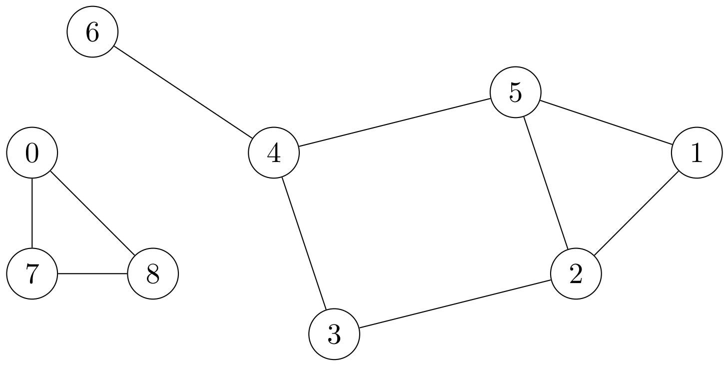
Podatkovne strukture in algoritmi 1: predizpit

8. 1. 2018

Čas pisanja je 120 minut. Možno je doseči 100 točk. Veliko uspeha!

1. naloga (25 točk)

Dan je graf G, prikazan na sliki spodaj. Vozlišča vedno obravnavamo po naraščajočih oznakah.



a) (5) Predstavi graf s seznamom sosedov ($G = [[\ldots], [\ldots], \ldots]$).

b) (7) Nariši DFS gozd grafa G. Kakšna je časovna zahtevnost DFS?

c) (7) Nariši BFS gozd grafa G. Kakšna je časovna zahtevnost BFS?

d) (6) Za vsak n konstruiraj primer povezanega grafa na n vozliščih, pri katerem BFS in DFS obiščeta vozlišča v enakem vrstnem redu.

Rešitev

a) $G = [[7, 8], [2, 5], [1, 5], [2, 4], [4, 5, 6], [1, 2, 5], [4], [0, 8], [0, 7]]$

b) $0 - 7 - 8$, $1 - 2 - 3 - 4 - 5$, $O(V + E)$

(pod 4: | 6)

c) $0 - 8$, $1 - 5 - 4 - 6$, $O(V + E)$

(pod 0: | 7; pod 1: | 3, $2 - 3$)

d) Npr. pot na n vozliščih oštevilčenih od leve proti desni; zvezda na n vozliščih, kjer začnemo v centru; zvezda, kjer začnemo v centru in kjer je zadnji krak pot...

2. naloga (25 točk)

Naj bo G omrežje brez zank z utežmi $w\colon E(G) \to \mathbb{R}$. Pazi, uteži so lahko tudi negativne.

a) (15) Razvij algoritem, ki v G najde najkrajšo pot iz kvečjemu k povezav od izbranega vozlišča s do vseh ostalih vozlišč v času $O(kE)$, če taka pot obstaja.
Namig: začni razmišljati pri $k = 1$ in $k = 2$ ter nadaljuj induktivno.

b) (5) Razvij algoritem za iskanje najkrajših poti iz vozlišča s do vseh ostalih vozlišč s časovno zahtevnostjo $O(VE)$. Utemelji njegovo pravilnost.

c) (5) Spremeni algoritem tako, da hkrati preveri še, ali lahko iz s dosežemo kakšen negativen cikel (in torej najkrajša pot ni definirana)?

Rešitev

a) Za $k = 1$ so najkrajše poti kar povezave do sosedov. Pri $k = 2$ imamo dve možnosti: ali je najkrajša pot do nekega vozlišča dolžine 1 ali pa je dolžine dva, v slednjem primeru je sestavljena iz poti dolžine 1 in povezave. Nadaljujemo enako, najkrajša pot dolžine k od s do t je lahko sestavljena iz najkrajše poti dolžine $k-1$ od s do u in povezave ut, za vse sosede u od t.

