Wprowadzenie do kombinatoryki algorytmicznej

Wojciech Rytter *

Skrypt ten zawiera szereg krótkich esejów opisujących proste, ale ciekawe algorytmy wielomianowe dla problemów związanych z generacją, zliczaniem lub obliczaniem elementarnych obiektów kombinatorycznych, których ilość jest z reguły wykładnicza. Inaczej mówiąc przedstawimy szereg małych zwycięskich potyczek z eksplozją kombinatoryczną.

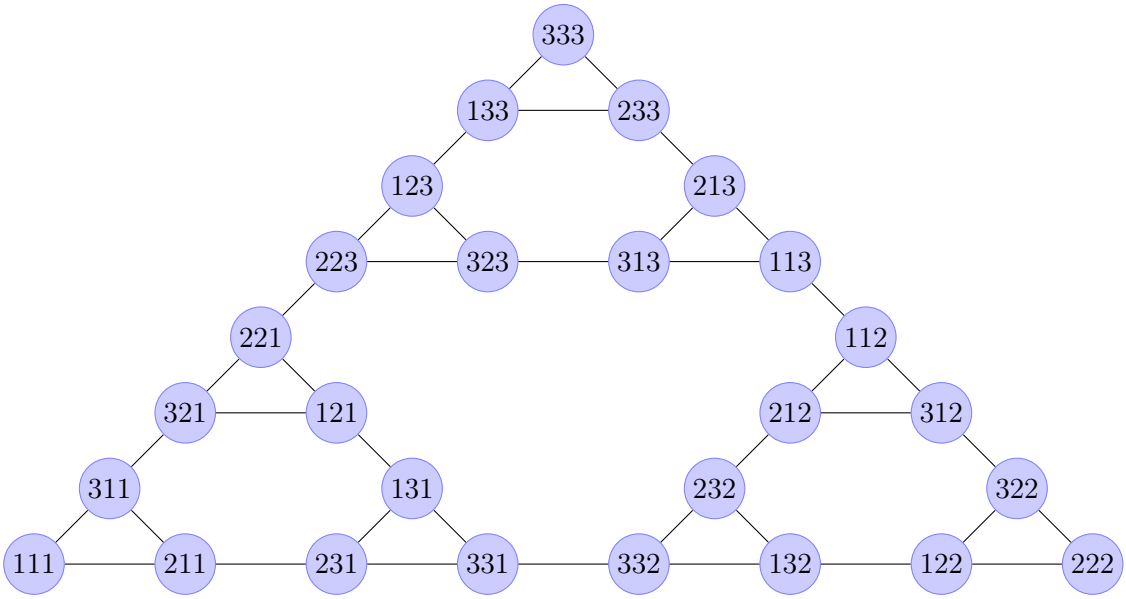
1 Grafy związane z wieżami Hanoi

Mamy n krążków (każdy o innym rozmiarze) na trzech wieżach. Początkowo wszystkie leżą na jednej wieży, w kolejności od największego u dołu do największego u góry. Trzeba je przełożyć na jakąś inną wieżę, ale nie można stawiać większego krążka na mniejszym.

Ustalmy przykładowo, że $n = 3$. Konfiguracja to trójka (a_1, a_2, a_3), oznaczająca położenie tych trzech krążków ($a_i \in \{1, 2, 3\}$ dla $i = 1 \ldots n$). Niech początkowa konfiguracja będzie $(1, 1, 1)$ (lub krócej 1^+) a końcowa $(3, 3, 3)$ (lub 3^+).

Budujemy graf $H_3 = (V, E)$, gdzie V – zbiór konfiguracji, krawędzie nieskierowane to dozwolone ruchy. Widać, że graf ma 3^n wierzchołków dla dowolnego n. Problem: znaleźć najkrótszą ścieżkę z konfiguracji początkowej do końcowej. Grafy H_n mają podobną strukturę jak trójkąty Sierpińskiego, powstające przez usunięcie z trójkąta Pascala elementów podzielnych przez 2.

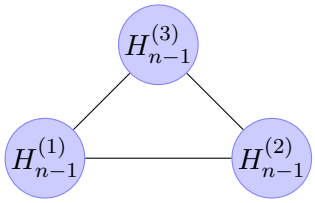
Początkowe iteracje tworzenia trójkąta Sierpińskiego.



Graf H_3 ma $3^3 = 27$ węzłów.

Graf H_n można definiować rekurencyjnie. Niech $H_m^{(j)}$ (dla dowolnego $m \geqslant 2$) będzie grafem H_m w którym każdy węzeł $(i_1, i_2 \ldots i_m)$ zamienimy na $((i_1, i_2 \ldots i_m,\ j)$. Wtedy H_n można zapisac rekurencyjnie jako:

*Przy znacznej pomocy technicznej Bartosza Szredera



1.1 Najkrótsze ścieżki i cykle Hamiltona w grafach Hanoi

Typy ruchów:

- α – przekładamy najmniejszy krążek na następny drążek (zgodnie z ruchem wskazówek zegara)
- β – przekładamy najmniejszy krążek na poprzedni drążek (przeciwnie do ruchu wskazówek zegara)
- γ – inny ruch (jest wyznaczony jednoznacznie)

Najkrótsza ścieżka

- n parzyste: $\alpha\gamma\alpha\gamma\alpha\gamma\ldots$
- n nieparzyste: $\beta\gamma\beta\gamma\beta\gamma\ldots$

Jak zapiszemy ciąg przełożeń kolejnych krążków, to wyjdą pozycje najmniej znaczącego zapalonego bitu w reprezentacjach binarnych kolejnych liczb od 1 do $2^n - 1$.

Ścieżka Hamiltona: $\beta^2\gamma\alpha^2\gamma\beta^2\gamma\alpha^2\gamma\ldots$

Cykl Hamiltona: ciąg ruchów: $(\delta\gamma)^3$, gdzie $\delta = (\beta^2\gamma\alpha^2\gamma)^k$, gdzie k tak dobrane, żeby długość tej ścieżki była 3^{n-1} (licząc odwiedzane wierzchołki). Budowanie cyklu Hamiltona należy zacząć od odpowiedniego wierzchołka — nie od narożnika, tylko od jednego z dwóch węzłów leżących na krawędzi łączącej dwa podgrafy H_{n-1}.

Uwaga! Wartość k w powyższym wykładniku nie musi być całkowita, np. dla przykładowego H_3 mamy z grubsza $k \approx \frac{3}{2}$, co przekłada się na $\delta = \beta^2\gamma\alpha^2\gamma\beta^2$.

Długa ścieżka Dla każdego $2^n - 1 \leqslant M \leqslant 3^n - 1$ istnieje **prosta** ścieżka z konfiguracji 1^+ do 3^+ mająca dokładnie M konfiguracji (taka, że żadna konfiguracja się nie powtarza).

Najkrótsza ścieżka z dowolnej konfiguracji Długość najkrótszej ścieżki z konfiguracji $(a_1, a_2, \ldots, a_n)$ do 3^+ wynosi

$$\sum_{a_i \neq 3} 2^{i-1}$$

1.2 Algorytm wyznaczania następnego ruchu

Dla n-tego ruchu możemy znaleźć wieżę początkową i docelową, odpowiednio z następujących wzorów:

$$\begin{aligned} &\text{początkowa: } (n\&(n-1)) \mod 3 \\ &\text{docelowa: } ((n|(n-1))+1) \mod 3 \end{aligned}$$

Gdzie `&` i `|` to bitowe operatory `AND` i `OR` i przy założeniu, że wszystkie krążki zaczynają na wieży o numerze 0 i docelowo trafiają na wieżę o numerze 1 albo 2 w zależności od tego, czy liczba krążków jest parzysta czy nie.

1.3 Algorytm wyznaczania konfiguracji

Jeśli chcemy się dowiedzieć jaki jest układ m krążków po n-tym ruchu, możemy zastosować następujący algorytm. Zapisujemy wiersze o długościach kolejno 1, 2, ... m w taki sposób, że wiersz o długości k ma konstrukcję $2\underbrace{1\ldots1}_{k-1}$, wiersz o długości $k-1$: $1\underbrace{2\ldots2}_{k-2}$ itd., czyli kolejne wiersze na zmianę:

- zaczynają się dwójką i są dopełniane jedynkami,
- zaczynają się jedynką i są dopełniane dwójkami.

Ponadto najdłuższy wiersz zaczyna się od dwójki i wiersze są wyrównane do prawej strony. Następnie zapisujemy liczbę n w postaci binarnej i usuwamy te wiersze, które odpowiadają bitowi zgaszonemu. Na końcu sumujemy wartości (mod 3) pozostałych wierszy w kolejnych kolumnach $1 \ldots m$ i otrzymujemy układ krążków na wieżach ponumerowanych kolejno 0, 1, 2.

Przykład $n = 23 = 10111_2$

				2	1
			1	2	1
		2	1	1	1
	~~1~~	~~2~~	~~2~~	~~2~~	0
2	1	1	1	1	1
2	1	3	3	6	mod 3
2	1	0	0	0	układ

Drugi algorytm wyznaczania konfiguracji

Założenie: przenosimy wszystkie krążki z wieży 0 na wieżę 2 w sposób optymalny. Następujący algorytm oblicza konfigurację po m ruchach:

wejście: $bit[1..n]$ – binarna reprezentacja m (najbardziej znaczący bit: $bit[n]$)
wyjście: $a[1..n]$ – konfiguracja n wież, $\forall_i\ a[i] \in \{0, 1, 2\}$

```
1  a[n] := 2bit[n];
2  x := a[n] − 1;
3  for i := n − 1 downto 1 do
4  |  if bit[i + 1] = bit[i] then
5  |  |  x := −x;
6  |  |  a[i] := a[i + 1];
7  |  else
8  |  |  a[i] := (a[i + 1] − x) mod 3;
9  |  end
10 end
```

Związki teorioliczbowe 3 wież Hanoi z trójkątem Pascala

W sekcji tej przedstawimy kilka ciekawych faktów i obliczeń teorioliczbowych związanych z trójkątem Pascala. Zajmiemy się liczeniem $\binom{n}{k}$ modulo liczba pierwsza p. Jeśli $p = 2$ to mamy bezpośredni związek z grafem H_k z poprzedniej sekcji.

Jeśli $\binom{n}{k} \equiv 1 \pmod 2$, to wierzchołek z trójkąta Pascala zostaje, w p.p. usuwamy go.
Niech H'_k będzie grafem który otrzymamy z trójkąta Pascala po usunięciu węzłów takich, że

$$\binom{i}{k} \pmod 2 \ = \ 0 \ \vee \ i > n.$$

Węzeł $\binom{i}{k}$ jest połączony nieskierowanymi krawędziami z istniejącymi węzłami $\binom{i'}{k'}$ takimi, że $|i-i'|+|k-k'| = 1$, podobnie jak w grafie H_n.

Fakt. Grafy $H_n,\ H'_n$ są izomorficzne.

Mamy proste kryterium stwierdzające, które pary $(i,k),\ i,k \leq n$ odpowiadają węzłom grafu H'_n.

Fakt.
Niech $W(m)$ będzie zbiorem pozycji zawierających jedynkę w zapisie binarnym liczby m. Zachodzi

$$\binom{n}{k} \pmod 2 \ = \ 1 \ \Leftrightarrow \ W(k) \subseteq W(n)$$

Przykład. Dla $n = 6 = [110],\ k = 4 = [010]$ mamy $W(n) = \{1,2\},\ W(k) = \{2\}$, zatem $W(k) \subseteq W(n)$ and $\binom{n}{k} \pmod 2 \ = \ 1$.

Powyższy fakt wynika prosto z następującego trudniejszego twierdzenia odkrytego przez Lucasa.

Twierdzenie 1. *Niech p – liczba pierwsza, oraz niech:*

$$r = (r_k,\, r_{k-1}, \ldots,\, r_0),\ c = (c_k,\, c_{k-1}, \ldots,\, c_0)$$

będą reprezentacjami liczb $c \leq r$ w systemie liczbowym o podstawie p. Zachodzi

$$\binom{r}{c} \equiv \prod_{i=0}^{r} \binom{r_i}{c_i} \pmod p.$$

Uzasadnienie twierdzenia.
Skorzystamy z następującej równości:

$$(1+x)^{p^m} \ = \ 1 + x^{p^m} \pmod p. \tag{1}$$

Uzasadninie pozostawiamy czytelnikowi.

Wykorzystamy też trochę manipulacji na wielomianach, przyrównując współczynnik przy c-tej potędze zmiennej x w pewnym wielomianie $W(x)$, obliczenia są modulo p:

$$W(x) \ = \ (1+x)^r \equiv \sum_{t=0}^{k} \binom{r}{t}$$

On the other hand, due to Równanie 1 mamy:

$$W(x) \ \equiv \ \prod_{i=0}^{k} [(1+x)^{p^i})]^{r_i} \ \equiv \ \prod_{i=0}^{k} (\sum_{j=0}^{r_i} \binom{r_i}{j} x^{j \cdot p^i} \ )$$

Współczynnik przy x^c w ostanim iloczynie wynosi

$$\prod_{i=0}^{r} (\binom{r_i}{c_i} \pmod p \ )$$

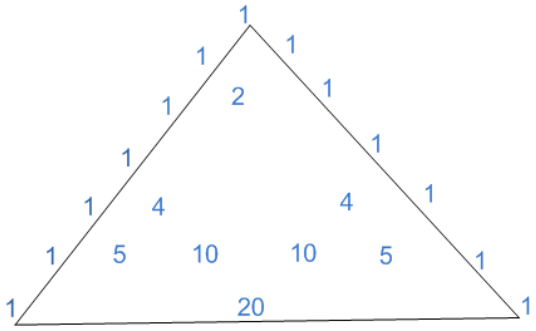
Jednocześnie ten wsółczynnik jest równy $\binom{r}{t}$. Stąd wynika praawdziwość tezy.

2 Więcej niż 3 wieże

Mamy n krążków i $m \geqslant 4$ wieże. Problem staję sie teraz dużo bardziej skomplikowany, nie jest znany żaden efektywny algorytm na optymalny cią ruchów.

Opiszemy pewną klasę algorytmów. Algorytm Frame-Stewart'a:

1. Rekurencyjnie przenieś stos $n - \zeta_n$ najmniejszych krążków z początkowej wieży do tymczasowej wieży T, używając wszystkich m wież.



Rysunek 1: Trójkąt Pascala z wyciętymi elementami podzielnymi przez 3.

2. Przenieś pozostały stos ζ_n największych krążków z początkowej wieży na docelową wieżę, używając $m-1$ wież (wszystkich poza wieżą T).

3. Rekurencyjnie przenieś stos $n - \zeta_n$ najmniejszych krążków z wieży T na wieżę docelową, używając wszystkich m wież.

Algorytm nie jest całkowicie wyspecyfikowany ponieważ nie podaje *explicite* wartości ζ_n rekurencyjnego podziału. Trzeba tak te wartości wybrac, żeby było optymalne w tej klasie. Dla 4 wież podamy dokładnie jak wyliczyć szybko ζ_n.

Niech wartość $FS(n, m)$ oznacza minimalną liczbę ruchów potrzebną do przeniesienia wszystkich n krążków z wieży początkowej do wieży końcowej, mając do dyspozycji m wież. Korzystając z powyższego algorytmu otrzymujemy wzór:

$$FS(n, m) = \begin{cases} 2^n - 1 & \text{dla } m = 3 \\ \min\limits_{1 \leqslant p < n} \{2FS(n-p, m) + FS(p, m-1)\} & \text{dla } m \geqslant 4 \end{cases}$$

Problem z tym algorytmem jest taki, że nie ma żadnego dowodu, że działa (ale wygląda, jakby działać miał). Eksperymentalnie sprawdzono jego poprawność dla $m = 4$ i $n \leqslant 30$.

W algorytmie Frame-Stewart obliczane jest najlepsze p, dla którego opłaca się wykonać operację przeniesienia najmniejszych $n-p$ krążków na tymczasowy stos. Obliczanie jest wykonywane niejako naiwnie, poprzez szukanie minimum po kolejnych $p = 1 \ldots n-1$. W przypadku 4 wież Hanoi odpowiednie p można znaleźć bezpośrednio.

Obserwacja 1. *Dla $n = 3 \ldots 5$ (czyli dla trzech kolejnych) mamy $p = 2$, dla $n = 6 \ldots 9$ (czyli dla czterech kolejnych) mamy $p = 3$ itd. Wartości n dla pierwszych „wystąpień" danej wartości p to odpowiednio* 3, 6, 10, 15 … *czyli $\Delta_k = \binom{k+1}{2} = \frac{k(k+1)}{2}$ dla kolejnych wartości k.*

Twierdzenie 2. *Dla $n = \Delta_k = \binom{k+1}{2}$ zachodzi $FS(n, 4) = (k-1)2^k + 1$.*

Dowód. Dowód indukcyjny. Łatwo sprawdzić, że zachodzi $FS(\Delta_1, 4) = 1$ i $FS(\Delta_2, 4) = 5$, zatem na początku *jest dobrze.* Weźmy teraz jakieś $FS(\Delta_i, 4)$ dla $i > 2$. Z wcześniejszej obserwacji wynika, że dla liczby krążków $n = \Delta_i, \Delta_i + 1, \ldots, \Delta_i + i = \Delta_{i+1} - 1$ zachodzi $p = i$. Zatem

$$\begin{aligned} FS(\Delta_i, 4) &= 2 \cdot FS(\Delta_i - i, 4) + FS(i, 3) \\ &= 2 \cdot FS(\Delta_{i-1}, 4) + 2^i - 1 \\ &= 2 \cdot \left((i-2)2^{i-1} + 1\right) + 2^i - 1 \\ &= (i-2)2^i + 2^i + 1 \\ &= (i-1)2^i + 1 \end{aligned}$$

□

Wniosek 1. *Dla problemu 4 wież Hanoi z liczbą krążków n, górne ograniczenie na liczbę ruchów wynosi $2^{c \cdot \sqrt{n}}$.*

Przykład. Dla każdego n istnieje konfiguracja 3 wież, w której wystarczy $2^{n-2}+1$ ruchów do uaktywnienia wszystkich krążków. Konfiguracja ta to $n-2$ najmniejsze krążki na pierwszej wieży, pozostałe dwa na drugiej wieży. Przekładamy krążek $n-1$ na trzecią wieżę, potem $n-2$ najmniejszych na trzecią wieżę, a następnie krążek n na pierwszą wieżę.

2.1 Dolne ograniczenie na liczbę ruchów (z pracy Mario Szegedy)

Mario Szegedy udowodnił, że dla 4 wież minimalna liczba ruchów x_n spełnia:

$$2^{c\cdot\sqrt{n}} \leqslant x_n \leqslant 2^{c'\cdot\sqrt{n}}$$

dla pewnych stałych c, c'.

Dowód opiera się na założeniu, że pomiędzy konfiguracją początkową a końcową każdy krążek przemieszcza się przynajmniej raz. Pierwsze przemieszczenie nazywamy *aktywacją* krążka. Dzięki takiemu podejściu można przeprowadzić dowód indukcyjny po liczbie wież. Wszędzie dalej zakładamy, że liczba krążków $n \geqslant 2$.

Dla trzech wież Weźmy dowolną początkową konfigurację n krążków i sekwencję ruchów $\mathcal{S}$, po których każdy krążek jest aktywny (co nie znaczy, że każdy krążek ruszył się tylko jeden raz). Powiedzmy, że największy krążek jest aktywowany w ruchu i-tym. W takim razie zarówno w ruchu $(i-1)$-szym oraz $(i+1)$-szym wszystkie pozostałe krążki znajdują się na jednej wieży. Podzielmy sekwencję ruchów $\mathcal{S}$ na trzy części $\mathcal{S} = \mathcal{S}_1\mathcal{S}_i\mathcal{S}_2$, gdzie $\mathcal{S}_i$ oznacza ruch i-ty, $\mathcal{S}_1$ oznacza prefiks $\mathcal{S}$, składający się z wszystkich ruchów wykonanych przed ruchem i-tym oraz $\mathcal{S}_2$ oznacza sufiks $\mathcal{S}$, składający się z wszystkich ruchów wykonanych po ruchu i-tym.

W zależności od konfiguracji początkowej, przedostatni co do wielkości krążek nie musi zostać aktywowany przed ruchem i-tym – jego pierwszy ruch może występować zarówno w $\mathcal{S}_1$ jak i $\mathcal{S}_2$. Bez straty ogólności przyjmijmy, że przedostatni co do wielkości krążek aktywowany jest gdzieś w $\mathcal{S}_2$. Oznacza to, że sekwencja ruchów $\mathcal{S}_2$ zawiera rozwiązanie dla problemu 3 wież Hanoi dla $n-2$ najmniejszych krążków, czyli $|\mathcal{S}_2| \geqslant 2^{n-2}$ ($2^{n-2}-1$ ruchów to standardowe rozwiązanie problemu wież Hanoi dla $n-2$ krążków, a $+1$ ponieważ jeszcze ruszyliśmy przedostatni co do wielkości krążek).

Z tego wynika, że $|\mathcal{S}| \geqslant 2^{n-2}+1$ ponieważ $\mathcal{S}$ zawiera przynajmniej jeden ruch więcej, niż $\mathcal{S}_2$ – jest to ruch i-ty. $\mathcal{S}_1$ może być puste.

Przykład. Dla każdego n istnieje konfiguracja trzecj wież, w której wystarczy $2^{n-2}+1$ ruchów do uaktywnienia wszystkich krążków. Konfiguracja ta to $n-2$ najmniejsze krążki na pierwszej wieży, pozostałe dwa na drugiej wieży. Przekładamy krążek $n-1$ na trzecią wieżę, potem $n-2$ najmniejszych na trzecią wieżę, a następnie krązek n na pierwszą wieżę.

Dla czterech wież Niech $H'_k(n)$ będzie minimalną liczbą ruchów potrzebnych do aktywacji n krążków na k wieżach, minimum bierzemy po wszystkich konfiguracjach. Wiemy że dla 3 wież zachodzi:

$$H'_3(n) = 2^{n-2}+1$$

Fakt zachodzenia ($\forall_{x\in A,\, y\in B}\ x < y$) zapisujemy jako $A < B$.

Stwierdzenie 1. *Jeśli mamy na 4 wieżach zbiór M_1 składający się z m_1 krążków i zbiór M_2 składający się z m_2 krążków, wszystkie krążki z M_2 na tej samej wieży oraz $M_1 < M_2$, to aktywacja wszystkich z M_2 wymaga* $\min\{H'_4(m_1),\, 2^{m_2-2}\}$ *ruchów.*

Dowód. Jeśli w czasie aktywacji M_2 wszystkie krążki z M_1 stają się aktywne to musimy wykonać co najmniej $H'_4(m_1)$ ruchów, w przeciwnym przypadku jeden z krążków z M_1 blokuje ciągle tę samą wieżę dla M_2 i do aktywacji M_2 musimy wykonać co najmniej $H'_3(M-2) \leqslant 2^{m_2-2}$ ruchów. □

Zacznijmy od oczywistego faktu rachunkowego.

Stwierdzenie 2. *Jeśli funkcja f spełnia*

$$\forall_{k\geqslant 1}\ f(k) \geqslant \min\{2\cdot f(k-1),\, \Delta\},\ f(1) = c' > 0$$

to zachodzi

$$f(k) \geqslant \min\{2^k\cdot c',\, \Delta\}$$

Twierdzenie 3. *$H'_4(n) \geqslant 2^{c\cdot\sqrt{n}}$ dla pewnej stałej $c > 0$*

Dowód. Ustalmy n i niech $\alpha = 8 \cdot \sqrt{n}$. Niech

$$f(k) = H'_4(k \cdot \alpha),\ \Delta = 2^{\sqrt{n}-2}$$

Udowodnimy, że f, Δ spełniają założenia stwierdzenia 2.

Na jednej z wież znajduje się zbiór Y co najmniej $\frac{\alpha}{4} = 2 \cdot \sqrt{n}$ spośród α największych krążków. Poza tym mamy zbiór M_1 składający się z $(k-1)\alpha$ najmniejszych krążków. Niech

$$M'_2 \cup M''_2 = Y$$

gdzie $M'_1 < M''_2$ będzie rozbiciem Y na dwie części. Musimy uaktywnić M'_1, a następnie M''_1. Za każdym razem zgodnie ze stwierdzeniem 1 wykonujemy co najmniej $\min\{f(k-1), 2^{m_2-2}\}$ ruchów. W sumie dwa razy tyle. Teraz teza wynika ze stwierdzenia 2. □

Oznaczmy w skrócie $FS(n, 4) = FS_4(n)$. Mamy z definicji

$$FS_4(n) = \min_{j\geqslant 1} 2 \cdot FS_4(n-j) + 2^j - 1.$$

Przyjmijmy $FS_4(0) = 0$ oraz

$$przyrost(n) = FS_4(n) - FS_4(n-1).$$

W poniższej tabelce kolejne liczby trójkątne są pogrubione:

n :	**1**	2	**3**	4	5	**6**	7	8	9	**10**	11	12	13	14	**15**	
$FS_4(n)$:	1	3	5	9	13	17	25	33	41	49	65	81	97	113	145	
$przyrost(n)$:	1	2	2	4	4	4	8	8	8	8	16	16	16	16	16	32

Ciąg $przyrost(n)$ jest bardzo regularny – mamy jedną jedynkę, dwie dwójki, trzy czwórki, cztery ósemki, pięć szsnastek itd. Inaczej mówiąc, kolejny blok potęg dwójki to $k+1$ potęg 2^k, dla $k = 0, 1, 2, 3, \ldots$.

Fakt. Załóżmy, że $0 \leqslant r \leqslant k$, wtedy:

(a) $FS_4(\Delta_k + r) = (r + k - 1)2^k + 1$

(b) $FS_4(\Delta_k + r) = 2 \cdot FS_4(\Delta_k + r - k) + 2^k - 1$

(b) Optymalną waartościa ζ_n jest liczba k.

Dygresja. Dla 5 wież ciąg przyrostów liczb $FS_5(n)$ wygląda początkowo następująco.

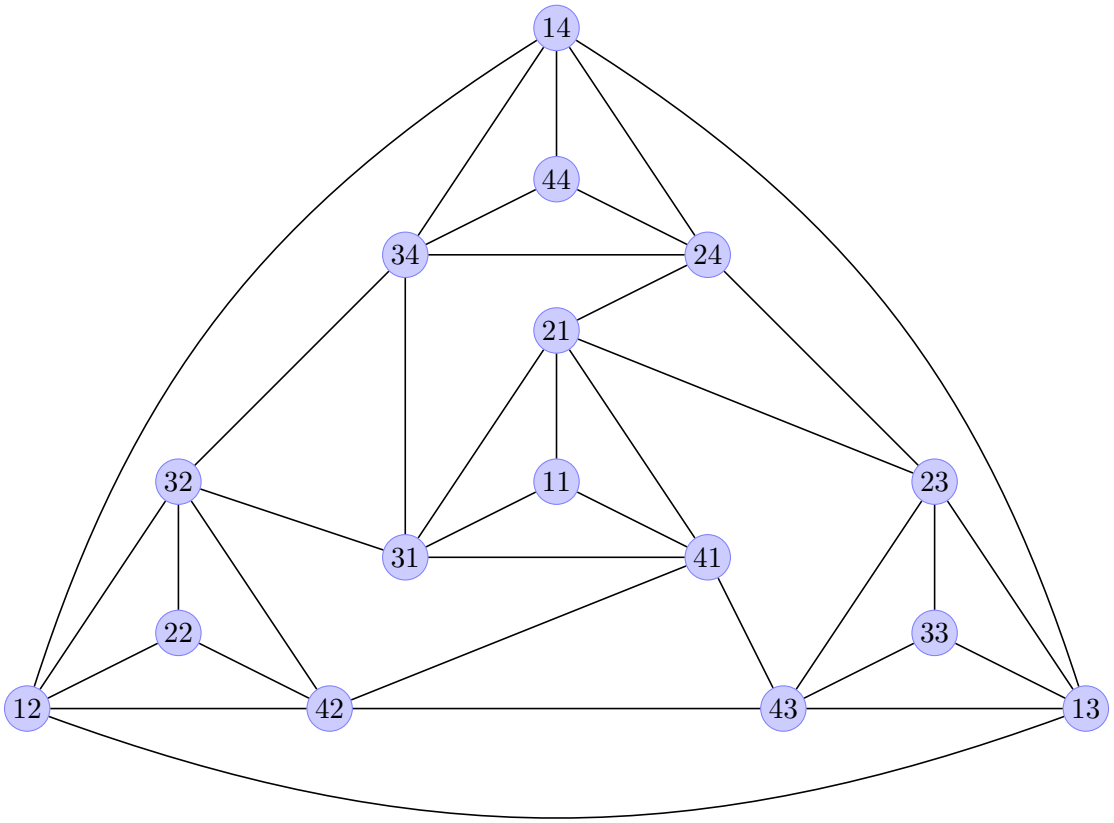
n :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$przyrost(n)$:	1	2	2	2	4	4	4	4	4	4	8	8	8	8	8

Mamy Δ_1 razy 2^0, Δ_2 razy 2^1, Δ_3 razy 2^2, Δ_4 razy 2^3 itd. Inaczej mówiąc, kolejny blok potęg dwójki to Δ_{k+1} potęg 2^k, dla $k = 0, 1, 2, 3, \ldots$.
Tak więc dla dowolnej ustaloneh liczby wież wyliczenie optymalnego parametru ζ_n dla podziału rekurencyjnego w algorytmie klasy FS jest kwestią dosyć żmudnych rachunków, ale jest komputerowo szybko obliczalne.

Hipoteza Frame'a-Stewarta. Algorytm typu FS z optymalnym doborem parametru ζ_n daje minimalną liczbe ruchów w problemie m wież Hanoi dla ustalonego $m \geq 4$.

2.2 Cykle Hamiltona w grafie 4 wież

Grafy dla 4 wież są skomplikowane i mają dużą liczbę cykli Hamiltona.



Graf dla 2 krążków i 4 wież, graf ten ma $6 \cdot 3^4 = 486$ cykli Hamiltona.

W każdym małym $\mathbf{K}_4$ (klika 4-wierzchołkowa jako podgraf indukowany) do cyklu Hamiltona możemy wybrać 3 lub 2 krawędzie. Zacznijmy od przypadku, gdy w każdym $\mathbf{K}_4$ bierzemy po 2 krawędzie (przykład na rysunku 2). Jest $a = 6$ takich cykli (przykładowy graf możemy odbić symetrycznie lewo-prawo, dodatkowo na 3 sposoby możemy wybrać, które krawędzie wybierzemy z górnej $\mathbf{K}_4$). Teraz popatrzmy na dwie krawędzie wychodzące z wierzchołka małego $\mathbf{K}_4$ (np. $(42) - (41) - (43)$). Możemy je ściągnąć do krawędzi $(42) - (43)$, natomiast opuszczony wierzchołek możemy odwiedzić wybierając 3 krawędzie w środkowym $\mathbf{K}_4$ (możemy to zrobić na $b = 2$ sposoby). W grafie możemy ściągnąć $i = 0 \ldots 4$ krawędzi, możemy je wybrać na $\binom{4}{i}$ sposobów. Zatem liczba cykli to

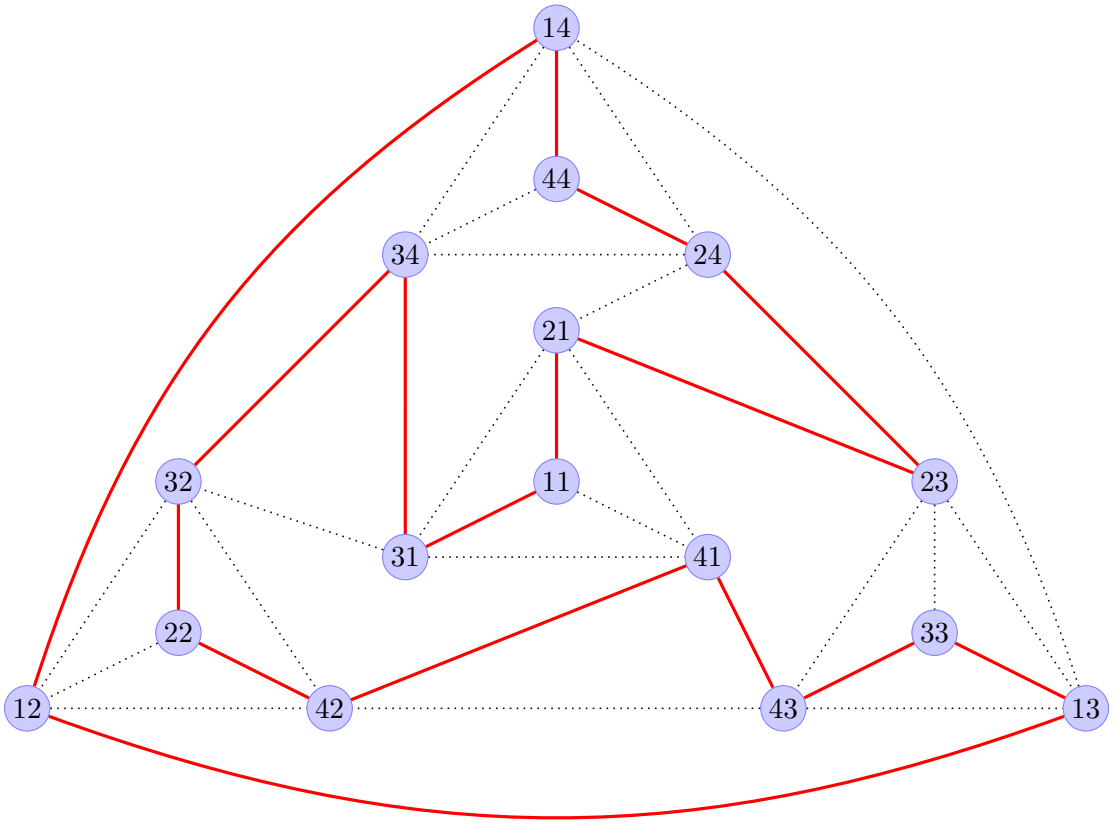
$$\sum_{i=0}^{4} a \cdot \binom{4}{i} \cdot b^i = a \cdot (1+b)^4 = 6 \cdot 3^4 = 486$$

Pozostaje pokazać, że cykli nie ma więcej. Weźmy dowolny cykl (np. ten z rysunku 3) i rozważmy $\mathbf{K}_4$, w którym wybrano 3 krawędzie (np. środkowe). Rozważmy wierzchołek (21), który nie łączy $\mathbf{K}_4$ z resztą grafu (zatem krawędzie $(21) - (24)$ i $(21) - (23)$ nie są wybrane). Teraz pokażemy, że krawędź $(24) - (23)$ musi być wybrana do cyklu; z tego wynika, że możemy dokonać na niej operacji odwrotnej do ściągnięcia, zatem każdy cykl powstaje za pomocą operacji ściągnięcia krawędzi.

