Algoritmi in podatkovne strukture 1 2020/2021

Seminarska naloga 1

Rok za oddajo programske kode prek učilnice je **sobota**, **28. 11. 2020.**

Zagovori seminarske naloge bodo (**predvidoma**) potekali v terminu vaj v tednu **30. 11. – 4. 12. 2020**.

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 iz paketa java.util).
- Razred naj bo v privzetem (default) paketu. Ne definirajte svojega.
- Uporabljajte kodni nabor utf-8.

Ocena nalog je odvisna od pravilnosti izhoda in učinkovitosti implementacije (čas izvajanja). Čas izvajanja je omejen na 2s za posamezno nalogo.

Naloga 1

Z avtomobilom želimo prevoziti pot dolžine D. Vozilo ima rezervoar kapacitete G. Po poti je razporejenih N bencinskih črpalk z oznakami od 1 do N. Za i-to črpalko je podana razdalja d_i od prejšnje črpalke (za prvo črpalko je podana razdalja od začetne točke poti) ter prodajna cena p_i za enoto goriva. Poraba vozila je ena enota goriva na eno enoto razdalje. Naloga je določiti, na katerih bencinskih črpalkah moramo dotakati gorivo, če želimo prevoziti celotno pot po **najnižji ceni**. Pri tem velja omejitev, da ob postanku na bencinski črpalki vedno napolnimo rezervoar vozila do konca. Če ima več sekvenc ustavljanja enako ceno, izberemo tisto z **najmanj postanki**.

Implementirajte razred **Naloga1**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]). Metoda naj prebere vhodne podatke, določi optimalno zaporedje ustavljanj in v izhodno datoteko zapiše oznake izbranih bencinskih črpalk.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici so zapisana tri cela števila, ločena z vejico. Zapis D,G,N določa dolžino poti, kapaciteto rezervoarja ter število bencinskih črpalk.
- V naslednjih N vrsticah so zapisani podatki o bencinskih črpalkah. Vsaka vrstica se začne z oznako črpalke, za njo pride dvopičje, nato sledi par celih števil, ločenih z vejico, ki določata razdaljo bencinske črpalke in prodajno ceno goriva.

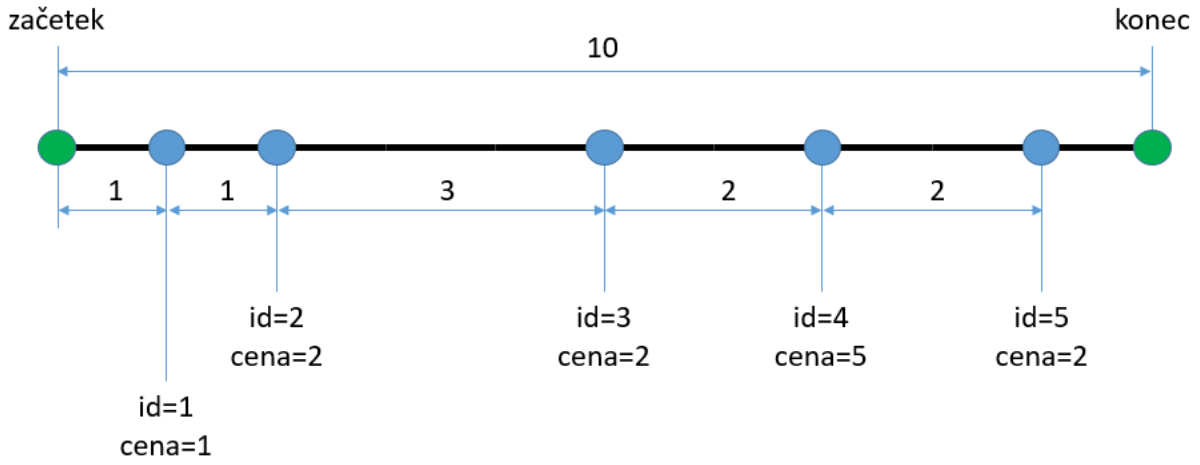
Tekstovna izhodna datoteka naj vsebuje eno samo vrstico z oznakami bencinskih črpalk, na katerih dolivamo gorivo. Oznake bencinskih črpalk naj bodo zapisane kot cela števila ločena z vejicami (brez dodatnih presledkov).

