Aufgabe 3: Wandertag

Team-ID: 00079

Team: Constantin Reinhold

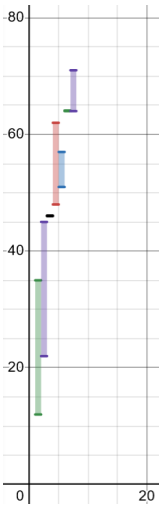
Bearbeiter/-innen dieser Aufgabe:
Constantin Reinhold

17. November 2024

Inhaltsverzeichnis

Lösungsidee....1
Umsetzung....1
Beispiele....2
Quellcode....2

Lösungsidee



Stellen wir uns wir repräsentieren das Problem in einem Koordinatensystem. Für wandern1.txt könnte das ganze so aussehen:
Wir können recht einfach erkennen, dass das wir im Optimalfall die Werte 22 bis 35, 51 bis 57 und 64 wählen. Damit schneiden wir 6 der 7 Bereiche. Diesen Ablauf wollen wir simulieren. Wir wollen also die drei Werte finden die mit möglichst vieler dieser Bereiche schneiden.

Umsetzung

Als erstes lesen wir die Aufgabendatei ein. Wir sortieren unsere Wert aufsteigend um möglichst einfach unsere Überschneidungen finden zu können. Danach rufen wir die Funktion *find_values* auf.

find_values

Die Funktion findet alle sinnvollen Rennlängen und die Überschneidungen mit den Wunschlängen durch einen Sweep-Line Algorithmus. Der Algorithmus geht jedes Minimum-Maximum-Paar durch und findet Überschneidungen und speichert für jede sinnvolle Rennlänge die Überschneidungen und die teilnehmenden Leute.

Diese Liste sortieren wir dann der Anzahl der Überschneidungen.

Danach rufen wir die Funktion *iter_combinations* auf.

iter_combinations

Die Funktion findet alle möglichen Kombination aus r items, in diesem Fall Listen von teilnehmenden Leuten zurück.

Für jede Kombination finden wir danach mit einem loop die Anzahl einzigartiger Teilnehmer. Diese Anzahl einzigartiger Teilnehmer sortieren wir absteigend und die Kombination mit den meisten einzigartigen Teilnehmern ist unsere optimale Lösung. Wir finden unsere Rennlängen für diese Kombination wieder und geben usere Lösung aus.

Beispiele

Wandern1.txt:

Beste 3 Werte sind: 22, 51, 64.

Dabei nehmen jeweils die Personen:

0, 1;

3, 4;

5, 6 teil.

Insgesamt würden 6 Leute teilnehmen.

Dies ist das oben besprochene Beispiel. Die Lösungen können ganz einfach auch selbst im Graphen gefunden werden. Kein Sonderfall.

Wandern2.txt:

Beste 3 Werte sind: 40, 60, 10.

Dabei nehmen jeweils die Personen:

5, 3, 2;

3, 2, 0;

4, 5 teil.

Insgesamt würden 5 Leute teilnehmen.

Wandern3.txt:

Beste 3 Werte sind: 66, 19, 92.

Dabei nehmen jeweils die Personen:

2, 1, 0, 5;

6, 8, 9;

4, 7, 3 teil.

Insgesamt würden 10 Leute teilnehmen.

Wandern4.txt:

Beste 3 Werte sind: 770, 221, 593.

Dabei nehmen jeweils die Personen:

75, 73, 16, 96, 51, 74, 8, 18, 13, 43, 63, 27, 48, 79, 87, 53, 55, 59, 5, 1, 62, 0;

41, 3, 65, 90, 11, 78, 76, 20, 45, 83, 28, 2, 60, 40, 42;

54, 6, 9, 80, 7, 19, 34, 37, 24, 15, 31, 14, 17, 26 teil.

Insgesamt würden 51 Leute teilnehmen.

Wir sehen, dass die Eingaben und Ausgaben deutlich länger werden.

Wandern5.txt:

Beste 3 Werte sind: 14359, 36951, 28194.

