Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Jakub Radoszewski
Nr albumu: 214565

Generowanie minimalnych leksykograficznie ciągów de Bruijna za pomocą słów Lyndona

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
prof. dr. hab. Wojciecha Ryttera
Instytut Informatyki UW

Wrzesień 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W niniejszej pracy zaprezentowano pełniejszy i dokładniejszy w stosunku do oryginalnego dowód twierdzenia Fredricksena i Maiorany, że konkatenacja słów Lyndona o długościach dzielących n w kolejności leksykograficznej daje minimalny leksykograficznie ciąg de Bruijna rzędu n. Nowym wynikiem pracy jest implementacja i analiza algorytmu opartego na tym twierdzeniu, wyznaczającego kolejne litery wspomnianego ciągu w pesymistycznej złożoności czasowej $O(1)$. Dodatkowo, techniki zastosowane w dowodzie twierdzenia zostały w pracy skutecznie wykorzystane do analizy struktury transformaty Burrowsa-Wheelera minimalnych leksykograficznie ciągów de Bruijna, której owocem jest efektywny algorytm wyznaczania dowolnej litery tej transformaty. Poza tym praca ma charakter przeglądowy i zawiera opis znanych faktów i algorytmów związanych ze słowami Lyndona i ciągami de Bruijna, w szczególności z ich zliczaniem.

Słowa kluczowe

ciąg de Bruijna, słowa Lyndona, cykl Eulera, transformata Burrowsa-Wheelera

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

G. Mathematics of Computing
G.2 DISCRETE MATHEMATICS
G.2.1 Combinatorics—Combinatorial algorithms
G.2.2 Graph Theory—Graph algorithms

F. Theory of Computation
F.2 ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY
F.2.2 Nonnumerical Algorithms and Problems

Tytuł pracy w języku angielskim

Generation of lexicographically minimal de Bruijn sequences with prime words

Spis treści

Wprowadzenie

Ciągiem de Bruijna rzędu n nad alfabetem Σ nazywamy takie słowo cykliczne, w którym każde n-literowe słowo nad Σ występuje dokładnie raz. Ciągi te zostały wprowadzone przez de Bruijna w pracy [8]. Od tego czasu znaleziono wiele ich zastosowań, między innymi do konstrukcji trybów pamięci w komputerach i innych urządzeniach, w modelach sieciowych, w algorytmach związanych z DNA, do generowania liczb pseudolosowych czy w nowoczesnych systemach kryptograficznych z kluczem publicznym (patrz [2], [6], [15] i [23]). Warto wspomnieć, że ciągi de Bruijna doczekały się pewnych uogólnień, na przykład na języki, w których pewne słowa są zabronione ([17], [19], [21]), oraz na inne niż słowa struktury kombinatoryczne, jak permutacje, partycje czy podzbiory zbioru o stałej mocy ([6]).

Najprostszym algorytmem generowania ciągów de Bruijna jest wyznaczanie cykli Eulera w grafie de Bruijna — metodę tę odkryli niezależnie dwaj naukowcy, de Bruijn [8] oraz Good [12]. Za pomocą utożsamienia ciągów z cyklami można wyznaczyć wzór na liczbę ciągów de Bruijna danego rzędu nad ustalonym alfabetem — patrz [8], [20]. Znanych jest też wiele metod generowania pewnych szczególnych ciągów de Bruijna, których przegląd można znaleźć w książce [15] oraz w pracy [9].

Spośród ciągów de Bruijna można wyróżnić ciągi minimalne leksykograficznie w swoich klasach. Nieco zaskakującym faktem związanym z tymi ciągami jest twierdzenie Fredricksena i Maiorany, że konkatenacja słów Lyndona o długościach dzielących n w kolejności leksykograficznej daje minimalny leksykograficznie ciąg de Bruijna rzędu n [11]. Twierdzenie to może zostać użyte do konstrukcji metody generowania ciągów de Bruijna dzięki wykorzystaniu algorytmu FKM — Fredricksena, Kesslera i Maiorany ([10], [11]) — generowania słów Lyndona składających się z co najwyżej n liter. Zastosowanie algorytmu FKM prowadzi przy tym do bardzo efektywnej metody generowania ciągów de Bruijna, w której każda litera jest wyznaczana w średnim czasie stałym [22].

W niniejszej pracy przedstawiono pełniejszy i dokładniejszy w stosunku do oryginalnego, ale także i innych dostępnych — niejasnego, a wręcz niepoprawnego, a także pomijającego kwestię leksykograficznej minimalności z pracy [18] oraz zaledwie naszkicowanego z książki [15] — dowód twierdzenia Fredricksena i Maiorany. Zaproponowany został także sposób usprawnienia algorytmu FKM, dzięki któremu każda kolejna litera minimalnego ciągu de Bruijna zostaje wygenerowana on-line w pesymistycznej złożoności czasowej $O(1)$, a także pewien bardzo prosty algorytm zachłanny generowania tego ciągu, który jest oparty bezpośrednio na przedstawionym dowodzie twierdzenia (alternatywne sformułowanie drugiego z tych algorytmów można znaleźć w [19]). Dodatkowo, techniki zastosowane w tym dowodzie zostały skutecznie wykorzystane do analizy struktury transformaty Burrowsa-Wheelera (BWT opisana jest w [3]) minimalnych leksykograficznie ciągów de Bruijna, której owocem jest efektywny algorytm wyznaczania dowolnej litery tej transformaty. Poza tym praca ma charakter przeglądowy i zawiera opis znanych faktów i algorytmów związanych ze słowami Lyndona i ciągami de Bruijna, w szczególności z ich zliczaniem.

W rozdziale 1 pracy zawarte są niezbędne do dalszych rozważań definicje związane ze sło-

wami oraz własności słów. W rozdziale 2 znajduje się opis i dowód poprawności algorytmu FKM, poprzedzony charakterystyką słów Lyndona oraz dowodami własności prefiksów Lyndona. W rozdziale 3 omówiono znane algorytmy generowania ciągów de Bruijna, w tym dwie wersje algorytmu wyznaczania cyklu Eulera w grafie de Bruijna, a także przedstawiono rozumowanie prowadzące do wzoru na liczbę ciągów de Bruijna danego rzędu nad zadanym alfabetem.

W rozdziale 4 zaprezentowano dowód twierdzenia Fredricksena i Maiorany. Został on podzielony na dwie części; pierwsza związana jest z poprawnością konstrukcji ciągu de Bruijna za pomocą słów Lyndona (sekcja 4.2), a druga — z minimalnością skonstruowanego ciągu (sekcja 4.3). Dalej przedstawiony jest algorytm oparty na tym twierdzeniu, wyznaczający kolejne litery minimalnego leksykograficznie ciągu de Bruijna w pesymistycznej złożoności czasowej $O(1)$ (sekcja 4.4), a także pewien bardzo prosty algorytm zachłanny generowania tego ciągu (sekcja 4.5). Wreszcie rozdział 5 zawiera opis struktury transformaty Burrowsa-Wheelera minimalnych ciągów de Bruijna wraz z efektywnym algorytmem pozwalającym na wygenerowanie dowolnej litery tej transformaty.

Rozdział 1

Podstawowe definicje i własności

1.1. Terminologia związana ze słowami

Niech dany będzie niepusty skończony zbiór Σ. Zbiór ten nazywamy *alfabetem*, a jego elementy — *literami* albo *symbolami*. Skończone ciągi liter z Σ nazywamy *słowami nad alfabetem* Σ. Zbiór wszystkich słów nad alfabetem Σ oznaczamy przez Σ^*. Zazwyczaj będziemy utożsamiać litery ze zbioru Σ z kolejnymi liczbami całkowitymi nieujemnymi: $0, 1, \ldots, M-1$, gdzie $M = |\Sigma|$. Dla uproszczenia dalszych zapisów literę $M-1$ będziemy także oznaczać przez Z (przez analogię do ostatniej litery zwykłego alfabetu).

Długością słowa u (oznaczaną przez $|u|$) nazywamy liczbę liter, z których składa się u. Zbiór wszystkich słów nad alfabetem Σ o długości n oznaczamy przez Σ^n. Jedyne słowo o długości 0 oznaczamy przez ϵ. Jeżeli $u = a_1 a_2 \ldots a_k$, to przez $u[i]$ (dla $1 \leqslant i \leqslant k$) oznaczamy i-tą literę słowa u, czyli a_i, natomiast przez $u[i..j]$ — słowo $a_i a_{i+1} \ldots a_j$. Jeżeli $i > j$, to zakładamy, że $u[i..j] = \epsilon$.

Podsłowem słowa $u = a_1 a_2 \ldots a_k$ nazwiemy dowolne słowo postaci $u[i..j]$ dla $1 \leqslant i \leqslant j \leqslant k$ albo słowo puste ϵ. Pewne rodzaje podsłów będą dla nas szczególnie istotne. *Prefiksem* słowa u nazywamy dowolne spośród słów $u[1..i]$ dla $i \in \{0, 1, \ldots, k\}$, natomiast *sufiksem* — dowolne spośród słów $u[i..k]$ dla $i \in \{1, \ldots, k, k+1\}$. Prefiks lub sufiks słowa u nazywamy *właściwym*, jeżeli jego długość jest liczbą dodatnią mniejszą niż $|u|$.

Dla słów $u = a_1 \ldots a_n$ i $v = b_1 \ldots b_k$ definiujemy operację *sklejenia (konkatenacji)*:

$$u \cdot v = a_1 \ldots a_n b_1 \ldots b_k.$$

Konkatenację słów u i v będziemy często zapisywać jako uv.

Mówimy, że słowo v jest *obrotem cyklicznym (rotacją cykliczną, równoważnikiem cyklicznym)* słowa u, jeżeli istnieje takie całkowite $1 \leqslant i \leqslant k$, że $v = u[i+1..k]u[1..i]$.

Zakładając, że zbiór Σ jest liniowo uporządkowany przez relację $\leqslant$, na zbiorze słów nad alfabetem Σ definiujemy *porządek leksykograficzny* w następujący sposób.

Definicja 1.1.1. *u jest **nie większe leksykograficznie** niż v, co zapisujemy jako $u \leqslant v$, jeżeli:*

- *u jest prefiksem v lub*
- *długość k najdłuższego wspólnego prefiksu słów u i v jest mniejsza niż* $\min(|u|, |v|)$ *oraz* $u[k+1] < v[k+1]$.

Można pokazać, że porządek leksykograficzny jest porządkiem liniowym na Σ^*. W naturalny sposób definiuje się następujące relacje stowarzyszone z porządkiem leksykograficznym: $<$, $\leqslant$ oraz $>$.

Powiemy, że liczba $p \in \{1, \ldots, n\}$ jest *okresem* słowa $u = a_1 a_2 \ldots a_n$, jeżeli $a_i = a_{p+i}$ dla $1 \leqslant i \leqslant n-p$. Terminem *okres* określamy także często słowo $u[1..p]$. Z kontekstu będzie zawsze jasno wynikać, w jakim znaczeniu użyty jest ten termin. Słowo v nazywamy *rozszerzeniem okresowym* niepustego słowa u, jeżeli jest ono równe słowu złożonemu z pierwszych $|v|$ liter nieskończonego ciągu $uuu\ldots$

Mówimy, że słowo u jest *k-tą potęgą* słowa v (co oznaczamy przez $u = v^k$), jeżeli u powstaje przez k-krotną konkatenację słowa v ze sobą. W takim wypadku v nazywamy *pierwiastkiem* słowa u. Zauważmy, że każdy pierwiastek słowa jest jego okresem, ale niekoniecznie każdy okres słowa jest jego pierwiastkiem. Najkrótszy spośród pierwiastków danego słowa nazywamy *pierwiastkiem pierwotnym* tego słowa. Mówimy, że słowo u jest *pierwotne*, jeżeli pierwiastkiem pierwotnym u jest u. Łatwo zauważyć, że pierwiastek pierwotny każdego słowa jest słowem pierwotnym.

1.2. Podstawowe własności słów

Niniejszą sekcję zaczynamy od trzech całkiem oczywistych własności porządku leksykograficznego, które pozostawiamy bez dowodu.

Fakt 1.2.1. *Trzy istotne własności porządku leksykograficznego:*

- *Jeżeli $u < v$, to dla dowolnego $w \in \Sigma^*$ zachodzi $wu < wv$.*
- *Jeżeli u nie jest prefiksem v i $u < v$, to dla dowolnych $w, z \in \Sigma^*$ zachodzi $uw < vz$.*
- *Jeżeli $uv \leqslant w \leqslant uz$, to u jest prefiksem w.*

Konkatenacja słów jest działaniem łącznym, ale niekoniecznie przemiennym. Można pokazać przez łatwą indukcję (patrz np. [16]), że:

Fakt 1.2.2. *Słowa u i v są przemienne, tj. $uv = vu$, wtedy i tylko wtedy, gdy są potęgami tego samego słowa.*

Ten prosty fakt ma wiele nietrywialnych konsekwencji, np.:

Fakt 1.2.3. *Jeżeli u jest słowem pierwotnym, to posiada ono $k = |u|$ różnych równoważników cyklicznych.*

Dowód. Gdyby pewne dwa równoważniki cykliczne u, np.

$$u[i+1..k] \cdot u[1..i] \text{ oraz } u[j+1..k] \cdot u[1..j] \text{ (dla } 1 \leqslant i < j \leqslant k)$$

były sobie równe, to przerzuciwszy kolejno i liter z końca każdego z tych równoważników na początek, otrzymalibyśmy, że u jest równe pewnej nietrywialnej rotacji cyklicznej u, czyli

$$u = u[l+1..k] \cdot u[1..l] \tag{1.1}$$

dla pewnego $1 \leqslant l < n$. Równość (1.1) można zapisać równoważnie jako:

$$u[1..l] \cdot u[l+1..k] = u[l+1..k] \cdot u[1..l], \tag{1.2}$$

co oznaczałoby, że słowa $u[1..l]$ i $u[l+1..k]$ są przemienne, czyli na mocy Faktu 1.2.2 istniałyby $t \in \Sigma^*$ oraz $a, b \geqslant 1$, dla których

$$u[1..l] = t^a \text{ oraz } u[l+1..k] = t^b.$$

Z tego wynikałoby, że $u = t^{a+b}$, gdzie $a + b \geqslant 2$, czyli u nie byłoby pierwotne, wbrew założeniu. ■

Następujący fakt zawiera jeszcze jedną istotną własność obrotów cyklicznych słów.

Fakt 1.2.4. *Jeżeli słowa u oraz v są swoimi równoważnikami cyklicznymi, to długości ich pierwiastków pierwotnych są takie same.*

Dowód. Oznaczmy $|u| = |v| = n$ i niech a oraz b o długości odpowiednio $|a| = k$ i $|b| = l$ będą pierwiastkami pierwotnymi odpowiednio u oraz v. W dowodzie wystarczy skupić się na przypadku, gdy $v = u[2..n] \cdot u[1]$. Skoro $u = a^\alpha$ dla pewnego $\alpha \geqslant 1$, to

$$\begin{aligned} v &= a[2..k] \cdot a^{\alpha-1} \cdot a[1] \\ &= a[2..k] \cdot (a[1] \cdot a[2..k])^{\alpha-1} \cdot a[1] \\ &= (a[2..k] \cdot a[1])^\alpha, \end{aligned}$$

co pokazuje, że $|b| \leqslant |a|$. Podobnie, skoro $v = b^\beta$ dla pewnego $\beta \geqslant 1$, to

$$\begin{aligned} u &= b[l] \cdot b^{\beta-1} \cdot b[1..l-1] \\ &= b[l] \cdot (b[1..l-1] \cdot b[l])^{\beta-1} \cdot b[1..l-1] \\ &= (b[l] \cdot b[1..l-1])^\beta, \end{aligned}$$

skąd $|a| \leqslant |b|$. Ostatecznie mamy więc $|a| = |b|$. ■

Rozdział 2

Słowa Lyndona

2.1. Definicja i własności

Definicja 2.1.1. *Niepuste słowo nazywamy* ***słowem Lyndona*** *(ang.* ***prime word****), jeżeli jest pierwotne i leksykograficznie najmniejsze w klasie swoich równoważników cyklicznych.*

Przykład 1. `001001` nie jest słowem Lyndona, gdyż nie jest pierwotne (`001001`=`001`^{2}). `010011` nie jest słowem Lyndona, gdyż istnieje równoważnik cykliczny tego słowa (`001101`), który jest leksykograficznie mniejszy od niego. Przykładami słów Lyndona są za to:

- wszystkie słowa jednoliterowe;
- `011111`, `001101` oraz `0101011`.

Można udowodnić, że następująca, alternatywna definicja słów Lyndona jest równoważna Definicji 2.1.1.

Stwierdzenie 2.1.1 (Równoważna definicja słów Lyndona)**.** *Niepuste słowo $w \in \Sigma^*$ jest słowem Lyndona wtedy i tylko wtedy, gdy jest mniejsze leksykograficznie od wszystkich swoich sufiksów właściwych.*

Słowa Lyndona mają szereg własności związanych z rozkładami według operacji konkatenacji. Przede wszystkim istnieje następująca charakteryzacja słów Lyndona.

Twierdzenie 2.1.2. *Niepuste słowo $w \in \Sigma^*$ jest słowem Lyndona wtedy i tylko wtedy, gdy $w \in \Sigma$ lub istnieją takie słowa Lyndona l oraz m, że $w = lm$ oraz $l < m$.*

Opisany rozkład nie musi być jednoznaczny, np.

$$\texttt{0011} = \texttt{0} \cdot \texttt{011} = \texttt{001} \cdot \texttt{1}.$$

Istnieje jednak pewna jednoznaczna reprezentacja każdego słowa w postaci konkatenacji słów Lyndona:

Twierdzenie 2.1.3 (Lyndon)**.** *Każde słowo $w \in \Sigma^*$ może zostać w sposób jednoznaczny przedstawione w postaci konkatenacji słów Lyndona:*

$$w = l_1 l_2 \dots l_k,$$

takich że $l_1 \geqslant l_2 \geqslant \dots \geqslant l_k$.

Co więcej, l_k jest wówczas najmniejszym leksykograficznie niepustym sufiksem słowa w, natomiast l_1 — najdłuższym prefiksem w będącym słowem Lyndona.

Dowody opisanych własności (2.1.1, 2.1.2 oraz 2.1.3) można znaleźć na przykład w książce [16], poza dowodem części „co więcej" Twierdzenia 2.1.3, który można znaleźć w książce [15].

