Algorithmen I
Tutorium 33

Woche 10 | 29. Juni 2018

Daniel Jungkind (daniel.jungkind@student.kit.edu)

INSTITUT FÜR THEORETISCHE INFORMATIK

Inhalt

Kürzeste Pfade

Dijkstra

Bellman-Ford

Zur Probeklausur

Durchschnitt: 58 % der Gesamtpunkte

- Recht gnädige Korrektur, weil sie ja zählt
- **Trotzdem**: verschenkte Punkte zuhauf

Zu Blatt #9 (bzw. #8, je nachdem...)

Durchschnitt: 69 % der Gesamtpunkte

- Beim Einfügen in einen (a, b)-Baum verwendet man *split*. **?**

- B: „*ohne* Datenliste“
- B: Bei 0-Indizierung $\lfloor \frac{n}{2} \rfloor$, nicht $\lceil \frac{n}{2} \rceil$ $\Rightarrow$ sonst **unbalanciert!**

Zu Blatt #9 (bzw. #8, je nachdem...)

Durchschnitt: 69 % der Gesamtpunkte

- Beim Einfügen in einen (a, b)-Baum verwendet man *split*. **Wahr.**

  Wieso macht ihr dann *balance*!?
- B: „*ohne* Datenliste“
- B: Bei 0-Indizierung $\left\lfloor \frac{n}{2} \right\rfloor$, nicht $\left\lceil \frac{n}{2} \right\rceil$ $\Rightarrow$ sonst **unbalanciert!**

Zu Blatt #9 (bzw. #8, je nachdem...)

Durchschnitt: 69 % der Gesamtpunkte

- Beim Einfügen in einen (a, b)-Baum verwendet man *split*. **Wahr.**

  Wieso macht ihr dann *balance*!?
- B: „*ohne* Datenliste“
- B: Bei 0-Indizierung $\lfloor \frac{n}{2} \rfloor$, nicht $\lceil \frac{n}{2} \rceil \Rightarrow$ sonst **unbalanciert**!

Zu Blatt #9 (bzw. #8, je nachdem...)

Durchschnitt: 69 % der Gesamtpunkte

- Beim Einfügen in einen (a, b)-Baum verwendet man *split*. **Wahr.**

  Wieso macht ihr dann *balance*!?
- B: „*ohne* Datenliste“
- B: Bei 0-Indizierung $\left\lfloor \frac{n}{2} \right\rfloor$, nicht $\left\lceil \frac{n}{2} \right\rceil$ $\Rightarrow$ sonst **unbalanciert**!

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

⇒ Passe BFS für gewichtete Kanten an:
  - $d[v]$: 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

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:
  - $d[v]$: 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

Karlsruher Institut für Technologie

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:

  - $d[v]$: 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

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:

  - $d[v]$: 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

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:
  - $d[v]$: 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

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

Korrektheitsbeweis – Invariante

IA.: Endgültiger kürzester Pfad zu s: Trivial ✓

IV.: Zu allen Knoten $v_1, \ldots, v_{i-1}$, die aus der PQ entnommen wurden, ist der **endgültige** kürzeste Pfad bekannt

Kürzeste Pfade – Dijkstra

Korrektheitsbeweis – Invariante

IA.: Endgültiger kürzester Pfad zu s: Trivial ✓

IV.: Zu allen Knoten $v_1, \ldots, v_{i-1}$, die aus der PQ entnommen wurden, ist der **endgültige** kürzeste Pfad bekannt

Kürzeste Pfade – Dijkstra

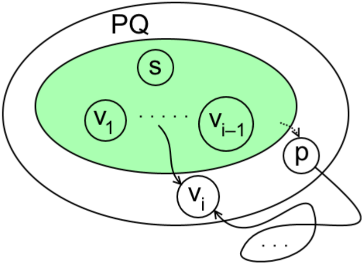
Korrektheitsbeweis – Invariante

IS.: Knoten v_i wird entnommen. Der bekannte kürzeste Pfad zu v_i führt „irgendwie“ über $v_1 \ldots v_{i-1}$.

