Kürzeste Wege

Navigationssysteme im Auto berechnen schnell den kürzesten Weg zwischen Start- und Zielort.

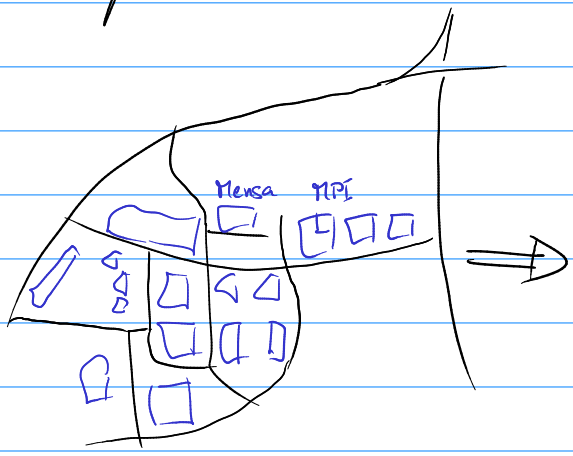
Heute: Wie finden Computer kurze Wege

Menschen können das recht gut und das Verfahren, das wir benutzen ist auch für Computer gut geeignet.

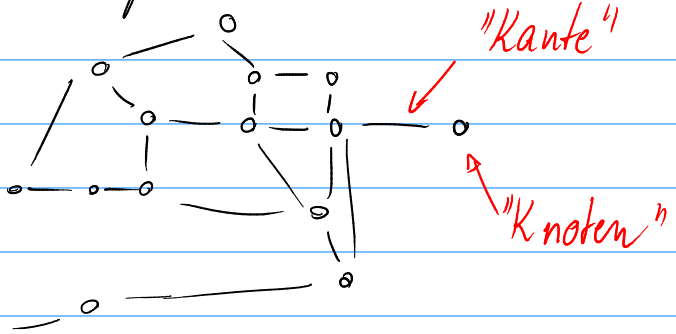
Karten:

Computer können schlecht Karten lesen, die für Menschen sind.
→ Bilder sind schwierig

Campus:



Stattdessen: Einfache Repräsentation
Kreuzungen → Punkte
Straßen → Linien



Abgespeichert als Adjazenzliste

u: $u'_1, u'_2 \dots$
v: $v'_1, v'_2 \dots$
⋮

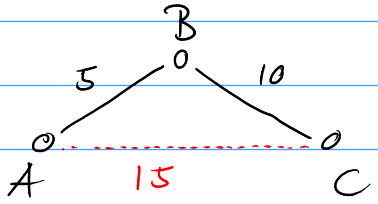
"Graph", "Netzwerk"

Für jede Kante speichern wir auch die Fahrzeit

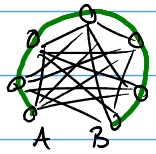
Straßennetzwerke sind groß! Kurze Wege kann man trotzdem in Sekunden berechnen.

Europa: 24 Millionen Knoten
58 Millionen Kanten

Ideen



Wenn ich in 5 Minuten von A nach B komme und in 10 Minuten von B nach C, dann komme ich in 15 Minuten von A nach C



Autobahn
Trampelpfad

→ Systematisches durchprobieren aller Wege dauert ewig
z.B. alle Wege mit 1, 2, 3 ... Kanten nacheinander

Beobachtung: Beim Durchprobieren vergessen wir nützliche Informationen

!‖ Ausprobieren von kurzen Wegen gibt uns Informationen über längere

→ Für jeden Knoten den Abstand speichern und nach und nach verbessern.

12

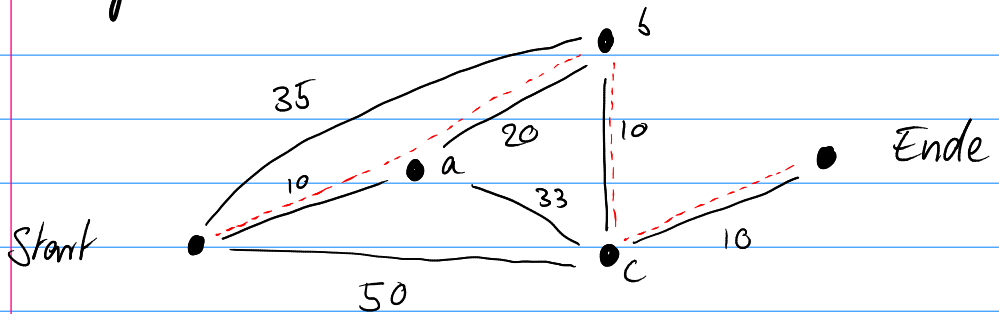
Arbeitsschritte

1. Setze $dist(start) = 0$ — Wir brauchen 0 Minuten zum Startort
2. Setze $dist(v) = \infty$ für alle $v \neq start$
3. Wähle einen Knoten u (mit $dist(u) < \infty$)
4. Für alle Kanten (u, v):
   Setze
   $$dist(v) = \min\{dist(v), dist(u) + Weg(u, v)\}$$
   "Entweder ich gehe den alten Weg zu v, oder es ist kürzer über u zu gehen"

