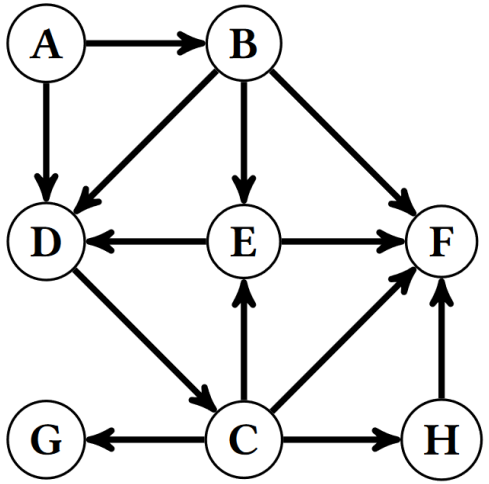
ALGORITHMS & DATA STRUCTURES

6th December 2021

TODAY’S PLAN

- Bonus Exercises
- Shortest Paths: Recap
- LeetCode Problem

EXERCISE 10.1



a) Provide the BFS order of visit of the nodes.

b) Provide the enter and leave time of each node.

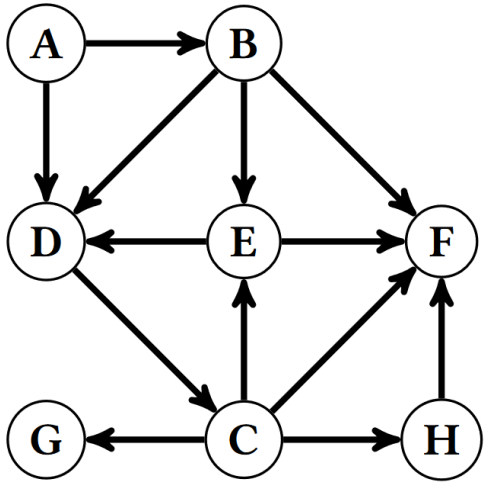
c) Indicate the shortest-path-tree that is obtained by BFS.

d) Determine the distance from A to every node in the graph.

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

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

	Entry	Leaving
A	1	2
B	3	5
C	9	12
D	4	8
E	6	10
F	7	11
G	13	15
H	14	16



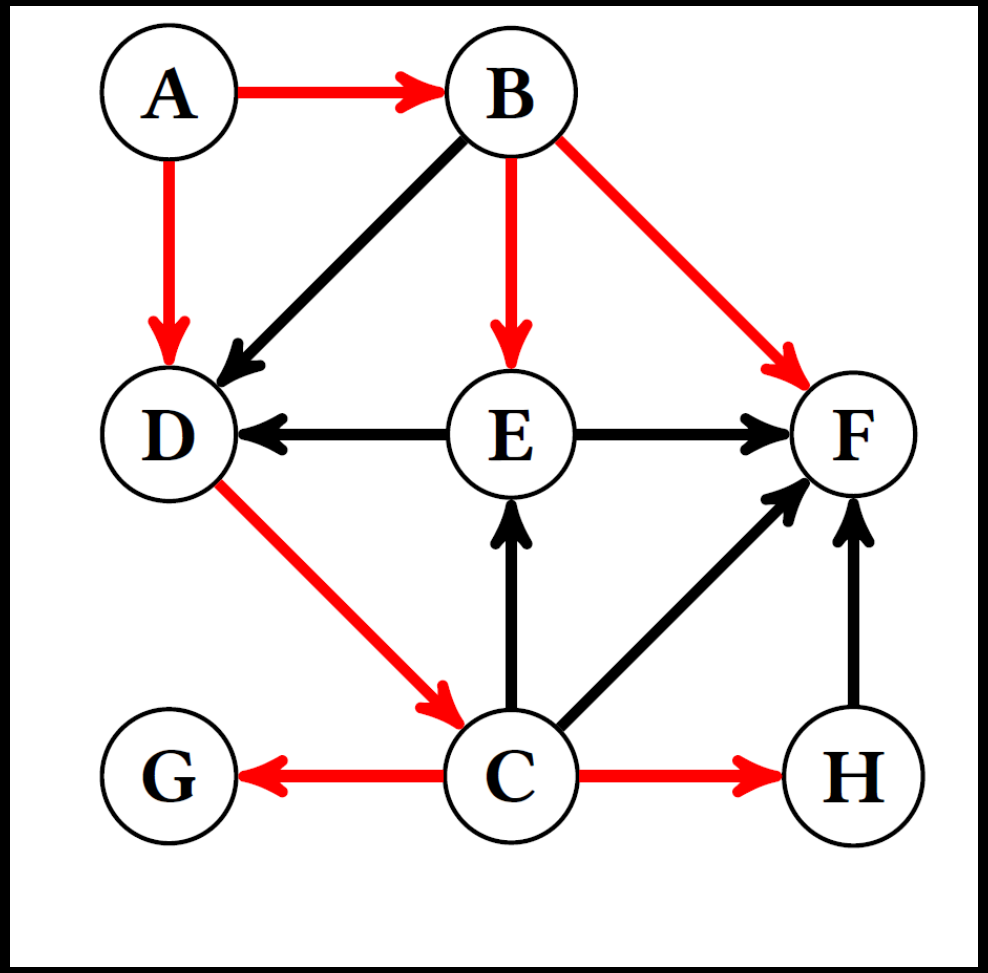
a) Provide the BFS order of visit of the nodes.

