Plan

- Administrative Information
- Short theory input
- Discussion Assignment 9
- Theory Recap
- Exercise 10.4 d.)
- Peer-Grading (ex. 10.4)

Administrative Information

- Graphcheatsheet updated (Bellman-Ford, Boruvka, Prim)
- Mock Exam (19.12, 16:00–18:00)
- Endterm exams (corrections might be delayed)

Theory Input I

- Runtime for accessing elements in an adjacency list is $O(1+deg(v))$
  - ↳ 1 for accesing the nested list
  - ↳ $deg(v)$ for accessing elements in that list
- Runtime of DFS follows from that:
  - ↳ $visit(v)$: $O(1+deg_{out}(v))$ (+ recursion)
  - ↳ Total: $\sum_{v \in V} O(1+deg_{out}(v)) = O(|V|+|E|)$

Discussion Assignment 9

- 9.2
  - was solved quite well
  - counterexamples weren't always correct (there was a longest path with a sink)
- 9.4
  - a lot of small mistakes
  - a,b assuming v_1 / v_n are the only source/sink → topological sorting not unique
    assuming the graph is connected / $(v_1 - v_n)$-path is unique
  - c using DP-Table that results in $O(n \cdot (n+m))$ runtime
- 9.5
  - u,v strongly connected ⇒ ∃ directed cycle between u and v ⇒ ∃ back edge
  - a lot of algorithms that weren't introduced in the course (make sure to understand & proof correctness/runtime)
    - ↳ usually exercises are solveable with ideas from the course
  - using non $O(1)$ datastructures for "visited"
  - be careful with ChatGPT for graph-theory
    - ↳ not well trained on this subject
  - assuming there are no cross-edges

Theory Input II

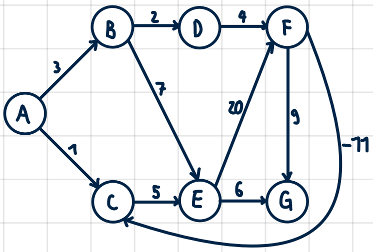
Bellman-Ford

- unlike Dijkstra also works with neg. Weights
- $O(n \cdot m)$ Runtime
  ↳ $(n-1)$ iterations, each iteration $O(m)$

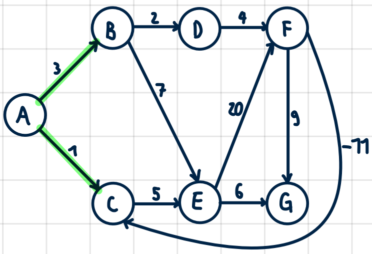
Algorithm 7 Bellman-Ford(s)

```
d[s] ← 0;   d[v] ← ∞ ∀v ∈ V \ {s}                    ▷ 0-gute Schranken
for i ∈ {1, . . . n − 1} do                           ▷ Verbessere Schranken (n − 1)-mal
    for (u, v) ∈ E do
        d[v] ← min{d[v], d[u] + c(u, v)}
```

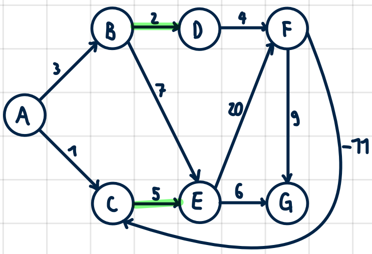
Example:



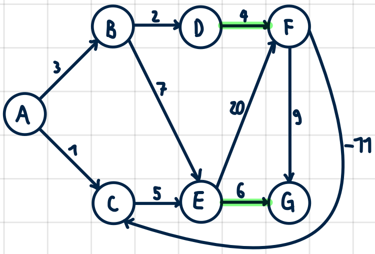
v:	A	B	C	D	E	F	G
d[v]	0	∞	∞	∞	∞	∞	∞



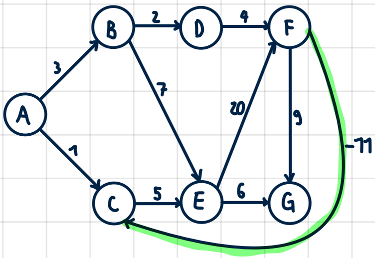
v:	A	B	C	D	E	F	G
d[v]	0	3	1	∞	∞	∞	∞



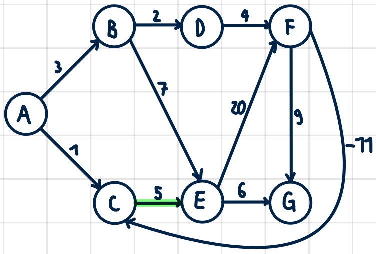
v:	A	B	C	D	E	F	G
d[v]	0	3	1	5	6	∞	∞



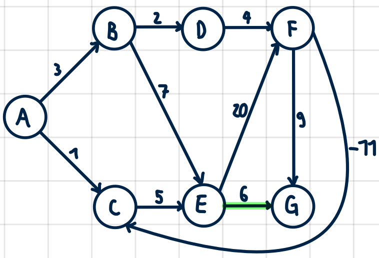
v:	A	B	C	D	E	F	G
d[v]	0	3	1	5	6	9	12



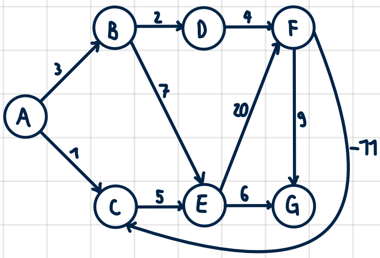
v:	A	B	C	D	E	F	G
d[v]	0	3	-2	5	6	9	12



v:	A	B	C	D	E	F	G
d[v]	0	3	-2	5	3	9	12



v:	A	B	C	D	E	F	G
d[v]	0	3	-2	5	3	9	9



v:	A	B	C	D	E	F	G
d[v]	0	3	-2	5	3	9	9

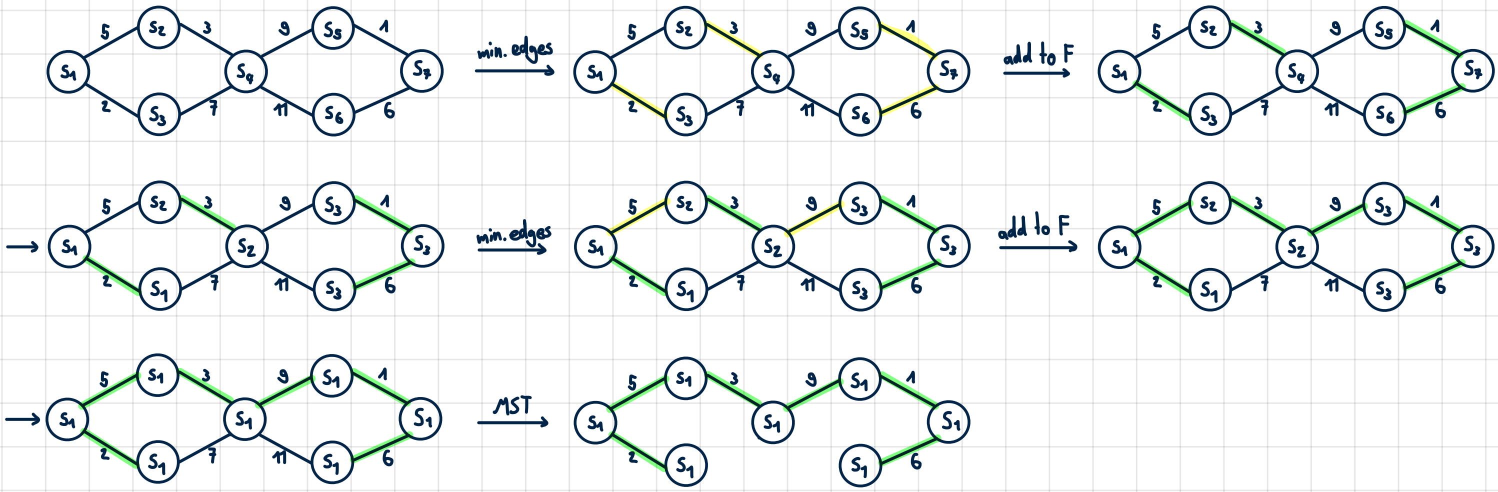
Boruvka

- Runtime $O((m+n)\cdot \log n)$
- Find MST
- usually only used on undirected graphs

