Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

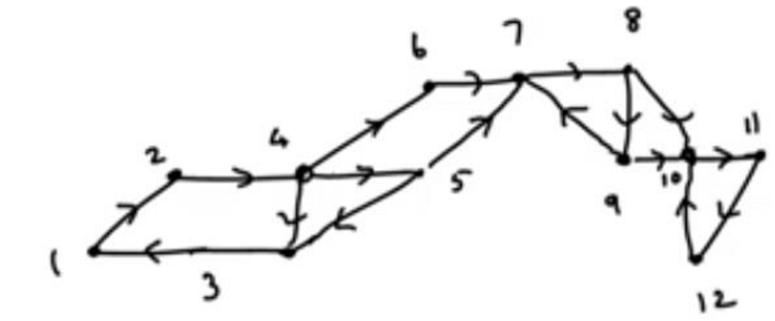
Department: Computer Science and Engineering

Institute: IISc

Week: 08

Lecture 34 Strong Connected Components Part 2

Namaskara what we have seen is when a graph is given whether it is strongly connected or not. The graph is strongly connected, we do not have to do any further stuffs. But if the graph is not strongly connected, it will consist of several strongly connected components. and we have to identify those components. Let us see an example and then figure out what are all the things to be done with the input graph okay. Let us consider this figure, let me write down the example vertex 1 and this is vertex 2, vertex 1, 2, this is vertex 4, 2, 6 and this will be vertex 3 and vertex to write it more beautifully. Let us put vertex 3 here and vertex 5, 5 to 3 and 6 to 7 and 7 to 8, 8 to 9, 9 to 7 and again 10 11, 9 to 10, 8 to 10, 10 to 11, 11 to 12, we write 10 little assigned this is vertex 10, this is vertex 10. So here is a graph with 12 vertices and this graph offers us a lot of insight into the way in which the algorithm works okay.

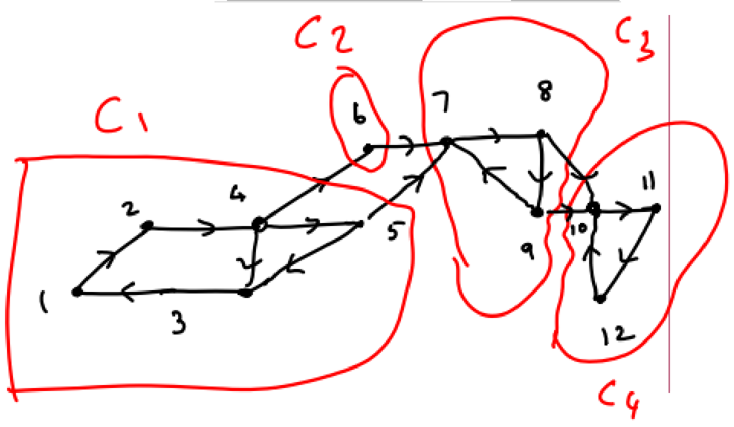


Now if you see the connected components strongly connected components. if you have a path from a to b and back b to a that means we will be able to find a kind of a cycle passing through these two vertices a to b, b to a.

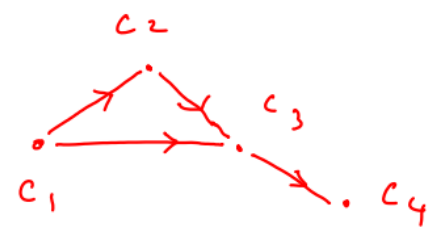
So the path may be different but we should be able to find a kind of a so if you see a big cycle 1 to 2, 2 to 4, 4 to 5, 5 to 3, 3 to 1. So if you find a cycle all vertices in the cycle are in the same component some basic intuition all vertices in a cycle will be in the same component that is because all vertices in the cycle are mutually related. because you can go from if there are 2 vertices in a cycle u and v from u to v you can go and from v to u also you can go along the cycle.

So the paths exist therefore they are related and this is true for all vertices in that cycle they are all mutually related that means they will all be put in the same component right. So you can see that 1, 2, 4, 3, 5 is in a component, so let us say I change this to this is in one component and 7, 8, 9 that will be in one component because 7 to 8, 8 to 9, 9 to 10 But once you go from 8 to 10 there is no way you can come back from 10 because the edge 9, 10 is going the other way right.

Therefore from 7 to 8 you can go 8 to 9, 9 to 7 but you will never be if you go anywhere else you cannot come back, therefore this will be another component. This vertex 6 by itself is a connected component because it is not related to anybody else okay. So it is an isolated vertex single vertex, this is a component and 10, 11, 12 that is another component, okay this also you could see that as. So call this as C1 C2, C3, C4, in this graph there are 4 strongly connected components,



Draw vertices, assume that all the vertices of the component are collapsed call that as C1, a vertex then C2, C3, C4 you can see that from C1 there is an edge going to C2. So I will draw this edge, from C1 also there is an edge going to C3, so I will draw this edge. From C2 there is an edge going to C3, so I will draw this edge and from C3 to C4 there is an edge, 9, 10 is going from this cluster to that cluster, from one group to another group multiple edges may be there, it does not matter, if there is an edge you draw that in the. So this is the condensed graph, if this is G this is the condensed graph of G, this is a DAG,



Condensed graph of G.

$\rightarrow$ DAG

This must be a DAG if there is a cycle then all the components in the cycles can be merged to form a super component therefore that would not be the case that is the reason why condensed graph is always a DAG. So we have to identify a strongly connected component means, we have to identify these set of vertices. So 1, 2, 4, 5, 3 is a set of vertices, it is a partition of the vertex set okay, 6 is and 7, 8, 9 and 10, 11, 12. These components we have to identify these are the strongly connected components.

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

How do we represent a strongly connected component? We will give a component number. Okay, so the component number for all these things component number will be 1, for all these things component number will be 2, this is 3, this is 4, connected component number 1, connected component number 2, so there are 4 components. How do we make this association we define an array, SCC, so for example SCC2 will be 1, SCC7 will be 3 because it is in the third connected component. So we will have a SCC array indexed by vertex set, SCCV is an integer and it will indicate the component number okay it is an identifier component number is just an identifier. So how do you get all vertices in a component, get all the vertices with the same SCC number, For example for 2, for 1, for 4, for 5, for 3, for all of them SCC will be 1. So when you collect all these vertices that will form a connected component. The name of the connected component is 1. So in this way we can have a connected component represented through an array. Okay, so if SCC of i equal to j, that would imply that vertex i is in the connected component j. So we are assuming that the vertex set is 1, 2, 3 up to n okay

$$V = \{1,2,3,4 \ldots n\}$$

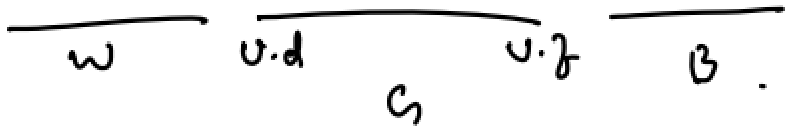
and the component numbers, strong component, component numbers are again 1, 2, 3 whatever is the connected number of connected components.

In our example we had 4 connected components therefore connected components will be numbered from 1 to 4 which are all the vertices in connected component 1 which are all in 2, 3 like this we will be able to identify. Our algorithm should identify that therefore the output for the algorithm is the SCC array, input G equal to VE, output SCC array for each vertex in which component it is in, that information you have to produce, that is the goal okay.

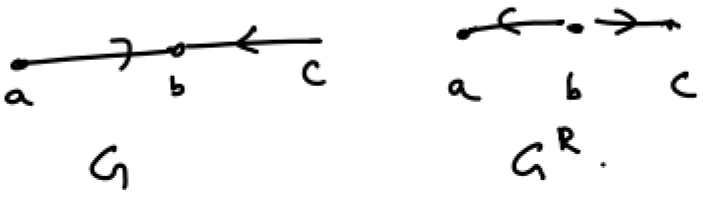
$$\text{Input} = G = (V, E)$$
$$\text{output} = SCC[\,]$$

So this would identify the strong component. So we do this in the following way, we will see the reason and other thing for it alright ,we will see proof of correctness of whatever we are doing. So that we do the following alright to identify the strong component.

On G perform a DFS, you can start the DFS from any vertex it does not matter, perform a DFS on G. So when you perform DFS for each vertex we have a discovery time and finish time. So, for each v belonging to V, we have v dot d discovery time and v dot f the finish time. Discovery time is when the DFS enters into that vertex for the first time. vf is the time stamp when it exit v and after that it will never return to v okay. So recall the DFS algorithm the way in which it works, so each v will be colored white. we will have white color up to some point from the initiation of the algorithm, from the start of the algorithm up to some point it will be white, then the discovery time it becomes gray then it will remain gray until finish time, after the finish time it will be black white gray and black. Before discovery time it is not even entered initially all vertices are white. In the exploration v is entered at a particular time, that is time stamped v dot d is that time and at the time it is set to grey okay and you do the exploration and v dot f and after that it is black, this is how the DFS is performed okay. So on G perform DFS and identify v dot d and v dot f for every vertex.



Second step okay, construct G reverse. What is G reverse, for every u flip that edge, that will become reverse, so G reverse and G will have same number of edges but every edge of G will be reversed okay. For example if abc if this is G how will G reverse will look like it will have the same vertex set, ab will be reversed abc, ab when you reverse you get ba bc will be reversed when bc is reversed you get when cb is reversed you get bc so this is G reverse.



So construct G reverse, okay. The third step is in G reverse perform DFS in the decreasing order of v dot f, the finish time, you start from the vertex which has got the highest v dot f, go through DFS all the reachable vertices will be obtained and then it will get stuck, you have to start again and where do you start again among the remaining thing the one from the highest f vertex, v dot f vertex, again from there you proceed you would visit all the vertices, some of the vertices. You might not have visited all the vertices, if that is a case again start from an unvisited vertex. But which vertex there may be several unvisited vertex okay, you must always start fresh DFS from a vertex with the highest v.f okay perform DFS in decreasing order of v, that is start every fresh search from an unvisited vertex with highest v dot f, highest finish time as done in step 1, as done in step 1 okay alright. Now when you do this, this is step 3 okay.

You have constructed GR, in GR you have performed a DFS in the decreasing order of v.f okay. Output of the vertices of each tree obtained in step 3 as component vertices. That is all it is done. So when you perform DFS, you start from a vertex, all vertices reachable will be explored there but you might not be, you might not have visited all the vertices all reachable vertices you would have accounted for, but in the graph still there may be some vertex and you have to make a fresh start and when you make a fresh start, you again from that vertex you will reach several vertices.

So, that way your DFS in general will consists of several trees, reachable from the starting vertices, every time you freshly start ok. If you have not visited all the vertices, you continue your DFS by starting from an unvisited vertex, okay, so put in a informal way. Start DFS from a tree from a vertex sorry, vertex v, this stops after visiting all vertices reachable from v, so this will be a tree. like this.

This stops after visiting all the vertices in a tree this will be a tree rooted at v, if this tree does not contain all vertices of G, we continue the DFS from another unvisited vertex, say u and then from u you continue your DFS and then this will be like this. This lead to a tree rooted at u, again if not all vertices are visited, not all vertices are covered, you start from one of the unvisited vertices say w

and from there you reach maximum number of vertices if all vertices are visited it is fine otherwise you have to continue, therefore in general DFS will result in several trees each rooted at a vertex and consisting of all vertices reachable from the root. This is how when the DFS ends you will have this, if all vertices are reachable then it will be a single tree. If not all vertices are reachable, DFS will result in several tree, that is the reason why the tree edges though we are calling as tree edges what are the edges, parent of v this set of tree edges form a forest in general, you will actually have a forest that means it will have several trees this is what we have to obtain.

So the tree vertices form a partition of the vertex set, the tree vertices form a partition of the vertex set, therefore DFS forest, in general DFS will result in a tree if all vertices are reachable from the vertex you started but if that is not the case, it will consist of several trees and that will form a DFS forest okay. So now this DFS forest that we get on GR are the components okay. We note that the components induced by DFS forest, DFS on GR are the strong connected components of G, this is the point that we have to prove. So we will see the procedure and how it works and then we will explain and then prove the correctness of the algorithm okay.

We will continue with the working of the example in our next session thank you.