Karlsruher Institut für Technologie

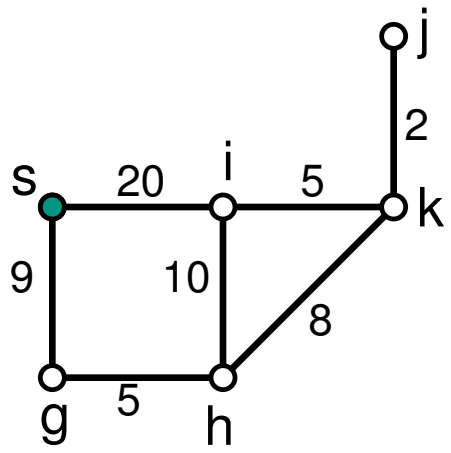
Tutorium Algorithmen 1

06 · Dijkstra · 3.6.2024
Peter Bohner Tutorium 3

Motivation

- Bisher: Alle Kanten haben Kosten 1, kürzester Weg geht über die kleinstmögliche Anzahl an Knoten
- jetzt: Kanten haben unterschiedliche Gewichte (Länge)
  - Gewichtsfunktion $len : E \mapsto \mathbb{Z}$
- Das ist sinnvoll für z.B. Straßennetze

Wieso funktioniert hier die Breitensuche nicht mehr?



Breitensuche findet (s,i,k) mit Länge 25,
(s,g,h,k) ist aber kürzer (Länge 22)

Problemstellung

- **Länge eines Pfades** $\langle v_0, \ldots, v_k \rangle$: $\sum_{i=1}^{k} \text{len}(v_{i-1}, v_i)$
- **Distanz** $\text{dist}(s, t)$: Länge des kürzesten st-Pfades

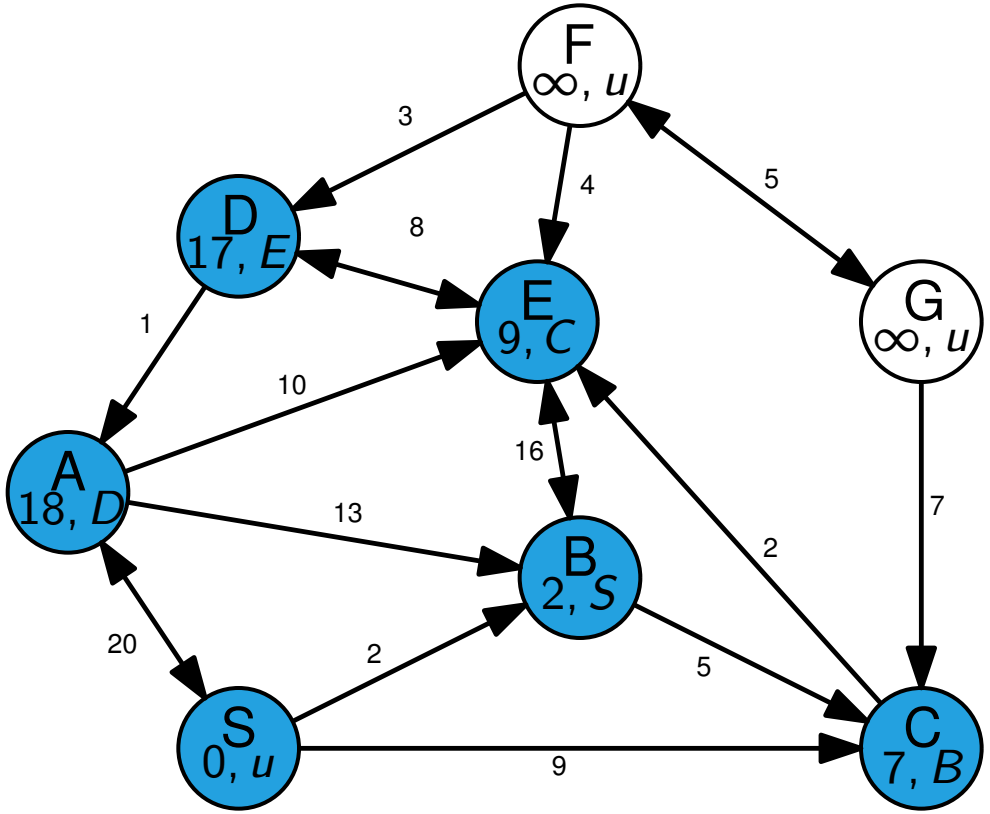
Problem: Single-Source Shortest Path (SSSP)
Gegeben einen gewichteten Graphen $G = (V, E)$ und einen Knoten $s \in V$, berechne $\text{dist}(s, t)$ für alle $t \in V$.

