Algoritmi in podatkovne strukture 1 2019/2020

Seminarska naloga 2

Rok za oddajo programske kode prek učilnice je **sobota**, **11. 1. 2020.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **13. 1. – 17. 1. 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.
- 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 6

Prevozno podjetje ima v lasti N vozil in zaposluje M voznikov, pri čemer velja M > N. Podjetje vodi evidenco o opravljenih vožnjah tako, da beležijo zapise oblike X-Y. X in Y predstavljata šifri voznika oziroma vozila (oba tipa String), pri čemer vrstni red ni določen. Na primer, zapis ab123-73f lahko pomeni, da je voznik z oznako ab123 vozil vozilo z oznako 73f, ali pa, da je voznik z oznako 73f vozil vozilo z oznako ab123. Način zapisa ni bil problematičen, ker so v podatkovni bazi imeli šifranta voznikov in vozil. Težava je nastala takrat, ko sta bila ob nadgradnji programske opreme po pomoti izbrisana šifranta. Naloga je za vsako šifro določiti, ali označuje voznika ali vozilo. Pri tem predpostavite, da so vse šifre enolično določene, da je vsak voznik vozil vsaj eno vozilo, ter da je vsako vozilo bilo uporabljeno vsaj enkrat.

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

Prva vrstica tekstovne vhodne datoteke vsebuje celo število V. Naslednjih V vrstic vsebuje zapise oblike A-B, pri čemer sta A in B poljubno dolga znakovna niza, sestavljena iz malih črk angleške abecede in števk 0-9 .

V tekstovno izhodno datoteko zapišite oznake voznikov. Vsako oznako zapišite v ločeno vrstico. Vrstni red izpisovanja ni pomemben. Če naloga ni rešljiva oziroma ne obstaja enolična rešitev (v smislu ločitve na voznike in vozila) , v izhodno datoteko samo zapišite -1.

Prvi primer:

Vhodna datoteka:	Izhodna datoteka:
10 gg-2fd 432h-gg gg-dfr2 2fd-nbj4 2fd-5s nbj4-432h nbj4-dfr2 432h-5s 5s-dfr2 432h-nmn1	gg 5s nbj4 nmn1

Drugi primer:

Vhodna datoteka:	Izhodna datoteka:
3 gg-2fd 432h-gg 2fd-432h	-1

Naloga 7

Podana je avto karta v obliki neusmerjenega grafa. Mesta so vozlišča grafa, ceste med mesti so pa povezave grafa. Nahajamo se v mestu A in želimo iti na izlet, na katerem ne bomo obiskali istih mest večkrat. Izlet želimo zaključiti v izhodiščnem mestu. Naše prevozno sredstvo ima goriva za B kilometrov. Naloga je prešteti vse različne izlete, ki jih lahko izvedemo. Pri tem velja, da sta izleta različna, če **množici** obiskanih mest nista enaki.

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

V tekstovni vhodni datoteki je v prvi vrstici zapisano celo število N, ki predstavlja število (dvosmernih) cest. V naslednjih N vrsticah so zapisani podatki o cestah. Vsaka vrstica je oblike M1,M2,D (prvi dve vrednosti sta celi števili, tretja vrednost je tipa double). M1 in M2 sta oznaki mest, ki ju ta cesta povezuje; D je pa dolžina cestnega odseka v kilometrih. Podatkom o cestah sledi vrstica oblike A,B, ki določa izhodiščno mesto in doseg vozila v kilometrih.

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 eksplicitno podane povezave iz M2 v M1, saj so povezave dvosmerne).

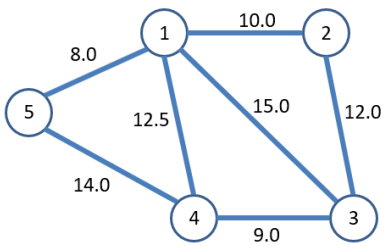
V izhodno datoteko zapišite število različnih izletov, ki jih lahko opravimo pod danimi pogoji.