2.2. Prefiksy Lyndona

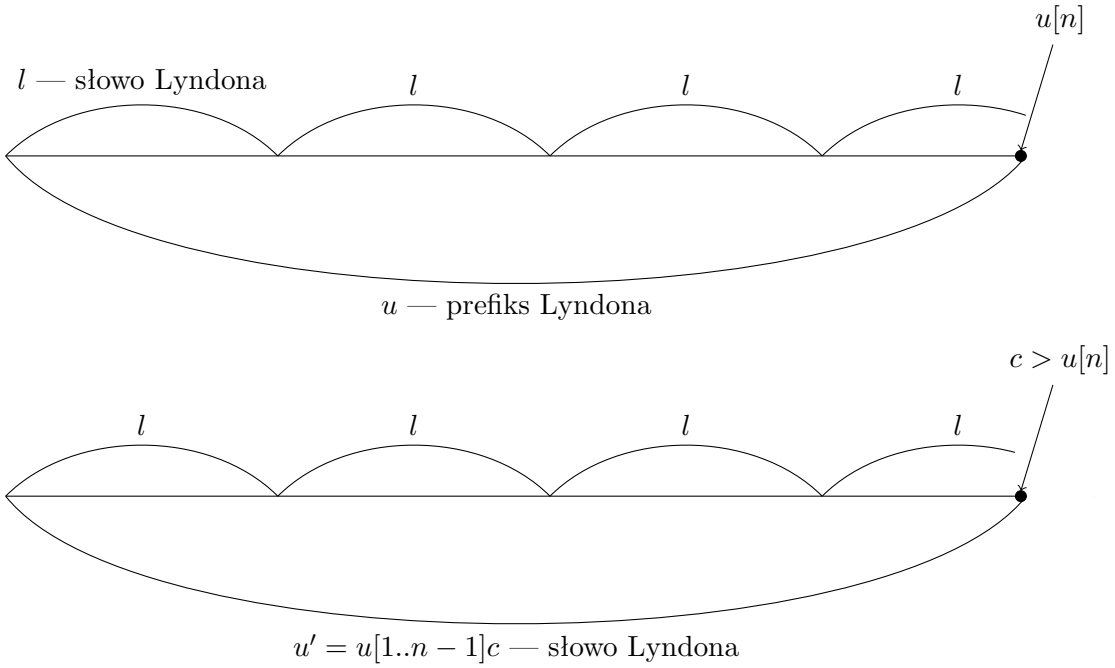
Definicja 2.2.1. *Słowo nazywamy **prefiksem Lyndona** (ang. **preprime**), jeżeli jest niepustym prefiksem pewnego słowa Lyndona.*

Będziemy dążyć do stworzenia alternatywnej definicji prefiksów Lyndona. Do pokazania równoważności definicji przyda nam się następujący lemat.

Lemat 2.2.1. *Niech l będzie słowem Lyndona, natomiast u — jego rozszerzeniem okresowym i $n = |u| \geqslant |l| = m$. Jeżeli powiększyć ostatnią literę słowa u, czyli zamienić u na*

$$u' = u[1..n-1]c, \text{ gdzie } u[n] < c \in \Sigma,$$

to stanie się ono słowem Lyndona.



Rysunek 2.1: Ilustracja tezy Lematu 2.2.1.

Dowód. Niech

$$n = a \cdot m + b, \text{ gdzie } 0 \leqslant b < m.$$

Pokażemy, że dowolny właściwy sufiks $u'[i..n]$ słowa u' jest większy niż u', co jest równoważne tezie Lematu.

Załóżmy najpierw, że $i \not\equiv 1 \pmod{m}$. Jeżeli $i \leqslant a \cdot m$, to

$$u'[i..\lceil i/m \rceil \cdot m] = u[i..\lceil i/m \rceil \cdot m] = l[q..m],$$

gdzie

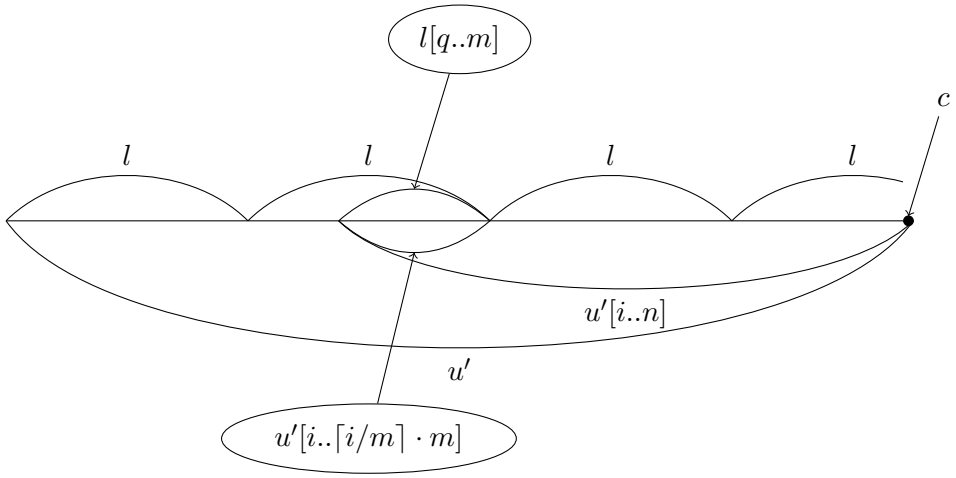
$$q = ((i-1) \bmod m) + 1 \tag{2.1}$$

jest równe $i \bmod m$, jeżeli $m \nmid i$, a m w przeciwnym przypadku (patrz Rys. 2.2). Stąd

$$u'[i..n] \geqslant u'[i..\lceil i/m \rceil \cdot m] = l[q..m] > l. \tag{2.2}$$

Pierwsza w powyższych nierówności zachodzi, ponieważ $u'[i..\lceil i/m\rceil \cdot m]$ jest prefiksem $u'[i..n]$, druga natomiast, gdyż l jest słowem Lyndona.

Mamy zatem, że $u'[i..n] > l$. Ponadto, l nie jest prefiksem $u'[i..n]$, gdyż prefiks $u'[i..\lceil i/m\rceil \cdot m]$ słowa $u'[i..n]$ jest krótszy niż l i leksykograficznie większy niż l (patrz (2.2)).

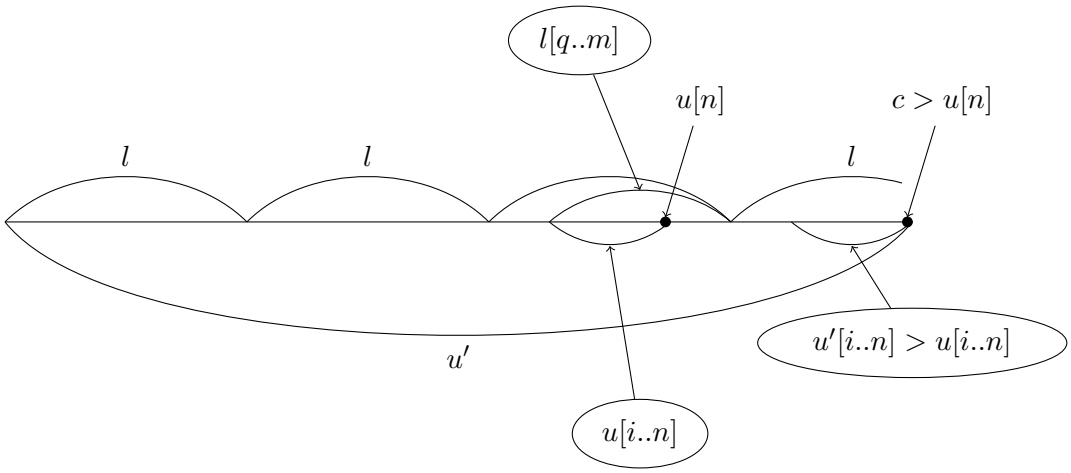


Rysunek 2.2: Ilustracja podprzypadku, w którym $i \not\equiv 1 \pmod m$ oraz $i \leqslant a \cdot m$.

Jeżeli zaś $i > a \cdot m$, to $u[i..n]$ jest prefiksem $l[q..m]$ (gdzie q jest zdefiniowane jak w (2.1)) — patrz Rys. 2.3. Ponieważ $u'[i..n] > u[i..n]$ i $|u'[i..n]| = |u[i..n]|$, to na mocy oczywistych własności porządku leksykograficznego (Fakt 1.2.1) zachodzi:

$$u'[i..n] > l[q..m].$$

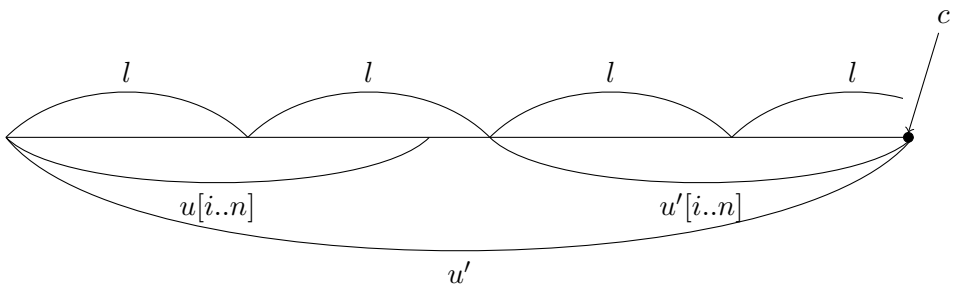
Podobnie jak w poprzednim przypadku, mamy więc $u'[i..n] > l$ i l nie jest prefiksem $u'[i..n]$, ponieważ $|u'[i..n]| < |l|$.



Rysunek 2.3: Ilustracja podprzypadku, w którym $i \not\equiv 1 \pmod m$ oraz $a \cdot m < i \leqslant n$.

W każdym z omówionych przypadków zachodzi zatem $u'[i..n] > u'$, gdyż pokazaliśmy, że $u'[i..n] > l$ oraz że l jest prefiksem słowa u', ale nie słowa $u'[i..n]$.

Jeżeli wreszcie $i \equiv 1 \pmod m$, to $u[i..n]$ jest prefiksem u. Ponieważ $u'[i..n] > u[i..n]$ oraz $|u'[i..n]| < |u|$, to na mocy oczywistych własności porządku leksykograficznego (Fakt 1.2.1) zachodzi $u'[i..n] > u'$ (patrz Rys. 2.4).



Rysunek 2.4: Ilustracja przypadku, w którym $i \equiv 1 \pmod{m}$.

■

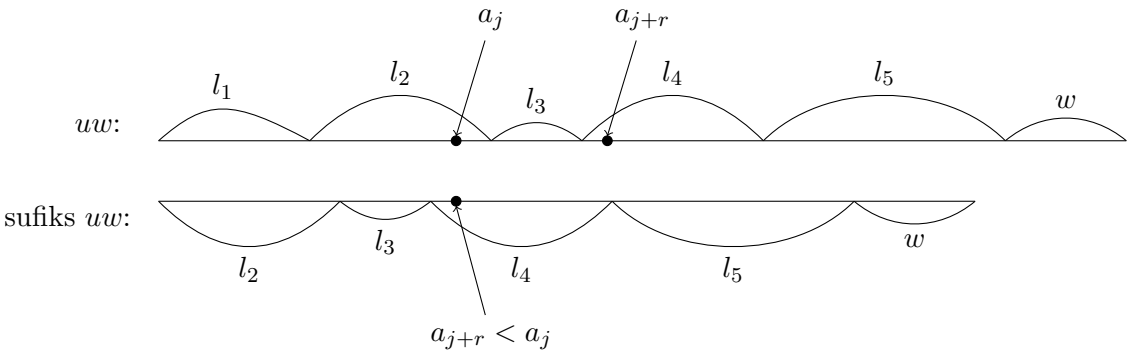
Stwierdzenie 2.2.2 (Równoważna definicja prefiksów Lyndona)**.** *Słowo u jest prefiksem Lyndona wtedy i tylko wtedy, gdy jest ono rozszerzeniem okresowym pewnego słowa Lyndona l o długości $|l| \leqslant |u|$. Co więcej, dla danego u słowo l jest wyznaczone jednoznacznie.*

Dowód. Niech $l_1 \dots l_k$ będzie rozkładem u na niemalejący ciąg słów Lyndona z Twierdzenia 2.1.3. Pokażemy, że u jest rozszerzeniem cyklicznym l_1, czyli że $|l_1| = r$ jest okresem słowa $l_1 \dots l_k$.

Niech w będzie dowolnym „uzupełnieniem" prefiksu Lyndona u, tzn. takim słowem, że uw jest Lyndona. Niech dalej $u = a_1 \dots a_n$, gdzie $a_i \in \Sigma$. Pokażemy nie wprost, że $a_i = a_{i+r}$ dla $i = 1, \dots, n-r$. Faktycznie, załóżmy, że $j \in \{1, \dots, n-r\}$ byłoby pierwszą pozycją, dla której $a_j \neq a_{j+r}$. Nie może zachodzić $a_j > a_{j+r}$, gdyż wówczas mielibyśmy

$$uw = l_1 l_2 \dots l_k w > l_2 \dots l_k w, \tag{2.3}$$

co stanowiłoby sprzeczność z tym, że uw jest słowem Lyndona.



Ilustracja wzoru (2.3).

Aby mogło zachodzić $a_j \neq a_{j+r}$, musiałoby więc być $a_j < a_{j+r}$. Ponieważ

$$a_1 \dots a_{j+r-1} a_j$$

jest rozszerzeniem okresowym słowa Lyndona l_1, to na mocy Lematu 2.2.1 słowo

$$a_1 \dots a_{j+r-1} a_{j+r}$$

byłoby słowem Lyndona. To jednak stanowi sprzeczność z Twierdzeniem 2.1.3, gdyż l_1 jest najdłuższym prefiksem u będącym słowem Lyndona. Ostatecznie mamy zatem, że szukana pozycja j nie istnieje, co pokazuje, że u jest rozszerzeniem okresowym l_1.

Na koniec pokażemy nie wprost, że słowo l z tezy Lematu jest wyznaczone jednoznacznie. Załóżmy więc, że u byłoby rozszerzeniem okresowym dwóch różnych słów Lyndona l i l', przy $|l'| > |l|$. W takim wypadku $l' = l^k m$, gdzie $k \geqslant 1$ i m jest prefiksem l. $m \neq \epsilon$, gdyż w przeciwnym wypadku l' nie byłoby pierwotne. Z definicji słów Lyndona mamy $m > l'$. Z drugiej strony, m jest prefiksem l, które to słowo jest z kolei prefiksem l', skąd $m \leqslant l'$. To daje sprzeczność postaci $m < m$. ■

Zauważmy, że wobec udowodnionego właśnie Stwierdzenia 2.2.2, teza Lematu 2.2.1 może zostać przeformułowana następująco:

Stwierdzenie 2.2.3. *Jeżeli powiększyć ostatnią literę prefiksu Lyndona, to otrzymuje się słowo Lyndona.*

2.3. Generowanie słów Lyndona

Stwierdzenie 2.2.2 wykazuje jednoznaczną odpowiedniość między słowami Lyndona o długości nie większej niż n i prefiksami Lyndona o długości n. Na jej podstawie można skonstruować algorytm generowania wszystkich słów Lyndona o długości nie większej niż n w kolejności leksykograficznej. Poniższy zapis tego algorytmu, autorstwa H. Fredricksena, I. J. Kesslera i J. Maiorany (algorytm pochodzi z prac [10] oraz [11]), jest wzorowany na książce [15].

```
1:  { Algorytm L: Generowanie słów Lyndona o długości ⩽n w porządku ⩽. }
2:  for i := 1 to n do a_i := 0;
3:  a_0 := −1;
4:  j := 1;
5:  while true do begin
6:    { Niezmiennik: a_1...a_n jest prefiksem Lyndona }
7:    { i rozszerzeniem okresowym słowa Lyndona a_1...a_j. }
8:    Wypisz(a_1...a_j);
9:    j := n;
10:   while a_j = Z do j := j − 1;
11:   if j = 0 then break;
12:   a_j := a_j + 1; { słowo a_1...a_j jest Lyndona }
13:   for k := j + 1 to n do a_k := a_{k−j}; { rozszerzenie okresowe }
14: end
```

Przeanalizujemy Algorytm L krok po kroku, co pozwoli nam uzasadnić jego poprawność. W wierszach 2.–4. następuje inicjacja wszystkich istotnych zmiennych: $a_1 \ldots a_n$ odpowiada po niej rozszerzeniu okresowemu najmniejszego leksykograficznie słowa Lyndona $a_1 \ldots a_j = 0$ — a zatem niezmiennik pętli **while** jest spełniony przy pierwszym obrocie. W wierszach 9.–12. słowo $a_1 \ldots a_n$ zostaje powiększone w najmniejszy możliwy sposób (przypomnijmy, że $\Sigma = \{0, 1, \ldots, Z\}$). Ponieważ dowolny prefiks prefiksu Lyndona jest oczywiście prefiksem Lyndona, to na mocy Stwierdzenia 2.2.3, po wykonaniu instrukcji z wiersza 12. słowo $a_1 \ldots a_j$ jest Lyndona; jest to zatem najmniejsze słowo Lyndona z Σ^* większe od aktualnego rozszerzenia okresowego. Znalezione w wierszu 13. rozszerzenie okresowe tego słowa jest zatem najmniejszym prefiksem Lyndona leksykograficznie większym od poprzedniego, co pokazuje, że wynikiem algorytmu jest ciąg wszystkich (Stwierdzenie 2.2.2) słów Lyndona o długości nie większej niż n w kolejności leksykograficznej.

Na koniec udowodnimy pewne własności Algorytmu L, które będą nam potrzebne w dalszych rozważaniach.

Lemat 2.3.1. *Jeżeli na początku pewnego kroku głównej pętli* ***while*** *(wiersz 8.) zachodzi* $j < n$ *i* $a_1 \neq Z$*, to* $s = a_1 a_2 \dots a_j$ *jest prefiksem wszystkich słów Lyndona skonstruowanych w algorytmie aż do momentu, gdy* $j = n$ *(wraz z tym momentem).*

Dowód. Jeżeli tylko w wierszu 8. algorytmu zachodzi $j < n$ i $a_1 = Z$, to w rozszerzeniu okresowym, które znajduje się wówczas w tablicy a, mamy $a_{j+1} = a_1 < Z$. To oznacza, że inkrementacja z wiersza 12. będzie miała miejsce dla $j' \geqslant j + 1$ — zmienna j wzrośnie, a prefiks s pozostanie bez zmian. Proces ten będzie trwał aż do momentu, gdy $j = n$. ■

Lemat 2.3.2. *Jeżeli w Algorytmie L zostało wygenerowane słowo Lyndona o długości* $k < n$ *oraz to słowo nie jest równe* 0*, to poprzednio wygenerowane słowo Lyndona miało długość* n *oraz posiadało sufiks* Z^{n-k}*.*

Dowód. Oczywisty. ■

Rozdział 3

Ciągi de Bruijna

3.1. Definicja

Definicja 3.1.1. ***Ciągiem de Bruijna*** *rzędu n nad alfabetem $\Sigma = \{0, 1, \ldots, M-1\}$ nazywamy słowo u o długości M^n, które spełnia następujący warunek: zbiór*

$$\{u[i..i+n-1] : 1 \leqslant i \leqslant M^n\}$$

zawiera wszystkie słowa z Σ^n. Słowo u traktujemy przy tym jako cykliczne, tzn. $u[M^n+1] = u[1]$, $u[M^n+2] = u[2]$ itd.

Oczywistą konsekwencją Definicji 3.1.1 jest to, że każde spośród słów z Σ^n występuje jako podsłowo słowa $uu[1..n-1]$ *dokładnie raz*. Ta właściwość stanowi o użyteczności ciągów de Bruijna do generowania wszystkich n-elementowych krotek, co ma szczególne zastosowanie w przypadku $M = 2$, kiedy ciąg de Bruijna rzędu n pozwala na szybkie przejrzenie reprezentacji binarnych wszystkich liczb n-bitowych.

