max planck institut informatik

Ideen und Konzepte der Informatik

Kürzeste und schnellste Wege

Wie funktioniert ein Navi?

Kurt Mehlhorn

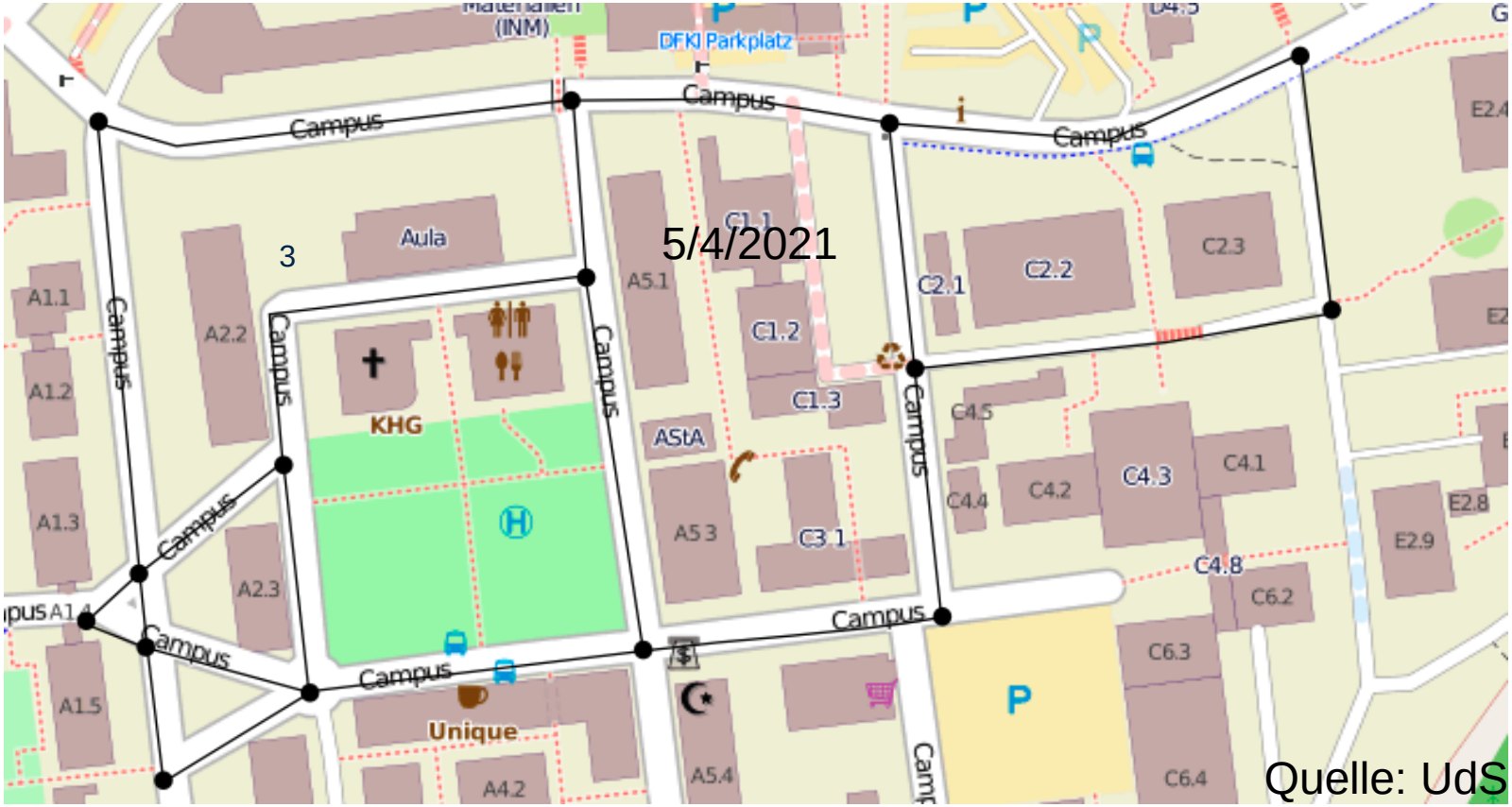
max planck institut informatik

Schnellste Wege – Routen finden im Navi

- Karten und Graphen
- Schnellste und kürzeste Wege sind das gleiche Problem; Länge einer Verbindung in Metern oder Sekunden. Schnell = kurz bzgl. Zeit.
- Algorithmen für schnellste Wege
  - Erster Versuch
  - Dijkstras Algorithmus
- Schnellste Wege in Straßengraphen

