Aufgabe 3: Eisbudendilemma

Teilnahme-ID: 56860

Bearbeiter/-in dieser Aufgabe:
Christopher Besch

11. April 2021

Inhaltsverzeichnis

1 Ein Wort über die Graphiken

Alle in dieser Dokumentation verwendeten Darstellungen verwenden einheitliche Symbole:

- Der See ist als schwarzer Kreis dargestellt.
- Die Häuser sind verschieden gefärbte Rechtecke, deren Adresse außerhalb des Kreises stehen:
  - Rot: Das Haus stimmt gegen eine Verlegung der Eisdielen.
  - Grün: Es stimmt für eine Verlegung.
  - Andere Farben werden verwendet, um bestimmte Häuser hervorzuheben.
- Die Adressen sind aufsteigend im Uhrzeigersinn angeordnet. Adresse 0, die Dorfkirche, befindet sich oben.
- Blaue Kreuze stellen die Positionen des Test-Arrangements dar und
- blaue Kreise die des Check-Arrangements. In beiden Fällen stehen die Adressen innerhalb des Kreises.

2 Lösungsidee

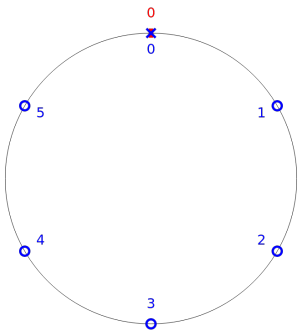
Das Ziel ist es, ein Arrangement bestehend aus drei Positionen für Eisdielen zu generieren, das in einer Abstimmung durch kein anderes Arrangement abgelöst werden kann. Diese Arrangements werden stabil genannt. Hierzu darf die Eisdielendistanz, die Strecke zwischen einem beliebigen Haus und der nächsten Eisdiele, 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 alle möglichen Arrangements auf Stabilität überprüft werden müssen. Diese werden Test-Arrangement genannt. Um die Stabilität zu bestimmen, muss das Test-Arrangement mit allen möglichen anderen Arrangements (Check-Arrangements genannt) verglichen werden. Wenn auch nur ein einziges Check-Arrangement gefunden wird, das mehr Ja- als Nein-Stimmen erhält, ist das getestete Test-Arrangement instabil. Es lässt sich leicht erkennen, dass dieser Algorithmus, der Durchgang aller möglichen Test-Arrangements, mit einer Laufzeit von $O(n^6)$ nicht verwendbar ist.

2.1 Sortierung der Arrangements

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 Eisdielendistanz aller Häuser. Nun stellt sich heraus, dass stabile Arrangements auch die niedrigsten Scores aller Arrangements aufweisen. Dies lässt sich damit erklären, dass je kleiner die Eisdielendistanz eines Hauses in einem Test-Arrangement ist, desto weniger Check-Arrangements existieren, die eine noch geringere Eisdielendistanz für das Haus generieren. Wenn die Eisdielendistanz beispielsweise 0 beträgt, existiert kein einziges Check-Arrangement, dem dieses Haus eine Ja-Stimme geben würde, da eine geringere Eisdielendistanz nicht möglich ist und ein Haus bei gleichbleibender Eisdielendistanz immer gegen einen Wechsel stimmt. Dies ist in Abbildung 1 gezeigt.

Abbildung 1: Das einzige Haus ist zufrieden mit der einzigen Eisdiele und lehnt jegliche Veränderung ab. Die Eisdielendistanz beträgt 0.



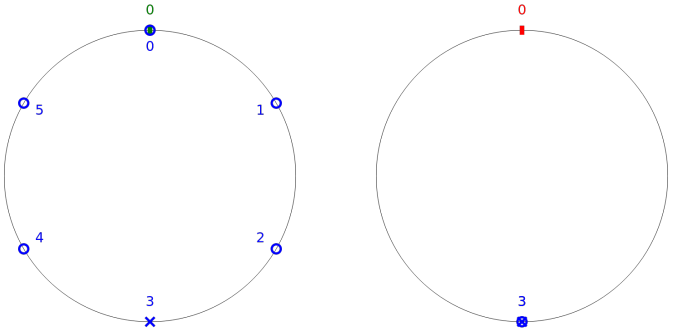
Wenn die Eisdielendistanz den maximalen Wert, dem halben Umfang des Sees, entspricht, stimmt es für alle Check-Arrangements (Abbildung 3a), abgesehen von denen, die die Eisdielendistanz nicht verändern (Abbildung 3b). Die durchschnittliche Eisdielendistanz lässt sich dementsprechend als „Zufriedenheitsgrad“ des Dorfes interpretieren. Je höher er ist, desto unwahrscheinlicher wird eine Verlegung durchgesetzt.

Allerdings muss dieser Wert nicht zwangsweise mit der Stabilität eines Arrangements übereinstimmen, was beispielsweise in Abbildung 4 gezeigt wird. Hieraus geht hervor, dass die durchschnittliche Eisdielendistanz nur eine Näherungslösung liefert. Trotzdem kann sie zur Generierung eines Satzes an Test-Arrangements, den x Arrangements mti der geringsten durchschnittlichen Eisdielendistanz, verwendet werden, die anschließend von dem bereits genannten Algorithmus auf Stabilität überprüft werden.

2.2 Bestimmung der Stabilität

Um die Stabilität eines Test-Arrangements zu berechnen, müssen alle möglichen Check-Arrangements durchgegangen werden. Es wird nur eine einziges Check-Arrangement gesucht, das das Test-Arrangement schlagen kann. Daher können zwei Optimierungen getroffen werden:

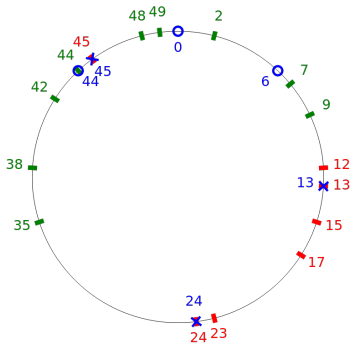
Abbildung 2: Unzufriedene Häuser



(a) Das Haus ist maximal unzufrieden, weshalb es für fast jede Verlegung stimmt. Jeder Kreis repräsentiert eine anderes Check-Arrangement, die alle von dem Haus angenommen werden.

(b) Dies ist der einzige Fall, in dem das Haus trotz seiner extremen Unzufriedenheit gegen eine Verlegung stimmt.

Abbildung 4: Trotz der für dieses Beispiel, *eisbuden3.txt*, minimalen durchschnittlichen Eisdielendistanz von $3,3125$ stimmen mehr Häuser für eine Verlegung.



1. Eisdielen sollten nicht übereinander liegen, da bei der Aufsplittung zweier aufeinanderliegender Eisdielen die Eisdielendistanz keines Häuses vergrößert wird.
2. Alle Dopplungen sind unnötig, da die Reihenfolge der Eisdielen für die Stimmen der Häuser irrelevant sind.

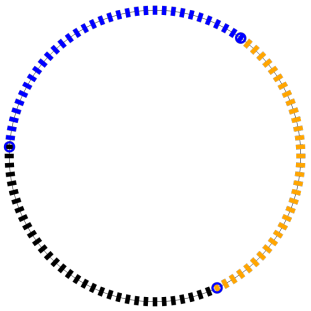
Deshalb darf die Bedingung gelten, dass die Adresse der zweiten Eisdiele größer als die der ersten und kleiner als die der dritten ist. Hieraus folgt, dass der See in drei Sektoren unterteilt ist (Abbildung 5).

2.2.1 Auszählung der Stimmen

Es zeigt sich, dass die Stimme der Häuser innerhalb eines Sektors ausschließlich durch die Größe und Position des Sektors und die in dem Sektor befindlichen Eisdielen des Test-Arrangements determiniert sind.

- Alle Test-Eisdielen außerhalb des Sektors sind von den Häusern immer weiter entfernt als die Ränder des Sektors, weshalb sich die ehemalige Eisdielendistanz ohne Test-Eisdielen innerhalb des Sektors nie verkürzen wird.
- Genauso ist für jedes Haus im Sektor eine der Sektorgrenzen immer die nächste Check-Eisdiele. Dementsprechend ist die Position der dritten Check-Eisdiele ebenfalls irrelevant. da immer nur die Eisdielendistanz betrachtet wird.

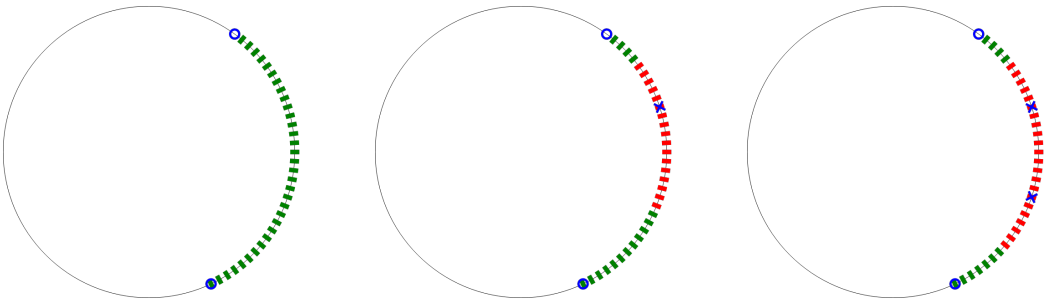
Abbildung 5: Einteilung in Sektoren



Aus diesen beiden Punkten geht hervor, dass ausschließlich die Ränder des Sektors und die sich im Sektor befindlichen Test-Eisdielen Einfluss auf die Stimme eines Hauses in dem Sektor hat.

Nun wird sich die Verteilung der Ja- und Nein-Stimmen betrachtet. In einem Sektor können sich keine (Abbildung 7a) nur eine (Abbildung 7b) oder mehr Eisdielen (Abbildung 7c) befinden.

Abbildung 6: Test-Eisdielen Anordnung im Sektor



(a) Keine Test-Eisdiele (b) Eine Test-Eisdiele (c) Zwei oder mehr Test-Eisdielen (nur die Äußersten dargestellt)

In dem letzten Fall von mehr als zwei Test-Eisdielen im Sektor, müssen nur die Äußersten betrachtet werden, da zwischen diesen beiden keine Check-Eisdiele stehen kann. Dies geht aus der Definition eines Sektors hervor. Somit stimmen immer alle Häuser zwischen diesen gegen eine Verlegung, unabhängig von der Menge and Test-Eisdielen.

Um die Anzahl an Nein- beziehungsweise Ja-stimmenden Häusern zu bestimmen wird müssen zwei verschiedene Methoden verwendet werden:

- Wenn sich keine Test-Eisdiele in dem Sektor befindet, stimmt kein Haus gegen eine Verlegung.
- In allen anderen Fällen befinden sich die einzigen Ja-stimmenden Häuser in den beiden Bereichen zwischen einem Rand un der nächsten Test-Eisdiele. Wenn diese Bereiche in der Mitte geteilt wird, befinden sich alle Ja-stimmenden Häuser in der Hälfte, die näher am Rand ist. Alle Nein-stimmenden Häuser liegen in der anderen Hälfte. Bei einer ungeraden Anzahl an Adressen in diesem Bereich wird der Nein-stimmenden Hälfte eine Adresse mehr zugeteilt, da bei gleichbleibender Eisdielendistanz das Haus Nein stimmen.

Auf diese Weise können effizient die Bereiche berechnet werden, in denen alle sich befindlichen Häuser Nein stimmen.

Es ergibt sich ein weiterer Vorteil: Um alle Position für die Check-Eisdielen durchzugehen, können zuerst zwei Positionen für die ersten beiden Eisdielen festgesetzt werden und daraufhin alle möglichen Positionen für die dritte Eisdiele verwendet werden, die die genannten Bedingungen erfüllt. Die verschiedenen Optionen für die ersten beiden Check-Eisdielen müssen in einem übergeordneten Schritt durchgegangen

werden. Somit ist die Größe und Position des ersten Sektors bei vielen Durchläufen gleich. Wenn bereits mindestenz die Hälfte der Häuser sich in diesem Sektor befinden und gegen eine Veränderung stimmen, ist das Check-Arrangement unfähig, das Test-Arrangement abzulösen. Somit kann in diesem Fall die Suche nach Positionen für die dritte Check-Eisdiele übersprungen werden und die Wahl der ersten beiden Check-Eisdielen sofort geändert werden.

3 Umsetzung

Die Lösungsidee wird in C++ implementiert. Der Übersichtlichkeit halber ist das Programm in drei Dateien unterteilt:

- **main.cpp** liest die Eingabe und gibt die Ergebnisse aus,
- **utils.h** enthält verwendete Structs und generell anwendbare Funktionen und Makros und
- **calculate_stable.h** enthält die eigentlichen Funktionen zur Bestimmung der stabilen Arrangements.

Die einzigen Eingaben, die das Programm benötigt ist der Pfad der Eingabedatei und die optionale Angabe der Menge an besten Arrangements, die auf Stabilität überprüft werden sollen. Je geringer dieser Wert ist, desto kürzer ist die Laufzeit. Allerdings ist ein zu niedriger Wert problemtisch, da wie bereits in Abbildung 4 gezeigt, die stabilen Arrangements nicht zwingendermaßen die geringste Eisdielendistanz aufweisen. Wenn kein Wert angegeben wird, werde die 600 besten Arrangements überprüft. Dieser Wert ist eine gute Wahl für die Beispieldateien.

3.1 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, das die Ausführung des Programms abbricht und eine möglichst informative Fehlermeldung zurückgibt.

Schlussendlich wird eine Instanz des **Lake** Structs erstellt. Dieses enthält:

- den Umfang des Sees,
- ein std::vector aller Häuser (repräsentiert durch **House** Instanzen, die jeweils die Adresse und Platz für die jeweilig notwendige Eisdielendistanz enthält),
- eine Karte des Sees, ein std::vector mit einem uint8_t pro Adresse. Wenn ein Haus bei dieser Adresse existiert, ist das entsprechende Element 1, sonst ist es 0. Diese Karte wird zur optimierten Zählung der Häuser in einem Bereich verwendet. Statt ein bool pro Element zu verwenden, wird ein uint8_t benutzt, da so die Speicherplatzspezifische Optimierung eines std::vector<bool> umgangen wird, was die Laufzeit verbessert.
- Und ein std::vector aller besten Arrangements, die jeweils als **Arrangement** Instanzen repräsentiert sind. Diese enthalten jeweils die Positionen der Eisdielen und den Score des Arrangements.

3.2 Scored Search

Der erste Schritt ist die Berechnung der Arrangements mit den besten Scores. Hierzu wird die Funktion **do_scored_search** verwendet. In ihr werden in drei verschachtelten Schleifen alle Test-Arrangements durchgegangen. Zu beachten ist, dass die erste Position angefangen von 0 hochzählt, während die zweite Position eine Adresse nach der ersten Position anfängt. Gleichen gilt für die dritte Position. Hierdurch werden die in Unterabschnitt 2.2 aufgezählten Optimierungen umgesetzt, die genauso für Test-Arrangements gelten können als auch für Check-Arrangements. Die Adressen der Eisdielen werden in einer neuen **Arrangement** Instanz gespeichert.

Für jedes Arrangement wird die Eisdielendistanz jedes Hauses mit **get_ice_cream_distance** berechnet und zusammenaddiert. In Unterabschnitt 2.1 wird zwar die durchschnittliche Eisdielendistanz berechnet, wofür die Summe durch die Menge an Häusern geteilt werden müsste. Allerdings werden im folgenden Code die Scores nur miteinander verglichen, weshalb die Berechnung des absoluten Wertes unnötig ist. Sie wird für die Optimierung weggelassen.

> **get_ice_cream_distance** berechnet mithilfe von **get_shortest_distance** die Entfernung von Einem Haus zu allen Eisdielen des Arrangements und wählt die kürzeste aus.
>
> **get_shortest_distance** berechnet die direkte und Entfernung zwischen zwei Adressen und die indirekte, wobei die direkte Entfernung von dem Umfang des Sees abgezogen wird. Schlussendlich gibt es die kürzere Entfernung zurück.

Dieser Score wird in der **Arrangement** Instanzen gespeichert. Um die Arrangements effizient zu sortieren, wird zuerst der std::vector in der **Lake** Instanz, der die besten Arrangements enthalten soll, mit so vielen Dummy Arrangements gefüllt, wie beste Arrangements gefordert sind. Wenn eine weitere **Arrangement** Instanz erstellt wurde, wird diese mit der Funktion **insert** an der richtigen Stelle in den std::vector eingefügt. Anschließend wird das letzte Element, das mit dem schlechtesten Score, entfernt. Diese vorgehensweise hat den Vorteil, dass der std::vector zu jedem Zeitpunkt sortiert ist.

Der Score der Dummy Arrangements entspricht dem Produkt aus Umfang und Menge an Häusern. Dieser Score ist schlechter als alle möglichen Scores, weshalb ein Dummy gegen jegliche Test-Arrangements ersetzt wird.

3.3 Überprüfung der Stabilität

Wenn die besten Test-Arrangements berechnet wurden, werden sie anschließend alle mit der Funktion **is_stable** auf Stabilität überprüft. Diese nimmt ein Test-Arrangement entgegen und geht alle möglichen Check-Arrangements durch, bis entweder ein Check-Arrangement gefunden wurde, dass gegen das Test-Arrangements in einer Abstimmung gewinnen und es ablösen würde, oder entdeckt wird, dass kein derartiges Check-Arrangement vorliegen kann.

Hierzu wird anfänglich für jedes Haus die Eisdielendistanz in dem Test-Arrangement berechnet und in den jeweiligen **House** Instanzen gespeichert. Genau wie in Unterabschnitt 3.2 werden die in Unterabschnitt 2.2 genannten Optimierungen implementiert. Der einzige Unterschied ist dass, wenn die ersten beiden Positionen gewählt wurden, die Suche nach einer Dritten nicht begonnen wird, wenn wie in Unterunterabschnitt 2.2.1 gezeigt bereits zu viele Stimmen gegen eine Verlegung in dem ersten Bereich gefunden werden.

3.3.1 Nein-Stimmenzählung in einem Sektor

Um die Menge an Nein-Stimmen im Sektor zwischen den ersten beiden Eisdielen zu berechnen, wird die Funktion **count_sector_nos** eingesetzt. Diese berechnet zuerst die nächste Test-Eisdiele innerhalb des Sektors von der linken Seite. Dafür wird **get_abs_distance** eingesetzt und mit der geringstern Entfernung weitergerechnet.

> **get_abs_distance** wird durch eine Instanze der Enum Class **Direction** auf Links- oder Rechtssuche eingestellt. Dementsprechend wird zurerst die direkte Distanz zwischen dem Startpunkt und dem Endpunkt berechnet. Wenn die direkte Distanz negativ ist, wird stattdessen die indirekte zurückgegeben. Hierdurch wird immer ein nicht-negativer Wert zurückgegeben.

Um zu detektieren, dass sich keine Test-Eisdiele in dem Sektor befindet, wird überprüft, ob die Breite des Sektors geringer als die Distanz zur nächsten Test-Eisdiele ist. Wenn dies der Fall ist, ist selbst die nächste Test-Eisdiele nicht im Sektor enthalten und Alle Häuser in dem Sektor stimmen für eine Verlegung, weshalb sofort 0 zurückgegeben wird.

Ist dies nicht der Fall, wird nach einer zweiten Test-Eisdiele in dem Sektor gesucht. Hierfür wird das selbe Verfahren angewendet, mit dem Unterschied, dass statt in die rechte Richtung, die Entfernung in die linke Richtung und statt der linken Grenze des Sektors, die rechte verwendet wird. So werden die beiden äußeren Test-Eisdielen berechnet, die die selbe sein können, wenn nur eine Test-Eisdiele im Sektor vorhanden ist.

Nun werden die in Unterunterabschnitt 2.2.1 vorgestellten Mitten berechnet und die Häuser in den beschriebenen Nein-stimmenden Bereichen gezählt.

> Diese Aufgabe übernimmt die Funktion **count_houses**, die die in der **Lake** Instanz gespeicherte Karte verwendet. Sie geht alle Adressen von dem Startwert bis zu der Adresse vor dem Endwert durch und addiert die in der Karte enthaltenen Elemente. In einer

Schleife wird hierdurch von einer Entfernung zum Anfangswert von 0 bis zu der Distanz zu dem Endwerte die Karte durchgegangen. Um ausgehend von der Entfernung zum Anfangswert die Adresse zu berechnen wird die Entfernung auf die Adresse des Anfangswertes addiert und der Rest der Division durch den Umfang des Sees verwendet.

So ergibt sich die Menge an Häusern in den festgelegten Bereichen, die zusammenaddiert zurückgegen werden.

3.3.2 Simulation der Abstimmung

Wenn im ersten Sektor nicht zu viele Nein-Stimmen enthalten sind, kann die selbe Funktion benutzt werden, um ebenfalls die Menge an Nein-Stimmen in den anderen beiden Sektoren zu berechnen. Allerdings stellt sich heraus, dass die Simulation der Abstimmung eine bessere Laufzeit aufweist. Hierfür wird die Funktion **is_better** verwendet. Sie geht alle Häuser durch und vergleicht die Eisdielendistanz mit dem Check-Arrangement eines Hauses mit dessen gespeicherte Eisdielendistanz. Diese wurde am Anfang von **is_stable** für das Test-Arrangement berechnet, damit sie nicht für jedes Check-Arrangement neu berechnet werden muss. Hierdurch wird die Stimme des Hauses bestimmt. Die Ja-Stimmen werden zusammenaddiert und mit der Menge an Nein-Stimmen, die der Differenz zwischen Anzahl an Häusern und Ja-Stimmen entspricht, verglichen.

Sind mehr Ja- als Nein-Stimmen vorhanden, wird die Suche nach einem Check-Arrangement abgebrochen und false ausgegeben, da die Instabilität bewiesen wurde. Nur in dem Fall, in dem die Suche alle möglichen Check-Arrangements durchlaufen hat und nicht abgebrochen wurde, ist das Test-Arrangement stabil. Alle stabilen Ergebnisse werden in der Konsole ausgegeben.

4 Zugeständnisse

Das Programm besteht aus zwei Teilen. Der eine, die Funktion **is_stable**, produziert bewiesenermaßen stabile Arrangements, ist allerdings zu langsam, um auf alle möglichen Arrangements angewendet zu werden. Auf der anderen Seite steht die Suche nach den besten Arrangements durch die durchschnittliche Eisdielendistanz, **do_scored_search**. Diese lässt sich nur empirisch beweisen und ist damit kein Garant für eine perfekte Lösung. Dafür ist sie äußerst schnell, weshalb es kein Problem ist, hiermit alle möglichen Arrangements durchzugehen.

Die Kombination dieser beiden Teile führt dazu, dass wenn das Programm ein stabiles Arrangement ausgibt, dieses definitiv stabil ist. Wenn das Programm allerdings kein stabiles Arrangement findet, besteht immer noch die Möglichkeit, dass es doch ein stabiles Arrangements gibt, dass nur eine hohe durchschnittliche Eisdielendistanz aufweist und daher noch nicht auf Stabilität geprüft wurde. Deshalb sollte die Menge an Arrangements, die überprüft werden soll, möglichst groß gewählt werden, um die Wahrscheinlichkeit, dass ein stabiles Arrangement nicht gefunden wird, minimiert wird. Dadurch ist dieses Programm nicht als perfekte Lösung des Problems zu betrachten.

Zudem ist an dem Programm auszusetzten, dass die Laufzeit stark von der Eingabedatei abhängt, während für *eisbuden1.txt* bereits nach etwa $200\mu s$ das erste stabile Ergebnis gefunden wird, benötigt es für *eisbuden7* $11s$. Die Laufzeit ist erheblich größer, wenn alle stabilen Arrangements gefordert sind ($1,5ms$ beziehungsweise $115s$).

5 Beispiele

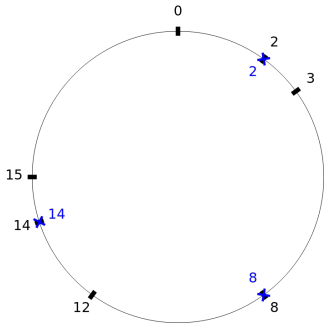
Nun wird das Programm mit allen Beispieldateien ausgeführt.

5.1 Eigene Beispiele

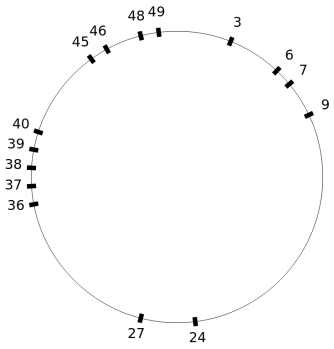
6 Quellcode

Dies sind die wichtigsten Funktionen:

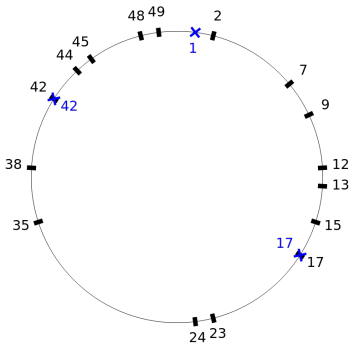
Abbildung 9: Beispiele vom BWINF



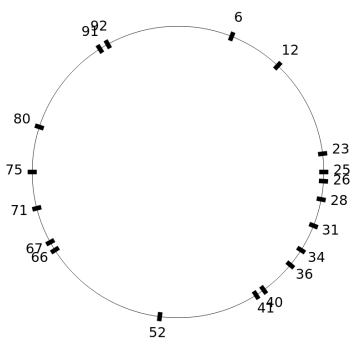
(a) eisbuden1.txt



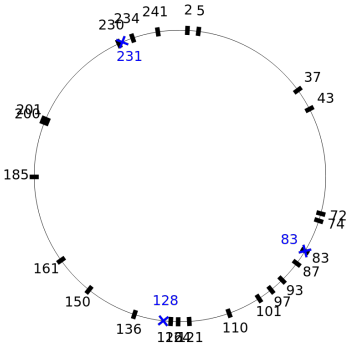
(b) eisbuden2.txt



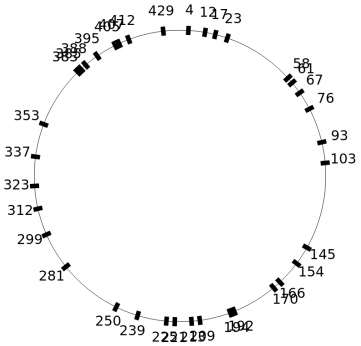
(c) eisbuden3.txt



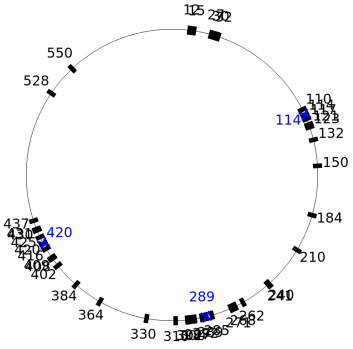
(d) eisbuden4.txt



(e) eisbuden5.txt



(f) eisbuden6.txt



(g) eisbuden7.txt