Primer:

Vhodna datoteka:	Izhodna datoteka:
7 1,2,10.0 1,3,15.0 1,4,12.5 1,5,8.0 2,3,12.0 3,4,9.0 4,5,14.0 1,40.0	7

Razlaga primera

Cestno omrežje izgleda takole:



Ob podanem izhodiščnem mestu z oznako 1 in dosegom 40 km, lahko izvedemo naslednje izlete:

1 → 2 → 1 (prevožena razdalja 20 km)
1 → 2 → 3 → 1 (prevožena razdalja 37 km)
1 → 3 → 1 (prevožena razdalja 30 km)
1 → 3 → 4 → 1 (prevožena razdalja 36.5 km)
1 → 4 → 1 (prevožena razdalja 25 km)
1 → 4 → 5 → 1 (prevožena razdalja 34.5 km)
1 → 5 → 1 (prevožena razdalja 16 km)

Izleti z izhodiščem v mestu z oznako 1, ki so predolgi za podani domet:

1 → 2 → 3 → 4 → 1 (razdalja 43.5 km)
1 → 2 → 3 → 4 → 5 → 1 (razdalja 53 km)
1 → 3 → 4 → 5 → 1 (razdalja 46 km)

Naloga 8

Podan je višinski zemljevid dimenzij A*A točk (A je potenca števila 2). Za podano višino vodne gladine želimo hitro izračunati delež potopljenih točk (to je točk, katerih nadmorska višina je manjša ali enaka gladini vode). Problem bomo reševali z uporabo drevesne strukture, v kateri vsako vozlišče predstavlja kvadraten segment znotraj zemljevida. Korensko vozlišče predstavlja celoten zemljevid. Vsa notranja vozlišča drevesa imajo natanko štiri sinove, ki predstavljajo razdelitev tega vozlišča na enako velike kvadrante (zgoraj levo, zgoraj desno, spodaj levo, spodaj desno). V vsakem vozlišču hranimo minimalno in maksimalno nadmorsko višino točk, ki jih to vozlišče pokriva. List drevesa pokriva bodisi področje velikosti 1x1 bodisi imajo vse točke v njem enako nadmorsko višino.

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

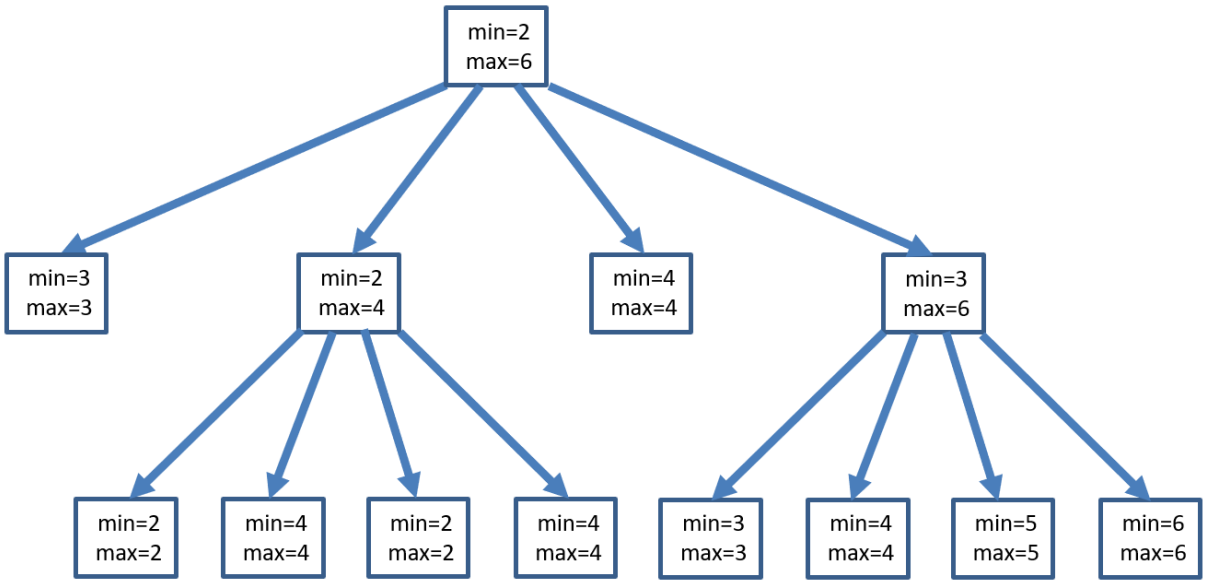
Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število A (dimenzija zemljevida). Vsaka izmed naslednjih A vrstic vsebuje A z vejico ločenih višin (višine so podane kot cela števila). Sledi celo število B, kateremu sledi B vrstic z višinami vodne gladine (celo število). Za vsako podano višino vodne gladine v tekstovno izhodno datoteko zapišite število potopljenih točk in (ločeno z vejico) število obiskanih vozlišč v predpisani strukturi (vozlišče se šteje za obiskano takoj, ko preberete njegovo minimalno ali maksimalno nadmorsko višino). Izračun je potrebno izvesti na optimalen način (želimo obiskati minimalno število vozlišč, da izvemo rezultat).

