Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Dr. Thomas Worsch, Dr. Simon Gog
Demian Hespe, Yaroslav Akhremstev

6. Übungsblatt zu Algorithmen II im WS 2017/2018

http://algo2.iti.kit.edu/AlgorithmenII_WS17.php
{hespe,sanders,simon.gog,worsch,yaroslav.akhremtsev}@kit.edu

Aufgabe 1 (*Analyse: Kleinaufgaben*)

a) Geben Sie die wesentlichen Unterschiede –laut Vorlesung– zwischen einer (normalen) *Priority Queue* und einer adressierbaren *Priority Queue* an.

b) Vergleichen Sie die Laufzeit einer `merge`-Operation für *Pairing Heaps* und *Binary Heaps*.

Aufgabe 2 (*Analyse: Laufzeitverhalten*)

a) Beweisen Sie allgemein für adressierbare *Priority Queues* die untere Laufzeitschranke von $\Omega(\log n)$ für `deleteMin` unter der Voraussetzung, dass `insert` konstante Laufzeit benötigt.

b) Warum muss diese untere Laufzeitschranke nicht gelten, wenn `insert` mehr Zeit benötigen darf?

Aufgabe 3 (*Analyse:* best-case *Verhalten*)

a) Geben Sie einen Zustand eines *Fibonacci Heaps* an, für den die nächsten n `deleteMin`-Operationen jeweils konstante Laufzeit benötigen (nicht amortisiert). Begründen Sie Ihre Antwort. Gehen Sie davon aus, dass zwischen den `deleteMin`-Operationen keine anderen Operationen ausgeführt werden.

b) Geben Sie einen Algorithmus an, welcher den von Ihnen angegebene Zustand für beliebige n erzeugt. Beweisen Sie die Korrektheit des Algorithmus.

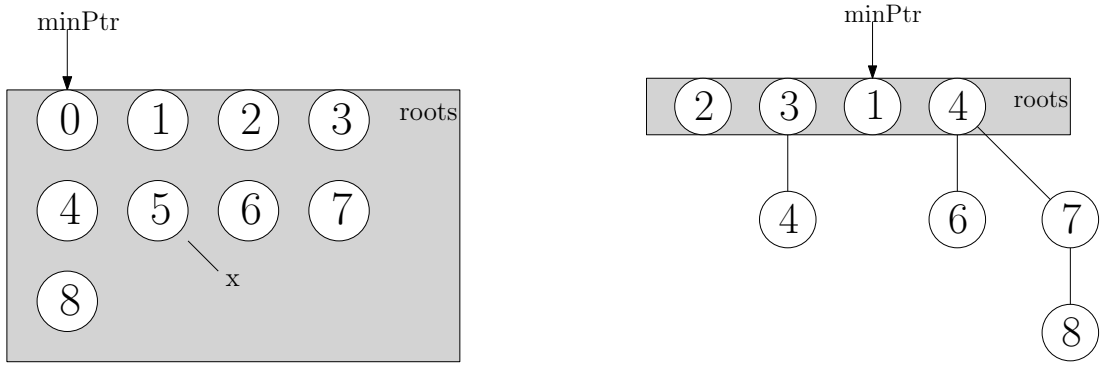
Aufgabe 4 (*Rechnen:* Fibonacci Heaps)

Gegeben sei ein *Fibonacci Heap* mit unten eingezeichnetem Zustand (a).

a) Geben Sie eine möglichst kurze Folge von Operationen an, die diesen Zustand erzeugt.

b) Führen Sie anschließend die Operationen `deleteMin()` auf dem Heap aus. Zeichnen Sie den Zustand des Heaps nach jedem Einfügen eines Baum in ein leeres Bucket und nach jeder Union-Operation.

c) Geben Sie eine möglichst kurze Folge von Operationen an, die den unten eingezeichneten Zustand (b) erzeugt. Tipp: der eingezeichnete Zustand lässt sich aus dem Heap in Abbildung (a) nach der `deleteMin`-Operation durch weitere Operationen erzeugen.



