

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.
- Uporabljamte kodni nabor utf-8.

Ocena nalog je odvisna od pravilnosti izhoda in učinkovitosti implementacije (čas izvajanja). Čas izvajanja je omejen na 1s za posamezno naložbo.

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.

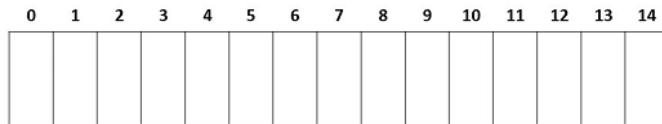
Z metodo `boolean alloc(int size, int id)` dodelimo `size` bajtov spremenljivki z oznako `id`. Zaporede bajtov, ki pripadajo isti spremenljivki, imenujemo **blok**. Pri izbiri prostora, ki naj se dodeli spremenljivki upoštevajte naslednje pravilo: dodeli se **prvih** `size` zaporednih bajtov, ki so na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

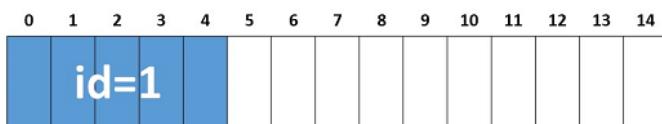
Metoda `void defrag(int n)` je namenjena reorganizaciji zasedenosti pomnilniškega prostora. Vhodni parameter določa, koliko korakov defragmentacije izvedemo. V enem koraku defragmentacije poiščemo prvi nezaseden prostor v polju. Če temu prostoru sledi zaseden blok (nezaseden prostor predstavlja vrzel),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

Ilustrativni primer:

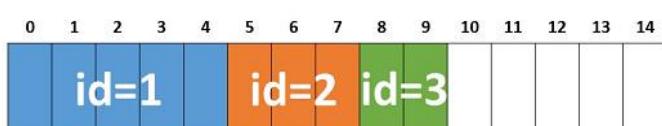
- klic `init(15)` zgradi naslednje polje:



- 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:



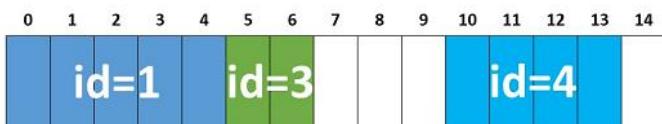
- klic `free(2)` sprosti blok 2:



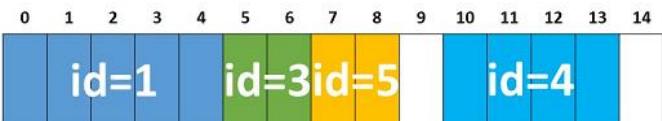
- klic `alloc(4, 4)` bo rezerviral prostor za spremenljivko z oznako 4 desno od bloka 3, ker med blokoma 1 in 3 ni dovolj prostora:



- 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)

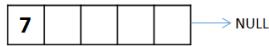
Metoda `void init(int N)` sprejme parameter N (kjer je $N > 1$), ki določa največje število elementov v posameznem členu seznama. Po klicu metode je seznam prazen. Konstruktor strukture avtomatsko kliče metodo `init` z vrednostjo $N=5$.

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 zavriže in metoda vrne `false`.

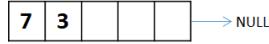
Funkcija za brisanje `boolean remove(int p)` prejme pozicijo elementa, ki ga želimo odstraniti iz seznama. Najprej poiščemo logično pozicijo, ki bo izbrisana (upoštevamo samo dejansko vstavljeni elemente in ne indeksov polj). Element na tem mestu izbrišemo in po potrebi izvedeno zamik elementov v levo, 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:

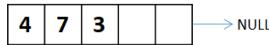
- po zaporedju klicev `init(5) in insert(7, 0)` bo struktura vsebovala samo en člen:



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



- po klicu `insert(4, 0)` se vsebina statičnega polja znotraj člena ustrezeno 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 ustrezeno 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 letem 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 $size \leq N$). 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	9
i,10,5	0
a,4,1	0
a,2,2	0
f,1	0
a,3,1	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.

Algoritmi in podatkovne strukture 1

2018/2019

Seminarska naloga 2

Rok za oddajo programske kode prek učilnice je **sobota, 12. 1. 2019.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **14. 1. – 18. 1. 2019.**

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 internalnih knjižnic java.* je dovoljena (vključno z javanskimi zbirkami iz paketa java.util).
- Razred naj bo v privzetem (default) paketu. Ne definirajte svojega.
- Uporabljamte 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 6

Transportno podjetje prevaža blago med mesti, povezanimi z dvosmernimi cestami. Nekatere ceste vključujejo predore, ki omejujejo prehod previsokim tovornjakom.

