Exercise Session 10

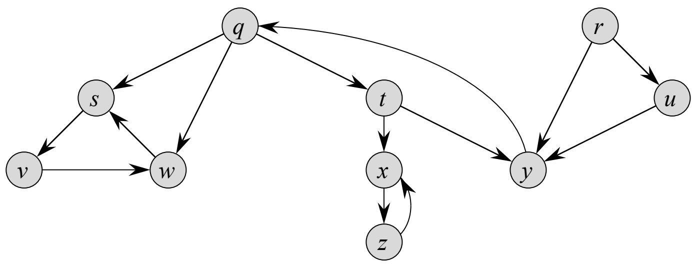
Exercise 1.
(CLRS 22.1-3) The transpose of a directed graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Describe efficient algorithms for computing G^T from G, for both adjacency-list and adjacency-matrix representations of G. Analyse the running time of your algorithms.

For an adjacency list we must create a nested for loop. The outer loop should iterate over the vertices that we have, the inner loop should iterate over the edges from each vertex. We should then swap $E = [u, v]$ to $E^T[v, u]$. The time complexity must be $\Theta(|V + E|)$

For an adjacency matrix A simply create a new 2d array B and create a nested for loop such that our $B[i, j] = A[j, i]$. The time complexity must be $\Theta(n^2)$

Exercise 2.
The diameter of a directed graph $G = (V, E)$ is defined as $\max\{\delta(v, u) : u, v \in V \text{ such that } v \rightsquigarrow u\}$, that is, the largest of all shortest-path distances between any two reachable nodes in G. Describe an algorithm that computes the diameter of a directed graph, and analyse its running time.

Exercise 3.
Consider the graph G depicted below.

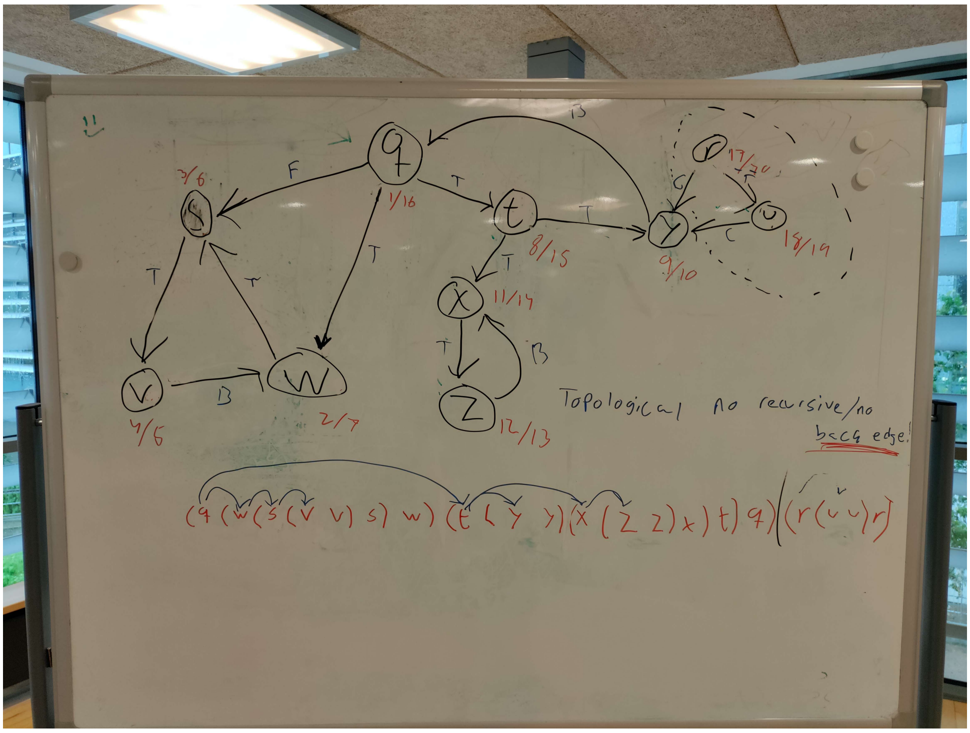


(a) Write the intervals for the discovery time and finishing time of each vertex in the graph obtained by performing a depth-first search visit of G.

(b) Write the corresponding "parenthesization" of the vertices in the sense of Theorem 22.7 in CLRS

(c) Assign with each edge a label T (tree edge), B (back edge), F (forward edge), C (cross edge) corresponding to the classification of edges induced by the DFS visit performed before.

(d) If G admits a topological sorting, then show the result of TOPOLOGICAL-SORT(G).



It does not admit to a topological sorting as the back edges does not follow the topological sorting as the graph creates multiple cycles with the back edges.

★ **Exercise 4.**
(CLRS 22.4-5) Another way to perform topological sorting on a directed acyclic graph $G = (V, E)$ is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph. Explain how to implement this idea so that it runs in time $O(|V| + |E|)$. What happens to this algorithm if G has cycles?

Exercise 5.
Assume G is a directed acyclic graph. Give an efficient algorithm to compute the graph of strongly connected components of G, and analyse the running time of your algorithm.

STRONGLY-CONNECTED-COMPONENTS(G)
1) call DFS(G) to compute finishing times $u.f$ for each vertex u
2) compute G^T
3) call $DFS(G^T)$, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4) output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

The overall time complexity of the algorithm is: $\theta|V| + |E|$. This is because steps 1, 2 and 3 all take $\theta|V| + |E|$.

Exercise 6.
(CLRS 22.5-1) How can the number of strongly connected components of a graph change if a new edge, say (u, v), is added? Discuss the following cases:

- if both u and v belong to the same component; Nothing happens and the no edge is connected to a new SCC
- if u and v belong to two distinct components.

  Case 1: There is already an edge from Component A to B but no edge from B to A. An edge from A to B is added to the Graph. This doesn't change the relationship of the SCC as A and B will still not have connected vertices such that we can go from A to B and from B to A Case

2: There is already an edge from Component A to B but no edge from B to A. An edge from B to A is added to the Graph. This changes the relationship of the SCC as A and B will now be connected as a single component. As there are edges such that you can travel from A to B and from B to A.