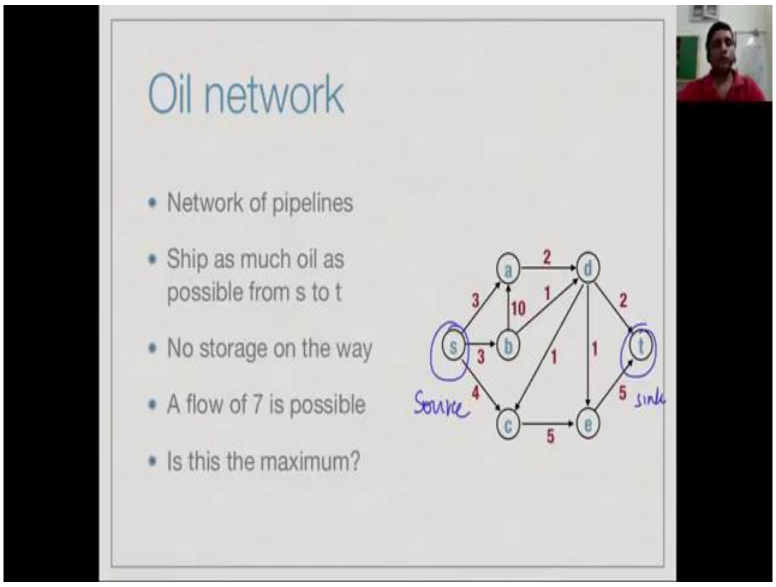
Design and Analysis of Algorithms, Chennai Mathematical Institute
Prof. Madhavan Mukund
Department of Computer Science and Engineering,

Week - 08
Module - 04
Lecture - 53

Network Flows

In the bandwidth allocation problem, we use one variable per path in order to encode a network flow problem as a linear program and we argued that does not an efficient way
to do it. So, let us look at a mode direct way to represent network flows in terms of linear programs.

(Refer Slide Time: 00:16)

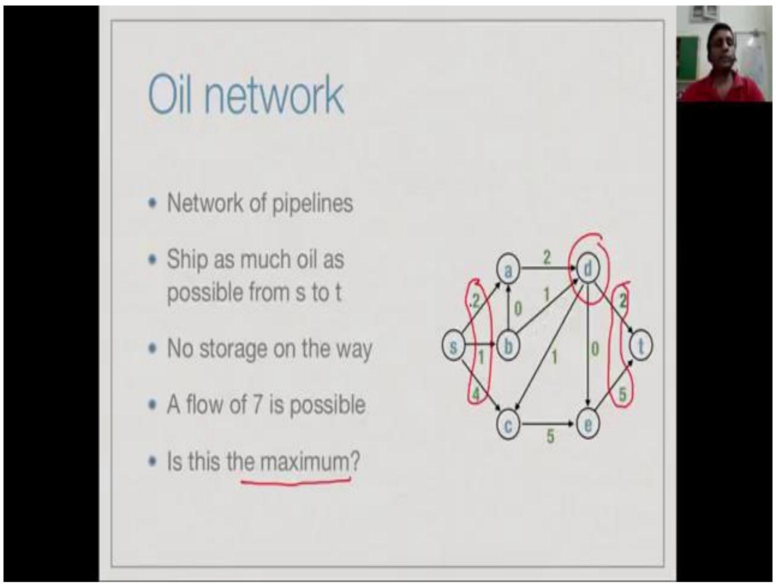


Oil network

- Network of pipelines
- Ship as much oil as possible from s to t
- No storage on the way
- A flow of 7 is possible
- Is this the maximum?

So, suppose we have an oil network which is shown as given in this directed graph. So, we have a source vertex s and we have a target or a sink vertex t and our aim is to shift as much oil as we can from s to t given the pipes that we are given. So, of course, one property of a flow is that it is must flow, so I cannot keep any quantity at any intermediate node. So, anything that enters b must leave b, anything that enters d must leave d.

(Refer Slide Time: 00:53)



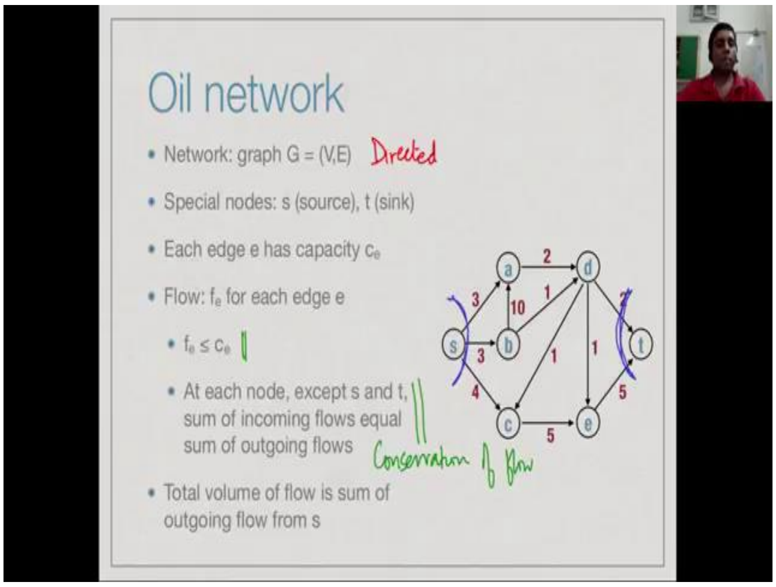
Oil network

- Network of pipelines
- Ship as much oil as possible from s to t
- No storage on the way
- A flow of 7 is possible
- Is this the maximum?

