Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 10b, Mittwoch, 26. Juni 2013

(Dijkstras Algorithmus)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

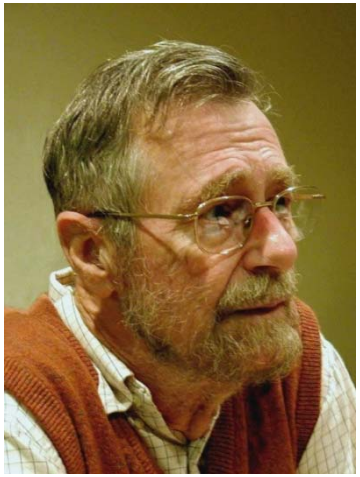
- Graphenalgorithmen, Teil 2
  - Dijkstras Algorithmus zur Berechnung kürzester Wege
  - Idee
  - Beispiel
  - Tipps zur Implementierung
  - Übungsblatt 10, Aufgabe 2: Implementieren !

Dijkstras Algorithmus 1/3

UNI FREIBURG

- Idee
  - Sei s der Startknoten und sei dist(s,u) die Länge des kürzesten Pfades von s nach u, für alle Knoten u
  - Besuche die Knoten in der Reihenfolge der dist(s,u)

    Wir werden gleich sehen: wenn alle Kantenlängen = 1 sind, ist das genau Breitensuche / BFS
- Ursprung
  - Edsger Dijkstra (1930 – 2002)
    Niederländischer Informatiker, einer der wenigen Europäer, die den Turing-Award gewonnen haben (für seine Arbeiten zur strukturierten Programmierung)
  - Der Algorithmus ist aus dem Jahr **1959**

