Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 12, Dienstag, 31. Januar 2012

(Kürzeste Wege, 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

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü11 (Breitensuche, Durchmesser)
  - Vorschlag zum Ausgleich für die C++ Programmierer
  - Infos zur Klausur
- Kürzeste Wege
  - Dijkstras Algorithmus zur Berechnung des kürzesten Weges zwischen zwei Knoten (in einem Graphen mit Kantenkosten)
  - Idee + Beispiel + Korrektheitsbeweis
  - Hinweise zur Implementierung
  - **Übungsaufgabe (Ü12):** Dijkstra implementieren
    - als zusätzliche Methode in unserer Graph Klasse

Ihre Erfahrungen mit dem Ü11 (BFS)

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback **Stand 31.1 15:40**
  - Vorlesung zum Thema gut verständlich
  - Implementierung anspruchsvoll, aber machbar
  - Mehr Implementierungshinweise in der Vorlesung geben
  - Manche fanden es auch sehr einfach
  - Einige hatten zuerst Rekursion versucht
  - Gefühlte 100 Stunden gebraucht, vielleicht zu wenig Schlaf
  - Mehrarbeit für C++ Leute ist unfair → nächste Folie
  - Keine Zeit wegen Klausurvorbereitung (Prüfungen ab 27. Feb)
  - Leichte Abweichungen bei Aufgabe 3 (Graphentheorie)
  - Aufgabe 1 war schwieriger als 2 und 3, aber für alle 6 Punkte
  - Diesmal uebungsblatt überall richtig geschrieben
  - build.xml so geändert, dass nur aktuelle Tests durchlaufen

Ausgleich für die C++ Programmierer

UNI FREIBURG

- So sehe ich das (Kommentare willkommen)
  - Die Dateien aus der Vorlesung sind immer in Java
  - Wenn die Vorgabe nur das Gerüst der Klasse ist, sehe ich den Unterschied als minimal an
  - Aber wenn auch Code vorgegeben ist, den wir in der VL zusammen geschrieben haben, ist es schon eine erhebliche Mehrarbeit, den erst in C++ zu übersetzen
  - Manchmal gibt es die analogen Dateien von der VL vor einem Jahr, aber in einigen Fällen habe ich dieses Jahr andere Sachen gemacht
  - **Vorschlag:** für die C++ Programmierer wird das praktische Übungsblatt (Ü3, Ü4, Ü5, Ü7, Ü9, Ü11, Ü12, Ü13) mit der schlechtesten Punktzahl aus der Wertung genommen

Infos zur Klausur

UNI FREIBURG

- Am Freitag, 30. März 2012, 14:00 - 16:00 Uhr
  - Im Hörsaal 026 **...** das ist **nicht** der Vorlesungshörsaal
  - 4 Aufgaben a jeweils 20 Punkte → K1, K2, K3, K4
  - Summe Übungspunkte auf max. 20 Punkte skaliert → Ü
  - Von Ü, K1, K2, K3, K4 werden die besten vier genommen und aufsummiert (also maximal 80 Punkte)
  - Art der Aufgaben wie bei der Klausur vom WS 2010/2011
    - ist auf dem Wiki von der alten Vorlesung verlinkt
    - **Achtung:** Stoff ist ähnlich aber nicht genau derselbe!
  - Drei Arten von Aufgaben
    - Algorithmus an einem Beispiel nachvollziehen
    - Einfaches Programm verstehen oder selber schreiben
    - Einfache Rechenaufgaben / Beweise

Graphen — Pfade (Wiederholung)

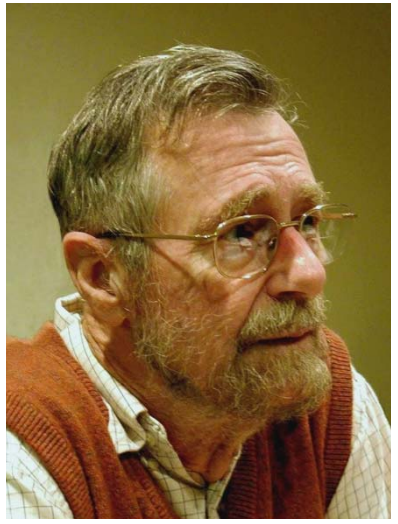
- Für einen Graphen $G = (V, E)$
  - Ein Pfad in G ist eine Folge $u_1, u_2, u_3, \ldots, u_l \in V$ mit
    - $(u_1,u_2), (u_2,u_3), \ldots, (u_{l-1},u_l) \in E$ [gerichteter Graph]
    - $\{u_1,u_2\}, \{u_2,u_3\}, \ldots, \{u_{l-1},u_l\} \in E$ [ungerichteter Graph]
  - Die **Länge des Pfades** (auch: Kosten des Pfades)
    - ohne Kantengewichte: Anzahl der Kanten
    - mit Kantengewichte: Summe der Gewichte auf dem Pfad
  - Der **kürzeste Pfad** (engl. shortest path) zwischen zwei Knoten u und v ist der Pfad $u, \ldots, v$ mit der kürzesten Länge
  - Der **Durchmesser** eines Graphen ist der längste kürzeste Pfad = $\max_{u,v}${Länge von P : P ist ein kürzester Pfad zwischen u und v}

Dijkstras Algorithmus

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)
- 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**



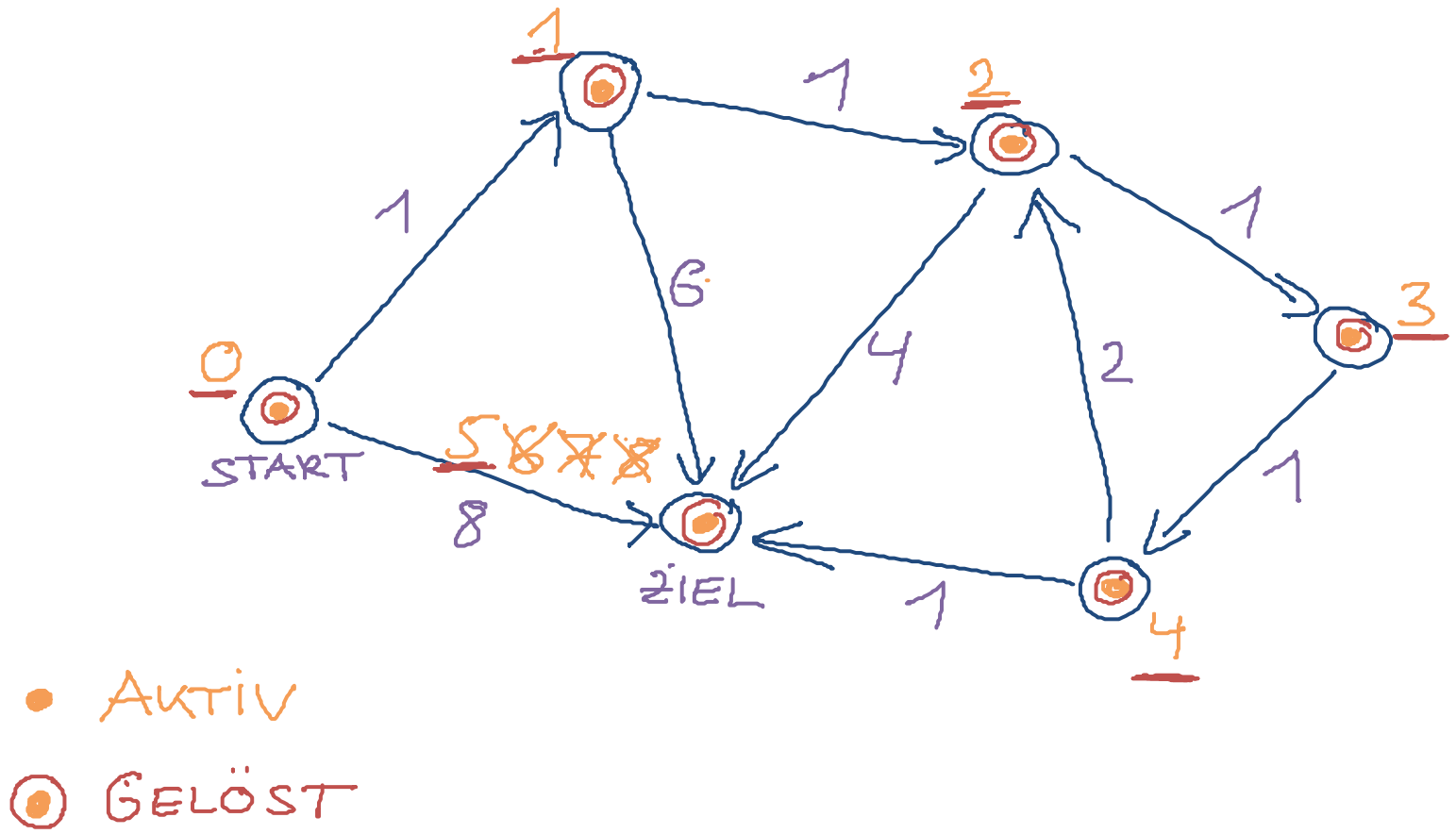
Dijkstras Algorithmus

UNI FREIBURG

- High-level Beschreibung des Algorithmus
  - Drei Arten von Knoten: gelöste, aktive und bisher unerreichte (auf Englisch: settled, active, unreached)
    - 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
  - 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

bei mehreren aktiven Knoten mit dem kleinsten Wert: irgendeinen

Dijkstras Algorithmus — Beispiel



Dijkstra — Korrektheitsbeweis 1/3

UNI FREIBURG

- Argumentationslinie
  - **Annahme 1:** Alle Kantenlängen sind ≥ 0, siehe Folie 18
  - **Annahme 2:** Die dist(s, u) sind **alle verschieden**
    - A2 erlaubt einen einfacheren und intuitiveren Beweis
    - Es geht auch ohne A2, siehe Referenzen (nur bei Interesse)
  - Mit A2 gibt es eine Anordnung u_1, u_2, u_3, ... der Knoten so dass gilt $dist(s, u_1) < dist(s, u_2) < dist(s, u_3) < \ldots$
  - 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

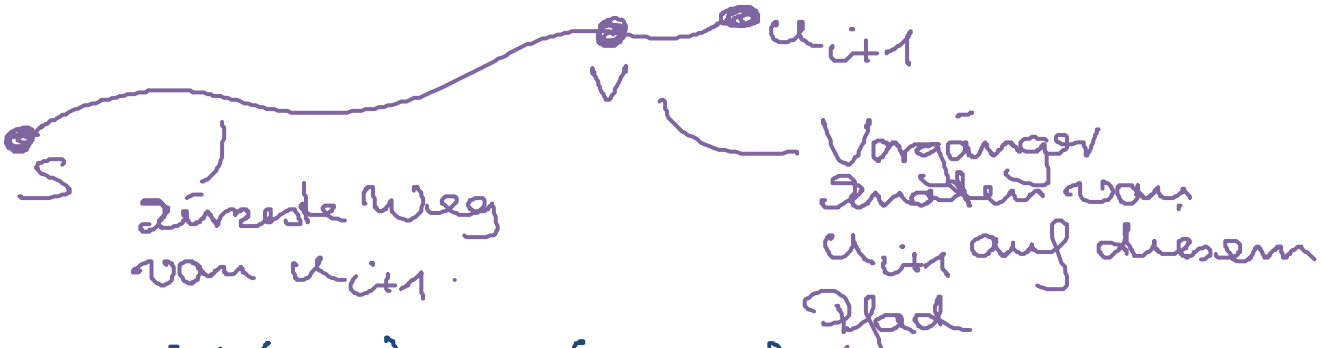
Dijkstra — Korrektheitsbeweis 2/3

Induktionsanfang: $i = 1$

$u_1 = s$; $dist(u_1) = 0 = dist(s, u_1)$ ☑

Induktionsschritt: $1, \dots, i \rightarrow i+1$

D.z. für $u_1, \dots, u_i$ stimmt die Aussage schon



$dist(s, u_{i+1}) = dist(s, v) + c(v, u_{i+1})$ Länge der Kante (v, u_{i+1})

$\Rightarrow dist(s, v) \leq dist(s, u_{i+1})$ weil $c(v, u_{i+1}) \geq 0$

$\Rightarrow$ v muss einer von den $u_1, \dots, u_i$ sein; nach A1 $v = u_j$

$\overset{IV}{\Rightarrow} dist(v) = dist(s, v)$ in Runde j und danach

$j \in \{1, \dots, i\}$

Dijkstra — Korrektheitsbeweis 3/3

Als in Runde j der Knoten v gelöst wurde wurde $dist(u_{i+1})$ auf $dist(v) + c(v, u_{i+1})$ oder kleiner gesetzt.

$$\Rightarrow dist(u_{i+1}) \leq dist(v) + c(v, u_{i+1})$$
$$= dist(s, v) + c(v, u_{i+1}) = dist(s, u_{i+1})$$

$$\Rightarrow dist(u_{i+1}) = dist(s, u_{i+1}) \text{ ab Runde } j.$$

$j > i+1$:

$$dist(u_j) \geq dist(s, u_j)$$
$$> dist(s, u_{i+1}) \quad \text{nach A2}$$
$$= dist(u_{i+1})$$

$\Rightarrow$ u_{i+1} hat den kleinsten dist-Wert unter $u_{i+1}, u_{i+2}, u_{i+3}, \ldots$ und wurde deshalb in Runde $i+1$ gelöst.

Dijkstras Algorithmus — Implementierung

UNI FREIBURG

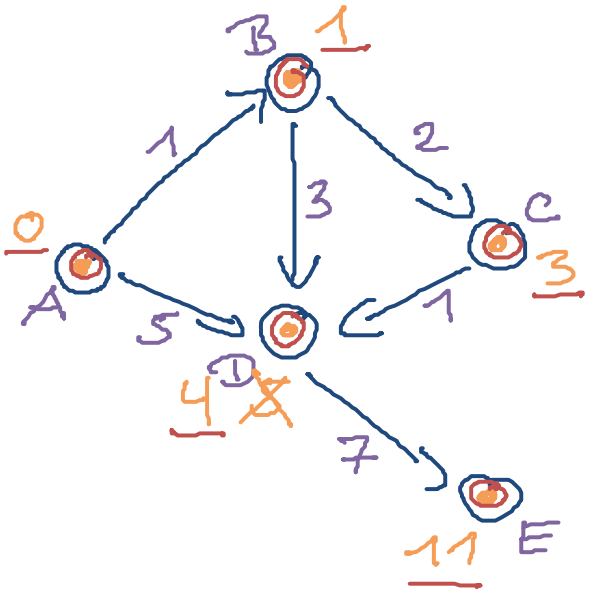
- Einige Hinweise (der Rest ist Übungsaufgabe)
  - Wir müssen die Menge der **aktiven Knoten** verwalten
  - Ganz am Anfang ist das nur der Startknoten
  - Am Anfang jeder Runde brauchen wir den **aktiven** Knoten u mit dem **kleinsten** Wert für dist(u)
  - Es bietet sich also an, die aktiven Knoten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

Problem Prioritätswarteschlange 1/2

UNI FREIBURG

- Wir hatten bisher nur insert, getMin und deleteMin
  - Mit so einer PW hat man gar keinen Zugriff auf ein **beliebiges** Element aus der PW (das könnte man dann löschen und mit neuem Schlüssel wieder einfügen)
  - **Alternative:** man fügt den Knoten einfach **nochmal** ein, mit dem neuen, niedrigeren dist Wert
  - 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**

Problem Prioritätswarteschlange 2/2



Für das 11-Blatt
können Sie Ihre eigene PW verwenden (ÜS)
oder die von JAVA
wie Sie möchten.

Dijkstras Algorithmus — Laufzeit

UNI FREIBURG

- Für einen Graph mit n Knoten und m Kanten, $m \geq 1$
  - 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 ($m \sim n^2$) Graphen ist das klar besser
    - in der Praxis ist aber oft $m = O(n)$ und dann ist der einfache Binary Heap ohne decrease key die bessere Wahl

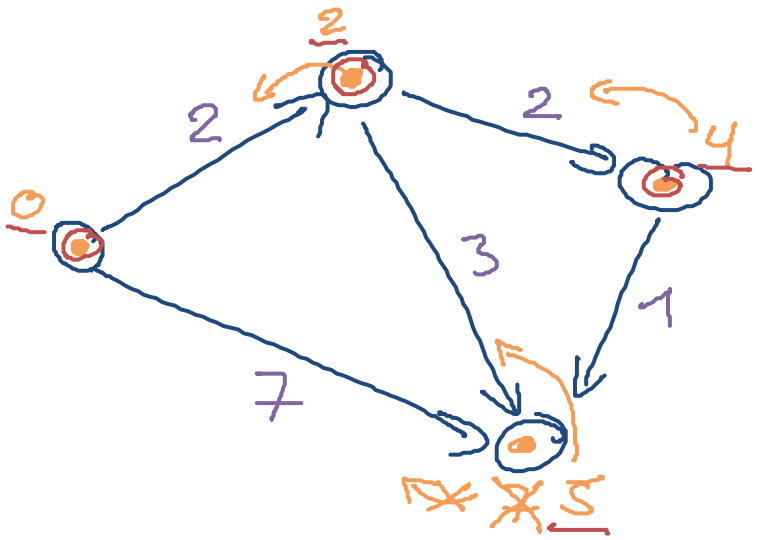
Dijkstra — Kommentare 1/3

UNI FREIBURG

- 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

Dijkstra — Kommentare 2/3

- Berechnung der kürzesten Pfade
  - So wie wir ihn bisher beschrieben haben, berechnet Dijkstras Algorithmus 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**



Wenn man die
↳ alle umdreht
bekommt man
einen Baum von s aus.

Dijkstra — Kommentare 3/3

UNI FREIBURG

- Erweiterungen
  - In unserem Beweis haben wir benutzt, dass die Kantenlängen alle **nicht-negativ** 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 + 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**:
    - zusätzlich gegeben: $h(u)$ = Schätzwert für $dist(u, t)$

Literatur / Links

- 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

    http://hr.wikipedia.org/wiki/Dijkstrin_algoritam