# McGill University COMP251: Assignment 4

Worth 10%. Due November 12 at the beginning of lecture (10am sharp!)

**Note** For this assignment, you can simply say "insert (an element) $x$ into (a linked list) $L$" without having to give the details of how it is implemented.

**Question 1** For both parts below, draw the graph with direction of the edges clearly marked, and write down for each vertex the adjacency list of its neighbors. If your graph has $n$ vertices, then they are numbered 1 to $n$, and this is the order in which the DFS algorithm loops through the vertices in the main for-loop. *Note that we are working only with simple graphs, i.e., there are no self-loop, nor multiple edges that connect two vertices.*

(a) Give a counter-example to the conjecture that if a directed graph $G$ contains a path from $u$ to $v$, and if $s[u] < s[v]$ in a DFS of $G$, then $v$ is a descendant of $u$ in the DFS forest.

(b) Give an example of a vertex $u$ in a directed graph $G$ such that $u$ has both incoming and outgoing edges, and $u$ belongs to the DFS tree that contains only $u$.

**Question 2** Here is an idea for topological sort, different from the algorithm given in lecture. Recall that given a directed graph $G$ we need to sort the vertices of $G$ in to a topological order. The idea is to repeatedly find a vertex with outdegree 0, insert it into a linked list $L$, and remove it and all of its incoming edges. Note that the removal of these edges causes update in the outdegrees of some other vertices. If there are vertices in $G$ left but we cannot find a vertex of (updated) outdegree 0, then we report that $G$ is not acyclic (and hence it is impossible to produce a topological ordering of its vertices). Otherwise, at the end the linked list $L$ contains the vertices of $G$ in a topological order.

You are asked to give an algorithm using this idea, and prove that it works correctly. The graph is given using adjacency list representation. Thus the input to your algorithm is a pair $(n, Adj)$ where $n$ specifies the vertices of the graph (i.e., $1, 2, 3, \ldots, n$), and $Adj$ is the array containing the adjacency lists: $Adj[v]$ is the head of the linked list of neighbors of node $v$, for $1 \leq v \leq n$.

(a) Give an algorithm that implements the idea given above. It must output "Containing cycle" if the given graph is not acyclic, and output the list $L$ otherwise. The algorithm must run in time $\mathcal{O}(|V| + |E|)$.

(b) Prove the correctness of your algorithm. This should consists of two parts:

(b1) If the algorithm reports "Containing cycle" then shows that the graph indeed contains a cycle.

(b2) If the algorithm outputs a linked list $L$, then show that if there is an edge $(u, v)$ in the graph, then $u$ must appear before $v$ in $L$.

**Question 3** Given a directed graph $G$, the **component graph** associated with $G$, denoted by $G^{SCC}$, is a directed graph defined as follows. Suppose that $C_1, C_2, \ldots, C_k$ are all strongly connected components of $G$. The vertex set $V^{SCC}$ of $G^{SCC}$ is the set $\{C_1, C_2, \ldots, C_k\}$, and the edge set $E^{SCC}$ of $G^{SCC}$ is the set of all ordered pairs $(C_i, C_j)$ such that in $G$ there is an edge from a vertex in $C_i$ to a vertex in $C_j$.

($G^{SCC}$ can be regarded as obtained from $G$ by contracting each strongly connected component to a single vertex, and removing duplicated edges if necessary.)

Note that $G^{SCC}$ is acyclic.

Give an algorithm that computes $G^{SCC}$ from $G$ in time $\mathcal{O}(|V|+|E|)$. $G$ is given using adjacency list representation, that is, the input is a pair $(n, Adj)$, where $n$ specifies the vertices of $G$ (the set of vertices of $G$ is $\{1, 2, \ldots, n\}$), and $Adj$ is the array of adjacency lists: $Adj[v]$ is the head of the linked list of neighbors of node $v$, for $1 \leq v \leq n$. The output of the algorithm is a pair $(k, B)$ where $k$ is the number of strongly connected components of $G$, and $B$ is an array of the adjacency lists of $G^{SCC}$: $B[i]$ is the head to the linked list of neighbors of $C_i$ in $G^{SCC}$. The running time of your algorithm must be $\mathcal{O}(|V| + |E|)$.

**Question 4** Suppose that we want to query the $k$-th smallest element in a binary search tree. One algorithm is to find the smallest element in the tree (using BST-Min), and then call the successor procedure (BST-Successor) $(k-1)$ times. This algorithm is not satisfactory because its running time depends on $k$: when $k$ is $\Omega(n)$ the running time is $\Omega(n)$.

Your task is to give an algorithm whose running time depends only on the height $h$ of the tree. In particular, it must run in time $\mathcal{O}(h)$. To do this you are allowed to modify the data structure by adding at most one field to the node object.

(a) Clearly describe the field you add (if you need it), and give your algorithm. Also verify that the running time of your algorithm is $\mathcal{O}(h)$.

(b) Give the algorithm $Insert(T, x)$ that inserts a new element $x$ into the tree $T$. The running time of your algorithm must be $\mathcal{O}(h)$.

(c) Given the algorithm $Delete(T, x)$ that deletes the element $x$ from tree $T$. The running time of your algorithm must be $\mathcal{O}(h)$.