UNIWERSYTET WARSZAWSKI

Wydział Matematyki,
Informatyki i Mechaniki

Łukasz Kowalik

ALGORYTMICZNE PROBLEMY ŚCIEŻKOWE W GRAFACH PLANARNYCH

Rozprawa Doktorska

Promotor rozprawy
dr hab. Krzysztof Diks
Instytut Informatyki

Styczeń 2005

Oświadczenie promotora rozprawy

Niniejsza rozprawa jest gotowa do oceny przez recenzentów.

Data Podpis promotora rozprawy

Oświadczenie autora rozprawy

Oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

Data Podpis autora rozprawy

Streszczenie

W niniejszej rozprawie zajmujemy się zagadnieniami algorytmicznymi związanymi z wyszukiwaniem w grafach planarnych ścieżek o zadanych własnościach. Głównym wynikiem jest nowa struktura danych umożliwiająca przetwarzanie zapytań postaci: *sprawdź, czy dwa zadane wierzchołki są odległe o nie więcej niż k i jeśli tak, zwróć odpowiednią najkrótszą ścieżkę.* Unikalną cechą zaproponowanej struktury jest to, że gdy k jest stałą niezależną od rozmiaru grafu, rozmiar struktury danych jest liniowy względem rozmiaru grafu, natomiast zapytania przetwarzane są w czasie stałym. Zaproponowane rozwiązanie jest niezwykle elastyczne: graf wejściowy może być zorientowany lub niezorientowany, a struktura danych może być szybko uaktualniana po dodaniu lub usunięciu krawędzi, a także po utożsamieniu dwóch wierzchołków.

Pokazujemy również, w jaki sposób wykorzystać opisaną strukturę danych do problemu wyszukiwania w grafie cykli o dowolnej długości. Jeżeli długość poszukiwanego cyklu jest stałą, uzyskujemy algorytm liniowy. W rzeczywistości czas działania algorytmu zależy bardzo silnie od długości szukanego cyklu, stąd wynik ten ma jedynie teoretyczne znaczenie. Dlatego stosując podobne, choć mniej ogólne metody, pokazujemy proste i praktyczne algorytmy wyszukiwania cykli o długościach co najwyżej 6.

Nasza struktura danych znalazła niespodziewane zastosowanie w problemach kolorowania grafów planarnych. Od 1959 roku znane jest twierdzenie Grötzscha, które mówi, że wierzchołki dowolnego grafu bez trójkątów można pokolorować trzema kolorami. Dotąd nie było jednak jasne, czy dla danego grafu można znaleźć takie kolorowanie w sposób efektywny, tj. w czasie bliskim liniowemu. W niniejszej rozprawie prezentujemy algorytm, który znajduje szukane kolorowanie w czasie $\mathcal{O}(n \log n)$.

Słowa kluczowe: Grafy planarne, najkrótsze ścieżki, wyrocznia, dynamiczne struktury danych, wyszukiwanie cykli, kolorowanie, twierdzenie Grötzscha.

Klasyfikacja tematyczna ACM: F.2.2, G.2.2.

Abstract

We deal with problems connected with finding in planar graphs paths of given properties. The main result of the thesis is a new data structure which allows processing queries of the form: *verify whether two given vertices are at distance at most k and if the answer is "yes", find one of the relevant shortest paths.* The number k is a constant independent of the input graph. The unique feature of our approach is constant query time and linear size of the data structure with respect to the size of the input. Our approach is very flexible: the input graph may be directed or undirected, our data structure can be updated very efficiently after deleting or inserting edges and after identifying vertices.

We also show how to apply the data structure mentioned above to the problem of finding cycles of arbitrary, but fixed, length in planar graphs. If the length is a constant the time complexity of our algorithm is linear. However, the running time of this algorithm depends very heavily on the length of the cycle. Hence this result is only of theoretical importance. Therefore using a similar, though not so general approach we show simple and practical algorithms for finding cycles of length at most 6.

Surprisingly, our data structure can be applied to coloring planar graphs. The famous Grötzsch's theorem, proved in 1959, says that vertices of any triangle-free planar graph can be colored with 3 colors. For the first time we show an algorithm which 3-colors such graphs in $\mathcal{O}(n \log n)$ time.

Keywords: Planar graphs, shortest paths, oracle, dynamic data structures, finding cycles, coloring, Grötzsch's theorem.

ACM Classification: F.2.2, G.2.2.

Niniejszą pracę dedykuję
mojej żonie Monice i mojej córce Zosi.

Spis treści

Wstęp

Tematem niniejszej rozprawy są algorytmy dla wybranych problemów ścieżkowych w grafach planarnych i ich zastosowania. Pojęcie *problemy ścieżkowe* odnosi się do zagadnień związanych z projektowaniem algorytmów wyszukujących w grafach ścieżki o zadanych własnościach. Skoncentrujemy się przede wszystkim na wyszukiwaniu najkrótszych ścieżek oraz ścieżek zadanej długości. Jest to jeden z najważniejszych, najbardziej naturalnych i najintensywniej badanych obszarów algorytmicznej teorii grafów. Problemy ścieżkowe pojawiają się jako podproblemy w wielu innych ważnych zagadnieniach algorytmiki (dobrymi przykładami są maksymalny przepływ i najliczniejsze skojarzenie). Efektywne algorytmy dla problemów ścieżkowych są także istotne w licznych zastosowaniach praktycznych, takich jak projektowanie architektury i protokołów dla sieci komputerowych, czy optymalizacja transportu w sieciach połączeń drogowych. Grafy, będące abstrakcjami tych oraz innych występujących w rzeczywistości sieci, bardzo często są planarne lub *bliskie* grafom planarnym. Dlatego nasze rozważania ograniczyliśmy do tej szczególnie interesującej klasy grafów, co pozwoliło na wykorzystanie ich specyficznych własności.

W praktyce często spotyka się sytuację, w której serwer przechowujący informacje o strukturze grafu (sieci) musi szybko przetwarzać duże ilości zapytań. Skupimy się na zapytaniach postaci: *znajdź najkrótszą ścieżkę łączącą wybraną parę wierzchołków*. Naturalnym podejściem jest wstępne przetworzenie grafu (*preprocessing*) w taki sposób, aby otrzymać strukturę danych, z której pomocą można szybko obsługiwać nadchodzące zapytania. Współczesne sieci bywają tak rozległe, że nie można zaakceptować podejścia polegającego na obliczeniu w fazie wstępnej wyników wszystkich możliwych zapytań. Badacze dążą więc do uzyskania możliwie krótkiego czasu odpowiedzi na zapytania, zachowując jednocześnie akceptowalną złożoność czasową i pamięciową fazy tworzenia struktury danych. Niemniej jednak, rezultaty w tej dziedzinie mają często formę kompromisu pomiędzy tymi wielkościami. Dla przykładu, Djidjev [19] podał strukturę danych zajmującą pamięć rozmiaru $\mathcal{O}(n^{4/3})$ i umożliwiającą przetwarzanie jednego zapytania w czasie $\mathcal{O}(n^{1/3} \log n)$. Gdy czas przetworzenia zapytania jest stały lub bliski stałemu, taką strukturę przyjęło się nazywać *wyrocznią*.

W rozdziale 1. opisujemy wyrocznię przetwarzającą zapytania specjalnej postaci, zaproponowanej po raz pierwszy przez Davida Eppsteina [22]: *sprawdź, czy dwa zadane wierzchołki są odległe o nie więcej niż k i jeśli tak, zwróć odpowiednią najkrót-*

szą ścieżkę. Liczba k jest ustaloną stałą, natomiast przetwarzany graf jest niezorientowanym grafem planarnym bez wag na krawędziach. W oparciu o nasz artykuł [39], zaprezentowany na konferencji STOC 2003, przedstawimy wyrocznię o rozmiarze $\mathcal{O}(n)$, konstruowaną w czasie $\mathcal{O}(n)$ i umożliwiającą przetwarzanie zapytań w czasie $\mathcal{O}(1)$. W dalszej kolejności zaprezentujemy istotne rozszerzenia i uogólnienia wprowadzonej struktury danych, opisane w naszej kolejnej pracy [38]. W szczególności pokażemy, w jaki sposób aktualizować strukturę wyroczni po dodaniu/usunięciu krawędzi lub wierzchołka grafu. W ten sposób nasza wyrocznia zyskuje unikalną możliwość pracy w środowisku dynamicznym. Złożoność czasowa aktualizowania wyroczni po usunięciu krawędzi wynosi $\mathcal{O}(1)$, natomiast po wstawieniu krawędzi $\mathcal{O}(\log^k n)$. Pokażemy również, że niewielkie modyfikacje struktury wyroczni pozwalają na jej wykorzystanie do zorientowanych grafów planarnych, bez pogorszenia złożoności czasowej i pamięciowej. Oprócz najkrótszych ścieżek, rozważymy także wyszukiwanie ścieżek zadanej długości, uzyskując analogiczne rezultaty. Obszerna dyskusja uzyskanych wyników na tle wcześniejszych rezultatów znajduje się na początku rozdziału 1.

W rozdziale 2. opiszemy interesujące zastosowanie przedstawionej wcześniej wyroczni krótkich ścieżek. Okazuje się, że nasza struktura danych może zostać użyta w kolorowaniu grafów, dziedzinie pozornie mało związanej z problemami ścieżkowymi. W problemie kolorowania grafu należy przypisać wierzchołkom grafu *kolory* tak, aby końce każdej krawędzi otrzymały inne kolory. Tematyka kolorowania wierzchołków grafów planarnych budzi ogromne zainteresowanie badaczy już od drugiej połowy XIX wieku. Odkąd twierdzenie o czterech barwach zostało udowodnione najważniejsze otwarte problemy poznawcze w tej dziedzinie dotyczą kolorowania z użyciem jedynie trzech kolorów. Od 1959 roku znane jest twierdzenie Grötzscha, które mówi, że wierzchołki dowolnego grafu planarnego bez trójkątów można pokolorować trzema kolorami. Dla badaczy zajmujących się algorytmiczną teorią grafów było jasne, że istniejące dowody tego twierdzenia odpowiadają wielomianowym algorytmom. Skonstruowanie efektywnego algorytmu, tj. bliskiego algorytmowi liniowemu, pozostawało jednak problemem otwartym. Okazało się, że użycie wyroczni krótkich ścieżek opisanej w rozdziale 1. pozwala przezwyciężyć kluczowe trudności na drodze do tego celu. Po zmodyfikowaniu dowodu twierdzenia i opracowaniu dodatkowych technik algorytmicznych udało się skonstruować algorytm o złożoności $\mathcal{O}(n \log n)$. Rozdział 2. zawiera dokładny opis tego algorytmu. Jego treść odpowiada naszemu artykułowi [37], zaprezentowanemu podczas konferencji ESA 2004.

W rozdziale 3. powracamy do problemów ścieżkowych. Naszą uwagę skoncentrujemy na wyszukiwaniu w grafach planarnych krótkich cykli, o długości co najwyżej 6. Jest to dość naturalny problem, który dodatkowo znajduje liczne zastosowania w innych obszarach algorytmicznej teorii grafów, takich jak kolorowanie, czy też testowanie stopnia spójności grafu. W ostatnich latach powstały algorytmy, które dla dowolnej stałej k znajdują cykl długości k w czasie liniowym. Jeden z takich algorytmów otrzymujemy przez zastosowanie naszej wyroczni wyszukującej ścieżki zadanej

długości. Niemniej jednak praktyczne znaczenie tych algorytmów wydaje się niewielkie ze względu na bardzo poważną zależność złożoności czasowej od wartości stałej k. Dlatego w ostatnim rozdziale skoncentrujemy się na konstruowaniu liniowych i w pełni praktycznych algorytmów wyszukiwania cykli o zadanych długościach. Przedstawiamy własne, nowe podejście do problemu, jak również rozwijamy wcześniejsze pomysły innych autorów. Zaimplementowaliśmy zamieszczone algorytmy i przeprowadziliśmy szereg testów potwierdzających ich praktyczną efektywność. Jako efekt uboczny naszych rozważań otrzymaliśmy także interesujące twierdzenie kombinatoryczne, które mówi, że maksymalna liczba cykli długości k w n-wierzchołkowym grafie planarnym wynosi $\Theta(n^{\lfloor \frac{k}{2} \rfloor})$. Treść ostatniego rozdziału pochodzi z naszego artykułu [36] zaprezentowanego podczas konferencji WG 2003.

Podziękowania

W pierwszej kolejności chciałbym podziękować mojemu promotorowi, prof. Krzysztofowi Diksowi. Jest on cierpliwym czytelnikiem wszystkich moich prac, nie szczędzącym wielu wnikliwych uwag. Jestem mu głęboko wdzięczny za całą pomoc okazaną mi podczas studiów doktoranckich. Zawsze robiła na mnie wrażenie wielka pasja, z jaką prof. Diks opowiada o algorytmice. Byłbym szczęśliwy, gdyby choć cień tej pasji można było odnaleźć w niniejszej rozprawie.

Równie gorąco dziękuję mojemu koledze z roku, Maćkowi Kurowskiemu. Wiele z prezentowanych tu wyników jest po prostu efektem naszej współpracy.

Praca nad niniejszą rozprawą była częściowo finansowana ze środków grantu Ministerstwa Nauki i Informatyzacji nr 3 T11C 008 26.

Pojęcia podstawowe, oznaczenia i założenia

Zanim podamy wykorzystywane w rozprawie najważniejsze definicje i oznaczenia najważniejszych pojęć chcielibyśmy zauważyć, że w teorii grafów wybór stosownej terminologii nie jest sprawą oczywistą. Jest to bowiem dziedzina operująca niezwykle dużą liczbą bardzo naturalnych pojęć. Nic więc dziwnego, że te same obiekty były „odkrywane" po wielokroć przez różnych autorów, którzy nie zawsze opisywali je tymi samymi słowami. Jako przykład można podać prace Tutte. Proces ujednolicania terminologii nie jest jeszcze niestety w pełni ukończony, o czym można się przekonać porównując aktualnie wydawane podręczniki teorii grafów w języku angielskim oraz polskie tłumaczenia niektórych z nich. W większości przypadków słownictwo i oznaczenia używane w niniejszej rozprawie są zgodne z podręcznikiem Cormena, Leisersona, Rivesta i Steina „Wprowadzenie do algorytmów" [16] oraz z podręcznikiem Wilsona „Wprowadzenie do teorii grafów" [56]. Odstępstwa pojawiają się w kilku przypadkach, uzasadnionych podwyższeniem czytelności i zgodnością z aktualnie powstającymi tekstami naukowymi. Dla wygody czytelnika, na końcu rozprawy zamieszczono indeks zawierający wszystkie zdefiniowane pojęcia wraz z numerami stron, na których można odszukać stosowne definicje.

Pojęcia podstawowe *Graf niezorientowany* definiujemy jako parę skończonych zbiorów (V, E). Elementy zbioru V nazywamy *wierzchołkami* grafu. Zbiór E zawiera nieuporządkowane pary wierzchołków, tzn.

$$E \subseteq \{\{u, v\} : \ u, v \in V \text{ oraz } u \neq v\}.$$

Elementy zbioru E nazywamy *krawędziami* grafu. Dla uproszczenia krawędź $\{u, v\}$ będziemy oznaczać krócej jako uv. Wierzchołki u i v nazywamy *końcami* krawędzi uv. Mówimy też, że wierzchołek u jest *incydentny* z krawędzią uv (podobnie wierzchołek v) oraz że u jest *sąsiadem* wierzchołka v. Zbiór sąsiadów wierzchołka u w grafie G oznaczamy przez $N_G(u)$ lub krótko $N(u)$, gdy jest jasne, o jaki graf chodzi. Liczbę krawędzi incydentnych z wierzchołkiem u nazywamy *stopniem* u w grafie G i oznaczamy przez $\deg_G(u)$. Będziemy używać oznaczeń $V(G)$ i $E(G)$ na zbiory wierzchołków i krawędzi grafu G. Przyjmujemy także konwencję, że $n = |V|$ oraz $m = |E|$,

kiedy oznaczenia te będą jednoznacznie określone. *Grafem pustym* nazwiemy graf o pustym zbiorze wierzchołków.

Ścieżką nazywamy niepusty graf $P = (V, E)$ zawierający wierzchołki $V = \{v_0, v_1, \ldots, v_t\}$ oraz krawędzie $E = \{v_0v_1, v_1v_2, \ldots, v_{t-1}v_t\}$. Wierzchołki $v_1, \ldots v_{t-1}$ nazywamy *wierzchołkami wewnętrznymi* ścieżki P, natomiast v_0 i v_t to *końce* ścieżki P. Mówimy, że ścieżka P *łączy* wierzchołki v_0 i v_t. Ścieżkę, w której $v_0 = v_t$ nazywamy *cyklem*. *Długością* ścieżki P nazywamy liczbę jej krawędzi i oznaczamy przez $|P|$. Ścieżkę długości k i cykl długości k będziemy nazywać krócej *k-ścieżką* i *k-cyklem*. Długość najkrótszego cyklu w grafie nazywamy *talią* tego grafu. Niech C będzie dowolnym cyklem w grafie. Krawędź grafu, która nie jest krawędzią cyklu C, ale łączy dwa wierzchołki cyklu, nazywa się *cięciwą cyklu C*. *Odległość* między wierzchołkami u i v w grafie G definiujemy jako długość najkrótszej ścieżki w G łączącej u i v, i oznaczamy przez $\mathrm{dist}_G(u, v)$. *Marszruta* w grafie G to niepusty naprzemienny ciąg $v_0e_0v_1e_1 \ldots e_{t-1}v_t$ wierzchołków i krawędzi grafu G taki, że $e_i = v_iv_{i+1}$, dla każdego $i = 0, \ldots, t-1$. Zauważmy, że w marszrucie te same wierzchołki mogą pojawiać się wiele razy. Jeśli wierzchołki są parami różne, to taka marszruta odpowiada ścieżce. Wierzchołki $v_1, \ldots, v_{t-1}$ będziemy nazywać *wierzchołkami wewnętrznymi*. Zauważmy, że zarówno v_0 jak i v_t mogą być wierzchołkami wewnętrznymi w przypadku gdy marszruta odwiedza je wielokrotnie. Jeżeli $v_0 = v_t$ mamy do czynienia z *marszrutą zamkniętą*.

Niech $G = (V, E)$ będzie grafem. *Grafem indukowanym przez zbiór $A \subseteq V$* nazywamy maksymalny (w sensie zawierania) podgraf grafu G o zbiorze wierzchołków A i oznaczamy przez $G[A]$.

Graf niezorientowany $G = (V, E)$ nazywamy *spójnym*, gdy dowolna para wierzchołków u i v grafu G jest połączona ścieżką. Maksymalne (w sensie zawierania) podgrafy spójne danego grafu nazywamy *spójnymi składowymi*. Jeżeli G jest grafem spójnym, to każdy wierzchołek v taki, że graf $G[V \setminus \{v\}]$ nie jest spójny, nazywamy *wierzchołkiem rozcinającym*[1]. Graf spójny G nie zawierający wierzchołków rozcinających nazywamy *dwuspójnym*. Dowolny zbiór wierzchołków $U \subset V$ grafu spójnego G nazwiemy *zbiorem separującym*, gdy graf $G[V \setminus U]$ nie jest spójny. Ogólnie, graf G nazywamy *k-spójnym*, gdy każdy zbiór separujący w G ma co najmniej k wierzchołków. Będziemy też używać pojęcia *cyklu separującego*, odnoszącego się do każdego cyklu C, takiego że zbiór $V(C)$ jest separujący.

Graf, w którym krawędziom przypisano kierunki nazywamy *grafem zorientowanym* lub krótko *digrafem*. Zbiór krawędzi grafu zorientowanego zawiera uporządkowane pary wierzchołków, tzn.

$$E \subseteq \{(u, v) : \ u, v \in V \text{ oraz } u \neq v\}.$$

Jeśli $e = (u, v)$ jest krawędzią w grafie zorientowanym, to mówimy, że krawędź e

[1] W niektórych podręcznikach wierzchołek rozcinający określa się jako *punkt artykulacji*.

wychodzi z u i *wchodzi do v*. *Ścieżką zorientowaną* od wierzchołka v_0 do wierzchołka v_t nazywamy niepusty digraf $P = (V, E)$, w którym $V = \{v_0, v_1, \ldots v_t\}$ oraz $E = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{t-1}, v_t)\}$. *Marszrutę zorientowaną* definiuje się podobnie jak w przypadku niezorientowanym, z tą różnicą, że każda krawędź e_i wychodzi z v_i i wchodzi do v_{i+1}.

Orientacje grafów Dla wierzchołka v, w grafie zorientowanym G, definiujemy jego *stopień wychodzący* jako liczbę krawędzi wychodzących z v oraz *stopień wchodzący* jako liczbę krawędzi wchodzących do v. Będziemy oznaczać te wartości przez, odpowiednio, $\text{outdeg}_G(v)$ oraz $\text{indeg}_G(v)$. Mówimy, że digraf G jest *d-zorientowany*, gdy stopień wychodzący każdego wierzchołka jest nie większy niż d. Niech G będzie grafem niezorientowanym i niech H będzie digrafem. Gdy graf H może zostać otrzymany z G poprzez zastąpienie każdej krawędzi $\{u, v\}$ grafu G przez krawędź (u, v), (v, u) lub parę (u, v) i (v, u) to mówimy, że H jest *bi-orientacją* grafu G. Najczęściej będziemy rozważać bi-orientacje nie zawierające jednocześnie krawędzi (u, v), (v, u). Wtedy mówimy po prostu o *orientacji* grafu G. Jeśli graf H jest d-zorientowany, nazywamy go także *d-orientacją* grafu G oraz mówimy, że G jest *d-orientowalny*. Przez $\overrightarrow{G}$ oznaczamy pewną, ale ustaloną, orientację grafu G. Graf niezorientowany G nazywamy *szkieletem* grafu zorientowanego $\overrightarrow{G}$.

Operacje na grafach Niech $G = (V_G, E_G)$ i $H = (V_H, E_H)$ będą grafami niezorientowanymi (analogiczne definicje stosują się do grafów zorientowanych). *Sumą* G i H nazywamy graf $G \cup H$, którego zbiorem wierzchołków jest $V_G \cup V_H$, a zbiorem krawędzi $E_G \cup E_H$. Niech $X \subseteq V_G$. Przez $G - X$ będziemy oznaczać graf $G[V_G - X]$. Niech $F = (V_F, E_F)$ będzie podgrafem G. *Różnicą grafów G i F* nazywamy graf o zbiorze wierzchołków V_G i zbiorze krawędzi $E_G \setminus E_F$. *Utożsamienie* u i v w grafie G, to operacja polegająca na zastąpieniu wierzchołków u, v nowym wierzchołkiem w połączonym ze wszystkimi wierzchołkami ze zbioru $V(G) \setminus \{u, v\}$, które były wcześniej sąsiadami u lub v.

Multigrafy Niekiedy wygodnie jest przyjąć, że graf niezorientowany może zawierać dwie lub więcej krawędzi łączących jedną parę wierzchołków. Takie krawędzie nazywamy *podwójnymi* (ang. *double edges*) lub *wielokrotnymi* (ang. *multiple edges*). Podobnie możemy rozważać grafy zorientowane, w których od dowolnego wierzchołka u do innego wierzchołka v może prowadzić wiele krawędzi. Mówimy wtedy o *multigrafach*. Jeżeli potrzebne jest wyraźne rozróżnienie, zdefiniowane wcześniej „zwykłe" grafy nazywamy *grafami prostymi*. Formalnie, multigraf definiujemy analogicznie jak graf prosty, ale z tą różnicą, że krawędzie multigrafu tworzą multizbiór, tzn. zbiór i funkcję, która każdemu elementowi zbioru przyporządkowuje dodatnią liczbę naturalną.

Zwróćmy także uwagę, że grafy proste nie zawierają *pętli*, czyli krawędzi, których oba końce są tym samym wierzchołkiem. W szczególnych sytuacjach będziemy rozszerzać pojęcia grafu i multigrafu tak, aby możliwe było istnienie pętli.

Grafy z wagami na krawędziach Będziemy czasem rozważać grafy wyposażone w funkcję, która każdej krawędzi przyporządkowuje pewną liczbę. Taką liczbę nazywamy *wagą* krawędzi. Intuicyjnie wagę krawędzi możemy rozumieć jako jej długość. Dlatego w grafie z wagami przypisanymi krawędziom, przez długość ścieżki rozumiemy sumę wag jej krawędzi.

Włożenia grafów i grafy planarne *Rysunkiem* lub *włożeniem* grafu w płaszczyznę (sferę, przestrzeń $\mathbf{R}^3$ itd.) nazywamy dowolne różnowartościowe odwzorowanie wierzchołków grafu w punkty płaszczyzny (sfery, przestrzeni) wraz z odwzorowaniem każdej krawędzi uv w łamaną zwyczajną na płaszczyźnie (sferze, w przestrzeni) o końcach w obrazach punktów u i v. Jeżeli obrazy dwóch krawędzi przecinają się w punkcie x, który nie jest obrazem wspólnego końca tych krawędzi to punkt x nazywamy *przecięciem*. Rysunek grafu na płaszczyźnie lub sferze nie zawierający przecięć, nazywamy *rysunkiem płaskim* lub *włożeniem płaskim*. Jeżeli istnieje rysunek płaski grafu G, to G nazywamy *grafem planarnym*. Zdarza się, że rysunek płaski grafu planarnego nazywa się także *grafem płaskim*. *Ściany* grafu płaskiego to maksymalne spójne podzbiory płaszczyzny, rozłączne z rysunkiem płaskim grafu. Często będziemy utożsamiać ścianę ze zbiorem krawędzi, których obrazy ograniczają daną ścianę. Będziemy mówić, że krawędź, która ogranicza ścianę f jest incydentna ze ścianą f. Podobnie, każdy z końców takiej krawędzi jest incydentny ze ścianą f. *Marszrutą ścianową* (ang. *facial walk*) w odpowiadającą ścianie f nazywamy najkrótszą marszrutę zamkniętą odwiedzającą wszystkie krawędzie incydentne z f. Jeżeli krawędzie incydentne z f tworzą cykl, to mówimy o *cyklu ścianowym*. *Długością ściany* f nazywamy długość marszruty w i oznaczamy przez $|f|$.

Kolorowanie grafów *Kolorowaniem grafu G z użyciem k kolorów* lub krócej *k-kolorowaniem*, nazywamy dowolną funkcję, która przypisuje wierzchołkom grafu G kolory ze zbioru $\{1, 2, \ldots, k\}$, w ten sposób, że końce każdej krawędzi mają różne kolory. Graf, dla którego istnieje takie kolorowanie nazywamy grafem k-kolorowalnym.

Model obliczeń W całej rozprawie stosujemy model RAM (maszyna z pamięcią o dostępie swobodnym, zobacz [1]). Z wyjątkiem miejsc, w których wyraźnie zaznaczono inaczej, przyjmujemy też powszechne w literaturze przedmiotu (zobacz np. dyskusję w [57]) założenia dotyczące złożoności obliczeń w obrębie algorytmów grafowych. Zakładamy mianowicie, że każde słowo pamięci składa się z w bitów, gdzie w jest wystarczająco duże, aby $\lfloor \log |V(G)| \rfloor + 1 \leq w$, dla dowolnego grafu G przetwarzanego lub konstruowanego przez algorytm. Wszelkie naturalne operacje na słowach pamięci wykonują się w czasie $\mathcal{O}(1)$. Oznacza to w szczególności, że możliwe jest porównanie identyfikatorów dwóch wierzchołków w czasie stałym oraz że graf o n wierzchołkach i m krawędziach zajmuje pamięć rozmiaru $\mathcal{O}(n + m)$.

Reprezentacja grafu w pamięci komputera We wszystkich prezentowanych algorytmach zakładamy, że grafy są reprezentowane przez listy sąsiedztwa, tj. dla każ-

dego wierzchołka grafu przechowywana jest dwukierunkowa lista jego sąsiadów. Jeżeli mamy do czynienia z grafem niezorientowanym, to w takiej reprezentacji każdej krawędzi $e = uv$ odpowiadają dwa elementy list sąsiedztwa: jeden na liście wierzchołka u, drugi na liście wierzchołka v. Zakładamy ponadto, że każde takie dwa elementy przechowują dodatkowo wskaźniki do siebie nawzajem. Pozwala to na usuwanie w czasie stałym dowolnej krawędzi reprezentowanej przez element pewnej listy sąsiedztwa.

Rozdział 1

Wyrocznia krótkich ścieżek

1.1 Wprowadzenie

Przypomnijmy, że mianem *wyroczni* przyjęło się nazywać strukturę danych umożliwiającą przetwarzanie zapytań w czasie stałym lub bliskim stałemu. W niniejszym rozdziale pokażemy wyrocznię dla zapytań o krótkie ścieżki łączące zadane pary wierzchołków. Poniżej przestawiamy uzyskane przez nas rezultaty na tle wcześniej znanych wyników.

1.1.1 Nowe wyniki

Najkrótsze ścieżki. Rozważamy następującą wersję problemu wyroczni najkrótszych ścieżek. Niech k będzie ustaloną stałą. Dany jest n-wierzchołkowy graf planarny G bez wag na krawędziach. Naszym celem jest zbudowanie struktury danych, która umożliwia sprawdzanie, czy dwa zadane wierzchołki są odległe w G o nie więcej niż k. Jeśli odpowiedź na to pytanie jest pozytywna, zwracana jest najkrótsza ścieżka. W niniejszym rozdziale pokażemy, w jaki sposób zbudować taką strukturę w czasie $\mathcal{O}(n)$. Rozmiar tworzonej struktury jest liniowy względem rozmiaru grafu G. Następnie zaprezentujemy algorytm, który z pomocą utworzonej struktury znajduje szukaną ścieżkę w czasie $\mathcal{O}(1)$.

Pokażemy także, w jaki sposób zaadoptować opisaną wyrocznię do pracy w środowisku dynamicznym. Po usunięciu dowolnej krawędzi lub wierzchołka z grafu możliwe jest zaktualizowanie struktury wyroczni w czasie stałym. Co więcej, jeśli w grafie pojawi się nowa krawędź, wyrocznia odświeża się w czasie polilogarytmicznym $\mathcal{O}(\log^k n)$. Jest to własność niezwykle istotna, ponieważ wiele rzeczywistych sieci podlega nieustannym zmianom i modelując je nie zawsze można pozwolić sobie na konstruowanie wyroczni od nowa po każdej modyfikacji struktury sieci. Ten sam argument dotyczy wykorzystywania wyroczni najkrótszych ścieżek jako pomoc-

niczej struktury danych w algorytmach rozwiązujących rozmaite problemy grafowe. Dobrym przykładem może być tu algorytm dla problemu kolorowania, którym zajmujemy się w rozdziale 2. Podczas działania algorytmu kolorowania, graf wejściowy podlega działaniu operacji wstawiania i usuwania krawędzi, a także utożsamiania par wierzchołków.

Dodatkowo opisywana struktura danych umożliwia ukrywanie i odkrywanie krawędzi oraz wierzchołków grafu w czasie $\mathcal{O}(1)$. Algorytm przetwarzający zapytania zachowuje się wtedy tak, jakby ukryte elementy zostały usunięte z grafu. Modeluje to pojawiającą się w praktycznych zastosowaniach sytuację, gdy węzły i połączenia w pewnej sieci mogą przez pewien czas pozostawać nieczynne, np. wskutek awarii.

Ścieżki o zadanej długości i cykle. Zaprezentujemy także algorytm przetwarzający w czasie stałym inne, równie naturalne zapytania. Dla danych dwóch wierzchołków u i v i liczby $t \leq k$, zwracana jest ścieżka łącząca u i v o długości t lub informacja, że graf G nie zawiera ścieżki o tej własności. Ponieważ w szczególności można położyć $v = u$, to w ten sposób znajdujemy również cykl zadanej długości zawierający u. Daje to także liniowy algorytm wyszukiwania cyklu danej długości w grafie planarnym.

Talia. Dla dowolnej stałej k nasza wyrocznia może być użyta do skonstruowania liniowego algorytmu, który sprawdza, czy dany graf planarny ma talię nie większą niż k i jeśli tak jest zwraca odpowiedni cykl. Sprawdzanie talii grafu planarnego jest istotne w wielu problemach kolorowania (patrz rozdział 3).

Uogólnienia. Należy zaznaczyć, że nasze rozważania koncentrujemy wokół niezorientowanych grafów bez wag na krawędziach jedynie w celu uproszczenia opisu. Zdecydowaliśmy się także skupić na grafach planarnych, ponieważ jest to klasa grafów powszechnie znana i szczególnie ważna w zastosowaniach. Niemniej jednak nasze wyniki mogą zostać użyte w ogólniejszym kontekście. Pokażemy, w jaki sposób rozszerzyć naszą wyrocznię tak, aby obsługiwała także grafy zorientowane i grafy z wagami na krawędziach (przy założeniu, że wagi są dodatnimi liczbami naturalnymi). Co więcej, rodzina grafów planarnych może zostać zastąpiona dowolną klasą grafów $\mathcal{C}$ spełniającą trzy następujące warunki:

(i) Grafy z klasy $\mathcal{C}$ są rzadkie, tzn. istnieje stała c taka, że dla dowolnego grafu $G \in \mathcal{C}$, $|E(G)| \leq c \cdot |V(G)|$,

(ii) $\mathcal{C}$ jest zamknięta na branie podgrafów,

(iii) jeśli graf G z klasy $\mathcal{C}$ jest podziałem (ang. *subdivision*)[1] grafu H, to $H \in \mathcal{C}$.

Dla przykładu, dla dowolnej stałej g możemy użyć naszej wyroczni dla grafów, które można narysować bez przecięć krawędzi na powierzchni *genusu* g. Intuicyjnie,

[1]Podział grafu zdefiniowano na stronie 31.

powierzchnia o genusie g jest homeomorficzna w sensie topologicznym ze sferą z g „rączkami". Dla przykładu torus ma genus 1. Inny przykład klasy grafów spełniających wymienione trzy warunki, to dowolna rodzina zamknięta na branie minorów. W szczególności grafy o szerokości drzewowej (ang. *tree-width*) ograniczonej przez stałą tworzą taką klasę (pojęcia minoru i szerokości drzewowej wykraczają poza zakres niniejszej rozprawy i nie zostaną w niej zdefiniowane[2]).

1.1.2 Tło

Wyrocznie najkrótszych ścieżek. Największym zainteresowaniem badaczy cieszy się ogólna wersja problemu, gdzie dany jest zorientowany graf planarny z wagami przypisanymi krawędziom i należy zbudować strukturę danych ułatwiającą przetwarzanie zapytań dotyczących najkrótszych ścieżek łączących zadane pary wierzchołków (bez ograniczeń na długość ścieżki). Rozważmy na początek podejścia skrajne. Jeśli nie stosujemy żadnego wstępnego przetwarzania grafu, to dla każdego zapytania najkrótszą ścieżkę można wyznaczyć w czasie $\mathcal{O}(n)$ za pomocą algorytmu Kleina i innych [35]. Drugi biegun to obliczenie w fazie wstępnej wyników wszystkich możliwych zapytań. Za pomocą wymienionego algorytmu dla każdego z n wierzchołków znajdziemy drzewo najkrótszych ścieżek, natomiast dla dowolnej pary wierzchołków zapamiętamy ich odległość w tablicy kwadratowej. Otrzymamy w ten sposób strukturę danych rozmiaru $\mathcal{O}(n^2)$, odległość między dwoma zadanymi wierzchołkami będzie odczytywana z tablicy w czasie stałym, natomiast najkrótszą ścieżkę otrzymamy w czasie proporcjonalnym do liczby jej krawędzi. Jak się okazuje, pomiędzy tymi dwiema skrajnościami możliwy jest płynny kompromis. Lipton i Tarjan [42], Frederickson [24], Arkati i inni [6] oraz Djidjev [19] pokazali, że jeśli możemy przeznaczyć pamięć rozmiaru s na strukturę danych, to zapytania mogą być przetwarzane w czasie $\mathcal{O}(n^2/s)$. Dla pamięci rozmiaru $s \in [n^{4/3}, n^{3/2}]$ Djidjev [19] poprawił czas przetwarzania zapytań do $\mathcal{O}(n/\sqrt{s}\log n)$. Chen i Xu [10] uogólnili wynik Djidjeva dla rozmiaru $s \in [n^{4/3}, n^2]$ otrzymując czas przetwarzania zapytań $\mathcal{O}(n/\sqrt{s}\log(n/s) + \alpha(n))$. Autorami aktualnie najlepszego wyniku w tej dziedzinie są Jittat Fakcharoenphol i Satish Rao [23]. Prezentują oni strukturę danych rozmiaru $\mathcal{O}(n \log n)$, tworzoną w czasie $\mathcal{O}(n \log^3 n)$, która umożliwia przetwarzanie zapytań w czasie $\mathcal{O}(\sqrt{n}\log^2 n)$. W dalszym ciągu nie wiadomo jednak, czy możliwe jest uzyskanie struktury danych o rozmiarze zbliżonym do liniowego zachowując równocześnie zbliżony do stałego czas obsługiwania zapytań.

Sytuacja poprawia się, jeśli przejdziemy do algorytmów aproksymacyjnych. Mikkel Thorup [54] zaprezentował wyrocznię zajmującą pamięć rozmiaru $\mathcal{O}(\varepsilon^{-1} n \log n \log \Delta)$, która umozliwia przetwarzanie zapytań o odległości ze stałą aproksymacji $(1+\varepsilon)$ i w czasie $\mathcal{O}(\log\log\Delta + \varepsilon^{-1})$, gdzie Δ jest największą skończoną odległością w danym grafie, przy założeniu że wagi wszystkich krawędzi są dodatnimi

[2]Definicje można znaleźć w książce R. Diestla „Graph Theory" [18] na stronach 16 i 257.

liczbami całkowitymi. W przypadku gdy dany jest niezorientowany graf planarny, konstrukcja wyroczni Thorupa upraszcza się. Dzięki temu rozmiar wyroczni maleje do $\mathcal{O}(\varepsilon^{-1} n \log n)$ natomiast czas przetwarzania zapytań wynosi $\mathcal{O}(\varepsilon^{-1})$. Obie wersje wyroczni można użyć do przetwarzania zapytań rozważanych w tym rozdziale. Aby otrzymać dokładne (a nie jedynie przybliżone) odpowiedzi na zapytania o najkrótsze ścieżki długości co najwyżej k w grafie bez wag, wystarczy położyć $\varepsilon = k^{-1}$. W ten sposób, jeśli k traktujemy jako stałą, rezultat Thorupa odpowiadają wyroczni dla grafów zorientowanych zajmującej pamięć rozmiaru $\mathcal{O}(n \log n \log \Delta)$ i przetwarzającej zapytania w czasie $\mathcal{O}(\log \log \Delta)$. Dla grafów niezorientowanych otrzymujemy natomiast stały czas przetwarzania zapytań i rozmiar wyroczni rzędu $\mathcal{O}(n \log n)$. Podobne wyniki, choć jedynie w wersji dla grafów niezorientowanych, przedstawił niezależnie Philip Klein [34].

W sposób jawny problem, którym zajmujemy się w tym rozdziale rozważał wcześniej David Eppstein [22]. W jego podejściu w czasie liniowym budowana jest wyrocznia liniowego rozmiaru, lecz zapytania obsługiwane są w czasie $\mathcal{O}(\log n)$. Algorytm Eppsteina łączy w sobie metody dekompozycji drzewowej (ang. *tree decomposition*) z technikami klastrowania Fredericksona [27]. Nasze podejście jest nie tylko efektywniejsze, lecz także prostsze w implementacji. W tej samej pracy Eppstein zaprezentował stosunkowo prosty algorytm, także wykorzystujący technikę dekompozycji drzewowej, który po wstępnym przetworzeniu grafu zajmującym czas $\mathcal{O}(n \log n)$, umożliwia otrzymywanie odpowiedzi na zapytania w czasie $\mathcal{O}(1)$. Warto zwrócić uwagę, że algorytmy Eppsteina wymagają, aby graf planarny był dany na wejściu w postaci tzw. *włożenia kombinatorycznego*, tzn. dla każdego wierzchołka v jego sąsiedzi muszą być podani w takiej kolejności, w jakiej pojawiają się wokół v w pewnym włożeniu grafu w płaszczyznę. Tymczasem dane wejściowe dla naszego algorytmu może stanowić lista krawędzi grafu podanych w dowolnej kolejności.

Czas przetwarzania zapytań	Czas konstrukcji wyroczni	Rozmiar wyroczni	Autorzy
$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Eppstein 1999, [22]
$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	Eppstein 1999, [22]
$\mathcal{O}(1)$	$\mathcal{O}(n \log^3 n)$	$\mathcal{O}(n \log n)$	Thorup 2001, [54]
$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Kowalik, Kurowski 2003, [39]

Tablica 1.1: Historia problemu wyroczni najkrótszych ścieżek o ograniczonej długości dla grafów planarnych.

