Aufgabe 3: Eisbudendilemma

Teilnahme-ID: 56860

Bearbeiter/-in dieser Aufgabe:
Christopher Besch

7. April 2021

Inhaltsverzeichnis

1 Lösungsidee

Das Ziel ist es, ein Arrangement bestehend aus drei Positionen für Eisdielen zu generieren, das in einer Abstimmung durch kein anderes Arrangement abgelöst werden kann. Diese Arrangements werden stabil genannt. Hierzu darf die Eisbudendistanz, die Strecke zwischen einem beliebigen Haus und der nächsten Eisbude, von nicht mehr als der Hälfte der Hauser durch ein anderes Arrangement verkürzt werden. Wäre dies der Fall, würde die Ablösung mehr Ja- als Nein-Stimmen erhalten. Hieraus geht hervor, dass für eine optimale Lösung für alle möglichen Arrangements alle Arrangements überprüft werden müssen. Das Arrangement, das auf Stabilität getestet wird, wird Test-Arrangement genannt, das, mit dem überprüft wird, Check-Arrangement. Es lässt sich leicht erkennen, dass ein derartiger Algorithmus mit einer Laufzeit von $O(n^6)$ nicht verwendbar ist.

Als Versuch der Optimierung werden bevor sie getestet werden, alle Arrangement sortiert. Hierzu wird für jedes mögliche Arrangement ein Score berechnet. Dieser entspricht der durchschnittlichen Eisbudendistanz aller Häuser. Nun stellt sich heraus, dass die stabilen Arrangement einen niedrigen Score aufweisen. Dies lässt sich damit erklären, dass je kleiner die Eisbudendistanz eines Hauses in einem Test-Arrangement ist, desto weniger Check-Arrangement existieren, die eine noch geringere Eisbudendistanz für das Haus generieren. Wenn die Eisbudendistanz beispielsweise 0 beträgt existiert kein einziges Check-Arrangement, dem dieses Haus eine Ja-Stimme geben würde, da ein Haus bei gleichbleibender Eisbudendistanz immer gegen einen Wechsel stimmt. Wenn die Eisbudendistanz ein den maximalen Wert, dem halben Umfang des Sees, entspricht, stimmt es für alle Check-Arrangement, abgesehen von denen, die die Eisbudendistanz nicht verändern.

Die durchschnittliche Eisbudendistanz lässt sich dementsprechend als „Zufriedenheitsgrad“ des Dorfes interpretieren. Je höher er ist, desto unwahrscheinlicher wird eine Veränderung durchgesetzt.

Allerdings muss dieser Wert nicht zwangsweise mit der Stabilität eines Arrangements übereinstimmen.

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.

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;
}
```