

1. Übungsblatt

Jasper Gude

Pia Röttgers

23. April 2026

/ 20

Aufgabe 1 – Spannbäume & Breitensuche

Sei $G = (V, E)$ ein zusammenhängender Graph mit Kantengewichten $w : E \rightarrow \mathbb{N}$ und $s \in V$ ein ausgezeichnete Knoten.

- a) Wenn $w(e) = 1$ für alle $e \in E$, dann ist der Breitensuchbaum mit Quelle s ein minimaler Spannbaum.

Die Breitensuche berechnet in diesem Fall den kürzesten Weg von jedem Knoten zum Knoten s , also den Breitensuchbaum. Dieser spannt also einen minimalen Spannbaum auf.

/ 2

- b) Wenn $w(e) = 1$ für alle $e \in E$, dann ist jeder minimale Spannbaum von G ein Breitensuchbaum mit Quelle s .

Falsch, siehe Abbildung 1.

/ 2

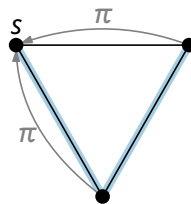


Abbildung 1: π -Zeiger des Breitensuchbaums und MSB blau hinterlegt.

- c) Wenn $w(e) \in \{1, 2, 3\}$ für alle $e \in E$, dann ist jeder minimale Spannbaum von G ein Tiefensuchbaum mit Quelle s .

Falsch, siehe Abbildung 2. Der Minimale Spannbaum kann kein Tiefensuchbaum sein.

/ 2

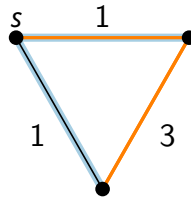


Abbildung 2: Tiefensuchbaum in orange und MSB blau hinterlegt.

Aufgabe 2 – Kreissuche

- a) Wähle Startknoten s und füge ihn in eine neue Queue Q ein.

Nimm den vordersten Knoten u aus Q und füge seine noch nicht entdeckten (weißen) Nachbarn ein und färbe sie grau. Wenn einer der Nachbarn schon entdeckt wurde, also grau ist, gibt es einen Kreis in G . Markiere den Knoten u als abgeschlossen (black) und entferne ihn aus Q .

Wiederhole den Schritt solange bis Q leer ist.

```

EINFACHERKREIS(Graph G, Vertex s)
  INITIALIZE(G, s) // So wie in der Breitensuche
  Q ← new QUEUE()
  Q.ENQUEUE(s)
  while Q ≠ ∅ do
    u ← Q.DEQUEUE()
    u.color ← gray
    foreach v ∈ Adj[u] do
      // Füge alle noch nicht entdeckten Knoten ein
      if v.color = white then
        v.color ← gray
        Q.ENQUEUE(v)
      // Wenn ein Knoten schon entdeckt wurde, gibt es einen Kreis
      else return true
    u.color ← black
  return false

```

/ 4

- b) Dadurch, dass wir nur Knoten einfügen, die noch nicht entdeckt wurden, können wir nie auf dem selben Pfad zu einem Knoten kommen. Das heißt, wenn wir einen schon entdeckten Knoten finden, haben wir einen Kreis im Graphen gefunden.

Der Algorithmus kann aufgrund der Struktur des Graphens (kein Multigraph, keine Selbstkanten) nur Kreise der Länge mindestens 3 finden.

Jeder Knoten wird nur einmal in die Queue eingefügt und nur einmal herausgenommen. Somit läuft der Algorithmus in $\mathcal{O}(|V|)$.

/ 2

- c) Solange es weiße Knoten im Graphen gibt, wählen wir einen neuen Startknoten für diese Zusammenhangskomponente. Als Ausgabe geben wir ein Array von Tupeln (s_i, c_i) zurück, wobei s_i der Startknoten einer Zusammenhangskomponente und $c_i \in \{\text{true}, \text{false}\}$ der Wahrheitswert, ob ein Kreis in der Komponente existiert.

/ 1

Aufgabe 3 – Eulerwege

Sei $G = (V, E)$ ein ungerichteter, zusammenhängender Graph. Dann gilt: G hat genau dann einen Eulerweg, wenn die Anzahl an Knoten $v \in V$, für die gilt, dass $\deg(v)$ ungerade ist, genau 0 oder 2 ist.

- ⇐ 1. Fall: 0 Knoten mit ungeradem Grad. Nach dem Satz in der Vorlesung gibt es einen Eulerkreis. Im Eulerweg sind also Start- und Endknoten identisch.
2. Fall: 2 Knoten mit ungeradem Grad. Die beiden Knoten bilden den Start- und Endknoten des Eulerwegs. Die Kante die den Eulerkreis schließen würde braucht genau zwei Knoten, zu denen sie inzident ist. Nehmen wir diese Kante weg, ergibt sich eine ungerader Grad an diesen beiden Knoten.
- ⇒ Ein Graph mit ungerader Anzahl an Knoten mit ungeradem Grad kann nicht existieren, da die Summe aller Knoten mit ungeradem Grad gerade ist.
- Für alle anderen Fälle gilt, wenn ein Knoten ungeraden Grad hat, dann gibt es keinen Weg aus dem Knoten heraus, wenn man hineingelaufen ist.

/ 4

Aufgabe 4 – Graphmodellierung

Sie betreuen ein Projekt, das sich aus vielen vordefinierten Aufgaben zusammensetzt. Manche Aufgaben können erst erledigt werden, wenn bestimmte andere Aufgaben abgeschlossen sind. Für jede Aufgabe ist vorher genau bekannt, von welchen Aufgaben sie abhängt. Ihr Projektteam kann immer nur eine Aufgabe gleichzeitig bearbeiten und eine angefangene Aufgabe wird immer abgeschlossen bevor eine neue Aufgabe begonnen werden kann.

Wir modellieren das Problem als gerichteten Graph. Die Aufgaben sind Knoten. Jede Aufgabe hat Kanten zu den Aufgaben, die von ihr abhängen.

Eine Reihenfolge können wir mithilfe einer Topologischen Sortierung finden.

/ 3