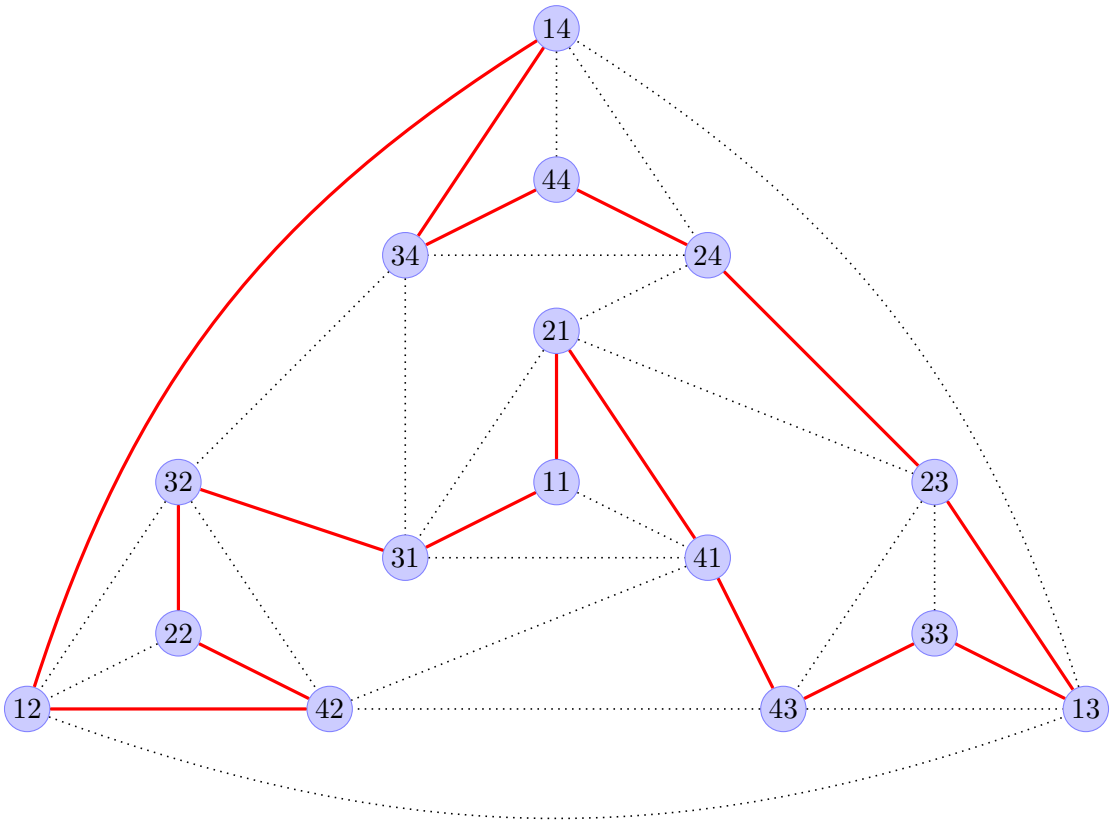
Dla małego $\mathbf{K}_4$ jeden z jego nieśrodkowych wierzchołków nazwiemy typu A, jeśli prowadzi do niego krawędź cyklu, która łączy $\mathbf{K}_4$ z resztą grafu. Jeśli $(24) - (23)$ nie była wybrana, to wierzchołki (24) i (23) muszą mieć po 2 wybrane krawędzie w swoich $\mathbf{K}_4$, zatem wierzchołki (34), (14), (43) i (13) są typu A. Ponieważ (41) jest typu A, to albo $(41) - (43)$ musi być wybrana, albo $(41) - (42) - (43)$ muszą być wybrane. W obu przypadkach (42) nie może być wierzchołkiem typu A. Analogicznie (32) nie może być typu A. Zatem lewy $\mathbf{K}_4$ ma co najwyżej jeden wierzchołek typu A, a to jest niemożliwe.

3 Generowanie obiektów kombinatorycznych

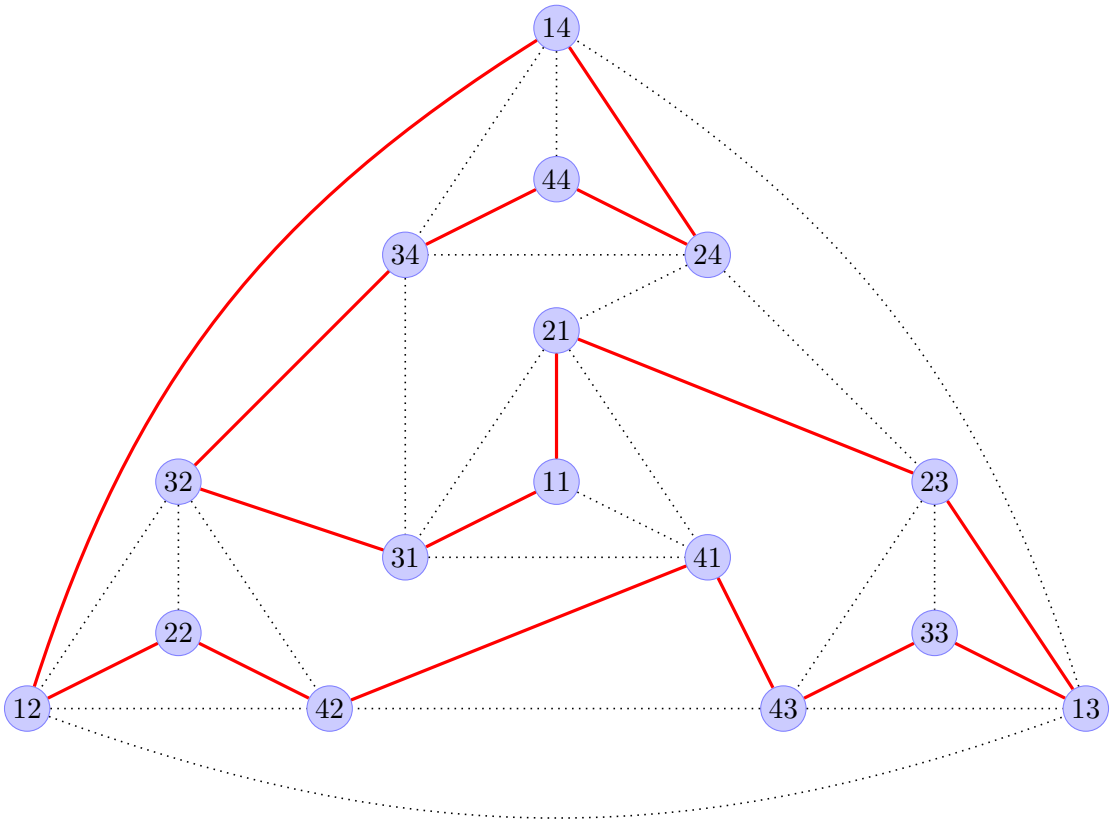
W rozdziale tym pokażemy jak generować ciągi prostych obiektów kombinatorycznych w taki sposób, aby kolejne dwa obiekty różniły się niewiele. Inaczej mówiąc szukamy teracyjnej metody generowania ścieżki/cyklu



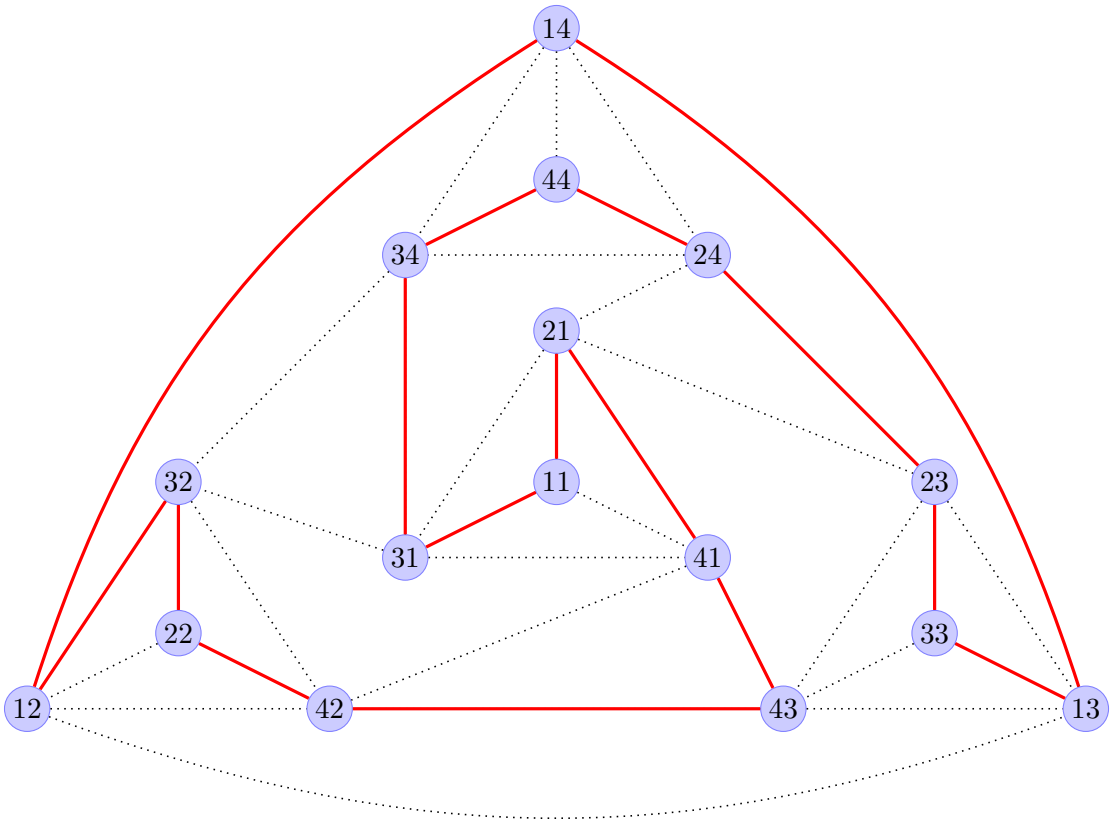
Rysunek 2: Przykład cyklu Hamiltona pierwszego typu.



Rysunek 3: Cykl Hamiltona drugiego typu, który wchodząc do każdego $\mathbf{K}_4$ przechodzi go w całości.



Rysunek 4: Przykład innego cyklu Hamiltona.



Rysunek 5: Jeszcze inny cykl Hamiltona.

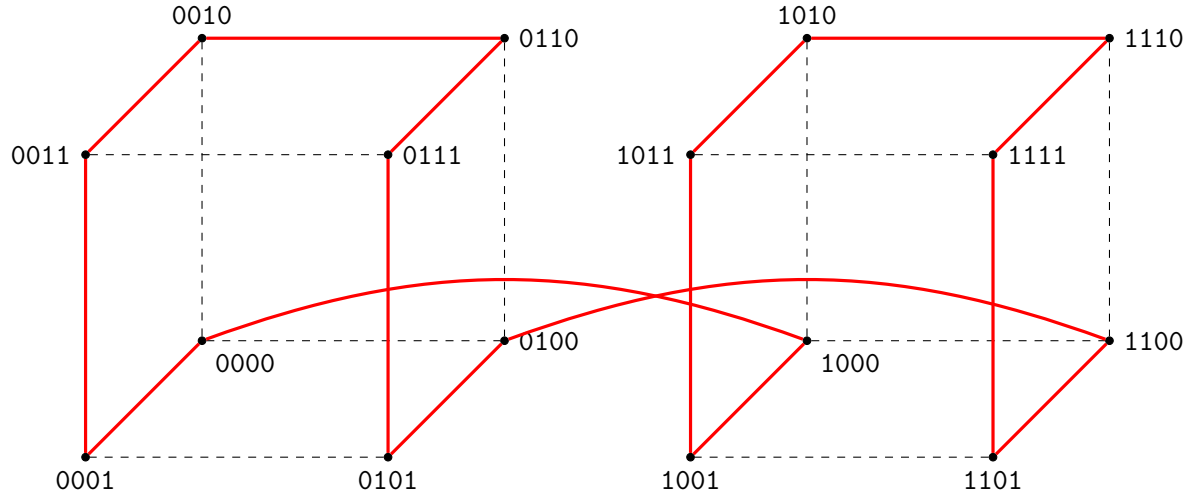
Hamiltona w grafie, którego krawędzie odpowiadają *bliskości* obiektów, a obiektami są kombinacje, permutacje itp. Charakterystyczne jest to, że nasze grafy są z reguły wykładniczej wielkości, a funkcja generacji kolejnego obiektu ma złożoność relatywnie małą (stałą lub liniową)

W szczególności niech $\mathcal{K}(n,\, k)$ oznacza rodzinę podzbiorów k-elementowych zbioru n-elementowego. Jeśli chcemy je generować np. leksykograficznie, tzn. jeśli zapiszemy wygenerowaną kombinację jako ciąg binarny, gdzie $b_i = 0$ oznacza nienależenie elementu i-tego do kombinacji, a $b_i = 1$ jego należenie, to generujemy kolejno wszystkie ciągi binarne n-elementowe o dokładnie k zapalonych jedynkach.

Najmniejsza odległość Hamminga pomiędzy tak wygenerowanymi kombinacjami wynosi 2. Jeśli stworzymy graf taki, że każdy węzeł jest tożsamy z jedną kombinacją, a krawędzie przebiegają między wierzchołkami, między którymi dla ich ciągów odległość Hamminga wynosi 2, to w takim grafie ścieżka Hamiltona wygeneruje wszystkie kombinacje.

3.1 Ciągi Graya – ścieżki Hamiltona w kostce n-wymiarowej

Ciąg Graya rzędu n oznaczamy przez $G(n)$ – lista wszystkich ciągów binarnych długości n, każdy ciąg występuje dokładnie raz, odległość Hamminga między kolejnymi obiektami wynosi 1 (minimalna).



Rysunek 6: Cykl Hamiltona w hiperkostce 4-wymiarowej. Dla czytelności pominięto krawędzie łączące dwa sześciany (poza krawędziami należącymi do cyklu).

Algorytm rekurencyjny:

$$G(0) = \emptyset; \quad G(n) = 0\,G(n-1);\ 1\,G(n-1)^R$$

Ten algorytm rekurencyjnie znajduje ścieżki Hamiltona w hiperkostkach coraz mniejszych wymiarów.

Algorytm iteracyjny:

- co drugi krok (poczynając od pierwszego) zamieniamy ostatni bit,
- w pozostałych krokach zmieniamy bit przed ostatnią (na prawo) jedynką.

Algorytm za pomocą wzoru na k-ty element ciągu $G(n)$ (konwertując liczby na zapisy binarne):

$$g(k) = k \oplus \left\lfloor \frac{k}{2} \right\rfloor$$

3.2 Generacja kombinacji poprzez wymiany dwóch bitów

Kolejne ciągi w kodzie Graya z k jedynkami dają generację k-podzbiorów z minimalnymi zmianami.

$$G(n,\, k) = 0\,G(n-1,\, k);\ 1\,G(n-1,\, k-1)^R$$

Jak z tego zrobić algorytm, w którym jedna iteracja jest w pamięci $\mathcal{O}(n)$ i czasie $\mathcal{O}(1)$? Opiszemy zupełnie inny algorytm. Niech π będzie ciągiem zerojedynkowym reprezentującym k-kombinacje n elementów.

Wprowadzamy wektor aktywności $A[1..n]$: $A[i] = 1$ gdy pozycja i aktywna. Oznaczmy przez $last(A)$ ostatnią (najbardziej na prawo) pozycję aktywną. Jeśli takiej pozycji nie ma, to $last(A) = \emptyset$. Niech operacja `UaktywnijPo(k)` oznacza uaktywnienie wszystkich pozycji większych od k.

Algorytm 1: Kombinacje przez zamiany

```
1  π := jakakolwiek kombinacja n po k;
2  UaktywnijPo(0);
3  repeat
4  |  wypisz kombinację π;
5  |  k := last(A);
6  |  if k = ∅ then
7  |  |  STOP;
8  |  W ciągu π wymień π[k] z przeciwnym bitem na jakiejkolwiek pozycji na prawo;
9  |  A[k] := 0;
10 |  UaktywnijPo(k);
11 until forever;
```

Rozważmy inny algorytm, w którym wykonujemy jedną zamianę bitów, a między zamienianymi pozycjami są same zera. Chcemy mieć taką *sztywniejszą* wersję poprzedniego algorytmu, w której pozycja na prawo od k, z którą jest wymieniany bit jest jak najwężej wyspecyfikowana.

Niech $PierwJed(k)$ oznacza pozycję pierwszej jedynki w π na prawo od k; jeśli na prawo nie ma jedynki to $PierwJed(k) = n + 1$. Żądamy, aby algorytm spełniał ponadto następujący niezmiennik: na prawo od k jest co najwyżej jeden blok jedynek, algorytm zaczyna i kończy się w sytuacji z jednym blokiem jedynek.

Algorytm 2: Kombinacje przez ścisłe zamiany

```
1  π := [1^k 0^(n−k)];
2  UaktywnijPo(0);
3  repeat
4  |  wypisz kombinację π;
5  |  k := last(A);
6  |  if k = ∅ then
7  |  |  STOP;
8  |  if π[k] = 1 then
9  |  |  Wymień π[k] z bitem na pozycji PierwJed(k) − 1;
10 |  else
11 |  |  Wymień π[k] z bitem na pozycji PierwJed(k);
12 |  end
13 |  A[k] := 0;
14 |  UaktywnijPo(k);
15 until forever;
```

Ponieważ algorytm jest dość „sztywny" można go zaimplementować tak, aby jedna iteracja była w czasie $\mathcal{O}(1)$ (pamięć liniowa – wektor A).

3.3 Generacje prefiksowe

Oznaczmy przez $shift_k$ operację cyklicznego przesunięcia k-tego prefiksu, polega ona na przesunięciu elementu k-tego na początek ciągu.

Opiszemy kilka algorytmow generacji – permutacji, kombinacji, permutacji multizbiorów, ciągów reprezentujących drzewa binarne. W tych algorytmach istotne będzie jaki jest pierwszy obiekt ($start$), a czasami również jaki ostatni ($finish$).

Generacja permutacji

Oznaczmy permutacje przez π (tablica od 1 do n). W tym przypadku $start = [1, 2, \ldots, n]$.

Algorytm 3: Prefiksowe generowanie permutacji

```
1  π := [1, 2, ..., n];
2  k := n + 1;
3  repeat
4  |  if (k ⩽ n) ∧ (π[k] = k) then
5  |  |  k := k − 1;
6  |  else
7  |  |  Wypisz permutację π;
8  |  |  k := n;
9  |  end
10 |  if k = 1 then
11 |  |  STOP;
12 |  π := shift_k(π);
13 until forever;
```

Dla $n = 4$ algorytm wygeneruje kolejno:

1234	4123	3412	2341	3124	4312	2431	1243	2314	4231	1423	3142
2134	4213	3421	1342	3214	4321	1432	2143	1324	4132	2413	3241

Generacja kombinacji

Oznaczmy kombinację typu n, k również przez π (tablica od 1 do n). Jest to ciąg zerojedynkowy mający k jedynek i $n - k$ zer. W tym przypadku $start = 1^k 0^{n-k}$, $finish = 1^{k-1} 0^{n-k} 1$. Potrzebna nam **funkcja pierwszego skoku** w ciągu, oznaczmy ją przez

$$PozSkoku(\pi) = \min\{k : (\pi[k] > \pi[k-1]) \vee (k = |\pi| + 1)\}$$

gdzie $|\pi|$ oznacza długość ciągu. Na przykład $PozSkoku([1, 1, 0, 0, 1, 0, 0]) = 5$

Algorytm 4: Prefiksowe generowanie kombinacji

```
1  π := [1^k 0^(n−k)];
2  repeat
3  |  Wypisz kombinację π;
4  |  k := PozSkoku(π);
5  |  if k = n then
6  |  |  STOP;
7  |  j := min{k + 1, n};
8  |  π = shift_j(π);
9  until forever;
```

Dla $k = 3,\ n = 6$ algorytm wygeneruje:

111000	011100	101100	110100	011010	101010	010110	001110	100110	110010
011001	101001	010101	001101	100101	010011	001011	000111	100011	110001

Zauważmy, że w pierwszym wierszu mamy sufiks 0, w drugim sufiks 1. Jeśli obetniemy ostatnie zero to otrzymamy ciąg dla $n = 5$, $k = 3$; jeśli obetniemy ostatnie 1 to otrzymamy ciąg dla $n = 5$, $k = 2$, ale zaczynający się w generacji w drugiej kombinacji.

Inaczej mówiąc w obu przypadkach mamy rekurencję, z tym że w przypadku ciągów z sufiksem 1 mamy rekurencyjny ciąg, ale cyklicznie przesunięty. Zauważmy, że w jednej iteracji zmieniamy co najwyżej 4 bity. Można jedną iterację zaimplementować tak, by działała w czasie $\mathcal{O}(1)$ i pamięci $\mathcal{O}(1)$.

Niech $\overrightarrow{\mathcal{A}}$ oznacza ciąg, w którym, pierwszy element staje się ostatnim, tzn. przykładowo:

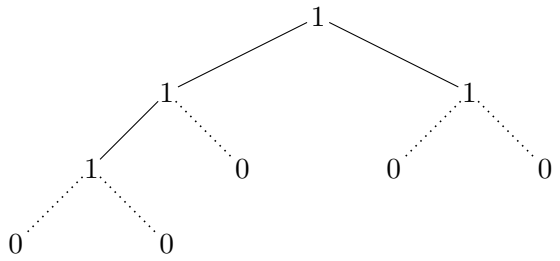
$\mathcal{A}$	$\overrightarrow{\mathcal{A}}$
0011	0101
0101	0110
0110	1001
1001	0011

Ten algorytm rekurencyjnie generuje ciąg binarny wszystkich kombinacji według schematu $\mathcal{R}_{t,\,s} = \mathcal{R}_{t,\,s-1}0; \overrightarrow{\mathcal{R}_{t-1,\,s}}1$. Podobne do generacji koleksykograficznej (leksykograficznej patrząc od końca ciągu)

$$colex_{k,\,t} = colex_{k,\,t-1}\,0;\ colex_{k-1,\,t}\,1$$

3.4 Generacja ciągów zrównoważonych, korzystając z $shift$

Tak naprawdę generacja pewnych kształtów drzew binarnych. Będzie to generacja analogiczna w pewnym sensie do kodów Graya – w drzewach binarnych będziemy zmieniać tylko stałą liczbę wskaźników generując kolejny element ciągu. Drzewa zwykle zapisuje się w postaci ciągu nawiasowego, tutaj zmieniamy (na 1 i) na 0. Sposób reprezentacji drzewa za pomocą takiego ciągu:



Przejście w porządku `preorder` daje ciąg 111000100 – zawsze jest jedno zero więcej. Zwykle będziemy z tego zapisu odcinać pierwszą cyfrę (jedynkę – korzeń) i ostatnią (zero – skrajnie prawy liść), nie tracąc żadnej informacji. Będziemy generować ciągi o takiej własności, że dowolny prefiks ciągu ma co najwyżej o jedno zero więcej niż jedynek.

Algorytm 5: Prefiksowe generowanie ciągów zrównoważonych

```
1  π := [01^{n-1}0^{n-1}];
2  repeat
3  |  Wypisz ciąg π;
4  |  k := PozSkoku(π);
5  |  if k = |π| + 1 then
6  |  |  STOP;
7  |  if shift_{k+1}(π) poprawny then
8  |  |  π := shift_{k+1}(π);
9  |  else
10 |  |  π = shift_k(π);
11 |  end
12 until forever;
```

Dla $n = 4$ algorytm wygeneruje:

0111000 1011000 1101000
0110100 1010100 0101100 1001100 1100100
0110010 1010010 0101010 1001010 0101010 1001010 1100010
1110000

W kolejnych wierszach (pomijając ostatni) mamy sufiksy 1000, 100, 10.

1	011 1000
2	101 1000
3	110 1000
4	0110 100
5	1010 100
6	0101 100
7	1001 100
8	1100 100
9	01100 10
10	10100 10
11	01010 10
12	10010 10
13	11000 10

Zauważmy, że w jednej iteracji zmieniamy $\mathcal{O}(1)$ bitów. Można jedną iterację zaimplementować tak, żeby działała w czasie $\mathcal{O}(1)$ i pamięci $\mathcal{O}(1)$.

Algorytm generuje kształty drzew binarnych, dla każdego drzewa binarnego dołączamy do każdego liścia dwóch synów — sztuczne dwa liście, oraz do każdego węzła z jednym synem dodatkowego sztucznego syna. Przechodzimy drzewo preorder i wypisujemy 1 gdy mamy oryginalny węzeł albo 0 gdy sztuczny. Pierwszą jedynkę i ostatnie zero obcinamy. W ten sposób mamy odpowiedniość między zrównoważonymi ciągami i drzewami binarnymi. Operacja przesunięcia prefiskowego zmienia w drzewie $\mathcal{O}(1)$ wskaźników typu $ojciec \leftrightarrow syn$.

3.5 Generacja anagramów – permutacje multizbiorów

Mamy alfabet składający się z m liter $\{1, 2, \ldots, m\}$. Przypuśćmy, że mamy f_i kopii litery i dla $1 \leqslant i \leqslant m$. Taki zbiór liter nazywamy multizbiorem M. Anagramem dla M jest dowolny ciąg (słowo) zawierające f_i razy literę i dla każdego i. Oznaczmy przz $Anag(M)$ zbiór wszystkich anagramów. Chcemy wygenerować wszystkie anagramy z $Anag(M)$ poprzez cyklicznie przesuwanie prefiksów. Niech $\max(M)$ oznacza leksykograficznie maksymalny anagram, tzn.

$$\max(M) = m^{f_m}(m-1)^{f_{m-1}} \ldots 2^{f_2} 1^{f_1}$$

Na przykład dla $M = \{1, 1, 2, 3, 3, 4, 4, 4\}$ mamy $\max(M) = 4\,4\,4\,3\,3\,2\,1\,1$. Niech n będzie długością anagramu.

Algorytm 6: Prefiksowe generowanie anagramów

```
1  π := shift_n(max(M));
2  repeat
3  |  Wypisz anagram π;
4  |  k := PozSkoku(π);
5  |  if k = |π| + 1 then
6  |  |  STOP;
7  |  if (k = n) ∨ (π[k − 1] < π[k + 1]) then
8  |  |  π := shift_k(π);
9  |  else
10 |  |  π = shift_{k+1}(π);
11 |  end
12 until forever;
```

Przykład: przypuśćmy, że $M = \{1, 1, 2, 2, 3\}$. Wtedy $\max(M) = 3\,2\,2\,1\,1$ i algorytm wygeneruje następujący ciąg anagramów:

13221	31221	23121	12321	21321	32121	13212	31212	13122	11322
31122	23112	12312	21312	12132	21132	32112	23211	22311	12231
21231	22131	12213	21213	21213	12123	11223	21123	22113	32211

4 Gwiazdowe generowanie permutacji

Rozważamy tylko transpozycje pewnego elementu z pierwszym, graf takich transpozycji jest *gwiazdą*. Załóżmy, że numerujemy pozycje permutacji π od zera. Interesuje nas generacja ciągu pozycji, które wymieniamy kolejno z elementem na pozycji 0. Niech E_n będzie ciągiem dla wygenerowania wszystkich permutacji n-elementowych. Chcemy, żeby E_n był prefiksem E_{n+1}, czyli otrzymujemy nieskończony ciąg E_∞.

Poniżej wypisujemy ciąg E_5 generujący wszystkie permutacje zbioru 5-elementowego. Pozycje permutacji numerujemy od zera. W i-tym kroku zamieniamy $\pi[E_5[i]]$ z $\pi[0]$.

1 2 1 2 1 3 2 1 2 1 2 3 1 2 1 2 1 3 2 1 2 1 2 4
3 1 3 1 3 2 1 3 1 3 1 2 3 1 3 1 3 2 1 3 1 3 1 4
1 2 1 2 1 3 2 1 2 1 2 3 1 2 1 2 1 3 2 1 2 1 2 4
3 1 3 1 3 2 1 3 1 3 1 2 3 1 3 1 3 2 1 3 1 3 1 4
1 2 1 2 1 3 2 1 2 1 2 3 1 2 1 2 1 3 2 1 2 1 2

Opiszemy jak gwiazdowo generować permutacje zbioru $\{1, 2, \ldots, n\}$, ciąg E wymienianych pozycji w trakcie algorytmu jest generowany. Oznaczmy

$$\rho_!(k) = \max\{j : j! \mid k\}$$

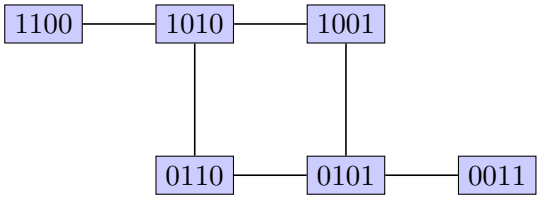
Na przykład $\rho_!(12) = 3$, $\rho_!(44) = 2$, $\rho_!(13) = 1$. Wykorzystamy tablicę (ciąg kontrolny) $B[1..n]$, początkowo będący identycznością.

Algorytm 7: Gwiazdowe generowanie permutacji

```
1  ∀_{i∈{1,2,...,n}} B[i] := 1;
2  π := [1, 2, ..., n];
3  Wypisz permutację π;
4  for i := 1 to n! − 1 do
5  |  k := ρ_!(i);
6  |  Zamień π[0] z π[B[k]];
7  |  Wypisz permutację π;
8  |  E_n[i] := B[k];
9  |  Odwróć kolejność elementów B[1..k − 1];
10 end
```

4.1 Generowanie podzbiorów k-elementowych przez sąsiednie wymiany

Chcemy wygenerować wszystkie zbiory z $\mathcal{K}(n, k)$ tak, że dla kolejnych dwóch zbiorów w ciągach je reprezentujących zamieniają się tylko sąsiednie bity. Np. dla $\mathcal{K}(4, 2)$:



Jak widać w takim grafie nie ma cyklu Hamiltona, bo istnieją węzły o stopniu 1. Możemy więc szukać ścieżki Hamiltona, ale okazuje się, że ona też istnieje tylko w niektórych grafach.

Lemat 1. *Graf $\mathcal{K}(n, k)$ jest dwudzielny.*

Dowód. Graf dwudzielny nie ma cykli o nieparzystej długości. Aby wychodząc z jakiegoś wierzchołka v można było do niego wrócić, należy wykonać parzystą liczbę zamian (czyli przejść po krawędziach), ponieważ każda zamiana zmienia o 2 odległość Hamminga. □

Twierdzenie 4. *Istnieje ścieżka Hamiltona w grafie $\mathcal{K}(n, k)$ wtedy i tylko wtedy, gdy n jest parzyste i k jest nieparzyste (oprócz specjalnego przypadku $k = 1 \vee k = n - 1$).*

Dowód. Poniższy dowód obejmuje konstrukcję ścieżki Hamiltona dla grafów z podanej klasy, ale nie obejmuje pokazania, że w grafach dla pozostałych wartości n i k cykl Hamiltona nie istnieje.

Graf $\mathcal{K}(n, k) = G(V, E)$ jest dwudzielny: $V = V_1 \cup V_2$. Dla takich grafów $|V_1| - |V_2| \in \{-1, 1\}$. Niech $\mathcal{K}(n, k) = A(n, k) \cup B(n, k) \cup C(n, k) \cup D(n, k)$, gdzie:

- $A(n, k)$ indukowany przez $11(0+1)^*$, $A(n, k) \equiv G(n-2, k-2)$
- $B(n, k)$ indukowany przez $00(0+1)^*$, $B(n, k) \equiv G(n-2, k)$
- $C(n, k)$ indukowany przez $10(0+1)^*$, $C(n, k) \equiv G(n-2, k-1)$
- $D(n, k)$ indukowany przez $01(0+1)^*$, $D(n, k) \equiv G(n-2, k-1)$

Ścieżka Hamiltona ma postać $1^+0^+ \xrightarrow{*} 0^+1^+$. Graf $\mathcal{K}(n, k)$ ma dwa wierzchołki o stopniu 1, które będą punktami początkowymi/końcowymi ścieżki Hamiltona.

Ponieważ podgrafy A i B są izomorficzne z mniejszymi grafami $\mathcal{K}$, to przez indukcję istnieje w nich ścieżka Hamiltona. Przejście z podgrafu A do B będzie przebiegać przez ścieżkę Hamiltona podgrafu $C \cup D$. Ścieżka Hamiltona przebiegająca przez A ma postać 111^*0^+ i kończy się na 110^+1^*, czyli zaczynamy z wszystkimi jedynkami po lewej stronie, a kończymy z wszystkimi poza dwoma ostatnimi po prawej. W ten sposób możemy przejść do podgrafu $C \cup D$. Analogicznie wychodzimy z $C \cup D$ do B (rysunek 7).

Grzebień to drzewo o maksymalnym stopniu wierzchołka co najwyżej 3 z dodatkową własnością – wszystkie wierzchołki stopnia 3 leżą na jednej ścieżce (*ścieżka główna*). Jeśli v jest wierzchołkiem na ścieżce głównej, to *zębem* grzebienia nazywamy najdłuższą ścieżkę, która przecina ścieżkę główną tylko raz i dokładnie w węźle v. Jeśli stopień $deg(v) < 3$, to ząb składa się z pojedynczego wierzchołka v (jest *trywialny*), w przeciwnym przypadku ma przynajmniej dwa wierzchołki.

Obserwacja 2. *Podgrafy $C(n, k)$ i $D(n, k)$ są rozpinane przez grzebienie.*

Ponieważ węzły z podgrafu C reprezentują ciągi $10w$, a węzły z podgrafu D reprezentują ciągu $01w$ dla odpowiednich $w \in (0+1)^*$, to możemy zbudować graf z grzebieni rozpinających C i D w taki sposób, że istnieje krawędź $10w \longleftrightarrow 01w$ dla odpowiednich w i nie naruszy to własności mówiącej o odległości Hamminga równej 2 pomiędzy ciągami binarnymi reprezentowanymi przez sąsiadujące wierzchołki.

Pozostaje zatem znalezienie ścieżki Hamiltona od 1010^+1^* do 0101^+0^* w grafie zbudowanym ze sklejonych grzebieni rozpinających podgrafy C i D. Wierzchołki $\{01, 10\}10^+1^*$ i $\{01, 10\}01^+0^*$ są punktami końcowymi grzebieni dla (odpowiednio) C i D. Algorytm generowania ścieżki Hamiltona zaczyna od pierwszego punktu końcowego grzebienia rozpinającego C i kończy na drugim punkcie końcowym grzebienia D. Przechodzenie odbywa się następująco: w danym węźle v będącym na ścieżce głównej grzebienia C zejdź w dół zęba, przejdź do odpowiadającego zęba grzebienia dla D, przejdź w górę zęba do ścieżki głównej D, przemieść się do kolejnego węzła na ścieżce głównej itd., aż do wylądowania na punkcie końcowym ścieżki głównej grzebienia dla D. Ponieważ mamy założenie, że zarówno n jak i k są parzyste w podgrafach C i D, a ścieżka główna grzebienia ma długość $1+(n-2)\cdot(k-1)$ [ćwiczenie: dlaczego?], czyli nieparzystą, to zawsze można taką ścieżkę wyznaczyć. □

5 Liczenie podziałów liczby: algorytm Eulera

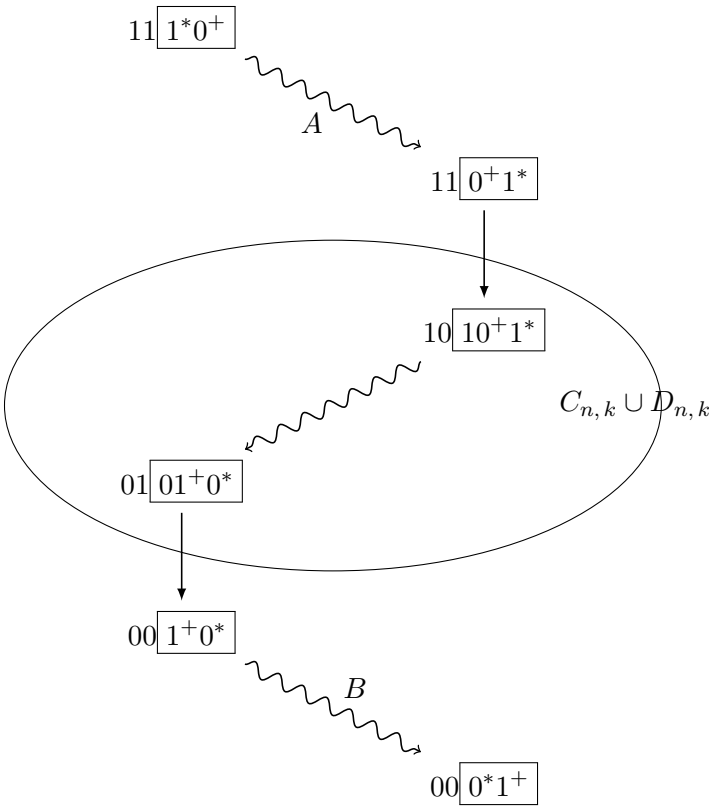
Podziały liczb są bardzo skomplikowanymi obiektami kombinatorycznymi. Przedstawimy dwa algorytmy liczenia takich oblektów. Pierwszy prosty algorytm będzie działał w czasie $\mathcal{O}(n^2)$ i pamięci $\mathcal{O}(n^2)$, natomiast drugi, pochodzący od Eulera i oparty na tzw. liczbach *pentagonalnych*, w czasie $\mathcal{O}(n\sqrt{n})$ i pamięci $\mathcal{O}(n)$.