Kürzeste Pfade – Dijkstra

Korrektheitsbeweis – Invariante

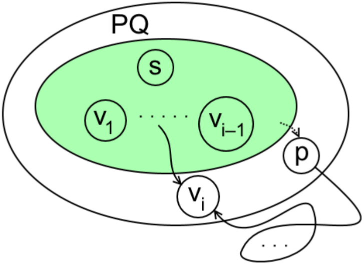
IS.: Knoten v_i wird entnommen. Der bekannte kürzeste Pfad zu v_i führt „irgendwie“ über $v_1 \ldots v_{i-1}$.
Ang., es gibt einen **echt** kürzeren Pfad zu v_i. Dieser **muss** dann über einen Knoten p aus der PQ zu v_i führen.



Kürzeste Pfade – Dijkstra

Korrektheitsbeweis – Invariante

IS.: Knoten v_i wird entnommen. Der bekannte kürzeste Pfad zu v_i führt „irgendwie" über $v_1 \ldots v_{i-1}$.
Ang., es gibt einen **echt** kürzeren Pfad zu v_i. Dieser **muss** dann über einen Knoten p aus der PQ zu v_i führen.

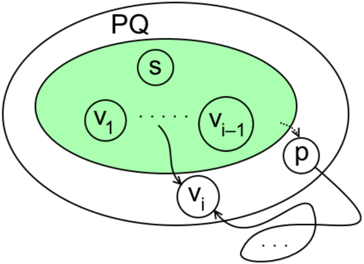


Dafür gibt es zwei Möglichkeiten:
Fall 1: Zu p gibt es einen kürzeren Pfad als zu v_i
$\Rightarrow$ p wurde **vor** v_i aus der PQ entnommen ↯

Kürzeste Pfade – Dijkstra

Korrektheitsbeweis – Invariante

IS.: Knoten v_i wird entnommen. Der bekannte kürzeste Pfad zu v_i führt „irgendwie“ über $v_1 \ldots v_{i-1}$.
Ang., es gibt einen **echt** kürzeren Pfad zu v_i. Dieser **muss** dann über einen Knoten p aus der PQ zu v_i führen.



Dafür gibt es zwei Möglichkeiten:

Fall 2: Der Pfad über p zu v_i ist kürzer als der kürzeste Pfad zu p
$\Rightarrow c\left((p, \ldots, v_i)\right) < 0$ ↯ (Keine negativen Kantengewichte erlaubt!) $\square$

Kürzeste Pfade – Dijkstra

Karlsruher Institut für Technologie

Korrektheitsbeweis – Terminierung

- In **jedem** Schleifendurchlauf wird ein Knoten v aus PQ entnommen. **Endgültiger** kürzester Pfad zu v bekannt $\Rightarrow$ v wird danach **nicht** mehr erneut in die PQ eingefügt

$\Rightarrow$ Nach maximal n Schleifendurchläufen ist PQ leer und der Algorithmus terminiert. Hurra. $\square$

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 – 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 – 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:

$(a, b) < (c, d) \;:\Longleftrightarrow\; a < c \,\vee\, (a = c \wedge b < d)$

Kürzeste Pfade

Aufgabe 2: Unerschlossenes Neuland

Ihr seid Administrator eines Computernetzwerks bestehend aus n Rechnern und Netzwerkverbindungen zwischen diesen. Leider sind die Netzwerkverbindungen nicht absolut zuverlässig: Jede Verbindung zwischen zwei Rechnern a und b hat eine Wahrscheinlichkeit $0 < p_{(a,b)} < 1$, dass ein auf dieser Verbindung gesendetes Paket ankommt. Die Verbindungen sind gerichtet, das heißt es kann sein, dass $p_{(a,b)} \neq p_{(b,a)}$ gilt.

Eure Aufgabe ist es nun, zwischen zwei gegebenen Rechnern einen möglichst zuverlässigen Pfad zu finden. Gebt dafür einen möglichst effizienten Algorithmus an.

Kürzeste Pfade

Karlsruher Institut für Technologie

Lösung zu Aufgabe 2

Führe **Dijkstra** auf dem Netzwerk-Graphen durch, allerdings mit...

