Aufgabe 3: Eisbudendilemma

Teilnahme-Id: 57063

Bearbeiter/-in dieser Aufgabe:
Raphael Gaedtke

5. April 2021

Inhaltsverzeichnis

1 Lösungsidee

1.1 Allgemeines

Definition 1. *Es sei t der Teichumfang in Schritten und g die Anzahl der Häuser mit $t \geq g$. Im Folgenden wird $t, g > 5$ angenommen, für $g \leq 5$ ist jede Anordnung von Eisbuden stabil, bei der sich jedes Haus eine Adresse mit einer Eisbude teilt (weil die Eisbuden im Inneren und die Häuser außerhalb des Uferweges liegen, können sich ein Haus und eine Eisbude eine Adresse teilen). Diese Anordnungen müssen stabil sein, weil die Häuser, die sich eine Adresse mit einer Eisbude teilen, eine Mehrheit der Häuser insgesamt bilden und immer gegen eine Verlegung stimmen.*
Die Standorte der Eisbuden werden dabei, genau wie die Adressen der Häuser, in der Anzahl von Schritten im Gegenuhrzeigersinn von der Adresse 0 gezählt und sind somit äquivalent zu den Adressen der Häuser.

Definition 2. *Eine Menge V mit $V = \{p_1, p_2, p_3\}$ und somit $|V| = 3$ sei eine Verteilung der drei Eisbuden auf die Adressen p_1, p_2 und p_3. Dabei sei $p_1, p_2, p_3 \in \mathbb{N}_0$ und $0 \leq p_1, p_2, p_3 < t$.*
Im Folgenden werden „Verteilungen der drei Eisbuden“ auch „Eisbudenverteilungen“ genannt.

Definition 3. *Für eine Eisbudenverteilung $V = \{p_1, p_2, p_3\}$ sei die Menge $I_k \subset \mathbb{N}_{<t} \cup \{0\}$ mit $k \in \{1, 2, 3\}$ der Einflussbereich der Eisbude mit der Adresse p_k. Dieser ist die Menge aller Adressen, für die die Eisbude mit der Adresse p_k die am nächsten gelegene Eisbude ist.*
Ist eine Adresse genau gleich weit von zwei Eisbuden mit den Adressen p_i und p_j mit $i < j$ entfernt, wird diese Adresse willkürlich eine der Mengen I_i oder I_j zugeordnet.
Es ist also insbesondere

$$I_1 \cap I_2 = I_1 \cap I_3 = I_2 \cap I_3 = \varnothing \tag{1}$$

und jede Adresse liegt im Einflussbereich genau einer Eisbude.

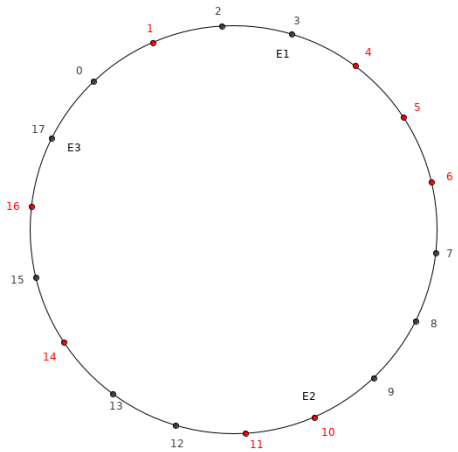

Abbildung 1: Ein Beispiel für ein Bergdorf mit $t = 17$ ung $g = 8$. Jeder Punkt stellt eine Adresse dar, der mit seiner Nummer beschriftet ist. Adressen, auf denen sich Häuser befinden, sind rot markiert und die Adressen der Eisbuden mit „E1“, „E2“ und „E3“ beschriftet.

Beispiel 1. *Im Beispiel aus Abbildung 1 ist $I_1 = \{1, 2, 3, 4, 5, 6\}$, $I_2 = \{7, 8, 9, 10, 11, 12, 13\}$ und $I_3 = \{14, 15, 16\}$, wobei die Adresse 1 nicht eindeutig zugeordnet werden kann.*

Definition 4. *Für eine Eisbudenverteilung $V = \{p_1, p_2, p_3\}$ und $i, j, k \in \{1, 2, 3\}$ sei der Zwischenraum zwischen den beiden Eisbuden auf den Adressen p_i und p_j die Menge der Adressen, deren Distanz zur Eisbude auf der Adresse p_k größer ist als die jeweilige Distanz zu den Eisbuden auf den Adressen p_i und p_j. Die Adressen p_i und p_j werden dabei nicht mitgezählt.*

Definition 5. *Für eine Eisbudenverteilung $V = \{p_1, p_2, p_3\}$ und $i \in \{1, 2, 3\}$ sei die Mitte des Einflussbereichs der Eisbude auf der Adresse p_i eine Teilmenge des Einflussbereichs I_i. Befinden sich in I_i ungerade viele Adressen, auf denen sich Häuser befinden, so enthält die Mitte dieses Einflussbereichs nur die Adresse des mittleren dieser Häuser.*
Befinden sich in I_i gerade viele Adressen, auf denen sich Häuser befinden, so enthält die Mitte dieses Einflussbereiches nur die Adressen der mittleren beiden Häuser und alle Adressen zwischen diesen beiden Häusern.

