Karlsruher Institut für Technologie

Tutorium 10

Algorithmen I SS 14

Institut für Theoretische Informatik

Kürzeste Wege

Bellman-Ford: Einsatz

- Wie Dijkstra: Finden der kürzesten Wege vom Startknoten zu allen anderen Knoten (single-source shortest paths)
- auch mit negativen Kantengewichten
- insbesondere Finden von negativen Kreisen im Graphen

Bellman-Ford-Algorithmus: Idee

- kürzeste Wege können maximal die Länge $|V| - 1$ haben
- die kürzesten Wege mit Länge k+1 sind aus denen der Länge k berechenbar

Bellman-Ford: Algorithmus

1. Initialisiere die Knotendistanz d des Startknotens mit 0 alle anderen mit ∞.
2. Iteriere $|V| - 1$-mal über alle Kanten:
   - Ist $d_v > d_u +$ Kantengewicht $e_{(u,v)}$, dann aktualisiere d_v
3. Iteriere nocheinmal über die Kanten. Ändern sich Werte so gibt es einen negativen Kreis

$\Rightarrow$ Laufzeit $O(|V||E|)$

Kürzeste Wege in DAGs

- Bessere Algorithmen für Spezialfälle?
- DAG: gerichteter, azyklischer Graph $\Rightarrow$ keine Kreise (insbesondere keine negativen kreise)
- Frage: Wie kann man das für eine bessere Laufzeit nutzen?

Kürzeste Wege in DAGs

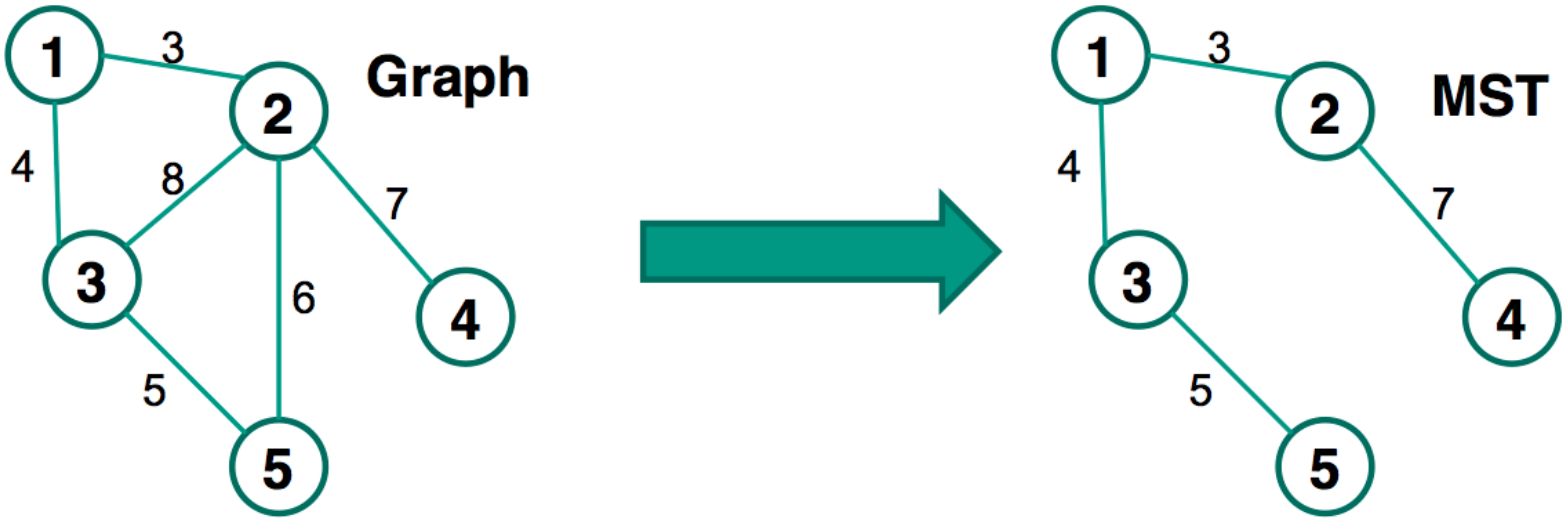
- Berechne topologische Sortierung ($\mathcal{O}(n+m)$)
- Relaxiere ausgehende Kanten der Knoten in der sortierten Reihenfolge (beginnend mit dem Startknoten)
- Gleicher Effekt wie bei Dijkstra: Nur Kanten von Knoten mit minimaler Entfernung werden Relaxiert
- Insgesamt Laufzeit $\mathcal{O}(n+m)$

