Yamátárájabhánasalagám

Wojciech GUZICKI, Warszawa

1. Ciągi de Bruijna

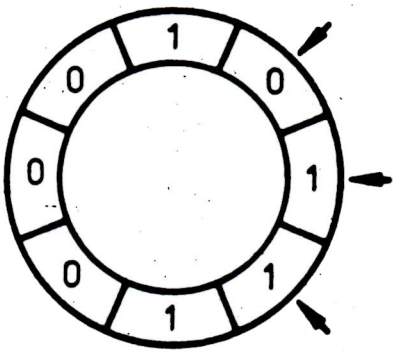
Tytuł jest sanskryckim słowem pomagającym w studiowaniu średniowiecznej poezji indyjskiej. Właściwie jest to słowo wymyślone specjalnie do tego celu, samo w sobie nie znaczy nic (czy jest więc rzeczywiście słowem sanskryckim?). Pomaga ono w zapamiętaniu nazw różnych rytmów. Składa się ono z dziesięciu sylab krótkich (nieakcentowanych) i długich (akcentowanych). Pierwsze trzy sylaby ya–má–tá dają rytm: krótka–długa–długa. Następnie, druga, trzecia i czwarta sylaba má–tá–rá dają rytm: długa–długa–długa. Podobnie trzecia, czwarta i piąta sylaba tá–rá–ja dają nowy rytm: długa–długa–krótka. Oczywiście jest dokładnie osiem różnych rytmów złożonych z trzech sylab krótkich lub długich. Wszystkie te rytmy występują w słowie yamátárájabhánasalagám, każdy tylko jeden raz. Nietrudno zauważyć, że najkrótsze takie słowo musi mieć co najmniej 10 sylab. Pierwsze dwie sylaby nie dają bowiem jeszcze żadnego rytmu trzysylabowego, każda następna sylaba daje co najwyżej jeden nowy rytm. Tych sylab musi więc być przynajmniej o dwie więcej niż wszystkich możliwych rytmów, zatem przynajmniej 10. Ponieważ mamy przykład słowa 10-sylabowego, więc możemy wypowiedzieć twierdzenie, że najkrótsze takie słowo ma dokładnie 10 sylab. Zauważmy jeszcze, że gdyby utworzyć analogiczne słowo w ten sposób, że najpierw wypiszemy trzy sylaby pierwszego rytmu, potem trzy sylaby drugiego itd., to otrzymalibyśmy słowo 24-sylabowe (8 rytmów po 3 sylaby). Mamy zatem znaczną oszczędność: zamiast 24 sylab tylko 10. O ile łatwiej jest zapamiętać słowo 10-sylabowe niż słowo 24-sylabowe, zwłaszcza nic nie znaczące słowo sanskryckie. Czytelnik może sobie wyobrazić trudności związane z zapamiętaniem (i wymówieniem) słowa dwa i pół raza dłuższego niż w tytule!

W hotelu Baltimore Hilton Inn wprowadzono zamki szyfrowe. Na drzwiach każdego pokoju znajduje się tarcza z dziesięcioma cyframi (taka jak w telefonie). Drzwi otwierają się po wykręceniu kodu składającego się z czterech cyfr. Zamek jest skonstruowany w taki sposób, że niezależnie od tego, jakie cyfry zostały wykręcone poprzednio, ważne są ostatnio wykręcone cztery cyfry. Jak powinien w najefektywniejszy sposób postępować lokator, który zapomniał szyfru do swojego pokoju? Może on oczywiście wykręcać po kolei wszystkie kombinacje czterech cyfr. Najpierw 0000, potem 0001, potem 0002 itd. Od razu widzimy, że ta metoda jest nieefektywna. Po wykręceniu 0000, wykręcenie następnych trzech zer już nic nie daje. Można było wykręcić jedynkę z takim samym efektem, jak poprzednio. Strategia najprostsza wymaga wykręcenia 40000 cyfr (10000 różnych kombinacji czterech cyfr). Na pewno jednak istnieją ciągi krótsze. Ile cyfr ma ciąg najkrótszy o tej własności, że występują w nim wszystkie możliwe czwórki cyfr? Powtórzymy znane już nam rozumowanie. Pierwsze trzy wykręcone cyfry nie dają nam jeszcze żadnej czwórki cyfr. Dokręcenie każdej kolejnej cyfry da nam co najwyżej jedną nową czwórkę. Najkrótszy ciąg otrzymamy, jeśli każda nowa cyfra da nam nową czwórkę. Tych czwórek jest 10000, więc musimy dokręcić też 10000. Łącznie z trzema początkowymi cyframi daje to 10003 cyfry. Tyle co najmniej cyfr musi mieć poszukiwany ciąg. Ale czy taki ciąg istnieje? Być może najkrótszy taki ciąg jest jednak dłuższy? Tu żadne sanskryckie słowo nam nie pomoże.

Zastąpmy w słowie yamátárájabhánasalagám każdą krótką sylabę zerem i długą jedynką. Otrzymamy ciąg 0111010001. Widzimy, że w tym ciągu każda trójka cyfr 0 i 1 występuje dokładnie jeden raz. Mamy zatem dwa przypadki szczególne tego samego problemu. Danych jest m symboli i dana jest liczba naturalna n. Szukamy najkrótszego ciągu złożonego z tych symboli, w którym występuje każdy ciąg złożony z n symboli. Interesuje nas przy tym, czy istnieje taki ciąg, w którym każdy ciąg n symboli występuje dokładnie jeden raz. Takie ciągi nazywamy ciągami de Bruijna. Podobnie jak wyżej pokazuje się, że wtedy długość takiego ciągu de Bruijna wyniesie $m^n + n - 1$. Przykład sanskrycki daje nam ciąg de Bruijna dla $m = 2$ i $n = 3$. Wtedy jego długość wynosi $2^3 + 3 - 1 = 10$. Dla Baltimore Hilton Inn, gdzie $m = 10$, $n = 4$, długość ciągu de Bruijna (o ile taki istnieje) wyniesie $10^4 + 4 - 1 = 10003$.

Zauważamy następnie, że dwie ostatnie cyfry ciągu de Bruijna 0111010001 są takie same, jak dwie pierwsze cyfry tego ciągu. Jest to zjawisko ogólne. Można pokazać, że w ciągu de Bruijna dla dowolnych liczb m i n pierwsze $n - 1$ i ostatnie $n - 1$ cyfr są

takie same. Dla dowodu oznaczmy ciąg pierwszych $n-1$ cyfr ciągu de Bruijna przez B_1, a ciąg ostatnich $n-1$ cyfr przez B_2. Zauważmy następnie, że każdy ciąg B złożony z $n-1$ cyfr, różny od B_1 i B_2 występuje w ciągu de Bruijna dokładnie m razy. Mianowicie po każdym wystąpieniu ciągu B pojawia się jeden z m symboli. Jeżeli choć jeden symbol powtórzy się dwa razy, to znaczy, że ten sam ciąg n-elementowy wystąpił dwukrotnie. Jeżeli ciągi B_1 i B_2 są różne, to w podobny sposób pokazujemy, że każdy z nich wystąpił tylko m razy (dla ciągu B_1 rozumujemy tak samo, dla ciągu B_2 patrzymy na symbole występujące przed B_2). Ciągów $n-1$ wyrazowych jest m^{n-1}, każdy z nich występuje w ciągu de Bruijna m razy, skąd wynika, że w całym ciągu de Bruijna występuje tylko m^n ciągów długości $n-1$. Ale to oznacza, że długość ciągu de Bruijna wynosi $m^n+(n-1)-1$, co jest niemożliwe. Zatem $B_1=B_2$ i ten ciąg występuje $m+1$ razy.



