Informatik II: Algorithmen und Datenstrukturen SS 2015

Vorlesung 10b, Mittwoch, 1. Juli 2015
(Dijkstras Algorithmus)

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

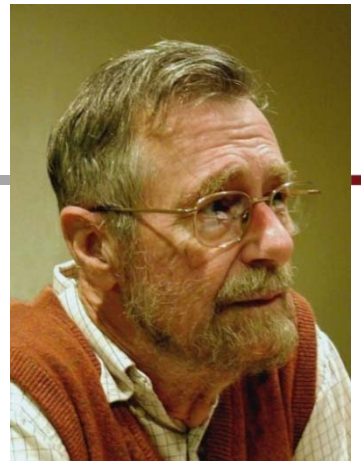
UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Inhalt
  - Fortsetzung von gestern … BFS, DFS, Z-Komponenten
  - Dijkstras Algorithmus
  - Korrektheitsbeweis
  - Implementierung + Laufzeit
  - Ü10, Aufgabe 2: Implementieren Sie Dijkstras Algorithmus zur (einfachen) Routenplanung auf Baden-Württemberg

Dijkstras Algorithmus 1/4



- Ursprung
  - Benannt nach 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 2/4

UNI FREIBURG

- Grundidee und Terminologie
  - 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)
  - Für jeden Knoten wird während der Ausführung eine vorläufige Distanz dist[u] gespeichert, zu Beginn ∞
  - Es gibt dann drei Arten von Knoten

    unerreicht: dist[u] = ∞

    aktiv: dist[u] ≥ dist(s, u) aber nicht ∞

    gelöst: siehe nächste Folie

    Auf Englisch: unreached, active, settled

Dijkstras Algorithmus 3/4

UNI FREIBURG

- Algorithmus
  - Zu Beginn ist nur s aktiv, mit dist[s] = 0
  - 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

    Das nennt man **Relaxieren** von v bzw. von (u, v)
  - Wiederhole, bis es keine aktiven Knoten mehr gibt

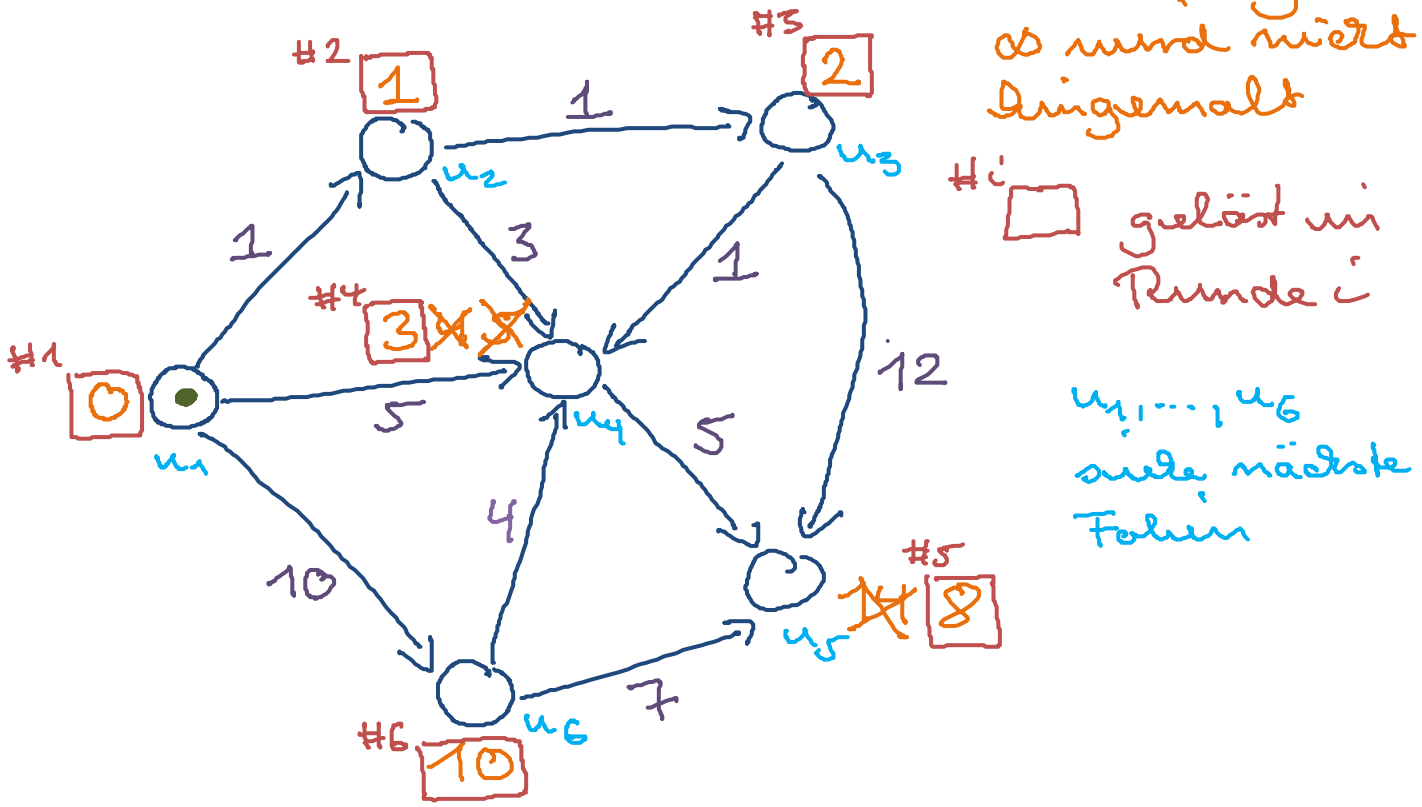
    Alle gelösten Knoten kennen dann ihre Entfernung von s