Zaradi pogostih vzdrževalnih del v predorih je treba spremnijati pot, po kateri bo blago prepeljano iz enega mesta v drugo. V ta namen želimo prešteti, na koliko različnih načinov lahko pridemo iz enega mesta do drugega. Pri tem preštevamo samo poti, ki ne vsebujejo ciklov (mesta se v njem ne ponavljajo) in so vsi morebitni predori na poti višji od višine vozila (podan parameter).

Želimo torej prešteti število različnih sekvenc id_1, id_2, \dots, id_n brez ponavljačih elementov, kjer je id_1 začetna lokacija, id_n pa končna lokacija. Pri tem mora veljati, da sta sosednja id-ja povezana s cesto, ki ne vsebuje predora, strogo nižjega od višine vozila.

V vhodni datoteki je v prvi vrstici zapisano število cest na območju M. V naslednjih M vrsticah so zapisani podatki o cestah. Vsaka vrstica je oblike ID_M1, ID_M2, H (vse vrednosti so cela števila). ID predstavlja oznako ceste; ID_M1 in ID_M2 sta oznaki mest ki ju ta cesta povezuje; H je višina predora (za ceste brez predorov je vrednost H enaka -1). Podatkom o cestah sledi vrstica oblike $ID1, ID2$, ki določata izhodiščno in ciljno mesto. Tej vrstici sledi vrstica z višino vozila, ki bo poslano na pot.

Lahko predpostavite, da bo za povezani mesti M1 in M2 samo en zapis v vhodni datoteki (če je v datoteki podana povezava iz M1 v M2, ne bo eksplisitno podane povezave iz M2 v M1, saj so povezave dvosmerne).

V izhodno datoteko zapisi število različnih poti med izhodiščnim in ciljnimi mestoma (to je $ID1$ in $ID2$).

Implementirajte razred **Naloga6**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke in sestavi izhodno datoteko.

Primer:

Vhodna datoteka	Izhodna datoteka
5 1,3,2 1,4,1 3,4,3 3,2,-1 4,2,3 1,2 2	2

Razlaga primera: Iz mesta 1 do mesta 2 lahko pridemo po poti 1-3-2 ali 1-3-4-2. Iz mesta 1 v mesto 4 ne moremo neposredno zaradi prenizkega predora.

Če bi v zadnji vrstici vhodne datoteke bila vrednost 1 (namesto 2), bi lahko ubrali tudi poti 1-4-2 in 1-4-3-2, tako da bi v izhodno datoteko zapisali vrednost 4.

Naloga 7

Želimo poskrbeti za lepši izris splošnih binarnih dreves (med elementi ni nujna urejenost). V ta namen želimo vsakemu vozlišču v drevesu določiti koordinati (x,y) v izrisu. Veljajo naslednja pravila:

- Koordinata y je enaka globini vozlišča v drevesu. Koren je na globini 0.
- Za vsako poddrevo s korenom k velja:
 - Koordinate x vozlišč v levem poddrevesu so manjše od koordinate x korena k .
 - Koordinate x vozlišč v desnem poddrevesu so večje od koordinate x korena k .
- Noben par vozlišč v drevesu nima enakih koordinat x .
- Zaloga vrednosti koordinat x je od 0 do $N - 1$, pri čemer je N število vozlišč v drevesu.

Drevo je podano v tekstovni vhodni datoteki. V prvi vrstici je zapisano celo število N , ki označuje število vozlišč v drevesu. V naslednjih N vrsticah so zapisani podatki o vozliščih v poljubnem vrstnem redu. Posamezna vrstica je oblike ID, V, ID_L, ID_R (vse vrednosti so cela števila). ID predstavlja identifikator vozlišča; V je vrednost, zapisana v tem vozlišču; ID_L je identifikator levega sina; ID_R je identifikator desnega sina. Za identifikatorje velja, da so enolično določeni. Identifikator -1 označuje prazno poddrevo (vozlišče nima ustreznegata).

Izhodna datoteka naj vsebuje N vrstic. Posamezna oblika naj bo sestavljena iz podatkov: vrednost v vozlišču, x koordinata vozlišča, y koordinata vozlišča (ločeno z vejicama). Vozlišča izpisujte v vrstnem redu, ki ustreza obhodu drevesa po nivojih - vozlišča izpisujemo nivo po nivo (vozlišča znotraj istega nivoja naj bodo izpisana od skrajno levega proti skrajno desnemu).

Implementirajte razred **Naloga7**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (`args[0]` in `args[1]`), prebere vhodne podatke in sestavi izhodno datoteko.

