

Algoritmi in podatkovne strukture 1

2020/2021

Seminarska naloga 2

Rok za oddajo programske kode prek učilnice je **sobota, 9. 1. 2021.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **11. 1. – 17. 1. 2021.**

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 (vključno z javanskimi zbirkami 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 2s za posamezno nalogo.

Naloga 6

Matej bo obiskal prijatelja. Potoval bo z vlakom. Ker si želi ogledati čim več različnih mest, obenem pa ne želi dolgo potovati, se je odločil, da bo poiskal takšno pot, kjer si bodo vsa mesta ločena z natanko enim vmesnim postankom čimbolj različna. Z drugimi besedami: iz trenutnega mesta ne bo šel v mesto, ki je podobno mestu iz katerega je prišel v trenutno mesto. Pomagaj Mateju najti najkrajšo takšno pot do prijatelja. Ena mesto lahko obišče tudi večkrat.

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

Vhodna datoteka:

Mesta so označena s številkami manjšimi od števila vseh mest **N**. V vseh vrsticah so števila ločena s presledki.

- V prvi vrstici so zapisana tri števila: število vseh mest **N**, število vseh povezav med mesti **P** in število skupin podobnih mest z več kot enim mestom **S**.
- V drugi vrstici sta zapisani števili, ki označujeta začetek in konec poti.
- V naslednjih **P** vrsticah so zapisane železniške povezave med mesti ter njihova dolžina, ki je pozitivno naravno število < 10000 .
- V naslednjih **S** vrsticah pa so zapisane skupine podobnih mest. Prvo število v vrstici določa število mest v skupini. Vsa naslednja števila pa določajo mesta v isti skupini. Vsaka vrstica označuje eno skupino mest, ki so si med seboj podobna. Ena mesto je lahko v največ eni skupini. Če je mesto različno od vseh ostalih mest, ne podamo skupine z enim mestom, ampak ga enostavno izpustimo.

Izhodna datoteka:

- Vsebuje eno vrstico, kjer je zapisana dolžina najkrajše poti do prijatelja.

Primer:

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

Razlaga: Najkrajša pot do prijatelja je 0,1,3. Ker sta si mesti 0 in 3 podobni, ta pot ni dovoljena. Najkrajša pot, ki zadostuje pogoju, je tako 0,2,4,1,3. Dolžina te poti je $1+3+2+2=8$.

Naloga 7

Med industrijsko cono in stanovanjskim predelom v mestu vsak dan potuje veliko avtomobilov. Zaradi zastarelosti ceste povzročajo zastoje, zato se je župan odločil, da jih bodo obnovili. Ker pa v občinskem proračunu ni dovolj denarja za obnovo vseh cest, bodo obnovili le tiste predele cest, ki so najbolj obremenjeni. Pomagaj županu poiskati odseke, ki so deli vseh poti med začetno lokacijo **A** in končno lokacijo **B**.

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

Vhodna datoteka:

Vsi predeli mesta (križišča) so zakodirani(a) s številci, dvosmerne povezave med njimi pa kot pari, kjer prvo število predstavlja začetek in drugo konec odseka.

- Prva vrstica določa število vseh predelov (križišč) in število vseh povezav **P**.
- Druga vrstica določa začetni položaj **A** in končni položaj **B**.
- Preostalih **P** vrstic pa določa obojesmerne povezave med različnimi deli mesta (križišči).

Izhodna datoteka:

- Vsebuje naj vse odseke, ki so deli **prav vseh** poti med **A** in **B**. Vsak odsek je zapisan v svoji vrstici kot par, kjer naj bo prvo število manjše od drugega. Odseke izpišite v naraščajočem vrstnem redu prvega števila.

V vhodni in izhodni datoteki so vsa števila zapisana v isti vrstici ločena s presledkom.

Primer:

Vhodna datoteka:	Izhodna datoteka:
6 6 2 0 0 1 0 3 2 3 3 4 3 5 4 5	0 3 2 3

Naloga 8

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 ciljnem 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 V_P iz P priredi vozlišče V_T iz T z enako oznako in stopnjo. Hkrati mora veljati, da se vsi sinovi vozlišča V_P preslikajo v sinove vozlišča V_T in so enako urejeni.