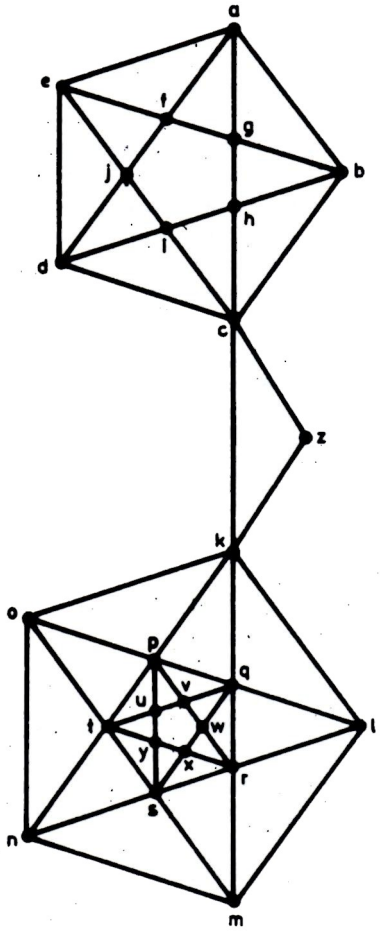
Rys. 1

Ciągi de Bruijna możemy zatem traktować jako ciągi cykliczne długości m^n. Zakładamy, że na końcu takiego ciągu powtórzy się pierwszych $n-1$ wyrazów. W takiej postaci ciągi de Bruijna mają szereg zastosowań. W jednym z nich służą do rozpoznawania położenia walca mogącego przyjmować 2^n różnych położeń. Umieszczamy na obwodzie walca 2^n cyfr 0 lub 1 tworzących cykliczny ciąg de Bruijna oraz instalujemy n czytników odczytujących kolejne cyfry na walcu. Każdemu z 2^n różnych położeń odpowiada inny ciąg zero-jedynkowy rozpoznany przez czytniki (patrz rysunek 1). Ciągi de Bruijna mają też zastosowanie w tworzeniu tzw. kodów korygujących błędy w teorii komunikacji.

Będziemy teraz zmierzać do pokazania, że dla dowolnych liczb m i n istnieją ciągi de Bruijna. W dowodzie wykorzystamy twierdzenia teorii grafów dotyczące tzw. grafów eulerowskich. Zajmiemy się więc teraz takimi grafami.

2. Grafy eulerowskie

Grafem będziemy nazywać dowolny skończony zbiór punktów wraz z pewnymi kreskami łączącymi te punkty. Ściślej, rozpatrujemy skończony zbiór G nazywany zbiorem wierzchołków grafu oraz zbiór E krawędzi tego grafu. Krawędź w grafie ma łączyć dwa wierzchołki, możemy przy tym rozważać krawędzie niezorientowane lub zorientowane. Krawędź zorientowana ma początek i koniec, jest to jakby ulica jednokierunkowa prowadząca od jednego wierzchołka do drugiego. Krawędź niezorientowana jest to jakby ulica dwukierunkowa łącząca dwa wierzchołki. Krawędź zorientowną o początku w wierzchołku x i końcu w wierzchołku y można więc utożsamić z parą uporządkowaną $\langle x,y\rangle$. Krawędź niezorientowaną łączącą wierzchołki x i y możemy utożsamić z parą nieuporządkowaną $\{x,y\}$. W dalszym ciągu będziemy rozpatrywać dwa rodzaje grafów: grafy niezorientowane, których wszystkie krawędzie są niezorientowane i grafy zorientowane, których wszystkie krawędzie są zorientowane. Oczywiście można rozpatrywać też tzw. grafy mieszane, w których występują oba rodzaje krawędzi. Takie grafy jednak nie wystąpią w rozważanych zastosowaniach i dlatego nie będziemy się nimi zajmować. Nie będziemy natomiast nakładać żadnych dalszych ograniczeń na rozpatrywane grafy, poza warunkiem skończoności zbioru krawędzi. Na przykład dopuścimy grafy, w których istnieje wiele krawędzi łączących dwa wierzchołki, tzw. krawędzie wielokrotne oraz grafy, w których istnieją krawędzie łączące dany wierzchołek z sobą samym, tzw. pętle. Będziemy rozpatrywać drogi w grafach. Droga łącząca dwa wierzchołki x i y jest to ciąg krawędzi o następujących własnościach:
- x jest początkiem pierwszej krawędzi,
- y jest końcem ostatniej krawędzi,
- koniec każdej krawędzi, z wyjątkiem ostatniej jest początkiem następnej krawędzi.

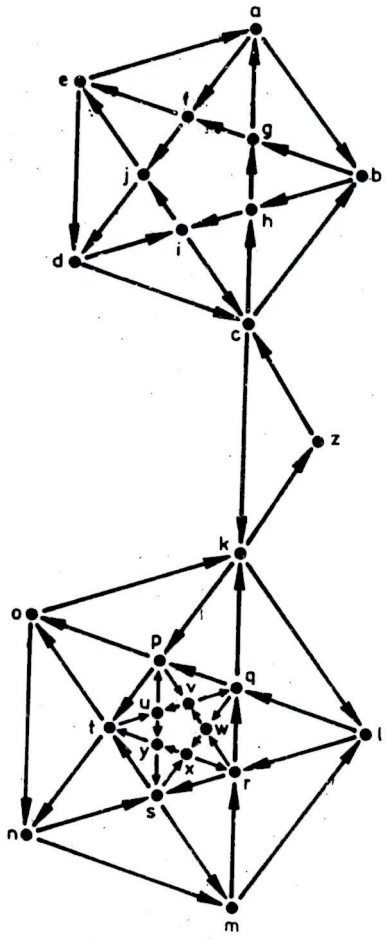


Rys. 2

Graf nazywamy spójnym, jeśli dla każdego wierzchołka istnieją drogi łączące go ze wszystkimi innymi wierzchołkami.

Jeśli droga łączy jakiś wierzchołek z sobą samym, to nazywamy ją cyklem. Cykl jest eulerowski, jeśli przechodzi przez wszystkie krawędzie grafu, przy czym przez każdą krawędź przechodzi dokładnie jeden raz. Naszym celem będzie zbadanie, w jakich grafach istnieją cykle eulerowskie i jak takie cykle znaleźć.

W grafie niezorientowanym stopniem wierzchołka x nazwiemy liczbę krawędzi wychodzących (lub wchodzących, bo to jest tym samym) z tego wierzchołka. Popatrzmy na graf G_1 przedstawiony na rysunku 2. W tym grafie stopień wierzchołków a, f, czy v wynosi 4, stopień wierzchołków k, p, czy t wynosi 6 i wreszcie stopień wierzchołka z wynosi 2.



Rys. 3

W grafie zorientowanym rozpatrujemy dwa rodzaje stopni: stopień wejściowy równy liczbie krawędzi wchodzących do danego wierzchołka i stopień wyjściowy równy liczbie krawędzi wychodzących z tego wierzchołka. Dla przykładu, w grafie G_2 przedstawionym na rysunku 3 stopień wyjściowy wierzchołka a wynosi 2, stopień wyjściowy wierzchołka p wynosi 3, a stopień wyjściowy wierzchołka z wynosi 1. Stopnie wejściowe tych wierzchołków są równe ich stopniom wyjściowym.

