Dr. S. Lewandowski **Wegeprobleme in Graphen** SS 2008

Übungsblatt 2

Ausgabe: 9.5. Abgabe: 29.5.

Ablauf der Übungen: Ausgabe der Übungen spätestens 8 Tage vor dem nächsten Übungstermin in der Vorlesung (pdf auf der Vorlesungsseite). Abgabe spätestens am Tag vor der Übung in der Vorlesung oder per E-Mail an `Lewandowski@fmi.uni-stuttgart.de`.
Scheinbedingungen: Regelmäßige und aktive Teilnahme an den Übungen und Abgabe von mehr als 50% der Aufgaben. Übungstermine: 9.5., 30.5., 13.6., 27.6., 11.7.

---

1. (leicht–mittel) **Kürzeste Wege in azyklischen Graphen:** Wir haben in der Vorlesung einen Algorithmus für die Kürzeste-Wege-Berechnung in azyklischen Graphen vorgestellt. Die Reihenfolge, in der die Knoten bearbeitet werden, entspricht einer topologischen Sortierung des Graphen.

   Überlegen Sie sich, wie Sie diesen Algorithmus implementieren würden. Berechnet man die topologische Sortierung vorab oder „on-the-fly“ oder noch anders?

   Beachten Sie dabei, dass wir davon ausgegangen sind, dass alle Knoten vom Startknoten v_0 aus erreichbar sind – dies lässt sich i.A. aber nicht garantieren (wählen Sie z.B. in einem azyklischen Graphen zwei beliebige Startknoten v_0 und v_0', so ist entweder v_0' von v_0 aus nicht erreichbar oder umgekehrt).

   Beachten Sie insbesondere, dass die Laufzeit nur $O(n+m)$ betragen darf!

2. (leicht) **Ein Linearzeitalgorithmus:** Zeigen Sie: Das SSSP-Problem mit $\gamma(u,v) = 1$, $(u,v) \in E$, lässt sich in Linearzeit durch den Bellman-Ford-Algorithmus lösen. Dabei gilt stets $D(v)=d(v)$ oder $D(v)=\infty$, d.h., erhält ein Knoten v zum ersten Mal einen Wert $D(v)$, so ist dieser bereits korrekt.

3. (mittel) **Dijkstra mit negativen Kantengewichten:** In der Grundvorlesung haben wir gelernt, dass der Dijkstra-Algorithmus mit negativen Kantengewichten i.A. nicht funktioniert. Dies war nur die halbe Wahrheit. Lässt man die explizite Verwaltung der Baummenge weg und nimmt Knoten bei Verringerung der Werte $D(\cdot)$ ggf. auch erneut in R auf (dies kann aufgrund negativer Kantengewichte vorkommen), so arbeitet Dijkstras Algorithmus auch mit negativen Kantengewichten korrekt – nur die Laufzeitabschätzungen stimmen nicht mehr.

   - Beweisen Sie, dass Dijkstras Algorithmus mit negativen Kantengewichten korrekt arbeitet, wenn man zulässt, dass Knoten nach Auswahl aus R zu einem späteren Zeitpunkt wieder in R aufgenommen werden dürfen.
   - Finden Sie Beispiele, für die der so modifizierte Algorithmus möglichst viele Schritte benötigt.

4. (leicht) **Eigenschaften des Dijkstra-Algorithmus:**

   - Zeigen Sie: Beim Dijkstra-Algorithmus werden die Knoten in aufsteigender Reihenfolge der Entfernung zum Startknoten betrachtet.
   - Zeigen Sie durch Reduktion auf das Sortierproblem, dass jede vergleichsbasierte Implementierung des Dijkstra-Algorithmus im Worst-Case $\Omega(m + n\log(n))$ Schritte benötigt.

5. (leicht+schwer) **Dijkstra-Algorithmus mit R als Heap:**

   - (leicht) Bei der Implementierung der Menge R als Heap muss man bei decrease_key-Aufrufen die Werte $D(\cdot)$ der Knoten ändern und deren Position im Heap anpassen. Überlegen Sie sich, wie dies implementiert werden kann – betrachten Sie dabei insbesondere die Frage, wie Sie die Position eines Knotens im Heap schnell finden können.
   - (schwer) In der Praxis lohnt sich die Verwendung von Fibonacci-Heaps gegenüber normalen Heaps in der Regel nicht. Dies liegt zum einen an der geringen erwarteten Anzahl von decrease_key-Aufrufen von $O(n\log(1+m/n))$ (Goldberg/Tarjan, Expected Performance of Dijkstra's Shortest Path Algorithm, 1996), zum anderen lässt sich zeigen, dass auch in einem normalen Heap die Laufzeit $O(m+n\log(n))$ gilt, wenn in den decrease_key-Aufrufen die Knoten im Mittel nur $O(1)$ Ebenen aufsteigen.

     Konstruieren Sie Graphen, bei denen viele (möglichst $\Theta(m)$) decrease_key-Aufrufe vorkommen und die betroffenen Knoten jeweils viele Ebenen (möglichst $\Theta(\log(n))$) im Heap aufsteigen.