- Zuverlässigkeiten $p_{(a,b)}$ als **Kantengewichten**,
- Suche nach dem „Longest Path“
  $\Rightarrow$ Verwende **Max**-Heap statt Min-Heap,
  $\Rightarrow$ relaxiere Kanten unter Bevorzugung „**längerer**“ (= zuverlässigerer) Pfade,
- **Multiplikation** statt Addition der Kantengewichte (Wahrscheinlichkeiten auf einem Pfad akkumulieren sich multiplikativ).

Dies funktioniert, da $0 < p < 1$ $\Rightarrow$ Pfade mit mehr Kanten haben also kleinere Zuverlässigkeitswerte.

Laufzeit: Wie Dijkstra in $\mathcal{O}((m+n)\log n)$.

Kürzeste Pfade

Lösung zu Aufgabe 2

Führe **Dijkstra** auf dem Netzwerk-Graphen durch, allerdings mit...

- Zuverlässigkeiten $p_{(a,b)}$ als **Kantengewichten**,
- Suche nach dem „Longest Path“
  $\Rightarrow$ Verwende **Max**-Heap statt Min-Heap,
  $\Rightarrow$ relaxiere Kanten unter Bevorzugung „**längerer**“ (= zuverlässigerer) Pfade,
- **Multiplikation** statt Addition der Kantengewichte (Wahrscheinlichkeiten auf einem Pfad akkumulieren sich multiplikativ).

Dies funktioniert, da $0 < p < 1 \Rightarrow$ Pfade mit mehr Kanten haben also kleinere Zuverlässigkeitswerte.

Laufzeit: Wie Dijkstra in $\mathcal{O}((m+n)\log n)$.

Kürzeste Pfade

Karlsruher Institut für Technologie

Lösung zu Aufgabe 2

Führe **Dijkstra** auf dem Netzwerk-Graphen durch, allerdings mit...

- Zuverlässigkeiten $p_{(a,b)}$ als **Kantengewichten**,
- Suche nach dem „Longest Path"
  $\Rightarrow$ Verwende **Max**-Heap statt Min-Heap,
  $\Rightarrow$ relaxiere Kanten unter Bevorzugung „**längerer**" ($=$ zuverlässigerer) Pfade,
- **Multiplikation** statt Addition der Kantengewichte (Wahrscheinlichkeiten auf einem Pfad akkumulieren sich multiplikativ).

Dies funktioniert, da $0 < p < 1 \Rightarrow$ Pfade mit mehr Kanten haben also kleinere Zuverlässigkeitswerte.

Laufzeit: Wie Dijkstra in $\mathcal{O}\big((m+n)\log n\big)$.

Kürzeste Pfade – Bellman-Ford

Rohe Gewalt: Bellman-Ford

- **Problem**: Dijkstra „erstickt“ an negativen Kantengewichten
- **Überlegung**: Längster (zyklenfreier) Pfad hat **maximal** $n - 1$ Kanten
- **Einleuchtend**: Jede Kante einmal zu relaxieren berechnet kürzeste Pfade der Länge 1 von jedem Knoten aus

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

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

Kürzeste Pfade – Bellman-Ford

Rohe Gewalt: Bellman-Ford

- **Problem**: Dijkstra „erstickt“ an negativen Kantengewichten
- **Überlegung**: Längster (zyklenfreier) Pfad hat **maximal** $n - 1$ Kanten
- **Einleuchtend**: Jede Kante einmal zu relaxieren berechnet kürzeste Pfade der Länge 1 von jedem Knoten aus

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

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

Kürzeste Pfade – Bellman-Ford

Rohe Gewalt: Bellman-Ford

- **Problem**: Dijkstra „erstickt“ an negativen Kantengewichten
- **Überlegung**: Längster (zyklenfreier) Pfad hat **maximal** $n - 1$ Kanten
- **Einleuchtend**: Jede Kante einmal zu relaxieren berechnet kürzeste Pfade der Länge 1 von jedem Knoten aus

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

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

Kürzeste Pfade – Bellman-Ford

Rohe Gewalt: Bellman-Ford

- **Problem**: Dijkstra „erstickt“ an negativen Kantengewichten
- **Überlegung**: Längster (zyklenfreier) Pfad hat **maximal** $n - 1$ Kanten
- **Einleuchtend**: Jede Kante einmal zu relaxieren berechnet kürzeste Pfade der Länge 1 von jedem Knoten aus

