Naloga 1

Podana je mreža črk in seznam besed. Vsaka beseda iz seznama je vpisana v mrežo v eni izmed osmih smeri: vodoravno, navpično in diagonalno, naprej in nazaj. Naloga je poiskati vse besede iz seznama tako, da posamezna črka v mreži pripada samo eni besedi.

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, poišče besede v mreži in v izhodno datoteko zapiše njihove pozicije.

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.
- V naslednji vrstici je celo število B. Ta predstavlja število besed, ki jih iščemo v mreži.
- V vsaki izmed naslednjih B vrstic je zapisana ena beseda.

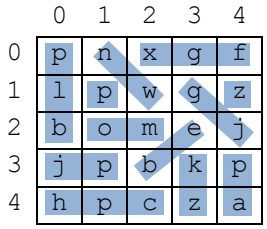
Tekstovna izhodna datoteka naj vsebuje B vrstic. V vsaki vrstici naj bo najprej zapisana beseda, nato še štiri cela števila, ločena z vejico. Prvi par števil V1,S1 določa pozicijo prve črke besede v mreži, drugi par V2,S2 pa pozicijo zadnje črke besede v mreži. **Pozor: indeksiranje začne z 0**.

Pri tej nalogi je možnih več enakovrednih rešitev.

Primer:

Vhodna datoteka:	Izhodna datoteka:
5,5 p,n,x,g,f l,p,w,g,z b,o,m,e,j j,p,b,k,p h,p,c,z,a 12 xgf mo zk jp plb nw hpc p pa z jg be	xgf,0,2,0,4 mo,2,2,2,1 zk,4,3,3,3 jp,3,0,3,1 plb,0,0,2,0 nw,0,1,1,2 hpc,4,0,4,2 p,1,1,1,1 pa,3,4,4,4 z,1,4,1,4 jg,2,4,1,3 be,3,2,2,3

Razlaga primera:



Naloga 2

Pri eni od najpogostejših tehnik mešanja kart (*the overhand shuffle)* karte najprej razdelimo na dva dela. Potem drugi del postopoma vstavljamo v prvi del na izbrano mesto; lahko na začetek, lahko na konec ali pa kam vmes. Postopek večkrat ponovimo.

Napišite program, ki bo mešal karte po tem postopku. Karte so označene s številkami in črkami (namesto barv, jokerjev, kraljev in podobnih znakov). Vsaka karta je označena s številko (<32768) in črko ('A' do 'Z'). Kup kart je predstavljen s seznamom, kjer je na prvem mestu napisana karta na vrhu kupa. V spodnjem primeru je 1A na vrhu kupa, na dnu pa je 3B:

`1A,2B,1C,2E,2K,17A,3A,1A,3B`

Postopek mešanja poteka na naslednji način. Kup se najprej razdelili na dva dela, recimo jima `K1` in `K2`. Potem se nekaj prvih kart iz kupa `K2` vstavi v `K1` na točno določeno mesto. Nato se naslednjih nekaj kart iz kupa `K2` vstavi na isto mesto v `K1` in se postopek ponavlja, dokler se `K2` ne izprazni. V vsakem koraku se vstavi isto število kart, razen v zadnjem, če preostalih kart ni dovolj. Vsako mešanje kart je določeno z vrednostmi *D*, *V* in *S*:

- *D* predstavlja karto, ki razdeli kup na `K1` in `K2`. Pri tem velja, da je *D* po razdelitvi zadnja karta v kupu `K1`. Če take karte v kupu ni, potem se premaknejo vse karte na `K2`, `K1` pa ostane prazen. Če se v kupu nahaja več kart *D*, se ob delitvi upošteva položaj prve pojavitve.

- Karta *V* določa mesto vstavljanja v `K1` (vstavlja se za karto *V*). Če karte v kupu `K1` ni, se vstavlja na začetek kupa `K1`; če jih je več, se spet upošteva prva pojavitev.

- *S* je število kart, ki se vstavijo v eni iteraciji postopka.

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, izvede mešanje in v izhodno datoteko zapiše dobljeno zaporedje kart.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici sta zapisani dve celi števili, ločeni z vejico. Prva vrednost določa število kart v začetnem kupu. Druga vrednost določa število mešanj.

- Druga vrstica vsebuje začetno zaporedje kart. Oznake kart so ločene z vejico.
- Sledijo navodila za mešanje. Vsako navodilo je v svoji vrstici predstavljeno s trojčkom *D*,*V*,*S*.

