

Algoritmi in podatkovne strukture 1

2022/2023

Seminarska naloga 2

Rok za oddajo programske kode prek učilnice je **sobota, 7. 1. 2023**.

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **9. 1. – 13. 1. 2023**.

Navodila

Oddana programska rešitev bo avtomatsko testirana, zato je potrebno strogo upoštevati naslednja navodila:

- Uporabite programski jezik java.
- 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.
- Uporablajte kodni nabor **utf-8**.

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

Naloga 6

Podan je neusmerjen povezan graf, kjer vozlišča predstavljajo lokacije v centru mesta, povezave pa poti med njimi. Podana je tudi množica dejstev oblike, A,B,C, kjer sta A in B (naravni števili) oznaki lokacij, C pa predstavlja število turistov, ki se bodo sprehajali od lokacije A do lokacije B. Vsak turist uporablja GPS in vedno izbere najkrajšo pot med dvema lokacijama. Če najkrajša pot ni enolično določena, turisti izberejo tisto, ki se pojavi prej v urejenem seznamu najkrajših poti. Urejen seznam dobimo tako, da razvrstimo poti padajoče najprej po velikosti oznake prve lokacije na poti, potem padajoče po velikosti oznake druge lokacije in tako naprej do oznake zadnje lokacije. Na vsaki lokaciji se nahaja stojnica, ki določenemu odstotku mimoidočih prodaja čaj. Poiščite stojnice, ki bodo prodale največ čaja ter stojnice, ki imajo najboljšo lokacijo (po številu mimoidočih turistov).

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

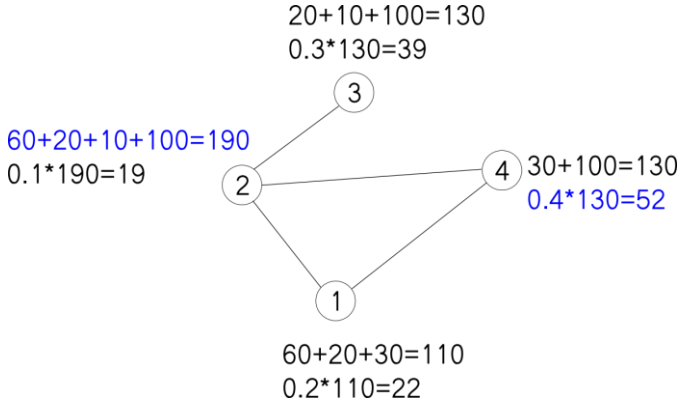
Vhodna datoteka ima v prvi vrstici zapisane uspešnosti prodaje čaja (v obliki deleža mimoidočih) po vrsti za vsako izmed lokacij. Števila so ločena z vejico. V drugi vrstici sta zapisani in z vejico ločeni naravni števili N in M, ki določata število povezav grafa (N) in število dejstev (M). V naslednjih N vrsticah so zapisane povezave neusmerjenega grafa, ločene z vejico. Nato pa v naslednjih M vrsticah dejstva oblike A,B,C.

Izhodna datoteka naj vsebuje eno vrstico v formatu A,B, kjer je A oznaka lokacije, ki prodaja največ čaja, in B oznaka lokacije, ki ima največ mimoidočih turistov. Če je več rešitev, zapišite tisto z najmanjšo oznako (npr., če največ čaja prodajo lokacije 3, 5 in 8, največ mimoidočih pa imata lokaciji 5 in 7, potem v izhodno datoteko zapišite 3,5).

Primer:

Vhodna datoteka:	Izhodna datoteka:
0.2, 0.1, 0.3, 0.4 4, 5 1, 2 1, 4 2, 3 2, 4 1, 2, 60 1, 3, 20 1, 4, 30 2, 3, 10 3, 4, 100	4, 2

Razlaga primera:



Naloga 7

V nekem mestu ima javni prevoz N linij. Po mestu je razporejenih M postajališč, ki so označena s celimi števili. Potek proge posamezne linije je podan s seznamom oznak postajališč. Vse linije so enosmerne. Za dani postaji A (vstop) in B (izstop) poiščite dve poti: pot z najmanjšim številom prestopanj in pot z najmanjšim številom postankov.