Beispiel 2. *Im Bergdorf aus Abbildung 1 enthält die Mitte des Einflussbereichs I_1 die Adressen 4 und 5, die Mitte des Einflussbereichs I_2 die Adressen 10 und 11 und die Mitte des Einflussbereichs I_3 die Adressen 14, 15 und 16.*

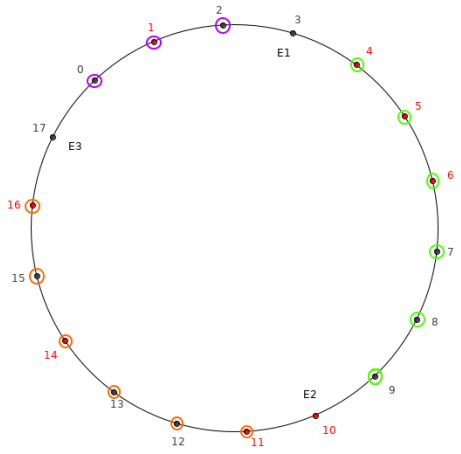

Abbildung 2: Im Bergdorf aus Abbildung 1 sind die Zwischenräume der Eisbuden farblich markiert: Grün steht für den Zwischenraum zwischen den Eisbuden E1 und E2, Orange für den Zwischenraum zwischen den Eisbuden auf E2 und E3 und Lila für den Zwischenraum zwischen den Eisbuden auf E3 und E1.

1.2 Eigenschaften stabiler Eisbudenverteilungen

Um in einem gegebenen Bergdorf stabile Eisbudenverteilungen zu finden oder deren Existenz auszuschließen, werden Eigenschaften beschrieben, die jede stabile Eisbudenverteilung haben muss und die somit notwendige Bedingungen für die Stabilität einer Eisbudenverteilung sind.

1.2.1 Erste Eigenschaft stabiler Eisdenverteilungen

In einer stabilen Eisbudenverteilung können sich im Zwischenraum zwischen zwei Eisbuden nur weniger als $\lfloor \frac{g}{2} \rfloor + 1$ Häuser befinden.

Trifft dies auf eine Eisbudenverteilung $V = \{p_i, p_j, p_k\}$, in der zwischen p_i und p_j mindestens $\lfloor \frac{g}{2} \rfloor + 1$ Häuser liegen, nicht zu, so lässt sich eine Eisbudenverteilung konstruieren, die in V eine Mehrheit hat, womit V nicht stabil sein kann.
Um diese zu konstruieren, werden die beiden Eisbuden auf den Adressen p_i und p_j um jeweils eine Adresse „in“ ihren Zwischenraum verschoben. Dadurch stimmen alle $\lfloor \frac{g}{2} \rfloor + 1$ Häuser innerhalb dieses Zwischenraums für die Verlegung. $\lfloor \frac{g}{2} \rfloor + 1$ Häuser bilden in der Menge der g Häuser allerdings eine Mehrheit, weshalb V nicht stabil sein kann.

1.2.2 Zweite Eigenschaft stabiler Eisbudenverteilungen

In einer stabilen Eisbudenverteilung müssen mindestens zwei Eisbuden in der Mitte ihres jeweiligen Einflussbereichs liegen.

Trifft dies auf eine Eisbudenverteilung $V = \{p_i, p_j, p_k\}$, für die die Eisbuden mit den Adressen p_i und p_j nicht in der Mitte ihres Einflussbereichs liegen, nicht zu, dann lässt sich eine Eisbudenverteilung konstruieren, die eine Mehrheit in V bekommt, womit V nicht stabil sein kann.
Um diese zu konstruieren, werden alle Eisbuden, die nicht in der Mitte ihres Einflussbereichs liegen, um eine Adresse in die Richtung verschoben, in der in ihrem Einflussbereich mehr Häuser liegen. Alle Häuser, auf die diese Eisbuden zugeschoben werden, stimmen für eine Verlegung. In den Einflussbereichen der beiden Eisbuden auf den Adressen p_i und p_j insgesamt stimmen also mindestens zwei Häuser mehr für eine Verlegung als dagegen.
Liegt auch die dritte Eisbude nicht in der Mitte ihres Einflussbereichs, ergibt sich durch ihre Verschiebung auch in diesem eine Mehrheit, im anderen Fall stimmt maximal ein Haus mehr gegen eine Verlegung als dafür, wenn die Eisbude um ein Adresse verschoben wird. Insgesamt ergibt sich dann aber immer noch eine Mehrheit aller Häuser.
In einer stabilen Eisbudenverteilung müssen also mindestens zwei Häuser in der Mitte ihres Einflussbereichs liegen.

