Efektywne struktury danych dla zapytań wewnętrznych w tekstach

Autoreferat rozprawy doktorskiej

Tomasz Kociumaka

Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego

1 Algorytmy i struktury danych do przetwarzania tekstów

Teksty (słowa, napisy), czyli skończone ciągi znaków z pewnego alfabetu, są wśród najbardziej podstawowych typów danych. Występują w językach naturalnych, jako sekwencje biologiczne, programy komputerowe oraz pliki generowane maszynowo. Używane są także do serializacji danych i do reprezentacji ruchu sieciowego, trajektorii, czy też szeregów czasowych. Tym samym zadania obliczeniowe dotyczące słów pojawiają się w wielu obszarach informatyki stosowanej, w tym w bioinformatyce, w kompresji danych, w eksploracji danych, w systemach wykrywania włamań, w narzędziach antyplagiatowych, czy w wyszukiwarkach. Mnogość zastosowań motywuje teoretyczne i eksperymentalne badania algorytmów i struktur danych do przetwarzania tekstów; podstawy tej dziedziny omówiono m.in. w podręcznikach [21, 14, 13].

Centralnym zagadnieniem przetwarzania tekstów jest problem wyszukiwania wzorca, czyli znajdowania wystąpień jednego podanego słowa (wzorca) w drugim podanym słowie (tekście), tzn. fragmentów tekstu, które pasują do wzorca. Algorytm wyszukiwania wzorca w czasie liniowym znany jest od roku 1970 [36], ale optymalizacja zużycia pamięci i praktycznej wydajności używanych metod doprowadziła do powstania kilkudziesięciu kolejnych procedur [17]. W wielu sytuacjach przydatne okazuje się indeksowanie, czyli wstępne przetwarzanie tekstu, tak aby skonstruować strukturę danych, która szybko odpowiada na zapytania o wystąpienia dowolnego podanego wzorca. Drzewo sufiksowe [46] to klasyczny indeks z liniowym czasem konstrukcji, obsługujący zapytania w czasie proporcjonalnym do długości wzorca. Niemniej jednak jego relatywnie duży rozmiar doprowadził do rozwoju bardziej ekonomicznych alternatyw, takich jak tablica sufiksowa [35] oraz indeksy skompresowane [40]. Standardowo zgodność (dopasowanie) słów zdefiniowana jest przez ich równość, jednak bada się także rozmaite słabsze relacje, pojawiające się na przykład w związku z problemem przybliżonego wyszukiwania wzorca [38].

Kolejna ważna rodzina problemów dotyczy kompresji słownikowej, czyli klasy metod bezstratnej kompresji danych obejmującej między innymi algorytmy Lempela-Ziva [47], które wykorzystywane są w wielu standardowych formatach danych (takich jak GIF, PNG, PDF, ZIP, gzip). Kluczowe zagadnienia tej tematyki to nie tylko wydajne algorytmy kompresji i dekompresji, lecz także procedury do przetwarzania skompresowanych tekstów [34].

Następny istotny kierunek badań stanowi poszukiwanie regularności, takich jak powtórzenia czy palindromy, które pojawiają się w słowach [44]. Takie struktury są przede wszystkim obiektem zainteresowania kombinatoryki słów. Niektóre występują również naturalnie w pewnych zastosowaniach (jak np. powtórzenia tandemowe w sekwencjach DNA), podczas gdy inne pojawiają się w rozwiązaniach pozornie niezwiązanych problemów przetwarzania tekstów.

W wielu współczesnych zastosowaniach rozmiar danych tekstowych mierzony jest w gigabajtach. Na przykład już pojedynczy ludzki genom składa się z ponad 3 miliardów znaków (par zasad), a zadania obliczeniowe często dotyczą wielu takich genomów. Ta ogromna ilość danych

wymaga szybkiego i wydajnego pamięciowo przetwarzania, co rodzi wiele wyzwań natury algorytmicznej. W związku z tym w algorytmice tekstów zazwyczaj optymalizuje się wszystkie składowe złożoności czasowej ponad minimum niezbędne do wczytania wejścia. Złożoność pamięciowa algorytmów i struktur danych jest nawet ważniejsza ze względu na ryzyko przekroczenia rozmiaru dostępnej pamięci operacyjnej. Z tego powodu celem wielu badań jest uzyskanie liniowego czasu działania i liniowego rozmiaru pamięci roboczej. Coraz częściej zadanie jest bardziej ambitne: aby pokonać tę naturalną barierę, na przykład za pomocą zrównoleglenia na poziomie bitów, które wykorzystuje się w większości tzw. zwartych struktur danych [39].

2 Zapytania wewnętrzne w tekstach

Szerokie spektrum problemów przetwarzania tekstów wymaga wielu technik algorytmicznych. Niemniej jednak pewne pomocnicze zadania pojawiają się w całej dziedzinie, co implikuje szerokie wykorzystanie rozwiązujących je narzędzi. Dobry przykład stanowi tu problem wyznaczania długości najdłuższego wspólnego prefiksu dowolnych dwóch sufiksów tekstu. Wśród wielu zastosowań tych zapytań o najdłuższy wspólny prefiks są przybliżone wyszukiwanie wzorca [33], konstrukcja indeksów, w tym drzew sufiksowych [16] i tablic sufiksowych [24], oraz wyznaczanie maksymalnych powtórzeń w tekstach [8].

