Aufgabe 2: Eisbudendilemma

Teilnahme-Id: 00041

Bearbeiter/-in dieser Aufgabe:
Sebastian Brunnert

23. Dezember 2020

Inhaltsverzeichnis

Lösungsidee

Letztlich kann gesagt werden, dass die möglichen Eisbuden so gefunden werden, dass jede mögliche Position für eine Eisbude reihum betrachtet wird und mit der aktuell besten Position verglichen wird. Ist eine bessere Position gefunden, so wird die aktuell best-mögliche Position neu definiert. Da aber für die Eisbuden mehrere Positionen gefunden werden sollen, wird also dieses Verfahren für jeden möglichen Startwert ausgeführt. Positionen, wo bereits eine Eisbude steht werden natürlich übersprungen. Es kann also gesagt werden, dass für jede Position eine Abstimmung simuliert wird und es geht immer eine andere Position voran, die zuvor definiert wurde.

Aber wie kann gesagt werden, dass eine Position sich besser eignet? Dazu muss letztlich für jedes Haus überprüft werden, ob dieses sein „Veto“ einlegen würde. Dies wird rechnerisch über die Distanz bestimmt. Da ja das Veto davon abhängt, wo die Eisbude vorher platziert war, ist es auch von Relevanz wo die anfänglichen Eisbuden sind. Daher wird der Algorithmus für jeden möglichen Startwert durchgeführt. Anfangs wird also von 0-n geprüft, wo sich möglicherweise eine Eisbude befindet, dann von 1-n-0, dann von 2-n-1. Bereits mögliche Eisbuden werden natürlich außer Acht gelassen.

Umsetzung

Die Umsetzung dieser Aufgabe geschieht in der Prorammiersprache Java auf Maven. Nachdem die Datei eingelesen und ausgewertet wurde, wird ein Objekt der Klasse Dorf erstellt. Diese beinhaltet die aus der Datei hervorgehenden Daten wie den Umfang oder die Platzierungen der Häuser. Außerdem existiert ein Integer eisbudenPosition (dieser wird im späteren Verlauf durch Kandidaten für eine Eisbude ersetzt). Mathematisch bestimmt die Methode distanz(int pos1, int pos2) die Distanz zwischen zwei Punkten auf dem Kreis. Wenn z.B. beim Umfang 200 die Distanz 101 beträgt, wird die Distanz zu 200-101=99 gesetzt, da „gegen die Richtung“ schneller zum Ziel gelangt werden kann. Außerdem wird die relevante Methode eisbudePruefen(int eisbudenPosition) eingeführt. Diese ermittelt, ob eine Eisbude errichtet werden würde im Vergleich zu der aktuellen Position der Eisbude. Dazu wird mittels der distanz-Funktion und einem for-Loop über die Haushalte ermittelt, wieviele Haushalte ihr Veto einreichen würden.

Nachdem nun das Objekt erstellt wurde, kann der eigentliche Algorithmus starten. Es wird die Liste results angelegt, in welche später Eisbuden kommen, die defintiv festlegen. Nun wird int startWert von 0 bis Umfang des Dorfs iteriert. Da für andere anfängliche Eisbuden bei diesem Algorithmus andere schlussendliche Eisbuden resultieren, wird dieser iteriert. Es sollen ja auch verschiedene Eisbuden ermittelt werden. Also wird bei dem Iterations-Durchlauf die anfängliche Eisbuden-Position gesetzt. In der Schleife wird nun auch durch jede mögliche Position für eine Eisbude iteriert, außer sie ist aktuell schon als best-mögliche Eisbude gesetzt (aufgrund des Startwertes eben). Durch die zuvor implementierte Methode eisbudePruefen(int eisbudenPositon) wird nun ermittelt, ob an dieser Stelle nun eine Eisbude gesetzt werden dürfte. Wenn ja, wird das auch gemacht. Also verändert sich die aktuelle Eisbuden-Position im Laufe des Schleifendurchlaufs in der Regeln mehrmals.

Wenn nun schlussendlich ein Resultat ermittelt wurde, wird es in die Liste results hinzugefügt, insofern es noch nicht vorhanden ist. Der Algorithmus wird nun für den nächsten Startwert durchgeführt und potenziell neue Eisbuden werden ermittelt. Zuletzt werden noch alle Resultate ausgegeben. Es handelt sich zwar definitiv nicht um die für jeden besten Ergebnisse. Nichtsdestotrotz würden alle Alternativ-Vorschläge abgelehnt werden bei folgenden Eisbuden:

Beispiele

eisbuden1.txt	Hier kann eine Eisbude stehen: 15
eisbuden2.txt	Hier kann eine Eisbude stehen: 45
eisbuden3.txt	Hier kann eine Eisbude stehen: 2 Hier kann eine Eisbude stehen: 3 Hier kann eine Eisbude stehen: 4 Hier kann eine Eisbude stehen: 5 Hier kann eine Eisbude stehen: 6 Hier kann eine Eisbude stehen: 7
eisbuden4.txt	Hier kann eine Eisbude stehen: 34
eisbuden5.txt	Hier kann eine Eisbude stehen: 93 Hier kann eine Eisbude stehen: 94 Hier kann eine Eisbude stehen: 95 Hier kann eine Eisbude stehen: 96 Hier kann eine Eisbude stehen: 97
eisbuden6.txt	Hier kann eine Eisbude stehen: 395
eisbuden7.txt	Hier kann eine Eisbude stehen: 285 Hier kann eine Eisbude stehen: 286 Hier kann eine Eisbude stehen: 287 Hier kann eine Eisbude stehen: 288 Hier kann eine Eisbude stehen: 289

Quellcode

Dorf.java

```
// Daten, die aus der Datei hervor gehen
private int umfang;
private int[] haueser;

// Für den Algorithmus wird die aktuelle Eisbude gesichert
private int eisbudenPosition;

public Dorf(int umfang, int[] haueser) {
    this.umfang = umfang;
    this.haueser = haueser;
    this.eisbudenPosition = 0;
}

public int getUmfang() {
    return umfang;
}

public int[] getHaueser() {
    return haueser;
}

// Verfahren zum Ermitteln der Distanz zweier Positionen
private int distanz(int pos1, int pos2) {
    int a = Math.abs(pos2-pos1);
    // Wenn z.B. beim Umfang 200 die Distanz 101 beträgt wird die Distanz zu 99 gesetzt, da "in die andere Richtung" es schneller geht
    if(a > this.umfang/2) {
        return this.umfang - a;
    } else {
        return a;
    }
```

```
    }

    // Algorithmus zum schauen, ob eine Eisbude (im Vergleich zur vorherigen) akzeptiert wird
    public boolean eisbudePruefen(int eisbudenPosition) {
        int vetos = 0;
        // Jedes Haus wird iteriert
        for(int haus : getHaueser()) {
            // Und anhand der Regeln wird ermittelt, ob dieses sein "Veto" einlegt
            if(distanz(haus,eisbudenPosition) >= distanz(haus,this.eisbudenPosition)) {
                vetos++;
            }
        }
        // Wenn alle "Vetos" ermittelt wurden wird überprüft, ob die Eisbude gesetzt werden kann
        return vetos < this.haueser.length-vetos;
    }

    public void setEisbudenPosition(int eisbudenPosition) {
        this.eisbudenPosition = eisbudenPosition;
    }

    public int getEisbudenPosition() {
        return eisbudenPosition;
    }
```

Eisbudendilemma.java

```
        try {
            BufferedReader bufferedReader = new BufferedReader(new FileReader(file));

            // Dorf wird anhand der Daten in Datei initialisiert
            dorf = new Dorf(Integer.parseInt(String.valueOf(bufferedReader.readLine().split(" ")[0])),
Arrays.stream(bufferedReader.readLine().split(" ")).mapToInt(Integer::parseInt).toArray());

        } catch (Exception e) {
            System.out.println("Die angegebene Datei kann nicht eingelesen werden.");
```

```
        System.exit(0);
    }

    // In diese Liste werden später alle möglichen Eisbuden abgelegt
    List<Integer> results = new ArrayList<>();

    // Da für andere anfängliche Eisbuden bei diesem Algorithmus andere schlussendliche Eisbuden resultieren, werden diese iteriert
    // Das heißt, dass dieser Algorithmus letztlich für jeden Startwert durchgeführt wird
    for(int startWert = 0; startWert < dorf.getUmfang(); startWert++) {
        // Also wird die anfängliche Eisbude auch zu dieser gesetzt
        dorf.setEisbudenPosition(startWert);

        // Nun wird durch jede Position iteriert ...
        for(int i = 0; i < dorf.getUmfang(); i++) {
            // ... außer sie ist aktuell gesetzt ...
            if(i != dorf.getEisbudenPosition()) {
                // ... wenn sie gewählt werden würde von den Dorfbewohnern ...
                if(dorf.eisbudePruefen(i)) {
                    // ... wird sie auch gesetzt. Der Wert kann bzw. wird sich später noch verändern
                    dorf.setEisbudenPosition(i);
                }
            }
        }
        // Dieses Verfahren findet also letztlich die letzte mögliche Eisbude. Es ist von Relevanz alle vorherigen auch durchzugehen,
        // da diese ja verglichen wird

        // Wenn diese Eisburde noch nicht gefunden wurde, wird diese gespeichert
        if(!results.contains(dorf.getEisbudenPosition())) {
            results.add(dorf.getEisbudenPosition());
        }
    }
```

```
for(int result : results) {
    System.out.println("Hier kann eine Eisbude stehen: " + result);
}
```