Algoritmi in podatkovne strukture 1 2018/2019

Seminarska naloga 1

Rok za oddajo programske kode prek učilnice je **sobota**, **24. 11. 2018.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **26. 11. – 30. 11. 2018**.

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 1s za posamezno nalogo.

Naloga 1

V mestu je zemljevid ulic predstavljen s koordinatami križišč. Koordinate so pari celih števil (x,y). Dolžina poti med dvema koordinatama je enaka Manhattanski razdalji med njima. V mestu se nahaja taksist, ki v svojem vozilu lahko istočasno pelje do N strank. Taksistu so znane koordinate M strank in koordinate njihovih ciljnih točk. Naloga je poiskati najkrajšo pot taksista, po kateri bo razvozil vse stranke do njihovih ciljev. Pri tem velja omejitev, da vsaka stranka lahko izstopi iz vozila samo na svoji ciljni točki.

Implementirajte razred **Naloga1**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]) in na podlagi prebranega vhoda poišče optimalno pot taksista, ki jo potem zapiše v izhodno datoteko.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici je celo število N. Ta predstavlja število strank, ki jih taksist lahko istočasno vozi.
- V drugi vrstici sta zapisani dve celi števili, ločeni z vejicami. Zapis X,Y predstavlja izhodiščno pozicijo taksija na koordinatah (X,Y).
- V tretji vrstici je celo število M. Ta predstavlja število strank, ki čaka na prevoz.
- V vsaki izmed naslednjih M vrstic je zapisanih pet celih števil, ločenih z vejicami. Zapis A, B,C,D,E predstavlja stranko z unikatno oznako A, ki čaka na koordinatah (B,C) in si želi priti na cilj s koordinatami (D,E).

Tekstovna izhodna datoteka naj vsebuje eno vrstico z zaporedjem oznak strank, v katerem jih taksist pobira oziroma odlaga med vožnjo. Oznake strank naj bodo med seboj ločene z vejicami. Prva pojavitev neke stranke v izhodnem zaporedju označuje, da jo bo taksist pobral na njenih izhodiščnih koordinatah. Druga pojavitev oznake stranke pomeni, da jo bo taksist odložil na njenih ciljnih koordinatah.

Primer:

Vhodna datoteka:	Izhodna datoteka:
2 5,5 3 1,3,7,5,7 2,9,2,9,7 3,1,1,2,8	2,3,3,1,1,2

Razlaga primera: taksist bo najprej pobral stranko z oznako 2. Potem bo pobral še stranko z oznako 3 in jo tudi odložil na želeni lokaciji. Zatem bo pobral in odložil stranko z oznako 1. Na koncu bo še odložil stranko z oznako 2. Prevožena pot bo: 7 + 9 + 8 + 2 + 2 + 4 = 32.

Naloga 2

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.

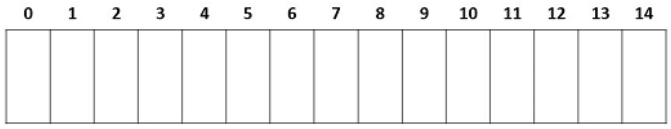
Z metodo `boolean alloc(int size, int id)` dodelimo *size* bajtov spremenljivki z oznako *id*. Zaporedje bajtov, ki pripadajo isti spremenljivki, imenujemo **blok**. Pri izbiri prostora, ki naj se dodeli spremenljivki upoštevajte naslednje pravilo: dodeli se **prvih** *size* zaporednih bajtov, ki so 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.

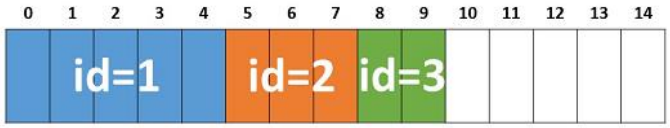
Ilustrativni primer:

- klic `init(15)` zgradi naslednje polje:

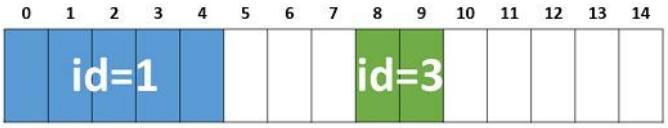


- klic `alloc(5,1)` rezervira blok 1 na začetku polja:

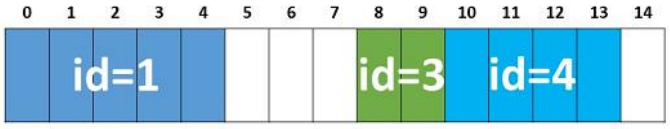
- po klicih alloc(3,2) in alloc(2,3) so v polju trije sosednji bloki, ki so dodeljeni spremenljivkam z oznakami 1, 2 in 3:



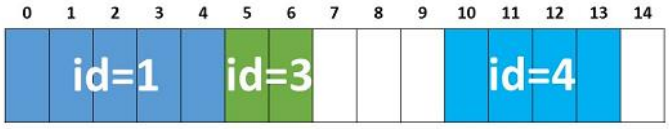
- klic `free(2)` sprosti blok 2:



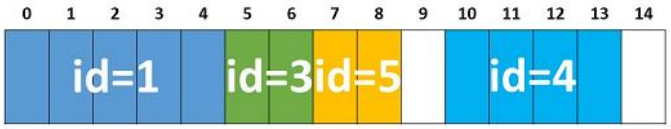
- klic `alloc(4,4)` bo rezerviral prostor za spremenljivko z oznako 4 desno od bloka 3, ker med blokoma 1 in 3 ni dovolj prostora:



- 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 **Naloga2**, 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 3

Podana je trenutna vsebina podatkovne strukture iz Naloge 2. S čim manj premeščenih bajtov želimo združiti vse dodeljene bloke tako, da med njimi ni več praznih mest in se nahajajo na začetku statičnega polja. Pri premikih upoštevajte, da se uporablja samo prostor znotraj dane strukture. Z drugimi besedami, vsi premiki se izvajajo znotraj rezerviranega polja. Premikajo se vedno bloki v celoti (cena premika je enaka velikosti bloka v bajtih).

Pri preurejanju pomnilniških blokov pazite, da mora biti ob zaključku vsebina blokov nespremenjena (spremeni se le njihova lokacija). Dovoljeni so premiki celotnega bloka v zadosti velik nezaseden prostor in premiki kjer se novi položaj bloka delno prekriva z njegovim starim položajem.

Implementirajte razred **Naloga3**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]) in na podlagi prebranega vhoda poišče optimalno preurejanje pomnilniških blokov, ki ga potem zapiše v izhodno datoteko.

Vhodna datoteka je v enakem formatu kot izhodna datoteka v Nalogi 2 in opisuje začetno stanje pomnilnika.

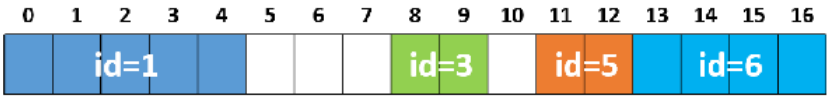
V izhodno datoteko zapišite zaporedje premikov oblike A,B, kjer A pomeni oznako bloka (id spremenljivke, kateri je blok dodeljen), B pa njegovo novo začetno lokacijo v pomnilniku (indeks prvega bajta bloka).

Primer:

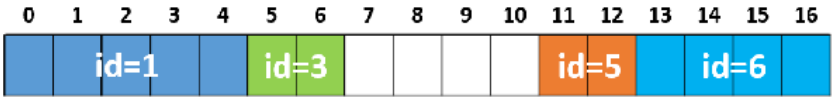
Vhodna datoteka:	Izhodna datoteka (ena izmed enakovrednih rešitev):
1,0,4 3,8,9 5,11,12 6,13,16	3,5 6,7

Razlaga primera:

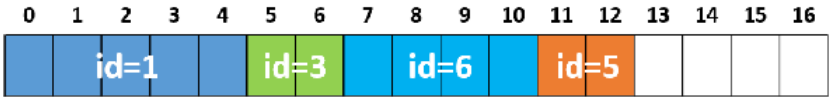
- izhodiščna situacija vsebuje štiri bloke.



- najprej premaknemo blok 3 tako, da začne na poziciji 5. To je premik dveh bajtov.



- nato premaknemo blok 6 tako, da začne na poziciji 7. To je premik 4 bajtov. Sedaj ni več praznih mest med bloki, skupna cena premikov pa je 2 + 4 = 6 bajtov.



Naloga 4

Želimo implementirati seznam celih števil s podatkovno strukturo, ki kombinira dobre lastnosti statičnih in dinamičnih polj. Za razliko od običajnega linearnega seznama s kazalci, kjer vsak člen seznama hrani en sam element, bo v naši strukturi vsak člen seznama hranil do N elementov.

Seznam bo tako predstavljen kot usmerjen linearni seznam, katerih členi vsebujejo statična polja velikosti N.

Podatkovna struktura naj podpira naslednje metode:

- `public void init(int N)`
- `public boolean insert(int v, int p)`
- `public boolean remove(int p)`

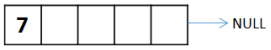
Metoda `void init(int N)` sprejme parameter `N` (kjer je `N>1`), ki določa največje število elementov v posameznem členu seznama. Po klicu metode je seznam prazen. Konstruktor strukture avtomatsko kliče metodo `init` z vrednostjo `N=5`.

Metoda `boolean insert(int v, int p)` prejme dva argumenta: vrednost (`v`), ki jo želimo vstaviti in pozicijo (`p`), na kateri naj se ta element vstavi (prvi element je na poziciji 0). Opozorilo: pozicija ni definirana v fizičnem, temveč v logičnem smislu - upoštevajo se le dejansko vstavljeni elementi in ne indeksi statičnih polj. Najprej poskusimo vrednost `v` vstaviti za elementom, ki se trenutno nahaja na poziciji `p-1` (s tem bo `v` postal `p`-ti element seznama, kar je naš cilj). To naredimo **izključno** v primeru, ko ima ciljni člen (tisti, ki vsebuje element na poziciji `p-1`) vsaj eno prazno mesto. V nasprotnem primeru poiščemo člen, v katerem se nahaja element na poziciji `p` (lahko, da bo to isti člen, ki vsebuje element na poziciji `p-1`) in poskusimo vrednost `v` vstaviti pred ta element (kar bo spet pripeljalo do tega, da bo `v` postal `p`-ti element seznama). Če ima statično polje v tem členu vsaj eno prazno mesto, vrednost `v` vstavimo na ustrezno pozicijo in zaključimo. Če pa je statično polje polno, ga razdelimo na dva dela tako, da ga nadomestimo z dvema členoma. Prvi člen vsebuje prvo polovico elementov (**zaokroženo navzdol**), preostanek pa je vsebovan v drugem členu. Sedaj ponovimo postopek vstavljanja elementa `v`, ki bo zagotovo končal v enem izmed ustvarjenih dveh členov. Po uspešnem vstavljanju metoda vrne vrednost `true`. V primeru vstavljanja na neveljavno lokacijo (negativna vrednost ali vrednost, večja od števila elementov v strukturi) se vrednost zavrže in metoda vrne `false`.

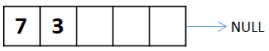
Funkcija za brisanje `boolean remove(int p)` prejme pozicijo elementa, ki ga želimo odstraniti iz seznama. Najprej poiščemo logično pozicijo, ki bo izbrisana (upoštevamo samo dejansko vstavljene elemente in ne indeksov polj). Element na tem mestu izbrišemo in po potrebi izvedeno zamik elementov v levo, da se izognemo vrzeli znotraj fizičnega polja. Če po brisanju število vstavljenih elementov v tem členu pade pod N/2 (**zaokroženo navzdol**), prenesemo iz morebitnega naslednjega člena toliko elementov, da dobimo v našem členu ravno N/2 (**zaokroženo navzdol**) zasedenih mest. Če je sedaj v naslednjem členu premalo elementov (manj kot N/2 **zaokroženo navzdol**), v trenutni člen prenesemo tudi vse preostale elemente iz naslednika in ga izbrišemo.

Ilustrativni primer izvajanja operacij na strukturi:

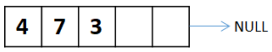
- po zaporedju klicev `init(5)` in `insert(7,0)` bo struktura vsebovala samo en člen:



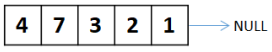
- po klicu `insert(3, 1)`:



- po klicu `insert(4,0)` se vsebina statičnega polja znotraj člena ustrezno zamakne:



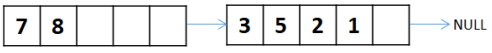
- po zaporedju klicev `insert(2,3)` in `insert(1,4)`:



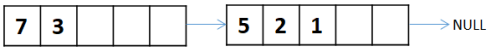
- po klicu `insert(5,3)` se člen razdeli na dva dela (4 in 7 gresta v en člen, preostali elementi pa v drugega), nato se element 5 vstavi na ustrezno mesto:

- po klicu `insert(8,2)`:

- po klicu `remove(0)` se elementi prvega člena zamaknejo:



- po klicu `remove(1)` ima prvi člen premalo elementov in se en element prenese iz naslednjega člena:



- po klicu `remove(3)`:

- če sedaj kličemo `remove(1)`, se člena združita, saj po prenosu elementa iz drugega člena v le-tem ostane premalo elementov:

- če bi namesto ukaza `remove(1)` klicali `remove(2)`, bi ostala dva člena, saj drugi člen nima naslednika, iz katerega bi prevzel elemente:

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 število K, s katero je določeno število ukazov, ki sledijo. V naslednjih K vrsticah sledi zaporedje klicev za vstavljanje in brisanje vrednosti iz podatkovne strukture.

Zapis *s,n* predstavlja klic metode `init(n)`. Zapis *i,v,p* predstavlja klic `insert(v,p)`. Zapis *r,p* predstavlja klic `remove(p)`.

Po zaključku izvajanja se v izhodno datoteko zapiše **fizična** vsebina podatkovne strukture. V prvi vrstici naj bo zapisano število členov strukture. V vsaki izmed naslednjih vrstic je zapisana vsebina enega člena v vrstnem redu od prvega do zadnjega. Ena takšna vrstica vsebuje z vejicami ločene trenutne vrednosti v statičnem polju člena. Nezasedena polja izpisujte kot NULL.

Primer:

Vhodna datoteka:	Izhodna datoteka:
12 s,5 i,7,0 i,3,1 i,4,0 i,2,3 i,1,4 i,5,3 i,8,2 r,0 r,1 r,3 r,2	2 7,3,NULL,NULL,NULL 1,NULL,NULL,NULL,NULL

Naloga 5

Želimo implementirati podatkovno strukturo v obliki enosmernega linearnega seznama, sestavljenega iz M členov. Členi so statična polja velikosti N bajtov.

Podatkovna struktura je namenjena dinamičnemu dodeljevanju pomnilniškega prostora spremenljivkam. Potrebno je realizirati sledeče funkcije:

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

Metoda `void init(m, n)` kreira linearni seznam, sestavljen iz *m* členov, ki vsebujejo statična polja velikosti *n* bajtov. Na začetku je ves prostor v členih nezaseden. Konstruktor strukture samodejno kliče `init(16,16)`.

Z metodo `boolean alloc(int size, int id)` spremenljivki z oznako *id* dodelimo *size* bajtov prostora (kjer je *size* <= N). Dodeli se prostor v tistem členu, v katerem bo po vstavljanju ostalo **najmanj** nezasedenega prostora. Kadar je takšnih členov več, se dodeli prostor v tistem, ki je **prvi** na poti od glave proti repu seznama.

Metoda `int free(int id)` sprosti prostor, ki ga trenutno zaseda spremenljivka z oznako *id*. Po sprostitvi se ostale spremenljivke v členu premaknejo na začetek polja tako, da med njimi ni nezasedenega prostora.

Implementirajte razred **Naloga5**, 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.

V prvi vrstici tekstovne vhodne datoteke je zapisano število K, ki določa število ukazov v datoteki. V naslednjih K vrsticah sledi zaporedje klicev za inicializacijo, dodeljevanje in sproščanje pomnilnika.

Zapis *i,m,n* označuje klic `init(m,n)`. Zapis *a,s,i* označuje klic `alloc(s,i)`. Zapis f,j označuje klic `free(j)`.

Ob zaključku izvajanja naj izhodna datoteka vsebuje N+1 vrstic. V prvi vrstici naj bo zapisano število členov, ki so popolnoma nezasedeni. V drugi vrstici naj bo zapisano število členov, ki imajo zaseden samo en bajt, in tako naprej, do zadnje vrstice, v kateri naj bo zapisano število popolnoma zasedenih členov.

Primer:

Vhodna datoteka:	Izhodna datoteka:
5 i,10,5 a,4,1 a,2,2 f,1 a,3,1	9 0 0 0 0 1

Razlaga primera:

Na začetku ima struktura 10 členov, vsak izmed katerih vsebuje polje velikosti pet bajtov. Najprej rezerviramo štiri bajte prostora za spremenljivko z oznako 1. Ta prostor se rezervira v prvem členu. Rezervacija prostora za spremenljivko z oznako 2 se izvede v drugem členu, saj v prvem členu ni dovolj nezasedenega prostora (na voljo je le en bajt). V naslednjem ukazu se prostor, ki ga v prvem

členu zaseda spremenljivka 1, sprosti (sedaj ima prvi člen ponovno prostih vseh pet bajtov). V zadnjem ukazu rezerviramo tri bajte za spremenljivko z oznako 1. Rezervacija se izvede v drugem členu, ki bo po tej rezervaciji popolnoma zaseden (rezervacija v tem členu ne pusti nezasedenega prostora - v vseh ostalih členih bi ostala nezasedena dva bajta). Na koncu imamo en popolnoma zaseden člen, vsi ostali členi pa so prazni.