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

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

Kürzeste Pfade – Bellman-Ford

function Bellman-Ford($G = (V, E),\ s \in V$)

- $d := (\infty, \ldots, \infty)$: **array**[1...n] **of** $\mathbb{R}$
- $parent := (\perp, \ldots, \perp)$: **array**[1...n] **of** V
- $parent[s] := s; \quad d[s] := 0$
- **do** $n - 1$ **times**
  - **foreach** $e = (u, v) \in E$ **do**
    - **if** $d[u] + c(e) < d[v]$ **then**
      - $d[v] := d[u] + c(e)$
      - $parent[v] := u$
- **foreach** $e = (u, v) \in E$ **do**
  - **if** $d[u] + c(e) < d[v]$ **then**
    - *// kleinerer zyklenfreier Pfad ist nicht möglich $\Rightarrow$ Negativer Zyklus!*
    - $d[v] := -\infty$
- **return** $(d,\ parent)$

Kürzeste Pfade

Aufgabe 3: Geld rotiert die Welt

Der ebenso geniale wie wirtschaftliche Superbösewicht Doktor Meta wittert das große Geld! Um seine Weltherrschaftspläne zu finanzieren, hat er einen teuflischen Plan ersonnen, mit dem er den internationalen Devisenhandel über den Tisch ziehen will: Zwischen je zwei Währungen i und j gibt es einen Wechselkurs, dargestellt als Umrechnungs-Faktor $T[i,j] > 0$. Doktor Meta will nun so lange Währungen tauschen, bis er am Ende mehr Geld in seiner Ausgangswährung „M$“ hat als zuvor; das heißt, er sucht eine „Umtausch-Kette“, deren Gesamt-Faktor > 1 ist.

Sein größter Widersacher Turing-Man (halb Mensch, halb Turingmaschine) ist ihm jedoch auf den Fersen und will den Finanzmarkt gezielt aufmischen, um die Pläne des Superbösewichts zu vereiteln. Dafür braucht er allerdings die genaue Tauschreihenfolge von Währungen, die Doktor Meta ausnutzen will. Helft Turing-Man und überlegt euch ein Verfahren, mit welchem ihr Metas genauen Plan enttarnen könnt.

Kürzeste Pfade

Lösung zu Aufgabe 3

- Definiere $V := \{\text{Währungen } i\}$, $E := \big\{(i,j) \bigm| T[i,j] \neq \perp\big\}$ und Kantengewichte $c(i,j) := -\log T[i,j]$.
- Starte **Bellman-Ford** von Knoten „M$" aus und suche nach einem **negativen Zyklus**. Dies ist der gewünschte Tauschzyklus.

Denn: Für eine Währungstauschfolge $(w_1, \dots, w_\ell)$ mit $w_\ell = w_1$ gilt:

$$\prod_{k=1}^{\ell-1} T[w_k, w_{k+1}] \overset{!}{>} 1 \iff \log \prod_k T[w_k, w_{k+1}] > 0$$

$$\iff -\log \prod_k T[w_k, w_{k+1}] < 0 \iff -\sum_k \log T[w_k, w_{k+1}] < 0$$

$$\iff \sum_k -\log T[w_k, w_{k+1}] < 0 \iff \sum_k c(w_k, w_{k+1}) < 0. \quad \square$$

Kürzeste Pfade

Lösung zu Aufgabe 3

- Definiere $V := \{\text{Währungen } i\}$, $E := \{(i,j) \mid T[i,j] \neq \perp\}$ und Kantengewichte $c(i,j) := -\log T[i,j]$.
- Starte **Bellman-Ford** von Knoten „M$“ aus und suche nach einem **negativen Zyklus**. Dies ist der gewünschte Tauschzyklus.