Tekstovna izhodna datoteka vsebuje zaporedje kart po vseh mešanjih. Oznake kart so ločene z vejico.

Primer:

Vhodna datoteka:	Izhodna datoteka:

`9,1`
`1A,2B,1C,2E,2K,17A,3A,1A,3B`
`2E,1A,3`
	`1A,1A,3B,2K,17A,3A,2B,1C,2E`

Razlaga primera:

Dan je kup devetih kart: `1A,2B,1C,2E,2K,17A,3A,1A,3B`. Podano je samo eno navodilo za mešanje po pravilu *D*=2E, *V*=1A in *S*=3, ki se izvede s sledečimi koraki.

Karte se razdelijo za karto 2E:
`K1: 1A,2B,1C,2E`
`K2: 2K,17A,3A,1A,3B`

Vrednost I=1A pomeni, da se karte vstavljajo za karto 1A. Nastavitev *S*=3 določa, da se prenašajo po tri karte naenkrat iz `K2` v `K1`. Ker je na začetku v `K2` pet kart, se celoten postopek izvede v dveh iteracijah.

Stanje po koncu prve iteracije:

`K1: 1A,2K,17A,3A,2B,1C,2E`
`K2: 1A,3B`

Stanje po koncu druge iteracije:

`K1: 1A,1A,3B,2K,17A,3A,2B,1C,2E`
`K2:`

`K2` je prazen, zato je mešanje zaključeno.

Naloga 3

Implementirajte seznam celih števil, ki podpira naslednje metode:

- public void preslikaj(char op, int val)
- public void ohrani(char op, int val)
- public void zdruzi(char op)

Metoda `void preslikaj(op,val)` se sprehodi čez elemente seznama ter vsak element `e` nadomesti z vrednostjo `e op val`. Parameter `op` lahko zavzame dve možni vrednosti: `'+'` in `'*'`. Drugače povedano, metoda preslikaj vsakemu elementu seznama bodisi prišteje vrednost `val` bodisi ga z `val` pomnoži.

Metoda `void ohrani(op,val)` v seznamu obdrži samo elemente za katere velja `e op val`. Parameter `op` lahko zavzame tri možne vrednosti: `'>'`, `'<'` in `'='`. Drugače povedano, metoda ohrani obdrži le tiste elemente seznama, ki so večji, manjši oziroma enaki parametru `val`.

Metoda `void združi(op)` bodisi sešteje bodisi zmnoži vse elemente seznama, v odvisnosti od vrednosti parametra `op` (`'+'` ali `'*'`). Ob zaključku seznam vsebuje en sam element, ki predstavlja rezultat operacije (seštevek ali zmnožek elementov).

Implementirajte razred **Naloga3**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]). Vhodna datoteka vsebuje začetno stanje seznama in navodila za operacije, ki naj se nad njim izvedejo. Izhodna datoteka vsebuje elemente seznama po končani izvedbi vsake zahtevane operacije.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- V prvi vrstici je zapisano začetno stanje seznama. Vrednosti posameznih elementov so ločene z vejico.
- V drugi vrstici je število operacij.
- Sledijo navodila za operacije. Vsako navodilo je v eni izmed spodnjih oblik:

  p,A,B označuje klic `preslikaj(A,B)`
  o,A,B označuje klic `ohrani(A,B)`
  z,A označuje klic `zdruzi(A)`

Tekstovna izhodna datoteka vsebuje enako število vrstic, kot je bilo navodil za operacije. Pri tem velja, da i-ta vrstica vsebuje zaporedje elementov, ločenih z vejico, ki predstavljajo stanje seznama po izvedbi i-te operacije.

Primer:

Vhodna datoteka:	Izhodna datoteka:

`5,8,3,2,9,3,4,8`
`4`
`o,>,5`
`p,+,2`
`o,=,10`
`z,*`
	`8,9,8`
	`10,11,10`
	`10,10`
	`100`

Naloga 4

Podatkovna struktura vreča (ang. bag) je vrsta seznama, pri katerem vrstni red elementov ni pomemben, ponavljanje elementov pa je dovoljeno. Primer vreče je denimo seznam z elementi {1,2,1,5,1,3,2}. V tem primeru se element 1 pojavi trikrat, element 2 dvakrat, elementa 3 in 5 pa po enkrat. Enako vrečo bi predstavljal tudi seznam {1,1,1,2,2,3,5}, saj je za vrečo pomembno le, kateri elementi so v njej in kolikokrat se ti pojavijo.

