Generowanie obiektów kombinatorycznych - materiały na analizę algorytmów

Przygotował Piotr Fabian (PCF@who.net) według notatek własnych, książek (m.in. [1]) itp.
Gliwice, styczeń 2000

1 Wprowadzenie

W wielu algorytmach zachodzi konieczność wygenerowania i sprawdzenia wszystkich albo wybranych elementów pewnej klasy obiektów kombinatorycznych. Niniejsze opracowanie przedstawia metody generacji **permutacji**, **podzbiorów** i **kombinacji**.

Algorytmy generujące obiekty kombinatoryczne można podzielić na dwie klasy:

- generujące wszystkie obiekty danej klasy, każdy dokładnie jeden raz,
- generujące losowy obiekt danej klasy w taki sposób, aby prawdopodobieństwo wylosowania każdego obiektu było jednakowe,
- generujące wybrany (n-ty) obiekt danej klasy.

Czas wykonania algorytmu generującego wszystkie obiekty danej klasy powinien być rzędu liczby tych obiektów (mocy zbioru tych obiektów). Taki algorytm nazywany jest **liniowym**. Czas generacji pojedynczego obiektu nie powinien zależeć od jego numeru (n). Wygenerowanie n-tego obiektu jest zwykle bardziej kłopotliwe niż sukcesywne generowanie wszystkich obiektów.

Większość algorytmów generujących wszystkie obiekty kombinatoryczne danej klasy składa się z trzech faz:

- inicjalizacji, połączonej z wygenerowaniem pierwszego obiektu,
- przetworzenia wygenerowanego obiektu,
- jeśli to możliwe, przekształcania bieżącego obiektu w następny.

Ogólna postać takich algorytmów jest więc następująca:

```
inicjalizacja(obiekt);
while not(ostatni_element(obiekt)) do
  przetwórz(obiekt);  { wykonanie żądanych operacji }
  przekształć_na_następny(obiekt);
endwhile;
```

Algorytmy generujące wszystkie obiekty mogą je generować w różnych uporządkowaniach. Jeśli różnice między kolejnymi obiektami są minimalne, algorytmy takie nazywane są *algorytmami minimalnych zmian*. Inne mogą generować obiekty uporządkowane *leksykograficznie*.

2 Generowanie permutacji

Permutacje należą do często generowanych obiektów kombinatorycznych. Dowolną permutację elementów zbioru $\{a_1, a_2, \ldots, a_n\}$ można zapisać jako ciąg indeksów tych elementów, czyli permutację zbioru liczb naturalnych $\{1, 2, \ldots, n\}$. Dlatego omówione zostaną algorytmy generujące właśnie permutacje liczb naturalnych, co nie zmniejsza ogólności rozwiązania problemu.

2.1 Permutacje w porządku minimalnych zmian

Minimalną zmianą prowadzącą do przekształcenia jednej permutacji w inną jest pojedyncza *transpozycja*, czyli zamiana pozycji pary elementów. Permutacje n elementów w takim porządku generuje następująca metoda rekurencyjna:

- Dla $n = 1$ permutacją jest (1).
- Dla $n > 1$ tworzymy przy użyciu tego algorytmu uporządkowaną listę permutacji $n-1$ elementów i oznaczamy je przez P_i $i = 1, \ldots, (n-1)!$.
- Tworzymy uporządkowaną listę permutacji n elementów generując n nowych permutacji z każdej permutacji P_i poprzez wstawianie liczbę n między elementy P_i. Miejsc, w które można wstawić ten nowy element, jest n. Dla nieparzystych i element n wstawiamy od prawej do lewej strony P_i, dla parzystych od lewej do prawej.

