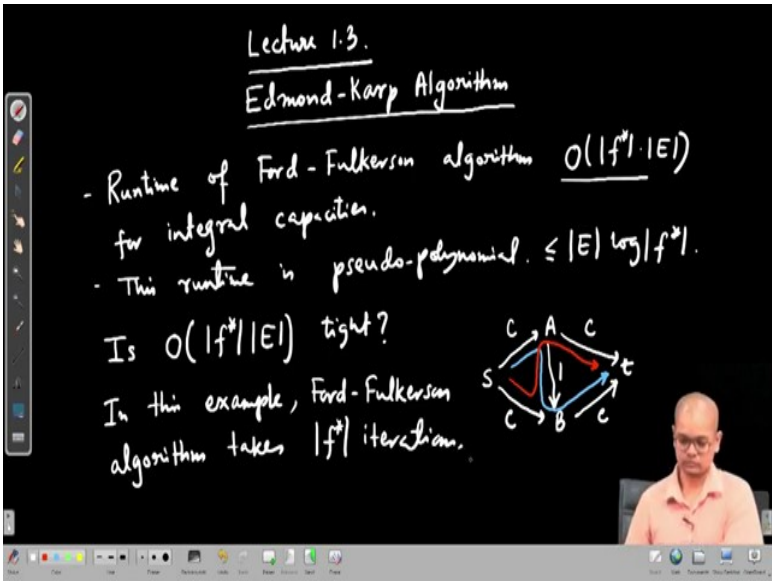
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology - Kharagpur

Lecture - 03
Edmond-Karp Algorithm

Welcome, so, in the last lecture we have seen the Ford-Fulkerson method for computing maximum flow, so, we will continue from there.

(Refer Slide Time: 00:37)



Today, we will see Edmond-Karp algorithm for computing Max flow. So, let us recall the runtime of Ford-Fulkerson algorithm is big O star sorry big O of value of Max flow times number of edges for integral capacities. So, this is not a polynomial time algorithm. So, this runtime is pseudo polynomial because it the runtime is proportional to the size of the maximum flow.

But the inputs are these numbers and whenever you have a number, the number of bits to represent that number is log of this. So, the size of the input is size of E times log of f star. This is the size of the input at most the size of the input. And hence with respect to the size of the input this runtime is exponential. So, these sort of running times are called pseudo polynomial running time.

Whenever we have an integer for polynomial running time, we want running time to be polynomial in the log of the value of this input integers which is not in the case. So, recall

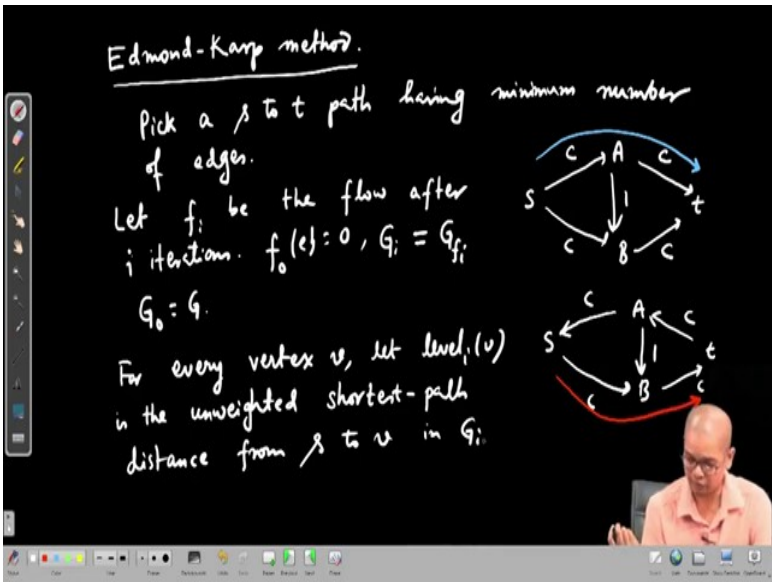
that Ford-Fulkerson method leaves the it does not specify which flow path to pick in the residual graph. And Edmond-Karp provides a particular method to pick a flow path in the residual graph to augment the current flow which results in fast fast computation of maximum flow.

And the runtime will be polynomial it does not depend on the maximum flow. But before that let us ask is the analysis of Ford-Fulkerson tight because it is a worst case and it is a big O bound. So, is it tight? So, is $O(f^* E)$ tight. That means do we have an example where the f Ford-Fulkerson method, indeed takes $O(f^*)$ many iterations. And yes, indeed, we have and we have already seen this example so, S to t A B.

So, suppose the capacities of these edges s to A are these are some very big numbers C, C is very large. And so, the value of the maximum flow is 2C. But if the Ford-Fulkerson method, picks the first flow path, S to A to B to t. And now in the residual graph there will be an edge from B to A and in the second iteration. If it picks S to A to C and in the residual graph after two iterations, this A to B edge again will appear and if it again speak S to B to t.

So, If in every odd iterations it if it picks A to B to t the blue path and it in every even iteration if it picks red path. Then you will see that in each iteration the value of the flow current flow increases by only 1. And hence to reach 2C, to reach the maximum flow, it will require size over value of maximum flow many iterations. So, in this example, the Ford-Fulkerson algorithm takes value of f^* many iterations.

(Refer Slide Time: 07:20)



So, this analysis of this Ford-Fulkerson method is indeed tight. So, now let us see, what is the Edmond-Karp method? So, Edmond-Karp method, it simply specifies which flow path in the residual graph to pick in each iteration and it says that pick are pick A s to t path having minimum number of edges. So, let us see in this example how it will happen, s to A, B, t.

So, in the first iteration itself the Edmund-Karp algorithm is not allowed to use S to A to B to t path because it involves 3 edges, whereas there is a path A to t or s to B to t which involves few edges. So, it picks one of the path involving minimum number of edges. So, whenever, when I am picking the flow path to augment a flow in the residual graph, I am not considering the capacities.

I am just picking the shortest path in terms of the minimum number of edges that it contain. So, the first part that it picks is s to A to t and it pushes C amount of flow and the resulting residual graph will look like this. There will be a edge from A to s C, A to t C. And in the next iteration again there is only one path and it pushes C amount of flow along the bottom path and hence it in two iterations it finds the maximum flow.

So, what we will show is that for any maximum flow problem this method of picking the path leads to a polynomial number of iterations to reach the maximum flow value. So, towards that so, let us define some notation, so, let f_i be the flow after i iterations. And so, in particular f_0 of every edge e is 0 and G_i is the residual graph of f_i, so, in terms in particular G_0 is G.

Now, for each vertex we define what is called a level of the vertex intuitively speaking what will basically show is that and any edge cannot be reversed many times. For example, if some edge disappears because that is, it is saturated in from the residual graph. Then, if at the reverse edge is there, if again in the later iteration, if any flow is pushed along the reverse edge then that edge will reappear?

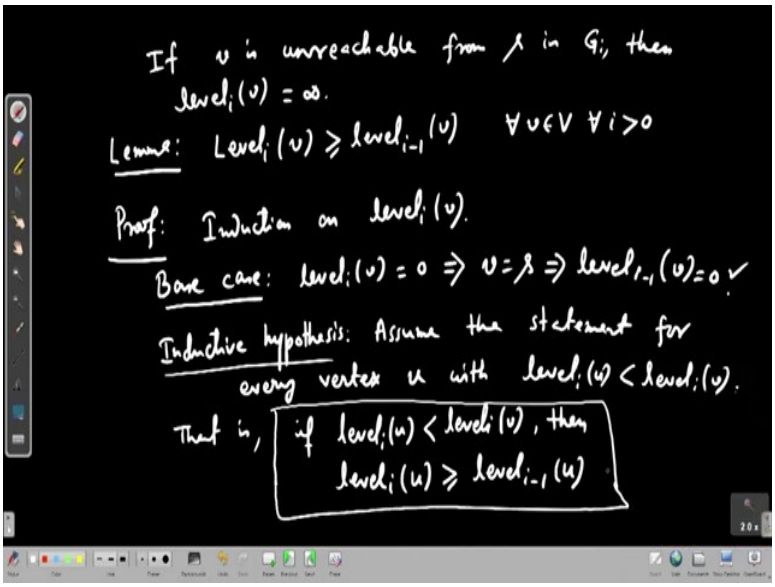
What will bound is we will show that any edge cannot appear and reappear too many times. Any edge can reappear at most size of v. The number of vertices is size of v that many times if it has n vertices, the each edge can reappear at most n times. That is the clean and we will

augment it with the fact that whenever we push a flow we at least 1 edge from the flow graph disappears.

Now, in each iteration at least, 1 edge disappears and each edge can disappear at most say $|V| \times |V|$. So that way, the number of iterations will be twice size of e number of edges in the residual graph times the number of vertices that is the high level idea. So, towards that let us define for every vertex v let level high of v is the unweighted shortest path distance from s to v in G_i .

That means look at the shortest path from s to v in terms of the number of edges that the path contains and that value let us call the level of that vertex v.

(Refer Slide Time: 14:43)

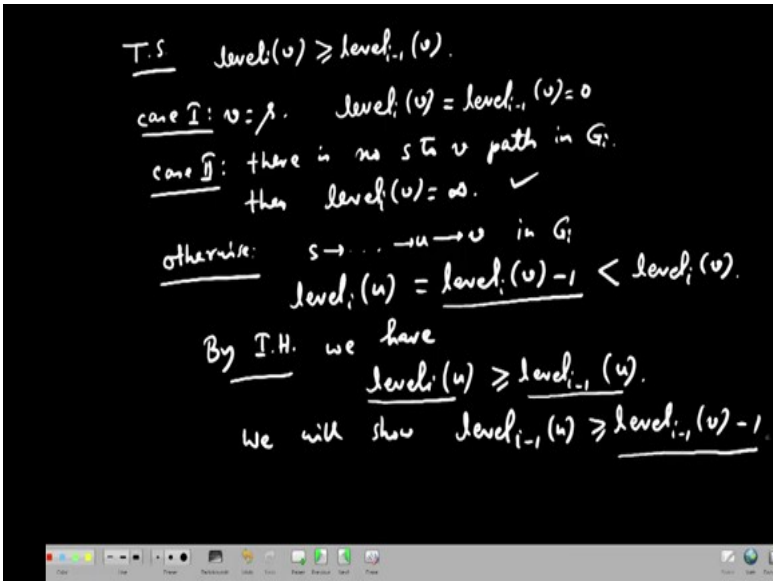


So, if v is unreachable from is s in G_i then we define level i of v to be infinity. So, first we prove that this level is a increasing function. Level only increases with i level of every vertex, so, first lemma level of level i of v is greater than equal to level i – 1 of v for all vertex v and for all i greater than 0. Proof, so, let i be any and v be any and will prove by induction on level i of v.

Not i not integer i, will induction will do induction on the level i of v. So, if level i of v is 0 then v must be an s, so, base case level i of v if it is 0 that is that can only be possible if v is s and then level i – 1 of v is also 0. And hence the inequality holds so the base case holds true. So, inductive hypothesis so, assume the statement for every vertex u with level i of u less than level i of v, that means what?

That means that is, if level i of u is less than level i of v then a level i of u is greater than equal to level i – 1 of u This is our assumption, inductive hypothesis, this we get from inductive hypothesis.

(Refer Slide Time: 19:26)



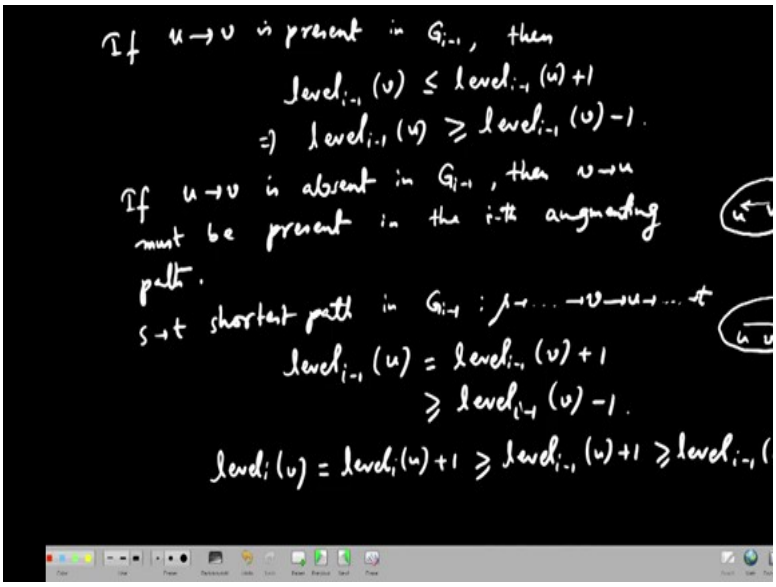
Now there are two cases, couple of cases, so, if so now, to show that level i of v this is greater than equal to level i – 1 of these 2 show. So, we will take some cases on v so, easy cases first. So, case I v = s, if v = S then level is 0, irrespective of i and hence the equality holds the inequality holds. Case II there is no path, no s to v path in G_i then level i of v is infinity and in this case also the inequality holds.

Otherwise, there is a path, so, otherwise let there is a path from in path from s to v in G_i . And suppose u is the last vertex see this is the shortest path, shortest path containing in terms this is the unweighted shortest path. This is the s to v path containing the least number of edges and u is the vertex just previous to v. So, level i of u is level i of v – 1 and in particular hence level i of u is strictly less than level i of v.

Now hence, from inductive hypothesis, we have level i of u is greater than equal to level i – 1 of u, Now, we will show basically, we will show what that level i – 1 of u this is greater than equal to level i – 1 of v – 1. So, if we show this then we get what we want because level i of u is nothing but level i of v – 1. So, this level i of u will be, is equal to level i of v – 1.

And if we can show that level i – 1 of u is this greater than equal to level i – 1 of v – 1. Then we can test this and we get level i of v is greater than equal to level i – 1 of v.

(Refer Slide Time: 24:06)



So, here also we make couple of bits some sub cases, so, we need to go to we need to consider G_{i-1} . So, we will show this so, for that I need to so, in this inequality involves i – 1 at level, so, i need to consider level i – 1 the graph i – 1. So, if this edge u v is present in G_{i-1} then again by shortest path property we have level i – 1 of v is less than equal to level i – 1 of u + 1 which is exactly what we need to show that level i – 1 of u is greater than this.

So that is level i – 1 of u is greater than equal to level i – 1 of v – 1. So, if this edge u v which is present in G_i if it was present in G_{i-1} also then it is very easy. Otherwise, if u v is absent in G_{i-1} then you see that this edge u to v was not there in G_{i-1} but it has come in G_i this edge u to v. Now, how can an edge appear? It is possible only if there was an edge from v to u in G_{i-1} . So, if u to v is absent in G_i .

Then v to u must be present in the ith augmenting path. It is not only present the augmented path also should contain this then only this edge the augmented path must also contain v to u edge. Then only we say edge u to v can reappear but the here we are using the fact that the Edmond-Karp algorithm only uses only augments shortest paths. So then, this u to v to u edge is A is contained in S to t sort a shortest path.

So then s to t shortest path in G_{i-1} look like this it starts from s and it goes to v then it uses this edge and then it reaches t. So then, this is exactly what I need to get that level i – 1 of u is nothing but level i – 1 of v + 1 which is greater than equal to level i – 1 of v – 1. So, this proves the statement now in both the subscripts then what we have is that. Now, if we patch this level i of v is level. So, here in this case, we had this inequalities level i of u is level i of v – 1.

So, level i of v is level i of u + 1 and this is greater than equal to this is because of induction hypothesis, i – 1 of u + 1 and is greater than equal to level i – 1 of v. So, this concludes the proof of this lemma. Hence the shortest path augumented shortest path distance from s to every vertex increases it did not decrease. And what we will show in the next lecture that between every disappearance this level increases by at least two.

And hence the number of iterations will be bounded because when the algorithm stops, the algorithm stops only when t is unreachable. So, we will continue from here in that next, let us play. Thank you.