Podatkovne strukture in algoritmi 1: 1. izpit

21. 1. 2021

Čas pisanja je 120 minut. Možno je doseči 80 točk. Veliko uspeha!

1. naloga (20 točk)

Spomni se union-find algoritma z rangom in kompresijo poti, pri čemer v primeru unije elementov z enakim rangom za starša naredimo tistega z nižjo številko.

a) (10) Imejmo 8 elementov, $[0, 1, 2, 3, 4, 5, 6, 7]$, ki naj bodo vsak v svoji množici z rangom 1. Izvedi naslednje operacije:

`union(1, 7), union(2, 3), union(3, 4), union(5, 1), union(7, 4)`

parent								
op \ el	0	1	2	3	4	5	6	7
0.								
1.								
2.								
3.								
4.								
5.								

rank								
op \ el	0	1	2	3	4	5	6	7
0.								
1.								
2.								
3.								
4.								
5.								

b) (10) Posamezna operacija `find` ali `union` lahko traja precej dolgo, toda potem so naslednje precej cenejše. Kakšna je časovna zahtevnost m zaporednih operacij na strukturi z n elementi? Konstruiraj primer zaporedja operacij, ki začnejo iz n singletonov, pri katerem vsaj ena izmed operacij traja $\Omega(\log n)$ časa. Odgovore ustrezno utemelji.

Rešitev:

parent								
op \ el	0	1	2	3	4	5	6	7
0.	0	1	2	3	4	5	6	7
1.	0	1	2	3	4	5	6	1
2.	0	1	2	2	4	5	6	1
3.	0	1	2	2	2	5	6	1
4.	0	1	2	2	2	1	6	1
5.	0	1	1	2	2	1	6	1

rank								
op \ el	0	1	2	3	4	5	6	7
0.	1	1	1	1	1	1	1	1
1.	1	2	1	1	1	1	1	1
2.	1	2	2	1	1	1	1	1
3.	1	2	2	1	1	1	1	1
4.	1	2	2	1	1	1	1	1
5.	1	3	2	1	1	1	1	1

Časovna zahtevnost zaporedja m operacij je $O(m \log^* n)$. (Pravilna je tudi boljša meja, z $\alpha(n)$.)
Ena možna rešitev b): Denimo, da imamo 2^k elementov, ki jih združujemo v polno dvojiško drevo. Za vsak sod i združimo i in $i + 1$, nato za vsak i, deljiv s 4, združimo i in $i + 2$, za vsak i, deljiv z 8, združimo i in $i + 4$. Tako dobimo (sicer malo neuravnoteženo) drevo z globino k. Nato poklicemo find za element $2^k - 1$, kar traja sorazmerno z globino drevesa, torej vsaj $\Omega(k)$. Razlagi pripada se slikica za npr. 8 elementov.

2. naloga (20 točk)

Reši problem najširše poti v grafu. Dan je graf G z utežmi $w\colon E(G) \to \mathbb{R}$ in vozliščema s in t. Iščemo st-pot, pri kateri je najmanjša povezava na poti čim večja.

a) (5) Pokaži, da najširša pot in najkrajša pot v grafu nista nujno enaki.

b) (7) Naj bo G neusmerjen. S pomočjo konstrukcije vpetega drevesa reši problem najširše poti. Utemelji pravilnost svojega algoritma. Analiziraj časovno in prostorsko zahtevnost razvitega algoritma.

c) (8) Naj bo G usmerjen. Spremeni Dijsktrov algoritem, tako da bo iskal najširšo pot. Utemelji pravilnost svojega algoritma. Analiziraj časovno in prostorsko zahtevnost razvitega algoritma.