Podział $\pi = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ to przedstawienie liczby $n = \lambda_1 + \lambda_2 + \ldots + \lambda_r$ w postaci

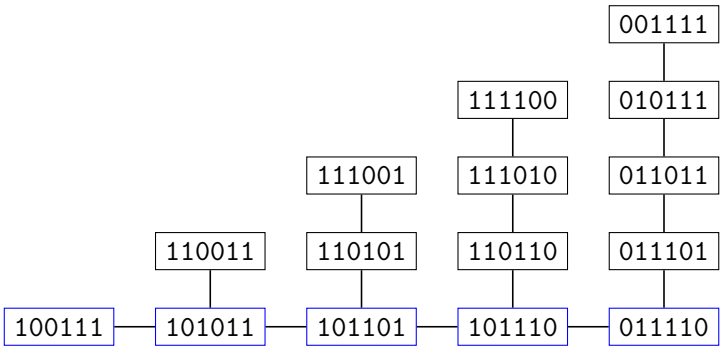
$$n = \lambda_1 + \lambda_2 + \ldots + \lambda_r, \quad \text{gdzie} \quad \lambda_1 \geqslant \lambda_2 \geqslant \ldots \lambda_r > 0$$

Wszystkie podziały liczby n, w porządku antyleksykograficznym, można wygenerować iteracyjnie następująco: szukamy pierwszego $\lambda_i \geqslant 2$ od prawej strony, zastępujemy λ_i przez $\lambda_i - 1$, a pozostałe części na prawo dajemy tak, aby sufiks na prawo od λ_i był jak największy. Na przykład podziały $n = 5$ w porządku antyleksykograficznym to:

$$5, \quad 4+1, \quad 3+2, \quad 3+1+1, \quad 2+2+1, \quad 2+1+1+1, \quad 1+1+1+1+1.$$



Rysunek 7: Schemat ścieżki Hamiltona w grafie $\mathcal{K}$.



Rysunek 8: Przykładowy grzebień rozpinający graf $\mathcal{K}(6, 4)$. Węzły ścieżki głównej zaznaczone są kolorem niebieskim.

Oznaczmy przez $p(n)$ liczbę podziałów liczby n, mamy:

n :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$p(n)$:	1	1	2	3	5	7	11	15	22	30	42	56	77	101	135	176	231

Dla $n < 0$ przyjmijmy czysto formalnie, że $p(n) = 0$, natomiast $p(0) = 1$. Oznaczmy przez $p(n,\, k)$ liczbę podziałów liczby n na k częsci (niezerowych). Algorytm o czasie kwadratowym liczenia $p(n)$ polega na policzeniu (kwadratowej liczby) wartości $p(n,\, k)$ na podstawie rekurencji:

$$p(n,k) = \begin{cases} 0 & \text{dla } k < n \vee k \leqslant 0 \\ p(n-1,\, k-1) + p(n-k,\, k) & \text{w p.p.} \end{cases}$$

Rekurencja wynika stąd, że mamy dwa przypadki:

($\lambda_k = 1$) Wtedy mamy $p(n-1,\, k-1)$ podziałów pomijając λ_k.

($\lambda_k > 1$) Wtedy możemy odjąć jeden od każdego λ_i otrzymując podział liczby $n-k$ na k części.

W celu szybszego policzenia $p(n)$ rozważymy podziały na różne części, tzn.

$$\lambda_1 > \lambda_2 > \ldots > \lambda_1.$$

Niech $\widetilde{p}(n)$ będzie liczbą takich podziałów. Przez p_{even}, $\widetilde{p}_{even}$, p_{odd}, $\widetilde{p}_{odd}$ oznaczmy liczbę podziałów na (odpowiednio) parzystą i nieparzystą liczbę części (o różnych rozmiarach w przypadku $\widetilde{p}$). Na przykład $\widetilde{p}(15) = 27$, $\widetilde{p}_{odd}(15) = 14$, $\widetilde{p}_{even}(15) = 13$, patrz rysunek 11. Zauważmy, że liczby $\widetilde{p}(n)$ są przeważnie znacznie mniejsze od liczb $p(n)$ (chociaż na początku niewiele się różnią).

Możemy rónież zdefiniować $\widetilde{p}(n,\, k)$ – liczbę podziałów n na różne części. Na przykład $\widetilde{p}(50,\, 7) = 522$, co Euler policzył prawie 300 lat temu bez komputera (ani kalkulatora) tę konkretną wartość odpwiadając na pytanie matematyka Ph. Naude.

Dygresja.
Liczba $\widetilde{p}(n)$ jest równa liczbie podziałów n na nieparzyste części (nie mylić z nieparzystą liczbą części).

Kluczową wartością jest zdefiniowana poniżej funkcja:

$$\Delta(k) = \widetilde{p}_{odd}(k) - \widetilde{p}_{even}(k)$$

Funkcje $p(n)$ i $\widetilde{p}(n)$ są bardzo skomplikowane, natomiast jest zadziwiające, że Δ jest bardzo prosta. Początkowe wartości to:

k :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
$\Delta(k)$:	1	1	0	0	-1	0	-1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	-1

Leonard Euler odkrył dwie istotne (dla liczenia $p(n)$) własności funkcji Δ:

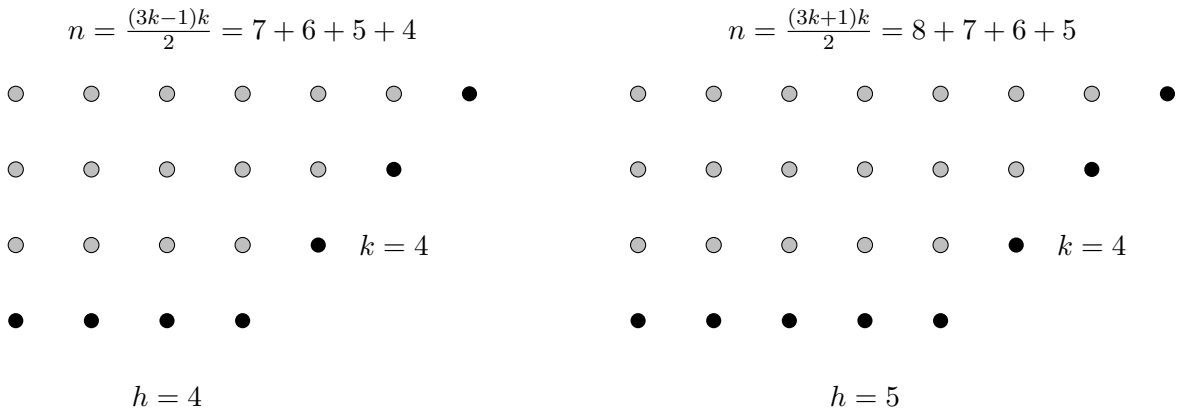
Własność 1: $p(n)$ spełnia rekurencję:

$$p(n) = \sum_{k=1}^{n} \Delta(k) \cdot p(n-k) \tag{2}$$

Własność 2.

$$\widetilde{p}_{odd}(k) - \widetilde{p}_{even}(k) \;=\; \Delta(k) \;=\; e(k). \tag{3}$$

Jak widać z początkowych wartości $\Delta(k)$ jest ciągiem bardzo *rzadkim* (prawie same zera). Jest on obliczalny łatwo za pomocą tzw. liczb pentagonalnch. Wartości ciągu to zera, $+1$ lub -1. Z tego, że ciąg $\Delta(n)$ jest bardzo rzadki wynika, że dla policzenia $p(n)$ tylko $\mathcal{O}(\sqrt{n})$ wartości k jest niezerowych. Zatem $p(n)$ liczymy w czasie $\mathcal{O}(\sqrt{n})$ znając $p(n-1)$, $p(n-2)$ $\ldots$, $p(0)$. W sumie mamy algorytm działający w czasie $\mathcal{O}(n\sqrt{n})$ i pamięci $\mathcal{O}(n)$, o ile potrafimy łatwo wylistować niezerowe wartości $\Delta(k)$.
Leonard Euler najpierw odkrył własności Δ heurystycznie, a dopiero po 10 latach znalazł dowód (być może nie zajmował się przez ten czas tym zagadnieniem).



Rysunek 9: Trapezy rzędu k, gdzie $k = 4$, pierwszego typu ma $pen(k)$ elementów, a drugiego typu ma $pen(k)+k$ elementów. Podziały odpowiadające tego typu trapezom nazywamy podziałami trapezowymi.

Możemy teraz zapisać algorytm (jedną iterację) liczenia $p(n)$ następująco:

$$p(n) = \sum_{i \geqslant 1} (-1)^{i+1} \cdot \big(p(n - pent(i)) + p(n - pent(i) - i)\big)$$

W równaniu tym korzystamy jedynie z wartości i takich, że $pent(i) \leqslant n$, mamy jedynie $\mathcal{O}(\sqrt{n})$ takich wartości. Liczby $pent(i)$ możemy łatwo policzyć.

Twierdzenie 5. *Liczby $p(1)$, $p(2)$, ..., $p(n)$ możemy policzyć w czasie $\mathcal{O}(n\sqrt{n})$ i pamięci $\mathcal{O}(n)$.*

Dowód własności 1

Uzasadnienie jest sprytną manipulacją algebraiczną, korzystającą z tego, że dwa wielomiany będące tą samą funkcją mają takie same współczynniki przy tych samych potęgach zmiennej. Sztuczka polega na tym, żeby te same wielomiany przedstawić na dwa różne sposoby. Z jednego wymnożenia otrzymujemy wynik, który przyrównujemy do wymnożenia w innej formie. Zdefiniujmy:

$$\phi(x) = (1 + x + x^2 + \ldots + x^n),\ \psi(x) = 1 - x$$
$$W_1(x) = \prod_{i=1}^{n} \phi(x^i),\ W_2(x) = \prod_{i=1}^{n} \psi(x^i)$$

Wprowadźmy notację $\stackrel{n}{=}$ dla równości wielomianów z dokładnością do potęg wyższych niż n. Inaczej mówiąc bierzemy resztę z dzielenia przez x^{n+1}. Zauważmy, że zachodzi dosyć łatwe równanie:

$$W_1(x) \cdot W_2(x) \stackrel{n}{=} 1. \tag{4}$$

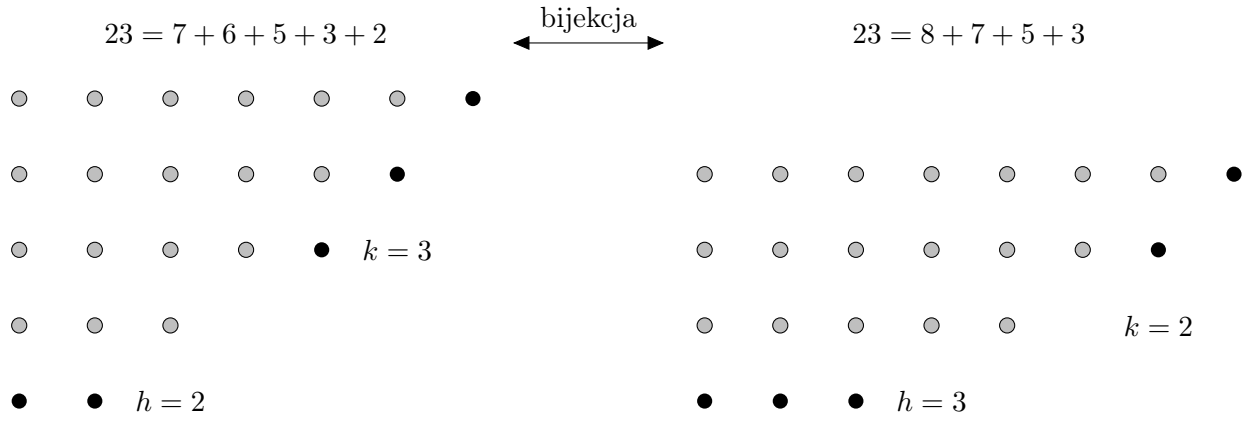
Powyższa równość trochę przypomina sytuację w równości $(1+x)(1-x) = 1-x^2$. Przedstawimy teraz te same wielomiany w innej formie.

$$W_1(x) \stackrel{n}{=} p(0)x^0 + p(1)x^1 + p(2)x^2 + \ldots + p(n)x^n,\ W_2(x) \stackrel{n}{=} 1 - \Delta(1)x^2 - \Delta(2)x^2 - \ldots - \Delta(n)x^n \tag{5}$$

Z równania (4) dla $n \geqslant 1$ wynika, że współczynnik przy x^n w iloczynie $W_1(x) \cdot W_2(x)$ wynosi zero. Korzystając z równania (5) możemy ten współczynnik przedstawić jako kombinacje iloczynów $p(i)$, $\Delta(j)$, gdzie $i + j = n$, w rezultacie otrzymujemy:

$$p(n) \cdot 1 - p(n-1) \cdot \Delta(1) - p(n-2) \cdot \Delta(2) - p(n-3) \cdot \Delta(3) \ldots - \Delta(n) \cdot p(0) = 0$$

Stąd wynika bezpośrednio równanie (2).



Rysunek 10: Działanie funkcji F.

Dowód własności 2

W części tej rozważamy tylko podziały o różnych częściach. Dowód wymaga rozważenia interpretacji geometrycznej podziałów.

Podział liczby może być przedstawiony w postaci diagramu zwanego diagramem Ferrersa – w kolejnym wierszu liczba elementów odpowiada liczbie λ_j. Diagramy Ferrersa dla przykładowych podziałów liczb 22 i 26 są przedstawione na rysunku 9. Są to bardzo szczególne podziały, które będziemy nazywać trapezowymi.

Podział trapezowy rzędu k pierwszego typu jest postaci $(k+k-1,\ k+k-2,\ k+k-3, \ldots,\ k)$ a drugiego typu postaci $(k+k,\ k+k-1,\ k+k-2, \ldots,\ k+1)$.

Obserwacja: Podział trapezowy mający $pen(k)$ lub $pen(k)+k$ elementów składa się z k części.

Liczbę n nazywamy liczbą trapezową, gdy istnieje podział n będący trapezowym. Zawsze jest co najwyżej jeden taki podział dla danego n.

Obserwacja: Liczby trapezowe są postaci $pen(j)$ lub $pen(j)+j$.

Dla trapezu π przez $k(\pi)$ oznaczmy liczbę elementów na prawej diagonali poczynając od górnej prawej strony. Jeśli odpowiadającym podziałem jest $(a_1,\ a_2, \ldots,\ a_r)$ to $k(\pi)$ jest największą liczbą naturalną taką, że $a_i - 1 = a_{i+1}$ dla $i = 1, 2, \ldots, k(\pi)-1$. Przez $h(\pi) = a_r$ oznaczmy liczbę elementów w najmniejszej części. Jeśli podział nie jest trapezowy oraz $h(\pi) \leqslant k(\pi)$ to $F(\pi)$ jest podziałem powstającym z π przez dodanie do każdej z pierwszych $k(\pi)$ części po jednym elemencie i usunięcie najmniejszej (dolnej) części, patrz rysunek 10.

Podziały o parzystej (nieparzystej) liczbie częsci nazywamy parzystymi (nieparzystymi). Zauważmy, że funkcja F zmienia parzystość podziału. Zachodzi następujący, dość oczywisty fakt.

Własność funkcji F: F jest bijekcją między nietrapezowymi podziałami n z $h(p) \leqslant k(\pi)$ i nietrapezowymi podziałami n z $h(p) > k(\pi)$.

$$\widetilde{p}_{odd}(k) - \widetilde{p}_{even}(k) = \begin{cases} 1 & \text{gdy } n \text{ jest nieparzystą liczbą trapezową} \\ -1 & \text{gdy } n \text{ jest parzystą liczbą trapezową} \\ 0 & \text{w przeciwnym przypadku.} \end{cases}$$

Z powyższej własności wynika:

$$\widetilde{p}_{odd}(k) - \widetilde{p}_{even}(k) = e(k) \text{ dla kazdego } k.$$

Więcej informacji na temat liczenia podziałów można znaleźć w:

`http://www.math.psu.edu/vstein/alg/antheory/preprint/andrews/chapter.pdf`

`http://www.rowan.edu/colleges/csm/departments/math/facultystaff/osler/89%20Surprising%20Connection%20Between%20Partitions%20and%20Divisors.pdf.pdf`

$14+1$	$\longleftrightarrow$	15	$7+5+2+1$	$\longleftrightarrow$	$8+5+2$
$13+2$	$\longleftrightarrow$	$12+2+1$	$7+4+3+1$	$\longleftrightarrow$	$8+4+3$
$12+3$	$\longleftrightarrow$	$11+2+1$	$9+3+2+1$	$\longleftrightarrow$	$10+3+2$
$11+4$	$\longleftrightarrow$	$10+4+1$	$8+4+2+1$	$\longleftrightarrow$	$9+4+2$
$10+5$	$\longleftrightarrow$	$9+5+1$	$6+5+3+1$	$\longleftrightarrow$	$7+5+3$
$9+6$	$\longleftrightarrow$	$8+6+1$	$6+4+3+2$	$\longleftrightarrow$	$5+4+3+2+1$
$8+7$	$\longleftrightarrow$	$7+6+2$	$6+5+4$		podział trapezowy

Rysunek 11: Zgrupowanie podziałów nietrapezowych liczby $n = 15$ korzystając z funkcji F.

`http://hkumath.hku.hk/mks/EulerHeuristicReasoning.pdf`

Jako ćwiczenie propnujemy dowód równości:

$$p(n,2) = \lfloor (n+1)/2 \rfloor,\ p(n,3) = \{(n+3)^2/12\},$$

where $\{x\}$ means the integer closest to x.

Nieoczekiwana relacja między funkcjami $p(n)$ i $\sigma(n)$

Jako ciekawostkę podamy (bez uzasadnienia) pewien związek dwóch pozornie odległych funkcji $p(n)$ i $\sigma(n)$, gdzie $\sigma(n)$ oznacza sumę dzielników liczby n (włącznie z n). Mamy

n :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\sigma(n)$:	0	1	3	4	7	6	12	8	15	13	18	12	28	14	24	24	31

Dla funkcji σ zachodzi prawie taka sama rekurencja jak dla $p(n)$, jedyna różnica to zastąpienie $p(0)$ przez n we wzorze (2). Funkcja $\sigma(n)$ spełnia rekurencję:

$$\sigma(n) = \sum_{k=1}^{n-1} \Delta(k) \cdot \sigma(n-k) + \Delta(n) \cdot n. \tag{6}$$

Przykład. Dla $n = 15$ mamy:

$$\begin{aligned}\sigma(15) &= \sigma(15-1) + \sigma(15-2) - \sigma(15-5) - \sigma(15-7) + \sigma(15-12) + 15\\ &= 24 + 14 - 18 - 15 + 4 + 15 = 24.\end{aligned}$$

$$\begin{aligned}p(15) &= p(15-1) + p(15-2) - p(15-5) - p(15-7) + p(15-12) + p(0)\\ &= 135 + 101 - 42 - 22 + 3 + 1 = 176.\end{aligned}$$

Ponieważ wartości $\Delta(k)$ są związane z liczbami pentagonalnymi tak jak poprzednio, to wszystkie wartości $\sigma(n)$, $\sigma(n-1)$, $\ldots$, $\sigma(1)$ można policzyć w czasie $\mathcal{O}(n\sqrt{n})$ i pamięci $\mathcal{O}(n)$, tak jak poprzednio zrobiliśmy to dla wartości $p(n)$. Zachodzi również inny zadziwiający związek:

$$\sigma(n) = \sum_{k=1}^{n} k \cdot \Delta(k) \cdot p(n-k).$$

Jeśli $n = \prod p_i^{m_i}$ gdzie p_i liczby pierwsze to

$$\sigma(n) = \frac{\prod (p_i^{m_i+1} - 1)}{\prod (p_i - 1)}$$

Wydaje się, że liczenie σ dla wszystkich liczb 1, 2,..., n jednak jest szybsze korzystając ze wzoru (6) i liczb pentagonalnych niż korzystając z ostatnego wzoru.

Dygresja. Liczby $\sigma(n)$ są związane z tzw. liczbami *doskonałymi* tzn. takimi, że $\sigma(n) = 2n$. Wiadomo, że n parzyste jest doskonałe wtedy i tylko wtedy, gdy $n = 2^{p-1}(2^p - 1)$, gdzie p i $2^p - 1$ są pierwsze, np. dla $p = 11213$, $p = 30402456$ (43. liczba doskonała). Pierwszych 7 liczb doskonałych to:

$$6, \quad 28, \quad 496, \quad 8128, \quad 33\,550\,336, \quad 8\,589\,869\,056, \quad 137\,438\,691\,328.$$

Pierwsze 4 liczby doskonałe policzył już Euklides. Kilka następnych Euler. Potem już był potrzebny komputer. Nie wiadomo czy liczb parzystych doskonałych jest nieskończenie wiele, ani też czy istnieje choćby jedna nieparzysta liczba doskonała.

Permutacje a liczby pentagonalne

Zdefiniujmy $b(n, k)$ jako liczbę n-permutacji mających k inwersji, oraz $c(n, j) = \binom{n+j-1}{j}$, gdzie $c(n, j) = 0$ dla $j < 0$. Wtedy dla $k \leq n$ mamy:

$$b(n, k) = c(n, k) + \sum_i (-1)^i \cdot (c(n, k - pent(i)) + c(n, k - pent(i) - i))$$

Otóż $b(n, k)$ jest równe liczbie ciągów $(a_1, a_2, \ldots, a_{n-1})$ takich, że $0 \leqslant a_i \leqslant i$, $\sum_i a_i = k$. Wynika to z analizy algorytmu sortowania przez wstawianie (*InsertionSort*).

Przekładając to na język wielomianów $b(n, k)$ jest współczynnikiem przy x^k w wielomianie

$$W_3(x) = (1 + x)(1 + x + x^2) \ldots (1 + x + x^2 + \ldots + x^{n-1})$$

Ale możemy ten wielomian zapisać jako $W_3(x) = W_2(x)/(1-x)^n$, a wielomian $W_2(x)$ ma, jak to już widzieliśmy przy liczeniu podziałów liczby, wiele wspólnego z liczbami pentagonalnymi, stąd zatem mamy relację między permutacjami i liczbami pentagonalnymi.

Poniże pokazujemy *tabelkę* poczatkowych wartości $b(n,k)$, kolumny odpowidają $k = 0, 1, 2, \ldots$, a wiersze kolejneym $n \geq 1$.

1															
1	1														
1	2	2	1												
1	3	5	6	5	3	1									
1	4	9	15	20	22	20	15	9	4	1					
1	5	14	29	49	71	90	101	101	90	71	49	29	14	5	1

Zauważmy, ze wiersze są symetryczne. Załóżmy, że jeśli $k < 0$ lub $k > \binom{n}{2}$ to $b(n, k) = 0$.
Wtedy dla $k \geq 0$ & $k > \binom{n}{2}$ zachodzi równość:

$$b(n, k) \;=\; b(n, k - 1) + b(n - 1, k) - b(n - 1, k - n).$$

6 Pierwiastkowanie permutacji

Rozważymy pewien problem dotyczący permutacji, rozwiązywalny w czasie liniowym za pomocą rozkładu na cykle. Dla zadanej permutacji π i liczby k określamy:

Pierwiastkowanie: znaleźć jakąkolwiek permutację γ taką, że $\gamma^k = \pi$ (oznaczmy takie γ przez $\sqrt[k]{\pi}$), ewentualnie stwierdzić, że nie ma pierwiastka. Może nie istnieć pierwiastek, np. dla $\pi = (2, 1, 3, 4)$ nie ma pierwiastka kwadratowego, a dla $\pi = (1, 2, 3, 4)$ mamy aż 10 takich pierwiastków.

Fakt. Rozkład permutacji na cykle można policzyć w czasie liniowym. Wystarczy reprezentować permutację jako graf skierowany, w którym do każdego wierzchołka wchodzi i z którego wychodzi dokładnie jedna krawędź. Następnie iterujemy przez wszystkie elementy permutacji szukając nieodwiedzonego wierzchołka. Dla każdego takiego węzła przechodzimy przez wszystkie elementy cyklu, oznaczając je po drodze jako odwiedzone. Możemy w ten sposób znaleźć wszystkie możliwe informacje dotyczące cykli w permutacji: ich liczbę i długość oraz który element należy do której.

Algorytm 8: Rozkład permutacji na cykle

```
1  L := pusta lista cykli;
2  for i := 1 to n do
3  |  if visited[i] = false then
4  |  |  C := pusty cykl;
5  |  |  j := i;
6  |  |  repeat
7  |  |  |  dodaj j do C;
8  |  |  |  visited[j] := true;
9  |  |  |  j := π[j];
10 |  |  until i = j;
11 |  |  dodaj C do L;
12 end
```

Zacznijmy od pierwiastka kwadratowego. Niech $\#cykle(\pi, k)$ oznacza liczbę cykli długości k w rozkładzie cyklowym permutacji π.

Fakt. π ma pierwiastek kwadratowy wtedy i tylko wtedy gdy dla każdego parzystego k liczba $\#cykle(\pi, k)$ jest parzysta.

Potrzebna nam będzie kluczowa operacja *interlace*. Jeśli mamy kilka rozłącznych cykli C_1, C_2, ..., C_t tej samej długości to $interlace(C_1, C_2, \ldots, C_t)$ otrzymujemy wstawiając kolejne elementy C_2 po elementach C_1, następnie kolejne elementy C_3 po elementach C_2 itd. Na przykład:

$$interlace((1, 2, 3), (4, 5, 6), (7, 8, 9)) = (1, 4, 7, 2, 5, 8, 3, 6, 9).$$

Jeśli mamy mamy cykl nieparzysty $(i_0, i_1, \ldots, i_{r-1})$, gdzie $r = 2k + 1$, to jego pierwiastkiem jest cykl $(j_0, j_1, \ldots, j_{r-1})$, gdzie $j_p = i_{p\cdot(k+1) \bmod r}$. Na przykład jeśli mamy cykl (1, 2, 3, 4, 5) to jego pierwiastkiem jest (1, 4, 2, 5, 3).

Natomiast jeśli mamy dwa cykle parzyste C_1 i C_2 tej samej długości, to pierwiastkiem kombinacji tych cykli jest $interlace(C_1, C_2)$. Pierwiastek kwadratowy liczymy w ten sposób, że dla każdego nieparzystego cyklu obliczamy jego cykl bedący pierwiastkiem, a dla każdej pary parzystych cykli tej samej długości zastepujemy je przez jeden cykl za pomoca operacji *interlace*.

Przykład. Niech

$$\pi = (2, 3, 4, 5, 1, 7, 8, 9, 6, 11, 12, 13, 10)$$

Rozkład na cykle to

$$(1, 2, 3, 4, 5),\ (6, 7, 8, 9),\ (10, 11, 12, 13)$$

Pierwszy cykl pierwiastkujemy otrzymując (1, 4, 2, 5, 3). Do dwóch pozostałych cykli (tej samej parzystej długosci) stosujemy *interlace* i otrzymujemy cykl (6, 10, 7, 11, 8, 12, 9, 13). Zatem wynikowa permutacja ma rozkład na cykle

$$(1, 4, 2, 5, 3),\ (6, 10, 7, 11, 8, 12, 9, 13)$$

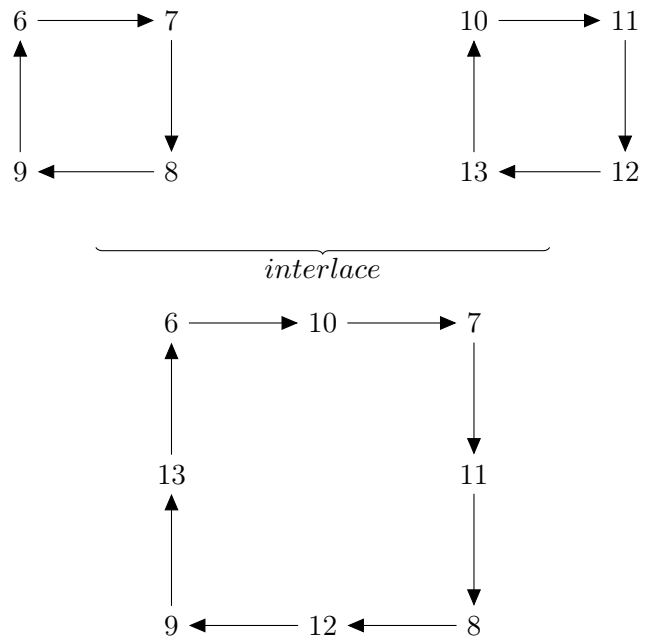
Ostatecznie

$$\sqrt[2]{\pi} = (4, 2, 5, 3, 1, 10, 7, 11, 8, 12, 9, 13, 6)$$

Systuacja jest podobna, choć bardziej skomplikowana, dla pierwiastka dowolnego stopnia m.

Systuacja jest podobna, choć bardziej skomplikowana, dla pierwiastka dowolnego stopnia m. Załóżmy, że faktoryzacja m na potęgi liczb pierwszych wygląda nastepująco:

$$m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_s^{\alpha_s}$$



Rysunek 12: Ilustracja operacji *interlace*.

Definiujemy:

$$((k,m)) \;=\; \prod_{p_i \mid k} p^{\alpha_i}.$$

Fact. π ma pierwiastek m-tego stopnia, gdzie $m > 1$, wtedy i tylko wtedy gdy dla każdego k liczba $((k,m))$ jest dzielnikiem liczby cykli długości k.

Inaczej mówiąc jeśli p_i dzieli $nwd(k,m)$ to $p_i^{\alpha_i}$ jest dzielnikiem liczby cykli długości k.

Problem ten jest opisany dokładniej w książce "Wyzwania algorytmiczne" z ICPC ACM, jako zadanie "Tasowanie" autorstwa P. Parysa.

Problem. Oblicz liczbę pierwiastków kwadratowych permutacji identycznosciowej rzędu n.

7 Generacja liczb pierwszych

Liczby pierwsze są jednymi z najciekawszych obiektów kombinatorycznych. Powszechna metoda sita Eratostenesa przetwarza tablicę liczb z przedziału $[2, n]$: w pętli znajdujemy pierwszą niewykreśloną liczbę, którą dodajemy do listy wynikowej, a następnie wykreślamy z tablicy wszystkie wielokrotności tej liczby. Łącznie wykonywanych jest $\mathcal{O}(n \log\log n)$ wykreśleń – iterując przez wielokrotności kolejno znajdowanych liczb pierwszych niektóre liczby wykreślimy wielokrotnie. Opiszemy algorytm z 1978 roku autorstwa Davida Griesa i Jayadeva Misry, który generuje wszystkie liczby pierwsze z przedziału $[2, n]$ wykonując jedynie liniową liczbę wykreśleń.

Zdefiniujmy operację *RemovePowers*(p, q, L), która z listy liczb naturalnych L usuwa wszystkie liczby postaci $p^i \cdot q$, dla $i \geqslant 1$. Chcemy, aby koszt tej operacji był proporcjonalny do liczby usuniętych elementów plus pewna stała. O ile usuwanie elementów z listy dwukierunkowej jest łatwo wykonalne w czasie stałym, o tyle trudne jest znalezienie elementu, który chcemy usunąć.

Na szczęście można skorzystać z dość prostego triku technicznego, pozwalającego szybko znajdować interesujące nas elementy listy. Określimy tablicę Ptr, która dla każdej liczby $x \in L$ zawiera wskaźnik do odpowiadającego mu elementu w L. Tablicę Ptr inicjujemy podczas konstrukcji L. Jeśli usuniemy liczbę x z L, to zapisujemy $Ptr[x] =$ `NULL`, co pozwala rozpoznać liczby już usunięte i nie duplikować wykreśleń.

Operację *RemovePowers*(p, q, L) możemy zaimplementować w taki sposób, aby dla każdej wykreślonej liczby x zapamiętać liczby p^i. Wtedy możemy dokonać faktoryzacji dowolnej liczby z zakresu $[2, n]$ w czasie

liniowym względem liczby czynników pierwszych x drogą sukcesywnego dzielenia przez spamiętane wartości p^i.

Algorytm 9: Gries-Misra

```
1  L := [2, n];
2  Inicjuj tablicę Ptr;
3  p := L.begin(); // p wskazuje na liczbę 2
4  while p ≠ NULL do
5  |  q := p;
6  |  while q ≠ NULL do
7  |  |  RemovePowers(p, q, L);
8  |  |  q := next(q);
9  |  end
10 |  p := next(p);
11 end
12 return L;
```

Fakt. Algorytm Gries-Misra wykonuje $\mathcal{O}(n)$ wykreśleń liczb – każda liczba, która trafiła do L albo jest liczbą pierwszą i nie zostaje wykreślona, albo zostaje wykreślona dokładnie raz i już do niej nie wracamy.

8 Kilka prościutkich problemów algorytmiczno-teorioliczbowych

Na cyfrach liczby zapisanej dziesiętnie można wykonywać proste operacje.

Możenie-Skreślanie. Dysponujemy operacjami skreślania zer i mnożenia przez dowolną liczbe naturalną.
Problem: Dla danej liczby n podać ciąg operacji $n \to^* 9$.

Posłużymy się rozwiązaniem **pomocnieczego problemu**: dla danej liczby n, $(n, 10) = 1$ można ją przemnożtć przez pewną liczbę otrzymując liczbę złożoną z samych jedynek. Zapiszmy to $n \to m = 1^k$ dla pewnego k.

Teraz nasz cią operacji dla $n \to^* 9$ wygląda następująco:

1. $n \to^* x$, dla pewnego x takiego, że $(x, 10) = 1$, wykonując pewną liczbę mnożeń przez 2 lub 5, oraz skreślen zer.
2. $x \to^* m = 1^k$
3. $m := m \times 82 = 9111...02$
4. skreślamy zero, mnożymy przez 9: $m \to^* 8200..8$
5. $482000...8 \to 828$, mnożymy przez 25, skreślmay zera, otrzymujemy 277.
6. mnożymy przez 4, skreślamy zera, otrzymujemy 18.
7. mnożymy przez 5, skreślamy zera. Otrzymujemy 9 !!!

Rozwiązanie problemu pomocniczego. Istnieją dwie liczby postaci 1^p, 1^q, gdzie $p > q$ dające tę samą reszte modulo n. Wtedy $1^p - 1^q$ jest podzielne przez n, pomijając zera mamy liczbe postaci 1^k podzialna przez n.

Dopisywanie-dzielenie. Dysponujemy operacjami dopisywania na końcu zera lub czwórki, oraz dzielenia liczby parzystej przez 2.
Problem: Dla danej liczby n podać ciąg operacji $4 \to^* n$.

Generacja liczb 1-10:

$$4 \to 2 \to 1,\ 2 \to 24 \to 12 \to 6 \to 3,\ 6 \to 64 \to 32 \to 16 \to 8,$$

$$2 \to 20 \to 10 \to 5,\ 10 \to 14 \to 7.$$

Pojedyńcze mnożenie. Dla danej liczby n zanaleźć $n^4 \geq x \geq 1$ takie, że $n \times x$ ma w zapisie dziesiętnym co najwyżej 4 różne cyfry.

Załóżmy, że $2^{k-1} \leq n < 2^k$. Niech S_k będzie zbiorem liczb mających k cyfr - same zera/jedynki. Istnieją dwie liczby $a < b$ w zbiorze S mające tę sama resztę modulo n, liczba $b - a$ jest podzielna przez n i ma w zapisie tylko cyfry 0,1,8,9. Weźmy $x = (b - a)/n$.

