



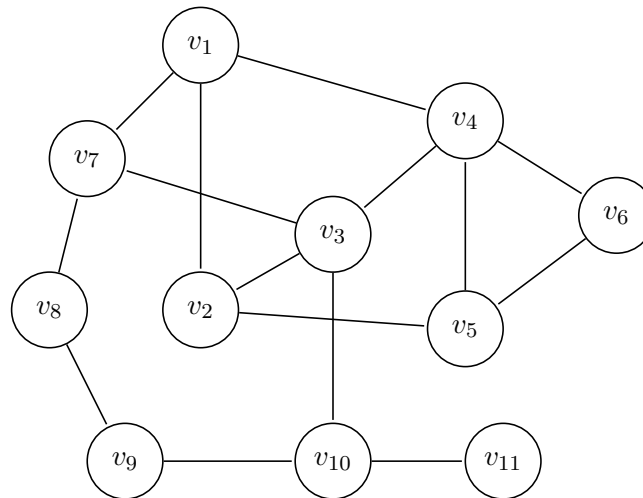
Algorithms and Datastructures

Sample Solution Exercise Sheet 8

Exercise 1: BFS

(5 Points)

Given the following undirected graph G :



- a) Provide G as an adjacency matrix. *(2 Points)*
- b) Provide G as an adjacency list. *(2 Points)*
- c) Perform a breadth-first search on G starting from node v_1 . Write the order in which the nodes are marked (i.e., colored gray) in the algorithm. To obtain a deterministic result, always add the node with the smaller index to the FIFO-queue first, that is, v_i before v_j if $i < j$. *(3 Points)*

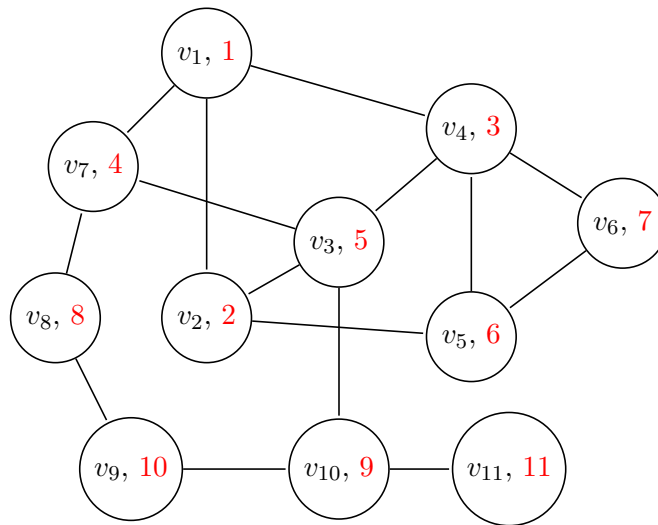
Sample Solution

a)

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	
0	1	0	1	0	0	1	0	0	0	0	v_1
1	0	1	0	1	0	0	0	0	0	0	v_2
0	1	0	1	0	0	1	0	0	1	0	v_3
1	0	1	0	1	1	0	0	0	0	0	v_4
0	1	0	1	0	1	0	0	0	0	0	v_5
0	0	0	1	1	0	0	0	0	0	0	v_6
1	0	1	0	0	0	0	1	0	0	0	v_7
0	0	0	0	0	0	1	0	1	0	0	v_8
0	0	0	0	0	0	0	1	0	1	0	v_9
0	0	1	0	0	0	0	0	1	0	1	v_{10}
0	0	0	0	0	0	0	0	0	1	0	v_{11}

- b)
- $v_1 : v_2, v_4, v_7$
 - $v_2 : v_1, v_3, v_5$
 - $v_3 : v_2, v_4, v_7, v_{10}$
 - $v_4 : v_1, v_3, v_5, v_6$
 - $v_5 : v_2, v_4, v_6$
 - $v_6 : v_4, v_5$
 - $v_7 : v_1, v_3, v_8$
 - $v_8 : v_7, v_9$
 - $v_9 : v_8, v_{10}$
 - $v_{10} : v_3, v_9, v_{11}$
 - $v_{11} : v_{10}$

c)



Exercise 2: DFS

(6 Points)

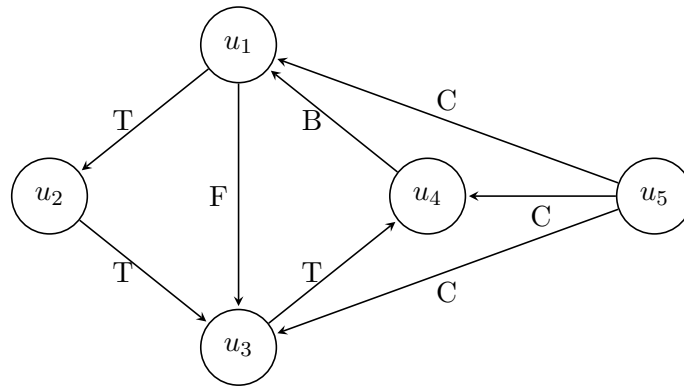
We define 2 timestamps for each node (as in Slide 29):

- $t_{v,1}$: Time when node v is colored gray by the DFS search
- $t_{v,2}$: Time when node v is colored black by the DFS search

Additionally, consider the following *directed* graph $G = (V, E)$ given with

- $V = \{u_1, u_2, u_3, u_4, u_5\}$
- $E = \{(u_1, u_2), (u_1, u_3), (u_2, u_3), (u_3, u_4), (u_4, u_1), (u_5, u_1), (u_5, u_3), (u_5, u_4)\}$

- a) Draw G . (2 Points)
- b) Write the processing interval $[t_{v,1}, t_{v,2}]$ for each node in G . Similar to part 1c), if multiple nodes could be visited next by the depth-first search, always choose the one with the smallest index (and thus we also start with u_1). (2 Points)
- c) For each edge, indicate whether it is a **Tree Edge**, **Backward Edge**, **Forward Edge**, or **Cross Edge**. (2 Points)



Sample Solution

ac) We label a Tree Edge by T , a Backward Edge by B (Backward Edge), a Forward Edge by F and a Cross Edge by C :

- b)
- $u_1 : [1, 8]$
 - $u_2 : [2, 7]$
 - $u_3 : [3, 6]$
 - $u_4 : [4, 5]$
 - $u_5 : [9, 10]$

Exercise 3: Cycle search

(9 Points)

- a) How many edges m can an undirected connected graph with n nodes have at most? Justify your answer. (2 Points)
- b) Show that every undirected connected graph which contains no cycle¹ has exactly $n - 1$ edges (where n is the number of nodes of the graph). (4 Points)
Hint: You can prove this statement, for example, by induction on $n \geq 1$.
- c) Given an undirected (not necessarily connected) graph $G = (V, E)$ with $n = |V|$. Provide an algorithm that decides in $\mathcal{O}(n)$ time whether G contains a cycle or not. Specify explicitly in which data structure G should be given. (3 Points)

Sample Solution

- a) A graph has the maximum number of edges when every node is connected to every other node. This means each node has a degree of $n - 1$. We now fix an order of the nodes $v_1, \dots, v_n$ and count the "not yet counted" edges for each. Thus, v_1 has exactly $n - 1$ edges, v_2 still has $n - 2$ edges (since the edge between v_1 and v_2 has already been counted), v_3 has $n - 3$ edges, and so on. Therefore, we have:

$$m \leq \sum_{i=1}^n (n - i) = \sum_{i=1}^{n-1} i = \frac{(n-1) \cdot n}{2}$$

Another approach would be to calculate how many 2-element subsets there are of an n -element set. There are exactly $\binom{n}{2} = \frac{n!}{2! \cdot (n-2)!} = \frac{n \cdot (n-1)}{2!} = \frac{(n-1) \cdot n}{2}$.

¹A cycle is a path $v_1, \dots, v_k \in V$ in a graph where there is also an edge between the start and the end node, i.e., $\{v_1, v_k\} \in E$.

- b) A connected graph without cycles has exactly $n - 1$ edges. Proof by induction.

Base case: For $n = 1$ the graph has no edges.

Induction hypothesis: Every such graph with $k \leq n - 1$ nodes has $k - 1$ edges.

Inductive step: We now show that the hypothesis also holds for a graph G with n nodes. Every graph G with n nodes can be composed of a node v which is connected to $l \geq 1$ disjoint subgraphs $G_1, \dots, G_l$ of G . Since G is acyclic, each of these subgraphs is also acyclic, and the only connection between two subgraphs is through the node v . Without loss of generality, let us say that G_i has exactly n_i nodes (for each of these subgraphs). Since $n_i \leq n - 1$ for all i , it follows from the induction hypothesis that G_i has exactly $n_i - 1$ edges. We can now calculate the number of edges m in G as follows:

$$m = \deg(v) + \sum_{i=1}^l (n_i - 1) = l + \sum_{i=1}^l n_i - \sum_{i=1}^l 1 = \sum_{i=1}^l n_i = n - 1$$

Here, $\deg(v) = l$, since v is connected to each of the l subgraphs, and $\sum_{i=1}^l n_i = n - 1$ because this is the sum over all nodes in G excluding v .

- c) This task could theoretically be solved using either depth-first or breadth-first search. For example we perform the breadth-first search "normally", but we also record for each node v the node p from which it was first reached. This node p is called the **parent** of v . When visiting v , if it has a marked neighbor w that is not its parent, then there is a cycle in the tree, and we return *Yes*. To see why this is true, notice that in this setup, there must be a node s from which there is a path to both w and v . The edge between w and v connects these paths into a cycle. If this never occurs, then there cannot be a cycle and we output No.

We assume that the graph is given as an adjacency list and that this datastructure stores the exact number of edges². At the beginning of our algorithm, if the number of edges is $> n - 1$, we immediately output Yes. By task 2,b) this is always correct, so assume the graph has at most $n - 1$ edges. But now both BFS and DFS run in time $O(n + m) \leq O(n + n - 1) = O(n)$, as desired.

²Note that if it didn't store the number of edges, we can easily add this feature while only adding an additive constant overhead to any operation.