Najkrótsze ścieżki omijające wadliwe elementy grafu i środowisko dynamiczne Demetrescu i Thorup [17] opisują wyrocznię umozliwiającą przetwarzanie zapytań postaci: *Jaka jest długość najkrótszej ścieżki z wierzchołka x do y, omijającej wadliwą krawędź (u, v) oraz która krawędź wychodząca z x znajduje się na początku takiej ścieżki?*. Jest to wyrocznia działająca dla dowolnych grafów zorientowanych,

zajmująca pamięć rozmiaru $\mathcal{O}(n^2 \log n)$ i umożliwiająca przetwarzanie zapytań w czasie $\mathcal{O}(\log n)$. W ten sposób, w przypadku gdy wagi krawędzi są liczbami całkowitymi, najkrótsza ścieżka długości k może zostać obliczona przez zadanie k zapytań w czasie $\mathcal{O}(k \log n)$. Niestety, opisana struktura danych pozwala na obecność w grafie tylko *jednej* wadliwej krawędzi. Dlatego, od chwili gdy w sieci modelowanej przez graf zostanie wykryta awaria, wyrocznia zwraca najkrótsze ścieżki unikające wadliwego połączenia, lecz cała struktura danych jest tworzona od nowa w tle. Ten przykład pokazuje jak trudne jest konstruowanie wyroczni najkrótszych ścieżek dla grafów podlegających zmianom. Wedle naszej najlepszej wiedzy nie pojawiły się dotąd żadne wyniki dotyczące wyroczni najkrótszych ścieżek dla grafów planarnych i działającej w środowisku dynamicznym.

Wyszukiwanie cykli zadanej długości i obliczanie talii. Problem wyszukiwania cykli określonych długości przyciągał uwagę wielu badaczy. Ponieważ w rozdziale 3. zajmujemy się właśnie wyszukiwaniem cykli długości od 3 do 6 w grafach planarnych, odsyłamy czytelnika do znajdującej się tam dyskusji wcześniejszych wyników. W tym miejscu wspomnijmy jedynie, że H. Djidjev [20] podał działający w czasie $\mathcal{O}(n^{5/4} \log n)$ algorytm obliczania talii dowolnego grafu planarnego.

1.2 Zarys pomysłu

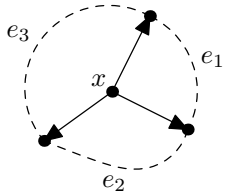
Niezwykle użyteczną cechą grafów planarnych jest fakt, że dla dowolnego grafu planarnego G można znaleźć jego 5-orientację $\overrightarrow{G_1}$ (tzn. orientację, w której z każdego wierzchołka wychodzi nie więcej niż 5 krawędzi). Można użyć w tym celu następującego prostego algorytmu. Dopóki graf G nie jest pusty, znajdujemy w G wierzchołek v stopnia co najwyżej 5 (zawsze taki istnieje), każdej krawędzi vw incydentnej z v w G nadajemy orientację (v, w) w $\overrightarrow{G_1}$ i usuwamy v z grafu G. Marek Chrobak i David Eppstein [14] pierwsi zwrócili uwagę na to, że takiej orientacji można użyć do przetwarzania zapytań postaci „czy wierzchołki x i y są sąsiednie". Zwróćmy bowiem uwagę, że $xy \in E(G)$ wtedy i tylko wtedy, gdy $(x, y) \in \overrightarrow{G_1}$ lub $(y, x) \in \overrightarrow{G_1}$. Stąd, po wstępnym przetworzeniu grafu w czasie liniowym, zapytania o sąsiedztwo są obsługiwane w czasie stałym.

Naszym celem będzie uogólnienie opisanego powyżej podejścia tak, by możliwe było sprawdzanie w czasie $\mathcal{O}(1)$, czy dwa zadane wierzchołki są odległe o nie więcej niż k i jeśli tak, otrzymanie odpowiedniej najkrótszej ścieżki.

Skoncentrujemy się teraz na przypadku $k = 2$, co pozwoli ukazać zarys podejścia stosowanego w ogólności. Niech u i v będą wierzchołkami odległymi o 2 w grafie G. Rozważmy ścieżkę p o długości 2 łączącą u i v. Krawędzie ścieżki p przyjmują jedną z trzech możliwych orientacji w grafie $\overrightarrow{G_1}$ (z wyłączeniem przypadku symetrycznego) przedstawionych na rysynku 1.1.

Rysunek 1.1: Możliwe orientacje ścieżki długości 2.

Widzimy, że jeżeli wystąpił przypadek (1) lub (2), to ścieżka p może zostać łatwo znaleziona w czasie stałym. Problem pojawia się gdy zajdzie przypadek (3), ponieważ stopnie wchodzące wierzchołków v i w mogą być dowolnie duże – nie możemy więc w poszukiwaniu wierzchołka pośredniego na ścieżce od v do w przejrzeć wszystkich krawędzi wchodzących do v i w. Zamiast tego utworzymy nowy, pomocniczy graf G_2. Zbiór wierzchołków G_2 pokrywa się ze zbiorem wierzchołków G, natomiast dwa wierzchołki v, w są połączone krawędzią w G_2 wtedy i tylko wtedy gdy istnieje wierzchołek x taki, że $(x, v) \in E(\overrightarrow{G_1})$ oraz $(x, w) \in E(\overrightarrow{G_1})$. Innymi słowy, dla każdej 2-ścieżki vxw w G odpowiadającej przypadkowi (3) utworzymy w G_2 krawędź vw. O wierzchołku x powiemy, że *wspiera* krawędź vw (patrz rysunek 1.2). Ponieważ stopnie



Rysunek 1.2: Krawędzie $e_1, e_2, e_3 \in E(G_2)$ są wspierane przez x.

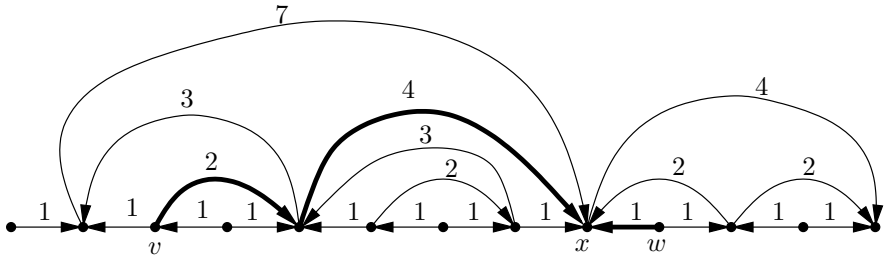
wychodzące w $\overrightarrow{G_1}$ są ograniczone przez stałą, każdy wierzchołek wspiera $\mathcal{O}(1)$ krawędzi w G_2. To pociąga za sobą, że G_2 może zostać utworzony w czasie liniowym. Jedyne czego teraz potrzebujemy to sprawdzanie sąsiedztwa w G_2 w czasie stałym. Chociaż G_2 nie musi być grafem planarnym, w podrozdziale 1.4 pokażemy, że G_2 jest $\mathcal{O}(1)$-orientowalny. W dalszej kolejności opiszemy, w jaki sposób otrzymać taką orientację w czasie liniowym. Przy pomocy $\mathcal{O}(1)$-orientacji grafu G_2 możemy więc wykrywać w czasie stałym ścieżki długości 2 w G, zorientowane zgodnie z przypadkiem (3).

Metoda opisana powyżej może zostać uogólniona dla dowolnej stałej k. W tym celu zbudujemy ciąg grafów $G_2, G_3, \ldots, G_k$ oraz ciąg orientacji tych grafów $\overrightarrow{G_2}, \overrightarrow{G_3}, \ldots, \overrightarrow{G_k}$. Każdy z grafów ma zbiór wierzchołków taki sam jak graf G. Dla dowolnego $t > 1$, krawędź uv należy do grafu G_t wtedy i tylko wtedy gdy istnieje $x \in V(G)$ taki, że $(x, u) \in E(\overrightarrow{G_i})$ oraz $(x, v) \in E(\overrightarrow{G_j})$, dla pewnych dodatnich liczb naturalnych i, j spełniających równość $i + j = t$. Intuicyjnie, każda krawędź w grafie G_t odpowiada pewnej marszrucie długości t w grafie G. Zwróćmy jednak uwagę, że jest to tylko niewielka część wszystkich takich marszrut pojawiających się w grafie G. Dla przykładu, jest $\Theta(n^2)$ ścieżek długości 2 w grafie planarnym $K_{1,n}$, natomiast każdy graf z ciągu $G_1, G_2, \ldots, G_k$ ma rozmiar liniowy względem rozmiaru G.

Sumę grafów $\overrightarrow{G_i}$ będziemy nazywać *grafem skrótów* i oznaczać przez $\overrightarrow{S_k(G)}$. Krawędzie grafu skrótów pochodzące z grafu $\overrightarrow{G_t}$ mają wagi równe t. Waga ścieżki w $\overrightarrow{S_k(G)}$ to suma wag jej krawędzi. Następujące własności pozwalają używać grafu skrótów jako wyroczni krótkich ścieżek:

(i) Każdy z grafów $\overrightarrow{G_i}$ jest $\mathcal{O}(1)$-zorientowany. Zauważmy, że w takiej orientacji, liczba wierzchołków osiągalnych w grafie skrótów z dowolnego wierzchołka v, przez ścieżki o wagach nie przekraczających k, jest ograniczona przez stałą.

(ii) Niech v i w będą wierzchołkami odległymi o $l \leq k$ w G. Wtedy w grafie skrótów istnieją dwie ścieżki zorientowane, $v \rightsquigarrow x$ i $w \rightsquigarrow x$ takie, że suma ich wag wynosi l.



Rysunek 1.3: Przykład grafu skrótów.

Łatwo zauważyć, że korzystając z własności (i) można znajdować ścieżki opisane w (ii) w czasie $\mathcal{O}(1)$. Prosty przykład grafu skrótów znajduje się na rysunku 1.3. Graf na wejściu jest ścieżką długości 12. Przyjmujemy $k = 7$. Wierzchołki v i w są odległe o 7. Ścieżki zorientowane, o których mowa w (ii), zostały na rysunku pogrubione.

1.3 Grafy rzadkie i ich własności

W tym podrozdziale sformułujemy używane często w całej rozprawie pojęcia i własności dotyczące grafów rzadkich. Rozważamy jedynie grafy proste. *Lesistość* (ang. *arboricity*) grafu G oznaczamy przez $arb(G)$ i definiujemy jako

$$arb(G) = \max_J \left\lceil \frac{|E(J)|}{|V(J)| - 1} \right\rceil,$$

gdzie J przebiega po wszystkich podgrafach G o co najmniej 2 wierzchołkach. Intuicyjnie, grafy o niskiej lesistości są jednolicie rzadkie. Ze wzoru Eulera wynika powszechnie znany fakt, sformułowany poniżej.

Fakt 1.3.1. *W dowolnym grafie planarnym G o co najmniej 3 wierzchołkach*

$$|(E(G)| \leq 3|V(G)| - 6.$$

■

Graf G o dwóch wierzchołkach ma co najwyżej 1 krawędź i wtedy $\left\lceil \frac{|E(G)|}{|V(G)|-1} \right\rceil \leq 1$. Z faktu 1.3.1 wnioskujemy, że w dowolnym grafie planarnym G o co najmniej 3 wierzchołkach $\left\lceil \frac{|E(G)|}{|V(G)|-1} \right\rceil \leq 3$. Każdy podgraf grafu planarnego także jest planarny, a więc także spełnia analogiczną nierówność. Dowodzi to następującego faktu:

Fakt 1.3.2. *Dowolny graf planarny ma lesistość nie przekraczającą 3.* ■

Poniższe twierdzenie dotyczące lesistości uzasadnia przy okazji stosowaną dla tego pojęcia nazwę:

Twierdzenie 1.3.3 (Nash-Williams [45]). *Dowolny graf o lesistości a jest sumą co najwyżej a lasów.* ■

Wniosek 1.3.4. *Każdy graf o lesistości a jest a-orientowalny.*

Dowód. Korzystając z twierdzenia 1.3.3 wystarczy podzielić graf na a lasów, w każdym drzewie w lesie wybrać dowolny wierzchołek jako korzeń i krawędzie każdego drzewa zorientować w kierunku korzenia. ■

Wniosek 1.3.5. *Dowolny graf planarny jest 3-orientowalny.* ■

Wprowadzimy jeszcze jedno pojęcie wiążące się z grafami rzadkimi. Graf G jest *k-zdegenerowany*, gdy dowolny niepusty podgraf H grafu G zawiera wierzchołek stopnia co najwyżej k. Użyteczne będą następujące proste fakty.

Lemat 1.3.6. *Dowolny graf d-orientowalny jest $2d$-zdegenerowany.*

Dowód. Niech G będzie dowolnym grafem d-orientowalnym, a H jego dowolną d-orientacją. Ponieważ $\sum_v \text{indeg}_H(v) = \sum_v \text{outdeg}_H(v)$, więc istnieje w G wierzchołek v taki, że $\text{indeg}_H(v) \leq \text{outdeg}_H(v)$. Wtedy $\deg_G(v) \leq 2d$. Teza wynika z faktu, że każdy podgraf grafu d-orientowalnego jest także d-orientowalny. ■

Lemat 1.3.7. *Dowolny graf d-zdegenerowany można d-zorientować w czasie liniowym. Co więcej, otrzymana orientacja będzie acykliczna.*

Dowód. Rozpoczynamy od grafu G, początkowo równemu grafowi wejściowemu oraz grafu H takiego, że $V(H) = V(G)$ oraz $E(H) = \emptyset$. Dopóki G jest niepusty wybieramy wierzchołek v stopnia nie większego niż d i dla każdej krawędzi $vw \in E(G)$ wstawiamy do grafu H krawędź (v, w). Następnie wierzchołek v jest usuwany z G. Widzimy, że po

zakończeniu algorytmu H jest d-orientacją grafu wejściowego. Acykliczność orientacji wynika z tego, że jeśli zawiera ona krawędź (x, y) to wiemy, że x był usunięty przed y, a więc istnienie cyklu prowadzi do sprzeczności.

W celu efektywnego wyszukiwania wierzchołków stopnia $\leq d$, dla każdego wierzchołka obliczamy i przechowujemy jego stopień. Obliczenie stopni przed uruchomieniem algorytmu zajmuje czas $\mathcal{O}(|V(G)| + \sum_v \deg_G(v)) = \mathcal{O}(|V(G)| + |E(G)|)$. Wierzchołki stopnia co najwyżej d wstawiane są do kolejki, dzięki czemu podczas działania algorytmu odnajdujemy je w czasie stałym. Po usunięciu wierzchołka z grafu stopnie jego sąsiadów są aktualizowane i w razie potrzeby nowe wierzchołki wstawiane są do kolejki. Zatem całkowity czas poświęcony na przetworzenie jednego wierzchołka stopnia c wynosi $\mathcal{O}(c)$. Sumując po wszystkich wierzchołkach ponownie otrzymujemy $\mathcal{O}(\sum_v \deg_G(v)) = \mathcal{O}(|E(G)|)$. Przedstawiony algorytm ma więc złożoność czasową $\mathcal{O}(|V(G)| + |E(G)|)$. ■

Wniosek 1.3.8. *Każdy graf d-zdegenerowany ma lesistość co najwyżej d.*

Dowód. Wniosek wynika natychmiast z acykliczości orientacji, której istnienie gwarantuje lemat 1.3.7. Wystarczy dla każdego wierzchołka ponumerować krawędzie wychodzące kolejnymi numerami, począwszy od 1. Widzimy, że cały graf jest sumą co najwyżej d lasów (i-ty las składa się z krawędzi, którym przypisalismy numer i). ■

Wniosek 1.3.9 (Aichholzer *i inni* [2]). *Dowolny graf d-orientowalny można $2d$-zorientować w czasie liniowym.*

Dowód. Dowód otrzymujemy natychmiast z lematów 1.3.6 i 1.3.7. ■

Lemat 1.3.10. *Dowolny graf d-orientowalny jest $(2d + 1)$-kolorowalny. Co więcej, takie kolorowanie można znaleźć w czasie liniowym.*

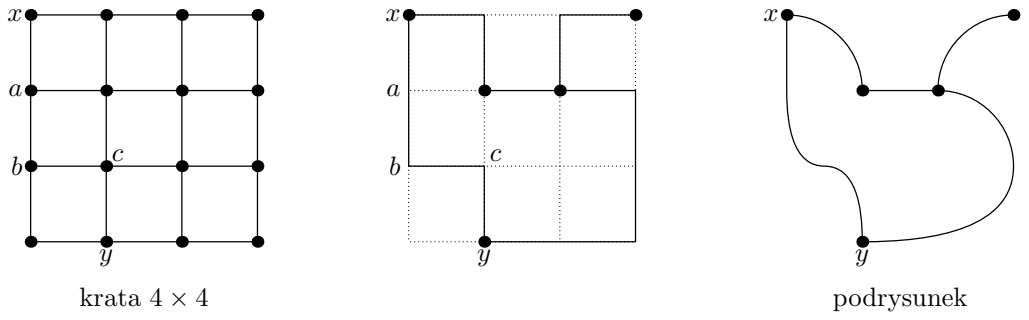
Dowód. Niech G będzie grafem d-orientowalnym. Dowód prowadzimy przez indukcję po liczbie wierzchołków. Dla grafu pustego teza jest trywialnie spełniona. Załóżmy więc, że graf jest niepusty. Wtedy z lematu 1.3.6 wynika, że graf G zawiera wierzchołek v stopnia co najwyżej $2d$. Możemy więc pokolorować indukcyjnie graf $G' = G - \{v\}$ i przypisać wierzchołkowi v „wolny" kolor, tzn. kolor ze zbioru $\{1, 2, \ldots, 2d+1\}$, którym nie został pokolorowany w grafie G' żaden sąsiad v. Pozostałe wierzchołki grafu G otrzymują takie same kolory, jak w grafie G'. Powyższy dowód w sposób oczywisty odpowiada rekurencyjnemu algorytmowi o złożoności liniowej (przy implementacji ponownie stosujemy techniki opisane w dowodzie wniosku 1.3.9). ■

1.4 Podrysunki i operacja ⊙

W celu omówienia pewnych własności grafu skrótów rozszerzymy używane w teorii grafów pojęcie podziału (ang. *subdivision*). Mówimy, że graf F jest *podziałem* grafu

H, gdy F może zostać otrzymany z H poprzez podział niektórych jego krawędzi, tj. poprzez zastąpienie pewnych krawędzi ścieżkami. Powiemy, że F jest *t-podziałem* grafu H, gdy F może zostać otrzymany z H poprzez zastąpienie pewnych krawędzi ścieżkami o długości co najmniej 1 i co najwyżej t. Powiemy też, że H jest *podrysunkiem* grafu G (niekoniecznie planarnego), jeśli pewien podział H jest podgrafem G. Podobnie, H jest *t-podrysunkiem* grafu G jeśli pewien t-podział H jest podgrafem G. Ustalmy pewien podgraf G, który jest podziałem H. Gdy krawędź $e \in E(H)$ została zastąpiona przez ścieżkę p w G, to mówimy, że e *odpowiada* ścieżce p. W takim przypadku będziemy oznaczać $e = e_H(p)$ oraz $p = p_G(e)$. Dla uproszczenia, jeżeli szkielet grafu zorientowanego H jest podrysunkiem grafu G, powiemy, że H jest podrysunkiem G.

Przykład Rysunek 1.4 przedstawia 5-podrysunek H „kraty" 4×4. Krawędź $xy \in E(H)$ odpowiada 4-ścieżce $xabcy$ w kracie G. Możemy napisać $xy = e_H(xabcy)$ oraz $xabcy = p_G(xy)$. Zauważmy, że gdy G jest grafem płaskim a H jego podrysunkiem to każda krawędź H może zostać narysowana na płaszczyźnie jako suma rysunków krawędzi G; otrzymamy w ten sposób włożenie płaskie grafu H.

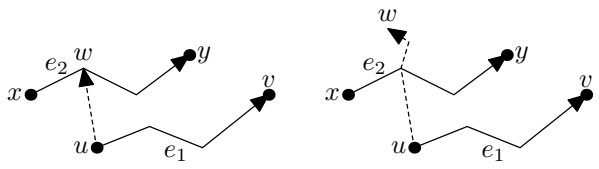


Rysunek 1.4: Przykładowy 5-podrysunek grafu kratowego 4×4.

Niech G i H będą dowolnymi grafami o wspólnym zbiorze wierzchołków V. Niech $\mathcal{R}$ będzie rodziną s krawędziowo rozłącznych podgrafów H, które sumują się do H, tzn. $\bigcup_{F \in \mathcal{R}} E(F) = E(H)$. Jeżeli każdy z grafów rodziny $\mathcal{R}$ jest t-podrysunkiem grafu G, to $\mathcal{R}$ nazywamy *(G,t)-reprezentacją H o grubości s*. Gdy H jest grafem zorientowanym i każdy element tej reprezentacji jest grafem d-zorientowanym powiemy, że reprezentacja $\mathcal{R}$ jest d-zorientowana.

Niech F i H będą zorientowanymi podrysunkami grafu G. Określimy pewną relację na zbiorze krawędzi F. Powiemy, że krawędź $e_1 = (u,v) \in E(F)$ *wskazuje za pomocą H* na inną krawędź $e_2 = (x,y) \in E(F)$, gdy istnieje krawędź $h = (u,w) \in E(H)$ taka, że co najmniej jeden wierzchołek ścieżki $p_G(h)$ jest wewnętrznym wierzchołkiem ścieżki $p_G(e_2)$, tzn. $V(p_G(h)) \cap (V(p_G(e_2)) - \{x,y\}) \neq \emptyset$ (patrz rysunek 1.5).

Poniższy lemat łączy ze sobą pojęcia podrysunku i relacji wskazywania.

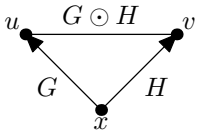


Rysunek 1.5: Krawędź e_1 wskazuje za pomocą H na krawędź e_2. Krawędzie grafu H oznaczono liniami przerywanymi.

Lemat 1.4.1. *Niech G będzie dowolnym grafem oraz niech A i B będą 1-zorientowanymi digrafami takimi, że A jest podrysunkiem G, a B jest b-podrysunkiem G. Wtedy dowolna krawędź grafu A wskazuje za pomocą B na co najwyżej b krawędzi grafu A.*

Dowód. Niech $e = (u, v)$ będzie dowolną krawędzią grafu A. Skoro B jest 1-zorientowany, to istnieje co najwyżej jedna krawędź w B wychodząca z u. Jeśli nie ma takiej krawędzi teza jest spełniona w sposób trywialny, w przeciwnym przypadku oznaczmy taką krawędź przez h. Zauważmy, że ścieżki w G odpowiadające dwóm krawędziom w A nie mogą mieć wspólnego wierzchołka wewnętrznego. Dlatego każdy z co najwyżej $b + 1$ wierzchołków ścieżki $p_G(h)$ jest wierzchołkiem wewnętrznym co najwyżej jednej ścieżki odpowiadającej krawędzi w A. Co więcej, u nie może być wierzchołkiem wewnętrznym żadnej takiej ścieżki. Wprost z definicji wynika, że e wskazuje za pomocą B na co najwyżej b krawędzi w A. ■

Zdefiniujemy teraz operację dwuargumentową $\odot$. Argumentami naszej operacji są grafy zorientowane G i H, natomiast wynikiem jest prosty graf niezorientowany o zbiorze wierzchołków $V(G) \cup V(H)$. Dwa wierzchołki u i v są sąsiednie w grafie $G \odot H$, wtedy i tylko wtedy gdy istnieje wierzchołek x taki, że $(x, u) \in E(G)$ oraz $(x, v) \in E(H)$. Mówimy, że wierzchołek x *wspiera* krawędź uv. Zauważmy, że jedna krawędź może być wspierana przez wiele wierzchołków.



Rysunek 1.6: Wierzchołek x wspiera krawędź uv grafu $G \odot H$.

W tej chwili jesteśmy już gotowi sformułować i udowodnić twierdzenie, które ukaże kluczowe własności zdefiniowanej właśnie operacji $\odot$.

Twierdzenie 1.4.2. *Niech G będzie dowolnym grafem niezorientowanym oraz niech A i B będą 1-zorientowanymi digrafami. Co więcej, niech grafy A i B będą, odpowiednio, a-podrysunkiem i b-podrysunkiem grafu G. Wtedy graf $A \odot B$ ma $(G, a + b)$-reprezentację o grubości $5 \cdot (2a+1)(2b+1)$. Co więcej, jeżeli graf G ma n wierzchołków*

oraz liczby a i b są ograniczone przez stałą, to taka reprezentacja może zostać znaleziona w czasie $\mathcal{O}(n)$.

Dowód. Z lematu 1.4.1 wynika, że każda krawędź A wskazuje za pomocą B na co najwyżej b innych krawędzi A. Innymi słowy zbiór krawędzi A oraz relacja wskazywania definiują graf b-zorientowany. Z lematu 1.3.10 możemy pokolorować krawędzie A za pomocą $2b+1$ kolorów w taki sposób, że jeśli pewna krawędź wskazuje inną krawędź, to mają one inne kolory. Podzielmy graf A na $2b+1$ grafów:

$$A = A_1 \cup A_2 \cup \ldots \cup A_{2b} \cup A_{2b+1},$$

takich, że każdy zawiera krawędzie pokolorowane tym samym kolorem. W analogiczny sposób dzielimy graf B na $2a+1$ grafów:

$$B = B_1 \cup B_2 \cup \ldots \cup B_{2a} \cup B_{2a+1}.$$

Powyższy podział gwarantuje, że w żadnym z grafów A_i nie ma krawędzi wskazującej za pomocą B na inną krawędź A_i (analogiczna własność jest spełniona dla grafów B_i). Zauważmy teraz, że

$$A \odot B = \bigcup_{\substack{1 \le i \le 2b+1 \\ 1 \le j \le 2a+1}} A_i \odot B_j.$$

Teraz wystarczy wykazać, że każdy z grafów $A_i \odot B_j$ ma $(G, a+b)$-reprezentację grubości 5. W tym celu przeprowadzimy jeszcze jeden podział. Rozważmy graf $A_i \cup B_j$ dla dowolnych i, j. Jest to graf 2-zorientowany. Stosując algorytm z lematu 1.3.10 otrzymujemy pewne 5-kolorowanie grafu $A_i \cup B_j$. Wierzchołki tego samego koloru nie są połączone ani w A_i, ani w B_j.

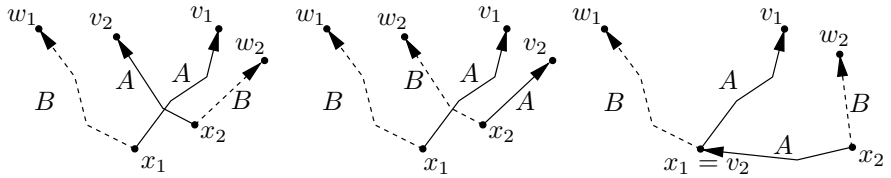
Niech c będzie dowolnym z pięciu kolorów wierzchołków. Oznaczmy przez $A_{i,c}$ graf złożony tylko z tych krawędzi A_i, które wychodzą z wierzchołków o kolorze c. Podobnie oznaczmy przez $B_{j,c}$ graf złożony tylko z tych krawędzi B_j, które wychodzą z wierzchołków o kolorze c. Prawdziwa jest następująca równość:

$$A_i \odot B_j = \bigcup_{1 \le c \le 5} A_{i,c} \odot B_{j,c}.$$

Teraz wystarczy wykazać, że dla dowolnych i, j, c graf $R_{i,j,c} = A_{i,c} \odot B_{j,c}$ jest $(a+b)$-podrysunkiem grafu G. W tym celu dla każdej krawędzi grafu $R_{i,j,c}$ wskażemy odpowiadającą jej ścieżkę w G. Niech vw będzie dowolną krawędzią w $R_{i,j,c}$. Zgodnie z definicją operacji $\odot$ istnieje wierzchołek x taki, że (x, v) jest krawędzią w $A_{i,c}$ oraz (x, w) jest krawędzią w $B_{j,c}$. Takich wierzchołków wspierających vw może być wiele, lecz w takim wypadku wybieramy dowolny z nich. Niech $w_G(vw)$ będzie marszrutą o początku w wierzchołku v i końcu w wierzchołku w, odpowiadającą sumie ścieżek $p_G(xv)$ i $p_G(xw)$. Oczywiście $w_G(vw)$ zawiera co najwyżej $a+b$ krawędzi. Zauważmy

jednak, że $w_G(vw)$ nie musi być ścieżką, gdyż ścieżki $p_G(xv)$ i $p_G(xw)$ mogą się wielokrotnie przecinać. Jako ścieżkę $p_G(vw)$ wybierzemy najkrótszą ścieżkę łączącą v i w w grafie $p_G(xv) \cup p_G(xw)$. Niemniej jednak, pojęcia marszruty $w_G(vw)$ będziemy jeszcze potrzebować w dalszej części dowodu.

W ten sposób każdej krawędzi grafu $R_{i,j,c}$ przypisaliśmy ścieżkę w G długości nie większej niż $a + b$. Niech H będzie grafem otrzymanym z $R_{i,j,c}$ przez zastąpienie krawędzi przypisanymi im ścieżkami. Oczywiście $H \subseteq G$. Aby wykazać, że $R_{i,j,c}$ jest $(a + b)$-podrysunkiem, należy jeszcze udowodnić, że H jest podziałem $R_{i,j,c}$, tj. nie istnieją dwie krawędzie $e_1, e_2 \in E(R_{i,j,c})$ takie, że pewien wierzchołek wewnętrzny $p_G(e_1)$ należy do $p_G(e_2)$. Wygodniej nam będzie pokazać bardziej ogólny fakt: udowodnimy, że nie istnieją dwie krawędzie $e_1, e_2 \in E(R_{i,j,c})$ takie, że pewien wierzchołek wewnętrzny marszruty $w_G(e_1)$ należy do wierzchołków $w_G(e_2)$. Załóżmy że taka para krawędzi jednak istnieje. Niech $e_1 = v_1w_1$ i $e_2 = v_2w_2$. Niech x_1 (odpowiednio x_2) będzie wierzchołkiem, który wspiera e_1 (odpowiednio e_2). Bez straty ogólności zakładamy, że $x_1v_1, x_2v_2 \in A_{i,c}$ oraz $x_1w_1, x_2w_2 \in B_{j,c}$. Do rozważenia są trzy przypadki (patrz rysunek 1.7):



Przypadek (i) Przypadek (ii) Przypadek (iii)

Rysunek 1.7: Przypadki (i)-(iii)

(i) Wewnętrzny wierzchołek ścieżki $p_G(x_1v_1)$ jest wierzchołkiem ścieżki $p_G(x_2v_2)$. Jest to niemożliwe, ponieważ $A_{i,c}$ jest podrysunkiem G. Podobnie nie może być tak, że wewnętrzny wierzchołek ścieżki $p_G(x_1w_1)$ jest wierzchołkiem ścieżki $p_G(x_2w_2)$.

(ii) Wewnętrzny wierzchołek ścieżki $p_G(x_1v_1)$ jest wierzchołkiem ścieżki $p_G(x_2w_2)$. Widzimy, że wtedy $x_2v_2 \in A_{i,c}$ wskazuje za pomocą B na $x_1v_1 \in A_{i,c}$, a to jest sprzeczność. Możemy od razu wykluczyć symetryczny przypadek, gdy wewnętrzny wierzchołek ścieżki $p_G(x_1w_1)$ jest wierzchołkiem ścieżki $p_G(x_2v_2)$.

(iii) Wierzchołek x_1 jest jednym z wierzchołków marszruty $w_G(e_2)$. Ponieważ wykluczyliśmy przypadki (i) i (ii), x_1 nie może być wierzchołkiem wewnętrznym ani $p_G(x_2v_2)$ ani $p_G(x_2w_2)$. Ponieważ grafy $A_{i,c}$ i $B_{j,c}$ są 1-zorientowane wiemy, że $x_1 \neq x_2$. W takim razie x_1 może być jedynie jednym z wierzchołków v_2, w_2. Oznacza to, że x_1 sąsiaduje z x_2 w grafie $A_{i,c}$ (gdy $x_1 = v_2$, tak jak na rysunku 1.7) lub x_1 sąsiaduje z x_2 w grafie $B_{j,c}$ (gdy $x_1 = w_2$). Żaden z tych

dwóch przypadków nie może się jednak zdarzyć, gdyż x_1 i x_2 mają ten sam kolor c.

Pokazaliśmy, że $R_{i,j,c}$ jest $(a+b)$-podrysunkiem G. To kończy dowód faktu, że zbiór

$$\mathcal{R} = \{R_{i,j,c} : 1 \leq i \leq 2b+1;\ 1 \leq j \leq 2a+1;\ 1 \leq c \leq 5\}$$

jest $(G, a+b)$-reprezentacją grafu $A \odot B$ o grubości $5 \cdot (2a+1) \cdot (2b+1)$. Reprezentacja $\mathcal{R}$ może zostać obliczona za pomocą algorytmu 1.4.1. Podany algorytm odpowiada konstrukcji przeprowadzonej w dowodzie. Łatwo się przekonać, że działa on w czasie liniowo zależnym od rozmiaru grafu G (liczby a i b traktujemy jako stałe). Do kolorowania grafów Γ_A, Γ_B (patrz algorytm 1.4.1) oraz $A \cup B$ używamy algorytmu opisanego w dowodzie lematu 1.3.10. ■

Algorytm 1.4.1 Znajdowanie $(G, a+b)$-reprezentacji $\mathcal{R}$ grafu $A \odot B$ o grubości $5(2a+1)(2b+1)$.

Wejście: 1-zorientowane grafy A i B; A jest a-podrysunkiem G; B jest b-podrysunkiem G; wraz z każdą krawędzią e grafów A i B przechowywana jest odpowiadająca jej ścieżka $p_G(e)$.

Wyjście: $(G, a+b)$-reprezentacja $\mathcal{R}$ grafu $A \odot B$ o grubości $5(2a+1)(2b+1)$.

1: $\mathcal{R} \leftarrow \emptyset$
2: Oblicz grafy Γ_A, Γ_B opisane relacją wskazywania na krawędziach A i B.
3: $(2b+1)$-koloruj graf Γ_A
4: $(2a+1)$-koloruj graf Γ_B
5: 5-koloruj graf $A \cup B$
6: **for** $i \leftarrow 1$ **to** $2b+1$ **do**
7: **for** $j \leftarrow 1$ **to** $2a+1$ **do**
8: **for** $c \leftarrow 1$ **to** 5 **do**
9: $A_{i,c} \leftarrow$ graf złożony z krawędzi A koloru i wych. z wierzchołków koloru c
10: $B_{j,c} \leftarrow$ graf złożony z krawędzi B koloru j wych. z wierzchołków koloru c
11: $R_{i,j,c} \leftarrow A_{i,c} \odot B_{j,c}$ ▷ Dla każdej krawędzi $R_{i,j,c}$ obliczana jest odpowiadająca jej ścieżka w G
12: $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_{i,j,c}\}$

Wniosek 1.4.3. *Niech G będzie dowolnym grafem n-wierzchołkowym. Niech A będzie digrafem z daną d_A-zorientowaną (G, a)-reprezentacją grubości $t(A)$ oraz niech B będzie digrafem z daną d_B-zorientowaną (G, b)-reprezentacją grubości $t(B)$. Wtedy graf $A \odot B$ ma $(G, a+b)$-reprezentację grubości $5 \cdot d_A \cdot d_B \cdot t(A) \cdot t(B) \cdot (2a+1) \cdot (2b+1)$. Co więcej, przy założeniu że liczby a i b są ograniczone przez stałe, taka reprezentacja może zostać obliczona w czasie $\mathcal{O}(n \cdot t(A) \cdot t(B) \cdot d_A \cdot d_B)$.*

Dowód. Niech $A = A_1 \cup A_2 \cup \ldots \cup A_{t(A)}$ oraz $B = B_1 \cup B_2 \cup \ldots \cup B_{t(B)}$ będą, odpowiednio, (G, a)-reprezentacją grafu A i (G, b)-reprezentacją grafu B. Zauważmy, że prawdziwa jest równość:

$$A \odot B = \bigcup_{i=1}^{t(A)} \bigcup_{j=1}^{t(B)} A_i \odot B_j.$$

Każdy z grafów A_i może być w dalszej kolejności przedstawiony jako suma 1-zorientowanych podrysunków G: $A_{i,1}, \ldots A_{i,d_A}$. Podobnie, dla każdego $j = 1, \ldots t(B)$, dzielimy graf B_j na 1-zorientowane podrysunki G: $B_{j,1}, \ldots B_{j,d_B}$. Zauważmy, że

$$A_i \odot B_j = \bigcup_{p=1}^{d_A} \bigcup_{q=1}^{d_B} A_{i,p} \odot B_{j,q}.$$

Ostatecznie otrzymujemy

$$A \odot B = \bigcup_{i=1}^{t(A)} \bigcup_{j=1}^{t(B)} \bigcup_{p=1}^{d_A} \bigcup_{q=1}^{d_B} A_{i,p} \odot B_{j,q}.$$

Teraz wystarczy zastosować twierdzenie 1.4.2 do każdego z $t(A) \cdot t(B) \cdot d_A \cdot d_B$ składników sumy. ■

1.5 Graf skrótów

Odtąd będziemy zakładać, że k jest ustaloną dodatnią liczbą całkowitą, natomiast G jest n-wierzchołkowym niezorientowanym grafem planarnym. Opiszemy strukturę danych $\overrightarrow{S_k(G)}$, którą będziemy nazywać *grafem k-skrótów dla grafu G* lub krócej *grafem skrótów*. Pokażemy, że graf skrótów ma następujące własności:

(i) budowanie $\overrightarrow{S_k(G)}$ zajmuje czas $\mathcal{O}(n)$,

(ii) rozmiar pamięci zajmowanej przez $\overrightarrow{S_k(G)}$ jest rzędu $\mathcal{O}(n)$,

(iii) dla dowolnych dwóch wierzchołków u, v grafu G, z pomocą grafu skrótów można w czasie $\mathcal{O}(1)$ sprawdzić, czy u i v leżą w G w odległości nie przekraczającej k.

1.5.1 Budowanie grafu skrótów

$\overrightarrow{S_k(G)}$ jest zorientowanym multigrafem o zbiorze wierzchołków $V(G)$. Krawędzie mają przypisane wagi ze zbioru $\{1, \ldots, k\}$. Podgraf grafu skrótów złożony z wszystkich krawędzi o wadze i będziemy oznaczać przez $\overrightarrow{G_i}$. Będziemy także odwoływać się do szkieletów grafów $\overrightarrow{S_k(G)}$ i $\overrightarrow{G_i}$ oznaczając je odpowiednio jako $S_k(G)$ i G_i.

Aby zbudować graf skrótów konstruujemy kolejno grafy $\overrightarrow{G_1}, \overrightarrow{G_2}, \ldots \overrightarrow{G_k}$. Dla każdego z grafów $\overrightarrow{G_i}$ obliczamy i przechowujemy jego 3-zorientowaną (G, i)-reprezentację. Cały algorytm polega na naprzemiennym wykonywaniu dwóch, opisanych poniżej, kroków.

Krok 1. Budowanie grafu G_i

Graf G_1 jest po prostu kopią grafu wejściowego G. Dla $i > 1$ graf G_i powstaje jako wynik stosowania operacji $\odot$ do par wcześniej utworzonych grafów, zgodnie z poniższym wzorem:

$$G_i = \bigcup_{\substack{1 \le p \le q < i \\ p+q=i}} \overrightarrow{G_p} \odot \overrightarrow{G_q} \tag{1.1}$$

Każdą z operacji $\overrightarrow{G_p} \odot \overrightarrow{G_q}$ wykonujemy osobno używając algorytmu z wniosku 1.4.3. Otrzymujemy w ten sposób także (G, i)-reprezentację grafu G_i. Zauważmy, że grubość tej reprezentacji jest funkcją i niezależną od n, a więc jest rzędu $\mathcal{O}(1)$.

Krok 2. Orientowanie krawędzi grafu G_i

Mamy daną (G, i)-reprezentację grubości $\mathcal{O}(1)$ grafu G_i. Każdy element tej reprezentacji jest podrysunkiem G, a więc grafem planarnym. Taki graf można bardzo łatwo 5-zorientować w czasie liniowym algorytmem opisanym w podrozdziale 1.2. Wiemy jednak z wniosku 1.3.5, że grafy planarne są 3-orientowalne. Taką 3-orientację także można znaleźć w czasie liniowym. Odpowiedni algorytm podali M. Chrobak i D. Eppstein [14].