Przykład 2. Poniższa tabela przedstawia przykłady ciągów de Bruijna dla różnych wartości parametrów n i M.

n	M	ciąg de Bruijna rzędu n nad $\Sigma = \{0, 1, \ldots, M-1\}$
2	1	`01`
2	2	`1001`
3	2	`01000111`
4	2	`0000100110101111`
5	2	`10010000011010100111011000101111`
6	2	`000101010010111011111000111111010110000001100111001001101101000001`
2	3	`110120022`
3	3	`210202200011002111201012122`
2	4	`1232130220011033`
3	4	`311031010220013213011133230232212223131203303000320121002021123 3`

Przyjrzyjmy się na przykład ciągowi `01000111` ($n = 3$, $M = 2$). Początkami wystąpień poszczególnych słów z $\{0,1\}^3$ w tym ciągu są:

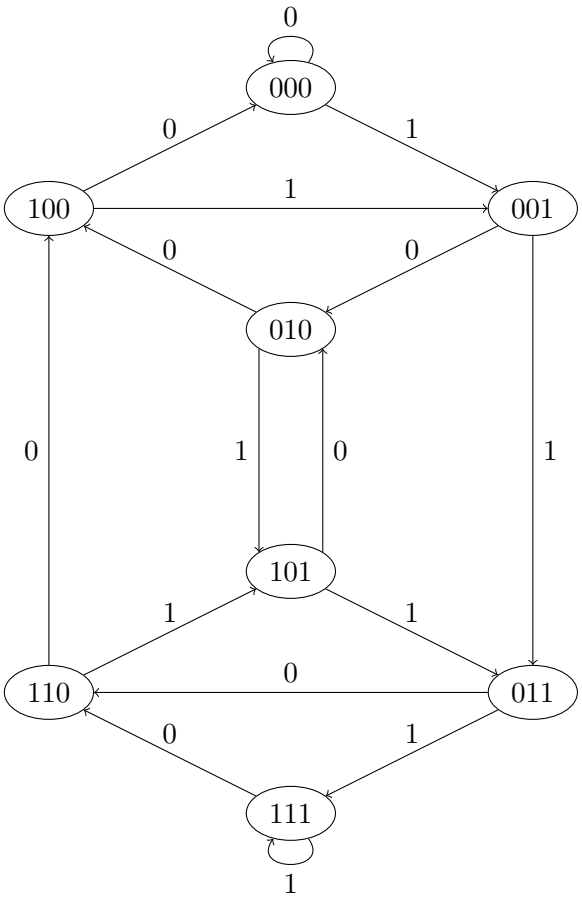
słowo	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
pozycja	3	4	1	5	2	8	7	6

3.2. Ciąg de Bruijna jako cykl Eulera

Klasyczna metoda generowania ciągów de Bruijna ([8], [12]) opiera się na poszukiwaniu cykli Eulera w pewnym szczególnym skierowanym grafie etykietowanym $G_B = (V, E)$, nazywanym często *grafem de Bruijna*. Wierzchołki G_B odpowiadają wszystkim słowom nad Σ długości $n-1$. Krawędzie są etykietowane literami z Σ. Z wierzchołka odpowiadającego słowu $a_1a_2 \dots a_{n-1}$ wychodzi po jednej krawędzi odpowiadającej każdej literze z Σ; krawędź z etykietą $a \in \Sigma$ prowadzi do wierzchołka $a_2 \dots a_{n-1}a$. Formalnie:

$$\begin{aligned} V &= \Sigma^{n-1} \\ E &= \{a_1a_2 \dots a_{n-1} \xrightarrow{a} a_2 \dots a_{n-1}a \mid a_1, \dots, a_{n-1}, a \in \Sigma\} \end{aligned}$$

Stopień wyjściowy każdego wierzchołka to M.



Przykładowy graf G_B dla $n = 4$, $M = 2$, $\Sigma = \{0, 1\}$.

Zauważmy, że stopień wejściowy każdego wierzchołka jest równy M, gdyż jedynymi krawędziami wchodzącymi do wierzchołka odpowiadającego $a_1 \dots a_{n-2}a_{n-1}$ są:

$$aa_1 \dots a_{n-2} \xrightarrow{a_{n-1}} a_1 \dots a_{n-2}a_{n-1} \text{ dla } a \in \Sigma,$$

wszystkie etykietowane a_{n-1}.

Lemat 3.2.1. *Graf G_B jest silnie spójny.*

Dowód. Rozważmy dowolne dwa wierzchołki G_B, $u = a_1 \dots a_{n-1}$ oraz $v = b_1 \dots b_{n-1}$. Ścieżkę z u do v konstruujemy, przechodząc kolejno po krawędziach o etykietach b_1, b_2, $\dots$, b_{n-1}. ■

Na podstawie Lematu 3.2.1 i wcześniejszych rozważań, z których wynika, że

$$\forall_{v \in V}\ indeg(v) = outdeg(v) = M,$$

możemy stwierdzić, że graf G_B jest eulerowski.

Stwierdzenie 3.2.2. *Istnieje bijektywna odpowiedniość między ciągami etykiet krawędzi na cyklach Eulera w G_B i ciągami de Bruijna.*

Dowód. Udowodnimy najpierw, że ciąg etykiet krawędzi na dowolnym cyklu Eulera w G_B jest ciągiem de Bruijna. Na wstępie zauważmy, że $|E| = M^n$. Niech $c = e_1 e_2 \dots e_{|E|}$ będzie słowem złożonym z etykiet kolejnych krawędzi na dowolnym cyklu Eulera w G_B. Pokażemy, że żadne dwa podsłowa długości n słowa $d = c \cdot c[1..n-1]$ rozpoczynające się od różnych pozycji nie są takie same.

Dowód przeprowadzimy przez sprowadzenie do sprzeczności. Załóżmy, że podsłowa d długości n rozpoczynające się od pozycji $i \leqslant n$ oraz $i < j \leqslant n$ byłyby takie same, tzn.:

$$d[i..i+n-1] = d[j..j+n-1].$$

Wówczas wierzchołki v_i oraz v_j grafu G_B, do których wchodzimy na rozważanym cyklu krawędziami odpowiadającymi $d[i+n-2]$ oraz $d[j+n-2]$, byłyby takie same. Ponieważ $d[i+n-1] = d[j+n-1] = e$, to zarówno $d[i+n-1]$, jak i $d[j+n-1]$ odpowiadałoby krawędzi o etykiecie e, wychodzącej z wierzchołka $v_i = v_j$. W G_B istnieje jednakże dokładnie jedna taka krawędź. To stanowi więc sprzeczność z założeniem, że c odpowiadało cyklowi Eulera, gdyż wówczas krawędzie odpowiadające etykietom

$$\{d[i+n-1] : 1 \leqslant i \leqslant |E|\}$$

muszą być wszystkie różne.

Pokażemy teraz, że dowolny ciąg de Bruijna c odpowiada ciągowi etykiet krawędzi na pewnym cyklu Eulera w G_B. Cykl ten zaczyna się mianowicie w wierzchołku

$$v_0 = c[M^n - (n-2)..M^n]$$

i przechodzi kolejno krawędziami

$$c_1, c_2, \dots, c_{M^n}.$$

Dlaczego opisana konstrukcja jest poprawna? Na wstępie zauważmy, że z każdego wierzchołka G_B wychodzą krawędzie o wszystkich możliwych etykietach z Σ, czyli c odpowiada pewnej ścieżce w G_B o początku w v_0. Ostatnie $n-1$ liter c, czyli słowo $c[M^n-(n-2)..M^n]$, jednoznacznie wyznacza wierzchołek v_0 jako końcowy tej ścieżki, co pokazuje, że c jest cyklem. Wreszcie dowolne podsłowo długości n słowa $c \cdot c[1..n-1]$ definiuje jedną krawędź G_B, a zatem, ponieważ c jest ciągiem de Bruijna, to każda krawędź grafu pojawia się na cyklu dokładnie raz. ■

Prostą, ale ważną konsekwencją Stwierdzenia 3.2.2 jest to, że dla dowolnych n oraz M istnieje ciąg de Bruijna rzędu n nad alfabetem $\Sigma = \{0, 1, \dots, M-1\}$. Ponadto, Stwierdzenie 3.2.2 daje prosty algorytm konstrukcji ciągów de Bruijna za pomocą klasycznych metod wyznaczania cyklu Eulera w grafie skierowanym.

3.3. Inne algorytmy generowania ciągów de Bruijna

Powołując się na książkę [15], wymienimy także inne metody generowania ciągów de Bruijna, nie konstruujące w tym celu grafu G_B. Złożoność pamięciowa każdego z tych algorytmów jest dużo mniejsza niż powyższego. Metody te mają jednakże jedną istotną wadę — pozwalają na konstrukcję jedynie pewnych specyficznych ciągów de Bruijna, a nie wszystkich, jak to ma miejsce w algorytmie wyznaczającym cykl Eulera w G_B. Ponieważ algorytmy te nie są kluczowe dla głównej części pracy, nie będziemy się wgłębiać w szczegóły techniczne tych metod ani w dowody ich poprawności.

Zauważmy, że każdemu ciągowi de Bruijna można jednoznacznie przypisać funkcję, która mając dane wierzchołek grafu G_B oraz krawędź, którą do tego wierzchołka weszliśmy, zwraca kolejną krawędź na cyklu Eulera wyznaczającym ten ciąg. Innymi słowy, argumentem dowolnej takiej funkcji f jest słowo z Σ^n, natomiast wartością — pewna litera z Σ. W ogólności opisane funkcje dla większości ciągów de Bruijna mogą być bardzo skomplikowane, jednakże dla pewnych szczególnych ciągów ich postać bywa prosta i łatwa do wyznaczenia. Na przykład, jeżeli M jest liczbą pierwszą, to dla dowolnego wielomianu

$$x^n - c_n x^{n-1} - \ldots - c_2 x - c_1$$

nieprzywiedlnego w $\mathbb{Z}_M$, następująca funkcja prawie rekurencyjna:

$$f(a_1 \ldots a_n) = \begin{cases} c_1 & \text{jeżeli } a_1 \ldots a_n = 0^n \\ 0 & \text{jeżeli } a_1 \ldots a_n = 1 \cdot 0^{n-1} \\ (\sum_{i=1}^{n} c_i \cdot a_i) \bmod M & \text{w przeciwnym przypadku} \end{cases}$$

generuje pewien ciąg de Bruijna. Wobec dużej liczby monicznych wielomianów nieprzywiedlnych nad $\mathbb{Z}_M$, zazwyczaj udaje się znaleźć taki wielomian, w którym liczba niezerowych współczynników jest mała. Istnieją także podobne metody, służące do generowania ciągów de Bruijna w przypadku, gdy M nie jest pierwsze.

Inne algorytmy opierają się na konstrukcjach rekurencyjnych. Jeżeli mamy daną funkcję f, generującą ciąg de Bruijna dla danych parametrów n oraz M, to istnieją krótkie nierekurencyjne algorytmy generujące na podstawie f pewne ciągi de Bruijna dla parametrów

$$n' = n + 1 \text{ i } M' = M$$

oraz

$$n'' = 2n \text{ i } M'' = M.$$

Po dokładne zapisy tych algorytmów i szkice dowodów poprawności odsyłamy do książki [15].

3.4. Zliczanie ciągów de Bruijna

Metoda zliczania wszystkich ciągów de Bruijna jest związana z algorytmem wyznaczania ich jako cykli Eulera w grafie G_B. Najpopularniejszy, klasyczny algorytm wyznaczania cyklu Eulera w grafie skierowanym G opiera się na metodzie przyrostowej: w grafie znajdowany jest jakiś cykl C, następnie w każdej silnie spójnej składowej $G \setminus C$ zostaje wyznaczony rekurencyjnie jej cykl Eulera, po czym te cykle zostają doklejone do C. Nasza metoda zliczania wymagać będzie jednakże innego algorytmu wyznaczania cyklu Eulera. Duża część poniższych rozumowań jest wzorowana na szkicach dowodów z książki [14].

Niech dane będą: skierowany graf eulerowski $G = (V, E)$ oraz pewien jego ustalony wierzchołek v_0.

Definicja 3.4.1. ***Skierowanym drzewem rozpinającym*** *(ang.* ***spanning arborescence****)* ***grafu*** G ***ukorzenionym w wierzchołku*** v_0 *nazywamy taki podgraf* G *o* $|V|-1$ *krawędziach, w którym stopień wyjściowy każdego wierzchołka poza* v_0 *jest równy* 1*:*

$$\forall_{v \in V \setminus \{v_0\}} \; outdeg(v) = 1$$

natomiast stopień wyjściowy v_0 *jest równy* 0*:*

$$outdeg(v_0) = 0$$

oraz w którym wierzchołek v_0 *jest osiągalny z każdego z pozostałych wierzchołków.*

Będziemy budować w G ścieżkę S o początku w v_0, stosując w każdym napotkanym wierzchołku v następujące kryterium.

Kryterium 3.4.1. *Niech* T *będzie skierowanym drzewem rozpinającym* G *ukorzenionym w* v_0*.*

Jako kolejną krawędź ścieżki S *w wierzchołku* v *wybieramy jakąkolwiek krawędź* $(v, w) \in E$ *spoza* T*, której nie ma jeszcze na ścieżce* S*, jeżeli tylko taka istnieje. Jeżeli nie, to wybieramy (jedyną możliwą z danego wierzchołka) krawędź* $(v, w) \in E_T$*, jeżeli* $(v, w) \notin S$*. Jeżeli nawet taka krawędź nie istnieje, to kończymy budowanie* S*.*

Lemat 3.4.1. *Ścieżka* S *skonstruowana zgodnie z Kryterium 3.4.1 jest cyklem Eulera w grafie* G*.*

Dowód. Ponieważ G jest eulerowski, to dla każdego wierzchołka $v \in V$ zachodzi $indeg(v) = outdeg(v)$. To oznacza, że S, jako ścieżka rozpoczynająca się w v_0, musi się także kończyć w v_0, gdyż w każdym innym wierzchołku v, po wejściu do v, musi wciąż istnieć krawędź wychodząca z v.

Wiemy już, że S jest cyklem. Aby pokazać, że jest także cyklem Eulera, uzasadnimy, dlaczego wszystkie krawędzie T leżą na S — jest to wystarczające wobec faktu, że w Kryterium 3.4.1 każda z krawędzi z E_T jest wybierana jako ostatnia możliwa. Aby S mogła się skończyć w v_0, po przejściu S wszystkie krawędzie wychodzące z v_0 muszą być już wykorzystane; ponieważ $indeg(v_0) = outdeg(v_0)$, to to oznacza, że wówczas muszą być także wykorzystane wszystkie krawędzie $\{(w, v_0) \in T : w \in V\}$. Krawędź postaci $(w, v_0) \in T$ zostaje wykorzystana na ścieżce S jako ostatnia wychodząca z w, a zatem wówczas wszystkie krawędzie wchodzące do w, w tym te należące do T, muszą już leżeć na S. Kontynuując to rozumowanie aż do dojścia do liści T (czyli wierzchołków o stopniu wejściowym równym 0), otrzymujemy, że warunkiem koniecznym na to, aby konstrukcja S mogła się zakończyć w v_0, jest to, aby wszystkie krawędzie z E_T leżały już na S. Podkreślmy, że możliwość dojścia do liści jest zagwarantowana przez to, że w drzewie skierowanym korzeń jest osiągalny z każdego z wierzchołków. ■

Dodajmy jeszcze, że aby znaleźć w grafie G jakiekolwiek skierowane drzewo rozpinające, wystarczy wykonać przeszukiwanie w głąb (DFS) w transponowanej wersji grafu G, rozpoczynając je z wierzchołka v_0. Ostatecznie otrzymaliśmy alternatywny algorytm znajdowania cyklu Eulera w grafie skierowanym.

Odtąd dążyć będziemy do sprowadzenia zliczania cykli Eulera w grafie skierowanym G do zliczania skierowanych ukorzenionych drzew rozpinających w G. Cykl Eulera rozumiemy przy tym jako ciąg numerów kolejnych wierzchołków na cyklu. *Ile różnych cykli Eulera można otrzymać z jednego ukorzenionego drzewa rozpinającego* T *w* G*?* Dla każdego wierzchołka $v \in V$ możemy ustalić permutację krawędzi ze zbioru $\{(v, w) : w \in V\} \cap E \setminus E_T$ ustalającą

pewną kolejność wyboru krawędzi wychodzących z v, a zatem likwidującą wszelką dowolność z Kryterium 3.4.1. Odpowiedzią na postawione pytanie jest więc

$$\prod_{v \in V \setminus \{v_0\}} (indeg(v) - 1)! \cdot indeg(v_0)! = M \cdot \prod_{v \in V} (indeg(v) - 1)! \tag{3.1}$$

Niech $E(v_0)$ oznacza liczbę cykli Eulera w G zaczynających się w v_0, natomiast $T(v_0)$ — liczbę drzew rozpinających ukorzenionych w v_0. Na podstawie sformułowania Kryterium 3.4.1 nietrudno zauważyć, że cykle Eulera wyznaczone dla różnych drzew rozpinających są różne. To stwierdzenie w połączeniu z (3.1) daje

$$E(v_0) \geqslant T(v_0) \cdot M \cdot \prod_{v \in V} (indeg(v) - 1)! \tag{3.2}$$

Zajmijmy się teraz dowodem nierówności w drugą stronę. Z każdym cyklem Eulera C zaczynającym się w v_0 można związać graf H, w którym każdemu wierzchołkowi $v \in V \setminus \{v_0\}$ przyporządkowujemy ostatnią krawędź, którą z niego wychodzimy na C.

Pokażemy, że H jest wówczas skierowanym drzewem rozpinającym G ukorzenionym w v_0, zgodnie z Definicją 3.4.1. Warunki tej Definicji związane ze stopniami wierzchołków są oczywiście spełnione. H nie może także zawierać cyklu. Faktycznie, gdyby w H istniał cykl

$$v_1 \to v_2 \to \ldots \to v_k \to v_1,$$

to dla każdego $i \in \{1, \ldots, k\}$ krawędź $v_{i-1} \to v_i$ musiałaby się pojawić na C przed $v_i \to v_{i+1}$ (ostatnie wyjście z v_i na C musi się odbyć po ostatnim wejściu tamże) — w tym stwierdzeniu indeksy wierzchołków traktujemy modulo k. Z przechodniości relacji „wyprzedzania się na cyklu C" wynikłoby stąd na przykład, że $v_1 \to v_2$ wyprzedza $v_2 \to v_3$, ale także i $v_2 \to v_3$ wyprzedza $v_1 \to v_2$, co jest oczywistą sprzecznością.

Skoro H nie zawiera żadnego cyklu, to z każdego wierzchołka $v \neq v_0$, idąc po krawędziach z H, w końcu dojdziemy do v_0 (jedyny wierzchołek bez krawędzi wyjściowych). To oznacza, że H jest faktycznie skierowanym drzewem rozpinającym G. Innymi słowy, pokazaliśmy, że każdy cykl Eulera G może zostać skonstruowany za pomocą naszego algorytmu, czyli że

$$E(v_0) \leqslant T(v_0) \cdot M \cdot \prod_{v \in V} (indeg(v) - 1)! \tag{3.3}$$

Na mocy (3.2) oraz (3.3) mamy, że:

$$E(v_0) = T(v_0) \cdot M \cdot \prod_{v \in V} (indeg(v) - 1)!$$

Łączna liczba cykli Eulera w G wyraża się zatem następująco za pomocą liczb $T(v)$:

$$\#\text{cykli Eulera} = M \cdot \sum_{v_0 \in V} T(v_0) \cdot \prod_{v \in V} (indeg(v) - 1)! \tag{3.4}$$

Musimy jeszcze pokazać, jak wyznaczać liczby $T(v)$. Dla grafu G skonstruujemy w tym celu *macierz stopni wejściowych* $D = (d_{ij})_{i,j \in V}$ następująco:

$$d_{ij} = \begin{cases} indeg(i) & \text{jeżeli } i = j \\ -1 & \text{jeżeli } (i,j) \in E \\ 0 & \text{w przeciwnym przypadku.} \end{cases} \tag{3.5}$$

Okazuje się, że prawdziwe jest wówczas następujące twierdzenie, po dowód którego odsyłamy do książki [14].

