WYKŁAD 7

Generowanie podziałów zbioru

Idea ostatniego algorytmu opiera się na tym, że każdy następny podział powstaje z poprzedniego przez przeniesienie pojedynczego elementu aktywnego do innego bloku.

Przedstawimy teraz inną strategię generowania wszystkich podziałów zbioru $\{1, 2, \dots , n\}$, która wykorzystuje pojęcie **funkcji wzrostu z ograniczeniami**.

Niech $n > 1$ i niech $\mathcal{R}(n)$ oznacza zbiór wszystkich funkcji

$$f : \{1, 2, \dots , n\} \to Z^+$$

spełniających warunki $f(1) = 1$ oraz dla $2 \le i \le n$

$$f(i) \le \max\{f(1), \dots , f(i-1)\} + 1$$

Funkcja $f \in \mathcal{R}(n)$ nazywa się *funkcją wzrostu z ograniczeniami* długości n i jest oznaczana krótko przez RGF (*Restricted Growth Function*). Czasami będziemy mówili, że funkcja spełnia własność RGF. Funkcję taką możemy reprezentować jako uporządkowaną n-tkę

$$(f(1), \dots , f(n))$$

złożoną z dodatnich liczb całkowitych. W zaprezentowanych poniżej dwóch procedurach taka n-tka będzie zapisana w postaci $(f_1, \dots , f_n)$.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

Kluczową obserwacją jest fakt, że dla dowolnej liczby $n \geq 1$ istnieje prosta **bijekcja** między zbiorem $\mathcal{S}(n)$ wszystkich podziałów zbioru $\{1, 2, \ldots, n\}$ a zbiorem $\mathcal{R}(n)$ funkcji RGF. Dla przykładu, dla $n = 4$ mamy

$(1, 1, 1, 1)$	$\{1, 2, 3, 4\}$
$(1, 1, 1, 2)$	$\{1, 2, 3\}, \{4\}$
$(1, 1, 2, 1)$	$\{1, 2, 4\}, \{3\}$
$(1, 1, 2, 2)$	$\{1, 2\}, \{3, 4\}$
$(1, 1, 2, 3)$	$\{1, 2\}, \{3\}, \{4\}$
$(1, 2, 1, 1)$	$\{1, 3, 4\}, \{2\}$
$(1, 2, 1, 2)$	$\{1, 3\}, \{2, 4\}$
$(1, 2, 1, 3)$	$\{1, 3\}, \{2\}, \{4\}$
$(1, 2, 2, 1)$	$\{1, 4\}, \{2, 3\}$
$(1, 2, 2, 2)$	$\{1\}, \{2, 3, 4\}$
$(1, 2, 2, 3)$	$\{1\}, \{2, 3\}, \{4\}$
$(1, 2, 3, 1)$	$\{1, 4\}, \{2\}, \{3\}$
$(1, 2, 3, 2)$	$\{1\}, \{2, 4\}, \{3\}$
$(1, 2, 3, 3)$	$\{1\}, \{2\}, \{3, 4\}$
$(1, 2, 3, 4)$	$\{1\}, \{2\}, \{3\}, \{4\}$

Dla łatwiejszej analizy weźmy pierwszych sześć funkcji RGF dla $n = 4$

$$
\begin{array}{ll}
(1,1,1,1) & \{1,2,3,4\} \\
(1,1,1,2) & \{1,2,3\},\{4\} \\
(1,1,2,1) & \{1,2,4\},\{3\} \\
(1,1,2,2) & \{1,2\},\{3,4\} \\
(1,1,2,3) & \{1,2\},\{3\},\{4\} \\
(1,2,1,1) & \{1,3,4\},\{2\}
\end{array}
$$

Zakładamy, że bloki podziału ponumerowane są od lewej strony kolejnymi liczbami naturalnymi (zatem numer bloku w tym przypadku nie musi odpowiadać numerowi bloku w poprzednio rozważanym algorytmie). I tak w piątej funkcji bloki mają numery (od lewej) 1, 2 i 3 a w szóstej funkcji numery bloków to 1 i 2.

Zauważmy, że $f(i)$ jest numerem bloku, do którego należy element i, $i = 1, 2, \dots, n$. I tak, w pierwszej funkcji na pierwszym, drugim, trzecim i czartym miejscu mamy 1, co oznacza, że elementy 1, 2, 3 i 4 należą do jednego bloku, który ma oczywiście numer 1. A np. w trzeciej funkcji 1,2 i 4 należą do jednego bloku o numerze 1 a 3 do drugiego o numerze 2.

I teraz kluczowa obserwacja: największa składowa funkcji jest liczbą bloków podziału, np. w szóstej funkcji największa składowa to 2 i rzeczywiście mamy podział na dwa bloki.

Zauważmy ponadto, że wypisane po lewej stronie funkcje RGF są uporządkowane leksykograficznie. W tym kontekscie możemy mówić o generowaniu wszystkich podziałów danego zbioru w porządku **leksykograficznym**.

Zanim będziemy generować wszystkie takie podziały, przedstawimy najpierw dwie proste procedury, które odpowiednio wyznaczają:

– z zadanego podziału zbioru $\{1, 2, \ldots, n\}$ na k bloków $\{B_1, \ldots, B_k\}$ odpowiadającą mu funkcję RGF $(f_1, \ldots, f_n)$

– z zadanej funkcji RGF $(f_1, \ldots, f_n)$ odpowiadający jej podział zbioru $\{1, 2, \ldots, n\}$ na stosowną liczbę bloków (wyliczaną w procedurze).

