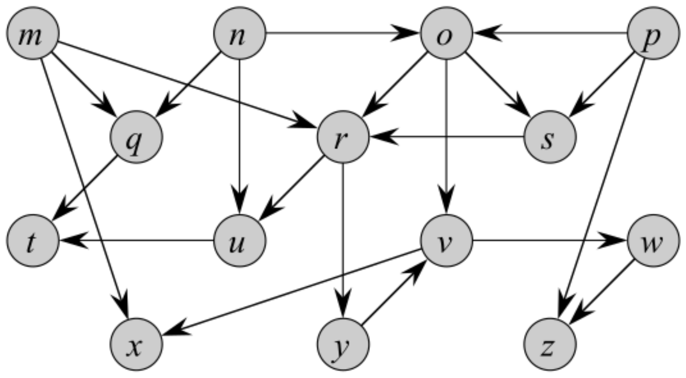
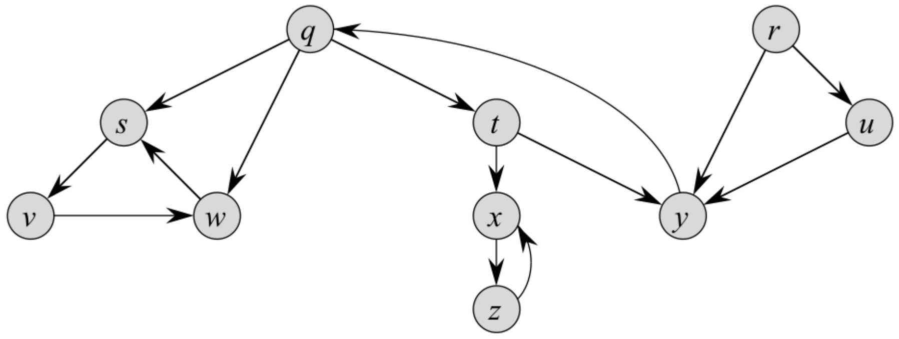
IITM-CS2800-2025 : Design and Analysis of Algorithms

Practice Problem Sheet #6 Released On : Mar 20, 1 PM

1. Show the ordering of vertices produced by Topological-Sort algorithm that we discussed in class when it is run on the DAG given below. (Process vertices in alphabetical order).



2. Show the output of the algorithm to find the strongly connected components using DFS that we discussed in the lecture for the following graph. Show the finish times after the first DFS. Show the transposed graph. Draw the SCC graph of G.



3. We saw a proof of existence of topological sorting in any directed acyclic graph via an argument that repeatedly finds a source node in the remaining graph, adds the source node to the topological order, and removes the node and the edges incident on it, and repeats this process. We argued that since G is a DAG, the subgraphs obtained in the intermediate stages are also DAGs, and consequently we will alwyas be able to find a source to remove (until all vertices in the graph G are exhausted). This algorithm can also be implemented to find a topological sorting of vertices of a directed acyclic graph G. Write down an implementation that runs in time $O(m + n)$. What happens to this algorithm if the graph G has cycles?

4. In this question, we analyse the implementation details of Dijkstra's algorithm. We recall the algorithm first:

```
Dijkstra's Algorithm (G(V,E),w)
    Let S be the set of explored nodes
    For each u in S, we store a distance d(u)
    Initially S = {s} and d(s) = 0
    While S not equal to V
        Select a node v \in S with at least one edge from S for which
        d'(v) = d(u) + w(e) is minimized over all u in S and e = (u,v) in E.
        Add v to S and define d(v) = d'(v)
    EndWhile
```

   (a) Notice that we need to compute $d'(v)$ by running over all edges. By a naive implementation using arrays, show that the algorithm runs in time $O(mn)$.

   (b) We improve the implementation using priority queues (which you have seen in PDS class). Recall that a priority queue maintains elements with a key value and can efficiently insert elements, delete elements, change an element's key, and extract the element with the minimum key. We will use it to maintain the data in such a way that the the v which achieves the minimum can be quickly found out (without spending $O(m)$ time in line 3 inside the while loop). After the initial setup of the priority queue, we will need the third and fourth of the above operations: `ChangeKey` and `ExtractMin`.

   Use the above to argue that Dijkstra's Algorithm can be implemented on a graph with n nodes and m edges to run in $O(m)$ time, plus the time for n number of `ExtractMin` and m number of `ChangeKey` operations.

   Note: Independent of this question, you may have seen in PDS class, an implementation of priority queue with $O(\log n)$ cost for `ChangeKey` *and $O(1)$ cost for* `ExtractMin`*. Using that, we can conclude that the algorithm runs in time $O(m \log n)$.*

6. Show that Dijkstra's algorithm does not work if the directed graph has weights which are negative. What about weight 0 edges?

7. Let $G(V, E)$ be a directed graph in which each vertex $u \in V$ is labeled with a unique integer $L(U)$ from the set $\{1, 2, \ldots, |V|\}$. For each vertex $u \in V$, let $R(u) = \{v \in V \mid u \rightsquigarrow v\}$ be the set of vertices that are reachable from u. Define $min(u)$ to be the vertex in $R(u)$ whose label is minimum, i.e., $min(u)$ is the vertex v such that $L(v) = \min\{L(w) \mid w \in R(u)\}$. Give an $O(m + n)$ algorithm that computes $min(u)$ for all vertices $u \in V$. *(Hint: Transpose and DFS)*

8. Modify the pseudocode for depth-first search so that it prints out every edge in the directed graph G, together with its type.

9. A directed graph $G(V, E)$ is *singly connected* if $u \rightsquigarrow v$ implies that G contains at most one simple path from u to v, for all vertices $u, v \in V$. Design an $O(mn)$ time algorithm for testing

whether or not a directed graph is singly connected. (*Hint: Do a topological sort. For each vertex, compute the list of ancestors with in-degree 0. Can you write a condition for G being singly connected in terms of these lists?*).

10. Prove or disprove: If a directed graph G contains cycles, then TOPOLOGICAL-SORT that we did in class produces a vertex ordering that minimizes the number of "bad" edges that are inconsistent with the ordering produced.

11. A depth-first forest classifies the edges of a graph into tree, back, forward, and cross edges. A breadth-first tree can also be used to classify the edges reachable from the source of the search into the same four categories.

    (a) Prove that in a breadth-first search of an undirected graph, the following properties hold:

    - There are no back edges and no forward edges.
    - For each tree edge (u, v), we have that $v.d = u.d + 1$.
    - For each cross edge (u, v), we have $v.d = u.d$ or $v.d = u.d + 1$.

    (b) Prove that in a breadth-first search of a directed graph, the following properties hold:

    - There are no forward edges.
    - For each tree edge (u, v), we have $v.d = u.d + 1$.
    - For each cross edge (u, v), we have $v.d \leq u.d + 1$.
    - For each black edge (u, v), we have $0 \leq v.d \leq u.d$.