1.3 Der Algorithmus

Um in einem gegebenem Bergdorf stabile Eisbudenverteilungen zu finden oder deren Existenz auszuschließen, werden alle möglichen Eisbudenverteilungen von drei untereinander nicht zu unterscheidenden Eisbuden auf die t Adressen im Dorf durchgegangen.
Für jede dieser Eisbudenverteilungen wird in einem ersten Schritt überprüft, ob sie die in Kapitel 1.2 beschriebenen Eigenschaften hat. Ist dies der Fall, wird sie auf Stabilität getestet, was in Kapitel 1.4 beschrieben wird.
Wird während dieses Verfahrens eine stabile Eisbudenverteilung gefunden, wird diese ausgegeben und das Programm beendet. Im anderen Fall gibt es keine stabile Eisbudenverteilung, was ebenfalls durch eine entsprechende Ausgabe zum Ausdruck gebracht wird.

1.4 Testen einer Eisbudenverteilung auf Stabilität

Um zu testen, ob eine gegebene Eisbudenverteilung $V = \{p_1, p_2, p_3\}$ stabil ist, wird überprüft, ob andere Eisbudenverteilungen eine Mehrheit in V bekommen. In der Menge dieser Eisbudenverteilungen soll es immer dann, wenn V nicht stabil ist, mindestens eine Eisbudenverteilung geben, die eine Mehrheit in V bekommt. Gleichzeitig soll diese Menge möglichst wenige Elemente enthalten, um die Laufzeit des Programms zu minimieren.
Ob eine Eisbudenverteilung W eine Mehrheit in V bekommt, wird festgestellt, indem für jedes Haus überprüft wird, ob die Distanz dieses Hauses zur nächsten Eisbuden in W geringer ist als in V. Ist die Anzahl der Häuser mit dieser Eigenschaft mindestens $\lfloor \frac{g}{2} \rfloor + 1$, bekommt W in V eine Mehrheit der g Stimmen.

Zunächst kann festgestellt werden, dass keine Eisbudenverteilung W getestet werden muss, in der es mindestens eine Adresse gibt, auf der sich sowohl in V als auch in W eine Eisbude befindet. Bekommt W nämlich eine Mehrheit in V, so stimmen in der entsprechenden Abstimmung alle Häuser im Einflussbereich der gleichbleibenden Eisbuden gegen die Verlegung. Bekommt W eine Mehrheit in V, so bekommt also eine Eisbudenverteilung, in der alle in W gleichbleibenden Eisbuden auf Adressen verschoben werden, die in V nicht von Eisbuden besetzt sind, ebenfalls eine Mehrheit in V, weil mindestens die gleichen Häuser wie in W einer Verlegung zustimmen.
Entsprechend werden Eisbudenverteilungen mit gleichbleibenden Eisbudenstandorten nicht getestet.

Für eine der verbleibenden Eisbudenverteilungen W können nun zwei Fälle unterschieden werden:

1. In W gibt es zwei Eisbuden, die im gleichen Zwischenraum zweier Eisbuden aus V liegen.
2. In W liegen alle Eisbuden in unterschiedlichen Zwischenräumen zweier Eisbuden aus V.

Gilt für eine Eisbudenverteilung W der erste Fall, stimmen im neuen Einflussbereich der beiden Eisbuden, die sich im gleichen Zwischenraum aus V befinden, nur die Häuser für eine Verlegung, die ebenfalls innerhalb dieses Zwischenraums liegen. Um zu überprüfen, ob es im ersten Fall Eisbudenverteilungen W gibt, die eine Mehrheit in V bekommen, reicht es also aus, Eisbudenverteilungen zu testen, bei denen die zwei Eisbuden im gleichen Zwischenraum um nur eine Adresse von den Eisbuden in V, die diesen Zwischenraum begrenzen, entfernt sind. Eisbudenverteilungen, bei denen alle drei Eisbuden im gleichen Zwischenraum liegen, müssen aus dem gleichen Grund ebenfalls nicht getestet werden.
Es gibt drei Möglichkeiten einen solchen Zwischenraumm auszuwählen. Für jeden dieser Zwischenräume werden zwei Eisbuden entsprechend positioniert, während für die dritte Eisbude jede mögliche Position außerhalb dieses Zwischenraums ausprobiert wird.
Damit sind alle Eisbudenverteilungen W bestimmt, die man auf Stabilität testen muss, um den ersten Fall abzudecken.
Im zweiten Fall werden dann alle Eisbudenverteilungen W, die diese Bedingung erfüllen, bei denen sich jede Eisbude also in einem unterschidelichen Zwischenraum aus V befindet, ausprobiert.