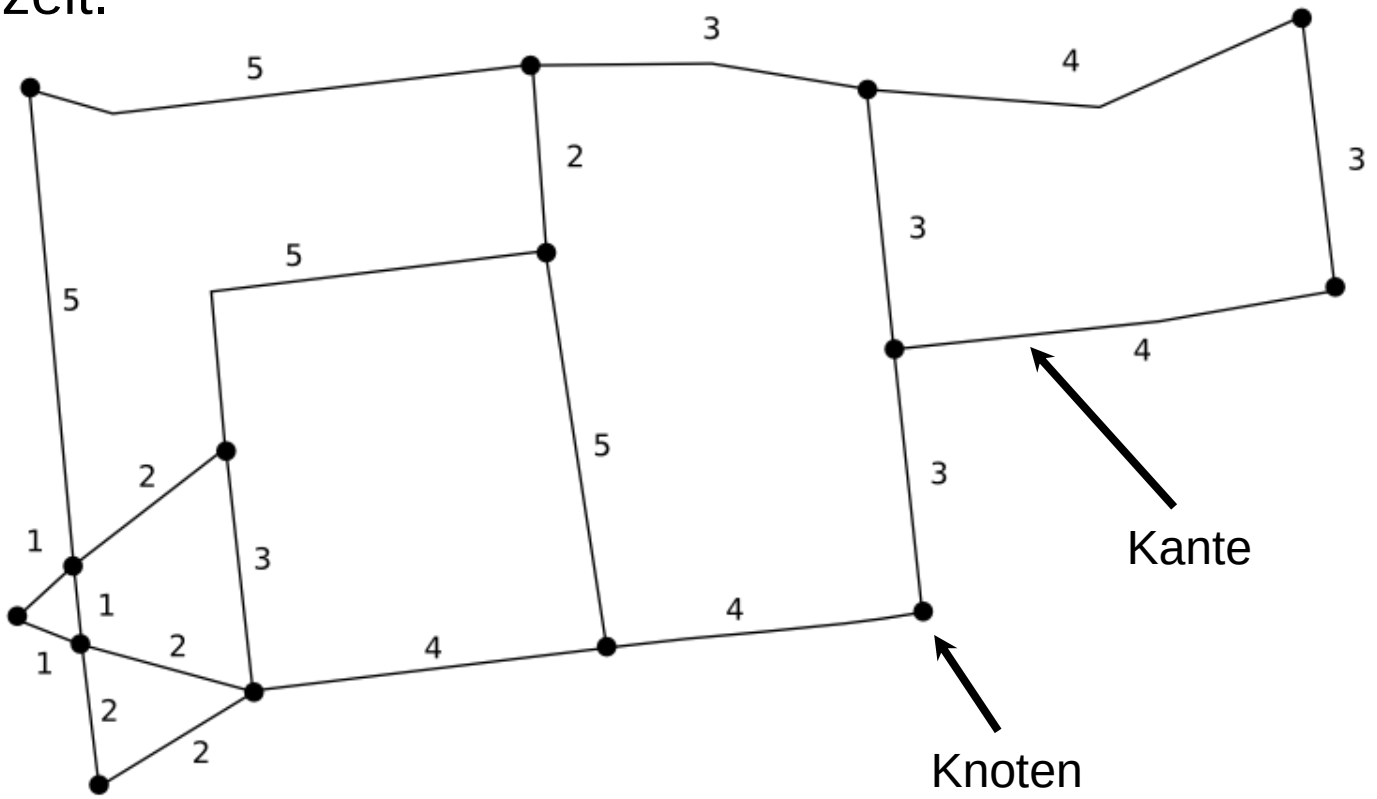
Landkarten

Landkarten enthalten sehr viele Informationen; nur Straßengraph ist wichtig.



Graphen als Abstraktion

Graphen bestehen aus Knoten und Kanten. Jede Kante hat eine Fahrzeit.



