Note: All exercises ending with a ☆ have previously appeared on an exam.

Part I
Graph Searching

1 Depth-first Search (p. 9)

Algorithm 1 DFS(G)

```
for all u ∈ V do
    color(u) = WHITE; π(u) = NIL
end for
time = 0;
for all u ∈ V do
    if color(u) = WHITE then
        DFS-VISIT(v)
    end if
end for
```

DFS-VISIT(u):

```
color(u) = GRAY; time = time + 1; d(u) = time
for all v ∈ Γ(u) do
    if color(v) = WHITE then
        π(v) = u; DFS-VISIT(v)
    end if
end for
color(u) = BLACK; time = time + 1; f(u) = time
```

Theorems and definitions

THEOREM 3.1 (*Parenthesis theorem*). Given $G = (V, E)$ and $u, v \in V$, then either of the following two cases holds after running the DFS algorithm:

1. The intervals $[d(u), f(u)]$ and $[d(v), f(v)]$ do not overlap and v is not a descendant of u.
2. The interval $[d(u), f(u)]$ completely engulfs the interval $[d(v), f(v)]$ and v is a descendant of u in the DFS tree/forest (or vice versa with the role of u and v switched).

Thus, v is a descendant of u if and only if $d(u) < d(v) < f(v) < f(u)$.

THEOREM 3.2 (*White path theorem*). Given that we perform a DFS search on $G = (V, E)$ with $u, v \in V$, v will be a descendant of u in the DFS tree if and only if at time $d(u)$ there exists a WHITE path from u to v in G.

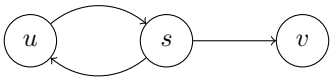
THEOREM 3.3. A DFS search of an undirected graph $G = (V, E)$ only encounters tree and back edges.

THEOREM 3.4. A directed graph G contains no cycles if a DFS traversal of the graph does not encounter a back edge.

1.1 DFS searches (p. 14)

1. Show that the following conjecture is false: If there is a path from u to v in a directed graph G and $d(u) < d(v)$ in a DFS search of G, then v is a descendant of u.

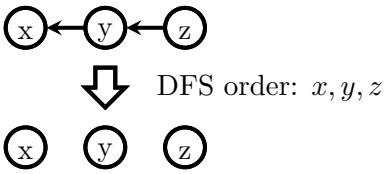
Solution: *Hint: theorem 3.2 (White path theorem) states that iff there is a white path from u to v at time $d(u)$, then v is a descendant of u.*



with DFS order: s, u, v

2. Give an example of a graph G such that a DFS search results in a forest where one of the trees contains only one vertex u that has both incoming and outgoing edges in G.

Solution: The node visiting order influences the DFS forest.



Note, that if we allow edges from a vertex to itself (we don't), we get the following trivial example:

3. Argue that a DFS on an undirected graph will result in a DFS forest where each tree is a connected component of G. Also, modify the DFS algorithm such that each vertex has a variable $cc(u)$ that indicates to which component it belongs.

Solution: A connected component (in an undirected graph $G = (V, E)$) is a subgraph (V', E') (meaning $V' \subseteq V$ and $E' \subseteq E$) such that each pair of nodes (u, v) in V' are connected by paths in E' and which is connected to no additional vertices in the supergraph G.

Each node u of one of the trees in the DFS forest is reachable with a path $s \to u$ from the source s of that tree (per DFS). This is an undirected graph, so the same path $s \to u$ can traversed in reverse order $u \to s$.

Modification of the DFS algorithm exists in creating a new ID for a new component before each call to DFS-Visit in the DFS routine. This ID must then be propagated during subsequent (recursive) calls to DFS-Visit.

DFSCC(G):

```
for all u ∈ V do
    color(u) = WHITE; π(u) = NIL
end for
time = 0; counter = 0
for all u ∈ V do
    if color(u) = WHITE then
        counter = counter + 1
        DFS-VISITCC(u, counter)
    end if
end for
```

DFS-VISITCC($u, counter$):

```
color(u) = GRAY; time = time + 1; d(u) = time
cc(u) = counter
for all v ∈ Γ(u) do
    if color(v) = WHITE then
        π(v) = u; DFS-VISITCC(v, counter)
    end if
end for
color(u) = BLACK; time = time + 1; f(u) = time
```

4. Give a $O(|V|)$ algorithm to check whether an undirected graph contains a cycle.

Solution: To detect whether an undirected graph $G = (V, E)$ contains a cycle, we run a modified version of DFS on G. Whenever we try to visit a non-WHITE node, we stop and return **true** (meaning we have a cycle). More specifically, encountering a GRAY neighbor means we already visited that node, so this means there is a cycle.

Encountering a BLACK node is impossible as G is undirected: theorem 3.3 states we can only encounter tree edges (no cycle) or back edges (cycle) in an undirected graph.

As this is an undirected graph, the traversal order does not matter. $O(|V|)$ means we can only visit each node a constant number of times. In the algorithm above, we visit at most $|V|-1$ edges before we can detect a cycle or each node in the graph is colored Gray or Black (no cycle).

In essence, this boils down to the detection of back edges (u, v) with $[d(u), f(u)] \subset [d(v), f(v)]$.

Note, that when no cycle exists in an undirected graph, we must have $|E| \leq |V| - 1$. We can speed up the algorithm by first checking whether $|E| \geq |V|$. If this is the case, then the graph contains a cycle. (The worst case complexity is still $O(|V|)$.)

5. Give an algorithm to test whether a selected vertex s is singly connected to all other vertices, that is, check whether there is at most one simple path from s to any other vertex $u \in V$. Use this algorithm to develop a $O(|V||E|)$ algorithm to check whether a graph is singly connected, meaning there is at most one simple path between any two vertices u and v.

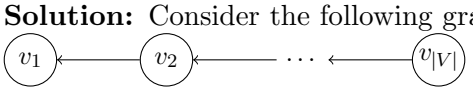
Solution: *Definition:* A simple path is a path containing no repeated vertices (each vertex on the path is visited only once). In other words: a simple path contains no cycles.
We assume this is a directed graph. We run a DFS on G starting at s until we've finished the DFS tree of s. During the run, we check whether there are forward edges (excepting tree edges) or cross edges (that is, we check whether we come across a black vertex). If any of those exist, then s is not singly connected: this is obvious in case of forward edges; for cross edges we note that here, a cross edge has to be an edge between two branches of the DFS tree of s.

We now run $|V|$ such algorithms every time starting at a different vertex. If we come across forward or cross edges, the graph is not singly connected. Otherwise, the graph is singly connected.
We potentially run this for each vertex in V, so the time complexity is $O(|V|(|V| + |E|)) \approx O(|V||E|)$.

G can still be singly connected if there are back edges. A back edge implicates only that there is an edge to an ancestor (hence there exists a cycle), which is still consistent with single connectedness.

6. Give an example of a graph G such that the DFS algorithm can produce k output trees for any $k = 1, \ldots, |V|$. ☆

Solution: Consider the following graph:



If the DFS discovers the nodes in the following order: $v_1, v_2, \ldots, v_{k-1}, v_{|V|}, v_{|V|-1}, \ldots, v_k$,

we get k different trees ($k-1$ trees consisting of a single vertex, and one tree consisting of the remaining vertices and edges between them).

7. An undirected connected graph $G = (V, E)$ is biconnected if G remains connected after removing any one of its vertices. Give an $O(|V|^2 + |V||E|)$ algorithm to check whether G is biconnected. Show that the removal of v, the root of a DFS tree, disconnects the graph if and only if v has 2 or more outgoing tree edges. ☆

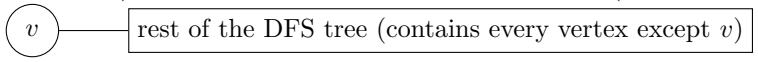
Solution: Part 1: We present three algorithms: 1) We can use the algorithm from exercise 3:

for $v \in V$ **do**
  Set $G' = (V \setminus \{v\}, E')$, where $E' = \{(s,t) \in E | s \neq v, t \neq v\}$.
  DFSCC(G')
  if *counter* > 1 **then**
    return false
  end if
end for
return true

Both the step where we set G' and the execution of DFSCC take $O(|V| + |E|)$ time. As we execute these two steps at most $|V|$ times, the time complexity of the algorithm is $O(|V|^2 + |V||E|)$.
2) Instead of setting G' above, we can set $color(v)$ = BLACK and run DFSCC(G) (without resetting $color(v)$ to WHITE).
3) We can leverage the property which we prove below for the following algorithm:

for $v \in V$ **do**
  Run DFS(G) starting from v
  if v has at least 2 edges in its DFS tree **then**
    return false
  end if
end for
return true

Part 2: $\Rightarrow$ Suppose v has one outgoing tree edge. Then removal of v (and its one tree edge) clearly doesn't disconnect the graph (see picture below).

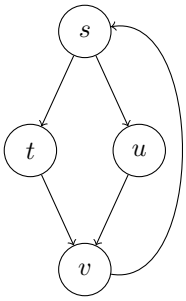


(If (v, w) is a tree edge in the picture above then every vertex is reachable from w without going through v).
$\Leftarrow$ Suppose now that v has at least two edges in its DFS tree, say $(v, u_1), \ldots, (v, u_k)$. Suppose u_1 is the first vertex that has been discovered after v. If there exists a path from u_1 to u_ℓ for some $\ell \in \{2, \ldots, k\}$ that doesn't go through v, then u_ℓ wouldn't be on a different branch, but it would be a descendant of u_1. This is due to the White path theorem (with $d(u_1) = d(v) + 1$). Therefore, the only path from u_1 to u_ℓ in G goes through v and thus the removal of v would make the graph disconnected.

In essence, this argument boils down to the fact that cross edges are impossible in undirected graphs.

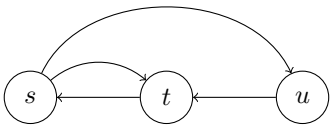
8. Let b be the number of back edges encountered during the execution of the DFS algorithm. Show that in general b is not equal to the number of cycles in the graph. ☆

Solution: Consider the following graph:



Clearly, it has two cycles (s, t, v, s and s, u, v, s). However, if we execute DFS starting at s on this graph, we will only encounter one back edge, namely (v, s). Note that the same example works if the graph is undirected, irregardless of whether we count an undirected cycle as one or two cycles (clockwise + counterclockwise).

The smallest possible example is



Clearly, it has two cycles (s, t, s and s, u, t, s). However, if we execute DFS starting at s on this graph, we will only encounter one back edge, namely (t, s).

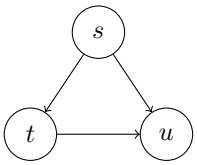
9. True or false? ☆

   (a) If one output of the DFS algorithm contains a back edge, all outputs contain a back edge.

   Solution: This is true for directed graphs by theorem 3.4: there exist cycles in the graph and thus every DFS search will encounter a back edge. In fact, the proof of theorem 3.4 can be repeated for undirected graphs. Therefore the statement is true.

   (b) If one output of the DFS algorithm contains a forward edge, all outputs contain a forward edge.

Solution: This is false. Consider for example the following graph:



If DFS discovers the vertices in the order s, t, u, then (s, u) is a forward edge. If, however, DFS discovers the vertices in the order u, t, s, then all edges are cross edges.

(c) If one output of the DFS algorithm contains a cross edge, all outputs contain a cross edge.

Solution: This is false, see for example the graph in exercise 6: the edges between different trees are cross edges, but we can also set $k = 1$ and get only one tree and no cross edges.