Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

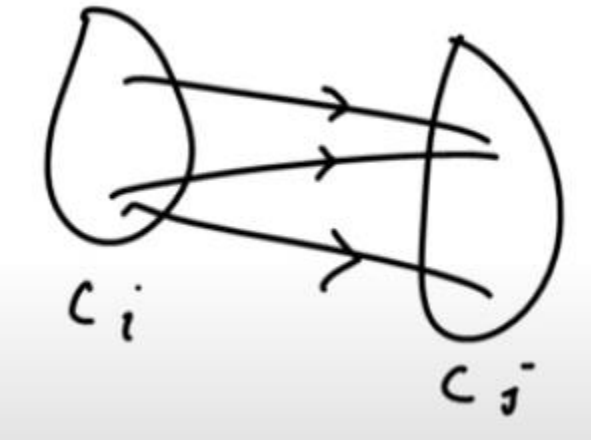
Week: 08

Lecture 36 Strong Connected Components Part 4

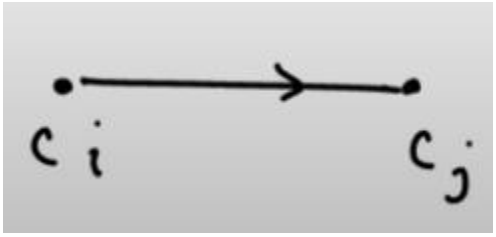
Namaskara, we have seen an interesting algorithm for finding the strong connected components of a directed graph. We have seen it is working through an example, we have used depth first search twice, once on G, again on G reverse and then finally obtained the partition corresponding to the strong connected components. Now we need to prove that it indeed works that is because it is not entirely obvious, how by processing the DFS in a specific order, that too in G reverse is giving the partition, is not clear right, it is not obvious or direct. Take a closer look at the proof it is a very interesting proof it is simple and insightful and one of the most elegant and wonderful proof. And this shows that the proof of an algorithm as an independent entity has got its own mathematical elegance, beauty, purpose and so on okay computation we can do. It is a procedure we can specify do this computation and that and so on and then you can give a description of the algorithm. Sometimes from the description of the algorithm its correctness is not clear. In fact the algorithm is derived based on these type of facts and observations. So if you do this computation we would get that kind of a confidence the designer gets. So, the discovery process is basically you build a mathematical properties and then in line, in conformity with those properties do some computation. So, that whatever is guaranteed by those properties is obtained by the algorithm, by doing those computation. So, they go hand in hand with one another, in certain simple algorithms, the proof will be direct and obvious but this is a case where it is pretty subtle okay.

So, we will see SCC, strong connected component proof of correctness. So first of all from the definition, we see that the strong connected component of G and strong connected component of G reverse are same, that is because the way in which the strong connected component is defined is for every pair there must be a path from u to v and v to u if there is a path from u to v in G there is a path from v to u in G reverse if there is a path from v to u in G there is a path from u to v in G reverse. The same definition works and the same connected components you will obtain when you work on G reverse as well. And we have seen that the condensed graph, it gives an interesting property of these components, okay.

The condensed graph is obtained in the following way, shrink all the vertices in a strong component to a single vertex and then if there is some edge between some vertex in one component to some vertex in the another component, in the condensed graph draw just an edge, okay so this may be a component and this may be a component, this may be Ci this may be Cj



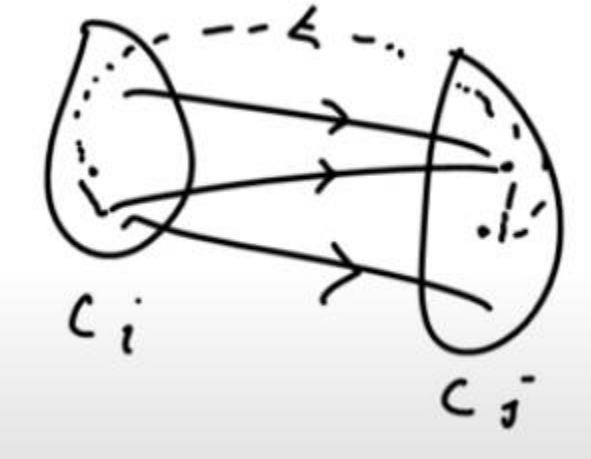
and there may be edges like this but when you condense you get Ci and Cj.



Notice that the edges can be only in one direction, you cannot have edges in both direction okay. That is because if you have edges in both direction, it will create a cycle and connection between all vertices in both directions okay. For example, if you can imagine some edge like this suppose I put that in a dotted one suppose there is an edge in the opposite direction.

Now from any vertex to any vertex you are going to have paths for example you want to go from here to here. right it is simple from here you can perhaps come here take an edge and reach here from here you can come over here right. So, you see that there is a way to go from one to another for the opposite direction you go from here to here take the opposite one and from there you can come over here. between all pairs of vertices paths exist. Therefore if there is an edge in the opposite direction it will merge Ci and Cj into one big super component, if there is any edge from Cj to Ci it would merge Ci and Cj to a big strong connected component, Ci union Cj, that is not a case, there is one component Cj is another component their union is not a component therefore there cannot be any

edge in the opposite direction you can go only from one component to another component one way there is no cycle okay.



That is the reason why the condensed graph is a directed acyclic graph, this is the major reason why condensed graph is a directed acyclic graph because of this reason okay, now let us derive one important property of the vertices in a connected component, we have to identify the vertices in a connected component.

So let fc equal to maximum of v dot f v belong to C.

$$f(c) = \text{Max}\{v.f \mid v \in c\}$$

So, C is a connected component it has got vertices each vertex has got a finish time, maximum finish time, among the vertices in a connected component, and discovery time earliest discovery and dc equal to minimum of u dot d, u belong to C

$$d(c) = \text{Min}\{u.d \mid u \in c\}$$

Each component has got vertices each vertex has got a discovery time and finish time. Let fc is the maximum finish time the one that has finished in the last and dc is a earliest discovery time, it is a discovery time of the first vertex in that group of vertices, in that set of vertices. Let fc and dc be defined in this way, then we have the following interesting lemma, if Ci and Cj are two strong connected component of G and if there is an edge from Ci to Cj, that is from some vertex in Ci to some vertex in Cj, that is what we mean by an edge from Ci to Cj, then finish time of Ci is greater than finish time of Cj.

$$f(C_i) > f(C_j).$$

So here is Ci, here is Cj, there is a vertex, there is an edge from Ci to Cj, we will consider 2 cases, 2 kinds of edge connections, okay.

Case 1: dCi is less than dCj, that is the vertex in Ci has got is discovered earlier than the vertices in Cj.

$$d(C_i) < d(C_j)$$

The first vertex, let us say u let u be the vertex with minimum d value. that is u is the first vertex in Ci, that is discovered by the DFS, when you perform the DFS this component has got several vertices, there is a vertex that has been discovered for the first time, let u be that vertex okay after entering u, what happens, after entering u, DFS reaches all vertices reachable from you and they will all be in the subtree rooted at u. All vertices reachable from u are in the subtree rooted at u because from u entered u and then u visited then u backtracked and then u. So, in this way if a vertex is reachable that will be in the subtree after it reaches all, all vertices reachable from u are in the subtree rooted at u in the DFS tree, it will be like this okay. Notice that C1 is a strong connected component that means all vertices here are reachable from you.

There is an edge from Ci to Cj some edge so through that edge it can reach vertices in Cj. Cj vertices form a strongly connected component, so any vertex here is reachable from this. Therefore the hence all vertices of Ci union Cj are reachable from u, all vertices of Ci union Cj are reachable from u. They are all here, therefore in the subtree Ci union Cj is in the subtree. So it will visit every vertex and then when it is finished it will backtrack.

Therefore du, when do you exit this and go to its parent, what is the finish time of u , finish time of u will be much later than finish time of all the vertices in the subtree because this start and finish earlier and backtrack and so on. And after visiting everything you backtrack from u, therefore u.f is greater than let us say v.f for all v belong to Ci union Cj, that means u.f is greater than f of Cj, because Cj is, here all vertices of Cj are here in the subtree, under maximum is this and this is larger than even that, u dot fs.

$$u.s > u.f \quad \forall u \in C_i \cup C_j$$
$$u.f > f(C_j)$$

If the discovery time if the finish time of u, u itself is larger there may be some other vertices with a higher finish time if that is the case f of Ci is greater than or equal to u dot f which is greater than f of Cj.

$$f(C_i) \geq u.f > f(C_j)$$

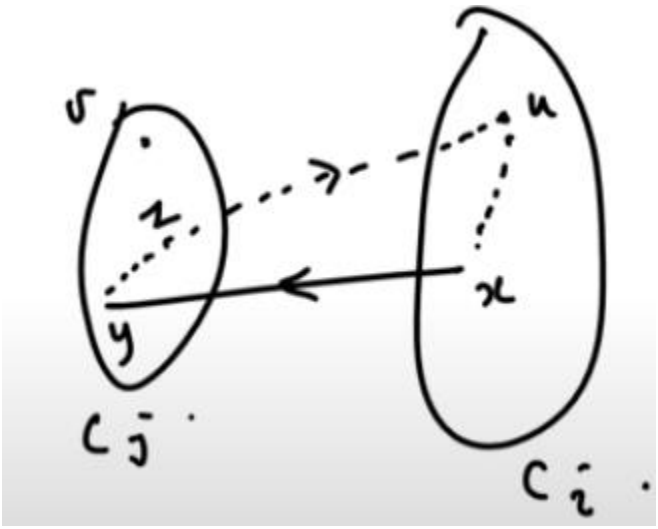
Because u belongs to Ci, u belongs to Ci therefore f of Ci is greater than or equal to u.f which is greater than f of Cj that means f of Ci.

$$f(C_i) > f(C$$

Case 2: In this case we have considered d of i is less than d of Cj right now we are going to assume d of i greater than d of Cj, d of Ci is greater than d of Cj.

$$d(C_i) > d(C_j)$$

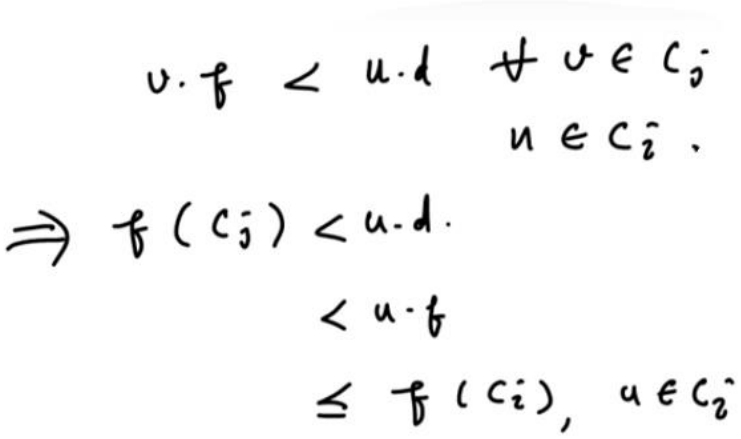
So, you Cj is here there is some earliest vertex v in C j and then you have d Ci there is a vertex u with d Ci. d of Ci is greater than d of Cj.



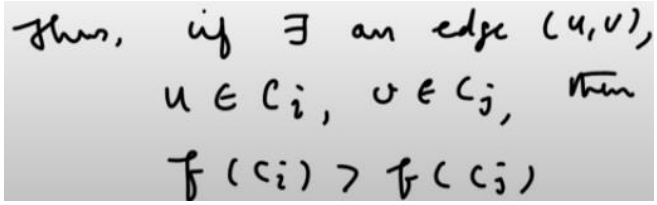
In this case we show that there is an edge here, let us say xy there is some edge from Ci to Cj. In this case, we note that no vertex of Cj can reach u, u in Ci, if any vertex can

proof by contradiction, we have seen already this proof is by contradiction, if there is a way to reach u, let us say u if there is a vertex that can reach u that path u to x, path y to so if there is a vertex z proof is by contradiction let z be a vertex which a path from Z to U then that path plus path from u to x, the edge xy and the path from y to z, it will, then this union contains a cycle and that means u is in c j right.

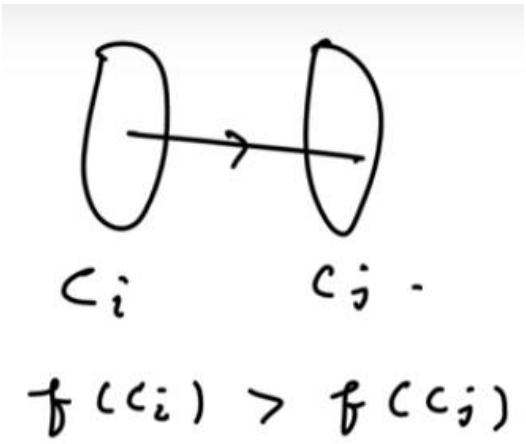
From every vertex in Cj, I can reach u, from u you can reach every vertex because of this cycle that means u is included in C j and this is a contradiction. The reason is Ci intersection Cj equal to empty, you cannot have u in Ci and u in Cj that is not possible Therefore thus no vertex of Cj is connected to u. So in the reachability of all these things u will never be there. Therefore all these things when they are done, they are going to, at the time the color of u will be white, it is not even reached. So you start with v, you explore all the vertices and in that exploration of the vertices u will be there always, because u is not touched there is no path. So no vertex of Cj is cannot, this implies DFS starting at v will never reach u, it will never reach u. therefor f of therefore v dot f is less than u dot d for all v belong to Cj and u belong to Cj. This implies f of Cj is less than u dot d, f of but u dot d is less than u dot f u dot f is less than or equal to f of Ci that is because u belongs to C.



So, in both cases we have shown that f of c j is thus, if there exist an edge u v with u n Ci and v in Cj, if there is exist then f of C i is greater than f of C j okay.



Pictorially there is an edge

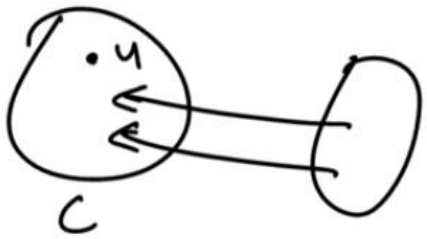


Now we use the contrapositive, if, what is the negation of this f of Ci less than f of Cj,

that is because f of Ci will never be equal to f of Cj, they are in different so the negation of this is this if this is the case if this is the case then there is no edge from Ci to Cj there is no edge. from Ci to Cj okay this is what we are going to use how we use it.

So start DFS in G power R with the vertex with highest finish time. Let u be such a vertex and C be the strong connected component containing u, let Cj be any other strong connected component of GR, strong connected component of GR is same as strong connected component of G. We start with the highest finish time therefore, f of C is greater than f of C j why this as a highest finish time.

So, the finish time of C j will be smaller than this, since there is no edge from Cj to C, there is no edge from C j to C in G. Therefore, there is no edge from C to Cj in G reverse. C j to C no edge means you reversed it. So, from C to C j there is no edge, hence the DFS starting at u will not go outside of C, you are starting with, this is C, this is the vertex with one there is no edge from C to any other one, if there are any other one, the edges are like this so they are not useful.



So from u when you start you cover all the vertices reachable, all vertices reachable will not go, hence the DFS starting at u visits all vertices of C and no other vertices of G, it will not go anywhere, it will remain there itself, this implies the spanning tree rooted at u as all vertices of the strong connected component and only these vertices are in the tree. Hence the forest generated by DFS in GR, hence the forest defines the strong connected component of G, okay. So you have a partition of, when you perform DFS you get a forest. So forest will have several trees, each tree will correspond to a strong connected component, done okay, hence the forest generated defines in what way each tree of DFS forest would correspond to one strong connected component of G. This is how therefore our algorithm works, the complexity is linear okay, construction of GR complexity order n plus m for constructing GR, again order n plus m for the two DFS we did in the algorithm that is all.

(O (n+m) for const G^R

Therefore, total complexity order n plus m, n is the number of vertices and m is the number of edges.

Total Comp (O(n+m)

So this completes the discussions on the linear time algorithm for strongly connected components. We have done this on a directed graph. We conclude the discussions on this session at this point. Thank you.