Vhodna datoteka v prvi vrstici vsebuje število linij javnega prevoza N . Nato so po vrsticah podana zaporedja postajališč, ki določajo potek posameznih linij od prve postaje do zadnje (ena linija v eni vrstici). Oznake postajališč so ločene z vejicami. V zadnji vrstici vhodne datoteke sta zapisani in z vejico ločeni celi števili A in B , ki predstavljata oznaki vstopnega in izstopnega postajališča.

Implementirajte razred **Naloga7**, ki vsebuje metodo **main**. Metoda prejme poti do vhodne in izhodne datoteke (`args[0]` in `args[1]`), prebere vhodne podatke o linijah javnega prevoza in nato v izhodno datoteko zapiše tri vrednosti, vsako v svoji vrstici.

V prvi vrstici naj bo zapisano minimalno število prestopanj, ki ga potrebujemo za pot od vstopnega postajališča A do izstopnega postajališča B (pri tem nas ne zanima dejanska dolžina poti oziroma število prevoženih postaj). Če sta obe postajališči na isti progi, je število prestopanj enako 0. Če z uporabo linij javnega prevoza ni možno priti iz enega postajališča do drugega, je število prestopanj enako -1.

V drugi vrstici naj bo zapisano število prevoženih postaj na najkrajši vožnji med vstopnim postajališčem A in izstopnim postajališčem B (pri tem nas ne zanima število prestopanj na tej poti). Če sta postajališči sosednji na isti liniji, je dolžina najkrajše poti enaka 1. Če z uporabo linij javnega prevoza ni možno priti iz enega postajališča do drugega, je dolžina najkrajše poti enaka -1.

V tretji vrstici naj bo zapisano število 1, če ima pot z najmanj postajami tudi najmanj prestopanj. Drugače povedano, v tretji vrstici naj bo zapisano število 1, če obstaja pot, ki je hkrati optimalna po kriteriju števila postaj in kriteriju števila prestopanj. V nasprotnem primeru naj bo zapisano število 0. Če z uporabo linij javnega prevoza ni možno priti iz enega postajališča do drugega, naj bo v tretji vrstici zapisano število -1.

Primer:

Vhodna datoteka:	Izhodna datoteka:
6	1
1, 2, 3, 4, 5	3
6, 7	0
8, 7, 4	
2, 6	
5, 7	
7, 1	
1, 7	

Razlaga primera:

Za pot med postajališči $A=1$ in $B=7$ je dovolj samo eno prestopanje (v postajališču 5 prestopimo s prve na peto linijo). Najkrajša pot zahteva 3 postaje ($1 \rightarrow 2$ s prvo linijo, $2 \rightarrow 6$ s četrto linijo, $6 \rightarrow 7$ s tretjo linijo). Pot z najmanj postajami ima dve prestopanji (iz linije 1 na linijo 4 in potem iz linije 4 na linijo 2), medtem, ko je najmanjše število prestopanj 1. Zato v tretjo vrstico zapišemo 0. S šesto linijo ni mogoče priti iz $A=1$ v $B=7$, ker so vse linije enosmerne.

Naloga 8

Turistično podjetje organizira ladijske prevoze med M otoki, ki jih označimo O_1, O_2, \dots, O_m , kjer je O_i naravno število. Posamezno ladijsko povezavo lahko zapišemo v obliki para $\langle O_a, O_b \rangle$, ki nam pove, da ladjica obojestransko povezuje otoka O_a in O_b .

Podjetje je v finančnih težavah in se je odločilo ukiniti redundantne povezave. Povezava med otokoma O_a in O_b je redundantna, če je možno od enega do drugega otoka priti s pomočjo preostalih povezav (na primer, če obstajata povezavi med otokoma O_a in O_c ter O_b in O_c , je povezava med O_a in O_b redundantna).

Napišite program, ki bo v vhodni datoteki prejel zapise o obstoječih ladijskih povezavah, identificiral tiste redundantne, in jih zapisal v izhodno datoteko. Redundantne povezave identificirajte tako, da dejstva o obstoječih povezavah upoštevate **v podanem** vrstnem redu.