Graph G = (V,E), Knoten V, Kanten E, E Teilmenge V x V

Straßennetzwerke

- Europa: 24 Millionen Knoten, 58 Millionen Kanten.
- Schnellste Wege kann man trotzdem in Sekunden berechnen,
- Oder in Millisekunden nachschlagen.
- Algorithmen arbeiten auf dem Graphen und zeigen das Ergebnis auf der Karte.
- Fahrzeit über eine Kante:
  - Länge/angenommene Durchschnittsgeschwindigkeit.
  - Tatsächlich bekannt durch Beobachtung.

Algorithmen für schnellste Wege: Grundidee

Wenn ich vom Startknoten in 30 Minuten nach A komme

und

von A nach B in 5 Minuten,

dann komme ich in 35 Minuten vom Startknoten nach B.

Ein erster Algorithmus

1. Fahrzeit von Start nach Start = 0 Minuten.

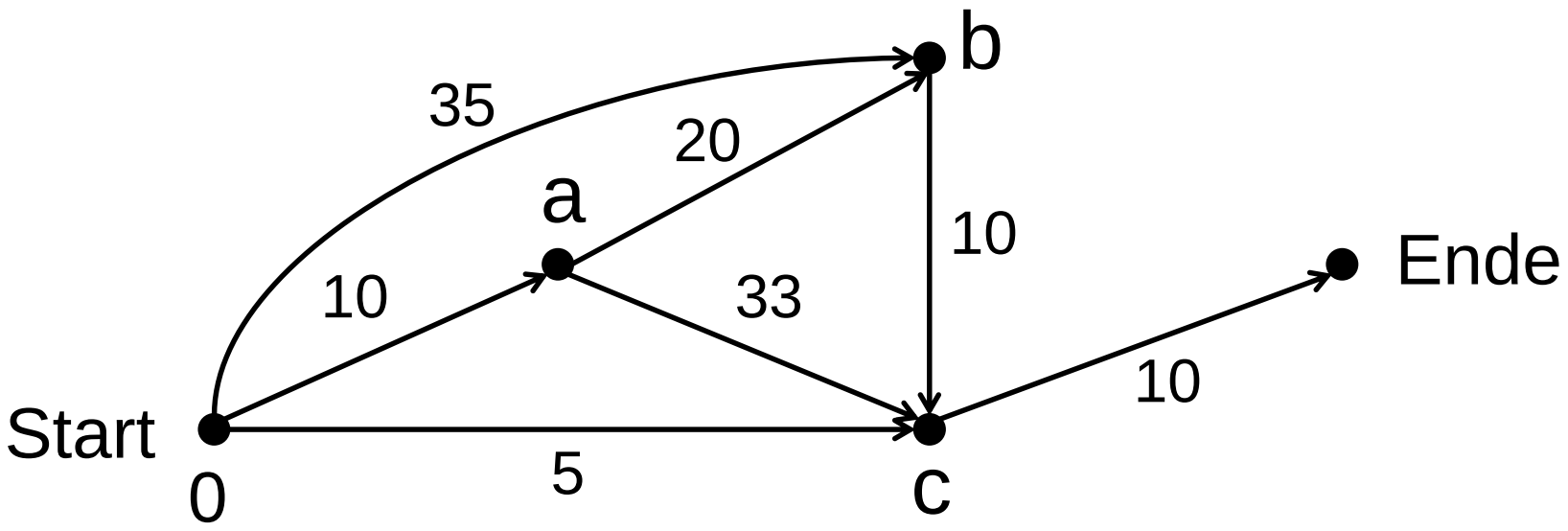
2. Für alle anderen Knoten weiß ich noch keinen Weg: also Fahrzeit = unendlich.

3. Falls Fahrzeit nach A = X Minuten und Straße A → B braucht Y Minuten, dann

   Fahrzeit nach B = min(X+Y, schon bekannte Fahrzeit nach B).

4. Wiederhole 3. solange noch eine Fahrzeit verbessert werden kann.

Beispiel



Ein erster Algorithmus (Pseudocode)

Setze dist[s] ← 0 und dist[v] ← ∞ für v ≠ s

Solange es eine Kante (u,v) gibt mit dist[u] + zeit(u,v) < dist[v]

    setze dist[v] auf dist[u] + zeit(u,v)

- Dabei, zeit(u,v) = Fahrzeit über die Kante von u nach v
- u, v sind Namen für Knoten
- dist[v] = beste bekannte Fahrzeit von s nach v
- s = Startknoten
- (u,v) heißt verbessernde Kante

Fragen

- Finden wir so immer den schnellsten Weg vom Startknoten s zu allen anderen Knoten? JA
- Pfad und nicht nur Fahrzeit.
- Wie lange dauert das? Kann sehr lange dauern.

Details auf den nächsten Folien

  - Laufzeit: exponentielle Laufzeit bei schlechter Wahl der verbessernden Kante
  - Pfad mitberechnen
  - Korrektheit: das ist etwas knifflig.

Wir können auch den Pfad mitberechnen

Setze dist[s] ← 0 und Pfad[s] ← (s) (Pfad, der nur aus s besteht)
Für v ≠ s, setze dist[v] ← ∞ und Pfad[v] ← kein Pfad

Solange es eine verbessernde Kante (u,v) gibt,
setze dist[v] auf dist[u] + zeit(u,v) und
Pfad[v] auf Pfad[u] verlängert um die Kante (u,v).

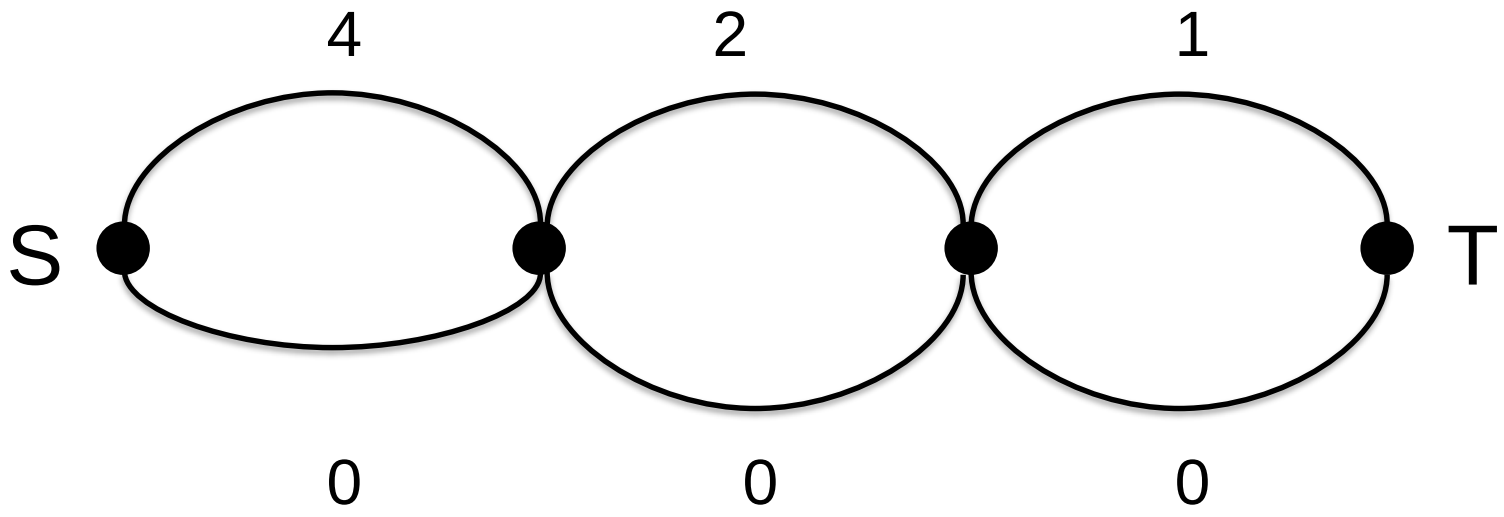
Beobachtung:

• Pfad[v] ist immer ein Pfad der Fahrzeit dist[v] von s nach v.

• Pfad[v] ist immer ein einfacher Pfad von s nach v, d.h. ein Pfad, in dem sich kein Knoten wiederholt.

• Der Algorithmus hält immer an, da es nur endlich viele einfache Pfade gibt.

Effizienz



- Fahrzeit nach T wird 8 mal geändert
- Ein Ort mehr: Laufzeit verdoppelt sich

Exponentielles Wachstum ist ein Killer

- 4 Orte: 8 Änderungen
- 5 Orte: 16 Änderungen
- 6 Orte: 32
- 7 Orte: 64
- 21 Orte: mehr als 1.000.000
- 41 Orte: mehr als 1.000.000.000.000

Mein Rechner kann 10^9 Operationen/sec.

Korrektheit

- Warum sollte man immer argumentieren, dass ein Algorithmus tatsächlich die Aufgabe löst, die er angeblich löst?
- Intellektuelle Redlichkeit
- Inkorrekte Algorithmen können großen Schaden anrichten
  - Ariane 5, Therac-25, AT&T crash of long-distance network, to keep a Boeing Dreamliner flying, reboot every 248 days, Boeing 737 MAX crashes.
  - Wikipedia: Liste von Programmfehlerbeispielen
  - Kosten für deutsche Industrie, 80 Mrd/Jahr

Korrektheit

Wissen schon: immer, dist[v] ≥ kürzeste Fahrzeit von s nach v und Alg hält an.

Beh: Wenn Alg anhält, dann gilt für alle Knoten v: dist[v] = Zeit[v] = kürzeste Fahrzeit von s nach v

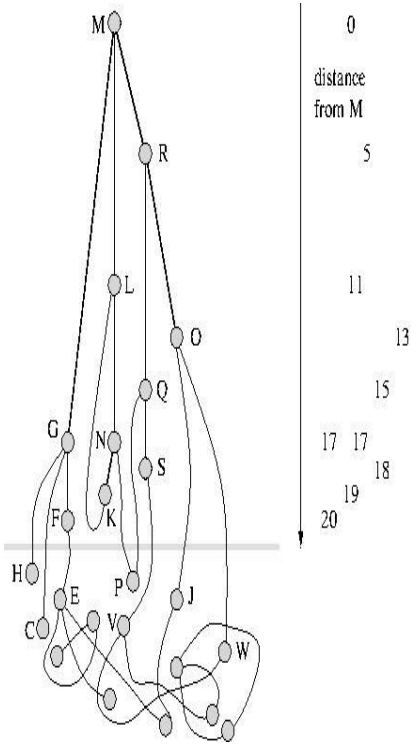
- Annahme: es gibt einen Knoten y für den dist[y] nie auf Zeit[y] verringert wird.
- y ist nicht der Knoten s.
- Betrachte schnellsten Weg von s nach y. Für s, wird dist[s] auf Zeit[s] gesetzt, für y ist das nicht der Fall.
- Also gibt es aufeinanderfolgende Knoten x und z auf diesem Weg, so dass dist[x] auf Zeit[x] gesetzt wird, aber dist[z] größer als Zeit[z] bleibt. Dann

  dist[z] > Zeit[z] = Zeit[x] + Zeit(x,z) = dist[x] + Zeit(x,z)

  und die Kante (x,z) ist verbessernd. Solange (x,z) verbessern ist, hält der Alg nicht an. Aber der Alg hält an. Aber Alg hält.

Das Fadenexperiment

- Ann: Alle Kanten sind in beide Richtungen benutzbar.
- Wir bauen ein Modell unseres Graphen
  - Knoten = Punkte mit Gewicht
  - Kanten = Fäden entsprechender Länge.
- Wir heben das Geflecht am Startknoten hoch.
- Was passiert mit den anderen Knoten? Wie weit liegen sie unter dem Startknoten?



Quelle: Peter Sanders

Das Fadenexperiment (dynamisch)

- Wir bauen das Modell schrittweise.
- Am Anfang sind alle Fäden unendlich lang. Wir hängen den Startknoten an die Decke.
- Dann verkürzen wir alle Fäden, die am Startknoten befestigt werden, auf ihre Länge.
- Gibt es einen Knoten außer dem Startknoten, der schon seine Endlage erreicht hat? Welcher Knoten ist das ?
- Nun kürzen wir alle Fäden ein, die an diesem Knoten befestigt sind. Und so weiter.

Geschicktes Auswählen

1. Nach s in 0 Min.
2. Wenn A in X Min. und A → B braucht Y Min., dann B in X+Y Min.
3. Wiederhole 2. solange nicht stabil

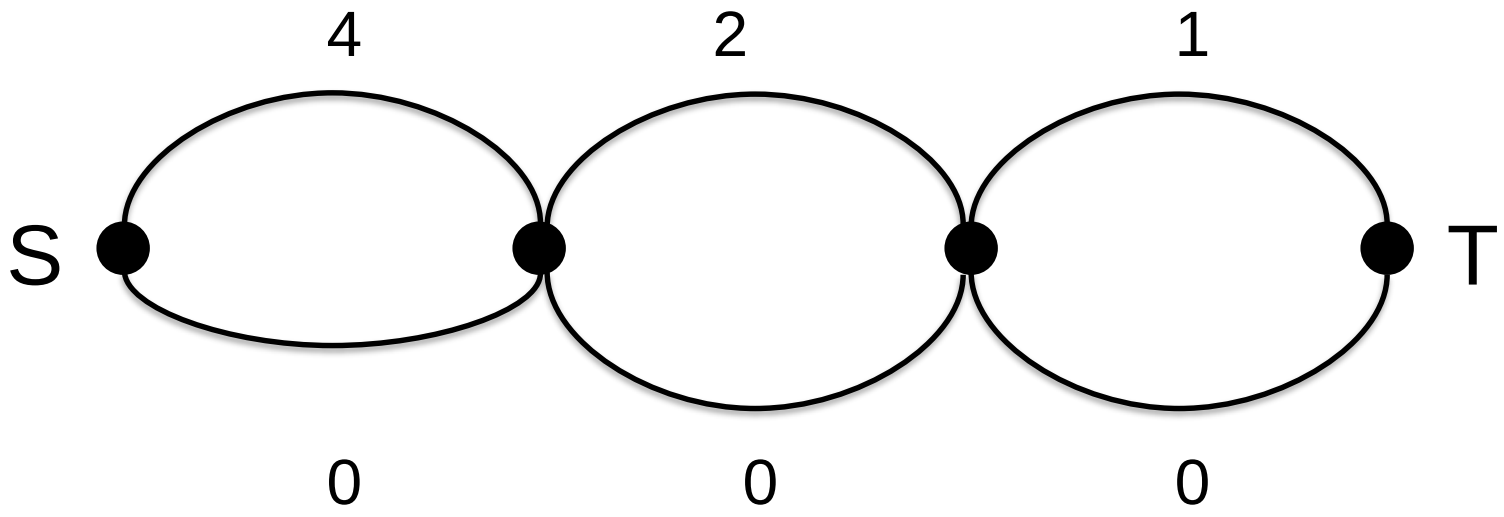
Wende 2) immer auf alle Kanten A → B an, die aus dem Knoten A mit der kleinsten Fahrzeit herausgehen (auf den wir das nicht schon vorher angewendet haben).

Dijkstras Algorithmus *(1959)*

Turing Award *(1972)*

Source: Wikipedia, CC BY-SA3.0

Beispiel



Fahrzeit wird über jede Kante
nur einmal propagiert

Pseudocode

Dijkstra: **dist[v] = beste bekannte Fahrzeit nach v**
Knoten v rot = v hat noch nicht propagiert

dist[s] ← 0 und färbe s rot
für alle v ≠ s:
    dist[v] ← unendlich und färbe v rot
solange es einen roten Knoten gibt:
    u ← der rote Knoten mit kleinstem Wert dist[u]
    färbe u schwarz
    für alle Kanten (u,v) tue:
        falls dist[u] + Zeit(u,v) < dist[v]:
            dist[v] ← dist[u] + Zeit(u,v)

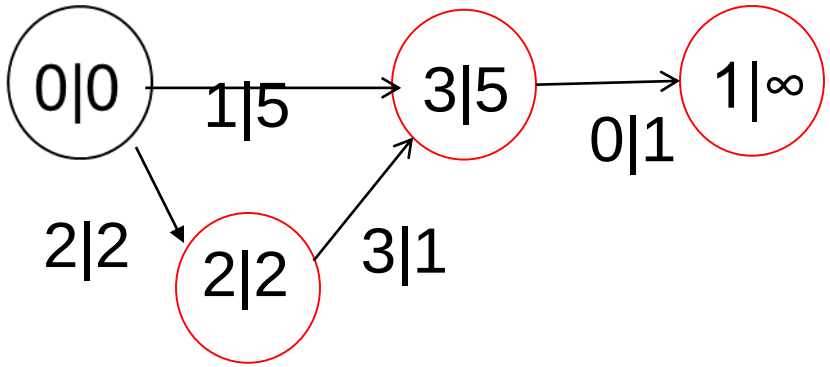
Vom Pseudocode zum Programm

- Der Pseudocode ist für Menschen verständlich, aber nicht für Computer.
- Wir geben Knoten Farben, wir suchen nach dem roten Knoten mit dem kleinsten dist-Wert, wir tun etwas für alle Kanten, die aus einem Knoten ausgehen.
- Wie sagen wir es einem Computer?