Rešitev:

Predpostavimo, da sta s in t v isti povezani komponenti, sicer je odgovor nedefiniran.
a) Zastonj točke, narišete praktično katerikoli graf, recimo pot na n vozliščih z utežmi 1. Najširša pot med začetkom in koncem je 1, dolžina pa je $n - 1$.

b) Algoritem: Najdemo maksimalno vpeto drevo. V tem drevesu obstaja natanko ena pot med s in t, to je naš odgovor. Časovna in prostorska zahtevnost sta odvisni od algoritma, ki ga uporabimo za iskanje vpetega drevesa: pri Kruskalu je časovna $O((V + E) \log E)$, prostorska $O(V + E)$, iskanje poti po drevesu pa je enostaven BFS, ki ima manjšo časovno in prostorsko zahtevnost od Kruskala, tako da ni pomemben.
Pravilnost: podobno kot pri dokazu pravilnosti za maksimalno vpeto drevo. Denimo, da imamo neko drugo st pot P' z boljšim ozkim grlom. Ta pot mora zaobiti trenutni minimum na poti P, ki je prišla iz maksimalnega vpetega drevesa. Toda, P in P' se vsaj dvakrat stakneta in tvorita cikel nek cikel C, ki vsebuje tudi povezavo, ki je ozko grlo na P. Sedaj lahko konstruiramo novo boljše vpeto drevo, tako da gremo po drugi strani cikla C in izpustimo najslabšo povezavo. To je protislovje s tem da je bilo začetno vpeto drevo maksimalno.
Na kratko: če bi obstajala boljša pot, bi lahko konstruirali še boljše vpeto drevo.

c) Dijkstrov algoritem spremenimo tako, da namesto razdalj od začetka hranimo ozka grla, torej najožjo povezavo, ki smo jo do sedaj srečali na poti do vsakega vozlišča. Vrsto tudi spremenimo v maksimalno prioritetno vrsto namesto minimalne. Ko smo na vozlišču u z razdaljo d od začetka in gledamo soseda v z utežjo povezave uv enako w, v vrsto namesto $d + w$ dodamo $\min(d, w)$, pri čemer d predstavlja najožjo povezavo do sedaj.
Časovna in prostorska zahtevnost je enaka kot običajna Dijsktra, torej $O((V + E) \log E)$ (lahko se dokaže, da je celo $O(V + E)$, ker so elementi v vrsti monotono padajoči).
Pravilnost: podobno kot utemeljitev pri Dijkstri. Najprej utemeljivom, da sprememba res računa najkrajšo pot: če je d ozko grlo poti od s do u, in w povezava med u in v, je ozko grlo poti med s in v enako $\min(d, w)$. Dijkstrov algoritem tako obravnava vse možne poti iz s po padajoči vrednosti ozkega grla. Prvič, ko pridemo do nekega vozlišča, je iskana pot najširša.
Krajša utemeljitev: $d \geq \min(d, v)$ za vsak $v, d \in \mathbb{R}$. Operacija kombiniranja uteži je monotona glede na urejenost kopice in posplošen algoritem za iskanje najkrajših poti deluje pravilno.

3. naloga (20 točk)

Dan je seznam n celih števil. Izračunati želimo večino seznama, tj. element, ki se v seznamu pojavi več kot $\frac{n}{2}$-krat, ali pa sporočiti, da tak element ne obstaja. Pozor: večinski element je vedno najpogostejši, ni pa vsak najpogostejši element večinski!

a) (5) Naj bodo števila v seznamu omejena na interval $[0, M-1]$, za nek (manjši) konstanten $M \in \mathbb{N}$. Zapiši algoritem in analiziraj časovno in prostorsko zahtevnost v odvisnosti od M in n.

b) (5) Sprostimo predpostavke glede lastnosti elementov. Elementi niso nujno števila, ampak splošni objekti, ki pa jih znamo med seboj primerjati glede na neko linearno urejenost $\leq$. Razvij $O(n \log n)$ algoritem za reševanje problema, ki porabi $O(1)$ dodatnega prostora.

c) (5) Še bolj sprostimo predpostavke glede lastnosti elementov. Denimo, da znamo primerjati objekte le za enakost. Razvij $O(n^2)$ algoritem za reševanje problema, ki ne porabi nič dodatnega prostora.

d) (5) Ovrži ali dokaži pravilnost naslednjega algoritma, ki poišče večinski element (nepraznega) seznama a.

```
m = a[0]
i = 1
for x in a[1:]:
    if m == x:
        i += 1
    else:
        i -= 1
        if i == 0:
            i = 1
            m = x

c = 0
for x in a:
    if x == m:
        c += 1
if c > len(a)/2:
    return m
else:
    return None
```

Rešitev:

a) Naredimo frekvenčno tabelo, pogledamo, ali se kakšen element pojavi več kot $n/2$-krat. Podobno kot counting sort, časovna zahtevnost $O(M + n)$, prostorska $O(M)$.

b) Elemente uredimo (z in-place sortom, npr. heapsort). Nato gremo od leve proti desni in štejemo koliko zaporednih enakih elementov imamo. Urejanje stane $O(n \log n)$, kasneje pa še $O(n)$ za procesiranje. Oboje porabi $O(1)$ prostora.

c) Za vsak element gremo čez cel seznam in preštejemo, koliko mu jih je enakih. $O(n^2)$ časa in $O(1)$ prostora.

d) Algoritem je pravilen in se imenuje Boyer-Moore majority element.
Najlažje intuitivno razumevanje algoritma je, da pogledamo vse pojavitve večinskega elementa in jih poskusimo "pokrajšati" z drugimi elementi.

Če je trenutni m ne-večinski element, potem ga pravilen večinski element "pokrajša" in se števec zmanjša. Če je trenutni m večinski element, potem ga bo vsaka druga element "pokrajšal" in se bo njegov števec zmanjšal. Vsaka pojavitev večinskega elementa, ki ni pokrajšana, šteje 1 h končnemu rezultatu, in ker se večinski element pojavi več kot $n/2$-krat, je nemogoče pokrajšati vse pojavitve. Velja omeniti, da se lahko pokrajšajo tudi pojavitve drugih elementov med sabo, vendar to samo še bolj koristi pojavitvam pravega elementa.

4. naloga (20 točk)

Dan je seznam a z n elementi, ki jih lahko primerjamo med seboj z neko linearno urejenostjo $\leq$. Preveriti želimo, ali so vsi elementi različni med seboj, pri čemer sprejmemo odgovor oblike "Da" ali pa "Ne, elementa na indeksih i in j sta enaka."

a) (5) Naj bo a urejen po velikosti. Dokaži, da vsak algoritem, ki pravilno reši zgornji problem, nujno naredi vsaj $n-1$ primerjav.
Namig: protislovje.

b) (5) Dokazano je, da vsak algoritem za reševanje zgornjega problema, ko je seznam neurejen, potrebuje vsaj $\Omega(n \log n)$ časa. Dokaži, da obstaja algoritem, ki potrebuje $\Theta(n \log n)$ časa.

c) (10) Naj bodo v seznamu nizi dolžine največ w iz malih črk angleške abecede. Razvij $O(nw)$ algoritem za reševanje problema in komentiraj, zakaj lahko tak algoritem obstaja.

Rešitev:

a) Ker je a urejen, imamo $n-1$ parov indeksov, kjer se elementi lahko razlikujejo. Denimo, da imamo pravilen algoritem A, ki naredi $n-2$ primerjav. Poženimo ga na seznamu $[0, 2, 4, \ldots, 2n]$. Ker naredi $n-2$ primerjav, gotovo obstajata dva zaporedna elementa, ki ju ni primerjal, a_i in a_{i+1}. Če nastavimo $a_{i+1} = a_i = 2i$, in A poženemo še enkrat, potem algoritem vrne enak odgovor kot prej, kar je narobe. Algoritem torej ni pravilen, kar vodi v protislovje.

b) Seznam uredimo in ga linearno pregledamo.

c) Uporabimo radix sort za stringe, kot smo ga delali na vajah (bucket sort po prvi črki, nato po drugi črki, ...), ki ima časovno zahtevnost $O(nw)$. Nato zopet linearno pregledamo. Tak algoritem ni v protislovju z a) točko, ker ne uporablja primerjav.