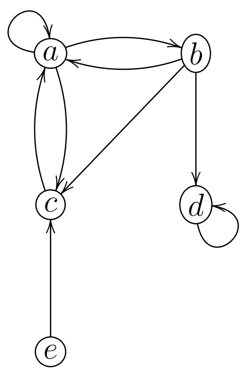
Solutions for Exercise Sheet 14

Handout: December 19th — Deadline: December 26th, 4pm

Question 14.1 (0.25 marks)

Perform a depth-first search on the following graph visiting nodes in alphabetical order. Assume that all adjacency lists are sorted alphabetically. Write down the timestamps and the π-value of each node.



Solution.

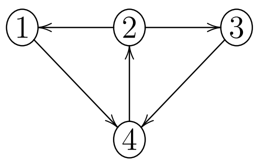
	d	f	π
a	1	8	NIL
b	2	7	a
c	3	4	b
d	5	6	b
e	9	10	NIL

Question 14.2 (0.5 marks))

Prove or refute the following claim: if some depth-first search on a directed graph yields precisely one back edge, then all depth-first searches on this graph yield precisely one back edge.

Solution.

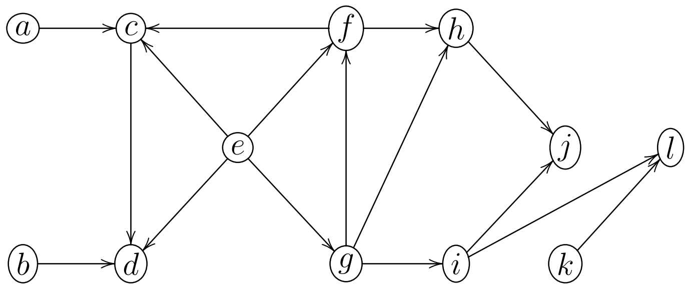
In the following graph, searching from 2 yields one backward edge; searching from 4 yields two.



This refutes the claim.

Question 14.3 (0.25 marks)

Run TOPOLOGICAL-SORT on the following directed acyclic graph. Assume that depth-first search visits nodes in alphabetical order and that adjacency lists are sorted alphabetically.



Solution.

Here's the output of depth-first search, processing nodes and adjacency lists in alphabetical order as stated in the instructions:

	d	f
a	1	6
b	7	8
c	2	5
d	3	4
e	9	22
f	10	15
g	16	21
h	11	14
i	17	20
j	12	13
k	23	24
l	18	19

This yields the list $(k, e, g, i, l, f, h, j, b, a, c, d)$.

Question 14.4 (0.5 marks)

Recall from the lecture that DFS can be used to check whether a directed graph $G = (V, E)$ is acyclic or not, and that DFS runs in time $\Theta(|V| + |E|)$.

Give an algorithm that checks whether or not an *undirected* graph $G = (V, E)$ is acyclic and that *runs in time only* $O(|V|)$.

Solution.

Run DFS and check whether you get a back edge (undirected graphs only have tree and back edges anyway). This is the case when you explore an edge (u, v) with v gray. Now G has no cycles at all if and only if there can be at most $|V|-1$ edges. Otherwise, just stop after exploring $|V|$ edges (a counter can be added for this).

Question 14.5 (1 mark)

Implement TOPOLOGICAL-SORT(G) for a given directed graph $G(V, E)$. The algorithm should return a topological sort if the graph is acyclic or that no topological sort exists if the graph contains a cycle. The input will be:

- first line: N M (the number of vertices and edges).
- M lines each containing a pair $v_i v_j$ meaning there is an edge $v_i \rightarrow v_j$.

You have to first build the adjacency list representing the graph with the required attributes (colour, .d, .f .π).

The algorithm should run in time $O(V + E)$.