So, if we try to do this, then for instance this green quantity represents one possible flow, I send two units from s to a 1 from s to b 4 from s to c and you can check that these are interact with in the capacity there could for sent a 3, 3 and 4 and I have set 2, 1 and 4 and I keep going. And therefore, now at this particular thing I have a total of 7 units which flows out of s and 7 units which comes into t. So, I am able to flow 7 units from s to t given this network.

So, we can just verify this locally then this is a flow by check instance at d, the total quantity flowing in 2 plus 1, 3 if nothing is stored because the total quantity flowing out is again 2 plus 1, 3. So, if we can do local checks like this, we can satisfy our self that this is a valid flow, the total amount is 7. But, the question is, is this the maximum? How do we know that we achieve the maximum or not?

(Refer Slide Time: 01:44)



Oil network

- Network: graph G = (V,E) Directed
- Special nodes: s (source), t (sink)
- Each edge e has capacity c_e
- Flow: f_e for each edge e
  - $f_e \leq c_e$
  - At each node, except s and t, sum of incoming flows equal sum of outgoing flows

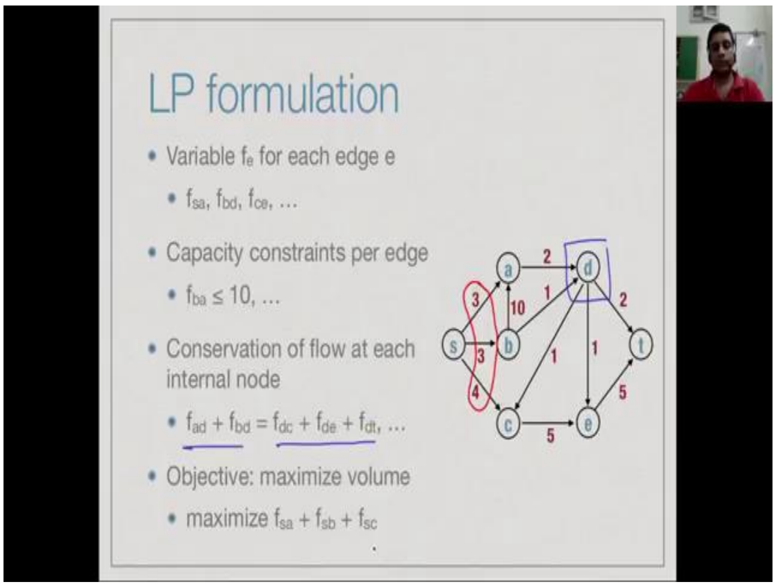
Conservation of flow

- Total volume of flow is sum of outgoing flow from s

So, the problem just to face it formally is that we have given a special type of graph, it is a directed graph and it has two special nodes, a source and a sink. The source has no incoming edges and the sink has no outgoing edges. Each edge has a capacity, which is a weight associated with the edge and our aims to come up with the flow, the flow is again the quantity that we will assign each edge. And now the flow must satisfy some basic conditions, the flow must always be less than the capacity, then we have no storage.

So, at every internal node the total amount flowing into the node must be equal to the total amount flowing out. So, this is called conservation of flow, we cannot lose anything or generate anything at an intermediate problems, anything that comes in must go out. And finally, what we are do is we want to optimize the total volume on flow, the total volume of flow is the amount of flow which is coming out of here which is also of course, the amount of which is going that in the flow can be lost. So, we can look at either n, but less just define it, to be the total volume of the outgoing flow from the source.

(Refer Slide Time: 03:01)


LP formulation

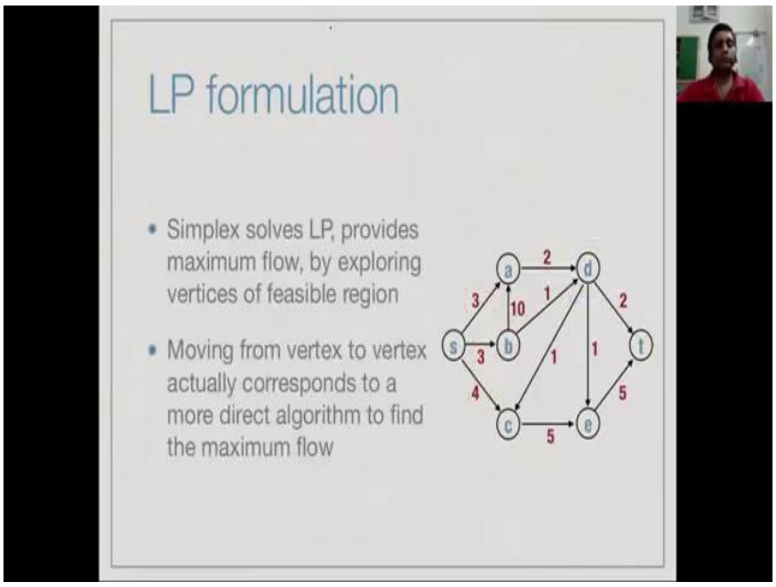
- Variable f_e for each edge e
  - f_{sa}, f_{bd}, f_{ce}, ...
- Capacity constraints per edge
  - $f_{ba} \leq 10$, ...
- Conservation of flow at each internal node
  - $\underline{f_{ad} + f_{bd}} = \underline{f_{dc} + f_{de} + f_{dt}}$, ...
- Objective: maximize volume
  - maximize $f_{sa} + f_{sb} + f_{sc}$

So, now we have this formulation remain, so we can now set up a linear program, what we associate do is we associate and said one variable for each edge. So, for instances for the edge s a we have f s a and then for the edge b d for example, we have f b d and then for c e, we have f c e and so on. So, we have 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 believe edges the 11 edges and therefore, we have the11 variables in this linear program.

