Datenstrukturen & Algorithmen Solution of Sheet 9 FS 12

Exercise 9.1 DFS & BFS.

a) A possible depth-first search order is A, B, C, G, H, F, D, E.
A possible breadth-first search order is A, B, F, C, H, G, D, E.

b) No: For every starting node in the graph, it is always possible to find a depth-first search order that is not a breadth-first search order.
The following orderings cannot be generated using breadth-first search.
  For A: A, B, C, ...
  For B: B, C, D, E, ...
  For C: C, D, E, ...
  For D: D, E, A, ...
  For E: E, A, B, ...
  For F: F, H, C, D, E, ...
  For G: G, H, C, ...
  For H: H, C, D, E, ...

(c) An example is the star (see below). If we begin the traversal in the middle, then it is easy to see that every breadth-first order is also a depth-first order, and vice versa. If the starting node is not the middle, then the only possible next choice is the middle, and we are back in the previous situation.

\begin{center}
\begin{tikzpicture}
  \node (a) at (0,0) [circle, fill=black] {};
  \node (b) at (1,1) [circle, fill=black] {};
  \node (c) at (1,-1) [circle, fill=black] {};
  \node (d) at (2,0) [circle, fill=black] {};
  \node (e) at (-1,1) [circle, fill=black] {};
  \node (f) at (-1,-1) [circle, fill=black] {};

  \draw (a) -- (b);
  \draw (a) -- (c);
  \draw (a) -- (d);
  \draw (a) -- (f);
  \draw (e) -- (a);
\end{tikzpicture}
\end{center}

d) Both breadth-first and depth-first searches have to visit every neighbor of a node at least once. Using adjacency lists, we need exactly as many steps as the number of neighbors. In this case, the running time is $O(n + m)$, i.e., $O(m)$ for connected graphs (as usual, we denote with $n$ the number of nodes, and with $m$ the number of edges).
Using an adjacency matrix, to find all the neighbors of a node we need to look through all the corresponding row/column. Since we need to do this for every node, the running time is $\Omega(n^2)$.
In a very dense graph with $m = \Theta(n^2)$, the asymptotic running time is the same. In „sparse” graphs, for example with $m = O(n)$, the use of an adjacency matrix results in a much worse asymptotic running time.

Exercise 9.2 Topological sorting using DFS.

a) Let $(u, v)$ be a backward edge found using depth-first search. This means that the depth-first search has already visited $v$ and it is now at $u$. In the DFS-tree, $v$ is an ancestor of $u$. The path from $v$ to $u$ together with the edge $(u, v)$ is a cycle in $G$.
Suppose now that $G$ contains a cycle. Let $v$ be the first node in the cycle that is visited by the depth-first search, and let $(u, v)$ be the (only) edge to $v$ in the cycle. When $u$ is visited by the depth-first search, the edge $(u, v)$ leads to an ancestor of $u$, and it is therefore a backward edge.
b) Let $T(v)$ be the time at which node $v$ is left, i.e., the number of steps of the depth-first search until $v$ is exited. We want to show that we obtain a topological sort if we arrange the nodes in descending order according to their exit times. This is the case if $T(u) > T(v)$ for every edge $(u, v)$.

From a), we know (since $G$ is acyclic), that the depth-first search does not encounter any backward edges. This means that every edge $(u, v)$ considered by the depth-first search leads from $u$ to an unvisited node or a node that has already been exited. The depth-first either already exited the node $v$, or visits and exits it before leaving $u$. Therefore, $T(u) > T(v)$ for every edge $(u, v)$.

c) We could use a depth-first search that remembers every exit time, and then sorts the nodes in descending order according to their exit times. It is much easier (and more efficient!) to add nodes to the solution as soon as they are exited during the depth-first search, without having to remember anything except the solution.

Pseudocode:

```
TOPOLOGICALSORT(G, v):
  input: directed Graph G, starting node v
  output: the nodes of G in topological order, or a warning that G has a cycle
  S ← ∅
  MARK v as ,,active```

```
  for all NEIGHBORS u of v:
    if u is ,,active```

```
      REPORT that G has a CYCLE AND ABORT the search
    if u is not ,,visited```

```
      S ← TOPOLOGICALSORT(G,u) ⊕ S
  MARK v as ,,visited```

```
  (INSTEAD of ,,active```

```
  return v ⊕ S```
```

d) The algorithm is essentially a depth-first search with the same running time as in 1d).

Note: In the implementation of the above pseudocode, you should avoid to copy partial solutions (i.e., the sorted subsequences) between recursive calls. Think about what the running time would be if you copy partial solutions around, and how to avoid to copy them.

Exercise 9.3 Union-find structures.

If we append smaller trees to larger trees, then for an union-find structure with height $h$ and $n$ nodes the invariant $n \geq 2^h$ applies (Lemma 6.3, Chapter 6.2.2).

The invariant states that a tree with height $h$ must contain at least $2^h$ nodes. We can construct a tree of height $h$ and exactly $2^h$ nodes as follows: a tree of height $h = 0$ consists of $n = 1 = 2^0$ nodes. To build a tree of height $h > 0$, we merge two trees of height $h - 1$, containing exactly $2^{h-1}$ nodes each. Since one of the two trees is appended to the other, the overall height increases by one. Therefore, the new tree has height $h$ and $2 \cdot 2^{h-1} = 2^h$ nodes. The number of the required UNION-operations is given by the following recursive equation:

\[
\begin{align*}
  u(0) &= 0 \\
  u(h) &= 2 \cdot u(h-1) + 1.
\end{align*}
\]

Thus, the overall number of UNION-operations is $u(h) = \sum_{i=1}^{h} 2^{i-1} = 2^h - 1$.

It is not possible to build a tree with $2^h$ nodes using less operations, since for each additional node in the tree we need at least one UNION-operation. Since a tree of height $h$ contains at least $2^h$ nodes, it is impossible to use less than $2^h - 1$ operations to obtain a tree with height $h$. 

2