Primer:

Vhodna datoteka:	Izhodna datoteka:
5 14,5,-1,210 302,2,14,-1 42,1,302,8 210,3,-1,-1 8,4,-1,-1	1,3,0 2,2,1 4,4,1 5,0,2 3,1,3

Izhodna datoteka ustreza naslednji razporeditvi vozlišč ob izrisu:

x	0	1	2	3	4
y				1	
0					
1			2		4
2	5				
3		3			

Naloga 8

Logistično podjetje razvaža blago med N mesti, ki so povezana z M dvosmernimi cestnimi odseki z zanimi dolžinami v kilometrih. Velja, da za poljubni mesti obstaja zaporedje cestnih odsekov, ki ju povezuje. Podjetje želi avtomatizirati svojo dejavnost z uporabo (N-1) avtonomnih električnih vozil. Vsako avtonomno vozilo je mogoče naučiti, da izvaja prevoz na enem cestnem odseku v obe smeri (tj. zagotavlja povezavo med mestoma). Vozila imajo omejen doseg, ki predstavlja najdaljšo dolžino cestnega odseka, ki ga vozilo še lahko prevozi z vgrajeno baterijo. Vozila imajo tudi določeno verjetnost, da bo med vožnjo prišlo do okvare. Ob vsakem prihodu vozila v mesto se vgrajena baterija napolni.

Vsa vozila imajo enak doseg D in enako verjetnost P okvare na prevoženih 1000 kilometrov.

Cilj naloge je vozilom dodeliti cestne odseke na tak način, da je omogočen transport med vsemi mesti (bodisi neposredno bodisi preko drugih mest), pri čemer pa naj bo skupna verjetnost okvare čim manjša. Če zaradi podanih omejitev takšna dodelitev ne obstaja in posledično transport med nekaterimi pari mest ni možen, mora program to situacijo prepoznati.

V vhodni datoteki je v prvi vrstici podano število cestnih odsekov M. V naslednjih M vrsticah sledijo opisi odsekov v obliki ID, ID_M1, ID_M2, C . ID predstavlja identifikator cestnega odseka; ID_M1 in ID_M2 sta identifikatorja mest, ki ju ta odsek povezuje; C pa predstavlja dolžino tega odseka. Vsi podatki v opisu cestnega odseka so cela števila. Lahko privzamete, da dve mestni povezuje največ en cestni odsek. Opisom cestnih odsekov sledi vrstica, ki vsebuje celo število D in predstavlja doseg vozil. V naslednji vrstici je zapisana verjetnost napake na 1000km, ki je podano kot decimalno število (kot decimalno ločilo je uporabljeni pik). Opozorilo: število mest N (in s tem število avtonomnih vozil N-1) lahko razberete iz podanih podatkov, saj do vseh mest pelje vsaj ena cesta.

V izhodno datoteko zapišite identifikatorje cestnih odsekov z dodeljenimi vozili, ločene z vejico. Identifikatorji naj bodo zapisani v naraščajočem vrstnem redu. Če naloga ni rešljiva, v izhodno datoteko zapišite samo vrednost -1.

Implementirajte razred **Naloga8**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke in sestavi izhodno datoteko.

Primer:

Vhodna datoteka:	Izhodna datoteka:
5 1,1,2,2 2,1,4,1 3,2,4,3 4,2,3,5 5,3,4,3 3 0.00001	1,2,5

Naloga 9

Podana je množica oseb $M = \{o_1, o_2, \dots, o_m\}$ in zaporedje dejstev o prijateljstvih $N = \{p_1, p_2, \dots, p_n\}$. Vsako prijateljstvo p_i je zapisano v obliki para $\langle o_{i1}, o_{i2} \rangle$ in določa, da sta osebi o_{i1} in o_{i2} prijatelja. Na podlagi zaporedja dejstev N lahko osebe iz M razdelimo v skupine prijateljev. Privzemite, da velja, da je prijatelj mojega prijatelja tudi moj prijatelj.

V zaporedju N je lahko navedba nekaterih dejstev odveč, saj te sledijo iz predhodnih dejstev. Naloga je odvečna dejstva identificirati in jih zapisati v izhodno datoteko. Odvečne trditve identificiramo tako, da dejstva iz N upoštevamo **v podanem vrstnem redu**.

Vhodna datoteka v prvi vrstici vsebuje celo število P , ki določa število dejstev o prijateljstvih. V naslednjih P vrsticah so zapisani pari oseb $O1, O2$, ki sta v prijateljskem odnosu. Pri tem velja, da sta $O1$ in $O2$ pozitivni celi števili, ločeni z vejico.