Algorithm 8 Boruvka(G)

$F \leftarrow \varnothing$ ▷ sichere Kanten
while F nicht Spannbaum **do** ▷ $\leq log(n)$ Iterationen, $\mathcal{O}(m+n)$ pro Iteration
  $(S_1, \dots, S_k) \leftarrow$ ZHKs von F
  $(e_1, \dots, e_k) \leftarrow$ minimale Kanten an $S_1, \dots, S_k$
  $F \leftarrow F \cup \{e_1, \dots, e_k\}$

Example:



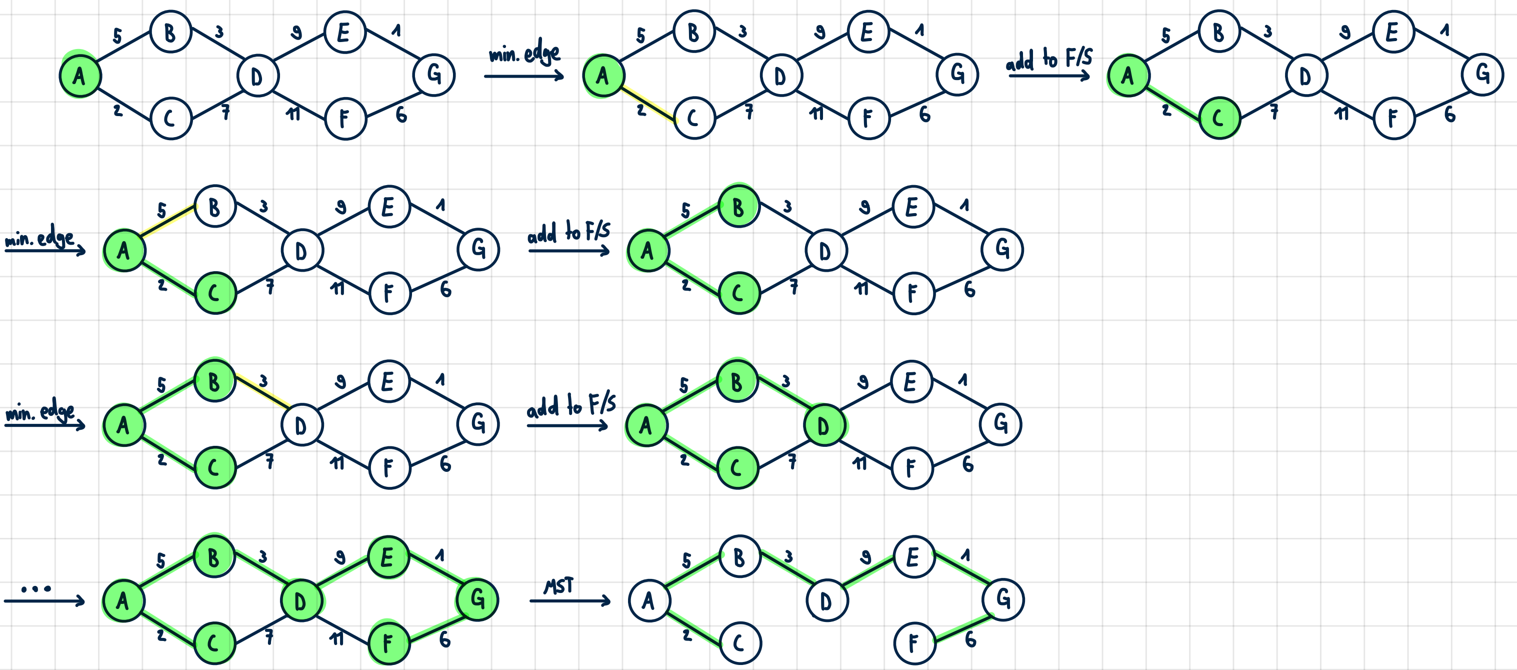
Prim

- Runtime $O((m+n)\cdot \log n)$
- Idea: Focus on one CC (ZHK)
  ↳ hence additional input s

Algorithm 9 Prim(G, s) (allgemeine Form)

$F \leftarrow \varnothing$
$S \leftarrow \{s\}$ ▷ ZHK von s in F
while F nicht Spannbaum **do**
  $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
  $F \leftarrow F \cup \{u^*v^*\}$
  $S \leftarrow S \cup \{v^*\}$

Example:



10.4.) d.) alternative solution (Kosarajus Algorithm)

Notation

- ADJ: adjacency list (array syntax used for readability)
- $\overrightarrow{ADJ}$: transposed adjacency list
- visited: boolean array of size n //flags vertices
- S: stack, vertices will be added in post-order
- SCC: int array of size n, entry is the corresponding SCC

Routines

```
visitS(u)
visited[u] := true
for v ∈ ADJ[u] with visited[v] = false
    visitS(v)
S.push(u)                      // here the post counter would be updated
```

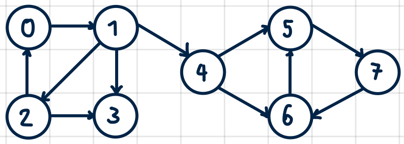
```
visitSCC(u)
visited[u] := true
SCC[u] := SCCcount
for v ∈ ADJ→[u] with visited[v] = false
    visitSSC(v)
```

```
Kosaraju(G)
S := ∅; SCC[v] := -1 ∀v∈V; visited[v] := false ∀v∈V; ADJ→ := transpose(ADJ)
for v ∈ V with visited[v] = false
    visitS(v)
visited[v] := false ∀v∈V
SCCcount := 0
while S ≠ ∅
    v := S.pop()
    if visited[v] = false
        visitSSC(v)
        SCCcount := SCCcount + 1
return SCC
```

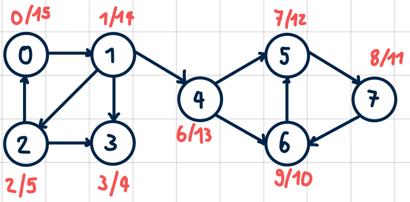
- initialization $O(|V|+|E|)$
- first DFS, $O(|V|+|E|)$
- reset flags, $O(|V|)$
- second DFS, computation of SCCs, $O(|V|+|E|)$

Runtime (follows from comments): $O(|V|+|E|)+O(|V|+|E|)+O(|V|)+O(|V|+|E|) = O(|V|+|E|)$, assuming adjacency list!

Example



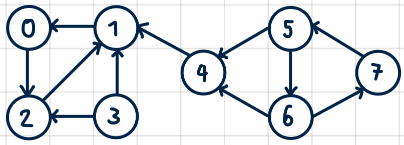
1. first DFS, build stack



top

⇒ $S = (0,1,4,5,7,6,2,3)$, is in reversed post order
(top of stack is starting vertex of one of the CCs (ZHKs), in our case only one CC)

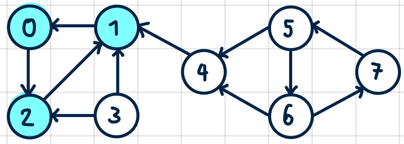
2. second DFS (in order of stack pops, on transposed graph)



$S = (0,1,4,5,7,6,2,3)$

v	0	1	2	3	4	5	6	7
SCC	-1	-1	-1	-1	-1	-1	-1	-1
Visited	F	F	F	F	F	F	F	F

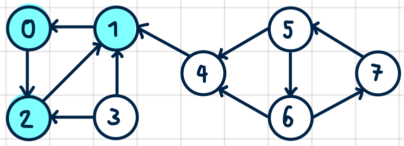
pop() → 0
visitSCC(0)



$S = (1,4,5,7,6,2,3)$

v	0	1	2	3	4	5	6	7
SCC	0	0	0	-1	-1	-1	-1	-1
Visited	T	T	T	F	F	F	F	F

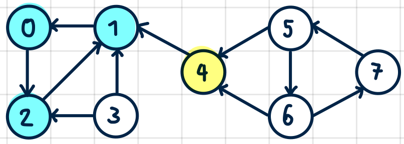
pop visited



$S = (4,5,7,6,2,3)$

v	0	1	2	3	4	5	6	7
SCC	0	0	0	-1	-1	-1	-1	-1
Visited	T	T	T	F	F	F	F	F

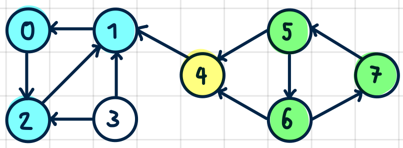
pop() → 4
visitSCC(4)



$S = (5,7,6,2,3)$

v	0	1	2	3	4	5	6	7
SCC	0	0	0	-1	1	-1	-1	-1
Visited	T	T	T	F	T	F	F	F

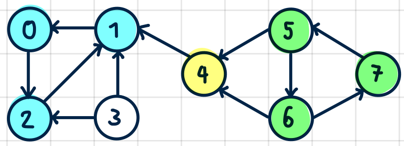
pop() → 5
visitSCC(5)



$S = (7,6,2,3)$

v	0	1	2	3	4	5	6	7
SCC	0	0	0	-1	1	2	2	2
Visited	T	T	T	F	T	T	T	T

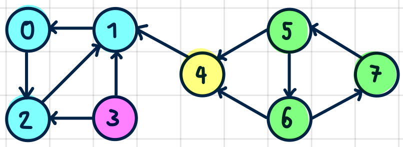
pop visited



$S = (3)$

v	0	1	2	3	4	5	6	7
SCC	0	0	0	-1	1	2	2	2
Visited	T	T	T	F	T	T	T	T

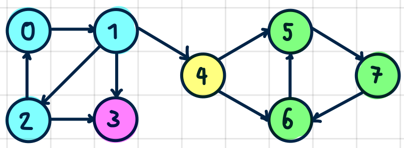
pop() → 3
visitSCC(3)



$S = \emptyset$

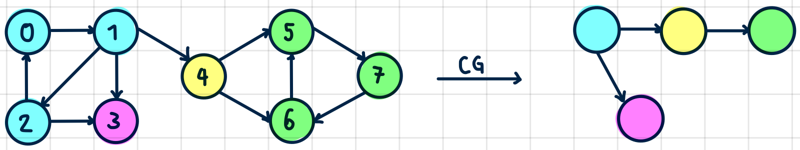
v	0	1	2	3	4	5	6	7
SCC	0	0	0	3	1	2	2	2
Visited	T	T	T	T	T	T	T	T

In orig. graph



Why does this work:

I.) consider the condensed graph (CG)



- G contains edge (u,v) with SCC[u] ≠ SCC[v] ⇒ CG contains edge (SCC[u], SCC[v])
- CG is a DAG, otherwise all SCCs in a cycle could've been merged into a single SCC

II.) Consider the traversal order

- CG is a DAG ⇒ CG has a top. sorting
  - If SCC2 goes after SCC1 then atleast one vertex of SCC1 will be higher in the stack than all vertices of SCC2

III.) Consider the transpose graph (TG)

- The TG has the same SCCs as the orig. graph
  - Semantically: only edges between different SCCs are inverted
- Inverting edges isolates SCCs
  - DFS can only enter the current SCC and SCCs already visited (which we ignore with the if-statement)