

2. Übungsblatt zu Algorithmen II im WS 2022/2023

http://algo2.iti.kit.edu/AlgorithmenII_WS22.php
 {sanders, moritz.laupichler, hans-peter.lehmann}@kit.edu

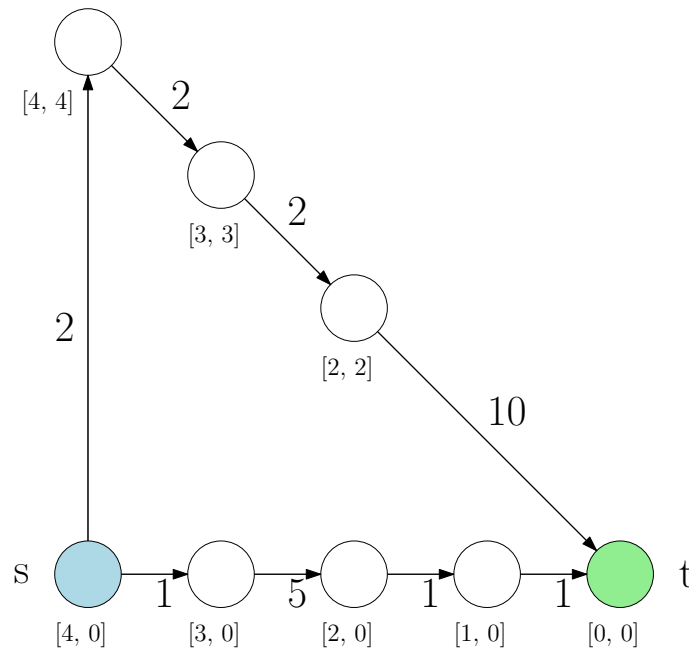
Aufgabe 1 (Rechnen: A* Suche)

Gegeben sei der unten abgebildete Graph. An den Kanten sind Kosten für die Nutzung der Verbindung eingetragen und die Knoten tragen Ortskoordinaten.

- a) Ergänzen Sie den gegebenen Graphen um Knotenpotentiale für eine A* Suche von s nach t . Verwenden Sie Knoten t als Landmarke und die Manhattan-Distanz ($\hat{=}$ Einsnorm $\|\cdot\|_1$) als Abschätzung für die Entfernung zum Ziel.

Hinweis: $\|\cdot\|_1 : \|(x_1, y_1), (x_2, y_2)\|_1 = y_2 - y_1 + x_2 - x_1$.

- b) Tragen Sie die reduzierten Kantengewichte in den Graphen ein.
- c) Wieviele `deleteMin` Operationen führt die A* Suche auf dem Graphen aus? Wieviele eine normale Suche mit Dijkstra's Algorithmus?



Aufgabe 2 (Einführung+Analyse: Bidirektionaler Dijkstra)

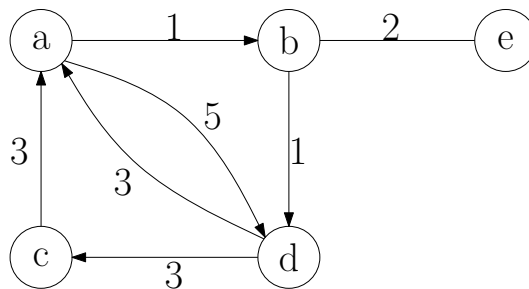
Gegeben sei – wie üblich – ein gerichteter Graph $G = (V, E)$ mit $|V| = n$ und $|E| = m$, sowie eine Kantengewichtungsfunktion $c : E \rightarrow \mathbb{R}_0^+$. Gesucht ist der kürzeste Pfad $p = \langle s, \dots, t \rangle$ zwischen zwei Punkten $s, t \in V$.

Eine bidirektionale Suche löst dieses Problem wie folgt: Es werden zwei unidirektionale Suchen mit Dijkstra's Algorithmus gestartet. Die *Vorwärtssuche* beginnt bei Knoten s und operiert auf dem normalen Graphen G , auch *Vorwärtsgraph* genannt. Die *Rückwärtssuche* beginnt bei Knoten t und operiert auf dem *Rückwärtsgraph* $G^r = (V, E^r)$ mit Kantengewichtungsfunktion c^r . Dieser Graph entsteht aus G durch Umkehrung aller Kanten. Der Algorithmus scannt abwechselnd einen Knoten in der Vorwärtssuche und in der Rückwärtssuche, beginnend mit der Vorwärtssuche.

