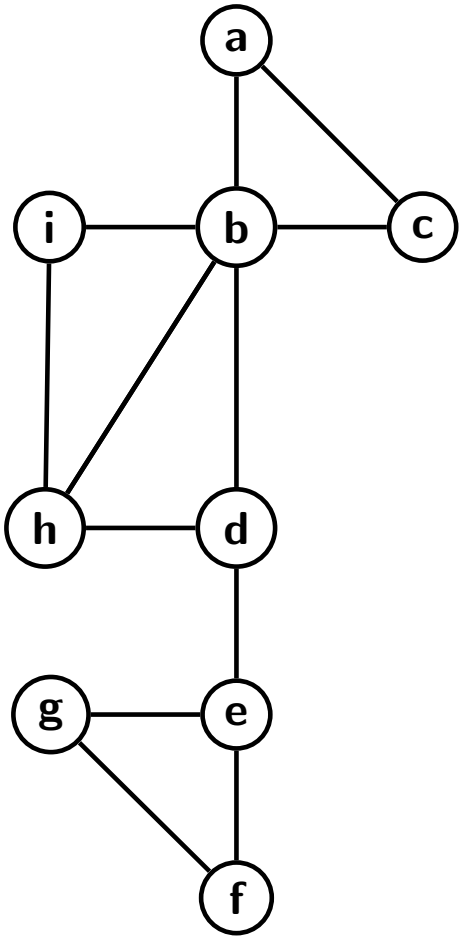
Classification of Edges by BFS

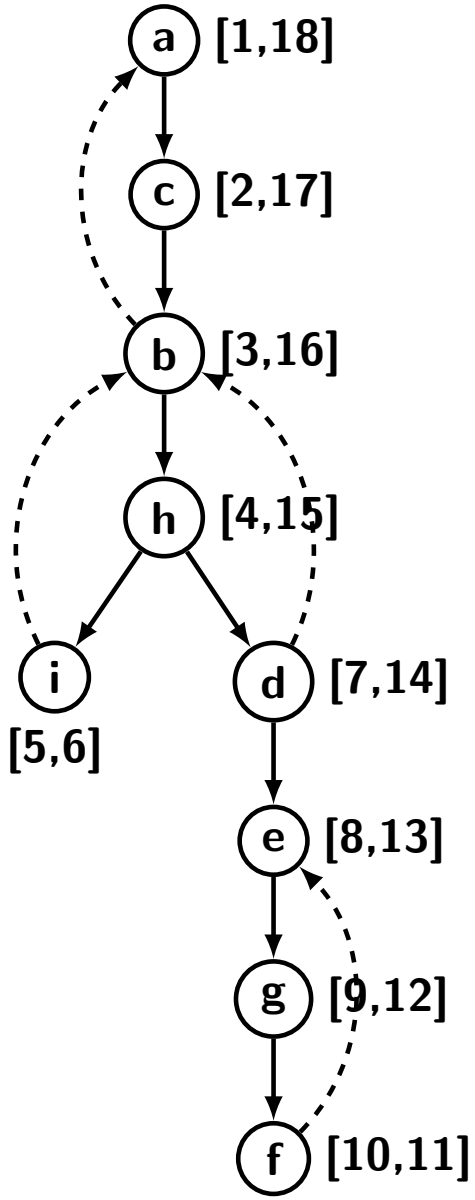
- There can be no back edges or forward edges in BFS of undirected graphs.
- For each tree edge (u, v), dist[v] = dist[u] + 1
- For each cross edge (u, v), dist[u] = dist[v] or dist[v] = dist[u] + 1

Parenthesis Theory

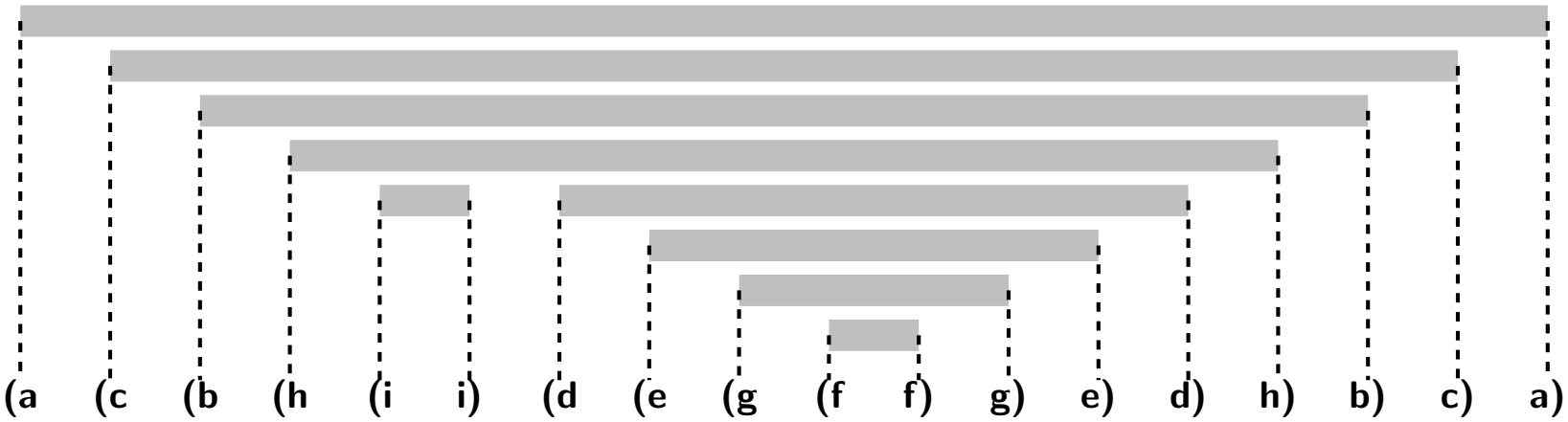
- Consider DFS numbers and reverse DFS numbers generated during DFS of a graph.
- Let u and v be any two vertices in the graph and let d[v] and f[u] respectively denote discovery time and finishing time of DFS then exactly one of the following three conditions hold.
  - If the intervals [d[u],f[u]] and [d[v],f[v]] are disjoint neither u nor v is a descendant of the other in DFS tree/forest.
  - If [d[v],f[v]] completely enclosed within [d[u],f[u]] then v is a descendant of u.
  - If [d[u],f[u]] completely enclosed within [d[v],f[v]] then u is a descendant of v.

Parenthesis Theory





Parenthesis Theory



- Opening parenthesis corresponds to discovery time d.
- Closing parenthesis corresponds to finish time f.
- Resulting expression is a valid parenthetical matching string.

Connected Components

- How to obtain connected components?
  - Using DFS/BFS it is possible.
- Outline of the algorithm is as follows:
  - Initialize a connected component number to 0.
  - Inside the second for-loop increment connected component number each time before calling DFS procedure.

Connected Components

```
index = 0;
count = 1; // initialize
for all (v ∈ V) {
    mark[v] = "unvisited";
}
for all (v ∈ V) {
    if (mark[v]=="unvisited"){
        DFS(G, v);
        increment(count); // Component number
    }
}
```

Depth First Search

```
DFS(G, v) {
    mark[v] = "visited";
    cID[v] = get(count); // Component ID
    dfn[v] = ++index; // DFS number
    for all (w ∈ ADJ_G(v)) {
        if (mark[w]=="unvisited") {
            parent[w] = v;
            DFS(G, w);
        }
    }
}
```

Topological Sorting

Definition

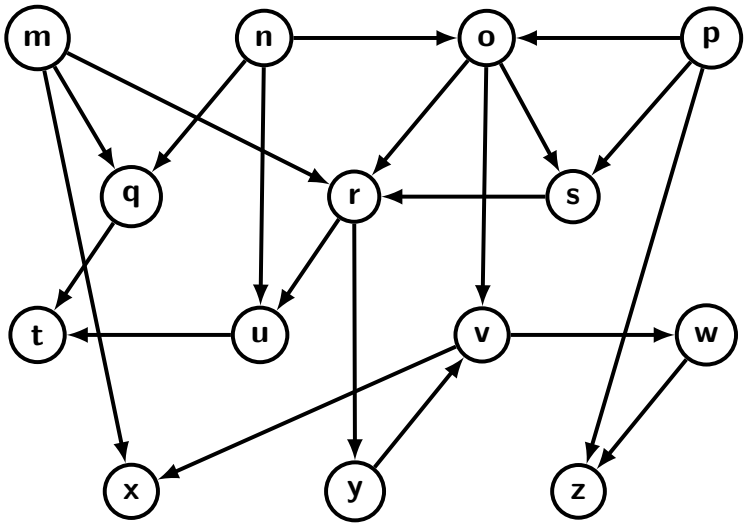
A linear total ordering of the vertices of directed graph, such that for each edge $u \rightarrow v$, u appears before v in the list.

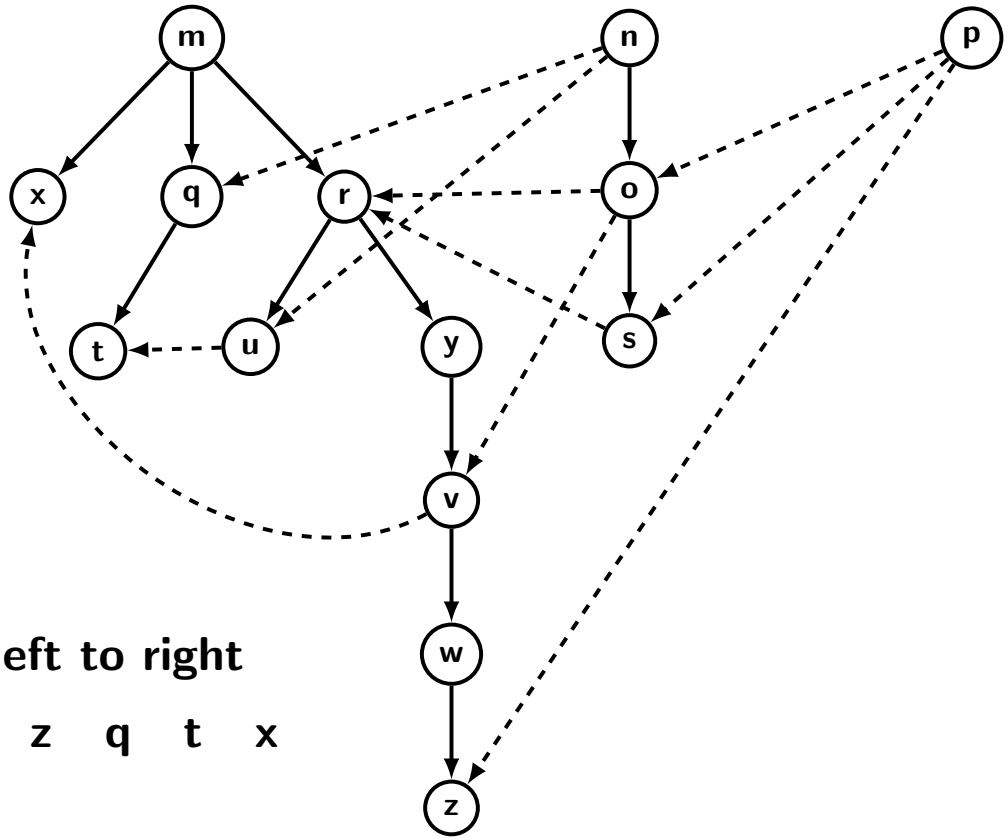
- Scheduling constraints between lectures.
- Pre-requisites of your B. Tech degree.
- Various stages or tasks related to completion of projects.

Topological Sorting

- Just call DFS to compute reverse DFS number.
- As numbering to a vertex get assigned insert it to the front of an initally empty linked list.
- Linked list gives the topological sorted sequence in decreasing order of finish time of the task.

Topological Sorting

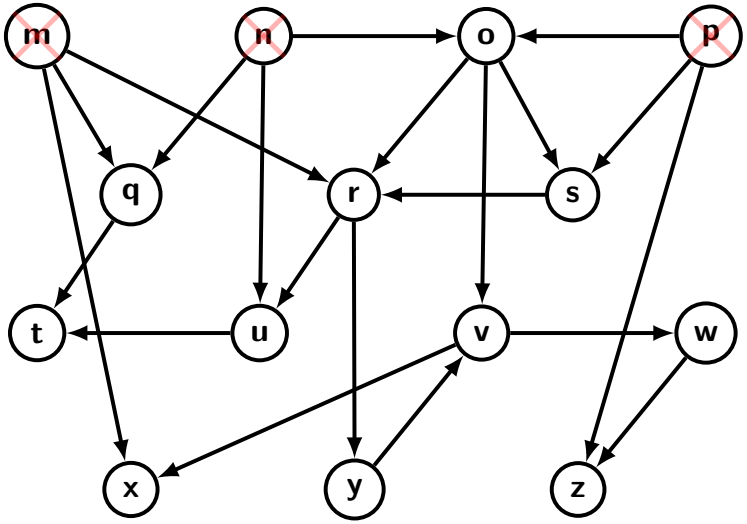




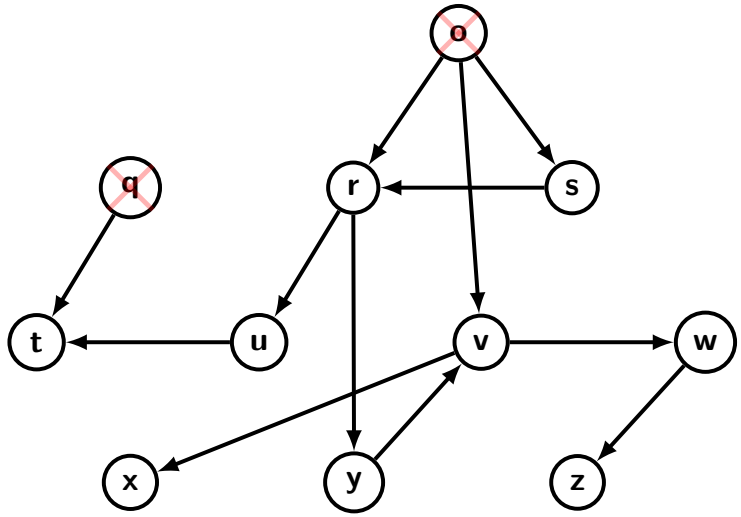
Topological Sort: all edges from left to right

p n o s m r u y v w z q t x

Topological Sorting



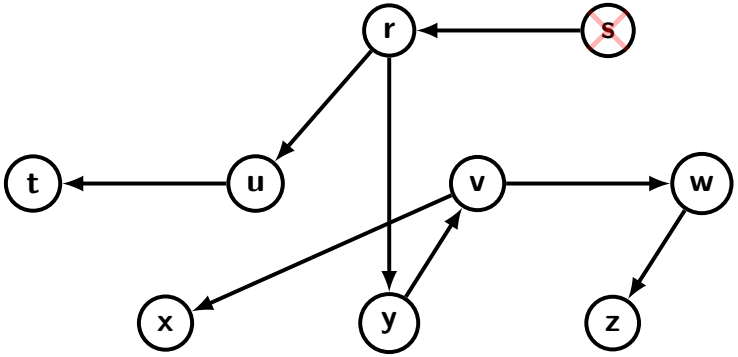
List: m, n, p



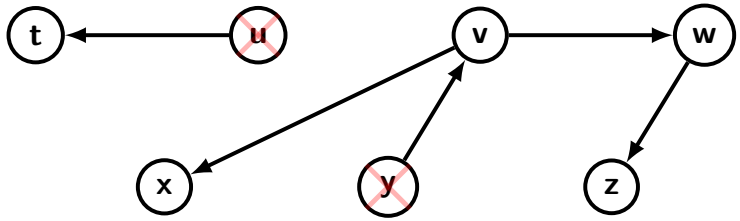
List: m, n, p, o, q

- Another simple way to get topological sort is as follows:
  - List out all vertices with no incoming edges.
  - Remove these vertices and keep repeating two step until all vertices as listed.

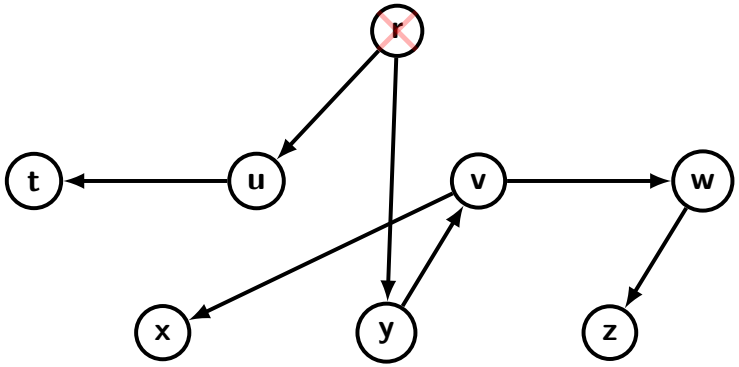
Topological Sorting



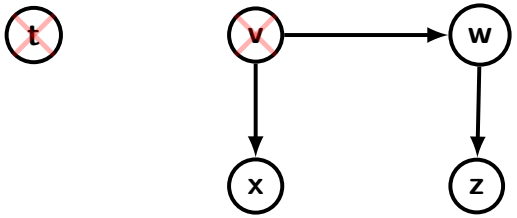
List: m, n, p, o, q, s



List: m, n, p, o, q, s, r, u, y

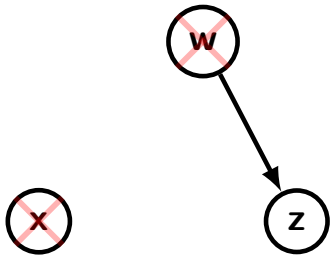


List: m, n, p, o, q, s, r



List: m, n, p, o, q, s, r, u, y, t, v

Topological Sorting



List: m, n, p, o, q, s, r, u, y, t, v, w, x

Final List:

m, n, p, o, q, s, r, u, y, t, v, w, x, z

- After deleting w and x only z is left out.
- Just append z to the list.
- As we can check the list orders that nodes such that edges alway directed from left to right.

Weighted Graphs

In introducing graph, as cost is implicitly assumed on links or edges. The cost is usually given as a numerical value. Formally, a weighted graph is defined as follows:

Definition (Weighted Graph)

If each edge of a undirected graph is associated with a weight then, it is known as a weighted graph. A weighted graph is defined in terms of a triple $G = (V, E, w)$, where $w : E \rightarrow val$. is a function that maps edges/directed edges to their associated values.

Typically, the value val is a real number.