Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 03

Lecture 14

Welcome to the 14th lecture of the second-level algorithms course. In the last class, we started a very important topic called maximum flow, and we will continue from there. In the last class, we claimed an important result where Given a flow, the total amount of flow leaving the source vertex S is the same as the total amount of flow going into the sink node, and the proof I gave as homework with a hint. So, let us begin today's lecture with a full proof of that claim. Let f be a flow in a network G equal to (V, E). From the source vertex S to the sink vertex T, then the total

amount of flow leaving S is the same as the total amount of flow going into T. The idea is to sum over all vertices the total flow into that vertex minus the total flow going out of the vertex. Consider the following: Sum S equal to sum over all vertices the flow into the vertex V, which is the sum of the flows of all the edges going into V minus the sum of the flows of all edges leaving V. Because of the conservation of flow property, we know that the sum is 0 for every vertex $v \in V \setminus \{S, T\}$. So, S equal to total flow into T minus total flow from S. On the other hand, every edge UV in A contributes to the sum S.

So here, you have an edge from u to v carrying a flow f(u,v). So f(u,v) contributes positively to the vertex v, and f(u,v) contributes negatively to the vertex u. which is 0. So, the net contribution of every edge to the sum S is 0. Hence, S equals 0. But we have already observed that S is the total flow into S minus the total flow from S. Hence, the total flow into T is the same as the total flow from S, thereby proving the claim. In the last lecture, we also saw a natural greedy algorithm for computing a maximum S-T flow. So, a natural question is: does the natural greedy algorithm for computing the maximum S-T flow work or not? So, what is the natural greedy algorithm? If there exists a path from S to T where every edge has positive residual capacity, then send as much flow as you can along that path and repeat.

That is the natural greedy algorithm. Actually, we will repeat. So, let us replace 'if' with 'while,' and we have seen that in our example, this algorithm actually worked and gave an optimal S-T path. But we claim that this natural greedy algorithm does not always output the maximum or a maximum S-T flow. So, let us prove this claim. To prove this claim, I need to come up with a counterexample.

That is an example of an instance of network flow where the greedy algorithm fails. So here is a counterexample. So, as we can see, the maximum flow of this counterexample from s to t is 2. We can send 1 unit of flow from s to a to t and 1 unit of flow from s to b to t. So, the value of the maximum s-t flow in the example above is 2.

Now think of a run of the greedy algorithm. And in the first iteration, greedy chooses a path from S to T, which is S to A to B to T, and sends one unit of flow. The greedy algorithm sends as much flow as you can along this path, which is 1. And so, if I now redraw the graph with residual capacities, that is how it looks. You see, there is no s to t path.

Such that every edge in the path has positive residual capacity. So the greedy algorithm terminates after sending 1 unit of flow from S to T, although the maximum flow that can be sent from S to T is 2 units. Hence, the greedy algorithm is incorrect. One may think that instead of picking any s to t path, if we pick some path cleverly, then maybe we can make the greedy algorithm work. Maybe instead of picking any arbitrary path, if we pick the s to t shortest path, then maybe the greedy algorithm works.

So, take it as homework to prove that even if we pick the shortest s to t path and follow the natural greedy algorithm, then also the greedy algorithm may fail to find the maximum s to t flow. And the problem is, you see, even in the fast iteration, the greedy algorithm may pick a path which can never lead to an optimal solution. So the idea is to have a mechanism to reverse the flow along some paths or along some edges. So we will see that we will make slight changes in the greedy algorithm, and it will work. We will just allow the flows to reverse along the edges.

So, to do this systematically, we will introduce the concept of a residual graph. So what is the residual graph? Given a flow network G equal to (V, E) and a flow f, The residual graph G_f is a graph on the same set of vertices, but we have a different set of edges. This is defined as follows.

It has two kinds of edges. The first one is called a forward edge, so for all edges uv in the input network graph where the flow value is strictly less than the capacity, we have an edge from U to V in the residual graph with the residual capacity of the edge uv, which is $c_{uv} - f_{uv}$. The second one is backward edges.

For all edges u, v in the given graph which carry some positive amount of flow, we have an edge from v to u in E_f. with capacity same as the flow value of u to v. So, in the next class, we will see how we can use this residual graph to systematically allow edge reversal or flow reversal along the edges, and we will also see some examples of residual graphs to make sure that we have understood this concept very clearly. This concept of residual graph is central in the design of many algorithms for computing maximum flow.

So, let us stop here. Thank you.