Po 3-zorientowaniu kolejno wszystkich grafów (G, i)-reprezentacji otrzymujemy graf $\overrightarrow{G_i}$ i jego 3-zorientowaną (G, i)-reprezentację. Zauważmy, że skoro grubość tej reprezentacji jest $\mathcal{O}(1)$, więc stopnie wychodzące wierzchołków w $\overrightarrow{G_i}$ są ograniczone przez stałą.

Wniosek 1.5.1. *Dla dowolnego grafu planarnego G graf skrótów $\overrightarrow{S_k(G)}$ spełnia następujące warunki:*

(i) Krawędź (u, v) z wagą t w $\overrightarrow{S_k(G)}$ oznacza, że w grafie G istnieje ścieżka o długości co najwyżej t łącząca u i v.

(ii) Graf $\overrightarrow{S_k(G)}$ jest $\mathcal{O}(1)$-zorientowany.

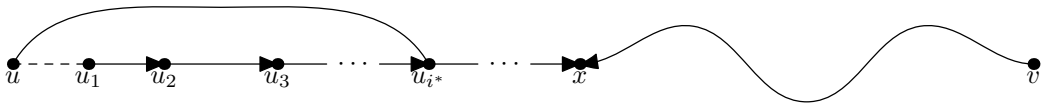
(iii) Algorytm budowania grafu $\overrightarrow{S_k(G)}$ ma złożoność czasową i pamięciową $\mathcal{O}(n)$.

1.5.2 Przetwarzanie zapytań

Wagą ścieżki P w grafie skrótów nazwiemy sumę wag jej krawędzi i będziemy ją oznaczać przez $w(P)$ (analogicznie definiujemy wagę marszruty). Udowodnimy teraz niezwykle istotną własność grafu skrótów.

Twierdzenie 1.5.2. *Niech G będzie grafem planarnym i niech $\overrightarrow{S_k(G)}$ będzie grafem skrótów. Jeżeli wierzchołki u i v są połączone ścieżką długości $l \leq k$ w grafie G, to w $\overrightarrow{S_k(G)}$ istnieją dwie zorientowane ścieżki, pierwsza od u do pewnego wierzchołka x i druga od v do x, takie że suma ich wag wynosi l.*

Dowód. Przeprowadzimy dowód przez indukcję po l. Twierdzenie jest prawdziwe w sposób trywialny dla $l = 0$. Przejdźmy teraz do kroku indukcyjnego. Przez P_1 i P_2 oznaczymy ścieżki zorientowane w grafie skrótów, których istnienie chcemy udowodnić. Niech P będzie ścieżką długości l łączącą u i v w grafie G. Oznaczmy przez u_1 sąsiada wierzchołka u na ścieżce P. Z założenia indukcyjnego wiemy, że w $\overrightarrow{S_k(G)}$ istnieją dwie zorientowane ścieżki, pierwsza od u_1 do pewnego wierzchołka x i druga od v do x, takie że suma ich wag wynosi $l - 1$. Oznaczmy te ścieżki odpowiednio przez Q_1 i Q_2. Niech $u_1, u_2, \ldots u_r = x$ będą kolejnymi wierzchołkami ścieżki Q_1 (patrz



Rysunek 1.8: Dowód twierdzenia 1.5.2.

rysunek 1.8). Zdefiniujmy zbiór Z jak następuje[3]:

$$Z = \{i : 1 \leq i \leq r \text{ oraz } uu_i \in E(G_{1+\mathrm{dist}_{Q_1}(u_1,u_i)})\}.$$

Skoro $uu_1 \in E(G_1)$, więc $1 \in Z$ i stąd $Z \neq \emptyset$. Niech $i^* = \max Z$.

Jeśli $i^* = r$, czyli $u_{i^*} = x$, to graf $\overrightarrow{G_{1+\mathrm{dist}_{Q_1}(u_1,x)}}$ zawiera albo krawędź (u, x) albo (x, u). Jeśli zawiera (u, x), kładziemy $P_1 = \{(u, x)\}$ i $P_2 = Q_2$. Wtedy $w(P_1) = 1 + w(Q_1)$ i $w(P_2) = w(Q_2)$. W przeciwnym przypadku kładziemy pustą ścieżkę jako P_1 i $P_2 = Q_2 \cup \{(x, u)\}$. Wtedy $w(P_1) = 0$ i $w(P_2) = w(Q_2) + 1 + w(Q_1)$. Widzimy, że w obu przypadkach $w(P_1) + w(P_2) = l$.

Załóżmy więc, że $i^* < r$. Niech $a = \mathrm{dist}_{Q_1}(u_{i^*}, u_{i^*+1})$ i $b = 1 + \mathrm{dist}_{Q_1}(u_1, u_{i^*})$. Wiemy, że $(u_{i^*}, u) \in E(\overrightarrow{G_b})$ lub $(u, u_{i^*}) \in E(\overrightarrow{G_b})$. Jeśli $(u_{i^*}, u) \in E(\overrightarrow{G_b})$, to $uu_{i^*+1} \in E(G_{a+b})$, a więc $i^* + 1 \in Z$ – sprzeczność. Ostatecznie zostaje $(u, u_{i^*}) \in E(\overrightarrow{G_b})$ i wtedy $P_2 = Q_2$, natomiast P_1 składa się z krawędzi (u, u_{i^*}) i kolejnych krawędzi ścieżki Q_1, tj. krawędzi $(u_{i^*}, u_{i^*+1}), \ldots, (u_{r-1}, u_r)$. Łatwo sprawdzić, że i tym razem $w(P_1) + w(P_2) = l$. ■

[3]Zapis $\mathrm{dist}_{Q_1}(a, b)$ oznacza odległość między wierzchołkami a i b na ścieżce Q_1, traktowanej jako graf z wagami na krawędziach.

Algorytm odpowiedzi na zapytanie o najkrótszą ścieżkę

Z twierdzenia 1.5.2 otrzymujemy prosty algorytm, który w czasie $\mathcal{O}(1)$ sprawdza, czy $\text{dist}_G(u,v) \leq k$ i jeśli tak jest, oblicza najkrótszą ścieżkę łączącą u i v. Pokazaliśmy, że stopnie wychodzące w grafie skrótów są ograniczone przez stałą. Oznaczmy tę stałą przez Δ. Niech $\overrightarrow{S(u)}$ będzie podgrafem grafu skrótów indukowanym przez wszystkie wierzchołki osiągalne w $\overrightarrow{S_k(G)}$ z u za pomocą ścieżek o wagach nie przekraczających k. Ponieważ każda taka ścieżka składa się z co najwyżej k krawędzi, graf $\overrightarrow{S(u)}$ ma nie więcej niż $\Delta^{k+1} = \mathcal{O}(1)$ krawędzi. Analogicznie definiujemy graf $\overrightarrow{S(v)}$, którego rozmiar jest również ograniczony przez stałą.

Niech graf $T(u,v)$ będzie sumą szkieletów grafów $\overrightarrow{S(u)}$ i $\overrightarrow{S(v)}$, tzn. $T(u,v) = S(u) \cup S(v)$. Twierdzenie 1.5.2 implikuje, że jeśli $\text{dist}_G(u,v) = l \leq k$, to istnieje ścieżka o wadze l w $T(u,v)$. Aby znaleźć taką ścieżkę używamy zwykłego algorytmu Dijkstry w grafie $T(u,v)$. Działa on w czasie stałym, ponieważ graf $T(u,v)$ ma ograniczony rozmiar. Niech P będzie najkrótszą ścieżką łączącą u i v w $T(u,v)$ znalezioną przez algorytm Dijkstry. Oczywiście P ma wagę $w(P) \leq l$.

P jest ścieżką w grafie skrótów. Najkrótsza ścieżka łącząca u i v w grafie skrótów może zostać odtworzona z P w czasie stałym jak następuje. Pamiętamy, że podczas obliczania (G,i)-reprezentacji grafu G_i dla każdej krawędzi ab w G_i znajdujemy odpowiadającą jej ścieżkę w G, długości co najwyżej i, oznaczaną przez $p_G(ab)$. Możemy więc każdą krawędź ścieżki P o wadze większej niż 1 zastąpić odpowiadającą jej ścieżką w G. W efekcie otrzymujemy marszrutę M długości nie większej niż $w(P)$ a więc także nie większej niż l. Łatwo widać, że długość tej marszruty wynosi dokładnie l oraz, że każdy wierzchołek pojawia się w marszrucie M dokładnie raz, gdyż jeśli któryś z tych dwóch warunków nie jest spełniony, to odległość między u i v w G jest mniejsza niż l. Oznacza to, że M odpowiada ścieżce długości l łączącej u i v. Jest jasne, że ta ostatnia faza algorytmu znajdowania najkrótszej ścieżki także zajmuje czas stały, przy założeniu, że $k = \mathcal{O}(1)$.

1.5.3 Dokładniejsze szacowanie złożoności

W tym podrozdziale pokażemy w jaki sposób złożoność fazy budowy wyroczni opisanej w podrozdziale 1.5.1 i algorytmu obliczania odpowiedzi na zapytania z podrozdziału 1.5.2 zależą od stałej k.

Dla każdego $i = 1, \ldots k$ oszacujemy z góry grubość (G,i)-reprezentacji grafu G_i, którą oznaczymy przez $t(G_i)$. Z równania (1.1) otrzymujemy:

$$\begin{cases} t(G_1) = 1 \\ t(G_i) \leq \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} t(\overrightarrow{G_j} \odot \overrightarrow{G_{i-j}}) \end{cases}$$

Następnie stosujemy kolejno wniosek 1.4.3 i nierówność między średnią arytmetyczną

i geometryczną:

$$\begin{aligned} t(G_i) &\leq \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} 5 \cdot 3^2 \cdot (2j+1)(2(i-j)+1) t(G_j) t(G_{i-j}) \\ &\leq 45 \cdot \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} \left(\frac{2j+1+2(i-j)+1)}{2} \right)^2 t(G_j) t(G_{i-j}) \\ &\leq 45 \cdot (i+1)^2 \cdot \sum_{j=1}^{\lfloor \frac{i}{2} \rfloor} t(G_j) t(G_{i-j}) \end{aligned}$$

Niech $c = 45 \cdot (k+1)^2$. Ponieważ $i \leq k$, to aby oszacować $t(G_i)$ wystarczy rozwiązać następujące równanie rekurencyjne:

$$\begin{cases} T_1 = 1 \\ T_i = c \cdot \sum_{j=1}^{i-1} T_j T_{i-j} \quad \text{dla } i \geq 2 \end{cases} \tag{1.2}$$

Niech C_n oznacza n-tą liczbę Catalana. Przypomnijmy definicję tych liczb:

$$\begin{cases} C_0 = 1 \\ C_i = \sum_{j=0}^{i-1} C_j C_{i-j-1} \quad \text{dla } i \geq 1 \end{cases} \tag{1.3}$$

Fakt 1.5.3. $T_i = c^{i-1} C_{i-1}$.

Dowód. Przeprowadzimy dowód przez indukcję po i. Oczywiście, $T_1 = 1 = c^0 C_0$. Co więcej, $T_i = c \cdot \sum_{j=1}^{i-1} T_j T_{i-j} = c \cdot \sum_{j=1}^{i-1} c^{j-1} C_{j-1} c^{i-j-1} C_{i-j-1} = c^{i-1} \sum_{j=0}^{i-2} C_j C_{i-j-2} = c^{i-1} C_{i-1}$. ■

W celu zbadania wartości T_n skorzystamy z powszechnie znanego oszacowania asymptotycznego liczb Catalana (zobacz np. [31]):

Fakt 1.5.4. $C_i = \frac{4^i}{\sqrt{\Pi i^3}}(1 + \mathcal{O}(1/i))$.

Ostatecznie otrzymujemy co następuje:

$$t(G_i) = \mathcal{O}((180(k+1)^2)^i) = 2^{\mathcal{O}(i \log k)}. \tag{1.4}$$

Wniosek 1.5.5. *Dla dowolnego $i \leq k$, stopnie wychodzące wierzchołków w grafie G_i są ograniczone przez $3 \cdot t(G_i) = 2^{\mathcal{O}(i \log k)}$ oraz liczba krawędzi grafu G_i nie przekracza $3n \cdot t(G_i) = 2^{\mathcal{O}(i \log k)}$.* ■

Wniosek 1.5.6. *Dla dowolnego n-wierzchołkowego grafu planarnego G, graf skrótów $\overrightarrow{S_k(G)}$ można zbudować w czasie $2^{\mathcal{O}(k \log k)} n$. Podobnie, rozmiar pamięci zajmowanej przez graf skrótów wynosi $2^{\mathcal{O}(k \log k)} n$. Z pomocą grafu skrótów algorytm z podrozdziału 1.5.2 oblicza odpowiedź na zapytanie o najkrótszą ścieżkę w czasie $2^{\mathcal{O}(k \log k)}$.* ■

1.5.4 Kompresja grafu skrótów

W poprzednim podrozdziale pokazaliśmy, że struktura danych $\overrightarrow{S_k(G)}$ zajmuje pamięć rozmiaru $2^{\mathcal{O}(k \log k)}$. W tym miejscu pokażemy, że po skonstruowaniu grafu skrótów można skompresować ten graf tak, aby otrzymany graf miał nie więcej niż $2|E(G)|$ krawędzi, zachowując równocześnie własności grafu skrótów istotne dla znajdowania najkrótszych ścieżek.

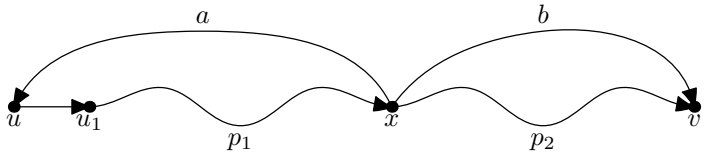
Opiszemy liniowy algorytm kompresji grafu $\overrightarrow{S_k(G)}$. Niech $\overleftrightarrow{G}$ będzie digrafem otrzymanym z G przez zastąpienie każdej krawędzi uv przez parę krawędzi skierowanych (u, v) i (v, u). Początkowo oznaczamy wszystkie krawędzie $\overleftrightarrow{G}$ jako *nieużywane*. Dodatkowo, wraz z każdą krawędzią $(u, v) \in E(\overrightarrow{G_1})$ przechowujemy wskaźniki do krawędzi (u, v) i (v, u) w $\overleftrightarrow{G}$. Zauważmy, że pozwala to na dostęp do dowolnej krawędzi grafu $\overleftrightarrow{G}$ w czasie stałym, ponieważ graf $\overrightarrow{G_1}$ jest 3-zorientowany. Nasz algorytm, dla każdej krawędzi $e = (x, y)$ grafu skrótów odczytuje odpowiadającą jej ścieżkę $p_G(e) = xx_1 \cdots y$ i oznacza krawędź (x, x_1) jako *używaną* w grafie $\overleftrightarrow{G}$. Następnie tworzymy nowy graf, do którego wstawiamy tylko te krawędzie grafu $\overleftrightarrow{G}$, które zostały oznaczone jako używane. Tak utworzony graf będziemy oznaczać przez $\overrightarrow{\sigma_k(G)}$. Jeżeli opisany algorytm zastosujemy jedynie do krawędzi o wadze nie przekraczającej i, to otrzymamy w rezultacie graf, który będziemy oznaczać przez $\overrightarrow{\sigma_i(G)}$. Widzimy, że graf $\overrightarrow{\sigma_k(G)}$ ma nie więcej niż $2|E(G)|$ krawędzi, niezależnie od wartości k. Co więcej, dla dowolnego wierzchołka v grafu G, $\mathrm{outdeg}_{\overrightarrow{\sigma_k(G)}}(v) \leq \mathrm{outdeg}_{\overrightarrow{S_k(G)}}(v)$.

Wniosek 1.5.7. *Dla dowolnego grafu planarnego G, graf $\overrightarrow{\sigma_k(G)}$ jest $\mathcal{O}(1)$-zorientowany.*

Aby pokazać, że graf $\overrightarrow{\sigma_k(G)}$ może być używany do wyszukiwania najkrótszych ścieżek podobnie jak graf skrótów, skorzystamy z następującego lematu.

Lemat 1.5.8. *Niech (u, v) będzie krawędzią grafu $\overrightarrow{G_i}$ odpowiadającą ścieżce $p_G(e) = u \ldots v_1 v$ o długości i. Wtedy graf $\overrightarrow{\sigma_i(G)}$ zawiera ścieżkę od u do v o długości i oraz ścieżkę od v_1 do u o długości $i - 1$.*

Dowód. Użyjemy indukcji względem i. Dla $i = 1$ wystarczy zauważyć, że $\overrightarrow{\sigma_1(G)} = \overrightarrow{G_1}$. Załóżmy, że $i > 1$. Pamiętamy, że istnieje wierzchołek x oraz krawędzie $(x, u) \in E(\overrightarrow{G_a})$, $(x, v) \in E(\overrightarrow{G_b})$ takie, że $a + b = i$. Co więcej, ponieważ $p_G(uv)$ ma długość i więc $p_G(xu) \cup p_G(xv) = p_G(uv)$. Niech u_1 będzie sąsiadem u na ścieżce $p_G(uv)$. Wtedy $p_G(xu) = x \ldots u_1 u$. Z założenia indukcyjnego w grafie $\overrightarrow{\sigma_a(G)}$ istnieje ścieżka p_1 o długości $a-1$ od u_1 do x (patrz rysunek 1.9). Podobnie z założenia indukcyjnego dostajemy ścieżkę p_2 o długości b od x do v w $\overrightarrow{\sigma_b(G)}$. Ponieważ $\overrightarrow{\sigma_a(G)} \cup \overrightarrow{\sigma_b(G)} \subseteq \overrightarrow{\sigma_i(G)}$ oraz $(u, u_1) \in E(\overrightarrow{\sigma_i(G)})$, szukaną ścieżką od u do v jest $\{(u, u_1)\} \cup p_1 \cup p_2$.



Rysunek 1.9: Dowód lematu 1.5.8.

Analogicznie pokazujemy istnienie ścieżki od v_1 do u. Korzystamy z tego, że graf $\overrightarrow{\sigma_b(G)}$ zawiera ścieżkę o długości $b-1$ od v_1 do x oraz graf $\overrightarrow{\sigma_a(G)}$ zawiera ścieżkę o długości a od x do u. Suma tych dwóch ścieżek daje szukaną ścieżkę od v_1 do u długości $i-1$. ■

Twierdzenie 1.5.9. *Dla dowolnego n-wierzchołkowego grafu planarnego G i liczby naturalnej k można w czasie $2^{\mathcal{O}(k \log k)}n$ znaleźć graf $\overrightarrow{\sigma_k(G)}$ – bi-orientację G – taki, że*

(i) stopnie wychodzące wierzchołków w $\overrightarrow{\sigma_k(G)}$ są ograniczone przez $2^{\mathcal{O}(k \log k)}$,

(ii) dla dowolnej pary wierzchołków u, v, odległych od siebie co najwyżej o k w grafie G, istnieje w G ścieżka p o długości $\mathrm{dist}_G(u,v)$ łącząca u i v oraz wierzchołek $x \in V(p)$ taki, że krawędzie p tworzą dwie ścieżki w $\overrightarrow{\sigma_k(G)}$: ścieżkę od u do x i ścieżkę od v do x.

Twierdzenie 1.5.9 wynika natychmiast z twierdzenia 1.5.2 i lematu 1.5.8. Implikuje ono, że zapytania o najkrótszą ścieżkę łączącą u i v możemy przetwarzać korzystając z grafu $\overrightarrow{\sigma_k(G)}$, podobnie jak w algorytmie z podrozdziału 1.5.2 korzystającego z grafu skrótów. Co więcej, otrzymujemy algorytm prostszy: teraz krawędzie nie mają wag, a więc wystarczy użyć algorytmu BFS w grafie $T(u,v)$. Dla $k = \mathcal{O}(1)$ algorytm działa oczywiście w czasie $\mathcal{O}(1)$.

Nie możemy w tym miejscu uniknąć interesującego pytania: czy możliwe jest znalezienie grafu $\overrightarrow{\sigma_k(G)}$ bez wcześniejszego obliczania grafu skrótów, korzystając jedynie z pamięci rozmiaru $\mathcal{O}(n)$ dla dowolnego k? Podejrzewamy, że jest to możliwe przy równoczesnym zwiększeniu złożoności czasowej w zależności od k.

1.6 Środowisko dynamiczne

Unikalną cechą opisywanej w tym rozdziale struktury danych jest to, że można ją dostosować do pracy w środowisku w pełni dynamicznym. Oznacza to, że możliwe jest szybkie aktualizowanie informacji przechowywanych przez wyrocznię po każdej operacji wstawienia lub usunięcia krawędzi grafu. Pokażemy, że po usunięciu z grafu G dowolnej krawędzi lub dowolnego wierzchołka, graf skrótów $\overrightarrow{S_k(G)}$ może zostać

zaktualizowany w czasie $\mathcal{O}(1)$. Opiszemy także, w jaki sposób można ukrywać i odkrywać krawędzie i wierzchołki grafu w dalszym ciągu potrzebując jedynie czasu stałego do wykonania koniecznych modyfikacji w grafie skrótów. W dalszej kolejności opiszemy bardziej złożoną operację odświeżania wyroczni po wstawieniu krawędzi do grafu. Czas potrzebny na jej wykonanie można oszacować przez $\mathcal{O}(\log^k n)$, lecz pytanie, czy jest to dokładne szacowanie, pozostaje nadal otwarte. W końcu zajmiemy się problemem odświerzania struktury wyroczni po utożsamieniu dwóch wierzchołków. Zdecydowaliśmy się rozważyć także tę operację, gdyż występuje ona bardzo często w algorytmach przetwarzających grafy planarne. W szczególności, algorytm 3-kolorowania grafów planarnych opisany w rozdziale 2. zawiera operacje utożsamiania wierzchołków i korzysta z wyroczni krótkich ścieżek. Warto równocześnie zaznaczyć, że algorytm przetwarzania zapytań podlega jedynie niewielkim zmianom i w dalszym ciągu działa w czasie $\mathcal{O}(1)$.

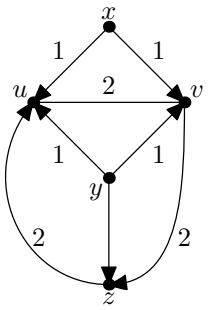
1.6.1 Usuwanie krawędzi

Wyobraźmy sobie, że usunięto krawędź $e = uv$ z grafu G. Zastanówmy się, które elementy grafu skrótów powinny zostać usunięte lub zmienione. Przypomnijmy sobie kolejne fazy budowania grafu skrótów. Na początku konstruowany jest graf $\overrightarrow{G_1}$ – orientacja grafu G. A więc w grafie $\overrightarrow{G_1}$ istnieje krawędź łącząca u i v, powiedzmy (u, v). Widzimy, że ta krawędź powinna zostać usunięta. Rozważmy jednak sytuację, w której z wierzchołka u w grafie $\overrightarrow{G_1}$ wychodzi jeszcze jedna krawędź, powiedzmy (u, w). Wtedy krawędź (u, v) i krawędź (u, w) gwarantują istnienie krawędzi vw w grafie G_2. Ta krawędź też powinna zostać usunięta, ale tylko wtedy, gdy nie ma innego, oprócz u, wierzchołka wspierającego vw. Usunięcie vw może w dalszej kolejności spowodować konieczność usuwania niepotrzebnych krawędzi z G_3, itd. Otrzymujemy więc cały zbiór krawędzi do usunięcia z grafów $\overrightarrow{G_1}, \overrightarrow{G_2}, \ldots, \overrightarrow{G_k}$. Musimy odpowiedzieć sobie na pytania: jak liczny może być ten zbiór i jak efektywnie znajdować krawędzie do usunięcia z grafu skrótów.

Rozszerzenie grafu skrótów

Zacznijmy od wprowadzenia kilku definicji. Niech $e_1 = (x, v)$ i $e_2 = (x, w)$ będą odpowiednio krawędziami grafów $\overrightarrow{G_i}$ i $\overrightarrow{G_j}$. Niech $e = vw$ będzie krawędzią grafu G_{i+j}. Powiemy, że krawędzie e_1 i e_2 są *rodzicami* krawędzi e oraz że krawędź e jest *dzieckiem* e_1 i e_2. Parę $\{e_1, e_2\}$ będziemy nazywać *parą rodziców* krawędzi e oraz powiemy, że e jest *wspólnym dzieckiem* krawędzi e_1 i e_2. Zauważmy, że krawędź może mieć więcej niż tylko jedną parę rodziców. Dla przykładu rozważmy rysunek 1.10. Krawędź uv grafu G_2 ma dwie pary rodziców: $\{(x, u), (x, v)\}$ oraz $\{(y, u), (y, v)\}$. Z drugiej strony, krawędź (u, y) grafu $\overrightarrow{G_1}$ ma dwoje dzieci: uv i uz. Dla uproszczenia, będziemy mówić, że e_1 i e_2 są rodzicami krawędzi (v, w) grafu $\overrightarrow{G_{i+j}}$, jeżeli e_1 i e_2 są rodzicami krawędzi

vw grafu G_{i+j}. Ma miejsce następujący fakt:



Rysunek 1.10: Rodzice i dzieci

Fakt 1.6.1. *Niech $\overrightarrow{S_k(G)}$ będzie grafem skrótów dla grafu planarnego G. Wtedy liczba dzieci dowolnej krawędzi $e = (v, w) \in E(\overrightarrow{S_k(G)})$ jest ograniczona przez stałą.*

Dowód. Oznaczmy przez Δ największy ze stopni wychodzących wierzchołków grafu skrótów (pamiętamy, że $\Delta = \mathcal{O}(1)$). Liczba krawędzi wychodzących z v różnych od e nie przekracza $\Delta - 1$. To pociąga za sobą, że liczba dzieci e także nie przekracza $\Delta - 1 = \mathcal{O}(1)$. ■

W celu efektywnego wykonywania operacji usuwania krawędzi wzbogacimy graf skrótów o dodatkowe informacje. Podczas konstruowania $\overrightarrow{S_k(G)}$, dla każdej krawędzi $e \in E(\overrightarrow{S_k(G)})$ zapamiętujemy dodatkowo:

(i) listę $P(e)$ wszystkich par rodziców krawędzi e,

(ii) listę $C(e)$ złożoną z par (c, p), w których c jest wspólnym dzieckiem e i pewnej krawędzi f oraz p jest wskaźnikiem do pary $\{e, f\}$ przechowywanej na liście $P(c)$.

Jest jasne, że konieczność przechowywania powyższych informacji nie zmienia asymptotycznej złożoności czasowej i pamięciowej fazy budowy wyroczni.

Algorytm usuwania krawędzi

Opiszemy rekurencyjny algorytm DeleteShortcut(e) służący do usuwania krawędzi e z grafu $\overrightarrow{G_t}$. Dla każdej pary $(c, p) \in C(e)$ wykonujemy kolejno następujące operacje: usuń z listy $P(c)$ parę $\{e, f\}$ wskazywaną przez wskaźnik p, usuń z listy $C(f)$ parę (c, p). Jeśli lista $P(c)$ stanie się pusta, usuwamy krawędź c rekurencyjnie (patrz pseudokod algorytmu 1.6.1). Ponieważ każda krawędź ma $\mathcal{O}(1)$ dzieci (fakt 1.6.1)

Algorytm 1.6.1 Usuwanie krawędzi uv z grafu skrótów

```
1: procedure DeleteShortcut(e)
2:     for each (c, p) ∈ C(e) do
3:         {e, f} ← para wskazywana przez wskaźnik p
4:         P(c) ← P(c) \ {e, f}
5:         C(f) ← C(f) \ (c, p)
6:         if P(c) = ∅ then
7:             DeleteShortcut(c)
8:     E(S_k(G)) ← E(S_k(G)) \ {e}
```

oraz głębokość rekursji jest ograniczona przez stałą k, cała operacja usunięcia krawędzi zajmuje czas $\mathcal{O}(1)$. W celu zaktualizowania grafu skrótów po usunięciu krawędzi z grafu G wystarczy usunąć jej zorientowaną wersję w $\overrightarrow{G_1}$ za pomocą powyższego algorytmu.

Zapytania o najkrótszą ścieżkę w środowisku dynamicznym

Pamiętamy, że podczas obsługiwania zapytania o najkrótszą ścieżkę za pomocą algorytmu z podrozdziału 1.5.2 potrzebujemy znaleźć ścieżkę w G odpowiadającą danej krawędzi w grafie skrótów. W środowisku statycznym wystarczyło dla każdej krawędzi e przechowywać jedną, ustaloną na zawsze ścieżkę $p_G(e)$. Jak się mogliśmy przekonać, w środowisku dynamicznym istnieje możliwość, że pewna krawędź ścieżki $p_G(e)$ zostanie usunięta, ale krawędź e pozostanie w grafie skrótów. Graf G zawiera wtedy inną ścieżkę długości nie przekraczającej wagi e. Dlatego w środowisku dynamicznym znajduje się ścieżkę odpowiadającą danej krawędzi e wykorzystując strukturę rodziców zapamiętaną w listach $P(\cdot)$. W tym celu zaczynamy od ścieżki $p = e$. Następnie, dopóki istnieje w p krawędź o wadze większej niż 1, zastępujemy ją dowolną parą jej rodziców. Jest jasne, że liczba takich zamian nie przekracza $k - 1$, a więc cała operacja odczytania ścieżki odpowiadającej krawędzi e wykonuje się w czasie $\mathcal{O}(1)$, nie zmieniając asymptotycznej złożoności czasowej algorytmu przetwarzania zapytań.

1.6.2 Usuwanie wierzchołków

Aby usunąć zadany wierzchołek v z grafu, najbardziej naturalnym podejściem wydaje się usunięcie kolejno wszystkich incydentnych z nim krawędzi. Takiej operacji nie można jednak wykonać w sposób efektywny, ponieważ incydentnych krawędzi może być $\Omega(n)$. Zamiast tego, usuwamy z grafu skrótów wszystkie krawędzie wychodzące z v za pomocą procedury rekurencyjnej opisanej w podrozdziale 1.6.1. Ta operacja zajmuje jedynie czas stały, ponieważ stopień wychodzący v w grafie $\overrightarrow{S_k(G)}$ jest ograniczony przez stałą. Następnie oznaczamy v jako wierzchołek usunięty. Na tym kończy się algorytm usuwania wierzchołka. Aby algorytm przetwarzania zapytań działał teraz

poprawnie wystarczy, żeby ignorował krawędzie wchodzące do usuniętych wierzchołków (albo, równoważnie, każda taka krawędź może być usuwana, jak tylko zostanie znaleziona przez algorytm przetwarzania zapytań).

1.6.3 Wstawianie krawędzi: przypadek $k = 1$

W tym i kolejnym podrozdziale zajmiemy się odpowiedzią na pytanie w jaki sposób zaktualizować graf $\overrightarrow{S_k(G)}$ po wstawieniu krawędzi e do grafu G. Naszym celem jest otrzymanie grafu, który mógłby zostać skonstruowany przez algorytm budowania grafu skrótów z grafem $G \cup \{e\}$ na wejściu.

Nasze rozważania rozpoczniemy od przypadku $k = 1$. Wtedy graf skrótów jest po prostu pewną $\mathcal{O}(1)$-orientacją $\overrightarrow{G_1}$ grafu G. Załóżmy, że chcemy żeby stopnie wychodzące wierzchołków w $\overrightarrow{G_1}$ nie przekraczały stałej D. Widzimy, że jeśli chcemy wstawić do grafu G krawędź uv oraz $\text{outdeg}(u) = \text{outdeg}(v) = D$ konieczna jest zmiana orientacji pewnej liczby krawędzi w $\overrightarrow{G_1}$.

Posłużymy się wynikiem Gertha Brodala i Rolfa Fagerberga [9]. Pokazują oni, w jaki sposób utrzymywać D-orientację grafu o lesistości ograniczonej przez stałą, $D = \mathcal{O}(1)$. Niech G będzie grafem o lesistości a i niech $D > 2a$. Załóżmy, że do grafu G wstawiono krawędź uv. Zaczynamy od wstawienia (u, v) do orientacji $\overrightarrow{G}$. Jeśli okaże się, że $\text{outdeg}_{\overrightarrow{G}}(u) = D + 1$, bierzemy kolejne wierzchołki stopnia większego niż D, dopóki takie istnieją, i dla każdego z nich zmieniamy orientację wszystkich krawędzi wychodzących (patrz pseudokod algorytmu 1.6.2). Przedstawiony algorytm przywracający D-orientację będziemy nazywać algorytmem REORIENT. Metodę wstawiania krawędzi wykorzystującą algorytm REORIENT nazwiemy algorytmem INSERT (tak jak w pseudokodzie 1.6.2). Po usunięciu krawędzi nie wykonuje się żadnych zmian orientacji krawędzi.

Algorytm 1.6.2 Utrzymywanie D-orientacji grafu G po wstawieniu krawędzi uv

```
1:  procedure INSERT(u, v)
2:      E(→G) ← E(→G) ∪ {(u, v)}
3:      if outdeg(u) = D + 1 then
4:          REORIENT (u)
5:  procedure REORIENT(w)
6:      S ← {w}
7:      while S ≠ ∅ do
8:          x ← POP(S)
9:          for each (x, y) ∈ E(→G) do
10:             Zmień orientację krawędzi (x, y) na (y, x).
11:             if outdeg(y) = D + 1 then
12:                 PUSH(S, y)
```

Autorzy wykazują, że gdy używamy opisanej metody dla początkowo pustego grafu o lesistości ograniczonej przez stałą, zamortyzowana liczba zmian orientacji krawędzi wynosi $\mathcal{O}(1)$ na jedną operację wstawienia krawędzi i $\mathcal{O}(\log n)$ na jedną operację usunięcia krawędzi. Taki sposób podania złożoności algorytmu podyktowany został zapewne tym, że gdy graf jest początkowo pusty, liczba operacji usuwania krawędzi nie przekracza liczby operacji wstawienia krawędzi. Niemniej jednak łatwo zmodyfikować oryginalną analizę, aby otrzymać inny, w naszym przypadku bardziej korzystny wynik. Poniższy lemat jest uogólnieniem lematu 1 z pracy [9]. Jego dowód pozostaje prawie niezmieniony, zamieszczamy go jednak dla pełności.

Lemat 1.6.2. *Niech σ będzie ciągiem operacji wstawiania i usuwania krawędzi wykonywanych na pewnym grafie. Załóżmy, że po wstawieniu krawędzi używamy algorytmu* Reorient *do utrzymywania D-orientacji tego grafu. Oznaczmy przez H_0 graf początkowy i przez $\overrightarrow{H_0}$ jego początkową orientację. Załóżmy, że graf $\overrightarrow{H_0}$ jest δ-zorientowany, dla pewnego δ takiego, że $D \geq 2\delta$. Niech H_i będzie grafem otrzymanym po i-tej operacji i niech t będzie liczbą wstawień.*

Jeśli istnieje ciąg δ-orientacji $\overrightarrow{H_0}, \overrightarrow{H_1}, \ldots, \overrightarrow{H_{|\sigma|}}$, w którym całkowita liczba zmian orientacji krawędzi nie przekracza r, to algorytm Reorient *wykonuje nie więcej niż*

$$(t+r)\frac{D+1}{D+1-2\delta}$$

zmian orientacji krawędzi dla ciągu operacji σ.

Dowód. Krawędź $uv \in E(H_i)$ nazwiemy *zgodną*, gdy jej orientacja ustalona przez algorytm Insert po wykonaniu i-tej operacji ciągu σ, zgadza się z jej orientacją w grafie $\overrightarrow{H_i}$. W przeciwnym przypadku krawędź uv nazwiemy *niezgodną*. Aby oszacować liczbę zmian orientacji wykonanych przez algorytm Insert rozważmy następujący, nieujemny potencjał:

$$\Phi(i) = \text{liczba niezgodnych krawędzi w zbiorze } E(H_i).$$

$\Phi(i)$ oznacza tu wartość potencjału po wykonaniu i-tej operacji ciągu σ. Oczywiście $\Phi(0) = 0$. Każda z t operacji wstawienia krawędzi i każda z r zmian orientacji krawędzi w ciągu δ-orientacji $\overrightarrow{H_0}, \overrightarrow{H_1}, \ldots, \overrightarrow{H_{|\sigma|}}$ podwyższa potencjał Φ co najwyżej o 1. Usuwanie krawędzi nie może zwiększyć Φ. Rozważmy pojedynczą iterację pętli **while**, w której zmieniają się orientacje krawędzi wychodzących z pewnego wierzchołka x o stopniu wychodzącym $D+1$. Wtedy może być co najwyżej δ zgodnych krawędzi wychodzących z x. Stąd podczas zmiany orientacji krawędzi wychodzących, co najwyżej δ krawędzi wychodzących może stać się niezgodnymi, natomiast pozostałych $D+1-\delta$ dotąd niezgodnych krawędzi wychodzących, po zmianie orientacji staje się zgodnymi. Z tego wynika, że podczas każdej takiej iteracji pętli **while** potencjał Φ maleje co najmniej o $D+1-2\delta$. Widzimy, że na wzrost potencjału wpływają jedynie wstawienia i zmiany orientacji krawędzi w ciągu $\overrightarrow{H_0}, \overrightarrow{H_1}, \ldots, \overrightarrow{H_{|\sigma|}}$, a więc suma wzrostów Φ nie

przekracza $t+r$. Stąd liczba iteracji pętli **while** nie przekracza $(t+r)/(D+1-2\delta)$. Wtedy liczba takich sytuacji, gdy zgodna krawędź staje się niezgodna wynosi nie więcej niż $\delta(t+r)/(D+1-2\delta)$, co pociąga za sobą, że liczba sytuacji, w których niezgodna krawędź staje się zgodna nie przekracza $t+r+\delta(t+r)/(D+1-2\delta)$. Razem otrzymujemy nie więcej niż $(t+r)(1+2\delta/(D+1-2\delta)) = (t+r)(D+1)/(D+1-2\delta)$ zmian orientacji krawędzi. ■

Lemat 1.6.3 (zobacz lemat 2 w [9]). *Niech $H=(V,E)$ będzie grafem o lesistości nie przekraczającej a i niech $\overrightarrow{H}$ będzie pewną orientacją grafu H. Niech $\delta > a$. Jeżeli stopień wychodzący pewnego wierzchołka u wynosi co najmniej δ, to w grafie $\overrightarrow{H}$ istnieje ścieżka o długości co najwyżej $\lceil \log_{\delta/a} |V| \rceil$ o początku w wierzchołku u i końcu w wierzchołku o stopniu wychodzącym mniejszym od δ.* ■

Wniosek 1.6.4. *Dla dowolnego grafu H o lesistości a można skonstruować jego $(3a-1)$-orientację $\overrightarrow{H}$ w czasie $\mathcal{O}(|V(H)|+|E(H)|)$, dodając kolejne krawędzie do $\overrightarrow{H}$ za pomocą algorytmu* Insert.

Dowód. Dla $i = 1,\ldots,|E(H)|$, niech H_i będzie grafem po i-tej operacji wstawienia krawędzi oraz niech H_0 będzie grafem o zbiorze wierzchołków $V(H)$ i pustym zbiorze krawędzi. Skonstruujemy pewien ciąg a-orientacji $\overrightarrow{H_0}, \overrightarrow{H_1}, \ldots, \overrightarrow{H_{|E(H)|}}$. Niech $\overrightarrow{H_{|E(H)|}}$ będzie a-orientacją grafu $H_{|E(H)|}$, której istnienie gwarantuje wniosek 1.3.4. Dla każdego $i \in \{0,\ldots,|E(H)|-1\}$, graf $\overrightarrow{H_i}$ otrzymujemy z grafu $\overrightarrow{H_{i+1}}$ przez usunięcie jednej krawędzi. W ten sposób skonstruowaliśmy ciąg orientacji nie zawierający ani jednej zmiany orientacji krawędzi. Kładziemy $\delta = a$ i $D = 3a-1$. Z lematu 1.6.2 wnioskujemy, że całkowita liczba zmian orientacji krawędzi wykonanych przez algorytm Insert podczas budowy $(3a-1)$-orientacji grafu H nie przekracza $(|E(H)|+0)\frac{3a-1+1}{3a-1+1-2a} = 3|E(H)|$. To implikuje, że faza budowy $\overrightarrow{H}$ zajmuje czas liniowy. ■

Wniosek 1.6.5. *Niech H będzie dowolnym n-wierzchołkowym grafem o lesistości a oraz niech $\overrightarrow{H}$ będzie jego początkową δ-orientacją, $\delta \geq 2a$. Dla dowolnego ciągu p wstawień i q usunięć krawędzi zastosowanego do grafu H takiego, że lesistość grafu po każdej operacji nie przekracza a, algorytm* Reorient *uruchomiony po każdym wstawieniu krawędzi utrzymuje $(3\delta-1)$-orientację, wykonując $\mathcal{O}(p\log n)$ zmian orientacji krawędzi.*

Dowód. Dla $i = 1,\ldots,p+q$, niech H_i będzie grafem po wykonaniu i-tej operacji wstawienia/usunięcia krawędzi i niech $H_0 = H$, $\overrightarrow{H_0} = \overrightarrow{H}$. Dla dowolnego $i > t$ zbudujemy δ-orientację $\overrightarrow{H_i}$. Jeżeli i-tą operacją ciągu σ było usunięcie krawędzi, otrzymujemy orientację $\overrightarrow{H_i}$ z orientacji $\overrightarrow{H_{i-1}}$ przez usunięcie odpowiedniej krawędzi. Załóżmy teraz, że i-tą operacją było wstawienie krawędzi uv. Przyjmijmy chwilowo $\overrightarrow{H_i} = \overrightarrow{H_{i-1}}$. Jeżeli $\mathrm{outdeg}_{\overrightarrow{H_i}}(u) = \delta$, to na podstawie lematu 1.6.3 w grafie $\overrightarrow{H_i}$ istnieje ścieżka p o długości co najwyżej $\lceil \log_{\delta/a} n \rceil$, o początku w u i końcu w wierzchołku w o stopniu

wychodzącym mniejszym od δ. Zmieniamy orientację wszystkich krawędzi na ścieżce p, otrzymując δ-orientację grafu H_{i-1} taką, że $\mathrm{outdeg}(u) = \delta - 1$. Teraz wystarczy już jedynie dodać krawędź (u, v), otrzymując δ-orientację grafu H_i. Zauważmy, że liczba zmian orientacji krawędzi w otrzymanym ciągu nie przekracza $p \cdot \lceil \log_{\delta/a} n \rceil$.