Zacznijmy od pierwszej z nich (od razu zaznaczam, że nie jest to jedyna możliwa konstrukcja).

PODZIAŁ-RGF $(n, k, \{B_1, \ldots, B_k\})$
  for $j \leftarrow 1$ **to** n
    do $f_j \leftarrow 0$
  $j \leftarrow 1$
  for $i \leftarrow 1$ **to** k
    do while $f_j \neq 0$
          do $j \leftarrow j + 1$
        $h \leftarrow 1$
        while $j \notin B_h$
          do $h \leftarrow h + 1$
        for each $g \in B_h$
          do $f_g \leftarrow h$
  return $(f_1, \ldots, f_n)$

Dla danego elementu j (zaczynamy od $j = 1$) szukamy numeru bloku zawierającego ten element (drugi **while**) a następnie wszystkim składowym z indeksami równymi elementom należącym do tego bloku nadajemy numer bloku (instrukcja **for each**). Czynność tę powtarzamy k razy biorąc pod uwagę kolejny element j zbioru, dla którego składowa f_j jest jeszcze równa zero. W tym celu przed wejściem do głównej pętli wszystkie składowe szukanej funkcji RGF muszą zostać wyzerowane.

W drugim przypaku, procedura generująca podział zbioru $\{1, 2, \ldots, n\}$ na stosowną liczbę bloków na podstawie zadanej funkcji RGF $(f_1, \ldots, f_n)$ ma następującą postać (ten sam komentarz co poprzednio).

RGF-PODZIAŁ $(n, (f_1, \ldots, f_n))$

  $k \leftarrow 1$
  for $j \leftarrow 1$ **to** n
    do if $f_j > k$
        then $k \leftarrow f_j$
  for $i \leftarrow 1$ **to** k
    do $B_i \leftarrow \emptyset$
  for $j \leftarrow 1$ **to** n
    do $B_{f_j} \leftarrow B_{f_j} \cup \{j\}$
  return $\{B_1, \ldots, B_k\}$

Pierwszy **for** wyznacza, na podstawie zadanej funkcji RGF, liczbę bloków w podziale odpowiadającym tej funkcji. Sprowadza się to, tak na prawdę, do wyznaczenia największej składowej. W ostatniej pętli **for** wstawiane są do kolejnych bloków podziału (które początkowo były zbiorami pustymi) stosowne elementy.

Przjdźmy teraz do omówienia algorytmu generowania wszystkich funkcji wzrostu z ograniczeniami długości n w uporządkowaniu leksykograficznym. Przypomnijmy, że RGF są to funkcje

$$f : \{1, 2, \dots , n\} \to Z^+$$

spełniające warunki $f(1) = 1$ oraz dla $2 \leq i \leq n$

$$f(i) \leq \max\{f(1), \dots , f(i-1)\} + 1$$

Zakładamy, że „aktualną” funkcję RGF przechowujemy w tablicy $f[1], f[2], \dots , f[n]$. Rozpoczynamy oczywiście z tablicą f zawierającą same jedynki. Algorytm wykorzystuje dodatkową tablicę $F[1..n]$, której element $F[j]$ oznacza maksymalną wartość jaką można przyporządkować składowej $f[j]$, przy zadanych wartościach $f[1], \dots , f[j-1]$, nie naruszając własności RGF. Dokładniej

$$F[j] = 1 + \max\{f[i] : \quad 1 \leq i \leq j-1\}$$

dla $j = 2, \dots , n$. Początkowo czynimy wszystkie elementy tablicy $F[1..n]$ równe dwa.

Idea algorytmu, podobnie jak w innych algorytmach generujących obiekty w porządku leksykograficznym, polega na

1. znalezieniu pierwszej pozycji z prawej strony tablicy f, dla której $f[j] \neq F[j]$
2. wartość elementu $f[j]$ zostaje zwiększona o jeden a wszystkie elementy leżące na prawo od niego czynimy równe jeden, tj.

   $$f[i] \leftarrow 1$$

   dla wszystkich i, takich że $j + 1 \leq i \leq n$
3. uaktualniamy tablicę F; polega to na tym, że jeżeli nowa wartość $f[j]$ jest równa wartości $F[j]$, to wszystkie elementy tablicy F leżące na prawo, tj. $F[i]$ dla $i = j+1, \ldots, n$ przyjmują wartość $F[j]+1$, a w przeciwnym razie ($f[j] \neq F[j]$) przyjmują one wartość $F[j]$
4. proces ten kontynujemy dopóty, dopóki nie będzie spełniony warunek

   $$f[j] = F[j]$$

   dla wszystkich j, takich że $2 \leq j \leq n$. To odpowiada ostatniej wygenerowanej funkcji FRG, która ma postać $(1, 2, \ldots, n)$.

Pseudokod algorytmu generowania wszystkich funkcji wzrostu z ograniczeniami długości n w uporządkowaniu leksykograficznym:

```
GENERUJ-RGF (n)
  for i ← 1 to n
    do f[i] ← 1
       F[i] ← 2
  koniec ← false
  while not koniec
      do wypisz f[1], ..., f[n]
         j ← n + 1
         repeat
           j ← j − 1
         until f[j] ≠ F[j]
         if j > 1
           then f[j] ← f[j] + 1
                for i ← j + 1 to n
                    do f[i] ← 1
                       if f[j] = F[j]
                          then F[i] ← F[j] + 1
                          else F[i] ← F[j]
           else koniec ← true
```