Rešitev 6. domače naloge (Štoparski vodnik po Manhattanu)

Rešitev »na prvo žogo«

Vhodno matriko preberemo v celoštevilsko tabelo `zemljevid` velikosti $m \times n$:

```
import java.util.Scanner;

public class StoparskiVodnik {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int stVrstic = sc.nextInt();
        int stStolpcev = sc.nextInt();
        int stZnamenitosti = sc.nextInt();

        int[][] zemljevid = new int[stVrstic][stStolpcev];
        for (int i = 0;  i < stVrstic;  i++) {
            for (int j = 0;  j < stStolpcev;  j++) {
                zemljevid[i][j] = sc.nextInt();
            }
        }
        ...
    }
}
```

Sedaj po vrsti poiščemo prvo, drugo, ... in k-to znamenitost. Pomagamo si z metodo `poisciZnamenitost`, ki poišče znamenitost s podano zaporedno številko in vrne tabelo z dvema elementoma: prvi element vsebuje indeks vrstice, drugi pa indeks stolpca iskane znamenitosti. Vzdržujemo še spremenljivko `pot`. Na začetku jo nastavimo na 0, potem pa ji vsakokrat prištejemo manhattansko razdaljo od prejšnje do trenutne Arthurjeve postaje, pri čemer upoštevamo, da naš štopar svoj obhod prične na točki (0, 0). Spremenljivka `pot` potemtakem vsebuje skupno manhattansko pot od izhodišča do trenutne znamenitosti. Ko se zanka zaključi, spremenljivka podaja celotno manhattansko pot.

```
public class StoparskiVodnik {

    public static void main(String[] args) {
        ...
        int vr = 0;
        int st = 0;
        int pot = 0;

        for (int i = 1;  i <= stZnamenitosti;  i++) {
            int[] polozaj = poisciZnamenitost(zemljevid, i);
            // (vr, st) je prejšnji Arthurjev položaj
            // (polozaj[0], polozaj[1]) je trenutni Arthurjev položaj

            pot += Math.abs(polozaj[0] - vr) + Math.abs(polozaj[1] - st);
            vr = polozaj[0];
            st = polozaj[1];
        }
        System.out.println(pot);
    }
    ...
```

```
}
```

Metoda `poisciZnamenitost` se sprehaja po celicah zemljevida, dokler ne naleti na iskano zaporedno številko. Ko se to zgodi, zapiše koordinati v tabelo in tabelo vrne kot svoj rezultat. Kot vemo, se mora vsaka metoda, ki nekaj vrne, zaključiti s stavkom `return` ali `throw`. Če smo program pravilno napisali in če testni primeri ustrezajo zahtevam, se izjema nikoli ne bo sprožila.

```
public class StoparskiVodnik {
    ...
    private static int[] poisciZnamenitost(int[][] zemljevid, int stevilka) {
        for (int i = 0;  i < zemljevid.length;  i++) {
            for (int j = 0;  j < zemljevid[i].length;  j++) {
                if (zemljevid[i][j] == stevilka) {
                    int[] polozaj = {i, j};
                    return polozaj;
                }
            }
        }
        throw new RuntimeException("Napaka!");
    }
}
```

Učinkovitejša rešitev

Pravkar prikazana rešitev se k-krat sprehodi po tabeli. Vsak sprehod v povprečju terja $mn/2$ korakov, od koder sledi, da časovna zahtevnost programa znaša $O(mnk)$. (To pomeni, da je poraba časa navzgor omejena s funkcijo $c \cdot mnk + d$, kjer sta c in d konstanti, neodvisni od vhodnih vrednosti m, n in k.) Če bolje pomislimo, pa lahko pridemo do rešitve s časovno zahtevnostjo zgolj $O(mn)$.

Pripravimo si tabelo `polozaji` velikosti $k \times 2$. Element `polozaji[s][0]` naj vsebuje indeks vrstice, element `polozaji[s][1]` pa indeks stolpca znamenitosti z indeksom s oziroma zaporedno številko $s + 1$. Tabelo napolnimo med branjem matrike: ko v i-ti vrstici in j-tem stolpcu matrike preberemo znamenitost z zaporedno številko z, shranimo vrednost i v element `polozaji[`z`-1][0]`, vrednost j pa v element `polozaji[`z`-1][1]`.

```
import java.util.Scanner;

public class StoparskiVodnik {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int stVrstic = sc.nextInt();
        int stStolpcev = sc.nextInt();
        int stZnamenitosti = sc.nextInt();

        int[][] polozaji = new int[stZnamenitosti][2];

        for (int i = 0;  i < stVrstic;  i++) {
            for (int j = 0;  j < stStolpcev;  j++) {
                int znamenitost = sc.nextInt();
                if (znamenitost > 0) {
                    polozaji[znamenitost - 1][0] = i;
```

```
                    polozaji[znamenitost - 1][1] = j;
                }
            }
        }
        ...
    }
}
```

Manhattansko pot lahko sedaj izračunamo s preprostim sprehodom po vrsticah tabele `polozaji`:

```
import java.util.Scanner;

public class StoparskiVodnik {

    public static void main(String[] args) {
        ...
        int vr = 0;
        int st = 0;
        int pot = 0;
        for (int i = 0;  i < stZnamenitosti;  i++) {
            pot += Math.abs(polozaji[i][0] - vr) + Math.abs(polozaji[i][1] - st);
            vr = polozaji[i][0];
            st = polozaji[i][1];
        }
        System.out.println(pot);
    }
}
```

Časovna zahtevnost polnjenja tabele `polozaji` znaša $O(mn)$, časovna zahtevnost računanja manhattanske poti pa $O(k)$. Ker velja $k \leqslant mn$, je časovna zahtevnost celotnega izboljšanega programa enaka $O(mn)$.