Aufgabe 2: Schwierigkeiten

Team-ID: 00153

Team-Name: ByteBusters

Bearbeiter/-innen dieser Aufgabe:
Nikolas Breins

17. November 2024

Inhaltsverzeichnis

1 Lösungsidee

Ich will einen Graphen bauen welcher dann die Schwierigkeiten der Aufgaben topologisch sortiert. Für das Problem mit den Konflikten dachte ich mir, wenn zwei Aufgaben leichter sind als die jeweils andere, dann sind sie gleich schwer. Dieses Verhalten kann ich dann mit anderen Aufgaben verknüpfen, um die Schwierigkeiten zu bestimmen.

2 Umsetzung

Ich habe einen Graphen gebaut, welcher die Beziehungen zwischen den Aufgaben darstellt. Wenn sich nun ein Zyklus bildet, wird dieser erkannt und die jeweiligen Aufgaben werden zu einer Gruppe zusammengefasst. Diese Gruppen werden dann mit den anderen Aufgaben wieder verknüpft und dann mithilfe einer topologischen Sortierung nach Kahn sortiert. Dann werden die gefragten Aufgaben in der Reihenfolge ausgegeben, wobei auch nur die gefragten aus den Gruppen ausgegeben werden. Dann werden noch alle 'Konflikte', welche bei mir gleich schwer sind, ausgegeben. Hierbei ist wichtig, dass auch mehrere Aufgaben gleich schwer sein können und dies auch erkannt und ausgegeben wird.

3 Beispiele

Wichtig:

Gleich schwere Aufgaben werden immer ausgegeben, falls es welche gibt, auch wenn sie nicht gefragt sind, werden jedoch nicht bei der Reihenfolge der gefragten Aufgaben ausgegeben.

3.1 Beispiel 1

Dies ist die Ausgabe von ‘schwierigkeiten0.txt‘:

```
Zu sortierende Aufgaben: B, C, D, E, F
Eine gute Anordnung wäre: B < E < D < F < C
Gleich schwere Aufgaben:
- Die Aufgaben F, G sind gleich schwierig
```

3.2 Beispiel 2

Dies ist die Ausgabe von ‘schwierigkeiten1.txt‘:

```
Zu sortierende Aufgaben: A, C, D, F, G
Eine gute Anordnung wäre: A < C < G < F < D
Keine Konflikte entdeckt.
```

3.3 Beispiel 3

Dies ist die Ausgabe von ‘schwierigkeiten2.txt‘:

```
Zu sortierende Aufgaben: A, B, D, E, F, G
Eine gute Anordnung wäre: D/E/A/B < F/G
Gleich schwere Aufgaben:
- Die Aufgaben C, D, E, A, B sind gleich schwierig
- Die Aufgaben F, G, H sind gleich schwierig
```

3.4 Beispiel 4

Dies ist die Ausgabe von ‘schwierigkeiten3.txt‘:

```
Zu sortierende Aufgaben: A, B, C, D, E, F, G, H, I, J, K, L, M, N
Eine gute Anordnung wäre: B/C/A/D < H/I/L < M/N < E < J < F < K < G
Gleich schwere Aufgaben:
- Die Aufgaben B, C, A, D sind gleich schwierig
- Die Aufgaben H, I, L sind gleich schwierig
- Die Aufgaben M, N sind gleich schwierig
```

3.5 Beispiel 5

Dies ist die Ausgabe von ‘schwierigkeiten4.txt‘:

```
Zu sortierende Aufgaben: B, W, I, N, F
Eine gute Anordnung wäre: B < I < F < N < W
Gleich schwere Aufgaben:
- Die Aufgaben K, O, M, N sind gleich schwierig
- Die Aufgaben Z, T, V, W, X, Y sind gleich schwierig
- Die Aufgaben I, H, S, G, J sind gleich schwierig
- Die Aufgaben F, P, Q, R sind gleich schwierig
- Die Aufgaben D, E sind gleich schwierig
```

3.6 Beispiel 6

Das Programm fügt keine neue Zeile ein, dies wurde hier nur gemacht, um die Ausgabe besser anzuzeigen.

Dies ist die Ausgabe von ‘schwierigkeiten5.txt‘:

```
Zu sortierende Aufgaben: A, B, C, D, E, F, G, H, I, J, K, L, M,
N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Eine gute Anordnung wäre: H < R < Z < S/C < Q < E < K < N < O/M < J
< L < F < V < P/X/D/U/T/G/A < B < W < I < Y
Gleich schwere Aufgaben:
- Die Aufgaben P, X, D, U, T, G, A sind gleich schwierig
- Die Aufgaben S, C sind gleich schwierig
- Die Aufgaben O, M sind gleich schwierig
```

4 Quellcode

Der wesentliche Teil des Programmes passiert hier:

```
# Funktion zur topologischen Sortierung nach Kahns Algorithmus
def top_sort(graph):
    # In-Degree definieren
    indegree = defaultdict(int)
    # In-Degree berechnen
    for node in graph:
        for neighbor in graph[node]:
            # Inkrementierung des In-Degree des Nachfolgeknotens
            indegree[neighbor] = indegree.get(neighbor, 0) + 1

    # Alle Knoten sollen im In-Degree sein (wenn keine Vorgaenger dann 0)
    indegree.update({node: indegree[node] for node in graph})

    # Alle Knoten mit dem Indegree 0 werden zur queue hinzugefuegt
    queue = deque([node for node in indegree if indegree[node] == 0])

    # Alle sortierten Aufgaben (Endergebnis)
    sorted_nodes = []

    # Bis die Warteschlange am Ende ist
    while queue:
        # Der erste Knoten aus der Warteschlange wird entnommen und anschlie end zu den sortierten Kno
        node = queue.popleft()
        sorted_nodes.append(node)

        # Alle In-Degrees der benachbarten Knoten werden um 1 herabgesetzt
        for neighbor in graph[node]:
            indegree[neighbor] -= 1
            # Falls der In-Degree der Nachbarn nun 0 ist, wird dieser zu den sortierten Knoten hinzugef
            if indegree[neighbor] == 0:
                queue.append(neighbor)

    return sorted_nodes
```

Hier wird der Graph topologisch sortiert und die gefragten Aufgaben ausgegeben.

```python
# Funktion zum Aufbauen des Graphen
def build_graph(exams):
    # Adjazenzliste erstellen
    graph = defaultdict(list)

    # Durch alle Klausuren iterieren
    for exam in exams:
        # Durch alle Aufgaben iterieren
        for i in range(len(exam) - 1):
            # Die Beziehungen finden
            node_from, node_to = exam[i], exam[i + 1]

            # Falls die Beziehung noch nicht existiert
            if node_to not in graph[node_from]:
                graph[node_from].append(node_to)

    # Rueckgabe des Graphen
    return graph

```

Hier wird der Graph aufgebaut und die Beziehungen zwischen den Aufgaben festgelegt. Dieser wird dann später durch andere Funktionen, wie das Finden von Zyklen, um diese zu Gruppen zusammenzufassen, verwendet.

Diese zwei Funktionen sind die wichtigsten Funktionen des Programmes, da sie den Graphen aufbauen bzw. am Ende richtig topologisch sortieren. Anhand der Kommentare sollte es verständlich sein, wie diese Funktionen funktionieren.