Naloga je implementirati metode, ki izvajajo operacije nad to podatkovno strukturo.

- `ustvari`: ustvari novo vrečo s podanim imenom in vsebino. Na primer, s tem ukazom bi lahko ustvarili vreči A = {1,1,1,2,2,3,5} in B = {1,1,2,2,2,4,5,5}.

- `združi`: v eno vrečo doda vsebino druge vreče. Po združevanju vreče A = {1,1,1,2,2,3,5} z vrečo B = {1,1,2,2,2,4,5,5} bi v vreči A bili naslednji elementi: {1,1,1,1,1,2,2,2,2,2,3,4,5,5,5}.

- `razlika`: iz ene vreče odstrani elemente druge vreče. Po izračunu razlike vreče A = {1,1,1,2,2,3,5} z vrečo B = {1,1,2,2,2,4,5,5} bi v vreči A ostali {1,3}.

- `skupno`: v vreči obdrži le elemente, ki so skupni v obeh vrečah. Izračun preseka vreče A = {1,1,1,2,2,3,5} z vrečo B = {1,1,2,2,2,4,5,5} bi v vreči A pustil {1,1,2,2,5}.

- `poreži`: število pojavitev elementov v vreči omeji na podano vrednost. Če vrečo A = {1,1,1,1,2,2,2,5} porežemo s konstanto 2 dobimo kot rezultat A = {1,1,2,2,5}.

- `obdrži`: v vreči ohrani elemente, ki se v njej pojavijo vsaj tolikokrat kot je vrednost podane konstante. Klic te funkcije nad vrečo A = {1,1,1,2,3,3,4,5} s konstanto 2 nam da novo stanje vreče A = {1,1,1,3,3}.

- `izpiši`: izpiše trenutno vsebino vreče.

Implementirajte razred **Naloga4**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]). Vhodna datoteka vsebuje opise operacij nad vrečami. V izhodni datoteki naj bodo izpisani rezultati operacij "izpiši".

Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število N, ki predstavlja število ukazov. V naslednjih N vrsticah sledijo opisi posameznih ukazov. Ukaz je lahko v enem izmed sledečih formatov:

- `U,V,`E_1`:`N_1`,`E_2`:`N_2`,...,`E_K`:`N_K - ustvari vrečo z imenom V, ki vsebuje N_1 pojavitev elementa E_1, N_2 pojavitev elementa E_2, ... , N_K pojavitev elementa E_K. Lahko predpostavite, da bodo elementi E_1,...,E_K in imena vreč V vedno cela števila. Prav tako naj velja, da bodo E_1,...,E_K med seboj različni.

- `Z,V1,V2` - vreča V1 je rezultat združevanja vreč V1 in V2.

- `R,V1,V2` - vreča V1 je rezultat izračuna razlike med vrečama V1 in V2.

- `S,V1,V2` - vreča V1 vsebuje skupne elemente vreč V1 in V2.

- `P,V,C` - v vreči V ostanejo elementi, ki ostanejo po klicu funkcije "poreži" s konstanto C

- `O,V,C` - rezultat operacije je nova vsebina vreče V, ki predstavlja rezultat klica "obdrži" s konstanto C nad vrečo V

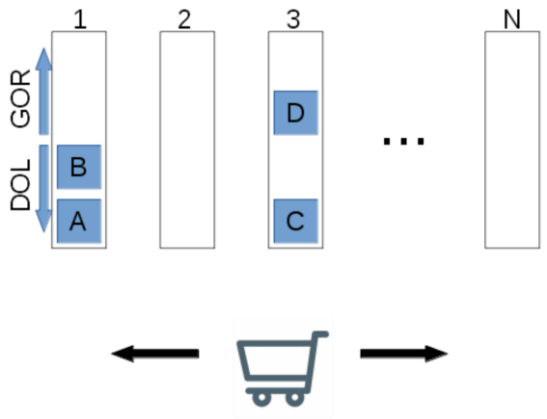
- `I,V` - v izhodno datoteko zapiše vsebino vreče V v formatu E_1:N_1,E_2:N_2,E_3:N_3,...,E_K:N_K pri čemer so E_1,...,E_K elementi, N_1,...,N_K pa njihova števila pojavitev. Pri izpisu naj velja $E_1 < E_2 < ... < E_K$.