1.5 Laufzeitüberlegungen

In einem Bergdorf mit einem Teichumfang von t gibt es $\binom{t}{3}$ voneinander verschiedene Eisbudenverteilungen, die in dem Fall, dass es keine stabile Eisbudenverteilung gibt, alle durchlaufen werden.
Für jede dieser Eisbudenverteilungen wird zuerst getestet, ob die notwendigen Bedingungen aus Kapitel 1.2 erfüllt sind, wobei schon nach der Prüfung für die erste notwendige Bedinung abgebrochen wird, wenn diese ein negatives Ergebnis hat. Mit dem in Kapitel 2.1 beschriebenen Verfahren zur Bestimmung der Anzahl von Häusern zwischen zwei Adressen laufen beide Prüfungen aber mit amortisiert konstanter Laufzeit.
Bei den Eisbudenverteilungen, die beide notwendige Bedinungen erfüllen, wird das in Kapitel 1.4 beschriebene Testverfahren durchgeführt. Der erste Teil des Testverfahrens hat dabei lineare und der zweite Teil des Testverfahrens kubische Laufzeit.
Die Gesamtlaufzeit ist abhängig von der Anzahl der Eisbudenverteilungen, die beide notwendige Bedingungen erfüllen und im Testverfahren abhängig von den Abständen der einzelnen Eisbuden zueinander, unterschreitet in jedem Fall aber $O(t^6)$, also die Laufzeit der Brute-Force-Lösung.

2 Umsetzung

Die Lösungsidee wird in C++ umgesetzt. Bei der Benennung einiger Variablen im Quelltext wird von einer Nummerierung der Eisbuden einer Eisbudenverteilung von 1 bis 3 ausgegangen, wobei eine höhere Nummer eine höhere Adresse bedeutet.

2.1 Speicherung der Eingabedaten

Der Teichumfang und die Anzahl von Häusern eines gegebenen Bergdorfes werden in Integern und die Hausnummern aller Häuser in einem Container des Typs „vector<int>“ mit der Bezeichnung „hausnummern“ gespeichert. Dieser Container enthält nach dem Einlesen der Daten genau g Elemente.
Parallel muss im Programm ohne großen Zeitaufwand festgestellt werden können, ob auf einer gegebenen Adresse ein Haus steht und wie viele Häuser im Zwischenraum zwischen zwei Adressen liegen. Aus diesem Grund werden der Container „adressebesetzt“ vom Typ „map<int, bool>“ und der Container „haeuserzahl“ vom Typ „map<int, int>“ initialisiert.
Der erste Container ordnet jede Adresse einem bool-Wert zu. Dieser ist True, wenn auf der Adresse ein Haus steht und False, wenn auf dieser Adresse kein Haus steht. Der zweite Container ordnet jeder Adresse die Anzahl der Häuser zu, die auf allen Adressen zwischen 0 und der gegebene Adresse (einschließlich der Randadressen) liegen. Mithilfe dieses zweiten Containers kann durch Addition und Subtraktion von zwei oder drei seiner Elemente die Anzahl von Häusern zwischen zwei beliebigen Adressen ermittelt werden.
Die Initialisierung dieser beiden Container erfolgt zu Beginn des Programms und benötigt lineare Laufzeit, das Abfragen einzelner Informationen kann im weiteren Programmverlauf in amortisiert konstanter Laufzeit realisiert werden.
Um einzelne Eisbudenverteilungen speichern zu können, wird die Datenstruktur „eisbudenverteilung“ definiert, die drei Integer für die Adressen dreier Eisbuden speichert.