Implementirajte razred `Naloga8`, 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 ciljnem drevesu.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

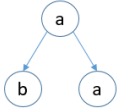
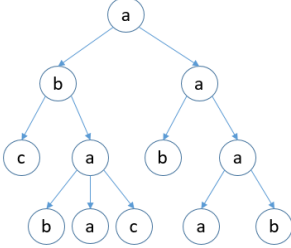
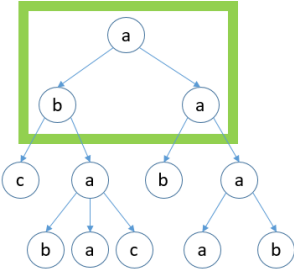
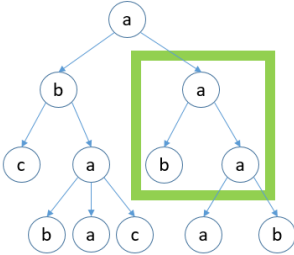
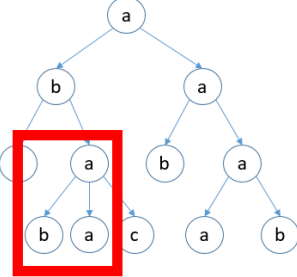
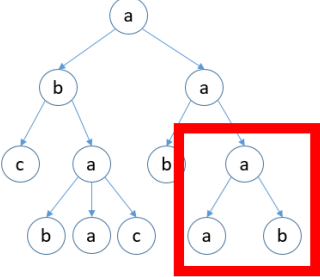
- V prvi vrstici je zapisano celo število N , ki določa število vozlišč v vzorcu P .
- V naslednjih N vrsticah so zapisani podatki o posameznih vozliščih vzorca P . Vsaka vrstica se začne z identifikacijsko številko vozlišča (integer), sledi oznaka v vozlišču (en znak tipa `char`). Če vozlišče ima sinove, 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 ciljnem 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 ciljnem 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

Analiza primera:

Vzorec P	
Ciljno drevo T	
Ujemanja	<div style="display: flex; justify-content: space-around;">   </div>
Primera neujemanj	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;">  <p>Razlaga: koren označenega poddrevesa je stopnje tri (koren v vzorcu P pa je stopnje dve)</p> </div> <div style="text-align: center;">  <p>Razlaga: najbolj levi sin korena označenega poddrevesa ima oznako 'a' (v vzorcu P pa ima oznako 'b')</p> </div> </div>

Naloga 9

Pri tej nalogi ponovno iščemo pojavitev vzorca P v ciljnem drevesu T (veljajo enaka pravila kot v nalogi 8). Vhodna datoteka je enake oblike kot v nalogi 8 s to razliko, da ima sedaj vsaj eno vozlišče vzorca P oznako 'X' (velika črka angleške abecede). Vozlišče, označeno z 'X', se lahko preslika v poljubno **neprazno** hierarhijo vozlišč¹. Pri tem velja omejitev, da se za uspešno ujemanje vzorca P morajo **vs**a vozlišča, označena z 'X', preslikati v **enako** hierarhijo vozlišč.

Naloga je poiskati pojavitev vzorca P v drevesu T, ki pokrije maksimalno število vozlišč ciljnega dresa T. Z drugimi besedami, zanima nas tista pozicija vzorca P, pri kateri se vozlišča 'X' preslikajo v največje (in med seboj enake!) drevesne strukture.

V izhodno datoteko zapišite premi (preorder) obhod največje drevesne strukture, ki se ujema z vzorcem P. Izpis naj vsebuje oznake v vozliščih, ločene z vejicami.