Dabei nehmen jeweils die Personen:

160, 12, 117, 113, 109, 27, 175, 121, 47, 66, 29, 124, 69, 170, 190, 152, 23, 128, 111, 127, 42, 188, 98, 199, 163, 106, 189, 118, 74, 21, 145, 177, 55, 126, 32, 195, 89;

26, 178, 94, 2, 1, 7, 78, 174, 191, 64, 180, 72, 103, 107, 102, 186, 168, 192, 123, 197, 44, 40, 5, 176, 71, 151, 194, 37, 110, 45, 91, 73, 144, 38, 105;

126, 32, 195, 89, 88, 135, 11, 95, 20, 97, 181, 17, 114, 133, 136, 182, 39, 86, 54, 67, 93, 96, 63, 115, 161, 104, 14, 108, 0, 26, 178, 94, 2 teil.

Insgesamt würden 97 Leute teilnehmen.

Und auch die Optimalwerte werden sehr groß.

Wandern6.txt:

Beste 3 Werte sind: 16208, 31426, 37994.

Dabei nehmen jeweils die Personen:

149, 145, 18, 128, 172, 294, 215, 199, 475, 385, 51, 390, 5, 196, 211, 406, 16, 73, 165, 456, 332, 37904, 279, 337, 311, 317, 36, 152, 266, 248, 260, 112, 299, 244, 64, 464, 249, 257, 125, 162, 415;
04, 279, 337, 311, 317, 36, 152, 266, 248, 260, 112, 299, 244, 64, 464, 249, 257, 125, 162, 415;
04, 279, 337, 311, 317, 36, 152, 266, 204, 279, 337, 311, 317, 36, 152, 266, 248, 260, 112, 299, 244, 64, 464, 249, 257, 125, 162, 415;

189, 288, 75, 399, 451, 236, 414, 194, 473, 46, 0, 360, 43, 421, 87, 344, 52, 233, 193, 457, 300, 319, 460, 130, 412, 181, 175, 487, 362, 94, 251, 54, 384, 88, 496, 278, 66, 120, 436, 70, 223;

120, 436, 70, 223, 301, 324, 117, 463, 307, 405, 41, 214, 137, 222, 114, 9, 26, 353, 447, 148, 485, 195, 282, 314, 144, 237, 228, 210, 110, 396, 284, 350, 121, 348, 136, 306, 132, 477, 205 teil.

Insgesamt würden 119 Leute teilnehmen.

Dieses und Wandern7.txt wurden auf der offizielen BWINF-Website als Sonderfälle angegeben, da die Eingabedateien so groß sind, dass sie mit einer simplen Brute-Force Methode ewig dauern würden. In meinem Programm konnte ich die Laufzeit des Programmes von 120s auf 0.2s herunter trimmen. Die Änderung die ich vorgenommen habe war die Überprüfung:

```
if intersections > 1 or len(scores) < 3:
        if minimum not in seen_minimums:
            scores.append([intersections, people, minimum])
            seen_minimums.add(minimum)
```

In der ersten Zeile wird sichergestellt, dass wir nur Werte speichern bei denen intersections mindestens 2 ist, außer wenn wir zwingend einen dritten Wert benötigen um unsere Kombination formen zu können. Diese Änderung sorgt dafür, dass wir ca. 90% der Werte garnicht mehr überprüfen müssen.
In der zweiten Zeile fügen stellen wir sicher, dass wir nur Werte speichern die wir nicht bereits in der Liste *seen_minimums* gespeichert haben. Wie auch die erste Zeile überspringen wir so eine Menge an Fällen, die sonst unnötiger Weise überprüft werden oder zu falschen Ergebnissen führen.

