Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Hans-Peter Lehmann, Daniel Seemaier

2. Übungsblatt zu Algorithmen II im WS 2021/2022

http://algo2.iti.kit.edu/AlgorithmenII_WS21.php
{sanders, hans-peter.lehmann, daniel.seemaier}@kit.edu

Aufgabe 1 (*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 2 (*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) Beweisen Sie die Behauptung aus der Vorlesung, dass für $m = \Omega(n \log n \log\log n)$ Dijkstra's Algorithmus mit einem *binary heap* eine durchschnittliche Laufzeit von $O(m)$ besitzt.

b) 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.

Aufgabe 3 (*Einführung+Analyse: Bidirektionaler Dijkstra*)

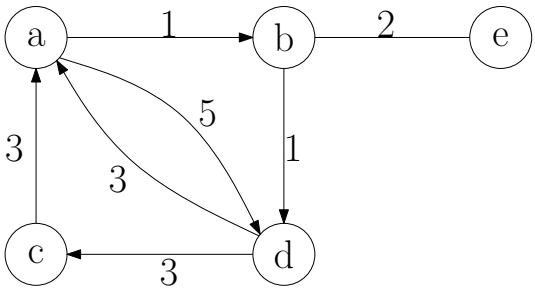
Gegeben sei –wie üblich– ein gerichteter Graph $G = (V,\ E)$ mit $|V| = n$ und $|E| = m$, sowie eine Kantengewichtungsfunktion $c : E \to \mathbb{R}_0^+$. Gesucht ist der kürzeste Pfad $p = \langle s, \ldots, t\rangle$ zwischen zwei Punkten $s,\ t \in V$.

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).

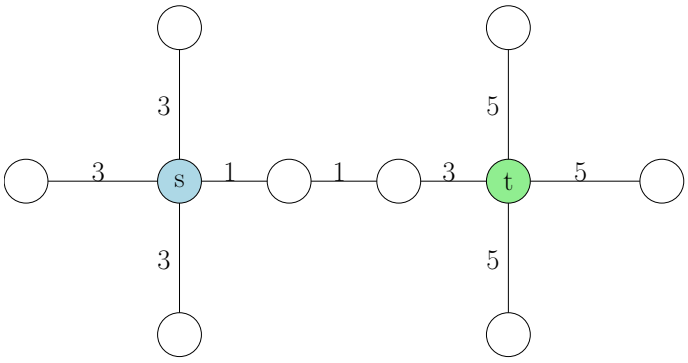
Sobald ein Knoten in einer Richtung gescannt werden soll, der bereits in der anderen Richtung gescannt worden ist, kann die Suche beendet werden (*Abbruchbedingung*). Die aktuelle minimale gefundene Distanz ist dann die tatsächliche minimale Distanz zwischen s und t.

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 4 (*Rechnen+Analyse: Suche in Strings*)

a) Zur Ausführung des KMP-Algorithmus muss zunächst ein sogenanntes *border-array* berechnet werden. Geben Sie das *border-array* für das Suchmuster $P =$ `abacababc` an.

b) Führen Sie den KMP-Algorithmus auf dem Text $T =$ `abacababbabacababc` mit obigem Suchmuster durch.

c) Wie oft muss der KMP-Algorithmus ein Muster P der Länge $|P|$ maximal an einen Text T der Länge $|T|$ anlegen, falls P nicht in T vorkommt? Wie oft minimal? Geben Sie das Ergebnis in Abhängigkeit von $|P|$ und $|T|$ an. Geben Sie außerdem jeweils ein Beispiel für P und T an.

d) Zeigen oder widerlegen Sie:
Das *border-array* kann keine drei aufeinanderfolgenden Einträge enthalten, die jeweils um eins kleiner sind als ihr Vorgänger, falls der erste von diesen drei Einträgen auf ein Suffix der Größe $k \geq 3$ verweist, welches gleichzeitig echtes Präfix ist.

Beispiel: $\text{border}[10] = 3,\ \text{border}[11] = 2,\ \text{border}[12] = 1$

Kein Beispiel: $\text{border}[10] = 2,\ \text{border}[11] = 1,\ \text{border}[12] = 0$

Aufgabe 5 (*Rechnen+Analyse: LCP-Array*)

Gegeben sei die Zeichenkette $s =$ `salsadipp$`.

a) Geben Sie den Suffixbaum für s an.

b) Geben Sie das Suffixarray für s an.

c) Geben Sie das LCP-Array für s an.

Im Folgenden sei ein String T sowie dessen Suffixarray SA[·] und dessen LCP-Array LCP[·] gegeben.

d) Wie kann der längste sich wiederholende Substring in T effizient bestimmt werden?
(der Substring darf sich dabei selbst überlappen)

e) Wie viele paarweise unterschiedliche Substrings kann ein String der Länge n maximal besitzen? Wie kann die tatsächliche Anzahl für einen konkreten String T bestimmt werden?

f) Ein String lässt sich schlecht komprimieren, wenn er wenig Redundanz besitzt. Ein Maß dafür ist die Anzahl paarweise unterschiedlicher Substrings normiert auf die mögliche Gesamtanzahl unterschiedlicher Substrings. Geben Sie an, wie dieses Maß für T berechnet werden kann.

Ausgabe: 08.11.2021
Abgabe: keine Abgabe, keine Korrektur