Algorithmen I
Tutorium 32

Eine Lehrveranstaltung im SS 2017 (mit Folien von Christopher Hommel)

Daniel Jungkind (ufesa@kit.edu) | 07. Juli 2017

INSTITUT FÜR THEORETISCHE INFORMATIK

Noch was zu DFS/BFS

- DFS/BFS finden **Pfade** von Startknoten s zu allen anderen erreichbaren Knoten
  $\Rightarrow$ *parent*-Array zum Rekonstruieren der Pfade
  ($parent[v]$: Vorgänger von v im Pfad zu v)
- DFS/BFS messen „**Distanz**“ der Knoten
  $\Rightarrow$ d-Array mit $d[v] =$ Anzahl Kanten auf dem Weg zu v

$\Longrightarrow$ **Rückgabewerte** von BFS/DFS im Pseudocode benutzbar:
$(parent, d) := \text{BFS}(G, s)$ *// DFS similar*
// Now use $parent[\cdot]$ *and* $d[\cdot]$

KÜRZESTE PFADE

Es kommt halt *doch* auf die Länge an...

Kürzeste Pfade – Dijkstra

Der unaussprechliche Algorithmus

- Gesucht: **Kürzeste gewichtete** Pfade
  von Startknoten $s \in V$ zu **allen** anderen Knoten
- *Breitensuche*: Findet kürzeste Pfade bei **ungewichteten** Kanten

$\Rightarrow$ Passe BFS für gewichtete Kanten an,
verwende zwei **arrays**:

  - $d[v]$: Länge des **bisher bekannten** kürzesten Pfades zu v
  - $parent[v]$: **Direkter** Vorgänger von v im **bisher bekannten** kürzesten Pfad zu v

- Rüste Queue Q auf zu einer **PriorityQueue** PQ (z.B. binärer Heap), Knoten v wird mit $d[v]$ gewichtet
- Wichtige Einschränkung: **Keine negativen Kantengewichte!**

Kürzeste Pfade – Dijkstra

```
function Dijkstra(G = (V, E), s ∈ V)
    d := (∞, ..., ∞) : array[1...n] of ℝ
    parent := (⊥, .., ⊥) : array[1...n] of V
    PQ = {s} : PriorityQueue
    parent[s] := s,    d[s] := 0
    while PQ ≠ ∅ do
        u := PQ.deleteMin()  // u wird jetzt „gescannt“
        foreach e = (u, v) ∈ E do      // „Relaxiere“ e
            if d[u] + c(e) < d[v] then
                d[v] := d[u] + c(e)
                parent[v] := u
                if v ∈ PQ then
                    PQ.decreaseKey(v)
                else
                    PQ.insert(v)
    return (d, parent)
```

Kürzeste Pfade – Dijkstra

Korrektheit

- **Invariante**: Wenn ein Knoten aus *PQ* entnommen wird, ist zu diesem der **endgültige** kürzeste Pfad bekannt
- Beweis der Invariante durch **vollständige Induktion** über die Schleifendurchläufe möglich

Kürzeste Pfade – Dijkstra

Laufzeit von Dijkstra

Im Worst-Case *m*-mal *decreaseKey*

$+$ Genau *n*-mal *deleteMin* und *insert*

$=$ Mit binärem Heap: $O\left((m+n)\log n\right)$

$=$ Mit Fibonacci-Heap: $O(m+n\log n)$ (amortisiert und mit höheren konstanten Faktoren)

Kürzeste Pfade

Aufgabe 1: Noch kürzere kürzeste Pfade

Gegeben sei ein (gerichteter oder ungerichteter) zusammenhängender Graph $G = (V, E)$ mit nichtnegativen Kantengewichten $\omega : E \longrightarrow \mathbb{R}^+$. Beschreibt einen effizienten Algorithmus, der für einen Startknoten s und alle Zielknoten $t \in V$ den Pfad mit den **wenigsten** Kanten unter allen kürzesten Pfaden von s nach t berechnet.

