Aufgabe 3: Wandertag

Team-ID: 00153

Team-Name: ByteBusters

Bearbeiter/-innen dieser Aufgabe:
Benedikt Wiesner

17. November 2024

Inhaltsverzeichnis

1 Lösungsidee **1**

2 Umsetzung **2**
2.1 Dateneinlese 2
2.2 Greedy-Algorithmus 2
2.3 Hilfsfunktionen 3

3 Beispiele **4**

4 Diskussion und Ausblick **4**

5 Quellcode **5**
5.1 Greedy-Algorithmus 5
5.2 Teilnehmerzahlen 5
5.3 Dateiauswahl und Verarbeitung 5

1 Lösungsidee

Die Kernidee unserer Lösung basiert auf einem Greedy-Algorithmus, der darauf abzielt, die maximale Anzahl von Teilnehmern mit drei optimal gewählten Streckenlängen abzudecken. Der Algorithmus wird in den folgenden Schritten ausgeführt:

1. **Erstellung der Streckenlängenliste**: Zunächst werden alle möglichen Streckenlängen ermittelt, indem die minimalen und maximalen Streckenlängen aller Teilnehmer gesammelt werden. Diese Streckenlängen stellen die potenziellen Werte dar, die für die Abdeckung der Teilnehmer erforderlich sind.

2. **Berechnung der Abdeckung für jede Streckenlänge**: Für jede dieser möglichen Streckenlängen wird berechnet, wie viele neue Teilnehmer sie abdeckt, die bisher noch nicht durch eine der bereits gewählten Streckenlängen erfasst wurden. Diese Berechnung erfolgt unter Berücksichtigung der individuellen Anforderungen der Teilnehmer.

3. **Auswahl der optimalen Streckenlänge**: Die Streckenlänge, die die größte Anzahl neuer Teilnehmer abdeckt, wird ausgewählt und zur aktuellen Lösung hinzugefügt. Dies stellt sicher, dass bei jeder Wahl die Abdeckung maximiert wird.

4. **Wiederholung des Auswahlprozesses**: Dieser Prozess wird wiederholt, bis insgesamt drei Streckenlängen ausgewählt wurden oder bis keine weiteren Teilnehmer mehr durch zusätzliche Streckenlängen abgedeckt werden können. Sollte es keine weiteren Teilnehmer geben, die durch eine zusätzliche Streckenlänge erfasst werden können, endet der Algorithmus vorzeitig.

Dieser Greedy-Ansatz maximiert in jeder Runde die Anzahl der neu abgedeckten Teilnehmer, was zu einer optimalen oder nahezu optimalen Lösung führt. Der Algorithmus trifft dabei lokale Entscheidungen, ohne langfristige Auswirkungen zu berücksichtigen. Aufgrund der begrenzten Anzahl an Streckenlängen wird er in akzeptabler Zeit ausgeführt und liefert in vielen Fällen eine Lösung, die der optimalen sehr nahekommt.

2 Umsetzung

Die Implementierung unserer Lösung wurde in Python realisiert und umfasst mehrere Hauptkomponenten, die in den folgenden Untersektionen beschrieben werden.

2.1 Dateneinlese

Zu Beginn des Programms werden die Eingabedaten aus einer Textdatei eingelesen, die im gleichen Verzeichnis wie das Skript abgelegt ist. Die erste Zeile enthält die Anzahl der Personen, gefolgt von den minimalen und maximalen Streckenlängen jeder Person.

2.2 Greedy-Algorithmus

Der Greedy-Algorithmus stellt den zentralen Bestandteil der Lösung dar. Das Hauptziel besteht darin, mit einer Auswahl von genau drei Streckenlängen die maximale Anzahl an Teilnehmern abzudecken. In jeder Iteration wird die Streckenlänge gewählt, die die größte Anzahl an neuen, bisher nicht abgedeckten Teilnehmern erreicht.

Formell betrachtet:

- Gegeben sei eine Menge von Teilnehmern $P = \{p_1, p_2, \ldots, p_n\}$ mit deren minimalen und maximalen Streckenlängen $[\min_i, \max_i]$.
- Zu Beginn sind noch keine Strecken gewählt und keine Teilnehmer abgedeckt. Das Ziel ist es, mit jeder neuen Streckenlänge die Zahl der abgedeckten Teilnehmer zu maximieren.