Now, what can we do in this variables, one first is we can say that each variable is constrained by the capacity of the corresponding edge. So, f b a, so b a is now this edge, it has a capacity of 10. So, whatever flow I finally arrive at from b to a must be less than 10 and the other think that we can say is that we must have a conservation of flow at each internal node. So, for instance if I look at this node d, then the incoming flow is a d plus b d, the outgoing flow is d c plus d e plus d t and these two quantities must be equal, f a d plus f b d must be equal to f d c plus f d e plus f d t.

And finally, so these are the constraints, so the constraint says that every edge can only carry or flow with in this capacity and in every internal node there is a conservation of flow. And finally, objective is to maximize what happens on these three edges f s a plus f s b plus f s c, this is our objective function.

(Refer Slide Time: 04:45)

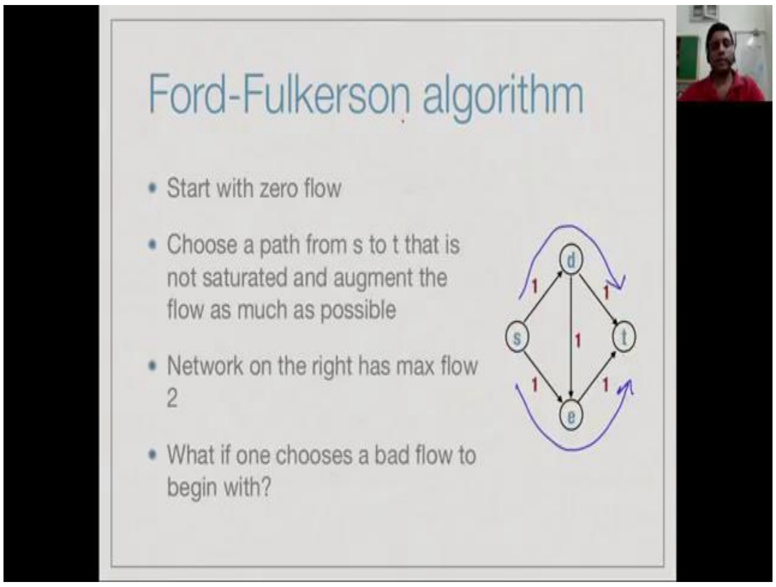


LP formulation

- Simplex solves LP, provides maximum flow, by exploring vertices of feasible region
- Moving from vertex to vertex actually corresponds to a more direct algorithm to find the maximum flow

So, of course as before we will just invoke a linear programming solved as a just simplex on these some and get an answer. But, what we will do now is to understand what this actually means, remember how simplex works. So, simplex start with the vertex of feasible region gives going from one vertex to an x. So, as among this increasing the flow is actually taking an existing flow and adding something do it and this can actual be interpreted directly in terms of the flow finding algorithm.

(Refer Slide Time: 05:19)

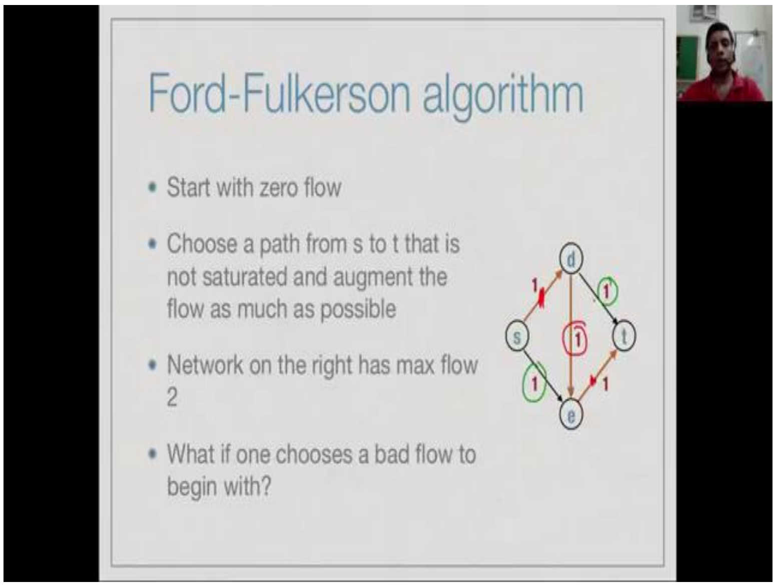


Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2
- What if one chooses a bad flow to begin with?

So, this is an algorithm called the Ford-Fulkerson algorithm which actually tries to directly solve a network flow problem by gradually building up an optimum flow. So, the algorithm assume you starts with 0 flow and then you choose some path on which there is pair capacity and then on this path, you augment the flow as much as possible, so that path become saturated. So, now if you look at the algorithm network on the right, it is very clear that there is a flow possible two, you can sent one unit flow that way, you can sent one unit of flow this way. But, Ford-Fulkerson algorithm says take any path which exists and starts flowing things to there.

(Refer Slide Time: 06:06)

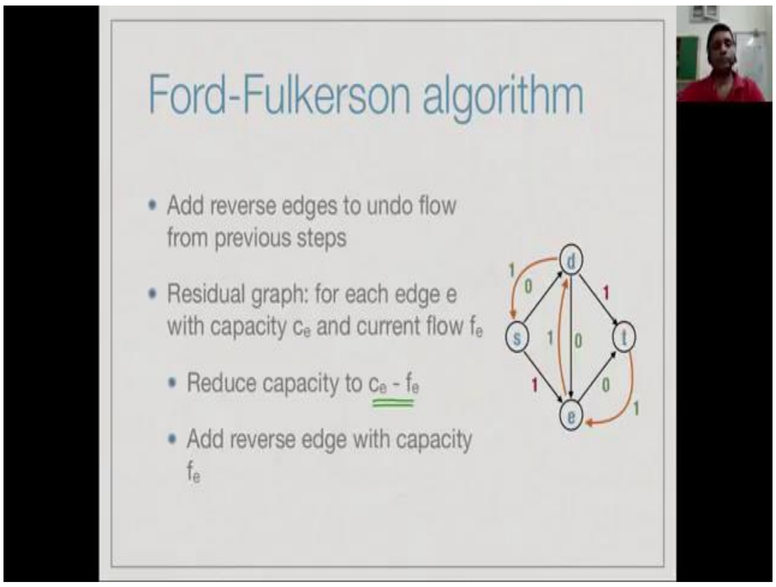


Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Network on the right has max flow 2
- What if one chooses a bad flow to begin with?

So, you could for instances begin with this path, the one that goes from s to d then from the d down to e and then to t. So, flows one unit, but now this point this edge is not saturated and this edge is not saturated and of course, this edge is in the wrong direction. So, I cannot make use of these two capacities to generate the second unit of flow, so it looks like the Ford-Fulkerson algorithm takes a bottle neck, if you choose the wrong path to start with.

(Refer Slide Time: 06:34)



Ford-Fulkerson algorithm

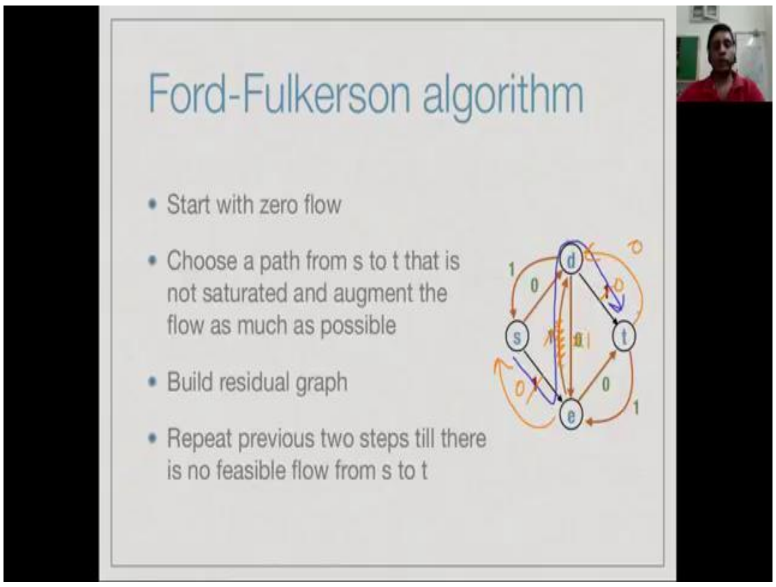
- Add reverse edges to undo flow from previous steps
- Residual graph: for each edge e with capacity c_e and current flow f_e
  - Reduce capacity to $\underline{\underline{c_e - f_e}}$
  - Add reverse edge with capacity f_e

So, the solution is to save that even if you are taken back path, one of the thinks we can do is reverse the decision we made earlier. So, we want to say that if we are flowing one through this, then we can reduce this flow. So, we can divert this flow back another way, so that is a bit complicated to describe, but one way to solve it is to actually setup and extra edge allowing us flow thinks back.

So, this is what we call the residual graph, so in the residual graph what we will do is we will actually take the flow that we just contracted and then we will change the capacities. So, the forward edge s to d which had capacity 1 and flow 1, now has residual capacity 0. So, we have a regular edge then we replaced it is weight by the actual amount that is still available. So, that is the capacity minus the current flow and in addition we add this new edges is backward edges, which corresponds to the flows that we have committed, but which we may want to change data.

So, we have sent of flow 1 from s to d, but now we can reduce that flow by sending some flow back from d to s that is what this is supposed. So, formally this is have a constructed is residual graph, you take the original graph then any flow that you have set up in an existing edge, you reduce the capacity of that edge by that flow and corresponding to that flow you setup of a reverse edge which allows as to later to undo this.

(Refer Slide Time: 08:09)

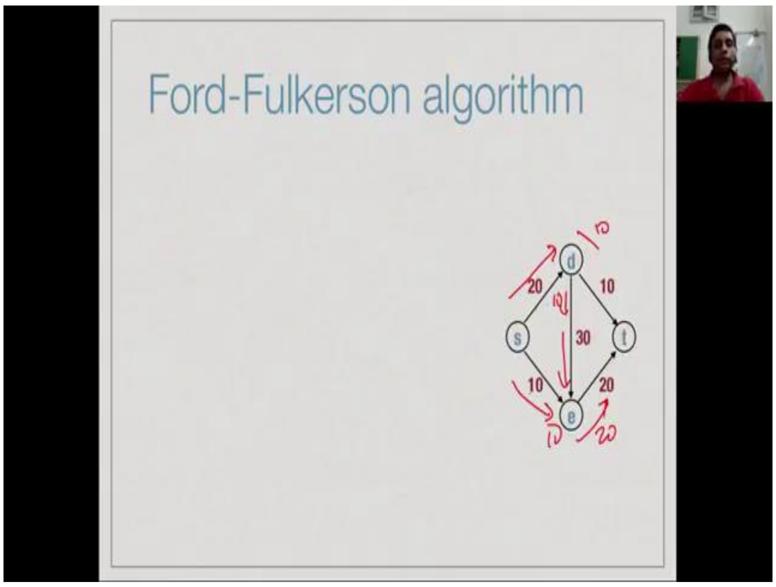

Ford-Fulkerson algorithm

- Start with zero flow
- Choose a path from s to t that is not saturated and augment the flow as much as possible
- Build residual graph
- Repeat previous two steps till there is no feasible flow from s to t

So, going back to this example, so what would do is we would first start with this wrong flow then we will say, but then we would residual graph and now we will observe that in this residual graph, there is path which goes like this. So, we are not talking about the original graph, we are only talk about residual graph at each stage. So, we will root build this now this will result in the new flows. So, this will end up reducing a 0 here and new edge back here similar 0 here and new edge back here and this one will now become canceled out.

Because now the path this flows this is show to one and this edge deserve it and now in this new graph will find that there are no edges left, because I have 0 capacity going from s to d as 0 capacity going to s to e. So, I cannot lean any flow to s, so this is my find of few, so this is the ford Fulkerson algorithm, let us look at it again in a slightly similar structure, but with us slightly different set of numbers.

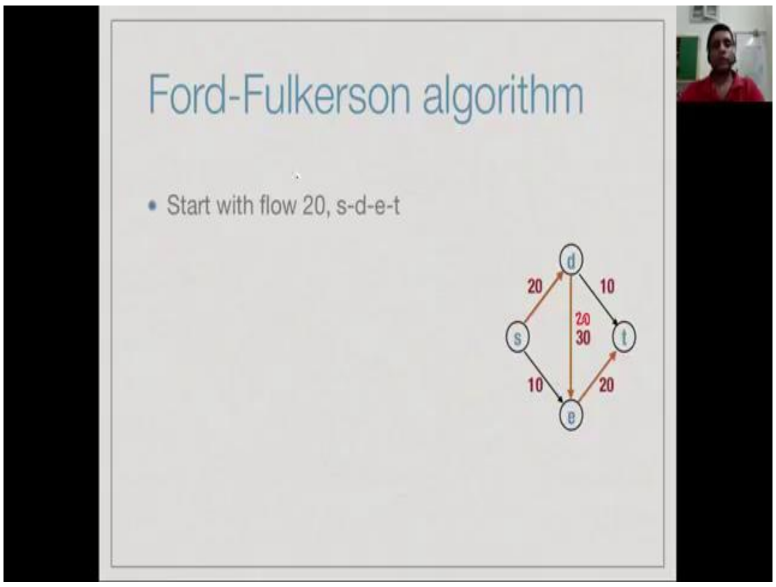
(Refer Slide Time: 09:07)



Ford-Fulkerson algorithm

So, this is a graph which has not once, but some 20s and 30s, so here intuitively with the claim is that all the 30 units can flow. But, the 30 units cannot flow unlike the earlier case along the edges of the bang, because if I take 20 units or then I must splited it us 10 plus 10. So, I must recognize that 10 must go down and 10 must go there, similarly if I put 10 here then this 10 and then incoming 10 combine from these 20, so this is have I get of flow of 30 in this graph.

(Refer Slide Time: 09:44)

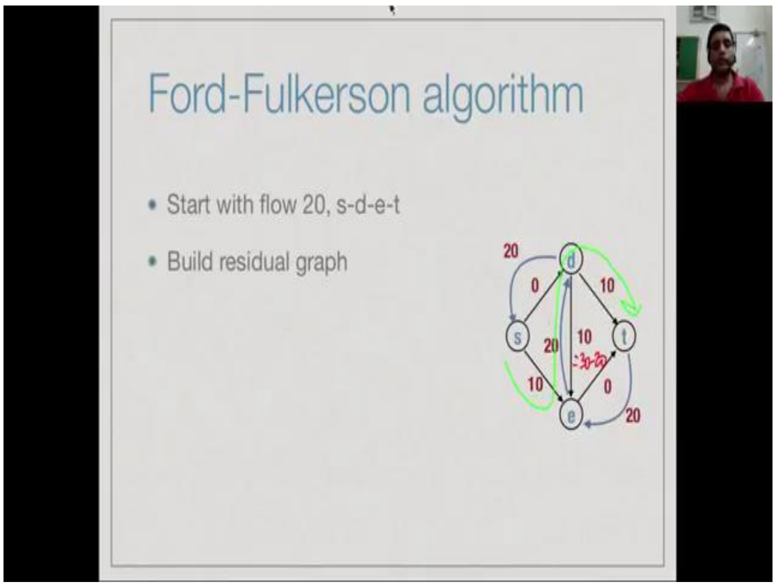


Ford-Fulkerson algorithm

- Start with flow 20, s-d-e-t

But, if I start the Ford-Fulkerson algorithm, it will try to saturated the path. So, supposing it identifies the path s to d to e to t, then if it identifies this path then it is put it 20 flow through this edge.

(Refer Slide Time: 10:00)

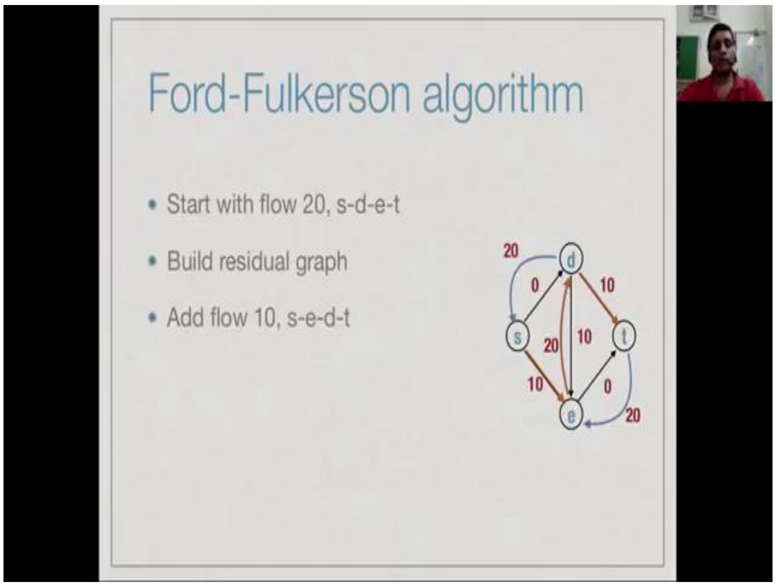


Ford-Fulkerson algorithm

- Start with flow 20, s-d-e-t
- Build residual graph

So, we start the in this flow and we build the residual graph, the residual graph says that this 10 is 30 minus 20. The residual capacity from d to e this now 10 because I had a capacity of 30 and I would 23. And the blue edge is now, the residual edge which allows me to reduce this 20 back, similarly from s to d I had earlier capacity of 20 have put 20 through it. So, it reduce capacity is now 0. But, I have a backward edge allows we to undo these later requirement and the same width e to t, e to t is reduce to 0, but I have a backward edge. Now, I look for another path in this graph, so for instances I find that this path is there I have a path from s to e to d to t.

(Refer Slide Time: 10:42)

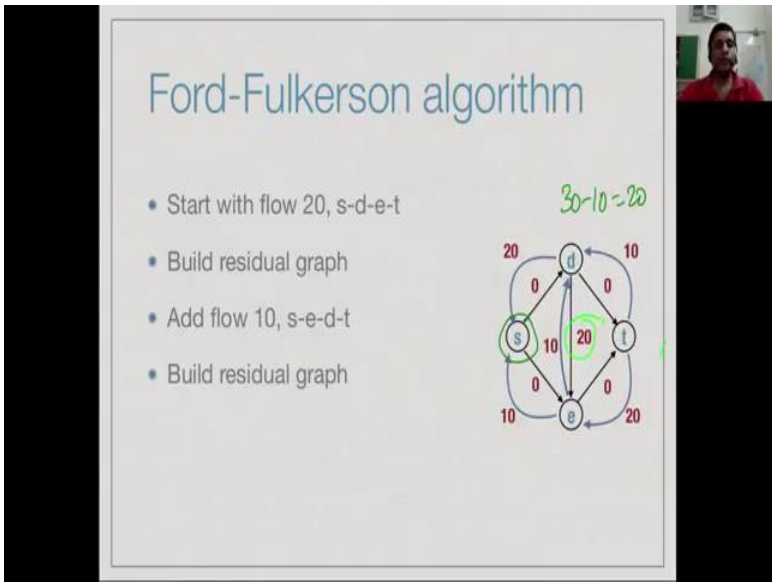


Ford-Fulkerson algorithm

- Start with flow 20, s-d-e-t
- Build residual graph
- Add flow 10, s-e-d-t

And ((Refer Time: 10:44)) is constraint is 10, because I have only ten flowing out of s to begin with, so I take that 10 and then I build residual graph.

(Refer Slide Time: 10:52)

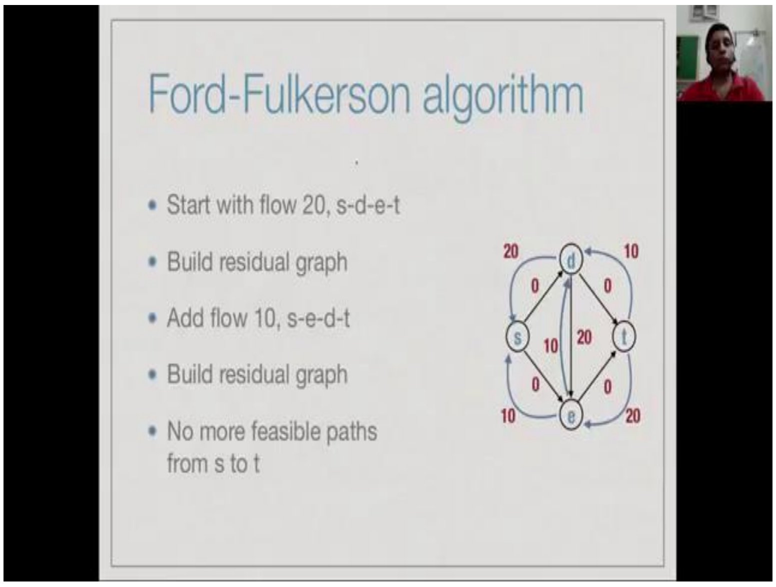


Ford-Fulkerson algorithm

- Start with flow 20, s-d-e-t
- Build residual graph
- Add flow 10, s-e-d-t
- Build residual graph

So, when I take that 10 these quantities here which are associated with this edge get list out back plug in. **Because** the total flow from d to e was 20 in the first round minus 10 in the second round is 10 therefore, the residues 30 minus 10, it will 20 and everywhere else now about 0 because a saturated, now if I look at s there is no outgoing flow possible in the residual graph.

(Refer Slide Time: 11:25)

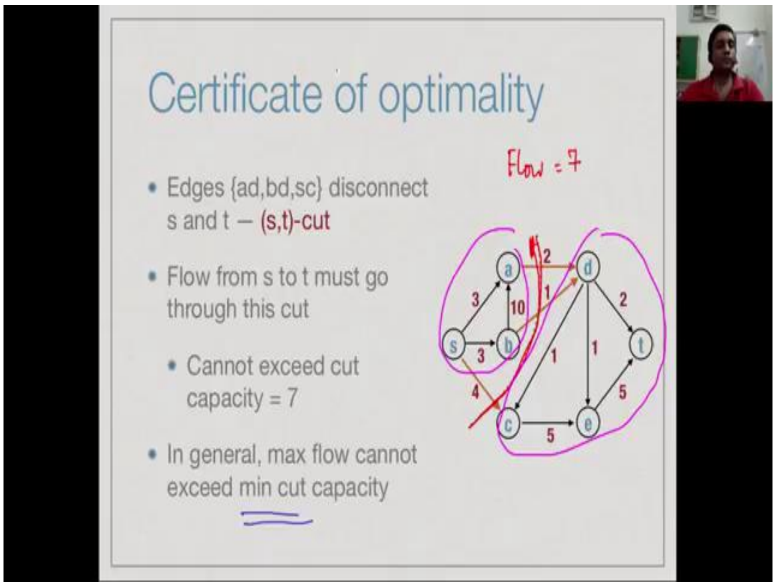


Ford-Fulkerson algorithm

- Start with flow 20, s-d-e-t
- Build residual graph
- Add flow 10, s-e-d-t
- Build residual graph
- No more feasible paths from s to t

So, then I say that there are no more feasible path and I stop and the residual edges are if I have been keeping track of that flow it will tell me that I have achieve flow of 30 in this graph.

(Refer Slide Time: 11:35)



Certificate of optimality

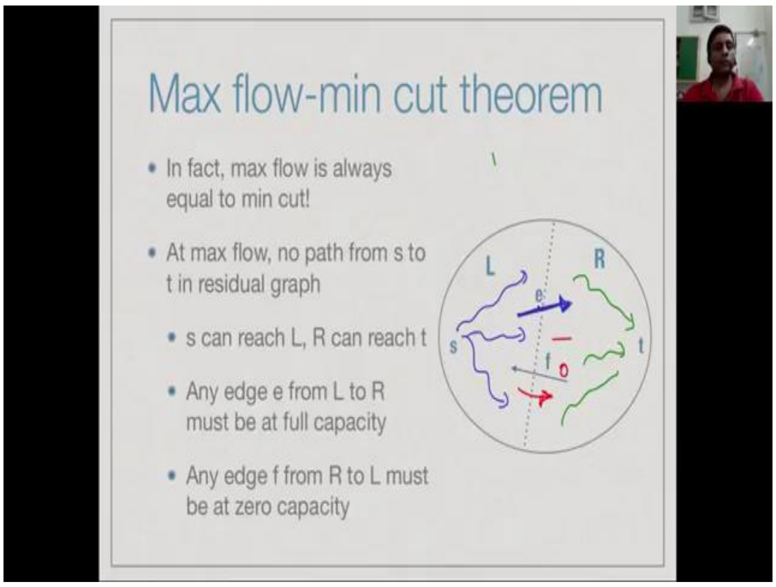
- Edges {ad,bd,sc} disconnect s and t – (s,t)-cut
- Flow from s to t must go through this cut
  - Cannot exceed cut capacity = 7
- In general, max flow cannot exceed min cut capacity

So, again if you want to ask the question as to while a given flow is optimal, we can ask for some quantity which is certificate of optimality. So, if we go back to the original oil shipping think we claim that we could setup a flow of 7. So, by hand we constructed a floor support, now let us look at these three edges, the edge k to d, the edge b to d, the edge s to c, if we disconnect this graph by cutting these edges, other if we cut these edges we discount this graph.

So, these three edges form what is called a cut between a set, now in this cut the total capacities 4 plus 1 plus 2, 7 at just any flow at all; however, it flows it must cross from this side to this side from left to the right. So, it can only flow from the left to the right through the edges with a part of that cut that the cut can only support flow of 7. So, therefore optimum flow can definitely not exceed 7 in this example, which we have already achieved.

So, we know that we can do flow of 7, but no more than 7 is possible, because this cut will prevent any think more than 7 from the flowing from left to the right. So, actually in this case the shows as that 7 is optimum, so in general if can look at various such cuts. So, cut is any set of edges with disconnect s from t and you can calculate the minimum cut across all of these and it is pretty clear that the maximum flow cannot exceed the minimum cut, because it has the cross this cut. So, you have to go from one side to rather it can only take that much of capacities, so the flow cannot exceed that.

(Refer Slide Time: 13:20)



Max flow-min cut theorem

- In fact, max flow is always equal to min cut!
- At max flow, no path from s to t in residual graph
    - s can reach L, R can reach t
    - Any edge e from L to R must be at full capacity
    - Any edge f from R to L must be at zero capacity

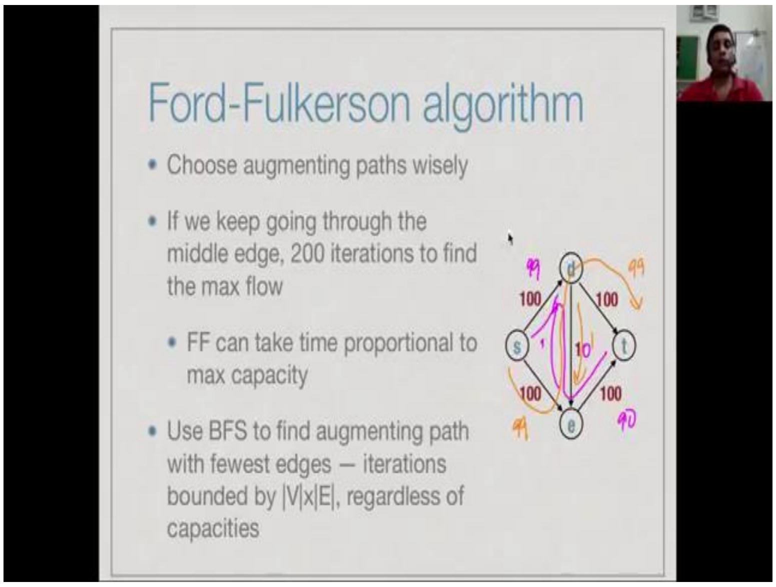
So, what is surprising that is actually that it will always in this example of the we did it was equal, but it always going to be equal. So, the max flow min cut theorem it says, the max flow actually always equal to the minimum cut. So, here is one way to understand this, so if we look at are l p solution, when we achieve the maximum flow, then s is going to be disconnected from t, there is no further path if I look at edge weight 0 and I remove those edges, then there is no further path.

So, s is disconnected from t, so there is a cut there are some edges which disconnected edge s from t. Now, let us look at any edge in the residual graph that point which goes from the left hand side to the right hand side. So, left is everything, so everywhere inside these I do have paths with nonzero edge and everywhere inside this I do have paths 50 with nonzero edge. So, now but I cannot get from the blue side to the green side.

So; that means, that in the forward direction all the edges must have saturated the capacity. So, every edge from l to r is actually are a full capacity, what about an edge in a residual graph f to r to l, the claim is a there was some capacity here then they would be a reverse side which goes this way, which would be a nonzero capacity. So, there would be a path from s to d whether no paths, so this must be at 0.

So, the reverse edges must be at 0 all the forward edges must be a full capacity. So, this gives a not very precise, but a reason why, but max flow will actually saturated this cut, but this could be any cut. So, therefore the minimum cut in particular be saturated and therefore, the max flow cannot exceed the minimum cut anyway, so the maximum must achieve the minimum cut value.

(Refer Slide Time: 15:07)



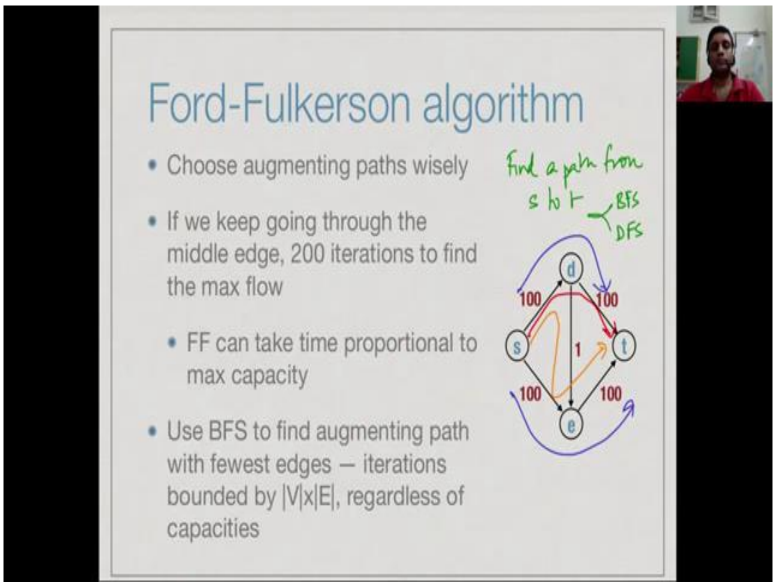
Ford-Fulkerson algorithm

- Choose augmenting paths wisely
- If we keep going through the middle edge, 200 iterations to find the max flow
    - FF can take time proportional to max capacity
- Use BFS to find augmenting path with fewest edges — iterations bounded by |V|x|E|, regardless of capacities

So, one think that one has to be careful about in the Ford-Fulkerson algorithm is to choice of how to increase the path. So, remember that in our pathological examples, we instead of going around the diamond you go through the center. So, supposing we keep doing that the model happen is here after one iteration, we go through this center and then we reduce this 99 this to 0 this to 99 and we will set up reverse edge of size 1.

Then this next iteration we will go in the reverse direction and we will again reduce this to 99 this to 99 and then will set up this reset this way back to one. So, this way can keep Zick-zacking. So, I will go from 99 to 98 and the down algorithm 99 to 98, so it will take the 200 iterations in order to find this path, this flow has 200 on the other hands is pretty clear that I could done it two iteration if had been cleaver.

(Refer Slide Time: 15:59)



Ford-Fulkerson algorithm

- Choose augmenting paths wisely
- If we keep going through the middle edge, 200 iterations to find the max flow
  - FF can take time proportional to max capacity
- Use BFS to find augmenting path with fewest edges — iterations bounded by |V|x|E|, regardless of capacities

Because I could a just I know initially there is a path of 200 there and the saturated. So, it is two edges and there is another path of 200 there, in a path of 100 here in the path of 100 here. So, in two iteration I can achieve 200, so it depends quite curiously and how I pick to path to augmented, how do I take the feasible path which still exist in the residual graph and choose which one to add it.

So, in general we cannot say anything good and Ford-Fulkerson is going to take time which is proportional to the capacity of the edges which is not a great idea. Because all though edges may be large you might may be could directly say this edge can take 100 and next edge can take 100. So, one sort I can say the whole path can take 100, if I have one at a time then it becomes extra parameter not the actual size of the graph. But, how do we do this feasible path business at all. So, every time we set up a residual graph we have to find a path from s to t and then augmented.

So, this we would have seen right to the beginning of this course, we will either do it through breath first search or depth first search, finding a path from one node to another node is typically this the exportation of the graph, if we use breath first search, then what we know we find the shortest path in terms of number of edges. So, one can prove that in this the Ford-Fulkerson algorithm, if we use breath first search at every iteration to decide which path to augment, then you will be augmenting always the shortest path in terms of edges.

So, for instances in this example here, if you had a choice between augmenting that path the 100 the path which goes about like this and the path which goes like this, then the breath first search will say that the first the red path has got two edges and the orange path as got three edges. So, the red path is shorter, so even augment that first, so if we use breath first search in the ford Fulkerson algorithm, it turns out that you will always get something which is propositional to the product on the vertex in the edges. So, it will be polynomial in the size of network independent of the capacities.