Wird während des Scans von Knoten u Kante (u, v) relaxiert, so wird überprüft, ob die Distanz $d_{\text{forward}}[v] + d_{\text{backward}}[v]$ kleiner ist als die momentan minimale gefundene Distanz von s nach t und diese gegebenenfalls angepasst ($d_{\text{forward}}[v]$ gibt die bisher kürzeste gefundene Distanz von s nach v in der Vorwärtssuche und $d_{\text{backward}}[v]$ die bisher kürzeste gefundene Distanz von v nach t in der Rückwärtssuche an).

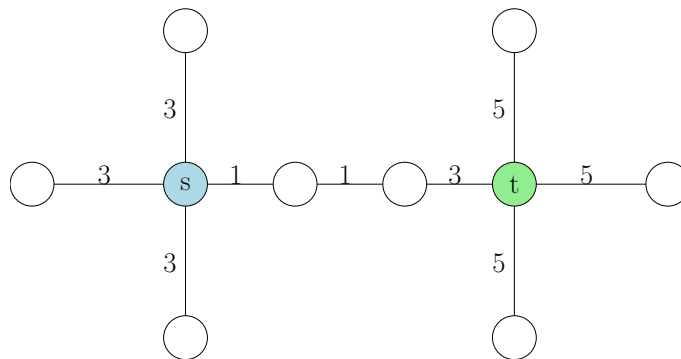
Sobald ein Knoten in einer Richtung gescannt werden soll, der bereits in der anderen Richtung gescannt worden ist, kann die Suche beendet werden (*Abbruchbedingung*). Die aktuelle minimale gefundene Distanz ist dann die tatsächliche minimale Distanz zwischen s und t .

- a) Zeichnen Sie den Rückwärtsgraph G^r zum angegebenen Graphen. Geben Sie die Kantengewichte $c(a, d)$, $c^r(a, d)$ sowie $c(b, e)$, $c^r(b, e)$ an.



(Kante (b, e) ist eine bidirektionale [bzw. ungerichtete] Kante)

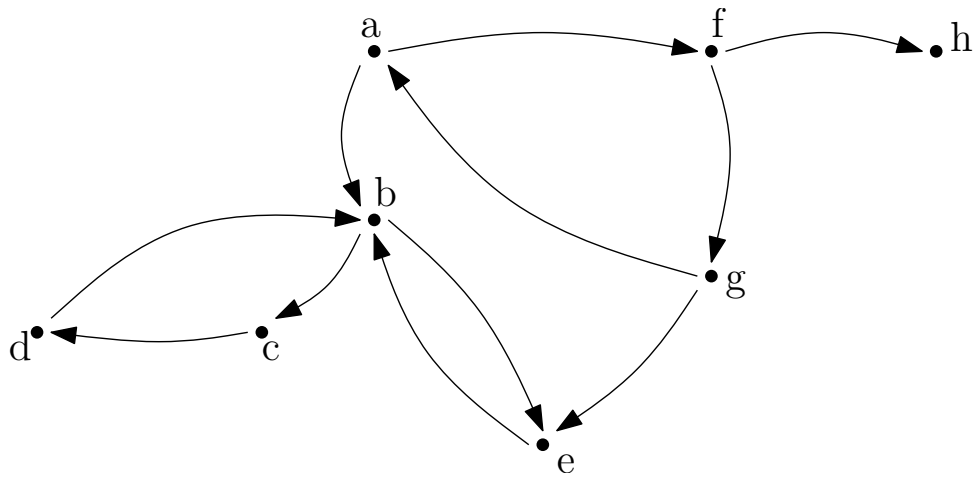
- b) Geben Sie an, in welcher Reihenfolge der unten angegebene Graph durchlaufen wird.



- c) Zeigen Sie, dass die Abbruchbedingung korrekt ist.
 d) Wann kann es passieren, dass die Suche nach dem Scan von Knoten u beendet wird, dieser aber nicht Teil des kürzesten Weges ist? Geben Sie ein Beispiel an.

Aufgabe 3 (Rechnen: SCC mit Tiefensuche)

Gegeben sei folgender Graph $G = (V, E)$:



Führen Sie den Algorithmus zur Bestimmung aller starken Zusammenhangskomponenten aus der Vorlesung auf dem Graph G aus. Geben Sie nach jedem Schritt den Zustand von `oReps`, `oNodes` und `component` an.

Aufgabe 4 (Analyse+Entwurf: Artikulationspunkte)

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph. Ein Knoten v des Graphen G wird als *Gelenkpunkt* bezeichnet, wenn dessen Entfernen die Zahl der Zusammenhangskomponenten erhöht.