2.2 Umsetzung des Algorithmus

Der in Kapitel 1.3 beschriebene Algorithmus wird in der main-Funktion durch drei ineinander verschachtelte for-Schleifen umgesetzt. Diese durchlaufen alle möglichen Eisbudenverteilungen in Form von Zahlentripeln für die Adressen der drei Eisbuden, wobei in den Abbruchbedingungen der Eisbudenverteilungen die erste notwendige Bedingung bereits teilweise überprüft wird.
Jede Eisbudenverteilung wird in eine Variable des Datentyps „eisbudenverteilung“ geschrieben und an Funktionen übergeben, die die notwendigen Bedingungen prüfen und die erzeugte Eisbudenverteilung ggf. auf Stabilität testen. Wird dabei eine stabile Eisbudenverteilung gefunden, werden die for-Schleifen abgebrochen und die Lösung ausgegeben.
Werden die Abbruchbedingungen der drei for-Schleifen erfüllt, ohne dass dabei eine stabile Eisbudenverteilung gefunden wurde, gibt es keine stabile Eisbudenverteilung, weshalb das Programm mit einer entsprechenden Ausgabe beendet wird.
Die Ausgabe erfolgt in jedem Fall an eine vorher vom Nutzer angegebene Ausgabedatei.

2.3 Überprüfen der notwendigen Bedingungen

Ob eine gegebene Eisbudenverteilungen die beiden notwendigen Bedingungen erfüllt, wird in zwei Schritten überprüft, die die beiden Bedingungen abbilden. Diese werden, insofern sie nicht wie oben beschrieben in den Abbruchbedinungen der for-Schleifen des Algorithmus stattfinden, in der Funktion „checkNotwendigeBedingungen“ durchgeführt, die einen bool-Wert zurückliefert.
Die erste dieser Prüfungen lässt sich dabei mithilfe des Containers „haeuserzahl“ umsetzen, durch den die Anzahl der Häuser zwischen zwei beliebigen Adressen bestimmt werden kann.
Für die zweite Prüfung wird zunächst die Anzahl der Adressen zwischen jedem Paar von Eisbuden berechnet und in den Integern „zwischen12“, „zwischen23“ und „zwischen13“ gespeichert. Im zweiten Schritt werden drei Adressen in den drei Integern „mitte12“, „mitte23“ und „mitte13“ gespeichert. Diese Adressen geben für eine ungerade Anzahl von Adressen zwischen zwei Eisbuden die Adresse an, die von beiden Eisbuden gleich weit entfernt ist. Ist die Anzahl von Adressen zwischen zwei Eisbuden gerade, speichern diese Variablen die Adresse, die am nächsten an der Eisbude mit der größeren Adresse ist, gleichzeitig aber im Einflussbereich der Eisbude mit der niedrigeren Adresse liegt.
Danach werden die drei Eisbuden durchlaufen, wobei in der Zählervariablen „inmitte“, die zu Beginn auf den Wert 0 gesetzt wird, festgehalten wird, wie viele Eisbuden in der Mitte ihres Einflussbereichs liegen. Für jede der Eisbuden wird in den zwei Integern „links“ und „rechts“ gespeichert, wie viele Häuser auf der jeweiligen Seite der Eisbude im Einflussbereich derselben liegen. Häuser, die sich eine Adresse mit der Eisbude teilen oder auf einem der „Mittelfelder“, insofern diese nicht zum Einflussbereich der Eisbude gehören, liegen, werden dabei nicht mitgezählt.
Dadurch sollen Edgecases abgefangen werden, in denen Häuser auf diesen Adressen unterschiedlichen Einflussbereichen zugeordnet werden können oder in diesem Einflussbereich zu verschiedenen Seiten der Eisbude gezählt werden dürfen. Stattdessen wird die Existenz dieser Häuser abgefragt, um eine eventuelle Ungleichheit zwischen den Werten der Variable „links“ und „rechts“ auszugleichen.
Lässt sich nach diesem Prozess eine Gleichheit dieser Werte feststellen, liegt die überprüfte Eisbude in der Mitte ihres Einflussbereichs und der Zähler „inmitte“ wird um 1 erhöht. Enthält dieser Zähler nach der Überprüfung aller Eisbuden den Wert 2 oder 3, ist auch die zweite notwendige Bedingung erfüllt.

