Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 21

Welcome to the 21st lecture of the second-level algorithms course. In the last class, we have seen the Dinic's algorithm. In this class, we will analyze the runtime of the Dinic's algorithm. Okay, so let's begin. The first claim that we will prove is the following lemma. Let δ_i be the distance of T from S in the residual graph at the beginning of the i-th iteration. δ_i is strictly less than δ_{i+1} for every i. That means, in every iteration, the distance of t from s is strictly increasing.

Again, by distance, we mean the number of edges involved in the shortest path from s to t. But the maximum distance of t from s if s and t are still connected can be at most n minus 1. So, this lemma gives us the corollary that the number of iterations of the Dinic's algorithm is at most $n-1$. because the algorithm terminates when there is no path from s to t. So, this is because the algorithm terminates when there is no path from s to t in the residual graph. So, all I need to do is prove this lemma. So, proof of lemma. But before that, let me clarify that the Dinic's algorithm finds the blocking flow not in the residual graph, but it computes the blocking flow in the layered graph. So, that is a very important point.

So, let me highlight it, note. Every iteration of the Dinic's algorithm. We compute a blocking flow in the layered residual graph. This is very important, not the residual graph, which I may have said before, it is the layered residual graph of that iteration. Let me highlight not in the residual graph. So, this is very important So you see that the concept of a layered graph was not used in the algorithm of the Edmunds-Karp algorithm. It was used only in the analysis of the Edmunds-Karp algorithm.

On the other hand, the concept of a layered residual graph is directly used in the design of the algorithm, in the working of the Dinic's algorithm. So, now let us prove that lemma. So, consider any arbitrary iteration i. and look at the layered graph. So, look at $G_f^L(i)$. So, this is the layered residual graph of the input instance at the beginning of the algorithm, at the beginning of iteration i. So, I have written i and the corresponding layered residual graph. So, S is in L_0. And these are the various levels. Suppose the distance of t from s in the i-th iteration is k, and there could be subsequent levels also. Now, here we are computing a blocking flow. So, a blocking flow means that there is no s-to-t path in this layered residual graph where, after sending the blocking flow in the path, every edge has greater than 0 residual capacity.

Recall in the Dinic's algorithm, we augment using a blocking flow of $G_f^L(i)$. Now, whenever we augment a flow, say along this path, So, in the next iteration, we will introduce many reverse edges because I am sending flow along So, in particular, in the next iteration, if I look at the shortest path or any path or the shortest path from S to T, how many edges does it contain? The shortest path from S to T cannot consist solely of the forward edges of $G_f^L(i)$.

So, that is the most crucial observation. So, all the edges introduced in the residual graph or layered residual graph in the next iteration are the backward edges with respect to $G_f^L(i)$. So, two important points. All the new edges introduced in the residual graph at the beginning of the (i+1)-th iteration are backward edges with respect to $G_f^L(i)$. Hence, the shortest path from S to T at the beginning of from s to t in the residual graph at the beginning of the (i+1)-th iteration must use at least one backward or cross edge with respect to $G_f^L(i)$. Now, let us go back to the pictorial diagram of the layered graph.

Now think of any path from s to t now if it uses any backward edge or any cross edge then that path using that path we cannot reach t from s using k hops that means the distance of t from s in the next iteration that means in the layered graph $G_f^L(i+1)$ is strictly more than k that is δ_i which is k this is less than δ_{i+1}. So, this concludes the proof of the lemma and thus the distance of t from s is strictly increasing in every iteration. Hence, the Dinic's algorithm makes at most $n-1$ iterations. This concludes that the first part of the running time. Next, what we need to show is that there exists a more efficient algorithm to compute a blocking flow in the layered residual graph. So, let us see that next.

We go of m n time algorithm to compute an S-T blocking flow. The first observation is that $G_f^L(i)$ is a directed acyclic graph. with source with s as a source and t one of its sink. So, observe that the layered residual graph is directed as cyclic graph. Such graphs are abbreviated as DAG. Why? Because edges only go from layer to a next layer okay so there cannot be any cycle because there is no back edge or cross edge so the algorithm is as follows we start our depth first search from s till we reach a sink node or t we perform depth first search from S till we reach T or any other sink node, say w, not equal to t. Sink node means a node which has only incoming edges but no outgoing edges. So, there could be two cases, the depth first search will terminate if it reaches t or if it reaches sink node. the depth first search which is abbreviated as DFS reaches t. In this case, the sequence of vertices in the DFS stack.

Recall, we use a stack data structure for performing a depth first search in any graph. So, if we reach t, then we look at the stack corresponding to the DFS and that sequence of vertex corresponds to a s to t path. we send as much flow as possible along this path This makes at least one edge saturated. Because we are finding a blocking flow, we cannot use saturated edges mode.

So, we remove the saturated edge from the layered graph and repeat the whole process. repeat the dfs search promise the second case case two in dfs search we it ends in the second case the dfs search ends at a sink node say W which is other than T. You see, if there is a sink node W which is other than T, then W cannot be part of any S to T path. In particular, we cannot send any flow from S to T via W.

So in this case, we simply delete w along with all the edges incident on it. Then we repeat, then we restart DFS from S. We continue this process until S has any outgoing how much time does any iteration takes because we are terminating the depth first search whenever we are finding a sink or the sink node t or any other sink node that means the length of the stack is a path from s to w the sequence of vertices in the stack is a path from s to w. every iteration takes $O(n)$ time instead of $O(m+n)$ for standard DFS because we are not running DFS fully. We are terminating whenever we are finding the first sink node.

So, that is why every iteration takes $O(n)$ times; in every iteration, we are removing at least one edge from the layered residual graph. So, the number of iterations can be at most the number of edges m. So, the overall runtime of our algorithm for finding a blocking flow in the layered residual graph is $O(mn)$. And this makes the runtime of Dinic's algorithm $O(mn^2)$.

The number of iterations is at most n, and the runtime of every iteration is $O(mn)$; thereby, the total runtime is $O(mn^2)$. So, let us stop here. Thank you.