Algorytmy i Struktury Danych, 7. ćwiczenia, przygotowanie do kolokwium

2023-11-20

Klasówka 2021 (1), zadanie 2

Zaproponuj optymalny ze względu na porównania algorytm sortowania siedmioelementowego ciągu $x_1, \ldots, x_7$, o którym wiadomo, że $x_1 < x_2$, $x_1 < x_3$, $x_4 < x_5$, $x_4 < x_6$. Udowodnij optymalność swojego algorytmu.

Rozwiązanie: Wszystkich permutacji 7 elementowych spełniających warunki zadania jest:

$$\binom{7}{3} \cdot 2 \cdot \binom{4}{3} \cdot 2 = 35 \cdot 2 \cdot 4 \cdot 2 = 560$$

Ponieważ:

- $\{x_1, x_2, x_3\}$ możemy wybrać z 7 elementów na $\binom{7}{3}$ sposobów
- mamy dwie możliwości na porównanie x_2 i x_3 ($x_2 < x_3$ lub $x_2 > x_3$)
- $\{x_4, x_5, x_6\}$ możemy wybrać z pozostałych 4 elementów na $\binom{4}{3}$ sposobów,
- mamy dwie możliwości na porównanie x_5 i x_6 ($x_5 < x_6$ lub $x_5 > x_6$)

Algorytm 1: Rozwiązanie

cmp(x_2, x_3) (bez straty ogólności $x_2 < x_3$)
cmp(x_5, x_6) (bez straty ogólności $x_5 < x_6$)
wstaw x_7 pomiędzy $x_1 < x_2 < x_3$ używając 2 porównań
scal dwa uporządkowane ciągu (4 i 3 elementowe) przy użyciu 6 porównań

Klasówka 2007 (1), zadanie 2

Udowodnij, że jeśli algorytm sortujący tablicę $A[1..n]$ porównuje i zamienia wyłącznie elementy odległe co najwyżej o 2007 (tzn. jeśli porównuje $A[i]$ z $A[j]$, to $|i - j| \leq 2007$), to jego pesymistyczny czas działania jest co najmniej kwadratowy.

Rozwiązanie: Pojedyncza zamiana usuwa $O(1)$ inwersji więc dla ciągu odwrotnie uporządkowanego (który ma $\Theta(n^2)$ inwersji) algorytm wymaga czasu $\Omega(n^2)$.

Klasówka 2020 (1), zadanie 1

Na uporządkowanym rosnąco n-elementowym ciągu $x_1, x_2, \ldots, x_n$ dokonano dokładnie k zmian wartości elementów w tym ciągu, $0 \leq k \leq n$. Ciąg otrzymany w ten sposób nazywamy *k-zaburzonym.*

Przykład

Uporządkowany ciąg $\langle 2, 6, 8, 12, 21\rangle$. Po zmianach wartości elementów - pierwszego na 7 i czwartego na 1 - dostajemy ciąg 2-zaburzony $\langle 7, 6, 8, 1, 21\rangle$.

Dana jest liczba całkowita k, $0 \leq k \leq n$, oraz *k*-zaburzony ciąg $x = \langle x_1, x_2, \ldots, x_n\rangle$. Interesuje nas wydajne sortowanie ciągu x ze względu na porównania.

a) Zaproponuj optymalne sortowanie ze względu na porównania dla $k = 1$ oraz

- $n = 4$
- $n = 5$

b) Zaproponuj asymptotycznie optymalny algorytm sortowania k-zaburzonych ciągów n-elementowych. Pamiętaj o uwzględnieniu obu parametrów - n i k.

Rozwiązanie: Punkt b)

Algorytm 2: Rozwiązanie$((x_1, \ldots, x_n), k)$

```
A = Y = ∅
for i = 1, ..., n do
    if A ≠ ∅ and Top(A) > x_i then
        ▷ co najmniej jeden elemenet z {Top(A), x_i} jest zaburzony, dla
          pewmości wrzucamy oba
        Push(Y, x_i)
        Push(Y, Pop(A))
    else
        Push(A, x_i)
▷ |Y| ≤ 2k, A zawiera ciąg uporządkowany rosnąco
Y' = sort(Y)
return Merge(A, Y')
```

Klasówka 2017 (1), zadanie 1

W liczbowym, różnowartościowym ciągu $< a_1, a_2, \ldots, a_n >$, $n > 2$, element a_i, $1 < i < n$, nazywamy lokalnym ekstremum gdy jest mniejszy lub większy od obu sąsiadów, tzn.