Denn: Für eine Währungstauschfolge $(w_1, \dots, w_\ell)$ mit $w_\ell = w_1$ gilt:

$$\prod_{k=1}^{\ell-1} T[w_k, w_{k+1}] \overset{!}{>} 1 \iff \log \prod_k T[w_k, w_{k+1}] > 0$$

$$\iff -\log \prod_k T[w_k, w_{k+1}] < 0 \iff -\sum_k \log T[w_k, w_{k+1}] < 0$$

$$\iff \sum_k -\log T[w_k, w_{k+1}] < 0 \iff \sum_k c(w_k, w_{k+1}) < 0. \quad \square$$

Kürzeste Pfade

Lösung zu Aufgabe 3

- Definiere $V := \{\text{Währungen } i\}$, $E := \{(i,j) \mid T[i,j] \neq \perp\}$ und Kantengewichte $c(i,j) := -\log T[i,j]$.
- Starte **Bellman-Ford** von Knoten „M$“ aus und suche nach einem **negativen Zyklus**. Dies ist der gewünschte Tauschzyklus.

Denn: Für eine Währungstauschfolge $(w_1, \ldots, w_\ell)$ mit $w_\ell = w_1$ gilt:

$$\prod_{k=1}^{\ell-1} T[w_k, w_{k+1}] \overset{!}{>} 1 \iff \log \prod_k T[w_k, w_{k+1}] > 0$$

$$\iff -\log \prod_k T[w_k, w_{k+1}] < 0 \iff -\sum_k \log T[w_k, w_{k+1}] < 0$$

$$\iff \sum_k -\log T[w_k, w_{k+1}] < 0 \iff \sum_k c(w_k, w_{k+1}) < 0. \quad \square$$

Aufgabe 4: DISHONEST WITCH HUNT!

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

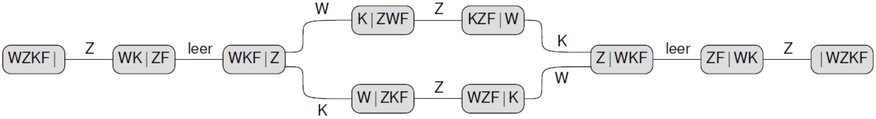
Außerdem frisst der Wolf die Ziege und die Ziege den Hohlkopf, wenn der Fährmann nicht dabei ist. Zum Glück mag der Wolf keine Nazis. Wie kann der Fährmann den Wolf, die Ziege und den Hohlkopf unbeschadet übersetzen?

Löst das Problem mithilfe eines Graphen zeichnerisch.

Graphen

Lösung zu Aufgabe 4

Ein Zustandsgraph: (**K** steht für **K**ushners Schwiegervater Trump)



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

Graphen

Karlsruher Institut für Technologie

Aufgabe 5: 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

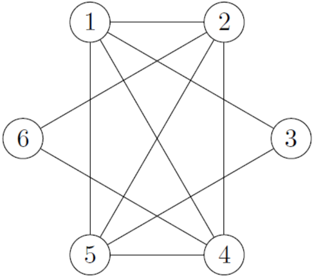
Ist es möglich, alle Steine als einen geschlossenen Ring anzuordnen, sodass nur gleiche Zahlen aneinanderliegen? Löst das Problem mithilfe eines Graphen zeichnerisch.

Graphen

Lösung zu Aufgabe 5

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

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



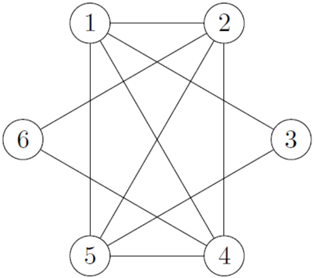
Allgemein lösbar, wenn Graph zusammenhängend ist und eulerschen Kreis enthält ($\Leftrightarrow$ nur gerade Knotengrade enthält).
(siehe Exkurs nächste Folie)

Graphen

Lösung zu Aufgabe 5

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

Zyklus $1 \to 4 \to 6 \to 2 \to 5 \to 3 \to 1 \to 2 \to 4 \to 5 \to 1$ 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. (Hamilton $\Rightarrow$ Hnoten :P)

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

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. (**H**amilton $\Rightarrow$ **H**noten :P)

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

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. (**H**amilton $\Rightarrow$ **H**noten :P)

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

Danke für eure Aufmerksamkeit! ☺

THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.

http://xkcd.com/612