Aufgabe 5 (*Entwurf: Datenstrukturen*)

a) Erweitern Sie die Datenstruktur *Pairing Heap* um die Operation `increaseKey(h: Handle, k: Key)`. Ihre Operation sollte amortisiert $O(\log n)$ Laufzeit benötigen. Geben Sie Pseudocode an. Wie würden Sie bei einem *Binary Heap* vorgehen?

b) Entwerfen Sie eine Datenstruktur welche die Operationen `insert` in $O(\log n)$, Median bestimmen in $O(1)$ und Median entfernen in $O(\log n)$ unterstützt. Eine Beschreibung in Worten ist ausreichend.

Aufgabe 6 (*Entwurf: Anwendung*)

Für ein großes –und wir meinen ein wirklich großes– Fest sind Sie für die Bar zuständig. Für diese Aufgabe haben Sie Ihren eigenen Barroboter entworfen, der automatisch wunderbare Cocktails mischen kann. Die Zutaten dafür werden in großen Kanistern bereitgestellt. Doch genau hier ist das Problem: In all der Hektik des Abends müssen Sie darauf achten, dass kein Kanister leerläuft. Sie wollen den Abend allerdings auch so gut wie möglich genießen und nicht andauernd die Kanister überprüfen. Um dieses Problem zu umgehen, gibt es nur eine Lösung: Eine Nachfüllanzeige muss her! Leider gab es nur noch Anzeigen, die es erlauben eine einzelne Zeile darzustellen.
Es ist klar, dass auf der Anzeige die am dringensten benötigte Zutat angezeigt werden sollte. Ziel ist es also einen Algorithmus zu entwerfen, der die Anzeige immer aktuell hält.

a) Überlegen Sie sich, welche Datenstruktur Sie als Grundlage für Ihren Algorithmus verwenden wollen, um ihn effizient implementieren zu können. Sie können davon ausgehen, dass die für einen Cocktail benötigten unterschiedlichen Zutaten wesentlich weniger sind als die Gesamtmenge an vorhandenen unterschiedlichen Zutaten.

b) Entwerfen Sie eine Funktion `MixDrink( recipe )`, die auf Ihrer Datenstruktur operiert. Geben Sie Pseudocode an. Sie müssen dabei nur Ihre Datenstruktur aktualisieren. Sonstige Funktionen des Roboters müssen Sie nicht berücksichtigen.

c) Wenn ein Kanister gewechselt wird, muss ihre Datenbasis natürlich auch aktualisiert werden. Beschreiben Sie, welche Auswirkungen das Wechseln auf Ihre Datenstruktur hat.

Aufgabe 7 (*Kleinaufgaben: A* Suche*)

a) Sei pot$(\cdot)$ eine gültige Potentialfunktion für die A* Suche nach Knoten t in Graph $G(V,\ E)$. Überprüfen Sie, ob

$$\text{pot}^c = \text{pot} + c,\ c = const.$$

ebenfalls eine gültige Potentialfunktion darstellt.

b) Kann es vorkommen, dass eine A* Suche mehr Knoten absucht als eine Suche mit Dijkstra's Algorithmus für die gleiche Anfrage? Begründen Sie warum nicht oder geben Sie ein Beispiel an.

Aufgabe 8 (*Rechnen: Monotone ganzzahlige* Priority Queues)

Bei einer Ausführung von *Dijkstra's Algorithmus* wird folgender Ausschnitt an *Priority Queue* Operationen protokolliert:

...

- `insert(a, 06 [00110] )` (Parameter: Knotenbezeichnung, Distanz [Distanz binär])
- `insert(b, 10 [01010] )`
- `insert(c, 07 [00111] )`
- `deleteMin()`
- `deleteMin()`
- `insert(d, 12 [01100] )`
- `deleteMin()`
- `insert(e, 16 [10000] )`

...