Dijkstra - Idee

Vorgehen

- Merke für jeden Knoten die Distanz zum Startknoten und den Knoten von dem aus er entdeckt wurde
  Am Anfang sind alle Distanzen ∞ und alle Herkunftsknoten null
- Setze die Distanz des Startknoten auf 0
- Wähle in jedem Schritt den noch nicht bearbeiteten Knoten mit der kleinsten Distanz zum Startknoten aus und:
  - Prüfe für alle noch nicht abgearbeiteten Nachbarn ob die Distanz des aktuellen Knoten + die Länge der Kante kleiner ist als die aktuelle Distanz
  - Falls ja update die Distanz des Nachbarn und setze den Herkunftsknoten auf den aktuellen Knoten
  - Das nennt man auch Kanten relaxieren

Beispiel



Dijkstra - Umsetzung

Karlsruher Institut für Technologie

Was müssen wir machen?

Attribut pro Knoten speichern	**können wir**
Den nächsten Knoten suchen den wir bearbeiten wollen	können wir auch, geht aber effizienter
Alle Nachbarn des aktuellen Knoten finden	**können wir**
Attribut pro Knoten anpassen	**können wir**

Einschub priority Queue

Wir wollen eine sogenannte Prioritätswarteschlange

- Speichert Elemente mit Priorität

Gewünschte Operationen:

- **push**(element, prio) fügt ein Element in die Queue ein
- **decPrio**(element, prio) ändert die Priorität eines Elements
- **popMin**() gibt das Element mit der niedrigsten Priorität zurück

push und **decPrio** in $\mathcal{O}(\log n)$, **popMin** in $\mathcal{O}(\log n)$ (andere Version mit **decPrio** in $\mathcal{O}(1)$)

Wie das funktioniert kommt später

Dijkstra - Umsetzung

```
DIJKSTRA(Graph G, Node s):
    d := [∞, ..., ∞]
    d[s] := 0
    Q := leere PriorityQueue
    for Node u ∈ G do
        Q.push(u, d[u])

    while ¬ Q.empty() do
        Node u := Q.popMin()
        for v ∈ N(u) do
            if d[u] + w(u, v) < d[v] then
                d[v] := d[u] + w(u, v)
                Q.decPrio(v, d[v])
```

Was müssen wir machen?

- Attribut pro Knoten speichern
- Den nächsten Knoten suchen den wir bearbeiten wollen
- Alle Nachbarn des aktuellen Knoten finden
- Attribut pro Knoten anpassen

Was fehlt hier noch?

- Herkunftsknoten Speichern
- Wie setzt man das um?
- Noch ein Array was für jeden Knoten den Index des Herkunftknotens speichert

Dijkstra - Laufzeit

```
DIJKSTRA(Graph G, Node s):
    d := [∞, ..., ∞]
    d[s] := 0
    Q := leere PriorityQueue
    for Node u ∈ G do
        Q.push(u, d[u])

    while ¬ Q.empty() do
        Node u := Q.popMin()
        for v ∈ N(u) do
            if d[u] + w(u, v) < d[v] then
                d[v] := d[u] + w(u, v)
                Q.decPrio(v, d[v])
```

- Wir fügen jeden Knoten in die PriorityQueue ein
  $\Rightarrow$n mal **popMin**
- Für jeden Knoten schauen wir alle Nachbarn an
  $\Rightarrow$jede Kante wird genau 2 mal angeschaut
  $\Rightarrow 2 \cdot m$ mal **decPrio**
- $\Theta((n \log n) + m)$ wenn **decPrio** in $\Theta(1)$
- $\Theta((n + m) \log n)$ wenn **decPrio** in $\Theta(\log n)$