Na rysunku 1 przedstawiono przykładowy przebieg algorytmu dla $n = 4$.

$$
\begin{array}{l}
1 \left\{ \begin{array}{l}
1\,\underline{2} \left\{ \begin{array}{l}
1\,2\,\underline{3} \left\{ \begin{array}{l} 1\,2\,3\,\underline{4} \\ 1\,2\,\underline{4}\,3 \\ 1\,\underline{4}\,2\,3 \\ \underline{4}\,1\,2\,3 \end{array} \right. \\
1\,\underline{3}\,2 \left\{ \begin{array}{l} \underline{4}\,1\,3\,2 \\ 1\,\underline{4}\,3\,2 \\ 1\,3\,\underline{4}\,2 \\ 1\,3\,2\,\underline{4} \end{array} \right. \\
\underline{3}\,1\,2 \left\{ \begin{array}{l} 3\,1\,2\,\underline{4} \\ 3\,1\,\underline{4}\,2 \\ 3\,\underline{4}\,1\,2 \\ \underline{4}\,3\,1\,2 \end{array} \right. \\
\underline{3}\,2\,1 \left\{ \begin{array}{l} \underline{4}\,3\,2\,1 \\ 3\,\underline{4}\,2\,1 \\ 3\,2\,\underline{4}\,1 \\ 3\,2\,1\,\underline{4} \end{array} \right.
\end{array} \right. \\
\underline{2}\,1 \left\{ \begin{array}{l}
2\,3\,\underline{1} \left\{ \begin{array}{l} 2\,3\,1\,\underline{4} \\ 2\,3\,\underline{4}\,1 \\ 2\,\underline{4}\,3\,1 \\ \underline{4}\,2\,3\,1 \end{array} \right. \\
2\,1\,\underline{3} \left\{ \begin{array}{l} \underline{4}\,2\,1\,3 \\ 2\,\underline{4}\,1\,3 \\ 2\,1\,\underline{4}\,3 \\ 2\,1\,3\,\underline{4} \end{array} \right.
\end{array} \right.
\end{array} \right.
\end{array}
$$

Rysunek 1: Generowanie permutacji w porządku minimalnych zmian. Podkreślone zostały elementy wstawiane w kolejnych krokach. Zależnie od parzystości numeru iteracji wstawiane są prawej do lewej bądź od lewej do prawej strony poprzedniej permutacji.

Taka procedura generuje permutacje w porządku minimalnych zmian, ale metodą rekurencyjną, nie stosującą tych zmian do generacji permutacji. Jest również możliwe iteracyjne generowanie permutacji, ale nie zostanie tutaj omówione[1].

[1] może w przyszłości

2.2 Permutacje w porządku leksykograficznym

Uwaga: ten podrozdział nie został omówiony na wykładzie.

Porządek leksykograficzny to taki porządek, w którym permutacje traktowane jako reprezentacje liczb naturalnych uporządkowane są rosnąco, np. 123, 132, 213, 231, 312, 321[2]. Najistotniejszą częścią algorytmu generującego wszsytkie permutacje jest sposób przekształcenia bieżącej permutacji w następną. Można opisać ten sposób następująco:

- Oznacz kolejne elementy bieżącej permutacji przez $(a_1, a_2, \ldots a_n)$.
- Przeglądając elementy od prawej strony, tzn. od a_n w kierunku a_1, znajdź pierwszą pozycję, dla której $a_i < a_{i+1}$.
- Przeglądając elementy położone na prawo od a_i znajdź element a_j taki, że jest on najmniejszy spośród elementów większych od a_i.
- Zamień miejscami element a_i z a_j.
- Odwróć kolejność elementów $(a_{i+1}, \ldots, a_n)$. Elementy te zostaną uporządkowane rosnąco. (indeks i **nie** ulega zamianie z j w poprzednim kroku).