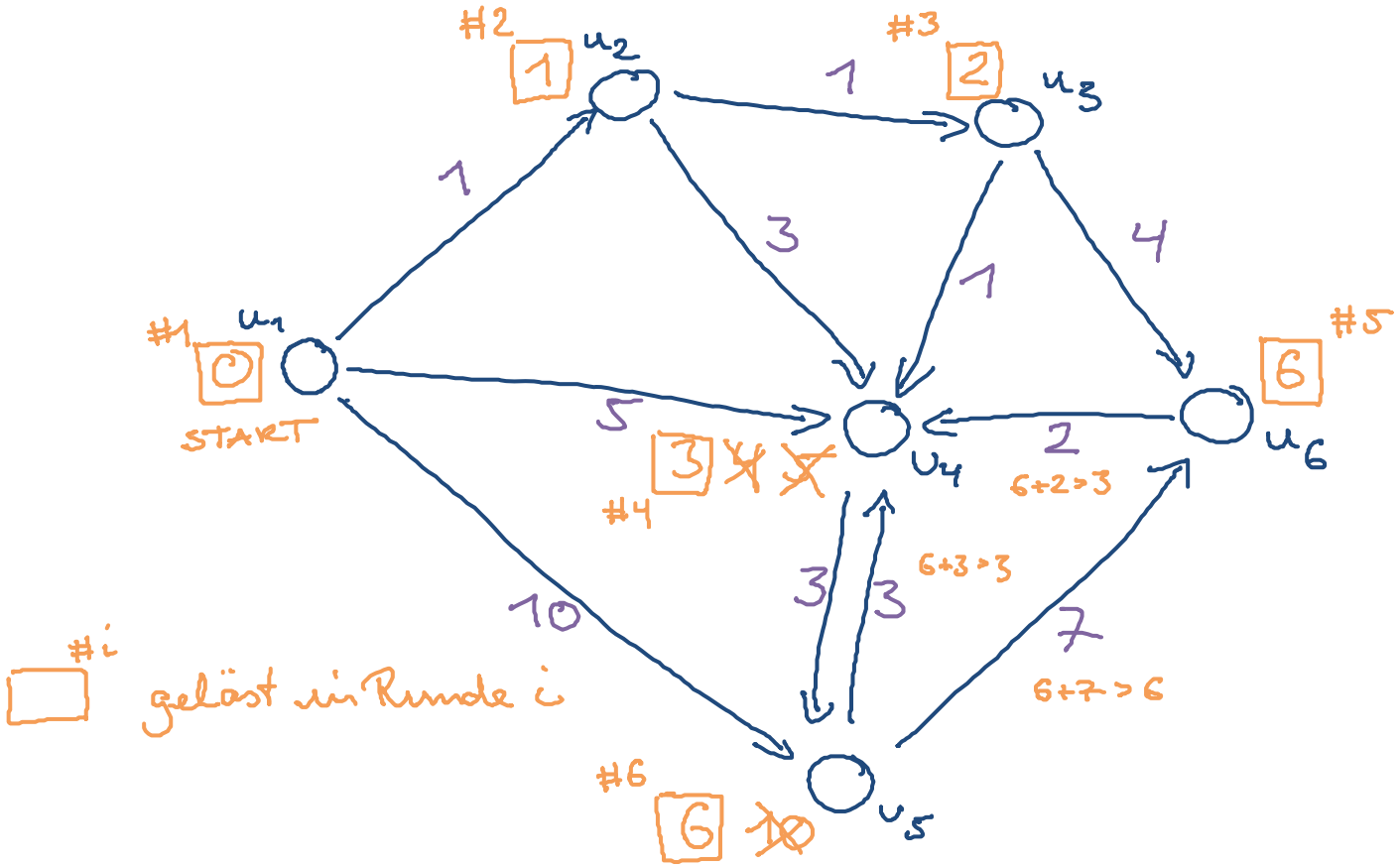


Dijkstras Algorithmus 2/3

- High-level Beschreibung des Algorithmus
  - Drei Arten von Knoten:
    - Für die **gelösten** Knoten u kennen wir dist(s, u)
    - Für die **aktiven** Knoten haben wir einen Pfad der Länge dist(u) ≥ dist(s,u) (kann optimal sein, muss aber nicht)
    - Die **unerreichten** Knoten haben wir noch nicht erreicht
    - Auf Englisch: settled, active, unreached
  - In jeder Runde holen wir uns den **aktiven** Knoten u mit dem **kleinsten** Wert für dist(u)
  - Den Knoten u betrachten wir dann als **gelöst**
  - Für jeden Nachbarn v von u prüfen wir, ob wir v über u schneller erreichen können als bisher = Relaxieren von (u, v)
  - Nächste Runde, bis es keine aktiven Knoten mehr gibt

Dijkstras Algorithmus 3/3

- Beispiel



Korrektheitsbeweis 1/3

im Beispiel auf der Folie vorher war das nicht so.

UNI FREIBURG

- Argumentationslinie
  - **Annahme 1:** Alle Kantenlängen sind > 0, siehe Folie 19
  - **Annahme 2:** Die dist(s, u) sind **alle verschieden**

    Das erlaubt einen einfacheren und intuitiveren Beweis

    Es geht aber auch ohne, siehe Referenzen … nur bei Interesse
  - Mit A2 gibt es eine Anordnung u_1, u_2, u_3, **...** der Knoten

    so dass gilt $dist(s, u_1) < dist(s, u_2) < dist(s, u_3) <$ **...**
  - Wir wollen zeigen, dass am Ende von Dijkstras Algorithmus

    $dist(u_i) = dist(s, u_i)$ für jeden Knoten u_i
  - Im Folgenden zeigen wir, durch Induktion über i, dass:

    … in der i-ten Runde gilt $dist(u_i) = dist(s, u_i)$

    … in der i-ten Runde wird Knoten u_i gelöst

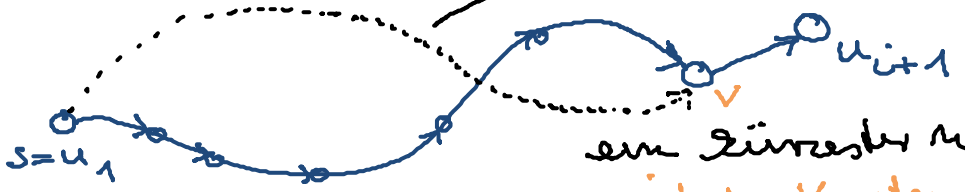
Korrektheitsbeweis 2/3

UNI FREIBURG

Induktionsanfang: $i = 1$

Im Runde 1 ist nun $u_1 = s$ aktiv, der wird dann gelöst mit $dist(u_1) = 0$ und das ist auch $dist(s, u_1) = 0$.

Induktionsschritt: $i \to i+1$, $i \geq 1$



sein anderer Weg zu v kann kürzer sein

ein kürzester Weg von s nach u_{i+1}

v ist der Knoten eins vor u_{i+1} auf diesem Weg. (v muss nicht u_i sein)

$dist(s, u_{i+1}) = dist(s, v) + \underbrace{cost(v, u_{i+1})}_{> 0 \text{ nach Annahme 1}}$

$\Rightarrow dist(s, v) < dist(s, u_{i+1})$

v muss also eins von den $u_1, \dots, u_i$ sein (aber nicht unbedingt u_i)

also $v = u_j$ für ein $j \in \{1, \dots, i\}$

Korrektheitsbeweis 3/3

$v = u_j \;,\; j \in \{1, \dots, i\}$

Also nach Induktionsvoraussetzung $dist(v) = dist(s,v)$
und zwar schon seit Runde j.

$\Rightarrow dist(u_{i+1}) = dist(s,v) + cost(v, u_{i+1}) = dist(s, u_{i+1})$
schon seit Runde j !
(aber erst in Runde $i+1$
können wir uns sicher
sein, dass es stimmt)

Wir müssen noch zeigen, dass in Runde $i+1$ auch
u_{i+1} gelöst wird und u_k mit $k > i+1$

Sei $k > i+1$, dann $dist(u_k) \geq dist(s, u_k) > dist(s, u_{i+1})$

$\Rightarrow u_{i+1}$ ist in Runde $i+1$ der mit dem kleinsten
dist-Wert (unter den aktiven, noch nicht gelösten)

Implementierungshinweise 1/3

UNI FREIBURG

- Einige Hinweise (der Rest ist Übungsaufgabe)
  - Wir müssen die Menge der **aktiven Knoten** verwalten
  - Ganz am Anfang ist das nur der Startknoten
  - Am Anfang jeder Runde brauchen wir den **aktiven** Knoten u mit dem **kleinsten** Wert für dist(u)
  - Es bietet sich also an, die aktiven Knoten in einer **Priori-tätswürgeschlange** zu verwalten, mit Schlüssel dist(u)
  - Folgendes Problem taucht dabei auf:

    Die Länge des aktuell kürzesten Pfades zu einem aktiven Knoten kann sich mehrmals ändern, bevor der Knoten schließlich gelöst wird

    Wir müssen dann seinen Wert in der PW verkleinern, **ohne** dass wir den Knoten rausnehmen