Minimale Spannbäume

Minimale Spannbäume (MST)

Gegeben: Ungerichteter Graph mit Knotenmenge V, Kantenmenge E und einer Gewichtsfunktion c

Gesucht: Baum mit Knotenmenge V, Kantenmenge F, $F \subseteq E$ mit minimaler $\sum_{e \in T} c(e)$, der alle Knoten verbindet



Schnitteigenschaft (Cut Property)

- Für eine beliebige Knotenmenge $S \subseteq V$ sei C die Schnittkantenmenge:

$$C := \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$$

- Die Kante mit dem geringsten Gewicht aus C ist im MST enthalten.
- Intuitiv klar, Teilmengen müssen irgendwo verbunden sein (formaler Beweis über Kreis in alternativem MST)

Kreiseigenschaft (Cycle Property)

- In einem Kreis in dem Graphen ist die schwerste Kante nicht im MST enthalten.
- Auch intuitiv klar, man kann die Kante einfach weglassen

Jarnik-Prim Algorithmus

- Beginne MST beginnend bei einem Startknoten schrittweise auf
- Nutze die Schnitteigenschaft: Wähle immer die minimale Kante zwischen dem (wachsenden) MST und dem Restgraph
- Speichere zu jedem Knoten die Entfernung zum MST (Knoten im MST haben Entfernung 0)
- Zusätzlich verwalte Knoten in Priority-Queue, mit dem Gewicht der leichtesten Kante zwischen dem MST und dem Knoten
- Fast wie Dijkstra, also Laufzeit $\mathcal{O}((m+n)\log n$ bzw. $\mathcal{O}(m+n\log n)$ mit Fib. Heaps

Jarnik-Prim Algorithmus

1. Füge Startknoten in Priority-Queue Q ein
2. Solange Q Elemente enthält:
   1. Nimm minimalen Knoten u aus Q und füge ihn in den MST ein.
   2. Betrachte alle Kanten (u, v) des Knotens.
   3. Für „bessere“ Kanten füge v in Q ein oder aktualisiere die Priorität (Kantengewicht).

Kruskals Algorithmus

1. Sortiere die Kanten aufsteigend nach Gewicht
2. Für jede Kante
    1. Überprüfe, ob die Kante zwei Knoten verbindet, die schon im MST liegen
    2. Wenn nicht, füge die Kante zum MST hinzu

Kruskals Algorithmus

- Benutzt sowohl Kreis- als auch Schnitteigenschaft
- Entscheidend: Verwaltung der Teilmengen mit geeigneter Datenstruktur
- $\Rightarrow$ **Union-Find**
- Laufzeit: $\mathcal{O}(m \log m)$ (beschränkt durch Sortieren)

Kreativaufgabe: Buy or Build

Gegeben ist eine Menge von Städten, die alle durch ein Netzwerk verbunden werden sollen. Zu jeder Stadt sind die Koordinaten gegeben, die Kosten um zwei Städte zu verbinden entsprechen dem Abstand. Zusätzlich existiert aber schon eine kleine Menge (≤ 8) von bereits vorhandenen Netzwerken, die gekauft werden können.
Aufgabe: Wie kann möglichst effizient berechnet werden, wie hoch die Kosten sind um alle Städte zu verbinden? (Zunächst: wie kann die Aufgabe durch Graphen dargestellt werden?)