Zusätzlich wissen Sie, dass das maximale Kantengewicht im Graphen $C = 6$ beträgt und dass vor der ersten protokollierten Operation das letzte enthaltene Element aus der *Priority Queue* entfernt wurde. Dieses hatte den Wert $min = 5$.

a) Führen Sie die Operationen auf einer *Bucket Queue* aus. Geben Sie den Zustand der Datenstruktur nach jeder Operation an.

b) Wieviele *Buckets* werden für eine Ausführung auf einem *Radix Heap* benötigt? Führen Sie die Operationen auf einem *Radix Heap* aus. Geben Sie den Zustand der Datenstruktur und den Wertebereich der *Buckets* nach jeder Operation an.

Aufgabe 9 (*Analyse: Laufzeit von Dijkstra's Algorithmus*)

Gegeben sei ein gerichteter Graph $G = (V,\ E)$ mit $|V| = n$ und $|E| = m$, sowie eine Kantengewichtungsfunktion $c : E \to \mathbb{R}_0^+$.

a) Eine spezielle *Priority Queue* habe folgende Laufzeiteigenschaften:

- `insert`: $O(\log n)$
- `decreaseKey`: $O(1)$
- `deleteMin`: $O(\sqrt{m})$

(ob eine Datenstruktur mit diesen Eigenschaften existiert und Dijkstra's Algorithmus mit ihr korrekt arbeitet, ist eine andere Frage, aber wir nehmen für diese Aufgabe an es ginge :-))

Geben Sie eine kleinste obere Schranke für die Laufzeit von Dijkstra's Algorithmus unter Verwendung dieser *Priority Queue* an. Unter welcher Bedingung an das Verhältnis der Anzahl Knoten n zu Kanten m wird die Laufzeit linear in der Eingabegröße? Die Eingabe erfolgt in Form einer Adjazenzliste.

b) Geben Sie eine Klasse von Graphen an, für welche die Anzahl an `deleteMin` Operationen in Dijkstra's Algorithmus von einem beliebigen Knoten s zu einem beliebigen erreichbaren Knoten t linear von der minimalen Pfadlänge $\mu(s, t)$ abhängt. Für die Klasse von Graphen muss weiter $m = \Theta(n \log n)$ gelten.

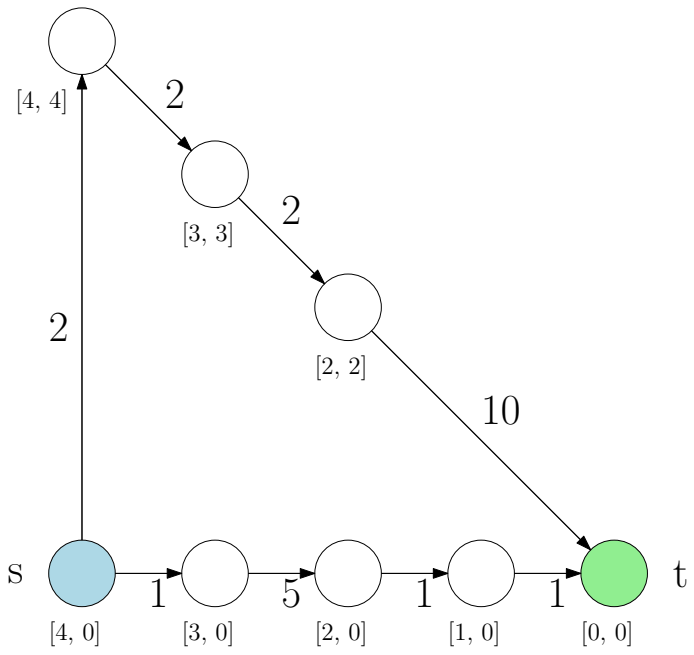
Aufgabe 10 (*Rechnen: A* Suche*)

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 Manhatten-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?

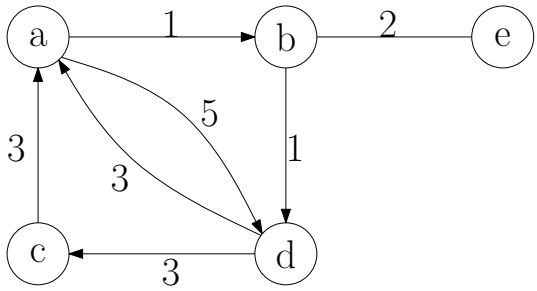


Aufgabe 11 (*Einführung+Analyse: Bidirektionaler Dijkstra*)

In Vorlesung wurde eine bidirektionale Variante von Dijkstra's Algorithmus angesprochen, die in dieser Aufgabe näher untersucht werden soll.

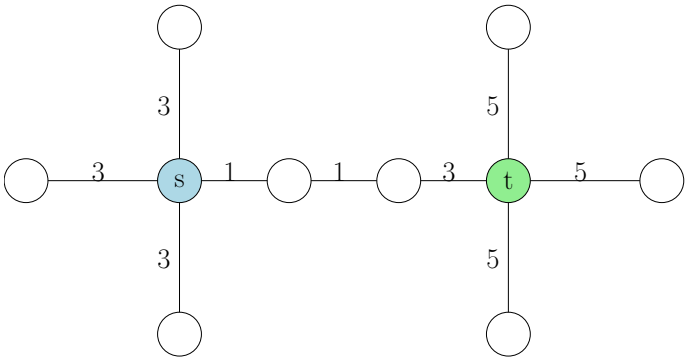
Zur Wiederholung:
Gegeben sei –wie üblich– ein gerichteter Graph $G = (V,\ E)$ mit $|V| = n$ und $|E| = m$, sowie eine Kantengewichtungsfunktion $c : E \to \mathbb{R}_0^+$. Gesucht ist der kürzeste Pfad $p = \langle s, \ldots, t\rangle$ zwischen zwei Punkten $s,\ t \in V$.
Eine bidirektionale Suche löst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

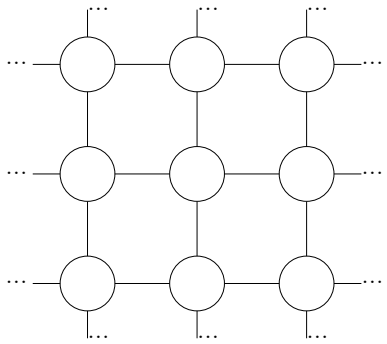


c) Zeigen Sie, dass die Abbruchbedingung korrekt ist.

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 12 (*Analyse: Bidirektionaler Dijkstra*)

a) Gegeben sei ein Gittergraph G mit allen Kantengewichten gleich 1. Wieviele Knoten wird eine bidirektionale Suche besuchen in Abhängigkeit von Abstand d zwischen Start und Ziel? Wieviele die unidirektionale Suche?



Beispiel eines Gittergraphen

b) Geben Sie ein Beispiel an, in dem die bidirektionale Suche von s nach t exponentiell weniger Knoten besucht als die unidirektionale Suche.

c) Geben Sie ein Beispiel, in dem die bidirektionale Suche von s nach t mehr Knoten besucht als die unidirektionale Suche.

d) Zeigen Sie, dass die bidirektionale Suche nie mehr als doppelt so viele Knoten besucht als die unidirektionale Suche.

Aufgabe 13 (*Analyse: Äquivalenzrelation*)

Gegeben sei ein gerichteter Graph $G = (V, E)$ und folgende Relation

$$v \overset{*}{\longleftrightarrow} w \qquad \text{gdw.} \qquad \text{es gibt Pfade } \langle v, \ldots, w\rangle \text{ und } \langle w, \ldots, v\rangle \text{ in } G$$

für $v, w \in V$.
Zeigen Sie, dass $\overset{*}{\longleftrightarrow}$ eine Äquivalenzrelation ist.

Aufgabe 14 (*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.

a) 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 $P_1 = \langle i, \ldots, j\rangle$ und $P_2 = \langle i, \ldots, j\rangle$ existieren, die bis auf die Endpunkte knotendisjunkt sind, d.h.: $P_1 \cap P_2 = \{i, j\}$.

b) 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.

c) 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.