Sumy-różnice kwadratów. Przedstawić liczbę n w postaci sumy typu

$$n \ = \ \pm 1^2 \pm 2^2 \pm 3^2 \ldots \pm m^2,$$

Można skorzystać z tożsamości:

$$k^2 - (k+1)^2 - (k+2)^2 + (k+3)^2 = 4$$

gdzie $m \leq 4n$.

Odejmowanie cykliczne. Dla ciągu liczb całkowitych $\gamma \ = \ (a_1, a_2, \ldots a_n)$, definiujemy operację

$$F(\gamma) \ = \ (a_1 - a_2, a_2 - a_3, \ldots a_{n-1} - a_n, a_n - a_1)$$

(w operacji tej wykonujemy działania na poprzednich wartościach liczb.)
Zachodzi następująca ciekawa własność tej operacji.

Fakt.. Jeśli n jest potęgą dwójki to istnieje takie k, że

$$F^k(\gamma) \ = \ (0, 0, 0, \ldots, 0).$$

9 Kombinatoryczne własności ciągu Thue-Morse'a

Niech τ_n będzie ciągime binarnym długości 2^n numerowanym od zera. Oznaczmy A_n, B_n pozycje zawierające odpowidnio 0,1. Zedfniujmy ciąg τ_n rekurencyjnie:

$$A_0 = \{0\}, \ A_{n+1} = B_n + 2^n, \ B_{n+1} = A_n + 2^n.$$

Problem Poucheta-Tarry'ego-Escota Następujący fakt dowodzimy indukcyjnie względem n:

Fakt.

$$(\forall\, 0 \leq k < n) \sum_{i \in B_n} i^k \ = \ \sum_{j \in A_n} j^k.$$

Konstrukcja kwadratów magicznych rzędu 2^n Ponumerujmy pola kwadratu $N \times N$, dla $N = 2^n$ kolejno wierszami *idąc* od lewej do prawej w każdym wierszu, pierwszy numer to zero. Pole o numerze i zawiera $i + 1$ jeśli $A_{n+1} = 0$, wpp. zawiera $2^{n+1} - i$.

Fakt. *Tak wypełniony kwadrat jest kwadratem magicznym.*

Wystarczy zauważyć, że każde kolejne cztery elementy wiersza (poczynając od lewej strony) mają taką samą sumę, podobnie każde kolejne cztery elementy kolumny mają tę samą sumę (poczynając od góry kwadratu).

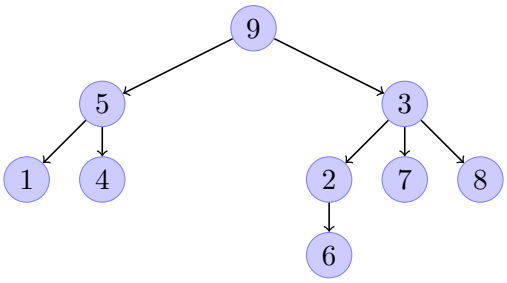
Natomiast przekątne są jednorodne, ich wszystkie pozycje należą do A_{n+1} albo do B_{n+1}.

10 DFS i BFS

Załóżmy (dla uproszczenia), że z wierzchołka zwanego *root* w danym grafie G można dojść do każdego innego (w grafie skierowanym lub nieskierowanym). Ze ścieżek dojścia z *root* do wszystkich wierzchołków można skonstruować drzewo (tak zwane drzewo DFS) będące rodzajem algorytmicznego „szkieletu" grafu.

DFS jest abstrakcją algorytmu przeszukującego graf „w głąb", startując z jakiegoś węzła „odwiedzamy" węzły, najpierw syna aktualnie odwiedzonego węzła. W każdym węźle odwiedzonym po raz pierwszy staramy się pójść do jego kolejnego nieodwiedzonego syna. Jeśli takiego nie ma to wycofujemy się. W czasie chodzenia po grafie otrzymujemy drzewo DFS, składające się z krawędzi pierwszego dojścia do węzłów. Niech $parent(v)$ oznacza poprzednik węzła v w drzewie. Mamy dwie podstawowe kolejności wierzchołków drzewa:

- *preorder*: wypisujemy w momencie pierwszego odwiedzenia
- *postorder*: wypisujemy w momencie ostatniego odwiedzenia
- *EulerTour*: wypisujemy wierzchołki w momentach pierwszego i ostatniego odwiedzenia (wchodzenie do i wychodzenie z wierzchołka). Kolejność ta jest złączeniem preorder i postorder.



Preorder	9, 5, 1, 4, 3, 2, 6, 7, 8
Postorder	1, 4, 5, 6, 2, 7, 8, 3, 9
EulerTour	9, 5, 1, 1', 4, 4', 5', 3, 2, 6, 6', 2', 7, 7', 8, 8', 3', 9'

Rysunek 13: Kolejności wierzchołków dla przykładowego drzewa

Graf nieskierowany jest 2-spójny gdy po usunięciu dowolnego wierzchołka jest spójny. Wierzchołki, które rozspójniają graf nazywamy węzłami rozdzielającymi lub punktami artykulacji. Krawędź nazywamy mostem (przewężeniem), gdy usunięcie tej krawędzi rozspójnia graf.

10.1 Zastosowanie DFS: silna orientacja grafu nieskierowanego

Graf jest silnie spójny jeśli istnieje skierowana ścieżka między każdymi dwoma węzłami. Wiadomo, że graf nieskierowany G można zorientować tak, aby był silnie spójny wtedy i tylko wtedy, gdy jest spójny i nie ma mostu. Następujący algorytm wykonuje orientację.

Algorytm 10: Silna orientacja grafu

wejście: $G = (V, E)$ – graf nieskierowany bez mostów
wyjście: $G = (V, E)$ – wejściowy graf po silnej orientacji

```
1 Policz preorder i DFS-drzewo T grafu G;
2 foreach e ∈ E do
3 |   if e ∈ T then
4 |   |   Zorientuj e w kierunku rosnącego preorder;
5 |   else
6 |   |   Zorientuj e w kierunku malejącego preorder;
7 |   end
8 end
```

10.2 Zastosowanie DFS: dwuspójność, węzły rozdzielające i mosty

Pokażemy jedynie jak obliczać węzły rozdzielające i mosty grafu nieskierowanego, wypisywanie dwuspójnych składowych zostawiamy jako ćwiczenie.

Załóżmy, że obliczyliśmy DFS-drzewo T oraz *postorder* i *preorder*. Utożsamiamy węzły z ich numerami w porządku *preorder*. Przechodzimy graf w porządku *postorder* i obliczamy:

$$low[v] := \min(\{v\} \cup \{w : (v, w) \in E - T\} \cup \{low[w] : parent(w) = v\})$$

Teraz v jest węzłem rozdzielającym jeśli dla pewnego syna w (tzn. $parent(w) = v$) zachodzi $low[w] \geqslant v$. Jeśli $low[w] = w$ to (v, w) jest mostem (krawędzią rozdzielającą).

10.3 "Ear-decomposition" grafu dwuspójnego

Pokażemy konstruktywnie następujący fakt:

każdy nieskierowany graf dwuspójny G możemy wygenerować startując z cyklu prostego, a następnie doklejając do grafu "uszy"(ścieżki mające tylko końcowe węzły w grafie).

Początkowo graf G' jest jakimkolwiek cyklem prostym w G.
while $G' \neq G$ **do**
wybieramy $(v, w) \notin G'$, $v \in G'$;
if $w \in G$ **then** doklejamy krawędź (v, w) do G'
else
usuwamy v, bierzemy ścieżkę od w do pewnego $w' \in G'$
na której pośrednie węzły nie są w G'; doklejamy tę ścieżkę do G';

10.4 Silnie spójne składowe grafu skierowanego - algorytm Tarjana

Tym razem graf jest skierowany, zatem DFS-drzewo składa się z zadanym oryginalnie skierowaniem krawędzi. Opiszemy algorytm Tarjana dzielącego graf wejściowy na podgrafy. Zaskakujące jest jego podobieństwo do algorytmu związanego z dwuspójnością. Teraz też zakładamy, że obliczyliśmy DFS-drzewo T oraz *postorder* i *preorder*.

Utożsamiamy węzły z ich numerami w porządku *preorder*. Dla każdego węzła v obliczamy predykat $repr[v]$ czy jest on najmniejszym węzłem w swojej silnie spójnej składowej (reprezentantem tej składowej). Początkowo *repr* zawiera same zera.

Algorytm 11: Tarjan

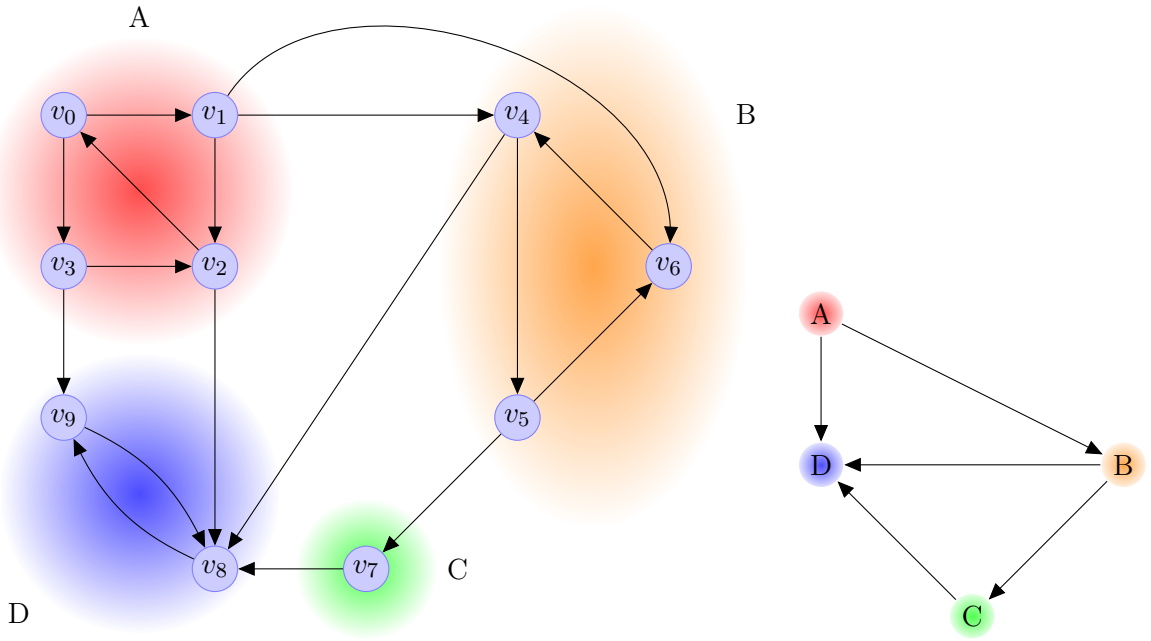
```
1 foreach v ∈ V w kolejności postorder do
2 |   low[v] := min({v} ∪ {w : (v, w) ∈ E − T} ∪ {low[w] : parent(w) = v});
3 |   if low[v] = v then
4 |   |   repr[v] := true;
5 |   |   U := zbiór węzłów w poddrzewie T_v;
6 |   |   V := V − U;
7 end
```

Teraz już łatwo możemy wypisać składowe silnie spójne:

Algorytm 12: Tarjan

```
1 SCC := ∅;
2 foreach v ∈ V, w kolejności postorder do
3 |   insert(v, SCC);
4 |   if repr[v] = true then
5 |   |   Wypisz składową SCC;
6 |   |   SCC := ∅;
7 end
```

W algorytmie Tarjana trzeba tak zaimplementować instrukcję $V := V - U$, aby nie usuwać tego samego węzła dwa razy. Wtedy algorytm działa w czasie liniowym.



Rysunek 14: Graf skierowany z pokolorowanymi silnie spójnymi składowymi, oznaczonymi kolejnymi literami od A do D. Po prawej graf silnie spójnych składowych. Jak widać pojedynczy wierzchołek także może być silnie spójną składową.

10.5 Silnie spójne składowe grafu skierowanego - algorytm Kosaraju

A teraz podamy inny algorytm, historycznie późniejszy i prostszy niż algorytm Tarjana.

Algorytm 13: Kosaraju

```
1 Przechodzimy graf DFSem i numerujemy w postorder;
2 odwracamy zorientowanie krawędzi grafu, otrzymując graf G^R;
3 foreach v ∈ V, w kolejności malejącego postorder do
4 |   if v nie jest usunięty then
5 |   |   Wypisz jako kolejną składową zbiór
6 |   |   wierzchołków osiągalnych z v w grafie G^R;
7 |   |   usuń wypisane wierzchołki i krawędzie prowadzące do nich;
8 end
```

Uzasadnienie poprawności:

Wystarczy udowodnić, że jeśli $v \to^* x$ w grafie G^R to $v \to^* x$ w grafie G. Rozważamy dwa przypadki, niech *preorder* będzie kolejnością odwiedzania w czasie pierwszego przechodzenia DFSem. Dowód przez zaprzeczenie, przypuśćmy, że nie zachodzi $v \to^* x$ w grafie G

Przypadek 1: $preorder(v) < preorder(x)$. Wtedy zakończymy v przed x, zatem $postorder(v) < x$, nie zaczniemy wykonywać szukania wierzchołków osiągalnych w grafie G^R z v. Sprzeczość.

Przypadek 2: $preorder(v) < preorder(x)$. Dowód podobny.

10.6 Zastosowanie DFS: przydział krawędzi incydentnych

Chcemy znaleźć funkcję różnowartościową P (przydział) ze zbioru wierzchołków w krawędzie, aby $P(v)$ było zawsze krawędzią incydentną. Załóżmy, że graf nie jest drzewem. Wybierzmy jako *root* wierzchołek na jakimkolwiek cyklu, stwórzmy drzewo DFS o korzeniu *root*. Wtedy $P[v] = (v, parent(v))$, dla $v \neq root$. Natomiast $P(root)$ to jakakolwiek krawędź incydentna z *root* jeszcze nie wybrana.

10.7 Zastosowanie postorder do najliczniejszego zbioru f-niezależnego

Niech $f : [n] \longrightarrow [n]$. Zbiór $X \subseteq [n]$ jest f-niezależny gdy $f(X) \cap X = \emptyset$. Mamy znaleźć najliczniejszy zbiór f-niezależny. Funkcja f może być traktowana jako zbiór pseudo-drzew skierowanych (od syna do ojca) — drzewa, w których korzeń ma swojego „ojca" w drzewie. Graf taki nazywamy *grafem funkcyjnym*.

Załóżmy, że naszej funkcji odpowiada jedno pseudo-drzewo (jeśli nie to wszelkie obliczenia są niezależne w rozłącznych pseudodrzewach). Dla ustalonej permutacji π zbioru $[n]$ definiujemy zbiór $X(\pi)$ jako wynik algorytmu:

```
1 DEAD := ∅;
2 for i := 1 to n do
3 |   if π[i] ∉ DEAD then
4 |   |   Insert(f(π[i]), DEAD);
5 |   end
6 end
7 return {1, 2, .., n} − DEAD.
```

Niech π będzie porządkiem *postorder* drzewa DFS dla grafu funkcji f oraz niech π' będzie cyklicznym przesunięciem π (ostatni element na początek). Wtedy najliczniejszym (jednym z wielu) jest większy ze zbiorów $X(\pi)$, $X(\pi')$.

Zastanówmy się teraz nad permutacją π, która minimalizuje $X(\pi)$. Załóżmy, że pseudodrzewo nie jest jedynie cyklem. Wybierzmy jako korzeń węzeł *root* taki, że istnieje $j \neq f(root)$, $f(j) = f(f(root))$. Wtedy porządkiem π minimalizującym $X(\pi)$ jest *preorder* w pseudodrzewie zakorzenionym w *root*.

10.8 Zastosowanie postorder do problemu najkrótszej ścieżki

Załóżmy, ze skierowany graf acykliczny G ma jedno *źródło* s (od słowa "source") i *ujście* t (od słowa "terminal"), oraz krawędzie grafu mają wagi liczbowe.
Naszym problemem jest znalezienie najkrótszej scieżki $s \longrightarrow^* t$.

Niech π będzie porządkiem *postorder* drzewa DFS dla grafu G w którym s jest ostatnim węzłem a t pierwszym. Niech $dist(t) = 0$. Następnie dla węzłów $v \neq t$ grafu G w porządku malejących wartości π wykonujemy:

$$dist(v) \ := \ \min \{ \ waga(v, w) + dist(w) \ : \ (v, w) \in E\}$$

Pokażemy teraz zastosowanie do (pozornie niezwiązanego) problemu liczenia najkrótszego bitonicznego cyklu Hamiltona w grafie skierowanym G, niekoniecznie acyklicznym. Dla uproszczenia załóżmy, że jakiś cykl Hamiltona w grafie G istnieje (chociaż nasz algorytm przy okazji to sprawdzi).
Definicja. Cykl Hamiltona jest **bitoniczny** jeśli startując od węzłą o numerze 1 najpierw numery węzłów rosną (do numeru n) a potem maleją do 1. Z punktu widzenia węzła 1 cykl ten składa isę z dwóch rozłącznych rosnących ścieżek od 1 do n Nasz algorytm będzie polegał na tym, że będziemy sie posuwać naraz na tych scieżkach w kierunku n, koncentrując sie na tzw. konfiguracjach *specjalnych*. Zaczniemy w węźle 1 i będziemy inkrematalnie budować te scieżki. Oznaczmy przez j konfigurację specjalną, która odpowiada sytuacji gdy jedna ze ścieżek kończy sie na węźle j, druga na $j - 1$ oraz zawierają (do tego momentu) wszystkie węzły od 1 do j. Istotne jest to, że porządek ścieżek jest pomijany, tzn. nie jestotne która jest pierwsza a która duga.

W następnym kroku na jednej ze scieżek idziemy *długą* krawędzią i dochodzimy do j'. Wtedy na druguej ścieżce musimy za pomocą *krótkich* krawędzi dojść do $j' - 1$.

Utworzymy graf acykliczny G' składający się z przejść miedzy konfiguracjami specjalnymi. Koszt krawędzi to koszt długiej krawędzi w G plus sumaryczny koszt krótkich krawędzi, co możemy wstępnie policzyć dla każdej długiej krawędzi. Pozostawimay szczegóły konstrukcji grafu G' czytelnikowi.

Graf G' ma tego samego rzędu liczbę węzłów i krawędzi co graf G. Ważna jego własnościa jest acykliczność.

Teraz stosujemy algorytm dla grafów acyklicznych i liczymy minimalny koszt ścieżki z konfiguracji początkowej do końcowej. Sytuacja początkowa odpowiada gdy obie ścieżki kończą sie (i zaczynają) w 1, nie jest ona specjalna ale nastepna po niej już jest.

Tak więc otrzymujemy algorytm o złożoności liniowej liczący minimalny bitoniczny cykl Hamiltona w dowolnym grafie. [1]

[1] Przydalby sie przyklad graficzny ilustrujacy algorytm i graf G'.

Opisany algorytm jest przedstawiony bardzo przystępnie w artykule J. Radoszewskiego w czasopośmie Delta (Październik 2012).

10.9 Zastosowanie DFS do iteracji funkcji

Niech $f : [n] \longrightarrow [n]$. Chcemy policzyć w czasie liniowym f^m dla zadanego m. Algorytm w czasie $\mathcal{O}(n \log m)$ jest oczywisty. Załóżmy dalej, że naszej funkcji odpowiada jedno pseudo-drzewo. Składa się ono z cyklu $(v[0], v[1] \ldots, v[k-1])$ oraz podwieszonych drzew. Niech T będzie drzewem podwieszonym w $root = v_0$. Najpierw liczymy odległość $D[v]$ od każdego wierzchołka v do $root$, jednocześnie pamiętając w tablicy POM ścieżkę do v.

```
1  D[root] := 0;
2  POM[0] := root;
3  foreach v ≠ root in preorder do
4  |  D[v] := D[f(v)] + 1;
5  |  POM[D[v]] := v;
6  |  if D[v] ⩾ m then
7  |  |  f^m(v) := POM[D[v] − m];
8  |  else
9  |  |  f^m(v) := v[(m − D[v]) mod k];
10 |  end
11 end
```

10.10 Obchodzenie drzewa 3-skokami

3-ścieżka (3-cykl) jest to taka permutacja węzłów, że każdy następny węzeł jest odległy od poprzeniego co najwyżej o 3 krawędzie.

Konstrukcja 3-cyklu Hamiltona w drzewie. Węzeł jest parzysty jeśli jego odległość (liczba krawędzi) od korzenia jest parzysta. Listujemy $EulerTour$, jeśli węzeł parzysty, to usuwamy jego wersję drugą (*postorder*), w przeciwnym razie pierwszą (*preorder*). Rozważmy drzewo z rysunku 13. Węzły na parzystych odległościach od korzenia to 9, 1, 4, 2, 7, 8. Mamy:

$$EulerTour = 9,\ 5,\ 1,\ 1',\ 4,\ 4',\ 5',\ 3,\ 2,\ 6,\ 6',\ 2',\ 7,\ 7',\ 8,\ 8',\ 3',\ 9'.$$

Dla węzłów parzystych wyrzucamy drugą kopię, dla nieparzystych pierwszą kopię. Zatem otrzymany 3-cykl Hamiltona to:

$$9,\ 1,\ 4,\ 5',\ 2,\ 6',\ 7,\ 8,\ 3'$$

Trzeba jeszcze usunąć oznaczenia kopii węzła, tzn. każde v' przepisujemy jako v. Dostajemy ostatecznie:

$$9,\ 1,\ 4,\ 5,\ 2,\ 6,\ 7,\ 8,\ 3.$$

Inna konstrukcja 3-cyklu Hamiltona. Kostruujemy cykl mający dodatkową własność: odległość między korzeniem i jednym (lub dwoma) sąsiednimi węzłami na cyklu wynosi dokładnie 1. Krawędź realizującą tę odległość nazwijmy *specjalną*. Wtedy rekurencyjnie konstruujemy cykle dla poddrzew zawieszonych w synach danego węzła v. Potem odpowiednio podłączmy cykle nawzajem do siebie używając jedynie v i krawędzi specjalnych, tak aby nowy cykl dla poddrzew z korzeniem w v miał dodatkową własność.

10.11 Obchodzenie drzewa 2-skokami

Przypuśćmy że chcemy przejść od wierzchołka u do v, gdzie $u \neq v$ 2-skokami realizując pewną ścieżkę Hamiltona. Algorytm jest bardzo prosty w działaniu, ale nietrywialny "dlaczego działa". Nie zawsze taka ścieżka istnieje.

Algorytm.
Zaczynamy w u i za każdym razem idziemy do jeszcze nie odwiedzonego węzła jak najbardziej odległego od v. W przypadku remisu preferujemy liść drzewa.

10.12 BFS

Podobny algorytm, w którym najpierw staramy się odwiedzić wszystkich synów danego wierzchołka nazywa się szukaniem wszerz (BFS). Tutaj również otrzymujemy drzewo zwane drzewem BFS, składające się z krawędzi pierwszego dojścia do węzłów.

10.13 Odległości od ustalonego wierzchołka w grafie nieskierowanym

Pierwszym naturalnym zastosowaniem jest liczenie odległości $D[v, \cdot]$ najkrótszych ścieżek w grafie nieskierowanym od wierzchołka v do każdego innego. Mając drzewo BFS startujące w v odległość od v jest numerem kolejnej warstwy, w której jest dany element.

10.14 Obliczanie średnicy w grafie nieskierowanym

Dla dowolnego węzła znajdujemy najbardziej oddalony od niego węzeł v, następnie najbardziej oddalony od v węzeł w, najkrótsza ścieżka między v i w jest (być może jedną z wielu) średnicą grafu.

10.15 Dla każdego wierzchołka odległość do najdalszego wierzchołka

W dalszym ciągu graf jest nieskierowanym drzewem. Wystarczy znaleźć średnicę i dla każdego wierzchołka policzyć maksimum z odległości do końców średnicy.

10.16 Sumy odległości w drzewie

W problemie tym dla każdego wierzchołka v danego drzewa chcemy policzyć wartość $suma(v)$ równą sumie odległości v do wszystkich innych wierzchołków. Można to łatwo policzyć w czasie liniowym przechodząc drzewo DFS najpierw po poziomach od dołu do góry a potem w przeciwnym kierunku.

10.17 Rozcyklanie grafu

Chcemy obliczyć minimalną moc zbioru $X \subseteq E$ krawędzi grafu nieskierowanego $G = (V, E)$ tak aby nie było cykli po susnięciu X. Załóżmy, że graf jest spójny. Znajdujemy drzewo DFS (lub BFS), pozostałe krawędzie tworzą zbiór X. Wynikiem jest moc tego zbioru.

Jest to bardzo proste. Nazwijmy ten algorytm Algorytmem A.

Sytuacja robi sie bardziej skomplikwana gdy wyróżnimy pewien zbiór wierzchołków N, nazwijmy je czerwonymi. oraz gdy zaążdamy jedynie żeby nie było cyklu przechodzącego przez wierzchołek czerwony. Krawędź której co najmniej jeden z końców jest czerwony nazwijmy czerwoną.

Mżna udwodnić, że najmniej liczny zbiór X krawędzi, którego usunięcie rozcykla graf w sensie czerwonym zawiera tylko czerwone krawędzi.

Algorytm teraz polega na tym, że końce nieczerwonych krawędzi sklejamy d tej pory aż pozostana tylko czerwone. Otrzymamy mniejszy graf G'. Teraz stosujemy algorytm A do grafu G'. Wynikiem jest minimlana moc zbioru X dla grafu G'.

Sprawdzanie sekwencji stopni węzłów

Ciąg niemalejący $(d_1, d_2, \ldots d_n)$ liczb naturalnych dodatnich jest sekwencją grafową gdy istnieje graf mający taki ciąg stopni węzłów (po posortowaniu).

Fakt. Załóżmy, że $\sum_i d_i$ jest liczba parzysta. Wtedy ciag $(d_1, d_2, \ldots d_n)$ jest grafowy wtedy i tylko wtedy gdy ciąg $(d_1, d_2, ..d_{n-k} - 1, d_{n-k+1} - 1, \ldots d_{n-1} - 1)$ po posortowaniu jest grafowy.

Daje to rekonstrukcję grafu (jedengo z wielu) z ciągu grafowego w czasie linowym ze względu na rozmiar grafu.

Istnieje inne kryterium sprawdzania, czy ciąg jest grafowy,

Fakt. [Erd¨s-Gallai]
Załóżmy, że $\sum_i d_i$ jest liczba parzysta oraz ciąg stopni jest posortowany nierosnąco. Wtedy ciag $(d_1, d_2, \ldots d_n)$ jest grafowy wtedy i tylko wtedy gdy

$$\forall\, k \sum_{i=1}^{k} \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Kryterium to daje łatwo algorytm liniowy sprawdzania czy ciąg jest grafowy.

Uzasadnienie. Weźmy k minimalne takie, że $d_k > d_{k+1}$ lub $k = n-1$. Zmniejszamy d_k, d_n. Nowy ciąg spełnia warunki, indukcyjnie istnieje graf. Potem przełączamy odpowiednio krawędzie dodajac po jednej krawędzi do k i n.

Kilka problemów związanych ze stpniami węzłów.

- Dla jakich n istnieje ciąg grafowy w którym wszystkie elementy są parami różne.
- Każdy graf o parzystej liczbie węzłów ma dwa węzły o parzystej liczbie wszystkich wsplnyc sąsiadów.
- Jesli graf jest spójny oraz liczba wspólnych sąsiadów dla każdej pary różnych węzłów wynosi 0 lub 5 to graf jest k-regularny dla pewnego k.

10.18 Dwuspójność grafu nieskierowanego

Operacja dzielenia krawędzi polega na wstawieniu nowego węzła w środek tej krawędzi, a operacja dodawania krawędzi plega na połączeniu krawędzią węzłów, które jeszcze nie są połączne krawędzią.

Fakt. Każdy graf dwuspójny mający $n > 2$ węzłów można otrzymac z trójkąta stosujac operacje dzielenia i tworzenia krawędzi.
Uzasadnienie. Zaczynamy od jakiegokolwiek cyklu prostego. Potem znajdujemy nieutworzony węzeł v do którego jest krawędź z węzła już utworzonego u. Usuwamy u, znajdujemy ścieżkę z v do jakiegoś węzła v' już utworzonego, pozostałe węzły tej ścieżki jeszcze nieutworzone. Dołączamy do grafu ścieżkę (ucho) od u do v a nastepnie do v'. Jak otrzymamy już wszystkie węzły to dołączamy pojedyncze orginalne krawędzie.

Fakt. Graf jest 2-spójny wtw każde dwa różne węzły leżą na pewnym cyklu prostym.

Kilka problemów związanych.

- Stopień spójności grafu H_k (hypercube k-wymiarowy) wynosi k.
- Każdy graf 14-wierzchołkowy spójny i bez cykli o długości co najwyżej 5 jest izomorficzny z grafem Heawooda. Węzły to 0..13. Węzły o numerach parzystych i łaczymy z $i + 5$ modulo 14.

11 Algorytmiczne tajemnice hiperkostki

11.1 Ścieżka na hiperkostce

Niech nasz graf $G = (V, E)$ będzie hiperkostką n-wymiarową z usuniętym k-elementowym zbiorem N wierzchołków. Chcemy sprwdzić, czy istnieje ścieżka z x y, gdzie $x, y \notin N$.

Niech $K = n \cdot k+1$ oraz niech $OgrDFS(x, y, K)$ będzie (ograniczonym) DFS, które sprawdza czy y jest osiagalne z x. Ograniczony DFS działa tak jak DFS, z tą różnicą, że jeśli odwiedzimy już K węzłów to zatrzymujemy się i zwracamy TRUE. Inaczej mówiąc: jeśli przejrzymy mniej niż K węzłów startując z x i DFS się zakończy bez znalezienia y (nie ma więcej wezlow do odwiedzenia) to OgrDFS zwraca FALSE, natomiast jeśli znajdziemy y lub odwiedzimy już K węzłów to OgrDFS się zatrzymuje i zwraca TRUE.

```
ALGORYTM
  k:= n k + 1;
  z1:= OgrDFS(x,y,K); z2:=OgrDFS(y,x,K)
  return (z1 and z2)
```

Dlaczego to dziala. Poprawność wynika z pewnej ciekawej i dosyc prostej własności grafu hiperkostki Q_n, którą nazwiemy **własnością podziałową:**

$$| \{(a,b) \in Q_n \ : \ a \in S, b \in V - S\} | \geq \min(|S|, |V - S|).$$

Przypuśćmy teraz że $OgrDFS(x, y, K)$ oraz $OgrDFS(y, x, K)$ nie wykryły ścieżki od x do y ale każde z nich odwiedziło co najmniej K węzłow. Wystarczy udowodnić, że w tej sytuacji x i y są w tej samej spójnej składowej w $Q_n - N$.

Dowód przez zaprzeczenie. Załóżmy, że x, y są w różnych składowych grafu $Q_n - N$. Bez straty ogólności niech S będzie mniejszą z tych dwóch składowych (być może jest więcej składowych). Rozważmy podział $(S, V - S)$ kostki Q_n. Zgodnie z własnością podziałową z S *wychodzi na zewnątrz* co najmniej $n \cdot k + 1$ krawędzi. Do zbioru N może wejśc co najwyżej $n \cdot k$ krawędzi, tak więc jest jakaś krawędź między S i $V - (S \cup N)$, przeczy to temu że S jest składową spójna $Q_n - N$, gdyż z definicji składowa jest *maksymalną* częścia spójną.

Udowodnimy teraz własność podziałową. Zamieńmy nasze krawędzie na krawędzie skierowane (każdą nieskierowaną zamieniamy na dwie skierowane). Ścieżki staną się skierowane. Dla każdych dwóch wezłów u, v przez ścieżkę standardową z u do v rozumiemy taką, ktora jest najkrótszej długości i odpowiada zmienianiu kolejnych niezgodnych bitów od lewej do prawej tak aby zamienić x na y. Na przykład dla $x = 0101$, $y = 0010$ ścieżka standardowa odpowiada zmienianiu kolejno bitów 2-gi, 3-ci, 4-ty.

$$0101 \to 0001 \to 0011 \to 0010$$

Zachodzi następujący fakt **(własność scieżek standardowych)**:
Przez zadaną skierowaną krawędź przechodzi dokładnie 2^{n-1} ścieżek standardowych.
Dowód tej własnosci: Rozważmy krawędź na standardowej scieżce od u do v. Jesli krawędź dotyczy zmiany na i-tym bicie, to ostatnie $n - i$ bitów u, oraz pierwsze $i - 1$ bitów węzła-napisu v są zdeterminowane przez tę krawędź. Poza tym krawędź determinuje i-ty bit węzłów u, v. Zatem pozostaje $n - 1$ możliwości na wybór niezdeterminowanych bitów węzłów u, v, co daje 2^{n-1} możliwości.

Dowód własności podziałowej. Załóżmy, że $|S| \leq |V - S|$. Niech $m = |S|$. Mamy $m \cdot (2^n - m)$, ścieżek standardowych z węzłow S do węzłów $V - S$. Każda z nich przechodzi przez dokładnie jedną krawędź między S i $V - S$. Przez jedną taką krawędź przechodzi co najwyżej 2^{n-1} ścieżek. Mamy $m \leq 2^{n-1}$. Zatem liczba krawędzi między S i $V - S$ jest co najmniej

$$m \cdot (2^n - m)/2^{n-1} \geq m \cdot 2^{n-1}/2^{n-1} = m = |S|$$

Koniec dowodu.

11.2 Przesuwanie żetonów na hiperkostce

In this problem we have token in vertices of the hypercube. We can take 2 tokens from one vertex and put one token on its neighbor. The goal is to put a token on a specified vertex. No moves are required if this specified vertex already has at least one token.

Define the operation $moves(u, v, W)$ which moves W from the vertex u to v, removing $2W$ tokens from u and more massive operation $MOVE(A, B, W)$, which for two disjoint subsets of the same size of the hypercube removes $2W$ tokens from A and places W tokens in vertices of B.

For a subset A define $odd(A)$ the number of vertices containing odd number of tokens and by $tokens(A)$ the total number of tokens in A.

Obserwacja 3.
The operation $MOVE(A, B, W)$ is possible if there is a perfect matching between A and B in the hypercube and $tokens(A) \geq 2W + odd(A)$.

We describe 2 functions, the first of them is placing (after some number of moves) a token on a vertex v, the second one places two tokens. Let X be a subhypercube and $Split(X, v)$ be the partition of X into 2 disjoint subhypercubes, v belongs to the first one.

function PlaceSingleToken(X,v)
$k := dim(X)$; // Assume $tokens(X) \geq 2^k$
if $k = 1$ or v already has a token then trivial solution and return;
$(A, B) := Split(X)$;
v' := neighbor of v in B;
if $tokens(A) \geq 2^{k-1}$ then PlaceSingleToken(A,v)
else if $odd(B) > tokens(A)$ then
PlaceTwoTokens(B,v'); move(v',v,1)
else
$MOVE(B, A, 2^{k-1} - tokens(A))$; PlaceSingleToken(A,v);

function PlaceTwoTokens(X,v)
$k := dim(X)$; // Assume $tokens(X) \geq 2^{k+1} - odd(X)$
if $k = 1$ or v already has a token then trivial solution and return;
$(A, B) := Split(X)$;
v' := neighbor of v in B;
if $tokens(A) \geq 2^k - odd(A)$ then PlaceTwoTokens(A,v)
else
if $tokens(A) < 2^{k-1}$ then $MOVE(B, A, 2^{k-1} - tokens(A))$
PlaceSingleToken(A,v); PlaceTwoTokens(B,v'); move(v',v,1)

The proof of the following theorem is by induction on the dimension of the hypercube together with Observation 3.

Twierdzenie
(a) If $tokens(Q_n) \geq 2^n$ then we can place a token on any specified vertex v.
(b) If $tokens(Q_n) > 2^{n+1} - odd(Q_n)$ then we can place 2 tokens on any specified vertex.