2.4 Testen einer Eisbudenverteilung auf Stabilität

Die Stabilität einer Eisbude wird in der Funktion „evPruefen“ getestet. Diese Funktion liefert einen bool-Wert zurück, der angibt, ob die als Argument an die Funktion übergebene Eisbudenverteilung stabil ist oder nicht.
Wie in Kapitel 1.4 beschrieben, wird sie dabei zunächst mit den Eisbudenverteilungen verglichen, bei denen sich zwei Eisbuden im gleichen Zwischenraum der ursprünglichen Eisbudenverteilung befinden. Für jede der drei Möglichkeiten, diesen Zwischenraum zu wählen, wird eine for-Schleife eingesetzt, die alle möglichen Eisbudenverteilungen mit dieser Eigenschaft durchläuft. Abschließend werden auch die Eisbudenverteilungen, bei denen alle Eisbuden in unterschiedlichen Zwischenräumen aus der ursprünglichen Eisbudenverteilung liegen, untersucht, wozu drei ineinander verschachtelte for-Schleifen verwendet werden.
Ob eine gegeben Eisbudenverteilung eine Mehrheit in einer anderen gegebenen Eisbudenverteilung bekommt, wird in der Funktion „abstimmen“ ermittelt, die wiederum einen bool-Wert zurückliefert.
Innerhalb dieser Funktion werden in einer for-Schleife die einzelnen Häuser durchlaufen und für jedes Haus mit einer entsprechenden Funktion die Distanz dieses Hauses zur nächstgelegenen Eisbude in der „neuen“ und der „alten“ Eisbudenverteilung bestimmt. Die Anzahl der Häuser, deren Distanz zur nächstgelegenen Eisbude geringer wird, wird in einer Zählervariablen gespeichert, die abschließend mit der notwendigen Mehrheit für eine Verlegung der Eisbuden, die sich aus der Anzahl der Häuser berechnen lässt, verglichen wird.

3 Beispiele

Ausgabe für die Eingabedatei „eisbuden1.txt“:

Es gibt mindestens einen stabilen Eisbudenstandort!
Dabei haben die Eisbuden die Adressen 2, 5 und 14.

Ausgabe für die Eingabedatei „eisbuden2.txt“:

Es gibt keine stabilen Eisbudenstandorte.

Ausgabe für die Eingabedatei „eisbuden3.txt“:

Es gibt mindestens einen stabilen Eisbudenstandort!
Dabei haben die Eisbuden die Adressen 0, 17 und 42.

Ausgabe für die Eingabedatei „eisbuden4.txt“:

Es gibt keine stabilen Eisbudenstandorte.

Ausgabe für die Eingabedatei „eisbuden5.txt“:

Es gibt mindestens einen stabilen Eisbudenstandort!
Dabei haben die Eisbuden die Adressen 83, 128 und 231.

Ausgabe für die Eingabedatei „eisbuden6.txt“:

Es gibt keine stabilen Eisbudenstandorte.

Ausgabe für die Eingabedatei „eisbuden7.txt“:

Es gibt mindestens einen stabilen Eisbudenstandort!
Dabei haben die Eisbuden die Adressen 114, 285 und 416.

Laufzeit:

Insgesamt benötigte das Programm auf meinem Computer für keine der fünf ersten Eingabedateien mehr als 20 Sekunden Rechenzeit.
Für die Eingabedatei „eisbuden6.txt“ wurden etwa zwei Minuten und für die Eingabedatei „eisbuden7.txt“ etwa drei Minuten Rechenzeit benötigt.

4 Quellcode

Die Definition des Datentyps für Eisbudenverteilungen:

```
struct eisbudenverteilung //Speichert eine Eisbudenverteilung durch die Adressen der Eisbuden
{
    int position1;
    int position2;
    int position3;
};
```

Die Funktion zur Ermittlung der Distanz eines Hauses zur nächstgelegenen Eisbude in einer gegebenen Eisbudenverteilung, die eine Implementierung der Betragsfunktion nutzt:

```
int distanzEisbude(eisbudenverteilung buden, int haus, int teichumfang)
{
    //berechnet fuer ein Haus die minimale Distanz zur naechsten Eisbude
    int distanz1 = min(betrag(haus-buden.position1), teichumfang-betrag(haus-buden.position1));
    //Distanz zur ersten Eisbude
    int distanz2 = min(betrag(haus-buden.position2), teichumfang-betrag(haus-buden.position2));
    int distanz3 = min(betrag(haus-buden.position3), teichumfang-betrag(haus-buden.position3));
    return min(distanz1, min(distanz2, distanz3));
}
```

Die Funktion „abstimmen“:

```
bool abstimmen(eisbudenverteilung statusquo, eisbudenverteilung neu, vector<int>* hausnummern,
               int haeuser, int teichumfang)
{
    //Diese Funktion liefert True zurueck, wenn die zweite Eisbudenverteilung eine Mehrheit
    //von der ersten aus bekommen wuerde und False sonst
    int mehrheit = haeuser/2 + 1;
    //Speichert, wie viele Haeuser mindestens fuer eine Umlegung zustimmen muessen
    int zustimmungen = 0; //Zahelt die Haueser, die einer Verlegung zustimmen
    for(int i : (*hausnummern))
    {
        if(distanzEisbude(neu, i, teichumfang) < distanzEisbude(statusquo, i, teichumfang))
        {
            zustimmungen++;
            //Bei Gleichheit stimmt das Haus gegen die Verlegung!
        }
    }
    if(zustimmungen >= mehrheit)
    {
        return true; //Die Mehrheit stimmt fuer eine Verlegung
    }
    return false;
}
```