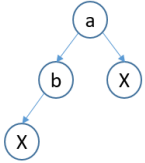
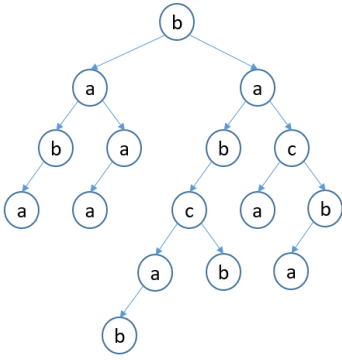
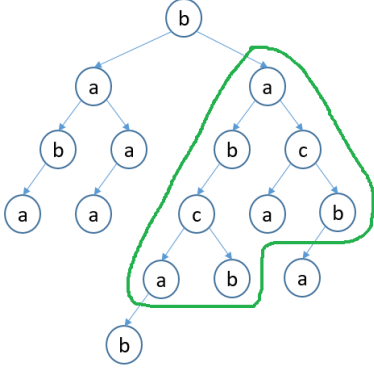
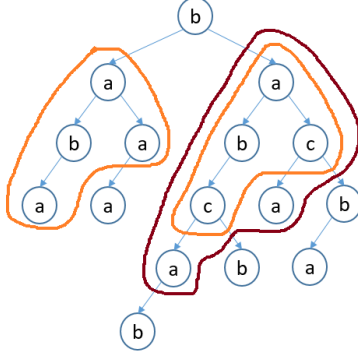
Implementirajte razred Naloga9, ki vsebuje metodo main. Metoda prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke in zapiše rezultat v izhodno datoteko.

Primer:

Vhodna datoteka:	Izhodna datoteka:
4 1,a,2,3 2,b,4 3,X 4,X 16 1,b,2,3 2,a,4,5 3,a,6,7 4,b,8 5,a,9 6,b,10 7,c,11,12 8,a 9,a 10,c,13,14 11,a 12,b,15 13,a,16 14,b, 15,a 16,b	a,b,c,a,b,c,a,b

¹ Pri razširjanju hierarhije moramo obstoječemu vozlišču-listu hierarhije dodati vse njegove sinove iz T.

Analiza primera:

Vzorec P	 <pre> graph TD a((a)) --> b((b)) a --> X1((X)) b --> X2((X)) </pre>
Ciljno drevo T	 <pre> graph TD b1((b)) --> a1((a)) b1 --> a2((a)) a1 --> b1_1((b)) a1 --> a1_1((a)) b1_1 --> a1_1_1((a)) a1_1 --> a1_1_1 a2 --> b2((b)) a2 --> c1((c)) b2 --> c2((c)) b2 --> a2_1((a)) c1 --> a2_1_1((a)) c1 --> b2_1((b)) a2_1 --> a2_1_1 </pre>
Maksimalno ujemanje	
<p>Svetlo rjava obroba označuje ujemanji, ki nista maksimalni.</p> <p>Temno rjava obroba ni pravilno ujemanje, ker mora vključevati vse sinove notranjih vozlišč (vključuje levega sina notranjega vozlišča 'c', ne pa tudi desnega).</p>	

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 najmanjšo 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 zaključí, 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 **najmanjšo** vrednostjo $v(b)$, poimenujmo ga (tisto vozlišče) b_min . V primeru, da ima več vozlišč enako oceno $v(b)$, se izbere tisto z **najmanjšo** 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 zaključí. Iteracija se prav tako zaključí 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 obliki 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).

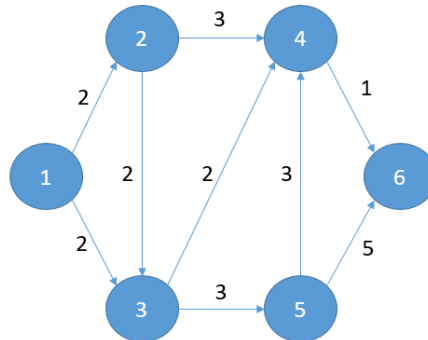
V izhodno datoteko za vsako iteracijo izpišite vozlišča, ki jih je obiskal agent (vključno z zadnjima dvema, identičnima, iteracijama).

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 4, 6, 1 1, 6	1, 2, 3, 4, 6 1, 2, 3, 4, 6 1, 3, 4, 6 1, 3, 4, 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 boljše 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 zapiše 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 ocenjeni 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 $v(5) = 3 + 0 = 3$. Ocenjeni 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 zapiše 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 agent 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 zapiše 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 zapišemo izbrano pot 1,3,4,6 in zaključimo.