Opomba: vhodna naloga ima lahko več enakovrednih rešitev. V tem primeru v izhodno datoteko zapišite samo eno rešitev.

Primer:

Vhodna datoteka:	Izhodna datoteka:
10,4,5 1:1,1 2:1,2 3:3,2 4:2,5 5:2,2	1,3,5

Razlaga primera:



Najcenejša sekvenca ustavljanja ima ceno 17 in zahteva 3 postanke (na črpalkah 1, 3 in 5). Na začetku ima avtomobil poln rezervoar, ki vsebuje 4 enote goriva. Vstavimo se na prvi črpalki in natočimo 1 enoto goriva (do polnega rezervoarja) po ceni 1. Ko se ustavimo pri tretji črpalki je rezervoar prazen, zato natočimo 4 enote goriva po ceni 2. Skupna cena poti je sedaj 1 + 4*2=9. Ko pridemo do pete črpalke je rezervoar ponovno prazen, zato natočimo 4 enote goriva po ceni 2. Skupna cena poti je sedaj 9 + 4*2=17. V avtomobilu imamo dovolj goriva, da pridemo do cilja in uspešno opravimo nalogo. Obstaja še ena sekvenca ustavljanja s ceno 17, ki vključuje črpalke 1, 2, 3 in 5 (cena poti je 1*1 + 1*2 + 3*2 + 4*2=17), a jo zavržemo, saj vsebuje 4 postanke.

Naloga 2

Podana je dvodimenzionalna mreža znakov. Poiščite najdaljšo pot v podani mreži, ki je sestavljena iz enakih znakov. Pot je sestavljena iz sosednjih polj, pri čemer velja, da sta polji sosednji, če iz enega polja lahko pridemo v drugega z enim samim premikom v smeri levo-desno ali god-dol (**ne pa tudi po diagonali!**). Vsako polje na poti obiščemo **samo enkrat**.

Implementirajte razred **Naloga2**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke, poišče najdaljšo pot v podani mreži in jo zapiše v izhodno datoteko.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici sta zapisani dve celi števili, ločeni z vejico. Zapis V,S določa dimenzije mreže, pri čemer je V število vrstic in S število stolpcev mreže.
- V naslednjih V vrsticah so zapisani elementi mreže. Vsaka vrstica vsebuje S znakov, ločenih z vejicami.

Tekstovna izhodna datoteka naj vsebuje dve vrstici.

- Zapis v prvi vrstici naj bo oblike Y,X in naj določa vrstico in stolpec prvega znaka na najdeni najdaljši poti v mreži. Polje v levem zgornjem kotu mreže je na koordinati 0,0.
- Druga vrstica naj vsebuje zaporedje premikov (LEVO, DESNO, GOR in DOL), s katerimi se sprehodimo po celotni poti. Premiki naj bodo ločeni z vejicami (brez dodatnih presledkov).

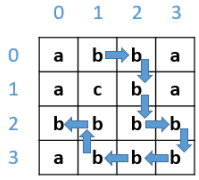
Opomba: vhodna naloga ima lahko več enakovrednih rešitev. V tem primeru v izhodno datoteko zapišite samo eno rešitev. Vsako pot je možno zapisati na dva načina (odvisno od tega, iz katerega konca začnemo). Ni pomembno, kateri konec izberete za začetno polje.

Primer:

Vhodna datoteka:	Izhodna datoteka:

`4,4`
`a,b,b,a`
`a,c,b,a`
`b,b,b,b`
`a,b,b,b`
	`0,1`
	`DESNO,DOL,DOL,DESNO,DOL,LEVO,LEVO,GOR,LEVO`

Razlaga primera:



Naloga 3

Podan je tekst, ki je zakodiran v treh korakih.

- **Korak 1**

  Najprej je vsak znak pretvorjen v ustrezno naravno število s pomočjo spodnje tabele.

A	B	C	Č	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	Š	T	U	V	Z	Ž
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

a	b	c	č	d	e	f	g	h	i	j	k	l	m	n	o	p	r	s	š	t	u	v	z	ž	_
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

Znak »_« predstavlja presledek.

- **Korak 2**

  Dobljen seznam naravnih števil napolnimo v **N** vrstic. Vsaka vrstica predstavlja seznam, kjer se znaki dodajajo na konec in jemljejo iz začetka (FIFO seznam). Znake dodajamo v vrstice po vrsti od prvega do zadnjega, in sicer tako, da **i**-ti znak dodamo v (**i % N**)-to vrstico. Pozor: šteti začnemo z nič.

- **Korak 3**

  Tako dobljeno tabelo seznamov nato premešamo v **K** korakih. Začnemo v vrstici $\mathbf{l_0=0}$. Iz te vrstice (seznama) odstranimo prvi element ($\mathbf{e[l_0]}$) in ga dodamo v $\mathbf{k_0}$-to vrstico, kjer je $\mathbf{k_0=((l_0 + e[l_0])\%N)}$. Nato se premaknemo v vrstico $\mathbf{l_1=(k_0+P)\%N}$ in ponovimo postopek. Torej vzamemo prvi element vrstice $\mathbf{l_1}$, tokrat ga označimo z $\mathbf{e[l_1]}$. In ga dodamo v vrstico $\mathbf{k_1=(l_1 + e[l_1])\%N}$. Nato se premaknemo v vrstico $\mathbf{l_2=(k_1+P)\%N}$ in ponovimo postopek še **K-2** krat, tako da je vseh premikov natanko **K**. Celi števili **P** in **K** sta parametra kodiranja. Kot rezultat pa vrnemo tudi $\mathbf{k_{K-1}}$, kamor postavimo zadnji element. Pozor: **P** je lahko negativen, zato je lahko $\mathbf{l_j}$ negativen. V tem tem primeru je $\mathbf{l_j=N-l_j}$. Z drugimi besedami, indeksa **k** in **l** sta ciklična.

Implementirajte razred **Naloga3**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne (args[0]) in izhodne datoteke (args[1]), prebere vhodne podatke, jih **dekodira** in zapiše besedilo v izhodno datoteko.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici so zapisana štiri cela števila, ločena z vejico. Zapis **N,V,K,P** določa število vrstic **N**, zadnjo vrstico kamor smo dodali znak **V**, število premikov pri kodiranju **K** in premik do nove vrstice **P**.
- V naslednjih **N** vrsticah so po vrsti zapisani elementi vrstic. Vsaka vrstica lahko vsebuje različno število elementov (naravnih števil) ločenih z vejico (brez presledkov).

V izhodni datoteki naj bo zapisano dekodirano besedilo.

Primer:

Vhodna datoteka:	Izhodna datoteka:

`3,1,100,-1`
`42,38,43,34,45`
`16,20,39,30,50,34`
`42,30`
	`Testni Primer`

Primer postopka kodiranja teksta »Testni Primer« z N=3, P=-1;

- Korak 1:
  {20,30,43,45,39,34,50,16,42,34,38,30,42}
- Korak 2:
  20,45,50,34,42
  30,39,16,38
  43,34,42,30
- Korak 3:

Premik 1	Premik 2	Premik 3	Premik 4
45,50,34,42 30,39,16,38 43,34,42,30,20	45,50,34,42 39,16,38,30 43,34,42,30,20	50,34,42,45 39,16,38,30 43,34,42,30,20	50,34,42,45,43 39,16,38,30 34,42,30,20

Naloga 4

Želimo implementirati podatkovno strukturo, ki simulira dodeljevanje pomnilniškega prostora spremenljivkam. Ta podatkovna struktura mora podpirati naslednje operacije:

- public void init(int size)
- public boolean alloc(int size, int id)
- public int free(int id)
- public void defrag(int n)

Metoda void init(int size) inicializira statično polje velikosti *size* bajtov. Po zaključenem klicu velja, da je celotno polje prosto. Privzamemo, da klic metode init vedno uspe.

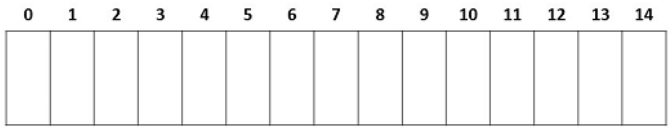
Z metodo boolean alloc(int size, int id) dodelimo *size* bajtov spremenljivki z oznako *id*. Zaporedje bajtov, ki pripadajo isti spremenljivki, imenujemo **blok**. Pri izbiri prostora, ki naj se dodeli spremenljivki upoštevajte naslednje pravilo: dodeli se **prvih** *size* zaporednih bajtov, ki so na voljo. Kadar zahtevanega prostora ni mogoče dodeliti ali spremenljivka s podanim *id* že obstaja, metoda vrne false, sicer je rezultat true.

Metoda int free(int id) ponovno sprosti prostor, ki ga zaseda spremenljivka z oznako *id*. Po zaključenem klicu se privzame, da je sproščen prostor nezaseden, metoda pa vrne število sproščenih bajtov. Če spremenljivke z oznako *id* ni, funkcija vrne 0.

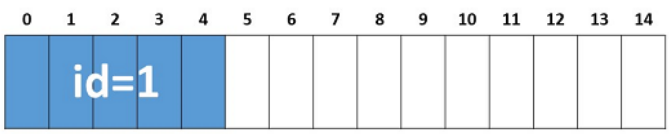
Metoda void defrag(int n) je namenjena reorganizaciji zasedenosti pomnilniškega prostora. Vhodni parameter določa, koliko korakov defragmentacije izvedemo. V enem koraku defragmentacije poiščemo prvi nezaseden prostor v polju. Če temu prostoru sledi zaseden blok (nezaseden prostor predstavlja vrzel), ta blok v celoti premaknemo na začetek najdenega praznega prostora.

Ilustrativni primer:

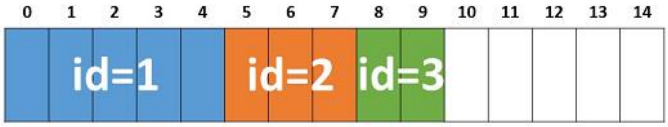
- klic init(15) zgradi naslednje polje:



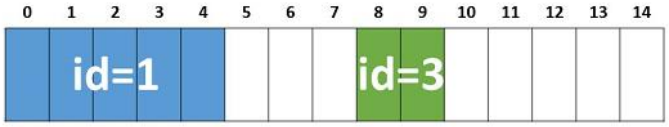
- klic alloc(5,1) rezervira blok 1 na začetku polja:



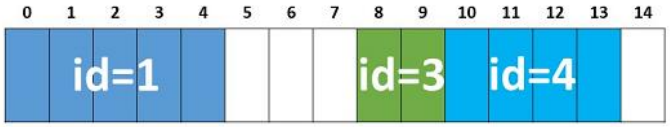
- po klicih alloc(3,2) in alloc(2,3) so v polju trije sosednji bloki, ki so dodeljeni spremenljivkam z oznakami 1, 2 in 3:



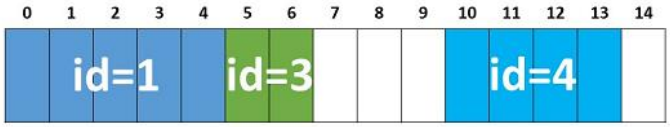
- klic free(2) sprosti blok 2:



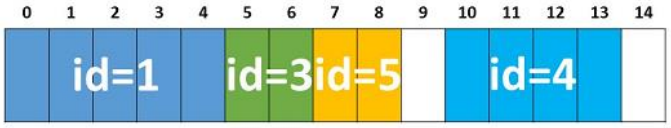
- klic alloc(4,4) bo rezerviral prostor za spremenljivko z oznako 4 desno od bloka 3, ker med blokoma 1 in 3 ni dovolj prostora:



- po klicu defrag(1) bo polje izgledalo takole:



- klic alloc(2,5) rezervira blok za spremenljivko 5 na prvem možnem mestu v polju:



Implementirajte razred **Naloga4**, ki vsebuje metodo **main**. Argumenti metode main vsebujejo poti do vhodne in izhodne datoteke (args[0] in args[1]). Metoda naj prebere ukaze iz vhodne datoteke, jih izvede in zapiše rezultat v izhodno datoteko.

Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število N, ki določa število ukazov v datoteki. V naslednjih N vrsticah sledi zaporedje klicev nad implementirano podatkovno strukturo. Vsaka vrstica sestoji iz ukaza (znak 'i' - init, 'a' - alloc, 'f' - free, 'd' - defrag) in enega ali dveh celoštevilčnih argumentov.

Primeri klicev:

- i,512 pomeni klic ukaza init(512), ki rezervira statično polje velikosti 512 bajtov
- a,4,55 pomeni klic ukaza alloc(4, 55), ki spremenljivki z oznako 55 dodeli 4 bajte prostora
- f,2 pomeni klic free(2), ki sprosti pomnilnik, dodeljen spremenljivki z oznako 2
- d,4 pomeni klic defrag(4), ki izvede 4 korake defragmentacije

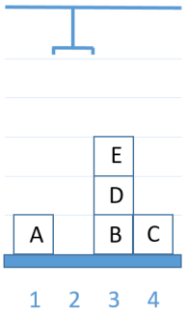
Ob zaključku se v izhodno datoteko zapiše trenutna vsebina pomnilnika. Za vsako spremenljivko v pomnilniku se v izhodno datoteko izpiše vrstica s tremi celimi števili, ločenimi z vejicami. Zapis A,B,C označuje spremenljivko z oznako A, kateri je dodeljen pomnilnik od indeksa B do indeksa C (oba vključno). Trenutna vsebina pomnilnika naj bo izpisana po naraščajočem indeksu B (po blokih z leve proti desni).

Primer:

Vhodna datoteka:	Izhodna datoteka:
8 i,15 a,5,1 a,3,2 a,2,3 f,2 a,4,4 d,1 a,2,5	1,0,4 3,5,6 5,7,8 4,10,13

Naloga 5

Skladišče ima P odstavnih položajev, na vsak položaj lahko naložimo N enako velikih škatel. Na spodnji sliki je prikazano skladišče, kjer sta P=4 in N=5.



Skladišče upravljamo z robotsko roko, s katero lahko dostopamo le do vrhnje škatle na posameznem odstavnem položaju. Robotska roka pozna naslednja ukaza:

- VZEMI *p*; robotska roka vzame vrhnjo škatlo na *p*-tem položaju ($1 \leq p \leq P$).
- IZPUSTI *p*; robotska roka položi škatlo, ki jo drži, na vrh kupa na *p*-tem položaju ($1 \leq p \leq P$). Če je *p*-ti položaj bil prazen, se škatla položi na tla.

Napišite program, ki sprejme začetno in končno konfiguracijo skladišča ter poišče **najkrajše zaporedje** ukazov, ki vodijo od začetne do končne konfiguracije. Robotska roka na začetku ne drži nobene škatle. Lahko predpostavite, da bo vedno možno doseči končno ureditev.

Implementirajte razred **Naloga5**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]). Metoda naj prebere vhodne podatke, poišče najkrajše zaporedje ukazov, ki rešijo nalogo, in jih zapiše v izhodno datoteko.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici bosta podana število odstavnih položajev P in višina skladišča N.
- V naslednjih P vrsticah je opisana vsebina posameznih odstavnih položajev začetne konfiguracije. Vsaka vrstica se začne s številko položaja, za njo pride dvopičje, sledijo oznake škatel ločene z vejico, kot si sledijo od tal proti vrhu kupa. Škatle so označene z eno črko ('A' do 'Z').
- V naslednjih P vrsticah je opisana vsebina posameznih odstavnih položajev končne konfiguracije.

Tekstovna izhodna datoteka naj vsebuje **najkrajše** zaporedje ukazov, ki dosežejo zahtevano ureditev. Vsak ukaz naj bo v ločeni vrstici.

Opomba: vhodna naloga ima lahko več enakovrednih rešitev. V tem primeru v izhodno datoteko zapišite samo eno rešitev.

Primer:

Vhodna datoteka:	Izhodna datoteka:
3,2 1:A,B 2: 3: 1: 2:A,B 3:	VZEMI 1 IZPUSTI 3 VZEMI 1 IZPUSTI 2 VZEMI 3 IZPUSTI 2

Razlaga primera: