Aufgabe 3: Eisbudendilemma

Teilnahme-ID: 55908

Jonathan Busch

19. April 2021

Inhaltsverzeichnis

1 Annahmen

- Die Häuser und Eisbuden bzw. ihre Zugänge sind punktförmig, sodass die Länge des Weges zwischen zwei Adressen exakt ihrer numerischen Differenz $\min(|a-b|, U-|a-b|)$ auf dem Kreisrand entspricht.
- Sowohl Häuser als auch Eisbuden können nur an ganzzahligen Adressen stehen, außerdem hat der See einen ganzzahligen Umfang. Somit ist der Abstand zwischen zwei Adressen auch immer ganzzahlig. Diese Annahme führt zu dem Schluss, dass es nur endlich viele Adressen gibt und ermöglicht damit den Brute-Force-Ansatz.
- Aus diesem Grund beziehen sich Intervallangaben nur auf die ganzen Zahlen (statt wie üblich auf $\mathbb{R}$). Außerdem sind die Intervalle nicht immer im üblichen Sinne zusammenhängend, da die Intervalle auch über den Adressensprung bei U und 0 hinausgehen können. Konkret heißt das: Wird ein Intervall x als $[a, b]$ (für andere Schreibweisen, z. B. $[a, b[$, gilt entsprechendes) notiert, dann ist

$$x = \begin{cases} \mathbb{Z} \cap \{z \mid a \leq z \leq b\} & a \leq b \\ \mathbb{Z} \cap (\{z \mid 0 \leq z \leq b\} \cup \{z \mid a \leq z < U\}) & b < a \end{cases}$$

2 Lösungsidee

2.1 Problemstellung

Gegeben ist der Umfang des Sees U sowie eine Menge von Häusern H, repräsentiert durch ihre Adressen. Außerdem gibt es 3 Eisbuden, die selbst bestimmte Adressen haben. Eine Konstellation von Eisbuden bezeichnet hier eine Menge von Eisbudenadressen mit 3 Elementen. Es ist eine Abstandsfunktion zwischen zwei Adressen a und b um den See definiert mit $dist(a, b) = \min(|a - b|, U - |a - b|)$ und eine Funktion, die den Abstand eines Hauses h zur nächsten Eisbude in der Konstellation K bezeichnet: $minDist(h, K) = \min_{k \in K}(dist(h, k))$. Außerdem gibt es eine Vergleichsfunktion

$$v(A, B) = \sum_{h \in H} \begin{cases} 1 & minDist(h, B) < minDist(h, A) \\ -1 \end{cases}$$

die angibt, wieviele *Ja*-Stimmen mehr als *Nein*-Stimmen die Konstellation B bei einer Abstimmung in der Konstellation A bekommen würde. Gesucht ist nun eine Konstellation X, sodass $\nexists Y : v(X, Y) > 0$.

2.2 Lösung

Die grundsätzliche Idee ist, für alle möglichen Konstellationen herauszufinden, ob sie stabil sind. Um das für eine Konstellation K herauszufinden, genügt es, eine Konstellation L zu finden, sodass $v(K, L) > 0$, oder nachzuweisen, dass $\nexists L$. Alternativ, um beide Fragen gleichzeitig zu beantworten, kann man ausgehend von K eine Konstellation L' generieren, bei der $v(K, L')$ maximal wird, d. h. dass sich der Weg zur nächsten Eisbude bei möglichst vielen Häusern

verkürzt. Ist $v(K, L') > 0$, dann gilt auch $L' = L$. Sonst, wenn $v(K, L') \leq 0$, dann $\nexists L$, da $v(K, L')$ mit L' den maximal möglichen Wert hat, woraus auch folgt, dass K stabil ist.

2.3 Generieren von L'

Vorab kann man sich überlegen, warum ausgeschlossen werden kann, dass L' gemeinsame Eisbudenadressen mit K hat: Angenommen, x ist eine Eisbude mit $x \in L' \wedge x \in K$.

Angenommen, es gibt ein Haus y, dessen nächste Eisbude in L' x ist. Dann kann es sein, dass

1. x auch eine nächste Eisbude (mit kürzestem Weg) in K ist. In diesem Fall verkürzt sich der Weg von y nicht, die Bewohner werden mit *Nein* stimmen.
2. x nicht die nächste Eisbude von y in K ist. Dann muss es in K eine andere Eisbude geben, die dichter an y liegt. In diesem Fall verlängert sich der Weg zur nächsten Eisbude mit Einführung von L': die Bewohner werden ebenfalls mit *Nein* stimmen.

Deshalb kann x den Weg von y nicht verkürzen. Nun gibt es zwei Möglichkeiten:

1. Auf mindestens einer Seite von x gibt es Häuser y, deren nächste Eisbude x ist. In diesem Fall kann man die Eisbude um 1 in eine Richtung verschieben, in der sich ein Haus befindet. Der Weg würde sich nun bei mindestens diesem Haus verkürzen, also würde die neue Konstellation ohne gemeinsame Eisbudenadressen mit K mehr Ja-Stimmen erhalten. Es folgt: $v(K, L')$ kann nicht maximal sein.
2. Auf keiner Seite der gemeinsamen Eisbudenadresse gibt es für (1) relevante Häuser, d. h. $\nexists y$. In diesem Fall kann man die Eisbude dennoch um 1 Position in eine beliebige Richtung verschieben, wobei die neue Konstellation ohne gemeinsame Eisbudenadresse genau so viele Ja-Stimmen erhält wie L'. In diesem Fall kann $v(K, L')$ zwar maximal sein, es gibt jedoch mindestens 1 andere Konstellation, die genau so viele Ja-Stimmen erhält und ohne gemeinsame Eisbudenadressen auskommt. Es gibt also mindestens ein anderes L'' mit $v(K, L') \leq v(K, L'')$, L' muss also in diesem Fall nicht berücksichtigt werden, wenn L'' generiert werden kann.

Um L' zu generieren, muss man die Eisbudenadressen so auswählen, dass sich der Weg zur nächsten Eisbude von möglichst vielen Häusern verkürzt. Offensichtlich verkürzt jede Eisbude den Weg nur in (maximal) einem zusammenhängenden Intervall, das auf jeder Seite bis zur Hälfte des Weges zur nächsten Eisbude reicht, aber (ebenso offensichtlich) niemals über die umgebenden Eisbuden hinaus. Um das zu verdeutlichen, zeigt Abbildung 1 mögliche Abstandsfunktionen.

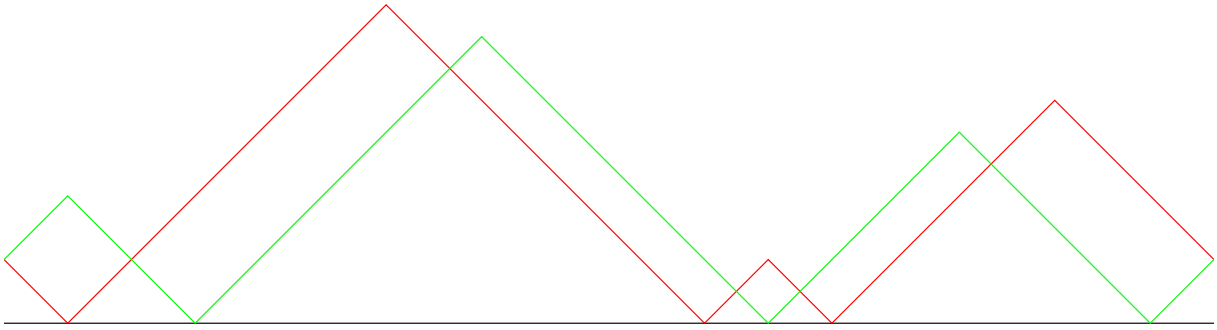


Abbildung 1: mögliche Abstandsfunktionen von K (rot) und L' (grün) ($minDist(x)$)

Der Bequemlichkeit halber werden diese Abstandsfunktionen ihrer Form nach im Folgenden „Kronenfunktion“ genannt, die Intervalle zwischen den Nullstellen (Eisbudenadressen) werden dementsprechend als Zacken bezeichnet. Die letzte Beobachtung lässt sich nun so formulieren: Das Intervall i, in dem eine Eisbude x die Abstandsfunktion von L' gegenüber der Abstandsfunktion von K verkürzt, liegt vollständig innerhalb der Zacke Z (von K), in der auch x liegt. Zudem gilt: Ist x die einzige Eisbude in Z, so ist i (asymptotisch) genau halb so groß wie Z.

Da schon ausgeschlossen wurde, dass K und L' gemeinsame Eisbudenadressen haben, kann man nun mehrere Fälle unterscheiden, wie sich die drei Eisbuden von L' in den Zacken von K verteilen lassen:

1. In jeder Zacke wird genau eine Eisbude platziert $(1+1+1)$. Hierbei gibt es 1 Möglichkeit für jedes K.
2. In einer Zacke werden zwei Eisbuden platziert $(2+1+0)$. Hierbei gibt es 6 Möglichkeiten für jedes K.
3. In einer Zacke werden alle Eisbuden platziert $(3+0+0)$.

In Bezug auf Möglichkeit (2) kann man beobachten, dass 2 Eisbuden innerhalb einer Zacke Z ausreichen, damit $Z = i$[1]. Das erreicht man, indem man die zwei Eisbuden an die Ränder von Z stellt (siehe Abbildung 2). Mit dieser Beobachtung ergibt (3) keinen Sinn mehr, da es offensichtlich sinnlos ist, drei Eisbuden in dieselbe Zacke zu stellen, obwohl schon zwei ausreichen, um die Abstandsfunktion in der gesamten Zacke zu verkürzen.

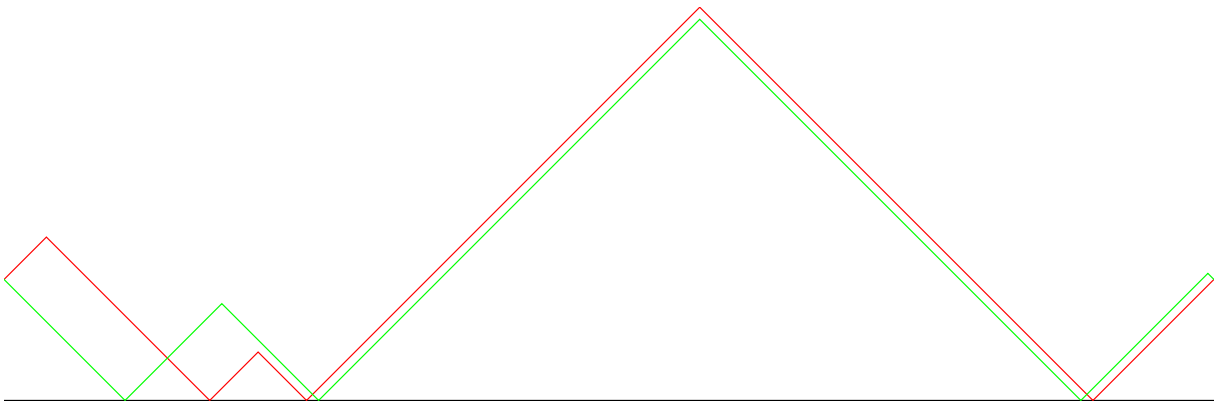


Abbildung 2: Beispiel für Möglichkeit (2)

Die Möglichkeiten, die untersucht werden müssen, sind also (1) und (2).

Zur Erinnerung: Da L' keinesfalls stabil sein muss, sondern nur $v(K, L')$ für ein vorgegebenes K maximal sein muss, ist unwichtig, um wie viel sich der Weg bei bestimmten Häusern verkürzt; wichtig ist nur, *dass* er sich verkürzt.

Um für (1) optimale Eisbudenadressen zu finden, genügt es, da die Zacken von K in dieser Hinsicht unabhängig voneinander sind, für jede Zacke Z einzeln das Intervall halb so groß wie Z zu finden, in dem die meisten Häuser stehen. Diese Informationen können vorberechnet werden, da sie unabhängig von K sind und mehrfach (für verschiedene K) gebraucht werden.

Um für (2) $(2+1+0)$ optimale Eisbudenadressen zu finden, muss für die Eisbude “1” (die alleine in einer Zacke steht, wie bei (1)) dieselbe Information verwendet werden wie in (1). Die

[1] siehe Annahme (3): insbesondere hier ist wichtig: nicht $Z = i$, sondern $Z \cup \mathbb{Z} = i \cup \mathbb{Z}$.

Adressen der anderen beiden Eisbuden herauszufinden ist trivial, diese stehen an den Rändern ihrer Zacke von K.

L' schließlich ermittelt man, indem man (1) und alle 6 Möglichkeiten für (2) generiert die Konstellation mit den meisten Stimmen als L' wählt.

2.4 Vorberechnungen

Da häufig Abfragen vorkommen, wie viele Häuser in einem Intervall stehen, bietet sich die Nutzung eines Prefixsummenarrays an. Darin steht am Index i, wie viele Häuser sich im Intervall $[0, i[$ befinden. Damit kann jede Abfrage in $\mathcal{O}(1)$ beantwortet werden.

Außerdem wird vorberechnet, wo eine Eisbude in einer Zacke Z stehen muss, damit sie den Weg zu möglichst vielen Häusern (Anzahl n) verkürzt, da Abfragen dieser Form zentraler Bestandteil des Generierens von L' sind. Um dies zu "berechnen", wird jede mögliche Position x innerhalb von $Z =]a, b[$ ausprobiert. Sei $i =]c, d[$ nun das Intervall, in dem sich der Weg verkürzt. c und d können in $\mathcal{O}(1)$ berechnet werden, da bekannt ist, dass c die mittlere Position zwischen a und x ist und d entsprechend die mittlere Position zwischen b und x. n kann danach mit dem Prefixsummenarray auch in $\mathcal{O}(1)$ berechnet werden.

3 Korrektheit

Eine *stabile* Konstellation ist gemäß der Aufgabenstellung eine Konstellation, die sich nicht mehr verändern kann, also für die keine Konstellation existiert, die bei einer Abstimmung eine Mehrheit bekäme.

Das Verfahren generiert für eine vorgegebene Konstellation K die Konstellation L', die bei Einführung den Weg zu möglichst vielen Häusern verkürzen würde. Daraus kann direkt auch die Anzahl der Stimmen für L' bei einer Abstimmung errechnet werden, da jedes Haus, bei dem sich der Weg verkürzt, *für* L' stimmen würde, und jedes andere Haus *dagegen*. Dass tatsächlich eine Konstellation L' mit maximalem $v(K, L')$ generiert wird, wurde bereits ausführlich im Teil Lösungsidee begründet.

Wenn nun L' keine Mehrheit bei einer Abstimmung bekäme, dann weiß man, dass auch keine andere Konstellation eine Mehrheit bekommen kann, da L' die maximal mögliche Stimmenzahl bekommt. Die Anzahl der Stimmen für L' reicht also als Entscheidungskriterium für die Stabilität einer Konstellation K.

Da außerdem alle möglichen Konstellationen K auf ihre Stabliltät geprüft werden, werden auch alle stabilen Konstellationen gefunden.

4 Laufzeitanalyse

Die Laufzeit ist abhängig von dem Unfang U des Sees.

Die Berechnung des Prefixsummenarrays ist in $\mathcal{O}(U)$ möglich.

Die Vorberechnung, wo die optimale Eisbudenadresse in einem bestimmten Intervall i ist, liefert U^2 Werte (für U^2 mögliche Intervalle), von denen jeder in $\mathcal{O}(|i|)$ berechnet wird. $|i|$ ist nur durch U beschränkt, daher ist $\mathcal{O}(|i|) \subset \mathcal{O}(U)$. Insgesamt läuft die Vorberechnung damit in $\mathcal{O}(U^3)$.

Es gibt $\binom{U}{3} \in \mathcal{O}(U^3)$ mögliche Konstellationen. Da L' für jede Konstellation in $\mathcal{O}(1)$ generiert werden kann, liegt die Laufzeit des Brute-Force-Teils des Algorithmus auch in $\mathcal{O}(U^3)$.

Insgesamt liegt die Laufzeitkomplexität des Algorithmus damit in $\mathcal{O}(U + U^3 + U^3) = \mathcal{O}(U^3)$.

Die Speicherkomplexität ist außerdem von der Anzahl R der stabilen Konstellationen abhängig.

Der Algorithmus benötigt insgesamt $\mathcal{O}(U^2 + R)$ Speicher, wobei R, theoretisch nur durch $\binom{U}{3}$ beschränkt (das kommt tatsächlich vor, z. B. wenn es keine Häuser gibt), in der Praxis vernachlässigbar ist.

5 Umsetzung

Da die ganze Lösungidee schon im gleichnamigen Teil erklärt wurde, kann zur Umsetzung nicht wesentlich mehr geschrieben werden, als dass das Programm in Java implementiert wird.

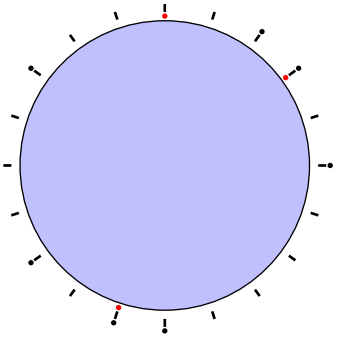
Die Implementation besteht aus der `main`-Methode sowie aus der Hilfsmethode `housesInInterval`.

Die Methode `main` enthält die Eingabe, den Algorithmus und die Ausgabe.

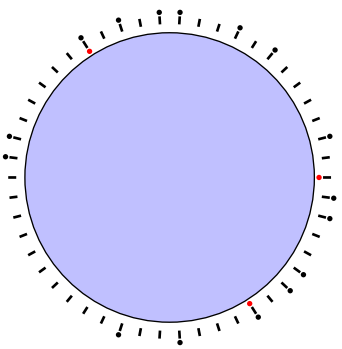
Die Methode `housesInInterval` ist eine Hilfsmethode zum Abrufen von korrekten Daten aus dem Prefixsummenarray. Das Attribut „korrekt“ ist an dieser Stelle sehr wichtig: Da der See kreisförmig ist und damit die Adresse $U - 1$ direkt neben der Adresse 0 liegt, können Adressenintervalle auch über diesen Adressensprung hinausgehen. Damit die Anzahl der Häuser dort trotzdem korrekt berechnet wird, ist eine Fehlerüberprüfung notwendig, die eben in `housesInInterval` stattfindet.

6 Beispiele

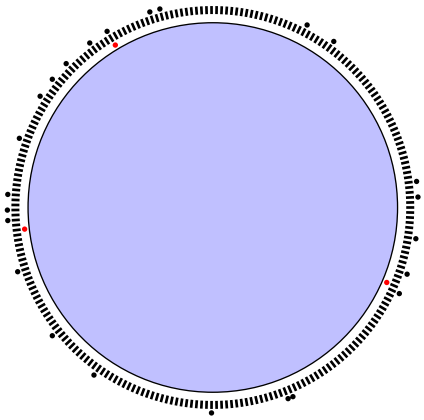
Zunächst die Beispiele von der BwInf-Webseite:



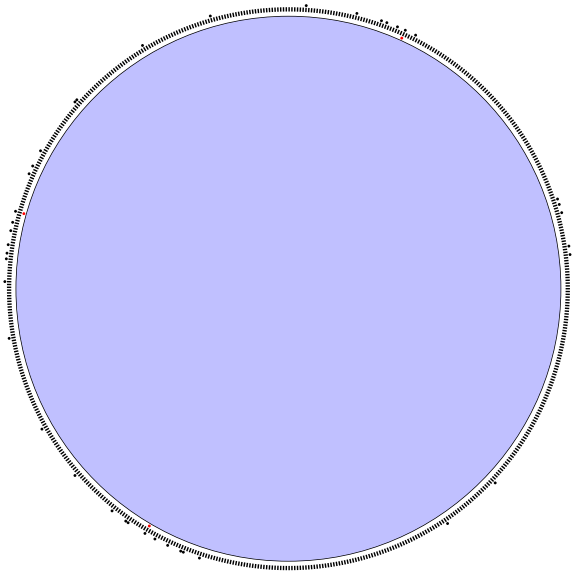
Beispiel 1



Beispiel 3



Beispiel 5



Beispiel 7

```
--- Befehl ---
$ java A3Main # Beispiel 1

--- INPUT  ---
20 7
0 2 3 8 12 14 15

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000002s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.000393s
Absuchen aller Eisbudenkonstellationen in 0.001861s
Gesamte Berechnung in 0.002934s
Es gibt 11 stabile Konstellationen:
[2, 5, 14]
[2, 6, 14]
[2, 7, 14]
[2, 8, 14]
[2, 9, 14]
[2, 10, 14]
[2, 10, 15]
[2, 11, 14]
[2, 11, 15]
[2, 12, 14]
[2, 12, 15]

--- Befehl ---
$ java A3Main # Beispiel 2

--- INPUT  ---
```

```
50 15
3 6 7 9 24 27 36 37 38 39 40 45 46 48 49

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000003s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.005393s
Absuchen aller Eisbudenkonstellationen in 0.015708s
Gesamte Berechnung in 0.021875s
Es gibt 0 stabile Konstellationen:
<empty list>

--- Befehl ---
$ java A3Main # Beispiel 3

--- INPUT  ---
50 16
2 7 9 12 13 15 17 23 24 35 38 42 44 45 48 49

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000003s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.005566s
Absuchen aller Eisbudenkonstellationen in 0.018590s
Gesamte Berechnung in 0.024932s
Es gibt 2 stabile Konstellationen:
[0, 17, 42]
[1, 17, 42]

--- Befehl ---
$ java A3Main # Beispiel 4

--- INPUT  ---
100 19
6 12 23 25 26 28 31 34 36 40 41 52 66 67 71 75 80 91 92

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000004s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.022344s
Absuchen aller Eisbudenkonstellationen in 0.035583s
Gesamte Berechnung in 0.058822s
Es gibt 0 stabile Konstellationen:
<empty list>

--- Befehl ---
$ java A3Main # Beispiel 5
```

```
--- INPUT  ---
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

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000007s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.093417s
Absuchen aller Eisbudenkonstellationen in 0.323203s
Gesamte Berechnung in 0.417594s
Es gibt 6 stabile Konstellationen:
[83, 128, 231]
[83, 129, 231]
[83, 129, 232]
[83, 130, 231]
[83, 130, 232]
[83, 130, 233]

--- Befehl ---
$ java A3Main # Beispiel 6

--- INPUT  ---
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

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000015s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.445505s
Absuchen aller Eisbudenkonstellationen in 1.143471s
Gesamte Berechnung in 1.590042s
Es gibt 0 stabile Konstellationen:
<empty list>

--- Befehl ---
$ java A3Main # Beispiel 7

--- INPUT  ---
625 40
12 15 27 30 32 110 114 117 121 123 132 150 184 210 240 241 262 268 271 285 289
    292 297 300 302 310 330 364 384 402 408 409 416 420 425 430 431 437 528 550

--- OUTPUT ---
Berechnung des Prefixsummenarrays in 0.000018s
Berechnung der optimalen Eisbudenposition zwischen zwei gegebenen Eisbuden in
    0.986344s
```

```
Absuchen aller Eisbudenkonstellationen in 3.111517s
Gesamte Berechnung in 4.099097s
Es gibt 16 stabile Konstellationen:
[114, 285, 416]
[114, 285, 417]
[114, 285, 418]
[114, 285, 419]
[114, 285, 420]
[114, 289, 416]
[114, 289, 417]
[114, 289, 418]
[114, 289, 419]
[114, 289, 420]
[115, 285, 420]
[115, 289, 420]
[116, 285, 420]
[116, 289, 420]
[117, 285, 420]
[117, 289, 420]
```

Nun eigene Beispiele. Zunächst Fälle mit sehr wenigen Häusern, für die natürlich trotzdem viele (z. T. alle möglichen) stabile Konstellationen existieren:

```
--- Befehl ---
$ java A3Main # Beispiel 1

--- INPUT  ---
5 0

--- OUTPUT ---
…
Es gibt 10 stabile Konstellationen:
[0, 1, 2]
[0, 1, 3]
[0, 1, 4]
[0, 2, 3]
[0, 2, 4]
[0, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]

--- Befehl ---
$ java A3Main # Beispiel 2

--- INPUT  ---
5 1
```

```
3

--- OUTPUT ---
…
Es gibt 6 stabile Konstellationen:
[0, 1, 3]
[0, 2, 3]
[0, 3, 4]
[1, 2, 3]
[1, 3, 4]
[2, 3, 4]

--- Befehl ---
$ java A3Main # Beispiel 3

--- INPUT  ---
5 2
1 4

--- OUTPUT ---
…
Es gibt 9 stabile Konstellationen:
[0, 1, 2]
[0, 1, 3]
[0, 1, 4]
[0, 2, 4]
[0, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]

--- Befehl ---
$ java A3Main # Beispiel 4

--- INPUT  ---
5 3
1 2 3

--- OUTPUT ---
…
Es gibt 7 stabile Konstellationen:
[0, 1, 2]
[0, 1, 3]
[0, 2, 3]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
```

```
[2, 3, 4]
```

Es zeigt sich, dass, wenn es nur wenige ($N \leq 3$) Häuser gibt, alle Konstellationen stabil sind, bei denen bei mindestens der Hälfte der Häuser eine Eisbude direkt vor der Haustür steht. So ist nämlich sichergestellt, dass die Bewohner für keine andere Konstellation mehr stimmen können – weil sich ihr Weg nicht mehr verkürzen kann.

Es folgt noch ein Beispiel, das zeigt, dass sich sogar mehrere Häuser an derselben Adresse befinden können. Das ist dadurch möglich, dass sämtliche Abfragen über die Häuserpositionen über das Prefixsummenarray durchgeführt werden, dem solche Sonderfälle keine Probleme bereiten.

```
--- Befehl ---
$ java A3Main # Beispiel 5

--- INPUT  ---
10 10
0 1 2 5 5 5 5 6 6 9

--- OUTPUT ---
…
Es gibt 30 stabile Konstellationen:
[0, 1, 5]
[0, 2, 5]
[0, 3, 5]
[0, 4, 5]
[0, 5, 6]
[0, 5, 7]
[0, 5, 8]
[0, 5, 9]
[1, 2, 5]
[1, 3, 5]
[1, 4, 5]
[1, 5, 6]
[1, 5, 7]
[1, 5, 8]
[1, 5, 9]
[2, 3, 5]
[2, 4, 5]
[2, 5, 6]
[2, 5, 7]
[2, 5, 8]
[2, 5, 9]
[3, 5, 6]
[3, 5, 9]
[4, 5, 6]
[4, 5, 9]
[5, 6, 7]
```

```
[5, 6, 8]
[5, 6, 9]
[5, 7, 9]
[5, 8, 9]
```

7 Quellcode

```
public static void main(String[] args) {
  // Umfang U des Sees, Anzahl N der Häuser
  int U, N;
  // Adressen der Häuser
  int[] adr;
  // kleinste Anzahl an Stimmen für Umzug
  int minN;
  List<int[]> stable = new ArrayList<int[]>();
  // Adressen sortieren
  Arrays.sort(adr);
  // Prefixsummenarray zur Abfrage: wie viele Häuser stehen im Intervall
  // [a,b[? als prefixS[b] - prefixS[a]
  int[] prefixS = new int[U + 1];
  // Zunächst: wie viele Häuser stehen an Position x?
  for (int x : adr) ++prefixS[x + 1];
  for (int i = 1; i <= U; ++i) prefixS[i] += prefixS[i - 1];
  // Gegeben Eisbuden a und x, die b Positionen weiter im Uhrzeigersinn von a
  // steht: Wo steht die (eine) "optimale" Eisbude zwischen a und b, sodass
  // sich der Weg zu möglichst vielen Häusern verkürzt? bzw. wie viele Häuser?
  int[][] optimalMiddleNum = new int[U][U];
  for (int a = 0; a < U; ++a) for (int b = 0; b < U; ++b) {
    int totalHouses = housesInInterval(prefixS, N, (a + 1) % U, (a + b) % U);
    if (totalHouses < 2) { // Trivialfälle
      optimalMiddleNum[a][b] = totalHouses;
      continue;
    }
    // initialisieren, u. a. für Trivialfälle b < 2
    int maxN = 0;
    // alle Positionen probieren
    for (int p = 1; p < b; ++p) {
      // erste Position im Bereich, in dem sich der Weg verkürzt
      int firstIn = ((p + 0 + 2) / 2 + a) % U;
      // erste Position, ab der sich der Weg nicht verkürzt
      int firstOut = ((p + b + 1) / 2 + a) % U;
      int numIn = housesInInterval(prefixS, N, firstIn, firstOut);
      if (numIn > maxN) {
        maxN = numIn;
      }
    }
    optimalMiddleNum[a][b] = maxN;
  }
  // alle Konstellationen durchgehen
  for (int a = 0; a < U; ++a)
    for (int b = a + 1; b < U; ++b) for (int c = b + 1; c < U; ++c) {
      int[] positions = new int[] { a, b, c };
      int numBest = -1;
      // alle Klassen für beste Möglichkeiten probieren:
      // 1. 2 + 1 Eisbuden: hier gibt es 6 Möglichkeiten
```

```
      for (int x = 0; x < 3; ++x) for (int z = 1; z < 3; ++z) {
        // x: Intervall, in dem 2 Eisbuden stehen
        // y: Intervall, in dem 1 Eisbude steht
        int y = (x + z) % 3;
        // Positionen der 2 Eisbuden: Anfang und Ende von x
        int p0 = (positions[x] + 1) % U;
        int p1 = (positions[(x + 1) % 3] + U - 1) % U;
        // Intervallstart und -länge von y
        int y0 = positions[y];
        int yl = (positions[(y + 1) % 3] + U - y0) % U;
        // Häuser, die in x und y stehen (= Häuser, bei denen sich der Weg
        // verkürzen wird)
        int inX = housesInInterval(prefixS, N, p0, (p1 + 1) % U);
        int inY = optimalMiddleNum[y0][yl];
        if (inX < 0 || inY < 0) continue; // die Intervallgröße war negativ
        // Anzahl Stimmen für diese Option
        int num = inX + inY;
        if (numBest < num) {
          numBest = num;
        }
      }
      // 2. 1 + 1 + 1 Eisbuden: nur eine Möglichkeit
      int num = 0;
      for (int i = 0; i < 3; ++i) {
        int i0 = positions[i];
        int i1 = (positions[(i + 1) % 3] + U - positions[i]) % U;
        num += optimalMiddleNum[i0][i1];
      }
      if (numBest < num) {
        numBest = num;
      }
      // generierte optimale Konstellation L' überprüfen:
      if (numBest < minN) {
        stable.add(positions);
      }
    }
  System.out
      .println("Es gibt " + stable.size() + " stabile Konstellationen:");
  if (stable.size() == 0) System.out.println("<empty list>");
  else for (int[] c : stable) System.out.println(Arrays.toString(c));
}

// Häuser im Intervall [start,end[
private static int housesInInterval(int[] prefixS, int N, int start,
    int end) {
  int num = prefixS[end] - prefixS[start];
  // Sonderfall beachten
  if (start > end) num += N;
  return num;
}
```