Notatki z AiSD. Nr 10. 4 kwietnia 2022

SORTOWANIE

IIUWr. II rok informatyki. *Opracował: Krzysztof Loryś*

Na dzisiejszym wykładzie poznamy algorytmy, sortujące w czasie niższym niż wynika to z dolnego ograniczenia poznanego na poprzednim wykładzie. Jest to możliwe z dwóch powodów. Po pierwsze algorytmy te zakładają pewne ograniczenia na postać danych, a po drugie wykonują one na sortowanych elementach operacje inne niż porównania.

1 Counting Sort

POSTAĆ DANYCH: ciąg $A[1..n]$ liczb całkowitych z przedziału $\langle 1, k\rangle$.

IDEA: $\forall_{x\in A[1..n]}$ obliczyć liczbę $c[x] = |\{y \ : \ y \in A[1..n] \ \& \ y \leq x\}|$.

```
procedure Counting − Sort(A[1..n], k, var B[1..n])
    for i ← 1 to k do c[i] ← 0
    for j ← 1 to n do c[A[j]] ← c[A[j]] + 1
    for i ← 2 to k do c[i] ← c[i] + c[i − 1]
    for j ← n downto 1 do  B[c[A[j]]] ← A[j]
                           c[A[j]] ← c[A[j]] − 1
```

UWAGA: W oczywisty sposób powyższa procedura może być zmodyfikowana do sortowania rekordów, w których klucz $A[j]$ jest jednym z wielu pól.

Definicja 1 *Metodę sortowania nazywamy* stabilną, *jeśli w ciągu wyjściowym elementy o tej samej wartości klucza pozostają w takim samym porządku względem siebie w jakim znajdowały się w ciągu wejściowym.*

Fakt 1 *Counting − sort jest metodą stabilną.*

KOSZT: $\Theta(n + k)$.

2 Sortowanie kubełkowe (bucket sort).

POSTAĆ DANYCH: Ciąg $A[1..n]$ liczb rzeczywistych z przedziału $\langle 0, 1)$ wygenerowany przez generator liczb losowych o rozkładzie jednostajnym.

IDEA: Podzielić przedział $\langle 0, 1)$ na n odcinków ("kubełków") jednakowej długości; umieścić liczby w odpowiadających im kubełkach; posortować poszczególne kubełki; połączyć kubełki.

```
procedure bucket − sort(A[1..n])
    for i ← 0 to n − 1 do B[i] ← ∅
    for i ← 1 to n do  dołącz A[i] do listy B[⌊nA[i]⌋]
    for i ← 0 to n − 1 do  posortuj procedurą select − sort listę B[i]
    połącz listy B[0], B[1], . . . , B[n − 1]
```

KOSZT: Oczekiwany czas działania: $\Theta(n)$.

UZASADNIENIE: Niech Y będzie zmienną losową równą liczbie porównań wykonanych podczas sortowania kubełków. Mamy

$$Y = Y_1 + \ldots + Y_n,$$

gdzie Y_i jest zmienną losową równą liczbie porównań wykonanych podczas sortowania i-tego kubełka. Z liniowości wartości oczekiwanej mamy

$$E[Y] = \sum_{i=1}^{n} E[Y_i].$$

Niech X_i będzie zmienną losową równą liczbie elementów w i-tym kubełku. Oczywiście X_i ma rozkład dwumianowy, w którym prawdopodobieństwo sukcesu wynosi $1/n$. Ponieważ do sortowania kubełków stosujemy *select-sort*, mamy $Y_i = X_i^2$. Stąd wystarczy teraz oszacować $E[X_i^2]$. □

3 Sortowanie leksykograficzne ciągów jednakowej długości (radix sort).

Porządek leksykograficzny na ciągach skończonej długości definiujemy w sposób analogiczny do porządku leksykograficznego na słowach.

Definicja 2 *Niech Σ - zbiór uporządkowany liniowo oraz $s_1, \ldots, s_p, t_1, \ldots, t_q \in \Sigma$.*

$$(s_1, \ldots, s_p) \leq (t_1, \ldots, t_q) \stackrel{df}{\Longleftrightarrow} \begin{array}{ll} \textit{(1)} & \exists_{1 \leq j \leq \min(p,q)}\, s_j < t_j \,\&\, \forall_{i<j}\, s_i = t_i \\ \textit{albo} & \\ \textit{(2)} & p \leq q \,\&\, \forall_{1 \leq i \leq p}\, s_i = t_i. \end{array}$$

Innymi słowy, ciąg S jest wcześniejszy leksykograficznie niż ciąg T, jeśli S jest właściwym prefiksem T albo, w przeciwnym wypadku, gdy na pierwszej różniącej się pozycji S ma wcześniejszy (w Σ) element niż T.

POSTAĆ DANYCH: $A_1, \ldots, A_n$ - ciągi elementów z $\Sigma = \{0, 1, \ldots, k-1\}$ o długości d.

```
procedure radix − sort(A_1, .., A_n)
    for i ← d downto 1 do
        metodą stabilną posortuj ciągi wg i−tego elementu
```

KOSZT: Jeśli w procedurze *Radix−sort* zastosujemy *counting−sort*, to jej koszt wyniesie $O((n+k)d)$. Jest to koszt liniowy, gdy $k = O(n)$.

4 Sortowanie leksykograficzne ciągów niejednakowej długości.

POSTAĆ DANYCH: $A_1, \ldots, A_n$ ciągi elementów z $\Sigma = \{0, 1, \ldots, k-1\}$.

Niech l_i oznacza długość A_i, a $l_{max} = \max\{l_i \,:\, i = 1, .., n\}$.