albo $a_{i-1} > a_i < a_{i+1}$, albo $a_{i-1} < a_i > a_{i+1}$

a) Udowodnij, że każdy algorytm sortujący przez porównania 4-elementowe ciągi z co najwyżej jednym lokalnym ekstremum wymaga wykonania w pesymistycznym przypadku co najmniej 4 porównań.

b) Zaproponuj algorytm sortowania 4-elementowych ciągów z co najwyżej 1 lokalnym ekstremum za pomocą co najwyżej 4 porównań

c) Zaproponuj algorytm, asymptotycznie optymalny ze względu na liczbę porównań, sortujący ciągi o co najwyżej k lokalnych ekstremach dla zadanego k, $0 < k < n$. Dowiedź optymalności swojego rozwiązania.

Klasówka 2021 (1), zadanie 1

Powiemy, że zbiór n liczb całkowitych jest prawie gęsty, jeśli zawiera podzbiór o rozmiarze większym niż $n/3$, w którym różnica pomiędzy największym i najmniejszym elementem jest mniejsza od n. Taki podzbiór nazywamy świadectwem. Dana jest dodatnia liczba całkowita n oraz n-elementowy zbiór liczb całkowitych S. Zaproponuj algorytm, który w czasie liniowym sprawdzi, czy S jest prawie gęsty.
Uwaga: 3 punkty uzyskasz za algorytm, który w czasie liniowym sprawdza, czy wskazany, dowolny element z S należy do jakiegoś świadectwa.

Rozwiązanie:

Algorytm 3: CzyNależy(S, n, x)

niech $S' = \{e \in S : |e - x| < n\}$
ponieważ wszystkie elementy z S' należą do przedziału $[e - n, \ldots, e + n]$ długości $2n + 1$ stąd możemy posortować S' w czasie liniowym
$S'' = sorted(S')$
for $i = 1, \ldots, m - \lceil \frac{n}{3} \rceil + 1$ **do**
    $j := i + \lceil \frac{n}{3} \rceil - 1$
    if $S''[j] - S''[i] < n$ **then**
        return *TAK* (ponieważ $s_i, \ldots, s_j$ + być ewentualnie x są świadectwem)
return *NIE*

Algorytm 4: Rozwiązanie(S, n)

foreach $k \in \{\lceil n/4 \rceil, \lceil n/2 \rceil, \lceil 3n/4 \rceil\}$ **do**
    $x :=$ DeterministicSelectKthElement(S, k)
    if *CzyNależy*(S, n, x) **then**
        return *TAK*
return *NIE*

Klasówka 2021 (1), zadanie 3

Dla dodatniej liczby całkowitej n kratownicą M_n nazywamy skierowany graf (V, E) bez pętli, w którym $V = \{(x, y) : x = 0, 1, ..., n$ oraz $y = 0, 1, ..., n\}$ i

$$E = \{(x, y) \to (x', y') : 0 \leq x' - x \leq 1 \text{ oraz } 0 \leq y' - y \leq 1\}.$$

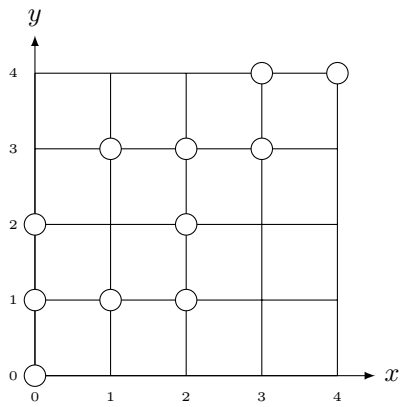
Wierzchołki grafu M_n pomalowano na biało lub czarno. Białą ścieżką nazwiemy każdą ścieżkę, na której wszystkie wierzchołki są białe. Dane są liczba całkowita $n > 0$, nieujemna liczba całkowita $m \leq (n + 1)^2$ oraz m różnych, białych wierzchołków $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$. Pozostałe wierzchołki są czarne. Zaproponuj wydajny (czasowo i pamięciowo) algorytm, który obliczy liczbę wszystkich białych ścieżek z wierzchołka $(0, 0)$ do wierzchołka (n, n).

