1. Runde

des

43. Bundeswettbewerbs Informatik

Aufgabe 3: Wandertag

Teamname: Crispy Clean

Team-ID: 42069

Bearbeitet

von

borekking
Teilnahme-ID: 42069

24. November 2024

Inhaltsverzeichnis

0. Einleitung

Dieses Dokument beinhaltet meine Dokumentation der dritten Aufgabe der ersten Runde des 43. Bundeswettbewerbs Informatik. Fuer diese Aufgabe wird zunaechst das gegebene Problem kurz mathematisch definiert. Darauf folgen drei Loesungsvorschlaege, welcher anschlieszend kurz auf ihre Zeit- und Platzkomplexitaet analysiert werden. Zuletzt werden Details zur Implementierung in C++ gegeben, woraufhin die Beispiele der BwInf Website und der Quellcode folgen.

1. Modellierung und Definition des Problems

Gegeben seien $n \in \mathbb{N}$ Tupel $(a_i,\ b_i)$ mit $a_i,\ b_i \in \mathbb{N}$ und $a_i \leq b_i$ für alle $1 \leq i \leq n$. Dabei ist a_i die minimale und b_i die maximale Streckenlaenge, die Mitglied i fordert, damit es teilnimmt. Genauer nimmt Mitglied i bei einer Strecke mit Streckenlaenge $l \in \mathbb{N}$ teil, wenn $a_i \leq l \leq b_i$ gilt. D.h., dass sowohl die minimale als auch die maximale Streckenlaenge inklusiv sind. Formal besteht das Problem der Aufgabenstellung nun darin, drei Streckenlaengen $l_1,\ l_2,\ l_3 \in \mathbb{N}$ zu finden, sodass die Anzahl der Mitglieder $1 \leq i \leq n$ die folgende Bedinung erfuellen maximal ist:

$$a_i \leq l_1 \leq b_i \text{ oder } a_i \leq l_2 \leq b_i \text{ oder } a_i \leq l_3 \leq b_i$$

Insbesondere wird ein Mitglied dabei nicht doppelt gezaehlt, wenn es bei mehr als einer Strecke teilnehmen wuerde, sondern genau dann wenn es bei mindestens einer Strecke teilnehmen wuerde. Auszerdem ist es insbesondere moeglich, dass ein Mitglied nur bei genau einer Streckenlaenge teilnimmt ($a_i = b_i$).

2. Loesungsvorschlag

Im Folgenden sollen drei Loesungsverfahren vorgestellt werden, wobei es sich beim ersten um ein Bruteforceverfahren handelt, waehrend die letzten beiden eine deutlich besser Laufzeit aufweisen. Dabei werden je die folgenden zwei Beobachtung gebraucht:

1. Es genuegt als Streckenlaengen alle der n minimalen Streckenlaengen zu betrachten, also die Menge $M \coloneqq \{a_i : 1 \leq i \leq n\}$. Denn alle moeglichen Streckenlaengen werden dadurch bereits abgedeckt: Jede Streckenlaenge $l \in \mathbb{N}$, die kleiner als $\min\{M\}$ ist, kann ohnehin ignoriert werden, da stets 0 Mitglieder die Bedingung $a_i \leq l \leq b_i$ erfuellen. Fuer jede Streckenlaenge $l \in \mathbb{N}$ mit $\min\{M\} \leq l$, wobei $l \notin M$, existiert stets eine maximale minimale Streckenlaenge $a \in M$ mit $\min(M) \leq a < l$. Dann gilt fuer jedes i mit $a_i \leq l \leq b_i$ auch $a_i \leq a \leq b_i$. Denn es gilt $a < l \leq b_i$ und a wurde maximal gewaehlt, sodass fuer keines der a_i gilt, dass $a < a_i < l$. Also gilt auch $a_i \leq a$, weil $a_i \leq l$ und $a_i = l$ ausgeschlossen ist (wegen $l \notin M$). Das bedeutet, dass bei Streckenlaenge a mindestens so viele Mitglieder teilnehmen wuerden, wie bei Streckenlaenge l. Also genuegt es in der Tat alle Streckenlaengen $a \in M$ zu betrachten.
2. Es genuegt Tripel $(l_1,\ l_2,\ l_3) \in M^3$ zu betrachten, fuer die $l_1 < l_2 < l_3$ gilt, denn die Bedingung $a_i \leq l_1 \leq b_i$ oder $a_i \leq l_2 \leq b_i$ oder $a_i \leq l_3 \leq b_i$ ist fuer alle Permutation von $(l_1,\ l_2,\ l_3)$ aequivalent. Auszerdem waere bei einer Gleichheit von zwei der drei Streckenlaengen die Anzahl mindestens genauso grosz, wuerde man ein beliebiges Element tauschen.

2.1. Loesungsvorschlag I - Bruteforceverfahren

Aufgrund der eben gemachten Beobachtungen, laesst sich das gegebene Problem sofort mit einem Bruteforceverfahren loesen. Dabei werden alle $\binom{|M|}{3}$ der paarweise verschiedenen Trippel $(c_1,\ c_2,\ c_3) \in M^3$ mit $c_1 < c_2 < c_3$ ausprobiert. Fuer jedes Tripel kann nun die Anzahl der Mitglieder berechnet werden, die die oben genannte Bedinung erfuellen, indem alle Mitglieder einmal geprueft werden. Dabei soll stets das Tripel mit der groeszten Anzahl an Mitgliedern, die teilnehmen wuerden und diese Anzahl gespeichert und ggf. neugesetzt werden (s. Algorithmus 1).

2.2. Loesungsvorschlag II

Nun wird $l_2 \in M$ fixiert und das Trippel $(l_1,\ l_2,\ l_3) \in M^3$ mit $l_1 < l_2 < l_3$ gesucht, das die Anzahl an Mitgliedern maximiert, die teilnehmen wuerden (also die Bedinung $a_i \leq l_1 \leq b_i$ oder $a_i \leq l_2 \leq b_i$ oder $a_i \leq l_3 \leq b_i$ erfuellen). Kann man dies fuer alle $l_2 \in M$ finden, so laesst sich das Problem loesen, indem die $|M|$ verschiedenen $l_2 \in M$ ausprobiert und die maximale Anzahl an Mitgliedern und die dazugehoerigen Streckenlaengen je updatet (s. Algorithmus 2).