Kürzeste Pfade

Lösung zu Aufgabe 1

Modifiziere Dijkstra: Definiere Kantengewichte um als **Tupel** $c'(e) := \big(c(e), 1\big)$ (mit komponentenweiser Addition) und folgender Ordnung:

$(a, b) < (c, d) \iff a < c \lor (a = c \land b < d)$

Kürzeste Pfade – Bellman-Ford

Rohe Gewalt: Bellman-Ford

- **Problem**: Dijkstra „erstickt“ an negativen Kantengewichten
- **Überlegung**: Längster (zyklenfreier) Pfad hat **maximal** $n - 1$ Kanten

$\Rightarrow$ Relaxiere jede Kante $(n - 1)$-mal $\Rightarrow$ **jeder** minimale zyklenfreie Pfad wurde bestimmt

- **Laufzeit**: $O(n \cdot m)$

Kürzeste Pfade – Bellman-Ford

```
function BellmanFord(G = (V, E), s ∈ V)
    d := (∞, ..., ∞) : array[1...n] of ℝ
    parent := (⊥, ..., ⊥) : array[1...n] of V
    parent[s] := s;   d[s] := 0
    do n − 1 times
        foreach e = (u, v) ∈ E do
            if d[u] + c(e) < d[v] then
                d[v] := d[u] + c(e)
                parent[v] := u
    foreach e = (u, v) ∈ E do
        if d[u] + c(e) < d[v] then
            // kleinerer zyklenfreier Pfad ist nicht möglich ⇒ Negativer Zyklus
            d[v] := −∞
    return (d, parent)
```

Graphen

Aufgabe 2: Der Klassiker

Ein Fährmann soll einen Wolf, eine Ziege und einen Kohlkopf von der linken auf die rechte Seite eines Flusses befördern. Sein kleines Boot hat aber nur Platz für ihn und ein weiteres Objekt.

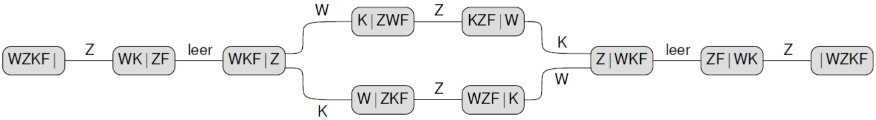
Außerdem frisst der Wolf die Ziege und die Ziege den Kohlkopf, wenn der Fährmann nicht dabei ist. Zum Glück mag der Wolf kein Gemüse. Wie kann der Fährmann den Wolf, die Ziege und den Kohlkopf unbeschadet übersetzen?

Löst das Problem mithilfe eines Graphen zeichnerisch.

Graphen

Lösung zu Aufgabe 2

Ein Zustandsgraph:



Weg von [WZKF|] nach [|WZKF] ist Lösung.

Graphen

Aufgabe 3: Domino Day

Ihr habt folgende Dominosteine gegeben:

1	4	1	2	1	5	1	3	4	6
4	2	4	5	6	2	2	5	5	3

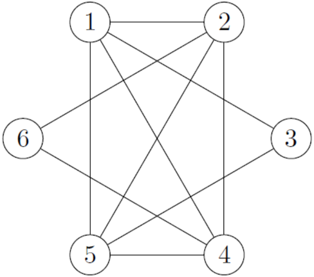
Ist es möglich, alle Steine als einen geschlossenen Ring anzuordnen, sodass nur gleiche Zahlen aneinanderliegen? Löst das Problem mithilfe eines Graphen zeichnerisch.

Graphen

Lösung zu Aufgabe 3

Pro Zahl ein Knoten, pro Stein eine Kante zwischen zwei Knoten.

Zyklus $1 \rightarrow 4 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1$ ist Lösung.



Allgemein lösbar, wenn Graph zusammenhängend ist und eulerschen Kreis enthält ($\Leftrightarrow$ nur gerade Knotengrade enthält).

(siehe Exkurs nächste Folie)

Exkurs Eulerkreis/Hamiltonkreis

Eulerkreis:
Kreis, der jede **Kante** genau einmal beschreitet. (**E**uler $\Rightarrow$ **E**dges $\Rightarrow$ Kanten)

Hamiltonkreis:
Kreis, der jeden **Knoten** genau einmal beschreitet.

$\exists$ Eulerkreis in $G \Leftrightarrow G$ hat nur gerade Knotengrade.
$\exists$ Hamiltonkreis in $G \Leftrightarrow$ Ausprobieren! :P (Gibt kein einfaches Kriterium)

Graphen

Aufgabe 4: Tiefensuche revisited

Implementiert Tiefensuche nicht-rekursiv als Pseudocode. Das asymptotische Laufzeitverhalten von Tiefensuche darf hierbei nicht überschritten werden.

Kürzeste Pfade

Lösung zu Aufgabe 4

Recursion-Faking mittels Stack:

procedure DFS$\big(G = (V, E),\ s \in V\big)$

- $S := \langle s \rangle$: Stack
- *visited* $:=$ (**false**, ..., **false**) : **array of** Boolean
- **while** $S \neq \emptyset$ **do**
  - $u := S$.pop()
  - **if not** $visited[u]$ **then**
    - $visited[u] :=$ **true**
    - **for** $(u, v) \in E$ **do**
      - **if not** $visited[v]$ **then**
        - S.push(v)