Quiz

```
DIJKSTRA(Graph G, Node s):
    d := [∞, ..., ∞]
    d[s] := 0
    Q := leere PriorityQueue
    for Node u ∈ G do
        Q.push(u, d[u])

    while ¬ Q.empty() do
        Node u := Q.popMin()
        for v ∈ N(u) do
            if d[u] + w(u, v) < d[v] then
                d[v] := d[u] + w(u, v)
                Q.decPrio(v, d[v])
```

- Dijkstra findet kürzeste Wege vom Startknoten zu allen anderen Knoten. ✗ nur zu erreichbaren
- Man kann bei gleicher Laufzeit für jedes erreichbare Ziel danach in $\mathcal{O}(n)$ den kürzesten Pfad rekonstruieren. ✓
- Der kürzeste Weg zwischen zwei Knoten ist immer eindeutig. ✗
- Kürzeste Wege enthalten keine Kreise, wenn alle Kanten positives Gewicht haben. ✓
- Kürzeste Wege enthalten keine Kreise, wenn alle Kanten nichtnegatives Gewicht haben. ✗
- Kürzeste Wege ändern sich nicht, wenn alle auf Kantengewichte ein $\alpha > 0$ addiert wird. ✗
- Kürzeste Wege ändern sich nicht, wenn alle Kantengewichte mit einem $\lambda > 0$ durchmultipliziert werden. ✓

Aufgabe

Sei $G = (V, E)$ ein zusammenhängender Graph mit positiven Kantengewichten.
Entwerfe einen Algorithmus, der für einen Startknoten s und alle Zielknoten $t \in V$ den kürzesten Pfad, der die wenigsten Kanten benötigt, ausgibt.

- Können Dijkstra verwenden, müssen aber immer die Anzahl der Kanten im Pfad mitzählen
- Dazu: Ersetze Gewichte von Kanten durch Tupel: $len'(e) = (len(e), 1)$
- Addition zweier Tupel komponentenweise $\Longrightarrow$ Gewicht eines Pfades: (Gewicht, Anzahl Kanten)
- Tupel lexikographisch vergleichen, d.h. $(l_1, k_1) < (l_2, k_2) \Leftrightarrow l_1 < l_2 \vee (l_1 = l_2 \wedge k_1 < k_2)$

$\Rightarrow$ falls 2 Pfade mit gleicher Länge zu einem Knoten führen wird der mit den wenigsten Knoten gewählt

Aufgabe

Gegeben:

- n Städte
- Straßen mit 3 Straßentypen: Autobahn, Landstraße, Feldweg
- Je 2 Städte können durch verschiedene Straßen (auch mehr als eine) verbunden sein
- Über eine bestimmte Straße i zu fahren kostet t_i Zeit
- Geldbeutel mit Kapazität für 10 Münzen
- Benutzen von Straßen kostet Münzen:
  - Autobahn: 4 Münzen
  - Landstraße: 3 Münzen
  - Feldweg: 1 Münze
- In jeder Stadt bekommt man 2 Münzen
- Ziel: mit Anfangsguthaben x möglichst schnell von Stadt A zu Stadt B kommen

Erstelle zu jeder solchen Situation einen Graphen, in dem man die Aufgabe mit Dijkstra lösen kann.

Tipps:

- Ob du eine Straße befahren kannst, hängt davon ab, wie viel Geld du in der jeweiligen Stadt hast
- Erstelle pro Stadt mehrere Knoten, die alle möglichen Guthaben modellieren

Lösung

- Modellierung als gerichteter Graph $G = (V, E)$
- Städte als Menge $S = \{s_1, \ldots, s_n\}$
- Für jede Kombination aus Stadt und Guthaben einen Knoten, also $V' = S \times \mathbb{Z}_{11}$
- Pro Straße und Guthaben (falls genug Münzen da) eine Kante mit der entsprechenden Stadt- und Guthabenänderung und der Dauer als Kantengewicht
- Da egal ist mit welchem Guthaben wir enden, noch einen Endknoten pro Stadt $V_{end} = S$
- Kanten von jedem Knoten der Stadt zum Endknoten mit Kosten 0

Finde schnellsten Weg mit Anfangsguthaben x von A nach B indem mit Dijkstra der kürzeste Weg von (s_a, x) zu s_b gefunden wird

Aufgabe

Gegeben ein Graph $G = (V, E)$, wobei Kanten entweder Gewicht 0 oder 1 haben.

