max planck institut informatik

Ideen und Konzepte der Informatik

Kürzeste und Schnellste Wege

Wie funktioniert ein Navi?

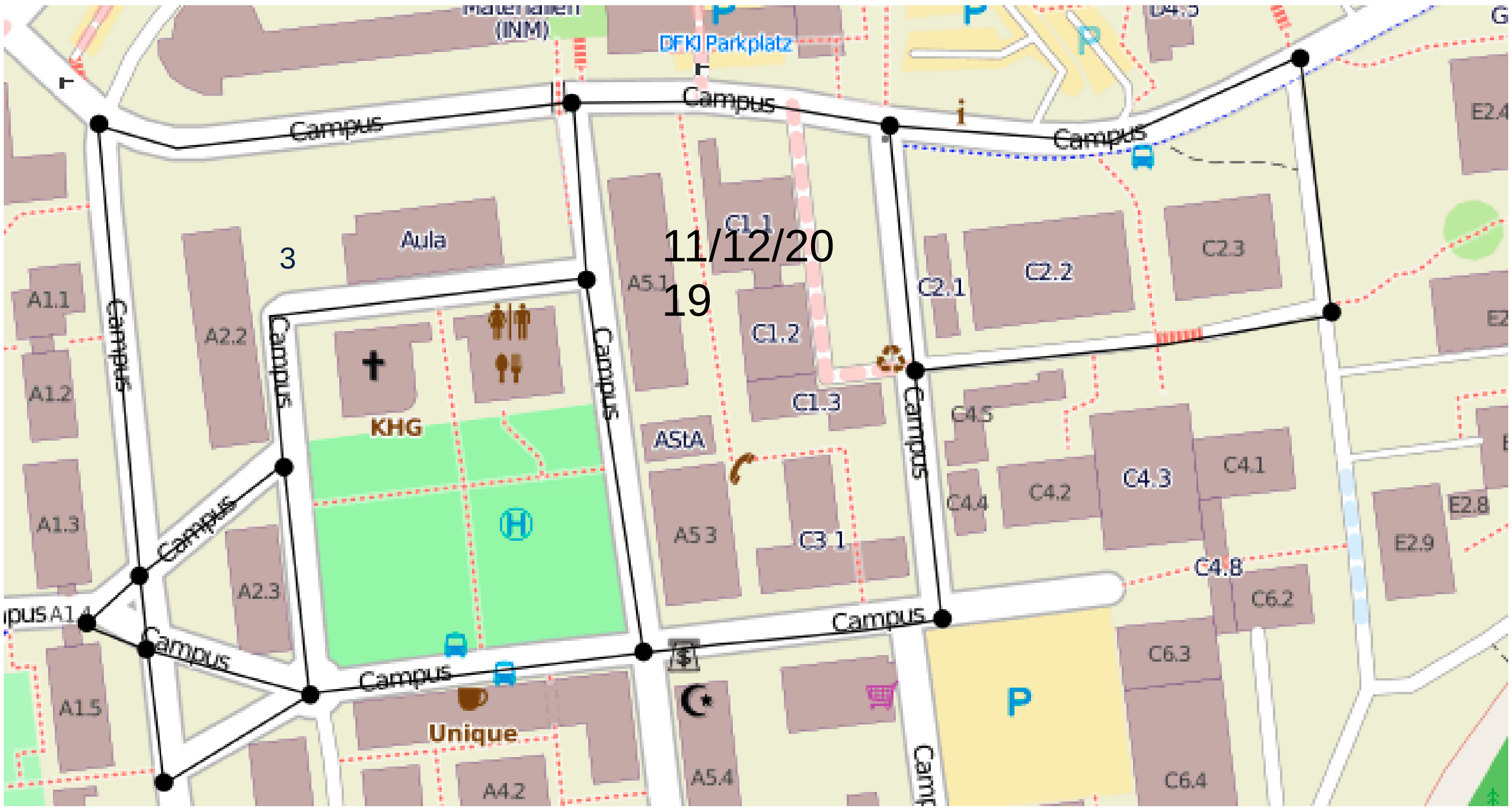
Kurt Mehlhorn

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

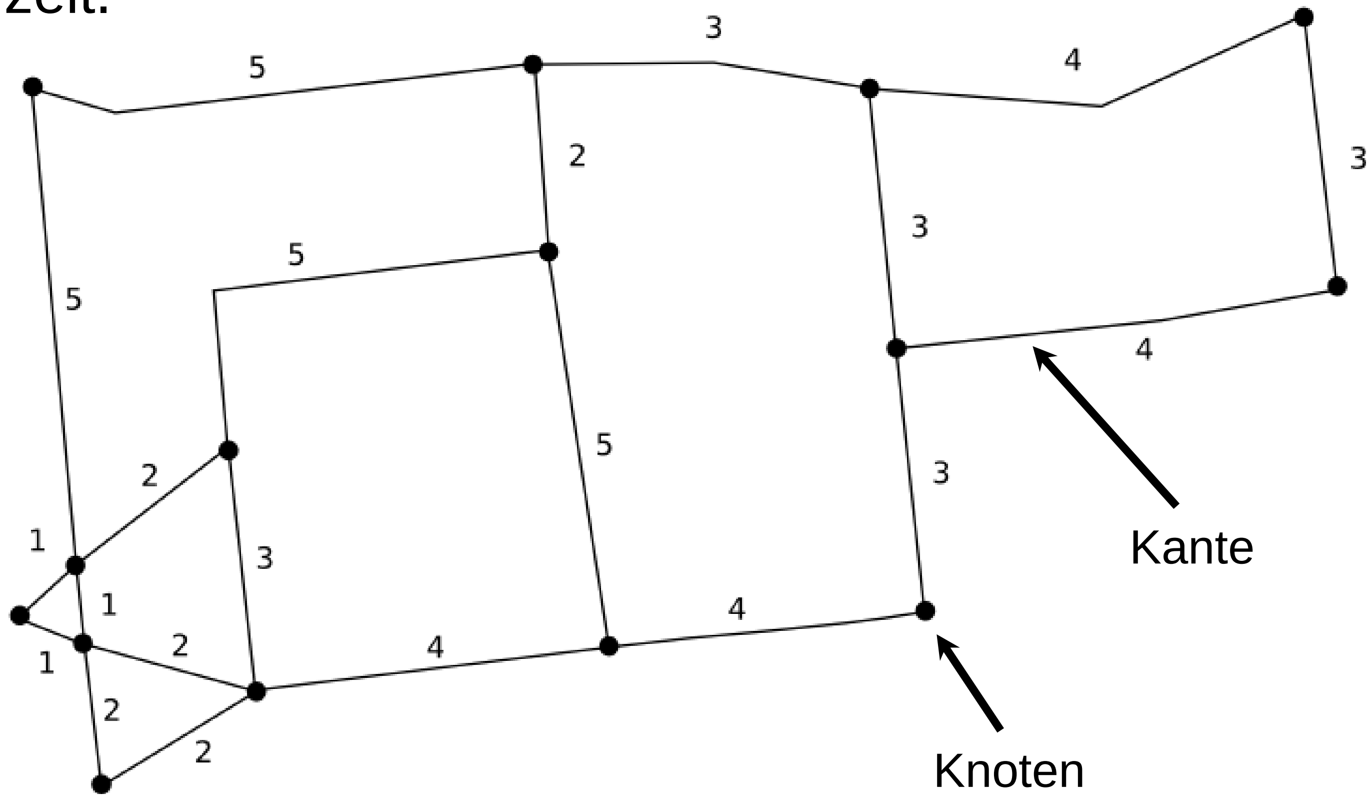
Landkarten

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.

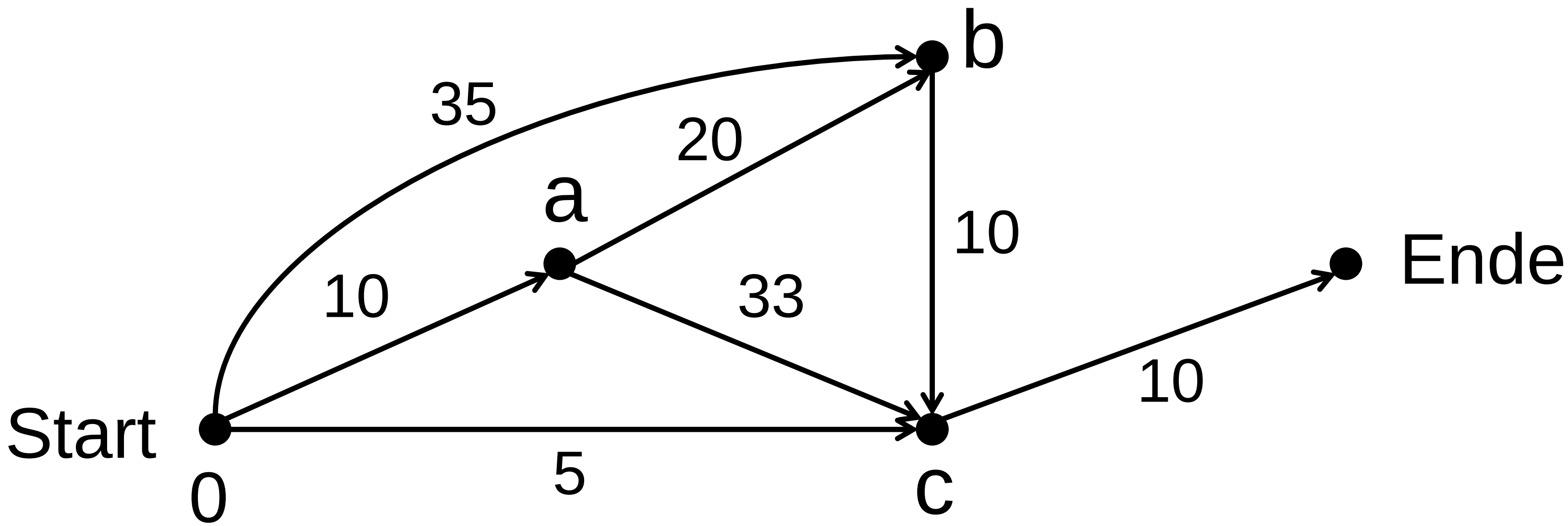
2. Für alle anderen Knoten weiß ich noch keinen Weg: also Fahrzeit = unendlich.

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] $\leftarrow$ 0 und dist[v] $\leftarrow \infty$ für $v \neq s$

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

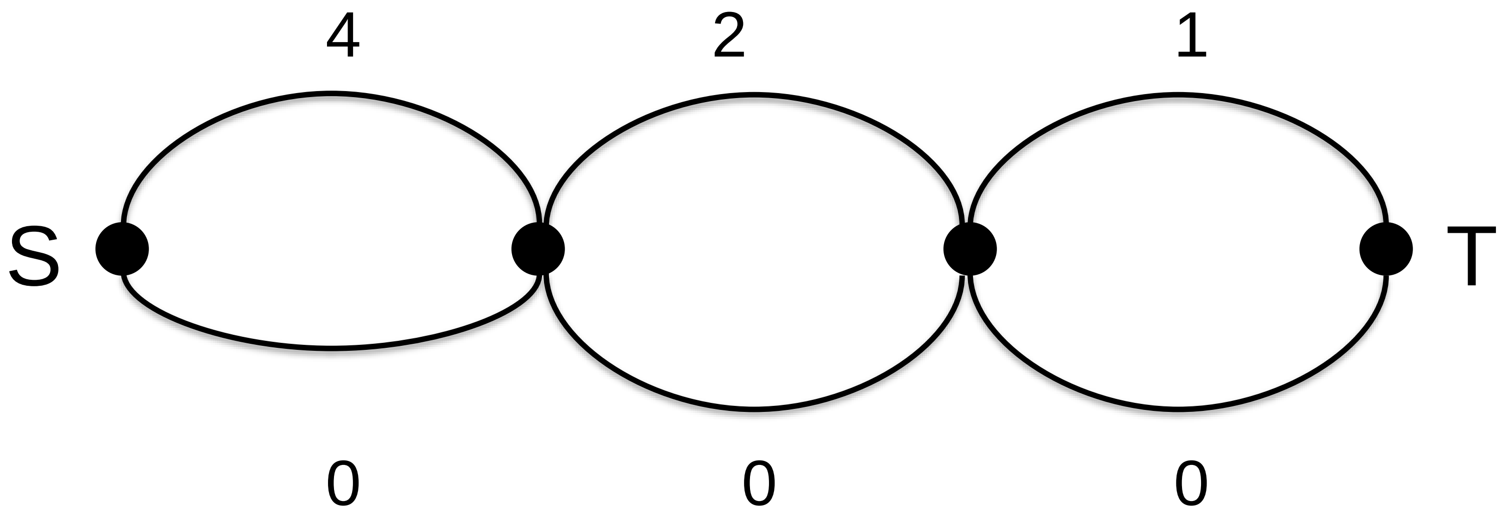
Fragen

- Finden wir so immer den schnellsten Weg vom Startknoten s zu allen anderen Knoten? JA

- Wie lange dauert das? Kann sehr lange dauern.

- Details auf den nächsten Folien
  - Warnung: Korrektheit ist ein bisschen knifflich. Die nächsten beiden Folien.
  - Laufzeit ist einfacher.

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.

Wir können auch den Pfad mitberechnen

Setze dist[s] $\leftarrow$ 0 und Pfad[s] $\leftarrow$ (s)   (Pfad, der nur aus s besteht)
Für v $\neq$ s, setze dist[v] $\leftarrow \infty$ und Pfad[v] $\leftarrow$ 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 den Knoten v.

Beobachtung:

• Pfad[v] ist immer ein Pfad der Länge 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.

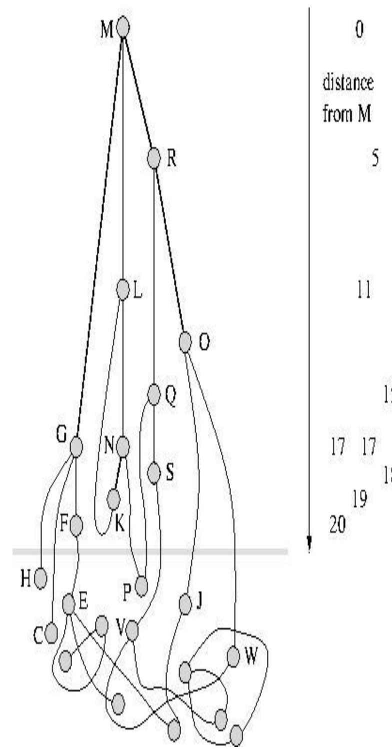
• Der Algorithmus hält immer an, da es nur endlich viele einfache Pfade gibt.

Korrektheit

- Beh: Immer gilt: dist[v] ≥ kürzeste Fahrzeit von s nach v. Wenn Alg anhält, dann dist[v] = kürzeste Fahrzeit von s nach v für alle v.
- Richtig für v = s, da dist[s] auf 0 gesetzt wird.
- Falls v ≠ s, betrachte schnellsten Weg von s nach v, etwa $s \to a \to b \to c \to v$.
- Wir wissen bereits, dass der Alg anhält. Solange dist[a] > kürzeste Fahrzeit von s nach a, ist (s,a) eine verbessernde Kante und der Alg hält nicht an. Also irgendwann dist[a] = ... Dann genauso für b, ...

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?



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

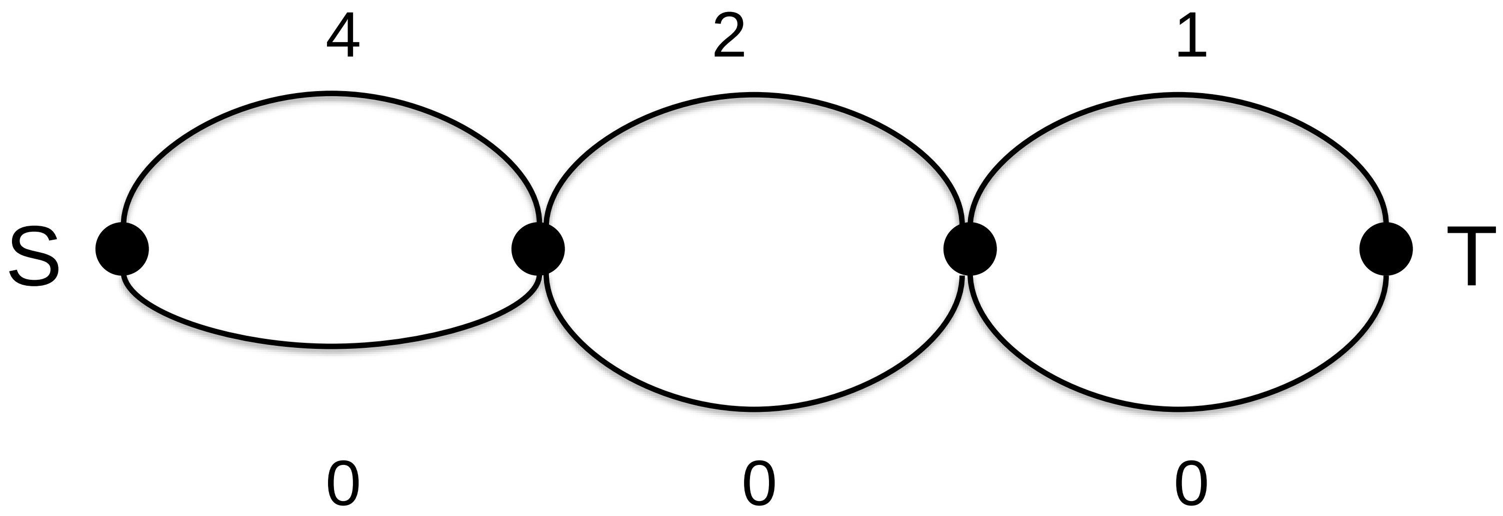
1. Nach s in 0 Min.
2. Wenn A in X Min. und A → B braucht Y Min., dann B in X+Y Min.
3. Wiederhole 2. solange nicht stabil

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)*

Beispiel



Fahrzeit wird über jede Kante
nur einmal propagiert

Pseudocode

Dijkstra(s): **dist[v] = beste bekannte Fahrzeit nach v**

dist[s] $\leftarrow$ 0 und färbe s rot

für alle v $\neq$ s:

    dist[v] $\leftarrow$ unendlich und färbe v rot

solange es einen roten Knoten gibt:

    u $\leftarrow$ 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] $\leftarrow$ dist[u] + Zeit(u,v)

Vom Pseudocode zum Programm

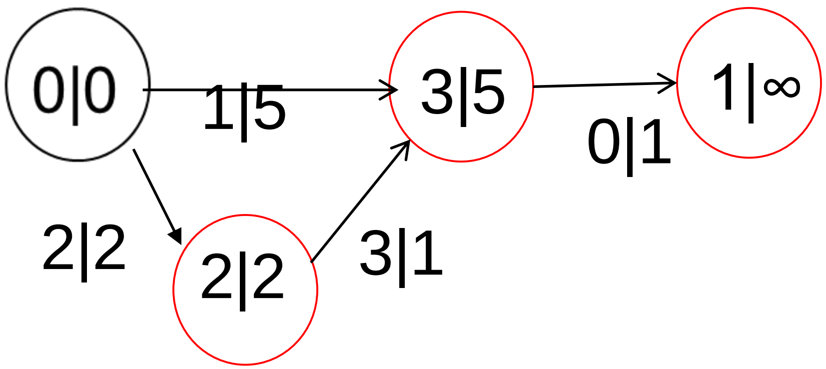
- Der Pseudocode ist für Menschen verständlich, aber nicht für Computer.
- Wir geben Knoten Farben, wir suchen nach dem roten Knoten mit dem kleinsten dist-Wert, wir tun etwas für alle Kanten, die aus einem Knoten ausgehen.
- Wie sagen wir es einem Computer?

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

	0	1	2	3
Knoten				
Kanten				
Farbe				
Dist				

	0	1	2	3
Kante				
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:

    dist[v] ← unendlich, färbe v rot   ~n

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

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.

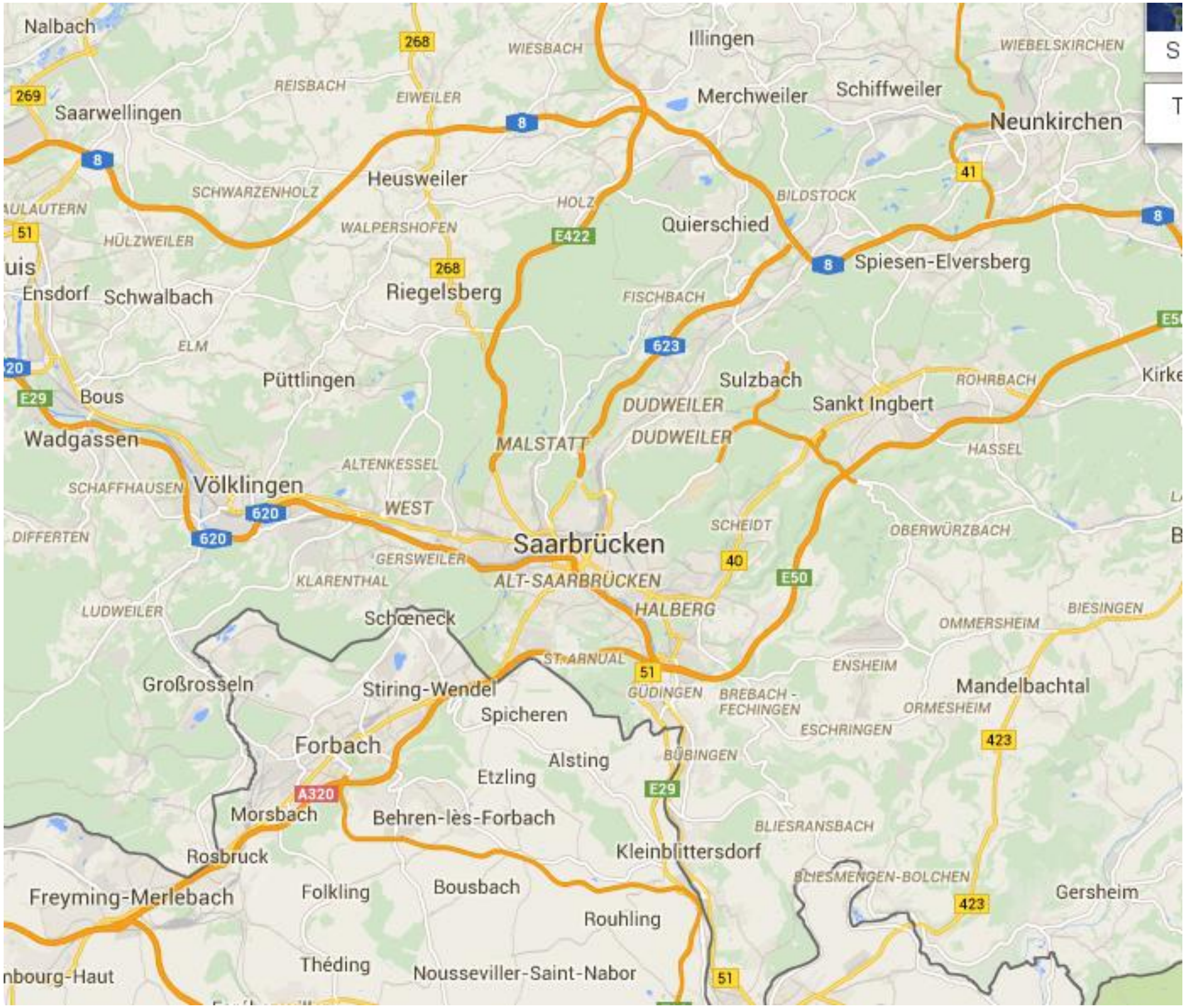
Transitknoten (Bast-Funke)

- 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.

Transitknoten

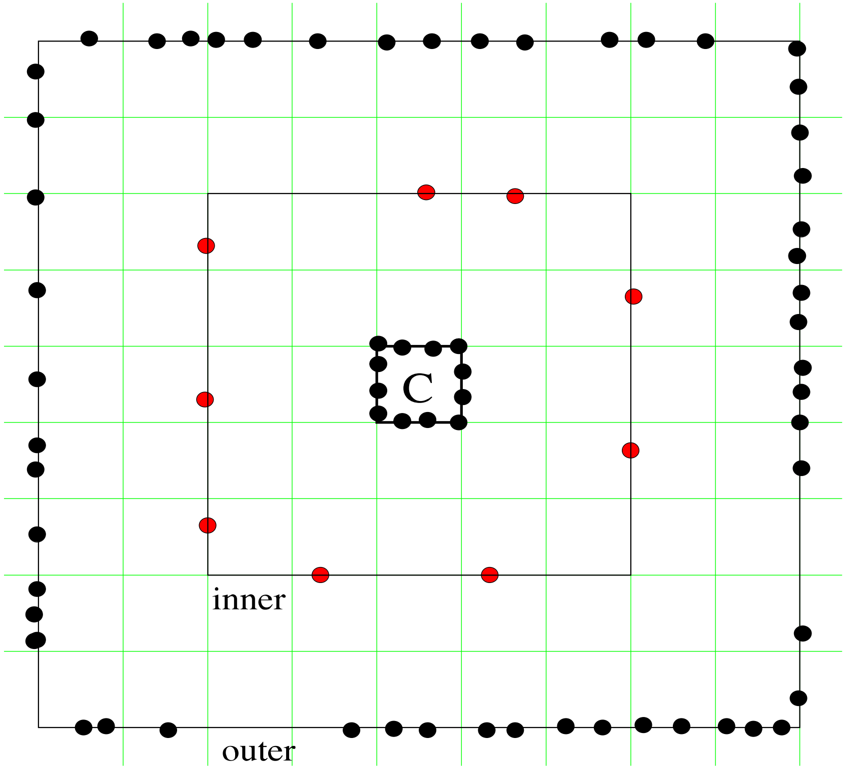


Transitknoten

- 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
  – Konyeptuelle 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

    $\mathrm{dist}(A, T_1) + \mathrm{dist}(T_1, T_2) + \mathrm{dist}(T_2, B)$.

  – 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.