V izhodno datoteko izpišite identificirana odvečna dejstva, po eno dejstvo na vrstico.

Implementirajte razred **Naloga9**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke in sestavi izhodno datoteko.

Primer:

Vhodna datoteka:	Izhodna datoteka:
6 1, 2 4, 2 5, 6 1, 4 3, 4 1, 3	1, 4 1, 3

Razlaga primera: trditev $\langle 1, 4 \rangle$ sledi iz dejstev $\langle 1, 2 \rangle$ in $\langle 4, 2 \rangle$, trditev $\langle 1, 3 \rangle$ pa iz dejstev $\langle 1, 2 \rangle$, $\langle 4, 2 \rangle$ in $\langle 3, 4 \rangle$.

Naloga 10

Podan je usmerjen graf z n vozlišči in m povezavami, kjer je vsaki povezavi (a,b) pripisana cena $c(a,b)$. Vsako vozlišče ima dodeljeno oznako id (pozitivno celo število). Želimo poiskati pot z najmanjo vsoto cen povezav od začetnega vozlišča s do končnega vozlišča t .

Naloge ne bomo reševali z Dijkstrovim algoritmom, ampak s postopkom, ki ponazarja premikanje agenta po grafu. Pri premikanju agent upošteva oceno razdalje vozlišč do cilja t (v nadaljevanju označena kot h). Na začetku postopka je ocena h za vsa vozlišča enaka 0.

Iskanje optimalne poti se izvaja v iteracijah. Vsako iteracijo agent prične v vozlišču s in se premika po povezavah, dokler ne prispe v vozlišče t , kar predstavlja konec iteracije. Med premiki agent posodablja ocene h obiskanih vozlišč. **V eni iteraciji agent obišče posamezno vozlišče kvečjemu enkrat.** Po zaključku ene iteracije se agent prestavi v izhodiščno vozlišče s in se prične nova iteracija. Celoten postopek se zaključi, ko agent dvakrat zapovrstjo ubere isto pot od s do t , ne da bi se ob tem spremenile ocene h obiskanih vozlišč.

Agent na svoji poti od s do t v vsakem vozlišču a izbira, v katero vozlišče so bo premaknil v naslednjem koraku. Najprej za vsa sosednja, **v trenutni iteraciji še ne obiskana**, vozlišča b (tista, v katera lahko pride z enim premikom) izračuna vsoto $v(b) = c(a,b) + h(b)$ (vrednost $v(b)$ predstavlja seštevek cene premika iz a v b in ocenjene razdalje med b in t). Izmed vseh sosednjih vozlišč agent izbere tisto z **najmanjo vrednostjo $v(b)$** , poimenujmo ga (tisto vozlišče) b_min . V primeru, da ima več vozlišč enako oceno $v(b)$, se izbere tisto z **najmanjo oznako id** . Agent se nato premakne v vozlišče b_min . Obenem se v primeru, da je $h(a)$ **manjše od $v(b_min)$** , nastavi $h(a) = v(b_min)$.

Kadar agent med premiki pride v vozlišče, ki ni cilj t in nima izhodnih povezav, se ocena h tega vozlišča nastavi na neskončno, trenutna iteracija pa se zaključi. Iteracija se prav tako zaključi v primeru, ko agent pride v **vozlišče, ki ni ciljno in ima vsaj eno izhodno povezavo, ampak s premiki ne more nadaljevati, ker so vsa sosednja vozlišča že obiskana**. V tem primeru se ob zaključku iteracije ocena h zadnjega vozlišča ne posodobi.

V vhodni datoteki je v prvi vrstici podano število povezav M . V naslednjih M vrsticah sledijo opisi povezav v obliku treh celih števil, ločenih z vejico. Zapis ID_V1, ID_V2, C predstavlja usmerjeno povezavo iz vozlišča z oznako ID_V1 v vozlišče z oznako ID_V2 s ceno C . Opisom povezav sledi vrstica oblike ID_S, ID_T , ki določa izhodiščno (ID_S) in ciljno vozlišče (ID_T).

V izhodno datoteko za vsako iteracijo izpišite vozlišča, ki jih je obiskal agent (vključno z zadnjima dvema, identičnima, iteracijama).

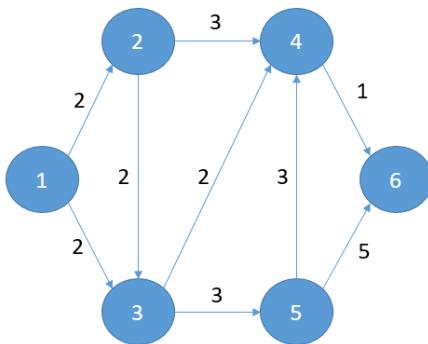
Implementirajte razred **Naloga10**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke in sestavi izhodno datoteko.