Przykładowo, rozważmy permutację ośmiu elementów $(1, 7, 8, 3, 6, 5, 4, 2)$. Przeglądamy elementy od prawej do lewej i znajdujemy sytuację $3 < 6$, czyli $a_i < a_{i+1}$ dla $i = 4$. Następnie wśród elementów na prawo od trójki szukamy większych od 3; znajdziemy 6, 5 i 4. Z tych elementów wybieramy najmniejszy. Jest to czwórka znajdująca się na pozycji 7, czyli $j = 7$. Teraz zamieniamy j-ty element (czwórkę) z i-tym (trójką), otrzymując $(1, 7, 8, 4, 6, 5, 3, 2)$. Dalej odwracamy kolejność elementów leżących na prawo od i-tego, czyli czwartego. Otrzymujemy permutację $(1, 7, 8, 4, 2, 3, 5, 6)$, która jest szukaną kolejną permutacją w porządku leksykograficznym następującą po $(1, 7, 8, 3, 6, 5, 4, 2)$.

Generację permutacji w porządku leksykograficznym należy zakończyć otrzymując po takim przekształceniu permutację $(n, n-1, \ldots, 1)$, która jest ostatnią permutacją w tym porządku.

2.3 Generowanie n-tej permutacji

Czasem zachodzi konieczność wygenerowania pojedyczej permutacji o zadanym numerze bez generowania permutacji ją poprzedzających. Numer jest indeksem permutacji w wybranym uporządkowaniu. Tutaj zostanie podana przykładowa metoda stosująca poznane już wcześniej[3] **tablice inwersji**. Jak zapewne wszyscy pamiętają, tablica inwersji odpowiadająca pewnej permutacji elementów $1 \ldots n$ ma n elementów. Na i-tej pozycji tej tablicy znajduje się liczba informująca, ile elementów permutacji znajdujących się na lewo od jej i-tego elementu jest większych od tego elementu. Przykładowo, dla permutacji $(4, 3, 5, 2, 1)$ tablica inwersji ma postać $(0, 1, 0, 3, 4)$. Jeśli numerujemy pozycje od 1 do n, to na i-tej pozycji tablicy inwersji mogą znaleźć się tylko całkowite liczby nieujemne mniejsze od i. Łatwo policzyć, że różnych n-elementowych tablicy inwersji jest dokładnie $n!$, czyli tyle, ile permutacji n elementów. Istnieje wzajemnie jednoznaczne odwzorowanie zbioru permutacji w zbiór tablic inwersji.

Wyznaczenie tablicy inwersji dla danej permuacji jest prostym zadaniem. Odwrotna operacja, wyznaczenie permutacji na podstawie tablicy inwersji, jest nieco trudniejsze[4]. Elementy permutacji należy uporządkować rosnąco: $(1, 2, \ldots, n)$. Szukaną permutację konstruuje się od ostatniego elementu. W tablicy inwersji na ostatnim, n-tym miejscu znajduje się liczba $t_n = k$ informująca, ile elementów szukanej permutacji na pozycjach od 1 do $n-1$ jest większych od n-tego elementu permutacji. Wobec tego na n-tej pozycji permutacji musi znaleźć się liczba $n-k$. Podobnie można znaleźć elementy $n-1$, $n-2$ itd. aż do 1.

[2] definicja nieścisła, ale intuicyjna

[3] m.in. przy okazji omawiania średniej złożoności algorytmów sortowania

[4] ale niewiele

Pozostaje więc do rozwiązania wyznaczenie n-tej tablicy inwersji, czyli przekształcenie liczby n na zawartość tablicy inwersji. Pomocne okazują się tu **niejednorodne systemy pozycyjne**. Przyzwyczajeni jesteśmy do stosowania pozycyjnego systemu dziesiętnego, który jest systemem **jednorodnym**, tzn. wagi kolejnych cyfr różnią się o stały czynnik, nazywany podstawą r ($r > 1$). Zapisując liczbę N jako ciąg cyfr d_i takich, że $0 \leq d_i < r$:

$$d_k d_{k-1} d_{k-2} \ldots d_1, \quad d_k \neq 0$$

wyznaczamy jej wartość N następująco:

$$N = d_0 + d_1 r + d_2 r^2 + \ldots + d_k r^k$$

Dla systemu dziesiętnego $r = 10$.

W niejednorodnych systemach pozycyjnych zamiast r stosowany jest ciąg liczb naturalnych r_0, r_1 itd. Wtedy podanemu wcześniej ciągowi odpowiada

$$N = d_0 + d_1 r_0 + d_2 r_0 r_1 + +d_3 r_0 r_1 r_2 \ldots + d_k \prod_{i=0}^{k-1} r_i$$

gdzie $0 \leq d_i < r_i$, $d_k \neq 0$ (oprócz $N = 0$). Przykładem takiego systemu jest zapis czasu w formacie DHMS (dni, godziny, minuty, sekundy). Dla takiego formatu $r_0 = 60$, $r_1 = 60$, $r_2 = 24$.

W omawianym problemie wyznaczania tablic inwersji przyda się tzw. **silnia-system**:

$$N = d_0 0! + d_1 1! + d_2 2! + \ldots + d_k k!$$

gdzie $0 \leq d_i < i + 1$, $d_k \neq 0$ (oprócz $N = 0$), $d_0 = 0$ (zawsze); $r_i = i + 1$. Cyfry tak zapisanej liczby N są bezpośrednio zawartością N-tej tablicy inwersji.

Algorytm przekształcania liczby N na liczbę w niejednorodnym systemie pozycyjnym (w szczególności *silnia-systemie*) jest następujący:

```
d0 := 0;
q  := N;
k  := 0;
while q<>0 do
  dk := q mod rk;
  q  := q div rk;
  k  := k+1;
endwhile;
if k<>0 then k:=k-1;
```

Tym sposobem mając dany numer permutacji n można najpierw przekształcić n na liczbę w silnia-systemie, wypełnić tablicę inwersji cyframi tej liczby i na podstawie tablicy inwersji utworzyć permutację.

2.4 Generowanie losowej permutacji

Znając metodę generowania k-tej permutacji, można wylosować liczbę z przedziału od 1 do $n!$ (gdzie n jest liczbą permutowanych elementów) i utworzyć k-tą permutację. Jeśli każdą liczbę generowaliśmy z takim samym prawdopodobieństwem, to również każda permutacja jest wtedy jednakowo prawdopodobna.

Inny sposób to dokonanie $n - 1$ transpozycji. Zaczynamy od permutacji o uporządkowanych rosnąco elementach $(1, 2, \ldots, n)$. Zamieniamy n-ty element z losowo wybranym elementem spośród 1 do n. Następnie element $n - 1$ zamieniamy z jednym z elementów spośród 1 do $n - 1$ itd. Można wykazać, że tutaj również wszystkie permutacje są jednakowo prawdopodobne[5].

[5] prosty dowód tego faktu zostanie przytoczony w kolejnej wersji materiałów

3 Podzbiory

Podzbiorów zbioru n-elementowego jest 2^n, licząc łącznie ze zbiorem pustym. Tworzenie podzbiorów jest równoważne tworzeniu n-bitowych ciągów binarnych. Element a_i należy do podzbioru wtedy i tylko wtedy, gdy i-ty element ciągu jest jedynką.

Zagadnienie generowania wszystkich podzbiorów sprowadza się do wygenerowania wszystkich n-bitowych liczb dwójkowych[6]. Generowanie losowego podzbioru to wylosowanie liczby z przedziału od 0 do $2^n - 1$ i zamiana tej liczby na postać dwójkową.

4 Kombinacje, czyli podzbiory k-elementowe zbioru n-elementowego

W sposób „brutalny" można wygenerować takie podzbiory generując wszystkie możliwe podzbiory (jak w poprzednim punkcie) i wybierając spośród nich te, które mają dokładnie k elementów. Jednak w ten sposób zawsze trzeba będzie przetworzyć 2^n podzbiorów, tymczasem kombinacji jest tylko $\binom{n}{k}$.

4.1 Porządek leksykograficzny

Przedstawiony zostanie metoda generacji kombinacji w porządku leksykograficznym. Na przykład, jeśli elementami zbioru będą liczby $\{1, 2, 3, 4, 5, 6\}$ i mają zostać wygenerowane trzyelementowe kombinacje, to będą one miały następujący porządek: 123, 124, 125, 126, 134, 135, 136, 145, 146, 156, 234, 235, 236, 245, 246, 256, 345, 346, 356, 456.

Algorytm jest następujący:

- Początkowa kombinacja to $(1, 2, \ldots, k)$
- Kolejną kombinację tworzymy z bieżącej w następujący sposób:
  - przeglądamy bieżącą kombinację od prawej do lewej strony. Znajdujemy element, który jeszcze *nie osiągnął maksymalnej wartości.*
  - Element ten zmiększamy o 1, a wszystkie elementy po jego prawej stronie otrzymują najniższe z dopuszczalnych wartości.

4.2 Porządek minimalnych zmian

Przy generacji kombinacji w porządku minimalnych zmian korzysta się z pojęcia **kodu Gray'a**. W zwykle stosowanej odmianie jest to sekwencja ciągów binarnych, w której kolejne dwa ciągi binarne różnią się dokładnie na jednej pozycji. Na przykład dla dwóch bitów może to być sekwencja 00, 01, 11, 10; dla trzech bitów 000, 001, 011, 010, 110, 111, 101, 100. Jak widać, sekwencje takie można tworzyć rekurencyjnie: przy trzech bitach wystarczy przepisać ciąg sekwencji dwubitowych najpierw w normalnym porządku dopisując zera, potem w odwrotnym porządku dopisując jedynki. Możliwe jest też tworzenie sekwencji o długościach innych niż 2^n, np. przez znajdowanie zamkniętych dróg w tzw. *tablicach Karnaugha*[7].

Do generacji kombinacji korzysta się z nieco innego rodzaju kodu Gray'a. Kolejne ciągi różnią się na dokładnie dwóch bitach i mają zawsze tę samą liczbę jedynek (równą k). Koncepcja generacji takich

[6] co jest tak prostym zadaniem, że przedstawianie tutaj jego rozwiązania mogłoby urazić słuchaczy

[7] były omawiane na kursie Teorii Automatów

sekwencji jest podobna do przedstawionego wcześniej rekurencyjnego generowania „zwykłych” kodów. Przez $G(n,k)$ oznaczymy sekwencję ciągów n-bitowych o k jedynkach. Wtedy:

$$G(n,k) = \begin{pmatrix} 0 & G(n-1,k) \\ 1 & G(n-1,k-1)^R \end{pmatrix}$$

Zapis $G(n-1,k-1)^R$ oznacza, że chodzi o sekwencję $G(n-1,k-1)$, ale w odwróconej kolejności.

$$G(n,0) = 0000\ldots0$$

Dla $G(n,0)$ ciągiem jest ciąg n zer.

$$G(n,n) = 1111\ldots1$$

Dla $G(n,n)$ ciągiem jest ciąg n jedynek.

Od Czytelnika oczekuje się przeanalizowania tego wzoru rekurencyjnego na prostym przykładzie, np. dla $G(4,2)$. Wynikową sekwencją jest: 0011, 0110, 0101, 1100, 1010, 1001. Czytelnik zechce również sprawdzić, że kolejne ciągi różnią się dokładnie na dwóch pozycjach.

4.3 Losowe podzbiory

Aby wygenerować k-elementowy, losowy podzbiór zbioru $S = \{a_1, a_2, \ldots, a_n\}$, należy wylosować jeden z n elementów tego zbioru i uzupełnić podzbiorem $k-1$ elementowym zbioru S pozbawionego wylosowanego elementu.

Bibliografia

[1] Edward M. Reingold, Jurg Nievergelt, Narsingh Deo, *Alorytmy kombinatoryczne*, Państwowe Wydawnictwo Naukowe, Warszawa 1985

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Wprowadzenie do algorytmów*, WNT, Warszawa 1997

Koniec wersji 0.023