b) Provide the enter and leave time of each node.

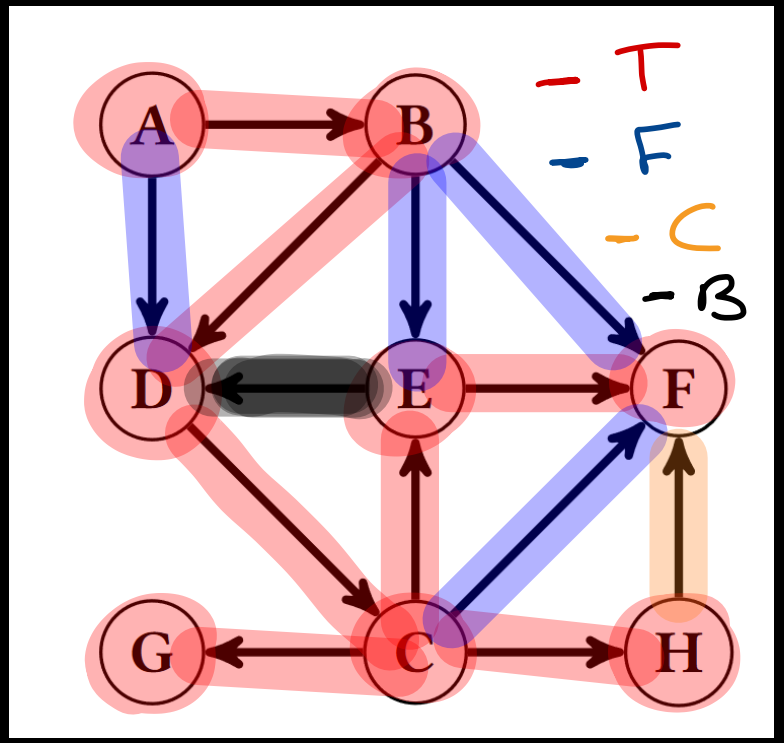
c) Indicate the shortest-path-tree that is obtained by BFS.

d) Determine the distance from A to every node in the graph.