Twierdzenie 3.4.2 (Tutte, 1948)**.** *Niech D będzie macierzą stopni wejściowych grafu skierowanego G. Wówczas liczba skierowanych drzew rozpinających G ukorzenionych w wierzchołku v jest równa wyznacznikowi podmacierzy D, powstałej poprzez usunięcie z D v-tego wiersza oraz v-tej kolumny.*

Innymi słowy, $T(v)$ jest równe v-temu minorowi macierzy D, czyli $\det D_v$.

Wyposażeni we wzór (3.4) i w Twierdzenie 3.4.2, możemy wyznaczyć liczbę ciągów de Bruijna rzędu n nad Σ, $|\Sigma| = M$. Interesować nas będzie oczywiście graf G_B, dla którego równość (3.4) przedstawia się następująco:

$$\#\text{ciągów de Bruijna} = M \cdot \sum_{v_0 \in V} T(v_0) \cdot (M-1)!^{M^{n-1}} \tag{3.6}$$

Pozostałą częścią wyprowadzenia jest wyznaczenie wzoru na v-ty minor macierzy D dla grafu G_B, zgodnie z wynikiem z Twierdzenia 3.4.2. Ponieważ jest to znany rezultat, a sam dowód jest dosyć żmudny, to przedstawiamy w tym miejscu jedynie sam wynik, a po dowód odsyłamy do pracy [8] (dla przypadku $M = 2$) lub [20].

$$\det D_v = M^{M^{n-1}-n}$$

Zauważmy, że wartość $\det D_v$ nie zależy od v.

Ostatecznie liczba ciągów de Bruijna w zależności od parametrów n oraz M wyraża się następująco:

$$\begin{aligned} \#\text{ciągów de Bruijna} &= M \cdot M^{n-1} \cdot M^{M^{n-1}-n} \cdot (M-1)!^{M^{n-1}} \\ &= M!^{M^{n-1}} \end{aligned}$$

Rozdział 4

Minimalne leksykograficznie ciągi de Bruijna

4.1. Związek ze słowami Lyndona

Spośród wszystkich ciągów de Bruijna szczególnym zainteresowaniem obdarzymy minimalne leksykograficznie w swoich klasach, czyli dla danych wartości parametrów n oraz M. Poniższa tabela zawiera przykłady takich ciągów.

n	M	minimalny leksykograficznie ciąg de Bruijna rzędu n nad $\Sigma = \{0, 1, \ldots, M-1\}$
2	2	0011
3	2	00010111
4	2	0000100110101111
5	2	00000100011001010011101011011111
6	2	0000001000011000101000111001001011001101001111010101110110111111
2	3	001021122
3	3	000100201101202102211121222
2	4	0010203112132233

Najprostszym sposobem generowania minimalnych leksykograficznie ciągów de Bruijna jest znajdowanie najmniejszego leksykograficznie cyklu Eulera w grafie G_B, opisanym w poprzednim rozdziale. Okazuje się jednak, że:

Twierdzenie 4.1.1 (Fredricksen, Maiorana)**.** *Konkatenacja wszystkich słów Lyndona o długościach dzielących n w kolejności leksykograficznej $L = l_1 l_2 \ldots l_t$ daje minimalny leksykograficznie ciąg de Bruijna rzędu n.*

Aby zademonstrować prawdziwość nieco zaskakującego Twierdzenia 4.1.1, przedstawiamy jeszcze raz tabelę z minimalnymi leksykograficznie ciągami de Bruijna, tym razem podzielonymi na odpowiednie słowa Lyndona:

n	M	rozkład ciągu de Bruijna na słowa Lyndona
2	2	0 01 1
3	2	0 001 011 1
4	2	0 0001 0011 01 0111 1
5	2	0 00001 00011 00101 00111 01011 01111 1
6	2	0 000001 000011 000101 000111 001 001011 001101 001111 01 010111...
2	3	0 01 02 1 12 2
3	3	0 001 002 011 012 021 022 1 112 122 2
2	4	0 01 02 03 1 12 13 2 23 3

Pozostała część niniejszego rozdziału zawiera przede wszystkim dowód Twierdzenia 4.1.1, podzielony na dwie części: dlaczego wynikiem sklejenia odpowiednich słów Lyndona jest ciąg de Bruijna (sekcja 4.2) i dlaczego uzyskany ciąg jest najmniejszy leksykograficznie w swojej klasie (sekcja 4.3). Na koniec (sekcje 4.4 i 4.5) zastanowimy się nad efektywnym zastosowaniem metody zawartej w tym Twierdzeniu do wygenerowania szukanego ciągu de Bruijna.

4.2. Dowód poprawności

Na początek zastanówmy się, dlaczego w ogóle długość ciągu $L = l_1 \dots l_t$ to M^n. Zauważmy, że istnieje bijektywna odpowiedniość między *słowami o długości n* a *słowami pierwotnymi o długościach dzielących n*. Faktycznie, dowolnemu słowu długości n można jednoznacznie przyporządkować jego pierwiastek pierwotny, a dowolne słowo pierwotne długości $d \mid n$ posiada dokładnie jedno rozszerzenie okresowe długości n. Z kolei *liczba słów pierwotnych o długościach dzielących n* jest równa *sumie długości wszystkich słów Lyndona o długościach dzielących n*, czyli $|l_1 l_2 \dots l_t|$. Jest tak dlatego, że dowolne słowo Lyndona l odpowiada, na mocy Faktu 1.2.3, dokładnie $|l|$ słowom pierwotnym — swoim równoważnikom cyklicznym. A zatem rzeczywiście $|l_1 \dots l_t| = M^n$.

Definicja 4.2.1. ***Pierwiastkiem Lyndona*** *słowa u (oznaczenie: $pL(u)$) nazywamy pierwiastek pierwotny minimalnego leksykograficznie równoważnika cyklicznego u.*

Fakt 4.2.1 (Natychmiastowy wniosek z Definicji 4.2.1)**.** *Na mocy Faktu 1.2.4 łatwo wnioskujemy, że pierwiastek Lyndona słowa u jest zawsze minimalnym leksykograficznie równoważnikiem cyklicznym pierwiastka pierwotnego słowa u. Co więcej, pierwiastek Lyndona każdego słowa jest zawsze słowem Lyndona.*

Wykorzystując wprowadzoną definicję, możemy przeformułować nasze rozumowanie ilościowe następująco: każdemu słowu z Σ^n odpowiada jednoznacznie jego pierwiastek Lyndona; z kolei każde słowo Lyndona o długości $d \mid n$ jest pierwiastkiem Lyndona dokładnie d słów długości n.

Fakt 4.2.2. *Równość $|L| = M^n$ pozwala przy okazji wyznaczyć wzór na liczbę słów Lyndona nad $\Sigma = \{0, \dots, M-1\}$ o długości n, czyli $L_M(n)$. Powyżej udowodniliśmy, że*

$$\sum_{d|n} dL_M(d) = M^n.$$

Korzystając ze wzoru na odwracanie z użyciem funkcji Möbiusa (patrz np. [13]), mamy zatem

$$L_M(n) = \sum_{d|n} \mu(d) M^{n/d}.$$

W dalszej części dowodu poprawności, dla każdego słowa $u \in \Sigma^n$ wskażemy pozycję w słowie L, na której występuje jako podsłowo. Rozważania podzielimy na kilka przypadków (przedstawiony podział został zaczerpnięty z książki [15], lecz rozumowania są inne niż tam zasugerowano).

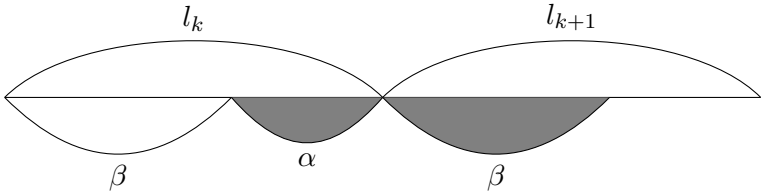
4.2.1. Przypadek A

Załóżmy, że słowo u jest pierwotne, czyli że jest rotacją cykliczną pewnego słowa Lyndona l_k. Załóżmy ponadto, że jeżeli $u = \alpha\beta$ i $l_k = \beta\alpha$, to α nie składa się z samych wystąpień litery Z, czyli $\alpha \neq Z^{|\alpha|}$.

Na mocy wymienionych warunków, w pewnym momencie działania Algorytmu L skonstruowane zostanie słowo l_k, czyli $a_1 \dots a_n = \beta\alpha$ oraz $j = n$. W kolejnym kroku pętli **while** z wiersza 5. zostanie wygenerowane słowo Lyndona posiadające prefiks β — jest tak dlatego, że $\alpha \neq Z^{|\alpha|}$. Stąd i na mocy Lematu 2.3.1, prefiksem słowa l_{k+1} jest β.

Uwaga 1. $l_k \neq Z$, więc słowo l_{k+1} istnieje.

A zatem $u = \alpha\beta$ jest podsłowem $l_k l_{k+1}$:



Przypadek A.

Uwaga 2. Zauważmy, że warunki Przypadku A zachodzą w szczególności, jeżeli $\beta = \epsilon$, czyli $u = \alpha$ jest słowem Lyndona l_k.

4.2.2. Przypadek B

W tym przypadku także zakładamy, że u jest pierwotne. Tym razem przyjmujemy jednak, że jeśli $u = \alpha\beta$ i $\beta\alpha$ jest słowem Lyndona, to $\alpha = Z^{|\alpha|}$. Pomijamy tutaj sytuację, w której $\alpha = \epsilon$, gdyż wtedy $u = \beta$ byłoby słowem Lyndona, a słowa Lyndona są pokryte przez warunki Przypadku A (patrz Uwaga 2). Z drugiej strony, $\beta \neq \epsilon$, gdyż wówczas $u = \alpha$ nie byłoby pierwotne.

Uwaga 3. Warunek $\alpha \neq \epsilon$, a zwłaszcza $\beta \neq \epsilon$, będzie nam potrzebny w sekcji 4.3.

Szczególnym podprzypadkiem niniejszego przypadku jest sytuacja, w której $\beta = 0^{|\beta|}$. Zauważmy jednak, że $l_1 = 0$, $l_2 = 0^{n-1}1$, $l_t = Z$, a $l_{t-1} = (Z-1)Z^{n-1}$. Stąd słowo $Z^n 0^n$ jest podsłowem $l_{t-1} l_t l_1 l_2$, więc wszystkie słowa odpowiadające wspomnianemu podprzypadkowi na pewno w L występują.

Odtąd zakładamy zatem, że $\alpha, \beta \neq \epsilon$ i $\beta \neq 0^{|\beta|}$. Niech l_k będzie najmniejszym słowem Lyndona o długości dzielącej n, spełniającym warunek $\beta \leqslant l_k$.

Uwaga 4. $\beta\alpha$ jest słowem Lyndona większym niż β, więc l_k istnieje.

Ponieważ $\beta \leqslant l_k \leqslant \beta\alpha$, to na mocy oczywistych własności porządku leksykograficznego (Fakt 1.2.1) β jest prefiksem l_k. Dalej, β jest prefiksem Lyndona, a zatem, na mocy Stwierdzenia 2.2.2, jest rozszerzeniem okresowym pewnego słowa Lyndona $\beta' \leqslant \beta$.

Przyjrzyjmy się teraz słowu l_{k-1}, które spełnia nierówność $l_{k-1} < \beta$.

Uwaga 5. $l_k \neq 0$, gdyż $\beta \neq 0^{|\beta|}$, a zatem słowo l_{k-1} istnieje.

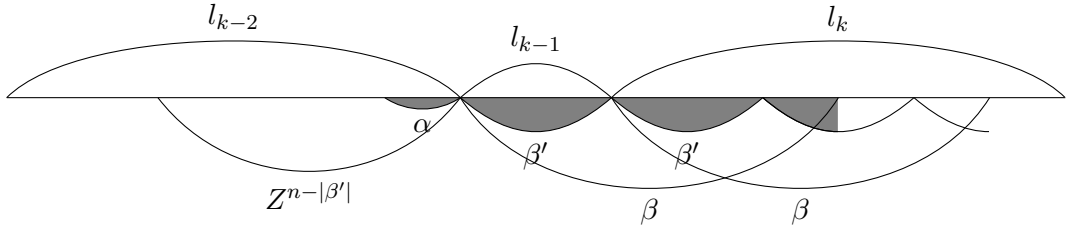
Jeżeli $l_{k-1} = \beta'$, to na mocy Lematu 2.3.2 poprzednim słowem Lyndona wygenerowanym w Algorytmie L musiało być l_{k-2} długości n, zakończone

$$n - |\beta'| \geqslant n - |\beta| = |\alpha|$$

wystąpieniami litery Z.

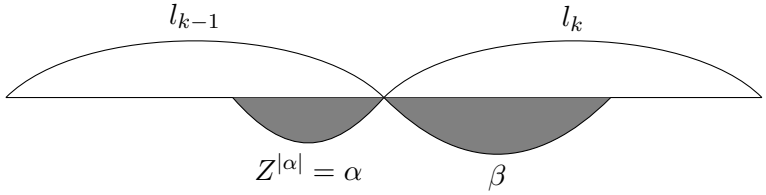
Uwaga 6. $\beta \neq 0^{|\beta|}$, a zatem $\beta' \neq 0$, co gwarantuje w tym przypadku istnienie słowa l_{k-2}.

Ponieważ β to rozszerzenie okresowe β', to $u = \alpha\beta$ jest wówczas podsłowem $l_{k-2}l_{k-1}l_k$.



Przypadek B_1: $l_{k-1} = \beta'$, u jest podsłowem $l_{k-2}l_{k-1}l_k$.

Jeżeli $l_{k-1} \neq \beta'$, to pokażemy, że $\alpha = Z^{|\alpha|}$ jest sufiksem l_{k-1}; stąd będzie już wynikać, że $u = \alpha\beta$ jest podsłowem $l_{k-1}l_k$.



Przypadek B_2: $l_{k-1} \neq \beta'$, u jest podsłowem $l_{k-1}l_k$.

Załóżmy najpierw, że $l_{k-1} < \beta'$. Słowo Lyndona wygenerowane w Algorytmie L tuż przed β' na mocy Lematu 2.3.2 ma długość n i sufiks $Z^{n-|\beta'|}$. Słowem tym musi więc być l_{k-1}, gdyż w przeciwnym przypadku istniałoby słowo Lyndona l długości n spełniające

$$l_{k-1} < l < \beta' \leqslant \beta \leqslant l_k,$$

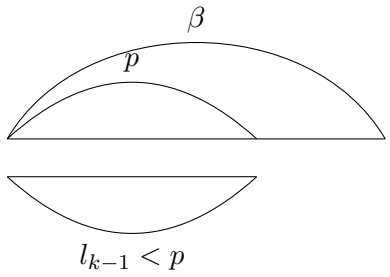
co jest niemożliwe wobec definicji ciągu l_i. Stąd $\alpha = Z^{|\alpha|}$ jest sufiksem l_{k-1}, gdyż

$$|\alpha| = n - |\beta| \leqslant n - |\beta'|,$$

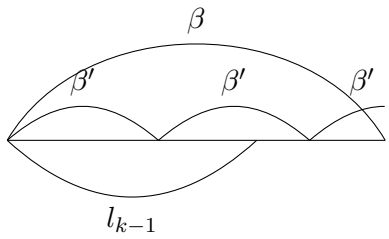
co kończy dowód w tym przypadku.

Odtąd zakładamy, że $\beta' \leqslant l_{k-1} < \beta$. Rozważmy kilka przypadków dotyczących postaci słowa l_{k-1}.

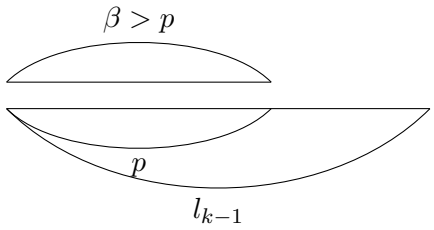
- Załóżmy, że $|l_{k-1}| \leqslant |\beta|$. Na mocy Lematu 2.3.1 istnieje słowo Lyndona l długości n, posiadające prefiks l_{k-1}. Gdyby prefiks p długości $|l_{k-1}|$ słowa β był leksykograficznie większy niż l_{k-1}, to słowo l spełniałoby nierówności $l_{k-1} < l < \beta \leqslant l_k$, co stanowiłoby sprzeczność z definicją ciągu l_i.



- Jeżeli więc $|l_{k-1}| \leqslant |\beta|$, to na mocy poprzedniego podpunktu l_{k-1} musiałoby być prefiksem β (przypomnijmy, że $l_{k-1} < \beta$). Z nierówności $\beta' \leqslant l_{k-1} < \beta$ i oczywistych własności porządku leksykograficznego (Fakt 1.2.1) wynikałoby wówczas, że β' byłoby prefiksem l_{k-1}. Wówczas jednak l_{k-1} byłoby rozszerzeniem okresowym dwóch różnych słów Lyndona — β' i l_{k-1} — co stanowiłoby sprzeczność ze Stwierdzeniem 2.2.2.



- Musi więc zachodzić $|l_{k-1}| > |\beta|$. Prefiks p długości $|\beta|$ słowa l_{k-1} musi być wówczas mniejszy leksykograficznie niż β, gdyż w przeciwnym przypadku zachodziłoby $l_{k-1} > \beta$.



  Okazuje się, że wówczas $|l_{k-1}| = n$ — w przeciwnym przypadku, ponownie na mocy Lematu 2.3.1, istniałoby słowo Lyndona l, posiadające prefiks l_{k-1} i spełniające $l_{k-1} < l < \beta \leqslant l_k$.

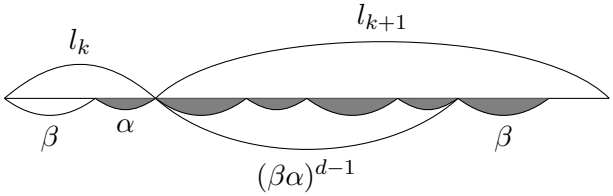
Wiemy zatem, że $|l_{k-1}| = n$. Teraz możemy już pokazać, że słowo $Z^{|\alpha|}$ jest sufiksem l_{k-1}. Gdyby tak nie było, to pewna z liter $l_{k-1}[|\beta| + 1], \ldots, l_{k-1}[n]$ byłaby różna od Z. Stąd, kolejnym słowem Lyndona wygenerowanym w Algorytmie L po l_{k-1} byłoby słowo l', zgodne na prefiksie długości $|\beta|$ z l_{k-1}. To, ponownie na mocy Lematu 2.3.1, oznaczałoby istnienie słowa Lyndona długości n, większego leksykograficznie od l_{k-1} i mniejszego od l_k, co jest niemożliwe.

4.2.3. Przypadek C

Rozważyliśmy już sytuację, w której u jest słowem Lyndona (Przypadki A i B). Odtąd zakładamy, że u jest potęgą swojego pierwiastka pierwotnego o wykładniku $d > 1$. W niniejszym przypadku $u = (\alpha\beta)^d$, gdzie $l_k = \beta\alpha$ jest słowem Lyndona, a $\alpha \neq Z^{|\alpha|}$ — jest to więc w pewnym sensie analogia Przypadku A.

W pewnym momencie działania Algorytmu L skonstruowane zostaje słowo Lyndona $l_k = \beta\alpha$; tablica a zostaje wówczas wypełniona rozszerzeniem okresowym tego słowa. W kolejnym kroku pętli **while** z wiersza 5. zostanie wygenerowane słowo Lyndona, posiadające prefiks $(\beta\alpha)^{d-1}\beta$ — jest tak dlatego, że $\alpha \neq Z^{|\alpha|}$. Stąd i na mocy Lematu 2.3.1, prefiksem słowa l_{k+1} jest właśnie $(\beta\alpha)^{d-1}\beta$.