Graphen im Computer I

- Graph mit n Knoten und m Kanten: wir bezeichnen die Knoten mit den Zahlen 0 bis n – 1 und die Kanten mit 0 bis m – 1.
- Attribute von Knoten und Kanten, etwa Farbe, dist, Fahrzeit, … Folge von n (m) Speicherzellen: i-te Zelle enthält die Information zum Knoten i (zur Kante i).

Graphen im Computer II



Knotennummer | Dist

Kantennummer | Fahrzeit

Knoten	0	1	2	3
Kanten				
Farbe				
Dist				

Kante	0	1	2	3
Länge				

Anmerkungen

- u ← der rote Knoten mit kleinstem Wert dist[u]

ist ausführlicher

mindist ← unendlich
Für i = 0 bis n – 1 tue
    falls (farbe[i] = rot und dist[i] < mindist)
    dann mindist ← dist[i]; u ← i;

- Pfade mitberechnen, Wellenausbreitung

Laufzeit (wieviele Knoten und Kanten betrachten wir?)

n = #Knoten, m = #Kanten,
m_u = Kanten aus u heraus

Dijkstra(s):

dist[s] ← 0, färbe s rot

für alle v ≠ s: } ~n

    dist[v] ← unendlich, färbe v rot

solange es aktiven Knoten gibt:

~n {
- u ← der rote Knoten mit kleinstem Wert dist[u]
- färbe u schwarz

~m_u {
- für alle Kanten (u,v) tue:
    - falls dist[u] + Zeit(u,v) < dist[v]:
        - dist[v] ← dist[u] + Zeit(u,v)

Laufzeit

- n mal Minimum finden: n mal n = n^2
- Alle Kanten verfolgen: m

Insgesamt: $m + n^2$

Kann verbessert werden auf $m + n \log n$

und braucht dann etwa 10 sec für Europa

Korrektheit

Ähnlich zum Grundalgorithmus, aber etwas komplexer.

Siehe nächste Folie.

Wenn Sie eine mathematische Ader haben, sollten Sie versuchen, das Argument nachzuvollziehen, sonst überspringen Sie die Folie.

Wir zeigen: wenn wir einen Knoten y schwarz färben, gilt:

dist[y] = schnellste Fahrzeit nach y.

Korrektheit (Beweis durch Widerspruch)

Annahme: y ist der erste Knoten, für den dist[y] > Zeit[y] = schnellste Fahrzeit nach y, wenn wir ihn schwarz färben.

Betrachte schnellsten Weg von s nach y. Sei z der erste rote Knoten auf diesem Weg unmittelbar bevor wir y schwarz färben und sei x sein Vörgänger. y = z ist möglich.

x existiert, da s der erste Knoten ist, der schwarz gefärbt wird.

Auch dist[x] = Zeit[x], wenn wir x umfärben, da wir x vor y schwärzen.

Wenn wir x schwärzen, setzen wir dist[z] auf dist[x] + Zeit(x,z) = Zeit[z].

Schließlich dist[z] = Zeit[z] ≤ Zeit[y] < dist[y], also wird z gewählt und nicht y.

Navigationssysteme

- Navigationssysteme im Auto benutzen Dijkstra + Straßenhierarchie.

- Auskunftssysteme (etwa Google Maps) berechnen Antworten zum Teil vor.

Nachschlagen statt Rechnen

- Idee: Alle Wege vorberechnen
  - Europa: $24 \cdot 10^6$ Knoten
  - 24 Millionen mal Dijkstra:

24 mal 10^6 mal 10 Sekunden; das ist etwas unter 8 Jahren.

- So einfach geht es nicht.

Transitknoten (Bast-Funke)

- Kurze Wege: on-the-fly mit Dijkstra
- Alle weiten Wege passieren eine kleine Menge von Transitknoten.
  - Gesamtmenge der Transitknoten ist klein.
  - Für jeden festen Startpunkt gibt es nur sehr wenige Transitknoten.

Hannah Bast, Stefan Funke, Saar LB Preis

Hannah ist jetzt Professorin in Freiburg, Stefan Prof in Stuttgart.