- Löse das Kürzeste Wege Problem (SSSP) in $\mathcal{O}((n + m) \log n)$
- Löse das Kürzeste Wege Problem in $\mathcal{O}(n + m)$

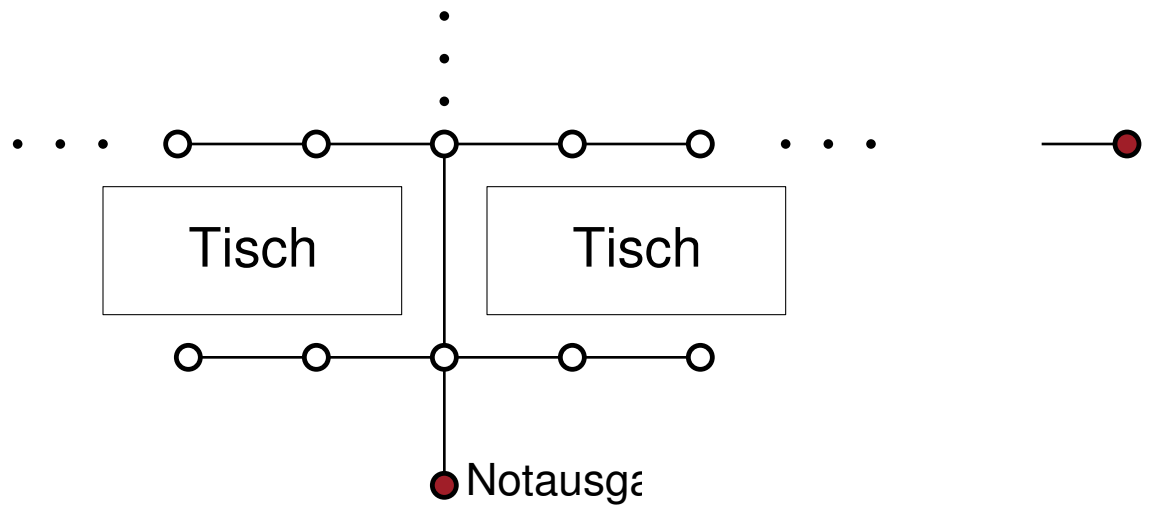
- $\mathcal{O}((n + m) \log n) \Rightarrow$ Dijkstra löst das Problem offensichtlich

$\mathcal{O}(n + m)$ Lösung

- Wir nutzen die 0,1-BFS
- Ähnlich wie bei Dijkstra nutzen wir eine Dequeue, diesmal aber für die Kanten
- Wenn wir einen Knoten über eine 0-Kante entdecken, fügen wir diese vorne ein und Knoten an einer 1-Kante hinten. Bei jeder Iteration poppen wir einen Knoten von vorne aus der Queue heraus und färben diesen.
- Die Laufzeit folgt, da wir jede Kante maximal zweimal betrachten und dadurch jeder Knoten maximal zweimal in der Queue auftauchen kann

Bonusaufgabe: Notausgänge

In der Mensa ist ein Feuer ausgebrochen! Doch der Hauptraum der Mensa hat mehrere Notausgänge, Gott sei Dank. Wir modellieren nun die Mensa als Graphen (siehe Bild).
Berechne in jedem Punkt der Mensa die Distanz zum nächsten Notausgang.



- Wir nutzen eine Breitensuche, wobei wir zu Beginn jeden Notausgang als Startknoten in die Queue einfügen

Zusammenfassung

Was haben wir gemacht?

