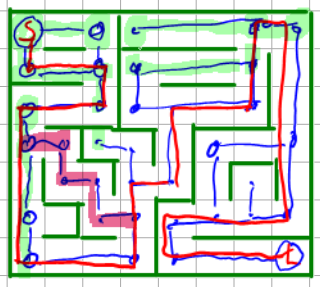
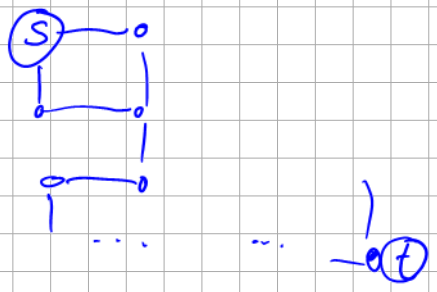
Kürzeste Wege

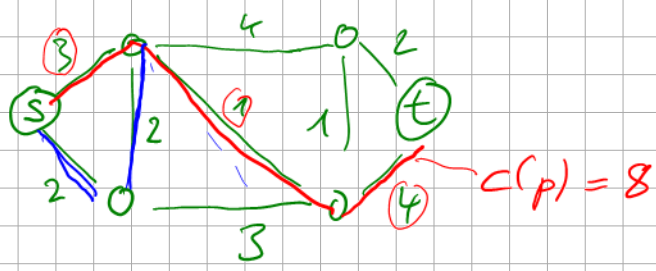
Labyrinth:





Gewichteter Graph $G = (V, E, c)$

$c : E \to \mathbb{R}$



gegeben: $s, t \in V$

gesucht: Pfad $p: s \rightsquigarrow t$ mit minimalem Gewicht $c(p)$

$p := \langle v_1 = s, v_2, \dots, v_k = t \rangle$, $(v_i, v_{i+1}) \in E \ \forall 1 \le i < k$

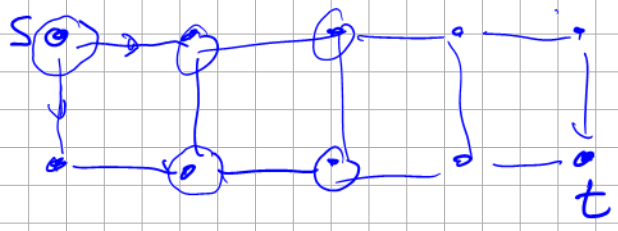
$c(p) := \sum_{i=1}^{k-1} c(v_i, v_{i+1})$

$$\delta(u,v) := \begin{cases} \infty, & \text{kein Pfad } u \rightsquigarrow v \\ \min \{ c(p) : u \overset{p}{\rightsquigarrow} v \} & \end{cases}$$

Beobachtungen

(1) Es muss keinen kürzesten Weg geben:

(2) Es kann exponentiell viele Wege von s nach t geben

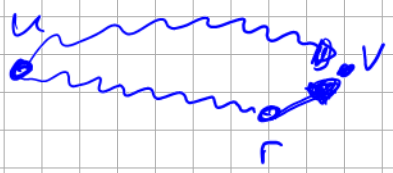


$\rightarrow 2^{n/2}$ Wege

(3) Dreiecksungleichung

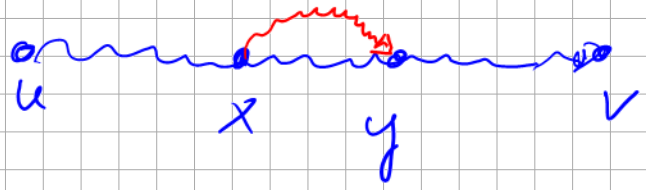
$$\delta(u,v) \leq \delta(u,r) + \delta(r,v)$$

insbesondere: $\delta(u,v) \leq \delta(u,r) + c(r,v)$



(4.) Optimale Substruktur

Teilpfade kürzester Pfade sind kürzeste Pfade



(5.) Kürzeste Pfade enthalten keine Zyklen

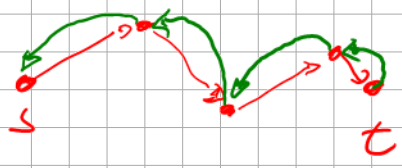
(a) neg. Zyklus $\Rightarrow$ $\exists$ kein kürzester Pfad