Zauważmy, że najdłuższy wspólny prefiks to pojęcie zdefiniowane dla dowolnych dwóch słów. Co więcej, jego długość łatwo obliczyć w optymalnym czasie liniowym, jeśli słowa te podane są jawnie na wejściu. W problemie zapytań o najdłuższy wspólny prefiks ograniczamy rodzinę dostępnych słów wejściowych do sufiksów ustalonego tekstu T i identyfikujemy każdy sufiks za pomocą jego pozycji początkowej. Ta transformacja czyni problem znacznie ciekawszym i przynosi wiele zastosowań, ponieważ teraz możemy wyznaczyć najdłuższy wspólny prefiks w czasie stałym po wstępnym przetworzeniu tekstu T w czasie liniowym [33, 16]. Proste uogólnienie tych zapytań pozwala operować nie tylko na sufiksach, lecz także na dowolnych podsłowach tekstu T. Każde takie podsłowo reprezentowane jest przez swoje wystąpienie w T, zwane *fragmentem* tekstu T i identyfikowane przez pozycje początkową i końcową. Zapytania o najdłuższy wspólny prefiks wykorzystywane są także do odpowiedzi na bardziej fundamentalne pytania dotyczące fragmentów tekstu T: do rozstrzygania, czy dwa fragmenty do siebie pasują, czyli czy są wystąpieniami tego samego podsłowa, oraz do porównywania fragmentów w porządku leksykograficznym.

W podobny sposób można przekształcić także inne problemy dotyczące jednego słowa lub większej ich liczby. Otrzymane w ten sposób pytania nazywamy *zapytaniami wewnętrznymi*. Formalnie, sformułowane w ten sposób zadanie polega na konstrukcji struktury danych dla podanego tekstu T, tak aby później można było rozwiązać instancje rozważanego problemu, których dane wejściowe stanowią podsłowa T wskazane jako fragmenty tego tekstu. W zastosowaniach praktycznych tekst T można interpretować jako „korpus” zawierający wszystkie dane zgromadzone na potrzeby prowadzonej analizy (np. jako konkatenację sekwencji biologicznych do porównania). Gdy z kolei zapytania wewnętrzne pojawiają się w toku działania algorytmu, wówczas T zazwyczaj składa się po prostu z danych wejściowych tego algorytmu.

W rozprawie przedstawiamy struktury danych dla kilku rodzajów zapytań wewnętrznych. Część z tych problemów pojawiała się już w literaturze, inne są wewnętrznymi odpowiednikami klasycznych zadań przetwarzania tekstów.

2.1 Zapytania o najdłuższy wspólny prefiks

Z perspektywy czasu, za początek zapytań wewnętrznych w tekstach należy przyjąć sformułowanie zapytań o najdłuższy wspólny prefiks, oryginalnie wprowadzonych przez Landaua i Vishkina na potrzeby przybliżonego wyszukiwania wzorca względem odległości edycyjnej [33]. Przypomnijmy, że ów pomocniczy problem polega na skonstruowaniu struktury danych, która obsługiwać będzie następujące zapytania dotyczące tekstu T:

Zapytania o najdłuższy wspólny prefiks

Dla danych pozycji i, i' tekstu T oblicz $\mathrm{LCE}(i, i') = \mathrm{lcp}(T[i\,..], T[i'\,..])$, tzn. długość najdłuższego wspólnego prefiksu sufiksów zaczynających się na pozycjach i oraz i'.

Klasyczna struktura danych dla zapytań o najdłuższy wspólny prefiks [33] ma rozmiar $\mathcal{O}(n)$, odpowiada na zapytania w czasie stałym i można ją skonstruować w czasie $\mathcal{O}(n)$ [16]. Czasy konstrukcji i zapytania są oczywiście optymalne, a rozmiar $\mathcal{O}(n)$ komórek pamięci (czyli $\mathcal{O}(n \log n)$ bitów) także jest w ogólności niezbędny. Niemniej jednak, to ostatnie ograniczenie dolne nie zachodzi, jeśli rozmiar alfabetu σ jest znacznie mniejszy niż długość tekstu n. Dokładniej, tekst zajmuje wtedy jedynie $\mathcal{O}(n \log \sigma)$ bitów, czyli $o(n \log n)$ o ile $\sigma = n^{o(1)}$. Co więcej, jeśli zezwolimy, aby taka *spakowana* reprezentacja tekstu T pojawiła się wejściu, otwieramy możliwość uzyskania czasu konstrukcji $o(n)$.

Niedawne prace kilku grup badawczych przyniosły progres w obsłudze zapytań o najdłuższy wspólny prefiks w tym modelu. Tanimura i in. [45] oraz Munro i in. [37] stworzyli struktury danych o stałym czasie zapytania i rozmiarach odpowiednio $\mathcal{O}(n\sqrt{\log n} \log \sigma)$ oraz $\mathcal{O}(n\sqrt{\log n \log \sigma})$ bitów. Drugie z tych rozwiązań posiada także efektywny algorytm konstrukcji, który, otrzymawszy tekst T w spakowanej reprezentacji, działa w czasie $\mathcal{O}(n/\sqrt{\log_\sigma n})$. W kolejnej pracy Birenzwige i in. [9] wykorzystali nasze techniki lokalnej zgodności (omówione w Sekcji 3), tak iż zapytania o najdłuższy wspólny prefiks w czasie stałym i pamięci $\mathcal{O}(n \log \sigma)$ bitów można otrzymać jako nietrudny wniosek. Niestety oryginalna implementacja używanych tam narzędzi daje jedynie randomizowany algorytm konstrukcji w czasie $\mathcal{O}(n)$. W rozprawie przedstawiamy deterministyczną konstrukcję zoptymalizowaną na potrzeby modelu spakowanych słów. Pozwala to otrzymać nasz pierwszy główny wynik:

Twierdzenie 1 *Dla każdego tekstu T długości n nad alfabetem wielkości σ istnieje struktura danych wielkości $\mathcal{O}(n \log \sigma)$ bitów (tj. $\mathcal{O}(n/\log_\sigma n)$ komórek pamięci), która odpowiada na zapytania o najdłuższy wspólny prefiks w czasie $\mathcal{O}(1)$. Można ją skonstruować w czasie $\mathcal{O}(n/\log_\sigma n)$ na podstawie spakowanej reprezentacji tekstu T.*

2.2 Zapytania o okresy

Jednym z podstawowych pojęć kombinatoryki słów jest *okres*. Liczba naturalna p jest okresem słowa w długości m, jeśli $1 \leq p \leq m$ oraz istnieje słowo u długości p, takie że w jest prefiksem u^k (czyli konkatenacji k kopii słowa u) dla wystarczająco dużej liczby naturalnej k. Równoważnie, słowo w ma okres p wtedy i tylko wtedy, gdy jego prefiks długości $m - p$ pasuje do sufiksu długości $m - p$; takie podsłowo, które występuje zarówno jako prefiks w jak i jako sufiks w, nazywamy *prefikso-sufiksem* słowa w.

Zbiór wszystkich okresów słowa jest wyznaczany w czasie $\mathcal{O}(m)$ jako element algorytmu Morrisa-Pratta do wyszukiwania wzorca [36]. Co więcej, o ile słowo długości m może mieć aż m okresów, posortowany ciąg tych wartości zawsze można podzielić na $\mathcal{O}(\log m)$ ciągów arytmetycznych. Rozmiar takiej reprezentacji ($\mathcal{O}(\log^2 m)$ bitów) jest asymptotycznie optymalny w pesymistycznym przypadku, w związku z czym używamy jej na wyjściu wyspecyfikowanej poniżej wewnętrznej wersji problemu wyznaczania okresów:

Zapytania o okresy

Dla danego fragmentu x tekstu T wyznacz wszystkie okresy podsłowa wskazanego przez x (reprezentowane za pomocą rozłącznych ciągów arytmetycznych).

Zapytania o okresy wprowadziliśmy w pracy [28]. W rozprawie przedstawiamy natomiast strukturę danych, która jest asymptotycznie optymalna dla tekstów nad alfabetami całkowitoliczbowymi wielkości wielomianowej (ze względu na n).

Twierdzenie 2 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na* zapytania o okresy *w czasie $\mathcal{O}(\log |x|)$. Można ją skonstruować w czasie $\mathcal{O}(n)$.*

Nasz algorytm obsługi zapytań opiera się na ścisłej zależności między prefikso-sufiksami a okresami słów. Aby odpowiedzieć na zapytanie o okresy fragmentu x, łączy wyniki następujących zapytań o prefikso-sufiksy zadawanych dla $x = y$, tak aby wyznaczyć wszystkie prefikso-sufiksy podsłowa reprezentowanego przez x.

Zapytania o prefikso-sufiksy

Dla danych fragmentów x i y tekstu T oraz liczby naturalnej d wyznacz wszystkie sufiksy y długości między d a $2d - 1$, które występują także jako prefiksy x (reprezentowane przez ciąg arytmetyczny ich długości).

Nasz pomocniczy wynik dla zapytań o prefikso-sufiksy został także wykorzystany w problemie dynamicznego wyznaczania najdłuższego wspólnego podsłowa [2] oraz do obliczania najdłuższego podsłowa bez nietrywialnych prefikso-sufiksów [27].

Twierdzenie 3 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na* zapytania o prefikso-sufiksy *w czasie $\mathcal{O}(1)$. Można ją skonstruować w czasie $\mathcal{O}(n)$.*

W wielu zastosowaniach znaczące są jedynie stosunkowo krótkie okresy. Odpowiadają one długim prefikso-sufiksom, co pozwala wyznaczyć je za pomocą niewielu zapytań o prefikso-sufiksy. Ta ograniczona wersja zapytań o okresy okazała się użyteczna w algorytmach wyznaczania powtórzeń z odstępem (ang. *gapped repeats*) i słabych powtórzeń (ang. *subrepetitions*) [31, 19]. Co więcej, nasze rozwiązanie dla szczególnie istotnego przypadku okresów $p \leq \frac{1}{2}|x|$ zostało wykorzystane do odtwarzania przybliżonych okresów [1], do znajdowania dwuwymiarowych maksymalnych powtórzeń [3] oraz do wyszukiwania wzorców z jedną zmienną [32].

2.3 Wewnętrzne wyszukiwanie wzorca

Główny element naszego rozwiązania dla zapytań o prefikso-sufiksy stanowi w rzeczywistości struktura danych dla wewnętrznej wersji problemu wyszukiwania wzorca, czyli do wyszukiwania wystąpień jednego podsłowa zlokalizowanych w obrębie innego podsłowa:

Wewnętrzne wyszukiwanie wzorca

Dla danych fragmentów x i y tekstu T spełniających warunek $|y| < 2|x|$ wyznacz pozycje początkowe fragmentów pasujących do x i zawartych w y (reprezentowane jako ciąg arytmetyczny).