Der Algorithmus wird in mehreren Schritten durchgeführt:

1. **Initialisierung**: Es wird eine Liste aller möglichen Streckenlängen (Minimal- und Maximalwerte für jede Person) erstellt. Diese Strecken bilden die Kandidaten für die Auswahl.
2. **Iterative Auswahl**: In jeder Iteration wird die Streckenlänge ausgewählt, die die größte Anzahl an neuen Teilnehmern abdeckt. Diese neuen Teilnehmer sind diejenigen, die noch nicht durch andere Streckenlängen abgedeckt wurden. Formal lässt sich die Anzahl der neuen Teilnehmer für eine Streckenlänge s als folgende Menge beschreiben:

$$\text{Neue Teilnehmer}(s) = \{p_i \mid \min_i \leq s \leq \max_i\} \setminus \text{abgedeckte Teilnehmer}.$$

   Die Streckenlänge s_{best} wird durch Maximierung der Anzahl neuer Teilnehmer ausgewählt:

$$s_{\text{best}} = \arg\max_s |\text{Neue Teilnehmer}(s)|.$$

3. **Abbruchbedingung**: Dieser Prozess wird fortgesetzt, bis entweder drei Streckenlängen ausgewählt wurden oder keine neuen Teilnehmer mehr abgedeckt werden können.

Der Algorithmus stellt sicher, dass in jedem Schritt die bestmögliche Entscheidung getroffen wird, um die Anzahl der abgedeckten Teilnehmer zu maximieren.

Die Laufzeit des Algorithmus ist in $O(m \cdot n)$, wobei m die Anzahl der Streckenlängen und n die Anzahl der Teilnehmer ist. Da maximal drei Streckenlängen ausgewählt werden, ergibt sich eine Gesamtkomplexität von $O(3 \cdot m \cdot n)$, was bei kleinen bis mittleren Eingabemengen effizient ist.

```python
def greedy_algorithm(personen):
    gew hlte_strecken = []
    abgedeckte_personen = set()
    strecken_kandidaten = set()

    for min_strecke, max_strecke in personen:
        strecken_kandidaten.add(min_strecke)
        strecken_kandidaten.add(max_strecke)

    for _ in range(3):
        beste_strecke = None
        beste_abdeckung = set()

        for strecke in strecken_kandidaten:
            aktuelle_abdeckung = set(teilnehmer_zahlen(strecke, personen))
            neue_abdeckung = aktuelle_abdeckung - abgedeckte_personen

            if len(neue_abdeckung) > len(beste_abdeckung):
                beste_strecke = strecke
                beste_abdeckung = neue_abdeckung

        if beste_strecke is None:
            break

        gew hlte_strecken.append(beste_strecke)
        abgedeckte_personen.update(beste_abdeckung)
        strecken_kandidaten.discard(beste_strecke)

    return gew hlte_strecken, abgedeckte_personen
```

In der Implementierung wird in jeder Iteration die Anzahl der abgedeckten Teilnehmer mit einer bestimmten Streckenlänge durch die Funktion `teilnehmer_zahlen` ermittelt. Anschließend wird die Strecke ausgewählt, die die größte Menge an nicht abgedeckten Teilnehmern erreicht. Die Laufzeit des Algorithmus wird durch die Anzahl der Iterationen und der zu prüfenden Streckenlängen bestimmt.

2.3 Hilfsfunktionen

Zur Unterstützung des Algorithmus wurden mehrere Hilfsfunktionen implementiert:

- `teilnehmer_zahlen`: Bestimmt, welche Teilnehmer eine bestimmte Strecke absolvieren können.
- `open_in_new_terminal`: Öffnet das Skript in einem neuen Terminalfenster (nur für Windows).
- `menü_anzeigen`: Zeigt das Hauptmenü mit ASCII-Art an.
- `option_1`: Verarbeitet die Auswahl der zu scannenden Datei und führt den Greedy-Algorithmus aus.
- `Laufzeitmessung`: Misst die Laufzeit des gesamten Programms, indem die Zeit vor und nach der Algorithmus-Ausführung erfasst und die Differenz berechnet wird.

Mit diesen Komponenten wurde eine effiziente und übersichtliche Lösung für das Problem implementiert.

3 Beispiele

Dateiname	Lösung	Laufzeit
wandern1.txt //kleine Datenmenge	• Gewählte Streckenlängen: [64, 35, 51] • Teilnehmende Personen: [0, 1, 3, 4, 5, 6]	0.0010 s
wandern7.txt //große Datenmenge	• Gewählte Streckenlängen: [52515, 86493, 30966] • Teilnehmende Personen: [2, 3, 4, 5, 7, 8, ..., 797, 799] (499 Personen)	0.2920 s
leer.txt //leere Datei	• Die Datei ist leer.	N/A
eine.txt // nur eine Eingabe	• Gewählte Streckenlängen: [50] • Teilnehmende Personen: [0]	0.0010 s
gleich.txt //gleiche Eingaben	• Gewählte Streckenlängen: [42] • Teilnehmende Personen: [0, 1, 2, 3, 4]	0.0029 s
keine.txt //keine Überschneidungen	• Gewählte Streckenlängen: [70, 40, 10] • Teilnehmende Personen: [0, 1, 3]	0.0030 s

