Datenstrukturen und Algorithmen

Exercise 10

FS 2020

No exercise session

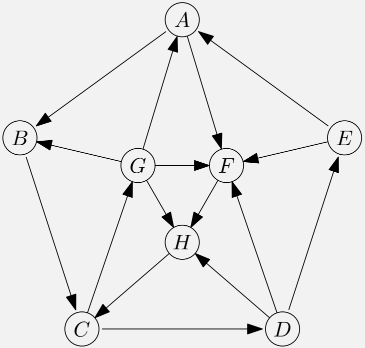
Even if no exercise sessions take place this week, we did not want to deny you the exercise slides.

Program of today

1. Feedback of last exercises
2. Recap Lecture Material
   - Algorithm Jarnik, Prim, Dijkstra
3. Programming Task

1. Feedback of last exercises

Depth-first-search and Breadth-first-search

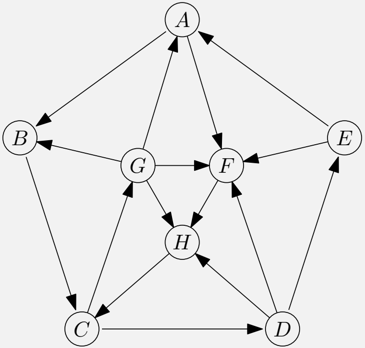


Starting at A

DFS: A, B, C, D, E, F, H, G

BFS: A, B, F, C, H, D, G, E

Depth-first-search and Breadth-first-search



Starting at A

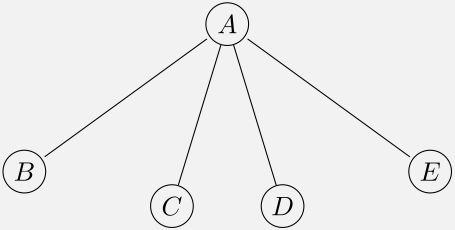
DFS: A, B, C, D, E, F, H, G

BFS: A, B, F, C, H, D, G, E

There is no starting vertex where the DFS ordering equals the BFS ordering.

Depth-first-search and Breadth-first-search

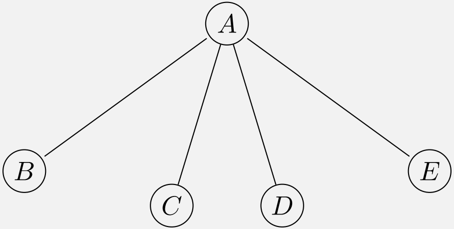
Star: DFS ordering equals BFS ordering



Starting at A
DFS: A, B, C, D, E
BFS: A, B, C, D, E

Depth-first-search and Breadth-first-search

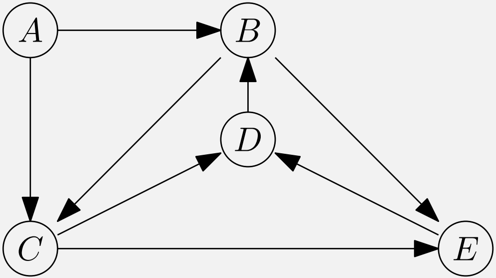
Star: DFS ordering equals BFS ordering



Starting at A
DFS: A, B, C, D, E
BFS: A, B, C, D, E

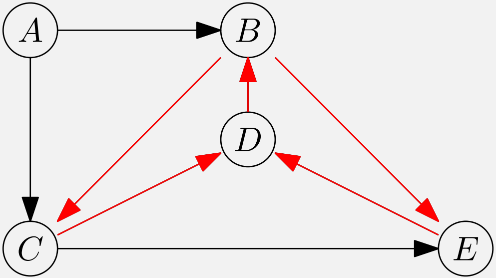
Starting at C
DFS: C, A, B, D, E
BFS: C, A, B, D, E

Topological Sorting



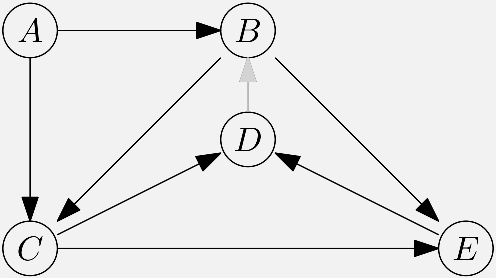
- Graph with cycles

Topological Sorting



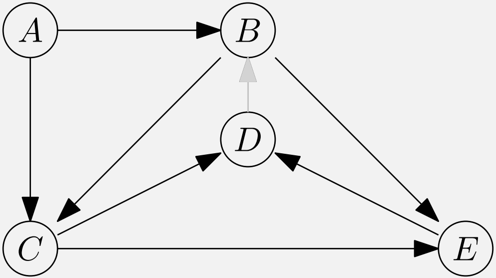
- Graph with cycles
- Two minimal cycles sharing an edge

Topological Sorting



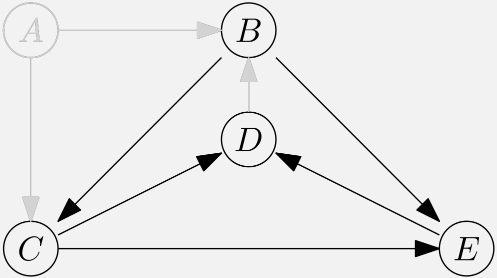
- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free

Topological Sorting



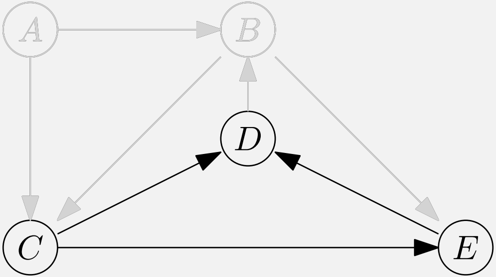
- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free
- Topological Sorting by "removing" elements with in-degree 0

Topological Sorting



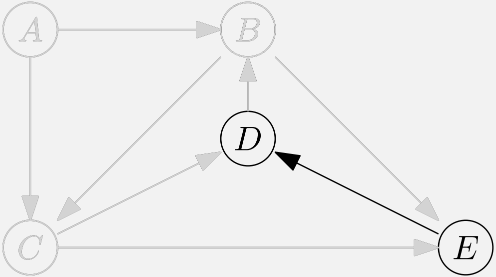
- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free
- Topological Sorting by "removing" elements with in-degree 0

Topological Sorting



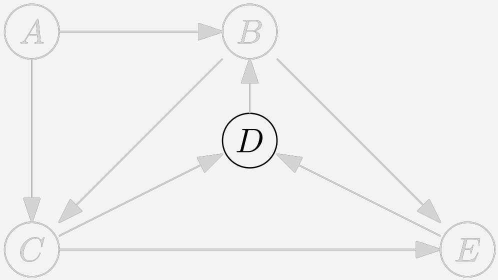
- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free
- Topological Sorting by “removing” elements with in-degree 0

Topological Sorting



- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free
- Topological Sorting by "removing" elements with in-degree 0

Topological Sorting



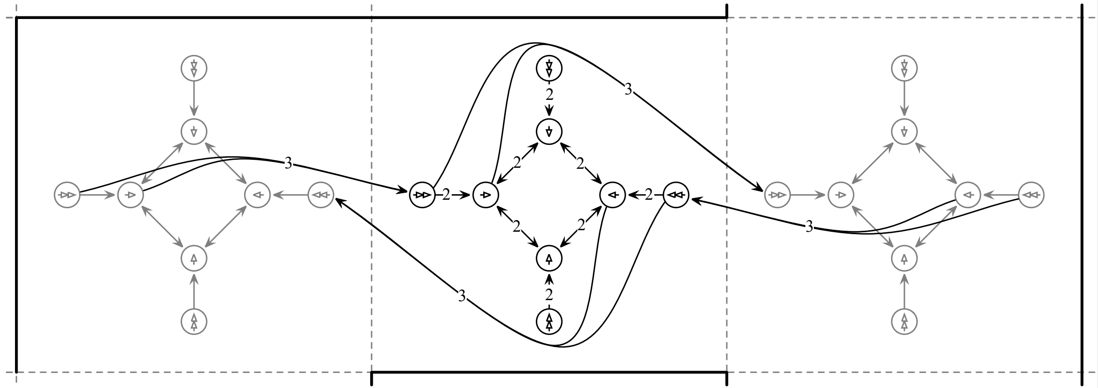
- Graph with cycles
- Two minimal cycles sharing an edge
- Remove edge $\implies$ cycle-free
- Topological Sorting by “removing” elements with in-degree 0

Exercise : Labyrinth

- Robot has to stop to change direction
- Interpret as shortest path problem

Exercise : Labyrinth

- position × direction × speed



- Runtime?

Exercise Labyrinth

- Let n be the number of squares. Graph has $|V| = 8n$ nodes
- Graph has at $|E| \leq 20n$ edges
- Therefore, Dijkstra $\mathcal{O}(|E| + |V| \log |V|)$ has runtime $\mathcal{O}(n \log n)$

2. Recap Lecture Material

A*-Algorithm($G, s, t, \hat{h}$)

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$, end point $t \in V$, estimate $\widehat{h}(v) \leq \delta(v, t)$
Output: Existence and value of a shortest path from s to t

foreach $u \in V$ **do**
  $d[u] \leftarrow \infty$; $\widehat{f}[u] \leftarrow \infty$; $\pi[u] \leftarrow$ null

$d[s] \leftarrow 0$; $\widehat{f}[s] \leftarrow \widehat{h}(s)$; $R \leftarrow \{s\}$; $M \leftarrow \{\}$
while $R \neq \emptyset$ **do**
  $u \leftarrow$ ExtractMin$_{\widehat{f}}(R)$; $M \leftarrow M \cup \{u\}$
  if $u = t$ **then return** success
  foreach $v \in N^+(u)$ with $d[v] > d[u] + c(u, v)$ **do**
    $d[v] \leftarrow d[u] + c(u, v)$; $\widehat{f}[v] \leftarrow d[v] + \widehat{h}(v)$; $\pi[v] \leftarrow u$
    $R \leftarrow R \cup \{v\}$; $M \leftarrow M - \{v\}$

return failure

DP Algorithm Floyd-Warshall(G)

Input: Acyclic Graph $G = (V, E, c)$
Output: Minimal weights of all paths d
$d^0 \leftarrow c$
for $k \leftarrow 1$ **to** $|V|$ **do**
  for $i \leftarrow 1$ **to** $|V|$ **do**
    for $j \leftarrow 1$ **to** $|V|$ **do**
      $d^k(v_i, v_j) = \min\{d^{k-1}(v_i, v_j), d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j)\}$

Runtime: $\Theta(|V|^3)$

Remark: Algorithm can be executed with a single matrix d (in place).

Algorithm Johnson(G)

Input: Weighted Graph $G = (V, E, c)$
Output: Minimal weights of all paths D.

New node s. Compute $G' = (V', E', c')$
if BellmanFord(G', s) $=$ false **then** return "graph has negative cycles"
foreach $v \in V'$ **do**
    $h(v) \leftarrow d(s, v)$ // d aus BellmanFord Algorithmus

foreach $(u, v) \in E'$ **do**
    $\tilde{c}(u, v) \leftarrow c(u, v) + h(u) - h(v)$

foreach $u \in V$ **do**
    $\tilde{d}(u, \cdot) \leftarrow$ Dijkstra($\tilde{G}', u$)
    foreach $v \in V$ **do**
        $D(u, v) \leftarrow \tilde{d}(u, v) + h(v) - h(u)$

Comparison of the approaches

Algorithm			Runtime
Dijkstra (Heap)	$c_v \geq 0$	1:n	$\mathcal{O}(\lvert E\rvert \log \lvert V\rvert)$
Dijkstra (Fibonacci-Heap)	$c_v \geq 0$	1:n	$\mathcal{O}(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$ *
Bellman-Ford		1:n	$\mathcal{O}(\lvert E\rvert \cdot \lvert V\rvert)$
Floyd-Warshall		n:n	$\Theta(\lvert V\rvert^3)$
Johnson		n:n	$\mathcal{O}(\lvert V\rvert \cdot \lvert E\rvert \cdot \log \lvert V\rvert)$
Johnson (Fibonacci-Heap)		n:n	$\mathcal{O}(\lvert V\rvert^2 \log \lvert V\rvert + \lvert V\rvert \cdot \lvert E\rvert)$ *

* amortized

Johnson is better than Floyd-Warshall for sparse graphs ($|E| \approx \Theta(|V|)$).

Union-Find Algorithm MST-Kruskal(G)

Input: Weighted Graph $G = (V, E, c)$
Output: Minimum spanning tree with edges A.

Sort edges by weight $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** m **do**
  MakeSet(k)

for $k = 1$ **to** m **do**
  $(u, v) \leftarrow e_k$
  if Find(u) $\neq$ Find(v) **then**
    Union(Find(u), Find(v)) // conceptual: $A \leftarrow A \cup e_k$
  else // conceptual: $R \leftarrow R \cup e_k$

return (V, A, c)

Implementation Union-Find

Index	1	2	3	4	5	6	7	8	9	10
Parent	**1**	1	1	6	**5**	**6**	5	5	3	**10**

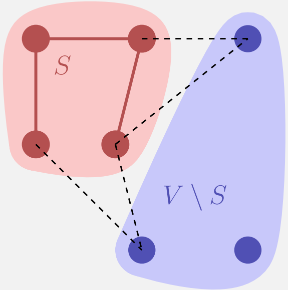
Operations:

- Make-Set(i): $p[i] \leftarrow i$; **return** i
- Find(i): **while** $(p[i] \neq i)$ **do** $i \leftarrow p[i]$; **return** i
- Union(i, j): $p[j] \leftarrow i$; **return** i

MST algorithm of Jarnik, Prim, Dijkstra

Idea: start with some $v \in V$ and grow the spanning tree from here by the acceptance rule.

$S \leftarrow \{v_0\}$
for $i \leftarrow 1$ **to** $|V|$ **do**
  Choose cheapest (u, v) mit $u \in S$, $v \notin S$
  // conceptual $A \leftarrow A \cup \{(u, v)\}$
  $S \leftarrow S \cup \{v\}$ // (Coloring)



Remark: a union-Find data structure is not required. It suffices to color nodes when they are added to S.

Running time

Trivially $\mathcal{O}(|V| \cdot |E|)$.

Improvements (like with Dijkstra's ShortestPath)

- Memorize cheapest edge to S: for each $v \in V \setminus S$. $\deg^+(v)$ many updates for each new $v \in S$. Costs: $|V|$ many minima and updates: $\mathcal{O}(|V|^2 + \sum_{v\in V} \deg^+(v)) = \mathcal{O}(|V|^2 + |E|)$
- With Minheap: costs $|V|$ many minima $= \mathcal{O}(|V| \log |V|)$, $|E|$ Updates: $\mathcal{O}(|E| \log |V|)$, Initialization $\mathcal{O}(|V|)$: $\mathcal{O}(|E| \cdot \log |V|.)$
- With a Fibonacci-Heap: $\mathcal{O}(|E| + |V| \cdot \log |V|)$.

3. Programming Task

Closeness Centrality

- Given: an adjacency matrix for an *undirected* graph on n vertices.
- Output: the *closeness centrality* $C(v)$ of every vertex v.

$$C(v) = \sum_{u \in V \setminus \{v\}} d(v, u)$$

Closeness Centrality

- Given: an adjacency matrix for an *undirected* graph on n vertices.
- Output: the *closeness centrality* $C(v)$ of every vertex v.

$$C(v) = \sum_{u \in V \setminus \{v\}} d(v, u)$$

- Intuition: If many connected vertices are close to v, then $C(v)$ is small.
- "How central is the vertex in its connected component?"

All Pairs Shortest Paths

- We require $d(u, v)$ for all vertex pairs (u, v).
- $\implies$ compute all shortest paths using Floyd-Warshall.

```
template<typename Matrix>
void allPairsShortestPaths(unsigned n, Matrix& m)
{
  // your code here
```

- Simply overwrite `m` with the distance values.
- Attention: initially 0 means “no edge”.
- Undirected graph: `m[i][j] == m[j][i]`

Closeness Centrality

```
vector<vector<unsigned> > adjacencies(n,
                 vector<unsigned>(n, 0));
vector<string> names(n);
// ...
allPairsShortestPaths(n, adjacencies);
for(unsigned i = 0; i < n; ++i) {
  cout << names[i] << ": ";
  unsigned centrality = 0;
  // your code here
  cout << centrality << endl;
}
```

Closeness Centrality: Input Data

- A graph that stems from collaborations on scientific papers.
- The vertices of the graph are the co-authors of the mathematician Paul Erdős.
- There is an edge between them if the authors have jointly published a paper.
- Source: `https://oakland.edu/enp/thedata/`

Closeness Centrality: Output

```
vertices: 511
ABBOTT, HARVEY LESLIE                    : 1625
ACZEL, JANOS D.                          : 1681
AGOH, TAKASHI                            : 2132
AHARONI, RON                             : 1578
AIGNER, MARTIN S.                        : 1589
AJTAI, MIKLOS                            : 1492
ALAOGLU, LEONIDAS*                       : 0
ALAVI, YOUSEF                            : 1561
...
```

Where does the 0 come from?

Task Union Find

- Input: *union* operations to be performed, followed by queries if they are located in the same set.
- Output: For each query, answer if they are in the same set.
- Make sure you can re-use your code in the next task.

Task Kruskal's MST algorithm

- Edges have to be sorted.

Task Kruskal's MST algorithm

- Edges have to be sorted.
- Create an *Edge* class that implements the comparison operator.
- Then use *std::sort*.

Questions?