Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 03

Lecture 15

Welcome to the 15th lecture of the second-level algorithms course. In the last class, we have seen how a natural greedy algorithm for computing the maximum flow of a network does not always output the maximum flow, and we have also defined the concept of a residual graph. Given a flow network and a flow in that network. So, in this lecture, we will understand the residual graph better, and we will see how we can use the concept of a residual graph to design algorithms for maximum S-T flow. So, let us begin. So, we begin this lecture by making some assumptions about the input graph, which we can make without loss of generality.

We assume the following without loss of generality. The first one is there is no self-loop in the input graph. So, if there is a vertex which has a self-loop and the capacity of the edge is c, what can we do?

We can replace this vertex and this self-loop with a vertex v, and we introduce another vertex put an edge from V to W and W to V. The capacity of both edges is C. And it is easy to see that the value of the maximum flow from S to T is the same in both graphs. Moreover, given a maximum ST flow in one graph, one can easily construct a maximum S-T flow in another graph. So, if the input graph contains some self-loop, we make these changes and make the graph have no self-loop. The second assumption is there are no parallel edges.

In the input graph. Again, suppose there exist parallel edges from U to V, and the capacities are C_1 and C_2. We can replace these parallel edges with one edge from U to V with capacity C_1+C_2. Again, we can prove that the value of the maximum s-t flow remains the same in both graphs. Moreover, given a flow of value f, from s to t in one graph, we can construct in polynomial time another flow of value f in the other graph. So, this also we can assume without loss of generality. If our input graph has parallel edges,

we make these transformations and assume that the input graph does not have any parallel edges. We remove all the parallel edges in this way.

We also assume that there are no anti-parallel edges. Again, if there are anti-parallel edges in the input graph, say between u and v, and the capacities of the edges are c_1 and c_2, what we can do is introduce another vertex for each anti-parallel edge. And replace anti-parallel edges with these gadgets locally. Again, it is provable that the value of the maximum s-t flow is the same in both graphs. And if we are given a flow f in one graph, one can easily construct another flow of value f in the other graph. So, we make these assumptions without loss of generality. We will see why we need these assumptions.

For example, we need this assumption of no self-loop to clearly define the residual graphs. The construction of residual graphs becomes very cumbersome if the input graph has parallel edges or anti-parallel edges. Or self-loops. So, if the input graph has any of these kinds of edges, we get rid of them by making these transformations and then have an input graph where there is no self-loop, parallel edges, or anti-parallel edges, okay? In the last class, we have defined the concept of residual graph.

So, now let us see how using the residual graph we can modify the greedy algorithm to compute a maximum s to t flow. So, let us take the example of the network flow problem where the natural greedy algorithm failed to output the maximum s to t flow. So, this is the input graph G. These are the capacity values, and it is an iterative algorithm. The natural greedy algorithm initially sets the flow to 0 if e is 0 for all edges in E, right? So, with respect to this flow, if I draw the G_f. We have only forward edges, and G_f is identical to G because no edge carries any flow; there are no backward edges.

The algorithm is that instead of looking for an s to t path in the graph G, where every edge of the path has a residual capacity, look for such a path in G_f. So, the modified greedy algorithm Instead of looking for an s to t path in G where every edge of the path has residual capacity We look for an s to t path in G_f. Let us call this path P. We do not need to write residual capacity because the construction of the residual graph ensures that every edge has a capacity.

Then we send as much flow as possible along the path P. This step is called flow augmentation. So, let us see the working of this modified greedy algorithm in the counterexample of the greedy algorithm. So, I am looking for an s to t path in G_f, and let us pick any s to t path, namely the path which ensures that the greedy algorithm fails, which is s to a to b to t. Okay, and we send as much flow as possible along this path. That

means our new flow value is: flow s to a is now 1, f_{ab} is 1, f_{vt} is 1, and the flow of the remaining edges is 0. So, this is the current flow value with respect to the current flow value. Let us draw the residual graph. So, what does G_f look like then? So, let us draw the forward edges. Which edges?

Forward edges are the edges of G which have residual capacity left. Only two edges have residual capacity left: S to B and A to T. And the capacity of these edges in the residual graph are their residual capacities, both of which are 1 in this case. In the residual graph, we also have backward edges. For every edge which carries some positive amount of flow, we have an edge in the reverse direction with the capacity same as the flow.

So, in this flow, we have a flow from S to A with capacity 1. So, we have a reverse edge of capacity 1. Same with B to A with capacity 1 reverse edge and T to B with capacity 1. So, this is how the flow or residual graph looks like after one iteration of the

Greedy algorithm or the modified greedy algorithm. Again, we look for an s to t path in this residual graph, and now we see that there exists an s to t path, namely s to b to a to t in G_f. We augment the flow along this path. That means, what? So, this is iteration 1.

Formally, what do we mean by flow augmentation? If I am adding, if I am sending a flow along the forward edge, then I will add the flow in the corresponding edge in G, and if I am sending a flow in a backward edge, then I subtract the flow value in the corresponding edge in G. So, S a the flow value was 1, and we are not sending any flow along forward or backward edge. So, the flow value of S A remains the same. Now, S B is the forward edge, and the flow value along that edge was 0, and I am sending 1 unit of flow.

So, S B is also 1. Similarly, f is 1 and f is 1, but f. So, what was the age? Let us see, the age was a to b. So, f

It was 1 after iteration 1 and in this iteration I am sending an 1 unit of flow in the reverse stage from B to A. So, the net flow along A to B is 1 minus 1 which is 0. So, this is the flow after iteration 2. Now, let us see how the residual graph look like. Let us first draw the forward edges. Forward edges are those edges which has residual capacity left and the only edge is A to B.

with the residual capacity 1. Now, let us draw the backward edges. Backward edges are those edges which carry positive amount of flow. So, the edge S to A carries 1 unit of flow. So, I have a backward edge from A to S with flow 1.

Similarly, B to S with flow 1. Similarly, not flow, capacity 1. Similarly, T to A with capacity 1 and T to B with capacity 1. Now, again I look for an S to T path in G_f and see that there is no S to T path in G_f. When this happens the algorithm terminates and outputs the current flow as a maximum

So, we have seen that in the counterexample for the greedy algorithm, the modified greedy algorithm works, but does this mean that it will work for every instance? The answer is yes, but that needs a proof, which is our next claim. The modified greedy algorithm is correct. It is correct if all the input capacities are integers. And because of this, this modified greedy algorithm is called the Ford-Fulkerson method.

So, it is not an algorithm if the capacity values are not integers or rational numbers. So, even this not only applies to integers but also to rational numbers. So, you can replace these integers with rational numbers. So, there are instances of network flow problems involving irrational numbers where the modified

greedy algorithm never terminates. That is why this modified greedy algorithm is called So, to prove its correctness, what we need to show is that when the capacities are rational numbers, the algorithm always terminates, and when does the algorithm terminate? The algorithm terminates when there is no s-to-t path in the residual graph G. So, what we will show is the following lemma. Given a flow network G equal to (V, E), and the capacity values and a flow

The following are equivalent. First, f is a maximum s-to-t flow in G. Second, there is no s-to-t path in the residual graph G_f. The third one is the value

of f, which is the total amount of flow leaving s. is the same as the capacity of the cut. where U is the set of vertices reachable from s in G_f. And what is the capacity of a cut? The capacity of $(U, V \setminus U)$ cut is the sum of all the edges uv such that u belongs to capital U and v belongs to capital $V \setminus U$, the capacities of those edges. That means here is this set of vertices V, here is a subset of vertices U, this has s U contains s because s is reachable from s. It does not contain t because there is no s-to-t path in G_f, and the capacity of this cut is you add the capacities of all the edges going from U to $V \setminus U$. So, in the next class, we will prove this lemma, and once this lemma is proved and if we show that the algorithm, the Ford-Fulkerson method, always terminates if the input capacities are all rational numbers, then this proves that the Ford-Fulkerson method is correct. So, we will do that in the next class.

So, let us stop here. Thank you.