Algorithmus 1 : Bruteforceverfahren

Input : Anzahl der Mitglieder $n \in \mathbb{N}$ und die Paare $(a_i,\ b_i)$
Output : Maximale Anzahl an Mitgliedern und Streckenlaengen $l_1,\ l_2,\ l_3$

```
1  M ← {a_i : 1 ≤ i ≤ n}
2  (l_1, l_2, l_3) ← (0, 0, 0)
3  b ← 0 // Maximalanzahl bisher (best)
4  foreach (c_1, c_2, c_3) in M^3 do
5      c ← 0 // Anzahl der Mitglieder (counter)
6      for i = 1 to n do
7          if a_i ≤ c_1 ≤ b_i oder a_i ≤ c_2 ≤ b_i oder a_i ≤ c_3 ≤ b_i then
8              c ← c + 1
9          end if
10     end for
11     if c < b then
12         (l_1, l_2, l_3) ← (c_1, c_2, c_3)
13         b ← c
14     end if
15 end foreach
16 return (b, l_1, l_2, l_3)
```

Nun sei also $l_2 \in M$ fixiert. Wir suchen $(l_1,\ l_2,\ l_3)$ mit $l_1 < l_2 < l_3$, sodass die Anzahl der Mitglieder, die teilnehmen koennen, maximal ist. Die Mitglieder $1 \leq i \leq n$ lassen sich nun durch drei verschiedene Faelle unterteilen:

1. Es gilt $a_i \leq b_i < l_2$. D.h., das Intervall $[a_i,\ b_i]$ befindet sich vollstaendig links von l_2. Dann kann Mitglied i potenziell nur bei Streckenlaenge l_1 teilnehmen, weil $a_i \leq b_i < l_2 < l_3$ gilt.
2. Es gilt $a_i \leq l_2 \leq b_i$. D.h., l_2 befindet sich im Intervall $[a_i,\ b_i]$. Dann nimmt Mitglied i bereits wegen Streckenlaenge l_2 teil.
3. Es gilt $l_2 < a_i \leq b_i$. D.h., das Intervall $[a_i,\ b_i]$ befindet sich vollstaendig rechts von l_2. Dann kann Mitglied i potenziell nur bei Streckenlaenge l_3 teilnehmen, weil $l_1 < l_2 < a_i \leq b_i$ gilt.

Somit genuegt es fuer l_1 die Mitglieder mit $a_i \leq b_i < l_2$ zu betrachten und fuer l_3 die Mitglieder mit $l_2 < a_i \leq b_i$ (s. Abbildung 1 - Mitglieder fuer l_1 in rot und Mitglieder fuer l_3 in blau). Findet man nun l_2 und l_3, sodass die

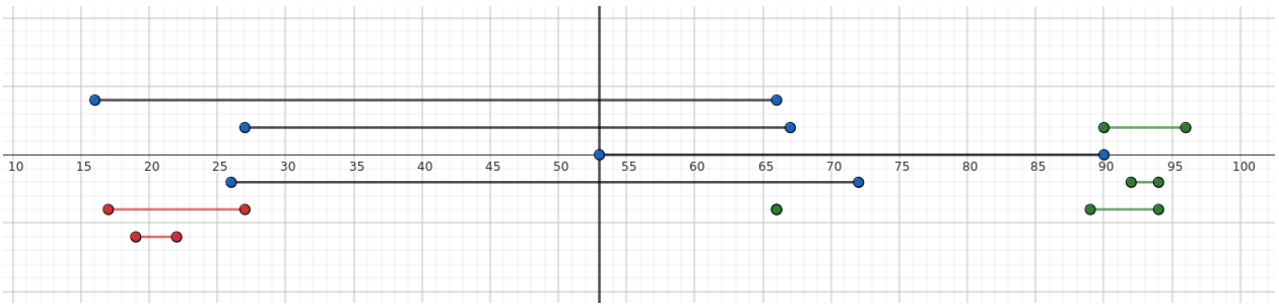

Abbildung 1: Oben beschriebene Aufteilung fuer $l_2 = 53$ in Beispiel *wandern3.txt*

Anzahl der Mitglieder die innerhalb der je betrachteten Menge teilnehmen koennen maximal ist, so hat man bereits, dass die Anzahl der Mitglieder die bei den Streckenlaengen $(l_1,\ l_2,\ l_3)$ teilnehhmen konnen, maximal ist (fuer gegebenes l_2) (s. Algorithmus 2, Funktion *foo*).

Also genuegt es, fuer eine gegebene Teilmenge an Mitgliedern $N \subset \{1,\ 2,\ \ldots\ ,\ n\}$ eine optimale Streckenlaenge finden zu koennen. Dies ist wie folgt in linearer Zeit moeglich: Iteriere durch alle a_i und b_i fuer $i \in N$ in aufsteigender Reihenfolge. Bei Gleichheit sollen erst die minimalen Streckenlaengen und dann die maximalen Streckenlaengen betrachtet werden. Wir wollen nun stets die aktuelle Anzahl der Mitglieder, die teilnehmen koennen, als Variable m speichern, die mit 0 initialisiert wird. Ist der aktuelle Wert eine minimale Streckenlaenge, dann erhoehe m um 1 und pruefe, ob das Maximum geandert werden muss. Handelt es sich um eine maximale Streckenlaenge, so verringere m um 1. Um zwischen minimalen und maximalen Streckenlaengen unterscheiden

zu koennen, kann man eine Liste an Tupeln anlegen, wobei der erste Wert a_i, bzw. b_i entspricht und der zweite Wert 0 ist, wenn es sich um eine minimale Streckenlaenge handelt und 1, wenn es sich um eine maximale Streckenlaenge handelt (s. Algorithmus 2, Funktion *bar*). Dies hat den Vorteil, dass bei Sortierung dieser Liste in aufsteigende Reihenfolge, automatisch oben gennante Reihefolge eingehalten wird, weil $(a,\ 0) < (a,\ 1)$.

2.3. Loesungsvorschlag III

Das vorherige Loesungsverfahren laesst sich verbessern, indem die Aufteilung der Mitglieder mit Intervall links bzw. rechts des aktuellen Punktes nicht implizit aufgeteilt werden, sondern nur durch geschicktes ignorieren aufgeteilt werden. Konkret bedeutet dies, dass bei einem fixierten l_2 wieder die Anzahl der Mitglieder die bei l_2 teilnehmen koennen, und anschlieszend die bestmoegliche Anzahl an Mitglieder berechnet, die links, bzw. rechts von l_2 erreicht werden kann. Dazu wird ein Verfahren benutzt, das komplett analog zum obigen ist. Aber dieses mal wird nicht jedes Mal eine neue Menge an Mitglieder erstellt, sondern es werden schlicht alle a_i und b_i durchgegangen, wobei diejenigen uebersprungen werden, deren Intervalle nicht vollstaendig links, bzw. vollstaendig rechts von l_2 liegen. Dies hat den Vorteil, dass die Liste der Mitglieder, die aktuell betrachtet werden nicht jedes Mal neu berechnet und sortiert werden muss. Es genuegt die Liste einmal mit allen Tripeln $(a_i,\ 0,\ i)$ und allen $(b_i,\ 1,\ i)$ zu fuellen und anschlieszend zu sortieren. Analog zu oben werden dadurch bei Gleichheit immer erst die a_i und danach die b_i betrachtet. (s. Algorithmus 3).

2.4. Analyse und Vergleich der Loesungsverfahren

Die Platzkomplexitaet der drei Loesungen ist jeweils linear in n. Dies ergibt sich sofort daraus, dass die benutzten Mengen, bzw. Listen jeweils maximal $2n$ Elemente haben. Insbesondere werden hier alle ganzzahligen Werte einem konstanten Platzverbrauch zu geordnet.
Die Laufzeitkomplexitaet der ersten Loesung ist $\mathcal{O}\left(n^4\right)$. Diese ergibts sich wie folgt: Die Schleife von Zeile 4 bis Zeile 15 wird genau $\binom{|M|}{3} = \frac{(|M|-2)\cdot(|M|-1)\cdot|M|}{6} = \mathcal{O}\left(|M|^3\right) = \mathcal{O}\left(n^3\right)$ Male ausgefuehrt (bemerke $|M| \leq n$). Weiter wird innerhalb dieser Schleife eine weitere Schleife ausgefuehrt, die genau n Male laeuft. Alle anderen Anweisungen sind konstant. Somit ergibt sich die Laufzeitkomplexitaet $\mathcal{O}\left(n^4\right)$.
Die Laufzeitkomplexitaet der zweiten Loesung ist $\mathcal{O}\left(n^2 \cdot \log(n)\right)$. Dafuer betrachte man zunaechst die Methode *bar*. Diese Funktion erstellt zunaechst eine Liste v, die maximal $2n$ Elemente hat, in linearer Zeit und sortiert diese, was in $\mathcal{O}\left(n \log(n)\right)$ moeglich ist. Die Schleife von Zeile 27 bis Zeile 38 laeuft dementspraechend maximale $2n$ Male, wobei jeder Durchlauf eine konstante Zeit braucht. Somit ergibt sich als Laufzeitkomplexitaet von *bar* $\mathcal{O}\left(n\right) + \mathcal{O}\left(n \log(n)\right) + \mathcal{O}\left(n\right) = \mathcal{O}\left(n \log(n)\right)$. Nun betrachte man *foo*. Das Erstellen der drei Mengen ist sofort in linearer Zeit moeglich. Die Aufruefe der Methode *bar* brauchen je $\mathcal{O}\left(n \log(n)\right)$ viel Zeit Somit ergibt sich fuer die Funktion die Laufzeitkomplexitaet $\mathcal{O}\left(n\right) + \mathcal{O}\left(n \log(n)\right) = \mathcal{O}\left(n \log(n)\right)$. Nun kann man den Hauptteil des Algorithmus betrachten. Dieser besteht bis auf die Schleife von Zeile 4 bis Zeile 10 aus Operationen mit konstanter Laufzeit. Jeder Durchlauf der Schleife benoetigt $\mathcal{O}\left(n \log(n)\right)$ viel Zeit, da die Methode *foo* aufgerufen wird und es sich sonst um konstante Operationen handel. Daraus ergibt sich insgesamt die Laufzeitkomplexitaet $\mathcal{O}\left(n^2 \log(n)\right)$.
Die Laufzeitkomplexitaet der dritten Loesung ist $\mathcal{O}\left(n^2\right)$. Die Liste v, die zunaechst erstellt wird, hat genau $2n$ Elemente, und wird anschlieszend sortiert. Dies ist in $\mathcal{O}\left(n \log(n)\right)$ moeglich. Die darauffolgende Schleife wird $|M| \leq n$ Male ausgefuehrt. Innerhalb der dieser Schleife werden neben einigen Operationen mit konstanter Laufzeit, zwei Schleifen ausgefuehrt, die jeweils genau $2n$ Male laufen, wobei jeder Durchlauf eine konstante Zeit benoetigt. Somit benoetigt jeder Durchlauf der aeuszeren Schleife $\mathcal{O}\left(n\right)$ viel Zeit. Insgesamt ergibt sich damit eine Laufzeitkomplexitaet von $\mathcal{O}\left(n^2\right)$
Somit ist ersichtlich, dass die zweite und dritte Loesung der Bruteforceloesung gegenüber eindeutig zu bevorzugen sind. Weiter zeigt sich, dass die Verbesserung vom zweiten Verfahren im dritten eine kleine aber sichbare Verbesserung der Laufzeitkomplexitaet mitsichgebracht hat.

3. Implementierung

Die vorgestellten Loesungsverfahren wurden einzeln in C++ implementiert. Dabei werden am anfang je aus dem Standart Input die Namen der Ein- und Ausgabedatei abgefragt und von dortan diese Datein fuer alle Ein- und Ausgaben verwendet. Auszerdem hat jede der drei Implementierungen eine Methode, die bei gegebenem Ergebnis in Form der drei Laengen und der maximalen Anzahl an Mitgliedern, alle gewuenschten Informationen in die angegebene Datei schreibt. Dabei handelt es sich um die Anzahl der Mitglieder, die teilnehmen koennen, die genutzten Laengen, die Mitglieder, die teilnehmen, die Mitglieder, die nicht teilnehmen und fuer jede Streckenlaenge, die Mitglieder, die potenziell teilnehhmen wuerden. Dabei wird ein Mitglied durch den Index (0-indexiert) angegeben, an dem er in der Eingabedatei vorkommt.

Algorithmus 2 : Loesungsvorschlag II

Input : Anzahl der Mitglieder $n \in \mathbb{N}$ und die Paare $(a_i,\ b_i)$

Output : Maximale Anzahl an Mitgliedern b und Streckenlaengen $l_1,\ l_2,\ l_3$

```
1  M ← {a_i : 1 ≤ i ≤ n}
2  (l_1, l_2, l_3) ← (0, 0, 0)
3  b ← 0 // Maximalanzahl bisher (best)
4  foreach c_2 in M do
5      (b_0, c_1, c_3) ← foo(l_2)
6      if b_0 > b then
7          (l_1, l_2, l_3) ← (c_1, c_2, c_3)
8          b ← b_0
9      end if
10 end foreach
11 return (b, l_1, l_2, l_3)
12
13 Function foo(l_2):
14     N_1 ← {i : 1 ≤ i ≤ n und b_i < l_2}
15     N_2 ← {i : 1 ≤ i ≤ n und a_i ≤ l_2 ≤ b_i}
16     N_3 ← {i : 1 ≤ i ≤ n und l_2 < a_i}
17     (best_1, l_1) ← bar(N_1)
18     (best_3, l_3) ← bar(N_3)
19     return (best_1 + |N_2| + best_3, l_1, l_3)
20
21 Function bar(N):
22     v ← {(a_i, 0) : i ∈ N} ∪ {(b_i, 1) : i ∈ N}
23     sort(v)
24     b ← 0 // Maximale Anzahl an Mitgliedern bisher (best)
25     l ← −1 // Laenge, die Anzahl b ermoeglicht (length)
26     c ← 0 // Aktuelle Anzahl an Mitgliedern (current)
27     foreach (x, y) in v do
28         if y = 0 then
29             c ← c + 1
30             if c > b then
31                 b ← c
32                 l ← x
33             end if
34         end if
35         else if y = 1 then
36             c ← c − 1
37         end if
38     end foreach
39     return (b, l)
```