11.3 Dopełnianie skojarzenia do cyklu Hamiltona na hiperkostce

Rozważamy następujący problem.

Wejście: skojarzenie pełne X hiperkostki $Cube_n$;
Wyjście: cykl C Hamiltona hiperkostki $Cube_n$ taki, że $X \subseteq C$.

Niech $\mathcal{K}(n)$ będzie *grafem pełnym* o tym samym zbiorze węzłów co $Cube_n$, każde dwa różne węzły są połączone krawędzią.
Wzmocnijmy nasz problem następująco.

Wejście: pełne skojarzenie X grafu $\mathcal{K}(n)$;
Wyjście: cykl C Hamiltona grafu $\mathcal{K}(n)$ taki, że $C - X$ jest pełnym skojarzeniem grafu $Cube_n$.

Inaczej mówiąc mając pełne skojarzenie w grafie pełnym chcemy je dopełnić krawędziami z hiperkostki do cyklu Hamiltona. Dla innych grafów może to być niewykonalne. Znacznie wzmocniliśmy nasz problem, dopuszczając na wejściu dowolny zbiór rozłącznych par wierzchołków.

Jeśli C jest cyklem Hamiltona oraz Z jest podzbiorem krawędzi cyklu i jednocześnie skojarzeniem (niekoniecznie pełnym) to przez $SEKW(C, Z)$ oznaczmy listę $[(x_1, y_1), (x_2, y_2) \ldots, x_k, y_k)]$ krawędzi skojarzenia Z w kolejności i skierowaniu tak jak na cyklu (w czasie tej operacji chwilowo zamieniamy nieskierowany cykl na skierowany).

Przykład. Jeśli $C = (1, 2, 4, 3, 6, 5, 1)$, $Z = \{(5, 6), (2, 4)\}$ to

$SEKW(C, Z) = [(2, 4), (6, 5)]$.

Głównym trikiem jest sprytne dołożenie pomocniczych krawędzi w pohiperkostkach (zbiory M_1, M_2), policzenie rekurencyjnie cykli w pod-hiperkostkach, następnie usunięcie pomocniczych krawędzi i dołożenie krawędzi ze zbioru $Z \subseteq X$ łączących pod-hiperkostki.

Algorytm Oblicz-Cykl(X, H); (H jest hiperkostką n-wymiarową)

Jeśli $n = 2$ to oblicz wynik naiwnie i zatrzymaj się;

podziel hiperkostkę H na dwie hiperkostki H_1, H_2 (o wymiarach $n-1$) między którymi jest jakakolwiek krawędź ze zbioru X;

$X_1 := X \cap H_1$; $X_2 := X \cap H_2$; $Z := X - X_1 - X_2$; (Z jest łącznikiem między H_1 i H_2);

$D :=$ zbiór wierzchołków w H_1 będących końcami krawędzi z Z;

$M_1 :=$ dowolne pełne skojarzenie podgrafu $\mathcal{K}(n)$ ograniczonego do zbioru D ($X_1 \cup M_1$ jest pełnym skojarzeniem w H_1);

Rekursja: $C_1 :=$ **Oblicz-Cykl**$(X_1 \cup M_1, H_1)$;

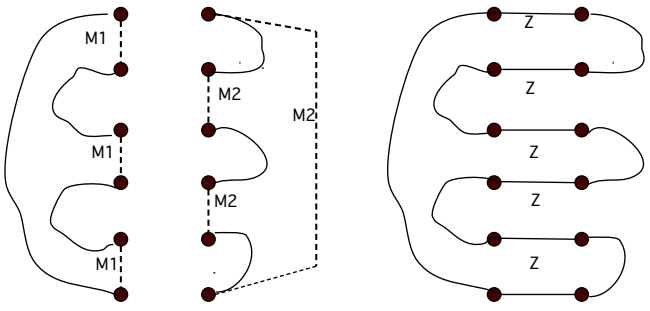
$[(x_1, y_1), (x_2, y_2) \ldots, x_k, y_k)] := SEKW(C_1, M_1)$;

niech $x'_i, y'_i \in H_2$ będą *partnerami* x_i, odpowiednio y_i, w łączniku Z;

$M_2 := \{(y'_1, x'_2), (y'_2, x'_3), (y'_3, x'_4), \ldots (y'_k, x'_1)\}$;

Rekursja: $C_2 :=$ **Oblicz-Cykl**$(X_2 \cup M_2, H_2)$;

return cykl Hamiltona $(C_1 - M_1) \cup (C_2 - M_2) \cup Z$.



Rysunek 15: Podział hiperkostki H_n na dwie hiperkostki H_1, H_2 o mniejszym wymiarze. Interpretacja graficzna częściowych sztucznie dodanych skojarzeń M_1, M_2 oraz łącznika Z: podzbioru X krawędzi pomiędzy H_1, H_2. Zauważmy że liczba tych krawędzi jest dodatnią liczbą parzystą. Algorytm znajduje rekurencyjnie dwa cykle w grafach H_1, H_2. Końcowy cykl powstaje przez usunięcie sztucznie dodanych krawędzi oraz dodanie krawędzi z łącznika.

Algorytm ma charakter rekurencyjny i korzysta z prostych rekurencyjnych własności hiperkostki n-wymiarowej, głownie z łatwego podziału kostki na dwie o mniejszym wymiarze. Złożoność czasowa algorytmu jest $O(n \log n)$, co wynika z rekurencji

$$T(n) = 2 \cdot T(n/2) + O(n).$$

Z rekurencją tego typu można się spotkać w klasycznym problemie sortowania przez scalanie (ang. *mergesort*).

Poprawność algorytmu jest dosyć oczywista (wystarczy sobie rozrysować schematycznie cykle generowane przez algorytm w pod-hiperkostkach.)

12 Uspójnianie grafu skierowanego

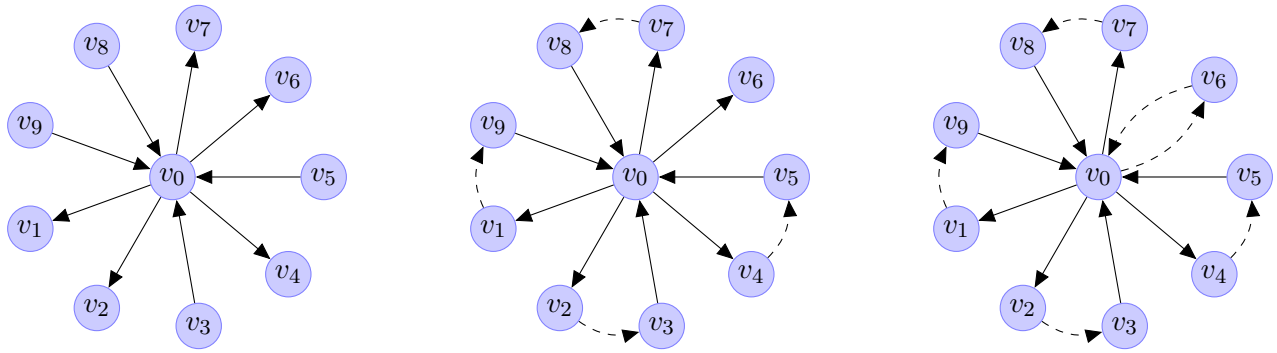
Na wejściu mamy graf skierowany $G = (V, E)$. Chcemy w czasie liniowym dodać minimalną liczbę krawędzi skierowanych tak, aby G był silnie spójny. W rozwiązaniu zredukujemy problem kolejno do grafu acyklicznego, potem do dwudzielnego, a potem do gwiazdy. Skierowany graf dwudzielny to taki, w którym V jest sumą rozłączną $V = A \cup B$, oraz $E \subseteq A \times B$. Gwiazda to graf, w którym jest jeden wierzchołek należący do wszystkich krawędzi, natomiast pozostałe końce krawędzi są parami różne. Zaczniemy opis algorytmu od końca:

1. Jeśli graf jest pojedynczą gwiazdą to rozwiązanie jest proste. Niech v_0 będzie środkiem gwiazdy. Wtedy

$$E = E_1 \cup E_2, \text{ gdzie } E_1 \subseteq \{v_0\} \times (V - \{v_0\}),\ E_2 \subseteq (V - \{v_0\}) \times \{v_0\})$$

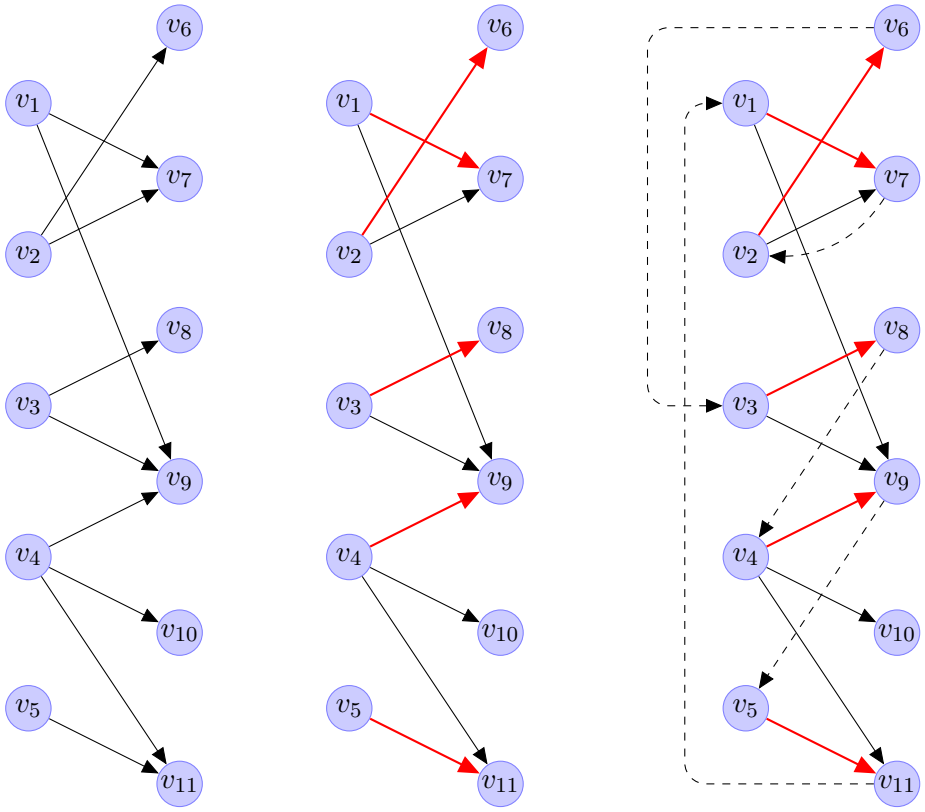
   (E_1 i E_2 to zbiory krawędzi wychodzących i wchodzących do v_0).

   Bez straty ogólności przyjmijmy, że $|E_1| \leqslant |E_2|$. Wtedy każdej krawędzi z $(v_0, a) \in E_1$ przyporządkowujemy krawędź $(b, v_0) \in E_2$ (zawsze inną) i dodajemy krawędź (a, b), tworząc w ten sposób cykl $v_0 \to a \to b \to v_0$. Dla krawędzi, których nie pogrupowaliśmy w pary dodajemy krawędzie odwrotne, tworząc cykle długości 2. W ten sposób robimy z gwiazdy graf silnie spójny za pomocą minimalnej liczby dodatkowych krawędzi.



2. Rozpatrujemy teraz grafy dwudzielne. Dla uproszczenia załóżmy w tej części, że graf nie ma odizolowanych węzłów, tzn. takich, że nie są początkiem ani końcem żadnej krawędzi. W trakcie algorytmu będziemy utrzymywać pewien roboczy zbiór wybranych krawędzi, krawędzie *wolne* to takie, które nie mają wspólnego węzła z żadną z wybranych krawędzi.

   Tworzymy *duży* cykl C w sposób zachłanny: dopóki istnieje jakakolwiek krawędź wolna, to dodajemy dowolną z nich do zbioru roboczego. W ten sposób otrzymujemy zbiór $\{(u_1, v_1), (u_2, v_2) \ldots, (u_k, v_k)\}$. Teraz dodajemy k nowych krawędzi $(v_1, u_2), (v_2, u_3), \ldots, (v_k, u_1)$ tworząc cykl C. Następnie ściągamy C do jednego węzła v_0 i otrzymujemy gwiazdę.



3. Jeśli G jest dowolnym grafem to znajdujemy silnie spójne składowe i każdą z nich *ściągamy* do pojedynczego węzła, otrzymując graf acykliczny (rys. 14). Niech A będzie zbiorem węzłów, do których nie wchodzi żadna krawędź. Analogicznie niech B oznacza zbiór węzłów, z których nie wychodzi żadna krawędź. Tworzymy nowy graf dwudzielny $G' = (A \cup B, E')$, gdzie $(u, v) \in E'$ gdy istnieje ścieżka skierowana od u do v w grafie acyklicznym.

Fakt. Problem minimalnego uspójnienia grafu skierowanego można rozwiązać w czasie liniowym. Wynika to z rozwiązywalności problemu podziału grafu na silnie spójne składowe w czasie liniowym (rozdział **??**).

Problem ten razem z rozwiązaniem został też przystępnie (choć trochę inaczej) opisany w artykule J.Radoszewskiego w numerze 09/2012 czasopisma Delta.

Minimalne drzewo skierowane.

Załóżmy, że mamy nieujemne wagi (koszty) $waga(i, j)$ potencjalnych skierowanych krawędzi $i \to j$ drzewa i chcemy skonstruowac drzewo ukorzenione o najmniejszej sumarycznej wadze. Zbiorem wierzchołów drzewa jest $V = \{1, 2, ..n\}$, oraz $r \in V$ jest ustalonym korzeniem. Formalne pisząc, chcemy zminimalizować wartość

$$MIN(v) \ = \sum_{i \neq r} waga(i, Parent(i)),$$

gdzie $Parent$ to poprzednik (ojciec) wierzchołka.

Zaczynamy of prostego podejścia zachłannego:

dla każdego $i \in V$, $i \neq r$ wybieramy najlżejszą krawędź wychodzącą z i.
Jeśli nie wygenerowaliśmy cyklu to orzymaliśmy minimalne drzewo skierowane.

Co się dzieje gdy wygenerowaliśmy cykl ($Parent$ generuje cykl, dla pewnego elementu $Parent^k(i) = i, k > 0$), którego zbiorem elementów jest zbiór $C \subseteq V - \{r\}$? Pozornie wydaje się, że nasze zachłanne podejście nie ma sensu. Zauważmy jednak (dowód w ćwiczeniach), że zbiór C ma bardzo pożyteczną własność:

($*$) Istnieje optymalne drzewo w ktorym z C wychodzi tylko jedna krawędź.

Możemy zatem potraktować zbiór C jako jeden *superwierzchołek*. Definiujemy:

$$MinOut(i) := \min\{waga(i,j) \ : \ j \in V\}.$$

Teraz możemy napisać co robimy gdy jest cykl C.

Uruchamiamy funkcję $CONTRACT$ która zamienia cykl na *superwierzchołek*. Funkcja ta działa następująco:

funkcja CONTRACT(C):
$v^* :=$ nowy element; $V := V - C \cup \{v^*\}$;
for each $k \in V$, $k \neq v^*$ **do**
  $waga(k, v^*) := \min\{waga(k,j) \ : \ j \in C\}$;
  $waga(v^*, k) := min\{waga(i,k) - MinOut(i) \ : \ i \in C\}$

Krawędzie cyklu C będa użyte w optymalnym drzewie poza dokładnie jedną krawędzią. Jeśli tą krawędzią jest $i \to k$ to wymieniając krawędź wychodzącą z i do elementu w cyklu (o wadze $MinOut(i)$) na krawędź $i \to$ zwiększamy całkowity koszt o wartość $waga(i,k) - MinOut(i)$. Stąd się bierze operacja definiująca wagi krawędzi wychodzących z superwierzchołka v^*.

Zdefiniujmy jeszcze $waga(C)$ jako sumę wag krawędzi skierowanych w cyklu C. Cały algorytm ma charakter rekurencyjny:

Funkcja Min-Greedy(V)
for each $i \neq r$ **do**
  $Parent(i) := j$, gdzie $waga(i,j) = MinOut(i)$;
if funkcja $Parent$ nie generuje cyklu **then**
  return $\sum_{i \neq r} waga(i, Parent(i))$
  else
    $C :=$ jakiś cykl generowany przez $Parent$;
    $CONTRACT(C)$;
    return $waga(C)$ + Min-Greedy(V)

Zauważmy, że policzyliśmy tylko minimalny koszt, tablicę $Parent$ można wyłuskać z algorytmu, wymaga to nieznacznej komplikacji. W tej wersji funkcja $Parent$ jest liczona lokalnie w każdej instancji i nie jest poprawna globalnie.

13 Generacja ciągów Lyndona i ciągów de Bruijna

Uwaga: dla uproszczenia rozważamy tylko teksty binarne.

Słowem (ciągiem) **de Bruijna** rzędu n jest ciąg binarny o długości 2^n, w którym (traktowanym jako ciąg cykliczny) każdy ciąg binarny długości n występuje dokładnie raz.

Słowa Lyndona są zwartymi reprezentacjami liniowymi słów cyklicznych. Dla słowa x niech y będzie minimalnym cyklicznym przesunięciem x. Wtedy pierwiastek pierwotny z słowa y jest słowem Lyndona. Słowo jest ciągiem **Lyndona** wtedy i tylko wtedy, gdy może powstać w ten sposób. Przypomnienie: pierwiastek pierwotny y to najkrótszy prefiks z słowa y taki, że y jest naturalną potęgą z.

Definicja równoważna Słowo jest Lyndona jeśli jest leksykograficznie najmniejsze ze swoich przesunięć cyklicznych (równoważnie, najmniejsze ze swoich sufiksów).

Dla danego n przez $ext(x,\ n)$ oznaczmy rozszerzenie okresowe słowa x do długości n, oraz przez $LastZero(x)$ oznaczamy najdłuższy prefiks słowa x kończący się zerem. Na przykład:

$$ext(00111,\ 13) = 00111\ 00111\ 001, \quad LastZero(0010111) = 0010.$$

Następujący algorytm generuje wszytkie słowa Lyndona o długości co najwyżej n:

Algorytm 14: Fredricksen-Maiorana

```
1 x := ''0'';
2 Wypisz słowo Lyndona x;
3 while x ≠ ''1'' do
4 |   x := LastZero(ext(x, n));
5 |   Zamień ostatni symbol x na jedynkę;
6 |   Wypisz słowo Lyndona x;
7 end
```

Niech $L_0 < L_1 < L_2 < \ldots < L_s$ będzie leksykograficznie posortowaną sekwencją wszystkich binarnych słów Lyndona o długości będącej dzielnikiem n. Niech $\mathcal{L}_n$ oznacza konkatenację

$$\mathcal{L}_n \;=\; L_0 \cdot L_1 \cdot L_2 \cdot L_3 \cdot \ldots \cdot L_s$$

Przykład Dla $n = 4$ algorytm FM wygeneruje:

$$0 \quad 0001 \quad 001 \quad 0011 \quad 01 \quad 011 \quad 0111 \quad 1$$

$$\mathcal{L}_4 = 0 \quad 0001 \quad 0011 \quad 01 \quad 0111 \quad 1$$

Powiemy, że L_k jest „małe" gdy $|L_k| < n$, w przeciwnym przypadku jest „duże". Z poprawności algorytmu FM (Fredricksena-Maiorany) wynikają własności:

1. $L_0 = 0,\ L_1 = 0^{n-1}1,\ L_{s-1} = 01^{n-1},\ L_s = 1$;
2. jeśli $L_k = \beta\alpha$ oraz α zawiera zero, to β jest prefiksem L_{k+1}
3. Jeśli L_k jest małe i $k > 0$ to
   - L_{k-1} jest duże;
   - L_{k-1} kończy się co najmniej $n - |L_k|$ jedynkami;
   - L_{k-1} jest bezpośrednio wygenerowane przed L_k w algorytmie FM.

Twierdzenie 6 (Fredricksen-Maiorana)**.** *Przypadek szczególny – rozgrzewka: jeśli n jest liczbą pierwszą, to $\mathcal{L}_n$ zawiera (cyklicznie) każde binarne słowo x długości n.*

Dowód. Przeprowadźmy dowód rozpatrując kilka przypadków.

Przypadek 1. Niech $x \in 1^*0^*$. Wtedy x jest podsłowem $L_{s-1}L_sL_0L_1 = 01^n0^n1$. Załóżmy zatem (do końca dowodu), że nie zachodzi przypadek 1. Słowo x jest cyklicznie równoważne pewnemu słowu L_r (równemu minimalnemu cyklicznemu przesunięciu x). Wtedy dla pewnych α, β

$$x = \alpha\beta, \quad L_r = \beta\alpha$$

Ustalmy do końca dowodu α i β.

Przypadek 2. α zawiera zero. Wtedy β jest prefiksem L_{r+1}, zatem x jest podsłowem L_rL_{r+1}, którego prefiksem jest $\beta\alpha\beta$.

Przypadek 3. $\alpha \in 1^+$. Załóżmy, że nie zachodzi przypadek 2. Wtedy $\beta \notin 0^+$. Istnieje zatem taki indeks k, że β jest prefiksem L_k, ale β nie jest prefiksem L_{k-1}. Niech γ będzie prefiksem L_{k-1} o długości β. Zapiszmy $L_{k-1} = \gamma\delta$. Udowodnimy:

Fakt. $\delta \in 1^+$. Dowód nie wprost. Przypuśćmy, że δ zawiera 0, wtedy zgodnie z algorytmem Fredricksena-Maiorany następne leksykograficznie słowo Lyndona ma prefiks γ. Wiemy, że $\beta \neq \gamma$. Natomiast następnym słowem z definicji jest L_k, które ma prefiks $\beta \neq \gamma$, o tej samej długości co γ. Sprzeczność.

Z powyższego faktu wynika, że $\delta = \alpha$, ponieważ są to słowa tej samej długości składające się z samych jedynek. Zatem x jest podsłowem $L_{k-1}L_k$ jako $\delta\beta$, w konsekwencji x jest podsłowem całego słowa $\mathcal{L}_n$. □

Twierdzenie 7 (Fredricksen-Maiorana)**.** *Przypadek ogólny, dowolne n. $\mathcal{L}_n$ zawiera (jako słowo cykliczne) każde binarne słowo x długości n.*

Dowód. Ponownie rozpatrujemy przypadki.

Przypadek 1: $x \in 1^*0^*$. Dowód bez zmian (w stosunku do dowodu przypadku szczególnego). Załóżmy, zatem (do końca dowodu), że nie zachodzi przypadek 1. **W przypadkach 2-4 zakładamy, że słowo x jest pierwotne.** Zakładamy również, że x nie jest równe żadnemu L_r. Wtedy x jest cyklicznie równoważne pewnemu słowu L_r (równemu minimalnemu cyklicznemu przesunięciu x). Wtedy dla pewnych niepustych α, β

$$x = \alpha\beta,\ L_r = \beta\alpha$$

Niech L_k będzie leksykograficznie pierwszym dużym słowem o prefiksie β.

Przypadek 2: $\alpha \notin 1^+$. Dowód bez zmian.

Przypadek 3: $\alpha \in 1^+$, L_{k-1} jest duże. Dowód bez zmian.

Przypadek 4: $\alpha \in 1^+$, L_{k-1} jest małe.

Rozważamy podprzypadki **A-C**:

(A) $|\beta| < |L_{k-1}|$. Wtedy L_{k-2} jest dużym słowem o prefiksie β co przeczy temu, że L_k jest najwcześniejsze. Zatem przypadek niemożliwy.

(B) L_{k-1} kończy się co najmniej $|\alpha|$ jedynkami i $\alpha\beta = x$ jest podsłowem $L_{k-1}L_k$.

(C) L_{k-1} kończy się mniej niż $|\alpha|$ jedynkami. Z definicji operacji okresowego rozszerzania wynika, że L_{k-1} jest okresem β. Jednocześnie L_{k-2} (duże słowo) kończy się co najmniej $n-|L_{k-1}| \geqslant |\alpha|$ jedynkami (gdyż $|\beta| \geqslant |L_{k-1}|$ oraz $|\alpha| = n - |\beta|$). Zatem $\alpha\beta = x$ jest podsłowem $L_{k-2}L_{k-1}L_k$.

Przypadek 5: Słowo x nie jest pierwotne. Wtedy dla pewnych $k > 1$ oraz r, α, β mamy $x = (\alpha\beta)^k$, $L_r = \beta\alpha$. Jeśli $\alpha \notin 1^+$, to ponieważ słowo L_r jest małe i rozszerzenie okresowe zostaje zaburzone dopiero w ostatnim α to L_{r+1} jest duże i ma prefiks $(\beta\alpha)^{k-1}\beta$. Zatem x jest podsłowem L_rL_{r+1}.

Jeśli $\alpha \in 1^+$ to L_{r-1} kończy się na α (ma dostatecznie dużo jedynek), a ponieważ (z rozszerzenia okresowego) L_{r+1} ma prefiks $(\beta\alpha)^{k-1}$ to x jest podsłowem $L_{r-1}L_rL_{r+1}$. □

Kilka wzorów. Niech zapisy $Lyn(n)$, $Pierw(n)$ oznaczają liczbę binarnych słów Lyndona oraz liczbę słów pierwotnych (nierozkładalnych) długości n. Niech μ będzie funkcją Möbiusa; spełnia ona wzór rekurencyjny:

$$\sum_{d|n} \mu(d) = [n = 1].$$

Niech ϕ będzie funkcją Eulera (ile jest liczb mniejszych od n względnie pierwszych z n). Przyjmujemy $\phi(1) = 1$. Użytecznym narzędziem kombinatorycznym jest formuła Möbiusa:

$$\forall_n\ f(n) = \sum_{d|n} g(d) \Rightarrow \forall_n\ g(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) f(d)$$

Mamy też wzory:

$$2^n = \sum_{d|n} Pierw(d), \quad n = \sum_{d|n} \phi(d).$$

Z formuły inwersyjnej Möbiusa i powyższego wzoru wynikają wzory:

$$Pierw(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) 2^d, \quad Lyn(n) = \frac{1}{n}\sum_{d|n} \mu\left(\frac{n}{d}\right) 2^d$$

Jeśli $\mathcal{L}_n = L_0 \cdot L_1 \cdot L_2 \cdot L_3 \dots L_s$ jest rozkładem na słowa Lyndona o długości dzielącej n to oznaczmy $\|\mathcal{L}_n\| = s+1$. Inaczej mówiąc $\|\mathcal{L}_n\|$ jest liczbą słów długości n cyklicznie nierównoważnych (liczba naszyjników binarnych z dokładnością do obrotu). Korzystając z poprzednich wzorów można udowodnić, że:

$$\|\mathcal{L}_n\| = \frac{1}{n} \sum_{d|n} \phi\left(\frac{n}{d}\right) 2^d.$$

Na przykład:

$$Lyn(6) = 9,\ Lyn(3) = 2,\ Lyn(2) = 1,\ Lyn(1) = 2 \tag{7}$$

$$|\mathcal{L}_6| = Lyn(1) + Lyn(2) + Lyn(3) + Lyn(6) = 14 \tag{8}$$

$$|\mathcal{L}_6| = \frac{1}{6}\left(\phi(1)\cdot 2^6 + \phi(2)\cdot 2^3 + \phi(3)\cdot 2^2 + \phi(6)\cdot 2^1\right) = \frac{1}{6}\left(1\cdot 2^6 + 1\cdot 2^3 + 2\cdot 2^2 + 2\cdot 2^1\right) \tag{9}$$

$$\tag{10}$$

Słuszność tych wzorów można prześledzić na przykładzie:

$$\mathcal{L}_6 = 0\ 000001\ 000011\ 000101\ 000111\ 001\ 001011 \tag{11}$$

$$001101\ 001111\ 01\ 010111\ 011\ 011111\ 1 \tag{12}$$

14 Grafy de Bruijna

Ciąg de Bruijna rzędu n nad alfabetem binarnym (dla uproszczenia) można też wygenerować korzystając z pewnej klasy grafów. Zbudujmy graf (tzw. graf de Bruijna rzędu n), którego wierzchołki etykietujemy wszystkimi możliwymi słowami binarnymi długości n. Krawędzie etykietujemy symbolami z alfabetu i prowadzimy je według następującej reguły:

- Weź wierzchołek opisany n-znakową etykietą $\alpha_1\alpha_2\dots\alpha_n$.
- Poprowadź krawędź etykietowaną 0 do wierzchołka $\alpha_2\dots\alpha_n 0$.
- Poprowadź krawędź etykietowaną 1 do wierzchołka $\alpha_2\dots\alpha_n 1$.

Innymi słowy symbol na krawędzi jest dodawany od prawej strony do słowa reprezentowanego przez bieżący wierzchołek, spychając jednocześnie skrajnie lewy znak.

Zatem przejście pewną sekwencją krawędzi

$$v_1 \xrightarrow{c_1} v_2 \xrightarrow{c_2} v_3 \dots v_k$$

możemy utożsamić z wygenerowaniem ciągu znaków $\alpha_1\alpha_2\dots\alpha_n c_1 c_2 c_3 \dots c_{k-1}$, gdzie oczywiście słowo $\alpha_1\dots\alpha_n$ stanowi etykietę wierzchołka v_1.

Takie rozumowanie prowadzi wprost do rozwiązania problemu: w zbudowanym grafie odnajdujemy cykl Hamiltona, z którego bezpośrednio otrzymujemy ciąg de Bruijna. Ponieważ przejście cyklem Hamiltona odwiedzi każdy wierzchołek grafu, więc wygenerowany w ten sposób ciąg będzie zawierał jako podsłowo każde binarne słowo długości n. Co prawda nie interesuje nas zawieranie każdego podsłowa w sensie dosłownym, lecz w sensie cykliczności ciągów de Bruijna, ale oznacza to, że w wynikowym ciągu wystarczy wyciąć n pierwszych albo ostatnich znaków. Przykład dla $n = 3$ można zobaczyć na rysunku 16.

Niestety problem znajdowania cyklu Hamiltona jest NP-zupełny. Zauważmy jednak dwie istotne własności grafów de Bruijna:

1. Grafy de Bruijna posiadają cykl Eulera – do każdego wierzchołka wchodzi tyle samo krawędzi ile wychodzi z niego.
2. Weźmy graf de Bruijna rzędu $n-1$ i skonstruujmy dla niego tzw. *graf krawędziowy* (szczegóły poniżej). Otrzymamy w ten sposób graf de Bruijna rzędu n.

Graf krawędziowy dowolnego grafu G, to taki graf G', w którym wierzchołkami są krawędzie grafu G, natomiast krawędziami jest relacja sąsiedztwa krawędzi w grafie G. Tzn. jeśli dwie krawędzie w grafie G mają wspólny wierzchołek, to odpowiadające tym krawędziom wierzchołki w grafie G' połączone są krawędzią.

Można łatwo pokazać, że cykl Eulera w pewnym grafie G ma jednoznaczne przełożenie na cykl Hamiltona w jego grafie krawędziowym G'. Z tego wynika, że możemy odnaleźć ciąg de Bruijna używając liniowego algorytmu znajdującego cykl Eulera w grafie rzędu mniejszego o jeden. Zatem dla $n = 3$ operujemy na grafie de Bruijna rzędu 2. Przykłady kilku kolejnych grafów podajemy poniżej.

14.1 Zastosowanie ciągów de Bruijna do konstrukcji słów o dużej liczbie podsłów.

Niech $MaxSub(n)$ oznacza maksymalną liczbę różnych podsłów w słowie binarnym długości n. Zachodzi:

Fakt.

$$\forall_{1\leqslant j\leqslant n}\ MaxSub(n) \leqslant 2^{j+1} - 1 + \binom{n-j+1}{2}$$

Dowód. Mamy co nawyżej $2^{j+1}-1$ podsłów binarnych długości co najwyżej j oraz co najwyżej $\binom{n-j+1}{2}$ podsłów długości co najmniej $j+1$. □

Wystarczy teraz znaleźć takie j dla którego zachodzi równość. Oznaczmy $\gamma_k = 2^k+k-1$. Zachodzi następujący fakt:

Fakt. *Jeśli $\gamma_k < n \leqslant \gamma_{k+1}$ to $MaxSub(n) \geqslant 2^{k+1} - 1 + \binom{n-k+1}{2}$. Słowo osiągające maksymalną liczbę podsłów można skonstruować w czasie liniowym.*

Dowód. Skonstruujemy słowo spełniające warunki:

- wszystkie podsłowa długości $k+1$ w π są parami różne (w konsekwencji wszystkie podsłowa o większej lub równej długości też są różne, jest ich $\binom{n-k+1}{2}$),
- π zawiera wszystkie (czyli 2^k) podsłowa binarne długości k (w konsekwencji wszystkie krótsze lub o tej samej długości słowa binarne, jest ich w sumie $2^{k+1}-1$).

Wtedy słowo π jest słowem długości n o maksymalnej liczbie podsłów, wynika to z poprzedniego faktu. □

Słowa spełniające powyższe warunki warunki nazywamy **gęstymi**.

Konstrukcja słowa gęstego Weźmy graf de Bruijna G rzędu k – wierzchołkami są wszystkie słowa binarne długości k. Znajdujemy w nim ciąg

$$\pi = a_1 a_2 \dots a_{n-k}$$

będący ciągiem etykiet krawędzi pewnego cyklu spełniającego warunki:

(A) cykl ten ma dokładnie $n-k$ krawędzi w G,

(B) przechodzi przez każdą krawędź co najwyżej raz,

(C) przechodzi przez każdy wierzchołek G co namniej jeden raz.

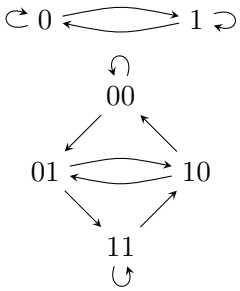
Fakt (Ciekawy fakt teoriografowy)**.** *Graf de Bruijna rzędu k posiada cykl π spełniający warunki **(A-C)** dla każdego $\gamma_k < n \leqslant \gamma_{k+1}$.*

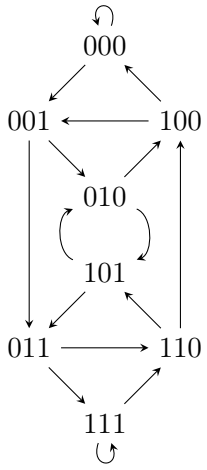
Uzasadnienie. Pokrycie cyklami prostymi to zbiór rozłącznych cykli prostych zawierających wszystkie węzły grafu.

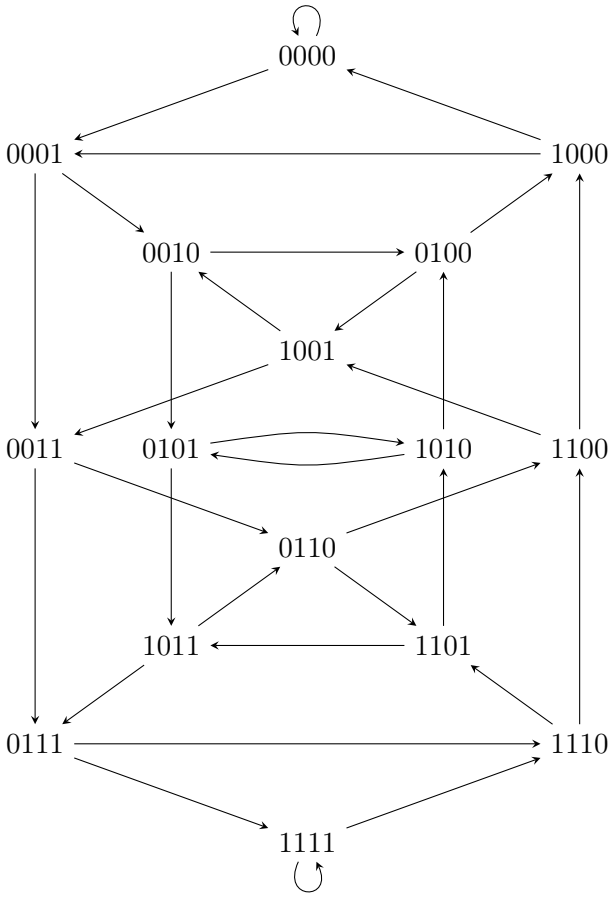
Pokażemy najpierw że jeśli C jest cyklem w grafie G_n to istnieje pokrycie cyklami zawierające C które można skonstruować w czasie liniwym.

