Računalništvo 1: 1. izpit

28. januar 2021

Čas reševanja je 90 minut. Veliko uspeha!

1. naloga (15 točk)

Dan je sklad `Sklad_A`, na katerem so padajoče zložena števila (največji podatek je na dnu sklada). Primer:

```
Sklad_A = dno : 658 : 128 : 67 : 55 : 32 : 11 : 8 : 3 : -5 : -22
```

Na voljo imamo še prazen sklad `Sklad_B` in prazno vrsto `Vrsta`.

Opišite postopek, kako števila iz `Sklad_A` premaknemo v `Vrsta` tako, da bodo na začetku padajoče urejena soda števila, nato pa naraščajoče urejena liha števila. Na koncu postopka naj bosta oba sklada prazna. V postopku ni dovoljeno uporabljati dodatnih podatkovnih struktur. Če postopek uporabimo na zgornjem zgledu, dobimo:

```
Vrsta = začetek : 658 : 128 : 32 : 8 : -22 : -5 : 3 : 11 : 55 : 67 : konec
```

Rešitve

Elemente premikamo z zanko `while`, ki jo ustavimo, ko je struktura prazna. Pri skladu elemente jemljemo z vrha in jih vstavljamo na vrh, medtem ko pri vrsti elemente dodajamo na začetek in jih jemljemo s konca. Če elemente vstavimo v sklad in nato poberemo, se njihov vrstni red obrne.

Najprej elemente premikamo iz `Sklad_A` v `Vrsta` in `Sklad_B`, tako da sode vstavimo v `Sklad_B` lihe pa v `Vrsta`.

```
Sklad_A = dno
Sklad_B = dno : -22 : 8 : 32 : 128 : 658
Vrsta = začetek : -5 : 3 : 11 : 55 : 67 : konec
```

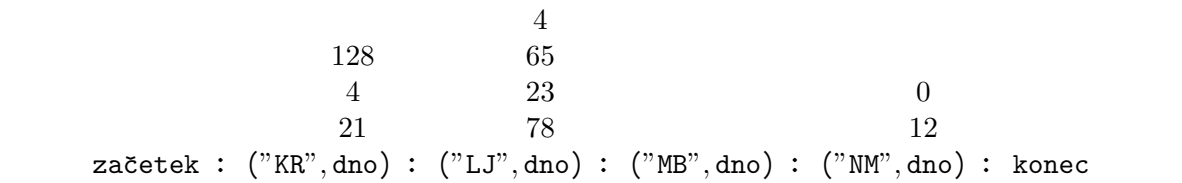
S tem smo obrnili vrstni red sodih števil (kar želimo), za liha pa moramo še poskrbeti. Liha števila pretočimo iz `Vrsta` v `Sklad A` in nato soda iz `Sklad_B` v `Vrsta`.

```
Sklad_A = dno : -5 : 3 : 11 : 55 : 67
Sklad_B = dno
Vrsta = začetek : 658 : 128 : 32 : 8 : -22 : konec
```

Liha števila je potrebno še obrnit preden jih vstavimo v vrsto, to naredimo tako, da jih iz `Sklad_A` premaknemo v `Sklad_B` in šele nato v `Vrsta`.

2. naloga (25 točk)

Potrebujemo podatkovno strukturo za pomnjenje elementov, ki pripadajo različnim kategorijam. Odločili smo se, da bomo uporabili *vrsto poimenovanih skladov*, kjer vrsta vsebuje elemente oblike `(ime, sklad)`. Zahtevamo, da je vrsta **urejena** glede na imena skladov (urejenost elementov v skladih ni pomembna). Primer:



Podrobno opišite implementacijo spodnjih metod (pazite na urejenost). Za vsako od metod določite in argumentirajte tudi časovno zahtevnost v O notaciji, kjer naj bo n število vseh elementov v vseh skladih, m število skladov v vrsti in k velikost največjega (glede na število elementov) sklada v vrsti.

