Prvi izpitni rok pri predmetu Programiranje 1
21. januar 2021

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) Napišite program, ki prebere celo število $n \in [1, 1000]$ in n celih števil z intervala $[1, 10^9]$ in za vsako od teh števil izpiše `DA`, če njegove števke tvorijo (ne nujno strogo) padajoče zaporedje, in `NE`, če to ne drži. Vsako vhodno število je zapisano v svoji vrstici, enako pa naj velja tudi za odgovore, ki jih izpiše vaš program.

Vaš program lahko uporablja le tipa `int` in `boolean`. V nasprotnem primeru bomo število dobljenih točk pri tej nalogi prepolovili!

Primer (vhod/izhod):

```
6
333
7210
2315432
1000000000
865312
9
```

```
DA
DA
NE
DA
NE
DA
```

(2) Vrtnar je nekoč znotraj pravokotnika velikosti $h \times w$ (višina krat širina) posadil nekaj dreves, do danes pa se je ta pravokotnik v obeh dimenzijah neskončnokrat razmnožil. Sprehajalec se postavi v zgornjo levo celico izhodiščnega pravokotnika (ta je v vseh testnih primerih prazna), nato pa se vsako uro premakne za a enot navzdol in b enot v desno ter se na kratko ustavi. Koliko ur bo hodil, da se bo ustavil pri k drevesih? Po 10^4 urah odneha, tudi če obišče manj kot k dreves.

Prva vrstica vhoda vsebuje cela števila $h \in [1, 100]$, $w \in [1, 100]$, $a \in [0, 10^4]$, $b \in [0, 10^4]$ in $k \in [1, 10^4]$, nato pa sledi h vrstic s po w števili 1 (drevo) in 0 (prazen prostor), ki podajajo izhodiščni pravokotnik. Vsa števila v isti vrstici so med seboj ločena s po enim presledkom. Program naj izpiše iskano število ur hoda.

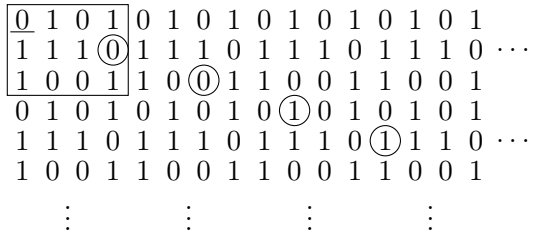
V 50% skritih testnih primerov velja, da sprehajalec ne bo zapustil izhodiščnega pravokotnika.

Primer (vhod/izhod):

```
3 4 1 3 2
0 1 0 1
1 1 1 0
1 0 0 1
```

```
4
```

Za lažje razumevanje primera si oglejte spodnjo sliko (obiskana polja so obkrožena):



(3) Podani so sledeči razredi:

```
class Opravilo {    // npr. rezanje lesa, pritrjevanje nog, lakiranje ipd.
    private String naziv;
    private int zahtevnost;  // večje število pomeni zahtevnejše opravilo
}
class Projekt {     // npr. izdelava mize
    private String naziv;
    private Opravilo[] opravila;    // dolžina ⩾ 1
}
class Delavec {
    private String ip;           // ime in priimek
    private int usposobljenost; // kako zahtevna opravila/projekte lahko izvaja
}
class Delavnica {
    private Delavec[] delavci;
}
```

Zahtevnost projekta je enaka zahtevnosti njegovega najzahtevnejšega opravila. Delavec lahko izvede projekt le tedaj, ko je njegova usposobljenost enaka zahtevnosti projekta ali večja.

Napišite sledeče metode:

- [34%] `public int zahtevnost()` v razredu `Projekt`:

  Vrne zahtevnost projekta `this`.

- [32%] `public int univerzalci(Projekt[] projekti)` v razredu `Delavnica`:

  Vrne število delavcev v delavnici `this`, ki bi lahko lastnoročno izvedli vse projekte v podani tabeli.

- [34%] `public boolean jePermutacijaOd(Object drugi)` v razredu `Projekt`:

  Vrne `true` natanko v primeru, če objekt `drugi` predstavlja projekt, ki vsebuje *ista* opravila (iste objekte!) kot projekt `this`, a ne nujno v istem vrstnem redu. Lahko predpostavite, da isti projekt ne vsebuje več kot enega kazalca na isto opravilo.

(4) Iz abstraktnega razreda `Lik` sta izpeljana razreda `Pravokotnik` in `Krog`, iz razreda `Pravokotnik` pa razred `Kvadrat`. Vsak lik ima svojo barvo, določeno s komponentami R (rdeča), G (zelena) in B (modra). Naravna urejenost likov je določena s ploščino: lik A sodi pred lik B natanko tedaj, ko ima lik A manjšo ploščino kot lik B. Da se izognete delu s tipom `double`, ploščino kroga zaokrožite z izrazom `(int) Math.round(...)`.

- [50%] Napišite vse potrebno, da bo metoda

  `public static void urediNaravno(List<Lik> liki)`

  podani seznam likov naravno uredila.

- [50%] Napišite vse potrebno, da bo metoda

  `public static Collection<Lik> poTipuInBarvi(Collection<Lik> liki)`

  vrnila zbirko, v kateri so liki iz zbirke `liki` urejeni po tipu (najprej pravokotniki, ki niso kvadrati, nato kvadrati in nazadnje krogi), v primeru enakih tipov pa po barvi. Barve uredimo po naraščajočih komponentah R, v primeru enakih komponent R po naraščajočih komponentah G, v primeru enakih komponent R in G pa po naraščajočih komponentah B. **Pozor:** metoda ne sme spreminjati zbirke `liki`!

V testnih primerih se vsi liki po urejevalnem kriteriju med seboj razlikujejo. Na primer, če like naravno urejamo, potem noben par likov nima iste ploščine.