Ostatecznie $u = (\alpha\beta)^d$ jest podsłowem $l_k l_{k+1}$:



Przypadek C.

Uwaga 7. Zauważmy, że warunki Przypadku C zachodzą w szczególności, jeżeli $\beta = \epsilon$, czyli α jest słowem Lyndona l_k. Jedynym wyjątkiem od tej uwagi jest $u = Z^n$, które to słowo jest uwzględnione w kolejnym przypadku.

4.2.4. Przypadek D

Podobnie jak w Przypadku C, i tym razem zakładamy, że u nie jest pierwotne: $u = (\alpha\beta)^d$, $d > 1$, a $l_k = \beta\alpha$ jest słowem Lyndona. Tym razem jednak $\alpha = Z^{|\alpha|}$. Przyjmujemy przy tym, że $\alpha \neq \epsilon$, gdyż wówczas $\alpha\beta$ jest Lyndona, a takie słowa zostały już rozpatrzone w Przypadku C (patrz Uwaga 7). Przypadek D stanowi analogię Przypadku B.

Podobnie jak w Przypadku B, i tutaj mamy pewne wyjątkowe słowo, które rozpatrzymy osobno. Jest nim Z^n, które to słowo jest oczywiście sufiksem $l_{t-1}l_t$. Rozpatrzenie tego słowa pozwala nam odtąd założyć, że $\beta \neq Z^{|\beta|}$, co w szczególności oznacza, że $\beta \neq \epsilon$.

W pewnym momencie działania Algorytmu L skonstruowane zostaje słowo Lyndona $l_k = \beta\alpha$; tablica a zostaje wówczas wypełniona rozszerzeniem okresowym tego słowa. Przyjrzyjmy się temu, jak wyglądają słowa l_{k-1} i l_{k+1}.

Uwaga 8. $\beta\alpha \neq 0$, gdyż $\alpha = Z^{|\alpha|} \neq \epsilon$; stąd l_{k-1} istnieje.

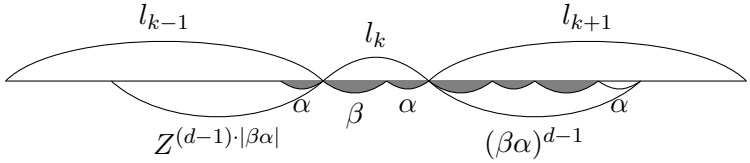
Uwaga 9. $\beta\alpha \neq Z$, gdyż $\beta \neq Z^{|\beta|}$; stąd l_{k+1} istnieje.

Ponieważ $|l_k| < n$, to słowo Lyndona, które zostało wygenerowane w Algorytmie L tuż przed l_k, musiało mieć długość n i być zakończone co najmniej $n - |l_k|$ wystąpieniami litery Z. To oznacza, że wspomniane słowo to właśnie l_{k-1}. Ponieważ

$$n - |l_k| = (d-1) \cdot |\alpha\beta| \geqslant |\alpha|,$$

to α jest sufiksem l_{k-1}.

Ponieważ $\beta\alpha \neq Z^{|\beta\alpha|}$, to słowo Lyndona wygenerowane w Algorytmie L tuż po l_k będzie miało prefiks $(\beta\alpha)^{d-1}$. Stąd i na mocy Lematu 2.3.1, także l_{k+1} ma prefiks $(\beta\alpha)^{d-1}$, czyli u jest podsłowem $l_{k-1}l_kl_{k+1}$.



Przypadek D.

4.3. Dowód minimalności

Definicja 4.3.1. *Jeżeli $u = a_1 \dots a_{n-1}a_n$ oraz $v = a_1 \dots a_{n-1}(a_n + 1)$, to dla uproszczenia będziemy pisać:*

$$v = u + 1 \quad \textit{lub równoważnie} \quad u = v - 1.$$

Aby udowodnić, że ciąg de Bruijna L jest minimalny leksykograficznie w klasie ciągów de Bruijna rzędu n nad $\Sigma = \{0, \ldots, M-1\}$, wystarczy pokazać, że:

Stwierdzenie 4.3.1. *Dla dowolnych dwóch słów u oraz $v = u + 1$ długości n nad alfabetem Σ, u występuje przed v jako podsłowo L.*

Zakładamy przy tym, że u nie jest postaci $Z^i 0^{n-i}$ $(i < n)$, gdyż o takich słowach można zakładać, że występują na samym początku L — w przeciwieństwie do $v = u+1$ — więc dla nich opisany warunek zachodzi.

Dowód. Dlaczego w celu pokazania leksykograficznej minimalności L wystarczy udowodnić Stwierdzenie 4.3.1? Załóżmy nie wprost, że istniałby ciąg de Bruijna L' rzędu n, spełniający $L' < L$. Wówczas prefiksy długości n słów L i L' musiałyby być oczywiście takie same i równe 0^n. Oznaczmy przez i, $i > n$, najmniejszy indeks litery, dla której $L'[i] < L[i]$. Wówczas

$$L'[i-n+1..i-1] = L[i-n+1..i-1]$$

oraz

$$L'[i-n+1..i] < L[i-n+1..i].$$

Jednakże na mocy Stwierdzenia 4.3.1, wszystkie ze słów

$$L[i-n+1..i-1] \cdot 0,\ L[i-n+1..i-1] \cdot 1,\ \ldots,\ L[i-n+1..i-1] \cdot (L[i]-1)$$

są podsłowami $L[1..i-1] = L'[1..i-1]$, a zatem także i $L'[i-n+1..i]$ byłoby podsłowem $L'[1..i-1]$, co stanowi sprzeczność z tym, że L' jest ciągiem de Bruijna. ■

Odtąd skupimy się na dowodzie Stwierdzenia 4.3.1. Przy okazji dowodu faktu, że $l_1 \ldots l_t$ jest ciągiem de Bruijna, dokonaliśmy klasyfikacji wszystkich podsłów z Σ^n tego ciągu na cztery kategorie: A, B (B_1 i B_2), C oraz D. W dalszej części dowodu pokażemy, że niezależnie od tego, do których z powyższych kategorii przynależą słowa u oraz $v = u + 1$, u występuje przed v w L. Na pierwszy rzut oka daje to $4^2 = 16$ (albo nawet $5^2 = 25$) przypadków do rozpatrzenia, niemniej jednak bardzo szybko zredukujemy tę liczbę.

Poniższa tabelka prezentuje wszystkie przypadki, jakie mamy do rozpatrzenia; przypadek XY oznacza, że zakładamy, że u jest kategorii X, natomiast v — kategorii Y.

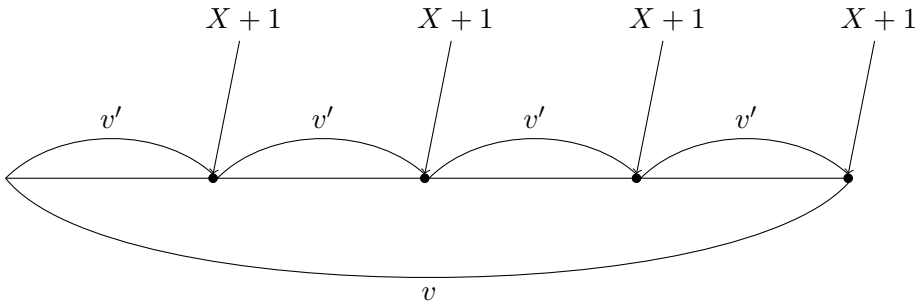
AA	AB	AC	AD
BA	BB	BC	BD
CA	CB	CC	CD
DA	DB	DC	DD

Rozumowanie rozpoczniemy od następującego, bardzo pomocnego lematu. Ideę przedstawionego dowodu lematu zasugerował mi Piotr Niedźwiedź.

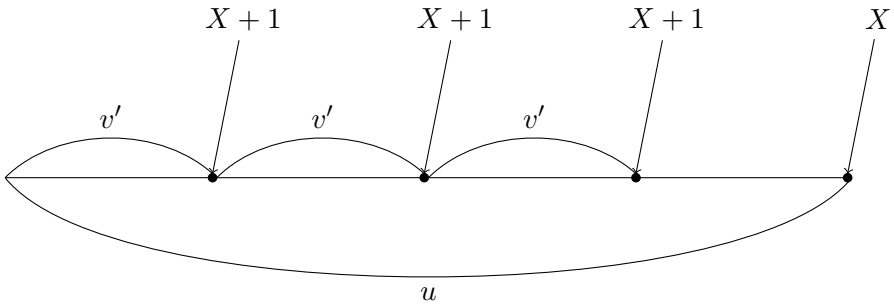
Lemat 4.3.2. *Jeżeli $v = u + 1$, to $pL(u) < pL(v)$.*

Dowód. Oznaczmy $a = pL(u)$, $b = pL(v)$.

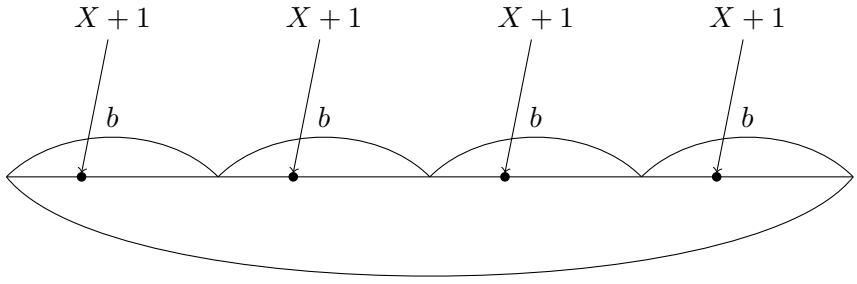
Niech v' będzie pierwiastkiem pierwotnym v.



Przypomnijmy, że $u = v - 1$.

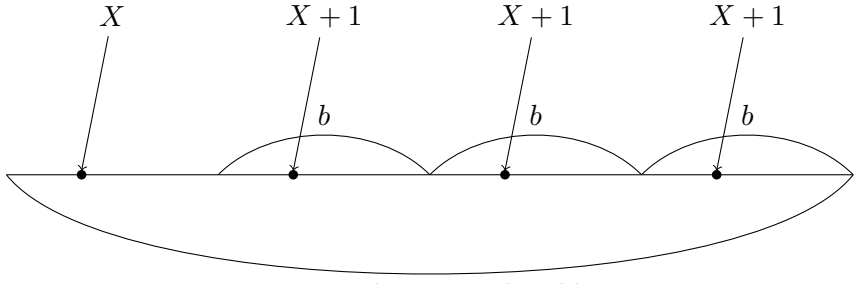


Pierwiastek Lyndona b słowa v jest rotacją cykliczną v', natomiast $b^{|v|/|b|}$ to rotacja cykliczna v:



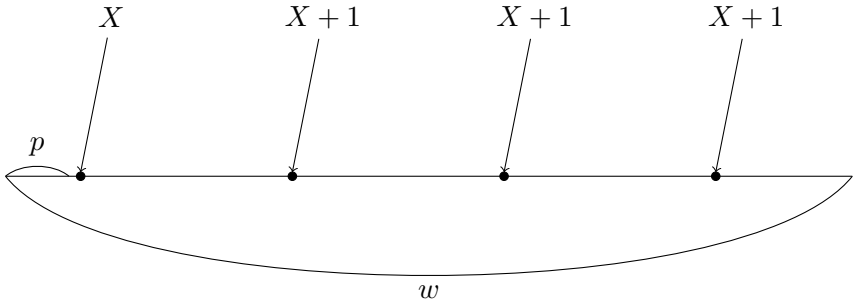
$b^{|v|/|b|}$ — pewien równoważnik cykliczny v

Niech i będzie pozycją w b, odpowiadającą końcowej literze $X + 1$ słów v oraz v'. Dokonajmy obrotu cyklicznego u o i liter; otrzymamy wówczas słowo w, mniejsze leksykograficznie niż $b^{|v|/|b|}$:

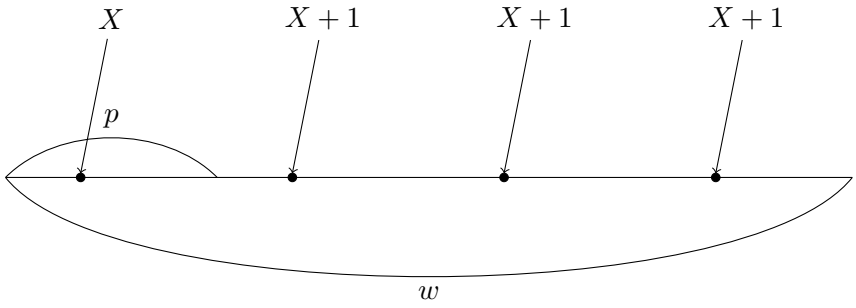


w — pewien równoważnik cykliczny u

Okazuje się, że dowolny prefiks p słowa w jest mniejszy leksykograficznie od b. Faktycznie, albo $|p| < i$ i p jest prefiksem b:



albo $|p| \geqslant i$ i $p[1..i] < b[1..i] \leqslant b$:



Stąd pierwiastek pierwotny w' słowa w, będący w szczególności prefiksem w, jest mniejszy leksykograficznie niż b. Ponieważ $a = pL(u) \leqslant w'$, to ostatecznie mamy $a \leqslant w' < b$, czyli $a < b$. ■

W przypadku słowa w z kategorii:

- A — wystąpienie słowa rozpoczyna się w L w słowie $l_k = pL(w)$;
- C — wystąpienie słowa rozpoczyna się w L w słowie $l_k = pL(w)$;
- D — wystąpienie słowa rozpoczyna się w L w słowie l_{k-1}, przy czym $l_k = pL(w)$.

Na mocy Lematu 4.3.2, w przypadkach AA, CC, DD, AC, CA, DA oraz DC teza jest prawdziwa, ponieważ wystąpienie u rozpoczyna się od słowa Lyndona położonego wcześniej niż odpowiadające słowo dla v. Tabelka nierozpatrzonych przypadków wygląda po tym spostrzeżeniu następująco:

~~AA~~	AB	~~AC~~	AD
BA	BB	BC	BD
~~CA~~	CB	~~CC~~	CD
~~DA~~	DB	~~DC~~	~~DD~~

Uwaga 10. Można pokazać, że żaden z przypadków: CC, DA, DC, DD nie może zajść. Ponieważ nie upraszcza to powyższego wywodu, to dowód tego faktu pomijamy.

Przejdźmy teraz do przypadków AD oraz CD. Oznaczmy $pL(u) = l_k$ i $pL(v) = l_m$. W każdym z Przypadków A i C zachodzi $u = (\alpha\beta)^d$, gdzie $l_k = \beta\alpha$, $d \geqslant 1$ oraz $\alpha \neq Z^{|\alpha|}$. Wystąpienie u w ramach L rozpoczyna się od słowa l_k. Z kolei $v = (\alpha'\beta')^{d'}$, gdzie $l_m = \beta'\alpha'$, $\alpha' = Z^{|\alpha'|}$, a wystąpienie v w ramach L rozpoczyna się od l_{m-1}.

Na mocy Lematu 4.3.2 mamy $l_k < l_m$; teza dla tego przypadku mogłaby więc nie zachodzić tylko wtedy, gdyby $l_k = l_{m-1}$. Jak łatwo zauważyć z analizy poszczególnych kategorii słów,

wystąpienie u w ramach L rozpoczyna się wówczas od sufiksu α słowa $l_k = l_{m-1}$, natomiast wystąpienie v — od sufiksu α' słowa $l_{m-1} = l_k$. α' nie mogłoby być dłuższe niż α, gdyż wówczas α byłoby sufiksem $\alpha' = Z^{|\alpha'|}$ i składałoby się wyłącznie z liter Z, co nie zachodzi. W takim razie $|\alpha| > |\alpha'|$, co oznacza, że wystąpienie u w L rozpoczyna się wcześniej niż wystąpienie v. A zatem, nawet jeżeli $l_k = l_{m-1}$, to żądana własność zachodzi dla przypadków AD oraz CD.

Tabelka nierozpatrzonych przypadków:

~~AA~~	AB	~~AC~~	~~AD~~
BA	BB	BC	BD
~~CA~~	CB	~~CC~~	~~CD~~
~~DA~~	DB	~~DC~~	~~DD~~

Uwaga 11. Można pokazać, że przypadek CD nie może zajść. Ponieważ nie upraszcza to powyższego wywodu, to dowód tego faktu pomijamy.

Zajmijmy się teraz przypadkami BX, gdzie $X \in \{A, B, C, D\}$. Niech więc $u = \alpha\beta$, gdzie $\beta\alpha$ jest Lyndona i $\alpha = Z^{|\alpha|}$. Wówczas

$$v = u + 1 = \alpha\beta + 1 = \alpha(\beta + 1)$$

Uwaga 12. Fakt, że w powyższej równości, ale także i w wielu kolejnych, możemy sobie pozwolić na zapisy $\beta + 1$ lub $\beta - 1$ dla słów $\alpha\beta$ z kategorii B, wynika z Uwagi 3.

Przyjrzyjmy się słowu $w = (\beta + 1)\alpha$. Jeżeli $w = Z^n$, to również i $v = Z^n$; v jest wówczas sufiksem L, więc bez wątpienia występuje w ramach L później niż u. Jeżeli zaś $w \neq Z^n$, to pokażemy, że w jest słowem Lyndona, co będzie oznaczało, że jeżeli u przynależy do kategorii B, to również i v do tej kategorii przynależy, gdyż wówczas $w = pL(u)$.

β jest prefiksem Lyndona, gdyż $\beta\alpha$ jest słowem Lyndona. Na mocy Stwierdzenia 2.2.3 słowo $\beta + 1$ jest więc słowem Lyndona. Zakładamy, że $(\beta + 1)\alpha = w \neq Z^n$, więc $\beta + 1$ nie składa się z samych wystąpień litery Z. To oznacza, że

$$(\beta + 1)[1] < Z, \tag{4.1}$$

gdyż $\beta + 1$ jest Lyndona. A zatem

$$\beta + 1 < Z \tag{4.2}$$

Na mocy Twierdzenia 2.1.2 i nierówności (4.2) otrzymujemy, że

$$(\beta + 1)Z$$

jest słowem Lyndona. Z (4.1) dalej wnioskujemy, że

$$(\beta + 1)Z < Z, \tag{4.3}$$

co na mocy Twierdzenia 2.1.2 pokazuje, że

$$(\beta + 1)Z^2$$

jest słowem Lyndona itd. Wykonawszy to przejście $|\alpha|$ razy, otrzymamy, że

$$(\beta + 1)Z^{|\alpha|} = (\beta + 1)\alpha = w$$

jest słowem Lyndona, do czego dążyliśmy.

Wykluczyliśmy zatem możliwość zajścia przypadków BA, BC oraz BD (poza wyjątkowym przypadkiem $u = Z^{n-1}(Z - 1)$ i $v = Z^n$ kategorii BD, w którym teza Stwierdzenia 4.3.1 oczywiście zachodzi).

~~AA~~	AB	~~AC~~	~~AD~~
~~BA~~	BB	~~BC~~	~~BD~~
~~CA~~	CB	~~CC~~	~~CD~~
~~DA~~	DB	~~DC~~	~~DD~~

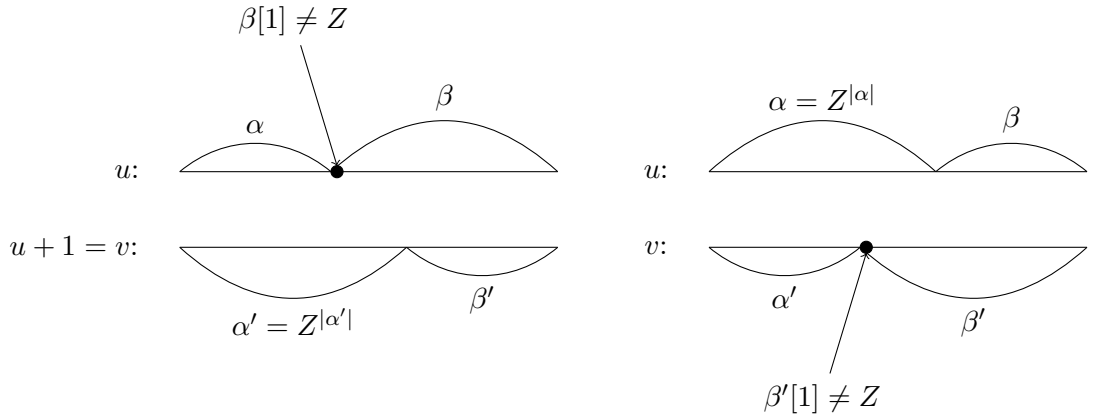
Spośród przypadków BX musimy jedynie rozpatrzyć BB. Niezależnie od tego, czy $u = \alpha\beta$ i $v = \alpha'\beta'$ spełniają warunki podprzypadku B_1, czy też B_2, wystąpienie u jako podsłowa L kończy się na słowie l_k,

$$\beta \leqslant l_k \leqslant \beta\alpha, \tag{4.4}$$

natomiast wystąpienie v — na słowie l_m,

$$\beta' \leqslant l_m \leqslant \beta'\alpha'. \tag{4.5}$$

Pokażemy, że $\alpha' = \alpha$ i $\beta' = \beta + 1$. Oczywiście $\beta[1] \neq Z$, gdyż w przeciwnym przypadku wszystkie litery β musiałyby być nie mniejsze niż Z (gdyż $\beta\alpha$ jest Lyndona), co oznaczałoby, że $u = \beta\alpha = Z^n$ i u nie byłoby pierwotne, wbrew założeniom Przypadku B. Podobnie, $\beta'[1] \neq Z$. Pokażemy teraz, że $\alpha = \alpha'$. Wiemy, że $\alpha = Z^{|\alpha|}$ i $\alpha' = Z^{|\alpha'|}$. Gdyby $|\alpha'| > |\alpha|$, toby $\beta[1] = Z$, co, jak pokazaliśmy, nie zachodzi; podobnie, gdyby $|\alpha'| < |\alpha|$, toby $\beta'[1] = Z$.



Skoro $\alpha' = \alpha$, to $\beta' = \beta + 1$. Na mocy nierówności (4.4) i (4.5) mamy zatem

$$l_k \leqslant \beta\alpha < (\beta + 1) = \beta' \leqslant l_m$$

Ostatecznie pokazaliśmy, że $l_k < l_m$, a zatem wystąpienie u w L kończy się przed końcem wystąpienia v w L.

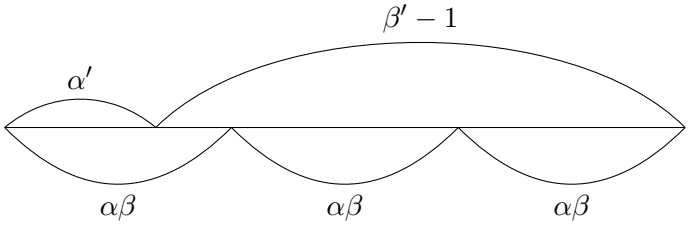
Pozostały nam już zatem tylko przypadki z rodziny XB, gdzie $X \in \{A, C, D\}$.

~~AA~~	AB	~~AC~~	~~AD~~
~~BA~~	~~BB~~	~~BC~~	~~BD~~
~~CA~~	CB	~~CC~~	~~CD~~
~~DA~~	DB	~~DC~~	~~DD~~

Niech $u = (\alpha\beta)^d$, gdzie $d \geqslant 1$ i $l_k = \beta\alpha$ jest słowem Lyndona. Niech $v = \alpha'\beta'$, $\alpha' = Z^{|\alpha'|}$, gdzie $\beta'\alpha'$ jest Lyndona. Wystąpienie u w ramach L kończy się na l_{k+1}, natomiast v — na słowie l_m takim, że $\beta' \leqslant l_m \leqslant \beta'\alpha'$.

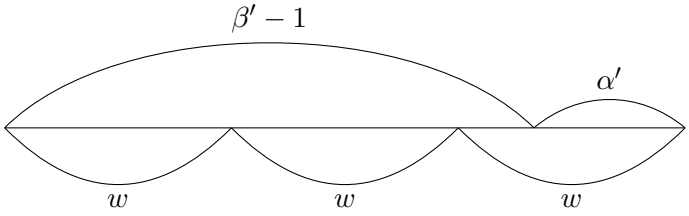
Na podstawie $v = u + 1$ oraz Uwagi 12 możemy napisać:

$$(\alpha\beta)^d = u = v - 1 = \alpha'\beta' - 1 = \alpha'(\beta' - 1).$$



Dwie reprezentacje słowa u.

Rozważmy słowo $s = (\beta' - 1)\alpha'$, będące pewną rotacją cykliczną u, i niech w będzie pierwiastkiem pierwotnym s.



Słowo $s = (\beta' - 1)\alpha'$ — rotacja cykliczna słowa u.

Na mocy Faktu 1.2.4, $|w| = |\beta\alpha|$.

Uwaga 13. Zauważmy, że $|w| > |\alpha'|$, gdyż w przeciwnym przypadku w byłoby sufiksem α' i wówczas zarówno s, jak i u byłoby postaci Z^n, co nie jest możliwe.

Co więcej, w jest oczywiście rotacją cykliczną $\beta\alpha$. Mamy zatem:

$$l_k = \beta\alpha \leqslant w \leqslant s = (\beta' - 1)\alpha' < \beta' \leqslant l_m,$$

gdzie pierwsza z nierówności wynika stąd, że $\beta\alpha$ jest Lyndona, druga — że w jest prefiksem s, a trzecia — z oczywistych własności porządku leksykograficznego (Fakt 1.2.1).

Mamy zatem, że $l_k < l_m$. Jeżeli też $l_{k+1} < l_m$, to tezy przypadków AB, CB i DB zachodzą. Przyjrzyjmy się więc sytuacji, gdy

$$l_{k+1} = l_m.$$

Wykluczymy najpierw, sprowadzając do sprzeczności, Przypadek B_1. Gdyby v spełniało warunki tego przypadku, to β' byłoby rozszerzeniem okresowym słowa $l_{m-1} = l_k$ (patrz ilustracja tego przypadku). Gdyby u było kategorii A, to taka sytuacja nie byłaby możliwa z prostego powodu — $|l_k| = n > |\beta'|$. W przeciwnym wypadku, ponieważ $|l_k| = |\beta\alpha| = |w|$ (Fakt 1.2.4) i w jest prefiksem właściwym $\beta' - 1$ (Uwaga 13), a więc także i β', to zachodziłoby $l_k = w$. To nie byłoby jednakże możliwe, gdyż $\beta' - 1$ jest rozszerzeniem okresowym w, więc β' nie może być także rozszerzeniem okresowym tego słowa. To daje żądaną sprzeczność.

Odtąd zakładamy, że v jest kategorii B_2. Wystąpienie v w ramach L rozpoczyna się od sufiksu $\alpha' = Z^{|\alpha'|}$ słowa l_{m-1}. Ponieważ $|l_{m-1}| = n$ (patrz opis Przypadku B_2), natomiast $|l_k| = |\beta\alpha|$, to to automatycznie eliminuje możliwość $l_k = l_{m-1}$ dla u kategorii C oraz D. Jeżeli zaś u spełnia warunki Przypadku A, to wystąpienie u w ramach L rozpoczyna się sufiksem α ($\neq Z^{|\alpha|}$) słowa $l_k = l_{m-1}$. Wówczas oczywiście $|\alpha| > |\alpha'|$, gdyż w przeciwnym przypadku α byłoby sufiksem α' i składałoby się wyłącznie z liter Z. To pokazuje, że nawet w przypadku, gdy $l_k = l_{m-1}$, u występuje przed v w ramach L.

To kończy dowód dla przypadków postaci XB, a zatem i cały dowód leksykograficznej minimalności L w klasie ciągów de Bruijna rzędu n.

~~AA~~	~~AB~~	~~AC~~	~~AD~~
~~BA~~	~~BB~~	~~BC~~	~~BD~~
~~CA~~	~~CB~~	~~CC~~	~~CD~~
~~DA~~	~~DB~~	~~DC~~	~~DD~~

Uwaga 14. Można pokazać, że przypadek CB nie może mieć miejsca, jednak nie upraszcza to w żaden sposób przeprowadzonego rozumowania.

Jako dodatek do dowodu prezentujemy wszystkie faktycznie możliwe przypadki XY dla $X, Y \in \{A, B, C, D\}$ wraz z przykładami konkretnych słów.

Przypadek	u	$v = u + 1$
AA	011100	011101
AB	110100	110101
AC	011010	011011
AD	101100	101101
BB	110010	110011
BD	111110	111111
CA	010010	010011
DB	100100	100101

4.4. Implementacja

W dwóch poprzednich sekcjach przeprowadzony został dowód Twierdzenia 4.1.1; w niniejszej sekcji zastanowimy się nad tym, jak wykorzystać ten teoretyczny wynik do konstrukcji algorytmu generującego ciąg L. Oczywistym sposobem jest wykorzystanie do tego celu Algorytmu L, w wersji, w której słowo $a_1 \dots a_j$ jest wypisywane (wiersz 8.) wtedy i tylko wtedy, gdy $j \mid n$.

W pracy [22] znajduje się szczegółowa analiza opisanego algorytmu, z której wynika, że jego złożoność czasowa może zostać oszacowana przez $O(|L|)$, co daje zamortyzowany koszt wygenerowania jednego symbolu ciągu L rzędu $O(1)$. W pracach [4] oraz [22] zaprezentowano także inne podobnie efektywne algorytmy. W niniejszej pracy stawiamy pytanie o pesymistyczną złożoność czasową wygenerowania jednego symbolu ciągu L. Przy badaniu tej złożoności kluczowe jest pojęcie maksymalnego *opóźnienia* opisanego algorytmu, które definiuje się jako *maksymalną liczbę słów Lyndona, jakie mogą zostać wygenerowane przez Algorytm L pomiędzy kolejnymi słowami Lyndona o długościach dzielących n.* Analizę maksymalnego opóźnienia Algorytmu L można znaleźć w pracy [10].

Fakt 4.4.1. *Maksymalne opóźnienie Algorytmu L to* $\lceil \frac{n}{2} \rceil - 1$.

Dowód. Dla uproszczenia przyjmijmy oznaczenie $m = \lceil \frac{n}{2} \rceil$. Na początek pokażemy ciąg długości $m-1$ kolejnych leksykograficznie słów Lyndona o długościach nie większych niż n i nie dzielących n, w przypadku, gdy $2 \mid n$. Zauważmy, że słowo

$$(Z-1)Z^{m-1}$$

jest Lyndona i jego długość dzieli n. Jego rozszerzeniem okresowym jest

$$(Z-1)Z^{m-1}(Z-1)Z^{m-1},$$

a zatem kolejnym słowem Lyndona wygenerowanym w Algorytmie L będzie

$$(Z-1)Z^{m}$$

o długości $(m+1) \nmid n$. Jego rozszerzeniem okresowym jest

$$(Z-1)Z^m(Z-1)Z^{m-2},$$

za pomocą którego wygenerowane zostaje kolejne słowo Lyndona:

$$(Z-1)Z^{m+1}$$

o długości $(m+2) \nmid n$. To rozumowanie można kontynuować aż do napotkania słowa

$$(Z-1)Z^{n-2},$$

które jest ostatnim słowem Lyndona spełniającym żądane warunki, po którym zostaje wygenerowane już słowo długości n:

$$(Z-1)Z^{n-1}.$$

Daje to łącznie

$$(n-2)-m+1 = (2 \cdot m-2)-m+1 = m-1$$

słów Lyndona między dwoma kolejnymi o długościach dzielących n.

Podobną konstrukcję można przeprowadzić dla $2 \nmid n$, otrzymując ciąg $m-1$ kolejno generowanych słów Lyndona:

$$(Z-1)Z^{m-1},\ (Z-1)Z^m,\ \ldots,\ (Z-1)Z^{n-2},$$

w którym długość żadnego słowa nie dzieli n.

Dlaczego opóźnienie Algorytmu L nie jest większe niż $m-1$? Uzasadnimy to jedynie w przypadku, gdy $2 \mid n$ — przypadek przeciwny wymaga analogicznego rozumowania.

Załóżmy nie wprost, że opóźnienie dłuższe niż $m-1$ zaczynałoby się od słowa Lyndona l długości $k < n$, $k \nmid n$. Przyjrzyjmy się dokładniej wartości k.

- Jeżeli $k < m$, to słowo Lyndona l' wygenerowane w Algorytmie L zaraz po l miałoby długość co najmniej $\lfloor \frac{n}{k} \rfloor \cdot k + 1$. Jest tak dlatego, że $l[1] < Z$, a ta litera powtarza się w rozszerzeniu okresowym l m.in. na pozycji $\lfloor \frac{n}{k} \rfloor \cdot k+1$ (warto w tym miejscu odnotować, że $\lfloor \frac{n}{k} \rfloor \cdot k < n$, gdyż $k \nmid n$). Zauważmy, że

  $$\left\lfloor \frac{n}{k} \right\rfloor \cdot k + 1 = n - n \bmod k + 1 \geqslant n-(k-1)+1 \geqslant n-(m-2)+1 = n-(m-3).$$

  Na podstawie Lematu 2.3.1 wiemy, że każde kolejne słowo wygenerowane po l' w Algorytmie L będzie co najmniej o jedną literę dłuższe od poprzedniego, aż do uzyskania słowa o długości n. To ogranicza długość opóźnienia algorytmu, rozpoczynającego się od słów l oraz l', przez $1+(m-3) = m-2 < m-1$.

- $k \neq m$, gdyż $m \mid n$.

- Jeżeli

  $$k \geqslant m+1 = (n-m)+1 = n-(m-1),$$

  to znów na podstawie Lematu 2.3.1 mamy, że długość opóźnienia rozpoczynającego się od l nie może przekroczyć $m-1$.

To kończy dowód, gdyż przeanalizowaliśmy wszystkie możliwe wartości $|l| = k$. ■

Fakt, że maksymalne opóźnienie Algorytmu L jest rzędu $\Theta(n)$, oznacza, że pesymistyczna złożoność czasowa wyznaczania pojedynczej litery słowa L jest rzędu $O(n^2)$. Z dowodu Faktu 4.4.1 można nawet wywnioskować, że analizowana złożoność to $\Theta(n^2)$, gdyż taki jest chociażby rząd wielkości łącznej liczby wykonań pętli **for** z wiersza 13. Algorytmu L dla ciągu słów Lyndona skonstruowanego w dowodzie.

W dalszej części niniejszej sekcji prezentujemy alternatywną implementację Algorytmu L, w której każdy symbol L jest generowany w pesymistycznej złożoności obliczeniowej stałej. W algorytmie wykorzystywane są dwie tablice rozmiaru $\Theta(n)$:

- $a_1 \dots a_n$ reprezentuje, podobnie jak w Algorytmie L, rozszerzenie okresowe bieżącego słowa Lyndona. W przeciwieństwie do Algorytmu L, tym razem całe rozszerzenie nie będzie uzupełniane od razu po znalezieniu nowego słowa Lyndona. Dokładniej, w dowolnym momencie działania algorytmu mamy zagwarantowane jedynie to, że $a_1 \dots a_i$ jest rozszerzeniem okresowym bieżącego słowa Lyndona, gdzie i to indeks aktualnie rozważanej pozycji słowa.

- Wartości $Z_0, Z_1, \dots, Z_n$ są zdefiniowane następująco:

$$Z_j = \max\{k \geqslant 0 : a_j = \dots = a_{j-k+2} = a_{j-k+1} = Z\}$$

  Podobnie jak to ma miejsce w przypadku tablicy a, również i tutaj zakładamy, że Z_j ma poprawną wartość jedynie dla $j \leqslant i$. $Z_0 = 0$ pełni rolę wartownika.

Na początku algorytmu zakładamy, że $a_1 = -1$, $Z_0 = 0$ (nie wymagamy żadnych konkretnych wartości pozostałych komórek tych tablic), natomiast $i = 0$.

Dalej, w algorytmie używane są zmienne *pop* oraz *nast*; przeznaczone są one do przechowywania indeksów ostatnich liter konstruowanych słów Lyndona: poprzedniego i następnego. Innymi słowy, zakładamy, że ostatnio zwiększaną komórką tablicy a była a_{pop}, natomiast kolejną komórką tej tablicy, która powinna zostać zwiększona, jest a_{nast}. Zwiększenia rozumiemy przy tym w sensie wiersza 12. Algorytmu L. Początkową wartością *pop* oraz *nast* jest 1.

Ostatnią zmienną wykorzystywaną w algorytmie jest zmienna boolowska *wypis*. Służy ona do obsługi wypisywania słowa Lyndona $a_1 a_2 \dots a_{pop}$, jeżeli $pop \mid n$ oraz $pop < n$. Wartością początkową tej zmiennej jest **false**.

Uwaga 15. Działanie mod jest w poniższym algorytmie oraz jego analizie zdefiniowane trochę inaczej niż zazwyczaj — jedyną różnicą jest to, że w przypadku, gdy $b \mid a$, jako a mod b zamiast 0 zwracane jest b.

```
1:  { Algorytm L′: Kolejne symbole słowa L wyznaczane w O(1). }
2:  function dajLiterę() : integer
3:    if a_1 = Z then KONIEC;
4:    i := (i + 1) mod n;
5:    if (pop | n) and (i = pop + 1) then begin
6:      if not wypis then begin
7:        { Początek wypisywania słowa Lyndona a_1...a_pop długości pop | n, pop < n. }
8:        i := 1; wypis :=true;
9:      end else
10:       { Słowo a_1...a_pop zostało wypisane, można kontynuować. }
11:       wypis :=false;
12:   end
```

13: $a_i := a_{i \bmod pop}$; { rozszerzenie okresowe słowa $a_1 \ldots a_{pop}$ }
14: **if** $i = nast$ **then** $a_i := a_i + 1$;
15: **if** $a_i < Z$ **then** $Z_i := 0$; **else** $Z_i := Z_{i-1} + 1$;
16: **if** $i = nast$ **then begin**
17: { Nastąpiła inkrementacja, mamy nowe słowo Lyndona. }
18: $pop := nast$;
19: $nast := n - Z_{n \bmod pop}$;
20: **end**
21: **return** a_i;
22: **end**

Niech $l_{j_1}, l_{j_2}, \ldots, l_{j_m}$ oznaczają wszystkie słowa Lyndona długości n, posortowane leksykograficznie. Wówczas, za wyjątkiem wierszy 5.–12., działanie algorytmu polega na konstruowaniu i wypisywaniu kolejnych słów l_{j_k}, czemu odpowiadają kolejne przejścia zmiennej i od 1 do n. Zastanówmy się, jak powinno wyglądać przejście od l_{j_k} do $l_{j_{k+1}}$. Niech $\lambda_1, \ldots, \lambda_{k'}$ oznaczają wszystkie słowa Lyndona, które są generowane w Algorytmie L między l_{j_k} a $l_{j_{k+1}}$. Z Lematu 2.3.1 wiemy, że

$$|\lambda_1| < |\lambda_2| < \ldots < |\lambda_{k'}| < |l_{j_{k+1}}| = n$$

Dalszą część dowodu przeprowadzimy przez indukcję po liczbie wywołań funkcji dajLiterę. Szeroko rozumianą bazę indukcji, czyli opis przebiegu działania algorytmu aż do momentu wygenerowania $l_{j_1} = 0^{n-1}1$, pomijamy.