Możemy teraz udowodnić twierdzenie mówiące o istnieniu cykli eulerowskich w grafie. Twierdzenie to pochodzi od Eulera.

Twierdzenie.

Warunkiem koniecznym i wystarczającym dla istnienia w spójnym grafie niezorientowanym cyklu eulerowskiego jest, by stopień każdego wierzchołka był parzysty.

Dla spójnego grafu zorientowanego warunkiem tym jest, by stopień wejściowy każdego wierzchołka był równy stopniowi wyjściowemu tego wierzchołka.

Dowód.
Prowadzimy dowód niezależnie od tego, czy rozpatrywany graf jest zorientowany, czy nie. Dowód okaże się być taki sam w obu przypadkach.

Przypuśćmy najpierw, że w grafie G istnieje cykl eulerowski. Popatrzmy na dowolny wierzchołek w tym grafie. Przez ten wierzchołek przechodzi cykl eulerowski, być może nawet wiele razy. Jednak za każdym razem, gdy poruszając się wzłuż tego cyklu dochodzimy do naszego wierzchołka, to zaraz musimy go opuścić inną krawędzią. Zatem liczba krawędzi, którymi weszliśmy do tego wierzchołka jest równa liczbie krawędzi, którymi z niego wyszliśmy.

Przypuśćmy teraz, że dany jest graf G spełniający warunek sformułowany w twierdzeniu. Będziemy dowodzić, że w tym grafie istnieje cykl eulerowski. Dowód prowadzimy przez indukcję ze względu na liczbę krawędzi grafu. Graf nie mający krawędzi i spójny jest jednym punktem i ma oczywiście cykl eulerowski, mianowicie cykl pusty. Przypuśćmy teraz, że dla grafów mających mniej krawędzi niż graf G twierdzenie jest prawdziwe.

Obierzmy w grafie G dowolny wierzchołek x i weźmy dowolny cykl zaczynający się i kończący w wierzchołku x. Taki cykl oczywiście istnieje. Możemy go znaleźć w następujący sposób. Bierzemy dowolną krawędź wychodzącą z x i prowadzącą do wierzchołka y. Jeżeli $y \neq x$, to bierzemy dowolną nową krawędź z y do wierzchołka z. Jeżeli $z \neq x$, to bierzemy następną nową krawędź. Na mocy założonego warunku, jeśli wejdziemy do wierzchołka różnego od x, to musimy mieć też drogę wyprowadzającą nas z niego. Ponieważ graf ma skończenie wiele krawędzi, więc tego postępowania nie możemy kontynuować w nieskończoność. Zakończyć się ono jednak może dopiero po powrocie do wierzchołka x i otrzymaniu w ten sposób szukanego cyklu.

Usuńmy z grafu wszystkie krawędzie tak wybranego cyklu C. Graf rozpadnie się być może na pewną liczbę mniejszych grafów spójnych, nazywanych składowymi uzupełnienia cyklu C w grafie G. Każdy z nich nadal spełnia warunek sformułowany w twierdzeniu. Mianowicie dla każdego wierzchołka y usunęliśmy z grafu G tyle samo krawędzi wchodzących do y, co wychodzących z y. Cykl C bowiem wchodzi do y tyle samo razy, ile z y wychodzi.

Otrzymane mniejsze grafy mają zatem cykle eulerowskie. Teraz konstruujemy cykl eulerowski w grafie G. Rozpoczynamy podróż po grafie w wierzchołku x. Poruszamy się wzdłuż wybranego cyklu C. W każdym momencie, gdy dojdziemy do którejś za składowych uzupełnienia C, przebiegamy tę składową wzdłuż istniejącego w niej cyklu eulerowskiego, powracając do cyklu C w tym samym miejscu. Kontynuujemy teraz naszą podróż wzdłuż C aż do napotkania następnej składowej, której nie obeszliśmy. Po zakończeniu w ten sposób podróży wzdłuż cyklu C, przerywanej od czasu do czasu wycieczkami w bok, po kolejnych składowych uzupełnienia C, dojdziemy do wierzchołka x. Z założenia indukcyjnego wynika, że obejdziemy wszystkie krawędzie każdej składowej uzupełnienia C oraz obejdziemy sam cykl C. Zatem obejdziemy wszystkie krawędzie grafu G, przy czym każdą dokładnie jeden raz. W ten sposób twierdzenie jest udowodnione.

Grafy, w których istnieją cykle eulerowskie nazywamy grafami eulerowskimi. Oczywiście możemy teraz zapytać, w jaki sposób znaleźć cykl eulerowski w grafie eulerowskim. Dowód twierdzenia daje nam pewien algorytm, zdefiniowany w sposób rekurencyjny. Spróbujemy zapisać ten algorytm w sposób precyzyjny. Musimy jednak przedtem zastanowić się nad sposobem reprezentacji grafu. Jest to szczególnie ważne wtedy, gdy chcemy napisać program komputerowy znajdujący cykl eulerowski.

Musimy przecież powiedzieć komputerowi jak wygląda graf, w którym ma on tego cyklu szukać. Przedstawienie graficzne oczywiście nie będzie odpowiednie dla komputera.

Na ogół grafy opisuje się podając zbiór wierzchołków (w przypadku grafów G_1 i G_2 przedstawionych na rysunkach 2 i 3 jest to zbiór liter alfabetu łacińskiego od a do z) oraz podając dla każdego wierzchołka listę wierzchołków, do których prowadzą krawędzie wychodzące z tego wierzchołka. Kolejność wierzchołków na tych listach jest dowolna. W przedstawionym niżej programie komputerowym grafy te zostały przedstawione w postaci tablicy G indeksowanej literami od a do z, w której przechowujemy słowa składające się z liter oznaczających wierzchołki połączone z danym. Tak na przykład w przypadku grafu G_1 element $G['c']$ tablicy G jest słowem *'bdhikz'*, co oznacza, że wierzchołek c jest połączony krawędziami z wierzchołkami b, d, h, i, k oraz z. Zauważmy, że w przypadku grafów niezorientowanych ta sama krawędź od wierzchołka x do wierzchołka y występuje w tej reprezentacji dwukrotnie, raz w słowie $G[x]$, drugi raz w słowie $G[y]$. W poniższych programach komputerowych tablica G jest wypełniana za pomocą procedury *ZapoczatkowanieGrafu*.

Programy te zawierają również pomocniczą procedurę *UsunKrawedz* usuwającą z grafu G krawędź o początku w wierzchołku x i końcu w wierzchołku y. Procedura ta będzie używana podczas generowania cyklu eulerowskiego w grafie G.

Sam cykl eulerowski C jest tworzony za pomocą procedury *CyklEulera1*. Procedura ta jest zdefiniowana rekurencyjnie i będzie korzystała sama z siebie. Formułujemy ją więc w sposób ogólny. Procedura ta dopisuje do przebytego fragmentu cyklu C cykl eulerowski wypełniający składową grafu G zawierającą wierzchołek x. Procedura ta w programie głównym zostanie wywołana dla całego grafu G (który jest z założenia spójny) i pustego fragmentu cyklu C. Efektem działania będzie cykl eulerowski wypełniający cały graf G, rozpoczynający się w wybranym wierzchołku. W naszym przypadku tym wierzchołkiem będzie a.

