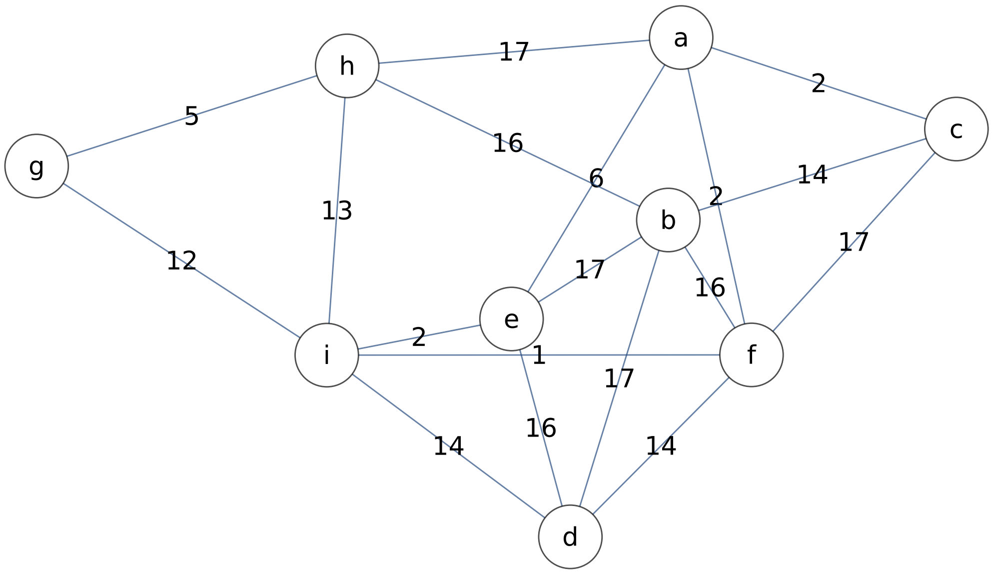
Podatkovne strukture in algoritmi 1: 1. izpit

4. 6. 2020

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

1. naloga (20 točk)

S spodnjo sliko je podan graf G.



Najdi minimalno vpeto drevo s pomočjo Primovega algoritma, pri čemer začnemo pri vozlišču "a" in sosede vedno obravnavamo po abecedi.

Algoritem izvajaj po iteracijah. Pri vsaki iteraciji algoritma simuliraj odvzemanje in dodajanje povezav v kopico. Za vsako povezavo, ki jo vzameš iz kopice, utemelji, zakaj je v končnem drevesu ali zakaj ni.

Rešitev: (Kopice ta rešitev ne simulira)
imamo vozlišče "a", označimo kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ac, 2), (af, 2), (ae, 6), (ah, 17)]
pop: [(ac, 2)], označimo "c" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(af, 2), (ae, 6), (ah, 17), (cb, 14), (cf, 17)] pop: [(af, 2)], označimo "f" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ae, 6), (ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (fi, 1)] pop: [(fi, 1)], označimo "i" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ae, 6), (ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (id, 14), (ie, 2), (ig, 12), (ih, 13)]
pop: [(ie, 2)], označimo "e" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ae, 6), (ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (id, 14), (ig, 12), (ih, 13), (eb, 17), (ed, 16)]
pop: [(ae, 6)], ne dodamo povezave, ker sta "a" in "e" že obiskana
vrsta: [(ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (id, 14), (ig, 12), (ih, 13), (eb, 17), (ed, 16)]
pop: [(ig, 12)], označimo "g" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (id, 14), (ih, 13), (eb, 17), (ed, 16), (gh, 5)]
pop: [(gh, 5)], označimo "h" kot visited, dodamo sosednje povezave do neobiskanih vozlišč
vrsta: [(ah, 17), (cb, 14), (cf, 17), (fb, 16), (fd, 14), (id, 14), (ih, 13), (eb, 17), (ed, 16), (hb, 16)]
pop: [(c, b)], označimo "b" kot visited, ni sosednjih neobiskanih povezav
pop: [(fd, 14)], ozačimo "d" kot visited. Sedaj smo obiskali vsa vozlišča, tako da lahko zaključimo.
Drevo vsebuje povezave: ac, af, fi, ie, ig, gh, cb, fd