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

Vhodna datoteka v prvi vrstici vsebuje celo število P , ki določa število povezav. V naslednjih P vrsticah so zapisani pari otok O_a, O_b , ki sta neposredno povezani z ladijsko povezavo. Pri tem velja, da sta O_a in O_b pozitivni celi števili, ločeni z vejico.

V izhodno datoteko izpišite identificirane redundantne povezave, po eno na vrstico.

Primer:

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

Razlaga primera:

Opazovana povezava $\langle 1, 4 \rangle$ je redundantna zato, ker že obstaja pot med otokoma 1 in 4, ki je sestavljena iz povezav $\langle 1, 2 \rangle$ in $\langle 4, 2 \rangle$ (podanih pred povezavo $\langle 1, 4 \rangle$). Opazovana povezava $\langle 1, 3 \rangle$ je redundantna zato, ker že obstaja pot med otokoma 1 in 3, ki je sestavljena iz povezav $\langle 1, 2 \rangle$, $\langle 4, 2 \rangle$ in $\langle 3, 4 \rangle$ (podanih pred povezavo $\langle 1, 3 \rangle$).

Naloga 9

Podan je višinski zemljevid dimenzij $A \times A$ 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 1×1 bodisi imajo vse točke v njem enako nadmorsko višino.

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

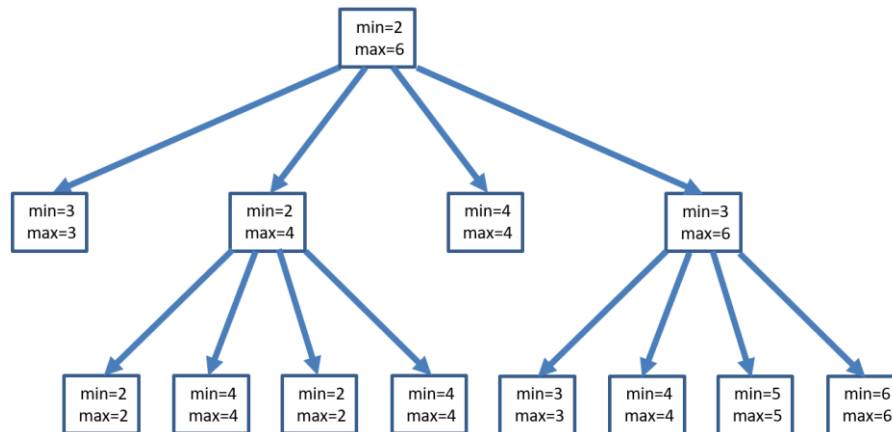
Tekstovna vhodna datoteka v prvi vrstici vsebuje celo število A (dimenzija zemljevida). Vsaka izmed naslednjih A vrstic vsebuje A z vejico ločenih višin (višine so podane kot cela števila). Sledi celo število B , kateremu sledi B vrstic z višinami vodne gladine (celo število). Za vsako podano višino 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 1:

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

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 $4 \times 4 = 16$ točk. Sinovi korenškega vozlišča pokrivajo področja velikosti $2 \times 2 = 4$ točke. Vnuki korenškega 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 korenškem 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 korenško vozlišče).

Pri drugi poizvedbi je višina vodne gladine nastavljena na 3. Pregledamo korenško 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 korenškega 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 korenškega vozlišča bo v celoti nad vodno gladino. Četrty 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 korenškega vozlišča, kar predstavlja 12 točk. Četrty 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 korenškega vozlišča).

Naloga 10

Podani sta drevesi P in T. Vozlišča obeh dreves vsebujejo oznako (mala črka angleške abecede) in so lahko poljubne stopnje (t. j. vsako vozlišče ima lahko 0 ali več potomcev). Vrstni red poddreves vozlišča je pomemben (z drugimi besedami: za vsako vozlišče vemo, kdo je njegov najbolj levi sin, in za vsakega sina vemo, kdo je njegov desni brat). Prav tako velja, da oznake v vozliščih niso unikatne – več vozlišč lahko vsebuje enako oznako.