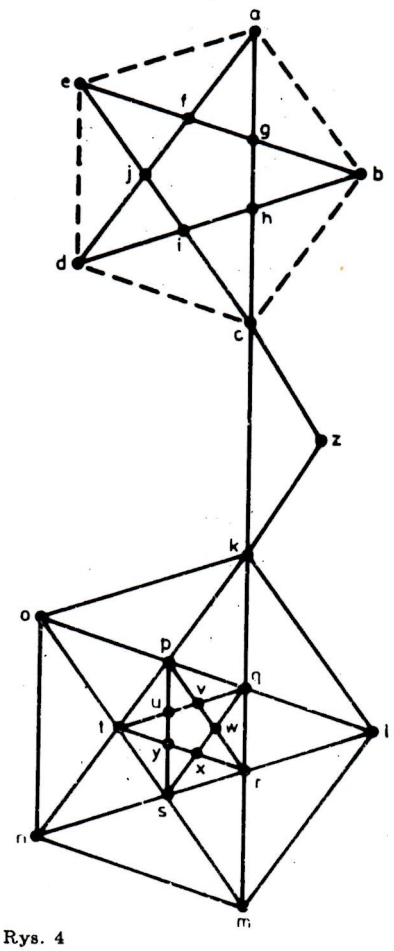
Teraz zastanówmy się nad czynnościami wykonywanymi przez tę procedurę. Jeżeli składowa zawierająca wierzchołek x jest pusta, to oczywiście nic nie robimy. Przypuśćmy teraz, że składowa zawierająca x jest niepusta. Wtedy najpierw znajdujemy cykl zaczynający się i kończący w wierzchołku x. Cykl ten zostanie zapamiętany w zmiennej *Pomoc*. Następnie wypisujemy ten cykl. W ten sposób będziemy mogli lepiej prześledzić działanie programu. Zauważymy też, że jednym z efektów działania programu będzie rozłożenie całego grafu G na sumę rozłącznych cykli.

W trakcie tworzenia cyklu *Pomoc* wszystkie jego krawędzie są usuwane z grafu G, co spowoduje ewentualny rozpad grafu na składowe. Teraz przebiegamy cykl *Pomoc* drugi raz. Każdą krawędź dopisujemy do cyklu C i od razu wywołujemy procedurę *CyklEulera1* dla wierzchołka końcowego tej krawędzi. Oczywiście w tym nowym wywołaniu procedury będziemy mieli do czynienia ze zmodyfikowanym grafem G, być może składającym się z wielu rozłącznych kawałków. Procedura *CyklEulera1* dopisze nam do cyklu C odpowiedni cykl obiegający składową, w której się znaleźliśmy, przed kontynuacją naszej drogi wzłuż cyklu *Pomoc*. Po obiegnięciu całego cyklu *Pomoc* zmienna C będzie zawierała cały cykl eulerowski dla grafu G. Wreszcie program wypisze długość tego cyklu i jego przebieg. Sam program komputerowy, napisany w języku Pascal (w wersji Turbo-Pascal) wygląda następująco:

```
program CykleEulera; {w grafie niezorientowanym}

type Cykl = String[255];
     ListaWierzcholkow = String[6];
     Graf = array ['a'..'z'] of ListaWierzcholkow;
var  G : Graf;
     C : Cykl;

procedure ZapoczatkowanieGrafu (var G : Graf);
    begin G['a']:='befg';   G['b']:='acgh';   G['c']:='bdhikz';
          G['d']:='ceij';   G['e']:='adfj';   G['f']:='aegj';
          G['g']:='abfh';   G['h']:='bcgi';   G['i']:='cdhj';
          G['j']:='defi';   G['k']:='clopqz'; G['l']:='kmqr';
          G['m']:='lnrs';   G['n']:='most';   G['o']:='knpt';
          G['p']:='koqtuv'; G['q']:='klprvw'; G['r']:='lmqswx';
          G['s']:='mnrtxy'; G['t']:='nopsuy'; G['u']:='ptvy';
          G['v']:='pquw';   G['w']:='qrvx';   G['x']:='rswy';
          G['y']:='stux';   G['z']:='ck'
    end; {ZapoczatkowanieGrafu}
```

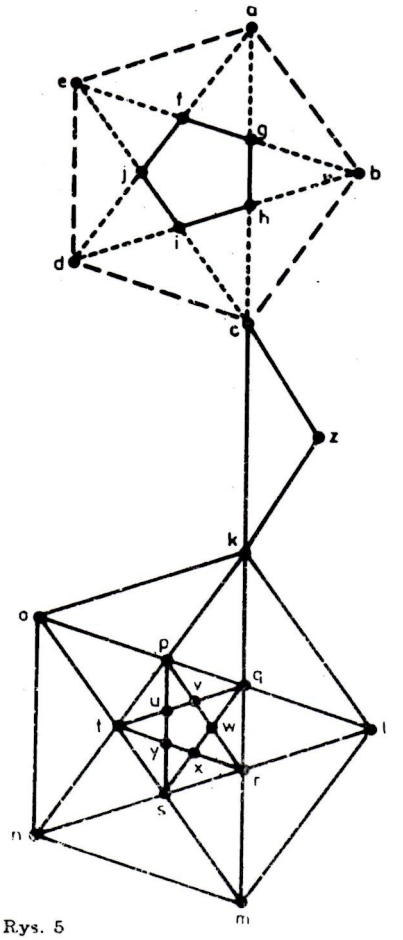


Rys. 4

```
procedure UsunKrawedz (var G : graf; x,y : char);
    var i : integer;
        Koniec : Boolean;
    begin i:=0;
        repeat i:=i+1;
            if i>Length(G[x]) then Koniec:=true
            else Koniec:=(G[x][i]=y)
        until Koniec;
        Delete(G[x],i,1)
    end; UsunKrawedz

procedure CyklEulerai (var G : Graf; var C : Cykl; x : char);
    var Pomoc : Cykl;
        Poczatek,Koniec : char;
        i : integer;
    begin
        if Length(G[x])>0 then begin
            Pomoc:=''; Poczatek:=x;
            repeat
                Koniec:=G[Poczatek][1];
                UsunKrawedz(G,Poczatek,Koniec);
                UsunKrawedz(G,Koniec,Poczatek);
                Pomoc:=Pomoc+Koniec;
                Poczatek:=Koniec
            until (Koniec=x);
            writeln(x,':',Pomoc);
            for i:=1 to Length(Pomoc) do begin
                C:=C+Pomoc[i];
                CyklEulerai(G,C,Pomoc[i]);
            end {for}
        end; {if}
    end; {CyklEulerai}

begin {program}
    ClrScr;
    ZapoczatkowanieGrafu(G);
    C:='';
    CyklEulerai(G,C,'a');
    writeln; writeln(Length(C)); writeln('a:',C)
end.
```



Rys. 5

Efektem działania tego programu będzie następujący wydruk:

a:bcdea
b:gafejdichb
g:fjihg
c:klmnokpotnsmrlqkzc
p:qrstp
q:vputysxrwq
v:uyxwv

58

a:bgfjihgafejdicklmnokpqvuyxwvputysxrwqrstpotnsmrlqkzchbcdea

Zauważamy rozkład grafu na cykle. Pierwszych siedem wierszy podaje nam te cykle. Każdy z nich jest zapisany w postaci: najpierw wierzchołek początkowy, a potem po dwukropku kolejne wierzchołki, przez które przechodzimy w tym cyklu. Dalej mamy długość cyklu i jego przebieg zapisany w taki sam sposób, jak powyżej zapisaliśmy poszczególne cykle. Teraz możemy prześledzić działanie programu. Najpierw został znaleziony cykl wychodzący z wierzchołka a i przechodzący przez wierzchołki *bcdea*. Usunęliśmy ten cykl z grafu (patrz rysunek 4) i rozpoczęliśmy podróżowanie wzdłuż niego. W wierzchołku b (dopisanym do pustego na początku cyklu C) od razu natknęliśmy się na niepustą składową i zaczęliśmy ją przebiegać. Cykl $a : bcdea$ został zapamiętany (komputer robi to automatycznie) i procedura *CyklEulerai* zaczęła poszukiwania nowego cyklu. Został znaleziony cykl $b : gafejdichb$. Usunęliśmy go (patrz rysunek 5), a następnie zaczęliśmy go przebiegać. Jednak znów już w pierwszym kroku (odpowiedni wierzchołek, tym razem g) znaleźliśmy niepustą składową i cykl $g : fjihg$. Zaczęliśmy więc przebiegać ten cykl, oczywiście po uprzednim usunięciu go z grafu. Teraz okazało się, że przebiegniemy go w całości nie napotykając na nowe niepuste składowe. Cały ten cykl będzie więc dopisany do cyklu C, którego początek

wygląda zatem następująco: $a : bgfjihg$. Teraz kontynuujemy drogę wzłuż zapamiętanego drugiego cyklu. Znów przekonamy się, że nie natkniemy się na żaden wierzchołek należący do nowej niepustej składowej do momentu, gdy znajdziemy się w wierzchołku c. Po dopisaniu przebytej drogi do cyklu C otrzymamy $a : bgfjihgafejdic$ i znów zaczniemy poszukiwania nowego cyklu. I tak dalej...

W definicji procedury *CyklEulera1* musimy pamiętać o tym, że każda krawędź występuje w grafie dwukrotnie i usuwać trzeba obydwie. W podobny sposób możemy zdefiniować procedurę *CyklEulera2* tworzącą cykl eulerowski w grafie zorientowanym. Będzie ona krótsza o jedną linijkę: tym razem każdą krawędź wystarczy usunąć tylko jeden raz.

Tak zdefiniowaną procedurę wykorzystamy w programie komputerowym wypisującym cykl eulerowski w grafie G_2 z rysunku 3. Oczywiście zmiany wymaga teraz też procedura zapoczątkowująca graf. Każdy wierzchołek ma tym razem mniej krawędzi niż w poprzednim przypadku.

Program komputerowy, napisany również w Turbo-Pascalu wygląda tym razem następująco:

```
program CykleEulera; w grafie zorientowanym
type Cykl = String[255];
     ListaWierzcholkow = String[6];
     Graf = array ['a'..'z'] of ListaWierzcholkow;
var  G : Graf;
     C : Cykl;

procedure ZapoczatkowanieGrafu (var G : Graf);
    begin G['a']:='bf'; G['b']:='gh'; G['c']:='bhk';
          G['d']:='ci'; G['e']:='ad'; G['f']:='ej';
          G['g']:='af'; G['h']:='gi'; G['i']:='cj';
          G['j']:='de'; G['k']:='lpz'; G['l']:='qr';
          G['m']:='lr'; G['n']:='ms'; G['o']:='kn';
          G['p']:='otv'; G['q']:='kpw'; G['r']:='qsw';
          G['s']:='mtx'; G['t']:='nou'; G['u']:='py';
          G['v']:='qu'; G['w']:='vx'; G['x']:='ry';
          G['y']:='st'; G['z']:='c'
     end; {ZapoczatkowanieGrafu}

procedure UsunKrawedz (var G : graf; x,y : char);
    var i : integer;
        Koniec : Boolean;
    begin i:=0;
        repeat i:=i+1;
            if i>Length(G[x]) then Koniec:=true
            else Koniec:=(G[x][i]=y)
        until Koniec;
        Delete(G[x],i,1)
    end; {UsunKrawedz}

procedure CyklEulera2 (var G : Graf; var C : Cykl; x : char);
    var Pomoc : Cykl;
        Poczatek,Koniec : char;
        i : integer;
    begin
        if Length(G[x])>0 then begin
            Pomoc:=''; Poczatek:=x;
            repeat
                Koniec:=G[Poczatek][1];
                UsunKrawedz(G,Poczatek,Koniec);
                Pomoc:=Pomoc+Koniec;
                Poczatek:=Koniec
            until (Koniec=x);
            writeln(x,':',Pomoc);
            for i:=1 to Length(Pomoc) do begin
                C:=C+Pomoc[i];
                CyklEulera2(G,C,Pomoc[i])
            end {for}
        end; {if}
    end; {CyklEulera2}
```

```
begin {program}
    ClrScr;
    ZapoczatkowanieGrafu(G);
    C:='';
    CyklEulera2(G,C,'a');
    writeln; writeln(Length(C)); writeln('a:',C)
end.
```

Efektem działania będzie tym razem następujący rozkład grafu na rozłączne cykle i ostateczny cykl eulerowski C:

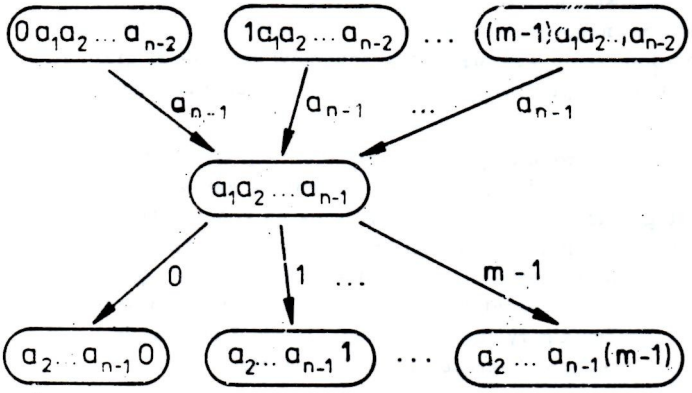
a:bga
b:hgfeafjdcb
h:ich
i:jedi
c:klqkpokzc
l:rqptnml
r:smr
s:tons
t:upvqwvuysxrwxyt

58
a:bhijedicklrstupvqwvuysxrwxytonsmrqptnmlqkpokzchgfeafjdcbga

Przeanalizowanie działania programu pozostawię tym razem Czytelnikowi. Zajmiemy się natomiast zastosowaniem grafów eulerowskich do problemu istnienia ciągów de Bruijna.

3. Grafy eulerowskie i ciągi de Bruijna

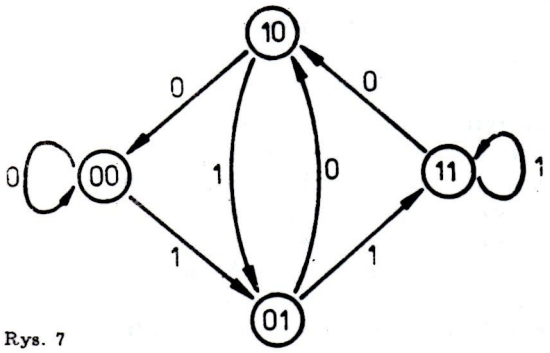
Chcemy skonstruować ciąg de Bruijna, którego wyrazy są dowolnymi spośród m symboli i taki, że każdy ciąg długości n wystąpi w tym ciągu de Bruijna dokładnie jeden raz. Będziemy tworzyli ciąg cykliczny. Założymy, że naszymi symbolami są liczby naturalne od 0 do $m-1$.



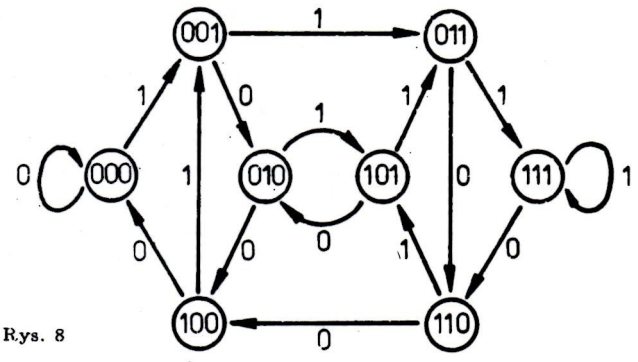
Rys. 6

Utworzymy graf, którego wierzchołkami są wszystkie ciągi długości $n-1$ o wyrazach ze zbioru $\{0,\ldots,m-1\}$. Takich wierzchołków jest więc m^{n-1}. Każdy wierzchołek będzie początkiem m krawędzi, numerowanych liczbami od 0 do $m-1$. Krawędź o numerze i wychodząca z wierzchołka $a_1a_2\ldots a_{n-1}$ prowadzi do wierzchołka $a_2\ldots a_{n-1}i$. Przejście tej krawędzi będziemy traktować jak dopisanie liczby i do ciągu, którego ostatnimi $n-1$ wyrazami są liczby $a_1, a_2, \ldots, a_{n-1}$. Zauważamy następnie, że każdy wierzchołek jest też końcem dokładnie m krawędzi. Wierzchołek $a_1a_2\ldots a_{n-1}$ jest końcem krawędzi wychodzących z wierzchołków $0a_1a_2\ldots a_{n-2}$, $1a_1a_2\ldots a_{n-2}$, $\ldots$, $(m-1)a_1a_2\ldots a_{n-2}$. Wszystkie te krawędzie są numerowane liczbą a_{n-1} (por. rysunek 6). Tak skonstruowany graf, nazywany dalej grafem de Bruijna, spełnia więc warunek sformułowany w twierdzeniu Eulera. Każdy wierzchołek ma bowiem stopień wejściowy i stopień wyjściowy równy m. Teraz wystarczy zauważyć, że wszystkich krawędzi jest w tym grafie dokładnie m^n oraz że numery krawędzi dowolnego cyklu eulerowskiego tworzą ciąg de Bruijna.

Na rysunkach 7 i 8 możemy obejrzeć grafy de Bruijna skonstruowane dla ciągów de Bruijna dla $m=2$ oraz odpowiednio $n=3$ (czyli dla yamátárájabhánasalagám) i $n=4$.



Rys. 7



Rys. 8

Znamy więc już rozwiązanie problemu istnienia ciągów de Bruijna dla dowolnych m i n. Znamy też jeden algorytm otrzymywania takich ciągów. Mianowicie tworzymy odpowiedni graf de Bruijna i następnie stosujemy znany już algorytm do otrzymania cyklu eulerowskiego, czyli tym samym do otrzymania ciągu de Bruijna. Niestety, algorytm ten jest bardzo nieefektywny. Wymaga on bowiem nie tylko zapisania w pamięci komputera całego grafu, który już dla niezbyt dużych m i n ma bardzo wiele wierzchołków, ale również zapamiętania tworzonych w czasie działania algorytmu cykli pomocniczych. Bardzo szybko może nastąpić więc przepełnienie pamięci komputera i działanie programu zostanie przerwane.

Będziemy zatem w dalszym ciągu zajmować się poszukiwaniem algorytmów bardziej efektywnych.

4. Rozgałęzienia

Każdy graf ma na ogół wiele różnych reprezentacji, różniących się porządkiem krawędzi wychodzących z wierzchołków. W powyższych programach grafy zapoczątkowywaliśmy tak, by dla każdego wierzchołka lista wierzchołków, do których z niego prowadzą krawędzie była uporządkowana alfabetycznie. Okazuje się, że zmiana kolejności tych wierzchołków może prowadzić do reprezentacji, dla której bardzo łatwo podać algorytm znajdowania cyklu eulerowskiego. Do opisu takich reprezentacji grafu służą tzw. rozgałęzienia. Opiszemy je tylko dla grafów zorientowanych.

Dla danego grafu zorientowanego G rozgałęzieniem o korzeniu a nazwiemy taki graf zorientowany R, który zawiera wszystkie wierzchołki grafu G, którego krawędzie są krawędziami grafu G (czyli R jest podgrafem grafu G) oraz taki, że:
- z każdego wierzchołka grafu R, różnego od a wychodzi dokładnie jedna krawędź, przy czym prowadzi ona do wierzchołka różnego od a,
- do każdego wierzchołka może prowadzić dowolnie wiele krawędzi,
- z wierzchołka a nie wychodzi żadna krawędź lub wychodzi tylko pętla do samego a.

Mając dany graf G bardzo łatwo znajdujemy pewne jego rozgałęzienie. Na początku do rozgałęzienia R zaliczamy wierzchołek a i ewentualną pętlę z a do a. Następnie wielokrotnie przeszukujemy graf G za każdym razem dołączając do grafu R jeden nowy wierzchołek i jedną krawędź. Dołączamy mianowicie pierwszy znaleziony wierzchołek, który nie został jeszcze zaliczony do R i z którego prowadzi przynajmniej jedna krawędź do pewnego wierzchołka już znajdującego się w grafie R. To postępowanie kończymy po dołączeniu wszystkich wierzchołków grafu G do grafu R. W zamieszczonym dalej programie komputerowym ten algorytm jest zrealizowany w procedurze *Rozgalezienie*.

Mając dane rozgałęzienie R grafu G dokonujemy zmiany reprezentacji G. Mianowicie dla każdego wierzchołka x grafu G, z wyjątkiem być może korzenia a, zamieniamy porządek listy „następników" tego wierzchołka w taki sposób, by na końcu listy znalazł się wierzchołek, do którego w rozgałęzieniu R prowadzi krawędź wychodząca z x. Taką reprezentację grafu G nazwiemy reprezentacją eulerowską. Oczywiście algorytm przejścia od dowolnej reprezentacji do reprezentacji eulerowskiej jest trywialny, o ile jest dane rozgałęzienie R grafu G. Ten algorytm jest w podanym dalej programie komputerowym zrealizowany w procedurze *ReprezentacjaEulerowska*.

Przypuśćmy teraz, że dany jest graf G wraz ze swoją reprezentacją eulerowską daną przez rozgałęzienie R o korzeniu a. Opiszemy algorytm przechodzenia krawędzi tego grafu. Zaczynamy od korzenia rozgałezienia. W każdym momencie, będąc w dowolnym wierzchołku x grafu G wybieramy krawędź prowadzącą do wierzchołka będącego na pierwszym miejscu na liście wierzchołków połączonych krawędzią z wierzchołkiem x. Przebytą krawędź usuwamy z grafu G. Zauważamy, że krawędź wychodząca z wierzchołka x w rozgałęzieniu R jest przechodzona w ostatniej kolejności; dopóki z wierzchołka x wychodzą inne krawędzie, wybieramy najpierw jedną z nich. Kolejność przechodzenia krawędzi wychodzących z dowolnego wierzchołka jest więc dowolna, z tym, że krawędź należąca do rozgałęzienia R jest przebywana na końcu. Przechodzenie grafu kończymy wtedy, gdy dojdziemy do wierzchołka, z którego nie wychodzi już żadna krawędź. Okaże się, że jest nim korzeń rozgałęzienia.

Czytelnik z łatwością zauważy, że powyższy algorytm przechodzenia grafu eulerowskiego wykorzystujący jego reprezentację eulerowską daje w wyniku cykl eulerowski. Dowód jest oparty na następujących kilku spostrzeżeniach. Po pierwsze, cykl rozpoczynający się w korzeniu a musi zakończyć się też w tym korzeniu. Wynika to stąd, że do każdego wierzchołka wchodzi tyle samo krawędzi, ile wychodzi z niego. Po drugie, jeśli jakaś krawędź grafu G nie została przebyta, to również nie została przebyta pewna krawędź rozgałęzienia R. Mianowicie krawędź należą do rozgałęzienia przechodzimy na końcu. Po trzecie, jeśli nie została przebyta krawędź

rozgałęzienia dochodząca do wierzchołka x, to również nie została przebyta krawędź rozgałęzienia wychodząca z x, o ile oczywiście x nie jest korzeniem rozgałęzienia. Wynika-to stąd, że jeśli nie została przebyta pewna gałąź dochodząca do wierzchołka x, to również nie może być przebyta pewna gałąź wychodząca z x, a zatem i gałąź należąca do rozgałęzienia. Wreszcie, po czwarte, wynika stąd, że nie została przebyta pewna gałąź wychodząca z korzenia. Oznacza to jednak, że możemy kontynuować naszą wędrówkę po grafie, co przeczy temu, że nasz algorytm zakończył działanie.

W poniższym programie opisany wyżej algorytm przechodzenia grafu jest zrealizowany w procedurze *CyklEulera3*. A oto sam program komputerowy:

```
program CykleEulera; {w grafie zorientowanym}
type Cykl = String[255];
     ListaWierzcholkow = String[6];
     Graf = array ['a'..'z'] of ListaWierzcholkow;
var  G,H,R : Graf;
     C : Cykl;

procedure ZapoczatkowanieGrafu (var G : Graf);
    begin G['a']:='bf';  G['b']:='gh';  G['c']:='bhk';
          G['d']:='ci';  G['e']:='ad';  G['f']:='ej';
          G['g']:='af';  G['h']:='gi';  G['i']:='cj';
          G['j']:='de';  G['k']:='lpz'; G['l']:='qr';
          G['m']:='lr';  G['n']:='ms';  G['o']:='kn';
          G['p']:='otv'; G['q']:='kpw'; G['r']:='qsw';
          G['s']:='mtx'; G['t']:='nou'; G['u']:='py';
          G['v']:='qu';  G['w']:='vx';  G['x']:='ry';
          G['y']:='st';  G['z']:='c'
     end; {ZapoczatkowanieGrafu}

procedure WypiszGrafy (G,R;H : Graf);
    var x : char;
    begin
        for x:='a' to 'z' do
            writeln(x,':',G[x]:8,R[x]:8,H[x]:8)
    end; {WypiszGraf}

procedure UsunKrawedz (var G : graf; x,y : char);
    var i : integer;
        Stop : Boolean;
    begin i:=0;
        repeat i:=i+1;
            if i>Length(G[x]) then Stop:=true
            else Stop:=(G[x][i]=y)
        until Stop;
        Delete(G[x],i,1)
    end; UsunKrawedz

procedure Rozgalezienie (G : Graf; var R : Graf);
    var Dolaczony,Stop : Boolean;
        x,y : char;
        i : integer;
    begin for x:='b' to 'z' do R[x]:=''; R['a']:='a';
        repeat Stop:=true; x:='a';
            repeat x:=succ(x);
                Dolaczony:=false;
                if Length(R[x])=0 then begin
                    Stop:=false; i:=0;
                    repeat i:=i+1;
                        y:=G[x][i]
                    until (i=Length(G[x])) or (Length(R[y])=1);
                    if Length(R[y])=1 then begin
                        Dolaczony:=true;
                        R[x]:=y
                    end {if}
                end {if}
            until Dolaczony or (x='z')
        until Stop
    end; {Rozgalezienie}
```

```
procedure ReprezentacjaEulerowska (var G,R,H : Graf);
    var x : char;
    begin H:=G;
        for x:='b' to 'z' do begin
            UsunKrawedz(H,x,R[x]);
            H[x]:=H[x]+R[x]
        end {for}
    end; {ReprezentacjaEulerowska}

procedure CyklEulera3 (var G : Graf; var C : Cykl; x : char);
    var Poczatek,Koniec : char;
    begin C:=''; Poczatek:=x;
        repeat
            Koniec:=G[Poczatek][1];
            UsunKrawedz(G,Poczatek,Koniec);
            C:=C+Koniec;
            Poczatek:=Koniec
        until (Koniec=x) and (Length(G[x])=0)
    end; {CyklEulera3}

begin {program}
    ClrScr;
    ZapoczatkowanieGrafu(G);
    Rozgalezienie(G,R);
    ReprezentacjaEulerowska(G,R,H);
    WypiszGrafy(G,R,H);
    CyklEulera3(H,C,'a');
    writeln; writeln(Length(C)); writeln('a:',C)
end.
```

Powyższy program tworzy reprezentację eulerowską znanego już nam grafu G_2, następnie wypisuje graf G, utworzone rozgałęzienie R oraz zrobioną z jego pomocą reprezentację eulerowską. Na końcu wypisuje długość cyklu eulerowskiego i jego przebieg. Wynik działania tego programu jest następujący:

a:	bf	a	bf
b:	gh	g	hg
c:	bhk	b	hkb
d:	ci	c	ic
e:	ad	a	da
f:	ej	e	je
g:	af	a	fa
h:	gi	g	ig
i:	cj	c	jc
j:	de	d	ed
k:	lpz	z	lpz
l:	qr	q	rq
m:	lr	l	rl
n:	ms	m	sm
o:	kn	k	nk
p:	otv	o	tvo
q:	kpw	k	pwk
r:	qsw	q	swq
s:	mtx	m	txm
t:	nou	n	oun
u:	py	p	yp
v:	qu	q	uq
w:	vx	v	xv
x:	ry	r	yr
y:	st	s	ts
z:	c	c	c

58
a:bhijedichgfjdcklrstonsxytuysmrwxrqptnmlqwvupvqkpokzcbgafea

W pierwszej kolumnie są wypisane wierzchołki grafu G_2, w drugiej znana już nam reprezentacja tego grafu, w trzeciej reprezentacja rozgałęzienia i w czwartej reprezentacja eulerowska grafu G_2. Zauważmy też, że otrzymaliśmy inny cykl eulerowski niż za pomocą programu poprzedniego.

Metoda generowania cykli eulerowskich oparta na pojęciu rozgałęzienia może być łatwo wykorzystana do generowania ciągów de Bruijna. Mianowicie zauważamy, że w opisanym poprzednio grafie de Bruijna bardzo łatwo możemy wskazać pewne rozgałęzienie. Wystarczy bowiem zauważyć, że wszystkie krawędzie numerowane liczbą 0 tworzą rozgałęzienie, w którym korzeń jest połączony pętlą ze sobą samym. Otrzymujemy zatem bardzo prosty algorytm generowania ciągów de Bruijna. Rozpoczynamy od ciągu $n-1$ zer. Następnie w każdym momencie patrzymy na ostatnie $n-1$ wyrazów otrzymanego ciągu i dopisujemy największą możliwą liczbę, której jeszcze do tego ciągu nie dopisywaliśmy. Postępowanie to kończymy, gdy powrócimy do ciągu $n-1$ zer i będziemy musieli dopisać również zero.