Implementierungshinweise 2/3

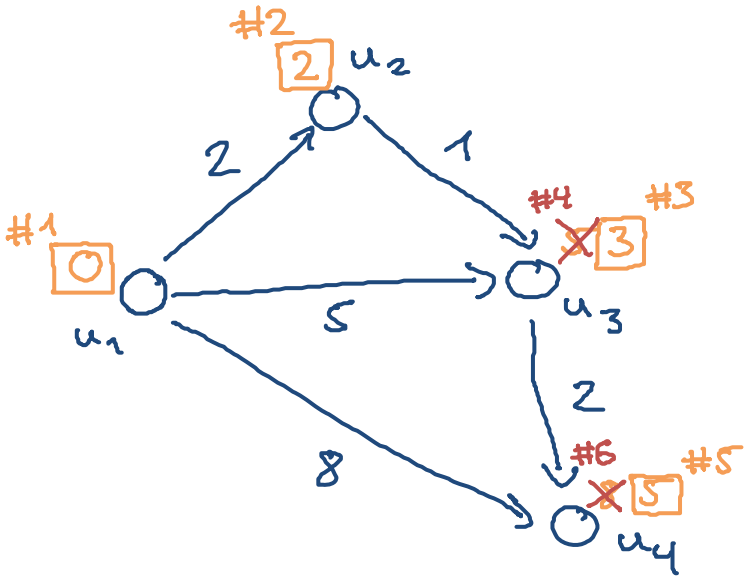
UNI FREIBURG

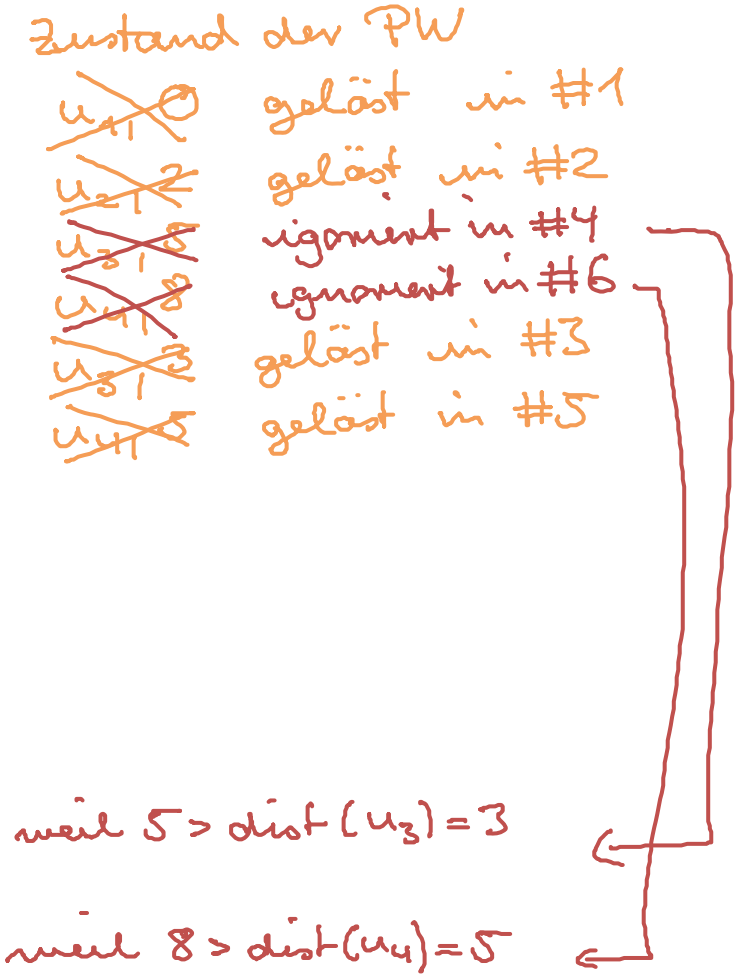
- Oft gibt es nur insert, getMin und deleteMin
  - Mit so einer PW hat man nur Zugriff auf das jeweils kleinste Element, nicht auf ein beliebiges
  - **Alternative:** Sieht man einen Knoten wieder, mit einem niedrigeren dist Wert, fügt man ihn einfach nochmal ein

    Bei einem gleichen oder höheren dist Wert macht man nichts !
  - Den Eintrag mit dem alten Wert lässt man einfach drin
  - Wenn der Knoten gelöst wird, dann mit dem niedrigsten Wert mit dem er in die PW eingefügt wurde
  - Wenn man dann später nochmal auf den Knoten trifft, mit höherem dist Wert, nimmt man ihn einfach heraus und macht **nichts**

Implementierungshinweise 3/3

- Beispiel für Dijkstra mit PW ohne decreaseKey





Laufzeitanalyse

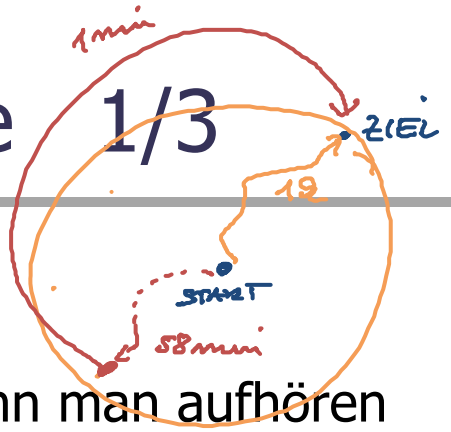
- Für einen Graph mit n Knoten und m Kanten
  - Jeder Knoten wird genau **einmal** gelöst
  - Genau dann werden seine ausgehenden Kanten betrachtet
  - Jede ausgehende Kante führt zu höchstens einem insert
  - Die Anzahl der Operationen auf der PW ist also $O(m)$
  - Die Laufzeit von Dijkstras Algorithmus ist also $O(m \cdot \log m)$
  - Weil $m \leq n^2$ ist das auch $O(m \cdot \log n)$ $\log n^2 = 2 \cdot \log n$
  - Mit einer aufwändigeren PW geht auch $O(m + n \cdot \log n)$

    Zum Beispiel mit sogenannten **Fibonacci-Heaps**

    Für große und dichte Graphen ($m \sim n^2$) ist das klar besser

    In der Praxis ist aber oft $m = O(n)$ und dann ist der einfache **Binary Heap** die bessere Wahl ... siehe Vorlesung 9a

Weiterführende Kommentare 1/3



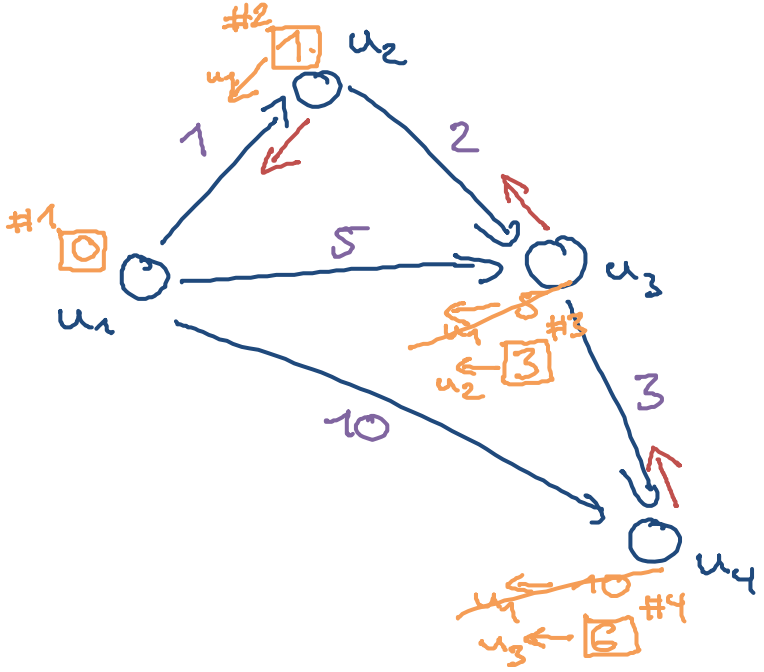
- Abbruchkriterium
  - Sobald der Zielknoten t gelöst wird kann man aufhören

    … aber nicht vorher, dann kann dist(t) > dist(s, t) sein!
  - Bevor Dijkstras Algorithmus t erreicht, hat er die kürzesten Wege zu **allen** Knoten u mit dist(s, u) < dist(s, t) berechnet
  - Dijkstras Algorithmus löst damit nicht nur das sogenannte single source **single** target shortest path Problem, sondern gleich das sogenannte single source **all** targets Problem
  - Das hört sich verschwenderisch an, es gibt aber für allgemeine Graphen keine (viel) bessere Methode

