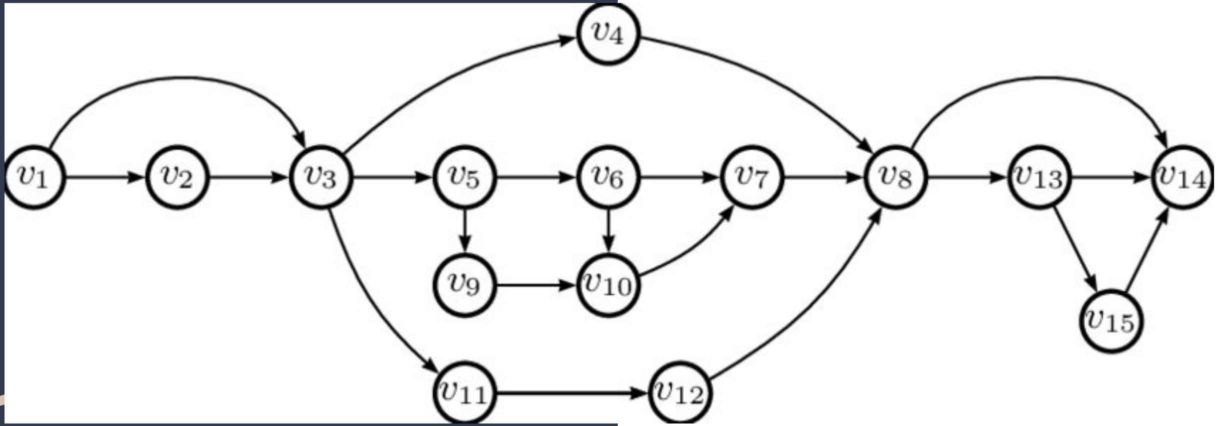
Topological Sorting

(Also called Toposort)

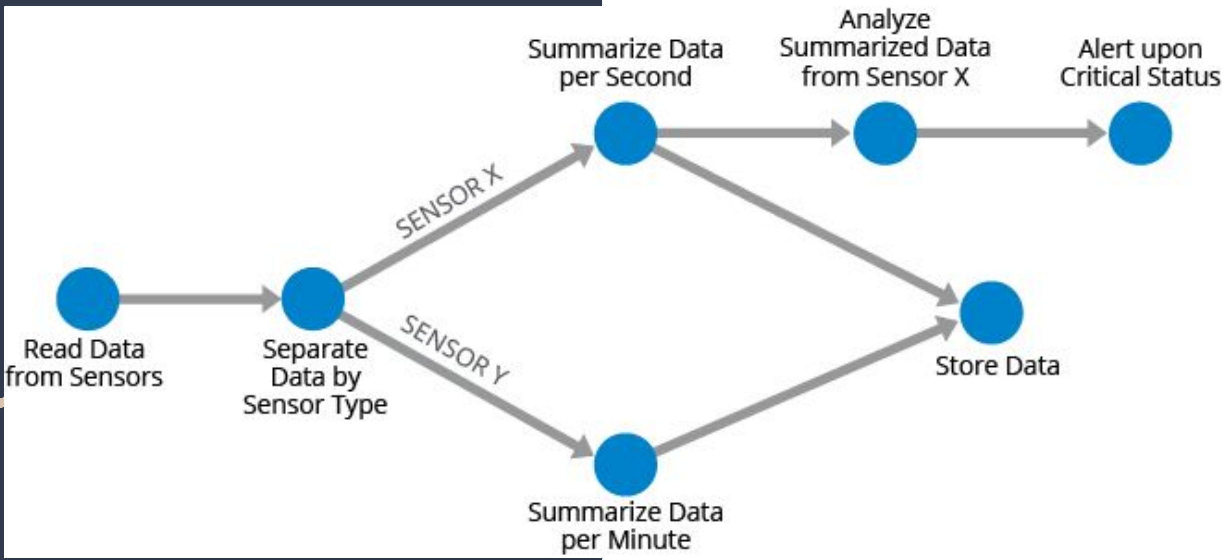
Directed Acyclic Graphs(DAGs)

A directed graph that has no cycles

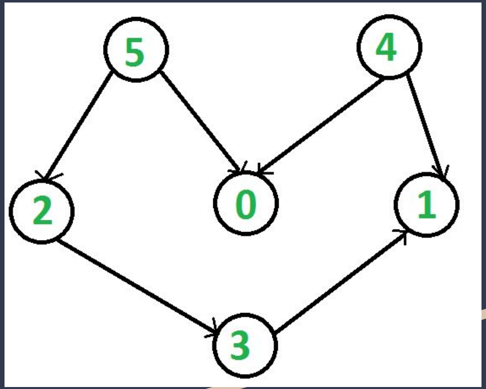


A **directed acyclic graph (DAG)** is a conceptual representation of a series of activities

Connecting Lines represent the flow of the graph



Topological Sorting



Example Graph

Linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Only possible in DAGs.

Example Topological sorts:

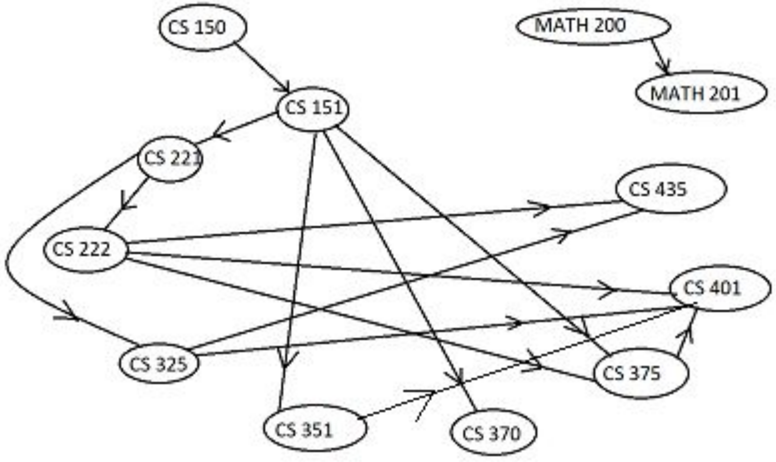
5 4 2 3 1 0

4 5 2 3 1 0

5 4 2 0 3 1

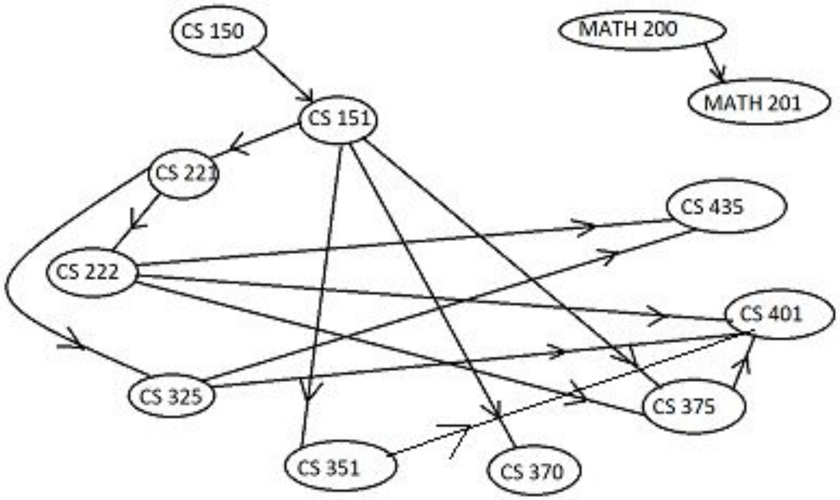
4 5 0 2 3 1

Main purpose of Toposort can be seen through the following problem statement:



Given this graph representing inter-dependencies on different courses, In what order should a student must complete the following set of courses?

Answer : Toposort



Order of completion :

MATH200, MATH201, CS150, CS151, CS221, CS222, CS325, CS435, CS351, CS370, CS375, CS401

Or any other possible Toposort of the given DAG.

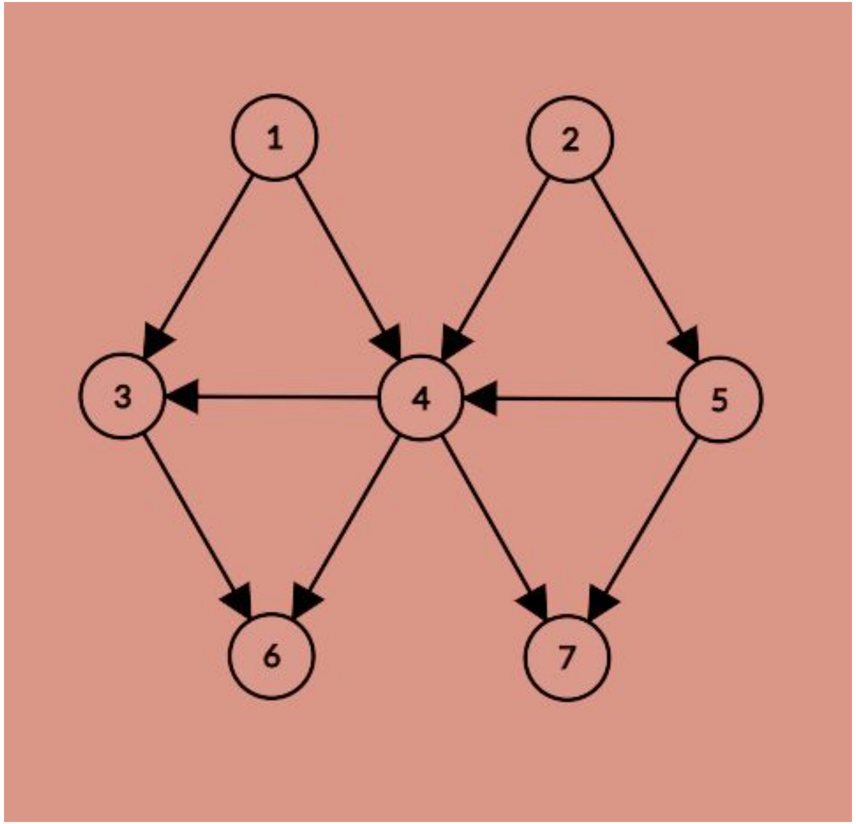
Code and Implementation

Two Approaches :

- Indegree Based
- DFS Based

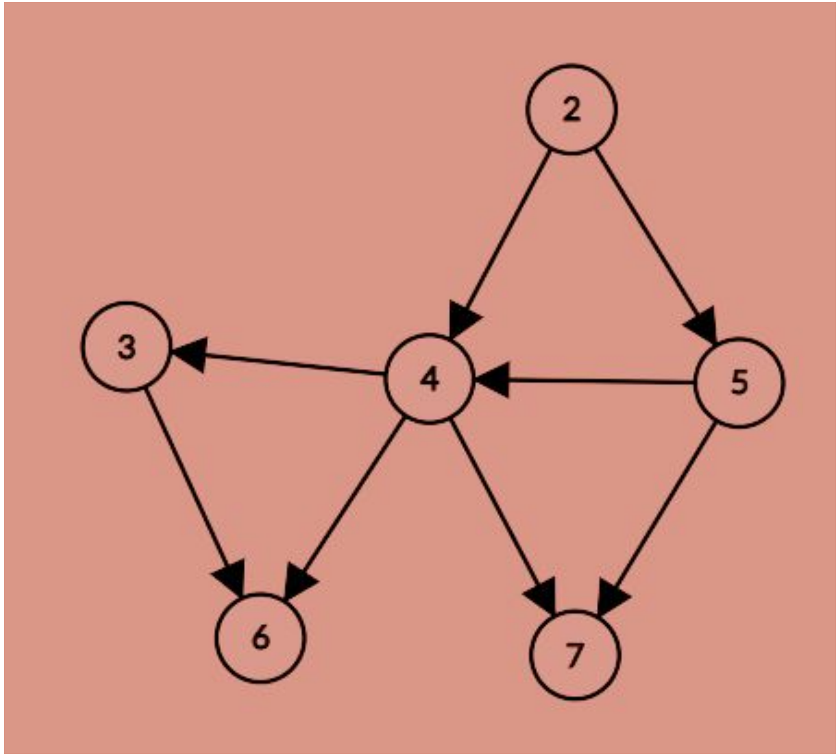
In-degree Approach / Kahn's Algorithm

1. Compute the indegrees of all vertices
2. Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)
3. Remove a vertex from the queue (Dequeue operation) and then. Increment count of visited nodes by 1. Decrease in-degree by 1 for all its neighboring nodes. If in-degree of a neighboring nodes is reduced to zero, then add it to the queue.
4. Repeat Step 3 until the queue is empty
   If count of visited nodes is not equal to the number of nodes in the graph then the topological sort is not possible for the given graph.



Indegrees
0: 1, 2
1: 5
2: 3, 6, 7
3: 4

Queue q -> 1 , 2

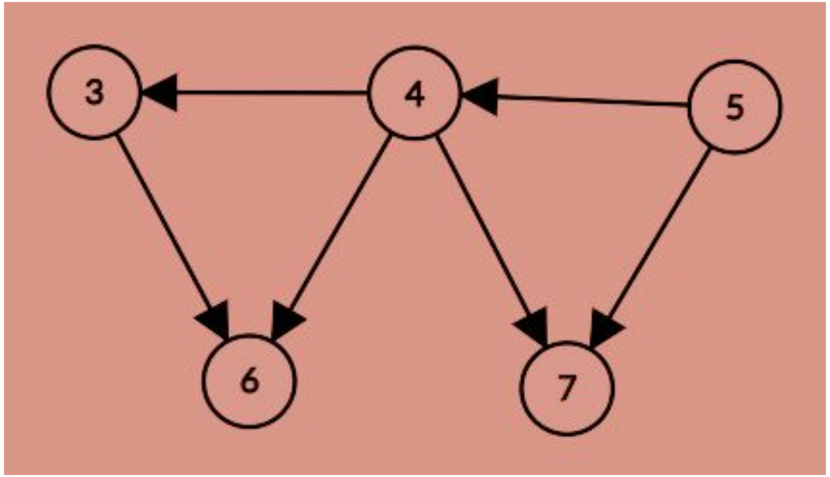


Node 1 dequeued and edges removed.

Indegrees
0: 2
1: 5,3
2: 6, 7, 4

Queue q -> 2

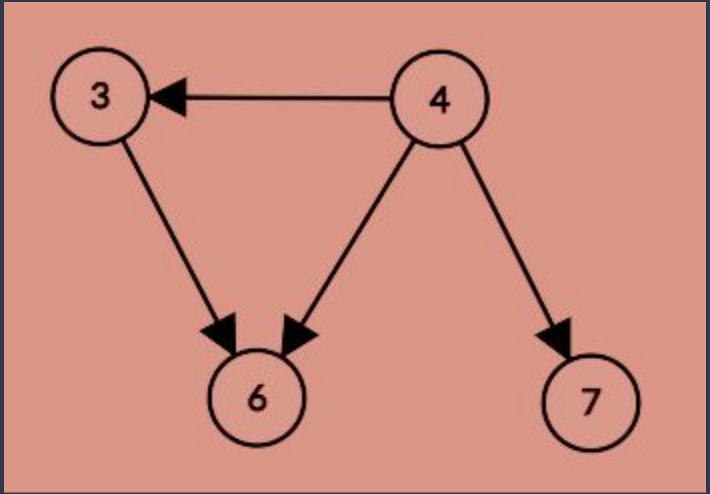
No new nodes with indegree 0 found



Node 2 dequeued and edges removed.

Indegrees
0: 5
1: 3,4
2: 6, 7

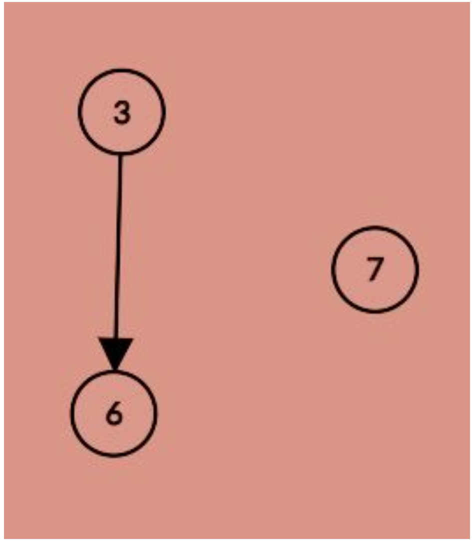
Queue q -> 5



Node 5 dequeued and edges removed.

Indegrees
0: 4
1: 3,7
2: 6

Queue q -> 4



Node 4 dequeued and edges removed

Indegrees
0: 3,7
1: 6

Queue q -> 3 , 7

Node 3 dequeued and edges removed.

Indegrees
0: 6,7

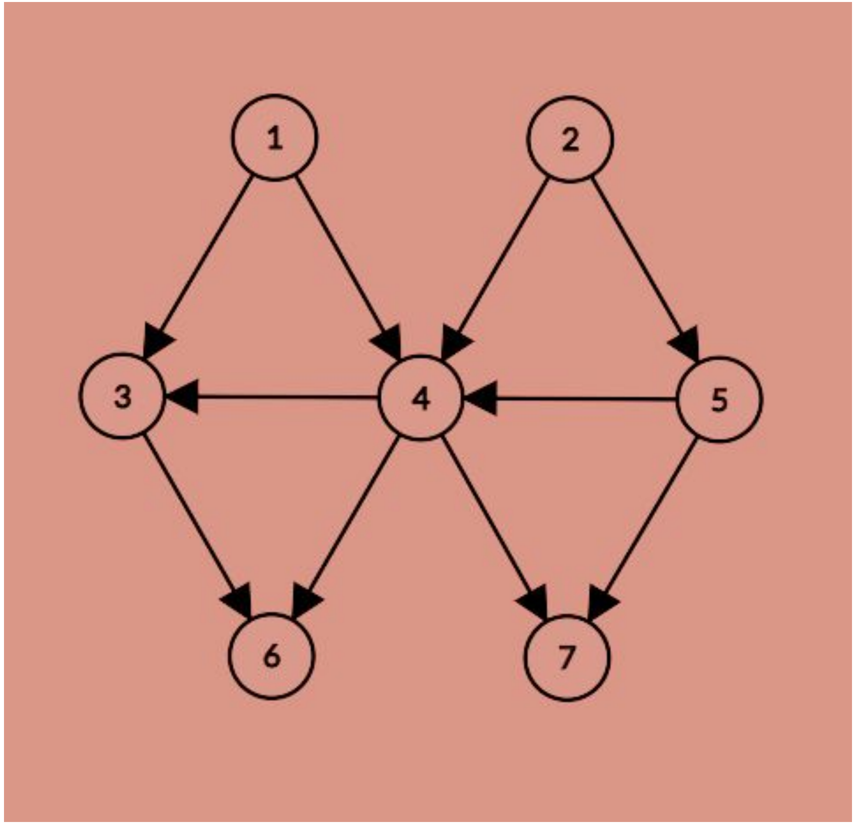
Queue q -> 7,6

Final few steps :

Node 7 dequeued.
Node 6 dequeued.

If all dequeue operations are printed the printed order becomes :

1,2,5,4,3,7,6



Try yourself : Indegree approach in a non-DAG

1,2,5,4,3,7,6

Code

```
int nodes = 7;
vector<int> adj[nodes];

// Vector to store indegrees, initialized to 0
vector<int> in_degree(nodes,0);

//filling indegrees
for(int i = 0;i<nodes;i++){
    for(auto it:adj[i]){
        in_degree[it]++;
    }
}


queue<int> q;

//enqueuing nodes with 0 indegree
for(int i=0;i<nodes;i++){
    if(in_degree[i]==0){
        q.push(i);
    }
}
```

initialization

Main code

```cpp
// count of visited vertices
int cnt = 0;

//store toposort in this
vector<int> ans;

while(!q.empty()){

    //dequeue from front
    int u = q.front();
    q.pop();

    //store popped element
    ans.push_back(u);

    //reduce indegree when edges of popped element are removed
    for(auto it:adj[u]){
        in_degree[it]--;
        if(in_degree[it]==0){
            q.push(it);
        }
    }
    //increment visited node
    cnt++;

}
```

Printing the answer or checking for cycle

```cpp
if(cnt!=nodes){
    cout<<"There exists a cycle in the graph\n";
    return 0;
}

//else print ans vector
```

Complexity Analysis of Indegree approach

Computation of Indegree : O(V+E)

Updating Indegree during dequeue per edge: O(1)
Total complexity of updating indegree during dequeue : O(V+E)

Total queue related operations : O(V+E)

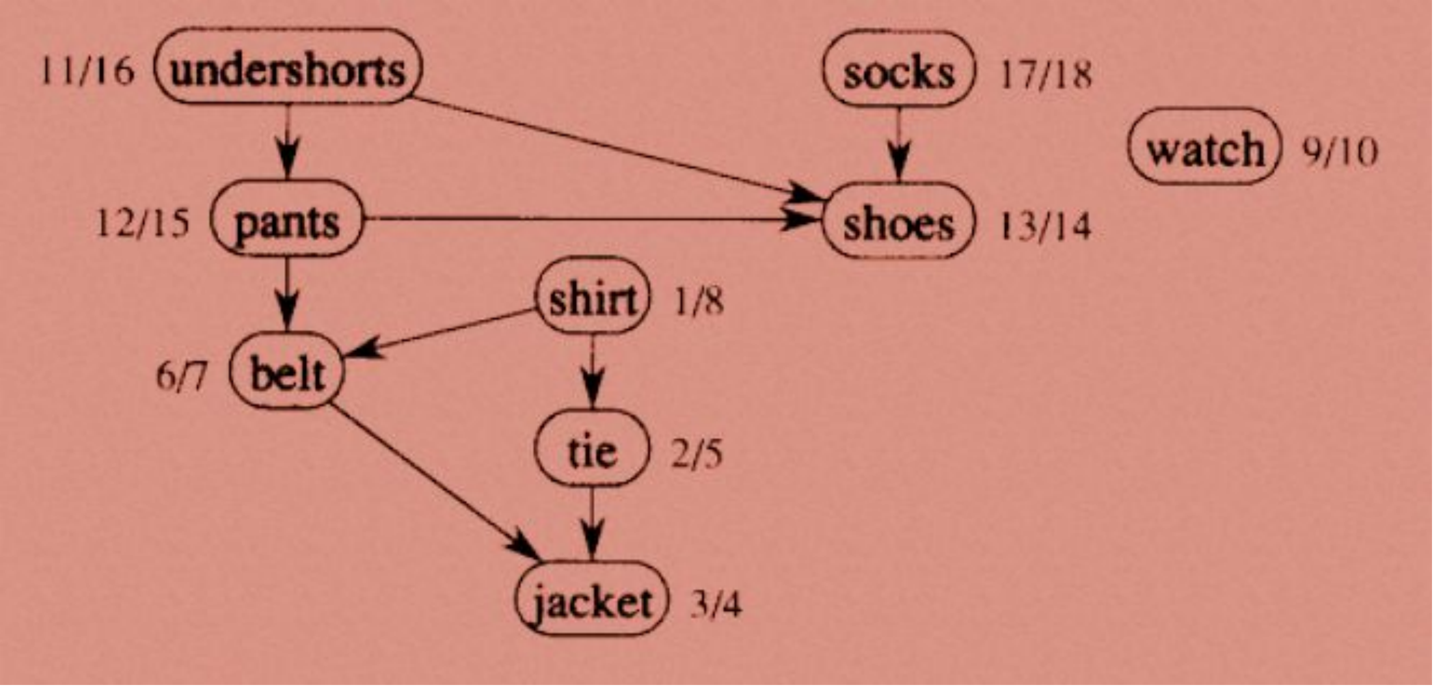
Net complexity of Indegree based Approach : O(V+E)

DFS Based Approach

Recall the concept of start time and finishing time in DFS from last class.

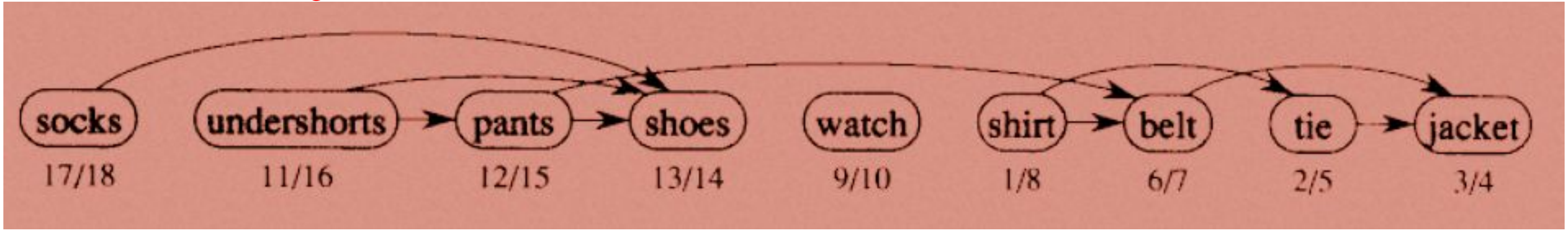
What do you get if you order the vertices in order of increasing start times? Answer is DFS Traversal order, pretty obvious this part!

Well , now we'll see that ordering the vertices in order of decreasing finish times gives you topological sort. How? See next slide ;)



No event dependent on a vertex can execute before the vertex's own execution.

Decreasing order of finish times



Ordering vertices in decreasing finish times

- **Option 1 : Run a DFS , note down finish times, sort to get output (Write extra code to make and sort pairs, a pretty simple task)**

- **Option 2 : Modify existing recursive DFS code (Even simpler, add 1 line of code to DFS,better complexity)**

```cpp
vector<int> adj[10];

//stack to store toposort , can use vector too, just remember to reverse order later
stack<int> toposrt;

int nodes = 7;
int visited[10];
```

Initialization

Driver code for topo() in main

```cpp
for(int i=0;i<nodes;i++){
    if(visited[i]==0){
        topo(i);
    }
}

//order reversed while popping
while(toposrt.size()){
    cout<<toposrt.top()<<" ";
    toposrt.pop();
}
```

Code for Modified DFS

```cpp
void topo(int src){
    visited[src] = 1;
    for(auto it: adj[src]){
        if(visited[it]==0){
            topo(it);
        }
    }

    //pushing nodes at finish times
    toposrt.push(src);

}
```

Complexity Analysis of DFS approach

You know, the