Tabelle 1: Ergebnisse der Dateiverarbeitung (alle BWINF Beispieleingaben wurden getestet)

4 Diskussion und Ausblick

Unser Greedy-Algorithmus liefert in den meisten Fällen eine optimale oder nahezu optimale Lösung. Er ist effizient und einfach zu implementieren. Allerdings gibt es einige Punkte zu beachten:

- Der Algorithmus garantiert nicht immer die global optimale Lösung, da er in jedem Schritt die lokal beste Wahl trifft.
- In Extremfällen, wo eine globale Betrachtung notwendig wäre, könnte der Algorithmus suboptimale Ergebnisse liefern.

Für zukünftige Verbesserungen könnten wir folgende Ansätze in Betracht ziehen:

- Implementierung eines exakten Algorithmus (z.B. dynamische Programmierung) für kleinere Datensätze, um die Optimalität zu garantieren.
- Entwicklung eines hybriden Ansatzes, der den Greedy-Algorithmus mit lokaler Suche oder Metaheuristiken kombiniert, um die Lösungsqualität zu verbessern.
- Parallelisierung des Algorithmus für größere Datensätze, um die Laufzeit zu reduzieren.

5 Quellcode

Der Quellcode ist in mehrere Abschnitte unterteilt, um die einzelnen Schritte des Algorithmus klar darzustellen:

5.1 Greedy-Algorithmus

```
def greedy_algorithm(personen):
    gew hlte_strecken = []
    abgedeckte_personen = set()

    strecken_kandidaten = set()
    for min_strecke, max_strecke in personen:
        strecken_kandidaten.add(min_strecke)
        strecken_kandidaten.add(max_strecke)

    for _ in range(3):
        if not strecken_kandidaten:
            break

        beste_strecke = None
        beste_abdeckung = set()

        for strecke in strecken_kandidaten:
            aktuelle_abdeckung = set(teilnehmer_zahlen(strecke, personen))
            neue_abdeckung = aktuelle_abdeckung - abgedeckte_personen

            if len(neue_abdeckung) > len(beste_abdeckung):
                beste_strecke = strecke
                beste_abdeckung = neue_abdeckung

        if beste_strecke is None:
            break

        gew hlte_strecken.append(beste_strecke)
        abgedeckte_personen.update(beste_abdeckung)
        strecken_kandidaten.discard(beste_strecke)

    return gew hlte_strecken, abgedeckte_personen
```

5.2 Teilnehmerzahlen

```
def teilnehmer_zahlen(strecke, personen):
    return [i for i, (min_strecke, max_strecke) in enumerate(personen) if min_strecke <= strecke <= max
```

5.3 Dateiauswahl und Verarbeitung

```
def option_1():
    script_dir = os.path.dirname(os.path.abspath(__file__))
    dateien = [f for f in os.listdir(script_dir) if f.endswith('.txt')]

    if not dateien:
        print("Keine␣Textdateien␣im␣Verzeichnis␣gefunden.")
        return

    print("Verf gbare␣Textdateien:")
    for idx, datei in enumerate(dateien, 1):
        print(f"{idx}.␣{datei}")

    try:
        auswahl = int(input("\nGeben␣Sie␣die␣Nummer␣der␣Datei␣ein,␣die␣verarbeitet␣werden␣soll:␣"))
        if 1 <= auswahl <= len(dateien):
            dateipfad = os.path.join(script_dir, dateien[auswahl - 1])
```

```
            # Startzeit messen
            start_zeit = time.time()

            with open(dateipfad, 'r') as datei:
                zeilen = datei.readlines()
                eintragsanzahl = int(zeilen[0].strip())
                personen = [tuple(map(int, zeile.strip().split())) for zeile in zeilen[1:eintragsanzahl

            print("\nVerarbeite Datei:", dateipfad)
            strecken, teilnehmer = greedy_algorithm(personen)

            print("Gew hlte Streckenl ngen:", strecken)
            print("Teilnehmende Personen:", list(teilnehmer))

            # Endzeit messen und Laufzeit berechnen
            end_zeit = time.time()
            laufzeit = end_zeit - start_zeit
            print(f"\nLaufzeit f r die Verarbeitung dieser Datei: {laufzeit:.4f} Sekunden")

        else:
            print("Ung ltige Auswahl.")
    except ValueError:
        print("Ung ltige Eingabe.")
```