Rozwiązanie: Aby uprościć rozumowanie załóżmy, że wszystkie operacje arytmetyczne możemy wykonywać w czasie stałym. Jest to o tyle istotne, że

wynikowa wartość może być większa niż $\binom{2n}{n}$ (to jest liczba ścieżek dla pełnego białego grafu M_n w których poruszamy się tylko w prawo lub w górę).
Przykład:
Dla $n = 4$ i $m = 11$ białych wierzchołków:

$$B = \{(0,0),(0,1),(0,2),(1,1),(1,3),(2,1),(2,2),(2,3),(3,3),(3,4),(4,4)\}$$

mamy następujący graf:



Niech $Ile(i,j)$ oznacza liczbę białych ścieżek z $(0,0)$ do (i,j) spełniających warunki zadania.

Jeśli $m = \Theta(n^2)$ to możemy w czasie $O(n^2)$ policzyć $Ile(n,n)$ korzystając z następujacych wzorów:

$$Ile(0,0) = \begin{cases} 1 & \text{jeśli wierzchołek } (0,0) \text{ jest biały} \\ 0 & wpp \end{cases}$$

$$Ile(\cdot,-1) = Ile(-1,\cdot) = 0$$

Jeśli (i,j) jest biały to:

$$Ile(i,j) = Ile(i-1,j) + Ile(i-1,j-1) + Ile(i,j-1)$$

Jeśli (i,j) jest czarny to:

$$Ile(i,j) = 0$$

Jeśli $m < n$ to odpowiedzią jest 0 (nie ma wystarczająco dużo białych węzłów).

Jeśli $m = \Omega(n)$ to możemy policzyć $Ile(n,n)$ w czasie $O(m)$. Załóżmy, że wierzchołki $(0,0)$ i (n,n) są białe (jeśli byłoby inaczej to oczywiście odpowiedzią jest 0).
Zdefiniujmy sobie 3 porządki ($<_{x,y}$, $<_{y,x}$, $<_d$) na parach liczb:

- $(x_1,y_1) <_{x,y} (x_2,y_2)$ wtw $(x_1 < x_2)$ lub $(x_1 = x_2) \wedge (y_1 < y_2)$
- $(x_1,y_1) <_{y,x} (x_2,y_2)$ wtw $(y_1 < y_2)$ lub $(y_1 = y_2) \wedge (x_1 < x_2)$
- $(x_1,y_1) <_d (x_2,y_2)$ wtw $(x_1 - y_1 < x_2 - y_2)$ lub $(x_1 - y_1 = x_2 - y_2) \wedge (x_1 < x_2)$

W pierszym kroku uporządkujmy leksykograficznie (czyli wg. $<_{x,y}$) ciąg B. Od teraz zakładamy, że:

$$(x_1, y_1) <_{x,y} (x_2, y_2) <_{x,y} \ldots <_{x,y} (x_m, y_m)$$

Przy takim uporządkowaniu definiujemy ILE$[i]$ jako liczbę białych ścieżek z $(0, 0)$ do (x_i, y_i).
Dodatkowo definiujemy pomocnicze wartości:

$$\text{GDZIEX}[i] = \begin{cases} j & \text{jeśli } (x_j = x_i) \wedge (y_j + 1 = y_i) \\ -1 & \text{jeśli takie } j \text{ nie istnieje} \end{cases}$$

$$\text{GDZIEY}[i] = \begin{cases} j & \text{jeśli } (y_j = y_i) \wedge (x_j + 1 = x_i) \\ -1 & \text{jeśli takie } j \text{ nie istnieje} \end{cases}$$

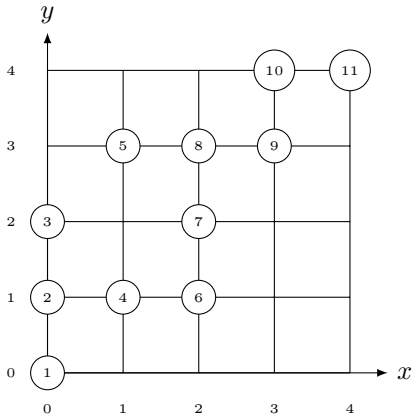
$$\text{GDZIED}[i] = \begin{cases} j & \text{jeśli } (x_j + 1 = x_i) \wedge (y_j + 1 = x_i) \\ -1 & \text{jeśli takie } j \text{ nie istnieje} \end{cases}$$

Wartości GDZIE$(X/Y/D)$ można obliczyć w czasie $O(m)$ sortując wierzchołki wg porządków $<_{x,y}$ / $<_{y,x}$ / $<_d$ (co możemy zrobić w czasie $O(m)$). Jedynym kandydatem na wartość Gdzie jest poprzedni wierzchołek w danym porządku (co możemy zweryfikować w czasie $O(1)$).
Gdy już mamy obliczone wartości Gdzie, wartości Ile obliczamy korzystając ze wzoru:

$$\text{ILE}[i] = \text{ILE}[GdzieX[i]] + \text{ILE}[GdzieY[i]] + \text{ILE}[GdzieD[i]]$$

(przy czym sztucznie definiujemy ILE$[-1] = -0$)

Przykład:



i	1	2	3	4	5	6	7	8	9	10	11
(x_i, y_i)	(0,0)	(0,1)	(0,2)	(1,1)	(1,3)	(2,1)	(2,2)	(2,3)	(3,3)	(3,4)	(4,4)
GDZIEX$[i]$	-1	1	2	-1	-1	-1	6	7	-1	9	-1
GDZIEY$[i]$	-1	-1	-1	2	-1	4	-1	5	8	-1	10
GDZIED$[i]$	-1	-1	-1	1	3	-1	4	-1	7	8	9
ILE$[i]$	1	1	1	2	1	1	3	4	7	11	18

1 Klasówka 2013 (1), zadanie 2

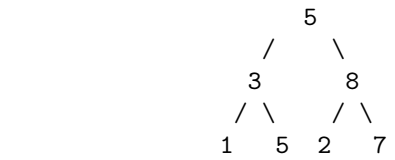
Drzewem klasówkowym nazywamy pełne drzewo binarne, w którym klucze są rozmieszczone zgodnie z następującą regułą: dla każdego węzła x najmniejszy klucz w poddrzewie o korzeniu x znajduje się w jego lewym podrzewie.

Zaproponuj implementację drzewa klasówkowego w sposób, który umożliwia wydajne wykonywanie operacji kolejki priorytetowej:

- Ini:: mając dane $n = 2^k - 1$ kluczy zbuduj n-węzłowe drzewo klasówkowe
- Min:: podaj wartość najmniejszego klucza w drzewie
- ChangeKey(x,k):: zmień wartość klucza we wskazanym węźle x na k

Uzasadnij poprawność swoich rozwiązań oraz dokonaj analizy ich złożności obliczeniowej.

Rozwiązanie: Przykład drzewa klasówkowego:



Wzbogacamy węzły v o dodatkowy atrybut min, który zawiera najmniejszy element z poddrzewa (łącznie z wartością $v.x$). Formuła na aktualizację tego atrybutu:

$$v.min := \min(v.left.min, v.right.min, v.x)$$

Tak jak w kopcu jesteśmy w stanie zdefiniować operację $DownHeap(v)$:

- jeśli v jest liściem to nic nie rób,
- jeśli $v.left.min > v.right.min$ -> $swap(v.left, v.right)$
- jeśli $v.x < v.left.min, v.right.min$ -> zamień $v.x$ i $v.left.x$, zaktualizuj $v.min$ i wykonaj $DownHeap(v.left)$

Analogicznie $UpHeap(v)$:

- jeśli v jest korzeniem to nic nie rób,
- niech $p = parent(v)$,
- zaktualizuj $p.min$,
- jeśli $p.left.min > p.right.min$ -> $swap(p.left, p.right)$
- wykonaj $UpHeap(p)$

Implementacja operacji:

- Ini:: utwórz drzewo a następnie wykonaj $DownHeap$ dla wszystkich węzłów idąc od warstwy k to 1 (tak jak w liniowym algorytmie tworzenia kopca)
- Min:: zwróć $root.min$
- ChangeKey:: zmień klucz i wykonaj $DownHeap(v)$ i $UpHeap(v)$

Klasówka 2022 (1), zadanie 1

Dla dodatniej liczby całkowitej n, w tablicy a[1..n] zapisano n różnych elementów z pewnego liniowo uporządkowanego uniwersum. Rangą elementu a[i], oznaczanego przez R(i), nazywamy pozycję tego elementu w tablicy a po jej uporządkowaniu rosnąco.

Przykład

Dla $a = [2, 5, 3, 4]$, rangami elementów $a[1], a[2], a[3]ia[4]$ są odpowiednio 1, 4, 2, 3.