20

5. Gehe zu 3, bis die Lösung stabil ist

Beispiel



25

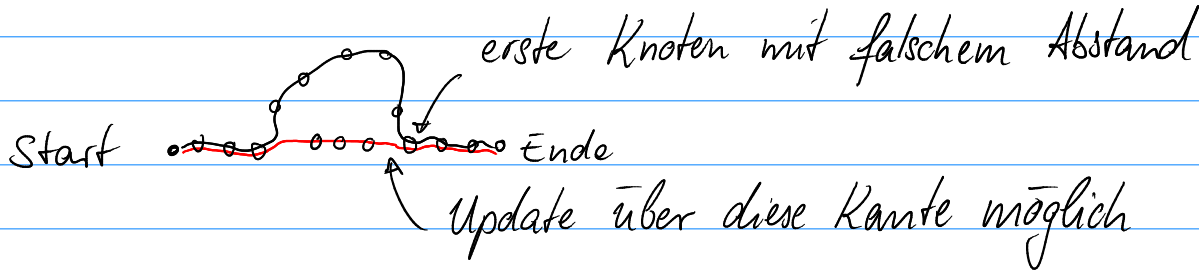
Fragen:

1. Finden wir immer den kürzesten Weg? Korrektheit
2. Finden wir ihn schnell? Effizienz

27

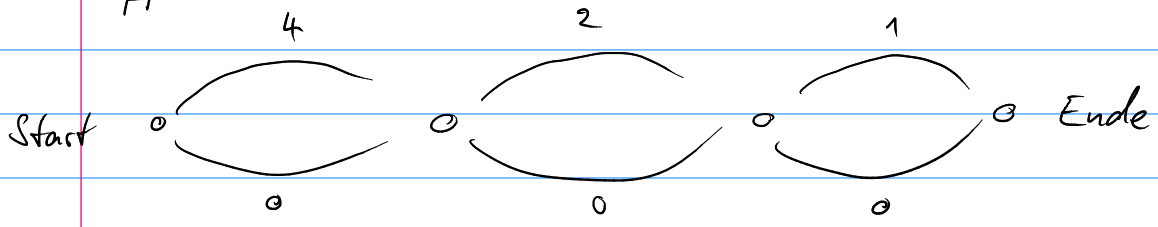
Korrektheit

- Abstände sind nie zu klein. → Es gibt zu jedem Abstand einen Weg
- Abstände sind nie zu groß. → Sonst ist die Lösung nicht stabil



35

Effizienz



Abstände ändern sich 8 mal (bei unglücklicher Kantenwahl)

→ Eine Kreuzung mehr doppelt so lange Laufzeit!

Orte	Änderungen
4	16
6	64
⋮	
41	1,000,000,000,000

10^9 Operationen/Sekunde

→ 1000 Sekunden

40

Unbrauchbar!

Warum braucht der Algorithmus so lange?

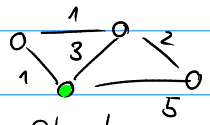
→ Unnütze Updates
Wenn die Distanz zu einem Knoten u noch nicht richtig ist, hat es keinen Sinn damit andere Distanzen zu berechnen.

45 → Nur Updates mit korrekten Distanzen machen

PAUSE

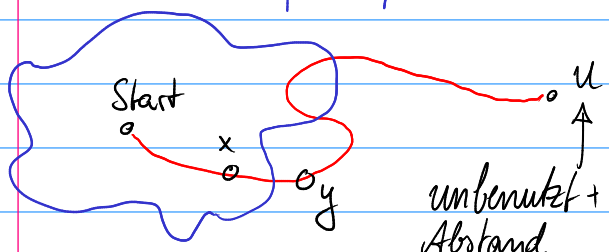
- Am Anfang ist die Distanz zu Start korrekt
- Nach Updates über alle benachbarten Kanten ist die Distanz zum nächsten Nachbarn korrekt (aber nicht für alle Nachbarn)

5 - Immer wahr:



Die Distanz zum Knoten mit dem kleinsten Abstand, über den noch keine Updates gemacht wurden, ist richtig

schon benutzt für Updates



Wissen schon: Wahr für Start
Angenommen nicht immer wahr.
→ Es gibt einen ersten Knoten u, für den es nicht stimmt

Betrachten wir den kürzesten Weg zwischen Start und u.
Weil u noch unbenutzt ist, muss es eine Kante x, y geben, wo der Weg den benutzten Bereich erstmals verlässt.
Der Abstand zu y ist schon richtig (wegen Update mit x) und muss kleiner sein als der Abstand zu u (es sind noch mehr Kanten auf dem Weg zwischen y und u).
Folglich ist y, nicht u, der Knoten mit dem kleinsten Abstand
– ein Widerspruch!

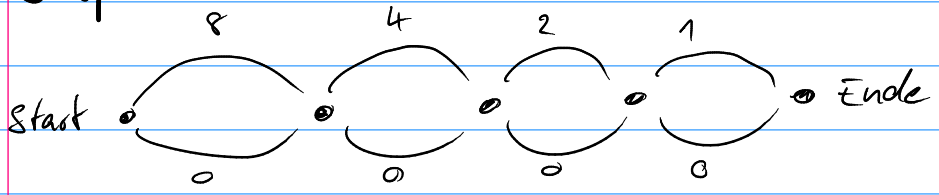
20

Dijkstras Algorithmus (1959)

Geschickt auswählen: Wähle immer die Kanten vom Knoten mit kleinstem Abstand (die noch nicht gewählt wurden)

Dijkstra (s):
dist[s] = 0, dist[v] = ∞ für v ≠ s
Markiere alle Knoten als unbenutzt
Solange aktive Knoten übrig:
  u = minimaler Distanz unter den unbenutzten } log n pro Knoten
  markiere u als benutzt
  für alle Kanten (u,v):
    wenn dist[u] + weg(u,v) < dist[v]: } insgesamt m Updates
      dist[v] = dist[u] + weg(u,v)

Beispiel



Weiteres Beispiel als Übungsaufgabe

Über jede Kante werden genau 2 Updates gemacht.

Laufzeit proportional zu $2m + n \cdot \log n$

Viel schneller als 2^n: Europa in etwa einer Minute

Weitere Verbesserung:

- Von Start und Ziel gleichzeitig suchen
- Bevorzugt in die richtige Himmelsrichtung suchen (A* Suche 1968)

→ Europa in Sekunden

Transitknoten (Hannah Bast)

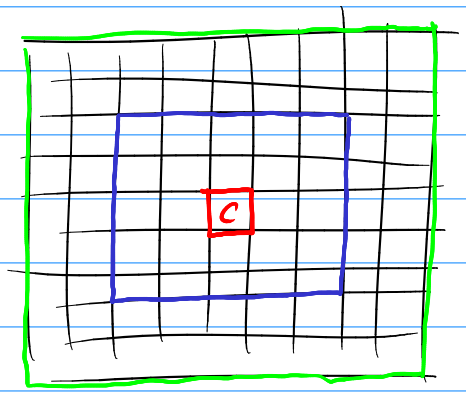
Idee: Routen nachschlagen statt zu rechnen

Nicht praktikabel: Europa $24 \cdot 10^6$ Knoten $\rightsquigarrow$ 10^{13} Wege
(10s/Dijkstra) $24 \cdot 10^6$ · Dijkstra $\rightsquigarrow$ 10 Jahre rechnen
100 kb/Weg $\rightsquigarrow$ 1000 Petabyte Platte
(≈ Googles gesamte Kapazität)

Idee: Lange Routen sind alle ähnlich, nutzen Autobahnen, Knotenpunkte
→ Nur Routen zwischen Knotenpunkten speichern

1) Knotenpunkte finden:

- Gitter über die Welt legen
- Routen von C nach außen
- Alle Knoten auf außen die benutzt werden sind Transitknoten für C



2) Vorberechnen

- Routen zwischen allen Transitknoten
- Routen von jedem Knoten in C zu "seinen" TK

3) Nachschlagen $A \mapsto B$

- A, B nah zusammen: Dijkstra
- Sonst: Minimiere

$$dist(A, T_1) + dist(T_1, T_2) + dist(T_2, B)$$

Europa:

1) Jeder Knoten hat etwa 10 Transitknoten
2) Insgesamt gibt es etwa 10 000 Transitknoten

1) $\sim 24 \cdot 10^7$ kurze Wege: 10 kb/Weg → 250 GB
2) $\sim 10^8$ lange Wege: 100 kb/Weg → 10 TB

→ passt auf eine Maschine!

A und B haben etwa 10 Transitknoten, es gibt also 100 Routen Routen zwischen A und B. Eine Route braucht bloß drei Werte in einer Tabelle zu addieren.

→ Routen in Millisekunden