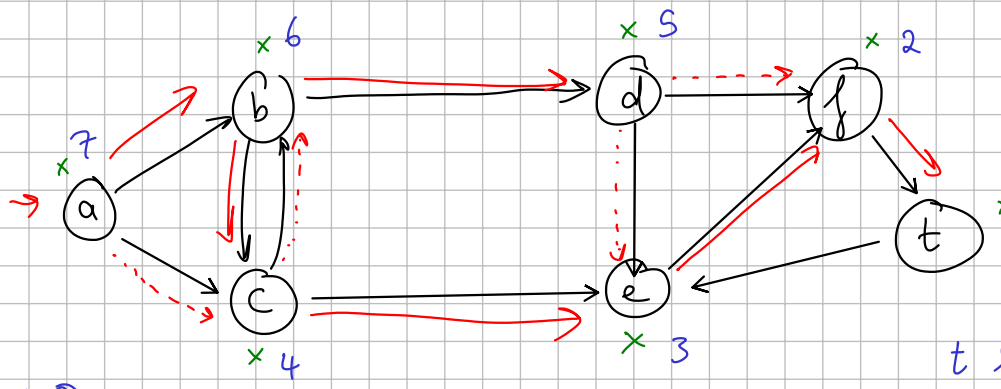
Tiefensuche (Depth-First-Search DFS)

```
main
for v∈V:
    if !entdeckt[v]:
        DFS(v)
```
wenn der ganze Graph besucht werden soll



reverse post-order entspricht einer top. sortierung

t f e c d b a

a b d c e f t

counter ← 0

entdeckt[...] ← false $(\forall v \in V : v.entdeckt = false)$

DFS(v): $O(n+m)$

```
    entdeckt[v] ← true
    für alle (v,u) ∈ E:
        if !entdeckt[u]:
            DFS(u)
            vorgänger[u] ← v
```
} intuitiv

Nachdem der Aufruf von DFS(v) terminiert, sind alle von v aus erreichbare Knoten als entdeckt markiert.

$$v \to u_1 \to u_2 \to \dots \to u_i \to u_{i+1} \to \dots \to u = u_k$$

Bew. Induktion über i (i-ter Knoten auf dem Pfad von v zu $u = u_k$ im DFS-Baum)

Basis $i=1$: ✓ u_1 wird durch DFS(v) entdeckt

Schritt: $i \to i+1$: Da die Kante (u_i, u_{i+1}) existiert, wird u_{i+1} bei DFS(u_i) entdeckt

```
DFS(v):
  if finished[v] return
  if visited[v] print("CYCLE")
  visited[v] = true
  for (v,u) ∈ E : DFS(u)
  finished[v] = true
  counter[v] = counter      } post-order-Nummerierung
  counter++
```

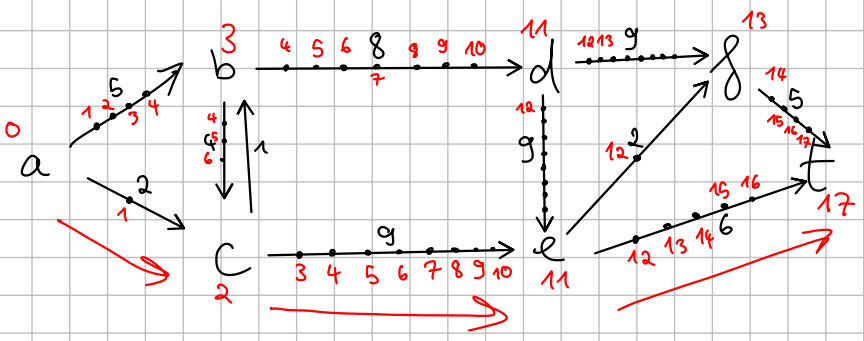
} Zyklen + top. Sortierung

Dijkstra's Algorithmus

BFS kann kürzeste Pfade (gemessen in # Kanten) berechnen.

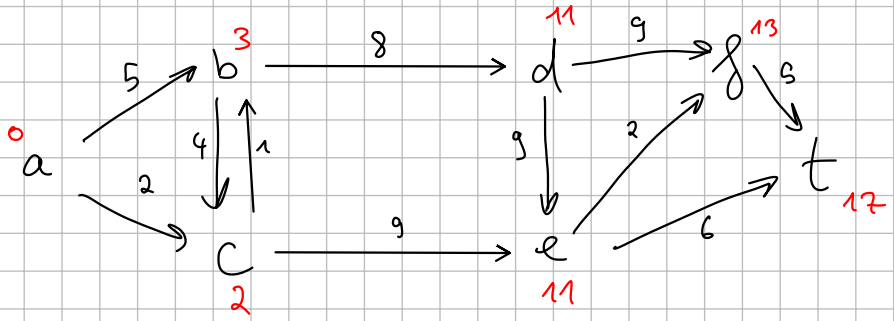
Wir wollen beliebige Kantengewichte c_{uv}

Dijkstra's Alg. funktioniert für $c_{uv} \geq 0$



$d(a,t) = 17$

Bei der BFS-Simulation schauen wir uns in jedem Schritt alle Kanten an, die von bereits entdeckten zu noch nicht entdeckten Knoten führen



$F = \{s\}$ $d[s] = 0$ (s Startknoten)

$D = 0$

while $D < \infty$

$O(n)$

$D \leftarrow \min \{ d[u] + c_{uv} \mid (u,v) \in E, u \in F, v \notin F \}$

$(\bar{u},\bar{v}) \leftarrow \underset{(u,v)}{\operatorname{argmin}} \{ d[u] + c_{uv} \mid (u,v) \in E, u \in F, v \notin F \}$ $O(n^2)$ $O(n^3)$

$F \leftarrow F \cup \{\bar{v}\}$

$d[\bar{v}] \leftarrow D$

nächstes mal $O(n^3) \rightarrow O(n^2)$