Dla nieujemnej liczby całkowitej k powiemy, że tablica jest k-zaburzona wtedy i tylko wtedy, gdy dla każdego $i = 1, 2, \ldots, n$, $|R(i) - i| \leq k$. Dana jest liczba całkowita k, $0 \leq k < n$, oraz k-zaburzona tablica a[1..n].

a) Zaproponuj wydajny algorytm, który zbuduje w tablicy a kopiec zupełny typu MIN.

b) Zaproponuj algorytm, optymalny ze względu na liczbę porównań, sortowania 1-zaburzonej tablicy a.

c) Zaproponuj asymptotycznie optymalny algorytm sortowania tablicy a.

Rozwiązanie: Obserwacja: w tablicy k-zaburzonej jeśli $i + 2k < j$ to mamy $A[i] < A[j]$.

Punkt a) utwórz kopiec na pierwszych $4k$ elementach (pozostałe elementy będą już tworzyć dobry porządek kopcowy.

Punkt b) w tablicy 1-zaburzonej tylko sąsiednie elementy mogą być zamienione. Więc wystarczy $n - 1$ porównań.

Punkt c) złożoność czasowa $O(n \log k)$:

Algorytm 5: Rozwiązanie(A, n, k)

```
i = 1
while i < n do
    HeapSort(A[i, ..., i + 4k])
    i+ = 2k
```

Klasówka 2022 (1), zadanie 2

W tym zadaniu rozważamy skończone słowa nad alfabetem $\{d, i, k, s\}$. Niech s będzie słowem i niech $s[j]$ będzie j-tym znakiem w tym słowie. Blokiem znaku $s[j]$ nazywamy maksymalne podsłowo s zawierające znak $s[j]$, w którym wszystkie znaki są takie same, równe $s[j]$. Taki blok oznaczamy przez $B(s, j)$.

Przykład

W słowie $s = abbaac$ mamy $B(s, 3) = bb$, $B(s, 4) = aa$.

O dwóch słowach s_1 i s_2 powiemy, ze są podobne wtedy i tylko wtedy, gdy $|s_1| = |s_2|$ oraz dla każdego $j = 1, \ldots, |s_1|$, bloki $B(s_1, j)$ i $B(s_2, j)$ są tej samej długości.

Zaprojektuj wydajny algorytm, który dla danego zbioru Q złożonego z n skończonych słów nad alfabetem $\{d, i, k, s\}$, wyznaczy wszystkie jego maksymalne (nie dające się rozszerzyć) podzbiory słów podobnych.

Rozwiązanie: Dla słów z Q definiujemy funkcję kodową $C(x_1, ldots, x_k) = (i_1, \ldots, i_p)$ takie, że $1 \leq i_1 \leq i_2 \leq i_p = k$ oraz $x[i_j] \neq x[i_{j+1}]$.

Następnie sotujemy wszystkie kody słów i liczymy ile różnych kodów otrzymaliśmy.

2 Klasówka 2019 (1), zadanie 2

Rozważamy klasyczne sortowanie przez wstawianie tablicy a[1..n], w porządku niemalejącym, w której zapisano pewną permutację liczb od 1 do n. Zaprojektuj wydajny algorytm, który dla danej zawartości tablicy a wyznaczy najmniejsze takie i, że podczas sortowania przez wstawianie a[i] jest porównywane najczęściej.

Rozwiązanie:

Element $a[i]$ jest porównany:

- niech $k = |\{j : a[j] > a[i] \text{ oraz } j < i\}|$ z k elementami podczas i–tej fazy sortowania, które zakończyły się zamianą elementów
- niech $m = |\{j : a[j] < a[i] \text{ oraz } j > i\}|$ z m elementami podczas kolejnych faz sortowania, które zakończyły się zamianą elementów
- niech p to liczba elementów z którymi porównywany jest $a[i]$ ale porównanie nie zakończyło się zamianą.

Jak policzyć wartości p:

---

foreach $x \in A$ **do**
  $p_x = 0$
$S := \emptyset$
for $i = 1..n$ **do**
  $x = A[i]$
  $y = \max\{y \in S : y < x\}$
  if *y jest zdefiniowane* **then**
    $p_x := p_x + 1$
    $p_y := p_y + 1$
  $S = S + \{A[i]\}$