4.1 Pierwszy sposób

Idea: Uzupełnić ciągi specjalnym elementem (mniejszym od każdego elementu z Σ), tak by miały jednakową długość i zastosować algorytm z poprzedniego punktu.

Koszt: $\Theta((n+k)\cdot l_{max})$.

Uwaga: Jest to metoda nieefektywna, gdy ciągów długich jest niewiele.

4.2 Drugi sposób

Chcemy opracować metodę sortującą w czasie liniowym względem rozmiaru danych, który jest równy $l_{total} = \sum_{i=1}^{n} l_i$.

Idea:

for $i \leftarrow l_{max}$ **downto** 1 **do**
  metodą stabilną posortuj ciągi o długości $\geq i$ wg i-tej składowej

Algorytm:

1. Utwórz listy $niepustekubelki[l]$ $(l = 1,..,l_{max})$ takie, że
   - $x \in niepustekubelki[l]$ iff x jest $l-$tą składową jakiegoś ciągu A_i.
   - $niepustekubelki[l]$ jest uporządkowana niemalejąco.
2. Utwórz listy $ciagi[l]$ $(l = 1,..,l_{max})$ takie, że $ciagi[l]$ zawiera wszystkie ciągi A_i o długości l.
3. $kolsłów \leftarrow \emptyset$
   for $j \leftarrow 0$ **to** $k-1$ **do** $q[j] \leftarrow \emptyset$
   for $l \leftarrow l_{max}$ **downto** 1 **do**
     $kolsłów \leftarrow concat(ciagi[l], kolsłów)$
     while $kolsłów \neq \emptyset$ **do**
       $Y \leftarrow$ pierwszy ciąg z $kolsłów$
       $kolsłów \leftarrow kolsłów \setminus \{Y\}$
       $a \leftarrow l$-ta składowa ciągu Y
       $q[a] \leftarrow concat(q[a], \{Y\})$
     for each $j \leftarrow niepustekubelki[l]$ **do**
       $kolsłów \leftarrow concat(kolsłów, q[j])$
       $q[j] \leftarrow \emptyset$

Operacja $concat(K_1, K_2)$ dołącza kolejkę K_2 do końca kolejki K_1.

Twierdzenie 1 *Powyższy algorytm można zaimplementować tak, by działał w czasie $O(k + l_{total})$.*

Uzasadnienie:
Jedynym niezupełnie trywialnym krokiem jest krok 1, w którym tworzone są listy *niepustekubelki*:

- tworzymy w czasie $O(l_{total})$ ciąg P zawierający wszystkie pary $\langle l, a\rangle$, takie, że a jest l-tą składową jakiegoś A_i;
- sortujemy leksykograficznie w czasie $O(k + l_{total})$ ciąg P;
- przeglądając P z lewa na prawo tworzymy $O(l_{total})$ listy *niepustekubelki*.

Krok 2 wymaga czasu $O(l_{total})$.

Aby oszacować czas wykonania kroku 3, oszacujemy czas wykonania dwóch jego pętli wewnętrznych:

- Wewnętrzna pętla while działa w czasie proporcjonalnym do sumarycznej (po wszystkich iteracjach pętli zewnętrznej) długości kolejek *kolslw*. Ponieważ w l-tej iteracji *kolslw* ma długość równą liczbie ciągów co najmniej l-elementowych, więc koszt while jest $O(l_{total})$.
- Wewnętrzna pętla for działa w czasie proporcjonalnym do sumarycznej (po wszystkich iteracjach pętli zewnętrznej) długości list *niepustekubelki*. Ponieważ w każdej iteracji *niepustekubelki* jest nie dłuższa od *kolslw*, czas pętli for jest również $O(l_{total})$.

□

4.3 Przykład zastosowania

PROBLEM:

Dane: T_1, T_2 - drzewa o ustalonych korzeniach,
Zadanie: sprawdzić, czy T_1 i T_2 są izomorficzne.

IDEA: Wędrując przez wszystkie poziomy (począwszy od najniższego) sprawdzamy, czy na każdym poziomie obydwa drzewa zawierają taką samą liczbę wierzchołków tego samego typu (wierzchołki będą tego samego typu, jeśli poddrzewa w nich zakorzenione będą izomorficzne).

ALGORYTM:

Bez zmniejszenia ogólności możemy założyć, że obydwa drzewa mają tę samą:

- wysokość,
- liczbę liści na każdym poziomie.

1. $\forall_{v\ -\ \text{liść w}\ T_i}\ kod(v) \leftarrow 0$
2. **for** $j \leftarrow depth(T_1)$ **downto** 1 **do**
3. $S_i \leftarrow$ zbiór wierzchołków T_i z poziomu j nie będących liśćmi
4. $\forall_{v \in S_i}\ key(v) \leftarrow$ wektor $\langle i_1, \ldots, i_k\rangle$, taki że
   - $i_1 \leq i_2 \leq \ldots \leq i_k$
   - v ma k synów $u_1, \ldots, u_k$ i $i_l = kod(u_l)$
5. $L_i \leftarrow$ lista wierzchołków z S_i posortowana leksykograficznie według wartości key
6. $L_i^{'} \leftarrow$ otrzymany w ten sposób uporządkowany ciąg wektorów
7. **if** $L_1^{'} \neq L_2^{'}$ **then return** ("nieizomorficzne")
8. $\forall_{v \in L_i}\ kod(v) \leftarrow 1 + rank(key(v), \{key(u) \mid u \in L_i\})$
9. Na początek L_i dołącz wszystkie liście z poziomu j drzewa T_i
10. **return** ("izomorficzne")

Twierdzenie 2 *Izomorfizm dwóch ukorzenionych drzew o n wierzchołkach może być sprawdzony w czasie $O(n)$.*