Hranimo si tabelo $\texttt{dist}(u)$, ki je na začetku nastavljena na ∞, razen $d(s) = 0$ in predstavlja razdaljo od s do u. Za rešitev k-krat ponovimo naslednji postopek: Gremo čez vse povezave $uv \in E$. Za vsako povezavo uv preverimo, ali je $\texttt{dist}(u)+w(uv) < \texttt{dist}(s)$, in če je, posodobimo $\texttt{dist}(s)$ na to vrednost. Na i-ti iteracije pregledamo vse možne najkrajše poti dolžine i, torej na koncu res najdemo najkrajše poti od s, ki so krajše od ali dolge k. Časovna zahtevnost je očitno dosežena, saj imamo dve neodvisni gnezdeni zanki do k in do $|E|$. V psevdokodi:

```
repeat k times:
  for each (u, v) with weight w in E:
    if d[u] + w < d[v]:
      d[v] := d[u] + w
      prev[v] := u
```

b) Vsaka pot je gotovo dolga manj ali enako $|V| - 1$ povezav. Poženemo algoritem iz točke a) za $k = |V| - 1$. Časovna zahtevnost sledi iz a).
c) Poženemo algoritem iz točke a) še enkrat. V $|V| - 1$ iteracijah je že našel najkrajše poti, zato se ne bi smelo nič spremeniti. Če se katera od vrednosti spremeni, imamo negativen cikel.

Ta algoritem je znan kot Bellman-Fordov algoritem.

3. naloga (25 točk)

Spomni se implementacije vrste, ki lahko vsebuje največ N elementov, s tabelo dolžine N, pri čemer hranimo indeks začetka in dolžino vrste. V praksi zgornje meje števila elementov ne poznamo, zato moramo, ko bi število elementov v vrsti preseglo velikost tabele, narediti novo večjo tabelo in vanjo prekopirati trenutno vrsto ter vstaviti nov element. Analiziraj dva načina povečevanja tabele.

a) (10) Izberimo fiksen $k \in \mathbb{N}$, $k \geq 1$. Začnimo s tabelo velikosti k. Nato izvedemo $m \gg k$ `push` operacij, pri čemer, ko zmanjka prostora, alociramo novo tabelo, ki ima za k elementov več prostora in vanjo skopiramo elemente vrste in dodamo nov element. Koliko je skupna časovna zahtevnost teh m operacij in koliko je povprečna zahtevnost posamezne operacije?

b) (10) Naredimo enak test kot v prejšnji podnalogi, le da tokrat, ko nam v tabeli velikosti n zmanjka prostora, naredimo novo tabelo velikosti $\lceil \alpha n \rceil$ za nek $\alpha > 1$. Koliko je sedaj skupna časovna zahtevnost teh m operacij in koliko je povprečna zahtevnost posamezne `push` operacije?

c) (5) Kateri pristop bi izbral? Komentiraj tudi razlike v prostorski zahtevnosti. Predlagaj strategijo za zmanjševanje tabele, ko naredimo veliko `pop` operacij, in oceni njeno povprečno časovno zahtevnost.

Rešitev

a) Na vsakih k operacij bomo kopirali celo tabelo, ki bo ob i-tem kopiranju velika ik. Teh kopiranj bo $d := \lfloor \frac{m}{k} \rfloor$. Ostale operacije so konstantne. Skupna zahtevnost je tako

$$T = \Theta(m) + \sum_{i=1}^{d} ik = \Theta\left(m + k\frac{d(d+1)}{2}\right) = \Theta(m + m^2/k) = \Theta(m^2/k).$$

Povprečna zahtevnost je $\Theta(m/k)$, kar je linearno v m.
b) Tabele so velikosti k, $\lceil \alpha k \rceil$, $\lceil \alpha \lceil \alpha k \rceil \rceil$, ... V novo tabelo kopiramo, ko stara tabela doseže približno $\alpha^i k$ elementov, dokler ni $\alpha^i k > m$, torej $i < \log_\alpha(m/k)$. Skupna zahtevnost je tako

$$T = \Theta(m) + \sum_{i=0}^{\lfloor \log_\alpha(m/k) \rfloor} \alpha^i k = \Theta(m) + k\left(\frac{\alpha^{\lfloor \log_\alpha(m/k) \rfloor + 1} - 1}{\alpha - 1}\right) =$$
$$= \Theta(m) + \Theta\left(k\frac{m/k - 1}{\alpha - 1}\right) = \Theta(m) + \Theta\left(\frac{m-k}{\alpha-1}\right) = \Theta(m)$$

Povprečna časovna zahtevnost je $O(1)$, saj je α konstanta.
c) Druga strategija je v splošnem boljša, saj ima `push` operacija konstantno povprečno časovno zahtevnost. Le v primeru, ko bi bili seznami zelo kratki, bi uporabili prvo strategijo. Je pa druga lahko prostorsko potratna, saj je lahko naše maksimalno število elementov tako, da ravno presežemo mejo in ostane npr. polovica nove tabele neuporabljene.

Za zmanjševanje bi uporabili analogno strategijo kot v drugi točki, ko je za nek $\beta \in (0, 1)$ tabela manjša kot $\lfloor \beta n \rfloor$, prestavimo vrsto v manjšo tabelo. Pametno je, da ta tabela ni velika točno $\lfloor \beta n \rfloor$, saj nam v tem primeru naslednja `push` operacija spet sproži realokacijo. Zato tabelo naredimo veliko npr. $\lfloor \frac{1+\beta}{2} n \rfloor$. Analiza časovne zahtevnost je enaka, le da vrsto seštejemo v drugo smer in z drugim faktorjem, in povprečna zahtevnost je zopet $O(1)$.

Zanimivost: Python uporablja strategijo z $n \mapsto n + n \gg 3$, kar je $\alpha \approx 1.125$, z dodatnimi specializacijami za sezname, krajše od 20 elementov.

4. naloga (25 točk)

Podan imamo stolpični diagram iz n stolpcev. Stolpec i je pravokotnik od koordinate $(i-1,0)$ do (i,h_i), za $i=1,\ldots,n$ in neke višine $h_i \in \mathbb{N}$, $h_i \geq 0$. Izračunati želimo ploščino največjega pravokotnika, ki ga lahko včrtamo na območju, ki ga pokrivajo stolpci.[1]

a) (10) Razvij algoritem, ki izračuna odgovor. Uporabi metodo deli in vladaj, pri čemer deli glede na najnižji stolpec. Predpostaviti smeš, da znamo z $O(n)$ predprocesiranja minimum strnjenega podseznama od i do j najti v $O(\log n)$ časa.

b) (10) Definirajmo količino $r_i \in [0, n-i]$, ki predstavlja "doseg v desno tik izpod vrha i-tega stolpca", definiran kot $r_i = \min\{j-i-1;\ h_j < h_i \text{ in } j > i\}$ oz. $r_i = n-i$, če je množica prazna. Razvij $O(n)$ algoritem za izračun vseh r_i. Utemelji, da algoritem res dosega iskano časovno zahtevnost.
Namig: sklad.

c) (5) Analogno definiramo in izračunamo "doseg v levo". S pomočjo obeh dosegov v času $O(n)$ reši začetni problem.

Rešitev

a) Zapišemo zvezo

$$\operatorname{maxp}(s,i,j) = \begin{cases} 0, & i \geq j \\ \max(m\cdot(j-i), \operatorname{maxp}(s,i,m), \operatorname{maxp}(s,m+1,j), \text{kjer } m = \operatorname*{argmin}_{i\leq k<j} s, & \text{sicer} \end{cases},$$

ki prestavlja maksimalno ploščino na $s[i:j]$. Pokličemo z $\operatorname{maxp}(s,0,len(s))$ in dobimo rezultat. Časovna zahtevnost: v najslabšem primeru je minimum vedno na robu in dobimo zahtevnost:

$$T(n) \leq O(n) + \sum_{i=1}^{n} (\log n + 2) = O(n \log n).$$

b) Procesiramo seznam od leve proti desni. Dokler stolpci padajo, jih dajemo na sklad. Ko pridemo do stolpca i, ki je večji kot zadnji na skladu, iz sklada odstranimo elemente in pri j-tem nastavimo $r_j = i-j-1$. Nato damo i-tega na sklad in nadaljujemo naprej do konca seznama. Preostalim elementom na skladu nastavimo $r_i = n-i$. Elementi na skladu so vedno padajoče urejeni. Časovna zahtevnost je $O(n)$, saj kljub temu da lahko na neki iteraciji zunanje zanke pregledamo skoraj cel sklad, vemo, da bo vsak element dan in vzet is sklada natanko enkrat in bo skupen čas dela s skladom reda $2n$. Ostale operacije so konstantne znotraj for zanke do n, skupna zahtevnost je tako $O(n)$.

c) Označimo z ℓ_i in r_i dosega v levo in desno. Vsak izmed n stolpcev s svojo višino in obema dosegoma definira nek pravokotnik, preprosto pogledamo kateri je največji. Odgovor je `max((r[i]+l[i]+1)*h[i]) for i in range(n))`. Celoten izračun traja trikrat $O(n)$ časa, kar je $O(n)$.

[1] Ta naloga se pogosto pojavlja na programerskih intervjujih za službo.