Einführung in die Programmierung mit C++

Übungsblatt 9

Klassen und Rekursion

Sebastian Christodoulou, Alexander Fleming und Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

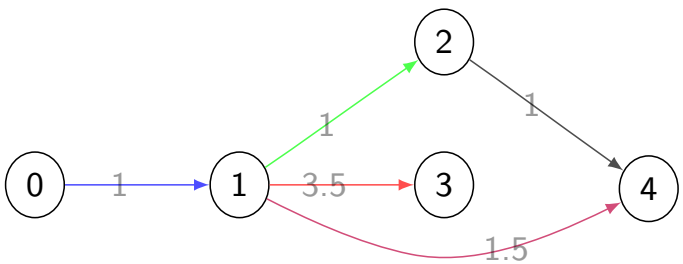
RWTH Aachen

Gegeben ist ein Graph. Anders als in voherigen Übungen haben dessen Kanten nun Kantengewichte. Diese repräsentieren die *Kosten*, sich entlang der Kante zu bewegen. Nicht-existente Kanten (0 in der Adjazenzmatrix) haben Kosten ∞

`my_graph_1.txt`

```
5
5
0 1 1
1 2 1
1 3 3.5
2 4 1
1 4 1.5
```

repräsentierter Graph



Adjazenzmatrix

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3.5 & 1.5 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Der erste Wert in `my_graph_1.txt` gibt die Anzahl der Knoten an, der zweite Wert Anzahl der Kanten. Die folgenden Zeilen beschreiben eine Kante durch `<Anfangsknoten> <Endknoten> <Kantengewicht>`

Uns interessiert die Sequenz an Kanten mit denen man von einem zum anderen Knoten gelangt, sodass die summierten Kantengewichte minimal sind. Wir nennen dies den kürzesten Pfad. Der kürzeste Pfad von Knoten 0 bis 4 kostet $1+1.5 = 2.5$

Zunächst wollen wir aber den Graphen zu einer Klasse machen. Diese hat

- die Adjazenzmatrix `std::vector<std::vector<double>> adjacency_matrix`, die wie auf Folie 2 die Kanten durch deren Gewichte repräsentiert. Der Wert 0 bedeutet darin, dass die Kante nicht vorhanden ist
- einen Konstruktor `Graph(int n)`, der eine mit 0 gefüllte Adjazenzmatrix für n Knoten initialisiert

Und folgende zu implementierende Funktionen

- eine Funktion `void add_edge(int i, int j, double weight)`, die eine Kante $i \to j$ mit Kantengewicht `weight` in die Adjazenzmatrix einträgt
- eine Funktion `double cost(int i, int j)` , die das Kantengewicht von $i \to j$ ausgibt. Hat die Adjazenzmatrix hier den Eintrag 0, ist das Gewicht ∞.
- eine Funktion `std::vector<int> reachable_nodes(int i)`, die in der Adjazenzmatrix nachschaut, welche anderen Knoten von Knoten i aus durch Kanten erreichbar sind. Diese Knoten werden in einem `std::vector` ausgegeben.

1. Vervollständige die oben beschriebenen Funktionen in der Klasse `Graph`
   - `void add_edge(int i, int j, double weight)`
   - `double cost(int i, int j)`
   - `std::vector<int> reachable_nodes(int i)`
2. Lese den Graph aus `my_graph_1.txt` ein, und initialisiere ein Objekt der Klasse `Graph`, die dessen Adjazenzmatrix hat.
   - Benutze dafür den Konstruktor `Graph(int n)` und `void add_edge(int i, int j, double weight)`
3. Zu diesem Zeitpunkt kann man testen ob die Implementierung funktioniert: Welche Knoten sind von Knoten 1 erreichbar? Zu welchen Kosten? Gib dies in der Kommandozeile aus.

4. Die Funktion `double shortest_path(const Graph& g, int i, int j)` soll implementiert werden. Durch Rekursion berechnet sie die Kosten des kürzesten Pfades von i nach j. Die Rekursion hat drei Fälle:
   - Base-Case 1 `i==j` . Hier ist die Rekursion am Ziel angekommen. Verbleibende Kosten von $i \to i$ sind 0. Deshalb wird 0 zurückgegeben.
   - Base-Case 2. Von i aus gibt es auf `g` keine erreichbaren Knoten. Auch hier endet die Rekursion, und gibt Kosten ∞ zurück.
   - Recursive Case. Hier werden alle erreichbaren Knoten $\{k_1, \ldots, k_p\}$ von Knoten i aus inspiziert:
     - Betrachte jedes k_q, $q \in \{1, \ldots, p\}$. Die Kosten von i zum Ziel j sind die Kosten der Kante $i \to k_q$ plus die Kosten des kürzesten Pfads von Knoten k_q nach j (rekursiver Aufruf).
     - Nachdem jedes erreichbare k_q, $q \in \{1, \ldots, p\}$ so betrachtet wurde, gib die minimalen betrachteten Kosten zurück.
5. Was sind die Kosten des kürzesten Pfads von Knoten 0 nach 4 in `my_graph_1.txt`? Zeige dies durch den Funktionsaufruf.
6. Erstelle nun auch ein Objekt der Klasse `Graph` aus der Textdatei `my_graph_2.txt`. Was sind dessen Kosten des kürzesten Pfads von 0 zu 8?

Abgabe:

- 9.cpp