Bildquellen: privat

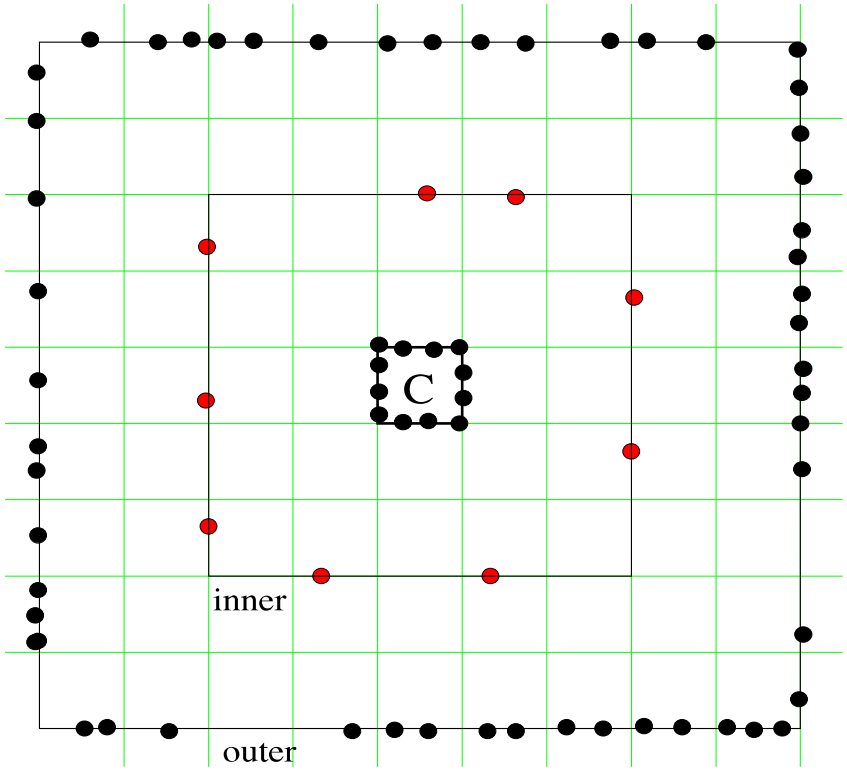
Transitknoten

Transitknoten

- KM wohnt in Scheidt
- Meine Transitknoten
  - nach Osten: Autobahnauffahrt St. Ingbert
  - Nach Süden: Kleinblittersdorf
  - Nach Westen: Goldene Bremm
  - Nach Nord-Westen: Stadtautobahn
  - Nach Norden und Nordosten: Autobahn Sulzbach
- Alle Bewohner von Scheidt benutzen die gleichen Transitknoten

Transitknoten

- Gitter über die Welt legen, Punkt = Kreuzung von Gitter und Straße.
- Für jede Zelle C Wege nach „outer” ausrechnen.
- Die Knoten aus „inner”, die benutzt werden sind Transitknoten für C.



Vorberechnen

- Europa: 24 Mio. Knoten, 10 000 Transitknoten.
- Schnellste Wege von jedem Knoten zu seinen Transitknoten:
  - Ungefähr 10/Knoten, ungefähr $24 \cdot 10^7$ Wege.
- Schnellste Wege zwischen allen Paaren von Transitknoten:
  - Ungefähr 10 000 Knoten, ungefähr 10^8 Wege.
- Passt auf einen Server!

Wege finden

- Kurze Wege: Dijkstra
- Lange Wege: A nach B
  - Konzeptuell zerlegen in A – T_1 – T_2 – B wobei T_1 Transitknoten für A und T_2 für B.
  - Probiere alle Möglichkeiten für T_1 und T_2 (jeweils etwa 10) und bestimme den minimalen Wert von

    dist(A, T_1) + dist(T_1, T_2) + dist(T_2, B).
  - Das sind 10 + 10 + 100 Speicherzugriffe.

Zusammenfassung

- Dijkstra ist ein sehr schneller Algorithmus für kürzeste Wege (wenige Sekunden in Graph mit 10 Mio. Knoten und 30 Mio. Kanten).
- Straßengraphen haben Struktur (Hierarchie der Straßen, Fast-Planarität); Struktur vereinfacht das Problem.
- Vorberechnen ist eine gute Idee, wenn man viele Anfragen zu beantworten hat.