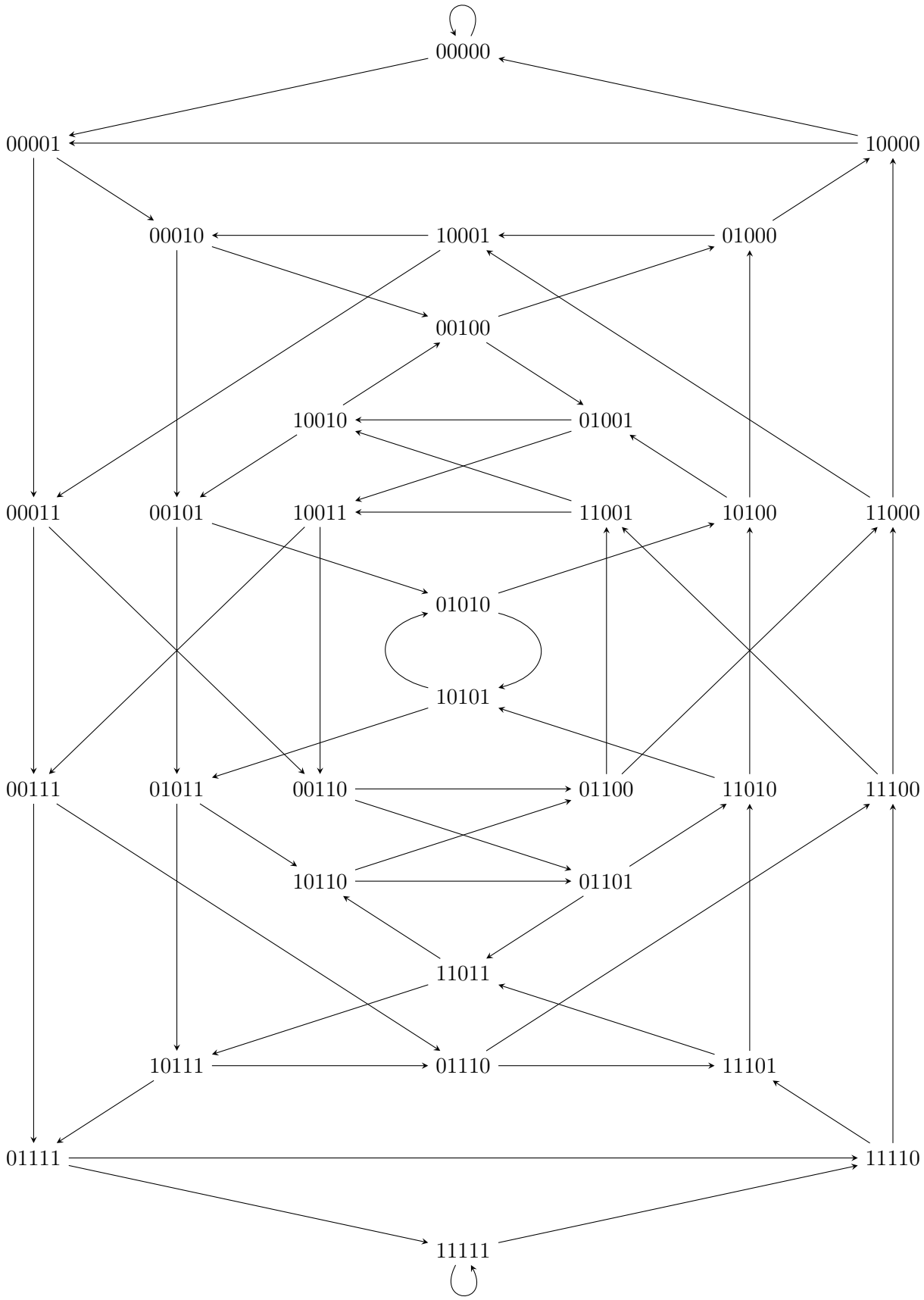
W grafie G_{n-1} jest częściowym cyklem Eulera C', Węzły C odpowiadają krawędziom C'. Usuwamy w G_{n-1} krawęzie C'. Otzrymany graf będzie pewną liczbą składowych Eulerowskich silnie spójnych.

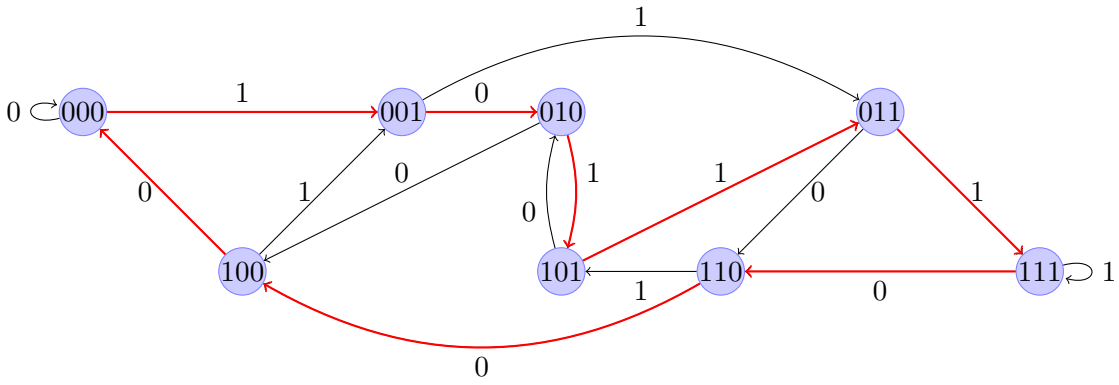
Grafy de Bruijna rzędu 1, 2, 3, 4, 5








00000
00001
10000
00010
10001
01000
00100
10010
01001
00011
00101
10011
11001
10100
11000
01010
10101
00111
01011
00110
01100
11010
11100
10110
01101
11011
10111
01110
11101
01111
11110
11111



Rysunek 16: Graf de Bruijna dla $n = 3$ nad alfabetem binarnym. Zaznaczono czerwonym kolorem cykl Hamiltona, który (rozpoczęty od wierzchołka o etykiecie 000) generuje ciąg `00010111000`. Po odcięciu n ostatnich znaków mamy `00010111`.

Weźmy cykle Eulera w każdej z nich. Te cykle, po przetłumaczeniu krawędzi na węzły w grafie G_n razem z C dają szukane pokrycie.

Pokażemy teraz jak mając pokrycie C znaleźć pokrycie rozłączne krawędziow z C. Zanjdujemy pokrycie P' zawierające C. Potem usuwamy krawędzie P' i znajdujem pokrycie P które jest automatycznie rozłączne krawędziowo z C. Po usnięciu krawędzi C każdy węzeł sspełnia $indeg(v) = oudeg(v) > 0$. Pokrycie P znajdujemy zachłannie.

Teraz bierzemy graf złożony z cyklu C i jego krawędzi oraz z krawędzi pokrycia P. Zamieniamy graf $P \cup C$ na częściowy cykl Eulera spełniający warunki **(A-C)**.

Wprowadzamy operację scalania cykli, jeśli z jakiegoś wierzchołka u cyklu C_1 jest krawędź do wierzchołka v innego cyklu C_2 te cykle można połączyć w jeden cykl o tym samym zbiorze węzłów stosując przekierowanie krawędzi. Pozostawiamy to jako ćwiczenie.

W ten sposób scalamy cykle po kolei otrzymując jeden częściowy cykl Eulera spełniający warunki **(A-C)**.

Mając częściowy cykl Eulera niech π będzie ciągiem etykiet jego krawędzi. Uliniawiamy π dodając na końcu k początkowych liter π, czyli słowo $a_1a_2 \dots a_k$. Słowo π ma teraz własności **(A-B)** i jest binarnym słowem długości n maksymalizującym liczbę podsłów.

Przykład. Dla $n = 14$ mamy $k = 3$, zaczynamy w wierzchołku (000) w grafie rzędu 3, a następnie konstruujemy cykl spełniający warunki **(A-C)** o długości $n - k = 11$. Ciąg etykiet krawędzi cyklu to (patrz rysunek grafu):

$$\pi = 1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,0\,0\,0$$

Dodajemy na końcu 3 początkowe litery i otrzymujemy wynik:

$$\pi = 1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,0\,0\,0\,000$$

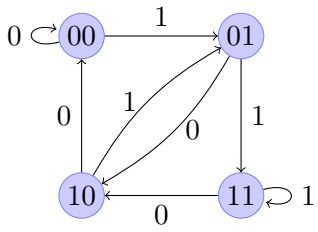
Ciąg ten jest binarnym słowem długości 14 o maksymalnej liczbie podsłów wynoszącej

$$2^{k+1} - 1 + \binom{n-k+1}{2} = 2^4 - 1 + \binom{12}{2} = 15 + 6 \cdot 11 = 81.$$

Słowa, których każdy prefiks jest gęsty w danym alfabecie nazywamy **super-gęstymi**. Zachodzi ciekawy fakt.

(a) Najdłuższym binarnym słowem super-gęstym jest słowo o długości 9.

(b) Jeśli dany alfabet ma więcej niż dwie litery to istnieje nieskończnoe słowo super-gęste. Dla danego n możemy skonstruować słowo super-gęste długości n w czasie $O(n)$.



Rysunek 17: Graf de Bruijna rzędu 2. Cyklowi Hamiltona z poprzedniego rysunku odpowiada cykl Eulera (00) – (00) – (01) – (10) – (01) – (11) – (11) – (10) – (00).

Przykład. W alfabecie 3 literowym słowem super-gęstym jest liniowe słowo de Bruijna 012, rozszerazmy je do słowa de Bruijna 012200211 0. Możemy to słowo rozszerzyć do liniowego słowa de Bruijna 012200211 000101112022212102 01 itd. Ponieważ poprzednie słowa są super-gęste to z własności ciągu de Bruijna wynika, że nastepne też są. Nie jest oczywistym, że zawsze tak możemy roszerać, ale jest to prawdziwe (korzystamy z kryteriów istnienia cyklu Eulera).

Redukcja problemu rzędu n do problemu rzędu $n-1$

Pokażemy na przykładzie obliczania ciągów de Bruijna w jaki sposób skomplikowany problem sprowadzamy do jednej instancji tego samego problemu dla danych mniejszego rozmiaru.

Niech $SubC(x)$, $SubC_k(x)$ oznaczają odpowiednio wszystkie podsłowa (wszystkie długości k) słowa xx. Inaczej mówiąc słowo x traktujemy jako ciąg cykliczny oraz podsłowa jako fragmenty cyklu.

Obserwacja 4. *Słowo binarne x jest (cyklicznym) ciągiem de Bruijna rzędu n gdy*

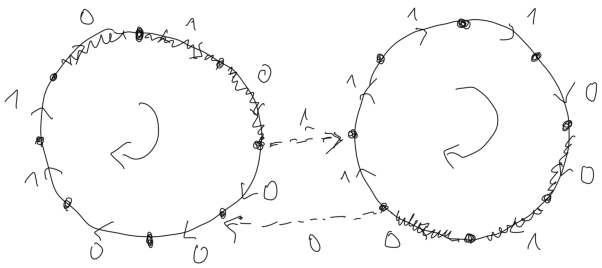
$$|x| = 2^n \ \& \ |SubC_n(x)| = 2^n.$$

Oznaczmy przez $DB(n)$ zbiór słow cyklicznych de Bruijna rzędu n. Oznaczmy jeszcze przez $Sync(x, \gamma)$ przesunięcie cykliczne ciągu x tak, aby na końcu było słowo γ, o ile $\gamma \in SubC(x)$.

Przykład. $Sync(001011101, 011) = 101001011$.

Dla dwóch słów binarnych u, v wprowadzamy operację zsynchronizowanego scalania tych słów za pomocą dodatkowego *synchronizującego* słowa $\gamma \in SubC(u) \cap SubC(v)$:

$$SyncMerge(u, v, \gamma) = Sync(u, \gamma) \cdot Sync(v, \gamma).$$



Rysunek 18: Graficzna ilutracja synchronicznego scalania $SyncMerge(u, v, 0101)$ dla słów cyklicznych $u = 00011010$, $v = 11110010$. Słowem sychronizującym jest $\gamma = 010$ (oba slowa kończą się na γ, tak więc są od razu zsynchronizowane). Wynikiem jest słowo cykliczne 0001101011110010. Każde słowo binarne długości 4 występowało (cyklicznie) dokładnie raz w rozłącznym zestawie słów cyklicznych u, v. Teraz występuje dokładnie raz w ich scaleniu.

Nastepna obserwacja wynika bezpośrednio z definicji operacji $SyncMerge$), patrz Rysunek 18.

Obserwacja 5. *Niech $n \geq 2$. Załóżmy, że dla słów binarnych u, v długości 2^n i słowa γ długości n zachodzi warunek:*
$(*)$ $|SubC_{n+1}(u) \cup SubC_{n+1}(v)| = 2^{n+1}$ & $\gamma \in SubC_n(u) \cap SubC_n(v)$.
Wtedy $w = SyncMerge(u, v, \gamma) \in DB(n+1)$, inaczej mówiąc w jest słowem de Bruijna rzędu $n+1$.

Dla ciągu binarnego w zdefiniujmy dziwną operację, której wynikiem jest następujący ciąg $\Psi(w)$ tej samej długości:

$$(\forall 1 \leq i \leq |w|)\ \Psi(w)[i] = (\sum_{j=1}^{i} w[j]) \bmod 2.$$

Przykład. $\Psi(11100101) = 10111001$.

Obserwacja. Możliwa jest sytuacja $neg(\Psi(x)) \neq \Psi(neg(x))$.

Niech α_n będzie słowem binarnym długości n zaczynającym się od zera i nie mających dwóch takich samych kolejnych liter. Dla ciągu binarnego z operacja $neg(z)$ polega na zanegowaniu kolejnych liter. Następna obserwacja wynika z definicji operacji Ψ:

Obserwacja 6.
Niech $x \in DB(n)$, $n \geq 2$. Wtedy warunek $(*)$ jest spełniony dla $X = \Psi(x)$ oraz $Y = neg(\Psi(x))$ i dowolnego $\gamma \in SubC_n(X) \cap SubC_n(Y)$. W szczególności mozna wziąć $\gamma = \alpha_n$

Powyższy fakt jest podstawą następującego algorytmu typu *redukcja problemu* konstrukcji ciągów de Bruijna (pewnego typu, jest wiele innych ciągów de Bruijna). Konstrukcję problemu dla n redukujemy do *jednej* instancji problemu rzędu $n-1$.

Algorytm DeBruijn(n) {Zakładamy, że $n \geq 2$}
if $n = 2$ **return** 0011;
$X := \Psi(DeBruijn(n-1))$; $Y := neg(X)$;
wybierz dowolne słowo $\gamma \in SubC_n(X) \cap SubC_n(Y)$.;
{ γ jest słowem synchronizującym długości n }
return $SyncMerge(X, Y, \gamma)$

Algorytm ten działa w czasie liniowym ze względu na rozmiar wyjścia.

14.2 Generowanie on-line binarnego słowa de Bruijna

Pokażemy jeszcze jedno zastosowanie porządku w alfabecie. Następujący algorytm generuje kolejne fragmenty n-elementowe ciągu liniowego de Bruijna rzędu n, jest to algorytm który, w odróżnieniu od algorytmu rekurencyjnego używa pamięcie $O(n)$, generując on-line symbole ciągu o długości 2^n.

Zakładamy, że alfanetem jest $\{0, 1\}$, oraz $\bar{b}$ oznacza negację bitu.

Jeśli $b_1 b_2 .. b_n$ jest aktualnym fragmentem to następny fragment otrzymujemy obcinając pierwszą literę b_1 i dopisując na końcu literę b. Litera b zależy od aktualnego ciągu w sposób następujący.

Algorytm On-Line generacji ciągu de Bruijna
przejście do kolejnej n-krotki: $b_1 b_2 \ldots b_n \to b_2 b_3 \ldots b_n b$
if $b_2 b_3 \ldots b_n 1$ jest ciągiem minimalnym w swojej klasie cykliczności **then** $b := \overline{b_1}$
else $b := b_1$;

Aby udowodnić poprawność wystarczy zinterpretować ten algorytm jako algorytm przechodzenia drzewa cykli rotacyjnych, patrz rysunek [2]. Cykle połączone są *dwukierunkowymi mostami.*

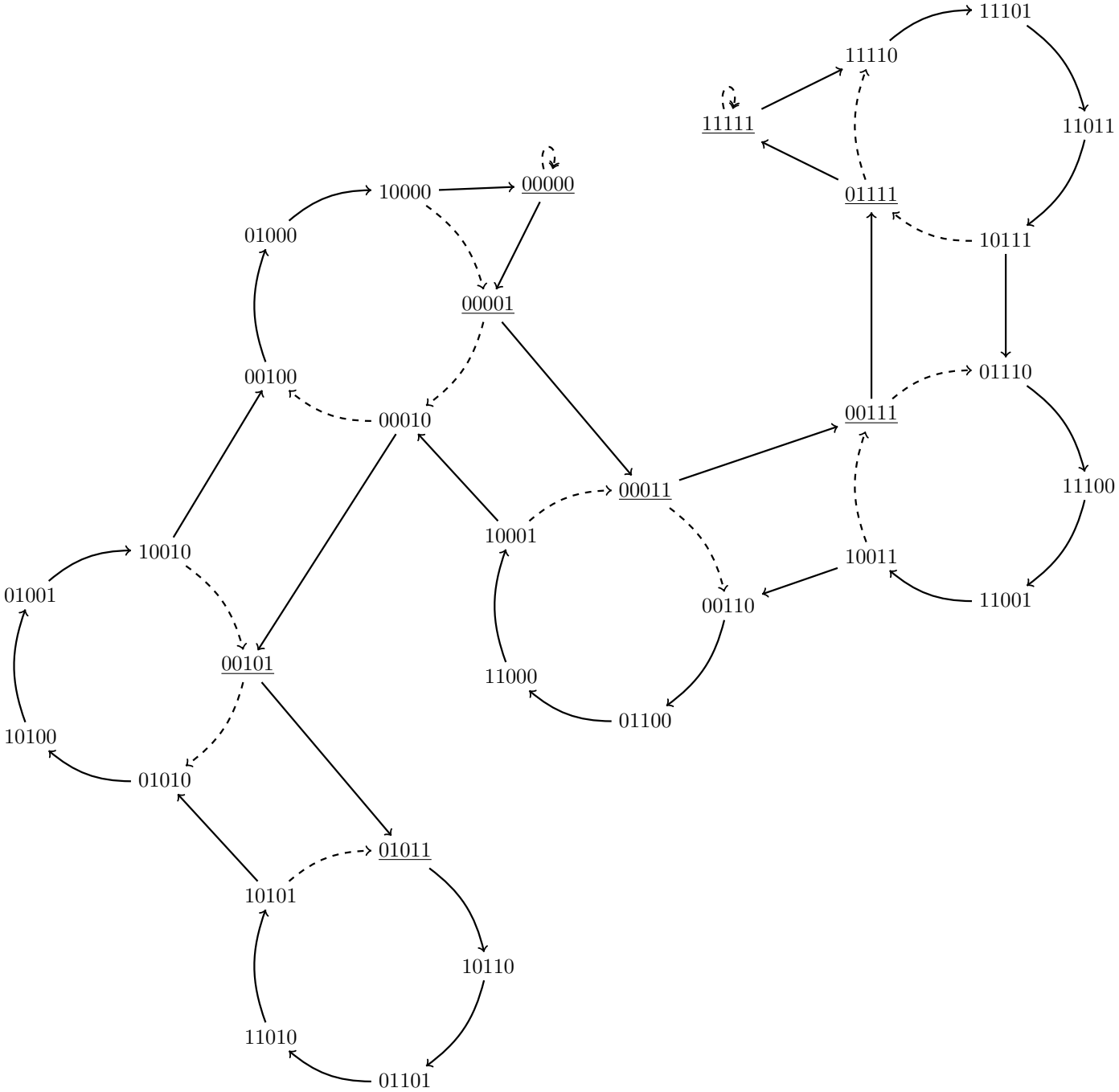
Drzewo cykli rotacyjnych jest podgrafem grafu de Bruijna $G_n = (V, E)$, gdzie $V = \{0, 1\}^n$ oraz mamy krawędź $a\alpha \to \alpha b$ dla każdego

$$a, b \in \{0, 1\},\ \alpha \in \{0, 1\}^{n-1}$$

Zachodzi następujący oczywisty fakt:

[2] Rysunek wykonany przez Bartosza Łukasiewicza

Obserwacja 7.
(1) *Każde słowo cykliczne w de Bruijna rzędu n odpowiada ciągowi etykiet krawędzi w cyklu Hamiltona grafu G_n, startując z sufiksu słowa w długości n.*
(2) *Każde słowo cykliczne w de Bruijna rzędu n odpowiada ciągowi etykiet krawędzi w cyklu Eulera grafu G_{n-1}, startując z sufiksu słowa w długości $n-1$.*



Rysunek 19: Graficzna ilustracja algorytmu generacji kolejnych n-krotek w ciągu de Bruijna wygenerowanym algorytmem on-line. Algorytm dopowiada obejściu drzewa cykli rotacyjnych. Cykle połączone są *dwukierunkowymi mostami*.

Alternatywny algorytm generacji ciągu de Bruijna. Rozważmy następującą wersję generacji on-line ciągu de Bruijna.

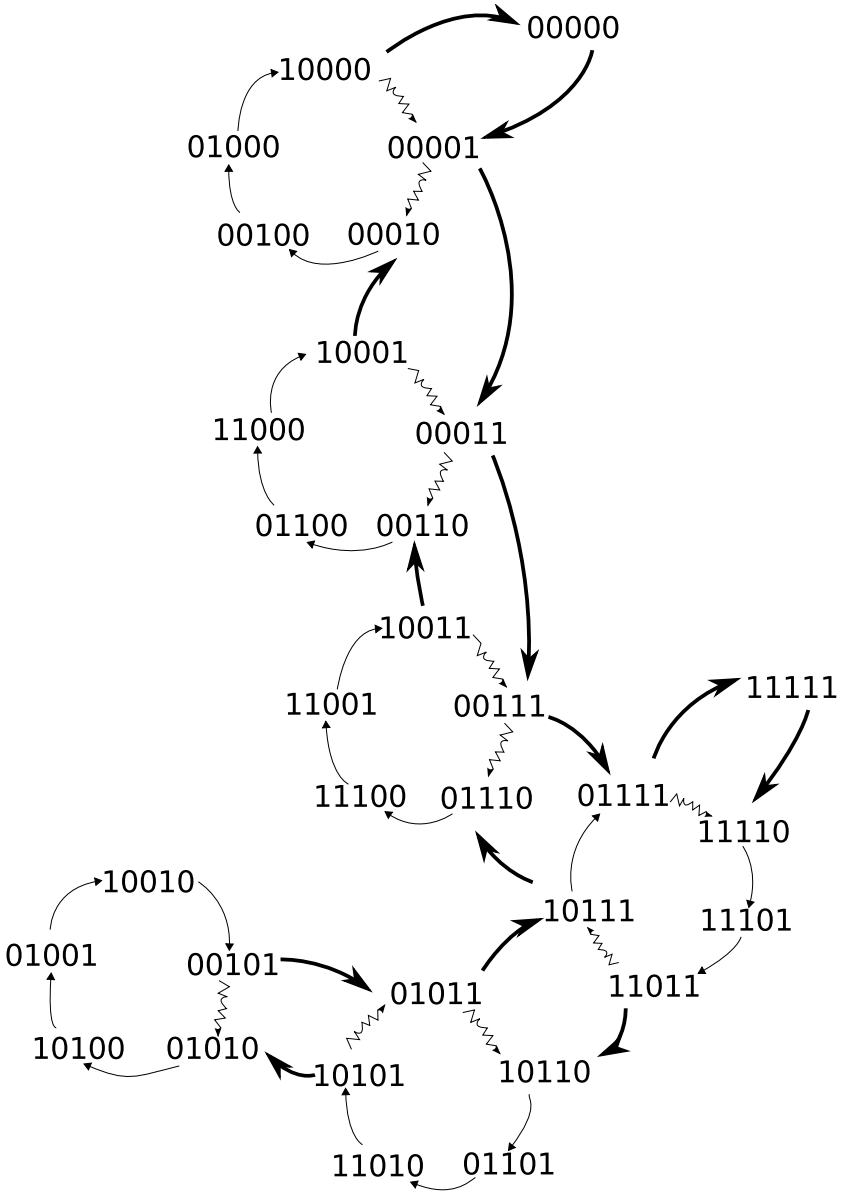
Algorytm On-Line generacji ciągu de Bruijna

przejście do kolejnej n-krotki: $b_1b_2\ldots b_n \to b_2b_3\ldots b_nb$

$0b_2b_3\ldots b_n$ jest ciągiem minimalnym w swojej klasie cykliczności

wtedy i tylko wtedy gdy $b = \overline{b_1}$.

Podobnie jak w poprzednim algorytmie można zinterpretować algorytm jako obchodzenie drzewa cykli rotacyjnych, patrz rysunek [3].



Rysunek 20: Graficzna ilustracja alternatywnego algorytmu generacji kolejnych n-krotek w ciągu de Bruijna wygenerowanym algorytmem on-line

15 Zastosowanie wyznacznika

Wyznacznik $\det(A)$ macierzy kwadratowej jest funkcją (od macierzy) o następujących własnościach.

- $\det(I_n) = 1$, gdzie I_n jest jednostkową $n \times n$ macierzą.
- Jeśli pomnożymy wiersz lub kolumnę przez stałą to wyznacznik też się pomnoży przez tę samą stałą.
- Przestawienie dwóch kolumn/wierszy zmienia znak wyznacznika.
- Dodanie do danej kolumny/wiersza innej kolumny/wiersza pomnożonej przez stałą nie zmienia znaku wyznacznika.
- Wyznacznik macierzy transponowanej jest taki sam.

[3] Rysunek wykonany przez Aleksandrę Jarmolińską

2	-1	-1				
	2		-1	-1		
		2			-1	-1
-1			2			
	-1	-1		2		
			-1	-1	2	
					-1	1

2				-2		
	2				-2	
		2				-2
-1			2			
	-1	-1		2		
			-1	-1	2	
					-1	1

2						
	2					
		2				
			2	-1		
	-1	-1		2	-1	-1
			-1	-1	2	
					-1	1

2						
	2					
		2				
	-1	-1	1			
	-1	-1		2	-1	-1
			-1	-1	2	
					-1	1

Tablica 1: Ilustracja trzech redukcji zastosowanych do macierzy dla grafu de Bruijna, początkowo mamy macierz L_n dla $n = 3$, po 3 redukcjach prawą dolną macierzą jest L_{n-1}.

15.1 Twierdzenie *Matrix-Tree* i wyznaczniki

Dla grafu skierowanego G o macierzy sąsiedztwa A przez $L(G)$ oznaczmy macierz $D(G) - A$, gdzie $D(G)$ jest macierzą, w której na przekątnej są wartości $D(G)_{i,i} = indegree(i)$. Dla $i \neq j$ mamy: $L(G)_{i,j} = -1$ wtedy i tylko wtedy gdy jest krawędź od i do j.

Twierdzenie 8 (**Matrix-Tree Theorem**). *Załóżmy, że G jest grafem skierowanym, spójnym i Eulerowskim oraz stopień wejściowy/wyjściowy każdego węzła jest co najwyżej 2. Wtedy liczba cykli Eulera jest równa liczbie drzew rozpinających skierowanych o (dowolnym) ustalonym korzeniu, oraz jest równa wyznacznikowi macierzy $L(G)$ z usuniętym pierwszym wierszem i pierwszą kolumną.*

15.2 Zastosowanie wyznacznika do liczenia ciągów de Bruijna

Pokażemy, jak policzyć liczbę wszystkich ciągów i cykli de Bruijna stosując wyznaczniki. Niech $G_n = (V, E)$ będzie grafem de Bruijna rzędu n:

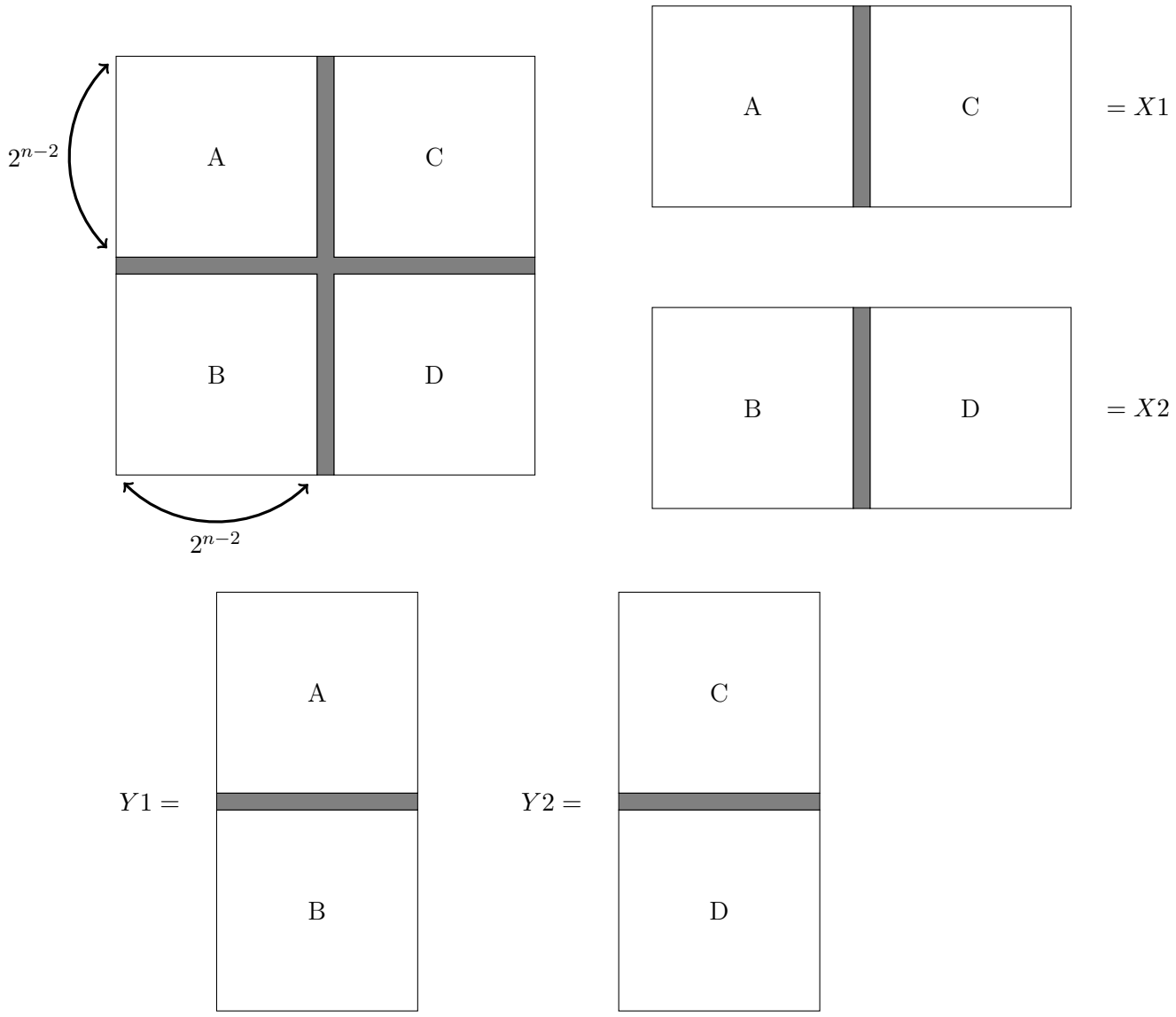
$$V = (0+1)^{n-1},\ E = \{a\alpha \to \alpha b : \alpha \in (0+1)^{n-2},\ a, b \in \{0, 1\}\}$$

$$B_n = \left[\begin{array}{cccccccc|cccccccc}
1 & -1 & & & & & & & & & & & & & & \\
 & 2 & -1 & -1 & & & & & & & & & & & & \\
 & & 2 & & -1 & -1 & & & & & & & & & & \\
 & & & 2 & & & -1 & -1 & & & & & & & & \\
 & & & & 2 & & & & -1 & -1 & & & & & & \\
 & & & & & 2 & & & & & -1 & -1 & & & & \\
 & & & & & & 2 & & & & & & -1 & -1 & & \\
 & & & & & & & 2 & & & & & & & -1 & -1 \\
\hline
-1 & -1 & & & & & & & 2 & & & & & & & \\
 & & -1 & -1 & & & & & & 2 & & & & & & \\
 & & & & -1 & -1 & & & & & 2 & & & & & \\
 & & & & & & -1 & -1 & & & & 2 & & & & \\
 & & & & & & & & -1 & -1 & & & 2 & & & \\
 & & & & & & & & & & -1 & -1 & & 2 & & \\
 & & & & & & & & & & & & -1 & -1 & 2 & \\
 & & & & & & & & & & & & & & -1 & 1
\end{array}\right]$$

Redukcja macierzy

1. $X1 := X1 - X2$;
2. $Y2 := Y2 + Y1$;
3. Dodajemy do środkowego wiersza wszystkie wiersze $X2$.

Po redukcji dostajemy, że wartością wyznacznika jest



Rysunek 21: Podział macierzy. Z macierzy usuwamy pierwszy wiersz i kolumnę, uzyskując macierz L_n o kształcie $(2^{n-1}-1) \times (2^{n-1}-1)$. Po pominięciu środkowego „krzyża" mamy cztery ćwiartki A, B, C, D.

$$\det(L_n) = 2^k \cdot \det(L_{n-1}), \text{ gdzie } k = 2^{n-2} - 1$$

Stąd mamy wzór na liczbę wszystkich cykli de Bruijna:

$$\det(L_n) = 2^{2^{n-1}-n}$$

Wzór na liczbę wszystkich ciagów de Bruijna jest prostszy: $\sqrt{2^{2^n}}$.

15.3 Inne grafy.

Oznaczmy przez Q_n graf będący kostką n-wymiarową. Stosując wyznaczniki podobnie jak poprzednio można pokazać następujący fakt.

Fakt. Liczba drzew rozpinających w grafie Q_n wynosi:

$$\prod_{S\subseteq[n],\,|S|\geqslant 2} 2\cdot|S| = \prod_{k=2}^{n}(2k)^{\binom{n}{k}}$$

Dla grafu pełnego K_n otrzymujemy:

Fakt. Liczba drzew rozpinających w grafie Q_n wynosi n^{n-2}.

Podobnie prosty wzór można otrzymać dla grafu pełnego dwudzielnego $K_{m,n}$.

Fakt. Liczba drzew rozpinających w grafie $K_{m\,n}$ wynosi $n^{m-2}m^{n-2}$

16 Dwa liniowe algorytmy konstrukcji binarnych drzew

W sekcji tej opiszemy dwa bardzo ładne i bardzo proste algorytmy działające w czasie liniowym, które dla zadanego ciągu $a_1, a_2, \ldots, a_n$ różnych liczb konstruują dwa typy drzew.

- Drzewo Kartezjańskie $K(a_1, a_2, \ldots a_n)$: korzeniem jest najmniejszy element a_k, jego lewe poddrzewo jest drzewem Kartezjańskim dla ciągu $a_1, a_2, \ldots, a_{k-1}$, a prawe dla ciągu $a_{k+1} \ldots a_n$.
- Drzewo wyszukiwań binarnych $BST(a_1, a_2, \ldots, a_n)$. Zakładamy tutaj, że elementy ciągu można posortować w czasie liniowym. Tak więc bez straty ogólności dla konstrukcji BST przyjmijmy, że ciąg jest permutacją. Korzeniem jest a_1, lewe poddrzewo jest BST dla podciągu elementów mniejszych od a_1, prawym poddrzewem dla podciągu elementów większych od a_1 (w kolejności tak jak w początkowej permutacji).