- Zeigen Sie, dass es in jedem Graphen G ohne Gelenkpunkte und mit $|V| \geq 3$ immer mindestens ein Knotenpaar (i, j) , $i, j \in V$, $i \neq j$ gibt, so dass zwei Pfade $P_1 = \langle i, \dots, j \rangle$ und $P_2 = \langle i, \dots, j \rangle$ existieren, die bis auf die Endpunkte knotendisjunkt sind, d.h.: $P_1 \cap P_2 = \{i, j\}$.
- Beweisen Sie in jedem Graphen G mit Gelenkpunkten die Existenz eines Knotens v , für den gilt: Man kann einen Knoten w entfernen, so dass es von v aus keine Pfade mehr zu mindestens der Hälfte der verbleibenden Knoten gibt.
- Zeigen Sie, dass in einem zusammenhängenden Graphen $G = (V, E)$ stets ein Knoten v existiert, so dass G nach Entfernen von v weiterhin zusammenhängend ist.
- Vervollständigen Sie den angegebenen allgemeinen DFS-Algorithmus, so dass er in $O(|V| + |E|)$ alle Gelenkpunkte eines ungerichteten Graphen berechnet. Geben Sie an, was die Funktionen `init`, `root(s)`, `traverseTreeEdge(v,w)`, `traverseNonTreeEdge(v,w)` und `backTrack(u,v)` machen.
Überlegen Sie sich zunächst, wie Sie mit Hilfe der DFS-Nummerierung Gelenkpunkte erkennen können.

Depth-first search of graph $G = (V, E)$

unmark all nodes

init

for all $s \in V$ **do**

if s is not marked **then**

 mark s

root(s)

 DFS(s,s)

end if

end for

procedure DFS(u,v : NodeID)

for all $(v, w) \in E$ **do**

if w is marked **then**

traverseNonTreeEdge(v,w)

else

traverseTreeEdge(v,w)

 mark w

 DFS(v,w)

end if

end for

backtrack(u,v)

end procedure

Aufgabe 5 (Kleinaufgaben: Eigenschaften von Flüssen)

a) Nach Vorlesung ist eine gültige Distanzfunktion $d(\cdot)$ für *Dinitz Algorithmus* gegeben durch:

- $d(t) = 0$
- $d(u) \leq d(v) + 1 \quad \forall (u, v) \in G_f$

Zeigen Sie, falls $d(s) \geq n$, existiert kein *augmentierender Pfad*.

b) In der Vorlesung wurde gezeigt, dass die Laufzeit von *Dinitz Algorithmus* für Graphen mit Kantengewichten gleich 1 (*unit edgeweights*) in $O((n + m)\sqrt{m})$ liegt. Vergleichen Sie diese Laufzeit zum *Ford Fulkerson Algorithmus*. Für welche Graphen mit *unit edgeweights* ist welcher der beiden Algorithmen schneller?

c) Sei $G = (V, E)$ ein gerichteter Graph, in dem maximale Flüsse berechnet werden sollen. Sei $e = (i, j) \in E$ ebenso wie $e' = (j, i) \in E$, d. h. G besitzt ein Paar entgegengesetzter Kanten. Außerdem sei $c(e) \geq c(e')$. Widerlegen Sie durch ein Gegenbeispiel:

Entfernt man e' aus E und reduziert $c(e) := c(e) - c(e')$, ändert sich der maximale Fluss nicht, d. h. man kann entgegengesetzte Kanten a-priori (für beliebige s und t) gegeneinander aufrechnen.

Aufgabe 6 (Rechnen: Segmentierung mit Flüssen)

Wir betrachten einen einfachen Fall für Bildbearbeitung. Die Vorder-/Hintergrundsegmentierung. Das Ziel dieses Prozesses ist es, ein Bild in Vorder und Hintergrund zu zerlegen. Die Transformation dafür weißt jedem Pixel $p_{i,j}$ des Bildes einen Knoten $v_{i,j}$ im Graphen zu. Für jedes Paar von benachbarten Pixeln $p_{i,j}$ und $p_{k,l}$ ($|i - k| + |j - l| = 1$) fügen wir eine ungerichtete Kante $(v_{i,j}, v_{k,l})$ ein. Zusätzlich fügen wir je einen Knoten s für Vordergrund (Quelle) und einen Knoten t für Hintergrund (Senke) ein. Von Knoten s existiert eine gerichtete Kante zu jedem Knoten $p_{i,j}$ und von jedem Knoten $p_{i,j}$ existiert eine gerichtete Kante zu Knoten t . Wir definieren darüber hinaus folgende Kantengewichte:

$$c(e = (u, v)) = \begin{cases} p_v(v) & u = s \\ p_h(u) & v = t \\ f(u, v) & \text{sonst} \end{cases}$$