Kładziemy $D = 3\delta - 1$. Z lematu 1.6.2 otrzymujemy, że całkowita liczba zmian orientacji krawędzi wykonanych przez algorytm REORIENT podczas wykonywania p wstawień nie przekracza $(p + p \cdot \lceil \log_{\delta/a} n \rceil) \frac{D+1}{D+1-2\delta} \leq 3p(1 + \lceil \log_2 n \rceil) = O(p \log n)$. ■

Powyższe wnioski oznaczają, że jeżeli użyjemy algorytmu INSERT do wstawiania krawędzi do grafu $\overrightarrow{G_1}$ (także podczas fazy konstruowania orientacji dla wejściowego grafu) to złożoność obliczania początkowej $\mathcal{O}(1)$-orientacji pozostanie liniowa, natomiast aktualizowanie orientacji po wstawieniu krawędzi będzie wykonywane w zamortyzowanym czasie $\mathcal{O}(\log n)$. Warto jednak zaznaczyć, że nie jest wiadome, czy logarytmiczne oszacowanie na zamortyzowany czas wstawienia jest dokładne, a jedynym znanym dolnym ograniczeniem jest $\Omega(1)$. Wspomnijmy jeszcze, że jeśli lesistość grafu G nie jest znana *a priori*, algorytm może mimo to działać zgodnie z oczekiwaniami, podwajając wartość D gdy wykonywanych jest więcej zmian orientacji niż wynika z przedstawionych oszacowań.

1.6.4 Budowanie grafu skrótów

W kolejnym podrozdziale opiszemy algorytm INSERTSHORTCUT służący do aktualizowania grafu k-skrótów po operacji wstawienia krawędzi do grafu G. Dysponując takim narzędziem moglibyśmy zrezygnować z używania algorytmu z punktu 1.5.1 do budowania grafu skrótów, zastępując go wykonaniem ciągu wstawień na początkowo pustym grafie. Jak się jednak okaże, algorytm INSERTSHORTCUT działa w zamortyzowanym czasie $\mathcal{O}(\log^k n)$, a więc faza budowania grafu k-skrótów zajmowałaby wtedy czas $\mathcal{O}(n \log^k n)$.

Aby czas konstruowania wyroczni pozostał liniowy, należy postępować ostrożniej. Algorytm INSERTSHORTCUT dokonuje zmian we wszystkich grafach $\overrightarrow{G_1}, \ldots, \overrightarrow{G_k}$. W ten sposób, przy podejściu opisanym powyżej podczas budowania grafu skrótów wszystkie grafy $\overrightarrow{G_i}$ są tworzone równocześnie. Aby poprawić oszacowanie złożoności czasowej, wystarczy budować grafy $\overrightarrow{G_1}, \ldots, \overrightarrow{G_k}$ po kolei. Dokładniej, zaczynamy od zbudowania grafu $\overrightarrow{G_1}$ tak, jak to było opisane w poprzednim podrozdziale. Znając graf $\overrightarrow{G_1}$ wiemy, które krawędzie wstawić do G_2. Budujemy więc graf G_2, utrzymując $\mathcal{O}(1)$-orientację w $\overrightarrow{G_2}$ za pomocą algorytmu REORIENT. Analogicznie, dla każdego $i < k$, po zbudowaniu grafów $\overrightarrow{G_1}, \ldots, \overrightarrow{G_i}$ znamy wszystkie krawędzie grafu G_{i+1}, zgodnie z równością 1.1. Każdą taką krawędź wstawiamy do G_{i+1}, utrzymując $\mathcal{O}(1)$-orientację w $\overrightarrow{G_{i+1}}$ za pomocą algorytmu REORIENT. Nazwijmy ten sposób budowania grafu $\overrightarrow{S_k(G)}$ algorytmem BUILDDYNAMIC. W następnym podrozdziale wykażemy, że złożoność czasowa tego algorytmu jest liniowa.

Oczywiście możliwe jest także korzystanie w środowisku dynamicznym z grafu skrótów zbudowanego z użyciem algorytmu z podrozdziału 1.5.1.

1.6.5 Wstawianie krawędzi: przypadek ogólny

Opiszemy teraz ogólny algorytm aktualizowania grafu skrótów $\overrightarrow{S_k(G)}$ po wstawieniu krawędzi, działający w czasie zamortyzowanym $\mathcal{O}(\log^k n)$.

Przedstawimy rekurencyjny algorytm InsertShortcut(uv,i), gdzie $i \in \{1, \ldots, k\}$. Zaczynamy od uruchomienia algorytmu Insert(uv) w grafie $\overrightarrow{G_i}$. Wprowadzamy jednak istotną zmianę: przed zmianą orientacji dowolnej krawędzi (x, y) uruchamiamy algorytm DeleteShortcut(xy) opisany w podrozdziale 1.6.1. Co więcej, kiedy dowolna krawędź (x, y) pojawia się w grafie G_i, zarówno po jej wstawieniu, jak i na skutek zmiany orientacji krawędzi (y, x), dla każdej innej krawędzi $(x, z) \in E(\overrightarrow{S_k(G)})$ o wadze $j \leq k - i$ wstawiamy yz do grafu G_{i+j} uruchamiając rekurencyjnie algorytm InsertShortcut(yz,$i + j$). To kończy opis algorytmu. Oczywiście, aby zaktualizować graf skrótów po wstawieniu krawędzi uv do G, wystarczy uruchomić InsertShortcut(uv,1). Poniższy fakt wyraża poprawność algorytmu InsertShortcut.

Fakt 1.6.6. *Niech $S_k(G)$ będzie grafem skrótów dla grafu G. Wtedy graf otrzymany na skutek uruchomienia algorytmu* InsertShortcut(uv,1) *jest grafem skrótów grafu $G \cup \{uv\}$.*

Dowód. Teza wynika natychmiast z tego, że algorytm InsertShortcut zachowuje równość (1.1) ze strony 38. ■

Lemat 1.6.7. *Załóżmy, że zbudowano $\mathcal{O}(1)$-zorientowany graf skrótów dla pewnego grafu G. Dla dowolnego $i \leq k$, po dowolnym ciągu usunięć i wstawień krawędzi wykonanych za pomocą algorytmów* DeleteShortcut *i* InsertShortcut*, graf $\overrightarrow{G_i}$ jest $\mathcal{O}(1)$-zorientowany i ma (G, i)-reprezentację grubości $\mathcal{O}(1)$.*

Dowód. Przeprowadzimy dowód przez indukcję po i. Prawdziwość tezy dla $i = 1$ jest oczywista. Przejdźmy do kroku indukcyjnego. Rozważamy sytuację po wykonaniu dowolnego ciągu usunięć i wstawień krawędzi. Załóżmy że lemat jest prawdziwy dla dowolnego $j < i$, tzn. graf $\overrightarrow{G_j}$ jest d_j-zorientowany i ma (G, j)-reprezentację grubości t_j, gdzie $d_j = \mathcal{O}(1)$ oraz $t_j = \mathcal{O}(1)$. Wniosek 1.4.3 pociąga za sobą, że dla dowolnych a, b takich, że $a + b = i$, graf $G_a \odot G_b$ ma $(G, a + b)$-reprezentację grubości $5 \cdot d_a \cdot d_b \cdot t_a \cdot t_b \cdot (2a + 1)(2b + 1) = \mathcal{O}(1)$. Ponieważ równość (1.1) jest spełniona, pociąga to za sobą, że graf G_i ma (G, i)-reprezentację grubości $t = \mathcal{O}(1)$. Dzięki temu, że lesistość grafu planarnego nie przekracza 3 (patrz fakt 1.3.2), graf G_i ma lesistość co najwyżej $3t = \mathcal{O}(1)$. Ponieważ rozważyliśmy dowolny ciąg operacji, wiemy, że po każdej z nich lesistość nie przekraczała pewnej wartości $a_i = \mathcal{O}(1)$. Załóżmy, że przed wykonaniem

ciągu wstawień i usunięć graf G_i był d_i-zorientowany. Wiemy, że $d_i = \mathcal{O}(1)$. Na podstawie wniosku 1.6.5 widzimy, że algorytm InsertShortcut utrzymuje stopnie wychodzące w grafie $\overrightarrow{G_i}$ ograniczone przez $3\max\{2a_i, d_i\} - 1 = \mathcal{O}(1)$. ■

Wniosek 1.6.8. *W środowisku dynamicznym umożliwiającym wstawianie i usuwanie krawędzi z grafu, czas przetwarzania zapytań o najkrótsze ścieżki pozostaje stały.* ■

Wniosek 1.6.9. *Algorytm* BuildDynamic *buduje graf k-skrótów dla n-wierzchołkowego grafu w czasie $\mathcal{O}(n)$, przy założeniu, że $k = \mathcal{O}(1)$.*

Dowód. Stosujemy lemat 1.6.7 zaczynając od 0-zorientowanego grafu pustego. Wnioskujemy, że dla każdego i liczba krawędzi grafu G_i nie przekracza $\mathcal{O}(n)$. Teza wynika natychmiast z wniosku 1.6.4. ■

Teraz pozostaje już tylko zbadać efektywność algorytmu InsertShortcut.

Lemat 1.6.10. *Dla dowolnego ciągu t wstawień krawędzi do grafu G (być może oddzielonych operacjami usuwania krawędzi) i dla dowolnego $i \leq k$ wykonuje się $\mathcal{O}(t \log^i n)$ zmian orientacji krawędzi w grafie $\overrightarrow{G_i}$, a także $\mathcal{O}(t \log^{i-1} n)$ wstawień/usunięć krawędzi w grafie G_i.*

Dowód. Skorzystamy z indukcji po i. Dla $i = 1$ teza wynika z wniosku 1.6.5. Dla $i > 1$ założenie indukcyjne pociąga za sobą, że dla każdego $j < i$ algorytm Insert wykonuje $\mathcal{O}(t \log^j n)$ zmian orientacji krawędzi i wstawień w grafie $\overrightarrow{G_j}$. Ponieważ stopnie wychodzące w grafie skrótów są ograniczone przez stałą, każda z tych zmian orientacji krawędzi i operacji wstawienia powoduje jedynie $\mathcal{O}(1)$ wstawień i usunięć w G_i. Stąd całkowita liczba wstawień/usunięć w grafie G_i wynosi $\mathcal{O}(t \log^{i-1} n)$. Z lematu 1.6.7 wynika, że graf G_i ma lesistość $\mathcal{O}(1)$, a więc wniosek 1.6.5 implikuje, że tych $\mathcal{O}(t \log^{i-1} n)$ wstawień wymaga $\mathcal{O}(t \log^i n)$ zmian orientacji w grafie $\overrightarrow{G_i}$. ■

Wniosek 1.6.11. *Zamortyzowany czas aktualizowania grafu skrótów po operacji wstawienia krawędzi wynosi $\mathcal{O}(\log^k n)$.* ■

Nasze rozważania kończymy uwagą na temat implementacji algorytmu Insert w sytuacji, gdy umożliwia się usuwanie wierzchołków z grafu. Pamiętamy, że aby zagwarantować efektywne wykonywanie tej operacji, wierzchołki nie są usuwane a jedynie oznaczane jako usunięte. W szczególności, gdy usuwamy wierzchołek v, w grafie pozostają wszystkie krawędzie wchodzące do v. Dlatego algorytm Insert zawsze oblicza stopnie wychodzące wierzchołków od nowa, w czasie $\mathcal{O}(1)$, ignorując (lub od razu usuwając) krawędzie prowadzące do usuniętych wierzchołków.

1.6.6 Utożsamianie wierzchołków: przypadek $k = 1$

Analogicznie jak w podrozdziale 1.6.3, przedstawimy teraz sposób utrzymywania D-orientacji $\overrightarrow{G}$ grafu G po operacji utożsamienia dwóch wierzchołków. Ponownie zakładamy, że zarówno przed, jak i po utożsamieniu lesistość grafu nie przekracza pewnej wartości a, gdzie $D > 2a$. Przyjmujemy też, że dla dowolnego wierzchołka grafu x możemy znaleźć w czasie $\mathcal{O}(\text{indeg}(x))$ wszystkie elementy list sąsiedztwa grafu $\overrightarrow{G}$ odpowiadające krawędziom wchodzącym do tego wierzchołka[4].

Załóżmy, że chcemy utożsamić dwa wierzchołki u i v. Opiszemy algorytm, który nazwiemy GLUE(u,v) – patrz pseudokod 1.6.3. Zakładamy, że $uv \notin E(G)$. Zaczynamy od dołączenia listy sąsiedztwa wierzchołka u w grafie $\overrightarrow{G}$ do listy sąsiedztwa wierzchołka v. Następnie, dla każdej krawędzi wchodzącej do u modyfikujemy odpowiadający jej element listy sąsiedztwa tak, aby reprezentował krawędź wchodzącą do v. Po tej operacji otrzymujemy orientację grafu G już po utożsamieniu u i v. Stopnie wszystkich wierzchołków $\overrightarrow{G}$ nie zmieniły się, z wyjątkiem stopnia wierzchołka v, który mógł przekroczyć D. W takiej sytuacji wykonujemy algorytm REORIENT (v) (patrz pseudokod 1.6.2).

Algorytm 1.6.3 Algorytm GLUE (u,v)

```
1: procedure GLUE(u, v)
2:     for each (u, x) ∈ E(G→) do
3:         E(G→) ← E(G→) \ {(u, x)}
4:         E(G→) ← E(G→) ∪ {(v, x)}        ▷ Przenosimy element listy sąsiedztwa
5:     for each (x, u) ∈ E(G→) do
6:         E(G→) ← E(G→) \ {(x, u)}
7:         E(G→) ← E(G→) ∪ {(x, v)}        ▷ Przenosimy element listy sąsiedztwa
8:     if outdeg(v) > D then
9:         REORIENT(v)
```

Aby przeanalizować złożoność czasową algorytmu GLUE, należy uogólnić lemat 1.6.2.

Lemat 1.6.12. *Niech σ będzie ciągiem operacji wstawiania i usuwania krawędzi, a także utożsamiania wierzchołków, wykonywanych na pewnym grafie. Załóżmy, że używamy algorytmów* INSERT *i* GLUE *do utrzymywania D-orientacji tego grafu, odpowiednio po wstawieniu krawędzi i po utożsamieniu pary wierzchołków. Oznaczmy przez H_0 graf początkowy i przez $\overrightarrow{H_0}$ jego początkową orientację. Załóżmy, że graf $\overrightarrow{H_0}$ jest δ-*

[4]Oczywiście nie jest to możliwe jeżeli dysponujemy jedynie listami sąsiedztwa grafu $\overrightarrow{G}$. Wystarczy jednak utrzymywać dla każdego wierzchołka x, listę $In(x)$ wskaźników do elementów list sąsiedztwa odpowiadających krawędziom wchodzącym do x. Nie zmieni to złożoności czasowej operacji wykonywanych na grafie, jeśli tylko wraz z każdym takim elementem list sąsiedztwa będziemy przechowywać wskaźnik do odpowiadającego mu elementu listy $In(x)$.

zorientowany, dla pewnego δ takiego, że $D \geq 2\delta$. Niech H_i będzie grafem otrzymanym po i-tej operacji i niech t będzie liczbą wstawień.

Jeśli istnieje ciąg δ-orientacji $\overrightarrow{H_0}, \overrightarrow{H_1}, \ldots, \overrightarrow{H_{|\sigma|}}$, w którym całkowita liczba zmian orientacji krawędzi nie przekracza r, to algorytm Reorient *wykonuje nie więcej niż*

$$(t+r)\frac{D+1}{D+1-2\delta}$$

zmian orientacji krawędzi dla ciągu operacji σ.

Dowód. Pomimo bardziej ogólnego sformułowania, dowód lematu 1.6.2 pozostaje poprawny także dla nowej wersji tezy. Wystarczy zauważyć, że samo utożsamienie wierzchołków, podobnie jak usuwanie krawędzi, także nie zmienia potencjału Φ. ■

Lemat 1.6.13. *Niech H będzie dowolnym n-wierzchołkowym grafem o lesistości a i niech $\overrightarrow{H}$ będzie jego początkową δ-orientacją, $\delta \geq 2a$. Dla dowolnego ciągu operacji składającego się z p wstawień krawędzi, q usunięć krawędzi i r utożsamień wierzchołków w grafie H takiego, że lesistość grafu po wykonaniu każdej operacji nie przekracza a, algorytmy* Insert *i* Glue *utrzymują $(3\delta-1)$-orientację, wykonując $\mathcal{O}((p+r\delta)\log n)$ zmian orientacji krawędzi.*

Dowód. Dowód przeprowadzamy analogicznie jak w przypadku wniosku 1.6.5. Dla $i = 1, \ldots, p+q+r$, niech H_i będzie grafem po wykonaniu i-tej operacji i niech $H_0 = H$, $\overrightarrow{H_0} = \overrightarrow{H}$. Dla każdego $i > 0$ zbudujemy δ-orientację $\overrightarrow{H_i}$. Jeżeli i-tą operacją ciągu σ było usunięcie krawędzi, otrzymujemy orientację $\overrightarrow{H_i}$ z orientacji $\overrightarrow{H_{i-1}}$ przez usunięcie odpowiedniej krawędzi. Jeżeli i-tą operacją było wstawienie krawędzi uv, postępujemy dokładnie tak samo, jak w dowodzie wniosku 1.6.5, otrzymując δ-orientację $\overrightarrow{H_i}$ kosztem nie więcej niż $\lceil \log_{\delta/a} n \rceil$ zmian orientacji krawędzi. Załóżmy wreszcie, że i-tą operacją było utożsamienie wierzchołków. Przyjmijmy chwilowo, że $\overrightarrow{H_i}$ jest grafem powstałym z $\overrightarrow{H_{i-1}}$ przez utożsamienie odpowiednich wierzchołków. Jeżeli $\mathrm{outdeg}_{\overrightarrow{H_i}}(u) > \delta$, to $\mathrm{outdeg}_{\overrightarrow{H_i}}(u) - \delta$ razy wykonujemy co następuje. Na podstawie lematu 1.6.3 w grafie $\overrightarrow{H_i}$ istnieje ścieżka p o długości co najwyżej $\lceil \log_{\delta/a} n \rceil$, o początku w u i końcu w wierzchołku w o stopniu wychodzącym mniejszym od δ. Zmieniamy orientację wszystkich krawędzi na ścieżce p. Po odwróceniu odpowiedniej liczby takich ścieżek otrzymujemy δ-orientację grafu H_i. Ponieważ liczba wykonanych iteracji nie przekracza δ, zawierają one w sumie nie więcej niż $\delta\lceil \log_{\delta/a} n \rceil$ zmian orientacji krawędzi. Podsumowując, liczba zmian orientacji krawędzi w otrzymanym ciągu nie przekracza $(p+r\delta)\lceil \log_{\delta/a} n \rceil$.

Kładziemy $D = 3\delta - 1$. Z lematu 1.6.12 otrzymujemy, że całkowita liczba zmian orientacji krawędzi wykonanych przez algorytmy Insert i Glue nie przekracza $(p + (p+r\delta)\cdot\lceil \log_{\delta/a} n \rceil)\frac{D+1}{D+1-2\delta} \leq 3(p + (p+r\delta)\lceil \log_2 n \rceil) = O((p+r\delta)\log n)$. ■

Powyższy lemat oznacza, że zamortyzowany czas wykonania algorytmu GLUE w n-wierzchołkowym grafie o lesistości nie przekraczającej a wynosi $\mathcal{O}(\deg_G u + a \log n)$.

1.6.7 Utożsamianie wierzchołków: przypadek ogólny

Opiszemy teraz algorytm IDENTIFYSHORTCUT (u,v) (patrz pseudokod 1.6.4), który uaktualnia graf skrótów $\overrightarrow{S_k(G)}$ po operacji utożsamienia wierzchołków u i v. Algorytm działa w k fazach, po i-tej fazie grafy $\overrightarrow{G_1}, \ldots, \overrightarrow{G_i}$ zawierają już uaktualnioną informację. Dla każdego i, fazę zaczynamy od znalezienia krawędzi, które powinny znaleźć się w grafie G_i wskutek modyfikacji grafów $G_1, \ldots G_{i-1}$, wykonanych w poprzednich fazach. Każdą taką krawędź dodajemy za pomocą algorytmu INSERTSHORTCUT. Następnie utożsamiamy wierzchołki u i v w grafie G_i używając algorytmu GLUE.

Algorytm 1.6.4 Algorytm IDENTIFYSHORTCUT (u,v)

```
1: procedure IdentifyShortcut(u, v)
2:     for i ← 1 to k do
3:         Glue(u,v,i)                          ▷ Wykonuje Glue(u,v) w grafie G_i
4:         for j ← 1 to i − 1 do
5:             for each (v, x) ∈ E(G_j→) do
6:                 for each (v, y) ∈ E(G_{i−j}→) do
7:                     if xy ∉ E(G_i→) then
8:                         InsertShortcut(xy,i)
```

Dowody poniższych faktów są takie same jak w przypadku ich odpowiedników sformułowanych dla algorytmu INSERTSHORTCUT z podrozdziału 1.6.5.

Fakt 1.6.14. *Niech $S_k(G)$ będzie grafem skrótów dla grafu G. Wtedy graf otrzymany na skutek uruchomienia algorytmu* IDENTIFYSHORTCUT(u,v) *jest grafem skrótów grafu powstałego z G przez utożsamienie wierzchołków u i v.* ■

Lemat 1.6.15. *Dla dowolnego $i \leq k$, po dowolnym ciągu usunięć i wstawień krawędzi oraz utożsamień wierzchołków wykonanych za pomocą algorytmów* DELETESHORTCUT, INSERTSHORTCUT *i* IDENTIFYSHORTCUT, *graf $\overrightarrow{G_i}$ jest $\mathcal{O}(1)$-zorientowany i ma (G,i)-reprezentację grubości $\mathcal{O}(1)$.* ■

Wniosek 1.6.16. *W środowisku dynamicznym umożliwiającym wstawianie i usuwanie krawędzi, oraz utożsamianie wierzchołków grafu, czas przetwarzania zapytań o najkrótsze ścieżki pozostaje $\mathcal{O}(1)$.* ■

Zbadamy teraz efektywność algorytmu IDENTIFYSHORTCUT.

Lemat 1.6.17. *Dla dowolnego ciągu t wstawień krawędzi do grafu G i s utożsamień wierzchołków (być może oddzielonych operacjami usuwania krawędzi) i dla dowolnego*

$i \leq k$ wykonuje się $\mathcal{O}((t+s)\log^i n)$ zmian orientacji krawędzi w grafie $\overrightarrow{G_i}$, a także $\mathcal{O}((t+s)\log^{i-1} n)$ wstawień krawędzi do grafu G_i wykonanych za pomocą algorytmu InsertShortcut.

Dowód. Rozumowanie będzie analogiczne jak w przypadku lematu 1.6.10. Zwróćmy uwagę, że w powyższym lemacie rozważamy jedynie wstawienia do grafu G_i wykonane za pomocą algorytmu InsertShortcut. Na tę liczbę składają się wstawienia, które wykonuje algorytm IdentifyShortcut w linii 8. Nie wliczamy tu jednak wstawień z linii 4 i 7 algorytmu Glue. Skorzystamy z indukcji po i. Dla $i = 1$ teza wynika z lematu 1.6.13. Przejdźmy teraz do kroku indukcyjnego.

Dla $i > 1$ założenie indukcyjne pociąga za sobą, że dla każdego $j < i$ algorytmy Insert i Glue wykonują $\mathcal{O}((t+s)\log^j n)$ zmian orientacji krawędzi i wstawień w grafie $\overrightarrow{G_j}$. Ponieważ stopnie wychodzące w grafie skrótów są ograniczone przez stałą, każda z tych zmian orientacji krawędzi i operacji wstawienia powoduje jedynie $\mathcal{O}(1)$ wstawień i usunięć w G_i. Stąd całkowita liczba tych wstawień/usunięć w grafie G_i wynosi $\mathcal{O}((t+s)\log^{i-1} n)$. Dodatkowo mamy jeszcze $\mathcal{O}(1)$ wstawień do G_i wykonanych przez algorytm IdentifyShortcut podczas każdej operacji utożsamienia. Razem otrzymujemy $\mathcal{O}((t+s)\log^{i-1} n + s) = \mathcal{O}((t+s)\log^{i-1} n)$ wstawień krawędzi do grafu G_i. Z lematu 1.6.15 graf G_i ma lesistość $\mathcal{O}(1)$, a więc lemat 1.6.13 implikuje, że tych $\mathcal{O}((t+s)\log^{i-1} n)$ wstawień wraz z s operacjami Glue wykonanymi w grafie G_i wymagają $\mathcal{O}((t+s)\log^i n)$ zmian orientacji krawędzi w grafie $\overrightarrow{G_i}$. ■

Lemat 1.6.18. *Dla dowolnego $i = 1, \ldots, k$, $\deg_{G_i}(u) = \mathcal{O}(\deg_G(u))$.*

Dowód. Skorzystamy z indukcji po i. Dla $i = 1$ teza jest trywialna, ponieważ $G_1 = G$. Rozważmy teraz krawędzie incydentne z u w G_i. Każda taka krawędź ux jest wspierana przez pewien wierzchołek w. Takich wierzchołków jest nie więcej niż $\sum_{j<i} \deg_{G_j}(u)$ czyli nie więcej niż $\mathcal{O}(\deg_G(u))$, co wynika z założenia indukcyjnego. Ponieważ stopnie wychodzące w grafie skrótów są ograniczone przez stałą, każdy z tych wierzchołków wspiera $\mathcal{O}(1)$ krawędzi incydentnych z u w G_i. Stąd $\deg_{G_i}(u) = \mathcal{O}(\deg_G(u))$. ■

Wniosek 1.6.19. *Zamortyzowany czas aktualizowania grafu skrótów po operacji utożsamienia wierzchołków u i v za pomocą algorytmu* IdentifyShortcut(u,v) *wynosi $\mathcal{O}(\log^k n + \deg_G(u))$.*

Dowód. Natychmiastowy z lematów 1.6.17 i 1.6.18. ■

1.6.8 Modelowanie awarii w sieci: ukrywanie krawędzi i wierzchołków

W rzeczywistych sieciach, takich jak sieci połączeń drogowych lub wielkie sieci komputerowe, lokalne awarie są na porządku dziennym. Skoncentrujemy się na sytuacji,

gdy pewne połączenia lub węzły sieci mogą nie nadawać się do użytku przez pewien czas. Jest naturalne, by wymagać wtedy od wyroczni najkrótszych ścieżek, aby odpowiedzi na zapytania dotyczyły sieci, z której usunięto wadliwie działające elementy. Najprostszym rozwiązaniem jest usuwanie wadliwych elementów z grafu i wstawianie ich ponownie, jak tylko awaria zostanie usunięta. Przy użyciu algorytmów zaproponowanych w poprzednim podrozdziale, taka para operacji zajęłaby czas $\mathcal{O}(\log^k n)$. Zaproponujemy inne rozwiązanie, wymagające czasu $\mathcal{O}(1)$. Wprowadzimy dwie nowe operacje, które można wykonywać na krawędziach i wierzchołkach grafu: *ukrywanie* i *odkrywanie* (tzn. „cofnięcie wcześniejszej operacji ukrycia").

Początkowo wszystkie wierzchołki i krawędzie oznaczone są jako *odkryte*. Dla każdej krawędzi e grafu skrótów, oprócz list $P(e)$ i $C(e)$, utrzymujemy dodatkową listę $dP(e)$. Lista $dP(e)$ przechowuje wszystkie pary krawędzi $\{e_1, e_2\}$ takie, że e jest wspólnym dzieckiem e_1 i e_2 oraz co najmniej jedna z krawędzi e_1, e_2 jest ukryta.

Algorytm 1.6.5 Ukrywanie krawędzi grafu G

```
1: procedure DISABLE(e)
2:     Enabled[e] ← False
3:     for each (c, p) ∈ C(e) do
4:         {e, f} ← para wskazywana przez wskaźnik p
5:         P(c) ← P(c) \ {e, f}
6:         dP(c) ← dP(c) ∪ {e, f}
7:         if P(c) = ∅ then
8:             DISABLE(c)
```

Operacja DISABLE(e), ukrywająca krawędź e, działa podobnie jak algorytm DELETESHORTCUT (patrz pseudokod algorytmu 1.6.5). Zaczynamy od oznaczenia e jako krawędzi ukrytej. Następnie dla każdej pary $(c, p) \in C(e)$, przenosimy parę $\{e, f\}$ wskazywaną przez wskaźnik p z listy $P(c)$ rodziców c na listę $dP(c)$, ukrytych rodziców c. Jeśli lista $P(c)$ stanie się pusta ukrywamy krawędź c rekurencyjnie. Aby ukryć wierzchołek v grafu wystarczy, podobnie jak w przypadku usuwania, oznaczyć go jako ukryty i ukryć wszystkie krawędzie wychodzące z v.

Operacja ENABLE(e), odkrywająca krawędź e, działa jak następuje (patrz pseudokod algorytmu 1.6.8). Zaczynamy od oznaczenia e jako krawędzi odkrytej. Następnie dla każdej pary $(c, p) \in C(e)$ wykonujemy następujące operacje. Niech $\{e, f\}$ będzie parą rodziców c wskazywaną przez wskaźnik p; jeśli f jest odkryta, przenosimy parę $\{e, f\}$ z listy $dP(c)$ na listę $P(c)$. Jeśli f jest odkryta, c ukryta oraz c wychodzi z odkrytego wierzchołka, odkrywamy krawędź c rekurencyjnie. Ponownie, aby odkryć wierzchołek v wystarczy oznaczyć go jako odkryty i odkryć wszystkie krawędzie wychodzące z v z wyjątkiem tych krawędzi e, dla których lista $P(e)$ jest pusta (zauważmy, że krawędzie grafu G, które zostały ukryte przez „użytkownika" nie zostaną odkryte).

Łatwo wykazać, używając podobnych argumentów jak w podrozdziałach o usu-

Algorytm 1.6.6 Odkrywanie krawędzi grafu G

```
 1: procedure ENABLE(e)
 2:     Enabled[e] ← True
 3:     for each (c, p) ∈ C(e) do
 4:         {e, f} ← para wskazywana przez wskaźnik p
 5:         if Enabled[f] then
 6:             dP(c) ← dP(c) \ {e, f}
 7:             P(c) ← P(c) ∪ {e, f}
 8:             if not Enabled[c] then
 9:                 v ← wierzchołek, z którego wychodzi c
10:                 if Enabled[v] then
11:                     ENABLE(c)
```

waniu krawędzi i wierzchołków, że powyższe operacje wykonują się w czasie $\mathcal{O}(1)$. Oczywiście zakładamy, że algorytm przetwarzania zapytań ignoruje ukryte elementy grafu skrótów. Z drugiej strony, ukryte elementy nie są ignorowane przez algorytm InsertShortcut. Zauważmy na koniec, że liczba równocześnie ukrytych elementów może być dowolnie duża.

1.6.9 Zastosowanie: obliczanie talii

Niech k będzie ustaloną stałą, a G grafem planarnym. Rozważmy następujący problem: sprawdzić czy talia grafu G nie przekracza k i jeśli tak, wyznaczyć jej wartość i odpowiedni najkrótszy cykl. Przedstawimy prosty algorytm liniowy dla tego problemu. Wykorzystamy graf skrótów $\overrightarrow{S_k(G)}$, dostosowany do ukrywania i odkrywania krawędzi. Po zbudowaniu grafu skrótów, dla każdej krawędzi $e = uv$ grafu G wykonujemy co następuje:

(i) ukryj krawędź e,

(ii) sprawdź czy $\mathrm{dist}_G(v, w) \leq k - 1$ i jeśli tak, znajdź cykl złożony z krawędzi e i najkrótszej ścieżki łączącej v i w; zapamiętaj ten cykl, jeśli jest krótszy niż znalezione wcześniej,

(iii) odkryj krawędź e.

Każdy z powyższych kroków wykonuje się w czasie stałym z użyciem opisanych wcześniej algorytmów. Dlatego złożoność całego algorytmu obliczania talii jest liniowa.

1.7 Wyszukiwanie ścieżek i cykli o zadanej długości

W tym podrozdziale pokażemy, w jaki sposób zmodyfikować strukturę naszej wyroczni, aby umożliwić przetwarzanie innego rodzaju zapytań, które nazwiemy *zapytaniami o ścieżki zadanej długości*. Dla danych dwóch wierzchołków u i v oraz liczby całkowitej $t \leq k$, wynikiem zapytania jest ścieżka długości t łącząca u i v lub komunikat, że nie ma takiej ścieżki w grafie. Czas potrzebny na zbudowanie takiej wyroczni pozostaje liniowy, a zapytania w dalszym ciągu są przetwarzane w czasie stałym. Ponieważ nic nie stoi na przeszkodzie, aby zadając zapytanie przekazać dwa razy ten sam wierzchołek, tzn. położyć $u = v$, wyrocznia wyszukuje w czasie $\mathcal{O}(1)$ także cykle zadanej długości zawierające zadany wierzchołek. Prowadzi to do liniowego algorytmu poszukiwania w grafie planarnym cyklu o zadanej, stałej długości. W rozdziale 3 opisujemy zastosowania tego naturalnego problemu.

1.7.1 Pętle w grafie skrótów

Graf skrótów budowany na potrzeby przetwarzania zapytań o najkrótsze ścieżki nie zawierał pętli. W tym podrozdziale w sposób naturalny rozszerzymy definicję grafu skrótów, umożliwiając pojawianie się w nim pętli, będą one bowiem źródłem informacji o niektórych cyklach w grafie wejściowym G. W tym celu wystarczy zmodyfikować definicję operacji $\odot$, pozostawiając całą dalszą konstrukcję grafu skrótów bez zmian. Niech A i B będą grafami zorientowanymi o tym samym zbiorze wierzchołków V. Wynikiem operacji $A \odot B$ będzie graf prosty R o zbiorze wierzchołków V, w którym każdy wierzchołek może być incydentny z co najwyżej jedną pętlą. Podobnie jak poprzednio, dwa wierzchołki u i v są połączone krawędzią w R, gdy istnieje wierzchołek x taki, że $(x, u) \in E(A)$ oraz $(x, v) \in E(B)$. Dodatkowo, dla dowolnego wierzchołka u graf R zawiera pętlę incydentną z u, gdy istnieje wierzchołek x taki, że $(x, u) \in E(A)$ oraz $(x, u) \in E(B)$. Jest jasne, że takie rozszerzenie nie zmienia czasu potrzebnego na zbudowanie grafu skrótów, a stopnie wychodzące w $\overrightarrow{S_k(G)}$ pozostają ograniczone przez stałą.

1.7.2 Generowanie marszrut

Niech W będzie marszrutą o wadze t w $S_k(G)$ i niech W zawiera krawędź $e = uv$ o wadze większej niż 1. Krawędź e ma co najmniej jedną parę rodziców. Niech $(x, u), (x, v)$ będzie taką parą. Zastępujemy krawędź e przez ścieżkę uxv otrzymując nową marsztutę o wadze t w $S_k(G)$. Taką operację możemy wykonać kilka razy, za każdym razem otrzymując nową marszrutę o wadze t. Niech W' będzie otrzymaną marszrutą. Powiemy, że W *generuje* W'.

Formalnie, marszruta W *generuje* marszrutę W', jeżeli istnieje ciąg marszrut

$W_0, W_1, \ldots, W_l$ taki, że $W_0 = W$, $W_l = W'$, oraz dla każdego $i = 1, \ldots, l$, marszruta W_i może zostać otrzymana z W_{i-1} poprzez zastąpienie pewnej krawędzi $e = uv \in E(W_{i-1})$ przez ścieżkę uxv taką, że e jest wspólnym dzieckiem krawędzi $(x, u), (x, v) \in \overrightarrow{S_k(G)}$.

Zauważmy, że gdy ciąg marszrut jest dostatecznie długi, otrzymujemy marszrutę w grafie wejściowym G. Ponieważ krawędź może mieć wiele par rodziców, wnioskujemy, że marszruta może generować wiele marszrut w G. Zauważmy, że w szczególności każda marszruta generuje samą siebie. Dla wygody, rozszerzymy pojęcie generowania na ścieżki, ponieważ dowolną ścieżkę możemy traktować jako odpowiednią marszrutę. Zatem ścieżka może generować ścieżkę lub marszrutę. Będziemy też mówić o generowaniu w odniesieniu do zorientowanych ścieżek: zorientowana ścieżka $\overrightarrow{P}$ generuje marszrutę W' wtedy i tylko wtedy gdy jej szkielet P generuje W'.

Załóżmy, że poszukujemy ścieżki długości t łączącej wierzchołki u i v. Jeżeli taka ścieżka istnieje, twierdzenie 1.5.2 implikuje, że istnieją dwie ścieżki zorientowane o początkach w wierzchołkach u, v i spotykające się we wspólnym wierzchołku x oraz takie, że ich wagi sumują się do t. Możemy takie ścieżki znaleźć. Każda z nich generuje pewną (przynajmniej jedną) marszrutę w G. Jeśli dla każdej z rozpatrywanych dwóch ścieżek wybierzemy jedną generowaną przez nią marszrutę, a następnie połączymy te marszruty, to otrzymamy marszrutę o długości t łączącą u i v w grafie G. Niestety nie możemy być pewni, czy otrzymana marszruta odpowiada ścieżce. Gdyby nawet rozważyć wszystkie pary marszrut generowane przez wszystkie pary ścieżek to twierdzenie 1.5.2 nie gwarantuje, że choć jedna z tych par marszrut odpowiada ścieżce. Widzimy, że potrzebujemy mocniejszego twierdzenia:

Twierdzenie 1.7.1. *Dla dowolnej ścieżki P o długości co najwyżej k, łączącej u i v w grafie G (być może $u = v$), istnieje wierzchołek $x \in V(P)$ i dwie ścieżki zorientowane w $\overrightarrow{S_k(G)}$, ścieżka $\overrightarrow{P_1}$ od u do x i ścieżka $\overrightarrow{P_2}$ od v do x takie, że marszruta $P_1 \cup P_2$ generuje ścieżkę P.*

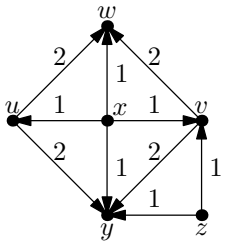
Dowód. Dowód jest analogiczny do dowodu twierdzenia 1.5.2. Używamy indukcji po długości scieżki P. Gdy P ma długość 1 twierdzenie jest oczywiste.