Wandern7.txt:

Beste 3 Werte sind: 21051, 13544, 26418.
Dabei nehmen jeweils die Personen:
386, 8, 174, 727, 76, 757, 184, 623, 16, 509, 215, 796, 302, 278, 236, 462, 252, 207, 762, 384, 788, 260, 527, 596, 379, 87, 324, 539, 433, 522, 436, 766, 157, 182, 446, 597, 245, 41, 399, 189, 363, 547, 170, 651, 520, 583, 177, 532, 625, 519, 516, 309, 382, 292, 441, 760, 789, 311, 241;
505, 503, 444, 498, 161, 160, 448, 617, 271, 26, 265, 633, 331, 577, 126, 322, 148, 319, 482, 722, 670, 728, 218, 200, 338, 232, 705, 214, 478, 115, 276, 159, 767, 77, 550, 78, 410, 261, 376, 611, 175, 4, 438, 112, 733, 313, 497, 154, 132, 559, 531, 449, 648;
70, 58, 643, 301, 488, 332, 179, 443, 94, 284, 374, 165, 627, 229, 13, 84, 90, 666, 360, 286, 141, 61, 453, 573, 353, 751, 155, 795, 425, 110, 294, 613, 414, 649, 486, 528, 42, 601, 569, 511, 504, 117, 173, 663, 65, 375, 680, 240, 320 teil.
Insgesamt würden 161 Leute teilnehmen.

Quellcode

```
filename = 'wandernX.txt'

# Datei einlesen
with open(filename, 'r') as file:
    lines = file.readlines()
# Erste Zeile wird nicht benötigt, andere Werte in Array speichern
number_array = [[index, tuple(map(int, line.strip().split()))] for index, line in
enumerate(lines[1:])]
```

```
# Aufsteigend nach erstem Wert sortieren
number_array = sorted(number_array, key=lambda x: x[1])

scores = []
seen_minimums = set()

def find_values(start_index):
    intersections = 0
    people = []
    minimum = 0
    maximum = 10**100

    for pair in number_array[start_index:]:
        if int(pair[1][0]) >= minimum and int(pair[1][0]) <= maximum:
            minimum = int(pair[1][0])
        else:
            if intersections > 1 or len(scores) < 3:
                if minimum not in seen_minimums:
                    scores.append([intersections, people, minimum])
                    seen_minimums.add(minimum)
            start_index += 1
            if start_index < len(number_array):
                find_values(start_index)
            return

        if int(pair[1][1]) >= minimum and int(pair[1][1]) <= maximum:
            maximum = int(pair[1][1])

        intersections += 1
        people.append(pair[0])
    if intersections > 1 or len(scores) < 3:
        if minimum not in seen_minimums:
            scores.append([intersections, people, minimum])
            seen_minimums.add(minimum)

find_values(0) # Ruft die Funktion das erste mal auf
scores = sorted(scores, key=lambda x: x[0], reverse=True) # Sortiert nach der Anzahl der
Überschneidungen

people = []
for score in scores:
    people.append(score[1]) # Speichert die teilnehmenden Leute für jeden möglichen Wert

def iter_combinations(iterable, r):
    pool = tuple(iterable)
    n = len(pool)
    if r > n:
```

```
        return
    indices = list(range(r))

    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r:
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)

combinations = list(iter_combinations(people, 3))

# Berechnet für jede Kombination die Anzahl an einzigartigen Teilnehmern
top_scores = []
for index,combination in enumerate(combinations):
    top_score = len(set(combination[0] + combination[1] + combination[2]))
    top_scores.append([index, top_score])

# Sortiert die Liste top_scores absteigend
top_scores = sorted(top_scores, key=lambda x: x[1], reverse=True)
top_scores = top_scores[0]

# Gibt die beste Kombination zurück
best_combination = combinations[top_scores[0]]

# Findet die zugehörigen Indexes für die teilnehmenden Leute der besten Kombination
matching_results = []
for index, score in enumerate(scores):
    if score[1] in best_combination:
        matching_results.append(score)

# Gibt das Ergebnis aus
print(f"""Beste 3 Werte sind: {matching_results[0][2]}, {matching_results[1][2]},
{matching_results[2][2]}.
Dabei nehmen jeweils die Personen:
{str(matching_results[0][1]).replace('[','').replace(']', '')};
{str(matching_results[1][1]).replace('[','').replace(']', '')};
{str(matching_results[2][1]).replace('[','').replace(']', '')} teil.
Insgesamt würden {top_scores[1]} Leute teilnehmen.""")
```