Algorithmus 3 : Loesungsvorschlag III

Input : Anzahl der Mitglieder $n \in \mathbb{N}$ und die Paare $(a_i,\ b_i)$

Output : Maximale Anzahl an Mitgliedern b und Streckenlaengen $l_1,\ l_2,\ l_3$

```
1  M ← {a_i : 1 ≤ i ≤ n}
2  v ← {(a_i, 0, i) : 1 ≤ i ≤} ∪ {(b_i, 1, i) : 1 ≤ i ≤}
3  sort(v)
4  (l_1, l_2, l_3) ← (0, 0, 0)
5  b ← 0 // Maximalanzahl bisher (best)
6  foreach c_2 in M do
7      // Mitglieder, die bereits abgedeckt sind
8      best_2 ← |{i : 1 ≤ i ≤ n und a_i ≤ c_2 ≤ b_i}|
9
10     // Maximalanzahl links von c_2
11     best_1 ← 0 // Beste Anzahl links
12     length_1 ← −1 // Laenge, die die beste Anzahl links ermoeglicht
13     c_1 ← 0 // Aktuelle Anzahl links
14     for i = 0 to 2n − 1 do
15         // Ignoriere, wenn [a_i, b_i] nicht links von c_2 liegt
16         if b_{v[i][2]} ≥ c_2 then
17             continue
18         end if
19         // Erhoehe bzw. Verringe die aktuelle Anzahl um 1
20         if v[i][1] = 0 then
21             c_1 ← c_1 + 1
22             if c_1 > best_1 then
23                 best_1 ← c_1
24                 length_1 ← a_{v[i][2]}
25             end if
26         end if
27         else if v[i][1] = 1 then
28             c_1 ← c_1 − 1
29         end if
30     end for
31
32     // Maximalanzahl rechts von c_2
33     best_3 ← 0 // Beste Anzahl rechts
34     length_3 ← −1 // Laenge, die die beste Anzahl rechts ermoeglicht
35     c_3 ← 0 // Aktuelle Anzahl rechts
36     for i = 0 to 2n − 1 do
37         [...] // Analog zum Teil links von c_2
38     end for
39
40     // Update wenn noetig
41     if best_1 + best_2 + best_3 > b then
42         (l_1, l_2, l_3) ← (length_1, c_2, length_3)
43         b ← best_1 + best_2 + best_3
44     end if
45 end foreach
46 return (b, l_1, l_2, l_3)
```

Weiter wird jeweils die Anzahl der Mitglieder als Integer n und die Paare $(a_i,\ b_i)$ in einem Vektor des Typs *pair<int, int>* gespeichert, waehrend die drei Streckenlaengen je in einem *tuple<int, int, int>* gespeichert werden. Dabei sei bemerkt, dass es sich bei den Streckenlaengen somit stets um ganzzahlige Werte handelt, da dies bei allen Eingabedateien ebenfalls der Fall ist. Weiter werden die in Loesungsverfahren II und III beschriebenen Listen, die Startpunkte, bzw. Endpunkte zusammen mit 0, oder 1, und ggf. dem dazugehoerigen Index als *vector<pair<int, bool»*, bzw. *vector<tuple<int, bool, int»* gespeichert.

4. Beispiele

Nun wird auf die Beispiele der BwInf Website eingegangen. Dabei haben alle drei Algorithmen dieselben Ausgaben ergeben, wobei das Bruteforceverfahren eine erheblich hoehere Laufzeit aufweiszt, als die anderen beiden Verfahren.

4.1. Beispiel 1 - „wandern1.txt“

1. Maximale Anzahl an Mitgliedern: 6 von 7
2. Streckenlaengen: 22, 51, 64
3. Mitglieder, die teilnehmen: 0, 1, 3, 4, 5, 6
4. Mitglieder, die nicht teilnehmen: 2
5. 2 Mitglieder, die bei Streckenlaenge 22 teilnehmen koennen: 0, 1
6. 2 Mitglieder, die bei Streckenlaenge 51 teilnehmen koennen: 3, 4
7. 2 Mitglieder, die bei Streckenlaenge 64 teilnehmen koennen: 5, 6

4.2. Beispiel 2 - „wandern2.txt’

1. Maximale Anzahl an Mitgliedern: 6 von 6
2. Streckenlaengen: 10, 60, 90
3. Mitglieder, die teilnehmen: 0, 1, 2, 3, 4, 5
4. Mitglieder, die nicht teilnehmen:
5. 2 Mitglieder, die bei Streckenlaenge 10 teilnehmen koennen: 4, 5
6. 3 Mitglieder, die bei Streckenlaenge 60 teilnehmen koennen: 0, 2, 3
7. 1 Mitglied, das bei Streckenlaenge 90 teilnehmen kann: 1

4.3. Beispiel 3 - „wandern3.txt“

1. Maximale Anzahl an Mitgliedern: 10 von 10
2. Streckenlaengen: 19, 66, 92
3. Mitglieder, die teilnehmen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
4. Mitglieder, die nicht teilnehmen:
5. 3 Mitglieder, die bei Streckenlaenge 19 teilnehmen koennen: 6, 8, 9
6. 4 Mitglieder, die bei Streckenlaenge 66 teilnehmen koennen: 0, 1, 2, 5
7. 3 Mitglieder, die bei Streckenlaenge 92 teilnehmen koennen: 3, 4, 7

4.4. Beispiel 4 - „wandern4.txt“

1. Maximale Anzahl an Mitgliedern: 79 von 100
2. Streckenlaengen: 524, 811, 922
3. Mitglieder, die teilnehmen: 0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 42, 43, 46, 47, 48, 50, 51, 53, 54, 55, 56, 57, 58, 59, 60, 62, 64, 65, 66, 67, 69, 71, 72, 73, 74, 75, 78, 80, 81, 82, 83, 86, 87, 88, 89, 90, 91, 92, 94, 96, 97, 98, 99
4. Mitglieder, die nicht teilnehmen: 2, 15, 17, 28, 33, 41, 44, 45, 49, 52, 61, 63, 68, 70, 76, 77, 79, 84, 85, 93, 95
5. 38 Mitglieder, die bei Streckenlaenge 524 teilnehmen koennen: 3, 6, 7, 9, 10, 11, 12, 19, 20, 23, 24, 25, 34, 37, 38, 39, 40, 42, 47, 54, 56, 57, 58, 60, 64, 65, 67, 71, 78, 80, 81, 82, 83, 90, 91, 97, 98, 99
6. 37 Mitglieder, die bei Streckenlaenge 811 teilnehmen koennen: 0, 1, 3, 4, 5, 8, 13, 14, 16, 18, 19, 20, 21, 25, 26, 27, 29, 31, 32, 34, 43, 48, 51, 53, 55, 59, 62, 64, 71, 73, 74, 75, 82, 87, 88, 96, 97
7. 20 Mitglieder, die bei Streckenlaenge 922 teilnehmen koennen: 3, 18, 22, 30, 34, 35, 36, 46, 48, 50, 51, 62, 66, 69, 72, 75, 86, 89, 92, 94

4.5. Beispiel 5 - „wandern5.txt“

1. Maximale Anzahl an Mitgliedern: 153 von 200
2. Streckenlaengen: 36696, 60828, 88584
3. Mitglieder, die teilnehmen: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 20, 21, 22, 24, 26, 27, 29, 30, 31, 32, 33, 34, 37, 38, 39, 40, 41, 42, 44, 45, 47, 48, 52, 53, 54, 56, 57, 59, 62, 63, 64, 65, 66, 67, 71, 72, 73, 76, 77, 78, 80, 81, 84, 85, 86, 87, 88, 89, 90, 91, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 104, 105, 106, 107, 108, 110, 111, 113, 114, 115, 116, 117, 118, 119, 120, 121, 123, 124, 126, 127, 128, 129, 130, 131, 132, 133, 135, 136, 137, 138, 139, 140, 142, 143, 144, 145, 147, 148, 149, 150, 151, 153, 154, 157, 158, 161, 162, 163, 165, 166, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 180, 181, 182, 184, 186, 187, 189, 190, 191, 192, 193, 194, 195, 196, 197, 199
4. Mitglieder, die nicht teilnehmen: 0, 12, 18, 19, 23, 25, 28, 35, 36, 43, 46, 49, 50, 51, 55, 58, 60, 61, 68, 69, 70, 74, 75, 79, 82, 83, 92, 99, 109, 112, 122, 125, 134, 141, 146, 152, 155, 156, 159, 160, 164, 167, 179, 183, 185, 188, 198
5. 86 Mitglieder, die bei Streckenlaenge 36696 teilnehmen koennen: 1, 2, 5, 7, 11, 14, 17, 20, 21, 26, 27, 29, 32, 37, 38, 39, 40, 42, 44, 45, 47, 54, 63, 64, 66, 67, 71, 72, 73, 78, 86, 88, 89, 91, 93, 94, 95, 96, 97, 98, 102, 103, 104, 106, 107, 108, 110, 111, 113, 114, 115, 117, 118, 121, 123, 124, 126, 127, 128, 133, 135, 136, 144, 145, 151, 161, 163, 168, 170, 174, 175, 176, 177, 178, 180, 181, 182, 186, 189, 190, 191, 192, 194, 195, 197, 199
6. 86 Mitglieder, die bei Streckenlaenge 60828 teilnehmen koennen: 1, 4, 5, 10, 11, 13, 20, 21, 24, 26, 29, 31, 32, 33, 34, 37, 39, 40, 41, 42, 45, 48, 52, 54, 57, 62, 63, 64, 65, 72, 81, 84, 85, 87, 91, 94, 96, 101, 102, 104, 105, 107, 108, 111, 113, 114, 117, 118, 124, 128, 129, 130, 131, 132, 133, 135, 137, 138, 143, 144, 145, 147, 149, 151, 154, 157, 158, 161, 162, 166, 168, 170, 171, 173, 175, 177, 178, 182, 184, 186, 191, 192, 193, 194, 195, 196
7. 52 Mitglieder, die bei Streckenlaenge 88584 teilnehmen koennen: 3, 5, 6, 8, 9, 10, 13, 15, 16, 20, 22, 26, 29, 30, 33, 34, 39, 53, 56, 59, 76, 77, 80, 84, 90, 94, 100, 105, 114, 116, 119, 120, 130, 131, 139, 140, 142, 144, 147, 148, 150, 151, 153, 157, 165, 169, 170, 172, 175, 184, 187, 191

4.6. Beispiel 6 - „wandern6.txt“

1. Maximale Anzahl an Mitgliedern: 330 von 500
2. Streckenlaengen: 42834, 74810, 92920

3. Mitglieder, die teilnehmen: 0, 2, 4, 6, 7, 9, 11, 12, 18, 19, 20, 25, 27, 29, 30, 31, 32, 33, 34, 35, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 58, 59, 60, 63, 64, 67, 69, 70, 71, 74, 75, 79, 81, 83, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 97, 98, 99, 100, 102, 103, 106, 107, 108, 110, 111, 112, 113, 114, 115, 117, 120, 121, 123, 125, 126, 128, 129, 131, 132, 135, 136, 137, 138, 140, 141, 142, 143, 144, 145, 146, 148, 149, 150, 151, 152, 154, 157, 163, 164, 165, 166, 168, 171, 174, 175, 176, 177, 178, 179, 180, 181, 182, 184, 185, 186, 188, 189, 190, 193, 194, 195, 196, 197, 198, 200, 202, 203, 205, 206, 210, 211, 213, 214, 215, 216, 221, 223, 224, 225, 226, 228, 229, 230, 231, 232, 234, 235, 236, 237, 238, 241, 243, 244, 247, 248, 249, 251, 253, 254, 255, 257, 258, 259, 260, 261, 262, 264, 266, 268, 272, 275, 276, 277, 278, 279, 280, 283, 284, 287, 288, 289, 290, 291, 292, 293, 295, 297, 298, 299, 300, 301, 305, 306, 307, 309, 311, 312, 314, 315, 316, 318, 319, 320, 322, 324, 325, 326, 328, 329, 330, 331, 334, 335, 336, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 356, 358, 360, 362, 363, 366, 367, 368, 369, 370, 371, 373, 374, 377, 379, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 393, 394, 395, 396, 397, 398, 399, 403, 404, 405, 409, 410, 412, 414, 415, 416, 417, 418, 419, 420, 425, 426, 428, 429, 430, 431, 432, 433, 435, 436, 438, 442, 443, 444, 447, 448, 449, 450, 451, 453, 454, 456, 457, 458, 460, 462, 463, 465, 467, 468, 469, 470, 473, 474, 476, 477, 478, 479, 480, 481, 482, 484, 486, 487, 490, 494, 496, 497

4. Mitglieder, die nicht teilnehmen: 1, 3, 5, 8, 10, 13, 14, 15, 16, 17, 21, 22, 23, 24, 26, 28, 36, 38, 48, 57, 61, 62, 65, 66, 68, 72, 73, 76, 77, 78, 80, 82, 89, 95, 96, 101, 104, 105, 109, 116, 118, 119, 122, 124, 127, 130, 133, 134, 139, 147, 153, 155, 156, 158, 159, 160, 161, 162, 167, 169, 170, 172, 173, 183, 187, 191, 192, 199, 201, 204, 207, 208, 209, 212, 217, 218, 219, 220, 222, 227, 233, 239, 240, 242, 245, 246, 250, 252, 256, 263, 265, 267, 269, 270, 271, 273, 274, 281, 282, 285, 286, 294, 296, 302, 303, 304, 308, 310, 313, 317, 321, 323, 327, 332, 333, 337, 338, 339, 340, 355, 357, 359, 361, 364, 365, 372, 375, 376, 378, 380, 392, 400, 401, 402, 406, 407, 408, 411, 413, 421, 422, 423, 424, 427, 434, 437, 439, 440, 441, 445, 446, 452, 455, 459, 461, 464, 466, 471, 472, 475, 483, 485, 488, 489, 491, 492, 493, 495, 498, 499

5. 168 Mitglieder, die bei Streckenlaenge 42834 teilnehmen koennen: 0, 9, 18, 25, 30, 32, 33, 40, 41, 43, 44, 45, 46, 50, 51, 52, 53, 54, 56, 58, 60, 64, 69, 70, 71, 74, 75, 81, 85, 86, 87, 88, 94, 99, 100, 102, 110, 112, 113, 114, 117, 120, 121, 125, 128, 129, 132, 136, 137, 144, 145, 148, 149, 150, 152, 157, 163, 165, 166, 174, 175, 179, 181, 184, 185, 189, 193, 194, 195, 196, 197, 202, 203, 205, 210, 211, 214, 215, 221, 223, 228, 235, 236, 237, 241, 244, 248, 249, 251, 254, 255, 257, 260, 261, 262, 266, 276, 277, 278, 279, 280, 283, 284, 288, 290, 293, 297, 299, 300, 301, 306, 307, 311, 312, 314, 315, 318, 319, 324, 325, 328, 341, 344, 346, 348, 350, 353, 358, 360, 362, 366, 367, 368, 373, 377, 379, 382, 384, 385, 390, 393, 396, 399, 404, 405, 412, 414, 415, 417, 418, 429, 431, 432, 436, 447, 451, 456, 457, 460, 463, 469, 473, 477, 478, 481, 487, 490, 496

6. 167 Mitglieder, die bei Streckenlaenge 74810 teilnehmen koennen: 2, 4, 7, 11, 12, 19, 25, 27, 29, 32, 33, 34, 35, 37, 39, 42, 46, 47, 51, 54, 55, 59, 67, 70, 71, 79, 81, 83, 87, 90, 93, 98, 102, 103, 106, 107, 108, 110, 111, 114, 115, 117, 120, 128, 129, 131, 132, 135, 140, 141, 142, 143, 145, 146, 148, 151, 152, 157, 164, 165, 174, 175, 176, 178, 179, 184, 186, 188, 194, 195, 197, 198, 200, 206, 213, 214, 215, 216, 221, 225, 229, 230, 234, 243, 244, 249, 254, 259, 268, 277, 284, 287, 291, 292, 295, 297, 298, 299, 301, 306, 309, 311, 316, 318, 320, 322, 326, 329, 330, 331, 334, 341, 342, 343, 346, 347, 349, 351, 352, 354, 366, 367, 368, 371, 374, 379, 381, 383, 385, 386, 389, 391, 397, 398, 403, 404, 409, 412, 416, 419, 428, 429, 430, 432, 435, 438, 442, 443, 444, 447, 449, 451, 453, 456, 458, 468, 469, 470, 474, 476, 477, 478, 480, 487, 490, 494, 496

7. 102 Mitglieder, die bei Streckenlaenge 92920 teilnehmen koennen: 6, 7, 11, 12, 20, 27, 31, 32, 46, 47, 49, 63, 70, 79, 84, 91, 92, 93, 97, 106, 115, 123, 126, 132, 138, 141, 142, 143, 154, 157, 168, 171, 177, 179, 180, 182, 184, 190, 195, 197, 198, 200, 206, 215, 224, 226, 231, 232, 238, 247, 253, 258, 264, 268, 272, 275, 289, 295, 297, 299, 305, 306, 322, 334, 335, 336, 342, 345, 346, 349, 356, 363, 369, 370, 381, 383, 387, 388, 394, 395, 409, 410, 420, 425, 426, 433, 435, 448, 450, 451, 454, 462, 465, 467, 469, 470, 479, 480, 482, 484, 486, 497

4.7. Beispiel 7 - „wandern7.txt“

1. Maximale Anzahl an Mitgliedern: 551 von 800

2. Streckenlaengen: 39520, 76088, 91584

3. Mitglieder, die teilnehmen: 0, 3, 4, 5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 19, 20, 21, 22, 25, 26, 27, 28, 29, 30, 32, 34, 37, 40, 41, 42, 43, 44, 45, 46, 49, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 68, 69, 70, 72, 73, 74, 75, 76, 78, 79, 80, 81, 82, 85, 86, 87, 88, 90, 91, 92, 94, 95, 98, 100, 101, 102, 103, 104, 106, 109, 110, 111, 113, 114, 115, 117, 119, 121, 122, 123, 124, 125, 126, 128, 130, 132, 133, 134, 135, 136, 138, 141, 145, 146, 147, 148, 150, 152, 153, 154, 157, 159, 160, 161, 162, 163, 164, 165, 166, 167, 169, 170, 171, 172, 173, 174, 175, 176, 179, 180, 182, 183, 184, 185, 186, 187, 188, 190, 191, 192, 194, 195, 197, 198, 201, 202, 204, 207, 208, 209, 210, 212, 213, 214, 215, 216, 217, 218, 219, 222, 224, 228, 229, 231, 233, 237, 238, 243,

244, 245, 248, 249, 251, 252, 254, 255, 256, 257, 258, 259, 261, 265, 267, 269, 270, 272, 275, 276, 277, 280, 282, 283, 284, 286, 287, 289, 290, 291, 292, 294, 295, 298, 300, 301, 303, 304, 305, 307, 308, 309, 311, 312, 313, 314, 315, 317, 318, 319, 320, 321, 325, 326, 331, 334, 336, 337, 339, 340, 342, 343, 346, 347, 348, 349, 351, 352, 353, 356, 357, 358, 359, 360, 362, 363, 365, 366, 368, 369, 370, 372, 373, 374, 375, 377, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 390, 391, 393, 394, 395, 396, 397, 398, 400, 401, 403, 405, 406, 407, 409, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 432, 433, 434, 436, 439, 440, 441, 444, 446, 448, 449, 452, 453, 454, 455, 456, 460, 462, 464, 466, 467, 468, 469, 471, 472, 475, 476, 478, 479, 480, 481, 485, 487, 488, 489, 491, 492, 493, 494, 496, 498, 500, 502, 504, 505, 506, 507, 508, 509, 510, 511, 512, 514, 517, 519, 520, 521, 523, 524, 526, 527, 528, 529, 530, 531, 532, 533, 534, 536, 537, 538, 539, 540, 541, 544, 545, 546, 547, 548, 549, 550, 551, 554, 555, 556, 557, 558, 559, 561, 562, 563, 565, 566, 569, 572, 573, 574, 576, 577, 580, 581, 582, 583, 584, 586, 587, 588, 589, 590, 591, 592, 594, 596, 600, 601, 603, 604, 605, 606, 607, 609, 610, 611, 613, 615, 617, 619, 620, 621, 622, 623, 624, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 640, 643, 644, 645, 646, 647, 648, 649, 651, 653, 654, 657, 658, 660, 663, 664, 665, 666, 668, 671, 672, 674, 675, 677, 678, 681, 682, 685, 686, 687, 688, 690, 691, 695, 696, 697, 698, 699, 700, 704, 705, 706, 707, 708, 710, 711, 712, 713, 715, 716, 718, 719, 723, 725, 727, 728, 729, 730, 732, 733, 735, 736, 737, 739, 740, 741, 742, 743, 744, 746, 747, 748, 749, 751, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 770, 772, 773, 775, 776, 778, 779, 780, 781, 782, 784, 785, 786, 787, 789, 790, 792, 794, 795, 797, 798, 799

4. Mitglieder, die nicht teilnehmen: 1, 2, 7, 13, 17, 18, 23, 24, 31, 33, 35, 36, 38, 39, 47, 48, 50, 51, 52, 67, 71, 77, 83, 84, 89, 93, 96, 97, 99, 105, 107, 108, 112, 116, 118, 120, 127, 129, 131, 137, 139, 140, 142, 143, 144, 149, 151, 155, 156, 158, 168, 177, 178, 181, 189, 193, 196, 199, 200, 203, 205, 206, 211, 220, 221, 223, 225, 226, 227, 230, 232, 234, 235, 236, 239, 240, 241, 242, 246, 247, 250, 253, 260, 262, 263, 264, 266, 268, 271, 273, 274, 278, 279, 281, 285, 288, 293, 296, 297, 299, 302, 306, 310, 316, 322, 323, 324, 327, 328, 329, 330, 332, 333, 335, 338, 341, 344, 345, 350, 354, 355, 361, 364, 367, 371, 376, 378, 389, 392, 399, 402, 404, 408, 410, 430, 431, 435, 437, 438, 442, 443, 445, 447, 450, 451, 457, 458, 459, 461, 463, 465, 470, 473, 474, 477, 482, 483, 484, 486, 490, 495, 497, 499, 501, 503, 513, 515, 516, 518, 522, 525, 535, 542, 543, 552, 553, 560, 564, 567, 568, 570, 571, 575, 578, 579, 585, 593, 595, 597, 598, 599, 602, 608, 612, 614, 616, 618, 625, 639, 641, 642, 650, 652, 655, 656, 659, 661, 662, 667, 669, 670, 673, 676, 679, 680, 683, 684, 689, 692, 693, 694, 701, 702, 703, 709, 714, 717, 720, 721, 722, 724, 726, 731, 734, 738, 745, 750, 752, 753, 768, 769, 771, 774, 777, 783, 788, 791, 793, 796

5. 285 Mitglieder, die bei Streckenlaenge 39520 teilnehmen koennen: 3, 4, 5, 8, 9, 10, 11, 15, 16, 21, 26, 27, 28, 29, 30, 37, 40, 41, 42, 44, 45, 58, 60, 61, 63, 65, 69, 70, 75, 76, 78, 82, 85, 87, 88, 90, 94, 95, 98, 103, 104, 106, 110, 114, 115, 117, 123, 126, 132, 134, 136, 141, 147, 148, 152, 154, 157, 159, 160, 161, 162, 163, 165, 169, 170, 173, 174, 175, 179, 180, 182, 184, 190, 191, 195, 207, 209, 210, 212, 214, 215, 218, 219, 224, 228, 229, 237, 238, 244, 245, 251, 252, 254, 261, 265, 269, 272, 276, 283, 284, 286, 287, 291, 292, 294, 298, 301, 303, 304, 307, 308, 309, 311, 313, 319, 320, 321, 325, 326, 331, 336, 340, 342, 347, 351, 353, 356, 357, 360, 363, 365, 370, 372, 374, 375, 379, 380, 381, 382, 384, 386, 390, 393, 396, 398, 400, 403, 407, 409, 412, 414, 421, 423, 425, 433, 436, 441, 444, 446, 448, 449, 453, 462, 464, 466, 469, 471, 475, 476, 478, 485, 488, 493, 496, 498, 500, 502, 504, 505, 509, 511, 512, 517, 519, 520, 523, 524, 527, 528, 529, 530, 531, 532, 533, 539, 541, 545, 547, 550, 558, 559, 566, 569, 573, 576, 577, 581, 582, 583, 584, 588, 594, 596, 601, 603, 611, 613, 615, 617, 619, 622, 623, 627, 630, 631, 633, 635, 637, 638, 640, 643, 648, 649, 651, 658, 663, 665, 666, 668, 671, 672, 675, 678, 681, 685, 688, 700, 704, 705, 707, 708, 710, 712, 723, 725, 727, 728, 730, 732, 733, 736, 739, 741, 742, 747, 751, 755, 756, 757, 760, 762, 766, 767, 773, 775, 778, 780, 782, 786, 789, 790, 794, 795, 797, 799

6. 278 Mitglieder, die bei Streckenlaenge 76088 teilnehmen koennen: 0, 6, 8, 10, 11, 12, 15, 20, 22, 25, 27, 32, 34, 37, 41, 46, 49, 54, 57, 58, 61, 62, 63, 64, 66, 68, 69, 73, 79, 80, 85, 86, 91, 92, 94, 98, 100, 109, 110, 111, 113, 115, 119, 121, 122, 124, 125, 126, 128, 133, 136, 138, 141, 145, 147, 152, 157, 166, 169, 170, 174, 179, 183, 184, 185, 186, 187, 188, 192, 194, 195, 197, 201, 202, 204, 208, 210, 216, 217, 219, 222, 224, 228, 229, 231, 233, 237, 243, 244, 248, 249, 251, 255, 256, 257, 258, 267, 270, 276, 277, 280, 284, 289, 290, 295, 298, 300, 304, 305, 307, 311, 312, 315, 317, 319, 325, 334, 337, 339, 342, 343, 346, 348, 349, 352, 358, 359, 360, 366, 368, 370, 373, 375, 379, 383, 385, 388, 394, 395, 403, 409, 417, 420, 422, 424, 425, 426, 427, 428, 434, 440, 449, 454, 455, 456, 460, 468, 469, 472, 475, 478, 479, 480, 481, 487, 488, 491, 494, 496, 498, 505, 506, 507, 532, 534, 536, 537, 538, 539, 541, 544, 549, 550, 551, 555, 561, 562, 563, 565, 566, 569, 572, 573, 577, 581, 582, 583, 584, 586, 588, 589, 590, 591, 592, 594, 600, 604, 605, 607, 609, 610, 611, 621, 622, 624, 628, 631, 634, 636, 638, 640, 644, 645, 646, 651, 657, 660, 663, 665, 671, 672, 677, 687, 690, 691, 695, 696, 698, 699, 700, 704, 706, 710, 711, 712, 715, 716, 718, 725, 727, 729, 732, 735, 737, 740, 741, 743, 746, 749, 755, 756, 757, 759, 762, 763, 764, 765, 766, 767, 770, 772, 776, 782, 784, 785, 790, 794, 798

7. 195 Mitglieder, die bei Streckenlaenge 91584 teilnehmen koennen: 11, 12, 14, 19, 27, 37, 43, 46, 53, 54, 55, 56, 57, 59, 61, 62, 64, 66, 68, 72, 74, 79, 80, 81, 92, 94, 100, 101, 102, 109, 111, 113, 121, 125, 130, 135,

141, 146, 150, 152, 153, 164, 167, 169, 171, 172, 176, 184, 185, 187, 188, 192, 194, 195, 197, 198, 204, 213, 217, 224, 228, 231, 233, 237, 243, 248, 249, 251, 255, 257, 259, 270, 275, 276, 277, 282, 314, 318, 334, 342, 343, 360, 362, 368, 369, 370, 375, 377, 379, 387, 391, 395, 397, 401, 405, 406, 411, 413, 415, 416, 417, 418, 419, 425, 429, 432, 439, 452, 456, 467, 469, 475, 478, 479, 480, 487, 488, 489, 492, 498, 505, 508, 510, 514, 521, 526, 538, 540, 541, 546, 548, 549, 550, 551, 554, 556, 557, 561, 565, 566, 569, 574, 580, 581, 582, 584, 587, 592, 605, 606, 607, 610, 620, 626, 629, 632, 636, 644, 646, 647, 653, 654, 657, 664, 674, 677, 682, 686, 695, 697, 698, 704, 711, 713, 718, 719, 729, 741, 744, 748, 754, 757, 758, 761, 764, 765, 776, 779, 781, 782, 784, 787, 790, 792, 794

5. Quellcode

Nun folgt der Quellcode der wichtigsten Teile der Implementationen der Loesungsverfahren in C++.

5.1. Loesungsvorschlag I - Bruteforceverfahren

Hier entspricht *starting_points[i]* dem Wert a_i.

```
// Bruteforce all triples of starting points
// Keep track of currently best triple
int best_amount = 0;
tuple<int, int, int> best_lengths;

for (int i = 0; i < n; i++) {
    for (int j = i+1; j < n; j++) {
        for (int k = j+1; k < n; k++) {
            // Current triple (l1, l2, l3)
            int l1 = starting_points[i];
            int l2 = starting_points[j];
            int l3 = starting_points[k];

            // Count the amount of members that would participate when using the current lengths
            int counter = 0;

            for (int c = 0; c < n; c++) {
                int start = values[c].first;
                int end = values[c].second;
                // A member participates if one of the three lengths matches his preferences
                if ((start <= l1 && l1 <= end) || (start <= l2 && l2 <= end)
                        || (start <= l3 && l3 <= end)) {
                    counter++;
                }
            }

            // Update the best triple if necessary
            if (counter > best_amount) {
                best_lengths = {starting_points[i], starting_points[j], starting_points[k]};
                best_amount = counter;
            }
        }
    }
}
```

5.2. Loesungsvorschlag II

Hier entspricht *values[i]* dem Tupel $(a_i,\ b_i)$.

```
  // Create vector only containing the starting point of each member and sort
    vector<int> starting_points;
    for (int i = 0; i < n; i++) {
        starting_points.push_back(values[i].first);
    }

    sort(starting_points.begin(), starting_points.end());

    // Keep track of the currently best triple
    tuple<int, int, int> best_lengths;
    int best_amount = 0;

```

```
    // Find the best possible triple by fixing the middle length
    for (int i = 0; i < n; i++) {
        int mid = starting_points[i];
        tuple<int, int, int> current_lengths;
        int current_amount;
        tie(current_lengths, current_amount) = foo(values, mid);

        if (current_amount > best_amount) {
            best_amount = current_amount;
            best_lengths = current_lengths;
        }
    }



  // Function finding the best triple (left, mid, right) given mid
pair<tuple<int, int, int>, int> foo(vector<pair<int, int>> &values, int mid) {
    int n = values.size();
    vector<int> left;
    vector<int> right;
    int counter = 0;

    // Calculate the members completely left and completely right
    // to mid and count amount of members covered by mid
    for (int i = 0; i < n; i++) {
        int start = values[i].first;
        int end = values[i].second;
        if (end < mid) {
            left.push_back(i);
        } else if (start <= mid && mid <= end) {
            counter++;
        } else if (mid < start) {
            right.push_back(i);
        }
    }

    // Get the best amount to the left and to the right
    // and return correspondingly
    int amount_left, length_left;
    tie(amount_left, length_left) = bar(values, left);

    int amount_right, length_right;
    tie(amount_right, length_right) = bar(values, right);

    int result = amount_left + counter + amount_right;
    return {{length_left, mid, length_right}, result};
}

  // Function returing the best possible amount of members and the corresponding length
// given a vector of indeces of members to involved
pair<int, int> bar(vector<pair<int, int>> &values, vector<int> &members) {
    // Get all needed values a_i and b_i corresponding to the given members
    vector<pair<int, bool>> lenghts;

    for (int i : members) {
        lenghts.push_back({values[i].first, false});
        lenghts.push_back({values[i].second, true});
    }

    sort(lenghts.begin(), lenghts.end());

    int best_amount = 0;
    int best_length = -1;
    int current_amount = 0;

    for (int i = 0; i < lenghts.size(); i++) {
        int length;
        bool type;
        tie(length, type) = lenghts[i];

        // Increase the counter if length is a staring point and decrease otherwise
        if (type == false) {
            current_amount++;

            // Update the best length if needed
            if (current_amount > best_amount) {
                best_amount = current_amount;
```

```
                best_length = length;
            }
        } else {
            current_amount--;
        }
    }

    return {best_amount, best_length};
}
```

5.3. Loesungsvorschlag III

Auch hier entspricht *values[i]* dem Tupel $(a_i,\ b_i)$.

```
    // Create vector only containing the starting point of each member and sort
    // and create vector containing all points a_i and b_i
    vector<int> starting_points;
    vector<tuple<int, bool, int>> list;
    for (int i = 0; i < n; i++) {
        starting_points.push_back(values[i].first);
        list.push_back({values[i].first, false, i});
        list.push_back({values[i].second, true, i});
    }

    sort(starting_points.begin(), starting_points.end());
    sort(list.begin(), list.end());

    // Keep track of the currently best triple
    tuple<int, int, int> best_lengths;
    int best_amount = 0;

    // Find the best possible triple by fixing the middle length
    for (int i = 0; i < n; i++) {
        int mid = starting_points[i];

        // Get current amount for mid
        int best_mid = 0;

        for (int j = 0; j < n; j++) {
            if (values[j].first <= mid && mid <= values[j].second) {
                best_mid++;
            }
        }

        // Get best possible amount to the left of mid
        int best_amount_left = 0;
        int best_length_left = -1;
        int current_amount_left = 0;

        for (int j = 0; j < 2*n; j++) {
            int length, index;
            bool type;
            tie(length, type, index) = list[j];

            // Dont consider member if its range is not completely to the left of mid
            if (values[index].second >= mid) {
                continue;
            }

            if (type == false) {
                current_amount_left++;

                if (current_amount_left > best_amount_left) {
                    best_amount_left = current_amount_left;
                    best_length_left = length;
                }
            } else {
                current_amount_left--;
            }
        }

        // Get best possible amount to the right of mid#
        int best_amount_right = 0;
```

```
        int best_length_right = -1;
61      int current_amount_right = 0;

63      for (int j = 0; j < 2*n; j++) {
            int length, index;
65          bool type;
            tie(length, type, index) = list[j];
67
            // Dont consider member if its range is not completely to the right of mid
69          if (values[index].first <= mid) {
                continue;
71          }

73          if (type == false) {
                current_amount_right++;
75
                if (current_amount_right > best_amount_right) {
77                  best_amount_right = current_amount_right;
                    best_length_right = length;
79              }
            } else {
81              current_amount_right--;
            }
83      }

85      // Update the overall best amount if needed
        int amount = best_amount_left + best_mid + best_amount_right;
87
        if (amount > best_amount) {
89          best_amount = amount;
            best_lengths = {best_length_left, mid, best_length_right};
91      }
    }
```