Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

5. Exercise for “Algorithmen, KI & Data Science 1”

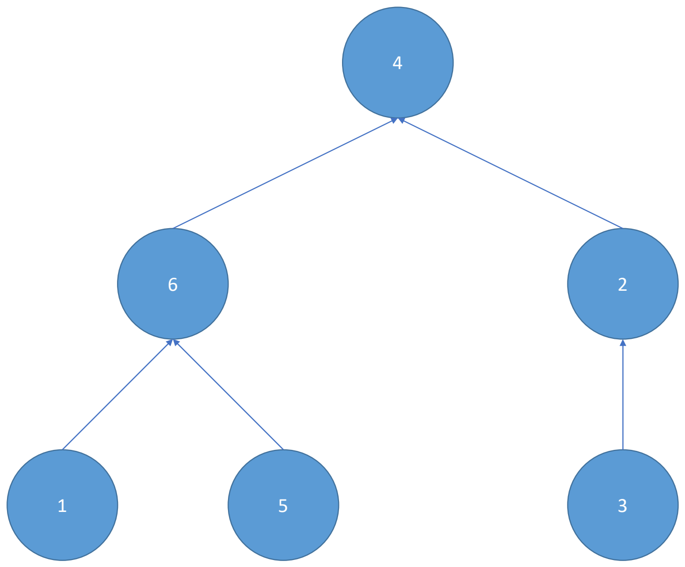
1 Breadth first search (BFS)

1. What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input?

   For each vertex u, we need to check for every $v \in V$ whether it is adjacent to u. This results in $O(V^2)$.

2. Does the BFS tree (see L10-slide 24) depend on the ordering of the adjacency lists? Explain by giving an example.

   Yes, it does. Changing the order in the adjacency list of node 4 (see L10-slide 24), results in the following BFS tree:



2 Depth First Search (DFS)

1. Give a counterexample to the conjecture that if a directed graph G contains a path from u to v, and if $u.vt < v.vt$ in a depth-first search of G, then v is a descendant of u in the depth-first tree/forest produced. Use the pseudocode of the recursive DFS shown in "L11 - Graph Algorithms".

   Explanation *DFS forest*: As the recursive variant of DFS (as shown in the lecture) might start from different roots, it possibly produces multiple DFS trees (i.e., a forest).

   Given a graph G with $V = w, u, v$ and $E = (w, u), (u, w), (w, v)$, let DFS start on w and u being in the adjacency list before v. Thus, $u.vt < v.vt$ and v is not a descendant of u.

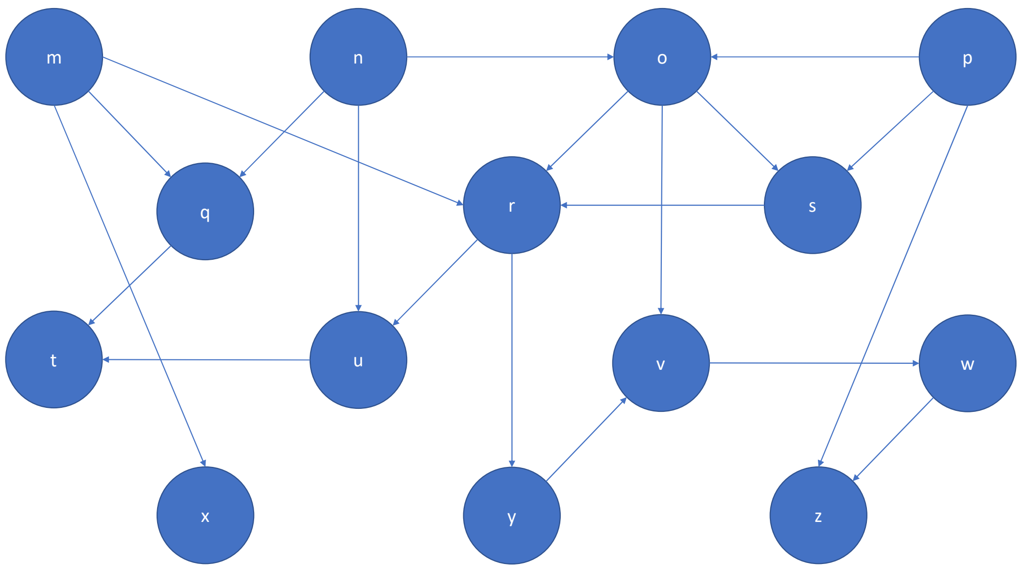
2. Give a counterexample to the conjecture that if a directed graph G contains a path from u to v, then any depth-first search must result in $v.vt \leq u.ft$. Use the pseudocode of the recursive DFS shown in "L11 - Graph Algorithms".

   Given a graph G with $V = w, u, v$ and $E = (w, u), (u, w), (w, v)$, let DFS start on w and u being in the adjacency list before v. Thus, $u.ft \leq v.vt$ because u finishes before v is visited.

3. Implement the recursive DFS as shown in L11 - slide 6. More precisely, implement the methods $dfs(G)$ and $dfs\_visit(G, u)$ in the provided $.ipynb$-file. Vertices and adjacency lists should be processed in lexicographical order.

3 Topological Sort

1. Show the ordering of vertices produced by topological-sort when it is run on the below graph. Include start and finish times for each vertex. Assume that the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically.



Vertex	vt	ft
m	1	20
q	2	5
t	3	4
r	6	17
u	7	8
y	9	16
v	10	15
w	11	14
z	12	13
x	18	19
n	21	26
o	22	25
s	23	24
p	27	28

List the vertices in descending order of ft: $p, n, o, s, m, x, r, y, v, w, z, u, q, t$

2. *Optional:* Give a linear-time algorithm (pseudocode) that, given a directed acyclic graph $G = (V, E)$ and two vertices $a, b \in V$, returns the number of simple paths from a to b in G. For example, the directed acyclic graph of exercise 3.1 contains exactly four simple paths from vertex p to vertex v: $<p, o, v>$, $<p, o, r, y, v>$, $<p, o, s, r, y, v>$, and $<p, s, r, y, v>$. Your algorithm needs only to count the simple paths, not list them.

Key idea is to run DFS for topological sorting until reaching node b. Then keeping track of the number of simple paths for each of the vertices on the way back to the source a. Coming back to the source a yields the total number of simple paths.

Following algorithm implements the idea. First, we initialize all vertices $v \in G.V$ with $v.cp = NIL$. The variable cp keeps track of the path count. We then run the algorithm $path\_count(G, a, b)$ to determine the number of simple paths from a to b.

Algorithm 1 $path\_count(G, a, b)$

```
if $a == b$ then
  return $1$
else if $a.cp \neq NIL$ then
  return $a.cp$
else
  $a.cp = 0$
  for $w \in G.Adj[a]$ do
    $a.cp = a.cp + path\_count(G, w, b)$
  end for
  return $a.cp$
end if
```

4 Strongly Connected Component

1. How can the number of strongly connected components of a graph change (increase, decrease or stay the same) if a new edge is added?

a) Increase: This is not possible. Adding a new edge can never remove an existing connection between two vertices.

b) Decrease: This is possible. $V = a, b, c$ and $E = (a, b), (b, c)$. This graph has three strongly connected components. Adding the edge (c, b) reduces the number of strongly connected components to two.

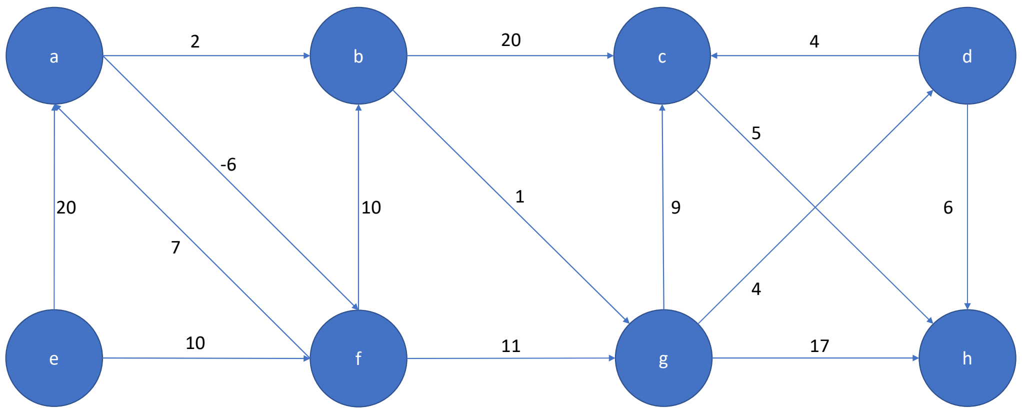
c) Stay the same: This is possible. Consider the same graph as for the decrease scenario and add edge (a, c). Adding this edge does not change anything.

2. Teaching Assistant Benedikt rewrites Kosaraju's algorithm for strongly connected components to use the original (instead of the transpose) graph in the second depth-first search and scan the vertices in order of increasing finish times. Give a counterexample to show that the algorithm is not correct.

Consider Graph G with $V = a, b, c$ and $E = (b, a), (b, c), (c, b)$. The strongly connected components are $\{a\}$ and $\{b, c\}$. However, if DFS starts on b, it could explore c before a. Hence, it holds that $c.ft < a.ft < b.ft$. Running the second DFS starting on c would reach all other vertices, resulting in a single strongly connected component. This is clearly not correct.

5 Bellman Ford

1. Consider the following graph:



Find the shortest path from vertex e to vertex h using Bellman-Ford algorithm based on the following edge order:

$\underset{2}{(a,b)}, \underset{-6}{(a,f)}, \underset{20}{(b,c)}, \underset{1}{(b,g)}, \underset{5}{(c,h)}, \underset{4}{(d,c)}, \underset{6}{(d,h)}, \underset{20}{(e,a)}, \underset{10}{(e,f)}, \underset{7}{(f,a)}, \underset{10}{(f,b)}, \underset{11}{(f,g)}, \underset{9}{(g,c)}, \underset{4}{(g,d)}, \underset{17}{(g,h)}$

To do so, complete the following table. Note that the not all columns may be needed.

	I		R_1		R_2		R_3		R_4		R_5		R_6	
	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$
a														
b														
c														
d														
e														
f														
g														
h														

	I		R_1		R_2		R_3		R_4		R_5		R_6	
	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$	$v.dist$	$v.prev$
a	∞		17	f										
b	∞		20	f	19	a								
c	∞		30	g	29	d	28	d						
d	∞		25	g	24	g								
e	0													
f	∞		10	e										
g	∞		21	f	20	b								
h	∞		38	g	31	d	30	d						

Shortest path from e to h:

$e \to f \to a \to b \to g \to d \to h$

Distance from e to h:

30

6 Dijkstra

1. Implement the Dijkstra algorithm as shown in L11. More precisely, implement the method $dijkstra(G, s)$ in the provided $.ipynb$-file.