Primer:

Vhodna datoteka:	Izhodna datoteka:
12 U,1,3:2,1:5,9:1,7:4 U,2,3:1,2:5,1:3,4:1,6:1 Z,1,2 I,1 O,1,4 I,1 P,2,3 I,2 S,1,2 I,1 R,2,1 I,2	1:8,2:5,3:3,4:1,6:1,7:4,9:1 1:8,2:5,7:4 1:3,2:3,3:1,4:1,6:1 1:3,2:3 3:1,4:1,6:1

Naloga 5

Na spodnji sliki je prikazan načrt skladišča.



Imamo N odstavnih trakov, na vsak trak lahko shranimo P enako velikih predmetov. Skladišče upravljamo 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.

Skladišče upravljamo z naslednjimi ukazi:

1) PREMIK i; premakne voziček do i-tega traku ($1 \leq i \leq N$).

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.

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 sprejme začetno in končno konfiguracijo skladišča ter poišče **najkrajše zaporedje** ukazov, ki vodijo od začetne do končne konfiguracije. Robotski voziček naj **vedno začne na lokaciji 1**. 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:

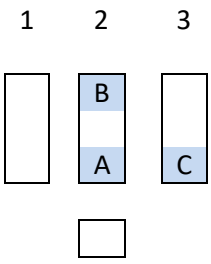
- V prvi vrstici bosta podana število trakov (N ≤ 5) in dolžina trakov (P ≤ 5).
- V naslednjih N vrsticah je opisana vsebina posameznih trakov začetne konfiguracije. Vsaka vrstica se začne s številko traku, za njo pride dvopičje, sledijo oznake zabojev ločene z vejico. Zaboji so označeni z eno črko ('A' do 'Z').
- V naslednjih N vrsticah je opisana vsebina posameznih trakov končne konfiguracije.

Tekstovna izhodna datoteka naj vsebuje najkrajše zaporedje ukazov, ki dosežejo zahtevano ureditev. Vsak ukaz naj bo v svoji vrstici. Pri tej nalogi je možnih več enakovrednih rešitev.

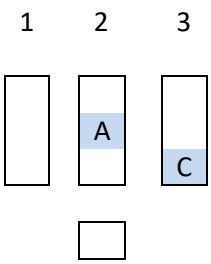
Primer:

Vhodna datoteka:	Izhodna datoteka:
3,3 1: 2:A,,B 3:C 1:C 2:A 3:	PREMIK 2 GOR DOL PREMIK 3 NALOZI PREMIK 1 ODLOZI

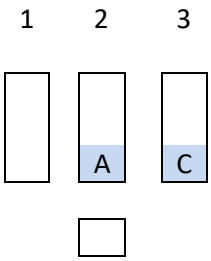
Razlaga primera:



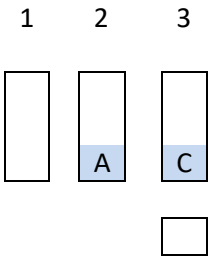
PREMIK 2



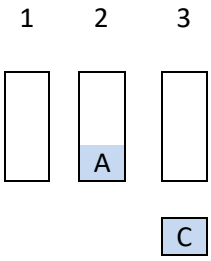
GOR



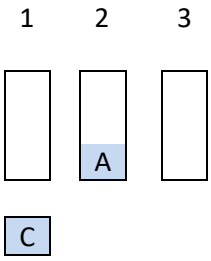
DOL



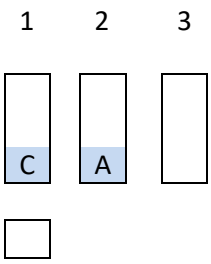
PREMIK 3



NALOZI



PREMIK 1



ODLOZI

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).

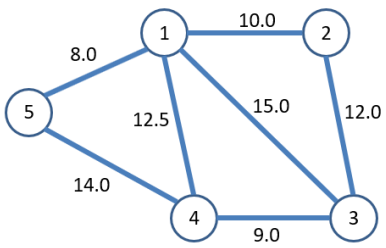
V izhodno datoteko zapišite število različnih izletov, ki jih lahko opravimo pod danimi pogoji.

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]).

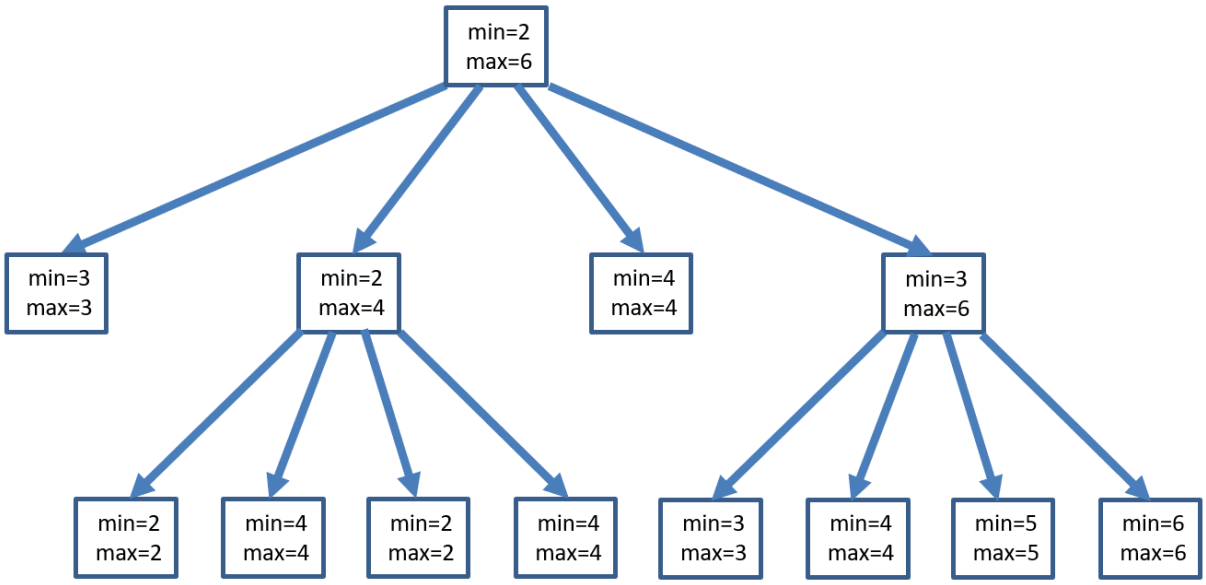
Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število A (dimenzija zemljevida). Vsaka izmed naslednjih A vrstic vsebuje A z vejico ločenih višin (višine so podane kot cela števila). Sledi celo število B, kateremu sledi B vrstic z višinami vodne gladine (celo število). Za vsako podano višino vodne 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.

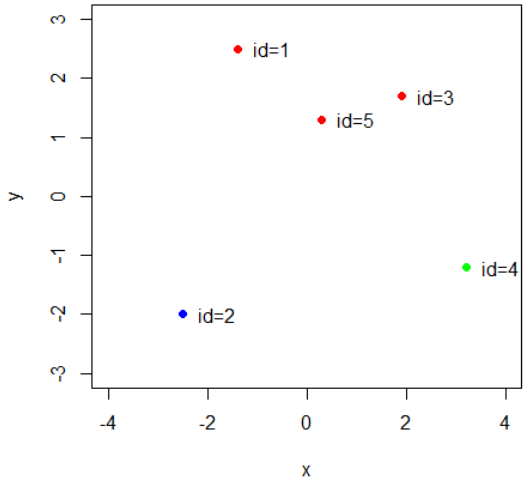
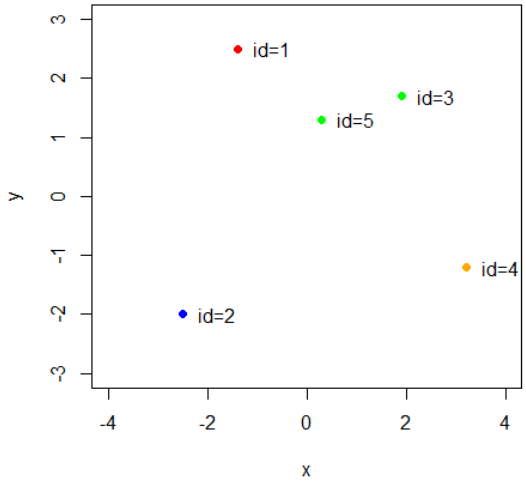
Tekstovna izhodna datoteka naj vsebuje K vrstic. V vsaki vrstici naj bodo izpisane oznake točk, ki pripadajo isti skupini. Oznake točk naj bodo izpisane v naraščajočem vrstnem redu in ločene z vejico. Vrstni red izpisanih skupin naj bo prav tako urejen naraščajoče glede na najmanjšo oznako točke, ki 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: