Aufgabe 3: Eisbudendilemma

Teilnahme-ID: 56860

Bearbeiter/-in dieser Aufgabe:
Christopher Besch

30. Dezember 2020

Inhaltsverzeichnis

1 Lösungsidee

Es werden alle Positionen gesucht, die gegen alle anderen Position in der Abstimmung nicht verlieren würden. Daher werden alle Positionen durchgegangen und überprüft. Bei jeder Überprüfung werden alle möglichen Positionen durchgegangen diese werden andere Position genannt und überprüft ob die zu testende Position in einer Abstimmung gegen diese andere Position verlieren und abgelöst werden würde. Ist dies der Fall, ist die zu testende Position keine stabile Position. Für die Überprüfung der Abstimmung wird der Abstand von der jeweiligen Position mit allen Häusern berechnet und verglichen. Wenn der Abstand durch die andere Position im Vergleich zum Abstand zur zu testenden Position verkürzt werden würde, stimmt das Haus gegen die zu testende Position. Wenn nun mehr Häuser gegen die zu testende Postion stimmen als für, verliert sie die Abstimmung. Dadurch hat die zu testende Position zwei Vorteile:

1. Wenn beide Positionen gleich weit von einem Haus entfernt sind, stimmt das betroffene Haus für die zu testende Position, da sie die aktuelle Position ist und nur abgelöst werden kann.
2. Wenn die Anzahl an Stimmen gegen die zu testende Position der Anzahl für diese entspricht, gewinnt die zu testende Position die Abstimmung.

Auf diese Weise ergibt sich die Menge aller stabile Positionen.

2 Umsetzung

Die Lösungsidee wird in C++ implementiert.

Einlese der Eingabedatei Als erster Schritt wird in der Funktion **read_file** die Eingabedatei gelesen. Hierbei wird überprüft, ob die Eingabedatei dem gegebenen Format entspricht, wenn nicht wird das Programm abgebrochen. Hierzu wird ein Makro **raise_error()** verwendet, dass die Ausführung des Programmes abbricht und eine möglichst informative Fehlermeldung zurückgibt. Dieses Makro wird ebenfalls für alle Methoden aller Klassen verwendet, um z.b. Segmentation Faults zu verhindern.

Es wird die Menge der Adressen aller Häuser in einem std::vector<int> und der Umfang des Sees in einem int gespeichert.

Bestimmung der stabilen Positionen Die Funktion **is_stable** geht alle möglichen Positionen durch und überprüft sie mit der Funktion **is_stable**. Alle gefundenen stabilen Positionen werden in einem std::vector gespeichert und zurückgegeben.

Die Funktion **is_stable** nimmt die Menge aller Häuser entgegen und eine zu testende Position. Es werden alle möglichen Positionen durchgegangen und überprüft, ob diese andere Position in einer Abstimmung gegen die zu testende Position gewinnen würde. Hierzu werden alle Häuser durchgegangen und mit der Funktion **vote** überprüft, ob dieses Haus gegen die zu testende Position stimmen würden, wenn als andere Option die andere Position gegeben ist.

Wenn die andere Position eine Mehrheit erhält, bricht die Funktion ab und es wird false zurückgegeben. Nur wenn nachdem alle anderen Postionen durchgegangen wurden und die zu testende Position nie eine Abstimmung verliert, wird true zurückgegeben.

Die Funktion **vote** vergleicht die Abstände eines Hauses mit zwei Positionen. Hierzu wird die Funktion **get_distance** verwendet. Wenn der Abstände zur ersten Position kürzer als zur zweiten Position ist, wird true zurückgegeben, sonst false.

Zuletzt gibt die Funktion **get_distance** den Abstand zwischen zwei Positionen zurück. Hierzu wird der Betrag der Differenz der Adressen der beiden Positionen, dieser Wert wird direkte Distanz genannt, mit der Differenz aus Umfang und der direkten Distanz verglichen. Der kleinere Wert wird zurückgegeben. So entscheidet das Programm, welcher Weg um den See kürzer ist, im oder gegen den Uhrzeigersinn.

3 Beispiele

Nun wird das Programm mit allen Beispieldateien ausgeführt.

eisbuden1.txt Mit der Eingabe
20 7
0 2 3 8 12 14 15
gibt das Programm aus, dass es keine stabilen Positionen gibt.

eisbuden2.txt Mit der Eingabe
50 15
3 6 7 9 24 27 36 37 38 39 40 45 46 48 49
gibt das Programm die Menge an stabilen Positionen aus:
45

eisbuden3.txt Mit der Eingabe
50 16
2 7 9 12 13 15 17 23 24 35 38 42 44 45 48 49
gibt das Programm die Menge an stabilen Positionen aus:
2, 3, 4, 5, 6 und 7

eisbuden4.txt Mit der Eingabe
100 19
6 12 23 25 26 28 31 34 36 40 41 52 66 67 71 75 80 91 92
gibt das Programm die Menge an stabilen Positionen aus:
34

eisbuden5.txt Mit der Eingabe
247 24
2 5 37 43 72 74 83 87 93 97 101 110 121 124 126 136 150 161 185 200 201 230 234 241
gibt das Programm die Menge an stabilen Positionen aus:
93, 94, 95, 96 und 97

eisbuden6.txt Mit der Eingabe
437 36
4 12 17 23 58 61 67 76 93 103 145 154 166 170 192 194 209 213 221 225 239 250 281 299 312 323 337 353 383 385 388 395 405 407 412 429
gibt das Programm aus, dass es keine stabilen Positionen gibt.

eisbuden7.txt

3.1 Eigene Beispiele

myeisbuden0.txt Mit der Eingabe
12 4
0 3 6 9
gibt das Programm die Menge an stabilen Positionen aus:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 und 11
Dies ergibt Sinn, da die Häuser gleichmäßig verteilt sind, womit nie eine Mehrheit gegen jegliche Positionen gefunden werden kann.

myeisbuden0.txt Mit der Eingabe
10 0
0, 1, 2, 3, 4, 5, 6, 7, 8 und 9
Dies ergibt Sinn, da keine Häuser vorhanden sind, die für eine Umlegung stimmen könnten.

4 Quellcode

Dies sind die wichtigsten Funktionen:

```
int get_distance(int circumference, int place_a, int place_b)
{
    int direct_distance = std::abs(place_a - place_b);
    // take shortest way, direct or the othe rdirection
    return std::min(direct_distance, circumference - direct_distance);
}

bool vote(int circumference, int house_place, int old_place, int new_place)
{
    // is new place better?
    if (get_distance(circumference, house_place, new_place) <
    get_distance(circumference, house_place, old_place))
        return true;
    return false;
}

bool is_stable(int circumference, std::vector<int> &houses, int test_place)
{
    // would any other place win an election against test_place?
    for (int other_place = 0; other_place < circumference; other_place++)
    {
        int trues = 0;
        for (int house : houses)
            if (vote(circumference, house, test_place, other_place))
                trues++;

        if (trues > houses.size() - trues)
            return false;
    }
    return true;
}

std::vector<int> get_stabel_places(int circumference, std::vector<int> &houses)
{
    // go thorugh all possible places
    std::vector<int> result;
    for (int test_place = 0; test_place < circumference; test_place++)
        if (is_stable(circumference, houses, test_place))
            result.push_back(test_place);
    return result;
}
```