Skonstruowanie tych drzew bezpośrednio z definicji daje naturalne algorytmy, które nie działają w czasie liniowym.

Algorytm 15: Algorytm konstrukcji $K(a_1, \ldots, a_n)$ (ciąg różnych elementów)

```
początkowo drzewo składa się z korzenia o wartości a_1;
for i = 2 to n do
    v := pierwszy od dołu węzeł na prawostronnej gałęzi taki, że v > a_i;
    leftson(a_i) := rightson(v); rightson(v) := a_i;
end
```

Algorytm 16: Algorytm konstrukcji $BST(a_1, \ldots, a_n)$ (ciąg jest permutacją)

```
rank(i) jest pozycją i w permutacji (a_1, a_2, ..., a_n);
lista L := (1, 2, ..., n);
for i = n downto 1 do
    if |L| > 1 then
        v := sąsiad a_i w liście L z większym rank;
        if v < a_i then rightson(v) := a_i;
        else leftson(v) := a_i;
        ;
        usuń a_i z listy L;
end
```

17 Kilka prostych gier kombinatorycznych.

Na początku pokażemy użyteczność operacji bitowej *xor* w grach kombinatorycznych. W grach tych chodzi przede wszystkim o to, żeby znaleźć algorytmicznie prostą funkcję odpowiadającą na pytania: czy zadana konfiguracja jest wygrywająca (zakładając perfekcyjną grę obu graczy). W bardziej skomplikowanych grach (szachy, wacaby, go) takiej prostej funkcji raczej nie ma. Skoncentrujemy się na bardzo prostych grach, w których funkcja konfiguracji wygrywającej związana jest z operacjami bitowymi na liczbach opisujących konfigurację.

Operacja $\oplus$, nazywana *xorem* jest użyteczna w wielu prostych grach kombinatorycznych w opisie tzw. funkcji Grundy'ego Gr.

Inną operacją jest operacja MEX (minimal excludant). Niech N będzie zbiorem liczb naturalnych z zerem.

$$MEX(X) = \min(N - X)$$

W szczególności

$$MEX(\emptyset) = 0$$

Załóżmy, że grze odpowiada **acykliczny**, z reguły **bardzo duży**, graf konfiguracji. Niech $x \to y$ będzie przejściem od konfiguracji x do konfiguracji y. W to zbiór konfiguracji wygrywających, P to zbiór konfiguracji przegrywających.
Funkcja całkowitoliczbowa Gr na zbiorze konfiguracji jest funkcją Grundy'ego gdy

$$g(x) = MEX\left(\{g(y) : x \to y\}\right)$$

Funkcja ta może istnieć nawet dla grafów z cyklami, tym niemniej zakładamy dalej w tej sekcji acykliczność. Funkcja g spełnia:

1. $x \in W \Leftrightarrow Gr(x) \neq 0$,
2. $x \in P \wedge x \to y \Rightarrow Gr(y) \neq 0$,
3. $x \in W \Rightarrow \exists_{x,\,y}\ x \to y, Gr(y) = 0$.

17.1 Gra NIM

Zbiorem konfiguracji jest zbiór wektorów k-wymiarowych o współczynnikach naturalnych, ruch polega na zmniejszeniu co najmniej o jeden którejś współrzędnej. Konfiguracja zerowa jest przegrana. W tym przypadku

$$Gr(x_1,\, x_2, \ldots,\, x_k) = x_1 \oplus x_2 \oplus x_3 \ldots \oplus x_k$$

17.2 Staircase (schodkowy) NIM

Gra ta jest wersją gry NIM, jest opisana jako zadanie „Gra" na XI OI. W tej grze możemy zmniejszyć (co najmniej o jeden) pierwszą współrzędną, lub wykonać dla $i > 1$:

`if` $x_i \geqslant \Delta > 0$ `then`
$x_i := x_i - \Delta$; $x_{i-1} := x_{i-1} + \Delta$

Inaczej mówiąc możemy przesuwać elementy ze schodka wyższego na bezpośrednio niższy lub usuwać z pierwszego schodka. W tym przypadku

$$Gr(x_1,\, x_2, \ldots,\, x_k) = x_1 \oplus x_3 \oplus x_5 \ldots \oplus \ldots$$

17.3 Monotoniczny NIM

Gra ta jest opisana jako zadanie „Kamyki" na XVI OI. Gra jest podobna do NIM z dodatkowym wymaganiem aby współrzędne wektora konfiguracji były zawsze niemalejące. W przypadku tej gry załóżmy, że k jest parzyste, ewentualnie dodając z przodu (gdy k nieparzyste) jedną „sztuczną" współrzędną zawsze zerową. W tym przypadku zachodzi:

$$Gr(x_1,\, x_2, \ldots,\, x_k) = (x_2 - x_1) \oplus (x_4 - x_3) \oplus \ldots (x_k - x_{k-1}).$$

17.4 Ograniczony NIM

Z dowolnego dokładnie jednego stosu możemy wziąć niezerową liczbę ale ograniczoną przez 4.

17.5 Twierdzenie Sprague-Grundy'ego

Grą jest graf acykliczny w którym każda ścieżka się gdzieś kończy. Jeśli mamy gry $G_1, \ldots, G_k$ to ich sumą $G_1 + G_2 + G_k$ jest gra, w której gracze kolejno wybierają jedną z G_i i wykonują w niej ruch. Konfiguracja gry to ciąg $(u_1, u_2, \ldots, u_k)$ konfiguracji w grach G_i.

Twierdzenie Sprague-Grundy'ego
Funkcja Grundy'ego konfiguracji sumy gier jest xor-sumą funkcji Grundy'ego dla poszczególnych komponentów.

Twierdzenie Boutona jest szczególnym przypadkiem. Dla komponentów mamy tutaj funkcję: $g(k) = k$.

17.6 Gry typu NIM i mocne twierdzenie Sprague-Grundy'ego

Zdefiniujmy teraz grę w inny sposób. W grze tej konfiguracjami będą multizbiory pewnego zbioru V.

Załóżmy, że mamy częściowo uporządkowany zbiór elementów V w którym nie ma nieskończonego ściśle malejącego ciągu, schodzenie do poprzednich wierzchołków kiedyś się kończy. Inaczej mówiąc mamy pewien graf acykliczny G z korzeniem. Korzeń to jest utożsamienie wszystkich elemenów minimalnych.

Mamy funkcję $PRED$ taką, że $PRED(v)$ jest skończonym zbiorem skończonych podzbiorów V. Jeśli $X \in PRED(V)$ to wszystkie elementy z X są mniejsze, w sensie częściowego porządku, od v (muszą być bezpośrednimi poprzednikami v w grafie G).

Obserwacja. Graf G jest tu właściwie nieistotny, ale wygodny.

W związanej z tym grze konfiguracjami K są skończone multizbiory V. Gracz wybiera $v \in K$, usuwa v i wstawia do multizbioru K jeden ze zbiorów $X \in PRED(v)$. Gracz, który nie ma ruchu przegrywa. Tak zdefiniowaną grę na multizbiorach nazywamy *grą typu NIM* (zadaną przez funkcję $PRED$). Inaczej mówiąc mamy grę w której gracz może zainicjować kilka wierzchołków, np. rozbić stos na dwie części o ustalonej relacji między nimi (np. różne rozmiary, lub rozmiar jednej 5 razy większy od rozmiaru drugiej części). Dodajemy graczowi dużo możliwości. Konfiguracja jest tak naprawdę *uogólnioną sumą gier*.

Oznaczmy przez $mult_K(u)$ wielokrotność (multiplicity) element u w konfiguracji K. Niech $\otimes$ oznacza operacje mnożenia niezależnie po bitach modulo 2 (bierzemy *xor* wiele razy).

Mocne twierdzenie Sprague-Grundy'ego
W grze typu NIM dla konfiguracji (multizbioru) K mamy

$$g(K) = XOR_{u \in K};\ mult_K(u) \otimes g(u)$$

Wniosek.
Funkcja Grundy'ego dla gry typu NIM dla $v \in V$ (dla singletonów) spełnia:

$$g(v) = MEX(\{XOR_{u \in X};\ g(u) : X \in PRED(v)\})$$

Daje to z reguły wielomianowy algorytm liczenia funkcji Grundy'ego dla singletonów. Zauważmy, że potencjalnych konfiguracji generowanych z singletona może być z reguły wykładniczo wiele.
Poprzednie (standardowe) twierdzenie Sprague-Grundy'ego wynika stąd, jeśli założymy, że V jest rozłączną sumą $V_1 \cup V_2 \ldots \uplus V_k$, gdzie V_i jest zbiorem lokalnych konfiguracji w grze G_i, oraz dla każdego elementu v funkcja $PRED(v)$ daje zbiór singletonów (poprzedników v) w grze G_i, gdzie $v \in V_i$.

17.7 Gra Grundy'ego

Kolejno gracze dzielą jeden ze stosów (rozmiaru n) na dwie nierówne i niepuste części o rozmiarach i, j. Liczba stosów może rosnąć. Gracz który nie ma ruchu przegrywa.

$$g(n) = MEX(\{g(i) \oplus g(j) : i \neq j,\ i + j = n,\ 0 < i, j < n\})$$

Początkowe wyrazy tego ciągu dla $n = 0..20$ to:

```
0  0  0  1  0  2  1  0  2  1  0  2  1  3  2  1  3  2  4  3  0
```

Otwarty problem i hipoteza. Ciąg Grundy'ego jest od pewnego miejsca okresowy (w każdym razie przynajmniej zbiór możliwych wartości jest $\mathcal{O}(1)$).

17.8 Gra PASKI

Mamy paski (prostokąty o kształcie $1 \times c_i$) długości c_1, c_2, c_3, jako narzędzia do wycinania. Na początku w grze mamy planszę o kształcie $1 \times n$ długości n, w trakcie gry mamy pewną liczbę plansz być może różnej długości. Jeden ruch polega na wybraniu jednej planszy i wycięciu z niej paska (podplanszy) długości c_i, gdzie $1 \leqslant i \leqslant 3$.
Liczymy funkcję Grundy'ego $g(n)$ dla każdej (spójnej) planszy bez dziur długości n.
W tym przypadku $PRED(n)$ jest rodziną zbiorów składających się z pary elementów (i, j), $0 \leqslant i, j < n$, takich, że po włożeniu ustalonego paska rozbijamy n na plansze spójne rozmiarów i, j (mogą być zerowe).

17.9 Gra „Obcinanie drzewa"

Artykuł Kulczyńskiego z Delty.
?????

17.10 Kayles

????????

```
      0+  0  1  2  3  1  4  3  2
1  4  2  6
     12+  4  1  2  7  1  4  3  2
1  4  6  7
     24+  4  1  2  8  5  4  7  2
1  8  6  7
     36+  4  1  2  3  1  4  7  2
1  8  2  7
     48+  4  1  2  8  1  4  7  2
1  4  2  7
     60+  4  1  2  8  1  4  7  2
1  8  6  7
     72+  4  1  2  8  1  4  7  2
1  8  2  7
```

17.11 Gra NIM$_K$ i Twierdzenie Moore'a

Ta gra jest tym samym co standardowy NIM, z tą różnicą, że gracz w jednym ruchu może zmniejszyć rozmiar co najwyżej $K \geqslant 1$ stosów jednocześnie, przy czym co najmniej jeden ze stosów musi się zmniejszyć.
Oznaczamy tę grę przez NIM_K.
Zdefiniujmy $\oplus_K$ jako operację dodawania modulo $K + 1$.

Twierdzenie Moore'a.(Eliakim Hastings Moore, 1909)
(E.H. Moore, "A Generalization of the Game Called Nim" Ann. Math. Princeton, Series 2, Vol. 11, pp. 93-94 (1909-1910))
W grze NIM$_K$ z m stosami zachodzi:

$$g(x_1, x_2, \ldots, x_m) = x_1 \oplus_K x_2 \oplus_k x_3 \ldots \oplus_K x_m$$

Pozycje przegrane to te z wartością $g(x_1, x_2, \ldots, x_m) = 0$.

17.12 Gra Wythoffa

17.13 Problem Josephusa: eliminacja cykliczna

Problem ten nie jest grą w dosłownym znaczeniu, ale można go tak potraktować. W tej grze mamy liczby od 1 do n umieszczone na okręgu, począwszy od liczby 1 usuwamy co drugi element aż zostaje jeden. Niech $J(n)$ będzie numerem tego elementu. Problem polega na szybkim policzeniu $J(n)$. Element o numerze $J(n)$ wygrywa (przeżywa), a pozostałe przegrywają (giną). Niech $shift(n)$ będzie liczbą powstałą z n przez przesunięcie pierwszego bitu na koniec. Na przykład

$$shift(12) = shift([1100]_2) = [1001]_2 = 9$$

Można pokazać, że $J(n) = shift(n)$.

17.14 Gra „kompletowanie zbioru"

Mamy m zbiorów n elementowych $X_1, X_2, \ldots, X_m$. Pierwszy gracz (wybieracz) wybiera pewien element, drugi gracz (usuwacz) usuwa jeden element dotychczas nie wybrany, i tak na przemian. Usuniety element jest usuwany ze wszystkich zbiorów do których należy. Pierwszy gracz wygrywa gdy skompletuje jeden ze zbiorów.

Przykład. Weźmy $m = 2^{n-1}$. Każdy ze zbiorów zawiera ten sam element a, oraz dokładnie jeden spośród a_i, b_i dla $i = 1, \ldots, n-1$. Mamy zatem 2^{n-1} podzbiorów, każdy mocy n. W takim zestawie 1-szy gracz zawsze wygrywa. Wybiera najpierw a a potem a_i gdy usuwacz wybierze b_i, lub b_i gdy usuwacz wybierze a_i.

Twierdzenie (Erdos-Selfridge 1973)
jeśli $m < 2^{n-1}$ to, zakładając, że usuwacz działa optymalnie, 1-szy gracz nie wygrywa.

Uzasadnienie. Załóżmy, że wybieracz wybrał już zbiór elementów Y. Potencjałem elementu $x \notin Y$ jest

$$\Phi(x) = \sum_{x \in X_i} 2^{|X_i \cap Y|}$$

W każdym kroku wybieracz usuwa element o maksymalnym potencjale. Wtedy wartość $\sum_i 2^{|X_i \cap Y|}$ po wykonaniu 2 kolejnych ruchów: (usuwacz; wybieracz) się nie zwiększa. Zatem wybieracz nigdy nie osiągnie żadnego kompletnego zbioru X_i (tzn. nigdy nie będzie $X_i \subseteq Y$), gdyż gdyby tak się stało to ta suma byłaby nie mniejsza niż 2^n, a jest ona mniejsza od 2^n po pierwszym ruchu wybieracza.

Rozważmy podobną grę. W tej grze wybieracz wybiera krawędzie z grafu pełnego K_n, a usuwacz usuwa te jeszcze nie wybrane. Wybieracz wygrywa gdy z wybranych krawędzi można utworzyć pełny graf K_r. Z Twierdzenia Erdosa-Selfridge'a wynika

Fakt. Jeśli $\binom{n}{k} < 2^{\binom{r}{2}-1}$ to wybieracz nie wygra (nie utworzy podgrafu K_r) przy optymalnej strategii usuwacza.

18 Sortowanie Berge'a

Mamy ciąg n-elementowy $\alpha_n = 10101010\ldots$, ostatnim elementem jest 1 wtedy i tylko wtedy gdy n nieparzyste. Zakładamy $n \geqslant 5$.

Operacja elementarna to przeniesienie 2 sąsiednich elementów na dwa wolne sąsiadujące pola, wszystkie (nieskończenie wiele) pola na lewo i na prawo ciągu α są początkowo wolne. Celem jest posortowanie ciągu w optymalnym czasie, oznaczmy przez $\widetilde{\alpha}_n$ ciąg posortowany.

Na przykład dla $n = 5$ optymalnym sortowaniem (w $\lceil \frac{n}{2} \rceil = 3$ ruchach) jest

$$\alpha_5 = 10101 \to 1\sqcup\sqcup0101 \to 1100\sqcup\sqcup1 \to 00111 = \widetilde{\alpha}_5$$

Fakt. Optymalny czas $\geqslant \lceil \frac{n}{2} \rceil$.

Dowód metodą zmniejszania potencjału - liczby elementów niepustych za którymi bezpośrednio na prawo jest wolne pole lub element innego koloru.

Sortowanie jest silne, gdy ciąg do posortowania używa tylko pól na których jest ciąg plus dwóch pustych pól na prawo, po posortowaniu ciąg przesuwa się o dwa pola na prawo.

Sortowanie Berge'a - schemat algorytmu
(1) Znajdujemy *ręcznie* w czasie $\mathcal{O}(1)$ silne sortowanie dla $n \in \{8, 10, 12, 14, 16\}$ oraz (niekoniecznie silne) sortowanie dla nieparzystych $5 \leqslant n < 17$.
(2) (rekurencja) Dla parzystych $n \geqslant 16$, sprowadzamy silne sortowanie α_n do silnego sortowania α_{n-8} korzystając z 4 ruchów.
(3) Sprowadzamy sortowanie (tym razem nie silne) dla nieparzystych $n \geqslant 17$ w 5 krokach do silnego sortowania α_{n-9} (mamy $n-9$ parzyste).

Opiszemy teraz dokładniej kroki (2), (3).

Rekurencyjny algorytm $n \geqslant 16$ parzyste.

$$1\,0\,1\,0\,\alpha_{n-8}\,1\,0\,1\,0 \to 1\sqcup\sqcup 0\,\alpha_{n-8}\,1\,0\,1\,0\,0\,1 \to 1\,1\,0\,0\,\alpha_{n-8}\sqcup\sqcup 1\,0\,0\,1$$

$$\to 1\,1\,0\,0\,\alpha_{n-8}\sqcup\sqcup 1\,0\,0\,1 \to^{rekursja} 1\,1\,0\,0\sqcup\sqcup\widetilde{\alpha}_{n-8}\,1\,0\,0\,1 \to$$

$$1\,1\,0\,0\,0\,0\,\widetilde{\alpha}_{n-8}\,1\sqcup\sqcup 1 \to 0\,0\,0\,0\,\widetilde{\alpha}_{n-8}\,1\,1\,1\,1 = \widetilde{\alpha}_n$$

Sprowadzenie przypadku nieparzystego do parzystego
Zakładamy, że $n \geqslant 17$ nieparzyste (wtedy $n-9$ jest parzyste).

$$\alpha_n = 1\,0\,1\,0\,1\,0\,\alpha_{n-9}\,1\,0\,1 \to 1\,0\,1\,0\,\alpha_{n-9}\,1\,0\,1\,1\,0 \to$$

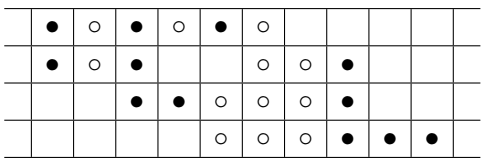
$$1\sqcup\sqcup 0\,\alpha_{n-9}\,1\,0\,1\,1\,0\,0\,1 \to 1\,1\,0\,0\,\alpha_{n-9}\sqcup\sqcup 1\,1\,0\,0\,1$$

$$\textit{(sortowanie dla przypadku parzystego)} \to^* 1\,1\,0\,0\sqcup\sqcup\widetilde{\alpha}_{n-9}\,1\,1\,0\,0\,1 \to$$

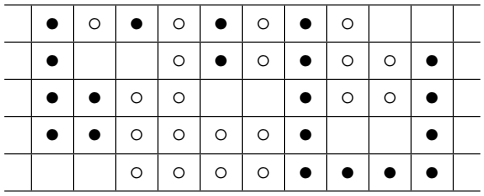
$$1\,1\,0\,0\,0\,0\,\widetilde{\alpha}_{n-9}\,1\,1\sqcup\sqcup 1 \to 0\,0\,0\,0\,\widetilde{\alpha}_{n-9}\,1\,1\,1\,1\,1 = \widetilde{\alpha}_n$$

Sortowania Berge'a dla $n = 6, 8, 10, 12, 14$

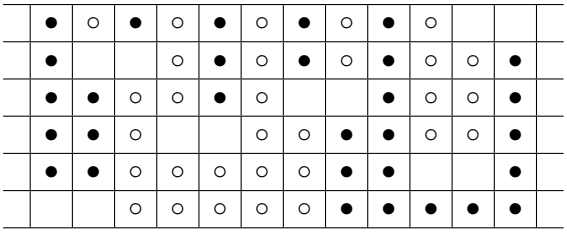
$n = 6$



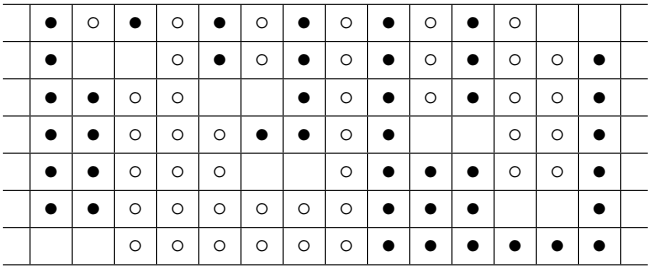
$n = 8$



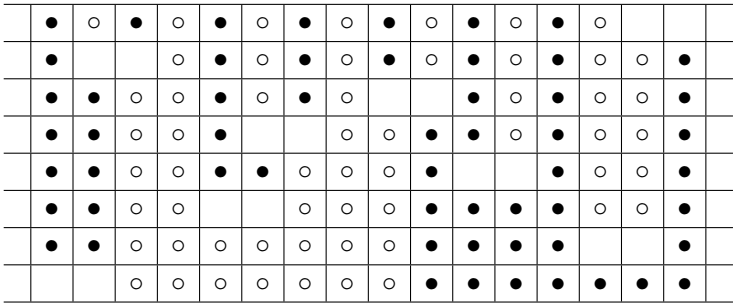
$n = 10$



$n = 12$



$n = 14$



19 Grafy szachowe

Rozważamy grafy pięciu figur szachowych: S – skoczek, K – Król, H – hetman, W – Wieża, G – goniec. Przez $F_{n,m}$ oznaczmy graf figury F na szachownicy $n \times m$. Jeśli $n = m$, to będziemy pisać F_n. Zajmiemy się (między innymi) liczeniem następujących parametrów dla grafów szachowych:

- $\gamma(G)$ – liczba dominacji grafu G, minimalna liczba figur które *biją* wszystkie pola;
- $\alpha(G)$ – liczba niezależności, maksymalna liczba figur które się nawzajem *nie biją*;
- $\omega(G)$ – liczba klikowa (rozmiar najliczniejszej kliki);
- $\chi(G)$ – liczba chromatyczna grafu G.

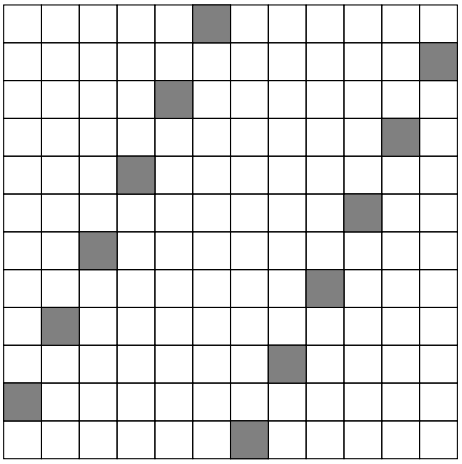
Poza tym zajmiemy się istnieniem cyklu (ścieżki) Hamiltona.

19.1 Graf hetmana

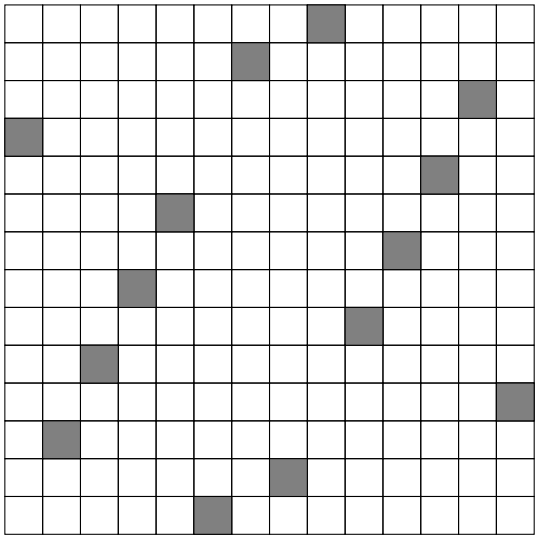
Mamy 92 zbiory niezależne rozmiaru 8 w grafie hetmana H_8. Nie ma ogólnego wzoru na liczbę zbiorów niezależnych w H_n, natomiast ogólnie zachodzi:

$$\alpha(H_n) = n \text{ dla } n \geqslant 5$$

Uzasadnienie Poniższe rysunki pokazują, że możemy dla $n \geqslant 6$ rozmieścić n hetmanów nawzajem niezależnych. Następnie zauważmy, że przekątne są wolne, zatem można rozszerzyć na szachownicę o boku o jeden większym.



Rysunek 22: Schemat ustawienia n niezależnych hetmanów dla szachownicy o boku $6k$ lub $6k+4$.



Rysunek 23: Schemat dla szachownicy o boku $6k+2$.

Wariacje problemu niebijących się hetmanów

- Szachownica cylindryczna. Na cylindrycznej szachownicy 8×8 ośmiu niezależnych hetmanów się nie ustawi.
- Niech HS będzie skrzyżowaniem skoczka z hetmanem, wtedy na szachownicy 8×8 nie ma 8 niezależnych figur HS, natomiast na szachownicy 10×10 można ustawić 10 takich niezależnych (niebijących się) figur HS.

Zajmiemy się teraz liczbą dominacji $\gamma(n) = \gamma(H_n)$. Dla małych wartości mamy

$$\gamma[2..13] \;=\; [1,\, 1,\, 2,\, 3,\, 3, \ldots]$$

Niech $diag(n)$ będzie minimalną liczbą hetmanów, które dominują szachownicę $n \times n$ i które są tylko na głównej przekątnej. Mówimy że zbiór liczb naturalnych $\mathcal{X}$ jest jedno-parzysty, gdy elementy $\mathcal{X}$ są tej samej parzystości. $\mathcal{X}$ jest bezśrodkowy jeśli

$$\forall_{a,\, b \in \mathcal{X}}\; a \neq b \Rightarrow \frac{a+b}{2} \notin \mathcal{X}$$

Niech $mid(n)$ będzie mocą maksymalnego jednoparzystego i bezśrodkowego podzbioru $[n]$. Zachodzi następujący fakt:

$$diag(n) = n - mid(n)$$

Zauważmy, że $\gamma(n) = diag(n)$ dla $n \leqslant 9$, natomiast $diag(10) > \gamma(n)$. Nie potrafimy policzyć szybko wartości $\gamma(n)$, natomiast mamy konstrukcję, która przybliża dobrze wartość γ.

$$\frac{1}{2}(n-1) \leqslant \gamma(n) \leqslant \left\lfloor \frac{2}{3}n \right\rfloor + n \bmod 3$$

Uzasadnienie

- **Górna granica:** możemy założyć, że n podzielne przez 3. Dzielimy planszę na 9 takich samych części, w lewej dolnej części umieszczamy $\frac{n}{3}$ hetmanów następująco: jeden w dolnym lewym rogu, pozostałe na głównej odwrotnej przekątnej przesuniętej o jeden w górę. W prawej górnej części umieszczamy hetmany na głównej odwrotej przekątnej.
- **Dolna granica:** argument kombinatoryczny, rachunki.

Liczba chromatyczna Wiadomo, że $8 \leqslant \chi(H_9) \leqslant 9$, tzn. istnieje kolorowanie za pomocą 9 kolorów, patrz rysunek. Czy istnieje za pomocą 8? Oczywiście 8 to dolna granica. Prawdopodobnie (ale nie wiadomo na pewno)

$$\chi(H_n) \in \{n,\, n+1,\, n+2\}$$

6	3	7	3	1	5	9	4
9	1	5	8	4	7	3	2
5	4	9	7	3	2	6	8
2	7	3	4	6	1	5	9
3	6	2	5	7	9	4	1
7	5	1	9	2	3	8	6
1	9	4	7	8	6	2	3
8	3	6	2	9	4	1	5

Rysunek 24: Kolorowanie grafu hetmana na szachownicy 8×8 za pomocą 9 kolorów.

Dwa niezależne hetmany Ustawiamy dwa hetmany losowo na szachownicy $n \times n$. Niech p_n oznacza prawdopododobieństwo, że się nie atakują. Wtedy $\lim p_n = \frac{1}{3}$. A co z trzema hetmanami?

19.2 Graf skoczka

Mamy $\alpha(S_{n,n}) = \lceil \frac{n}{2} \rceil$. Natomiast nie ma sensownego wzoru na $\gamma(S_{n,n})$.

					S		
	S	S		S	S		
		S					
		S	S		S		
		S			S	S	

Rysunek 25: $\gamma(S_8) = 12$

Zajmiemy się istnieniem i konstrukcją cyklu Hamiltona w $S_{n,m}$.

Twierdzenie 9 (Schwenk)**.** *Załóżmy $m \leqslant n$. Istnieje cykl Hamiltona na $S_{n,m}$ wtedy i tylko wtedy, gdy nie zachodzi żaden z następujących warunków.*

1. *$n \leqslant 2$ lub $m \leqslant 2$;*

2. n, m *obie nieparzyste;*

3. $m = 3,\ n < 10$*;*

4. $m = 4$.

Dowód. Jeśli $m \leqslant 2$, to szachownica nie jest dość „szeroka" na istnienie cyklu — z niektórych wierzchołków grafu skoczka wychodzi tylko jedna krawędź. Punkt (2) jest oczywisty — jeśli obie współrzędne szachownicy są nieparzyste, to łączne pole też jest nieparzyste. A ponieważ graf skoczka jest dwudzielny (przeskoki pomiędzy czarnymi i białymi polami), to nie może istnieć cykl o długości nieparzystej.

Udowodnienie nieistnienia cyklu Hamiltona dla pozostałych z wyżej wymienionych przypadków jest nietrywialne. □

Reszta dowodu jest konstruktywna – polega na konstrukcji cyklu w czasie liniowym jeśli istnieje. Jeden dowód Parberry'ego (ale tylko dla parzystych n, m) poprzez dzielenie na cztery (prawie) ćwiartki. Drugi dowód z oryginalnej pracy Schwenka przez dodawania 4 kolumn lub czterech wierszy. Mamy 9862 cykli na S_6 oraz 13 267 364 410 532 cykli na S_8.

Twierdzenie 10. *Na szachownicy skoczka $m \times n$ będącej torusem zawsze jest cykl Hamiltona.*

Twierdzenie 11. *Na szachownicy skoczka $m \times n$ (n kolumn) będącej cylindrem (ostatnia kolumna sąsiaduje z pierwszą) jest cykl Hamiltona wtw. gdy nie zachodzi zaden z warunków:*

1. $m = 1$ *oraz* $n > 1$

2. $m \in \{2, 4\}$ *oraz n parzyste*

19.3 Graf króla

Mamy

$$\gamma(K_{n,n}) = \left\lceil \frac{n+2}{3} \right\rceil$$

W szczególności

$$\gamma(K_7) = \gamma(K_8) = \gamma(K_9) = 0$$

Również łatwo się liczy $\alpha(K_{n,n})$ i $\chi(K_{n,n})$.

20 Relokacja (przesuwanie) na grafie – uogólnienie gry *Piętnastka*

Mamy planszę $n \times n$, oznaczamy ją przez G_n. Graf G_n jest *gridem* rzędu n. W prawym dolnym rogu jest puste pole, pozostałe zawierają liczby od 1 do $n^2 - 1$. Konfiguracja jest permutacją π tych liczb. Z jednej konfiguracji możemy przejść do innej przesuwając liczbą na sąsiednie wolne pole.

Twierdzenie 12. *Z konfiguracji π możemy otrzymać konfigurację identycznościową wtedy i tylko wtedy, gdy π ma znak dodatni (parzysta liczba transpozycji, permutacja parzysta). Jeśli można otrzymać, to wystarczy i czasami trzeba $\Theta(n^3)$ ruchów.*

Są dwa dowody tego faktu, oba konstruktywne. Jeden Aarona Archera, drugi Parberry'ego, z tym że ten drugi sprowadza problem do $n = 3$ i wtedy korzysta z dowodu Aarona Archera. R.M. Wilson uogólnił ten probem na dowolne grafy nieskierowane. W dowodzie Aarona Archera korzystamy z następującego faktu kombinatorycznego.

Twierdzenie 13. *Zbiór wszystkich permutacji cyklicznych postaci $(k, k+1, k+2)$ generuje dokładnie zbiór wszystkich permutacji parzystych zbioru $[n]$.*

20.1 Algorytm rozwiązywania „piętnastki” rozmiaru 3×3

Idea algorytmu opiera się na zdefiniowaniu 5 operacji, będących ciągami elementarnych ruchów, jakie można wykonywać przesuwając sąsiedni element na puste pole. Z operacji tych zostanie zbudowane rozwiązanie postawionego problemu. Poniższy diagram przedstawia jak poszczególne operacje zmieniają stan łamigłówki:

1. operacja `A1`

a	b	c
d	e	f
g	h	0

$\Rightarrow$

a	e	f
d	b	c
g	h	0

2. operacja `B1`

a	b	c
d	e	f
g	h	0

$\Rightarrow$

a	c	b
d	f	e
g	h	0

3. operacja `A2`

a	b	c
d	e	f
g	h	0

$\Rightarrow$

a	b	c
g	h	f
d	e	0

4. operacja `B2`

a	b	c
d	e	f
g	h	0

$\Rightarrow$

a	b	c
e	d	f
h	g	0

5. operacja `C`

a	b	c
d	e	f
g	h	0

$\Rightarrow$

a	c	b
d	e	f
h	g	0

Operacje `A2` i `B1` są symetryczne do (odpowiedno) `A1` i `B1`. Zauważmy, że za pomocą operacji `A1`, `B1`, `A2` i `B2` możemy przesunąć dowolny klocek z prawego górnego kwadratu 2×2 do lewego dolnego kwadratu 2×2 i odwrotnie. Jest tak, ponieważ złożenie operacji `A1` i `B1` powoduje cykliczne obracanie elementów w prawym górnym kwadracie 2×2, a złożenie operacji `A2` i `B2` powoduje analogiczny skutek w lewym dolnym kwadracie 2×2. W takim razie, dowolny element z prawego górnego kwadratu 2×2 możemy przenieść w miejsce o współrzędnych $(2, 2)$ za pomocą ciągu operacji `A1` i `B1`. Analogicznie możemy robić z lewym dolnym kwadratem 2×2 i operacjami `A2` i `B2`. Algorytm składa się z następujących kroków:

1. Przesuwamy klocek z numerem jeden w lewy górny róg i wracamy pustym miejscem w prawy dolny róg.
2. Mamy następującą sytuację:

1	b	c
d	e	f
g	h	0

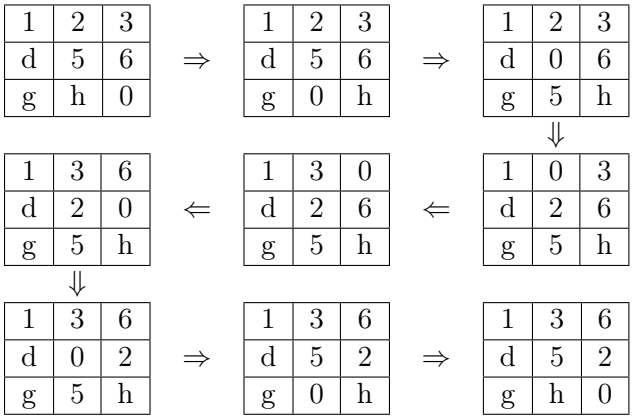
Korzystając z operacji `A1`, `B1`, `A2` i `B2` „przerzucamy” elementy z prawego górnego kwadratu 2×2 do lewego dolnego kwadratu 2×2 i odwrotnie, tak aby uzyskać w prawym górnym kwadracie 2×2 elementy ze zbioru $\{2, 3, 5, 6\}$, a w lewym dolnym kwadracie 2×2 elementy ze zbioru $\{4, 5, 7, 8\}$. Zauważmy, że w takim układzie elementów na pozycji o współrzędnych $(2, 2)$ musi znajdować się element o wartości 5.

3. Zauważmy, że prawy górny kwadrat 2×2 może mieć jedną z sześciu postaci:

2	3		6	2		3	6	
5	6		5	3		5	2	

2	6		3	2		6	3	
5	3		5	6		5	2	

Trzy górne możliwości wymagają parzystej liczby zamian elementów, a trzy dolne wymagają nieparzystej liczby zamian do osiągnięcia porządku $(2, 3, 5, 6)$. Zauważmy, że możemy przechodzić cyklicznie między trzema górnymi stanami stosując kilkakrotnie poniższe ruchy (nazwijmy je operacją `D`):



Analogicznie można przechodzić między dolnymi stanami. Symetryczna sytuacja ma miejsce w lewym dolnym kwadracie 2×2. Zauważmy, że mamy cztery możliwe przypadki:

I. W prawym górnym kwadracie 2×2 jest parzysta permutacja i w lewym dolnym kwadracie 2×2 jest parzysta permutacja.

II. W prawym górnym kwadracie 2×2 jest parzysta permutacja i w lewym dolnym kwadracie 2×2 jest nieparzysta permutacja.

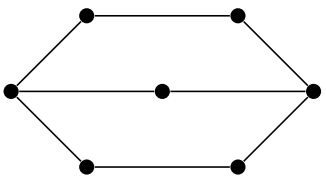
III. W prawym górnym kwadracie 2×2 jest nieparzysta permutacja i w lewym dolnym kwadracie 2×2 jest parzysta permutacja.

IV. W prawym górnym kwadracie 2×2 jest nieparzysta permutacja i w lewym dolnym kwadracie 2×2 jest nieparzysta permutacja.

W takim razie w przypadkach II i III układanka nie ma rozwiązania, ponieważ permutacja dla całego kwadratu 3×3 jest nieparzysta. W przypadku I możemy doprowadzić prawy górny kwadrat 2×2 do porządku (2, 3, 5, 6) za pomocą operacji `D` i lewy dolny kwadrat do porządku (4, 5, 7, 8) za pomocą operacji symetrycznej do `D`. W przypadku IV stosujemy operację `C`, która zmienia parzystość permutacji w prawym górnym kwadracie 2×2 i w lewym dolnym kwadracie 2×2, dając w efekcie przypadek I. Ostatecznie osiągamy układankę ułożoną w porządku (1, 2, 3, 4, 5, 6, 7, 8).

20.2 Uogólnienie „piętnastki" na dowolne grafy dwuspójne

Twierdzenie 14 (Wilson)**.** *Jeśli G jest prostym grafem dwuspójnym, różnym od cyklu oraz różnym od grafu Θ_0:*



to każdą permutację parzystą da się osiągnąć. Gdy graf jest niedwudzielny, to można wszystkie permutacje.

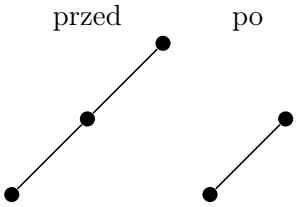
Mamy graf G (różny od cyklu i różny od Θ_0) etykietowany $\{1, \ldots, n-1\} \cup \{\emptyset\}$. $\Gamma(x)$ – zbiór permutacji powstałych z marszrut z x do x. Zbiór ten stanowi pewną podgrupę grupy permutacji:

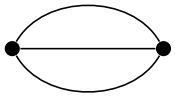
- łączność – złożenie marszrut $(P_1 \circ P_2) \circ P_3 = P_1 \circ (P_2 \circ P_3)$,
- element odwrotny: marszruta odwrotna (zapuszczona do tyłu).
- element neutralny: pusta marszruta.

Pojęcia

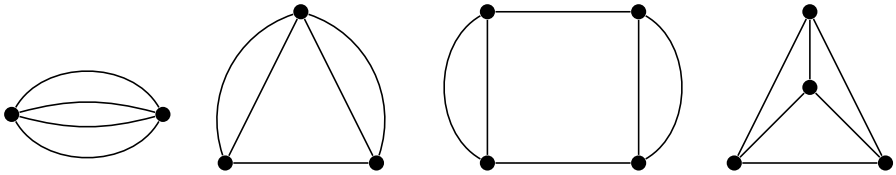
- G – graf dwuspójny. Niech $\beta(G) = m - n + 1$. Jeśli $\beta(G) = 1$, to graf jest cyklem. Jeśli $\beta(G) = 2$, to graf jest Θ-grafem. Dla $\beta(G) \geqslant 3$ krok indukcyjny.

- Ściągnięcie (zwinięcie łańcuchów krawędzi w pojedyncze krawędzie) grafu G w graf G' (potencjalnie dostaniemy multigraf):



Ściągnięcie Θ-grafu to 

Ściągnięcie grafu dla $\beta = 3$:



Lemat 2. *Niech G' ściągnięty dwuspójny graf o $\beta(G) \geqslant 3$. Istnieje krawędź w G', po usunięciu której graf pozostaje dwuspójny oraz jest różny od Θ_0 po rozciągnięciu.*

Grupa H działająca na zbiorze X jest tranzytywna, jeśli potrafi przeprowadzić każdy element na każdy element. Grupa $\Gamma(x)$ działająca na $V\backslash\{x\}$ jest tranzytywna, jeśli umiemy wstawić każdy element $V\backslash\{x\}$ pojedynczo wszędzie.

Twierdzenie 15. *$\Gamma(x)$ jest tranzytywna $\Leftrightarrow$ gdy graf G jest dwuspójny.*

Grupa H działająca na zbiorze X jest 2-tranzytywna, jeśli potrafi przeprowadzić dowolną parę różnych elementów z X na dowolną parę różnych elementów z X.

Lemat 3. *Rozważmy zbiór X, $|X| \geqslant 3$; weźmy dowolne u, $v \in X$. Zbiór 3-cykli postaci (uvx) dla $x \in X\backslash\{u, v\}$ generuje $alt(X)$ – permutacje parzyste.*

Lemat 4. *Niech Σ będzie zbiorem 3-cykli na zbiorze X $|X| \geqslant 3$. Oznaczmy przez Σ^* grupę generowaną przez Σ. Nastepujące warunki są równoważne:*

1. $\Sigma^* = alt(X)$
2. *Σ^* jest tranzytywna*

Lemat 5. *Niech H będzie 2-tranzytywną grupą permutacji (niekoniecznie wszystkich permutacji! może być podzbiór permutacji, które stanowią grupę), która zawiera co najmniej jeden 3-cykl. Wtedy $alt(X) \subseteq H$.*

Z powrotem do kroku indukcyjnego twierdzenia początkowego Mamy graf G, dwuspójny, $\beta(G) \geqslant 3$. Z któregoś lematu dostajemy „ucho” do usunięcia (łańcuch pomiędzy dwoma wierzchołkami x, $y \in V$), otrzymując graf H. Wiemy, że graf H jest dwuspójny, $\beta(H) \geqslant 2$, H jest różny od Θ_0 (z założenia), zatem możemy skorzystać z założenia indukcyjnego: mamy permutacje parzyste w H, czyli mamy 3-cykl.

Nasza para (y, z), gdzie z to dowolny wierzchołek w H różny od x i y. Bierzemy (a, b):

- b wsadzamy na z,
- jeśli a jest na uchu, to kręcimy cyklem, a trafia do y
- w p.p. a jest w G, to umiemy przełożyć (z, a) na (z, y), bo z założenia indukcyjnego H jest 2-tranzytywna

21 Problemy wagowe

Zajmiemy sie następującym problemem, dany jest zbiór Z odważników. Chcemy obliczyć minimalną liczbę odważników do zważenia zadaenj liczby x. Mamy dowolnie wiele odważników każdego typu.

21.1 Waga trójkowa

Niech Z będzie multizbiorem potęg trójki. Problem można zinterpretować jako zapis liczby naturalnej m w specjalnej reprezentacji trójkowej: w postaci tej bazą jest liczba 3 a cyfry są ze zbioru {-1, 0, 1}. Oznaczmy tę reprezentację przez $repr(m)$.

Mamy $repr(m) = (a_0, a_1, a_2, \ldots, a_k)$, gdzie $a_i \in \{-1,\ 0,\ +1\}$, $a_k \neq 0$ oraz

$$m = \sum_{i=0}^{k} a_i \cdot 3^i$$

Mając obliczoną reprezentację na prawej szalce kladziemy odważniki 3^i dla których $a_i = 1$, natomiast na lewej szalce kładziemy obiekt o masie m i odważniki 3^i dla $a_i = -1$.

Oznaczmy przez $[m]_3$ normalną reprezentację trójkową liczby ma

W algorytmie mamy do czynienia z bardzo dużymi liczbami, tak że dodtakowym utrudnieniem jest implemnatcja arytmetyki na reprezentacjach dziesiętnych i trójkowych dużych liczb.

Obserwacja. Rozwiązanie opiera się na następującym spostrzeżeniu. Załóżmy, że mamy liczby naturalne $m > 0$ i liczbę x taką, że $[x]_3$ składa się z samych $k+1$ jedynek oraz $x \geq m$. Jeśli $[x+m]_3 = (b_0, b_1, b_2, \ldots, b_k)$ to $repr(x) = (b_0 - 1, b_1 - 1, b_2 - 1, \ldots b_k - 1)$, z dokładnością do pominięcia nieznaczących ostatnich zer.

Opis algorytmu.

Obliczmy najmniejszą liczbę x, taką że $[x]_3$ składa się z samych jedynek oraz $x \geq m$.
Niech $y = m + x$.
Obliczmy $[y]_3 = (b_0, b_1, b_2, \ldots, b_k)$.
return $repr(m) = (b_0 - 1, b_1 - 1, b_2 - 1, \ldots b_k - 1)$

Przykład.
Niech $m = 42$. Mamy $[42]_3 = (0, 2, 1, 1)$, oraz $x = 121$, $[x]_3 = (1, 1, 1, 1, 1)$,
$y = m + x = 42 + 121 = 163$, gdzie $y = (1, 0, 0, 0, 2)$.
Tak więc $repr(42) = (0, -1, -1, -1,\ 1)$.
A więc na prawej szalce kładziemy 2^4 a na lewej obiekt o masie 42 i odważniki $2^1,\ 2^2,\ 2^3$.

21.2 Waga czwórkowa

W tym przypadku zbiorem odważników jest zbiór potęg czwórki.

Skoncentrujmy się najpierw na opracowaniu metody wyznaczenia minimalnej liczby odważników, jaka jest potrzebna do zważenia przedmiotu o zadanej wadze n. Oznaczmy tę liczbę $M(n)$.

Ponadto niech $n_{1\ldots i}$ oznacza liczbę ułożoną z pierwszych (najbardziej znaczących) i cyfr liczby n w zapisie czwórkowym. Konkretnie $n_{1\ldots i} = \lfloor n \cdot 4^{-(|n|-i)} \rfloor$, gdzie $|n|$ to ilość cyfr liczby n zapisanej w systemie czwórkowym.

Kluczową obserwacją prowadzącą do rozwiązania tego zadania jest to, że na podstawie $M(n_{1\ldots i})$ (ozn. x_i) **oraz** $M(n_{1\ldots i} + 1)$ (ozn. y_i) możemy w prosty sposób obliczyć $M(n_{1\ldots i+1})$ (czyli x_{i+1}) oraz $M(n_{1\ldots i+1} + 1)$ (czyli y_{i+1}). Jeśli przez n_i oznaczymy i-tą najbardziej znaczącą cyfrę liczby n, to sposób obliczania wygląda następująco:

$$\begin{aligned}
n_{i+1} = 0 &\implies (x_{i+1}, y_{i+1}) = (x_i, x_i + 1) \\
n_{i+1} = 1 &\implies (x_{i+1}, y_{i+1}) = (x_i + 1, \min(x_i + 2, y_i + 2)) \\
n_{i+1} = 2 &\implies (x_{i+1}, y_{i+1}) = (\min(x_i + 2, y_i + 2), y_i + 1) \\
n_{i+1} = 3 &\implies (x_{i+1}, y_{i+1}) = (y_i + 1, y_i)
\end{aligned}$$

Pierwszy przypadek dla x_{i+1} mówi m.in. tyle, że jeśli do dotychczas rozpatrywanej wartości dopisujemy zero, to optymalny sposób jej zważenia polega na zastąpieniu wszystkich odważników czterokrotnie cięższymi. Gdyby bowiem istniała lepsza strategia, wówczas i liczbę $n_{1...i}$ dałoby się odważyć przy pomocy mniej niż x_i odważników, gdyż moglibyśmy wszystkie odważniki tej strategii dla $n_{1...i+1}$ zastąpić czterokrotnie lżejszymi, gdyż nie ma w niej ani jednego odważnika o wadze 1.

Drugi przypadek mówi, że przy dopisaniu jedynki zamieniamy wszystkie odważniki na cięższe oraz dokładamy jeden o wadze 1 na szalkę przeciwną do tej, na której leży sztabka.

Trzeci przypadek jest nieco bardziej skompilowany. Jeśli dopisujemy dwójkę, to sytuację możemy rozwiązać na dwa sposoby: albo zamieniamy wszystkie odważniki na cięższe i dokładamy dwie jedynki, albo też bierzemy optymalną konfigurację odważającą $(n_{1...i} + 1)$, mnożymy wszystkie wagi przez 4, i dokładamy dwie jedynki na tą samą szalkę, na której leży sztabka.

Czwarty przypadek – gdy dopisujemy 3, mówi, że najpierw należy w sposób optymalny odważyć $(n_{1...i} + 1)$, następnie zamienić odważniki na cięższe, a następnie na szalce ze sztabką położyć odważnik jednostkowy.

Aktualizacja zmiennych y_i przebiega analogicznie. U podstaw przedstawionej metody leży obserwacja, iż jeśli ostatnią cyfrą rozpatrywanej liczby jest 0, to w optymalnej sytuacji zawsze użyjemy dokładnie 0 odważników jednostkowych. Podobnie, jeśli liczba kończy się na 1 lub 3, to użyjemy jednego odważnika jednostkowego. Jeśli za to kończy się na 2, to trzeba zawsze użyć dwóch odważników jednostkowych.

Pozostaje jeszcze policzyć, na ile sposobów można uzyskać optymalne konfiguracje. Ale to robimy w podobny sposób. Niech X_i oznacza liczbę optymalnych sposobów zważenia liczby $n_{1...i}$, a Y_i liczbę optymalnych sposobów dla $(n_{1...i} + 1)$. Wówczas wykorzystujemy następujące zależności, wynikające z powyższego opisu i wcześniejszych wzorów dla x_i oraz y_i.

$$\begin{aligned}
n_{i+1} = 0 &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i) \\
n_{i+1} = 1, x_i < y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i) \\
n_{i+1} = 1, x_i = y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i + Y_i) \\
n_{i+1} = 1, x_i > y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, Y_i) \\
n_{i+1} = 2, x_i < y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, Y_i) \\
n_{i+1} = 2, x_i = y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i + Y_i, Y_i) \\
n_{i+1} = 2, x_i > y_i &\implies (X_{i+1}, Y_{i+1}) = (Y_i, Y_i) \\
n_{i+1} = 3 &\implies (X_{i+1}, Y_{i+1}) = (Y_i, Y_i)
\end{aligned}$$

Warto również wspomnieć o warunkach początkowych:
$x_0 = 0$, $y_0 = 1$, $X_0 = 1$, $Y_0 = 1$.

Jeżeli pominąć koszt implementacji własnej arytmetyki dużych liczb, to powyższe rozwiązanie ma złożoność czasową $O(\log n)$.

21.3 Waga Fibonacciego

W tym przypadku zbiorem odważników jest zbiór liczb Fibonacciego $\{1, 2, 3, 5, 8, 13, \ldots\}$.

Algorytm działa w sposób zachłanny, w każdej iteracji bierzemy odważnik będący najbliższą liczbą Fibonacciego Fib_i względem danego przedmiotu o aktualnej wadze n. Inaczej mówiąc wykonujemy:

$wynik := 0$;
while $n \neq 0$ do
  wybierz i minimalizujące $|n - Fib_i|$
  $n := |n - Fib_i|$;
  $wynik := wynik + 1$

21.4 Identyfikacja fałszywej monety za pomocą ważeń

Mamy $n = (3^n - 3)/2$ monet, są jednakowe poza jedną, za pomocą n ważeń na wadze szalkowejz znajdujemy te monetę i stwierdzamy czy jest lżejsza, czy tez cięższa od normalnych.

Niech S_n bedzie zbiorem ciągów n cyfr trójkowych typu $1^+2...$, $2^+0..$ lub $0^+1..$ (kropki oznaczaja dowolny ciąg, ale razem ma być n cyfr).

Niech $S_n(k,0)$, $S_n(k,2)$ będzie zbiorem ciągów mających na k-tej pozycji 0 lub odpowiednio 2. Identyfikujemy monety z elementami zbioru S_n.

Algorytm ważenia

for $k = 1$ **to** n **do**

Wykonujemy ważenia zbioru $S_n(k,0)$ (lewa szalka) kontra $S_n(k,2)$ (prawa szalka).

Tworzymy ciąg cyfr trójkowych W o długości n.

Gdy lewa szalka cięższa zapisujemy na k-tej pozycji 0, jeśli prawa to 2, a jeśli wagi równe to 1.

Jeśli W jest identyfikatorem pewnej monety to jest to fałszywa mneta, wagę ma lżejszą, w przeciwnym wypadku jest to moneta o identyfikatorze $\bar{W}$, wage ma większą niz normalna.
$\bar{W}$ oznacza zamiane $2 \leftrightarrow 0$ na każdej pozycji ciągu W.

21.5 Problemy wagowe Munchausena

W XVIII wieku żył baron Hieronymus Carl Friedrich von Münchhausen (znany też jako Munchausen), który wsławił się opowiadaniem różnych bardzo dziwnych opowieści. Między innymi znany jest z wprowdzenia następującego problemu typu wagowego.

Mamy wagę szalkową i n elementów o wagach 1, 2, ..., n. Baron Munchausen wie który element jest który i chce udowodnić widowni za pomocą jedynie wagi szalkowej, jaka jest waga pewnego pojedynczego elementu, który może sobie dowlnie wybrać. Minimalną liczbę potrzebnych ważeń oznaczmy przez $b(n)$.

Munchausen podał rozwiązanie dla $n = 8$ – wystarczy jedno nastepujące ważenie:

$$1 + 2 + 3 + 4 + 5 = 7 + 8$$

W ten sposób zidentyfikujemy element o wadze 6. Mamy więc $b(8) = 1$.

W ogólnym przypadku działa następujące twierdzenie:

Twierdzenie 16. $1 \leqslant b(n) \leqslant 2$.

Problem można skomplikować następująco. Niech $a(n)$ będzie minimalną liczbą ważeń pozwalającą na identyfikację wszytkich elementów. Dalej zakładamy, że mają one wagi 1, 2, ..., n. Na przykład $a(6) = 2$ ponieważ możemy wykonać dwa ważenia, jedno po drugim:

$$1 + 2 + 5 < 3 + 6 \quad 1 + 3 < 5.$$

lub dwa ważenia:

$$1 + 2 + 3 = 6 \quad 1 + 6 < 3 + 5$$

Zachodzi ogólne nietrywialne twierdzenie.

Twierdzenie 17. $a(n) \leqslant 2\lceil \log n \rceil$.

22 Skojarzenia i systemy różnych reprezentantów

$G = (V, E)$ – graf. $M \subseteq V$ to skojarzenie, jeśli żadne dwie krawędzie w M nie mają wspólnego końca. Zwykle szukamy największego skojarzenia. Skojarzenie, które pokrywa wszystkie wierzchołki to skojarzenie doskonałe.

22.1 System Różnych Reprezentantów

$\mathcal{I} = \langle S_1, S_2, \ldots, S_m \rangle$ – rodzina niepustych skończonych zbiorów. Niech $\mathcal{I}' \subseteq \mathcal{I}$, oznaczmy przez $S_{\mathcal{I}'}$ podsystem $\langle S_i : i \in \mathcal{I}' \rangle$. Przez $S_{\mathcal{I}'} \cup S_{I''}$ oznaczmy podsystem $\langle S_i : i \in \mathcal{I}' \cup \mathcal{I}'' \rangle$. Podsystem $S_{\mathcal{I}'}$ jest *krytyczny* gdy $|\sum_{i \in \mathcal{I}'} S_i| = |\mathcal{I}'|$. Czy istnieje taki wektor $\langle a_1, \ldots, a_m \rangle$ parami różnych elementów, że $a_i \in S_i$ dla $i = 1, \ldots, m$?

Twierdzenie 18 (Hall)**.** *Rodzina zbiorów $\mathcal{I}$ posiada SRR gdy spełniony jest tzw. warunek Halla:*

$$(\forall_{\mathcal{I}'\subseteq\mathcal{I}}) \quad \left|\sum_{i\in\mathcal{I}'} S_i\right| \geqslant \left|\mathcal{I}'\right|.$$

Fakt.

1. *Jeśli dwa podsystemy są krytyczne, to ich suma też.*
2. *Jeśli system spełnia warunek Halla oraz krotność każdego elementu jest* 1*, to wybierając dowolny element z każdego zbioru otrzymujemy SRR (system różnych reprezententów).*

Dowód. Udowodnimy, że jeśli system $\langle S_1, S_2, \ldots S_n\rangle$ spełnia warunek Halla, oraz $a \in S_1 \cap S_2$, to system spełnia warunek Halla po usunięciu a z S_1 lub a z S_2. Przypuśćmy, że tak nie jest. Przykładowo, niech

$$|S_1 - a,\, S_3,\, S_4,\, S_5,\, S_6| < 5, \quad |S_1 - a,\, S_3,\, S_4,\, S_7,\, S_8| < 5$$

Wtedy podsystemy $\langle S_3, S_4, S_5, S_6\rangle$, $\langle S_3, S_4, S_7, S_8\rangle$ są krytyczne oraz

$$S_1 - a \subseteq S_3 \cup S_4 \cup \ldots \cup S_8$$

No ale na mocy wcześniejszego faktu mamy, że moc sumy podsystemów $\langle S_3, S_4, S_5, S_6\rangle$, $\langle S_3, S_4, S_7, S_8\rangle$ jest mniejsza równa 6 (suma indeksów), nawet jak dodamy element a to $|S_1 \cup S_2 \ldots S_8| \leqslant 6 + 1 = 7$, co przeczy oryginalnemu założeniu o warunku Halla. Zatem system sprowadza się do rodziny zbiorów, w której każdy element występuje tylko raz, a zatem system posiada SRR. □

Załóżmy, że spełniony jest warunek Halla. Można powyższe rozumowanie wyrazić algorytmem niewielomianowym:

```
1 while istnieje element a należący do dwóch różnych zbiorów do
2 |   Usuń a z jednego z tych dwóch zbiorów tak, aby warunek Halla nadal zachodził (to jest zawsze
  |     możliwe);
3 end
4 Usuń z każdego zbioru wszystkie elementy, poza jednym dowolnym;
5 Wypisz otrzymany system jako SRR dla wejściowego systemu;
```

22.2 Oszacowania na liczbę SRR

Załóżmy, że $\min_i |S_i| = t$ i spełniony jest warunek Halla.

Teza 1. *$\mathcal{I}$ posiada*

$$\begin{array}{lll} t! & SRR & gdy\ t \leqslant m \\ \frac{t!}{(t-m)!} & SRR & gdy\ t > m \end{array}$$

22.3 Grafy regularne

$G = (V, E)$ dwudzielny, regularny stopnia $\Delta \geqslant 3$. Chcemy znaleźć skojarzenie doskonałe.

```
1  przypisz każdej krawędzi w grafie wagę 1;
2  while podgraf rozpięty na krawędziach z dodatnimi wagami nie jest lasem do
3  |   C := dowolny cykl (elementarny);
4  |   M1, M2 – doskonałe skojarzenia na C
5  end
6  if waga M1 ⩾ waga M2 then
7  |   od wagi każdej krawędzi z M1 odejmij 1, do wagi każdej krawędzi z M2 dodaj 1
8  else
9  |   odejmujemy z M2 i dodajemy do M1
10 Niezmiennik pętli: suma wag krawędzi wychodzących z wierzchołka wynosi Δ.
```

22.4 Najliczniejsze skojarzenie

$G = (V, E)$ graf prosty, spójny. $M \subset E$ – skojarzenie, gdy żadne dwie krawędzie nie mają wspólnego końca. Znaleźć M o największej liczności. Większość algorytmów znajdujących największe skojarzenie działa przez szukanie ścieżek powiększających — gdy taka nie istnieje, skojarzenie jest największe.

$G = (W, K, E)$ – graf dwudzielny, gdzie W to wiersze $\{1, 2, \ldots, w\}$, K kolumny $\{1, 2, \ldots, k\}$ (reprezentacja macierzowa). Zakładamy bez straty ogólności $|W| \geqslant |K|$. Budujemy drzewa rozpinające ukorzenione w wierszach. Silne drzewo rozpinające to takie drzewo rozpinające, w którym każda kolumna-liść musi być synem korzenia. Jeśli T to silne drzewo rozpinające, to niech $Q(T)$ oznacza zbiór kolumn-liści.

Nie każdy graf dwudzielny ma silne drzewo rozpinające. Modyfikujemy graf $G \to G^a$:

- $G^a = (W \cup \{0\}, K, E^a)$
- $E^a = E \cup \{(0, j) : j \in K\}$

Korzeniem silnego drzewa rozpinającego T będzie zawsze zero. Kolumnę h nazwiemy kandydatem, gdy:

1. posiada dwóch synów w T,
2. istnieje ścieżka w G z h do pewnego liścia-kolumny taka, że pierwsza krawędź prowadząca poza drzewo $T(h)$ jest postaci $wiersz \to kolumna$.

gdzie $T(h)$ to poddrzewo T ukorzenione w h. Niech $C(T)$ to będzie zbiór kandydatów. Przez $Z(T)$ oznaczamy skojarzenie indukowane przez T. Jeśli $C(T)$ jest puste, to skojarzenie jest największe.

Definiujemy operację *pivot* na krawędzi (l, h) typu $wiersz \to kolumna$, gdzie h jest korzeniem $T(h)$, a l jest jego rodzicem w drzewie T. Operacja polega na usunięciu krawędzi (l, h) z T i dodaniu zamiast tego krawędzi z warunku drugiego istnienia kandydata.

Obserwacja 8. *Jeżeli h ma $\geqslant 2$ synów, to po wykonaniu zamiany mamy nadal silne drzewo rozpinające.*

Obserwacja 9. *Rozmiar skojarzenia indukowanego przez nowe drzewo **może** być o 1 większe od poprzedniego.*

Dominatory: kolumny o co najmniej dwóch synach, nie posiadające właściwych przodków o tej własności. Oznaczenie: $D(T)$. Jeśli $D(T)$ puste, to mamy najliczniejsze skojarzenie. $W^D(K^D)$ – wiersze zdominowane. Powiemy, że silne drzewo rozpinające jest zamknięte, gdy

1. $D(T) = \emptyset$ lub $Q(T) = \emptyset$
2. lub z faktu, że $a \in W^D$ oraz $(a, b) \in G$ wynika, że $b \in K^D$.

Twierdzenie 19. *Jeżeli T jest drzewem zamkniętym, to każde skojarzenie indukowane przez T jest największe.*

22.5 Kolorowanie

Dla danego nieskierowanego grafu $G = (V, E)$ szukamy jego liczby chromatycznej $\chi(G)$, czyli minimalnej liczby różnych kolorów, jakich potrzeba na pokolorowanie krawędzi grafu w taki sposób, żeby w żadnym wierzchołku nie spotykały się dwie krawędzie tego samego koloru. Czasami oprócz liczby chromatycznej chcemy wyznaczyć samo kolorowanie – nie zawsze jest to proste. Wprowadzamy $\Delta(G)$ jako maksymalny stopień wierzchołka w G.

Twierdzenie 20 (Vizing)**.** *W każdym prostym grafie (bez pętli i nie-multigraf) G zachodzi $\chi(G) \leqslant \Delta(G) + 1$.*

Dowód indukcyjny: przyjmiemy, że istnieje $(\Delta + 1)$-kolorowanie grafu G', gdzie G' to graf G pozbawiony dowolnego wierzchołka v. Wtedy wystarczy pokazać, że z istnienia takiego kolorowania grafu G' można uzyskać $(\Delta+1)$-kolorowanie grafu G. Powiemy, że dany kolor α jest *dostępny* w wierzchołku v, jeśli nie istnieje krawędź incydentna z v pokolorowana kolorem α. Pomocniczy...

Lemat 6. *Niech G będzie grafem prostym, v jakimś jego wierzchołkiem i $k \geqslant \Delta(G)$ pewną liczbą całkowitą. Załóżmy, że graf $G' = G - v$ ma k-kolorowanie krawędziowe takie, że każdy sąsiad (bez co najwyżej jednego) v z grafu G ma przynajmniej dwa kolory dostępne. Pojedynczy wyjątek może mieć tylko jeden kolor dostępny. Wtedy graf G także jest k-kolorowalny.*

Lemat jest prawdziwy gdy $k = \Delta + 1$, bo wtedy każdy sąsiad v w grafie G' ma stopień co najwyżej $\Delta - 1$. Twierdzenie Vizinga można rozszerzyć na multigrafy bez pętli.

Twierdzenie 21 (Brooks)**.** *Jeśli G jest grafem spójnym, to $\chi(G) \leqslant \Delta(G)$, chyba że G jest grafem pełnym albo cyklem długości nieparzystej - wtedy $\chi(G) = \Delta(G) + 1$.*

22.6 Algorytmy kolorowania dla grafów dwudzielnych

22.6.1 Algorytm najprostszy $\mathcal{O}(nm)$

Dla każdego wierzchołka grafu dwudzielnego G ustalamy zbiór kolorów dostepnych $\{1, 2, \ldots, \Delta\}$. Będziemy po kolei dokonywali kolorowania każdej krawędzi grafu. Powiedzmy, że bierzemy teraz krawędź (u, v). W wierzchołku u na pewno nie użyliśmy jeszcze wszystkich kolorów, więc bierzemy dowolny kolor α dostepny w u. Podobnie bierzemy dowolny kolor β dostępny w v. Jeśli $\alpha = \beta$, to już jest dobrze: kolorujemy krawędź tym kolorem i usuwamy go ze zbioru kolorów dostępnych w wierzchołkach u i v.

Jeśli jednak $\alpha \neq \beta$, to szukamy ścieżki powiększającej w grafie G postaci $v \to u_1 \to v_1 \to u_2 \to \ldots$ takiej, że krawędzie, po których przechodzimy, są na przemian koloru α i β. Po znalezieniu takiej ścieżki odwracamy kolorowanie krawędzi (krawędzie koloru α kolorujemy na β i odwrotnie), dzięki czemu w wierzchołku v „zwolnił się" kolor α i mamy przypadek pierwszy.

22.7 Algorytm $\mathcal{O}(\Delta m)$

Załóżmy, że graf dwudzielny G jest k-regularny. Z grafami nieregularnymi za moment sobie poradzimy. Z twierdzenia Halla wynika, że każdy k-regularny graf dwudzielny ma skojarzenie doskonałe. Indukcyjnie można znaleźć skojarzenie doskonałe w grafie k-regularnym, usunąć je i następnie zajmować się grafem $(k-1)$-regularnym, aż do usunięcia wszystkich krawędzi. W ten sposób otrzymamy k rozłącznych krawędziowo skojarzeń doskonałych. Jeśli dla każdego z tych skojarzeń dobierzemy inny kolor, to mamy k-kolorowanie grafu.

Niestety znajdowanie kolejnych skojarzeń nie jest wystarczająco szybkie. Użyjemy algorytmu typu „dziel i zwyciężaj":

1. Jeśli k jest nieparzyste, znajdź skojarzenie doskonałe i usuń je – możemy to zrobić w czasie $\mathcal{O}(km)$. W ten sposób redukujemy problem do grafu $(k-1)$-regularnego.

2. Jeśli k jest parzyste, znajdź cykl Eulera. Podziel krawędzie cyklu na dwa zbiory (krawędzie o parzystych i nieparzystych numerach), otrzymując w ten sposób dwa $\frac{k}{2}$-regularne podgrafy.

Jeśli mamy graf $H = (U, V;\ E)$, który nie jest regularny, to weźmy $d = \Delta(H)$ i stworzymy graf G, który będzie d-regularny. Z kolorowania grafu G można odtworzyć kolorowanie grafu H. Działamy następująco:

1. Dopóki jest więcej niż jeden wierzchołek w U (lub odpowiednio V) stopnia co najwyżej $\frac{d}{2}$, to wybierz dowolne dwa z nich i sklej je w jeden wierzchołek.

2. Dodaj „fałszywe" wierzchołki i krawędzie, żeby powstał graf d-regularny. Można w ten sposób uzyskać multigraf, ale to nie przeszkadza.

Literatura

[1] Aaron F. Archer: *A Modern Treatment of the 15 Puzzle*

[2] J. A. Bondy, *Short Proofs of Classical Theorems*

[3] Bette Bultena, Frank Ruskey: *Transition Restricted Gray Codes*, 1995

[4] Peter Eades, Michael Hickey, Ronald C. Read: *Some Hamilton Paths and a Minimal Change Algorithm*, Journal of the Association for Computing Machinery, Vol. 31, No. 1, January 1984

[5] Alon Itai: *Producing Permutations and Combinations in Lexicographical Order*, 2002

[6] E. Moreno, *On the theorem of Fredricksen and Maiorana about de Bruijn sequences*, Adv. in Appl. Math., 2004

[7] D. E. Knuth, *The Art of Computer Programming*, Volume 4, Fascicle 2, Addison-Wesley, 2005

[8] Richard E. Korf, Ariel Felner: *Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem*

[9] H. Fredricksen, J.Maiorana, *Necklaces of beads in k colors and k-ary de Bruijn sequences*, Discrete Math. 23, 1978

[10] Ian Parberry: *An efficient algorithm for the Knight's tour problem*, 1994

[11] Ian Parberry: *A Real-Time Algorithm for the* (n^2-1)*-Puzzle*, 1997

[12] Andrzej Proskurowski, Frank Ruskey: *Generating Binary Trees by Transpositions*, 1995

[13] J. Radoszewski, *Generowanie minimalnych leksykograficznie ciągów de Bruijna za pomocą słów Lyndona*, praca mgr. (2008)

[14] Frank Ruskey, Aaron Williams: *An Explicit Universal Cycle for the* $(n-1)$*-Permutations of an* n*-Set*

[15] Frank Ruskey, Aaron Williams: *The Coolest Way to Generate Combinations*

[16] Alexander Schrijver, *Bipartite edge-colouring in* $\mathcal{O}(\Delta m)$ *time*

[17] Allen J. Schwenk: *Which Rectangular Chessboards Have a Knight's Tour?*, Mathematics Magazine, Vol. 64, No. 5, December 1991

[18] Mario Szegedy: *In How Many Steps the k Peg Version of the Towers of Hanoi Game Can Be Solved?*, 1999

[19] Richard M. Wilson: *Graph Puzzles, Homotopy, and the Alternating Group*, 1973

[20] *Generating Balanced Parentheses and Binary Trees by Prefix Shifts*, 2008

[21] http://www.math.psu.edu/vstein/alg/antheory/preprint/andrews/chapter.pdf

[22] http://hkumath.hku.hk/~mks/EulerHeuristicReasoning.pdf