Die Funktion zum Überprüfen der notwendigen Bedingungen für eine gegebene Eisbudenverteilung:

```
bool checkNotwendigeBedingungen(eisbudenverteilung test, int haeuser, int teichumfang,
                                map<int, int>* haeuseranzahlen, map<int, bool>* adressebesetzt)
{
    //Diese Funtion ueberprueft, ob eine Eisbudenverteilung die notwendigen Bedingungen
    //fuer eine stabile Eisbudenverteilung erfuellt
    //Ist dem so. liefert sie True zurueck und False sonst
    //Erst ueberpruefen, ob "zu viele" Haueser zwischen zwei Eisbuden liegen
    // - deren Adresse sind aufsteigend geordnet
    int noetigFuerMehrheit = haeuser/2 + 1;
    //Liegen zwischen zwei Eisbuden mindestens so viele Haueser,
    //kann die Eisbudenverteilung test nicht stabil sein
    if((*haeuseranzahlen)[test.position1-1] + (*haeuseranzahlen)[teichumfang-1] -
       (*haeuseranzahlen)[test.position3] >= noetigFuerMehrheit)
       {
           //Eisbudenverteilung test kann nicht stabil sein
           return false;
       }
    //Diese Bedingung wird teilweise schon in der main-Funktion getestet
```

```
20      //Nun zaehlen, wie viele Eisbuden in der Mitte ihres Einflussbereichs liegen:
        int inmitte = 0; //Zaehlt die in der Mitte ihres Einflussbereichs liegenden Eisbuden
22      int zwischen12 = test.position2-1-test.position1;
        //Anzahl der Adressen zwischen den Eisbuden 1 und 2
24      int zwischen23 = test.position3-1-test.position2;
        int zwischen13 = teichumfang-zwischen12-zwischen23-3;
26      //Jede der folgenden Variablen bezieht sich jeweils auf den Zwischenraum zwischen zwei Eisbuden
        //Liegen in diesem gerade viele Adressen, enthaelt sie die letzte Adresse,
28      die im Einflussbereich der Eisbude mit der niedrigeren Adresse liegt
        //Im anderen Fall enthaelt sie die Adresse, die von beiden Eisbuden gleich weit entfernt ist.
30      int mittelfeld12, mittelfeld23, mittelfeld13;
        mittelfeld12 = test.position1+(zwischen12/2)+(zwischen12%2);
32      mittelfeld23 = test.position2+(zwischen23/2)+(zwischen23%2);
        mittelfeld13 = (test.position3+(zwischen13/2)+(zwischen13%2))%teichumfang;
34      if(zwischen13%2 == 0)
        {
36          mittelfeld13 = (mittelfeld13+1)%teichumfang;
        }
38
        //Variablen fuer die zwei Haelften des Einflussbereichs einer Eisbude
40      //bzw. die Anzahl der Haeuser darin
        //Links steht fuer die Richtung, in der die Adressnummern abnehmen
42      int links, rechts;

44      //Nun werden die drei Eisbuden durchlaufen:

46      //Eisbude Nr. 1
        if(mittelfeld13 < test.position1)
48      {
            if(zwischen13%2 == 0)
50          {
                links = (*haeuseranzahlen)[test.position1-1]-(*haeuseranzahlen)[mittelfeld13-1];
52          }
            else
54          {
                links = (*haeuseranzahlen)[test.position1-1]-(*haeuseranzahlen)[mittelfeld13];
56          }
        }
58      else
        {
60          //Aehnliche Berechnungen, die hier gekuerzt wurden
        }
62      if(zwischen12 % 2 == 1)
        {
64          rechts = (*haeuseranzahlen)[mittelfeld12-1]-(*haeuseranzahlen)[test.position1];
        }
66      else
        {
68          rechts = (*haeuseranzahlen)[mittelfeld12]-(*haeuseranzahlen)[test.position1];
        }
70
        if((*adressebesetzt)[mittelfeld13] && zwischen13%2 == 1 && links < rechts)
72      {
            links++;
74      }
        if((*adressebesetzt)[mittelfeld12] && zwischen12%2 == 1 && rechts < links)
76      {
            rechts++;
78      }
        if(betrag(rechts - links) == 1 && (*adressebesetzt)[test.position1])
80      {
            rechts = links;
82      }

84      if(rechts == links)
        {
86          inmitte++;
        }
88
        //Es folgen aequivalente Anweisungen fuer die Eisbuden Nr. 2 und Nr. 3
90      //Diese sind hier gekuerzt
```

```
92      if(inmitte < 2)
        {
94          //Es liegen mindestens zwei Eisbuden ausserhalb der Mitte ihres Einflussbereichs
            return false;
96      }

98      return true;
}
```

Die Funktion, die eine gegebene Eisbudenverteilung auf Stabilität testet:

```
1  bool evPruefen(eisbudenverteilung test, vector<int>* hausnummern, int haeuser, int teichumfang)
   {
3      //Ueberprueft eine Eisbudenverteilung auf Stabilitaet
       //Diese Funktion liefert True zurueck, wenn die zu untersuchende Eisbudenverteilung stabil ist
5      //und False sonst

7      //Durchlaufen von Eisbudenverteilungen, in denen sich zwei Eisbuden zueinander bewegt haben:
       //1 und 2 bewegen sich aufeinander zu:
9      if(test.position2 - test.position1-1 >= 2) //Die beiden koennen sich zueinander bewegen
       {
11         for(int p = test.position2+1; (p%teichumfang) != test.position1; p++)
           {
13             eisbudenverteilung vergleich;
               vergleich.position1 = test.position1+1;
15             vergleich.position2 = test.position2-1;
               vergleich.position3 = (p%teichumfang);
17             if(abstimmen(test, vergleich, hausnummern, haeuser, teichumfang))
                   return false;
19         }
       }
21     //Aehnliche Berechnungen fuer 2 und 3 bzw 1 und 3, die hier gekuerzt sind

23     //Durchlaufen von Eisbudenverteilungen, bei denen sich alle Eisbuden
       //in die gleiche Richtung bewegt haben:
25     for(int a = test.position1+1; a < test.position2; a++)
       {
27         for(int b = test.position2+1; b < test.position3; b++)
           {
29             for(int c = test.position3+1; (c%teichumfang) != test.position1; c++)
               {
31                 eisbudenverteilung vergleich;
                   vergleich.position1 = a;
33                 vergleich.position2 = b;
                   vergleich.position3 = (c%teichumfang);
35                 if(abstimmen(test, vergleich, hausnummern, haeuser, teichumfang))
                       return false;
37             }
           }
39     }

41     //Die Eisbudenverteilung ist stabil!!!
       return true;
43 }
```

Die main-Funktion:

```
1  int main()
   {
3      //Variablen fuer das Speichern der eingegebenen Daten:
       int teichumfang; //Umfang des Sees in Schritten
5      int haeuser; //Anzahl der Haeuser
       vector<int> hausnummern (0); //Die Adressen der Haeuser um den See
7      string ausgabedatei; //Der Name der Ausgabedatei

9

       //Einlesen der Eingabe:
11     eingabeeinlesen(&teichumfang, &haeuser, &hausnummern, &ausgabedatei);

13     //Container fuer das Ueberpruefen der notwendigen Bedingungen:
       map<int, int> haeuserzahl; //Ordnet jeder Adresse die Anzahl der Haueser im Bereich
```

```
15      //zwischen der Adresse 0 und dieser Adresse (inklusive) zu
        map<int, bool> adressebesetzt; //Gibt fuer jede Adresse an, ob sich ein Haus auf ihr befindet.
17      init_container(&haeuserzahl, &adressebesetzt, &hausnummern, haeuser, teichumfang);
        //Initialisieren der beiden Container
19
        bool gefunden = false; //True, wenn eine stabile Eisbudenverteilung gefunden wurde
21      eisbudenverteilung evstabil; //Speichert eine evtl. existierende stabile Eisbudenverteilung

23      //Nun durch Eisbudenverteilungen gehen:
        for(int a = 0; a < teichumfang && !gefunden; a++)
25      {
            for(int b = a+1; b < teichumfang && haeuserzahl[b-1] - haeuserzahl[a] < haeuser/2 + 1 &&
27          !gefunden; b++)
            {
29              for(int c = b+1; c < teichumfang &&
                haeuserzahl[c-1] - haeuserzahl[b] < haeuser/2 + 1 && !gefunden; c++)
31              {
                    eisbudenverteilung test;
33                  test.position1 = a;
                    test.position2 = b;
35                  test.position3 = c;
                    if(checkNotwendigeBedingungen(test, haeuser, teichumfang,
37                     &haeuserzahl, &adressebesetzt))
                    {
39                      testcounter++;
                        if(evPruefen(test, &hausnummern, haeuser, teichumfang))
41                      {
                            //Eine stabile Eisbudenverteilung wurde gefunden!
43                          gefunden = true;
                            evstabil = test;
45                      }
                    }
47              }
            }
49      }

51      //Ausgabe:
        ausgabe(ausgabedatei, evstabil, gefunden);
53
        return 0;
55  }
```