wenn ja, dann dist[v] updaten

Falls alle Kanten gleiche Kosten haben bzw. Kosten 1, ist das Vorgehen wie BFS.

Dijkstras Algorithmus 4/4

- Beispiel



dist Werte in Orange ∞ wird nicht dringemalt

#i gelöst in Runde i

u1, ..., u6 siehe nächste Folien

Korrektheitsbeweis 1/6

UNI FREIBURG

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

    Es gibt dann eine Anordnung $u_1, u_2, u_3, \ldots$ der Knoten

    so dass gilt $dist(s, u_1) < dist(s, u_2) < dist(s, u_3) < \ldots$
  - Es geht auch mit Kantenlängen ≥ 0 und ohne Annahme 2

    Beweis dazu siehe Referenzen (Mehlhorn/Sanders)

    Mit den Annahmen ist der Beweis einfacher und intuitiver und enthält trotzdem alles Wesentliche

Korrektheitsbeweis 2/6

UNI FREIBURG

- Argumentationslinie
  - 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
    - In der i-ten Runde gilt $dist[u_i] = dist(s, u_i)$
    - In der i-ten Runde wird Knoten u_i gelöst

Korrektheitsbeweis 3/6

UNI FREIBURG

- Induktionsanfang: $i = 1$
  - In Runde 1 ist nur $u_1 = s$ aktiv
  - $dist[u_1] = 0 = dist(s, u_1)$
  - u_1 wird als einziger aktiver Knoten gelöst

Korrektheitsbeweis 4/6

UNI FREIBURG

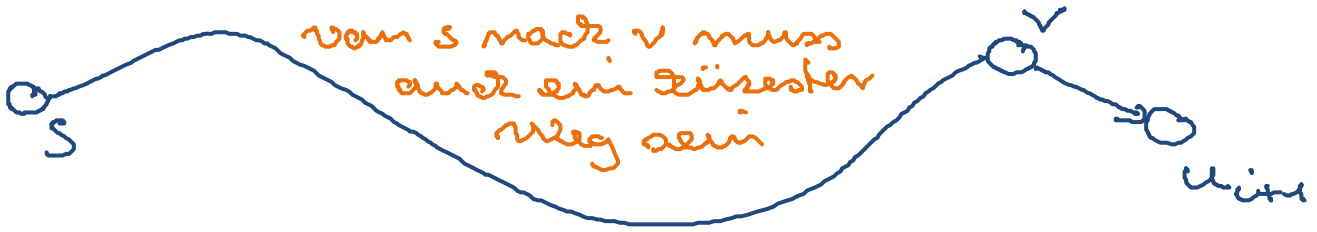
- Induktionsschritt: $i \rightarrow i + 1$ für $i \geq 1$
  - Wir betrachten einen kürzesten Weg von s nach u_{i+1}

    Wir nehmen nicht an, dass unser Algorithmus diesen Weg kennt, aber wir können ihn im Beweis trotzdem betrachten
  - Sei v der Knoten direkt vor u_{i+1} auf diesem Weg … dann:

    $dist(s, u_{i+1}) = dist(s, v) + cost(v, u_{i+1}) > dist(s, v)$

    Hier benutzen wir Ann. 1 = alle Kantenkosten positiv
  - v muss also einer von $u_1, \ldots, u_i$ sein (aber nicht unbedingt u_i)

    Hier benutzen wir Ann. 2 = $dist(s, u_1) < dist(s, u_2) < \ldots$



Korrektheitsbeweis 5/6

- Induktionsschritt: $i \rightarrow i + 1$ für $i \geq 1$
  - Nach Induktionsvoraussetzung gilt seit spätestens Runde j

    $dist[u_j] = dist(s, u_j)$
  - In der Runde hat man dann, nach Relaxieren von (u_j, u_{i+1})

    $dist[u_{i+1}] = dist(s, u_j) + cost(u_j, u_{i+1}) = dist(s, u_{i+1})$

    Das gilt schon seit Runde j, aber erst in Runde i + 1 kann sich der Algorithmus sicher sein, dass es nicht besser geht

Korrektheitsbeweis 6/6

UNI FREIBURG

- Induktionsschritt: $i \rightarrow i + 1$ für $i \geq 1$
  - Wir müssen noch zeigen, dass in Runde $i + 1$ auch u_{i+1} gelöst wird, und nicht u_k mit $k > i + 1$
  - Aber für $k > i + 1$ gilt nach Annahme 2 (Monotonie):

    $dist[u_k] \geq dist(s, u_k) > dist(s, u_{i+1})$
  - Also ist u_{i+1} in Runde $i+1$ der aktive Knoten mit dem kleinsten dist Wert und wird also in der Runde gelöst

Implementierung 1/8

UNI FREIBURG

- Grundprinzip
  - 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 **Prioritätswürgeschlange** zu verwalten

    Mit Schlüssel dist(u) und Wert u

Implementierung 2/8

UNI FREIBURG

- Update von dist[u]
  - **Beobachtung:** der dist Wert eines aktiven Knotens kann sich mehrmals ändern, bevor er schließlich gelöst wird

    Wir müssen dann seinen Wert in der PW verkleinern, ohne dass wir den Knoten rausnehmen
  - Das geht mit der Operation changeKey ohne Weiteres

    Allerdings steht diese Operation nicht bei allen PWs zur Verfügung, z.B. bei der std::priority_queue von C++

Implementierung 3/8

UNI FREIBURG

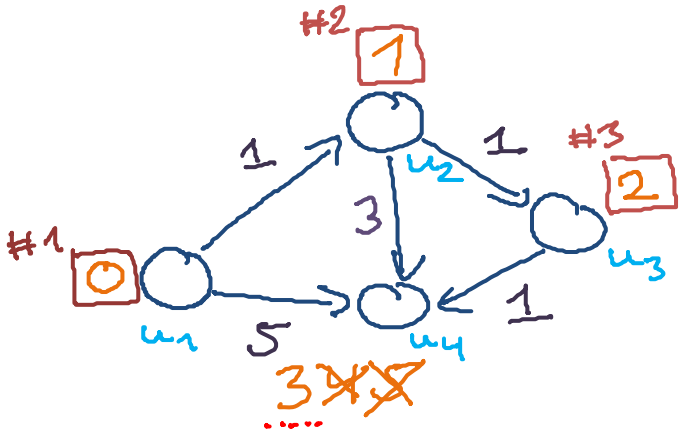
- Implementierung ohne changeKey
  - Statt changeKey macht man einfach ein insert mit dem neuen (niedrigeren) dist Wert

    Den Eintrag mit dem alten Wert lässt man einfach drin

    Bei gleichen oder höheren dist Wert macht man nichts
  - 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**

Implementierung 4/8

- Beispiel für Dijkstra mit PW ohne changeKey



Elemente in der PW

~~u_1 0~~ gelöst in Runde 1

~~u_2 1~~ gelöst in Runde 2

~~u_4 5~~ ignoriert, weil 5 > 3

~~u_3 2~~ gelöst in Runde 3

~~u_4 4~~ ignoriert, weil 4 > 3

~~u_4 3~~ gelöst in Runde 4

Implementierung 5/8

UNI FREIBURG

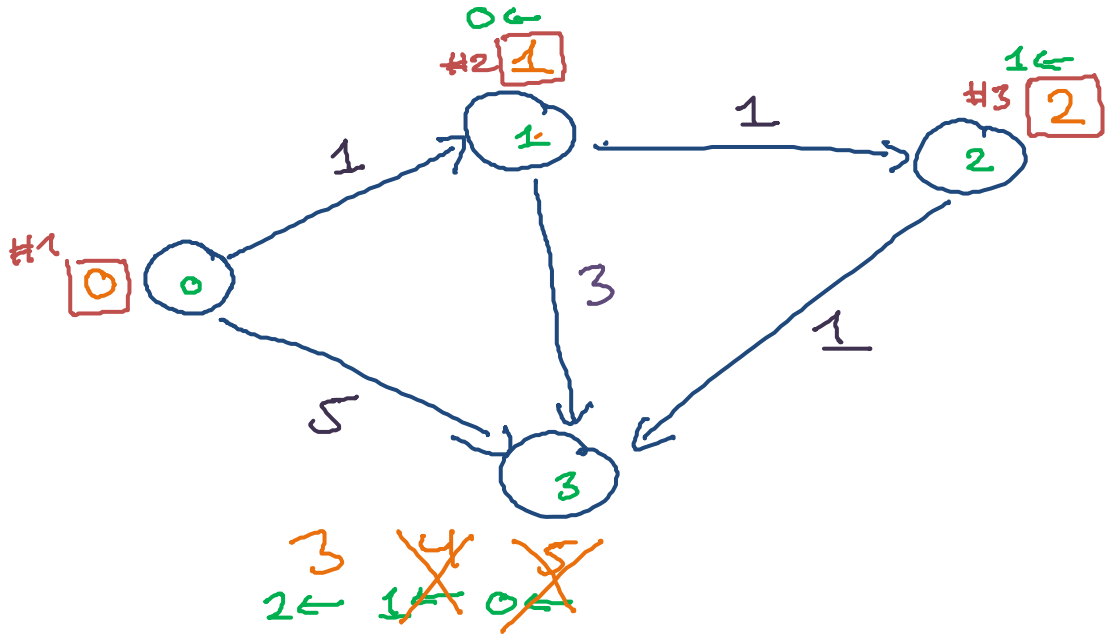
- Berechnung der kürzesten Pfade
  - So wie wir Dijkstras Algorithmus bisher beschrieben haben, berechnet er nur die **Länge** des kürzesten Weges
  - Wenn man sich bei jeder Relaxierung den Vorgängerknoten auf dem aktuell kürzesten Pfad merkt, kriegt man aber auch leicht die tatsächlichen **Pfade**

    Es reicht für jeden Knoten ein Zeiger, weil jeder Präfix eines kürzesten Weges selber ein kürzester Weg ist

    Um den kürzesten Weg zu bekommen, kann man dann einfach die Zeiger bis nach s zurückverfolgen

Implementierung 6/8

- Berechnung der kürzesten Pfade, Beispiel



Implementierung 7/8

UNI FREIBURG

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

    Aber nicht vorher, dann kann noch 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
  - Das hört sich verschwenderisch an, es gibt aber für allgemeine Graphen keine (viel) bessere Methode

    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

Implementierung 8/8

UNI FREIBURG

- Laufzeit dieser Implementierung
  - 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 $O(m \cdot \log n)$
  - Mit einer komplizierteren PW geht auch $O(m + n \cdot \log n)$

    In der Praxis ist aber oft $m = O(n)$ ... dann asymptotisch gleich, und man nimmt besser die einfachere PW

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