    Intuitiv: erst wenn man **alles** im Umkreis von dist(s, t) um den Startknoten s abgesucht hat, kann man sicher sein, dass es keinen kürzeren Weg zum Ziel t gibt

Weiterführende Kommentare 2/3

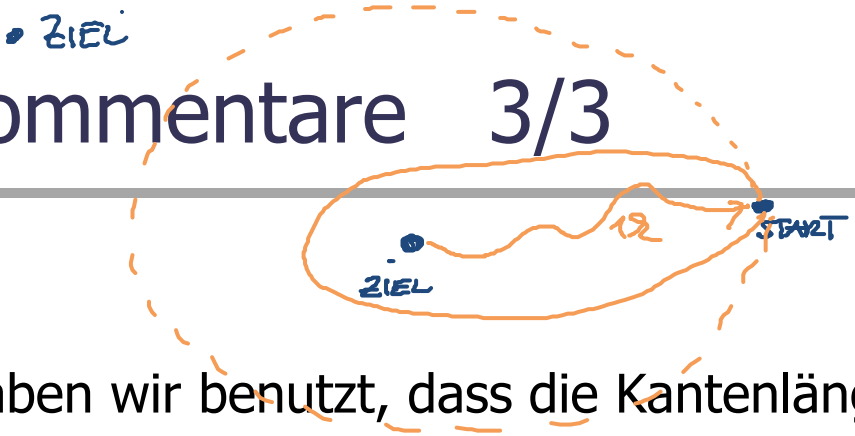
- Berechnung der kürzesten Pfade
  - So wie wir Dijkstras Algorithmus bisher beschrieben haben, berechnet er nur die **Länge** des kürzesten Weges
  - Wenn man sich bei jeder Relaxierung den Vorgängerknoten auf dem aktuell kürzesten Pfad merkt, kriegt man aber auch leicht die tatsächlichen **Pfade**



am Ende hat man (zusätzlichen zu den Distanzen) nur noch die roten Pfeile = einen Knoten

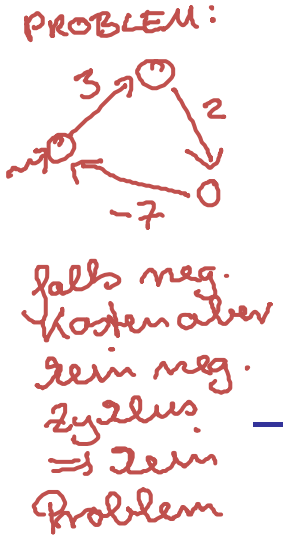
(das reicht, weil jeder Präfix eines kürz. Weges auch ein kürz. Weg ist)

Weiterführende Kommentare 3/3



- Erweiterungen
  - In unserem Beweis haben wir benutzt, dass die Kantenlängen alle **positiv** sind
  - Bei negativen Kantenkosten kann es negative Zyklen geben, um mit denen umzugehen braucht man andere Algorithmen
    - Zum Beispiel den Bellman-Ford Algorithmus
    - Wenn der Graph **azyklisch** ist, reicht auch einfach topologisches Sortieren (mit DFS) + Relaxieren der Knoten in der Reihenfolge dieser Sortierung
  - Eine (nicht nur) in der künstlichen Intelligenz häufig benutzte Variante von Dijkstras Algorithmus ist der **A* Algorithmus**:

    Dann zusätzlich gegeben: h(u) = Schätzwert für dist(u, t)



Literatur / Links

UNI FREIBURG

- Kürzeste Wege und Dijkstras Algorithmus
  - In Mehlhorn/Sanders:

    10 Shortest Paths
  - In Wikipedia

    http://en.wikipedia.org/wiki/Shortest_path_problem

    http://en.wikipedia.org/wiki/Dijkstra's_algorithm