Algoritmi in podatkovne strukture 1 2021/2022

Seminarska naloga 1

Rok za oddajo programske kode prek učilnice je **sobota**, **27. 11. 2021.**

Zagovori seminarske naloge bodo potekali v terminu vaj v tednu **29. 11. – 3. 12. 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 (razen javanskih zbirk 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 1

Plantažni delavec mora prenesti orodje iz severo-zahodne plantaže preko plantažnega polja do jugo-zahodne plantaže. Plantažno polje in plantaže so kvadratne oblike, tako da ima vsaka plantaža (razen robnih) štiri sosednje plantaže. Ker je orodje težko, se lahko delavec dvigne na plantažo višjo za največ **2m** in spusti na plantažo nižjo za največ **3m**. Pomagaj mu najti pot do končne plantaže.

Implementirajte razred **Naloga1**, ki vsebuje metodo **main**. Metoda v argumentih prejme poti do vhodne in izhodne datoteke (args[0] in args[1]), prebere vhodne podatke, poišče pot od začetne do končne plantaže in v izhodno datoteko zapiše višinsko razliko, za katero se delavec na poti dvigne.

Tekstovna vhodna datoteka je podana v naslednjem formatu:

- v prvi vrstici je zapisano celo število **N**, ki določa dimenzije plantažnega polja **NxN**.
- v naslednjih **N** vrsticah so v **decimetrih** zapisane relativne višine posameznih plantaž glede na začetno plantažo. Vsaka vrstica vsebuje **N** vrednosti, ločenih z vejicami.

V tekstovno izhodno datoteko zapišite celo število, ki je seštevek vseh dvigov na poti od začetne do končne plantaže v **decimetrih** (seštevek spustov nas ne zanima).

Opomba: vhodna naloga ima lahko več enakovrednih rešitev. V tem primeru v izhodno datoteko zapišite samo eno rešitev.

Primer:

Vhodna datoteka:	Izhodna datoteka:
3,3 0,-29,-20 14,70,-27 -10,3,14	38

Razlaga primera:

Edina dovoljena pot je dol, dol, desno, desno. Na poti so višinske razlike sledeče: 14, -24, 13, 11, tako da je vsota vseh vzponov 14+13+11=38.

Naloga 2

Rok se igra s tekstom. Napisal je program, ki obrne vrstni red črk v besedah, zamenja sode in lihe besede (prvo in drugo, tretjo in četrto,...) v povedi, obrne vrstni red povedi ter obrne vrstni red lihih odstavkov (medtem ko sodi odstavki ostanejo na mestu). Napiši program, ki bo pretvoril besedilo v prvotno stanje.

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, jih postavi v pravilni vrstni red, ter jih izpiše v izhodno datoteko.

Tekstovna vhodna datoteka vsebuje zakodirano besedilo. Tekstovna izhodna datoteka naj vsebuje dekodirano besedilo.

Opombe: Ločila se obravnavajo kot deli besed, katerih se držijo. Vhodni podatki bodo takšni, da bo rešitev enolična. Razmisli, kdaj ni moč enolično rešiti naloge.

Primer:

Vhodna datoteka:	Izhodna datoteka:
`ap oT agurd ej .devop ej oT intset tsorperp z remirp .mokvatsdo mine`	`To je preprost testni primer z enim odstavkom. To pa je druga poved.`

Naloga 3

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)`

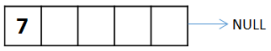
Metoda `void init(int N)` sprejme parameter `N` (kjer je `N>1`), ki določa največje število elementov v posameznem členu seznama. Po klicu metode je seznam prazen. Konstruktor strukture avtomatsko kliče metodo `init` z vrednostjo `N=5`.

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 ciljni člen (tisti, ki vsebuje element na poziciji `p-1`) ima 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 zavrže in metoda vrne `false`.

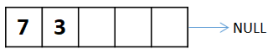
Funkcija za brisanje `boolean remove(int p)` prejme pozicijo elementa, ki ga želimo odstraniti iz seznama. Najprej poiščemo logično pozicijo, ki bo izbrisana (upoštevamo samo dejansko vstavljene elemente in ne indeksov polj). Element na tem mestu izbrišemo in po potrebi izvedeno zamik elementov v levo,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

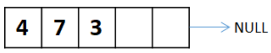
- po zaporedju klicev `init(5)` in `insert(7,0)` bo struktura vsebovala samo en člen:



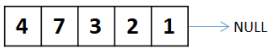
- po klicu `insert(3, 1)`:



- po klicu `insert(4,0)` se vsebina statičnega polja znotraj člena ustrezno zamakne:

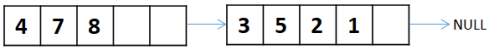


- po zaporedju klicev `insert(2,3)` in `insert(1,4)`:

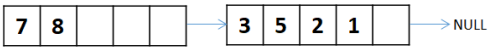


- po klicu `insert(5,3)` se člen razdeli na dva dela (4 in 7 gresta v en člen, preostali elementi pa v drugega), nato se element 5 vstavi na ustrezno mesto:

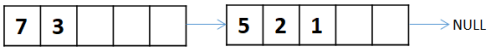
- po klicu `insert(8,2)`:



- po klicu `remove(0)` se elementi prvega člena zamaknejo:

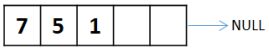


- po klicu `remove(1)` ima prvi člen premalo elementov in se en element prenese iz naslednjega člena:



- 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 **Naloga3**, 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 4

Napisali bomo simulator dogajanja v frizerskem salonu. V salonu se nahaja samo en frizerski stol, kar pomeni, da se v vsakem trenutku lahko striže samo ena stranka. Salon ima tudi čakalnico, ki deluje po sistemu FIFO (first-in-first-out).

V salon periodično vstopajo nove stranke, ki so opisane z id-jem in potrpljenjem (koliko so pripravljene sedeti v čakalnici preden nepostrižene odidejo iz salona). Če je ob vstopu stranke v salon čakalnica polna, stranka odide in je ni več nazaj. V nasprotnem primeru se usede kot zadnja v vrsti (če je čakalnica prazna in se trenutno nobeden ne striže, se usede kar na frizerski sol).

Ko se striženje ene stranke začne, se izvede do konca. Po končanem striženju stranka zapusti salon in se prične s striženjem naslednjega v vrsti (če le-ta obstaja). Vsako striženje dodatno utrudi frizerja, kar povzroči podaljševanje časa naslednjih striženj.

Vhodni parametri simulacije so:

- T – število korakov simulacije
- N – število stolov v čakalnici
- S – trajanje striženja (v korakih simulacije)
- K – za koliko se podaljša striženje vsake naslednje stranke (v korakih simulacije)
- L_A – seznam z zamiki prihodov strank (v korakih simulacije). Seznam L_A lahko vsebuje enega ali več elementov. Na primer, če velja L_A = [X, Y, Z], pomeni, da bodo stranke vstopale v salon v korakih X, X+Y, X+Y+Z, X+Y+Z+X, X+Y+Z+X+Y, X+Y+Z+X+Y+Z,…
- L_W – seznam s podatki o potrpljenju strank (v korakih simulacije). Seznam L_W lahko vsebuje enega ali več elementov. Na primer, če velja L_w = [A,B], pomeni, da bo prva stranka imela potrpljenje A, druga stranka B, tretja stranka A, četrta stranka B in tako naprej.

Simulacija se izvaja po korakih. Posamezen korak se izvede v tem zaporedju:

1. Če imamo stranko za frizerskim stolom in se je njeno striženje zaključilo (je preteklo S korakov od začetka), stranka zapusti salon in frizerski stol ostane prazen. Trajanje naslednjega striženja se podaljša za K korakov (torej, S = S + K).
2. Če je frizerski stol prazen in imamo stranko v čakalnici, se le-ta usede na frizerski stol in prične s striženjem.
3. Vse stranke v čakalnici, ki predolgo čakajo, zapustijo salon in se več ne vračajo.
4. Če je nastopil čas za prihod nove stranke, jo kreiramo v skladu s podatki v L_A in L_W. Prva stranka dobi id=1, vsaka naslednja stranka ima vrednost id povečano za 1. Kreirana stranka vstopi v salon. Če je frizerski stol prazen, se usede vanj in prične s striženjem. V nasprotnem primeru se stranka usede na stol v čakalnici kot zadnja v vrsti. Če je čakalnica polna (vseh N stolov je zasedenih), stranka odide in se več ne vrača.

Implementirajte razred **Naloga4**, ki vsebuje metodo **main**. Argumenti metode main vsebujejo poti do vhodne in izhodne datoteke (args[0] in args[1]).

Tekstovna vhodna datoteka v prvih štirih vrsticah vsebuje celoštevilčne vrednosti parametrov T, N, S in K. V peti vrstici so zapisani celoštevilčni elementi seznama L_A, ločeni z vejico. Šesta vrstica vsebuje celoštevilčne elemente seznama L_W, ki so prav tako ločeni z vejicami.

V tekstovno izhodno datoteko zapišite id-je postriženih strank (v istem vrstnem redu kot so strižene). Posamezne vrednosti ločite z vejicami. Upoštevajte samo stranke, ki so postrižene do konca (ignorirajte stranko na frizerskem stolu ob izteku simulacije).

Primer:

Vhodna datoteka:	Izhodna datoteka:
20 2 3 1 2,3,1 4,5	1,2,3,5

Razlaga primera:

Korak	Dogodki
1	
2	Prihod stranke 1 (potrpljenje 4). Začetek striženja stranke 1 (trajalo bo 3 korake).
3	
4	
5	Striženje stranke 1 se je zaključilo. Prihod stranke 2 (potrpljenje 5). Začetek striženja stranke 2 (trajalo bo 4 korake).
6	Prihod stranke 3 (potrpljenje 4). Stranka 3 se je usedla na stol v čakalnici.
7	
8	Prihod stranke 4 (potrpljenje 5). Stranka 4 se je usedla na stol v čakalnici.
9	Striženje stranke 2 se je zaključilo. Začetek striženja stranke 3 (trajalo bo 5 korakov).
10	
11	Prihod stranke 5 (potrpljenje 4). Stranka 5 se je usedla na stol v čakalnici.
12	Prihod stranke 6 (potrpljenje 5). Čakalnica je polna in je stranka 6 odšla.
13	Stranka 4 se je naveličala in zapustila čakalnico.
14	Striženje stranke 3 se je zaključilo. Začetek striženja stranke 5 (trajalo bo 6 korakov). Prihod stranke 7 (potrpljenje 4). Stranka 7 se je usedla na stol v čakalnici.
15	
16	
17	Prihod stranke 8 (potrpljenje 5). Stranka 8 se je usedla na stol v čakalnici.
18	Stranka 7 se je naveličala in zapustila čakalnico. Prihod stranke 9 (potrpljenje 4). Stranka 9 se je usedla na stol v čakalnici.
19	
20	Striženje stranke 5 se je zaključilo. Začetek striženja stranke 8 (trajalo bo 7 korakov). Prihod stranke 10 (potrpljenje 5). Stranka 10 se je usedla na stol v čakalnici.

Naloga 5

Dva igralca igrata igro z naslednjimi pravili. Na mizo položita M kupčkov s po n_1, n_2, …, n_M kamenčki in izmenoma vlečeta poteze. Igralec na potezi izbere en kupček in iz njega odstrani poljubno število kamenčkov (vsaj enega). Zmaga igralec, ki lahko zadnji izvede potezo.

Oba igralca želita zmagati, zato če je le možno vlečeta poteze, ki nasprotniku ne puščajo nobenih možnosti. Igralca, ki začenja igro, zanima, koliko potez je potrebnih, da bo zagotovo premagal svojega nasprotnika. Pri rezultatu štejemo poteze **obeh** igralcev.

Implementirajte razred **Naloga5**, ki vsebuje metodo **main**. Argumenti metode main vsebujejo poti do vhodne in izhodne datoteke (args[0] in args[1]). Metoda naj iz vhodne datoteke prebere začetno konfiguracijo kamenčkov na mizi, nato naj zapiše rezultat v izhodno datoteko.

Tekstovna vhodna datoteka v prvi vrstici vsebuje število kupčkov K. V naslednjih K vrsticah je zapisano število kamenčkov v posameznem kupčku. V izhodni tekstovni datoteki naj bo zapisan rezultat (število potez do neizogibnega poraza drugega igralca) oziroma vrednost -1, če bo prvi igralec zagotovo izgubil igro ob optimalnih potezah nasprotnika.

Namig: uporabite rekurzijo za pregledovanje vseh možnih potekov igre. Pri sistematičnem pregledovanju boste velikokrat generirali identične situacije, ki jih je nesmiselno vsakič znova pregledovati. Najdite način za shranjevanje rezultatov pregledovanja, ki jih boste pozneje uporabili ob ponovnem srečanju z istimi situacijami.

Pri štetju potez najdaljše igre je potrebno upoštevati, da prvi igralec želi čim prej zmagati. Drugi igralec prav tako želi zmagati, če pa zmaga ni možna, se bo trudil čim kasneje izgubiti.

Primer:

Vhodna datoteka:	Izhodna datoteka:
3 2 2 2	5

Razlaga primera: če prvi igralec začne tako, da pobere samo en kamenček iz nekega kupčka, bo zagotovo izgubil igro (ob predpostavki, da bo drugi igralec vlekel optimalne poteze). Zato bo prvi igralec najprej iz enega kupčka vzel 2 kamna (prva poteza). Na mizi ostaneta dva kupčka s po dvema kamenčkoma. Če sedaj drugi igralec pobere dva kamenčka iz enega izmed kupčkov, bo v naslednji potezi poražen. Da bi se čim dlje upiral porazu, bo iz enega kupčka vzel samo en kamenček (druga poteza). Prvi igralec bo nato iz drugega kupčka vzel en kamenček (tretja poteza). Sedaj sta na mizi dva kupčka s po enim kamenčkom. Drugi igralec pobere kamenček iz enega kupčka (četrta poteza), nato prvi igralec pobere zadnji kamen z mize in zmaga (peta poteza). To je najdaljše zaporedje potez, ki ga drugi igralec lahko izsili, preden izgubi igro.