Primer:

Vhodna datoteka:	Izhodna datoteka:
4 3,3,2,4 3,3,2,4 4,4,3,4 4,4,5,6 3 1 3 4	0,1 7,13 14,9

Razlaga primera

Na podlagi vhodnih podatkov zgradimo drevesno strukturo, ki izgleda takole:



V vsakem vozlišču hranimo razpon vrednosti znotraj področja, ki ga vozlišče pokriva. Korensko vozlišče pokriva celoten zemljevid, ki je v konkretni nalogi velikosti 4x4 = 16 točk. Sinovi korenskega vozlišča pokrivajo področja velikosti 2x2 = 4 točke. Vnuki korenskega vozlišča pa pokrivajo eno samo točko. Sedaj lahko začnemo z obdelavo poizvedb.

Pri prvi poizvedbi je višina vodne gladine nastavljena na 1. Že po pregledu vrednosti v korenskem vozlišču lahko sklepamo, da ne bo potopljena niti ena točka, saj so vse točke na zemljevidu nad višino 1. Zato v prvo vrstico izhodne datoteke zapišemo 0 (število potopljenih točk) in 1 (pregledali smo samo korensko vozlišče).

Pri drugi poizvedbi je višina vodne gladine nastavljena na 3. Pregledamo korensko vozlišče in ugotovimo, da bo delno potopljeno (višina vodne gladine se nahaja med min in max vrednostjo v korenu), zato moramo pregledati še njegove sinove. Prvi (najbolj levi) sin korenskega vozlišča bo v celoti potopljen – to področje ustreza štirim točkam. Drugi sin bo delno potopljen, zato pregledamo še njegove sinove. Potopljena bosta prvi in tretji sin – to sta še dve potopljeni točki. Tretji sin korenskega vozlišča bo v celoti nad vodno gladino. Četrti sin bo delno potopljen, zato pregledamo še njegove sinove. Potopljen bo samo njegov prvi sin, ki ustreza eni potopljeni točki. V izhodno datoteko zapišemo 7 (4+2+1 potopljenih točk) in 13 (pregledali smo vsa vozlišča drevesa).

Pri tretji poizvedbi je višina vodne gladine nastavljena na 4. Sedaj bodo popolnoma potopljeni prvi, drugi in tretji sin korenskega vozlišča, kar predstavlja 12 točk. Četrti sin bo delno potopljen, zato pregledamo še njegove sinove. Potopljena bosta dva sina, kar prinese še dve potopljeni točki. V izhodno datoteko zapišemo 14 (12+2 potopljenih točk) in 9 (pregledali smo vsa vozlišča drevesa razen potomcev drugega sina korenskega vozlišča).

Naloga 9

Podana je množica N-točk v 2d prostoru. Vsaka točka je podana s koordinatama (x, y). Točke želimo razdeliti v K skupin. Razdalja med skupinama S1 in S2 je definirana kot evklidska razdalja med najbližjima točkama t1 iz S1 in t2 iz S2. Naloga je poiskati dodelitev točk skupinam tako, da so razdalje med skupinami maksimalne. Ideja: na začetku vsaka točka predstavlja svojo skupino, nato združujemo najbližje skupine dokler ne dobimo zahtevanega števila skupin.

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

Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število N, ki predstavlja število točk. V naslednjih N vrsticah sledijo koordinate točk. Posamezna koordinata je podana v obliki X,Y. Pri tem velja, da sta X in Y števili tipa double. Vsem točkam se implicitno dodelijo enolične oznake (id): prva točka ima id=1, druga točka ima id=2 in tako naprej to zadnje točke, ki ima id=N. V zadnji vrstici vhodne datoteke je zapisano celo število K, ki določa število zahtevanih skupin.

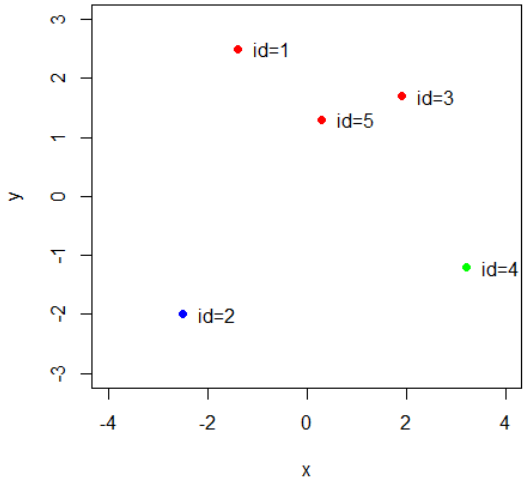
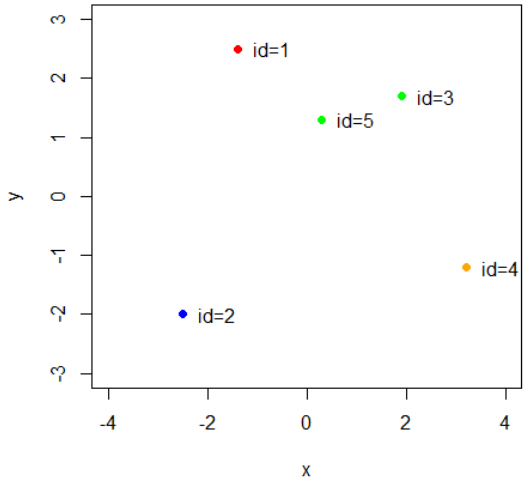
Tekstovna izhodna datoteka naj vsebuje K vrstic. V vsaki vrstici naj bodo izpisane oznake točk, ki pripadajo isti skupini. Oznake točk naj bodo izpisane v naraščajočem vrstnem redu in ločene z vejico. Vrstni red izpisanih skupin naj bo prav tako urejen naraščajoče glede na najmanjšo oznako točke, ki pripada skupini. To pomeni, da bo v prvi vrstici izpisana skupina, ki vsebuje točko z id=1. V vsaki naslednji vrstici bo izpisana tista skupina, ki vsebuje točko z najmanjšo oznako, ki še ni izpisana v prejšnjih vrsticah. Drugače povedano, prve vrednosti v vsaki izpisani vrstici bodo naraščale od prve do zadnje vrstice.

Prvi primer:

Vhodna datoteka:	Izhodna datoteka:
5 -1.4,2.5 -2.5,-2.0 1.9,1.7 3.2,-1.2 0.3,1.3 3	1,3,5 2 4

Drugi primer (enake točke ampak drugo število skupin):

Vhodna datoteka:	Izhodna datoteka:
5 -1.4,2.5 -2.5,-2.0 1.9,1.7 3.2,-1.2 0.3,1.3 4	1 2 3,5 4

Izris prvega primera (točke ene skupine so izrisane z isto barvo):	Izris drugega primera (točke ene skupine so izrisane z isto barvo):
	

Naloga 10

V vhodni datoteki je podano zaporedje celih števil, ki ustreza vmesnem (inorder) izpisu elementov nekega binarnega drevesa. Za to drevo vemo, da je delno urejeno - za vsako poddrevo velja, da je v korenu največji element tega poddrevesa. Vemo tudi to, da drevo ni nujno poravnano niti delno poravnano.

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, rekonstruira drevo (rešitev je enolično določena) in v izhodno datoteko zapiše elemente drevesa po nivojih.

Primer:

Vhodna datoteka:	Izhodna datoteka:
`7,5,10,3,2,17,1,12,14`	`17,10,14,7,3,12,5,2,1`

Rekonstruirano drevo iz vhodne datoteke izgleda takole: