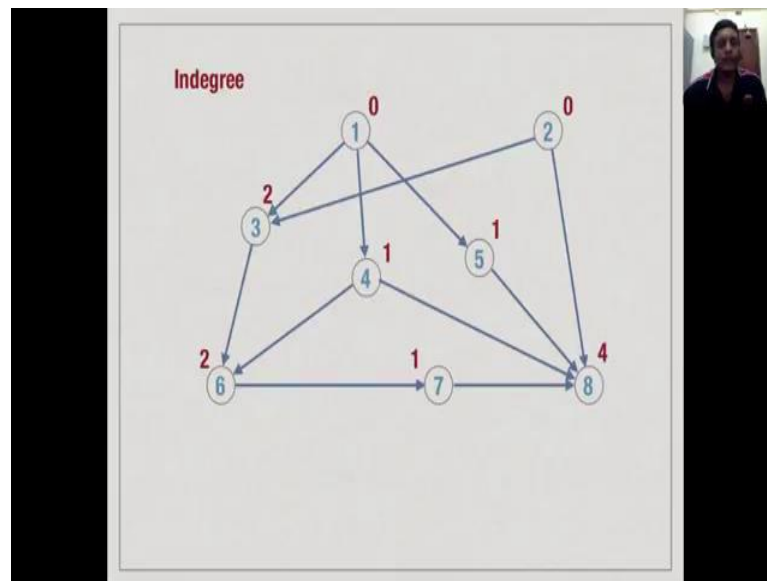


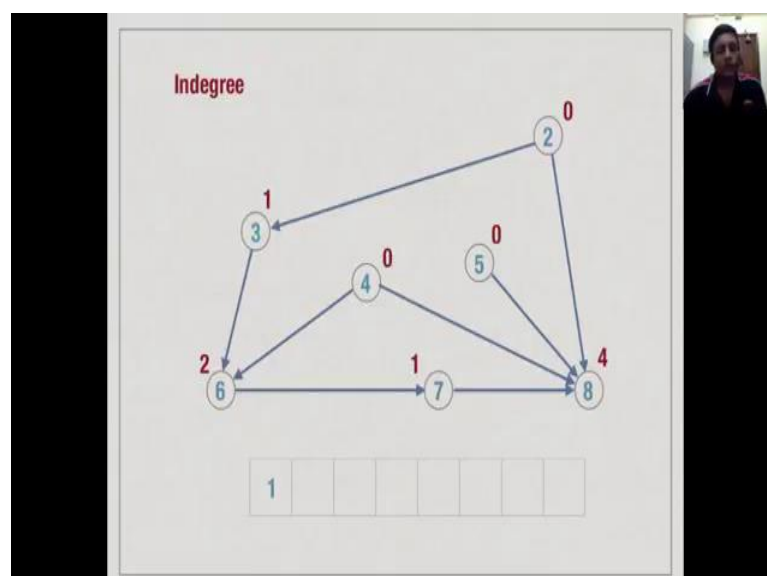
(Refer Slide Time: 11:00)



So, let us apply the strategy to this DAG, so we first begin by labelling every vertex by its in degree. So, we have indicated the in degree of every vertex. So, for instance 1 and 2 have no incoming edges. So, they have in degree 0, vertex 3 has 2 edges coming as in degree 2, vertex 8 has 4 edges coming inside and in degree as 4 and so on, now we have to pick up a vertex of in degree 0 enumerated and eliminated.

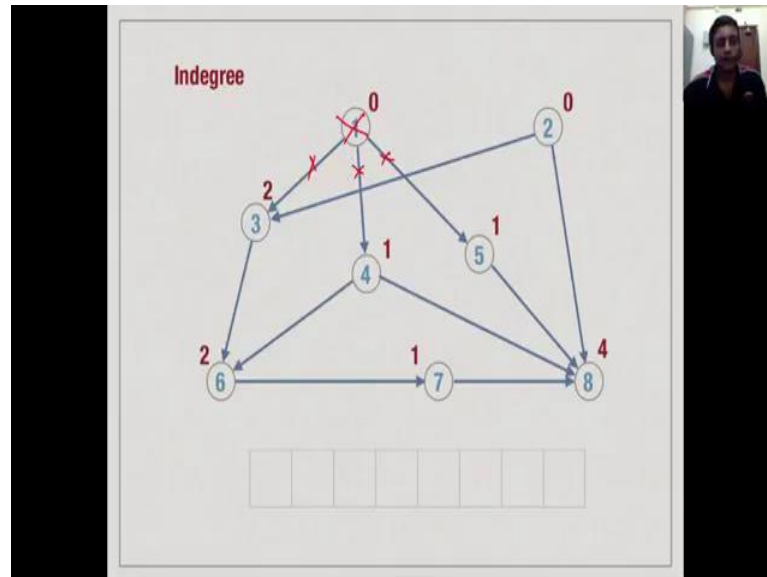
So, we have a choice between 1 and 2, so let us suppose we start with 1, so we start with 1 we eliminated and now when we eliminated we also eliminate the edges going out of it. So, the edges coming into the 3, 4 and 5 will reduce by 1, because a vertex 1 is gone, so the edges coming into them reduce by 1 their in degree is also reduced by 1.

(Refer Slide Time: 11:57)



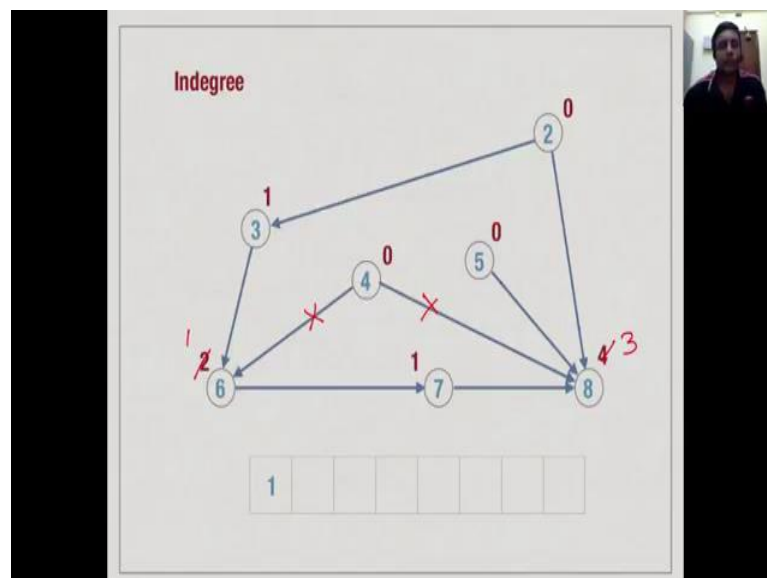
So, what happens on eliminate 1 is that we enumerated and we reduce the in degrees of 3, 4 and 5 from 2, 1 1 to 1 0 0.

(Refer Slide Time: 12:04)



So, recall that before that the in decrease or 2 1 and 1, now these edges which are coming into them have been deleted. So, when we delete this, we also delete the incoming edges.

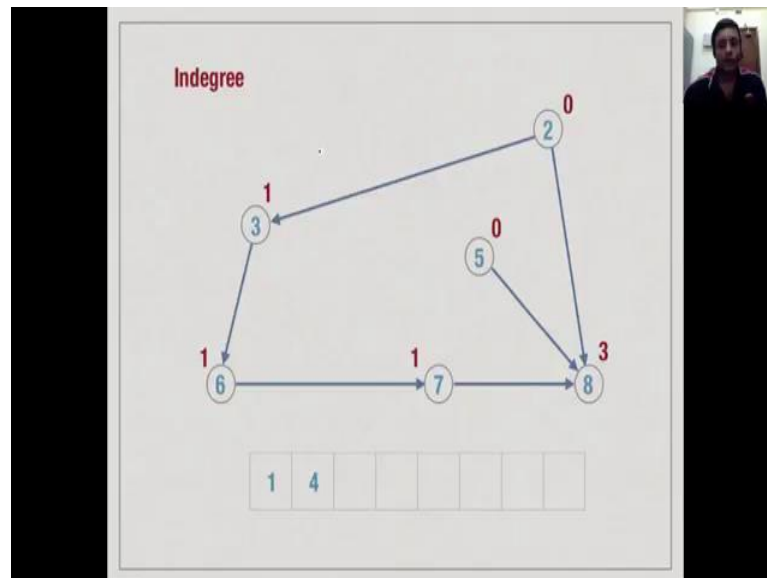
(Refer Slide Time: 12:14)



So, now we have 1 0 0, now we have three choices, two the original one which are in degree 0 and we have two new vertices 4 and 5 which correspond to tasks if you want to call them, whose prerequisite has been completed. So, tasks 1 was a only pre requested for 4, task 1 is only prerequisite ed for 5 it has been completed. So, 4 and 5 or now

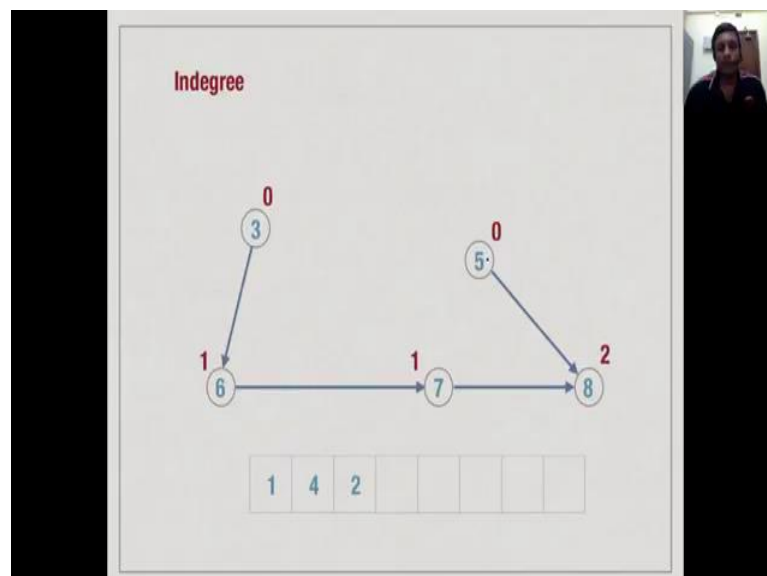
available, so we can choose any of 2, 4 and 5 it does not matter. So, let us suppose we choose 4 then again these two edges will go. So, this will reduce to 1 and this will reduce to 3.

(Refer Slide Time: 12:48)



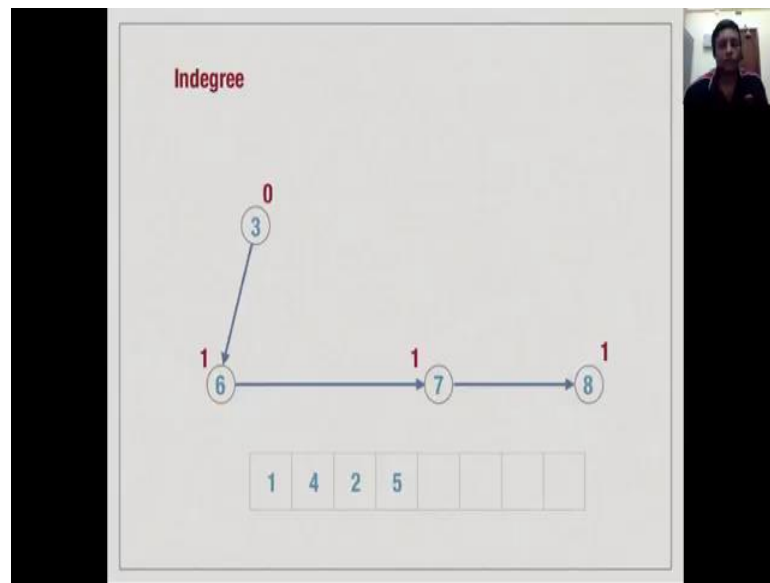
So, we can do that eliminate 4 and reduce the in degree of 6 from 2 to 1 in degree of 8 from 4 to 3, now perhaps which is decide to eliminate task 2.

(Refer Slide Time: 13:00)



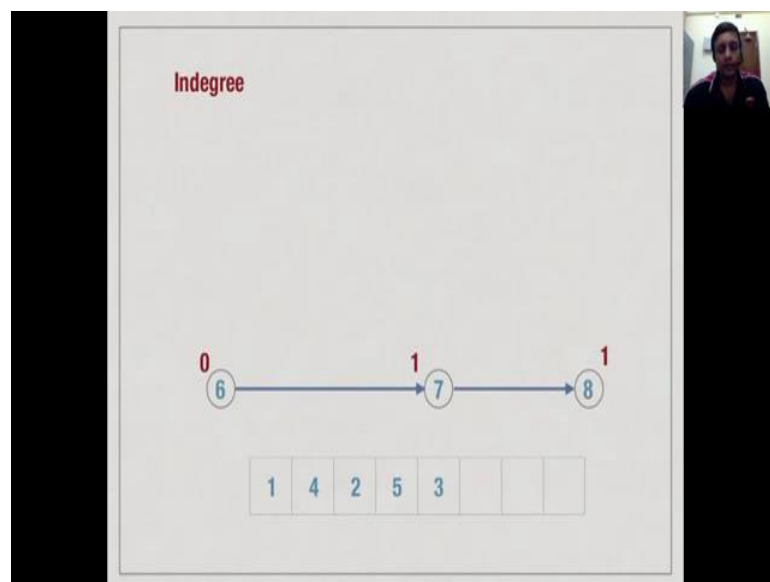
So, when you do pass 2 then the in degree of 3 reduces and the in degree of 8 again reduces. So, notice that 8 still has to pending in requirements only 5 and 7, so it cannot be done yet, but 3 and 5 are the available.

(Refer Slide Time: 13:14)



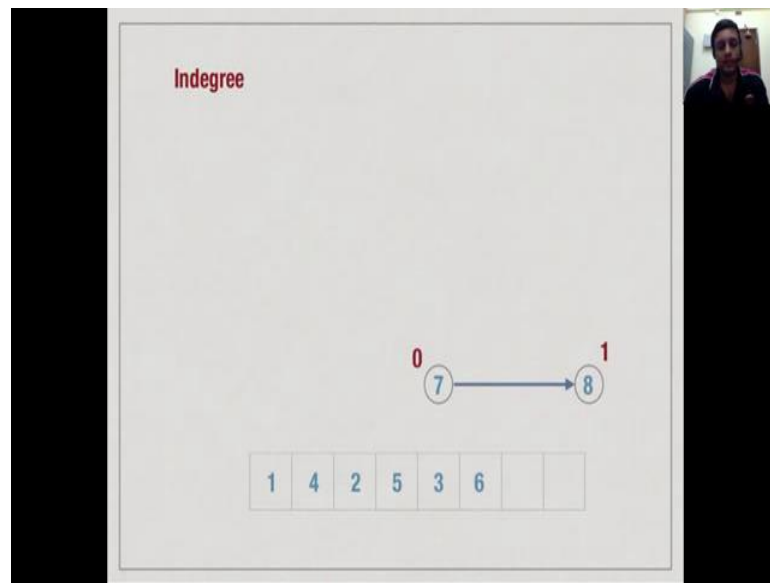
So, perhaps I do 5, so now 8 plus 1 and now have no choice, the only task with in degree 0 is 3. So, then I do 3, so now I can see that this is actually the way it is drawn is that is the sequence, I must do 3 before 6, 6 before 7, 7 before 8. So, I have no choice to this point I must enumerate as 3.

(Refer Slide Time: 13:32)



Then, 6.

(Refer Slide Time: 13:33)



Then, 7.

(Refer Slide Time: 13:34)



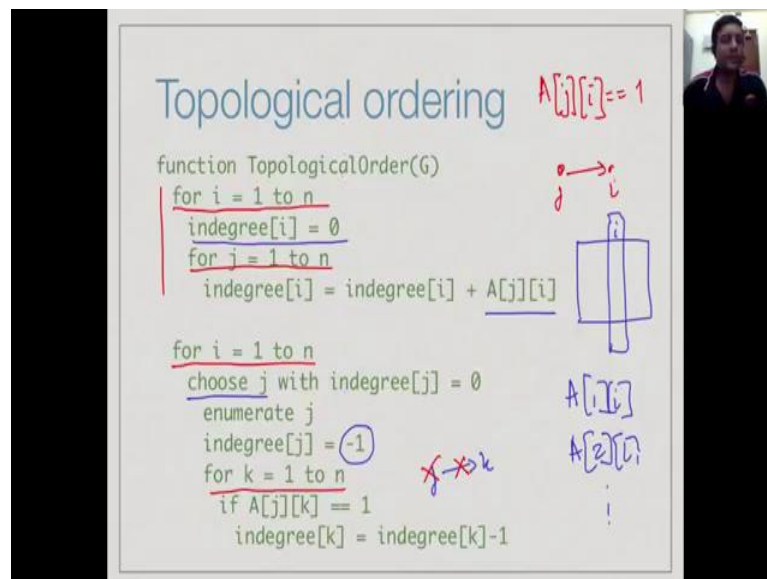
Then finally, 8.

(Refer Slide Time: 13:36)



At this point my graph is empty and I have to obtain a sequence of vertices which is a valid topological ordering, because every pair of vertices which occurs an edge in my original graph is order. So, that source of the edge appears before the target to the edge.

(Refer Slide Time: 13:52)



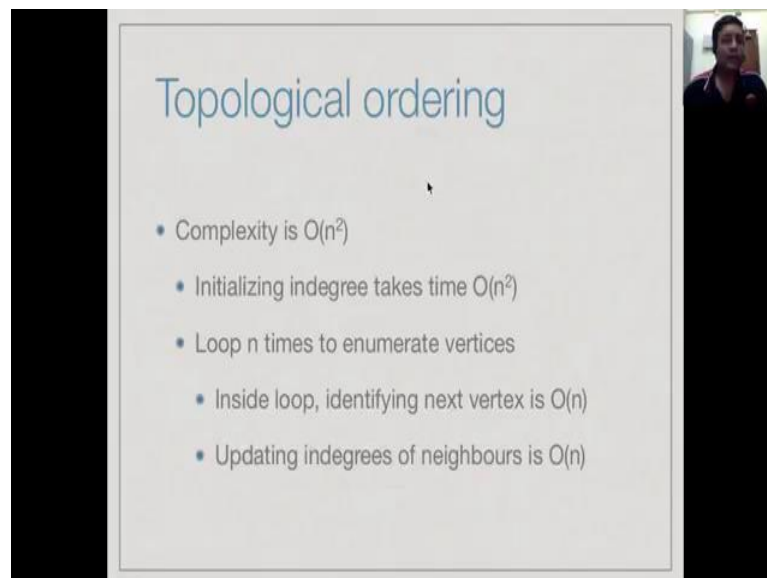
So, let us look at some pseudo code for the algorithm which we just executed by DAG. So, in this particular algorithm we first start by computing the in degree, so in order to compute the in degree, we need to find out how many for a vertex i we need to find out how many j satisfy the property that $A[j][i]$ is equal to 1, because this corresponds to an edge from j to i . So, when an adjacency matrix this corresponds to looking at a column containing line.

Because, in the column containing I , you will have entries at the form A_{1i} , A_{2i} and so on. So, we will have these entries and we want to scan all of these and then add up all the 1s. So, we start by setting in degree equal to 1, a in degree i equal to 0 and then for every row j we add A_{ji} . So, it is either 0 or 1 and so we therefore, collect all the incoming edges which pointing to i as the in degree of i , now we start enumerating. So, we know this is a DAG, so we know there is at least 1 j with in degree 0 at an every point.

So, we choose any such j , choose a j which has in degree 0 enumerate it, now when we enumerate we want to eliminated from the graph. Rather than, going an actually modifying the graph itself, we will just work with in degree as a kind of approximate version of that modified graph. So, we first set the in degree to minus 1 for this particular vertex. So, minus 1 means it cannot be in the graph, because you cannot have minus 1 edges pointing can have at least 0 edges of mode.

So, this effectively means that the j is not going to be considered hence 4 and now for every outgoing edge from j . So, wherever we have j pointing to k we want to decrement this, because we are going to eliminate this edge eliminating j , we eliminate this edge. So, for every k from 1 to n we scan out going neighbours of j and if $j k$ is an edge A_{jk} is 1 we reduce the in degree of k by 1.

(Refer Slide Time: 15:48)



Topological ordering

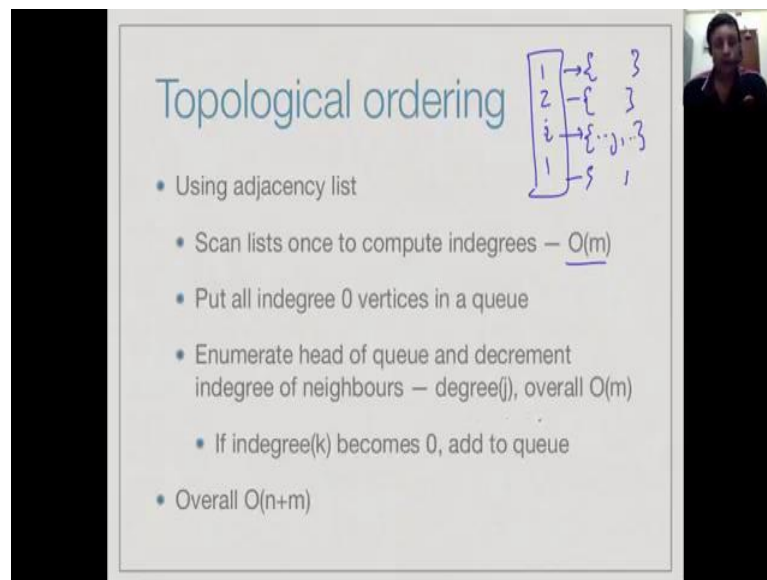
- Complexity is $O(n^2)$
 - Initializing indegree takes time $O(n^2)$
 - Loop n times to enumerate vertices
 - Inside loop, identifying next vertex is $O(n)$
 - Updating indegrees of neighbours is $O(n)$

So, what is the complexity of the algorithm is fairly easy to see that for this adjacency matrix or presents it is n square, as we saw initializing the in degree itself takes time n square ((Refer Time: 16:01)). Because, we have one outer loop from 1 to n and then for

each outer loop we have an inner loop from 1 to n , so this is a n^2 loop. And then, when we enumerate the vertices again we have an outer loop which will enumerate every vertex once.

And then for the inner loop, we have to enumerate check all its neighbours and decrement. So, we have $2n^2$ loops and therefore, this whole thing takes order n^2 .

(Refer Slide Time: 16:29)



Topological ordering

- Using adjacency list
- Scan lists once to compute indegrees — $O(m)$
- Put all indegree 0 vertices in a queue
- Enumerate head of queue and decrement indegree of neighbours — $\text{degree}(j)$, overall $O(m)$
- If $\text{indegree}(k)$ becomes 0, add to queue
- Overall $O(n+m)$

Handwritten diagram showing an adjacency list structure:

```

graph LR
    1 --> 3
    2 --> 3
    i --> j
    j --> i
    1 --> 1
  
```

Now, as we saw with BFS and DFS, if we use an adjacency list we can be a little more clever and we can bring down this time to linear from n^2 . We can bring it down to order $n + m$. So, how do we do this? Well, we have this list, so we have a list say 1, 2 and for each of these we have a list of its neighbours. So, if you go through this as we said each edge in this, now this is a directed graph. So, each edge is represented only once if an edge from i to j it will appear as an entry j in the list for i .

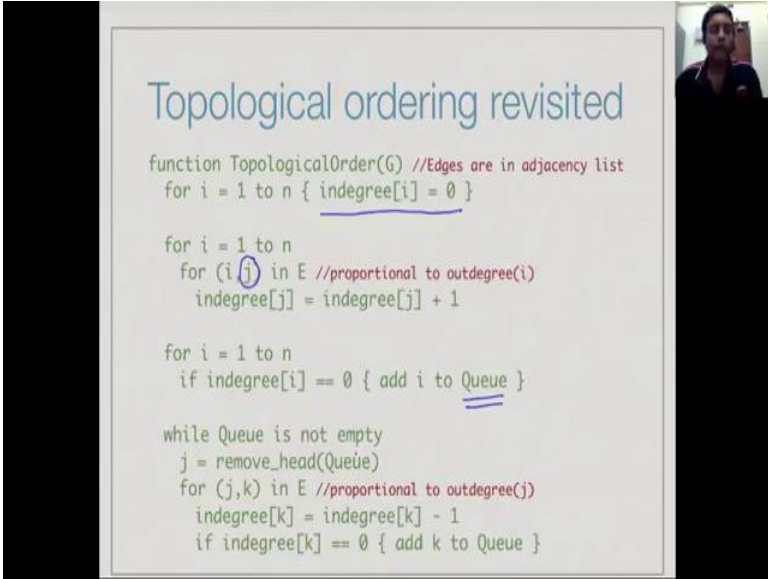
So, if you scan these lists every time we see a j , we know there is an edge pointing in to j and we will increment. So, we start off by setting all the indegrees to 0, we scan all the lists and every time we see an entry in a list, we increment this indegree. So, one scan of the list that is in time order n , we can find the indegrees. Now, we have the list of indegrees, so we can put all the indegree 0 vertices into a queue, this makes it easy to find which vertex to enumerate next.

So, at the end of this scan we have done an order m scan to find all the indegrees, now and an order n scan we can put all the 0 degree vertices in to a queue. Now, we can do

the rest pretty much as we did much before, we enumerate the first vertex in the queue and then we go to its list which is now explicitly available to a adjacency list is outgoing neighbours, decrement its in degree and if any of those in degree is become 0 we can added to the queue.

So, we know it is to be processed now, so this becomes overall it take order m time to scan the list takes order n time to start the queue of and then this is the loop of order n where across all the updates we will overall update in degrees order n times, so this is order n plus m.

(Refer Slide Time: 18:26)



Topological ordering revisited

```

function TopologicalOrder(G) //Edges are in adjacency list
  for i = 1 to n { indegree[i] = 0 }

  for i = 1 to n
    for (i, j) in E //proportional to outdegree(i)
      indegree[j] = indegree[j] + 1

  for i = 1 to n
    if indegree[i] == 0 { add i to Queue }

  while Queue is not empty
    j = remove_head(Queue)
    for (j, k) in E //proportional to outdegree(j)
      indegree[k] = indegree[k] - 1
      if indegree[k] == 0 { add k to Queue }
  
```

So, here is the corresponding pseudo code, so the first step is to initialise the in degree to 0 for every vertex in our graph, then we go through all the edges. So, we do this by looking at each adjacency list. For each vertex we look at each neighbour i, j and the adjacency list and we for each of these we increment the in degree of j, because we are looking at edges pointing into j, not pointing out of i. So, this pointing into j will come in different list.

But, as and when we encounter them for each of them we will account for them and add one to it. Now, we go through the list one more time a list of vertices and every time will see an in degree 1, 0 we add i to the queue. And now we do this loop, till the queue becomes empty we know this at least one at every point remember along the graph a is DAG is not an empty, we know there is at least one in degree vertex with in degree i 0. So, they must be in the queue, because we adding them all originated the queue and each

one we generate we will add to the queue.

So, along with a queue is not empty we take of the first element of the queue, then we look at its adjacency list, decrement the in degrees of all those vertices and if any of them happens to now become in degree 0, we add to the queue. So, this becomes now a linear implementation of topological sort, using adjacency list and a queue to process the elements. Because, the reason why we need this queue is that otherwise we have to scan all the vertices every time to determine whether a vertex or become in degree 0, then that becomes an order n scan within this become order n square again.

So, we need the queue to make sure that we do not spend time trying to identify the next vertex to enumerate. We do not have to go through all the vertices and check the in degrees, when we see the in degree 0 we put it into the queue. So, automatically it will come out without having to do any further check. So, it gets observed in this order m work that we are doing here.