2. naloga (20 točk)

Označimo s $\mathbb{P}$ množico vseh praštevil. Predpostavili bomo, da vse aritmetične operacije na številih trajajo konstantno časa.

a) (5) Napiši algoritem za preverjanje ali je naravno število n praštevilo ($n \in \mathbb{P}$), ki teče v $O(\sqrt{n})$ časa.

b) (5) Dokaži, da z algoritmom iz prejšnje naloge traja $\Theta(n^{\frac{3}{2}})$ časa, da za vsa števila od 1 do n ugotovimo, ali so praštevila.

c) (5) Denimo, da naprej skonstruiramo Eratostenovo rešeto, nato pa za vsako število preverimo, ali je praštevilo. Koliko časa traja računanje rešeta in nato posamezna poizvedba?
V spomin: Eratostenovo rešeto naredimo tako, da vsa števila od 2 do n napišemo v tabelo, nato pa po vrsti črtamo večkratnike do sedaj neprečrtanih števil. Števila, ki ostanejo so praštevila.
Pomoč: Mertensov drugi izrek pravi:

$$\lim_{n\to\infty}\left(\sum_{\substack{p\leq n\\ p\in\mathbb{P}}}\frac{1}{p}-\ln\ln n-M\right)=0,$$

kjer je $M \approx 0.261497$ Meissel–Mertensova konstanta.

d) (5) Denimo, da si v rešetu za vsako število zapomnimo, katero število je bilo prvo, ki je bilo krivo, da smo ga prečrtali (če pa ga nismo, si zapomnimo neko drugo vrednost, npr. -1). Za rešeto od 2 do 27 bi dobili npr.

$$[-1, -1, 2, -1, 2, -1, 2, 3, 2, -1, 2, -1, 2, 3, 2, -1, 2, -1, 2, 3, 2, -1, 2, 5, 2, 3]$$

S pomočjo že zgrajene take strukture za vsa števila do nekega n_0, izpelji algoritem za razcepljanje števil manjših od n_0 na prafaktorje. Pokaži, da traja $O(\log n)$ časa, da razcepimo število $n \leq n_0$. Konstruiraj družino števil, ki realizira to časovno zahtevnost.

Rešitev:

a) Primer implementacije:

```
def is_prime(n):
    i = 2
    while i*i <= n:
        if n % i == 0:
            return False
        i += 1
    return True
```

Preverjati je potrebno samo do $\sqrt{n}$, ker so pari deliteljev simetrični okrog te vrednosti. Zanka se v najslabšem primeru, ko je $n \in P$ ali popoln kvadrat izvede $\lfloor\sqrt{n}\rfloor$-krat, vsakič je konstantno operacij, kar je $O(\sqrt{n})$.

b) Če za vsako število ponovimo enak postopek, traja vse skupaj

$$T = \sum_{i=1}^{n} \sqrt{i}$$

Lahko je pokazati da je $T = O(n\sqrt{n})$, saj velja $T \leq \sum_{i=1}^{n} \sqrt{n} = n \log n$ Za Θ moramo oceniti tudi v drugo smer. Lahko npr. s pomočjo integrala

$$T \geq \int_{i=1}^{n} \sqrt{x}\, dx = \frac{2}{3} n^{\frac{3}{2}} - \frac{2}{3}$$

ali pa z argumentom, kjer ocenimo vsako polovico seznama posebej.

V resnici za zaporedje n števil vemo, da ne morejo biti vsa praštevila, in se za veliko števil funkcija `is_prime` konča hitreje. V resnici je časovna zahtevnost enaka $\sum_{i=1}^{n} d_{min}(n)$, kjer je d_{min} najmanjši delitelj, ampak to ni bila poanta naloge...