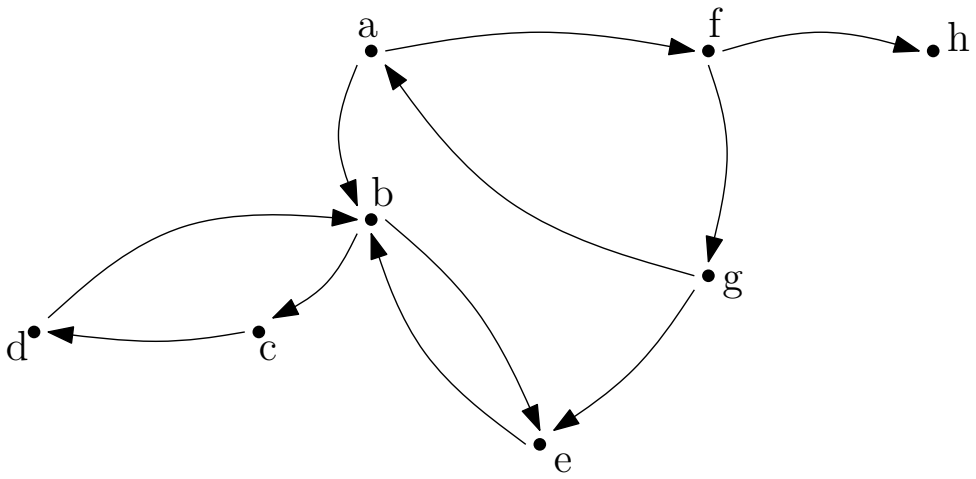
d) Vervollständigen Sie den angegebenen allgemeinen DFS-Algorithmus, so dass er in $O(|V| + |E|)$ alle Gelenkpunkte eines ungerichteten Graphen berechnet. Geben Sie an, was die Funktionen `init`, `root(s)`, `traverseTreeEdge(v,w)`, `traverseNonTreeEdge(v,w)` und `backTrack(u,v)` machen.
Überlegen Sie 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) ∈ 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 15 (*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 16 (*Analyse: Kürzeste Wege*)

a) Betrachten Sie eine Suche mit bidirektionalem Dijkstra. Geben Sie eine Familie von Graphen an, bei der ein ausgezeichneter Knoten u eine Anzahl an `decreaseKey` Operationen erfährt, die linear in der Länge des kürzesten Weges für jede mögliche Anfrage ist.

b) Bei manchen Algorithmen kann es sinnvoll sein, unterschiedliche Potentialfunktionen zu kombinieren. Zeigen Sie in diesem Zusammenhang: Sind π_1 und π_2 gültige Potentialfunktionen, so ist auch $\pi = \frac{\pi_1+\pi_2}{2}$ eine gültige Potentialfunktion.

c) Bei der Durchführung von *Dijkstras Algorithmus* können kürzeste Wege gespeichert werden, indem Vorgängerknoten gespeichert werden. Diese werden immer dann geändert, wenn eine bessere vorläufige Distanz gefunden wird. Dieses kann auch auf einem Graphen durchgeführt werden, der negative Kantengewichte enthält. Das Stopkriterium von Dijkstras Algorithmus muss dafür durch ein schwächeres Kriterium ersetzt werden: Es wird gestoppt, sobald keine Verbesserung mehr gefunden werden kann. Zeigen Sie, wenn auf diese Art ein Kreis entsteht, so ist dieser negativ.

d) (*) Dijkstras Algorithmus ist ein Spezialfall des allgemeinen *Labeling Algorithmus*. Der allgemeine Labeling Algorithmus wählt eine beliebige Kante aus, die das Label des Zielknotens ver-

bessert. Geben Sie einen Graphen sowie eine Reihenfolge der Kanten an, so dass bei positiven Kantengewichten eine exponentielle Zahl von Schritten ausgeführt wird.

Hinweis: Achtung, schwierige Knobelaufgabe!

Ausgabe: 16.01.2018
Abgabe: keine Abgabe, keine Korrektur