(b) pos. Zyklus $\Rightarrow$ Weglassen verbessert die Lösung

(c) Zyklen mit Gewicht 0

Konvention: weglassen

Datenstrukturen



$n = |V|$

$d_s \in \mathbb{R}^n$ Distanz, initial $d_s[s] = 0$

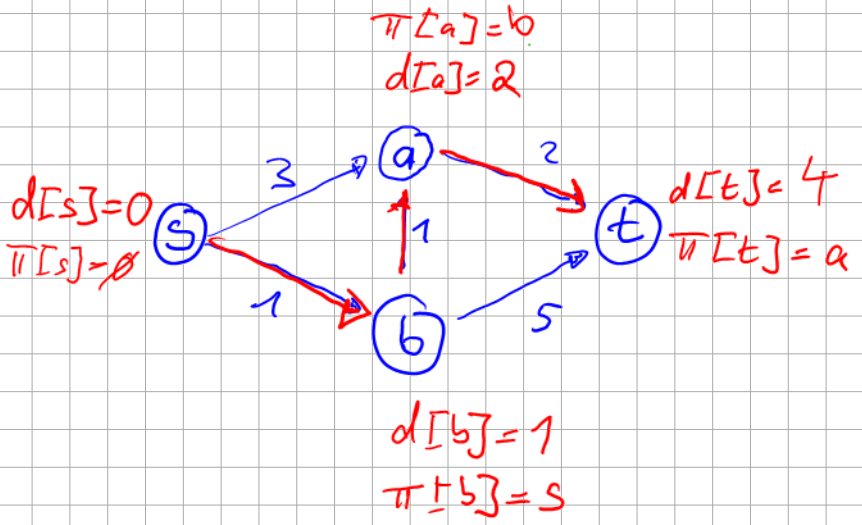
$d_s[v] = \infty \quad \forall v \neq s$

$\pi_s \in V^n$ Vorgänger

$\pi_s[v] = \text{null} \quad \forall v \in V$

Ziel: $d_s[t] = \delta(s,t)$

$d_s[v] = \delta(s,v) \quad \forall v \in V$



Allgemeines (Relaxier-) Algorithmus

① for each $u \in V$
  | $d[u] \leftarrow \infty$
  | $\pi[u] \leftarrow \text{null}$
  Init
$d[s] \leftarrow 0$

② Wähle eine Kante $(u,v) \in E$ — welche?
  if $d[u] + c(u,v) < d[v]$
    $d[v] \leftarrow d[u] + c(u,v)$
    $\pi[v] \leftarrow u$
    } Relax(u,v)

③ Wiederhole ② bis keine Relaxierungen mehr

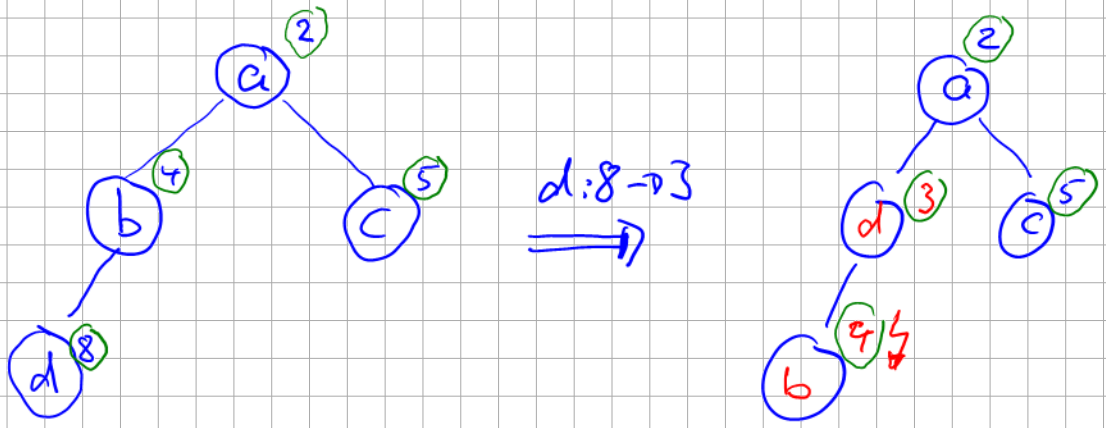
Dijkstra-Algorithmus

① foreach $u \in V$
  | $d[u] \leftarrow \infty$, $\pi[u] \leftarrow \text{null}$
$d[s] \leftarrow 0$
$R \leftarrow \{s\}$ — R: Minheap

② while $R \neq \emptyset$
  $u \leftarrow$ extractMin(R) [$R \leftarrow R - \{u\}$] — $n \log n$
  foreach $v \in N^+(u)$
    if $d[u] + c(u,v) < d[v]$ ← Relax(u,v)
      $d[v] \leftarrow d[u] + c(u,v)$
      $\pi[v] \leftarrow u$
      $R \leftarrow R \cup \{v\}$ ← decreaseKey $n \log n$ / Anpassen des Minheaps / Hinzufügen zu R $n \log n$

Laufzeit: $\mathcal{O}((n+m) \log n)$

Heaps und DecreaseKey



Problem: Position von d im Heap ist unbekannt

Lösungen

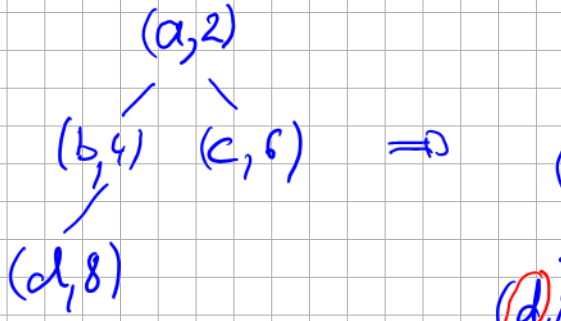
Position des Knoten gespeichert wird

(a) Position im Heap speichern an Knoten

(b) " " " mit Hashtabelle Knoten → Position

(c) Lazy Deletion

Paare speichern (node, d), neuen einfügen



$m \log n$

$\Rightarrow$

$m \log m = \Theta(m \log n)$

$m \in \Theta(n^2)$