Przejdźmy teraz do kroku indukcyjnego. Dla dowolnych wierzchołków $a, b \in V(P)$ niech $P[a, b]$ oznacza ścieżkę łączącą a i b składającą się z krawędzi P. Tak jak poprzednio, niech u_1 będzie sąsiadem u na ścieżce P. Z założenia indukcyjnego istnieją dwie ścieżki zorientowane w $\overrightarrow{S_k(G)}$, ścieżka $\overrightarrow{Q_1}$ od u_1 do x i ścieżka $\overrightarrow{Q_2}$ od v do x takie, że marszruta $Q_1 \cup Q_2$ generuje ścieżkę $P[u_1, v]$. Niech $u_1, u_2, \ldots u_r = x$ będą kolejnymi wierzchołkami ścieżki Q_1. Niech $\mathrm{dist}_{Q_1}(a, b)$ oznacza odległość wierzchołków a i b na ścieżce Q_1, taktowanej jako graf z wagami na krawędziach. Określmy zbiór Z jak następuje:

$$Z = \{i : 1 \leq i \leq r \text{ oraz } E(G_{1+\mathrm{dist}_{Q_1}(u_1,u_i)}) \text{ zawiera krawędź } uu_i \text{ generującą } P[u, u_i]\}.$$

Ponownie, ponieważ $uu_1 \in E(G_1)$ i uu_1 generuje samą siebie, tj. ścieżkę $P[u, u_1]$, to wnioskujemy, że $1 \in Z$, a zatem $Z \neq \emptyset$. Od tego miejsca możemy dokończyć dowód dokładnie tak samo jak dowód twierdzenia 1.5.2. ■

1.7.3 Trudności na drodze do efektywnego algorytmu

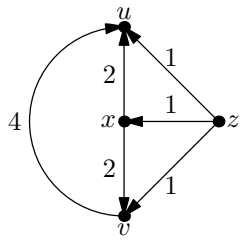


Rysunek 1.11: Poszukiwanie ścieżki długości 4 łączącej u i v.

Rozważmy graf skrótów przedstawiony na rysunku 1.11. Załóżmy, że poszukujemy w grafie G ścieżki o długości $t = 4$ łączącej wierzchołki u i v. Istnieje dokładnie jedna taka ścieżka, mianowicie ścieżka $uxyzv$. Twierdzenie 1.7.1 mówi nam, że jest ona generowana przez pewną parę ścieżek o początkach w u i v spotykających się w tym samym wierzchołku. Widzimy, że istnieje ścieżka o wadze 2 od u do w, składająca się z jednej krawędzi (u, w). Istnieje także ścieżka o wadze 2 od v do w – ona także składa się z jednej krawędzi (u, w). Choć wagi tych ścieżek sumują się do 4, nie generują one żadnej ścieżki.

Na szczęście to, że nie wszystkie pary ścieżek zorientowanych generują ścieżki nie jest dużym problemem na drodze do algorytmu, który w stałym czasie przetwarza zapytania. Nasz algorytm może bowiem znaleźć *wszystkie* pary odpowiednich ścieżek, których wagi sumują się do t. Ponieważ stopnie wyjściowe w grafie skrótów są ograniczone, to liczba takich par także jest ograniczona. Powróćmy jednak do sytuacji na rysunku 1.11. Graf skrótów zawiera jeszcze jedną parę ścieżek: pierwsza ścieżka składa się z krawędzi (u, y), natomiast druga z krawędzi (v, y). Pierwsza ścieżka generuje ścieżkę w G: ścieżkę uxy. Krawędź (v, y) ma natomiast dwie pary rodziców, tj. $\{(x, y), (x, v)\}$ i $\{(z, y), (z, v)\}$. Zatem rozważana para ścieżek generuje dwie marszruty w G: $uxyxv$ and $uxyzv$. Tylko druga z nich odpowiada ścieżce. Widzimy, że nie wystarczy dla każdej pary ścieżek sprawdzać po jednej generowanej przez nią marszrucie. Z drugiej strony, skoro krawędź może mieć $\Omega(n)$ par rodziców to nawet ścieżka składająca się tylko z jednej krawędzi może generować $\Omega(n)$ marszrut. Nasz algorytm nie może zatem sprawdzać wszystkich tych marszrut jeżeli chcemy, by działał w czasie stałym. Rozwiążemy ten problem w następujący sposób.

- Zauważymy, że pewne pary rodziców są „zbędne". Dla przykładu rozważmy graf skrótów z rysunku 1.12. Krawędź $uv \in E(G_4)$ ma jedną parę rodziców, tj.



Rysunek 1.12: Zbędna para rodziców.

$(x,u),(x,v)$. Mimo to, ścieżka złożona z krawędzi $(x,u),(x,v)$ nie generuje żadnej ścieżki w G. Takich zbędnych par rodziców nie będziemy przechowywać w grafie skrótów.

- Nasz algorytm przetwarzania zapytań znajdzie pewien zbiór marszrut generowaych przez pary ścieżek. Będą to jednak tylko niektóre, starannie wybrane marszruty, a ich liczba będzie ograniczona przez stałą (zależną jedynie od stałej k, ale nie od rozmiaru grafu). Okaże się, że jeśli żadna z tych wybranych marszrut nie jest ścieżką to w grafie G nie ma ścieżki żądanej długości łączącej dane wierzchołki.

1.7.4 Przetwarzanie zapytań

Dla $l \leq k$ powiemy, że graf $S_k(G)$ jest *l-czysty*, gdy dla każdej krawędzi e o wadze $w \leq l$, dla dowolnej pary jej rodziców $r = \{e_1, e_2\} \in P(e)$, istnieje ścieżka w G, która rozszerza r. Taką ścieżkę będziemy nazywać ścieżką *odpowiadającą* parze $\{e_1, e_2\}$. Zwróćmy uwagę, że długość tej ścieżki jest równa wadze krawędzi e. Będziemy zakładać, że w l-czystym grafie skrótów, dla dowolnej krawędzi e, każda para rodziców $r \in P(e)$ przechowuje odpowiadającą jej ścieżkę (dowolną jeśli takich ścieżek jest wiele) i tą przechowywaną ścieżkę będziemy oznaczać przez $p_G(r)$.

Opiszemy teraz algorytm wyszukiwania w grafie G ścieżki o zadanej długości t, łączącej daną parę wierzchołków u i v. Nasz algorytm będzie korzystał z informacji przechowywanych w grafie skrótów. Zakładamy, że graf skrótów $S_k(G)$ jest k-czysty. W punkcie 1.7.5 pokażemy, w jaki sposób skonstruować k-czysty graf skrótów w czasie liniowym.

Tak jak sygnalizowaliśmy w poprzednim punkcie, w naszym algorytmie rozważane są wszystkie pary $\overrightarrow{P_1}, \overrightarrow{P_2}$ w $\overrightarrow{S_k(G)}$ takie, że $\overrightarrow{P_1}$ jest zorientowaną ścieżką od u do pewnego wierzchołka x i $\overrightarrow{P_2}$ jest zorientowaną ścieżką od v do x oraz wagi ścieżek $\overrightarrow{P_1}$ i $\overrightarrow{P_2}$ sumują się do t. Ponieważ graf $\overrightarrow{S_k(G)}$ jest $\mathcal{O}(1)$-zorientowany, więc liczba takich par ścieżek jest ograniczona przez stałą.

Załóżmy, że poszukiwana ścieżka łącząca u i v istnieje. Oznaczmy ją przez

P. Dla każdej pary scieżek $\overrightarrow{P_1}, \overrightarrow{P_2}$ opisanej powyżej rozważamy marszrutę w $S_k(G)$ odpowiadającą sumie ścieżek P_1 i P_2. Twierdzenie 1.7.1 implikuje, że jedna z tych marszrut generuje P. Jest jasne, że możemy skoncentrować się jedynie na marszrutach, które odpowiadają ścieżkom w $S_k(G)$. Niech Q będzie taką ścieżką w $S_k(G)$. Pozostaje pokazać, jak w stałym czasie znaleźć w G ścieżkę generowaną przez Q.

Niech λ będzie pewną stałą, którą określimy później. Marszrutę W nazwiemy *marszrutą rzędu* λ, gdy każda krawędź W albo ma wagę 1 albo ma co najmniej λ par rodziców. *ścieżką rzędu* λ będziemy nazywać ścieżkę odpowiadajcą marszrucie rzędu λ.

Algorytm znajdowania w G ścieżki generowanej przez ścieżkę Q z grafu skrótów składa się z dwóch kroków.

Krok 1: od ścieżki Q w $S_k(G)$ do ścieżki Q_λ rzędu λ

Zauważmy, że ponieważ λ jest ograniczone, to Q generuje ograniczoną liczbę marszrut rzędu λ. Zatem znalezienie wszystkich takich marszrut zajmuje jedynie czas stały. Jeżeli żadna z nich nie jest ścieżką, to w grafie G nie ma ścieżki generowanej przez Q. W przeciwnym wypadku, niech Q_λ będzie dowolną ścieżką rzędu λ generowaną przez Q. Jeśli Q_λ jest ścieżką w G, tzn. wszystkie krawędzie w Q_λ mają wagi 1, znaleźliśmy żądaną ścieżkę i możemy zakończyć algorytm. Jeśli tak nie jest, wykonujemy krok 2.

Krok 2: wyszukiwanie ścieżki w G generowanej przez Q_λ

W celu znalezienia ścieżki w G generowanej przez Q_λ stosujemy następującą metodę (patrz algorytm 1.7.1: λ-Search). Dopóki Q_λ zawiera krawędź e o wadze większej od 1 próbujemy zastąpić e ścieżką odpowiadającą jednej z par rodziców e. Jeżeli tak otrzymana marszruta nie jest ścieżką, wykonujemy kolejną próbę, z następną parą rodziców e. Poniższy lemat gwarantuje, że gdy stała λ jest wystarczająco duża, to po sprawdzeniu nie więcej niż λ rodziców otrzymamy ścieżkę. To pociąga za sobą, że algorytm 1.7.1 *zawsze* znajduje ścieżkę w G, która rozszerza m i działa w czasie stałym.

Lemat 1.7.2. *Niech $S_k(G)$ będzie l-czystym grafem skrótów. Wtedy istnieje stała $\lambda(k)$ taka, że dla dowolnej ścieżki Q_λ rzędu λ i takiej, że każda krawędź Q_λ ma wagę co najwyżej l algorytm 1.7.1 znajduje ścieżkę w G generowaną przez Q_λ.*

Dowód. Powiemy, że krawędź $e \in S_k(G)$ *przecina* wierzchołek x, gdy istnieje ścieżka r w G rozszerzająca e i taka, że x jest wierzchołkiem wewnętrznym r. Rozważmy dowolny wierzchołek x grafu G. Graf $\overrightarrow{S_k(G)}$ jest $\mathcal{O}(1)$ zorientowany a więc wierzchołek x wspiera $\mathcal{O}(1)$ krawędzi $S_k(G)$. To pociąga za sobą, że istnieje stała $\lambda_1(k)$ taka, że dowolny wierzchołek x przecina nie więcej niż $\lambda_1(k)$ krawędzi w $S_k(G)$. Niech $\lambda_2(k)$

Algorytm 1.7.1 (λ-Search)

Wejście: ścieżka Q_λ w $S_k(G)$; Q_λ jest ścieżką rzędu λ.
Wyjście: ścieżka P w G taka, że Q_λ generuje P.

```
1: procedure λ-SEARCH(Q_λ)
2:     P ← Q_λ
3:     while istnieje krawędź xy ∈ E(P) o wadze > 1 do
4:         X ← zbiór dowolnych λ par rodziców krawędzi xy
5:         for each r = {(z,x),(z,y)} ∈ X do
6:             if (V(p_G(r)) \ {x,y}) ∩ V(P) = ∅ then
7:                 Zastąp krawędź xy ścieżki P przez ścieżkę p_G(r).
8:                 break                          ▷ Przerywa działanie pętli for
9:     return P
```

będzie stałą, która ogranicza stopnie wychodzące w $\overrightarrow{S_k(G)}$. Kładziemy

$$\lambda(k) := k(\lambda_1(k) + \lambda_2(k)) + 1.$$

Niech xy będzie krawędzią wybraną przez algorytm 1.7.1. Algorytm bierze parę rodziców $r = \{(z,x),(z,y)\}$ i sprawdza czy po zastąpieniu xy w ścieżce P przez ścieżkę $p_G(r)$ otrzymamy ścieżkę. Jeśli tak się stanie parę r będziemy nazywać *szczęśliwą*. W przeciwnym przypadku r nazywamy *chybioną* i algorytm wykonuje kolejną próbę dla następnej pary rodziców. Zwróćmy uwagę, że ponieważ ścieżka P zawiera co najwyżej k wierzchołków, to liczba rodziców krawędzi xy, którzy przecinają jeden z nich nie przekracza $k\lambda_1(k)$. Co więcej, liczba rodziców xy wychodzących z jednego z wierzchołków P nie przekracza $k\lambda_2(k)$. Każda chybiona para rodziców zawiera krawędź jednego z tych dwóch rodzajów. Z kolei każda krawędź może być elementem co najwyżej jednej pary rodziców xy ponieważ grafy G_i są proste. Otrzymujemy w ten sposób, że liczba chybionych par rodziców xy wynosi nie więcej niż $k(\lambda_1(k) + \lambda_2(k))$. To pociąga za sobą, że algorytm 1.7.1 znajdzie szczęśliwą parę rodziców po sprawdzeniu co najwyżej $\lambda(k)$ par. ■

Kroki 1 i 2 składają się na algorytm, który wyszukuje w grafie G ścieżkę P generowaną przez ścieżkę Q w grafie skrótów. Niech $mw(Q)$ będzie maksymalną spośród wag krawędzi na ścieżce Q. Zauważmy, że na to, żeby nasz algorytm działał poprawnie i w czasie stałym potrzeby jedynie, żeby graf skrótów był $mw(Q)$-czysty. Wykorzystamy tę własność w kolejnym punkcie.

1.7.5 Czyszczenie grafu skrótów

Pozostaje jedynie opisać, w jaki sposób otrzymać k-czysty graf skrótów w czasie liniowym. Zauważmy, że $S_k(G)$ jest zawsze 2-czysty. Aby $S_k(G)$ stał się 3-czysty, wykonujemy następujący algorytm.

Niech e będzie dowolną krawędzią o wadze 3 i niech $r = \{e_1, e_2\}$ będzie jedną z par rodziców e. Niech Q będzie ścieżką składającą się z krawędzi e_1, e_2. Zauważmy, że maksymalna spośród wag krawędzi na ścieżce Q nie przekracza 2. Zatem wykonując kroki 1 i 2 z poprzedniego punktu możemy w czasie stałym sprawdzić, czy Q generuje ścieżkę w G. Jeśli tak, znalezioną ścieżkę przechowujemy jako $p_G(e)$. W przeciwnym przypadku usuwamy r z grafu skrótów. Jeśli e nie ma już więcej par rodziców, w czasie stałym usuwamy e z grafu skrótów korzystając z algorytmu z podrozdziału 1.6.1. Jest jasne, że jeśli zastosujemy opisaną tu metodę kolejno do każdej krawędzi ze zbioru $E(G_3)$ to otrzymamy w czasie liniowy 3-czysty graf skrótów.

Podobnie mając c-czysty graf skrótów możemy w czasie liniowym otrzymać graf $(c+1)$-czysty. Po $k-2$ takich fazach otrzymujemy k-czysty graf skrótów.

1.8 Uogólnienia

1.8.1 Grafy z wagami na krawędziach

Rozważmy sytuację, w której w grafie wejściowym G każdej krawędzi e przypisano pewną wagę $w(e)$. Wagę krawędzi utożsamiamy wtedy z jej *długością*; podobnie pojęcie długości ścieżki oznacza wtedy sumę wag jej krawędzi. Dla takiego grafu chcielibyśmy przetwarzać zapytania opisane w tym rozdziale. Wydaje się mało prawdopodobne, aby podejście zastosowane do konstrukcji naszej wyroczni można było uogólnić do sytuacji, w której wagi krawędzi mogą być dowolnymi liczbami rzeczywistymi (czy wymiernymi). Niemniej jednak warto zaznaczyć, że nasza metoda w sposób trywialny stosuje się do grafów, w których krawędziom przypisano wagi będące liczbami naturalnymi. W fazie preprocessingu zaczynamy wtedy od usunięcia wszystkich krawędzi o wagach większych od k – nie będą one elementami poszukiwanych ścieżek. Następnie każdą krawędź o wadze w zastępujemy przez ścieżkę długości w. Jest jasne, że otrzymany graf jest liniowego rozmiaru względem grafu wejściowego, gdyż k traktujemy jako stałą. Ścieżki będące wynikami zapytań w takim grafie łatwo tłumaczymy na ścieżki w wejściowym grafie z wagami na krawędziach.

1.8.2 Grafy zorientowane

Uogólnienie naszych wyników na grafy zorientowane jest bardzo naturalnym pomysłem. Możemy to łatwo osiągnąć dokonując jedynie niewielkich modyfikacji w konstrukcji grafu skrótów. Niech G będzie wejściowym grafem zorientowanym natomiast G^u jego szkieletem. Przyjmujemy wtedy $G_1 = G^u$ i niech $\overrightarrow{G_1}$ będzie pewną $\mathcal{O}(1)$-orientacją grafu G^u. Oczywiście zwykle $\overrightarrow{G_1} \neq G$. Co więcej, teraz dla każdej krawędzi $e = xy \in E(S_k(G))$ przechowujemy *dwie* ścieżki zorientowane grafu G odpowiadające e: ścieżkę $p_G(x, y)$ prowadzącą od x do y oraz ścieżkę $p_G(y, x)$ prowadzącą od y do x.

Co najwyżej jedna z tych ścieżek może nie istnieć; wtedy w odpowiedniej zmiennej przechowujemy ścieżkę pustą. W przypadku zorientowanym grafy G_i, dla $i > 1$, są zdefiniowane jak następuje: $uv \in E(G_i)$ wtedy i tylko wtedy gdy istnieje wierzchołek x i liczby a, b takie, że $a + b = i$, $(x, u) \in E(\overrightarrow{G_a})$ oraz $(x, v) \in E(\overrightarrow{G_b})$ oraz co najmniej jeden z następujących warunków jest spełniony: (i) obie ścieżki $p_G(u, x)$ i $p_G(x, v)$ są niepuste; (ii) obie ścieżki $p_G(v, x)$ i $p_G(x, u)$ są niepuste. Oczywiście kiedy warunek (i) jest spełniony, jako $p_G(u, v)$ przechowujemy ścieżkę zawartą w marszrucie $p_G(u, x) \cup p_G(x, v)$. Analogicznie, gdy spełniony jest warunek (ii), jako $p_G(v, u)$ przechowujemy ścieżkę zawartą w marszrucie $p_G(v, x) \cup p_G(x, u)$. Wtedy dla dowolnego $i > 1$

$$G_i \subseteq \bigcup_{\substack{1 \leq p \leq q < i \\ p+q=i}} \overrightarrow{G_p} \odot \overrightarrow{G_q}.$$

Na podstawie wniosku 1.4.3, dla dowolnego i takiego, że $1 < i \leq k$, graf G_i ma (G, i)-reprezentację grubości $\mathcal{O}(1)$ i podobnie jak poprzednio można znaleźć w czasie liniowym jego $\mathcal{O}(1)$-orientację $\overrightarrow{G_i}$. W ten sposób, krawędź (u, v) w grafie $\overrightarrow{G_i}$ oznacza, że w grafie G istnieje zorientowana ścieżka łącząca u i v o długości co najwyżej i. Taka ścieżka może prowadzić od u do v ale także od v do u; mogą również istnieć dwie ścieżki łączące u i v, każda zorientowana w inną stronę.

Niech p będzie ścieżką zorientowaną w grafie $\overrightarrow{S_k(G)}$. Powiemy, że p jest *ścieżką w przód*, jeżeli dla każdej krawędzi $(x, y) \in E(p)$ ścieżka $p_G(x, y)$ jest niepusta. Podobnie p będziemy nazywać *ścieżką wstecz*, jeżeli dla każdej krawędzi $(x, y) \in E(p)$ ścieżka $p_G(y, x)$ jest niepusta. Zauważmy, że p może być równocześnie ścieżką w przód i ścieżką wstecz. Analogicznie definiujemy krawędzie w przód i krawędzie wstecz. Możemy teraz sformułować odpowiednik twierdzenia 1.7.1 dla grafów skierowanych.

Twierdzenie 1.8.1 (Odpowiednik twierdzenia 1.7.1 dla digrafów). *Niech G będzie zorientowanym grafem planarnym, a $\overrightarrow{S_k(G)}$ grafem skrótów dla G. Dla dowolnej ścieżki $\overrightarrow{P}$ o długości co najwyżej k, prowadzącej od u do v, w grafie G istnieje wierzchołek $x \in V(P)$ i dwie ścieżki zorientowane w $\overrightarrow{S_k(G)}$, ścieżka w przód $\overrightarrow{P_1}$ od u do x i ścieżka wstecz $\overrightarrow{P_2}$ od v do x takie, że $P_1 \cup P_2$ generuje P.*

Dowód. Dowód jest analogiczny jak dowód twierdzenia 1.7.1. ■

Oczywiście w algorytmie przetwarzającym zapytanie o najkótszą ścieżkę od u do v w wersji zorientowanej wyznaczamy graf $\overrightarrow{S(u)}$ korzystając jedynie z krawędzi w przód, natomiast graf $\overleftarrow{S(v)}$ korzystając jedynie z krawędzi wstecz. Podobnie modyfikujemy algorytm wyszukujący ścieżki zadanej długości z podrozdziału 1.7.4. Twierdzenie 1.8.1 gwarantuje poprawność obu algorytmów w wersji zorientowanej. Wyniki dotyczące środowiska dynamicznego przenoszą się natychmiast na grafy zorientowane – jedyna konieczna modyfikacja, to uwzględnienie nowych zasad pojawiania się i znikania krawędzi w grafach G_i.

1.8.3 Nie tylko grafy planarne

Jest jasne, że wszystkie algorytmy zaprezentowane w tym rozdziale mogą być wykonane dla dowolnego, niekoniecznie planarnego grafu danego na wejściu. Niemniej jednak analiza złożoności została przeprowadzona dla grafów planarnych. Możemy zadać naturalne pytanie: jakie własności grafów planarnych były konieczne w naszych dowodach? Innymi słowy, jakie warunki powinna spełniać klasa grafów $\mathcal{C}$, aby można byłą nią zastąpić grafy planarne w naszych rozważaniach, otrzymując wyniki tej samej jakości? Dociekliwy czytelnik może łatwo sprawdzić, że planarność była używana tylko dwukrotnie. Za pierwszym razem, w podrozdziale 1.5.1, podczas orientowania krawędzi grafu G_i („krok 2") korzystaliśmy z tego, że podrysunek grafu planarnego jest grafem planarnym oraz z faktu, że graf planarny można $\mathcal{O}(1)$-zorientować w czasie liniowym. Za drugim razem, gdy opisywaliśmy algorytm uaktualniania grafu skrótów (podrozdziały 1.6.3 i 1.6.5) wykorzystywaliśmy fakt, że lesistość grafów planarnych jest ograniczona przez stałą. Oczywiście graf o lesistości $\mathcal{O}(1)$ jest $\mathcal{O}(1)$-orientowalny, na podstawie wniosku 1.3.4 i można znaleźć taką orientację w czasie liniowym, co dostajemy z wniosku 1.3.9. Zatem klasa $\mathcal{C}$ powinna spełniać następujące warunki:

(a) $\mathcal{C}$ jest zamknięta na branie podrysunków,

(b) grafy z klasy $\mathcal{C}$ mają lesistość $\mathcal{O}(1)$.

Rozważmy następujące trzy prostsze własności:

(i) Grafy z klasy $\mathcal{C}$ są rzadkie, tzn. istnieje stała c taka, że dla dowolnego grafu $G \in \mathcal{C}$, $|E(G)| \leq c \cdot |V(G)|$,

(ii) $\mathcal{C}$ jest zamknięta na branie podgrafów,

(iii) jeśli graf G z klasy $\mathcal{C}$ jest podziałem grafu H, to $H \in \mathcal{C}$.

Twierdzimy, że koniunkcja warunków (a) i (b) jest równoważna koniunkcji warunków (i)-(iii). Warunek (b) implikuje (i) ponieważ $arb(G) \geq \frac{|E(G)|}{|V(G)|-1} \geq \frac{|E(G)|}{|V(G)|}$. Z (a) natychmiast mamy (ii) ponieważ podgraf grafu G jest 1-podrysunkiem grafu G. Podobnie (a) implikuje (iii), gdyż jeśli G jest podziałem H, to H jest podrysunkiem G. Z drugiej strony wprost z definicji (ii) i (iii) implikują (a). Podobnie z definicji lesistości (i) i (ii) dają (b). To kończy uzasadnienie równoważności warunków (a)-(b) oraz (i)-(iii).

Wniosek 1.8.2. *Dla dowolnej rodziny grafów spełniającej warunki (i)-(iii), graf skrótów jest tworzony w czasie liniowym. Zapytania o najkrótsze ścieżki oraz o ścieżki zadanej długości obsługiwane są w czasie stałym. Uaktualnienie grafu skrótów po usunięciu wierzchołka lub krawędzi oraz po dodaniu wierzchołka zajmuje czas stały, natomiast po wstawieniu krawędzi zajmuje zamortyzowany czas* $\mathcal{O}(\log^k n)$.

1.9 Etykietowanie wierzchołków

C. Gavoille, D. Peleg, S. Pérennes i R. Raz rozważają w artykule [29] następujące zagadnienie. Dla danego grafu należy przypisać jego wierzchołkom etykiety w taki sposób, aby dla dowolnych dwóch wierzchołków u i v możliwe było obliczenie odległości między nimi jedynie na podstawie przypisanych im etykiet. Naturalnie, celem jest uzyskanie możliwie *krótkich* etykiet, na podstawie których można *szybko* uzyskać informację na temat odległości. W kontekście wyroczni najkrótszych ścieżek otrzymujemy niezwykle naturalne pytanie: czy informacje składające się na strukturę danych wyroczni można *równomiernie rozproszyć* w grafie? W wymienionym artukule rozważano grafy bez wag na krawędziach i pokazano szereg dolnych i górnych granic na rozmiar etykiet dla rozmaitych klas grafów. Dla grafów planarnych, korzystając z twierdzenia o separatorze [42], pokazano, w jaki sposób skonstruować etykiety rozmiaru $\mathcal{O}(\sqrt{n}\log n)$. Czynnik $\log n$ pojawia się tu ponieważ taka jest liczba bitów konieczna do zapamiętania liczby wielkości n, czyli np. identyfikatora wierzchołka[5]. Co ważne, udowodniono także, że dolna granica na rozmiar etykiet dla grafu planarnego jest $\Omega(n^{1/3})$.

Problem etykietowania grafów rozważali także wcześniej Kannan, Naor i Rudich [33]. Zaprojektowali oni algorytmy etykietowania wierzchołków rozmaitych klas grafów tak, aby na podstawie etykiet dwóch dowolnych wierzchołków można było się przekonać, czy są one połączone krawędzią. Dla grafów planarnych autorzy uzyskali etykiety rozmiaru $\mathcal{O}(\log n)$.

Łatwo zauważyć, że nasze rezultaty przedstawione w tym rozdziale można wyrazić jako schematy etykietowania wierzchołków i algorytmy uzyskiwania konkretnych informacji na podstawie etykiet. Jako etykietę dowolnego wierzchołka v wystarczy przyjąć informacje przechowywane w strukturze wyroczni związane z wierzchołkami leżącymi w odległości nie większej niż k od wierzchołka v. Innymi słowy, etykieta wierzchołka v zawiera podgraf grafu skrótów $\overrightarrow{S_k(G)}$ indukowany przez wierzchołki leżące w odległości nie większej niż k od v wraz z informacjami związanymi z krawędziami (wagi, listy rodziców $P(e)$, listy dzieci $P(e)$ itd.). Rozmiar takiego podgrafu jest ograniczony przez stałą, a więc rozmiar etykiet nie przekracza $\mathcal{O}(\log n)$ bitów. Tak przedstawiony wynik jest uogólnieniem rezultatów z pracy [33] i alternatywą dla etykietowania opisanego w [29].

Jak pamiętamy, istota naszego podejścia polega na tym, że chcąc znaleźć ścieżkę łączącą dwa wierzchołki, powiedzmy u i v, łączymy ze sobą *dwie* ścieżki, z których jedną znajdujemy w sąsiedztwie u (czy też odczytujemy z etykiety u), natomiast drugą wyszukujemy w sąsiedztwie (etykiecie) wierzchołka v. Niezależnie taki sam sposób etykietowania wierzchołków grafu odkryli E. Cohen, E. Halperin, H. Kaplan i U.

[5]Pamiętamy, że w naszych rozważaniach korzystaliśmy z modelu, w którym identyfikatory wierzchołków zajmują pamięć rozmiaru $\mathcal{O}(1)$ – patrz akapit „model obliczeń” na str. 20

Zwick [15]. Swoje etykiety nazwali *etykietami 2-skokowymi*(ang. *2-hop labels*). Dla dowolnego wierzchołka v jego etykieta jest zbiorem ścieżek („skoków") o początku w v. Autorzy nie zajmowali się jednak grafami planarnymi gdzie, jak się przekonaliśmy, tego rodzaju etykiety mają szczególnie interesujące własności (przede wszystkim mały rozmiar). Wspominają jedynie, że struktura wyroczni zaproponowanej przez Thorupa [54] może zostać potraktowana jako schemat etykietowania, w którym całkowity rozmiar pamięci potrzebnej do przechowywania etykiet jest $\mathcal{O}(n \log^2 n)$. Autorzy pracy [15] zajmują się zagadnieniem konstruowania etykiet 2-skokowych na podstawie dowolnego zbioru ścieżek (w szczególności zbioru najkrótszych ścieżek). Ich celem jest minimalizowanie całkowitej liczby skoków składających się na etykiety. Pokazują m. in., że problem skonstruowania etykietowania o najmniejszej liczbie skoków jest NP-trudny i pokazują efektywny algorytm $\log n$-aproksymacyjny. Badają również eksperymentalnie efektywność wprowadzonego etykietowania dla próbek grafów modelujących duże, występujące w praktyce sieci, takie jak sieć komputerowa dużego dostawcy usług internetowych, czy sieć drogowa pewnego regionu stanu Kalifornia.

Rozdział 2

Kolorowanie grafów planarnych bez trójkątów trzema kolorami

2.1 Wprowadzenie

Słynne twierdzenie o czterech barwach mówi, że wierzchołki każdego grafu planarnego można pokolorować czterema kolorami. Artykuł Robertsona, Sandersa, Seymoura i Thomasa [48] zawiera odpowiedni algorytm 4-kolorowania, działający w czasie $\mathcal{O}(n^2)$. Ten wynik wydaje się być trudny do poprawienia, ponieważ bardziej efektywny algorytm wymagałby prawdopodobnie istotnie nowego dowodu twierdzenia o czterech barwach. Z drugiej strony znany jest cały wachlarz liniowych algorytmów 5-kolorowania grafów planarnych (zobacz [12, 13, 26, 43, 55]). Jak wiemy, problemy 1- i 2-kolorowania dowolnych grafów mają łatwe, również liniowe rozwiązania. Dlatego najciekawszym obszarem badawczym w dziedzinie kolorowania grafów planarnych wydaje się poszukiwanie efektywnych algorytmów, które kolorują takie grafy używając jedynie trzech kolorów (jeżeli jest to możliwe). Ogólny problem decyzyjny, w którym należy sprawdzić, czy dany graf planarny jest 3-kolorowalny, okazuje się być NP-zupełny [28]. Niemniej jednak, znane jest twierdzenie Grötzscha [32], które mówi, że każdy graf planarny bez trójkątów (tzn. nie zawierający podgrafu izomorficznego z K_3) jest 3-kolorowalny. Najprostsze znane dowody tego twierdzenia przedstawił w ostatnich latach Carsten Thomassen (zobacz [50, 52]). Wśród badaczy zajmujących się tematyką kolorowania wydaje się być powszechnie uznane, że dowody Thomassena można dość łatwo zamienić na algorytmy działające w czasie $\mathcal{O}(n^2)$. W tym rozdziale pokażemy dużo bardziej efektywny algorytm o złożoności $\mathcal{O}(n \log n)$.

W podrozdziale 2.2 zaprezentujemy nowy dowód twierdzenia Grötzscha, oparty na technikach użytych przez Thomassena [50]. Przedstawiony dowód jest indukcyjny, a więc odpowiada algorytmowi rekurencyjnemu. Dlatego został on sformułowany w specyficzny sposób, tak, aby mógł być odczytywany także jako algorytm. Zamieszczony dowód można właściwie traktować jako opis algorytmu przemieszany z

dowodem jego poprawności. W podrozdziale 2.3 opisujemy, jak w sposób efektywny zaimplementować przedstawiony algorytm. W tym celu podajemy szczegóły realizacji nietrywialnych operacji, jak również struktury danych niezbędne do ich szybkiego wykonywania. Jedną z używanych struktur danych jest wyrocznia najkrótszych ścieżek opisana w rozdziale 1. Warto podkreślić, że korzystamy tu jedynie z bardzo uproszczonej wersji wyroczni, wyszukującej jedynie ścieżki długości 1 i 2, dzięki czemu złożoność czasowa algorytmu nie kryje w sobie ogromnych stałych, o których była mowa w podrozdziale 1.5.3. Niestety nie wszystkie oszacowania złożoności czasowej algorytmów związanych z wyrocznią przedstawione w rozdziale 1 wystarczają aby udowodnić, że nasz algorytm 3-kolorowania działa w czasie $\mathcal{O}(n \log n)$. Dlatego w podrozdziale 2.4 przeprowadzamy analizę efektywności pracy wyroczni dla szczególnego ciągu wykonywanych na niej operacji, który jest wymuszony przez algorytm kolorowania.

Niedawno pojawiły się dwa nowe twierdzenia o brzmieniu podobnym do twierdzenia Grötzscha. Pierwsze z nich mówi, że dowolny graf planarny nie zawierający 5-cykli ani trójkątów leżących w odległości co najwyżej 4 jest 3-kolorowalny (zobacz [8]). Drugie twierdzenie podaje, że każdy graf planarny bez cykli długości od 4 do 7 jest 3-kolorowalny (zobacz [7]). Twierdzimy, że techniki opisane w tym rozdziale mogą zostać użyte do przełożenia tych dowodów na algorytmy 3-kolorowania o złożoności, odpowiednio, $\mathcal{O}(n \log^5 n)$ i $\mathcal{O}(n \log^7 n)$. Precyzyjna analiza złożoności prawdopodobnie pomogłaby poprawić te oszacowania, podobnie jak to ma miejsce w przypadku algorytmu opisanego w niniejszym rozdziale.

Terminologia Wprowadzimy w tym miejscu kilka pojęć, które będą używane jedynie w niniejszym rozdziale. Niech C będzie cyklem w grafie płaskim G. Cykl C dzieli płaszczyznę na dwa zbiory otwarte, D i E, gdzie zbiór D jest homeomorficzny z kołem otwartym w $\mathbb{R}^2$. Zbiór złożony ze wszystkich wierzchołków grafu G należących do D i wszystkich krawędzi przecinających ten zbiór będziemy nazywać *wnętrzem cyklu* i oznaczać przez $\mathrm{int}(C)$. Zwróćmy uwagę, że $\mathrm{int}(C)$ nie musi być grafem, podczas gdy $C \cup \mathrm{int}(C)$ jest zawsze podgrafem G. W dowolnym grafie planarnym G dokładnie jedna ściana jest nieograniczona. Taką ścianę nazywamy *ścianą zewnętrzną*, natomiast wszystkie pozostałe ściany określamy jako *wewnętrzne*.

2.2 Nowy dowód twierdzenia Grötzscha

Zaprezentujemy teraz nowy dowód twierdzenia Grötzscha. Nasz dowód jest oparty na pomysłach Carstena Thomassena [50][1]. Opracowanie nowego dowodu było konieczne ze względu na to, że dowód oryginalny odpowiada jedynie algorytmowi o złożoności $\mathcal{O}(n \log^3 n)$, przy założeniu, że przy jego implementacji stosujemy techniki algoryt-

[1]Zamiast adaptować dowód Thomassena [50] z roku 1994 moglibyśmy dostosować także najnowszy dowód [52], opublikowany w roku 2003. Podejrzewamy jednak, że otrzymany w rezultacie algorytm byłby bardziej skomplikowany i trudniejszy do opisania.

miczne opisane w kolejnych podrozdziałach. W algorytmie uzyskanym na podstawie poniższego dowodu wyeliminowano konieczność wyszukiwania w grafie ścieżek o długościach 3 i 4, co zmniejszyło złożoność czasową do $\mathcal{O}(n \log n)$. W naszym dowodzie twierdzenia Grötzscha skorzystamy z lematu 2.2.1. W dowodzie lematu wykorzystujemy szeroko stosowaną technikę *rozładowywania* (ang. *discharging*), użytą m.in. w dowodzie twierdzenia o czterech barwach (patrz [5, 49]). Zwykle technika rozładowywania jest używana do wykazania istnienia określonego obiektu w grafie planarnym. Technika ta polega na przypisaniu pewnym elementom grafu (wierzchołkom, krawędziom, ścianom) liczb, nazywanych *ładunkami*, w taki sposób, żeby całkowity ładunek wszystkich elementów grafu był ujemny. Następnie, za pomocą rozmaitych reguł przesyłania, ładunek przepływa z jednych elementów grafu do innych. Oczywiście w czasie tej fazy całkowity ładunek w grafie nie ulega zmianie. Celem jest pokazanie, że gdyby pożądanego obiektu nie było w grafie, to po zakończeniu cyrkulacji ładunków całkowity ładunek byłby dodatni, co z kolei prowadziłoby do sprzeczności. Aby ten cel osiągnąć należy w odpowiedni sposób dobrać zestaw reguł przesyłania ładunków.

Lemat 2.2.1. *Niech G będzie dwuspójnym grafem płaskim, w którym każda ściana wewnętrzna ma długość 5, natomiast ściana zewnętrzna C ma długość $4 \leq |C| \leq 6$. Załóżmy ponadto, że każdy wierzchołek spoza $V(C)$ ma stopień co najmniej 3 oraz że G nie zawiera pary wierzchołków stopnia 2 połączonych krawędzią. Wtedy G zawiera cykl ścianowy C' taki, że $V(C') \cap V(C) = \emptyset$ oraz wszystkie wierzchołki ściany C' są stopnia 3, z wyjątkiem co najwyżej jednego, który ma stopień co najwyżej 5.*

Dowód. Każdy wierzchołek v grafu G otrzymuje ładunek równy $\deg_G(v) - 4$ jednostek. Podobnie, każda ściana q grafu G otrzymuje $|q| - 4$ jednostek ładunku. Ściana zewnętrzna dostaje 7 dodatkowych jednostek ładunku. Niech n, m, f oznaczają, odpowiednio, liczby wierzchołków, krawędzi i ścian grafu G oraz niech V i F będą, odpowiednio, zbiorami wierzchołków i ścian tego grafu. Korzystając ze wzoru Eulera łatwo obliczamy całkowitą wartość ładunku w grafie G:

$$\sum_{v \in V} (\deg_G(v) - 4) + \sum_{q \in F} (|q| - 4) + 7 = 2m - 4n + 2m - 4f + 7 = -1. \qquad (2.1)$$

Następnie przesyłamy ładunek z wierzchołków do ścian, w taki sposób, że każdy wierzchołek v wysyła $\frac{\deg_G(v)-4}{\deg_G(v)}$ jednostek ładunku do każdej ściany incydentnej z v. Niech d_4 i d_5 oznaczają, odpowiednio, liczby wierzchołków stopnia 4 i 5 w $V(C)$. Ponieważ z założeń wnioskujemy, że w grafie G mogą znajdować się nie więcej niż 3 wierzchołki stopnia 2, więc ściana zewnętrzna otrzymuje co najmniej $|C| - 4 + 7 + 3 \cdot (-1) + 3 \cdot (-\frac{1}{3}) + d_4 \cdot \frac{1}{3} + d_5 \cdot (\frac{1}{3} + \frac{1}{5}) = |C| - 1 + d_4 \cdot \frac{1}{3} + d_5 \cdot \frac{8}{15}$ jednostek ładunku.

Zauważmy, że wszystkie ściany grafu mają teraz nieujemny ładunek, z wyjątkiem, być może, 5-ścian z 4 wierzchołkami stopnia 3 i jednym stopnia między 3 a 5 (ładunek co najmniej $-\frac{2}{3}$) lub 5-ścian zawierających wierzchołek stopnia 2 (ładunek co najmniej $-\frac{4}{3}$).

Następnie, ściana zewnętrzna przesyła każdej krawędzi z $E(C)$ taki sam ładunek równy $1-\frac{1}{|C|} \geq \frac{3}{4}$. W dalszej kolejności każda z tych krawędzi przesyła otrzymany ładunek do drugiej incydentnej z nią ściany (różnej od C). Zanotujmy, że ściana zewnętrzna przechowuje po tej operacji co najmniej $d_4 \cdot \frac{1}{3} + d_5 \cdot \frac{8}{15}$ jednostek ładunku.

Widzimy, że teraz wszystkie ściany, które mają choć jedną krawędź wspólną ze ścianą zewnętrzną, przechowują ładunek nieujemny: ściany z wierzchołkiem stopnia 2 mają co najmniej $-\frac{4}{3} + 2 \cdot \frac{3}{4} = \frac{1}{6}$ jednostek ładunku, podczas gdy pozostałe takie ściany mają $\geq -\frac{2}{3} + \frac{3}{4} = \frac{1}{12}$ jednostek ładunku.

Zwróćmy uwagę, że jeżeli istnieje teraz ściana q z ujemnym ładunkiem, która ma pewien wierzchołek x w zbiorze $V(C)$, to $\deg x \in \{4, 5\}$ i pozostałe wierzchołki q nie mogą należeć do $V(C)$. Każdej takiej ścianie q przesyłamy dodatkowy ładunek ze ściany zewnętrznej. Jeśli $\deg x = 4$, to przesyłamy $\frac{1}{3}$ jednostek ładunku, natomiast gdy $\deg x = 5$, przesyłamy ładunek równy $\frac{2}{15}$. W obu przypadkach ładunek ściany q staje się nieujemny. Ponieważ ściana zewnętrzna w tej ostatniej fazie oddaje nie więcej niż $d_4 \cdot \frac{1}{3} + d_5 \cdot 2 \cdot \frac{2}{15}$ jednostek ładunku, a więc ona także zachowuje nieujemny ładunek.

Widzimy teraz jasno, że gdyby graf G nie zawierał cyklu ścianowego o podanych własnościach, to całkowity ładunek w grafie G byłby nieujemny, a to jest sprzeczne z równością (2.1). ■

Zamiast bezpośrednio dowodzić twierdzenie Grötzscha łatwiej nam będzie pokazać poniższą, bardziej ogólną tezę. Chociaż uzyskaliśmy ten wynik niezależnie, jego treść wynika także z twierdzenia 5.3 z artykułu [30].

3-kolorowanie cyklu C o długości nie przekraczającej 6 będziemy nazywać *bezpiecznym*, gdy $|C| < 6$ lub gdy $|C| = 6$ i kolejność kolorów pojawiających się na cyklu jest różna od $(1, 2, 3, 1, 2, 3)$ i $(3, 2, 1, 3, 2, 1)$.

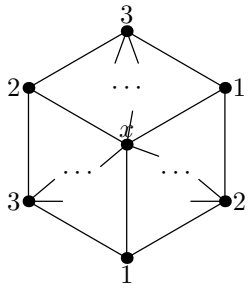
Twierdzenie 2.2.2. *Dowolny spójny graf płaski bez trójkątów G jest 3-kolorowalny. Ponadto, jeśli marszruta ścianowa ściany zewnętrznej C grafu G jest cyklem długości nie przekraczającej 6, to każde bezpieczne 3-kolorowanie grafu $G[V(C)]$ może zostać rozszerzone do 3-kolorowania całego grafu G.*

Dowód. Dowód prowadzimy przez indukcję po $|V(G)|$. Wierzchołki grafu G są albo wszystkie niepokolorowane, albo wierzchołki ściany zewnętrznej są pokolorowane, natomiast wszystkie pozostałe nie mają przypisanych kolorów. W pierwszej sytuacji pokażemy, że istnieje 3-kolorowanie grafu G. Zakładamy, że druga sytuacja może zdarzyć się jedynie gdy marszruta ścianowa ściany zewnętrznej C grafu G jest cyklem długości nie przekraczającej 6. Zakładamy ponadto, że kolorowanie wierzchołków ściany zewnętrznej jest bezpiecznym 3-kolorowaniem grafu $G[V(C)]$ i pokażemy, że to kolorowanie można rozszerzyć do 3-kolorowania całego grafu G.

Zakładamy, że G zawiera co najmniej jeden niepokolorowany wierzchołek, bowiem w przeciwnym przypadku teza jest spełniona. W trakcie dowodu rozważymy szereg przypadków. Podczas rozważania każdego z nich zakładamy, że żaden z wcześniej rozpatrzonych przypadków nie stosuje się już do grafu G.

Przypadek 1. G zawiera niepokolorowany wierzchołek x stopnia co najwyżej 2. Wtedy możemy skorzystać z założenia indukcyjnego i otrzymać 3-kolorowanie grafu $G' = G[V(G) \setminus \{x\}]$. (Jeżeli x jest wierzchołkiem rozcinającym, stosujemy założenie indukcyjne do każdej ze spójnych składowych G'.) Do grafu G' wstawiamy następnie x oraz usunięte wcześniej krawędzie incydentne z x. Otrzymujemy graf G, w którym jedynie wierzchołek x nie został pokolorowany. Co więcej, sąsiedzi x wykorzystują co najwyżej 2 kolory, a więc zostaje co najmniej 1 *wolny kolor*, którym możemy pokolorować wierzchołek x.

W dalszych rozważaniach możemy więc wykluczyć przypadek 1. Zauważmy, że gdy marszruta ścianowa ściany zewnętrznej ma długość co najwyżej 6 i graf nie zawiera trójkątów oraz gdy przypadek 1 nie zachodzi to marszruta ściany zewnętrznej musi być cyklem (a nie jedynie marszrutą zamkniętą). W szczególności odtąd możemy zakładać, że jeśli graf G zawiera pokolorowane wierzchołki to marszruta ścianowa ściany zewnętrznej jest cyklem.



Rysunek 2.1: Przypadek 2.

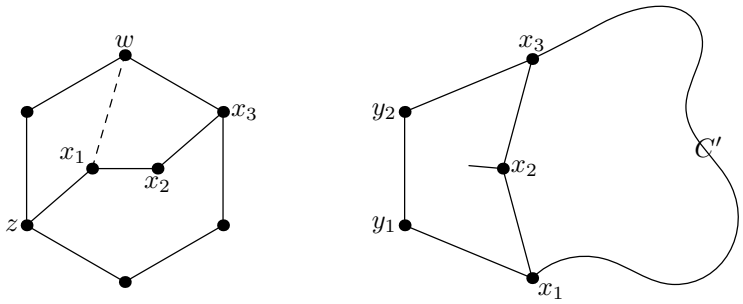
Przypadek 2. G zawiera niepokolorowany wierzchołek x połączony krawędziami z dwoma lub trzema pokolorowanymi wierzchołkami (patrz rysunek 2.1). Zauważmy, że gdy x ma trzech pokolorowanych sąsiadów, to oznacza, że $|C| = 6$ oraz ci sąsiedzi nie mogą mieć przypisanych wszystkich trzech kolorów, ponieważ kolorowanie C jest bezpieczne. Dzięki temu zawsze możemy rozszerzyć 3-kolorowanie C do 3-kolorowania $G[V(C) \cup \{x\}]$. Zauważmy, że dla każdego cyklu ścianowego grafu $G[V(C) \cup \{x\}]$ otrzymane kolorowanie jest bezpieczne. Pozwala to na zastosowanie założenia indukcyjnego do każdej ze ścian $G[V(C) \cup \{x\}]$ wraz z ich wnętrzami.

Przypadek 3. Cykl C jest pokolorowany i ma cięciwę. Postępujemy wtedy analogicznie jak w przypadku 2.

Przypadek 4. W G istnieje marszruta ścianowa $C' = x_1 x_2 \cdots x_k x_1$ taka, że $k \geq 6$ oraz co najmniej jeden wierzchołek C' nie jest pokolorowany. Ponieważ wy-

kluczyliśmy przypadek 2, obaj sąsiedzi niepokolorowanego wierzchołka C' nie mogą być równocześnie pokolorowani. Stąd C' zawiera dwa kolejne niepokolorowane wierzchołki. Bez straty ogólności możemy założyć, że są to wierzchołki x_1 i x_2.

Przypadek 4a. Załóżmy, że x_3 jest pokolorowany i x_1 ma pokolorowanego sąsiada z (patrz rysunek 2.2). Ponieważ wykluczyliśmy przypadek 2, x_1 oraz x_2 nie



Rysunek 2.2: Przypadki 4a (z lewej) i 4b (z prawej).

mogą mieć pokolorowanych sąsiadów, z wyjątkiem, odpowiednio, z i x_3. Wtedy graf $G[V(C) \cup \{x_1, x_2\}]$ ma dokładnie dwie ściany wewnętrzne, każda o długości co najmniej 4. Oznaczmy przez C_1 i C_2 marszruty ścianowe odpowiadające tym ścianom tak, że $|C_1| \leq |C_2|$. Widzimy, że $|C_1| \leq 6$, ponieważ w przeciwnym przypadku $|C| \geq 7$ i cykl C nie może być pokolorowany. Bez straty ogólności możemy założyć, że C_1 jest cyklem separującym, ponieważ w przeciwnym przypadku $C_1 = C'$, $|C_1| = 6$, $|C_2| = 6$ i cykl C_2 jest separujący. Niech $G' = G - V(\text{int}(C_1))$. Gdy C_1 jest cyklem i ma długość 6, dodajemy nową krawędź do G' łącząc wierzchołek x_1 i wierzchołek w odległy o 3 od x_1 w cyklu C_1. Zauważmy, że taka krawędź nie utworzy trójkąta w G'. Następnie stosujemy założenie indukcyjne do grafu G'. Zwróćmy uwagę, że otrzymane 3-kolorowanie C_1 jest bezpieczne, ponieważ gdy $|C_1| = 6$, wierzchołki x_1 i w otrzymują różne kolory. Co więcej, ponieważ C_1 nie ma cięciw w G, to otrzymane kolorowanie C_1 jest także poprawnym kolorowaniem $G[V(C_1)]$, możemy więc z pomocą indukcji rozszerzyć je do kolorowania grafu $C_1 \cup \text{int}(C_1)$.

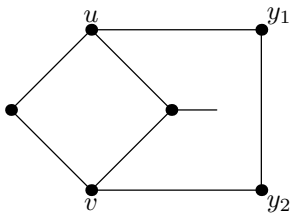
Przypadek 4b. Istnieje ścieżka $x_1y_1y_2x_3$ lub $x_1y_1x_3$ różna od $x_1x_2x_3$ (patrz rysunek 2.2). Ponieważ G nie zawiera trójkątów, więc $x_2 \notin \{y_1, y_2\}$. Niech C'' będzie, odpowiednio, cyklem $x_3x_2x_1y_1y_2x_3$ lub $x_3x_2x_1y_1x_3$ oraz niech $G_1 = G - V(\text{int}(C''))$. Wtedy $|V(G_1)| < |V(G)|$, ponieważ $\deg_G(x_2) \geq 3$. Korzystamy z założenia indukcyjnego i otrzymujemy 3-kolorowanie grafu G_1 (jeśli wierzchołki cyklu C są pokolorowane to rozszerzamy to kolorowanie na cały graf G_1). Otrzymane w ten sposób kolorowanie C'' jest bezpiecznym kolorowaniem $G[V(C'')]$, ponieważ $|C''| \leq 5$ i C'' nie ma cięciw. Dlatego w celu znalezienia 3-kolorowania grafu $C'' \cup \text{int}(C'')$ możemy ponownie zastosować indukcję.

Przypadek 4c. W sytuacji, gdy przypadki 4a i 4b nie mają miejsca, możemy utożsamić wierzchołki x_1 i x_3, nie tworząc przy tym cięciwy cyklu C ani trójkąta

w otrzymanym grafie G'. Następnie wystarczy skorzystać z indukcji aby otrzymać kolorowanie G'. Wszystkie wierzchołki w G otrzymują następnie takie same kolory jak ich odpowiedniki w G' (wierzchołki x_1 i x_3 otrzymują taki sam kolor jak wierzchołek powstały po ich utożsamieniu).

Przypadek 5. W G istnieje cykl ścianowy C' o długości 4. Ponadto, jeśli cykl C jest pokolorowany, zakładamy, że $C' \neq C$. Zauważmy, że C' zawiera dwa naprzeciwległe wierzchołki, u, v takie, że jeden z nich nie jest pokolorowany i utożsamienie u i v nie powoduje powstania krawędzi łączącej dwa pokolorowane wierzchołki. Gdyby tak nie było, G zawierałby trójkąt lub zachodziłby jeden z przypadków 2 lub 3.

Przypadek 5a. Istnieje ścieżka uy_1y_2v, $y_1 \notin V(C')$ (patrz rysunek 2.3). Ponie-



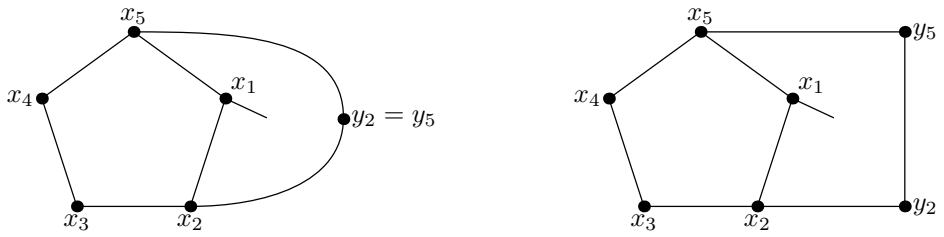
Rysunek 2.3: Przypadek 5a.

waż G nie zawiera trójkątów, więc $y_2 \notin V(C')$. Wtedy wymieniona ścieżka wraz jedną z dwóch ścieżek łączących w C' wierzchołki u i v tworzy cykl separujący C'' długości 5. Możemy wtedy dokończyć dowód podobnie jak przypadku 4b.

Przypadek 5b. Skoro przypadek 5a został wykluczony, możemy utożsamić u i v nie tworząc przy tym trójkąta. Jeżeli jeden z utożsamianych wierzchołków jest pokolorowany, nowo utworzony wierzchołek (leżący w marszrucie ścianowej nowej ściany zewnętrznej) kolorujemy tym samym kolorem. Zwróćmy uwagę, że gdy C jest pokolorowany oraz $|C| = 6$, to kolorowanie C pozostaje bezpieczne. Pozwala to na skorzystanie z założenia indukcyjnego dla tak utworzonego grafu G'. Graf G możemy wtedy pokolorować na podstawie kolorowania G' podobnie jak w przypadku 4c.

Przypadek 6 (główna redukcja). Zauważmy, że ponieważ wykluczyliśmy poprzednie przypadki, to wszystkie ściany wewnętrzne G mają długość 5, ściana zewnętrzna ma długość od 4 do 6 oraz G nie zawiera wierzchołków stopnia 1. Implikuje to, że gdyby G zawierał wierzchołek rozcinający to G zawierałby trójkąt lub ścianę długości co najmniej 8. Zatem G jest dwuspójny. Co więcej, wszystkie wierzchołki w G mają stopnie co najmniej 3, z wyjątkiem, być może, wierzchołków marszruty ściany zewnętrznej. Nie istnieje też para wierzchołków stopnia 2 połączonych krawędzią, gdyż musiałyby one być incydentne ze ścianą zewnętrzną, a wtedy marszruta 5-ściany wewnętrznej incydentnej z tymi wierzchołkami zawierałaby cięciwę cyklu C lub niepokolorowany wierzchołek sąsiadujący z dwoma pokolorowanymi wierzchołkami (przypadki 2 i 3, odpowiednio). Widzimy, że spełnione są założenia lematu 2.2.1, a więc graf G zawiera cykl ścianowy $C' = x_1x_2x_3x_4x_5x_1$ taki, że

$\deg(x_1) = \deg(x_2) = \deg(x_3) = \deg(x_4) = 3$, $\deg(x_5) \leq 5$ oraz $V(C) \cap V(C') = \emptyset$. W szczególności wierzchołki x_i nie są pokolorowane. Oznaczmy przez y_i sąsiada wierzchołka x_i w grafie $G - V(C')$, dla $i = 1, 2, 3, 4$. Ponadto, niech $y_5, \ldots y_m$ będą sąsiadami x_5 w $G - V(C')$.



Rysunek 2.4: Przypadki 6a (z lewej) i 6b (z prawej)

Przypadek 6a. $y_i = y_j$ dla pewnych $i \neq j$ (patrz rysunek 2.4). Ponieważ graf G nie zawiera trójkątów, x_i i x_j leżą w odległości 2 na cyklu C'. Niech x_k będzie wspólnym sąsiadem x_i i x_j leżącym na cyklu C'. Wtedy 4-cykl $x_i y_i x_j x_k x_i$ jest separujący, ponieważ wykluczyliśmy przypadek 5. Możemy więc zakończyć dowód analogicznie jak w przypadku 4a.

Przypadek 6b. $y_i y_j \in E(G)$ dla pewnych $i \neq j$ (patrz rysunek 2.4). Wtedy graf G zawiera cykl separujący długości 4 lub 5 i ponownie możemy postąpić podobnie jak w przypadku 4a.

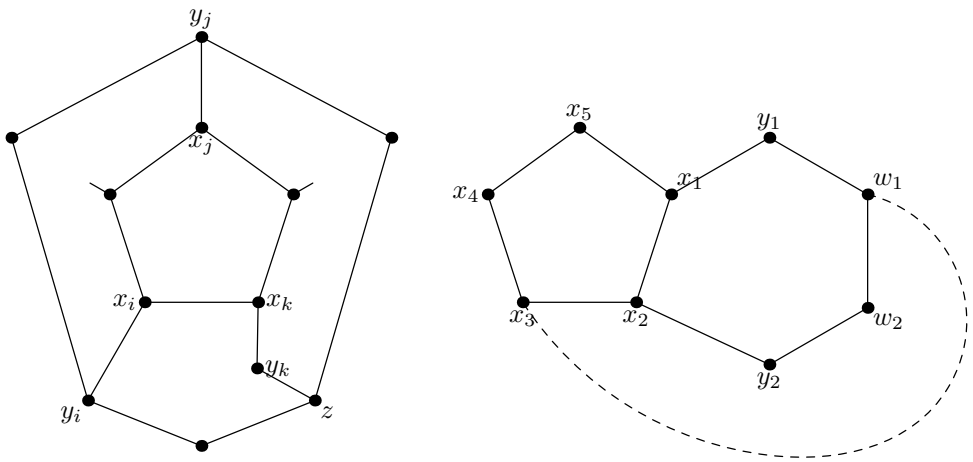
Przypadek 6c. Istnieją trzy różne wierzchołki $x_i, x_j, x_k \subset \{x_1, \ldots, x_5\}$ takie, że każdy z nich ma pokolorowanego sąsiada. Wtedy co najmniej jeden z cykli grafu $G[V(C) \cup \{x_i, x_j, x_k\}]$ jest cyklem separującym długości od 4 do 6 i znów możemy postąpić tak jak w przypadku 4a.

Przez symetrię możemy założyć, że jeden z wierzchołków y_1, y_2 nie jest pokolorowany (tzn. jeśli tak nie jest to zmieniamy oznaczenia wierzchołków $x_1, \ldots, x_4$ oraz $y_1, \ldots, y_4$).

Przypadek 6d. Dla różych indeksów $i, j, k \in \{1, \ldots, 5\}$ wierzchołki y_i, y_j i z są pokolorowane, gdzie z jest sąsiadem y_k. Ponadto, y_i i z mają ten sam kolor oraz x_i jest sąsiadem x_k (patrz rysunek 2.5). Ponieważ y_i i z mają ten sam kolor, to ich odległość na cyklu C wynosi co najmniej 2. Ponownie widzimy, że jeden z cykli grafu $G[V(C) \cup V(C') \cup \{y_k\}]$ jest cyklem separującym o długości od 4 do 6 i postępujemy tak jak w przypadku 4a.

Przypadek 6e. y_2 i sąsiad y_1 mają ten sam kolor (albo y_1 i sąsiad y_2 mają ten sam kolor). Ponieważ wykluczyliśmy przypadek 6d y_3 i y_4 nie są pokolorowane. Zamieniamy wtedy rolami wierzchołki x_1 i x_4 oraz x_2 i x_3. Zauważmy, że gdy przypadek 6e nie stosuje się już do G, utożsamienie wierzchołków y_1 i y_2 nie powoduje powstania krawędzi o obu końcach tego samego koloru.

Przypadek 6f. y_3 jest pokolorowany i x_5 ma pokolorowanego sąsiada. Ponieważ



Rysunek 2.5: Przypadki 6d (z lewej) i 6g (z prawej).

wykluczyliśmy przypadki 6c i 6d, więc y_2 nie jest pokolorowany oraz y_4 nie ma sąsiada o takim samym kolorze jak y_3. Zamieniamy wtedy rolami wierzchołki x_1 i x_4 oraz x_2 i x_3. Dzięki temu możemy założyć, że utożsamienie y_1 z y_2 oraz y_3 z x_5 nie powoduje powstania krawędzi o obu końcach tego samego koloru.

Przypadek 6g. Istnieje ścieżka $y_1w_1w_2y_2$ różna od $y_1x_1x_2y_2$ (patrz rysunek 2.5). Wtedy rozważamy cykl $C' = y_1w_1w_2y_2x_2x_1y_1$. Ponieważ G nie zawiera trójkątów, $\deg x_1 = \deg x_2 = 3$, oraz $y_1y_2 \notin E(G)$, więc cykl C' nie ma cięciw w G. Skoro $|C| = 6$ i wykluczyliśmy już przypadek 4, możemy założyć, że C' jest cyklem separującym i postąpić analogicznie jak w przypadku 4a. Jedyna różnica polega na tym, że tym razem próbujemy połączyć x_2 z w_1 aby po skorzystaniu z założenia indukcyjnego otrzymać bezpieczne kolorowanie C'. Jeżeli jednak istnieje 2-ścieżka w $G - V(\mathrm{int}(C'))$ łącząca x_2 z w_1, nie możemy połączyć tych wierzchołków, bo spowodowałoby to powstanie trójkąta. Ponieważ $\deg x_2 = 3$ oraz G nie zawiera trójkątów, taką 2-ścieżką jest $x_2x_3w_1$. Wtedy 5-cykl $x_1x_2x_3w_1y_1x_1$ jest separujący i możemy postąpić jak w przypadku 4a.

Przypadek 6h. Istnieje ścieżka $x_5y_kwy_3$ dla pewnego $k \in \{5, \ldots, m\}$. Rozważamy wtedy cykl $C' = y_kx_5x_4x_3wy_k$ i postępujemy podobnie jak w przypadku 6g.

Niech G' będzie grafem otrzymanym z G poprzez usunięcie wierzchołków x_1, x_2, x_3, x_4 oraz utożsamienie x_5 z y_3 oraz y_1 z y_2. Ponieważ wykluczyliśmy przypadki 6c-6h, G' nie zawiera trójkątów i graf indukowany przez pokolorowane wierzchołki G' jest pokolorowany poprawnie. Możemy więc skorzystać z założenia indukcyjnego aby otrzymać 3-kolorowanie c grafu G'. Następnie rozszerzamy c do 3-kolorowania grafu G. Ponieważ przypadek 6b został wykluczony, (częściowe) kolorowanie G odziedziczone z G' jest poprawne. Jeżeli $c(y_1) = c(x_5)$, to kolorujemy kolejno wierzchołki x_4, x_3, x_2, x_1, za każdym razem używając wolnego koloru. Jeżeli natomiast $c(y_1) \neq c(x_5)$, to kładziemy $c(x_2) = c(x_5)$ i kolorujemy kolejno wierzchołki x_4, x_3, x_1, za każdym razem

używając wolnego koloru. To kończy dowód. ■

2.3 Algorytm

Dowód twierdzenia Grötzscha przedstawiony w poprzednim podrozdziale będziemy traktować jak schemat algorytmu. Ponieważ dowód jest indukcyjny, w naturalny sposób otrzymujemy algorytm rekurencyjny. Zastanówmy się jakie wnioski, ważne dla algorytmu 3-kolorowania, możemy wysnuć z przedstawionego dowodu. Choć dowód i odpowiadający mu algorytm są dość skomplikowane[2], zauważmy, że liczba istotnie różnych operacji, które muszą zostać zaimplementowane jest niewielka. Możemy wymienić takie czynności jak usuwanie wierzchołków i krawędzi, wstawianie krawędzi, utożsamianie par wierzchołków i wyszukiwanie ścieżek o długości od 1 do 3 łączących zadane pary wierzchołków. Należy zbadać, w jaki sposób możemy te operacje zrealizować i jaka będzie ich złożoność czasowa. Musimy zastanowić się także, w jaki sposób efektywnie stwierdzać zachodzenie kolejnych przypadków oraz jak zrealizować rekursję, w szczególności w sytuacji, gdy procedura rekurencyjna jest wywoływana dla pewnego fragmentu grafu, musimy rozważyć jak szybko budować struktury danych opisujące wybrany fragment. W tym rozdziale odpowiemy na postawione tu pytania. Zaczniemy od opisania struktur danych używanych w algorytmie. Następnie zastanowimy się, w jaki sposób efektywnie korzystać z rekursji i jak implementować nietrywialne operacje pojawiające się w algorytmie. Odtąd, aż do końca rozdziału, G odnosi się do grafu danego na wejściu procedury rekurencyjnej, natomiast n jest liczbą wierzchołków grafu wejściowego.

2.3.1 Struktury danych

Graf dany na wejściu, listy sąsiedztwa. Bez straty ogólności możemy założyć, że graf wejściowy jest spójny, gdyż w przeciwnym przypadku możemy znaleźć spójne składowe w czasie liniowym za pomocą algorytmu przeszukiwania w głąb (DFS) i uruchomić algorytm 3-kolorowania osobno dla każdej składowej. Zakładamy, że graf jest dany na wejściu w postaci list sąsiedztwa. Oczekujemy ponadto, że listy sąsiedztwa są zgodne z pewnym włożeniem kombinatorycznym grafu, tzn. że sąsiedzi każdego wierzchołka v pojawiają się na odpowiedniej liście sąsiedztwa w takim samym porządku jak prowadzące do nich krawędzie w pewnym włożeniu płaskim grafu, podane w kolejności zgodnej z ruchem wskazówek zegara.

Ściany i kolejki ścian. Zauważmy, że korzystając z włożenia kombinatorycznego grafu zapisanego w listach sąsiedztwa możemy łatwo w czasie liniowym znaleźć wszystkie ściany danego na wejściu grafu płaskiego. Ponieważ jest to graf spójny, każda

[2]Warto zaznaczyć, że najprostsze znane dowody nie są istotnie krótsze.

ściana odpowiada pewnej marszrucie ścianowej. Dla każdej krawędzi uv istnieją co najwyżej dwie ściany incydentne z uv. Ściana f jest *prawą ścianą incydentną z* (u, v), gdy u poprzedza v w ciągu wierzchołków marszruty ścianowej f podanych w kolejności zgodnej z ruchem wskazówek zegara. W przeciwnym przydapku f jest *lewą ścianą incydentną z* (u, v). Każdą ścianę f grafu przechowujemy jako właściwą dla niej marszrutę ścianową, tj. listę wskaźników do elementów list sąsiedztwa odpowiadających kolejnym krawędziom marszruty podanych w kolejności zgodnej z ruchem wskazówek zegara. Dla każdego takiego elementu e odpowiadającego sąsiadowi v wierzchołka u, ściana f jest prawą ścianą incydentną z (u, v). Dodatkowo, element e przechowuje wskaźnik do ściany f. Z każdą ściana przechowywana jest także jej długość, tzn. długość odpowiedniej marszruty ścianowej.

W naszym algorytmie będziemy używać trzech kolejek $Q_4, Q_5, Q_{\geq 6}$, przechowujących ściany o długościach, odpowiednio, 4, 5 i co najmniej 6, spełniające warunki opisane w przypadkach 5, 6 i 4 dowodu twierdzenia 2.2.2.

Kolejka wierzchołków niskiego stopnia. W celu szybkiego wykrywania przypadku 1, będziemy utrzymywać kolejkę przechowującą wierzchołki stopnia co najwyżej 2.

Wyrocznia krótkich ścieżek. W celu szybkiego wyszukiwania najkrótszych ścieżek w grafie użyjemy wyroczni opisanej w rozdziale 1. Choć w algorytmie niezbędne jest wyszukiwanie ścieżek o długościach od 1 do 3, okaże się, że wystarczy w tym celu zbudować graf 2-skrótów. Można sprawdzić, że za każdym razem interesuje nas *najkrótsza ścieżka*, ponieważ zawsze gdy szukamy ścieżki długości 2 wiemy już, że dana para wierzchołków nie jest połączona krawędzią. Nie będziemy więc potrzebować rozbudowanej wersji wyroczni opisanej w podrozdziale 1.7. Będziemy jednak używać wyroczni działającej w środowisku dynamicznym, opisanej w podrozdziale 1.6.

Warto odnotować poniższy, oczywisty fakt.

Fakt 2.3.1. *Wszystkie struktury danych używane w algorytmie* 3*-kolorowania możemy utworzyć w czasie liniowym na podstawie grafu danego na wejściu.*

2.3.2 Rekursja

Zauważmy, że w algorytmie rekurencyjnym wynikającym z dowodu podanego w podrozdziale 2.2 pojawia się konieczność rozdzielenia grafu G na dwie części. Po podziale, każda z części jest przetwarzana osobno przez uruchamiane kolejno wywołania rekurencyjne. Przez rozdzielanie grafu rozumiemy tu podział informacji przechowywanych w listach sąsiedztwa i innych strukturach danych opisanych w poprzednim podrozdziale. Ponieważ pesymistyczna liczba wywołań rekurencyjnych wynosi $\Theta(n)$, naiwne podejście (tzn. budowanie odpowiednich struktur danych od nowa) do tego problemu wymagałoby poświęcenia na dzielenie grafu całkowitego czasu rzędu $\Theta(n^2)$. Zamiast

tego, przed podzieleniem informacji dotyczących G, nasz algorytm znajduje mniejszą ze wspomnianych dwóch części. Możemy to łatwo zrealizować przez dwa wywołania algorytmu DFS, przeszukujące obie części w sposób „równoległy". Dokładniej, za każdym razem gdy znajdziemy wierzchołek w jednej części, zawieszamy przeszukiwanie i wznawiamy działanie algorytmu DFS w drugiej części. Takie podejście pozwala na znalezienie mniejszej części A w czasie liniowym ze względu na rozmiar A. Oznaczmy drugą, większą część przez B. Mając dany zbiór wierzchołków A łatwo, w czasie $\mathcal{O}(|V(A)|)$, rozdzielamy listy sąsiedztwa, kolejki ścian i kolejkę wierzchołków niskiego stopnia. Wierzchołki i krawędzie cyklu (lub ścieżki), które są wspólne dla obu części są kopiowane i kopie umieszczane są w strukturach danych odpowiadających części A. Zwróćmy uwagę, że liczba takich wierzchołków nie przekracza 6 (podobnie liczba krawędzi). W dalszej kolejności usuwamy wszystkie wierzchołki zbioru $V(A) \setminus V(B)$ z wyroczni krótkich ścieżek. Nazwiemy tę czynność operacją Cut. W wyniku operacji Cut otrzymujemy wyrocznię krótkich ścieżek dla B. Wyrocznia dla A jest budowana od nowa, na podstawie list sąsiedztwa A. Jak pamiętamy z rozdziału 1, usuwanie wierzchołka ze struktury wyroczni wykonuje się w czasie $\mathcal{O}(1)$, natomiast budowa nowej wyroczni zajmie czas $\mathcal{O}(|A|)$. Podsumowując, cała operacja rozdzielenia grafu zajmuje czas $\mathcal{O}(|A|)$.

Lemat 2.3.2. *Całkowity czas zużyty przez algorytm na dzielenie struktur danych przed wywołaniami rekurencyjnymi wynosi* $\mathcal{O}(n \log n)$.

Dowód. Możemy założyć, że graf jest zawsze dzielony na dwie części – sytuację opisaną w przypadku 2 dowodu, gdzie mogą pojawić się trzy części możemy traktować jak dwa kolejne podziały. Wierzchołki wspólne dla obu części nazwiemy *wierzchołkami zewnętrznymi*, natomiast pozostałe wierzchołki leżące w mniejszej części A nazwiemy *wierzchołkami wewnętrznymi*. Całkowity czas zużyty na dzielenie struktur danych jest liniowy ze względu na sumaryczną liczbę wierzchołków wewnętrznych i zewnętrznych. Ponieważ mamy $\mathcal{O}(n)$ podziałów, a przy każdym z nich występuje co najwyżej 6 wierzchołków zewnętrznych, ich całkowita liczba wynosi $\mathcal{O}(n)$. Ponadto, podczas podziału grafu k-wierzchołkowego, mamy do czynienia z co najwyżej $\lfloor k/2 \rfloor$ wierzchołkami wewnętrznymi. To implikuje, że każdy wierzchołek wejściowego grafu może stać się wierzchołkiem wewnętrznym nie więcej niż $\log n$ razy. Wnioskujemy, że całkowita liczba wierzchołków wewnętrznych nie przekracza $\mathcal{O}(n \log n)$. ■

2.3.3 Nietrywialne operacje

Utożsamianie wierzchołków. Opiszemy tutaj, w jaki sposób nasz algorytm aktualizuje struktury danych wymienione w punkcie 2.3.1 podczas wykonywania operacji utożsamienia pary wierzchołków u, v. Utożsamienie dwóch wierzchołków może być wykonane za pomocą ciągu operacji usuwania i wstawiania wierzchołków i krawędzi

grafu. Dokładniej, operacja utożsamienia IDENTIFY(u,v) jest wykonywana za pomocą następującego algorytmu. Zaczynamy od porównania stopni wierzchołków u i v w bieżącym grafie G. Załóżmy, że $\deg_G(u) \leq \deg_G(v)$. Jeśli tak nie jest zamieniamy oznaczenia wierzchołków. Następnie dla każdego sąsiada x wierzchołka u usuwamy krawędź ux i wstawiamy nową krawędź vx, chyba że jest ona już obecna w grafie.

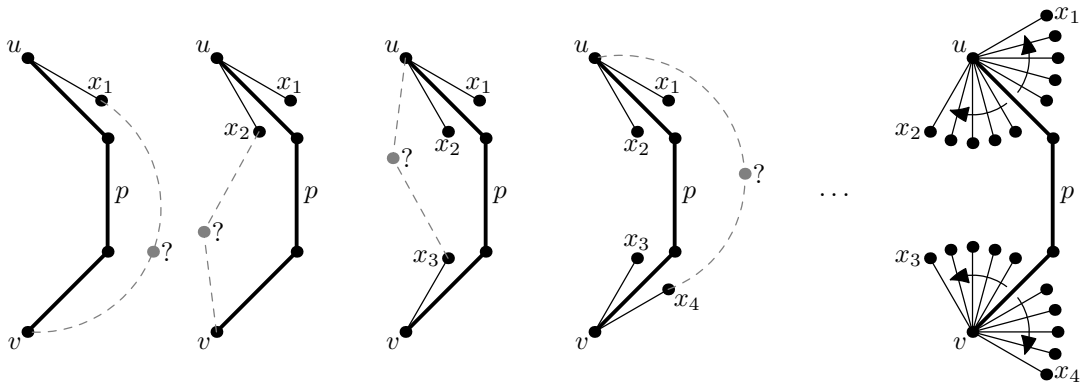
Lemat 2.3.3. *Całkowita liczba par operacji wstawiania/usuwania krawędzi wykonywanych w algorytmie* 3*-kolorowania przez algorytm* IDENTIFY *jest ograniczona przez* $\mathcal{O}(n \log n)$.

Dowód. Załóżmy na chwilę, że algorytm 3-kolorowania nie wykonuje innych usunięć niż te związane z operacją utożsamiania wierzchołków. Czynność polegającą na usunięciu krawędzi ux i wstawieniu krawędzi vx podczas wykonywania operacji IDENTIFY będziemy nazywać *przenoszeniem krawędzi* ux. Widzimy, że każda krawędź grafu wejściowego może zostać przeniesiona co najwyżej $\lceil \log n \rceil$ razy, bowiem w przeciwnym przypadku pojawiłby się wierzchołek stopnia większego niż n. Wnioskujemy, że całkowita liczba par operacji wstawiania/usuwania krawędzi związanych z wykonywaniem algorytmu IDENTIFY wynosi $\mathcal{O}(n \log n)$. Jest jasne, że wartość ta nie może wzrosnąć, jeśli rozważymy dodatkowe usunięcia krawędzi wykonywane przez algorytm. ■

Ponieważ w naszym algorytmie zawsze utożsamiane są wierzchołki z tej samej marszruty ścianowej, możemy łatwo zaktualizować listy sąsiedztwa tak, aby w dalszym ciągu reprezentowały włożenie grafu w płaszczyznę. Nietrudno także w czasie stałym uaktualnić informacje dotyczące ścian grafu. W przypadku 4c pewna marszruta ścianowa skraca się o 2 elementy, w przypadku 5b z grafu znika ściana o długości 4, natomiast w przypadku 6 rozdzielamy w czasie stałym odpowiednią marszrutę ścianową tworząc dwie marszruty. Każdą ze zmodyfikowanych marszrut ścianowych umieszczamy we właściwej kolejce, jeżeli spełnione są odpowiednie warunki. Lemat 2.3.3 implikuje, że na wszystkie te aktualizacje potrzebujemy łącznie czasu $\mathcal{O}(n \log n)$. W podrozdziale 2.4 pokażemy, w jaki sposób efektywnie aktualizować strukturę wyroczni krótkich ścieżek po IDENTIFY. Podsumowując, całkowity czas związany z modyfikowaniem struktur danych po utożsamianiu wierzchołków można oszacować przez $\mathcal{O}(n \log n)$.

Wyszukiwanie krótkich ścieżek. W naszym algorytmie wyszukiwane są najkrótsze ścieżki długości od 1 do 3 łączące zadane pary wierzchołków. Zauważmy, że liczba wszystkich takich zapytań podczas wykonywania algorytmu wynosi $\mathcal{O}(n)$. Jak pamiętamy z rozdziału 1, zapytania o ścieżki o długościach 1 i 2 mogą zostać obsłużone w czasie stałym z pomocą grafu 2-skrótów. Pozostaje jeszcze zająć się ścieżkami długości 3. Nie chcemy korzystać z grafu 3-skrótów, ponieważ nie wiadomo, czy czas poświęcony przez algorytm na aktualizowanie takiej struktury danych po modyfikacjach grafu, można oszacować przez $\mathcal{O}(n \log n)$.

Opiszemy algorytm PATH3, którego będziemy używać do wyszukiwania 3-ścieżki łączącej parę wierzchołków u,v. Możemy założyć, że mamy daną ścieżkę p długości 2 (przypadki 4b i 5a) lub długości 3 (przypadek 6g, wtedy oczywiście szukamy innej ścieżki) łączącą u i v. Bez straty ogólności możemy założyć, że $\deg(u) \leq \deg(v)$. Niech $A(u), A(v)$ oznaczają, odpowiednio, listy sąsiedztwa wierzchołków u i v. Zaczynamy od przypisania zmiennym x_1, x_2 tych elementów listy sąsiedztwa $A(u)$, które odpowiadają krawędziom ścieżki p incydentnym z u. Podobnie, przypisujemy zmiennym x_3, x_4 te elementy listy sąsiedztwa $A(v)$, które odpowiadają krawędziom ścieżki p incydentnym z v. Następnie zaczyna się pętla. Wewnątrz pętli przypisujemy na x_1 element listy $A(u)$, który poprzedza element dotychczas przechowywany w x_1. Podobnie na x_2 przypisujemy element następujący po x_2 w liście $A(u)$. Analogicznie, x_3 przechodzi na poprzedzający element, a x_4 na następny element w liście $A(v)$ (patrz rysunek 2.6).



Rysunek 2.6: Kolejność wykonywania zapytań w algorytmie PATH3.

W tym momencie używamy wyroczni krótkich ścieżek, szukając ścieżek o długości 2 łączących x_1 z v, x_2 z v, x_3 z u oraz x_4 z u. Jeżeli wyrocznia znajdzie jedną z tych ścieżek, łatwo przedłużamy ją o jedną krawędź (odpowiednio ux_1, ux_2, vx_3 lub vx_4) i przerywamy działanie algorytmu PATH3 zwracając otrzymaną ścieżkę. Jeżeli wyrocznia nie znajdzie żadnej ścieżki, wykonywany jest kolejny obrót pętli, chyba że sprawdzono już wszystkich sąsiadów u.

Lemat 2.3.4. *Całkowity czas poświęcony na wykonywanie algorytmu* PATH3 *podczas całego algorytmu* 3*-kolorowania grafu wynosi* $\mathcal{O}(n \log n)$.

Dowód. Podzielmy wszystkie wywołania operacji PATH3 na dwie grupy. Jeżeli szukana 3-ścieżka została znaleziona, operację nazwiemy *pomyślną*, a w przeciwnym przypadku *chybioną*. Pamiętamy, że gdy szukana ścieżka łącząca u i v nie istnieje, wierzchołki u i v zostają utożsamione. Ponieważ złożoność czasowa pojedynczego wykonania algorytmu PATH3(u,v), dla $\deg(u) \leq \deg(v)$, wynosi $\mathcal{O}(\deg u)$ i wszystkie krawędzie incydentne z u zostają usunięte podczas wykonania IDENTIFY(u,v), więc

całkowity czas spędzony na wykonywaniu chybionych operacji PATH3 jest liniowy w stosunku do liczby usunięć krawędzi spowodowanych utożsamieniami wierzchołków. Tymczasem lemat 2.3.3 gwarantuje, że jest $\mathcal{O}(n \log n)$ takich usunięć.

Pozostaje jedynie oszacować czas potrzebny na wykonanie pomyślnych operacji. Rozważmy jedną z nich, wywołaną dla wierzchołków u i v połączonych ścieżką p, $\deg(u) \leq \deg(v)$. Niech C' będzie cyklem separującym złożonym ze ścieżki p i znalezionej ścieżki o długości 3. Niech H będzie grafem $C' \cup \text{int}(C')$. Pamiętamy z dowodu twierdzenia 2.2.2, że jeden z wierzchołków u i v nie jest pokolorowany, powiedzmy v. Liczba zapytań zadanych wyroczni nie przekracza $4 \cdot \deg_H(v)$. Zauważmy, że v zostanie pokolorowany w grafie H, tuż przed wywołaniem rekurencyjnym dla H. W takim razie każdej krawędzi pojawiającej się w grafie G możemy przyporządkować co najwyżej 8 zapytań (po 4 dla każdego z końców) w taki sposób, że każde z zapytań pojawiających się w trakcie wykonywania pomyślnych operacji PATH3 jest przypisane pewnej krawędzi. Tymczasem, licząc nawet krawędzie pojawiające się przy okazji realizowania operacji IDENTIFY, podczas całego wykonania algorytmu mamy $\mathcal{O}(n \log n)$ różnych krawędzi. Stąd czas poświęcony na przetwarzanie pomyślnych zapytań nie przekracza $\mathcal{O}(n \log n)$. ■

Usuwanie wierzchołka stopnia co najwyżej 3. W przypadkach 1 i 6 usuwamy wierzchołki stopnia 1, 2 lub 3. Całkowita liczba tego rodzaju operacji wynosi $\mathcal{O}(n)$. Ponieważ stopnie usuwanych wierzchołków są ograniczone, sumaryczny czas poświęcony na aktualizowanie list sąsiedztwa i wyroczni krótkich ścieżek jest rzędu $\mathcal{O}(n)$. Musimy także odświeżyć informacje związane ze ścianami grafu. Może się zdarzyć, że trzeba połączyć dwie lub trzy marszruty ścianowe w jedną. Tę operację można łatwo zaimplementować, aby działała w czasie stałym, sklejając odpowiednie listy. Pojawia się tu jednak pewien problem. Każda krawędź (tzn. element listy sąsiedztwa) w marszrucie ścianowej przechowuje wskaźnik do incydentnej z nią prawej ściany. Po połączeniu dwóch marszrut ich krawędzie dalej przechowują wskaźniki do dwóch różnych ścian. Dlatego używamy algorytmu FIND AND UNION (zobacz np. [16]) do łączenia dwóch ścian i do wyszukiwania prawej ściany incydentnej z daną krawędzią. Koszt zamortyzowany każdej z tych dwóch operacji można oszacować przez $O(\alpha(n))$, gdzie $\alpha(n)$ jest niezwykle wolno rosnącą odwrotnością funkcji Ackermanna i oczywiście $\alpha(n) = o(\log n)$. Ponieważ całkowita liczba wywołań tych operacji nie przekracza $\mathcal{O}(n)$, więc nie zmieniają one złożoności czasowej naszego algorytmu 3-kolorowania.

Wyszukiwanie ścian. Aby szybko znajdować ściany o własnościach określonych w przypadkach 5, 6, 4 dowodu twierdzenia 2.2.2 używamy, odpowiednio, kolejek $Q_4, Q_5, Q_{\geq 6}$. Kolejki są inicjalizowane w czasie liniowym, a wyszukiwanie elementu w kolejce trwa oczywiście czas stały.

Uwagi końcowe. Aby efektywnie rozpoznawać przypadki 2, 3, 4a, 6c-6f wystarczy, w przypadku gdy cykl ścianowy C ściany zewnętrznej jest pokolorowany, przekazywać C

przy wywołaniach rekurencyjnych. Wtedy sprawdzenie każdego z wymienionych przypadków zajmuje czas stały (przy niektórych z nich korzystamy z wyroczni krótkich ścieżek), ponieważ liczba pokolorowanych wierzchołków nigdy nie przekracza 6.

2.4 Graf skrótów w algorytmie kolorowania

Dla uproszczenia algorytmu możemy założyć, że do budowania $\overrightarrow{G_1}$ i $\overrightarrow{G_2}$ używany jest algorytm BuildDynamic opisany w podrozdziale 1.6.4. Przypomnijmy, że złożoność czasowa tego algorytmu jest liniowa ze względu na liczbę wierzchołków grafu, tak jak zakładaliśmy dowodząc lematu 2.3.2.

Podczas działania algorytmu konieczne jest uaktualnianie grafów skrótów po wykonaniu operacji wstawienia i usunięcia krawędzi, a także operacji utożsamienia wierzchołków. Na początek zastanówmy się, ile takich operacji jest wykonywanych. Przyjmijmy, że nie będziemy wliczać do całkowitej liczby wstawień tych operacji wstawiania, które wykonały się w ramach utożsamiania wierzchołków (analogiczne zakładamy o operacjach usunięcia krawędzi). Podobnie, nie musimy już przejmować się wstawieniami związanymi z budowaniem grafów skrótów oraz usunięciami związanymi z operacjami Cut, gdyż wiemy już z lematu 2.3.2, że ich złożoność czasowa wynosi $\mathcal{O}(n \log n)$.

Lemat 2.4.1. *Podczas działania algorytmu 3-kolorowania wykonywanych jest*

(i) $\mathcal{O}(n)$ *operacji wstawienia krawędzi (nie licząc wstawień użytych do budowy grafów skrótów przed wywołaniami rekurencyjnymi),*

(ii) $\mathcal{O}(n)$ *operacji usunięcia krawędzi (nie licząc usunięć w ramach operacji* Cut*),*

(iii) $\mathcal{O}(n)$ *operacji utożsamienia wierzchołków.*

Dowód. Operacje wstawiania krawędzi pojawiają się wtedy, gdy graf jest rozcinany na dwie części wzdłuż cyklu separującego (np. przypadek 4a). Takich rozcięć jest $\mathcal{O}(n)$ i na każde z nich przypada co najwyżej 1 operacja wstawienia krawędzi, a więc liczbę wstawień można oszacować przez $\mathcal{O}(n)$. Daje to od razu oszacowanie na liczbę usunięć, gdyż nie może ona przekraczać liczby krawędzi w początkowym grafie powiększonej o liczbę wstawień.

W naszym algorytmie pojawia się $\mathcal{O}(n)$ wywołań rekurencyjnych. Przy każdym takim wywołaniu tworzymy kopie nie więcej niż sześciu wierzchołków. Dlatego całkowita liczba wierzchołków we wszystkich utworzonych grafach skrótów może przekroczyć n, ale z drugiej strony można ją oszacować przez $\mathcal{O}(n)$. Ponieważ przy każdym utożsamieniu liczba wierzchołków zmniejsza się o 1, mamy $\mathcal{O}(n)$ utożsamień. ■

Oczywiście po każdej operacji wstawienia lub usunięcia krawędzi a także po operacji utożsamienia wierzchołka, uaktualniamy odpowiedni graf skrótów za pomocą operacji, odpowiednio, InsertShortcut, DeleteShortcut lub IdentifyShortcut. Jeżeli skorzystamy z wyników udowodnionych w rozdziale 1, to jako zamortyzowany czas aktualizowania grafu 2-skrótów po operacji wstawienia krawędzi musimy przyjąć $\mathcal{O}(\log^2 n)$. W rezultacie otrzymalibyśmy oszacowanie $\mathcal{O}(n \log^2 n)$ na całkowity czas działania algorytmu. Ten podrozdział poświęcimy dokładniejszemu oszacowaniu złożoności operacji wykonywanych na grafie skrótów. Pokażemy, że wykorzystując własności szczególnego ciągu operacji pojawiającego się w algorytmie 3-kolorowania możemy oszacować całkowity czas potrzebny na aktualizowanie grafu 2-skrótów przez $\mathcal{O}(n \log n)$.

Oszacujemy teraz liczbę zmian orientacji krawędzi w $\overrightarrow{G_1}$. Przyjrzyjmy się, jak funkcjonują grafy skrótów podczas wykonywania algorytmu kolorowania. Na początku, przed pewnym wywołaniem rekurencyjnym, budowany jest graf skrótów dla pewnego grafu H. Następnie w wyniku działania algorytmu, krawędzie i wierzchołki grafu H są usuwane. Ten proces kończy się, gdy w grafie przekazanym w pewnym wywołaniu rekurencyjnym nie ma już niepokolorowanych wierzchołków, a więc zostały jedynie pokolorowane wierzchołki, których liczba nie przekracza 6. Dla uproszczenia analizy będziemy zakładać, że wszystkie krawędzie grafów, na podstawie których tworzymy grafy skrótów są w końcu usuwane.

Lemat 2.4.2. *Niech σ będzie ciągiem operacji wstawiania/usuwania krawędzi i utożsamiania wierzchołków wykonywanych w grafie $\overrightarrow{G_1}$ przez algorytm kolorowania po utworzeniu grafu skrótów. Niech t będzie liczbą operacji wstawiania krawędzi w ciągu σ. Wtedy całkowita liczba zmian orientacji krawędzi w grafie $\overrightarrow{G_1}$, wykonanych w algorytmie* Reorient, *wynosi $\mathcal{O}(t)$.*

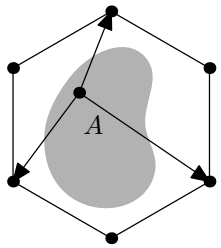
Dowód. Niech G^i będzie grafem G_1 po i-tej operacji. Zbudujemy ciąg 6-orientacji $\mathcal{G} = \overrightarrow{G^1}, \overrightarrow{G^2}, \ldots \overrightarrow{G^t}$. Zauważmy, że $\overrightarrow{G^t}$ nie ma krawędzi, a więc jest 6-zorientowany. Dla każdego $i = 1, \ldots, t-1$, opiszemy 6-orientację $\overrightarrow{G^i}$ korzystając z $\overrightarrow{G^{i+1}}$.

Jeżeli σ_{i+1} jest wstawieniem krawędzi uv, graf $\overrightarrow{G^i}$ otrzymujemy z $\overrightarrow{G^{i+1}}$ przez usunięcie uv.

Jeżeli $\sigma_{i+1} =$ Identify(u,v), to krawędzie incydentne z wierzchołkiem x powstałym w $\overrightarrow{G^{i+1}}$ po utożsamieniu dzielimy na dwie grupy w $\overrightarrow{G^i}$, nie zmieniając ich orientacji (pamiętamy, że u, v nie są sąsiednie w G^i). Oczywiście $\mathrm{outdeg}_{\overrightarrow{G^i}} u \leq \mathrm{outdeg}_{\overrightarrow{G^{i+1}}} x$ oraz $\mathrm{outdeg}_{\overrightarrow{G^i}} v \leq \mathrm{outdeg}_{\overrightarrow{G^{i+1}}} x$.

Pozostaje przypadek, gdy σ_{i+1} jest operacją usunięcia krawędzi. Rozpatrzmy najpierw przypadek, gdy nie jest to usunięcie związane z operacją Cut. Pamiętamy z dowodu twierdzenia 2.2.2, że takie usunięcie jest spowodowane usuwaniem wierzchołka stopnia co najwyżej 3. Oznaczmy ten wierzchołek przez x. Widzimy, że wystarczy zorientować usuwaną krawędź w grafie $\overrightarrow{G^i}$ tak, żeby wychodziła z x.

Zamiast rozważać pojedyncze usunięcie krawędzi związane z operacją CUT, weźmiemy pod uwagę od razu cały ciąg usunięć spowodowanych przez jedną taką operację. Załóżmy, że G^i i G^j są, odpowiednio, grafami przed i po wykonaniu operacji CUT. Niech A będzie grafem indukowanym przez wierzchołki usunięte z G^i. Ponieważ A jest grafem planarnym, więc na podstawie wniosku 1.3.5 można go 3-zorientować. Tak zorientowany graf dodajemy do grafu $\overrightarrow{G^j}$, wstawiając kolejne krawędzie. Aby otrzymać $\overrightarrow{G^i}$ wystarczy jeszcze dodać krawędzie łączące wierzchołki A z wierzchołkami z G^j. Każdą taką krawędź xy, dla $x \in V(A)$, orientujemy jako (x, y). Zauważmy, że graf A leży wewnątrz cyklu długości co najwyżej 6, a więc każdy wierzchołek A sąsiaduje z nie więcej niż trzema wierzchołkami G^j, gdyż w przeciwnym przypadku graf G^i musiałby zawierać trójkąt (patrz rysunek 2.7). Stąd $\overrightarrow{G^i}$ jest 6-zorientowany.



Rysunek 2.7: Wierzchołki grafu A sąsiadują z nie więcej niż 3 wierzchołkami G^j.

W ten sposób opisaliśmy pewien ciąg 6-orientacji $\overrightarrow{G^1}, \ldots, \overrightarrow{G^t}$. Zauważmy, że w tym ciągu nie ma nawet jednej zmiany orientacji krawędzi. Niech $\overrightarrow{G^0}$ będzie początkową orientacją grafu G_1. Wiemy, że graf $\overrightarrow{G^0}$ jest d-zorientowany dla pewnego $d = \mathcal{O}(1)$. Rozważmy teraz ciąg $\overrightarrow{G^0}, \ldots, \overrightarrow{G^t}$. Orientacje $\overrightarrow{G^0}$ i $\overrightarrow{G^1}$ mogą się znacznie różnić. Aby zlikwidować to zjawisko, modyfikujemy orientacje $\overrightarrow{G^1}, \ldots, \overrightarrow{G^t}$. Dla każdego $i = 1, \ldots, t$ każdą krawędź $uv \in E(G^i) \cap E(G^0)$ orientujemy w grafie $\overrightarrow{G^i}$ tak, jak była zorientowana w $\overrightarrow{G^0}$. W ten sposób otrzymamy ciąg $(d+6)$-orientacji $\overrightarrow{G^0}, \ldots, \overrightarrow{G^t}$. Zauważmy, że w tym ciągu nie ma zmian orientacji krawędzi. Korzystamy z lematu 1.6.12, kładąc $\delta = d+6$ i $D = 3\delta - 1$. Wnioskujemy, że liczba zmian orientacji krawędzi wykonywanych w algorytmie REORIENT podczas wykonywania naszego ciągu operacji nie przekracza $\mathcal{O}(t)$. ■

Lemat 2.4.3. *Całkowita liczba zmian orientacji krawędzi w grafie $\overrightarrow{G_2}$, wykonanych w algorytmie* REORIENT *podczas realizowania ciągu operacji zawierającego t wstawień krawędzi i utożsamień wierzchołków, wykonanego na grafie n-wierzchołkowym wynosi $\mathcal{O}(t \log n)$.*

Dowód. Graf G_2 jest n-wierzchołkowym grafem o lesistości $\mathcal{O}(1)$. Algorytmy INSERT i GLUE uruchamiane podczas wstawiania wierzchołków i utożsamiania krawędzi utrzymują $\mathcal{O}(1)$-orientację grafu $\overrightarrow{G_2}$. Z lematu 2.4.2 wnioskujemy, że w grafie G_2 wykonywanych jest $\mathcal{O}(t)$ wstawień krawędzi za pomocą algorytmu INSERT. Wszystkie te

fakty, łącznie z lematem 1.6.13 implikują, że całkowita liczba zmian orientacji w grafie $\overrightarrow{G_2}$ wynosi $\mathcal{O}(t \log n)$. ■

Wniosek 2.4.4. *Całkowity czas poświęcony przez algorytm 3-kolorowania na uaktualnianie informacji w grafach skrótów można oszacować przez* $\mathcal{O}(n \log n)$.

Dowód. Lematy 2.4.1, 2.4.2 i 2.4.3 implikują, że całkowity czas poświęcony na wykonywanie operacji REORIENT można oszacować przez $\mathcal{O}(n \log n)$. Oszacujmy teraz czas potrzebny na uaktualnianie grafów skrótów, z wyłączeniem czasu zajętego przez operacje REORIENT. Operacje wstawiania krawędzi zajmą wtedy czas liniowy. Z lematu 2.3.2 wiemy już, że operacje usuwania krawędzi związane z wykonywaniem CUT zajmują czas $\mathcal{O}(n \log n)$, natomiast lemat 2.4.1 implikuje, że pozostałe usunięcia zajmą jedynie czas liniowy. Podczas wykonywania operacji IDENTIFYSHORTCUT(u,v, wykonujemy dwukrotnie operację GLUE, raz w grafie G_1, drugi raz w grafie G_2. Oprócz wywołania procedury REORIENT, algorytm GLUE wykonuje pewną liczbę par operacji usunięcia i wstawienia krawędzi w grafach G_1 i G_2. Każda taka para wykonuje się oczywiście w czasie stałym. Z lematu 2.3.3 wiemy, że całkowita liczba par usunięć i wstawień wykonanych w grafie G (a więc także w G_1) nie przekracza $\mathcal{O}(n \log n)$. Pozostaje jeszcze oszacować liczbę takich par operacji wykonanych w G_2. Liczba par usunięć i wstawień pojawiających się podczas wykonywania operacji GLUE(u,v) w grafie G_1 wynosi $\mathcal{O}(\deg_{G_1}(u))$. Dla grafu G_2 będzie to natomiast $\mathcal{O}(\deg_{G_2}(u))$. Na podstawie lematu 1.6.18 mamy jednak $\mathcal{O}(\deg_{G_2}(u)) = \mathcal{O}(\deg_{G_1}(u))$. Dzięki temu całkowita liczba par usunięć i wstawień w G_2 także nie przekracza $\mathcal{O}(n \log n)$. To kończy dowód lematu. ■

2.5 Podsumowanie

W niniejszym rozdziale przedstawiliśmy algorytm 3-kolorowania grafów planarnych bez trójkątów. Nasz algorytm działa w czasie niemal liniowym, t.j. $\mathcal{O}(n \log n)$. Pojawia się naturalne pytanie o algorytm liniowy dla tego problemu. Sytuacja wydaje się tu podobna do pytania o efektywny algorytm dla 4-kolorowania grafów planarnych, gdzie znany jest algorytm o złożoności $\mathcal{O}(n^2)$. Tak jak już pisaliśmy we wstępie, wydaje się, że uzyskanie bardziej efektywnego algorytmu wymaga istotnie nowego dowodu twierdzenia o czterech barwach. Analogicznie przypuszczamy, że uzyskanie liniowego algorytmu 3-kolorowania grafów bez trójkątów wymaga istotnie nowego dowodu twierdzenia Grötzscha. Carsten Thomassen przedstawił ostatnio nowy dowód twierdzenia mówiącego, że każdy graf planarny bez trójkątów i 4-cykli jest 3-kolorowalny [52]. Wydaje się, że na podstawie tego dowodu można skonstruować dość skomplikowany, ale liniowy, algorytm 3-kolorowania takich grafów. Thomassen [53] nie wie jednak, czy można napisać analogiczny dowód dla grafów, które mogą zawierać cykle długości 4.

Rozdział 3

Krótkie cykle

3.1 Wstęp

Problem wyszukiwania podgrafu izomorficznego (ang. *graph isomorphism problem*) polega na określeniu, czy dany graf G, zwany „tekstem", zawiera inny graf H, zwany „wzorcem"[1]. Jest to jeden z najważniejszych i najbardziej naturalnych problemów algorytmicznej teorii grafów, znajdujący także wiele zastosowań. W niniejszej rozprawie koncentrujemy się na sytuacji gdy graf G jest planarny i rozważamy jedną z najbardziej naturalnych klas wzorców – cykle. Przez wielomianową redukcję problemu cyklu Hamiltona nietrudno jest wykazać, że nawet tak ograniczona wersja problemu jest NP-zupełna. Niemniej jednak, możemy traktować wzorzec jako ustalony graf i badać złożoność algorytmów jedynie w zależności od rozmiaru grafu tekstu. David Eppstein [22] przedstawił algorytm o złożoności liniowej, znajdujący dowolny ustalony wzorzec w grafie planarnym. Niestety, dla wzorca zawierającego w wierzchołków algorytm ten w czasie swego działania generuje $\Omega(w^{3w+9})$ obiektów kombinatorycznych. Dlatego, pomimo ogromnego znaczenia teoretycznego, uwzględniając możliwości współczesnych komputerów, algorytm Eppsteina nie nadaje się do zastosowania w praktyce nawet dla wzorców 4-wierzchołkowych. Daje to podstawę aby sądzić, że poszukiwanie efektywnych algorytmów dla konkretnych, wybranych wzorców jest w dalszym ciągu interesującym i ważnym obszarem badawczym.

3.1.1 Wcześniejsze prace

Problem wyszukiwania cykli określonych długości w grafach planarnych przyciągał uwagę wielu badaczy. Papadimitriou i inni [46] podali pierwszy liniowy, ale skomplikowany algorytm dla znajdowania 3-cykli. Dwa różne liniowe, a zarazem proste w implementacji algorytmy zaproponowali następnie Chiba i Nishizeki [11] oraz Chrobak i Eppstein [14]. W pierwszym z tych artykułów znaduje się również prosty liniowy

[1] Precyzyjnie należałoby powiedzieć „zawiera podgraf izomorficzny z H".

algorytm wyszukujący 4-cykle w grafach planarnych. Oba algorytmy z tego artykułu mogą zostać zastosowane również dla grafów d-zdegenerowanych. Jeśli taki graf ma m krawędzi, złożoność czasową obu algorytmów można oszacować przez $\mathcal{O}(d \cdot m)$. W dalszej kolejności Richards [47] podał algorytm o złożoności $O(n \log n)$ na znajdowanie w grafie planarnym cykli C_5 i C_6. Alon i inni [4] przedstawili algorytm wyszukujący 5-cykl (tylko jedno wystąpienie) w grafie d-zdegenerowanym o m krawędziach, działający w czasie $\mathcal{O}(d^2 \cdot m)$, co daje czas liniowy dla grafów planarnych.

Oprócz wyniku Eppsteina [22] znanych jest jeszcze kilka algorytmów służących do wyszukiwania cykli *dowolnej* długości w grafach planarnych. Monien [44] podał algorytm działający w czasie $O(m \cdot n)$ dla dowolnego grafu o n wierzchołkach i m krawędziach. Alon i inni [3] zaprojektowali randomizowany algorytm dla grafów planarnych działający w oczekiwanym czasie liniowym i algorytm deterministyczny o pesymistycznym czasie $\mathcal{O}(n \log n)$. Pamiętamy również, że graf skrótów, opisany w rozdziale 1, może być użyty do skonstruowania liniowego algorytmu wyszukiwania cykli dowolnej długości. Niestety jeśli przyjrzymy się zależności czasu działania tego algorytmu od długości wyszukiwanego cyklu, okaże się że jest ona podobna do złożoności algorytmu Eppsteina.

3.1.2 Nasze wyniki

Nowe podejście. Przedstawimy nowe podejście do problemu wyszukiwania krótkich cykli w grafach planarnych. Pokażemy, że nasza metoda pozwala uzyskać algorytmy liniowe w przypadku cykli o długościach od 3 do 6. Dokładniej, dla każdego $k = 3, \ldots, 6$ pokażemy algorytm, który w czasie liniowym znajduje k-cykl w danym grafie planarnym lub stwierdza, że takiego cyklu nie ma. Pokażemy także, w jaki sposób rozszerzyć każdy z tych algorytmów tak, aby znajdował wszystkie cykle w n-wierzchołkowym grafie planarnym w czasie liniowym ze względu na rozmiar wyjścia, tj. $\mathcal{O}(n + \#_c)$, gdzie $\#_c$ jest liczbą cykli w grafie. Co więcej, algorytmy wyszukujące cykle długości 3, 4 i 5 działają także dla grafów d-zdegenerowanych. Złożoności tych algorytmów wynoszą wtedy odpowiednio $\mathcal{O}(d^2 \cdot m)$ dla znajdowania jednego cyklu oraz $\mathcal{O}(d^2 \cdot m + \#_c)$ w przypadku gdy znajdujemy wszystkie cykle w grafie.

Liczba cykli. Jako efekt uboczny naszej metody, w podrozdziale 3.3 udowodnimy, że maksymalna liczba k-cykli w n-wierzchołkowym grafie planarnym wynosi $\Theta(n^{\lfloor k/2 \rfloor})$, gdzie k traktujemy jako stałą. Jest to nie tylko interesujący wynik kombinatoryczny, lecz również oszacowanie złożoności czasowej naszych algorytmów w przypadku, gdy wyszukujemy *wszystkie* cykle w grafie. Wcześniej podobny problem kombinatoryczny rozważał David Eppstein. Pokazał on [21], że graf H występuje $\mathcal{O}(n)$ razy jako podgraf dowolnego grafu planarnego G wtedy i tylko wtedy gdy H jest 3-spójny.

Jeszcze jeden algorytm. W podrozdziale 3.6 pokażemy natomiast bardzo naturalne rozszerzenie algorytmu Chiby i Nishizeki'ego [11] do przypadku 5-cykli otrzy-

mując algorytm bardzo efektywny i prosty w implementacji. Po dokonaniu drobnych modyfikacji otrzymamy także algorytm zliczający 5-cykle w d-zdegenerowanym grafie o m krawędziach w czasie $\mathcal{O}(d^2 \cdot m)$. Oba nasze algorytmy wyszukujące 5-cykle mają taką samą złożoność co algorytm Alona i innych [4], są jednak znacznie prostsze w implementacji.

Żaden z przedstawionych w tym rozdziale algorytmów nie wymaga, aby wejściowy graf planarny był dany wraz z włożeniem w płaszczyznę (wejście stanowi lista krawędzi, podanych w dowolnej kolejności).

3.1.3 Zastosowania do kolorowania grafów

Okazuje się, że problemy wyszukiwania krótkich cykli w grafach planarnych są w nieoczekiwany sposób powiązane z klasycznym kolorowaniem wierzchołkowym oraz listowym kolorowaniem wierzchołkowym grafów planarnych. Przypomnijmy, że w problemie *kolorowania listowego* z każdym wierzchołkiem grafu związana jest lista (w rzeczwystości: zbiór) dozwolonych kolorów. Należy przypisać wierzchołkom dozwolone dla nich kolory tak, aby końce każdej krawędzi otrzymały inne kolory. Mówimy, że graf jest *k-wybieralny*, gdy zawsze można go pokolorować, pod warunkiem, że dla każdego wierzchołka określono co najmniej k dozwolonych kolorów. Problem rozstrzygnięcia, czy dany graf planarny jest 3-wybieralny jest NP-zupełny, podobnie jak rozstrzyganie zwykłej 3-kolorowalności. Z rozdziału 2 pamiętamy, że grafy nie zawierające cykli długości 3 są 3-kolorowalne. Okazuje się, że grafy nie zawierające cykli długości 3 i 4 są 3-wybieralne [51]. Co więcej, dla każdego k takiego, że $3 \leq k \leq 6$, dowolny graf bez k-cykli jest 4-wybieralny [41, 40, 25]. Dzięki temu algorytmy, które wyszukują krótkie cykle mogą zostać użyte do efektywnego rozpoznawania szerokich klas 3-kolorowalnych, 3- i 4-wybieralnych grafów planarnych.

3.1.4 Zarys pomysłu

Zaczynamy od przetworzenia informacji o wejściowym grafie G otrzymując dwa nowe grafy, GP_2 i GP_3. Każda krawędź w GP_2 i GP_3 odpowiada pewnej ścieżce długości, odpowiednio, 2 lub 3 w grafie G. Rozmiary grafów GP_2 i GP_3 są liniowe względem rozmiaru G. Oznacza to, że ich krawędzie reprezentują tylko niektóre ścieżki (graf planarny może zawierać nawet $\Omega(n^2)$ ścieżek długości 2). Pamiętamy, że podobnie było w przypadku grafów G_2 i G_3 wchodzących w skład grafu skrótów z rozdziału 1. Pewne analogie występują także w sposobie budowania grafów G_2, G_3 oraz GP_2, GP_3, nie są to jednak te same grafy. W szczególności, grafy GP_2 i GP_3 przechowują *pełną* informację o cyklach w grafie G. Dzięki temu, dowolny 4-cykl w G odpowiada pewnej parze krawędzi łączących te same wierzchołki w GP_2. Z kolei dowolny 6-cykl w G odpowiada pewnemu trójkątowi w GP_2 lub pewnej parze krawędzi łączących te same wierzchołki w GP_3. Wykorzystując tego typu fakty, w podrozdziale 3.4 pokażemy

stosunkowo proste algorytmy wyszukiwania cykli długości od 3 do 6.

3.2 Grafy ścieżek długości 2 i 3

Pseudokod 3.2.1 opisuje algorytm TRANSFORM, który buduje grafy GP_2 i GP_3 na podstawie wejściowego grafu planarnego G. Algorytm usuwa kolejno wierzchołki o niskich stopniach z grafu G, zapisując w grafach GP_2 i GP_3 informacje dotyczące ścieżek przechodzących przez usuwane wierzchołki. $N(v)$ oznacza zbiór sąsiadów wierzchołka v w aktualnej wersji grafu G (zauważmy, że ten graf zmienia się podczas działania algorytmu), natomiast $N_0(v)$ oznacza zbiór sąsiadów v w wejściowym grafie G. Wraz z każdą krawędzią dodawaną do grafów GP_2 i GP_3, przechowujemy odpowiadającą jej ścieżkę długości, odpowiednio, 2 lub 3. Krawędź $xy \in E(GP_2)$ związana ze ścieżką xvy w G będzie oznaczana przez $x[v]y$. Podobnie, $x[vw]y$ oznacza krawędź xy w GP_3, związaną ze ścieżką $xvwy$ w G. Zauważmy, że podczas wykonywania algorytmu do grafów GP_2 i GP_3 może zostać dodanych wiele krawędzi łączących te same wierzchołki (choć odpowiadających różnym ścieżkom). Co więcej, zauważmy, że gdy dodajemy krawędź xy do grafu GP_3 w linii 10, może się zdarzyć, że $x = y$. Mówiąc precyzyjnie, mamy tutaj do czynienia z multigrafami zawierającymi pętle.

Algorytm 3.2.1 Budowanie grafów GP_2 i GP_3 na podstawie wejściowego grafu G

```
 1: procedure TRANSFORM(G)
 2:     GP2 ← ∅
 3:     GP3 ← ∅
 4:     while G ≠ ∅ do
 5:         v ← wierzchołek o najniższym stopniu w G
 6:         for each x ∈ N(v) do
 7:             for each y ∈ N0(v) − N(v) do
 8:                 GP2 ← GP2 ∪ {x[v]y}
 9:             for each v[w]y ∈ GP2 do
10:                 GP3 ← GP3 ∪ {x[vw]y}
11:             Usuń krawędź vx z G
12:         Usuń wierzchołek v z G
```

Twierdzenie 3.2.1. *Dowolny cykl w grafie wejściowym składa się z krawędziowo rozłącznych ścieżek odpowiadających pewnym krawędziom grafów GP_2 i GP_3.*

Dowód. Niech G_0 oznacza graf wejściowy, natomiast niech G będzie jego aktualną wersją podczas wykonywania algorytmu. Pokażemy przez indukcję po liczbie usuniętych wierzchołków, że za każdym razem gdy w algorytmie zaczyna się lub kończy wykonywanie pętli **while**, każdy cykl w G_0 składa się z pewnej liczby krawędzi grafu G oraz z pewnej liczby krawędziowo rozłącznych ścieżek odpowiadających pewnym

krawędziom z GP_2 lub GP_3. Taki niezmiennik natychmiast implikuje prawdziwość twierdzenia, ponieważ w chwili zakończenia algorytmu $E(G) = \emptyset$.

Zanim jakikolwiek wierzchołek grafu G_0 został usunięty, nasz niezmiennik jest spełniony w sposób trywialny, ponieważ wtedy $G_0 = G \cup GP_2 \cup GP_3$. Załóżmy teraz, że nasz niezmiennik jest spełniony na początku pętli **while** (linia 5) i pokażemy, że pozostaje on spełniony na końcu pętli (linia 12). Niech $C_0 \subseteq G_0$ będzie dowolnym cyklem. Z założenia indukcyjnego, istnieje cykl $C \subseteq G \cup GP_2 \cup GP_3$ odpowiadający cyklowi C_0 (jako suma krawędziowo rozłącznych ścieżek). Możemy założyć, że wierzchołek v wybrany w linii 5 jest incydentny z pewną krawędzią $va \in E(G) \cap E(C)$, gdyż w przeciwnym przypadku cykl C pozostaje niezmieniony podczas rozpatrywanego obrotu pętli. Niech $b \in V(C)$ będzie sąsiadem v w cyklu C, różnym od a. Rozważymy trzy przypadki w zależności od tego, z którego z grafów G, GP_2, GP_3 pochodzi krawędź vb.

Gdy $vb \in E(G)$, w linii 8 dodajemy krawędź $a[v]b$ do GP_2. Jest jasne, że wtedy cykl $C' = C - \{va, vb\} \cup \{a[v]b\}$ odpowiada cyklowi C_0. Podobnie, gdy $vb \in E(GP_2)$, krawędź ab zostaje dodana do GP_3 w linii 10. Ponownie widzimy, że cykl $C'' = C - \{va, vb\} \cup \{ab\}$ odpowiada C_0. Wreszcie, gdy $v[cd]b \in E(GP_3)$, to w linii 8 dodajemy krawędź $a[v]c$ do GP_2. Co więcej, niezależnie od tego, w jakiej kolejności wierzchołki c i d były usuwane z grafu, GP_2 zawiera obie krawędzie $v[c]d$ i $c[d]b$. Dzięki temu, przed zakończeniem wykonywania pętli **while** cykl $C''' = C - \{va, v[cd]b\} \cup \{a[v]c, c[d]b\} \subseteq G \cup GP_2 \cup GP_3$ odpowiada C_0. To kończy dowód. ■

Fakt 3.2.2. *Niech G będzie dowolnym d-zdegenerowanym grafem i niech GP_2, GP_3 będą grafami utworzonymi przez algorytm* Transform. *Wtedy graf GP_2 zawiera nie więcej niż $2d \cdot |E(G)|$ krawędzi, natomiast graf GP_3 zawiera nie więcej niż $2d \cdot |E(GP_2)|$ krawędzi.*

Dowód. Ponownie oznaczmy przez G_0 graf wejściowy. Rozważmy dowolny wierzchołek v wybrany w linii 5 algorytmu Tansform. Ponieważ graf jest d-zdegenerowany, więc w chwili wyboru tego wierzchołka $\deg_G(v) \leq d$. Liczba wszystkich krawędzi postaci $x[v]y$, dodanych do GP_2 w linii 8 nie przekracza więc $\deg_G(v) \cdot \deg_{G_0}(v) \leq d \deg_{G_0}(v)$. Sumując po wszystkich wierzchołkach grafu wejściowego G_0, otrzymujemy:

$$|E(GP_2)| \leq \sum_{v \in V(G_0)} d \deg_{G_0}(v) = 2d \cdot |E(G_0)|.$$

Analogiczne rozumowanie daje oszacowanie na $|E(GP_3)|$. ■

Wniosek 3.2.3. *Niech G będzie dowolnym d-zdegenerowanym grafem i niech GP_2, GP_3 będą grafami utworzonymi przez algorytm* Transform. *Wtedy $|E(GP_2)| \leq 2d \cdot |E(G)|$ oraz $|E(GP_3)| \leq 4d^2 \cdot |E(G)|$.* ■

Wniosek 3.2.4. *Algorytm* Transform *można zaimplementować tak, aby działał w czasie $\mathcal{O}(d^2|E(G)|)$, dla dowolnego d-zdegenerowanego grafu G.* ■

Wniosek 3.2.5. *Jeśli G jest planarny, to $|E(GP_2)| \leq 10|E(G)|$ oraz $|E(GP_3)| \leq 100 \cdot |E(G)|$.* ■

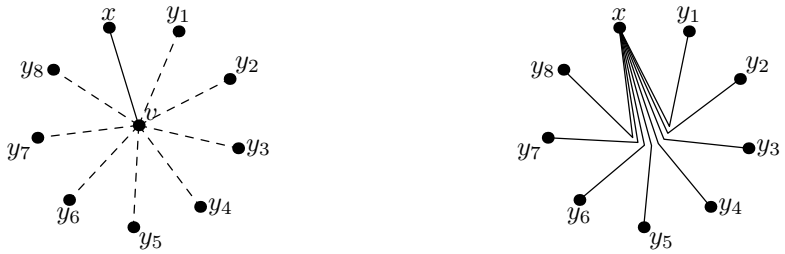
Lemat 3.2.6. *Dla dowolnego grafu planarnego G, graf GP_2 utworzony przez algorytm* TRANSFORM *jest sumą nie więcej niż 20 grafów planarnych.*

Dowód. Niech $G = (V, E)$. Wiemy z twierdzenia o czterech barwach, że G jest 4-kolorowalny. Niech $K : V \to \{1, 2, 3, 4\}$ będzie dowolnym 4-kolorowaniem grafu G. Rozważmy dowolny wierzchołek v grafu G. Podczas działania algorytmu TRANSFORM w pewnym momencie wierzchołek v zostaje wybrany w linii 5. Ponieważ grafy planarne są 5-zdegenerowane, miał on wtedy nie więcej niż pięciu sąsiadów. Każdemu z nich przypiszemy inny numer ze zbioru $\{1, \ldots, 5\}$. Dla każdego takiego sąsiada x wierzchołka v niech $no_v(x) \in \{1, \ldots, 5\}$ oznacza przypisany mu numer. Podobną numerację sąsiadów określamy dla wszystkich wierzchołków grafu. Pozwala to zdefiniować podział grafu GP_2 na 20 grafów:

$$GP_2 = \bigcup_{\substack{i \in \{1,\ldots,4\} \\ j \in \{1,\ldots,5\}}} GP_2^{i,j}$$

taki, że graf $GP_2^{i,j}$ zawiera krawędź $x[v]y$, dodaną do GP_2 w linii 8, wtedy i tylko wtedy gdy $K(v) = i$ oraz $no_v(x) = j$.

Teraz wystarczy jedynie pokazać, że każdy z grafów $GP_2^{i,j}$ jest planarny. Weźmy dowolne włożenie płaskie G i rozważmy dowolny z grafów $GP_2^{i,j}$. Pokażemy, że ten graf także ma włożenie płaskie. Wierzchołki $GP_2^{i,j}$ odwzorowujemy w płaszczyznę dokładnie tak samo jak wierzchołki G. Natomiast każdą krawędź $x[v]y \in E(GP_2^{i,j})$ odpowiadającą ścieżce P w grafie G rysujemy *blisko* włożenia ścieżki P w taki sposób, że dla dowolnego $v \in V$ i jego dowolnego ponumerowanego sąsiada x, żadne dwie spośród krawędzi odpowiadających ścieżkom postaci xvy się nie przecinają (patrz rysunek 3.1).



graf G graf $GP_2^{K(v),1}$

Rysunek 3.1: Rysowanie krawędzi GP_2 incydentnych z wierzchołkiem x.

Rozważmy teraz dowolną krawędź $e = x[v]y \in E(GP_2^{i,j})$ i załóżmy, że rysunek e przecina rysunek pewnej innej krawędzi $e' = x'[v']y' \in E(GP_2^{i,j})$. Gdyby $v = v'$

to obie krawędzie miałyby wspólny koniec – byłby to sąsiad v o numerze j. Wtedy jednak nie mogłyby się przecinać, gdyż zostały narysowane właśnie tak, aby się nie przecinały (zgodnie z rysunkiem 3.1). W takim razie $v \neq v'$. Widzimy, że wystarczy rozważyć sytuację, gdy $v = x'$ (pozostałe są symetryczne). Wtedy v i v' są sąsiednie w G. Taka sytuacja nie może jednak powstać, gdyż v i v' miałyby różne kolory i krawędzie e i e' nie mogłyby znaleźć się w tym samym grafie $GP_2^{i,j}$. ■

Zamiana multigrafów na grafy proste. W algorytmach dla cykli długości 4, 5 i 6 wygodnie będzie zamienić multigrafy GP_2, GP_3 na grafy proste. W tym celu zamieniamy krawędzie wielokrotne na pojedyncze. Otrzymane grafy proste oznaczmy odpowiednio przez $\widehat{GP}_2$ i $\widehat{GP}_3$. Dokładniej, dla dowolnych różnych wierzchołków x i y w grafie $\widehat{GP}_2$ istnieje krawędź xy wtedy i tylko wtedy gdy w GP_2 istnieje choć jedna krawędź łącząca x i y (analogicznie definiujemy $\widehat{GP}_3$). Przy tej operacji nie chcemy jednak tracić informacji zapisanej w oryginalnych multigrafach. Dlatego, wraz z każdą krawędzią xy grafu $\widehat{GP}_2$ przechowujemy ścieżki długości 2 przechowywane wraz ze *wszystkimi* krawędziami łączącymi x i y w GP_2 (analogicznie w $\widehat{GP}_3$).

Pozostaje jeszcze opisać, jak wykonać taką zamianę w czasie liniowym. Opiszemy algorytm dla grafu GP_2, dla GP_3 postępujemy podobnie. Załóżmy, że wierchołki grafu GP_2 tworzą liniowy porządek. Każdą krawędź $x[v]y \in E(GP_2)$, dla $x < y$, zamieniamy na trójkę (x, y, v). Otrzymujemy w ten sposób ciąg $|E(GP_2)|$ trójek. Następnie wystarczy posortować trójki w porządku leksykograficznym. W tym celu stosujemy klasyczny algorytm sortowania pozycyjnego (ang. *radix-sort*), opisany np. w podręczniku Cormena i innych [16]. Algorytm działa w czasie $\mathcal{O}(|V(GP_2)| + |E(GP_2)|) = \mathcal{O}(n + d \cdot m)$. Z posortowanego ciągu łatwo budujemy graf $\widehat{GP}_2$ w czasie linowym. Analogicznie dla GP_3 otrzymujemy czas $\mathcal{O}(|V(GP_3)| + |E(GP_3)|) = \mathcal{O}(n + d^2 \cdot m)$. Oczywiście tworząc odpowiedni ciąg czwórek dla GP_3, ignorujemy pętle.

Wniosek 3.2.7. *Dla dowolnego grafu planarnego G, graf $\widehat{GP}_2$ ma lesistość 60 i jest 120-zdegenerowany.*

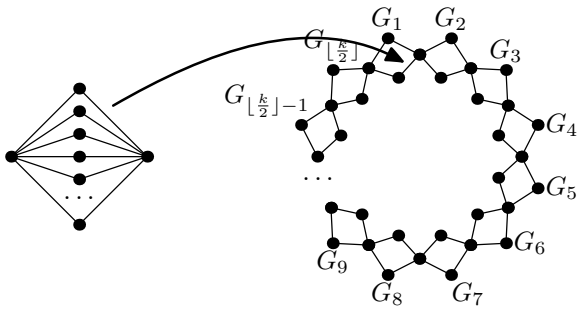
Dowód. Dowód otrzymujemy natychmiast stosując kolejno lemat 3.2.6, wniosek 1.3.4 i lemat 1.3.6. ■

3.3 Maksymalna liczba cykli danej długości

W tym podrozdziale pokażemy w jaki sposób wykorzystać własności grafów GP_2 i GP_3 aby udowodnić następujące twierdzenie.

Twierdzenie 3.3.1. *Niech $k \geq 3$ będzie ustaloną stałą, niezależną od n. Wtedy maksymalna liczba k-cykli w n-wierzchołkowym grafie planarnym wynosi $\Theta(n^{\lfloor k/2 \rfloor})$.*

Dowód. Zaczniemy od dolnego ograniczenia. Skonstruujemy n-wierzchołkowy graf planarny zawierający $\Omega(n^{\lfloor k/2 \rfloor})$ cykli długości k. Załóżmy na początek, że k jest parzyste. Zaczynamy od cyklu C długości $\frac{k}{2}$. Następnie zastępujemy każdą krawędź uv cyklu C przez l ścieżek długości 2 łączących u i v (patrz rysunek 3.2). To kończy opis konstruowanego grafu.



Rysunek 3.2: Konstrukcja n-wierzchołkowego grafu planarnego zawierającego $\Theta(n^{\lfloor k/2 \rfloor})$ cykli długości k.

Zauważmy, że $n = (l+1) \cdot \frac{k}{2}$ a więc $l = \Omega(n)$. Jest jasne, że otrzymany graf ma $\Omega(l^{\frac{k}{2}}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cykli długości k, tak jak chcieliśmy. Gdy k jest nieparzyste konstrukcja przebiega podobnie. Zaczynamy od cyklu C długości $\lfloor \frac{k}{2} \rfloor + 1$ i wybieramy dowolną krawędź $e \in C$. Następnie każda krawędź cyklu, z wyjątkiem e, jest zastępowana przez l ścieżek długości 2. Dostajemy $\Omega(l^{\lfloor \frac{k}{2} \rfloor}) = \Omega(n^{\lfloor \frac{k}{2} \rfloor})$ cykli długości k.

Górne ograniczenie dostajemy jako w wniosek z twierdzenia 3.2.1 i faktu 3.2.2. Mianowicie, niech G będzie dowolnym grafem planarnym. Wystarczy zauważyć, że dowolny k-cykl w G może składać się z co najwyżej $\lfloor \frac{k}{2} \rfloor$ rozłącznych krawędziowo ścieżek długości 2 i 3 przechowywanych w grafach GP_2 i GP_3. Ponieważ jest $O(n)$ takich ścieżek (Fakt 3.2.2) i każdy cykl w G może zostać podzielony na takie ścieżki, całkowita liczba cykli długości k w G nie przekracza $O(n^{\lfloor \frac{k}{2} \rfloor})$. ■

3.4 Wyszukiwanie cykli o długościach od 3 do 6

W tym podrozdziale pokażemy, w jaki sposób można używać grafów GP_2 i GP_3 do wyszukiwania krótkich cykli. Algorytmy dla cykli długości 3, 4 i 5 stosują się do d-zdegenerowanych grafów i mają liniową złożoność czasową gdy $d = \mathcal{O}(1)$. Tej własności nie ma algorytm dla cykli długości 6: przy analizie jego złożoności wykorzystujemy lemat 3.2.6. Dlatego ten algorytm działa w czasie liniowym jedynie dla grafów planarnych. Przy opisywaniu kolejnych algorytmów będziemy zakładać, że na wejściu pojawia się d-zdegenerowany (lub planarny) graf G zawierający n wierzchołków i m krawędzi.

3.4.1 Cykle długości 3

Jako natychmiastowy wniosek z twierdzenia 3.2.1 otrzymujemy, że aby otrzymać 3-cykle w G wystarczy wyszukać pętle w grafie GP_3. Cały algorytm działa więc w czasie liniowym ze względu na rozmiar GP_3, a więc $\mathcal{O}(d^2 \cdot m)$.

3.4.2 Cykle długości 4

Wiemy z twierdzenia 3.2.1, że każdy cykl $xvywx$ długości 4 w G składa się z dwóch ścieżek, powiedzmy xvy i ywx, z których każda jest przechowywana wraz z pewną krawędzią łączącą x i y w GP_2. Dzięki temu, aby znaleźć 4-cykl w G wystarczy odszukać krawędź w $\widehat{GP_2}$ przechowującą dwie ścieżki. Zauważmy, że graf GP_3 jest tu zbędny i możemy z niego zrezygnować. Algorytm znajdujący pojedynczy cykl działa więc w czasie $\mathcal{O}(d \cdot m)$.

Podobnie, jeśli chcemy znaleźć więcej cykli (lub wszystkie cykle), dla każdej krawędzi grafu $\widehat{GP_2}$ rozważamy wszystkie pary przechowywanych wraz z nią ścieżek. Algorytm działa wtedy w czasie $\mathcal{O}(d \cdot m + \#_c)$, gdzie $\#_c$ jest liczbą wypisanych cykli, zatem z twierdzenia 3.3.1 wiemy, że $\#_c = \mathcal{O}(n^2)$.

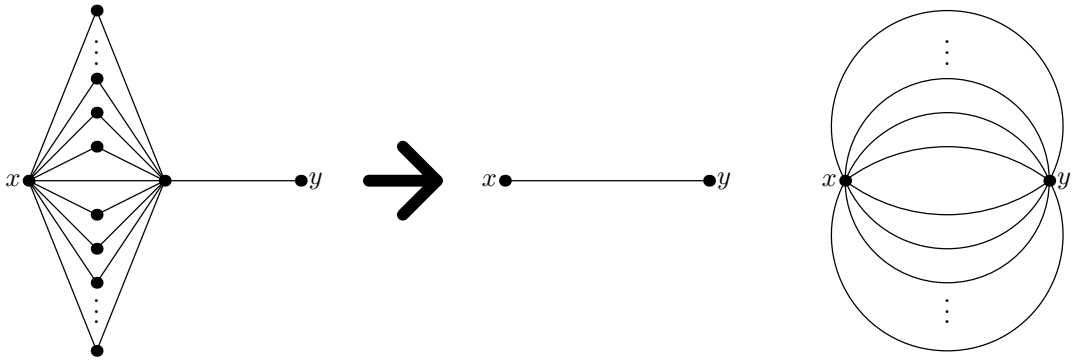
3.4.3 Cykle długości 5

Twierdzenie 3.2.1 implikuje, że w grafie G każdy cykl długości 5 składa się z dwóch ścieżek długości 2 i 3 przechowywanych wraz z pewnymi krawędziami grafów GP_2 i GP_3. Zauważmy jednak, że wynikanie odwrotne nie jest tu prawdziwe, tzn. dwie ścieżki długości, odpowiednio 2 i 3, przechowywane wraz z krawędziami $e_1 = xy \in E(GP_2)$ oraz $e_2 = xy \in E(GP_3)$ nie zawsze tworzą cykl. Dla jednej krawędzi $x[v]y \in E(GP_2)$ może być nawet $\Theta(n)$ krawędzi postaci $x[wv]y \in E(GP_3)$ (patrz rysunek 3.3). Widzimy, że odpowiednie pary ścieżek w G przecinają się w wierzchołku v i nie tworzą cykli (tworzą marszruty zamknięte).

Na szczęście możemy spojrzeć na sytuację z drugiej strony, dostrzegając następujący fakt:

Fakt 3.4.1. *Dowolna ścieżka długości 3 łącząca x i y w G może przecinać się wewnętrznie z nie więcej niż dwoma ścieżkami długości 2 łączącymi x i y.*

Aby znaleźć pojedynczy cykl długości 5 w G, dla każdej krawędzi $xy \in E(\widehat{GP_3})$ i dla każdej ścieżki długości 3, przechowywanej wraz z xy, sprawdzamy co najwyżej 3 ścieżki przechowywane wraz z krawędzią xy w $\widehat{GP_2}$. Wcześniej, dla każdej krawędzi xy grafu $\widehat{GP_3}$, obliczamy wskaźnik do krawędzi xy w grafie $\widehat{GP_2}$ (jeśli taka istnieje). Takie wskaźniki łatwo znaleźć w czasie liniowym ponownie wykorzystując sortowanie pozycyjne. Cały algorytm ma więc złożoność czasową liniową względem sumy rozmiarów GP_2 i GP_3, a więc $\mathcal{O}(d^2 \cdot m)$.



Rysunek 3.3: Jedna ścieżka długości 2 łącząca x i y może przecinać się z $\Theta(n)$ ścieżek długości 3 łączących x i y.

Analogicznie postępujemy chcąc znaleźć więcej cykli w G. Wtedy, dla każdej ścieżki długości 3 łączącej x i y, przechowywanej w $\widehat{GP_3}$, sprawdzamy kolejne ścieżki łączące x i y w $\widehat{GP_2}$. Dla każdej z rozważanych ścieżek długości 3 znajdą się co najwyżej dwie ścieżki długości 2, które nie utworzą cyklu. W ten sposób otrzymujemy algorytm wyszukujący wszystkie $\#_c$ 5-cykli w grafie w czasie $\mathcal{O}(d^2 \cdot m + \#_c)$. Może się zdarzyć, że algorytm zwróci jeden cykl kilka razy. Aby pozbyć się niepotrzebnych kopii nie zmieniając złożoności czasowej algorytmu, wystarczy posortować znalezione cykle za pomocą sortowania pozycyjnego. Każdy cykl C zamieniamy na ciąg $(v_1, v_2, v_3, v_4, v_5)$ taki, że $v_1, v_2, \ldots, v_5$ są kolejnymi wierzchołkami C, $v_1 = \min V(C)$ oraz $v_2 = \min N_C(v_1)$.

3.4.4 Cykle długości 6

W tym punkcie zakładamy, że G jest n-wierzchołkowym grafem planarnym. Na podstawie twierdzenia 3.2.1 wiemy, że każdy 6-cykl w G składa się z dwóch 3-ścieżek łączących te same wierzchołki i przechowywanych w GP_3 lub z trzech ścieżek długości 2 przechowywanych w GP_2 i takich, że odpowiadające im krawędzie grafu GP_2 tworzą trójkąt. Ponownie będziemy rozważać dwie wersje algorytmu. Pierwsza w czasie liniowym znajduje jeden cykl w grafie, lub stwierdza, że takiego cyklu nie ma. Druga wersja znajduje wszystkie $\#_c$ cykli w czasie $\mathcal{O}(n + \#_c)$, gdzie $\#_c = \mathcal{O}(n^3)$. W drugiej wersji ponownie może się zdarzyć, że ten sam cykl zostanie znaleziony dwukrotnie: raz jako suma dwóch 3-ścieżek, drugi raz jako suma trzech 2-ścieżek. Niepotrzebnych kopii pozbywamy się analogicznie jak w przypadku 5-cykli.

Wyszukiwanie trójkątów w $\widehat{GP_2}$

Zauważmy, że ponieważ graf $\widehat{GP_2}$ jest $\mathcal{O}(1)$-zdegenerowany (patrz wniosek 3.2.7), możemy użyć algorytmu z punktu 3.4.1, aby w czasie liniowym znaleźć wszystkie trójkąty w $\widehat{GP_2}$ (ich liczba nie przekracza $\mathcal{O}(n)$). Możemy także użyć innych algorytmów wyszukujących trójkąty w grafach rzadkich w czasie liniowym, np. algorytmu Chiby i Nishizeki'ego [11] lub algorytmu Chrobaka i Eppsteina [14]. Powiemy, że dwie ścieżki są *wewnętrznie rozłączne*, gdy ich wspólne wierzchołki są równocześnie końcami każdej z nich. Zauważmy, że trójka ścieżek długości 2, odpowiadająca pewnemu trójkątowi w $\widehat{GP_2}$ tworzy 6-cykl w G wtedy i tylko wtedy, gdy te ścieżki są parami wewnętrznie rozłączne, co nie zawsze musi być spełnione. Pokażemy, jak dla każdego z $\mathcal{O}(n)$ znalezionych trójkątów znaleźć w sposób efektywny te trójki ścieżek długości 2, które tworzą cykle długości 6.

Rozważmy dowolny trójkąt T znaleziony w $\widehat{GP_2}$. Niech v_1, v_2, v_3 będą jego wierzchołkami. Zauważmy, że możemy ignorować 2-ścieżki $v_1v_2v_3$, $v_1v_3v_2$ oraz $v_2v_1v_3$, gdyż żadna z nich nie może być częścią cyklu długości 6 odpowiadającego trójkątowi T. Wszystkie inne ścieżki długości 2 przechowywane wraz z krawędziami T będziemy nazywać *potrzebnymi*. Dla dowolnej krawędzi e trójkąta T niech $Need(e)$ oznacza zbiór potrzebnych krawędzi przechowywanych wraz z e. Aby znaleźć 6-cykle odpowiadające trójkątowi T użyjemy algorytmu 3.4.1 przedstawionego poniżej.

Algorytm 3.4.1 Wyszukiwanie trójek 2-ścieżek, które tworzą cykle długości 6

```
1: Posortuj krawędzie trójkąta T tak, aby |Need(e1)| ≤ |Need(e2)| ≤ |Need(e3)|.
2: for each p ∈ Need(e1) do
3:     for each q ∈ Need(e2) do
4:         if p i q są wewnętrznie rozłączne then
5:             for each r ∈ Need(e3) do
6:                 if p, q i r tworzą 6-cykl w G then
7:                     Wypisz znaleziony cykl.
```

Fakt 3.4.2. *Algorytm 3.4.1 zwraca pierwszy cykl po czasie stałym. Co więcej, po wypisaniu dowolnego cyklu, po czasie stałym algorytm znajduje następny cykl lub kończy działanie.*

Dowód. Niech $T = v_1v_2v_3v_1$ będzie trójkątem w $\widehat{GP_2}$. Rozważmy dowolne ścieżki $v_1xv_2 \in Need(v_1v_2)$ i $v_2yv_3 \in Need(v_2v_3)$. Załóżmy, że te ścieżki nie są wewnętrznie rozłączne. Ponieważ są potrzebne, wnioskujemy, że $x = y$. W takim razie możemy zauważyć, że mogą być co najwyżej dwie pary takich ścieżek, gdyż każde 3 takie pary tworzą $K_{3,3}$, co, zgodnie z twierdzeniem Kuratowskiego, przeczyłoby planarności G.

Wiemy już, że mogą być co najwyżej 2 pary ścieżek $p \in Need(e_1)$, $q \in Need(e_2)$, które nie są wewnętrznie rozłączne. Rozważmy dowolną parę wewnętrznie rozłącznych ścieżek $p = abc$ i $q = cde$ sprawdzaną przez algorytm. Widzimy,

że algorytm może znaleźć co najwyżej dwie ścieżki $r \in Need(e_3)$, które nie tworzą 6-cyklu z p i q – są to ścieżki *eda* i *eba*. To implikuje, że wśrod dowolnych trzech kolejnych sprawdzanych trójek co najmniej jedna tworzy cykl. To kończy dowód. ■

Wniosek 3.4.3. *Wszystkie* 6*-cykle grafu* G *odpowiadające trójkątom w* $\widehat{GP_2}$ *mogą zostać znalezione w czasie* $\mathcal{O}(n + \#_c)$*, gdzie* $\#_c$ *oznacza liczbę wszystkich takich cykli w* G*. Pierwszy cykl jest znajdowany po czasie* $\mathcal{O}(n)$*.*

Sprawdzanie par ścieżek związanych z tą samą krawędzią w $\widehat{GP_3}$.

Pośród wszystkich k ścieżek przechowywanych wraz z jedną krawędzią grafu $\widehat{GP_3}$ chcemy znaleźć wszystkie pary ścieżek wewnętrznie rozłącznych. Gdy $k < 11$ liczba wszystkich par do sprawdzenia jest ograniczona przez stałą i możemy wybrać interesujące nas pary w czasie $\mathcal{O}(1)$. W przeciwnym przypadku skorzystamy z następującego faktu:

Fakt 3.4.4. *Niech* e *będzie krawędzią grafu* $\widehat{GP_3}$ *przechowującą* $k \geq 11$ *różnych ścieżek długości 3 w* G*. Istnieje algorytm działający w czasie* $\mathcal{O}(k)$*, który albo stwierdza, że żadna z par ścieżek nie tworzy* 6*-cyklu w* G*, albo zwraca* $\Theta(k)$ *cykli długości* 6 *i co najmniej* 2 *ścieżki takie, że żadna z nich nie jest już częścią żadnego cyklu w* G*.*

Dowód. Niech p i q będą dowolnymi, różnymi ścieżkami przechowywanymi wraz z $e = xy$. Widzimy, że może być nie więcej niż 8 ścieżek długości 3 łączących x i y, które nie są wewnętrznie rozłączne ani z p ani z q. Na początek sprawdzamy, czy p i q tworzą cykl i jeśli tak, wypisujemy go. Następnie, dla każdej 3-ścieżki $r \notin \{p, q\}$ przechowywanej wraz z e sprawdzamy, czy tworzy ona cykl z p lub q – znalezione cykle wypisujemy. W ten sposób znajdziemy co najmniej $k - 10$ cykli, sprawdzając $2k$ par krawędzi. ■

Wniosek 3.4.5. *Wszystkie* 6*-cykle grafu* G *odpowiadające parom krawędzi łączącym te same wierzchołki w* GP_3 *mogą zostać znalezione w czasie* $\mathcal{O}(n + \#_c)$*, gdzie* $\#_c$ *oznacza liczbę wszystkich takich cykli w* G*. Pierwszy cykl jest znajdowany po czasie* $\mathcal{O}(n)$*.*

3.5 Dłuższe cykle

Z twierdzenia 3.2.1 widzimy, że każdy cykl długości 7 składa się z pewnych dwóch 2-ścieżek przechowywanych w GP_2 i jednej 3-ścieżki przechowywanej w GP_3. Algorytm wyszukiwania 7-cykli redukuje się więc do wyszukiwania 3-cykli w $\widehat{GP_2} \cup \widehat{GP_3}$. Niestety nie jest jasne, czy $\widehat{GP_3}$ jest grafem rzadkim. Jeśli tak jest, otrzymujemy liniowy algorytm wyszukiwania 7-cykli w grafie planarnym, analogiczny do algorytmu wyszukiwania 6-cykli pochodzących od trójkątów w GP_2.

3.6 Inny algorytm wyszukujący cykle długości 5

W tym podrozdziale opiszemy jeszcze jeden algorytm wyszukiwania cykli długości 5 w grafach rzadkich, który dla grafów planarnych działa w czasie liniowym. Jest on prostszy w implementacji od algorytmu opisanego w punkcie 3.4.3. Wydaje się również, że w praktyce jest on szybszy (patrz podrozdział 3.7). Kolejna ważna cecha nowego algorytmu polega na tym, że każdy cykl jest zwracany dokładnie raz (dodatkowe sortowanie pozycyjne nie jest tu potrzebne). Pokażemy również, w jaki sposób zmodyfikować algorytm tak, aby efektywnie zliczał liczbę cykli w grafie rzadkim (dla grafów planarnych w czasie liniowym ze względu na rozmiar grafu). Nasz algorytm wykorzystuje podejście Chiby i Nishizeki'ego [11], którzy pokazali jak znajdować cykle długości 3 i 4. Algorytm dla 5-cykli otrzymamy przez wzbogacenie ich metody o technikę $\mathcal{O}(1)$-orientowania grafu, nawiasem mówiąc również używaną do znajdowania cykli w grafach planarnych[2]. Pseudokod algorytmu przedstawiamy poniżej.

Algorytm 3.6.1 Znajdowanie wszystkich cykli długości 5 w grafie d-zdegenerowanym

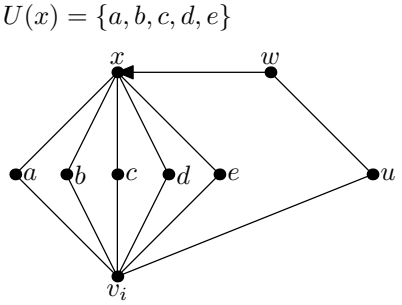
```
1:  Zorientuj krawędzie G tak, aby otrzymać d-orientację G→.
2:  Posortuj wierzchołki G tak, aby deg(v_1) ≥ deg(v_2) ≥ ... ≥ deg(v_n).
3:  for each v ∈ V do
4:      U(v) ← ∅
5:  for i ← 1 to n do                          ▷ Znajdowanie cykli zawierających v_i
6:      for each u ∈ N(v_i) do
7:          for each w' ∈ N(u) − {v_i} do
8:              U(w') ← U(w') ∪ {u}
9:      for each u ∈ N(v_i) do
10:         for each w ∈ N(u) − {v_i} do
11:             for each (w, x) ∈ E(G→) do
12:                 if x ≠ u then
13:                     for each y ∈ U(x) do
14:                         if y ∉ {u, w} then
15:                             Wypisz cykl v_i u w x y v_i.
16:     for each w such that U(w) ≠ ∅ do
17:         U(w) ← ∅
18:     Usuń v_i z G (wraz ze wszystkimi incydentnymi krawędziami).
```

Fakt 3.6.1. *Algorytm 3.6.1 poprawnie znajduje wszystkie cykle w grafie.*

Dowód. Pokażemy, że nasz algorytm w i-tym obrocie pętli znajduje wszystkie cykle zawierające wierzchołek v_i. Przyjrzyjmy się i-temu obrotowi pętli. W liniach 6–8 dla każdego wierzchołka w' połączonego z v_i ścieżką długości 2 obliczany jest zbiór

[2]Algorytm wyszukiwania trójkątów sformułowany przez Chrobaka i Eppsteina [14] korzysta z $\mathcal{O}(1)$-orientacji.

$U(w')$. Jest to zbiór wszystkich wierzchołków incydentnych zarówno z v_i, jak i z w'. Innymi słowy, korzystając z $U(w')$ można wygenerować wszystkie ścieżki długości 2 łączące w' z v_i. Weźmy teraz dowolny 5-cykl zawierający v_i i nazwijmy jego kolejne wierzchołki: v_i, u, w, x, a. Pokażemy, że zostanie on znaleziony przez algorytm. Do rozważenia są dwa symetryczne przypadki, w zależności od kierunku krawędzi wx w grafie $\overrightarrow{G}$. Załóżmy, że $(w, x) \in E(\overrightarrow{G})$, tak jak na rysunku 3.4. Pętla zaczynająca się



Rysunek 3.4: Działanie algorytmu 3.6.1.

w linii 9 wybiera kolejno wszystkich sąsiadów v_i, a więc w końcu wybierze u. Z kolei pętla zaczynająca się w linii 10 wybierze sąsiadów u różnych od v_i, a więc także w. Pętla z linii 11 znajdzie wierzchołek x. Ponieważ a jest równocześnie sąsiadem v_i i x, więc $a \in U(x)$ i wierzchołek a zostanie wybrany w pętli z linii 13 i nasz cykl zostanie wypisany. Pokazaliśmy w ten sposób, że wszystkie 5-cykle w grafie zostaną znalezione. Widzimy również, że każda piątka wierzchołków wypisanych przez algorytm tworzy pewien cykl (m.in. dzięki sprawdzaniu odpowiednich warunków w liniach 12 i 14), a więc algorytm wypisuje dokładnie wszystkie 5-cykle w grafie (i nic więcej). ■

Lemat 3.6.2 (Chiba, Nishizeki [11]). *Niech G będzie dowolnym grafem o m krawędziach i lesistości a. Wtedy*

$$\sum_{uv \in E} \min\{d(u), d(v)\} \leq 2am.$$

■

Fakt 3.6.3. *Dla dowolnego d-zdegenerowanego grafu G o m krawędziach, algorytm 3.6.1 działa w czasie $\mathcal{O}(d^2 \cdot m + \#_c)$, gdzie $\#_c$ oznacza liczbę cykli długości 5 w G. Co więcej, algorytm znajduje pierwszy cykl (lub kończy działanie, gdy G nie zawiera 5-cykli) w czasie $\mathcal{O}(d^2 \cdot m)$.*

Dowód. Linię 1 algorytmu możemy łatwo zaimplementować w czasie liniowym ze względu ma rozmiar grafu (patrz lemat 1.3.7). W linii 2 pojawia się sortowanie n liczb. Ponieważ są to liczby całkowite ze zbioru $0, \ldots, n-1$, możemy zastosować algorytm sortowania przez zliczanie (zobacz np. [16]), który zajmie czas $\mathcal{O}(n)$.

Pozostaje wykazać, że pętla **for** o początku w linii 5 wykona się w czasie liniowym. Rozważmy teraz wierzchołek v_i i jego sąsiada u, wybranego przez w pętli **for** w linii 9. Niech G_0 będzie grafem wejściowym a G aktualną wersją G_0 (po usunięciu wierzchołków $v_1, \ldots, v_{i-1}$). Wtedy $\deg_G(u) \leq \deg_{G_0}(u) = \min\{\deg_{G_0}(u), \deg_{G_0}(v_i)\}$. Zauważmy także, że na podstawie wniosku 1.3.8 graf G ma lesistość co najwyżej d. W takim razie lemat 3.6.2 gwarantuje, że pętla o początku linii 10 wykona nie więcej niż $2dm$ obrotów podczas działania całego algorytmu. To z kolei implikuje, że pętla o początku linii 11 wykona nie więcej niż $2d^2 \cdot m$ obrotów. Co więcej, wśród wszystkich sprawdzanych piątek wierzchołków (v_i, u, w, x, y), co najwyżej $4d^2m$ piątek nie tworzy cyklu. To kończy dowód. ■

Algorytm 3.6.1 może zostać łatwo zmodyfikowany w celu otrzymania algorytmu zliczającego w czasie $\mathcal{O}(d^2 \cdot m)$ wszystkie 5-cykle w d-zdegenerowanym grafie o m krawędziach – daje to czas liniowy dla grafów planarnych. W tym przypadku $U(w)$ będzie oznaczało liczbę ścieżek długości 2 od v_i do w. W algorytmie 3.6.1, zawsze gdy znajdziemy 3-ścieżkę $p = v_iuwx$, musimy znaleźć te ścieżki długości 2 zapisane w $U(x)$, które są wewnętrznie rozłączne z p (linia 14). Możemy łatwo pozbyć się tego czasochłonnego fragmentu. Przed sprawdzeniem wszystkich 3-ścieżek postaci $v_iu\ldots$ zmniejszamy wartość $U(w)$ dla każdego sąsiada w wierzchołka u. Dzięki temu, kiedy znajdziemy 3-ścieżkę $p = v_iuwx$ mamy pewność, że istnieje dokładnie $U(x)$ ścieżek długości 2 łączących v_i i x, z których żadna nie przechodzi przez u. Jeśli teraz w jest incydentny z v_i, to dokładnie jedna z tych ścieżek przecina p (w wierzchołku w). W przeciwnym przypadku wszystkie z tych $U(x)$ ścieżek tworzą cykl z p. Pseudokod algorytmu 3.6.2 zawiera wszystkie szczegóły.

Fakt 3.6.4. *Algorytm 3.6.2 poprawnie zlicza w czasie $\mathcal{O}(d^2 \cdot m)$ wszystkie 5-cykle w danym d-zdegenerowanym grafie o m krawędziach.*

Dowód. Analiza złożoności algorytmu 3.6.2 przebiega analogicznie jak w przypadku algorytmu 3.6.1. Poprawność wynika z przeprowadzonych powyżej rozważań. ■

3.7 Eksperymenty

Ponieważ motywacją badań przedstawionych w niniejszym rozdziale było opracowanie algorytmów, które mogą być używane w praktyce, w tym podrozdziale prezentujemy wyniki eksperymentów dotyczących przedstawionych algorytmów. Algorytmy były testowane na losowej triangulacji zawierającej 100 000 wierzchołków, wygenerowanej za pomocą programu *Stanford GraphBase* autorstwa Donalda Knutha. Obliczenia zostały wykonane na komputerze wyposażonym w procesor Pentium II o częstotliwości 500 MHz i 2 GB pamięci operacyjnej. Poniższa tabela przestawia czasy działania wszystkich przedstawionych algorytmów, jak również liczbę znalezionych cykli ($\#_c$). Algorytm wyszukujący cykle długości 6 używa algorytmu Chiby i Nishizeki'ego [11]

Algorytm 3.6.2 Zliczanie cykli długości 5 w grafie d-zdegenerowanym

1: $\#_c \leftarrow 0$
2: Zorientuj krawędzie G tak, aby otrzymać d-orientację $\overrightarrow{G}$.
3: Posortuj wierzchołki G tak, aby $\deg(v_1) \geq \deg(v_2) \geq \ldots \geq \deg(v_n)$.
4: **for each** $v \in V$ **do**
5: $\quad U(v) \leftarrow 0$
6: **for** $i \leftarrow 1$ **to** n **do**
7: $\quad$ **for each** $u \in N(v_i)$ **do**
8: $\quad\quad$ **for each** $w' \in N(u) - \{v_i\}$ **do** $U(w') \leftarrow U(w') + 1$
9: $\quad$ **for each** $u \in N(v_i)$ **do**
10: $\quad\quad$ **for each** $w \in N(u) - \{v_i\}$ **do** $U(w) \leftarrow U(w) - 1$
11: $\quad\quad$ **for each** $w \in N(u) - \{v_i\}$ **do**
12: $\quad\quad\quad$ **for each** $(w, x) \in E(G')$ **do**
13: $\quad\quad\quad\quad$ **if** $x \neq u$ **then**
14: $\quad\quad\quad\quad\quad$ **if** $(w, v_i) \in E(G')$ **or** $v_i \rightarrow w \in E(G')$ **then**
15: $\quad\quad\quad\quad\quad\quad \#_c \leftarrow \#_c + U(x) - 1$
16: $\quad\quad\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad\quad\quad \#_c \leftarrow \#_c + U(x)$
18: $\quad\quad$ **for each** $w \in N(u) - \{v_i\}$ **do** $U(w) \leftarrow U(w) + 1$
19: $\quad$ **for each** w **such that** $U(w) \neq 0$ **do** $U(w) \leftarrow 0$
20: $\quad$ Usuń v_i z G (wraz ze wszystkimi incydentnymi krawędziami).

do znajdowania trójkątów w $\widehat{GP_2}$. Jak widzimy, wyniki eksperymentów potwierdzają praktyczną przydatność algorytmów. Należy zwrócić uwagę, że algorytm wyszukujący cykle długości 4 jest znacznie szybszy niż algorytm wyszukujący trójkąty – jest to spowodowane brakiem potrzeby generowania grafu GP_3 podczas wyszukiwania 4-cykli.

algorytm	C_3	C_4	C_5	C_6	C_5 (Alg. 3.6.1)	zliczanie C_5 (Alg. 3.6.2)
czas [s]	28,03	20,70	51,55	93,43	26,94	5,89
$\#_c$	201 136	315 333	689 705	2 065 391	689 705	689 705

Bibliografia

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[2] O. Aichholzer, F. Aurenhammer, G. Rote. Ein dreifarbensatz fur dreikreisfreie netze auf der kuzel. SFB-Report F003-51, SFB 'Optimierung und Kontrolle', TU Graz, Austria, 1995.

[3] N. Alon, R. Yuster, U. Zwick. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. *Proc. 26th Annual ACM Symposium on Theory of Computing*, strony 326–335. ACM, 1994.

[4] N. Alon, R. Yuster, U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[5] K. Appel, W. Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[6] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. H. M. Smid, C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. *European Symposium on Algorithms*, strony 514–528, 1996.

[7] O. V. Borodin, A. N. Glebov, A. Raspaud, M. R. Salavatipour. Planar graphs without cycles of length from 4 to 7 are 3-colorable. 2003. Submitted to J. of Comb. Th. B.

[8] O. V. Borodin, A. Raspaud. A sufficient condition for planar graphs to be 3-colorable. *Journal of Combinatorial Theory, Series B*, 88:17–27, 2003.

[9] G. S. Brodal, R. Fagerberg. Dynamic representations of sparse graphs. *Proc. 6th Int. Workshop on Algorithms and Data Structures*, wolumen 1663 serii *LNCS*, strony 342–351. 1999.

[10] D. A. Chen, J. Xu. Shortest path queries in planar graphs. *Proc. of the 32nd Annual ACM Symposium on Theory of Computing*, strony 469–478, 2000.

[11] N. Chiba, T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[12] N. Chiba, T. Nishizeki, N.Saito. A linear algorithm for five-coloring a planar graph. *J. Algorithms*, 2:317–327, 1981.

[13] M. Chrobak, K. Diks. Two algorithms for coloring planar graphs with 5 colors. Raport instytutowy, Columbia University, January 1987.

[14] M. Chrobak, D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, 1991.

[15] E. Cohen, E. Halperin, H. Kaplan, U. Zwick. Reachability and distance queries via 2-hop labels. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, strony 937–946. Society for Industrial and Applied Mathematics, 2002.

[16] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Wprowadzenie do Algorytmów.*

[17] C. Demetrescu, M. Thorup. Oracles for distances avoiding a link-failure. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, strony 838–843. Society for Industrial and Applied Mathematics, 2002.

[18] R. Diestel. *Graph Theory.*

[19] H. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. *Proc. 22nd Int. Worksh. Graph-Theoretic Concepts in Computer Science (WG 1996)*, strony 151–165, 1996.

[20] H. Djidjev. Computing the girth of a planar graph. *Proc. 27th International Colloquium on Automata, Languages and Programming*, strony 821–831, 2000.

[21] D. Eppstein. Connectivity, graph minors, and subgraph multiplicity. *J. Graph Theory*, 17:409–416, 1993.

[22] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms & Applications*, 3(3):1–27, 1999.

[23] J. Fakcharoenphol, S. Rao. Planar graphs, negative weight edges, shortest paths, near linear time. *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, strony 232–241, 2001.

[24] G. N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, Gru. 1987.

[25] G. Fijavz, M. Juvan, B.Mohar, R. Skrekovski. Planar graphs without cycles of specified lengths, 2002.

[26] G. Frederickson. On linear-time algorithms for five-coloring planar graphs. *Information Processing Letters*, 19:219–224, 1984.

[27] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM Journal on Computing*, 26(2):484–538, April 1997.

[28] M. R. Garey, D. S. Johnson. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, February 1976.

[29] C. Gavoille, D. Peleg, S. Pérennes, R. Raz. Distance labeling in graphs. *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, strony 210–219. Society for Industrial and Applied Mathematics, 2001.

[30] J. Gimbel, C. Thomassen. Coloring graphs with fixed genus and girth. *Transactions of the AMS*, 349(11):4555–4564, November 1997.

[31] R. Graham, D. Knuth, O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[32] H. Grötzsch. Ein dreifarbensatz fur dreikreisfreie netze auf der kuzel. Raport instytutowy, Wiss. Z. Martin Luther Univ. Halle Wittenberg, Math.-Nat. Reihe 8, 1959.

[33] S. Kannan, M. Naor, S. Rudich. Implicit representation of graphs. *Proc. 20th Symposium on Theory of Computing*, strony 334–343. ACM, May 1998.

[34] P. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, strony 820–827. Society for Industrial and Applied Mathematics, 2002.

[35] P. Klein, S. Rao, M. Rauch, S. Subramanian. Faster shortest-path algorithms for planar graphs. *Proc. 26th Symposium on Theory of Computing*, strony 27–37. ACM, 1994.

[36] Ł. Kowalik. Short cycles in planar graphs. H. Bodlaender, redaktor, *Proc. 29th Int. Worksh. Graph-Theoretic Concepts in Computer Science (WG 2003)*, wolumen 2880 serii *Lecture Notes in Computer Science*, strony 284–296. Springer-Verlag, 2003.

[37] Ł. Kowalik. Fast 3-coloring triangle-free planar graphs. S. Albers, T. Radzik, redaktorzy, *Proc. 12th Annual European Symposium on Algorithms (ESA 2004)*, wolumen 3221 serii *Lecture Notes in Computer Science*, strony 436–447. Springer-Verlag, 2004.

[38] Ł. Kowalik, M. Kurowski. Oracles for bounded length shortest paths in planar graphs. *Przyjęte do druku w ACM Transactions on Algorithms*.

[39] Ł. Kowalik, M. Kurowski. Shortest path queries in planar graphs in constant time. *Proc. 35th Symposium on Theory of Computing*, strony 143–148. ACM, June 2003.

[40] P. C. B. Lam, B. Xu, J. Liu. The 4-choosability of plane graphs without 4-cycles. *Journal of Combinatorial Theory, Series B*, 76:117–126, 1999.

[41] K.-W. Lih, W. Wang. Choosability and edge choosability of planar graphs without five cycles. *Applied Mathematics Letters*, 15:561–565, 2002.

[42] R. J. Lipton, R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, Kwi. 1979.

[43] D. Matula, Y. Shiloach, R. Tarjan. Two linear-time algorithms for 5-coloring a planar graph. Raport instytutowy, Stanford University, 1980.

[44] B. Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239–254, 1985.

[45] C. S. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39:12, 1964.

[46] C. H. Papadimitriou, M. Yannakakis. The clique problem for planar graphs. *Information Processing Letters*, 13(4–5):131–133, 1981.

[47] D. Richards. Finding short cycles in planar graphs using separators. *Journal of Algorithms*, 7:382–394, 1986.

[48] N. Robertson, D. P. Sanders, P. Seymour, R. Thomas. Efficiently four-coloring planar graphs. *Proc. 28th Symposium on Theory of Computing*, strony 571–575. ACM, 1996.

[49] N. Robertson, D. P. Sanders, P. Seymour, R. Thomas. The four-color theorem. *Journal of Combinatorial Theory, Series B*, 70:2–44, 1997.

[50] C. Thomassen. Grötzsch's 3-color theorem and its counterparts for the torus and the projective plane. *Journal of Combinatorial Theory, Series B*, 62:268–279, 1994.

[51] C. Thomassen. 3-list coloring planar graphs of girth 5. *Journal of Combinatorial Theory, Series B*, 64:101–107, 1995.

[52] C. Thomassen. A short list color proof of Grötzsch's theorem. *Journal of Combinatorial Theory, Series B*, 88:189–192, 2003.

[53] C. Thomassen. Prywatna korespondencja. 2004.

[54] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, strony 242–251, 2001.

[55] M. Williams. A linear algorithm for colouring planar graphs with five colours. *Comput. J.*, 28:78–81, 1985.

[56] R. J. Wilson. *Wprowadzenie do teorii grafów.* PWN, 2002.

[57] U. Zwick. Exact and approximate distances in graphs - a survey. *Proceedings of the 9th Annual European Symposium on Algorithms*, strony 33–48. Springer-Verlag, 2001.

Indeks