W praktyce ten algorytm możemy zrealizować w taki sposób, że dla każdego $(n-1)$-wyrazowego ciągu zapamiętujemy największą liczbę, którą wolno nam dopisać. Oczywiście na początku wszystkie te liczby będą równe $m-1$. Następnie rozpoczynamy przechodzenie grafu de Bruijna w korzeniu rozgałęzienia, czyli w ciągu $n-1$ zer. Za każdym razem będąc w kolejnym wierzchołku grafu patrzymy na odpowiadającą temu wierzchołkowi liczbę, wypisujemy ją i zmniejszamy ją o 1. Postępowanie to kończymy wtedy, gdy liczbą odpowiadającą ciągowi $n-1$ zer będzie -1, co oznacza, że za tym ciągiem już dopisaliśmy zero. Algorytm ten jest zrealizowany w następującym programie komputerowym:

```
program CiagiDeBruijna;

const m = 3;
      m1 = 2; m-1
type  Symbole = -1..m1;
      Graf = array [0..m1,0..m1,0..m1] of Symbole;
var   G : Graf;

procedure ZapoczatkowanieGrafu (var G : Graf);
    var i,j,k : Symbole;
    begin
        for i:=0 to m1 do
            for j:=0 to m1 do
                for k:=0 to m1 do
                    G[i,j,k]:=m1
    end; ZapoczatkowanieGrafu

procedure deBruijn (G : Graf);
    var c1,c2,c3,c : Symbole;
    begin
        c1:=0; c2:=0; c3:=0;
        repeat
            c:=G[c1,c2,c3]; write(c);
            G[c1,c2,c3]:=c-1;
            c1:=c2; c2:=c3; c3:=c
        until (G[0,0,0]=-1)
    end; {deBruijn}

begin {program}
    ClrScr;
    ZapoczatkowanieGrafu(G);
    deBruijn(G)
end.
```

Efektem działania tego programu będzie ciąg de Bruijna dla $m=3$ oraz $n=4$. Ten 81-cyfrowy ciąg wygląda następująco:

222122202211221022012200212120211121102101210020201120102001200011110110010100 00.

5. Algorytmy oszczędzające pamięć

Opisany powyżej algorytm generowania ciągów de Bruijna wymaga też bardzo dużej pamięci komputera. Mianowicie musimy zapamiętać największą możliwą do dopisania liczbę dla każdego ciągu $(n-1)$-elementowego. Wymaga to zatem zapamiętania co najmniej m^{n-1} liczb naturalnych. Pamięć potrzebna do zrealizowania tego algorytmu rośnie więc wykładniczo wraz ze wzrostem n. Powstaje naturalne pytanie, czy istnieje algorytm generujący ciągi de Bruijna wymagający mniej pamięci.

Jeden taki algorytm został podany przez A. Ralstona. Jest on zbyt skomplikowany, by go tu przytoczyć w całości, zadowolimy się tylko szkicem jego działania i obejrzymy jego działanie na jednym wybranym przykładzie. Algorytm ten przedstawia się mniej więcej następująco:

1. Wypisz pojedynczą liczbę $m-1$.
2. Dopisz $m-1$ ciągów n-elementowych począwszy od ciągu $m-1, m-1, \ldots,$ $m-1, m-2$ aż do $m-1, m-1, \ldots, m-1, 0$.
3. Dopisz $(m-1)^2$ ciągów n-elementowych począwszy od ciągu $m-1, m-1, \ldots,$ $m-1, m-2, m-2$ aż do $m-1, m-1, \ldots, m-1, 0, 0$.
4. Kontynuuj dopisując kolejne ciągi n-elementowe mające coraz mniej rozmaicie rozmieszczonych liczb $m-1$.
5. Następnie dopisz ciąg de Bruijna skonstruowany według tego samego algorytmu dla $m-1$ i tego samego n.

Punkt 5 tego algorytmu pokazuje, że jego definicja jest rekurencyjna. Należy więc wiedzieć, co zrobić dla $m=1$. Otóż oczywiste jest, że wtedy mamy po prostu wypisać n zer.

W powyższym opisie niejasny jest punkt 4. Wymaga on dokładnego sprecyzowania, co właśnie stanowi zasadniczą trudność. W tym punkcie też uwidoczni się różnica pomiędzy sformułowaniem algorytmu dla liczb n będących liczbami pierwszymi i dla liczb n będących liczbami złożonymi. Algorytm ten dla liczb pierwszych n jest znacznie prostszy. Teraz prześledzimy działanie algorytmu dla $m=3$ (tzn. symboli 0, 1 i 2) oraz $n=5$.

1. Wypisujemy liczbę 2.
2. Dopisujemy 2 ciągi 5-elementowe: 22221 22220.
3. Dopisujemy 4 ciągi 5-elementowe: 22211 22210 22201 22200.
4. Dopisujemy dalsze ciągi 5-elementowe zawierające mniej inaczej rozmieszczonych dwójek:

a. 3 dwójki „oddzielnie":	22121 22120 22021 22020,
b. 2 dwójki „razem":	22111 22110 22101 22100
	22011 22010 22001 22000,
c. 2 dwójki „oddzielnie":	21211 21210 21201 21200
	20211 20210 20201 20200,
d. 1 dwójka:	21111 21110 21101 21100
	21011 21010 21001 21000
	20111 20110 20101 20100
	20011 20010 20001 20000.

5. Dopisujemy ciąg de Bruijna dla $m=2$ i $n=5$ utworzony w analogiczny sposób.

Dalsze szczegóły tego algorytmu pominiemy. Zachęcam zainteresowanych Czytelników do samodzielnego opracowania szczegółów tego algorytmu. Warto natomiast wiedzieć, że algorytm ten nie wymaga tak wielkiej pamięci jak wszystkie algorytmy opisane poprzednio.

6. Wskazówki bibliograficzne

Historię słowa yamátárájabhánasalagám znajdzie Czytelnik w rozdziale 9 książki Shermana K. Steina *Mathematics. The Man-made Universe*. Tam też można znaleźć wiele informacji o historii badań nad istnieniem ciągów de Bruijna. Opis zamków szyfrowych w Baltimore Hilton Inn zaczerpnąłem z artykułu A. Ralstona *The Decline of Calculus - The Rise of Discrete Mathematics* znajdującego się w książce *Mathematics Tomorrow* wydanej pod redakcją Lynn A. Steena. Tam też znajduje się szkic algorytmu Ralstona. Dokładny opis tego algorytmu znajduje się w pracy A. Ralstona *A New Memoryless Algorithm for De Bruijn Sequences*, J. of Algorithms 2, str. 50-62. Interesujący przegląd algorytmów generujących ciągi de Bruijna znajduje się w artykule A. Ralstona *De Bruijn Sequences - A Model Example of the Interaction of Discrete Mathematics and Computer Science*, Mathematics Magazine 55, No. 3 (1982), str. 131–143. Tam też znajduje się dość obszerna literatura dotycząca tych algorytmów. Informacje o teorii grafów i grafach eulerowskich może Czytelnik znaleźć w książkach R.J. Wilsona *Wprowadzenie do teorii grafów*, PWN Warszawa 1985 oraz N. Deo *Teoria grafów i jej zastosowania w technice i informatyce*, PWN Warszawa 1980. W tej ostatniej książce znajdzie również Czytelnik informacje o rozgałęzieniach i ich zastosowaniach do znajdowania cykli Eulera.