Nakładamy ograniczenie $|y| < 2|x|$ na długości fragmentów w zapytaniu, aby zapewnić, że pozycje początkowe wystąpień x w y tworzą jeden ciąg arytmetyczny, więc w szczególności można je reprezentować w pamięci stałej. Jeśli $|y| \geq 2|x|$, można zadać $\mathcal{O}(|y|/|x|)$ zapytań wewnętrznego wyszukiwania wzorca i zwrócić na wyjściu $\mathcal{O}(|y|/|x|)$ ciągów arytmetycznych.

Twierdzenie 4 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na zapytania* wewnętrznego wyszukiwania wzorca *w czasie $\mathcal{O}(1)$. Można ją skonstruować w czasie $\mathcal{O}(n)$.*

O ile problem wewnętrznego wyszukiwania wzorca nie był wcześniej rozważany, Keller i in. [25] badali jego wersję decyzyjną i otrzymali kilka struktur danych różniących się przyjętym kompromisem między wielkością a czasem obsługi zapytania. Jedno z ich rozwiązań ma rozmiar $\mathcal{O}(n)$, czas konstrukcji $\mathcal{O}(n\sqrt{\log n})$ i odpowiada na zapytania w czasie $\mathcal{O}(\log^{\varepsilon} n)$ (dla dowolnej

stałej $\varepsilon > 0$). Wspomniany tu czas zapytania jest w mocy dla dowolnych długości $|x|$ oraz $|y|$, a więc nie da się bezpośrednio porównać efektywności tej struktury danych ze stworzoną przez nas. Keller i in. [25] wprowadzili także bardziej ogólne zapytania o najdłuższy występujący prefiks zdefiniowane w następujący sposób:

Zapytania o najdłuższy występujący prefiks
Dla danych fragmentów x i y tekstu T znajdź najdłuższy prefiks p podsłowa x, który występuje w y.

Połączywszy nasz wynik dla wewnętrznego wyszukiwania wzorca z technikami pracy [25], otrzymaliśmy szybszą implementację zapytań o najdłuższy występujący prefiks, poprawiając czas zapytania z $\mathcal{O}(\log^{1+\varepsilon} n)$ do $\mathcal{O}(\log^{\varepsilon} n)$.

Twierdzenie 5 *Dla każdego tekstu T długości n i stałej $\varepsilon > 0$ istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na zapytania o najdłuższy występujący prefiks w czasie $\mathcal{O}(\log^{\varepsilon} n)$. Można ją skonstruować w czasie $\mathcal{O}(n\sqrt{\log n})$.*

2.4 Kompresja podsłów

Oryginalną motywację dla zapytań o najdłuższy występujący prefiks stanowiła problematyka kompresji podsłów, czyli zapytania wewnętrzne o skompresowaną reprezentację podsłów. Ta rodzina problemów została wprowadzone przez Cormode'a i Muthukrishnana [12], a niektóre wyniki poprawili później Keller i in. [25]. Zapytania kompresji podsłów mają stosunkowo bezpośrednią motywację: Rozważmy serwer utrzymujący długi i bogaty w powtórzenia tekst T oraz klientów pytających o podsłowa T (np. fragmenty, które należy wyświetlić). Ograniczona przepustowość kanału komunikacji uzasadnia stosowanie kompresji.

Obie wspomniane powyżej prace wykorzystują klasyczny algorytm kompresji LZ77 [47]. Badają zapytania wewnętrzne o faktoryzację LZ wskazanego fragmentu x oraz o uogólnioną faktoryzację jednego fragmentu x w kontekście innego fragmentu y. Uogólniona faktoryzacja jest tu zdefiniowana jako część odpowiadająca x w faktoryzacji LZ słowa $y\#x$, gdzie # jest specjalnym symbolem-strażnikiem, który nie występuje nigdzie indziej. Zapytania o najdłuższy występujący prefiks naturalnie pojawiają się przy rozwiązywaniu poniższego problemu:

Uogólniona kompresja podsłów metodą LZ77
Dla danych fragmentów x i y tekstu T oblicz uogólnioną faktoryzację x względem y.

W ten sposób nasze usprawnienia dla zapytań o najdłuższy występujący prefiks natychmiast dają rozwiązanie problemu uogólnionej kompresji podsłów metodą LZ77 w czasie $\mathcal{O}(C \cdot \log^{\varepsilon} n)$, poprawiając uprzednio znany czas odpowiedzi $\mathcal{O}(C \cdot \log^{\varepsilon} n \cdot \log \frac{|x|}{C})$; wartość C oznacza tutaj liczbę fraz w wyznaczonej faktoryzacji LZ. Zauważamy także, że kilka innych wariantów kompresji LZ77 można zbudować w czasie $\mathcal{O}(C \cdot \log^{\varepsilon} n)$, czyli takim samym jak uzyskany wcześniej dla podstawowej wersji kompresji podsłów metodą LZ77 [25].

O ile oryginalne prace o kompresji podsłów skupiają się na metodzie LZ77, Cormode i Muthukrishnan [12] pozostawili inne schematy kompresji jako temat do dalszych badań. W szczególności, wspomnieli w tym kontekście o algorytmach opartych na transformacie Burrowsa-Wheelera [11]. W rozprawie podążamy we wskazanym przez nich kierunku i rozważamy kompresję podsłów względem najprostszej procedury kompresji wykorzystującej tę transformatę, gdzie przekształcony za jej pomocą tekst jest kodowany metodą długości serii (ang. *run-length encoding*).

Kompresja podsłów metodą BWT+RLE
Dla danego fragmentu x tekstu T wyznacz kodowanie długości serii transformaty Burrowsa-Wheelera podsłowa reprezentowanego przez x, czyli wartość $\mathrm{RLE}(\mathrm{BWT}(x))$.

Nasza struktura danych ma czas odpowiedzi powiększony o czynnik logarytmiczny względem rozmiaru wyjścia.

Twierdzenie 6 *Dla każdego tekstu T długości n istnieje struktura danych wielkości liniowej, która odpowiada na zapytania* kompresji podsłów metodą BWT+RLE *w czasie $\mathcal{O}(C \cdot \log |x|)$, gdzie C jest rozmiarem kodowania długości serii transformaty Burrowsa-Wheelera podsłowa x. Można ją skonstruować w czasie $\mathcal{O}(n\sqrt{\log n})$.*

2.5 Ranking i selekcja sufiksów podsłów

Głównym zastosowaniem struktury danych, której używamy do kompresji podsłów metodą BWT+RLE, jest jednak symulacja tablicy sufiksowej [35] i odwrotnej tablicy sufiksowej wskazanego podsłowa. Wyznaczenie wartości k-tej komórki takiej tablicy sufiksowej można sformułować jako problem odpowiedzi na następujące zapytania wewnętrzne:

Selekcja sufiksów podsłów

Dla danego fragmentu x tekstu T oraz liczby naturalnej k wyznacz k-ty sufiks x w porządku leksykograficznym.

Problem symulacji odwrotnej tablicy sufiksowej polega natomiast na wyznaczeniu rangi wskazanego sufiksu podsłowa x wśród wszystkich sufiksów x, czyli na określeniu liczby sufiksów mniejszych lub równych od podanego. Nasze zapytania rankingu sufiksów podsłów, zdefiniowane poniżej, są nieco ogólniejsze: pozwalają obliczyć rangę dowolnego podsłowa T.

Ranking sufiksów podsłów

Dla danych fragmentów x oraz y tekstu T oblicz rangę y wśród sufiksów x w porządku leksykograficznym.

Łącząc nasz komponent do wewnętrznego wyszukiwania wzorca z technikami stosowanymi do przeszukiwania obszarów ortogonalnych, otrzymujemy następujący wynik:

Twierdzenie 7 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na zapytania* selekcji sufiksów podsłów *i* rankingu sufiksów podsłów *w czasie $\mathcal{O}(\log |x|)$. Można ją skonstruować w czasie $\mathcal{O}(n\sqrt{\log n})$.*

O ile logarytmiczny czas odpowiedzi nie musi być optymalny, to wiadomo, iż nie da się go poprawić o więcej niż czynnik $\mathcal{O}(\log \log n)$, nawet kosztem niewielkiego zwiększenia rozmiaru struktury danych.

Stwierdzenie 8 *Każda struktura danych wielkości $\mathcal{O}(n \log^{\mathcal{O}(1)} n)$ musi wymagać czasu pesymistycznego $\Omega(\frac{\log n}{\log \log n})$ dla zapytań* selekcji sufiksów podsłów *i* rankingu sufiksów podsłów.

2.6 Zapytania o najmniejszy sufiks i najmniejszą rotację cykliczną

Dla $k = 1$ oraz $k = |x|$ zapytania selekcji sufiksów podsłów redukują się do problemu wyznaczenia leksykograficznie najmniejszego i największego sufiksu wskazanego podsłowa.

Zapytania o najmniejszy (największy) sufiks

Dla danego fragmentu x tekstu T wyznacz najmniejszy (największy) leksykograficznie niepusty sufiks podsłowa reprezentowanego przez x.

Klasyczny algorytm Duvala [15] oblicza zarówno najmniejszy jak i największy sufiks słowa w czasie liniowym i w stałej pamięci roboczej. Jego proste rozszerzenie pozwala także wyznaczyć takie sufiksy dla każdego prefiksu podanego na wejściu tekstu. W czasie kwadratowym można zatem uzyskać odpowiedzi na wszystkie możliwe zapytania o najmniejszy oraz największy sufiks w tekście T.

Pierwsze nietrywialne struktury danych dla tych zapytań przedstawili Babenko i in. [7], pokazując, jak odpowiadać na zapytania o najmniejszy sufiks w czasie $\mathcal{O}(\log^{1+\varepsilon} n)$, a o największy sufiks — w czasie $\mathcal{O}(\log n)$, w obu przypadkach przy użyciu pamięci liniowej. Późniejsza praca [5] poprawiła czas zapytania do stałego, zachowując jednocześnie liniowy rozmiar struktur danych. Stworzony tam komponent dla zapytań o największy sufiks posiada także algorytm konstrukcji w czasie $\mathcal{O}(n)$, co czyni go optymalnym rozwiązaniem dla alfabetów całkowitoliczbowych wielkości wielomianowej. Konstrukcja struktury odpowiadającej w czasie stałym na zapytania o najmniejszy sufiks wymaga z kolei czasu $\mathcal{O}(n \log n)$. W rozprawie przedstawiamy pierwsze rozwiązanie dla zapytań o najmniejszy sufiks, które jednocześnie uzyskuje czas konstrukcji $\mathcal{O}(n)$ i czas zapytania $\mathcal{O}(1)$.

Twierdzenie 9 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na zapytania o najmniejszy sufiks w czasie $\mathcal{O}(1)$. Można ją skonstruować w czasie $\mathcal{O}(n)$.*

Stworzonych w tym celu technik używamy także do wyznaczania najmniejszej i największej leksykograficzne rotacji cyklicznej podsłowa. Obie wersje tych zapytań są symetryczne (z dokładnością do odwrócenia porządku nad alfabetem), więc skupiamy się na wersji minimalizacyjnej:

Zapytania o najmniejszą rotację cykliczną

Dla danego fragmentu x tekstu T wyznacz najmniejszą leksykograficznie rotację cykliczną podsłowa reprezentowanego przez x.

Pierwszy algorytm obliczający w czasie liniowym najmniejszą rotację cykliczną podanego słowa zaproponował Booth [10]. Później Duval [15] przedstawił implementację charakteryzującą się dodatkowo stałym rozmiarem pamięci roboczej. Następnie Apostolico i Crochemore [4] opisali procedurę wyznaczającą w czasie liniowym najmniejszą rotację cykliczną każdego prefiksu podanego na wejściu tekstu.

Nasza struktura danych dla zapytań o najmniejszą rotację cykliczną jest pierwszym rozwiązaniem tego problemu, ale jednocześnie jest ona optymalna dla tekstów nad alfabetem całkowitoliczbowym wielkości wielomianowej.

Twierdzenie 10 *Dla każdego tekstu T długości n istnieje struktura danych wielkości $\mathcal{O}(n)$, która odpowiada na zapytania o najmniejszą rotację cykliczną w czasie $\mathcal{O}(1)$. Można ją skonstruować w czasie $\mathcal{O}(n)$.*

Zapytania o najmniejszy sufiks i najmniejszą rotację cykliczną mają kilka naturalnych zastosowań. Tych pierwszych używa się między innymi do wyznaczania faktoryzacji Lyndona wskazanego podsłowa. Główną motywacją zapytań o najmniejszą rotację cykliczną jest z kolei klasyfikacja podsłów ze względu na równoważność cykliczną.

3 Nowe techniki

Oprócz przedstawionych powyżej wyników efektem naszych badań jest także rozwój technik przetwarzania tekstów. Naszą główną innowacją i kluczowym narzędziem stojącym za strukturami dla zapytań o najdłuższy wspólny prefiks i wewnętrznego wyszukiwania wzorca jest nowa implementacja *lokalnej zgodności* — pomysłu, aby łamać symetrię między pozycjami i tekstu T na podstawie znaków $T[j]$ na pozycjach j bliskich pozycji i. Dzięki temu możemy zapewnić, że pasujące do siebie fragmenty tekstu są przetwarzane w ten sam sposób.

Poprzednie implementacje lokalnej zgodności opierają się na parsowaniu tekstu. Klasyczne parsowanie lokalnie zgodnie (ang. *locally consistent parsing*) Sahinalpa i Vishkina [42] zostało z sukcesem wykorzystane do wielu problemów takich jak równoległa konstrukcja drzew sufiksowych, przybliżone wyszukiwanie wzorca i indeksowanie dynamicznych tekstów [42, 43, 41]. Znacznie później Jeż stworzył *rekompresję* [23]: technikę, która daje prostsze i efektywniejsze parsowanie. Poza nowymi zastosowaniami lokalnej zgodności, między innymi w rozwiązywaniu równań na słowach [23] i w wyszukiwaniu wzorca w tekstach skompresowanych gramatycznie [22], rekompresja pozwoliła także poprawić i rozszerzyć kilka wyników opartych na wcześniejszych parsowaniach lokalnie zgodnych [20, 18].

Metody te nie są niestety odpowiednie do implementacji zapytań wewnętrznych w czasie stałym. Wynika to przede wszystkim z faktu, że uzyskanie użytecznych informacji o fragmencie tekstu wymaga czasu proporcjonalnego do głębokości parsowania, która jest zwykle logarytmiczna. Co więcej, rozmiar kontekstu na każdym poziomie parsowania jest wyrażony za pomocą liczby fraz, których długość możne znacznie różnić się między poszczególnymi częściami tekstu. Aby pokonać te trudności, zamiast używać parsowania lokalnie zgodnego definiujemy *funkcje synchronizujące*, które wybierają $\mathcal{O}(n/\tau)$ fragmentów wskazanej długości τ, bazując na zawartości kontekstu wielkości $\mathcal{O}(\tau)$. Jednolita długość kontekstu i stały czas ewaluacji to najważniejsze cechy charakterystyczne naszej techniki.

Silnie okresowe fragmenty wymagają szczególnej uwagi w wielu metodach przetwarzania tekstów. Dotyczy to także zastosowań idei lokalnej zgodności, ponieważ łamanie symetrii bywa niemożliwe obrębie okresowych części tekstu. Parsowanie lokalnie zgodne automatycznie znajduje i kompresuje okresowe fragmenty tekstu, ale konstruując funkcje synchronizujące, potrzebujemy zewnętrznego narzędzia do lokalizacji i obsługi takich fragmentów. Wykorzystujemy w tym celu fakt, że strukturę tych fragmentów kodują maksymalne powtórzenia (ang. *runs*) [30]. To pojęcie kombinatoryki słów było w ciągu ostatnich dwóch dekad przedmiotem intensywnych prac, które doprowadziły niedawno do przełomowych wyników [8]. Nasz wkład w rozwój tego szczególnego kierunku badań stanowi przede wszystkim stworzenie pomocniczej struktury danych pozwalającej szybko rozszerzyć wskazany okresowy fragment tekstu do maksymalnego powtórzenia o tym samym okresie.

4 Publikacje związane z rozprawą

Większość rezultatów zawartych w rozprawie pochodzi z trzech prac konferencyjnych:

- Praca [29] składa się z wyników dla wewnętrznego wyszukiwania wzorca, zapytań o okresy i kompresji podsłów metodą LZ77. Struktura danych dla zapytań o najdłuższy wspólny prefiks jest nowym zastosowaniem wprowadzonych w tej publikacji technik. Ponadto oryginalne randomizowane algorytmy konstrukcji w rozprawie zastępujemy nowymi deterministycznymi.

- Praca [6] zawiera wyniki dla selekcji sufiksów podsłów, rankingu sufiksów podsłów oraz kompresji podsłów metodą BWT+RLE.

- Praca [26] składa się z wyników dla zapytań o najmniejszy sufiks i najmniejszą rotację cykliczną.

Dwie spośród wymienionych wyżej publikacji [29, 6] zostały zaprezentowane na konferencji *26th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2015). Trzecia [26] zdobyła nagrodę *Alberto Apostolico Best Paper Award* na konferencji *27th Annual Symposium on Combinatorial Pattern Matching* (CPM 2016).

Bibliografia

[1] A. Amir, M. Amit, G. M. Landau i D. Sokol. Period recovery of strings over the Hamming and edit distances. *Theoretical Computer Science*, 710:2–18, 2018. DOI: 10.1016/j.tcs.2017.10.026.

[2] A. Amir, P. Charalampopoulos, S. P. Pissis i J. Radoszewski. Longest common factor made fully dynamic, 2018. arXiv: 1804.08731.

[3] A. Amir, G. M. Landau, S. Marcus i D. Sokol. Two-dimensional maximal repetitions. [W:] Y. Azar, H. Bast i G. Herman, redaktorzy, *Algorithms, ESA 2018*, tom 112 serii *LIPIcs*, 2:1–2:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. DOI: 10.4230/LIPIcs.ESA.2018.2.

[4] A. Apostolico i M. Crochemore. Optimal canonization of all substrings of a string. *Information and Computation*, 95(1):76–95, 1991. DOI: 10.1016/0890-5401(91)90016-U.

[5] M. Babenko, P. Gawrychowski, T. Kociumaka, I. Kolesnichenko i T. Starikovskaya. Computing minimal and maximal suffixes of a substring. *Theoretical Computer Science*, 638:112–121, 2016. DOI: 10.1016/j.tcs.2015.08.023.

[6] M. Babenko, P. Gawrychowski, T. Kociumaka i T. Starikovskaya. Wavelet trees meet suffix trees. [W:] P. Indyk, redaktor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, strony 572–591. SIAM, 2015. DOI: 10.1137/1.9781611973730.39.

[7] M. Babenko, I. Kolesnichenko i T. Starikovskaya. On minimal and maximal suffixes of a substring. [W:] J. Fischer i P. Sanders, redaktorzy, *Combinatorial Pattern Matching, CPM 2013*, tom 7922 serii *LNCS*, strony 28–37. Springer, 2013. DOI: 10.1007/978-3-642-38905-4_5.

[8] H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda i K. Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. DOI: 10.1137/15M1011032.

[9] O. Birenzwige, S. Golan i E. Porat. Locally consistent parsing for text indexing in small space. Nieopublikowany manuskrypt, 2017.

[10] K. S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4–5):240–242, 1980. DOI: 10.1016/0020-0190(80)90149-0.

[11] M. Burrows i D. J. Wheeler. A block-sorting lossless data compression algorithm. Sprawozdanie techniczne 124, Digital Equipment Corporation, Palo Alto, California, 1994. URL: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[12] G. Cormode i S. Muthukrishnan. Substring compression problems. [W:] *16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, strony 321–330. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070478.

[13] M. Crochemore, C. Hancart i T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007. DOI: 10.1017/cbo9780511546853.

[14] M. Crochemore i W. Rytter. *Jewels of Stringology*. World Scientific, 2003. DOI: 10.1142/4838.

[15] J.-P. Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. DOI: 10.1016/0196-6774(83)90017-2.

[16] M. Farach-Colton, P. Ferragina i S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *Journal of the ACM*, 47(6):987–1011, 2000. DOI: 10.1145/355541.355547.

[17] S. Faro i T. Lecroq. The exact online string matching problem: A review of the most recent results. *ACM Computing Surveys*, 45(2):13:1–13:42, 2013. DOI: 10.1145/2431211.2431212.

[18] M. Gańczorz, P. Gawrychowski, A. Jeż i T. Kociumaka. Edit distance with block operations. [W:] Y. Azar, H. Bast i G. Herman, redaktorzy, *Algorithms, ESA 2018*, tom 112 serii *LIPIcs*, 33:1–33:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. DOI: 10.4230/LIPIcs.ESA.2018.33.

[19] P. Gawrychowski, T. I, S. Inenaga, D. Köppl i F. Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes: finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory of Computing Systems*, 62(1):162–191, 2018. DOI: 10.1007/s00224-017-9794-5.

[20] P. Gawrychowski, A. Karczmarz, T. Kociumaka, J. Łącki i P. Sankowski. Optimal dynamic strings. [W:] A. Czumaj, redaktor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, strony 1509–1528. SIAM, 2018. DOI: 10.1137/1.9781611975031.99.

[21] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. DOI: 10.1017/cbo9780511574931.

[22] A. Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. DOI: 10.1145/2631920.

[23] A. Jeż. Recompression: A simple and powerful technique for word equations. *Journal of the ACM*, 63(1):4:1–4:51, 2016. DOI: 10.1145/2743014.

[24] J. Kärkkäinen, P. Sanders i S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006. DOI: 10.1145/1217856.1217858.

[25] O. Keller, T. Kopelowitz, S. Landau Feibish i M. Lewenstein. Generalized substring compression. *Theoretical Computer Science*, 525:42–54, 2014. DOI: 10.1016/j.tcs.2013.10.010.

[26] T. Kociumaka. Minimal suffix and rotation of a substring in optimal time. [W:] R. Grossi i M. Lewenstein, redaktorzy, *Combinatorial Pattern Matching, CPM 2016*, tom 54 serii *LIPIcs*, 28:1–28:12. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. DOI: 10.4230/LIPIcs.CPM.2016.28.

[27] T. Kociumaka, R. Kundu, M. Mohamed i S. P. Pissis. Longest unbordered factor in quasilinear time. [W:] W.-L. Hsu, D.-T. Lee i C.-S. Liao, redaktorzy, *Algorithms and Computation, ISAAC 2018*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. arXiv: 1805.09924.

[28] T. Kociumaka, J. Radoszewski, W. Rytter i T. Waleń. Efficient data structures for the factor periodicity problem. [W:] L. Calderón-Benavides, C. N. González-Caro, E. Chávez i N. Ziviani, redaktorzy, *String Processing and Information Retrieval, SPIRE 2012*, tom 7608 serii *LNCS*, strony 284–294. Springer, 2012. DOI: 10.1007/978-3-642-34109-0_30.

[29] T. Kociumaka, J. Radoszewski, W. Rytter i T. Waleń. Internal pattern matching queries in a text and applications. [W:] P. Indyk, redaktor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, strony 532–551. SIAM, 2015. DOI: 10.1137/1.9781611973730.36.

[30] R. M. Kolpakov i G. Kucherov. Finding maximal repetitions in a word in linear time. [W:] *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, strony 596–604. IEEE Computer Society, 1999. DOI: 10.1109/SFFCS.1999.814634.

[31] R. Kolpakov, M. Podolskiy, M. Posypkin i N. Khrapov. Searching of gapped repeats and subrepetitions in a word. *Journal of Discrete Algorithms*, 46–47:1–15, 2017. DOI: 10.1016/j.jda.2017.10.004.

[32] D. Kosolobov, F. Manea i D. Nowotka. Detecting one-variable patterns. [W:] G. Fici, M. Sciortino i R. Venturini, redaktorzy, *String Processing and Information Retrieval, SPIRE 2017*, tom 10508 serii *LNCS*, strony 254–270. Springer, 2017. DOI: 10.1007/978-3-319-67428-5_22.

[33] G. M. Landau i U. Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. DOI: 10.1016/0022-0000(88)90045-1.

[34] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. DOI: 10.1515/gcc-2012-0016.

[35] U. Manber i E. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. DOI: 10.1137/0222058.

[36] J. H. Morris Jr. i V. R. Pratt. A linear pattern-matching algorithm. Sprawozdanie techniczne 40, Department of Computer Science, University of California, Berkeley, 1970.

[37] J. I. Munro, G. Navarro i Y. Nekrich. Text indexing and searching in sublinear time, 2017. arXiv: 1712.07431.

[38] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001. DOI: 10.1145/375360.375365.

[39] G. Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016. DOI: 10.1017/cbo9781316588284.

[40] G. Navarro i V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007. DOI: 10.1145/1216370.1216372.

[41] S. C. Sahinalp i U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. [W:] *37th IEEE Annual Symposium on Foundations of Computer Science, FOCS 1996*, strony 320–328. IEEE Computer Society, 1996. DOI: 10.1109/SFCS.1996.548491.

[42] S. C. Sahinalp i U. Vishkin. On a parallel-algorithms method for string matching problems. [W:] M. A. Bonuccelli, P. Crescenzi i R. Petreschi, redaktorzy, *Algorithms and Complexity, CIAC 1994*, tom 778 serii *LNCS*, strony 22–32. Springer, 1994. DOI: 10.1007/3-540-57811-0_3.

[43] S. C. Sahinalp i U. Vishkin. Symmetry breaking for suffix tree construction. [W:] F. T. Leighton i M. T. Goodrich, redaktorzy, *26th Annual ACM Symposium on Theory of Computing, STOC 1994*, strony 300–309. ACM, 1994. DOI: 10.1145/195058.195164.

[44] W. F. Smyth. Computing regularities in strings: A survey. *European Journal of Combinatorics*, 34(1):3–14, 2013. DOI: 10.1016/j.ejc.2012.07.010.

[45] Y. Tanimura, T. Nishimoto, H. Bannai, S. Inenaga i M. Takeda. Small-space LCE data structure with constant-time queries. [W:] K. G. Larsen, H. L. Bodlaender i J. Raskin, redaktorzy, *Mathematical Foundations of Computer Science, MFCS 2017*, tom 83 serii *LIPIcs*, 10:1–10:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. DOI: 10.4230/LIPIcs.MFCS.2017.10.

[46] P. Weiner. Linear pattern matching algorithms. [W:] *14th Annual Symposium on Switching and Automata Theory, SWAT 1973*, strony 1–11. IEEE Computer Society, 1973. DOI: 10.1109/SWAT.1973.13.

[47] J. Ziv i A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. DOI: 10.1109/TIT.1977.1055714.