Poizvedbe so samo vpogled v seznam in trajajo $O(1)$.

c) Za konstrukcijo rešeta moramo za vsako praštevilo $p \leq n$ prečrtati vse njegove večkratnike do n, teh je manj kot $\lfloor n/p \rfloor$. Ostali deli algoritma kot so npr. inicializacija tabele so linearni. Tako lahko ocenimo

$$T = O(n) + \sum_{\substack{p \in \mathbb{P} \\ p \leq n}} O(\lfloor n/p \rfloor) \leq O(n) + n \sum_{\substack{p \in \mathbb{P} \\ p \leq n}} O(1/p) = O(n) + O(n \log \log n) = O(n \log \log n),$$

kjer smo uporabili Mertensov izrek.

d) Za vsako število pogledamo v seznam, kaj je njegov najmanjši prafaktor. Če je to -1, smo zaključili, sicer pa ga lahko s faktorjem delimo in ponovimo enak postopek. Vsakič število vsaj razpolovimo, tako da je gotovo konec v $O(\log_2 n)$ korakih. To realizirajo števila oblike 2^k.

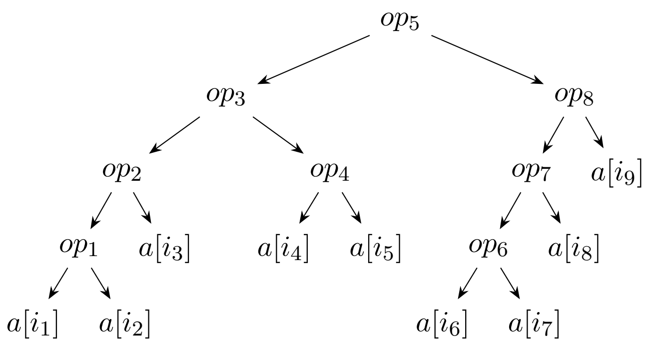
3. naloga (20 točk)

Dana je tabela $a[0\,..\,n-1]$ dolžine n, katere elementi so cela števila.

a) (5) Najdi algoritem, ki najde maksimum izraza $a[i] - a[j]$ v $O(n^2)$ časa, pri čemer mora veljati $0 \leq i < j < n$. Algoritem izboljšaj, da porabi $O(n)$ časa. Za oba algoritma utemelji pravilnost in časovno zahtevnost.

b) (10) Najdi algoritem, ki najde maksimum izraza $a[i] - a[j] + a[k] - a[l]$ v $O(n^4)$ časa, pri čemer mora veljati $0 \leq i < j < k < l < n$. Algoritem izboljšaj, da porabi $O(n)$ časa. Za oba algoritma utemelji pravilnost in časovno zahtevnost.

c) (5) Najdi algoritem, ki najde maksimum splošnega izraza iz e ($e \leq n$) elementov tabele in dvojiških operatorjev (predstavljenega z drevesom). Označimo splošne indekse z $i_1, \ldots, i_e$ in $e-1$ operatorjev z $op_1, \ldots, op_{e-1}$. Primer izraza:



V drevesu so indeksi in operatorji vedno oštevilčeni od leve proti desni. Operacije lahko izbiramo iz množice $\{+, -, \max, \min\}$.
Najdi algoritem, ki maksimizira splošen izraz pri pogoju $0 \leq i_1 < \cdots < i_e < n$ in oceni njegovo časovno zahtevnost.
Rešitev:
a) V $O(n^2)$ časa imamo dve for zanki (v ustreznih mejah), ki pogledata vse možnosti.
Ko pri nekem $a[j]$ gledamo, kaj največ lahko dosešemo, bi bilo dobro vedeti maximum seznama $a[0 : j]$. Te maximume si lahko kumulativno poračunamo vnaprej v neko tabelo $maxod[i]$ z zvezo $maxod[i] = \max(a[i], maxod[i-1])$. Nato se še enkrat zapeljemo čez tabelo in izračunamo $\min_j(maxod[j] - a[j])$.
(obstajajo tudi rešitve, ki gredo samo enkrat čez seznam brez dodatnega spomina, ampak ta se dobro posploši; ekvivalentno lahko tudi naredimo z druge strani s tabelo $minod[i] = \min(a[i : n]))$
b) V $O(n^4)$ časa zopet pogledamo vse možnosti s 4 for zankami. Hranimo 4 tabele: m_1 hrani kumulativne maksimume $a[i]$, m_2 hrani maksimume $a[i]-a[j]$, m_3 hrani maksimume $a[i]-a[j]+a[k]$, m_4 pa kumulativne maksimume celotnega izraza.

```
for (int i = n - 1; i >= 0; i--)
    m1[i] = max(m1[i+1],a[i]);
for (int i = n - 2; i >= 0; i--)
    m2[i] = max(m2[i+1], m1[i + 1] - a[i]);
for (int i = n - 3; i >= 0; i--)
    m3[i] = max(m3[i+1], m2[i + 1] + a[i]);
for (int i = n - 4; i >= 0; i--)
    m4[i] = max(m4[i+1], m3[i + 1] - a[i]);
```

c) Za vsak operator hranimo podobno tabelo kumulativnih maksimumov, $m_k[i]$, ki hrani maksimum podizraza na tabeli $a[0 : i]$, če je sam levi otrok ali pa na $a[i : n]$ če je desni otrok. V primeru da je desni otrok $-$ operatorja, hrani minimume namesto maksimumov. Tako lahko računamo maksimume od otrok do korena, za vsakega porabino n časa, za e operatorjev torej $O(ne)$. a) in b) dela sta posebna primera, ko sta drevesi verigi in lahko to naredimo z zaporednimi for zankami.

4. naloga (20 točk)

Naj bo $G = (V, E)$ utežen usmerjen graf brez negativnih ciklov z utežjo $w\colon E \to \mathbb{R}$.

a) (10) Denimo, da bi se želeli negativnih uteži znebiti tako, da definiramo novo utež $w'\colon E \to [0, \infty)$, tako da stari uteži prištejemo ustrezno velik c:

$$w'(uv) = w(uv) + c.$$

Dokaži, da ta postopek ne deluje. Konstruiraj družino grafov G_i, tako da gre $n(G_i)$ proti neskončno in da Dijkstrov algoritem za noben c ne deluje pravilno (za vsak c obstaja vsaj en par vozlišč, kjer najde napačno najkrajšo pot).

b) (10) Naj bo s poljubno vozlišče. Denimo, da smo izračunali razdalje od s do vseh drugih vozlišč. Drugače povedano, izračunano imamo funkcijo d

$$d(v) = \min \left\{ \sum_{u \in P} w(u) \ \mid \ P \text{ pot med } s \text{ in } v \right\}$$

Dokaži, da lahko uteži spremenimo tako, da bodo vse pozitivne in da bo Dijkstrov algoritem deloval pravilno (najkrajša pot v novem grafu, najdena z Dijkstro in nakrajša pot v starem grafu sovpada za vsak par vozlišč).

Rešitev:

a) Težava je, povezave z različno koraki različno kaznujemo. En možen primer družine: $s \xrightarrow{i} t \xleftarrow{-i} u$, poleg tega pa s in u povežemo z i vozlišči s povezavami teže 1.

Najkrajša pot je prek $s \to \cdots \to u \to t$ dolga $(i+1) - i = 1$. Če prištejemo poljuben c (ki mora biti večji od i), pa je zgornja pot dolga $i + c$, spodnja pa $(i+1)(c+1)$, kar je (precej) več.

b) Če bi nas zanimale samo poti od s, bi lahko vse uteži delili z dolžino najkrajše in tako enako kaznovali vse dolžine poti.

Ideja je, da definiramo $w'(uv) = w(uv) + d(u) - d(v)$. Tako vsako pot $au_1u_2 \dots u_kb$ od a do b utežimo enako: $w'(au_1u_2 \dots u_kb) = w(a) + d(a) - d(u_1) + w(p_1) + d(u_1) - d(u_2) + \cdots + w(b) + d(u_k) - d(b) = (w(a) + w(p_1) + \cdots + w(p_k) + w(b)) + d(a) - d(b)$, kar je originalna cena poti, povečana za $d(a) - d(b)$. To je neodvisno od poti in torej enako za vse poti od a do b, zato ne vpliva na pravilnost Dijkstre.

Algoritem je znan kot Johnsonov algoritem.[1]

[1] `https://en.wikipedia.org/wiki/Johnson\%27s_algorithm`