Wobei mit $p_v(x)$ die Wahrscheinlichkeit gegeben ist, dass x Vordergrundknoten ist, mit $p_h(x)$ die Wahrscheinlichkeit für einen Hintergrundknoten und mit $f(x, y)$ eine Penaltyfunktion für das Trennen der beiden Knoten x und y . Für ein Graustufenbild B definieren wir

$$p_v(x, y) = B[x, y]^2, \quad p_h(x, y) = (4 - B[x, y])^2 \quad \text{sowie} \quad f((x_1, y_1), (x_2, y_2)) = (4 - |B[x_1, y_1] - B[x_2, y_2]|)^2.$$

Hinweis: Diese Modellierung ist nur ein Beispiel und keine allgemeingültige Modellierung. Sie soll nur verdeutlichen, wie Flow-Algorithmen für andere Probleme eingesetzt werden können.

- Geben Sie den Flussgraphen für das unten angegebene Graustufenbild an.
- Führen Sie einen augmenting Path Algorithmus auf dem entstandenen Graphen aus.
- Wie würde die Segmentierung in Vorder- und Hintergrund im Bild als Ergebnis aussehen?

4	4	1
4	2	0
0	0	0

Aufgabe 7 (Analyse+Entwurf+Rechnen: Grenzüberwachung)

Eine (eindimensionale) Grenzlinie soll durch ein Sensornetz überwacht werden. Zu diesem Zweck wurde eine große Anzahl an Sensorknoten unregelmäßig an der Grenze ausgebracht. Jeder Knoten kann einen Bereich der Grenze für eine gewisse Zeit proportional zu seiner Batteriekapazität überwachen. Die Grenze gilt als vollständig gesichert, wenn jeder Abschnitt der Grenzlinie von mindestens einem Sensorknoten abgedeckt ist. Aufgrund der großen Menge an Knoten sind ihre Überwachungsbereiche stark überlappend. Daher müssen nicht immer alle Knoten aktiv sein, um eine vollständige Sicherung der Grenze zu gewährleisten. So kann Energie gespart werden und die maximale Dauer der Grenzsicherung erhöht werden.

Durch die unregelmäßige Anbringung der Knoten und durch große Fertigungstoleranzen in der Batteriekapazität und dem Überwachungsbereich (*man hat unbedingt beim billigsten Hersteller einkaufen müssen...*) ist zunächst nicht klar, wie lange die Grenze maximal vollständig gesichert werden kann. Glücklicherweise wurden die Positionen der Knoten und ihre jeweiligen Kapazitäten und Detektionsbereiche protokolliert und können verwendet werden, um diese Frage zu beantworten.

- a) In der Vorlesung haben Sie Flussprobleme mit beschränkten Kantenkapazitäten $c(e)$ kennengelernt. Ebenso können Flussprobleme mit beschränkten Knotenkapazitäten $c(v)$ sinnvoll sein. In diesem Fall darf für einen gültigen Fluss die Summe der in den Knoten ankommenden bzw. ausgehenden Flüsse die Kapazität des Knotens nicht überschreiten. Außerdem muss wie bisher für jeden Knoten (außer der Quelle und Senke) die Summe der ankommenden Flüsse gleich der Summe der ausgehenden Flüsse sein.

Erklären Sie, wie maximale Flüsse mit Knotenkapazitäten berechnet werden können. Begründen Sie kurz, warum Ihr Ansatz einen zulässigen und optimalen Fluss berechnet.

- b) Konstruieren Sie ein Flussnetzwerk, das das oben beschriebene Problem der Bestimmung einer maximalen Dauer für die vollständige Grenzüberwachung lösen kann.

Hinweis: Jeder Knoten entspricht einem Sensorknoten. Batteriekapazität kann als äquivalent zur Flussmenge betrachtet werden.

- c) Erstellen Sie ein Flussnetz, das dem folgenden Sensornetz entspricht. Wie lange kann dieses Netz die Grenze im Bereich $[0, 13]$ überwachen? Welche Sensorknoten müssen wann aktiv sein?

Format der Angaben: $x_{nodeID} = \{[begin_range, end_range], capacity\}$

- $x_1 = \{[0, 5], 4\}$
- $x_2 = \{[0, 7], 3\}$
- $x_3 = \{[4, 9], 2\}$
- $x_4 = \{[3, 8], 5\}$
- $x_5 = \{[8, 13], 5\}$
- $x_6 = \{[7, 11], 3\}$
- $x_7 = \{[11, 15], 2\}$

Hinweis: Bevor Sie langwierig einen maximalen Fluss berechnen, versuchen Sie ihn durch *scharfes Hinschauen* zu bestimmen.

Ausgabe: 22.11.2022

Besprechung: 06.12.2022