- Kürzeste Wege auf gewichteten Graphen (SSSP)

Worauf könnt ihr euch nächste Woche freuen?

- Kürzeste Wege mit negativen Kanten
- All-Pair Shortest Path (APSP)
- Priority Queues

Umfrage

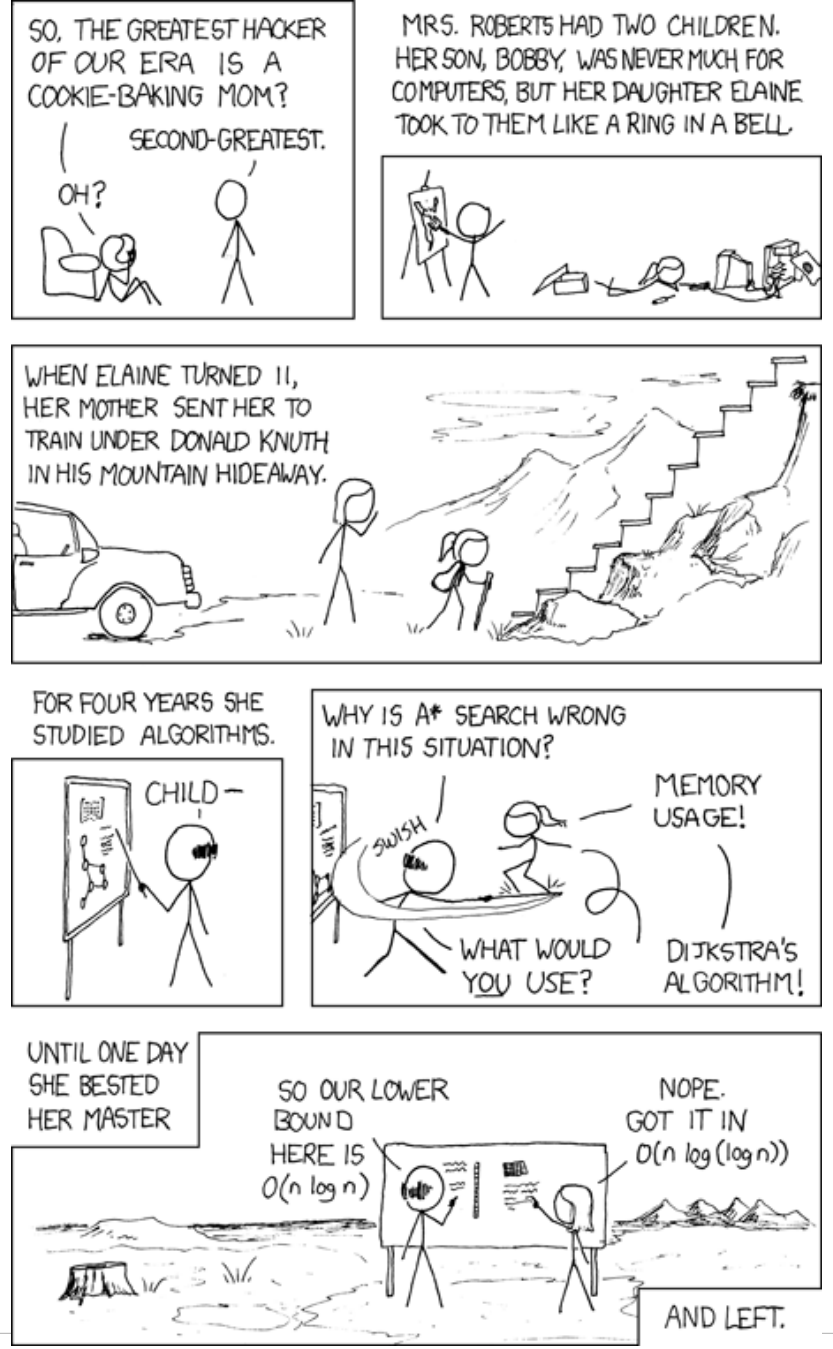
https://forms.gle/h1engX2ny1CUbmJW9

Fragen?

Karlsruher Institut für Technologie

Fragen!

Ende



https://xkcd.com/342/