Podatkovne strukture in algoritmi 1: Izpit 1

20. Januar, 2022

Čas reševanja je 120 minut. Pojasnite vse odgovore.

Sedež (2.02)

1	
2	
3	
4	
Σ	

Ime in priimek

Vpisna številka

1. naloga (20 points)

Kuhar peče palačinke in jih odlaga na kup. Ker ni najbolj natančen so vse palačinke različnih velikosti. Na koncu želi palačinke postreči v urejenem vrstnem redu, torej od največje spodaj do najmanjše zgoraj. Na voljo ima le lopatko s katero lahko dvigne zgornjih k palačink in jih obrne. Tej operaciji bomo rekli **Obrni**(k).

a) Opiši algoritem, ki uredi kup n palačink ter uporablja le operacijo **Obrni**(k). Algoritem naj uporabi $O(n)$ **Obrni**(k) operacij. Kolikšna je celotna časovna zahtevnost algoritma?

Najdemo indeks i največje palačinke. Z obrni(i) jo spravimo na vrh. Potem z obrni(n) spravimo največjo na dno, Nadaljujemo z drugo največjo.

⇒ 2 × obrni za vsako palačino
$O(n)$ obrni operacij.

Na vsakem koraku iščemo max, zato
celotna zahtevnost $O(n) \cdot O(\text{obrni})$

b) Kuhar zažge eno stran vsake palačinke. Želi jih postreči, tako da so vse zažgane strani obrnjene navzdol. Modificiraj algoritem iz prejšnje točke, kjer morajo biti vse palačinke z zažgano stranjo obrnjene navzdol.

Ko je na vrhu po potrebi kličemo obrni(1).

c) Za vsak $n \geq 5$ opiši stolp z n palačinkami, kjer v vsakem primeru potrebujemo $\Omega(n)$ **Obrni**(k) operacij.

IDEJA: če sta 2 sosednji palačinki v stolpu, ki nista skupaj v urejenem stolpu moramo vsaj 1x dat lopatico med njiju!

n sod:

Stolp: $(2, 4, \ldots, n, 1, 3, \ldots, n-1)$

Podobno za n l.h.

2. naloga (20 points)

Na divjem zahodu se n kavbojcev postavi v vrsto en zraven drugega. Vsak ima po dve pištoli in vsako usmeri v eno stran (skrajno levi ter skrajno desni imata le po eno pištolo, ki jo usmerita proti sosedu). Ko zazvoni zvonik v bližnji cerkvi vsi istočasno ustrelijo. Vsak kavbojec zadane prvega višjega kavbojca na desni in levi strani. Vsak kavbojec, ki je zadet umre in pade na tla.

a) Razvij algoritem tipa deli in vladaj, ki kot vhod sprejme višine kavbojcev V[1,...,n] in vrne 2 seznama L[1,...,n], D[1,...,n] z levimi in desnimi tarčami za vsakega kavbojca. Algoritem naj deluje v $O(n \log n)$, kar moraš tudi dokazati.

IDEJA: Podobno kot Merge sort.
Delimo glede na srednjega kavbojca. Levi del bo imel leve tarče že pravilne, desni del pa desne.
Ko združujemo je potrebno torej levemu delu nastavit manjkajoče desne tarče (in desnemu delu manjkajoče leve tarče)
To storimo, tako da se 1x zapeljemo čez dva seznama, kar vas stane $O(n)$.

$\Rightarrow T(n) = 2T(n/2) + O(n)$, po glavnem izreku torej $T(n) = O(n \log n)$

b) Pokaži, da umre vsaj $\lfloor \frac{n}{2} \rfloor$ kavbojcev. Torej rezultat prejšnjega algoritma vrne vsaj $\lfloor \frac{n}{2} \rfloor$ različnih vrednosti.

Pogledamo 2 sosednja kavbojca:

$\rightsquigarrow$ vsaj eden izmed njiju bo tarča drugega.

c) (Bonus 8 točk) Razvij algoritem kot v točki a), ki deluje v $O(n)$.
Namig: Sklad.

Za leve tarče:
- Dodajamo na sklad dokler višine padajo.
Ko imamo element, ki je višji kot zgornji element v skladu, vzemamo iz sklada elemente dokler so manjši. Vsi te imajo tarčo element, ki ga dodajamo. Nato element dodamo.
Vsak element je enkrat dan na sklad.

Enako za desne točke $\Rightarrow 2n = O(n)$.

3. naloga (20 točk)

Sinonim (ali sopomenka) je beseda, ki ima enak pomen kot kakšna druga beseda. Npr.

- ideja-zamisel
- dimenzija-razsežnost
- tlak-pritisk

Sinonime imamo podane kot seznam parov besed $S = [(s_1, t_1), \ldots, (s_n, t_n)]$. Prav tako imamo podan seznam poizvedb, ki je tudi podan kot seznam parov besed $P = [(p_1, q_1), \ldots, (p_m, q_m)]$. Vaša naloga je, da sestavite podatkovno strukturo, ki bo omogočala dodajanje sinonimov in bo znala odgovarjati na vprašanja oziroma poizvedbe: *Ali sta besedi p in q sinonima*?

Pozor: sinonimi so tranzitivni v smislu: če sta $(a, b) \in S$ in $(b, c) \in S$ sinonima sta potem sinonima tudi (a, c). Ni pa nujno, da se bo par (a, c) nahajal v seznamu S.

a) Razvij zgoraj opisano podatkovno strukturo in v njo najprej dodaj vse sinonime v $O(n)$ časa. Nato odgovori na vsako poizvedbo v P v konstantnem času. Torej celoten čas izvajanja naj bo $O(n+m)$.

Besede so vozlišča, sinonimi so povezave.
Naredimo graf in z BFS/DFS določimo povezane komponente.
(neusmerjen)
Shranimo slovar, kjer je ključ beseda in vrednost komponenta, ki ji pripada.
N poizvedbe odgovarjamo, tako da preverimo če dve besedi pripadata isti povezani komponenti.
Čas: $O(n)$ (graf) $+ O(m)$ (poizvedbe)

b) Recimo, da sinonime in poizvedbe dobivamo kot tok podatkov. Razvij podatkovno strukturo, ki sproti dodaja nove sinonime in odgovarja na poizvedbe. Na vsako poizvedbo odgovori le glede na sinonime, ki so bili dodani do takrat. Skupen čas naj bo skoraj linearen v n in m (velikost toka).

Sedaj uporabimo disjoint set podatkovno strukturo.
Sinonime dodajamo z union(s_i, t_i) na poizvedbe pa odgovarjamo z find(p_i) = find(q_i).

Čas: $O(n \log^* n) + O(m \log^* n) =$

$= O(\log^*(n+m))$

4. naloga (20 točk)

Naj bo S množica n naravnih števil. Razvij algoritem, ki preveri, če se da S zapisati kot disjunktno unijo dveh množic z enako vsoto. Kakšna je časovna zahtevnost? Posploši algoritem na k množic. Kakšna je časovna zahtevnost algoritma za k množic?

Namig: Če rešitev obstaja, v kateri množici se lahko nahaja zadnji element v S?

$S = [s_1, \ldots, s_n]$; $\text{sum}S = \sum_{i=1}^{n} s_i$

Definiramo: $f(n, A, B)$ = Ali lahko elemente $\{s_1, \ldots, s_n\}$ razdelimo v 2 množici z vsoto A in B

Rekurzivna zveza:

n-ti element damo v prvo množico

$$f(n, A, B) = f(n-1, A - s_n, B) \text{ OR } f(n-1, A, B - s_n)$$

n-ti element damo v drugo množico

robni pogoji:

$f(0,0,0) = \text{True}$

$f(a,b,c) = \text{False}$, če je katera koli $a,b,c < 0$

Kličemo: $f(n, \text{sum}S/2, \text{sum}S/2)$

Časovna zahtevnost: $O(n \cdot \text{sum}S^2)$

Opomba: Da se še na več različnov!
(0/1 nahrbtnik, ...)

Za splošen k:

$$f(n, A_1, \dots, A_k) = f(n-1, A_1 - s_n, \dots, A_k) \text{ OR } \dots \text{ OR } f(n-1, A_1, \dots, A_k - s_n)$$

za vsako stvar k delu!

čas: $O(n \cdot k \cdot \text{sums}^k)$

Tisti, ki ste delali z nahrbtnikom ste imeli tu težavo. Rekli ste:

kličemo nahrbtnik($\frac{\text{sums}}{k}$) in nato nadaljujete (k-1 krat)

Ampak to ni ok:

IDEJA zakaj ne: Prvi nahrbtnik vam lahko najde množico z vsoto $\frac{\text{sums}}{k}$, ki pa potem onemogoča rešitve za preostalih k-1 množic.

Primer: $S = \{1, 2, 3, 4, 5, 6\}$ s_1 s_2 s_3

Deliteu na 3 množice obstajajo: $\{4,3\}, \{5,2\}, \{6,1\}$

Sedaj iz S_1 vzamemo 3 in jo zamenjamo s 2 in 1.

$\Rightarrow S_1 = \{4, 1, 2\}$, ostane $\{3, 5, 6\}$.

Tega nikakor ne moremo razdeliti na 2 množici z vsoto 7.