(a) `dodaj_sklad(ime)` v vrsto vstavi prazen sklad z izbranim imenom. Predpostavite lahko, da `ime` še ni uporabljeno.

(b) `vstavi_na_dno(ime, x)` poskusi na *dno* sklada z imenom `ime` vstaviti element `x`, vrne pa logično vrednost, ki opisuje uspeh (vrne `False`, če sklada z imenom `ime` ni v vrst oziroma `True`, če je postopek uspel). Metoda naj ima *prostorsko* zahtevnost $O(k)$.

(c) `vsote_skladov()` vrne *novo* vrsto, ki vsebuje pare `(ime, vsota)`, kjer je `vsota` seštevek vseh elementov sklada z imenom `ime`.

(d) `preimenuj(staro_ime, novo_ime)` preimenuje izbrani sklad (če obstaja).

Vedno lahko predpostavite, da se imena skladov ne ponavljajo.

Rešitve

(a) *Ker moramo ohranjati sklade v vrsti v pravem vrstnem redu, ne smemo novega sklada vstaviti kar na konec. Tokrat opišemo prostorsko manj učinkovito rešitev (saj v točki (b) sledi učinkovitejši pristop).*

Naredimo pomožno vrsto. Z zanko elemente jemljemo iz začetka glavne vrste. Če ima odstranjen element manjše ime kot argument, odstranjen element vstavimo v pomožno vrsto. Ko prvič naletimo na element z večjim imenom, v pomožno vrsto najprej vstavimo `(ime, dno)` (poimenovan prazen sklad) in šele nato element, ki smo ga odvzeli iz vrste. Nato premaknemo še preostanek elementov v pomožno vrsto.

Če smo prišli do konca in še nismo opazili primernega mesta, prazen sklad dodamo sedaj. Nato elemente iz pomožne vrste pretočimo nazaj v izvorno.

Vse sklade smo pretočili iz ene vrste v drugo in nazaj, gledali pa smo le imena, torej je časovna zahtevnost $O(m)$.

(b) Da obdržimo prostorsko zahtevnost $O(k)$ ne smemo uporabiti pomožne vrste. Ker lahko predpostavimo enoličnost imen, si zapomnimo ime prvega elementa, ter elemente premikamo iz začetka na konec vrste. Ko na začetku opazimo zapomnjeno ime, vemo da smo preiskali celotno vrsto (in da so vsi skladi na isti poziciji kot pred iskanjem).

Vstavljanje je preprosto: če med vrtenjem vrste opazimo sklad s primernim imenom, "vstavimo"element na dno sklada. Vrsto moramo kljub temu prevrteti do konca, zato si *zapomnimo uspeh vstavljanja kot dodatno spremenljivko, ki jo vrnemo na koncu.* Če v vrsti ni sklada z primernim imenom vrnemo `False`.

Na dno sklada vstavimo tako, da elemente z zanko prestavimo na pomožen sklad, dodamo `x` in nato vse elemente premaknemo nazaj na sklad.

V metodi prevrtimo celo vrsto, kar je $O(m)$, in vstavimo na dno sklada, kar je $O(k)$ zaradi dveh pretakanj. Skupaj torej $O(m + k)$. Ker nismo uporabili pomožne vrste, potrebovali pa smo pomožen sklad, je prostorsko $O(k)$.

(c) Naredimo novo vrsto. Za sprehod po vrsti spet uporabimo trik s premikanjem na konec dokler ni na začetku spet prvo ime. Vsakič, ko premaknemo sklad na konec vse elemente sklada tudi seštejemo (z zanko jih premečemo na pomožen sklad in nazaj, spotoma pa beležimo vsoto).V novo vrsto vstavimo element z imenom in vsoto.

Sprehod po vrsti je $O(m)$ elementov, na skladu pa kvečjemu $O(k)$. Vendar hkrati vsak element pogledamo zgolj $O(1)$ (premik iz sklada na sklad, prištejemo, premaknemo nazaj), torej imamo z elementi kvečjemu $O(n)$ dela. Ocena $O(m \cdot k)$ je previsoka, ocena $O(n)$ je napačna (ko se sprehajamo po vrsti moramo obiskati tudi prazne sklade, ki nis všteti v n), če ocenimo $O(m + n)$ pa bo ravno prav.

(d) Lotimo se podobno kot pri `dodaj_sklad`. V prvem koraku gremo čez vrsto in sklad z imenom (če obstaja) vzamemo ven iz vrste. V drugem koraku gremo ponovno po vrsti in ga vstavimo z novim imenom. Če ga zgolj preimenujemo, je namreč lahko na napačni lokaciji.

Potrebujemo dva sprehoda po vrsti, torej $O(m)$.

3. naloga (20 točk)

Na voljo imamo *iskalno drevo s ponovitvami*, kjer za vsako vozlišče (x je vrednost podatka v vozlišču) velja:

- vsi podatki v levem poddrevesu so manjši ali enaki x;
- vsi podatki v desnem poddrevesu so večji ali enaki x.

Opišite algoritem, ki sprejme določeno vrednost in vrne število ponovitev te vrednosti v drevesu.

Rešitve

Problema se lotimo s preprosto rekurzivno funkcijo. Denimo, da iščemo vrednost x.

1. Če funkcija naleti na prazno drevo, vrne vrednost 0.
2. V primeru vozlišča z vrednostjo y, kjer $x \neq y$. Tedaj vrnemo število ponovitev x iz primernega poddrevesa (bodisi levega ali desnega, odvisno ali je x večji ali manjši).
3. V primeru, da ima vzlišče vrednost y in $y = x$, pa izračunamo število ponovitev x za **obe** poddrevesi (saj imamo v definiciji $\leq$ in $\geq$). Te števili seštejemo, prištejemo še 1 (za trenutno vozlišče) in vrednost vrnemo.

4. naloga (20 točk)

Psička Nara je na drugem koncu travnika opazila prečudovito palico. Do palice želi priti čim hitreje, vendar je travnik poln zanimivih vonjav, ki jo ovirajo pri hitrejšem napredovanju.

Travnik predstavimo z matriko, kjer na vsakem polju označimo, koliko časa Nara potrebuje, da preduha tisti košček travnika.

$i \setminus j$	0	1	2	3	4	5
0	1	2	1	2	2	1
1	3	6	0	2	7	0
2	0	1	3	4	8	3
3	2	5	7	0	1	4

Nara se nahaja v levem zgornjem kotu, palica pa v desnem spodnjem kotu. Pri tem se lahko psička premika desno, dol, ali pa diagonalno desno-in-dol. Opišite učinkovit algoritem, ki vrne poljubno pot, po kateri si Nara lahko prilasti palico v najkrajšem možnem času. Pot predstavimo kot seznam parov (i, j), ki predstavljajo indekse za polja v matriki.

Za zgornji travnik je primer optimalne poti:

```
[(0, 0), (0, 1), (1, 2), (2, 2), (3, 3), (3, 4), (3, 5)]
```

Algoritem naj bo ustrezno utemeljen, po možnosti z ustrezno Bellmanovo enačbo.

Rešitve

Problem je variacija problema kovancev v trikotniku oz. problema požrešne miške iz vaj.

Z Bellmanovo enačbo (zaželeno)

Enačba ima napram skici zamik indeksov za ena, da se spodnje desno polje ujema z dimenzijo matrike.

$T \ldots$ matrika travnika

$(m, n) \ldots$ dimenzija matrike T

$\check{C}as(i, j) \ldots$ minimalni čas od polja (i, j) do polja (m, n)

$$\check{C}as(i,j) = T(i,j) + \min(\{\check{C}as(i+1,j), \check{C}as(i,j+1), \check{C}as(i+1,j+1)\}) \quad (\text{za } 1 \le i < m \text{in } 1 \le j < n)$$

$$\check{C}as(m,j) = T(m,j) + \check{C}as(m,j+1) \quad (\text{za } 1 \le j < n)$$

$$\check{C}as(i,n) = T(i,n) + \check{C}as(i+1,n) \quad (\text{za } 1 \le i < m)$$

$$\check{C}as(m,n) = T(m,n)$$

Vrednost optimalne rešitve je $\check{C}as(0, 0)$. Za rekonstrukcijo poti, si moramo pri izračunu minimuma beležiti še katera od opcij je optimalna.

Z opisom algoritma

Ker mora program vrniti pot in ne zgolj optimalne vrednosti, na podlagi katere se na vsakem koraku odločamo, bo naša funkcija vračala optimalno vrednost *in* optimalno pot.

Da se izognemo kopiranju podmatrik naša funkcija kot argumenta sprejme `i` in `j` polja na katerem se trenutno nahajamo. Začnemo v polju `(0, 0)`.

Na vsakem polju preverimo, kam se še lahko premaknemo (če smo v spodnji vrstici nam premik dol in diagonalno ne prideta v poštev). Za vsako opcijo premika rekurzivno izračunamo, koliko časa potrebujemo od tam do palice (zraven dobimo tudi optimalno nadaljevanje poti). Na podlagi vrednosti izberemo najboljšo od opcij. Optimalni vrednosti prištejemo vrednost trenutnega polja v matriki, optimalni poti pa na začetek dodamo trenutno polje.

Če smo na spodnjem desnem polju zgolj vrnemo vrednost polja in pot z enim korakom, ki je enak poziciji spodnjega desnega polja (zaustavitveni pogoj).

Za učinkovitost algoritem memoiziramo.

5. naloga (20 točk)

Rešujemo problem trgovskega potnika na omrežju:

$$\begin{bmatrix} \infty & 3 & 2 & 1 & 7 \\ 4 & \infty & 2 & 3 & 3 \\ 1 & 5 & \infty & 2 & 6 \\ 2 & 1 & 3 & \infty & 2 \\ 7 & 2 & 5 & 4 & \infty \end{bmatrix}$$

(a) Koliko stane krožna pot `1-4-3-5-2-1`?

(b) Podajte spodnjo mejo cene optimalne krožne poti za zgornje omrežje po metodi reduciranja matrike. Uporabite lahko poljuben vrstni red redukcij (najprej stolpci nato vrstice ali obratno).

(c) Kakšna je ocena vrednosti krožnih poti (po metodi reduciranja matrike), če zahtevamo, da krožna pot vsebuje segment `1-3-2`?

Rešitve

(a) Cena je 1 + 3 + 6 + 2 + 4 = 16, kdor pa ponesreči gleda `1-2-5-3-4-1` pa dobi 15.

(b) V obeh primerih dobimo oceno 8.

	1	1	2	1	2
0	∞	2	0	0	5
0	3	∞	0	2	1
0	0	4	∞	1	4
0	1	0	1	∞	0
1	5	0	2	2	∞

	0	0	0	0	1
1	∞	2	1	0	5
2	2	∞	0	1	0
1	0	4	∞	1	4
1	1	0	2	∞	0
2	5	0	3	2	∞

(c) Denimo, da se odločimo za redukcije stolpcev in nato vrstic. Pričnemo z že zreducirano matriko iz točke (b) in najprej nastavimo segment `1-3`. Ta povezava oceni doda 3. Ko dodamo še povezavo `3-2` dobimo skupno oceno 8 + 3 + 5 = 16.

	1	0	0	1	0	
0	∞	∞	0	∞	∞	
1	1	∞	∞	0	0	
0	∞	4	∞	0	4	→ 3
0	0	0	∞	∞	0	
0	4	0	∞	1	∞	

	0	4	0	0	0	
0	∞	∞	0	∞	∞	
0	1	∞	∞	0	0	
0	∞	0	∞	∞	∞	→ 5
0	0	∞	∞	∞	0	
1	3	∞	∞	0	∞	