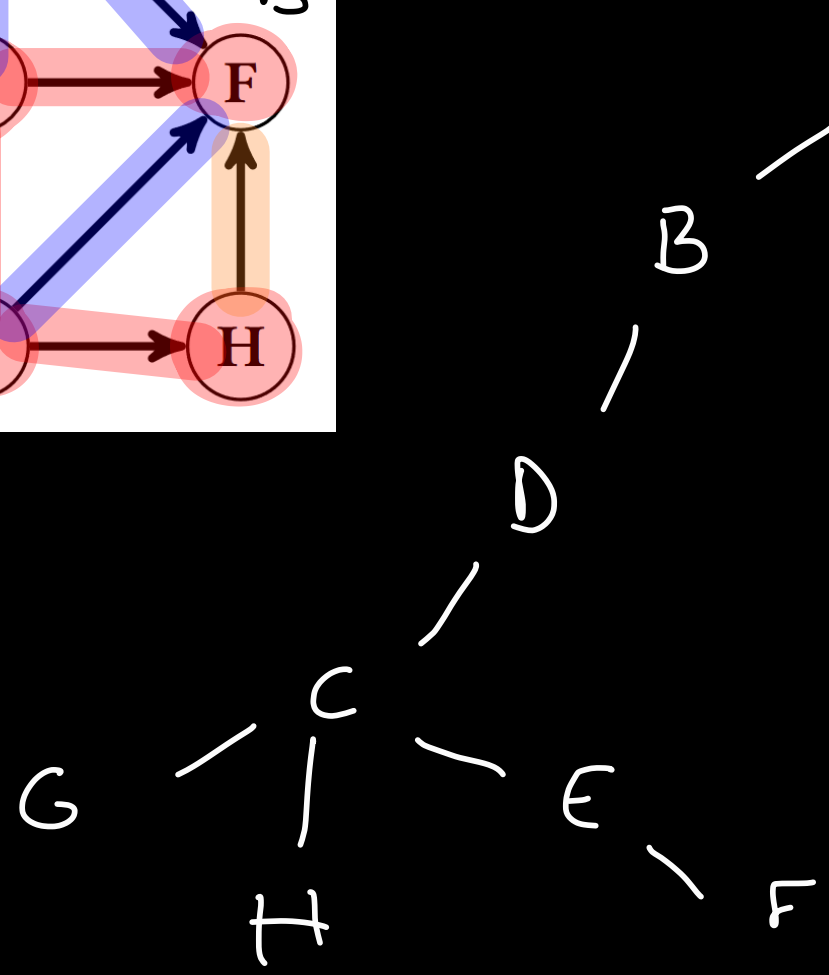
A	B	C	D	E	F	G	H
0	1	2	1	2	2	3	3

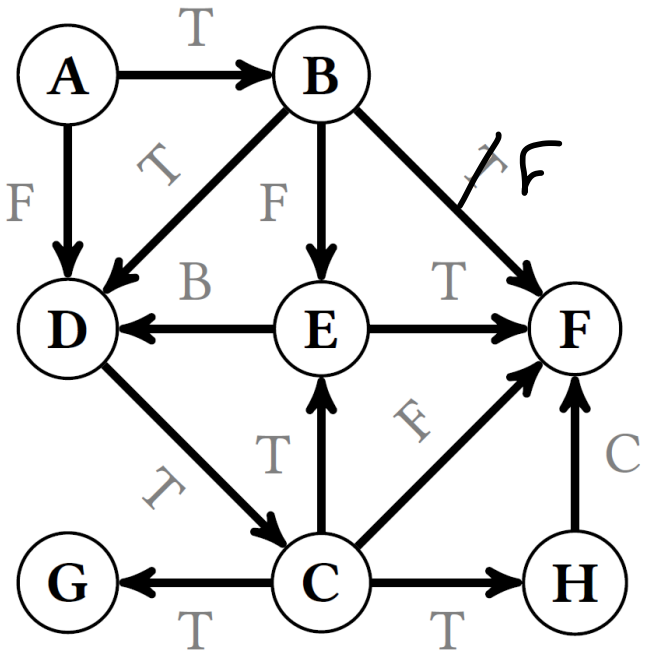


EXERCISE 10.2



A ; B ; D ; C ; E ; F ; G ; H

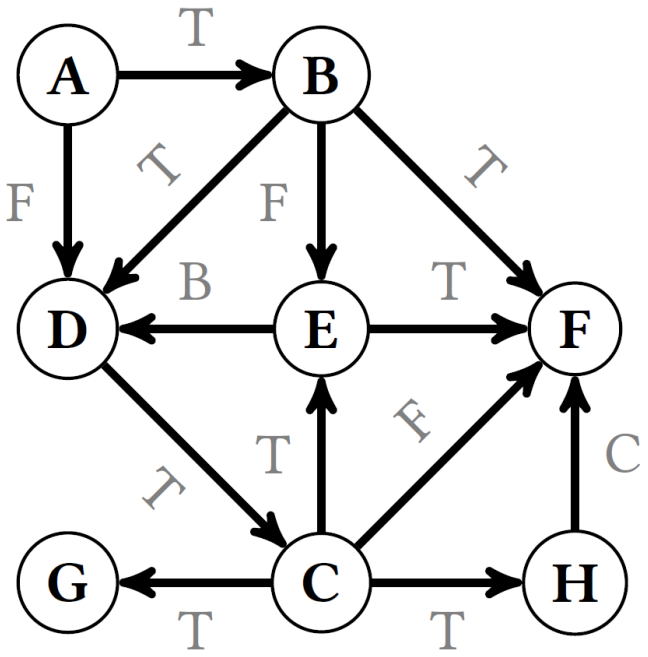




b) For each vertex, give its *pre*- and *post*-number.

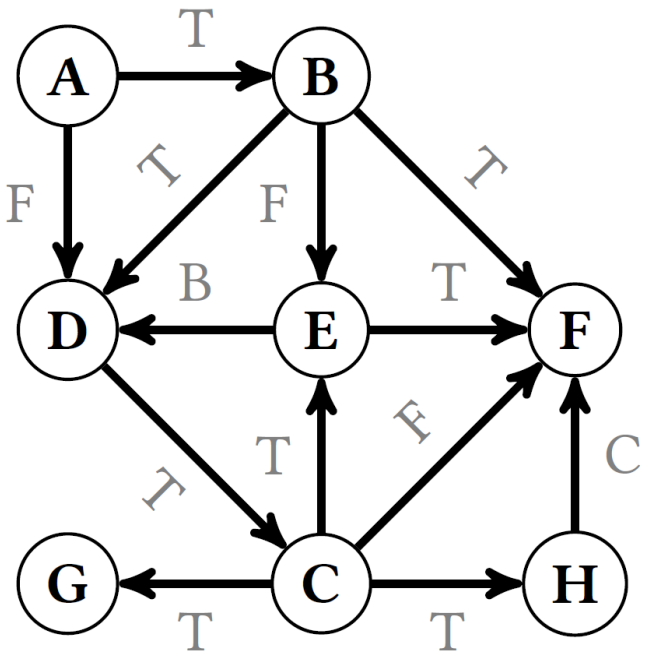
A	1	16
B	2	15
C	4	13
D	3	14
E	5	8
F	6	7
G	9	10
H	11	12

A



b) For each vertex, give its *pre*- and *post*-number.

Solution: A (1,16), B (2,15), D (3,14), C (4,13), E (5,8), F (6,7), G (9,10), H(11,12)

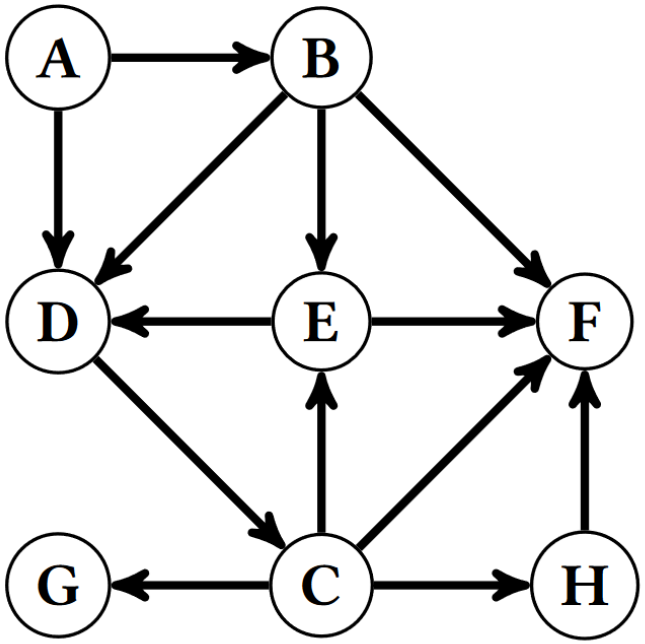


b) For each vertex, give its *pre-* and *post*-number.

Solution: A (1,16), B (2,15), D (3,14), C (4,13), E (5,8), F (6,7), G (9,10), H(11,12)

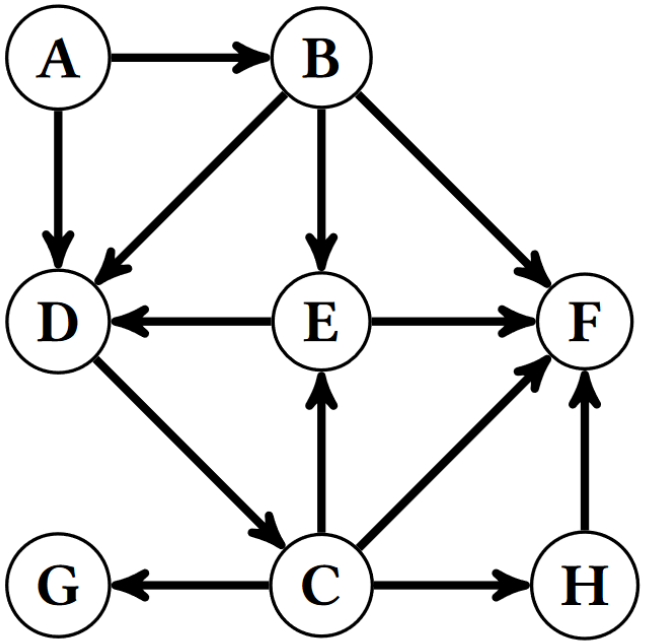
c) Give the vertex ordering that results from sorting the vertices by pre-number. Give the vertex ordering that results from sorting the vertices by post-number.

Solution: Pre-ordering: A, B, D, C, E, F, G, H. Post-ordering: F, E, G, H, C, D, B, A.



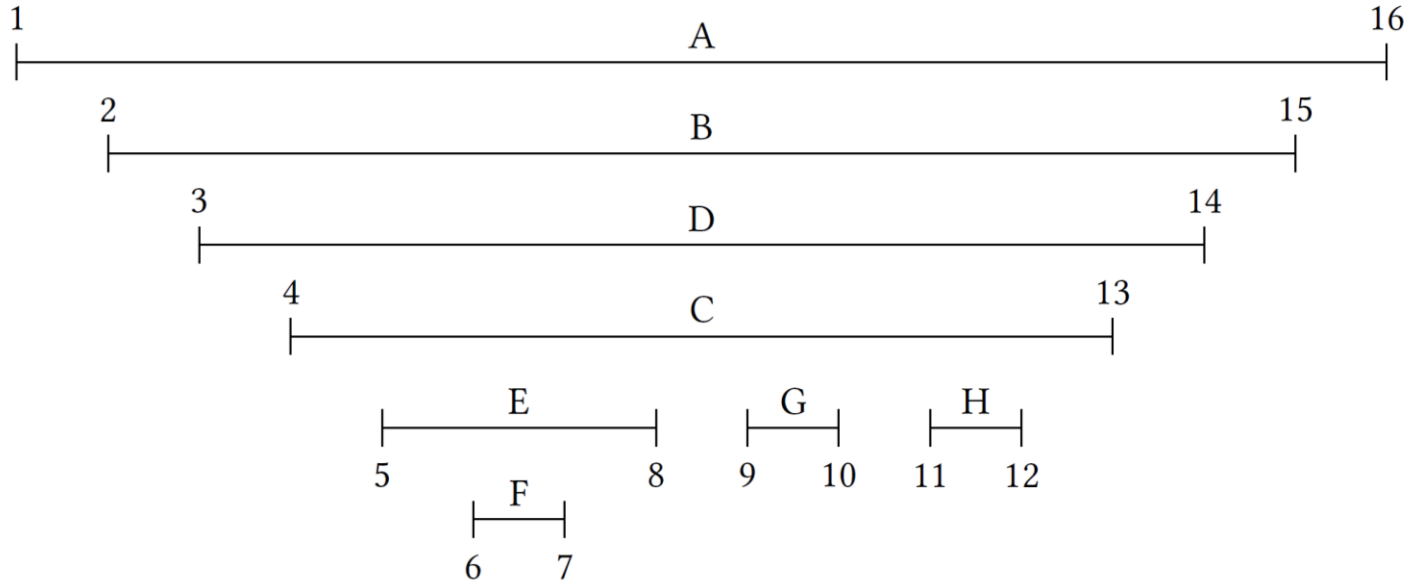
e) Does the above graph have a topological ordering? How can we use the above execution of depth-first search to find a directed cycle?

Solution: The decreasing order of the post-numbers gives a topological ordering, whenever the graph is acyclic. This is the case if and only if there are no back edges. If there is a back edge, then together with the tree edges between its end points it forms a directed cycle. In our graph, the only back edge is E $\rightarrow$ D, and the tree edges from D to E are D $\rightarrow$ C and C $\rightarrow$ E. Together they form the directed cycle (D $\rightarrow$ C $\rightarrow$ E $\rightarrow$ D).

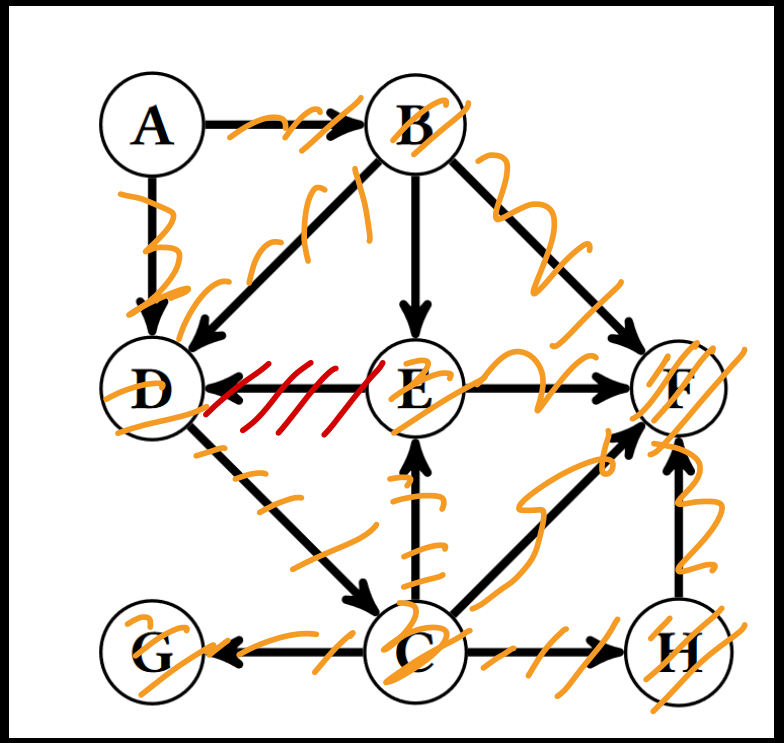


f) Draw a scale from 1 to 16, and mark for every vertex v the interval I_v from pre-number to post-number of v. What does it mean if $I_u \subset I_v$ for two different vertices u and v?

Solution:



If $I_u \subset I_v$ for two different vertices u and v, then u is visited during the call of DFS-Visit(v).



A ; B ; D ; C ; E ; G ; H ; F

g) Consider the graph above with the edge from E to D removed. How does the execution of depth-first search change? Which topological sorting does the depth-first search give? If you sort the vertices by *pre-number*, does this give a topological sorting?

Solution: The execution of depth-first search doesn't change. The topological sorting is: A, B, D, C, H, G, E, F. The pre-ordering is A, B, D, C, E, F, G, H.; it does not give a topological ordering, since there is an edge (H, F) in the graph.

EXERCISE 10.5

In order to encourage the use of train for long-distance traveling, the Swiss government has decided to make all the m highways between the n major cities of Switzerland one-way only. In other words, for any two of these major cities C_1 and C_2, if there is a highway connecting them it is either from C_1 to C_2 or from C_2 to C_1, but not both. The government claims that it is however still possible to drive from any major city to any other major city using highways only, despite these one-way restrictions.

a) Model the problem as a graph problem. Describe the set of vertices V and the set of edges E in words. Reformulate the problem description as a graph problem on the resulting graph.

b) Describe an algorithm that verifies the correctness of the claim in time $O(n + m)$.

Hint: *You can make use of an algorithm from the lecture. However, you might need to modify the graph described in part (a) and run the algorithm on some modified graph.*

$G = (V, E)$ $\quad$ $G' = (V, E')$

$(v,u) \in E' \Leftrightarrow \exists$ eine Strasse von u nach v

V sind die Städte

$(u,v) \in E \Leftrightarrow \exists$ eine Strasse von u nach v

Naive: n DFS (eine von jeden Knoten) kontrolliere, dass von Stadt u, alle andere Städte erreichbar sind.

Bessere Idee: kontrolliere, dass

① Von "Zürich" alle andere Städte erreichbar sind

② Von jede Stadt, "Zürich" ist erreichbar

Wenn ① und ② gelten, dann $\exists$ Pfad für jede (A,B) $(A,B \in V)$

$\Rightarrow$ Wir gehen von A nach Zürich und dann von Zürich nach B

① DFS von Zürich in G

① gilt $\Leftrightarrow$ alle Städte besucht werden

② DFS von Zürich in G'

② gilt $\Leftrightarrow$ alle Städte besucht werden

SHORTEST PATHS: A RECAP

Fact Es gilt das "Optimal Subproblem Pintip"

$(v_1 \dots v_\alpha, v_\beta)$ ist ein Shortest Path, dann auch $(v_1 \dots v_\alpha)$ ist ein Shortest Path

$$d[s,t] = \min_{\substack{u \in V \\ (u,t) \in E}} \left[d[s,u] + c(u,t) \right] \quad (*)$$

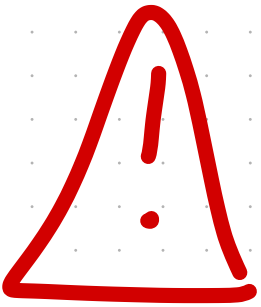
$\Rightarrow$ DP ☺

☹ Reihenfolge ist nicht immer klar

Fälle, wo DP funktioniert: DAG [Löse rekurrent (*) in topologische Sortierung]

Nich-negative Gewichte $\Rightarrow$ Dijkstra

Situation	Algorithm	Runtime
One-to-~~one~~ all, all edges with the same weight	Modified BFS	$\mathcal{O}(\|V\| + \|E\|)$
One-to-~~one~~ all, directed acyclic graph with arbitrary weights	DP and TopoSort	$\mathcal{O}(\|V\| + \|E\|)$
One-to-~~one~~ all, only with non-negative weights	Dijkstra	$\mathcal{O}(\|V\| \log \|V\| + \|E\|)$
One-to-~~one~~ all, also with negative weights	Bellman-Ford	$\mathcal{O}(\|V\|\|E\|)$
~~One~~ All-to-all	Floyd-Warshall	$\mathcal{O}(\|V\|^3)$
~~One~~ All-to-all	Johnson	$\mathcal{O}(\|V\|\|E\| + \|V\|^2 \log \|V\|)$



One-to-one existiert nicht wegen Optimal Subproblem Prinzip

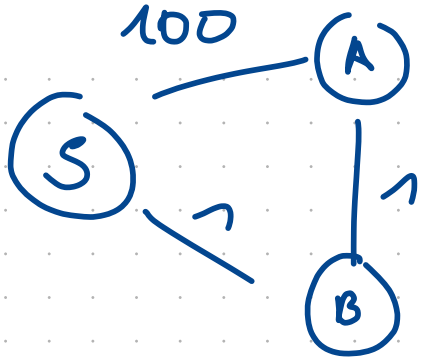
1

Falls
- keine Gewichte ($c(u,v) = 1 \; \forall (u,v) \in E$
- uniforme Gewichte

Algorithm 24: SP-BFS(G, s)

```
d[s] ← 0;
d[v] ← ∞ for v ∈ V \ {s};
Q ← ∅;
ENQUEUE(s,Q);
while Q ≠ ∅ do
    u ← DEQUEUE(Q);
    for each (u,v) ∈ E do
    if d[v] = ∞ then
        d[v] ← d[u] + c(u,v);
        ENQUEUE(v,Q);
```

$O(n+m)$



d	S	A	B
	0	100	1

Algorithm 25: SP-DAG(G, s)

$d[s] \leftarrow 0$;
$d[v] \leftarrow \infty$ for $v \in V \setminus \{s\}$;
for $v \in V$ *in topological order* **do**
  for all $(v, w) \in E$ **if** $d[w] > d[v] + c(v, w)$ **then**
    $d[w] = d[v] + c(v, w)$;

$O(n+m)$

(3) S wo wir die kürzeste Distanz kennen

In jede Iteration, wir hinzufügen ein Knoten zu S,

bis $S = V$

$d[s] = 0 \qquad d[v] = \infty \quad \forall v \neq s$

PRIORITY-QUEUE P

P.ENQUEUE$(s, 0)$ $\qquad S = \emptyset$

while $S \neq V$

$u \leftarrow$ EXTRACT-MIN(P) Take element in V/S with minimal $d[s]$

$S \leftarrow S \cup \{u\}$

for all $(u, v) \in E$

$d[v] \leftarrow \min(d[v], d[u] + c(u, v))$

and change priorities in P accordingly

Algorithm 26: Dijkstra(G, s)

```
for each v ∈ V \ {s} do d[v] = ∞;
d[s] = 0;
V \ S ← V;
while V \ S ≠ ∅ do
    choose u ∈ V \ S with minimal d[u];
    V \ S − {u};
    for (u, v) ∈ E do
        if d[v] > d[u] + c(u, v) then
            d[v] = d[u] + c(u, v);
```

$\mathcal{O}(|V| \cdot \text{Add} + |V| \cdot \text{Extract-Min} + |E| \cdot \text{Decrease-Key})$

- **Heaps:** $\mathcal{O}((|V| + |E|) \log |V|)$
- **Fibonacci Heaps:** $\mathcal{O}(|V| \log |V| + |E|)$

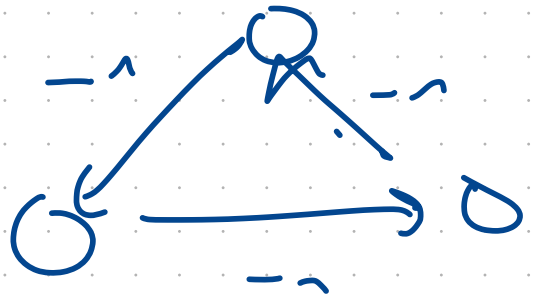
Add $O(\log |V|)$
Extract-min $O(\log |V|)$
Decrease-key $O(\log |V|)$

Decrease-key $O(1)$

4

Algorithm 27: Bellman-Ford(G, s)

```
for each v ∈ V \ {s} do d[v] = ∞;
d[s] = 0;
for i ← 1, 2, ..., |V| − 1 do
    for each (u, v) ∈ E do
        if d[v] > d[u] + c(u, v) then
            d[v] ← d[u] + c(u, v)
for each (u, v) ∈ E do
if d[v] > d[u] + c(u, v) then
    return Negative cycle;
```



$O(|V| \cdot |E|)$

LEETCODE PROBLEM

1971. Find if Path Exists in Graph

Easy 427 19 Add to List Share

There is a **bi-directional** graph with `n` vertices, where each vertex is labeled from `0` to `n - 1` (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [u_i, v_i]` denotes a bi-directional edge between vertex `u_i` and vertex `v_i`. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `start` to vertex `end`.

Given `edges` and the integers `n`, `start`, and `end`, return `true` *if there is a* ***valid path*** *from* `start` *to* `end`, *or* `false` *otherwise.*

Example 1:


```java
class Solution {
    public boolean algo(int n, ArrayList<Integer>[] A, int start, int end, boolean[] vis){
        vis[start] = true;
        boolean res = false;
        for(int i = 0; i < A[start].size(); i++){
            int v = A[start].get(i);
            if(v == end) return true;
            if(!vis[v]) res = res || algo(n, A, v, end, vis);
        }
        return res;
    }
    public boolean validPath(int n, int[][] edges, int start, int end) {
        if(start == end) return true;
        ArrayList<Integer>[] A = new ArrayList[n];
        for(int i = 0; i < n; i++) A[i] = new ArrayList<Integer>();
        for(int i = 0; i < edges.length; i++){
            int u = edges[i][0];
            int v = edges[i][1];
            A[u].add(v);
            A[v].add(u);
        }
        boolean[] vis = new boolean[n];
        for(int i = 0; i < n; i++) vis[i] = false;
        return algo(n, A, start, end, vis);
    }
}
```

Your previous code was restored from your local storage. Reset to default

TAKE HOME MESSAGE: DATA STRUCTURES MATTER

- Adjacency Matrix → Memory Error
- List of edges → Time Limit Error
- Adjacency List → Good enough