Drevo P predstavlja vzorec, ki ga iščemo v ciljnim drevesu T. Naloga je poiskati vse pojavitve vzorca P v drevesu T. Pri ujemanju upoštevamo tako oznake kot tudi povezanost vozlišč. Če vzorec P najdemo v drevesu T, pomeni, da obstaja bijektivna preslikava, ki vsakemu vozlišču VP iz P priredi vozlišče VT iz T z enako oznako in stopnjo. Hkrati mora veljati, da se vsi sinovi vozlišča VP preslikajo v sinove vozlišča VT in so enako urejeni.

Implementirajte razred **Naloga10**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (`args[0]` in `args[1]`). Metoda naj prebere vhodne podatke (vzorec P in ciljno drevo T) in v izhodno datoteko zapiše število pojavitev danega vzorca v ciljnim drevesu.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

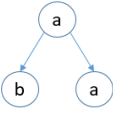
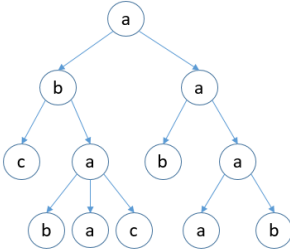
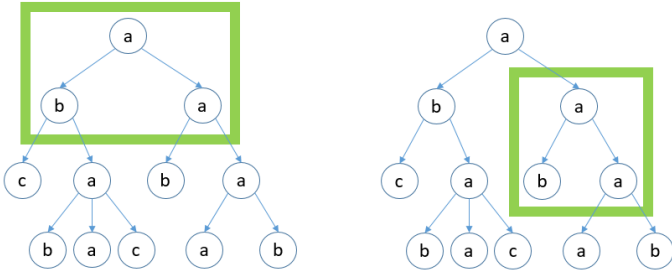
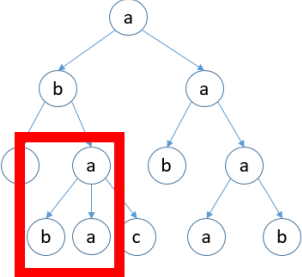
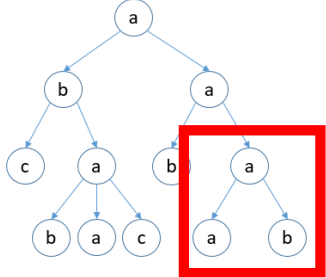
- V prvi vrstici je zapisano celo število N, ki določa število vozlišč v vzorcu P.
- V naslednjih N vrsticah so zapisani podatki o posameznih vozliščih vzorca P. Vsaka vrstica se začne z identifikacijsko številko vozlišča (integer), sledi oznaka v vozlišču (en znak tipa char). Če vozlišče ima sinove, sledijo njihove identifikacijske številke (podane od najbolj levega sina proti desni). Vse vrednosti v eni vrstici so ločene z vejicami.
- Sledi vrstica s celim številom M, ki določa število vozlišč v ciljnim drevesu T.
- V naslednjih M vrsticah so zapisani podatki o posameznih vozliščih ciljnega drevesa T (podani na enak način kot pri vzorcu P).

Tekstovna izhodna datoteka naj vsebuje eno samo vrstico s številom pojavitev vzorca P v ciljnim drevesu T.

Primer:

Vhodna datoteka:	Izhodna datoteka:
3 1, a, 2, 3 2, b 3, a 12 1, a, 2, 3 2, b, 4, 5 3, a, 6, 7 4, c 5, a, 8, 9, 10 6, b 7, a, 11, 12 8, b 9, a 10, c 11, a 12, b	2

Razlaga primera:

<p>Vzorec P</p>	
<p>Ciljno drevo T</p>	
<p>Ujemanja</p>	
<p>Primeri neujemanj</p>	<div style="display: flex; justify-content: space-around;"> <div data-bbox="542 1093 845 1368">  <p>Razlaga: koren označenega poddrevesa je stopnje tri (koren v vzorcu P pa je stopnje dve)</p> </div> <div data-bbox="925 1093 1252 1368">  <p>Razlaga: najbolj levi sin korena označenega poddrevesa ima oznako 'a' (v vzorcu P pa ima oznako 'b')</p> </div> </div>