Algoritmi in podatkovne strukture 1 2017/2018

Seminarska naloga 1

Rok za oddajo (1) programske kode in (2) izjave o avtorstvu prek učilnice je 25.11.2017**.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu 27.11.2017 – 1.12.2017.

Navodila

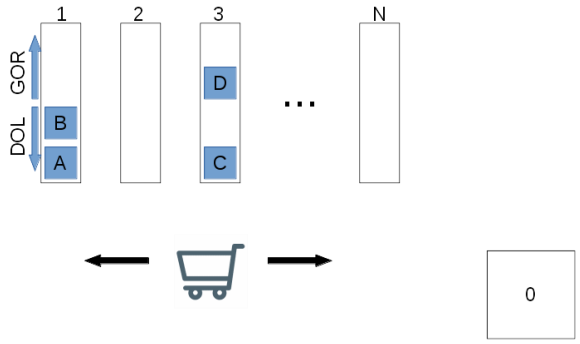
Oddana programska rešitev bo avtomatsko testirana, zato je potrebno strogo upoštevati naslednja navodila:

- Uporabite programski jezik java (program naj bo skladen z različico JDK 1.8).
- Rešitev posamezne naloge mora biti v eni sami datoteki. Torej, za pet nalog morate oddati pet datotek. Datoteke naj bodo poimenovane po vzorcu NalogaX.java, kjer X označuje številko naloge.
- Uporaba zunanjih knjižnic ni dovoljena. Uporaba internih knjižnic java.* je dovoljena (razen javanskih zbirk (collections) iz paketa java.util).
- Razred naj bo v privzetem (default) paketu. Ne definirajte svojega.
- Uporabljajte kodni nabor utf-8.
- V prvi seminarski nalogi boste vse vhodne podatke prebrali iz standardnega vhoda in vse izhode zapisali na standardni izhod.
- Vsaki nalogi je priloženo nekaj testnih primerov. Vendar ti testni primeri še zdaleč ne pokrivajo vseh možnih situacij, zato razmislite in uporabite še svoje testne primere.

Ocena nalog je odvisna od pravilnosti izhoda in učinkovitosti implementacije (čas izvajanja). Čas izvajanja je omejen na 5s za posamezno nalogo.

Naloga 1

Slika na desni prikazuje načrt skladišča. Imamo N odstavnih trakov, na vsak trak lahko shranimo D enako velikih predmetov. Do predmetov dostopamo z robotskim vozičkom, s katerim lahko dostopamo le do prvega predmeta na posameznem traku. Trak se torej obnaša po LIFO principu: v primeru traku 1 lahko v tem trenutku dostopamo le do elementa A, do B ne moremo priti. Med predmeti imamo lahko tudi prazen prostor, kot je to v primeru tretjega traku. Lokacija 0 označuje vhodni plato, kamor prihajajo novi predmeti, ki jih bomo shranjevali v skladišče.



Skladišče upravljamo z naslednjimi ukazi:

1) PREMIK i; premakne voziček do i-tega traku
2) NALOZI; naložimo trenutni zaboj iz sklada na voziček. Če voziček ni prazen, se ne zgodi nič.
3) ODLOZI; odložimo zaboj iz vozička na trenutni trak. Če na traku ni prostora, zaboj ostane na vozičku. Prav tako se ne zgodi nič, če poskušamo odlagati na lokacijo 0.
4) GOR; trenutni trak (kjer je voziček) premakne vse zaboje za en korak gor (glej sliko). Zaboj, ki je pred to akcijo na zadnjem mestu, pade iz traku in izgine. Npr., če bi 2x izvedli ukaz GOR na traku 3, bi s tem izgubili zaboj D.
5) DOL; trak se premakne dol. Zaboj na prvem mestu se, tako kot zadnji zaboj pri GOR, uniči.

Napišite program, ki simulira izvajanje ukazov in izpiše končno stanje skladišča. Skladišče je na začetku vedno prazno, robotski voziček se nahaja na lokaciji 0.

Vhod

Vhodne podatke boste brali s standardnega vhoda. V prvi vrstici bosta podana število trakov (N<=10) in dolžina trakov (D<=1000), v drugi vrstici bo z vejico ločen seznam imen zabojev (največ 10000), ki jih prejmemo na lokaciji 0. Zaboje jemljemo po vrsti; začnemo s prvim, potem vzamemo drugega, itd. Ime zaboja bo imelo največ 16 znakov. Sledijo ukazi (en ukaz na vrstico, največ 100000).

Primer:

```
5 4
B,D,A,C
NALOZI
PREMIK 1
ODLOZI
GOR
PREMIK 0
NALOZI
```

```
PREMIK 3
ODLOZI
GOR
PREMIK 0
NALOZI
PREMIK 2
PREMIK 1
GOR
DOL
ODLOZI
PREMIK 0
NALOZI
PREMIK 3
GOR
ODLOZI
```

Izhod

Izpišite končno stanje trakov 1 … N, vsak trak v svoji vrstici. Vrstica se začne s številko traku, za njo pride dvopičje, sledijo oznake zabojev ločene z vejico.

Za zgornji primer dobimo rezultat iz začetne slike:

```
1:A,B
2:
3:C,,D
4:
5:
```

Naloga 2: optimizacija skladišča

Napišite program, ki sprejme začetno in končno konfiguracijo skladišča ter poišče **najkrajše** zaporedje ukazov, ki vodijo od prve do druge. Robotski voziček naj vedno začne na lokaciji 0, vendar te lokacije pri preurejanju ne sme uporabljati. Lahko predpostavite, da bo vedno možno doseči končno ureditev.

Vhod

V prvi vrstici bosta dve celi števili N(<=5) in D(<=5). Sledita začetna in končna ureditev.

Primer:

```
3 3
1:A,,B
2:
3:C
1:A
2:C
3:
```

Izhod

Izpišite ukaze. Vsak ukaz naj bo v svoji vrstici. Pri tej nalogi je možnih več enakovrednih rešitev.

Primer rešitve zgornjega primera:

```
PREMIK 1
GOR
DOL
PREMIK 3
NALOZI
PREMIK 2
ODLOZI
```

Naloga 3: vlaki

Vlak je predstavljen v obliki dvosmernega linearnega seznama s kazalci. Glava seznama predstavlja lokomotivo, ki je opisana z največjo dovoljeno maso (v tonah) privezane kompozicije vagonov. Elementi seznama predstavljajo posamezne vagone v kompoziciji vlaka.

Posamezni vagon je opisan z največjo dovoljeno maso (v tonah) in trenutnim tovorom. Tovor je seznam predmetov, opisanih s tipom in težo v tonah. Lahko predpostavite, da na vagonu ne bo več predmetov istega tipa. Vlak je torej seznam seznamov.

Kompozicija vlaka je veljavna, če masa tovora v posameznem vagonu ne presega maksimalne teže vagona in skupna masa na vagonih ne presega največje dovoljene mase, ki je opredeljena z lokomotivo.

Napišite program, ki prebere začetno kompozicijo vlaka in zna interpretirati naslednje operacije:

- ODSTRANI_LIHE odstrani vagone z lihimi indeksi (odstrani vagon z indeksom 1 (drugi vagon), potem odstrani vagon z indeksom 3, itd.)
- ODSTRANI_HET N (odstrani vagone, ki vsebujejo N ali več različnih tipov tovora)
- ODSTRANI_ZAS P (odstrani vagone, ki imajo zasedeno P ali več odstotkov svoje max. teže)
- OBRNI (obrne vrstni red vagonov)
- UREDI (uredi vagone glede na težo – od vagona z najmanjšo skupno težo vsega tovora do vagona za največjo skupno težo. Če imata dva vagona enako težo, mora njun medsebojni vrstni red ostati enak kot pred urejanjem – torej gre za stabilno urejanje.)
- PREMAKNI Tip Vagon1 Vagon2 (premakne ves tovor tipa Tip iz Vagon1 na Vagon2, kjer sta Vagon1 in Vagon2 indeksa ustreznih vagonov. Če je na Vagon2 že ta tip tovora, se teži seštejeta, če še ni, se tovor doda na koncu. Če na Vagon1 ni tega tipa, se ne zgodi nič. Če vagon z indeksom ne obstaja, se ne zgodi nič.)

Vhod

V prvi vrstici vhoda bosta dve vrednosti: maksimalna teža celotnega vlaka in število vagonov (<=10000). Vse teže in kapacitete bodo podane kot cela števila. Sledi opis vagonov. Začnemo z maksimalno težo vagona in številom različnih tipov tovora naloženih na ta vagon. Naslednje vrstice opisujejo tovor: vsak tip je opisan v svoji vrstici s tipom tovora (*String*) in težo tovora v tonah. Sledijo ukazi (največ 20).

Primer (v [] so komentarji in niso del vhodnih podatkov):

```
100 3  [lokomotiva lahko pelje 100 ton, imamo 3 vagone]
40 2
A 20
B 10
30 3
A 5
B 20
C 5
50 3
```

XYZ 26
ZYX 26
A 3
ODSTRANI_ZAS 101
PREMAKNI A 1 0
OBRNI
ODSTRANI_LIHE

Izhod
Izpišite končno kompozicijo vlaka, kjer uporabite enak format kot pri vhodu. Na koncu izpišite še, če je kompozicija veljavna (DA/NE).

Izhod, ki ustreza zgornjemu primeru:
100 1
30 2
B 20
C 5
DA

Naloga 4: koliko vagonov?
Napišite program, ki bo poiskal najcenejšo kompozicijo vlaka za dani tovor. Cena vagona je:

1 + št. različnih tipov naloženih na vagon

Cena vlaka je vsota cen vseh vagonov.

Vhod

V prvi vrstici na vhodu bo število kosov tovora (<=30) in največja dovoljena teža vagonov(<=10) (vsi vagoni imajo enako), v naslednjih vrsticah je opis tovora. Vse podane teže bodo cela števila. Primer:

8 7 [8 kosov tovora, 7 je največja teža na vagonu]
A 6
A 3
A 2
B 2
B 2
B 2
B 2
B 1

Izhod

Izpišite ceno najcenejše kompozicije. Za gornji primer je rešitev [vagon1: A6, vagon2: A3,A2,B2, vagon3: B2,B2,B2,B1]:

7

Naloga 5: labirint

Labirint je predstavljen z dvodimenzionalnim poljem. Celice polja vsebujejo binarne vrednosti. Vrednost 1 označuje zid, vrednost 0 prazen prostor. V labirintu se nahaja robot, ki je obrnjen v eno izmed štiri smeri: sever(0), vzhod(1), jug(2), zahod(3) - v dvodimenzionalni predstavitvi labirinta so to smeri gor, desno, dol, levo. Robot lahko izvede šest ukazov:

1) FWD - premik za eno celico naprej v smeri, v kateri je obrnjen. Če se pred robotom nahaja prazna celica, se robot premakne vanjo in se ukaz uspešno zaključi. Če se pred njim nahaja zid, se robot ne premakne in se ukaz neuspešno zaključi.

2) RGT - rotacija robota za 90° v desno (v dvodimenzionalni predstavitvi labirinta to ustreza rotaciji za 90° v smeri urinega kazalca).

3) LFT - rotacija robota za 90° v levo (v dvodimenzionalni predstavitvi labirinta to ustreza rotaciji za 90° v nasprotni smeri od urinega kazalca).

4) FUN *n* - klic funkcije *n*. Funkcija lahko kliče tudi samo sebe (rekurzija).

5) SETJMP – shrani trenutni kontekst izvajanja v vrsto tipa FIFO. Shrani torej vse, tudi celotni sklad, da boš kasneje lahko od tu nadaljeval.

6) JMP – v trenutni kontekst izvajanja naloži naslednji kontekst iz vrste kontekstov. Če je vrsta kontekstov prazna, se trenutni kontekst ohrani, ukaz pa se neuspešno zaključi. (glej točko 5).

Robot izvaja program, ki je sestavljen iz funkcij. Funkcije so sestavljene iz šestih zgoraj opisanih ukazov (torej, funkcijska koda je zaporedje osnovnih ukazov). Izvajanje funkcije se zaključi:

- ko se izvedejo vsi ukazi v kodi funkcije, ali
- ko se ukaz v funkcijski kodi neuspešno zaključi (v primeru ukaza FWD oz. JMP)

Program vedno začne z izvajanjem ukazov FUN 1.

Cilj naloge je implementirati simulator robota s podanim programom v podanem labirintu.

Vhod

Postavitev labirinta je podana na standardnem vhodu:

1. V prvi vrstici sta podani dimenziji dvodimenzionalnega polja (število vrstic, število stolpcev).
2. Nato sledi vsebina posameznih vrstic dvodimenzionalnega polja (od najvišje proti najnižji), ki je podana kot zaporedje ničel in enic (od leve proti desni), kjer nič označuje prazno polje, enica pa zid. Spodnja leva celica labirinta je na koordinatah [0, 0]. Lahko predpostavite, da bo polje vedno ograjeno in da robot ne more "pasti ven".
3. Sledi opis programa. Najprej je podatek o številu funkcij, nato sledijo njihovi opisi. Opis funkcije se začne s številom ukazov, nato sledijo posamezni ukazi. V programu so funkcije označene z zaporedno številko; FUN 1 predstavlja prvo funkcijo, FUN 2 drugo, itd. Lahko predpostavite, da v programu ne bo sintaktičnih napak.

4. V naslednji vrstici bodo tri vrednosti (x, y, orientacija), ki določajo začetno pozicijo robota. Orientacija ima lahko eno od štirih vrednosti: 0 (sever), 1(vzhod), 2(jug), 3(zahod).

5. V zadnji vrstici je število korakov izvajanja. Tu štejte le korake FWD, RGT in LFT (tudi neuspešno izvedene).

Primer vhoda za 5x5 mrežo in program z dvema funkcijama:

```
5 5
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
2
4
SETJMP
FWD
FUN 1
FUN 2
3
RGT
FWD
JMP
1 1 0
7
```

Izhod

Izpišite končno lokacijo robota in njegovo orientacijo. Za zgornji vhod je pravilen izhod:

```
3 3 1
```