Primer:

Vhodna datoteka:	Izhodna datoteka:
9 1,2,2 1,3,2 2,3,2 2,4,3 3,4,2 3,5,3 5,4,3 5,6,5	1,2,3,4,6 1,2,3,4,6 1,3,4,6 1,3,4,6

$4, 6, 1$	
$1, 6$	

Razlaga primera:



Na začetku velja $h(1) = h(2) = h(3) = h(4) = h(5) = h(6) = 0$. Agent se nahaja v vozlišču 1 in oceni neposredno dosegljivi vozlišči 2 in 3. Dobljeni oceni sta $v(2) = 2 + 0 = 2$ in $v(3) = 2 + 0 = 2$. Vozlišči sta ocenjeni enako, zato se agent premakne v vozlišče 2, ki ima manjšo oznako. Obenem agent nastavi novo hevristično oceno za vozlišče 1 na $h(1) = 2$. Iz vozlišča 2 sta neposredno dosegljivi vozlišči 3 in 4. Agent ju oceni in dobi $v(3) = 2 + 0 = 2$ ter $v(4) = 3 + 0 = 3$. Ker je vozlišče 3 bolje ocenjeno, se agent premakne vanj in nastavi $h(2) = 2$. Iz vozlišča 3 sta neposredno dosegljivi vozlišči 4 in 5. Agent ju oceni in dobi $v(4) = 2 + 0 = 2$ ter $v(5) = 3 + 0 = 3$. Sedaj je vozlišče 4 tisto bolje ocenjeno, zato se agent premakne vanj in nastavi $h(3) = 2$. Iz vozlišča 4 je dosegljivo le vozlišče 6. Agent ga oceni in dobi $v(6) = 1 + 0 = 1$, se premakne vanj ter nastavi $h(4) = 1$. S tem se zaključi prva iteracija in se v izhodno datoteko zapise sekvenca 1,2,3,4,6.

Na začetku druge iteracije je agent spet v vozlišču 1. Agent oceni vozlišči 2 in 3 in dobi $v(2) = 2 + 2 = 4$ ter $v(3) = 2 + 2 = 4$. Ponovno izbere premik v vozlišče 2 in nastavi $h(1) = 4$. V vozlišču 2 agent oceni vozlišči 3 in 4 ter dobi $v(3) = 2 + 2 = 4$ in $v(4) = 3 + 1 = 4$. Ker sta oceni enaki, agent izbere vozlišče 3 (manjša oznaka) in se premakne vanj, obenem nastavi $h(2) = 4$. V vozlišču 3 agent oceni vozlišči 4 in 5 in dobi $v(4) = 2 + 1 = 3$ ter $v(5) = 3 + 0 = 3$. Oceni sta enaki, zato agent izbere vozlišče 4 (manjša oznaka) in se premakne vanj, obenem nastavi $h(3) = 3$. V vozlišču 4 agent izračuna $v(6) = 1 + 0 = 1$ ter se premakne v vozlišče 6, ocena $h(4)$ ostane nespremenjena. S tem se je zaključila druga iteracija in se v izhodno datoteko zapise sekvenca 1,2,3,4,6. Agent je dvakrat zapored izbral enako pot, vendar se je med sprehodom vsaj ena hevristična ocena $h(v)$ spremenila, zato postopek še vedno ni zaključen.

Na začetku tretje iteracije agent oceni vozlišči 2 in 3 ter dobi $v(2) = 2 + 4 = 6$ in $v(3) = 2 + 3 = 5$, se premakne v vozlišče 3 in nastavi $h(1) = 5$. V vozlišču 3 oceni vozlišči 4 in 5 ter dobi $v(4) = 2 + 1 = 3$ in $v(5) = 3 + 0 = 3$. Izbere vozlišče 4 (manjša oznaka) ter se premakne vanj, ocena $h(3)$ pa ostane nespremenjena. Iz vozlišča 4 je neposredno dosegljivo samo vozlišče 6, agent ga oceni in dobi $v(6) = 1 + 0 = 1$. Agent se premakne v vozlišče 6, ocena $h(4)$ ostane nespremenjena, iteracija se zaključi in se v izhodno datoteko zapise sekvenca 1,3,4,6.

Četrta iteracija se izvede na enak način kot tretja, s to razliko, da se nobena hevristična ocena $h(v)$ ne spremeni. To je pogoj za konec postopka. V izhodno datoteko zapisemo izbrano pot 1,3,4,6 in zaključimo.