Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

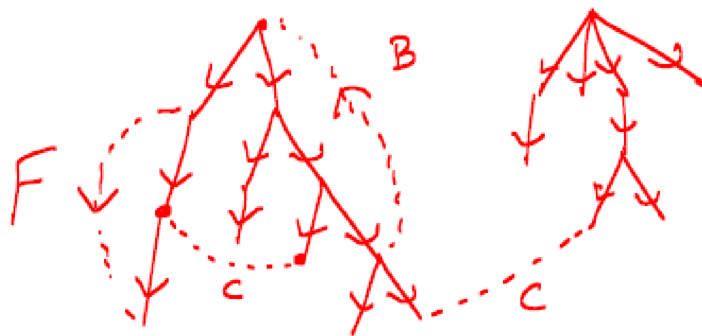
Institute: IISc

Week: 07

Lecture 32 DFS in Directed Graph

Namaskara, in this lecture I am going to discuss about the depth first search done on directed graph, depth first search in directed graph. The skeletal structure is same. much same like undirected one the difference comes in dealing with non-tree edges. In the case of undirected graph there is only one kind of non-tree edge called the back edge, they always go from a descendant to an ancestor, however for the directed graph the non-tree edges can fall into 3 different categories, there are 3 types of non-tree edges. Let us say we have the depth first search done in a directed graph and assume that we have got the trees. Something like this and you know that when the DFS is done from wherever you have started, all the vertices that are reachable, that is if there is a path, then that will occur in the DFS tree with that node as a root but if not all the vertices are reachable.

In order to explore the graph you have to make a fresh start from one of the vertices that you have not reached or not visited and that fresh start may give rise to another kind of a tree, it is in the same graph except that you made the second starting point and then you have proceeded with this and in this way you may have several DFS trees starting at different vertices. It is a same graph okay.

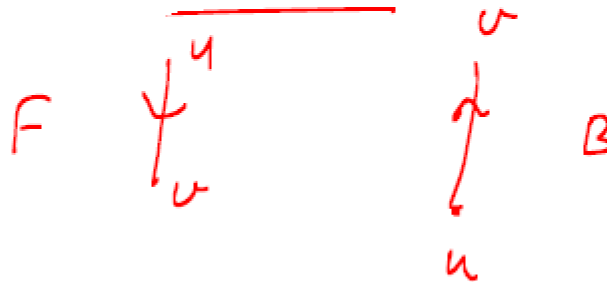


The non-tree edges can be of 3 different categories. One of the non-tree edges could be something like this and this is called the forward edge, I have written F, in near that edge ,it is a non-tree edge, it is not part of the tree, it goes from a visited node to another visited node, right therefore it is a non tree edge. From a visited to an unvisited node

when an edge goes that will be a tree edge, from a visited node to an unvisited node when you go that defines a tree edge.

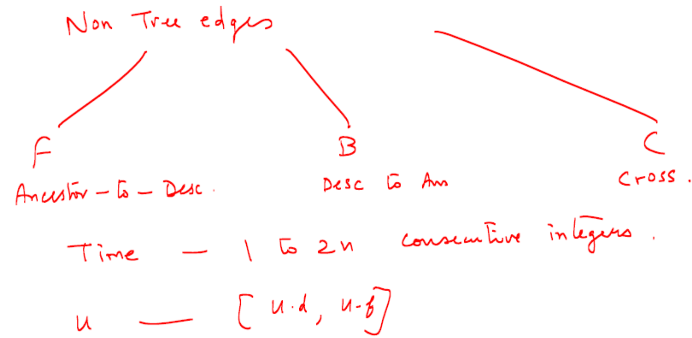
From a visited node to another visited node when an edge goes those edges are non-tree edges. So here is an edge, it is a non-tree edge, this is called the forward edge. I am going to draw another one this way it is called backward edge or back edge, I have labeled that as B and then there can be certain edges like this, they are called cross edges they go across. okay for example this is an example of a crossing edge or a cross edge. So the non-tree edges are of three kinds forward edge, forward edges go from ancestor to descendant, backward edge they go from descendant to ancestor, it is going back but it is going to an ancestor. So this will go from descendant to ancestor okay that is why this is back edge. The third kind cross edge the cross edge connect 2 vertices that are not related by ancestor descendancy type okay they are in different branches. they are in completely different subtrees. The subtrees at these end vertices are disjoint one is not going to include the other okay that is the reason why such edges are called cross edges. So in the picture you can see that these cross edges they they are not related by ancestor descendant.

So if an edge e equal to uv if u is ancestor and v is a descendant u is an ancestor and v is a descendant this is forward edge. And if u is descendant and v is ancestor, so this is forward edge and this is backward edge or back edge.



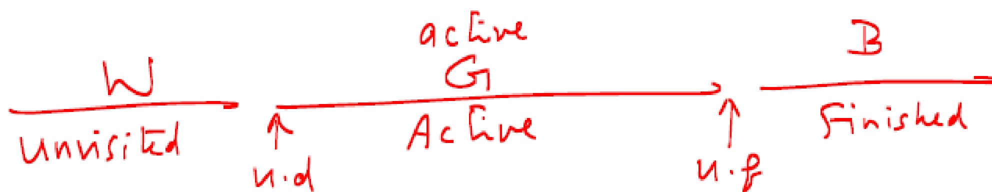
If u and v are not related, they are in the different subtrees or one is not the ancestor of other, this is in the same subtree but you can see that these two vertices. They are neither ancestor nor descendant. Any edge connecting these type of vertices they are called cross edges.

So we need to recognize when a non-tree edge would be a back edge or a forward edge and so on. The nodes that are active during a DFS, when a node, see if you look into the entire span of DFS, the timestamp starts from 1 and for every entry and for every exit the time is incremented by 1. Therefore the time is going to take consecutive integer values 1 to 2n consecutive integers and these values are assigned to the vertices as a discovery time and finish time. So if you take a vertex u, u.d is the discovery time, u.f is the finish time. The depth first search enters u, explores everything and then exits, finish time.



So u.d, u.f defines a time interval, an integer interval. This is the period in which u is active, so DFS happens from 1 to 2n various vertices are active. In various sub intervals u may be active from here to here, v may be active from here to here and another vertex may be active from here to there and so on. Various vertices are active in various time intervals, so before it is active it is unvisited, it is not visited and its color will be white. When it is active, the color will be gray, once it is finished, it will be black, the vertex will have a color attribute set to black, so white, gray and black.

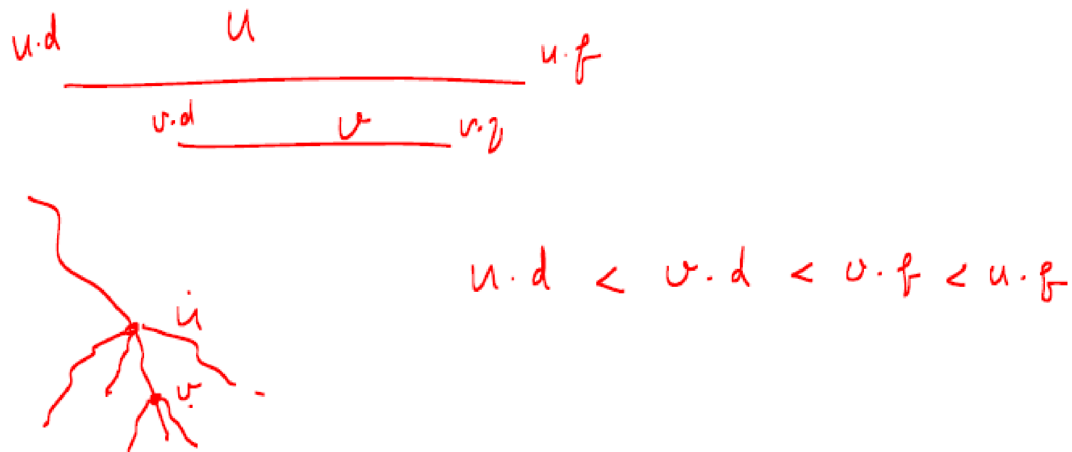
So if you look into the execution, a vertex will remain white until it is visited. The moment it is visited, it is set to gray in the first step and you can see when it is set to black, when all the neighbors are visited, when the entire visit related to that vertex is over, it is set to black. After that, that vertex will never be visited again. It is finished. So once it, after this point of time, So this is discovery point, this is the finish point ,after that it will remain black, this true for every vertex.



These intervals the time intervals related to the vertices they have some very interesting properties either they are disjoint or one is contained in the other it is going to be like this. or like this one is going to be contained in or other or it is disjoint. It will never be like this kind of there would not be any overlapping intervals, one is going to contain the other or they are going to be disjoint, take any two vertices every vertex has got an interval associated with this, the interval is defined by its discovery time and finish time and those intervals are either disjoint or one containing other but they will never overlap like this okay. When one is contained in the other, let us say this is the interval of u and this is the interval for v, v is a descendant of u descendant means u is discovered first then the exploration continues, backtrack, then the exploration continues and backtrack

So somewhere here v will be there, so v is going to be discovered after u is discovered ,then it backtracks and then it continues and then it is finished.

So u.d will be less than v.d because this is discovered earlier, v is discovered later and v is finished earlier and u is finished later, so u dot d is less than therefore this is u dot d this is v dot d, this is v dot f, this is u dot f, you can see that it is like this



If this is the case then v is descendant of u. The ancestor would flip just the same relation it will be still contained in another right because if v is a descendant of u, u is an ancestor of v. So ancestors intervals will be bigger descendants interval will be smaller okay. If two vertices have no ancestor descendant relation then there will be disjoint like this if this is u and this is v, u will be in one part v will be in another part one will not be an ancestor or descendant because if it is an ancestor or descendant one interval completely goes into another interval it is not like that.

v and u are not related by ancestor descendant relation that means they are going to remain disjoint it will never overlap like this. So these are all some basic properties of the intervals associated with the vertex. You do the time stamping, the time stamping naturally gives rise to an interval associated with a vertex and when you have the information related to that interval ,you can say whether one is an ancestor or one is a descendant whether an edge is a back edge or a forward edge, because back edge and forward edge are only between ancestor and descendants okay. So you can see that if u is an ancestor of v, u is finished after v is finished. So u will be gray when v is finished it is still live, v is finished and then go back and then it may continue at u.

Therefore you have very interesting basic properties that are allowing us to characterize and recognize when an edge is a back edge, when an edge is a forward edge and when an edge is a cross edge and so on. So the control point for non-tree edges now split into three further options. It is a non-tree edge alright. If it is a cross edge what you have to do if it

is a back edge what you have to do they may be different, in that case you have to put them in the appropriate control points. So we will give now a more elaborate version of the depth first search of a directed graph with the conditions for non-tree edges of each category explicitly mentioned okay, the pseudo code is here.

This is called the driver code DFSG for each u belong to vertex set of G, u.color equal to white u dot p the parent is null there is no parent each vertex is not even connected time is 0 we set it to 0 it is a global variable for each u in v, if u dot color equal to white then DFS Gu. So you start from a vertex u if it reaches all other vertices everything will be colored black. So this for loop, for each u you start with one u if that accounts for all other vertices there is no further trigger of DFS Gu is possible, if there are still some vertices that are white a fresh DFS will start and a tree from that node as a root will get built up okay.

```
DFS (G)
    For each u ∈ V
        u.col = white
        u.p   = NULL.

    Time = 0
    For each u ∈ V
        if (u.col == white)
            DFS (G,u).
```

So this is the DFS Gu how are we going to do DFSGu. Suppose you start from a vertex u what do you do, first time stamp update the time stamp u.d this is a discovery time, this is the first time you are entering u.d is time and u.color equal to gray, it has become active because just you have entered into this, for each v belonging to adjacency list of u. If v dot color equal to white, it is an unvisited vertex, then this is a tree edge, v dot p equal to u, this means uv is a tree edge and DFS Gv, if v dot color is not white so else uv is a non-tree edge.

```
DFS (G, u)
    Time = Time + 1;
    u.d = Time;
    u.col = Gray;
    For each v ∈ Adj[u]
        if (v.col == white)
            v.p = u;
            \\ (u, v) is a Tree
                           edge.
            DFS (G, v).
        else \\ (u, v) is a
                 Non-Tree edge.
```

If it is a non-tree edge we have further things to look at so it is not white therefore it could be gray or it could be black the color of V could be grey or black therefore in the uv is a non-tree edge, if v.color equal to grey. because color of v is not white therefore it could be gray or it could be black. If v dot color equal to gray that means uv is a back edge okay. When you come to this statement, well you can write like this or you can put even an else statement here it does not matter, so if v.f is less than u.d, then uv is a cross edge, because v is already finished and u has started later, that means v is in some other part of the tree and then it is not in the branch or it is not in the path from root to v at all, it is already done okay. Therefore u dot v is a cross edge, else it is a forward edge, you can write simply like this else forward edge, either it is a back edge or a cross edge or a forward edge. You can write down the condition for any two and put in the else class the third one there are only three types okay.

```
else        \\ (u, v) is a non tree edge.
    if (v.col == Gray)
        \\ (u, v) is a back edge. <--
    if (v.f < u.d)
        \\ (u, v) is a cross edge. <--
    else
        \\ forward edge.
```

Therefore if you want to add a computation that is to be done with respect to a back edge those computational steps are to be included here, if you want to consider doing something only when you find a cross edge those computations are to be done when in this part, in this after this if statement, it is a skeleton fitting the appropriate computations where it is to be carried out. The logic involved in the solution of the problem will tell you what are the things to be done and where those codes are to be fitted here okay it will determine that.

So all that I have done is that I have, this an elaboration of control point related to non-tree edges. We have seen a canonical example of using this kind of a skeleton to solve cut vertex problem for undirected graph. We have already seen, how to solve the cut vertex problem for undirected graph. For directed graph, we are going to take a look at an algorithm for finding strongly connected component of a directed graph. We will see an interesting solution for this, there are several algorithms for this, we will take a closer look at an algorithm in which the computation to be done to determine the strong components are characterized in terms of the non tree edge related computations. Now, we know when, we know how to recognize a particular kind of a non tree edge and the computation to be done there must be fitted there, once you do that you have an algorithm for finding the strongly connected components. We will take a closer look at one such algorithm in our next class, thank you.