Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

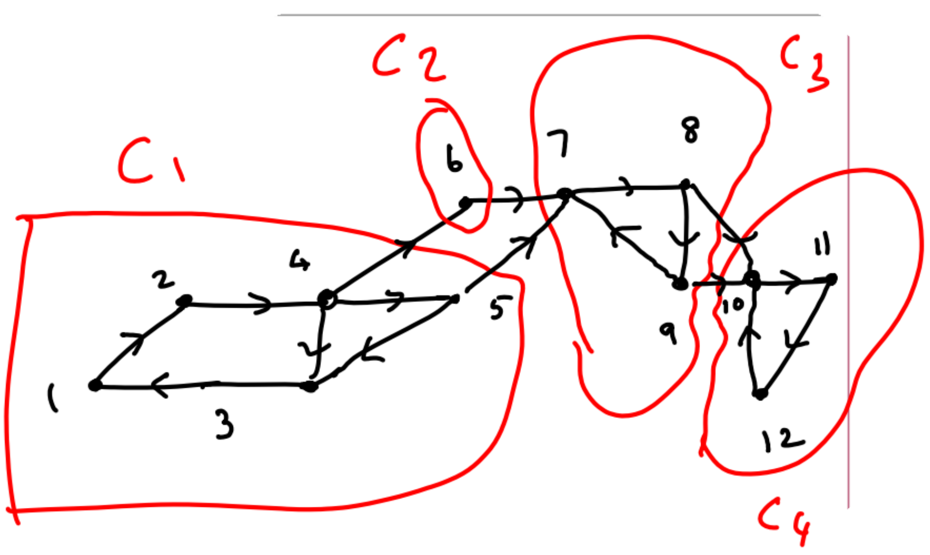
Department: Computer Science and Engineering

Institute: IISc

Week: 08

Lecture 35 Strong Connected Components Part 3

Namaskara we have seen a high level description of the strong connected components algorithm we will see the way in which it works. We will understand the working details of the algorithm by running that algorithm on this example, okay. So what we do is we have to find the discovery time and finish time of all these vertices performing a DFS. That is what we have to do, so consider this graph, that is given here,

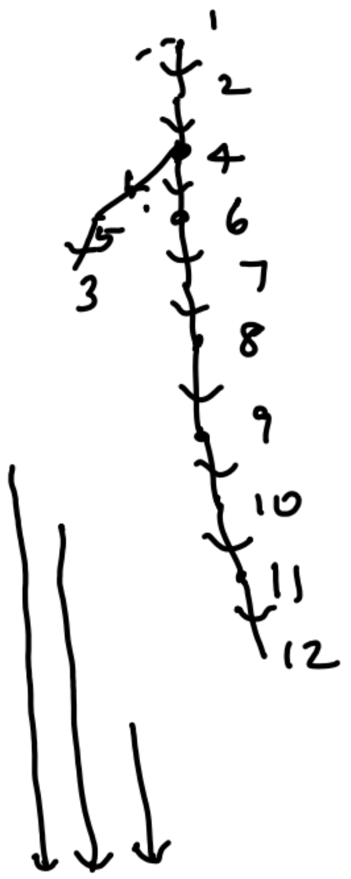


run DFS and find finish time find discovery time and finish time for every vertex v okay. So, let us do that so vertex v and discovery time v dot d and v dot f; v, v dot v, v dot f.

Let us say we are starting at vertex 1. We assume that the adjacency list has got all the vertices, all the neighbors labeled in increasing order. For example, 1 has got only 2 as a neighbor, right, outgoing. So that will have, if you look at 4, 4 has got 3 neighbors, 3, 5 and 6, so adjacency list of 4 will have 3, 5 and 6, listed in that order. If you look at 8, 8 has got several neighbors 9 and 10, so 8 will have its neighbors 9 and 10 listed in the increasing order okay. So, in this way each vertex and their neighbors are listed in increasing order, so we are going to do DFS looking at the adjacency list alright. You start from 1, v dot discovery time is 1, v dot discovery time is 1. From where, from there you have to go to an unvisited neighbor, 2 is the unvisited neighbor, so you will go to 2, the next vertex to be visited will be 2 and we will, 1 2 3 4 5 6 7 8, these are the vertices, so from 1 you are going to go to 2, therefore discovery time of 2 will be 1 and that is only neighbor it has got from 2 you are going to 4, sorry the discovery time will be 2.

2, from 2 you are going to 4. So, let us draw also from 1 you had gone to 2, from 2 you had gone to 4, the discovery time for 4 will be 3, and from 4, 4's adjacent neighbors list are 3, 5 and 6, first from 4 you will go to 3, 3 is not visited yet, so from 4 you go to 3, so discovery time is the next time. So, this is the fourth vertex discovered, as you can see 1, 2, 4, 3, discovery time of 3 will be 4 and from 3 you will see whether there is any unvisited neighbor, 1 is a neighbor but 1 is already visited. So, since all neighbors of 3 are visited, you are going to exit 3, from 3, actually this will be a 3 to 1, this is a non tree edge, that is the only neighbor, that is a non-tree edge and all neighbors of 3 are visited, you are done with the 3 therefore you will now finish with the 3 okay.

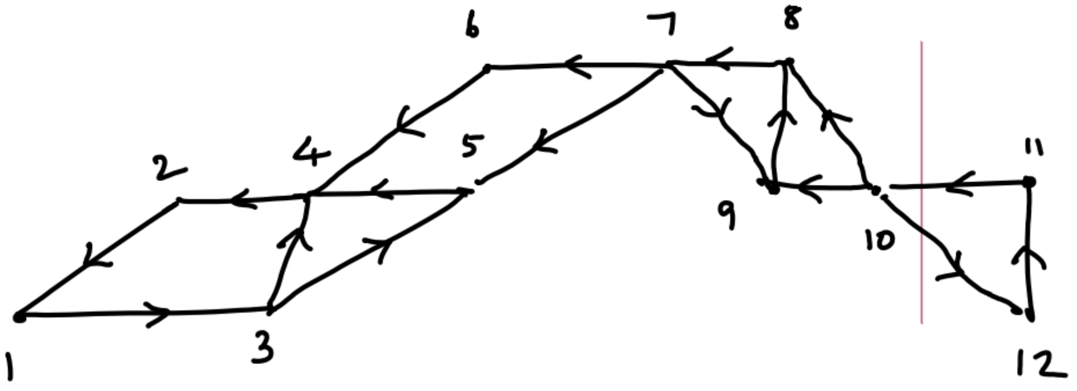
v	v.d	v.f
1	1	24
2	2	23
3	19	20
4	3	22
5	18	21
6	4	17
7	5	16
8	6	15
9	7	14
10	8	13
11	9	12
12	10	11



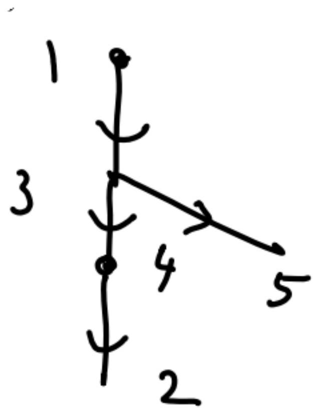
Oh I am sorry in this example from 4 it had not gone to 3, has gone to 6 my apologies, forget this, this is not necessary. Now there is a systematic, way but the way in which this example from 4 we had not gone, I just have to mild backtrack from 4 you had gone to 6 from 4 not to 3.

From 4 you had gone to 6, 1 to 4, 4 to 6 you had gone, so 6 will have discovery time 4 4 as a discovery time 3 the next discovered vertex was 6, you can see that it has gone to 6, from 4 you had gone to 6, from 6 you had gone to 7. So, the next vertex discovered is 7, which is the discovery time will be 5, discovery time for 7 is 5. discovery time of 8 is 6, the next vertex discovered is 8, therefore 6 is the discovery time for 8, from 8 you had gone to 9, from 8 you had gone to 9. So the discovery time for 9 is 7 and from 9 you had gone to 10, therefore discovery time for 10 is 8. From 10 you had gone to 11, therefore the discovery time for this the depth first search. 11 from 11 you had gone to 12 therefore discovery time for 12 is 10. Now all neighbors of 12 are visited, 12 has got only one neighbor namely 10 and 10 is already visited so you will finish 12, finish time of 12 will be 11. You backtrack, you had come from this you had gone to 12 you backtrack to 11 and at 11 you have no other vertex, so at 11 also you will finish, now you finish 11 at 12, the finish time of 11 will be 12 and then you backtrack to 10, the finish time of this 10, because 10 has got no other neighbors, 10 has got 11, 11 is already visited therefore, no other neighbors of 10 are available. So, you from 10 you backtrack to 9, at 9 also you have 7, which is already visited 10, that is already visited, all neighbors are visited. So, you finish at 9 and you backtrack to 8 and 8 also you have no other, 9 is visited 10 is visited all neighbors are visited. Hence you finish at 8, finish time of 8 is 15, you backtrack to 7, And at 7 there are again no unvisited neighbors, it will be 16 and you come back to 6 and 6 also have no other neighbor, so that will be 17, from 6 you come back to 4, right from 6 you come back to 4, this is 4, 4 has got several unvisited neighbors, okay, 4 has got, 3 is not visited, 5 is not visited and it is going to take 5 now okay from 4 you are going to visit 5 now. from 4, 5 is not visited so from therefore the discovery time for 5 is 18 and from 5 there is an unvisited neighbor 3, so discovery time for 3 will be 19, okay this just now 3 is discovered, from 3 you try to go to 1 but 1 is already visited, so from 5 you had gone to 3, 3 is visited, its visit is complete, so at 3 you visit, you finish the visit, therefore finish time is 20, you backtrack to 5, just 5 have any other unvisited visit vertex no, so 5 will finish, now 21 you come back to 4 and when you come back to 4 is there any unvisited neighbor. Now 4 has got 5, 3, 6 all of them are visited so you will finish at 4 also and at 22 and then you are going to backtrack to 2, at 2 all neighbors are visited therefore you finish at 2, at 23 the next time slot, then you backtrack to 1, 1 has no other neighbors so you finish at, these are the finish times. So discovery time and finish time of all the 12 vertices are determined Step 1 just perform the DFS on the input graph and identify the vertices that are, this is step 1 okay.

Now we have to construct the reverse, okay, each edge is going to be reversed, okay, so let us write down the same graph with each edge reversed. So I need a so construct G reverse, it is very easy to construct okay for every edge you generate the reverse edge you create a adjacency list for reverse graph that is all. So I will draw pictorially here, but in algorithm you can very easily do that each edge you just scan through the adjacency list just put the reverse of it in the another fresh adjacency list you are going to create a new adjacency list for G of R, construct G of R how do you do that, construct new adjacency list for G of R, this is how you will do it in the program but here I will draw the picture. Earlier you had an edge from 1 to 2, but that edge reversed will become an edge from 2 to 1. Earlier you had an edge from 3 to 1, now that will become an edge from 1 to 3. Earlier you had an edge from 2 to 4, now that will become an edge from 4 to 3. Then this reverse will have this, vertex 5 and 3 to 5 ok. So, 3 to 5 will be, then the 5 to 4 will be the edge and 6 to of the edge 4 to 6 here you will have 6 to 4 instead of edge from 6 to 7 you will have the edge from 7 and then 7 to 5, earlier you had an edge from 5 to 7 for example let us see, here you can see that edges from 5 to 7, in the reverse it will be from 7 to 5 you can see that here I have drawn from 7 to 5 and now you have 8 to 7 instead of 7 to 8 and instead of 8 to 9 you will have 9 to 7, you will have 7 to 9 and then 10, instead of 9 to 10 you will have 10 to 9, instead of 8 to 10 you will have 10 to 8. So, this is how the reverse, and then 10 to 11, now you will have 11 to, this is vertex 11 and 12 to 11 and 10 to 12 okay. So, this is G reverse every edge I have just reversed it okay.



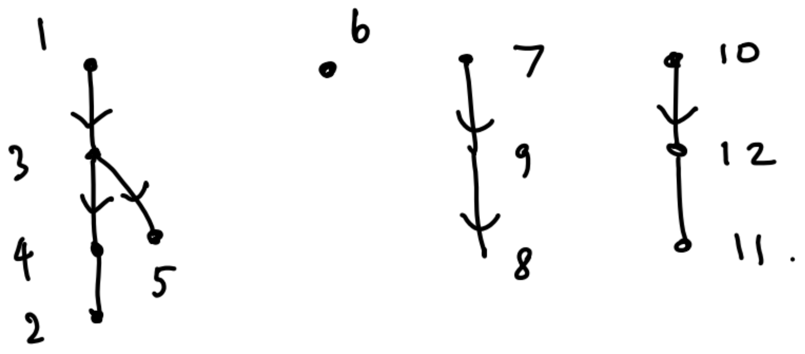
On G reverse I am going to do depth first search okay. Third step perform DFS on G reverse but how are you going to perform DFS, where will you start and how would you proceed? You would always start with a vertex with maximum, an unvisited vertex with maximum finish time okay. Perform ,how are you going to perform, start always with an unvisited vertex with maximum v.f, so where do you begin with. Now no vertex is visited, right no vertex is visited, so which vertex has got the maximum. So here you see the maximum finish time is 24 and that is with vertex 1, therefore max finish time is 24 and that is for vertex 1. So, start DFS from 1 on G reverse, so start the DFS from 1. So when you start the DFS from 1, from 1 where can you go, there is only one neighbor you go to 3, okay so from 3 you have several options, you can either go to 4 or you can go to 5, let us say you had gone to 4. From 4 you can go to 2, that is the only option you have right, from 4 you can go to 2, from 2 you can go to 1, that is already visited, so backtrack to 4. Now all neighbors of 4 are visited, backtrack to 3 and when you backtrack to 3, now from 3 you can go to 5, but from 5 you can go to 4, but that is already visited and there are no other neighbors therefore you backtrack to 3, at 3 all neighbors are visited therefore backtrack to 1 and 1 is the route that is where you have started okay. So the DFS stops at this point, the DFS has stopped this point because DFS has visited all the vertices reachable from 1, all the vertices reachable from 1 are already been reached therefore the DFS has come at 1 and then it has stopped,



This set of vertices 1 3 4 5 2 is a connected component of original graph right, which you can see 1 3 4 5 2 a connected component, strong connected component of G. I worked on G reverse and when I worked on G reverse, I started a DFS and the DFS started at 1 and it has visited all the vertices that are reachable but then it has not visited all the vertices, I have to continue my DFS where do I continue, from which unvisited vertex I should continue, here is the logic start always with an unvisited vertex with maximum vf ,so 1, 2, 3, 4, 5 are all visited among the other vertices which one has got the maximum one. So go back let us see the so 1, 2, 3, 4, 5 are already visited okay. Among the remaining which one has got the maximum finish time, so all these things are, Yeah, 6 as the maximum finish time you can see that these are the finish time of the remaining vertices, 17 is the maximum finish time. Therefore, I have to start at 6, among the remaining vertices 6 has maximum finish time, namely 17, hence continue the DFS on G reverse from 6, okay, you should start from 6, 6 has got one neighbor, which is 4, that is already visited it does not have any other neighbor, therefore when you start at 6, it is just stopping there, that is all, the DFS second, I mean continuation of the DFS is at 6 but from 6 the tree does not grow right therefore this is the one and hence this is the second tree okay. This is the second tree in the DFS this has got only one vertex therefore this is the second component. This is a strong connected component of G, 6 is a strong connected that single vertex 1, 2, 3, 4, 5 is a strong connected component, 6 is a strong connected component you can see that in the original picture you can see C1 is 1, 2, 3, 4, 5, C2 has got only 6, that is a connected component okay, let us continue.

So among the remaining vertices, I have to start, I have to continue my DFS I have to continue the DFS until I visit all the vertices I started at 1, I reached all vertices reachable from 1, I got stuck again I started from 6, I got stuck there, again I have to continue. I have to start from one of the unvisited vertices but then our policy is always start from the unvisited vertex with maximum vf, therefore let us look back, even 6 is gone and these are the remaining one 16 to this these are the remaining one. The largest one is 16, therefore we start from 7, so start from 7, 7 has got 2 neighbors, 6 is already visited, 9 is another option so you go from 7 to 9, from 9 you go to 8, from 8 there is only one neighbor 7 that is visited, all neighbors are visited backtrack you come to 9 all neighbors of 9 visited, you backtrack you come to 7, all neighbors of 7 there is another neighbor 5 that is also visited. So at 7 also the visit is complete therefore this is another connected component 7, 9, 8 is another, is a strong connected component of G. strong component of G. So you start from 7, among the remaining vertices, 7 has the maximum finish time, start from 7 and you get 7, 9, 8. So everything is visited, among the remaining vertices 7, 8, 9 is all 10, 11, 12 is there so you start from 10.

So when you start from 10. you go to 12 and 12 to 11 and 11 has 10 as a neighbor, that is visited and backtrack 12 and this is visited done so 10. The DFS on GR is now complete. The DFS on G reverse is now complete, the DFS, the depth first search has generated a forest with 4 trees, okay, first tree has got root at 1, how was the tree looking like 1 to 3 to 4 to 2, 1 to 3 to 4 to 2 and 3 to 5 this is one forest another forest is only 6. another sorry another tree is only 6, another is 7, 9, 8, another tree is 10, 12, 11.



So the DFS on G reverse has generated 4 trees each set of vertices will form a connected component 1 3 4 2 5 6 7 9 8, 10, 12, 11 each one of them is a strong connected component of G and that is what we have seen in the picture right therefore this works.

$$\underline{\underline{\{1,3,4,2,5\},\ \{6\},\ \{7,9,8\},\ \{10,12,11\}}}$$

A SCC of G.

But we have to prove the correctness of this method right this is how it works, so, on GR we are going to work by exploring the vertices in a specific order. Whenever you perform DFS if there is still an unvisited vertex, in general, you can choose any unvisited vertex and continue your DFS, here we are continuing in a specific way alright. So to summarize, perform DFS on G and determine v dot f for all v belonging to V, construct G reverse. perform DFS on G reverse and generate the DFS forest. Each tree has vertices defining the vertices of strong connected component of G. Generate the DFS forest okay. In 3 we always start DFS from a vertex with highest v dot f, from a unvisited vertex. So in general DFS, you start from a vertex it will visit few vertices and get stuck if there are still some vertices the general DFS puts no constraint, you can continue the DFS from any unvisited vertex, but for this particular algorithm we are going to choose that unvisited vertex which has got the highest v.f, then do that it is still whenever you are stuck and if there are still some more unvisited vertex, we are going to continue our DFS from one of the unvisited vertices and then reach all the nodes that are reachable from that and so on. So in this way a DFS forest will be generated each tree will have certain vertices the DFS forest is generating a partition of the vertex set that partition is nothing but this strong connected component partition.

This example clearly shows the way in which it works. In our next session we will see a very simple and elegant proof of correctness of this method, how this works, why it should work, you do a DFS and you got a just do DFS you get for every vertex a finish time and then the start time. Then you constructed reverse and on the reverse you have done a particular way of depth first search and you got it. The complexity is very simple, it is linear because you do only 2 DFS and then you construct reverse. Reverse can also be constructed in linear time for each edge you have to produce its reverse and get GR.

So overall complexity is linear, so in linear time we have an algorithm that is determining the connected components of a directed graph, okay, very interesting algorithm we will see the proof of correctness in our next session thank you.