Izredni izpitni rok pri predmetu Programiranje 1
28. marec 2022

Oddajte datoteke `Prva.java`, `Druga.java`, `Tretja.java` in `Cetrta.java`. Testirate jih lahko takole:

(1) `tj.exe Prva.java . .` (2) `tj.exe Druga.java . .` (3) `tj.exe` (4) `tj.exe`

(1) Na vhodu je najprej zapisano celo število $n \in [4, 10^5]$, nato pa sledi n vrstic, od katerih vsaka vsebuje celoštevilski koordinati x in y z intervala $[-10^9, 10^9]$. Teh n vrstic tvori zaporedje opisov neznanega števila večkotnikov. Vsak večkotnik je opisan z zaporedjem (x_1, y_1), (x_2, y_2), $\ldots$, (x_k, y_k), (x_1, y_1), kjer je k število njegovih oglišč, (x_i, y_i) pa koordinati i-tega oglišča. V primeru na desni (`vhod01.txt`) imamo dva večkotnika: trikotniku z oglišči $(-1, -3)$, $(6, 2)$ in $(4, -5)$ sledi štirikotnik z oglišči $(5, -8)$, $(-7, 0)$, $(0, 4)$ in $(3, -2)$.

Napišite program (`Prva.java`), ki za podani vhod izpiše število večkotnikov. Vsak večkotnik na vhodu ima vsaj tri oglišča, poleg tega pa noben par njegovih oglišč ne sovpada.

Vhod:

```
9
-1 -3
6 2
4 -5
-1 -3
5 -8
-7 0
0 4
3 -2
5 -8
```

Izhod:

```
2
```

(2) Pika (»piksel«) dvojiške slike je *mejna*, če ima vrednost 1, poleg tega pa se nahaja na robu slike ali pa ima vsaj ena od njenih osmih sosed vrednost 0. V primeru na desni (`vhod07.txt`) so označene vse mejne pike.

Napišite program (`Druga.java`), ki za podano dvojiško sliko izpiše število mejnih pik. Slika je na vhodu zapisana tako: v prvi vrstici sta podani celi števili $h \in [1, 500]$ in $w \in [1, 500]$, nato pa sledi h vrstic s po w števili iz množice $\{0, 1\}$.

V 20% skritih testnih primerov velja $h \leqslant 2$ ali $w \leqslant 2$. V nadaljnjih 40% primerov velja $h \geqslant 3$ in $w \geqslant 3$, vendar pa imajo vse pike na robu slike vrednost 0.

Vhod:

```
7 10
0 0 0 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0
0 0 0 1 1 1 1 1 1 1
0 0 0 1 1 1 0 1 1 0
1 0 0 0 1 1 1 1 0 0
```

Izhod:

```
30
```

(3) Oddajniki (objekti tipa `Oddajnik`) pošiljajo znake svojim sprejemnikom (objektom tipa `Sprejemnik`). Vsak sprejemnik prejema znake od natanko enega oddajnika, oddajnik pa jih lahko pošilja več sprejemnikom. Te povezave se določijo ob izdelavi sprejemnikov in se nato več ne spreminjajo.

Ob klicu *oddajnik*`.oddaj(`*znak*`)` naj oddajnik podani znak pošlje vsem svojim sprejemnikom. Klic *sprejemnik*`.koliko()` naj vrne število vseh znakov, ki jih je sprejemnik doslej prejel. Klic *sprejemnik*`.odZadnjic()` naj vrne niz, ki po vrsti vsebuje vse znake, ki jih je sprejemnik prejel od prejšnjega klica te metode oziroma od začetka, če gre za prvi tak klic.

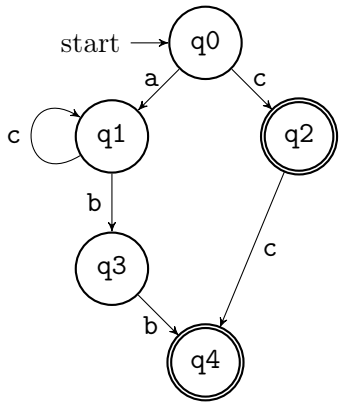
V datoteki `Tretja.java` dopolnite razreda `Oddajnik` in `Sprejemnik`. Pomagajte si s sledečim primerom (`Test07.java`):

```
Oddajnik krvavec = new Oddajnik();
Oddajnik nanos = new Oddajnik();
krvavec.oddaj('a');                          // tega nihče ne sprejme
Sprejemnik ana = new Sprejemnik(krvavec);    // ana posluša krvavec
krvavec.oddaj('b');                          // to sprejme ana
```

```
Sprejemnik bojan = new Sprejemnik(krvavec); // tudi bojan posluša krvavec
Sprejemnik cvetka = new Sprejemnik(nanos);  // cvetka ima raje nanos
nanos.oddaj('c');                           // to sprejme cvetka
krvavec.oddaj('d');                         // to sprejmeta ana in bojan
System.out.println(ana.odZadnjic());        // bd
System.out.println(cvetka.odZadnjic());     // c
krvavec.oddaj('e');                         // to sprejmeta ana in bojan
System.out.println(ana.odZadnjic());        // e
System.out.println(bojan.odZadnjic());      // de
System.out.println(bojan.odZadnjic());      // (prazno)
System.out.println(ana.koliko());           // 3
System.out.println(bojan.koliko());         // 2
```

Lahko predpostavite, da vsakemu oddajniku pripada kvečjemu po 1000 sprejemnikov in da vsak sprejemnik v času svojega obstoja prejme kvečjemu po 1000 znakov. V 60% skritih testnih primerov je vsak oddajnik povezan s kvečjemu enim sprejemnikom, v polovici od teh primerov pa ni nobenega klica metode `odZadnjic`.

(4) *Končni avtomat* je stroj, ki po vrsti bere znake vhodnega niza (*besede*) in pri tem spreminja svoje stanje. Na primer, avtomat, ki ga ponazarja diagram na desni (`Test01.java`, `Test04.java`), je na začetku v stanju `q0` (*začetno stanje*), po branju besede `acb` pa pristane v stanju `q3` ($\texttt{q0} \xrightarrow{\texttt{a}} \texttt{q1} \xrightarrow{\texttt{c}} \texttt{q1} \xrightarrow{\texttt{b}} \texttt{q3}$). Stanji `q2` in `q4` sta *sprejemni stanji*; če avtomat po branju besede pristane v enem od njih, pravimo, da besedo *sprejme*. Naš avtomat potemtakem sprejme besede `c`, `cc`, `abb`, `acbb`, `accbb`, `acccbb` itd.



Končni avtomat je predstavljen kot objekt razreda `Avtomat` v datoteki `Cetrta.java`, ki vsebuje sledeče atribute:

```
private String zacetnoStanje;
private Set<String> sprejemnaStanja;
private Map<String, Map<Character, String>> prehodi;
```

Slovar `prehodi` pove, v katero stanje preide avtomat pri podanem izhodiščnem stanju in podanem znaku. V našem primeru bi klic `prehodi.get("q1").get('b')` vrnil niz `q3`, klic `prehodi.get("q2").get('a')` pa vrednost `null`, saj ta prehod ne obstaja. Klic `prehodi.get("q4")` bi vrnil prazen slovar.

Razred `Avtomat` dopolnite s sledečimi metodami:

- [32%] `public boolean jeSprejemno(String stanje)`

  Vrne `true` natanko v primeru, če je podano stanje sprejemno.

- [34%] `public String kam(String beseda)`

  Vrne stanje, v katerem se avtomat nahaja po branju podane besede. Če kateri od prehodov ne obstaja, naj metoda vrne `null`.

- [34%] `public static Avtomat zaBesedo(String beseda)`

  Vrne končni avtomat, ki sprejme samo podano besedo. Namig: i-to stanje naj bo kar `Integer.toString(`i`)`.

  Javni testni razredi preverjajo to metodo z vašima, skriti pa z našima (zanesljivo pravilnima) implementacijama metod `jeSprejemno` in `kam`.