Załóżmy, że w pewnym momencie algorytmu zostało wygenerowane (i wypisane) słowo l_{j_k}. To oznacza, że wówczas $i = pop = n$. Fakt, że wówczas $nast = |\lambda_1|$, wynika z następującego lematu.

Lemat 4.4.2. *Wzór z wiersza 19. Algorytmu L′ jest poprawny. Dokładniej, jeżeli w pewnym momencie działania algorytmu zachodzi $i = pop$, gdzie pop oznacza indeks litery słowa a, która została ostatnio powiększona (wiersz 16.), to kolejną literą tablicy a, która powinna zostać powiększona, jest*

$$n - Z_{n \bmod pop}.$$

Dowód. Przede wszystkim zauważmy, że $n \bmod pop \leqslant pop$ (nierówność nieostra ze względu na Uwagę 15), a równość $i = pop$, wobec definicji tablicy Z, implikuje, że wartość $Z_{n \bmod pop}$ jest aktualna (a dowód poprawności tej definicji w oparciu o wiersz 15. algorytmu jest trywialny).

Słowo Lyndona $a_1 \ldots a_{nast}$ powinno zostać skonstruowane z poprzedniego — $a_1 \ldots a_{pop}$ — za pomocą rozszerzenia okresowego, a następnie inkrementacji najdalszej możliwej litery mniejszej niż Z. W rozszerzeniu okresowym długości n słowa $a_1 \ldots a_{pop}$ ostatnimi $n \bmod pop$ literami są

$$a_1 \ldots a_{n \bmod pop}.$$

Ponieważ $a_1 < Z$, to

$$Z_{n \bmod pop} < (n \bmod pop) - 1$$

i $Z_{n \bmod pop}$ jest równe liczbie końcowych liter Z szukanego rozszerzenia okresowego. To oznacza, że inkrementacja litery, prowadząca do otrzymania słowa $a_1 \ldots a_{nast}$, powinna nastąpić właśnie na pozycji $n - Z_{n \bmod pop}$. ■

Przyjrzyjmy się kolejnym krokom algorytmu od momentu wygenerowania l_{j_k}. Zmienna i będzie przebiegać wartości $1, \ldots, |\lambda_1| - 1$ (wiersz 4.), generując kolejne litery λ_1 za pomocą rozszerzenia okresowego l_{j_k} (wiersz 13.). Kiedy $i = |\lambda_1| = nast$, to nastąpi przepisanie (wiersz 13.) i inkrementacja (wiersz 14.) litery a_i, dzięki czemu $a_1 \ldots a_i = \lambda_1$. Z Lematu 2.3.1 wiemy, że λ_1 jest prefiksem $l_{j_{k+1}}$, a zatem zwracane w kolejnych krokach litery λ_1 odpowiadają zarazem literom $l_{j_{k+1}}$.

Po wypisaniu całego λ_1, zmienna *nast* (znów na podstawie Lematu 4.4.2) przyjmie wartość $|\lambda_2| > |\lambda_1|$ i rozpocznie się generowanie słowa λ_2, poczynając od litery o indeksie $|\lambda_1|+1$. Po wygenerowaniu λ_2, które na mocy Lematu 2.3.1 również jest prefiksem $l_{j_{k+1}}$, rozpocznie się generowanie λ_3 itd. Na końcu tego procesu zostanie wygenerowane $\lambda_{k'}$, a następnie dokończone zostanie samo słowo $l_{j_{k+1}}$.

Na koniec pozostała nam analiza wierszy 5.–12. Odpowiadają one sytuacji, kiedy długość któregoś ze słów $\lambda_1, \ldots, \lambda_{k'}$ jest podzielna przez n.

Uwaga 16. Można pokazać, że takim słowem może być tylko i wyłącznie λ_1, ale nie jest to istotne dla analizy poprawności Algorytmu L′.

Jeżeli $|\lambda_j| = pop \mid n$, to to oznacza, że fragment $a_1 \ldots a_{pop}$ musi się pojawić w ciągu L wielokrotnie — przy okazji λ_j oraz przy okazji $l_{j_{k+1}}$, którego λ_j jest prefiksem. Zmienna *wypis* odpowiada za obsługę takich sytuacji. Konkretniej, po wykryciu słowa Lyndona o długości dzielącej n, ale mniejszej niż n (wiersz 5.), zmienna *wypis* staje się prawdą i rozpoczyna się wypisywanie słowa λ_j, które odbywa się przez $|\lambda_j|$ kolejnych kroków algorytmu (rozpoczęcie tego procesu stanowi przypisanie $i := 1$ w wierszu 8.). Na końcu wykonanie algorytmu wraca na poprzedni tor (wiersz 11.). To pokazuje, że opisana sytuacja jest faktycznie obsługiwana poprawnie.

Zakończenie działania algorytmu następuje, gdy na początku danego wywołania funkcji dajLiterę() zachodzi $a_1 = Z$ (wiersz 3.), co faktycznie oznacza, że zostało już wygenerowane największe leksykograficznie słowo Lyndona nad Σ. To, wraz z wcześniejszą analizą, kończy wyjaśnienie działania i szkic dowodu poprawności Algorytmu L′.

Natychmiastowym wnioskiem z analizy pseudokodu Algorytmu L′ jest to, że *złożoność obliczeniowa każdego wywołania funkcji* dajLiterę() *to* $O(1)$.

Na koniec dodajmy, że w pracy [22] zaprezentowano pewien algorytm generowania minimalnych leksykograficznie słów długości n, co do którego istnieje hipoteza, że może zostać zaimplementowany tak, że pesymistyczna złożoność czasowa wygenerowania pojedynczego słowa jest stała. Jeżeli hipoteza jest prawdziwa, to ten algorytm mógłby także posłużyć do generowania kolejnych liter ciągu L w $O(1)$.

4.5. Algorytm zachłanny

W poprzedniej sekcji skupiliśmy się na konstrukcji bardzo efektywnej wersji algorytmu generowania ciągu L. W niniejszej sekcji zaprezentujemy inny algorytm konstrukcji L, oparty na Stwierdzeniu 4.3.1, który jest z kolei bardzo prosty i krótki w zapisie.

W algorytmie będziemy utrzymywać zbiór S tych słów z Σ^n, które wystąpiły już jako podsłowa wygenerowanego fragmentu ciągu, a dokładniej jego cyklicznej wersji. Konstrukcję rozpoczynamy od słowa $s = Z^n$ — sufiksu słowa L. W każdym z kolejnych M^n kroków dokładamy kolejną literę ciągu, wybierając ją jako najmniejszy symbol, który, umieszczony na końcu ciągu, nie tworzy słowa obecnego już w S.

1: { **Algorytm R:** Metoda zachłanna generowania ciągu L. }
2: $s := Z^n$; $S := \emptyset$;

3: **for** $i := 1$ **to** M^n **do begin**
4: $c := 0$;
5: **while** $s[2..n]c \in S$ **do** $c := c + 1$;
6: $s := s[2..n]c$;
7: Wypisz(c);
8: $S.insert(s)$;
9: **end**

Oczywiste jest, że jeżeli Algorytm R wygeneruje słowo złożone wyłącznie z liter z Σ (czyli jeżeli w pętli **while** z wiersza 5. zachodzi niezmiennik $c < M$), to słowo to będzie minimalnym leksykograficznie ciągiem de Bruijna rzędu n nad Σ. Fakt, że c nigdy nie przekroczy Z, wynika ze Stwierdzenia 4.3.1, gdyż na jego mocy w słowie $L = l_1 \dots l_t$ każda kolejna litera jest najmniejszą niepowodującą powstania już wygenerowanego słowa długości n. To kończy dowód poprawności Algorytmu R.

Okazuje się, że na Algorytm R można także spojrzeć w inny sposób, który został opisany w pracy [19]. Otóż działanie tego algorytmu jest bardzo podobne do zastosowania standardowej metody poszukiwania cyklu Eulera w grafie G_B, z jedną istotną różnicą — po znalezieniu cyklu zakładamy, że musi to być już cykl Eulera, tymczasem w standardowym algorytmie po znalezieniu cyklu zaczyna się dobudowywanie do niego rekurencyjnie wyznaczanych cykli Eulera w silnie spójnych składowych pozostałej części grafu.

Aby wyjaśnić fenomen polegający na tym, że Algorytm R działa poprawnie pomimo wspomnianej istotnej różnicy, zinterpretujemy go jeszcze inaczej, a mianowicie pod kątem Kryterium 3.4.1. Rozważmy podgraf H grafu G_B, wyznaczony przez krawędzie o etykietach Z. Zauważmy, że jeżeli $u \in \Sigma^{n-1}$ kończy się i wystąpieniami litery Z, to słowo v, do którego prowadzi z u krawędź z H, kończy się $\min(i+1, n-1)$ wystąpieniami litery Z. To pokazuje, że słowa z Σ^n możemy ustawić poziomami, dzieląc je na:

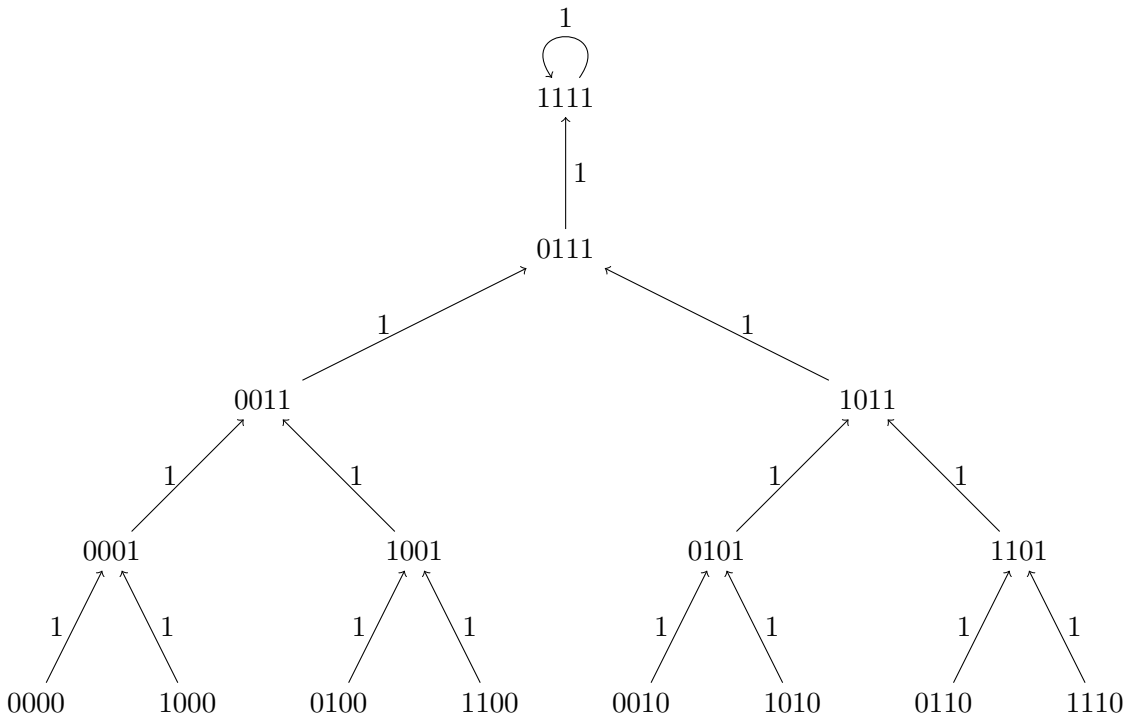
- słowa, które nie kończą się literą Z (poziom 0);
- słowa, które kończą się jednym wystąpieniem litery Z (poziom 1);
- słowa, które kończą się dwoma wystąpieniami litery Z (poziom 2);
- ...
- jedyne słowo Z^{n-1}, które kończy się $n-1$ wystąpieniami litery Z (poziom $n-1$).

Na i-tym poziomie, z każdego słowa wychodzi dokładnie jedna krawędź z H do jakiegoś słowa poziomu $i+1$, poza poziomem $n-1$, na którym ze słowa wychodzi pętla.

Na podstawie przeprowadzonej analizy struktury H łatwo wnioskujemy, że H jest skierowanym drzewem ukorzenionym w Z^{n-1} z dodaną jedną pętlą (patrz przykład takiego grafu na końcu sekcji).

Niech zatem T będzie skierowanym drzewem rozpinającym G_B, ukorzenionym w wierzchołku odpowiadającym Z^{n-1}, powstałym z H poprzez usunięcie jedynej pętli. W G_B będziemy poszukiwać cyklu Eulera za pomocą algorytmu opartego na Kryterium 3.4.1, wykorzystując drzewo T. Dodatkowo, posortujemy krawędzie wychodzące z wierzchołków G_B niemalejąco po wartościach etykiet i w tej kolejności będziemy wybierać kolejne krawędzie w przypadku niejednoznaczności warunku pochodzącego z Kryterium. Jak łatwo zauważyć, kolejno przechodzone w algorytmie krawędzie będą (wliczając też krawędzie z T, które zawsze są wybierane jako ostatnie) zawsze najmniejsze leksykograficznie spośród niezużytych. Ponieważ poszukiwanie cyklu zaczynamy od korzenia drzewa, to znaczy od słowa Z^{n-1}, to algorytm

ten działa rzeczywiście dokładnie tak samo jak Algorytm R. To pokazuje, że Algorytm R jest w gruncie rzeczy alternatywną (dużo prostszą) implementacją algorytmu poszukiwania cyklu Eulera w G_B.



Przykład podgrafu H indukowanego przez krawędzie z etykietami $Z = 1$ dla $n = 5$, $M = 2$.

Rozdział 5

BWT minimalnych ciągów de Bruijna

Transformata Burrowsa-Wheelera [3] to algorytm powszechnie używany w kompresji bezstratnej. Wynik transformacji $BWT(s)$ danego słowa s otrzymuje się, sortując leksykograficznie wszystkie rotacje cykliczne s i konkatenując ostatnie litery tych rotacji w posortowanej kolejności. O przydatności transformaty stanowi fakt, że dla wielu rodzin słów $BWT(s)$ jest o wiele lepiej kompresowalne za pomocą algorytmów typu *move-to-front* (MTF) oraz *run-length encoding* (RLE) niż samo s. Transformacja słowa $s \in \{0, 1, \ldots, M-1\}^l$ może zostać wykonana w złożoności czasowej $O(l \log l + M)$, a transformacja odwrotna — w złożoności $O(l + M)$ [3].

Zauważmy, że transformację Burrowsa-Wheelera ciągów de Bruijna można przeprowadzić efektywnie bez użycia wyrafinowanych algorytmów. Faktycznie, niech s będzie ciągiem de Bruijna rzędu n nad $\Sigma = \{0, 1, \ldots, M-1\}$, $|s| = M^n = N$, i niech $S = \{s_1, \ldots, s_N\}$ będzie zbiorem wszystkich rotacji cyklicznych s. $BWT(s)$ można wyznaczyć z definicji w złożoności czasowej $O(N^2)$ przez posortowanie pozycyjne (ang. *Radix Sort*) słów ze zbioru S. Zauważmy jednakże, że dokładnie taki sam wynik otrzymamy, posortowawszy leksykograficznie słowa ze zbioru $S' = \{f(s_1), \ldots, f(s_N)\}$, gdzie

$$f(u) = u[1..n] \cdot u[N].$$

Wynika to stąd, że każde słowo z Σ^n jest prefiksem dokładnie jednego słowa z S, więc dowolne dwa różne słowa z S różnią się na co najmniej jednej pozycji spośród $1, \ldots, n$. Wreszcie sortowanie słów ze zbioru S' można zrealizować w złożoności czasowej $O(Nn) = O(M^n n)$.

W niniejszym rozdziale zbadamy strukturę BWT dla minimalnych leksykograficznie ciągów de Bruijna. Na początek przyjrzyjmy się kilku przykładom tych transformat.

n	M	BWT ciągu de Bruijna rzędu n nad $\Sigma = \{0, 1, \ldots, M-1\}$
2	2	1010
3	2	10011010
4	2	1001010110101010
5	2	10010101010101011010010110101010
6	2	10010101010101010101010101010110101010011001011010011010101010
2	3	201021201
3	3	201021021012021012201021201
2	4	3012031201323012
3	4	30120312031203120123031201320132012301230132012330120312013230 12

Poniższa tabela przedstawia posortowane leksykograficznie rotacje cykliczne minimalnego ciągu de Bruijna rzędu 3 dla $M = 3$, czyli słowa

$$L = \texttt{000100201101202102211121222}.$$

Ostatnie litery kolejnych rotacji składają się na transformatę tego ciągu, czyli

$$BWT(L) = \texttt{201021021012021012201021201}.$$

i	i-ty obrót cykliczny s
1	000100201101202102211121222
2	001002011012021022111212220
3	002011012021022111212220001
4	010020110120210221112122200
5	011012021022111212220001002
6	012021022111212220001002011
7	020110120210221112122200010
8	021022111212220001002011012
9	022111212220001002011012021
10	100201101202102211121222000
11	101202102211121222000100201
12	102211121222000100201101202
13	110120210221112122200010020
14	111212220001002011012021022
15	112122200010020110120210221
16	120210221112122200010020110
17	121222000100201101202102211
18	122200010020110120210221112
19	200010020110120210221112122
20	201101202102211121222000100
21	202102211121222000100201101
22	210221112122200010020110120
23	211121222000100201101202102
24	212220001002011012021022111
25	220001002011012021022111212
26	221112122200010020110120210
27	222000100201101202102211121

5.1. Struktura BWT ciągów de Bruijna

Stwierdzenie 5.1.1. *BWT ciągu de Bruijna s można podzielić na M^{n-1} kolejnych podsłów długości M: $u_1 \dots u_{M^{n-1}}$, z których każde jest permutacją zbioru Σ.*

Dowód. Każde z podsłów u_i odpowiada, zgodnie z definicją $BWT(s)$, M kolejnym (leksykograficznie) rotacjom cyklicznym s. Zauważmy, że wszystkie te rotacje zaczynają się od tego samego słowa $v_i \in \Sigma^{n-1}$, będącego zapisem liczby $i-1$ w układzie pozycyjnym o podstawie M. Jest tak dlatego, że v_i występuje jako podsłowo $s \cdot s[1..n-1]$ dokładnie M razy, co jest związane z wystąpieniami słów: v_i0, v_i1, …, v_iZ. Z definicji ciągu de Bruijna wynika, że litera poprzedzająca v_i w każdym z tych wystąpień jest inna. I właśnie te poprzedzające litery składają się na rozważane podsłowo u_i. ■

Skupimy się odtąd na analizie struktury podsłów u_i dla transformat minimalnych leksykograficznie ciągów de Bruijna. Pokażemy w szczególności, że określone przez nie permutacje zbioru Σ nie mogą być całkowicie dowolne.

Niech więc u_i będzie podsłowem $BWT(L)$ (gdzie L oznacza minimalny leksykograficznie ciąg de Bruijna) odpowiadającym $v_i \in \Sigma^{n-1}$ jak w dowodzie Stwierdzenia 5.1.1. To oznacza, że podsłowami (cyklicznej wersji) L są słowa:

$$u_i[1]v_i0,\ u_i[2]v_i1,\ \ldots,\ u_i[n]v_iZ.$$

Na mocy Stwierdzenia 4.3.1 mamy, że słowa:

$$v_i0,\ v_i1,\ \ldots,\ v_iZ$$

występują jako podsłowa L właśnie w tej kolejności (przypomnijmy, że zakładamy, że słowa Z^j0^{n-j} dla $j > 0$ występują na samym początku L). Stąd również i słowa:

$$u_i[1]v_i,\ u_i[2]v_i,\ \ldots,\ u_i[n]v_i$$

występują w tej właśnie kolejności jako podsłowa L (z jednym wyjątkiem — kiedy $v_i = Z^{n-1}$, to Zv_i zawsze występuje na samym końcu L). Permutacja wyrażona przez słowo u_i określa więc po prostu kolejność występowania w ramach L słów $0v_i,\ 1v_i,\ \ldots,\ Zv_i$.

Dla uproszczenia dalszych zapisów przyjmijmy następującą definicję, podobną do Definicji 4.3.1.

Definicja 5.1.1. *Jeżeli $u = a_1a_2\ldots a_n$ oraz $v = (a_1+1)a_2\ldots a_n$, to będziemy używać zapisu:*

$$v = 1 + u \quad \textit{lub równoważnie} \quad u = -1 + v.$$

Okazuje się, że prawdziwy jest następujący lemat, podobny do Lematu 4.3.2.

Lemat 5.1.2. *Jeżeli $v = 1 + u$, to $pL(u) < pL(v)$.*

Dowód. Niech

$$u' = u[2..n] \cdot u[1],\ v' = v[2..n] \cdot v[1]$$

będą rotacjami cyklicznymi odpowiednio u oraz v. Wówczas $v' = u' + 1$, więc na mocy Lematu 4.3.2 oraz definicji pierwiastka Lyndona mamy

$$pL(u) = pL(u') < pL(v') = pL(v).$$

■

Co więcej, zachodzi także następujące stwierdzenie, podobne do Stwierdzenia 4.3.1.

Stwierdzenie 5.1.3. *Dla dowolnych dwóch słów u oraz $v = 1 + u$ długości n nad alfabetem Σ, jeżeli $v[1] < Z$, to u występuje przed v jako podsłowo L.*

Dowód. Nierówność $v[1] < Z$ oznacza, że każde ze słów u, v jest kategorii A lub C (przyjmujemy takie samo nazewnictwo kategorii jak w dowodzie Twierdzenia 4.1.1). To oznacza, że wystąpienie u w ramach L zaczyna się od słowa $pL(u)$, natomiast wystąpienie v — od $pL(v)$ (patrz rys. związane z odpowiednimi przypadkami albo dowód Stwierdzenia 4.3.1). Na mocy Lematu 5.1.2 mamy $pL(u) < pL(v)$, co kończy dowód. ■

Na mocy Stwierdzenia 5.1.3 oraz dotychczasowych rozważań o permutacjach wyrażonych przez słowa u_i, jeżeli z L usunąć wszystkie wystąpienia litery Z, to w wyniku otrzymuje się słowo

$$(01\ldots(Z-1))^{M^{n-1}}.$$

Pozostało do rozstrzygnięcia, w jaki sposób rozmieszczone są wystąpienia litery Z w $BWT(L)$. Już same przykłady pokazują, że ich układ nie jest tak samo regularny, jak to ma miejsce dla pozostałych liter.

Zastanówmy się jednak, jakich regularności można się w ich rozmieszczeniu dopatrzeć. Niech $u, v \in \Sigma^n$, $v = 1 + u$ oraz $v[1] = Z$. Jeżeli v nie jest kategorii B, to pokażemy, że w ramach L u zawsze poprzedza v. Przypomnijmy, że na mocy Lematu 5.1.2 mamy $pL(u) < pL(v)$ i oznaczmy $l_k = pL(u)$, $l_m = pL(v)$. Zauważmy, że u może być kategorii A lub C, natomiast v — dowolnej spośród: A, C, D. Prawdziwość tezy w przypadkach AA, AC, CA oraz CC wynika natychmiast z nierówności $l_k < l_m$ (patrz także dowód Stwierdzenia 5.1.3). Przypadki AD oraz CD można rozpatrzyć analogicznie jak w dowodzie Stwierdzenia 4.3.1; dokładniej, jeżeli $l_k < l_{m-1}$, to teza oczywiście zachodzi, a w przypadku $l_k = l_{m-1}$ wystąpienie słowa u w ramach l_k musi się z pewnych przyczyn zaczynać wcześniej niż wystąpienie v w ramach $l_{m-1} = l_k$.

Widać więc, że jedyne nieregularności mogą wystąpić, gdy v spełnia warunki Przypadku B. Poniższe stwierdzenie pokazuje, że w pewnych specjalnych okolicznościach także i te nieregularności zanikają.

Stwierdzenie 5.1.4. *Jeżeli $M = 2$, to pierwsza połowa słowa L jest postaci*

$$10(01)^a.$$

Dowód. Zacznijmy od tego, że prefiks 10 słowa $BWT(L)$ bierze się stąd, że $1 \cdot 0^{n-1}$ występuje w L (tuż) przed 0^n.

Pierwsza połowa słowa $BWT(L)$ pochodzi z ostatnich liter rotacji cyklicznych L zaczynających się od $0t$ dla $t \in \Sigma^{n-1}$. Musimy więc pokazać, że poza wyżej opisanym wyjątkiem, dla dowolnego $w \in \Sigma^{n-2}$ słowo $u = 00w$ występuje w L przed $v = 1 + u = 10w$. u może więc być kategorii A lub C, natomiast v — dowolnej.

Na mocy dotychczasowych rozważań, jeżeli tylko v nie jest kategorii B, to niezależnie od szczególnej postaci słów u i v, u występuje przed v w ramach L. Przejdźmy więc do przypadków z rodziny XB, $X \in \{A, C\}$. Jeżeli v jest kategorii B, to $v = \alpha\beta$, gdzie $\alpha = 1$ i $\beta = 0w$ oraz $l_i = pL(v) = \beta\alpha$ jest Lyndona.

Uwaga 17. $|l_i| = n$, więc l_{i-1} istnieje.

Jaka jest ostatnia litera słowa l_{i-1}? Jedynym słowem Lyndona nad Σ zakończonym zerem jest 0. Jeśli $l_{i-1} = 0$, to $l_i = 0^{n-1} \cdot 1$ i $v = 1 \cdot 0^{n-1}$; ten przypadek został już jednakże rozpatrzony na początku dowodu. W każdej innej sytuacji l_{i-1} kończy się jedynką, a wówczas v jest podsłowem $l_{i-1}l_i$ — wystąpienie v w ramach L zaczyna się na ostatniej literze przed l_i. Na mocy Lematu 5.1.2 mamy teraz $pL(u) < pL(v) = l_i$. Początek wystąpienia słowa u w ramach L znajduje się w ramach słowa $pL(u)$, gdyż u jest kategorii A lub C, a stąd wynika już, że u występuje przed v jako podsłowo L. ■

5.2. Efektywny algorytm wyznaczania p-tej litery $BWT(L)$

W niniejszej sekcji prezentujemy algorytm, za pomocą którego dla danej liczby p, $1 \leqslant p \leqslant M^n$, można wyznaczyć $BWT(L)[p]$ w złożoności czasowej $O(n \log M + M)$. Ze względu na pewien poziom skomplikowania algorytmu, nie prezentujemy jego pseudokodu, a jedynie opis słowny.

Na początku algorytmu wyznaczamy indeks i słowa u_i, którego pewna litera odpowiada p-tej literze $BWT(L)$ (patrz Stwierdzenie 5.1.1):

$$i = \left\lceil \frac{p}{M} \right\rceil.$$

Zauważmy, że p-ta litera $BWT(L)$ jest dokładnie

$$(p - (i-1) \cdot M)\text{-tą}$$

literą u_i. Naszym celem będzie więc odtąd wyznaczenie słowa u_i.

Dla danego i możemy w złożoności czasowej $O(M)$ wyznaczyć słowo v_i, gdyż jest ono zapisem liczby $i-1$ w układzie pozycyjnym o podstawie M. (Zakładamy przy tym, że działania na liczbach rzędu $O(M^n)$ możemy wykonywać w stałej złożoności czasowej). Słowo u_i określa kolejność wystąpień słów $0v_i,\ 1v_i,\ \ldots,\ Zv_i$ w ramach L. Jak już pokazaliśmy, u_i zawiera podciąg $0, 1, \ldots, Z-1$ i jedyną niewiadomą w tym słowie jest położenie litery Z. Pozycję tej litery wyznaczymy za pomocą wyszukiwania binarnego (w związku z tym do złożoności czasowej algorytmu dojdzie nam czynnik $\log M$). W tym celu będziemy musieli wielokrotnie odpowiadać na pytanie, czy dla danej litery $X < Z$ słowo Xv_i występuje przed Zv_i jako podsłowo L, czy też nie.

Przydatna nam do tego będzie umiejętność przyporządkowywania słowom ich kategorii.

Lemat 5.2.1. *Dla danego słowa $u \in \Sigma^n$, w złożoności czasowej $O(n)$ można wyznaczyć:*

- *kategorię, do której to słowo przynależy (A, B, C lub D);*
- *zapis tego słowa w postaci $u = (\alpha\beta)^d$, gdzie $\beta\alpha$ jest słowem Lyndona ($d = 1$ dla kategorii A oraz B).*

Dowód. Zacznijmy od tego, że pierwiastek pierwotny v słowa u można wyznaczyć w złożoności $O(n)$ za pomocą funkcji prefiksowej P z algorytmu Knutha-Morrisa-Pratta [7]. Ponieważ jest to klasyczny algorytm tekstowy, to przypomnijmy jedynie, że jeżeli $n - P(n) \mid n$, to pierwiastek pierwotny słowa u ma długość $n - P(n)$, a w przeciwnym przypadku u jest pierwotne. Na tej podstawie wyznaczamy wartość d, a zarazem to, czy u jest kategorii A lub B, czy też C lub D.

Wiemy, że $v = \alpha\beta$. Aby wyznaczyć słowa α i β, policzymy liczbę liter, o które trzeba obrócić cyklicznie v, aby stało się równe pierwiastkowi Lyndona u, czyli $pL(u) = \beta\alpha$. W tym celu potrzebujemy algorytmu, który mając dane słowo pierwotne, wyznaczy jego najmniejszą leksykograficznie rotację cykliczną.

Maksymalny leksykograficznie sufiks słowa można wyznaczyć liniowo względem jego długości za pomocą algorytmu Duvala [1] lub odpowiedniej modyfikacji algorytmu wyznaczającego funkcję prefiksową [7]. Okazuje się, że używając takiego algorytmu jako czarnej skrzynki (ang. *Black Box*), można znaleźć szukany obrót słowa v. W tym celu wystarczy odwrócić porządek na literach i uruchomić jeden ze wspomnianych algorytmów na słowie vv.

Uzasadnijmy poprawność opisanego algorytmu. Zauważmy przede wszystkim, że wynikowy sufiks będzie miał długość większą niż v, co wynika z faktu, że dla $0 \leqslant i \leqslant |v|$ sufiks słowa vv długości $|v| + i$ jest większy leksykograficznie niż sufiks długości i — gdyż drugi z tych sufiksów jest prefiksem pierwszego. Każde dwa sufiksy vv o długościach większych niż $|v|$ różnią się na jednej spośród pierwszych $|v|$ liter, gdyż ich prefiksy długości $|v|$ są rotacjami cyklicznymi v, a v jest pierwotne (patrz też Fakt 1.2.3). Przy porównywaniu słów, z których żadne nie jest prefiksem drugiego, odwrócenie porządku na literach oznacza dokładnie odwrócenie porządku słów (co bardzo prosto wynika z definicji porządku leksykograficznego). To oznacza, że prefiks długości $|v|$ znalezionego sufiksu będzie najmniejszym leksykograficznie obrotem cyklicznym v, czyli szukanym słowem Lyndona równoważnym cyklicznie v. ■

Na mocy dotychczasowych rozważań, jeżeli Zv_i nie jest kategorii B, to nawet bez wyszukiwania binarnego można orzec, że $u_i = 01\ldots Z$. Niech więc Zv_i spełnia warunki Przypadku B; musimy umieć sprawdzać, czy słowo to występuje jako podsłowo L przed słowem Xv_i, czy też po nim. Xv_i jest kategorii A lub C, więc wystąpienie tego słowa w ramach L rozpoczyna się do słowa Lyndona $l_k = \beta\alpha$ (które na mocy Lematu 5.2.1 możemy wyznaczyć w złożoności czasowej $O(n)$), a kończy się na słowie l_{k+1}. Co więcej, na podstawie rozważań poczynionych dla przypadków A oraz C w rozdziale 4 można łatwo obliczyć, o ile liter słowo Xv_i nachodzi na l_{k+1} — będzie to

$$|\beta|$$

w Przypadku A oraz

$$(d-1)\cdot(|\alpha|+|\beta|)+|\beta|$$

w Przypadku C. Z kolei wystąpienie $Zv_i = \alpha'\beta'$ kończy się w ramach L na l_m, czyli na najmniejszym słowie Lyndona spełniającym $\beta' \leqslant l_m$.

Do uzupełnienia rozumowania jest nam potrzebna umiejętność stwierdzenia, które ze słów l_{k+1} i l_m jest leksykograficznie mniejsze. Kłopot tkwi w tym, że teza Lematu 5.2.1 nie pomaga w policzeniu ani l_{k+1}, ani l_m. Okazuje się, że efektywne wyznaczenie l_m nie jest proste; autor tej pracy nie zna żadnego algorytmu o złożoności $o(n^2M)$, który na podstawie β' obliczałby l_m. Z tego względu zajmiemy się jedynie wyznaczeniem l_{k+1} i spróbujemy zaprojektować pozostałą część algorytmu bez dokładnej znajomości słowa l_m.

Lemat 5.2.2. *Słowo l_{k+1} można wyznaczyć na podstawie l_k w złożoności czasowej $O(n)$.*

Dowód. Szukany algorytm konstruuje się bardzo prosto na podstawie Algorytmu L′, za pomocą którego generowaliśmy kolejne litery L w złożoności $O(1)$. Wystarczy w tym celu:

- wypełnić tablicę a rozszerzeniem okresowym l_k;
- ustawić wskaźniki: *pop* na $|l_k|$, *nast* na najdalsze pole tablicy a zawierające literę mniejszą niż Z, i na $|l_k|$;
- usunąć z algorytmu tablicę Z;
- wywoływać funkcję dajLiterę() aż do końca wypisywania kolejnego słowa Lyndona, czyli do dojścia do wierszy 17.–19.

Ponieważ opisana modyfikacja algorytmu jest raczej techniczna niż koncepcyjna, to jej dokładniejszy opis oraz dowód poprawności pomijamy. ■

Załóżmy, że wyznaczyliśmy już słowo l_{k+1}. Mamy do rozważenia kilka przypadków:

a) jeżeli $l_{k+1} < \beta'$, to $l_{k+1} < l_m$ i wówczas Xv_i występuje przed Zv_i w ramach L;

b) jeżeli $l_k \geqslant \beta'$, to $l_m \leqslant l_k < l_{k+1}$ i Zv_i występuje przed Xv_i w ramach L;

c) jeżeli wreszcie $l_k < \beta' \leqslant l_{k+1}$, to $l_{k+1} = l_m$ i jeszcze nie wiadomo, które ze słów Xv_i, Zv_i występuje wcześniej.

Zajmijmy się — jedynym wciąż nierozstrzygniętym — podpunktem c). Aby porównać położenia Xv_i oraz Zv_i w tym podpunkcie, musimy jeszcze tylko wyznaczyć liczbę liter, o które Zv_i nachodzi na l_m. Tę wartość łatwo policzyć w każdym z podprzypadków B_1, B_2, tylko najpierw trzeba ustalić, z którym z tych podprzypadków mamy faktycznie do czynienia. Zauważmy, że w sytuacji z podpunktu c) znamy słowa $l_m = l_{k+1}$ oraz $l_{m-1} = l_k$. To pozwala

już bardzo łatwo dokonać żądanej klasyfikacji, gdyż Przypadek B_1 tym różni się od B_2, że słowo l_m jest rozszerzeniem okresowym l_{m-1} (patrz opisy tych przypadków), a ten warunek już bardzo łatwo sprawdzić w złożoności czasowej $O(n)$.

Udało nam się ostatecznie wyznaczyć sposób odpowiadania na pytania pojawiające się w wyszukiwaniu binarnym w złożoności czasowej $O(n)$. Stąd złożoność czasowa całego algorytmu to $O(n \log M + M)$.

Bibliografia

[1] L. Banachowski, K. Diks, W. Rytter, *Algorytmy i struktury danych*, Wydawnictwa Naukowo-Techniczne, 2006

[2] J. C. Bermond, R. W. Dawes, F. Ö. Ergincan, *De Bruijn and Kautz bus networks*, Networks 30, 1997

[3] M. Burrows, D. Wheeler, *A block sorting lossless data compression algorithm*, Technical Report 124, Digital Equipment Corporation, 1994

[4] K. Cattell, F. Ruskey, J. Sawada, C. R. Miers, M. Serra, *Fast Algorithms to Generate Necklaces, Unlabelled Necklaces and Irreducible Polynomials over GF(2)*, J. Algorithms 37, 2000

[5] C. Choffrut, J. Karhumäki, *Combinatorics of words*, Handbook of formal languages (ed.: G. Rozenberg, A. Salomaa), Springer, 1998

[6] F. Chung, P. Diaconis, R. Graham, *Universal cycles for combinatorial structures*, Discrete Math. 110, 1992

[7] M. Crochemore, W. Rytter, *Jewels of Stringology*, World Scientific Press, 2002

[8] N. G. de Bruijn, *A combinatorial problem*, Proc. Nederl. Akad. Wetensch. 49, 1946

[9] H. Fredricksen, *A survey of full length nonlinear shift register cycle algorithms*, SIAM Rev. 24, 1982

[10] H. Fredricksen, I. J. Kessler, *An algorithm for generating necklaces of beads in two colors*, Discrete Math. 61, 1986

[11] H. Fredricksen, J. Maiorana, *Necklaces of beads in k colors and k-ary de Bruijn sequences*, Discrete Math. 23, 1978

[12] I. J. Good, *Normal recurring decimals*, Journal of the London Mathematical Society 21, 1946

[13] R. L. Graham, D. E. Knuth, O. Patashnik, *Matematyka konkretna*, Wydawnictwo Naukowe PWN, 2006

[14] D. Jungnickel, *Graphs, Networks and Algorithms*, Springer, 2005

[15] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 2*, Addison-Wesley, 2005

[16] M. Lothaire, *Combinatorics on Words*, Cambridge University Press, 1983

[17] E. Moreno, *Lyndon words and de Bruijn sequences in a subshift of finite type*, proceedings of WORDS, 2003

[18] E. Moreno, *On the theorem of Fredricksen and Maiorana about De Bruijn sequences*, Adv. in Appl. Math., 2004

[19] E. Moreno, M. Matamala, *Minimum de Bruijn Sequence in a Language with Forbidden Substrings*, Workshop on Graph, 2004

[20] V. R. Rosenfeld, *Enumerating De Bruijn sequences*, MATCH Communications in Mathematical and in Computer Chemistry, 2002

[21] F. Ruskey, J. Sawada, *Generating necklaces and strings with forbidden substrings*, Lect. Notes Comput. Sci. 1858, 2000

[22] F. Ruskey, C. Savage, T. Wang, *Generating Necklaces*, J. Algorithms 13, 1992

[23] S. K. Stein, *The mathematician as an explorer*, Sci. Amer. 204, 1961

[24] Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki`