Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 05

Lecture 25

Welcome to the 25-th lecture of the second-level algorithms course. In the last lecture, we saw the pseudocode of the push-relabel algorithm. In this lecture, let us begin with an example of the execution of the push-relabel algorithm in a toy example to ensure that we all fully understand the working of the push-relabel algorithm. That means, how does the algorithm work? After that, we will see why the algorithm is correct and what the running time is.

So, let us begin. So, let us take an example of max flow. So, suppose this is the input instance. So, we initialize the height function and the preflow value as follows. So, there are 4 vertices, so n equals 4. The height of s is initialized at n, which is 4, and the height of every other vertex is 0. Then, the preflow is initialized as all the outgoing flow from s carries

the full amount of flow, and the other edges carry 0 flow, okay. So, what is the residual graph? Let us draw it. So, this is G; let us call it G^0. And this is G_f, the residual graph.

Let us write 0 in the superscript. So, S to A and S to B edges are saturated. So, they do not appear in the residual graph. The corresponding reverse edges appear. So, this is how after initialization the residual graph looks like.

We can see that all edges go either from height 0 to height 0—that is, the edges from a to t, b to a, b to t—or from height 0 to height 4—that is, from a to s, b to s. We ask: Are there any vertices having excess flow other than s and t? Yes, there are two vertices, a and b, which have an excess flow of 1. Among all vertices having excess flow, we are supposed to pick a vertex having maximum height. a and b have the same height, so we can pick either; let us pick a. And I am looking for any downhill neighbor of A. What is a downhill neighbor?

That means a neighbor whose height is 1 less than the height of A. So, does A have any downhill neighbor? A has only two outgoing edges. One is from A to t, which is at the same height, and the other is to s, which is at height 4. So, it currently does not have any downhill neighbor. So, we cannot push the excess flow at A. Therefore, we increase the height of A by 1.

So, the height of a becomes 1, the height of other vertices remain the same, okay. So, in the next iteration, so this also is G_f because preflow value has not changed, we have not made any push operation, we have only relay built. So, the residual graph remains same, but now among the two vertices having excess flows namely A and B, A is has the maximum height and A has a downhill neighbor namely T.

height of T is 0. So, we push the excess flow at A and the minimum of excess flow at A and the minimum of the residual capacity these two thing quantities both bound the amount of flow that we can push both are 1 in this case. So, I push 1 unit of flow along the edge A to T. Then the resulting graph look like, resulting preflow is the following. This is the graph G, S to A carries 1 unit of flow, S to B carries 1 unit of flow, A to T carries 1 unit of flow.

these are capacity values all 1, height of S is 4, height of A is 1, height of B is 0, height of T is 0, ok. So, then how does the residual graph look like? Now again ask does there exist any vertex having excess flow? The answer is yes. B has an excess flow of 1.

Then again ask does B have any downhill neighbor? A has 3 neighbors. S whose height is 4, A whose height is 1, T whose height is 0, but the height of B is 0. So, B does not have any downhill neighbor. So, we increase the height of B by 1.

So, the new height of B becomes 1, the height of other vertices remain the same. Again in the next iteration, so this is GF 3 also. We ask, is there any vertex having excess flow? Answer is yes, B. Does it have any downhill neighbor? The answer is again yes, namely T. We push as much flow, as much excess flow along the edge B to T as possible.

So, we push one unit of flow from B to T. And then this is how the flow graph now look. Now let us draw the residual graph. Again ask, does there exist any vertex having excess flow other than s and t? Answer is no. So, the preflow is a flow and hence the algorithm terminates.

Okay, so now that the working of the algorithm is clear, we can code it, and now we see the proof of correctness of the algorithm—why the algorithm always outputs a maximum

ST flow for every instance. First, we claim that the invariant is always maintained by the algorithm. So, we have seen that the invariant is maintained before entering the while loop, and whatever changes we do in the while loop do not violate the invariant. So, first claim. The invariants 1, 2, and 3 are maintained.

After every iteration, proof. We have already observed that the initialization ensures that the invariant holds before entering the while loop. In every iteration of the while loop, we perform either a push operation or a relabel operation.

So, case 1: if a push operation is performed along an edge uv in the residual graph. Then, you see, in this iteration, we are not changing the height of any vertex. So, all the edges that were present in the new residual graph and that were present in the original residual graph, the invariant continues to hold.

Of the residual graph after this iteration that were present. Before this iteration, the invariant was satisfied. You see, there are three conditions in the invariant. The first two are regarding the height of s and t, which we are not changing. So, if they were satisfied at the beginning of the iteration, they continue to hold after this iteration.

Actually, the height of s and t is never changed throughout the run of the algorithm. So, the only condition that we need to ensure is satisfied is that the edges go only uphill or at the same level, and if it goes downhill, it can go down by at most one level. So, if this is the residual graph after the i-th iteration and if this is the residual graph after $i-1$ iterations, then for every edge u to v of G_f^i (the residual graph after the ith iteration) that was present in the residual graph after the $i-1$th iteration, because the height of no vertex has changed—in particular, the height of the vertices u and v have not changed. Then the invariant is satisfied for this edge.

Now, are there any new edges added? Are there any new edges that are in G_f^i (the residual graph after the ith iteration) which were not there in the residual graph after the i minus 1th iteration? Yes, because we are pushing—so let us call it u prime v prime. So, we are pushing the flow in the push operation along the edge u to v. Now, if it is a forward edge, then a reverse edge may appear in the residual graph; if it is a reverse edge, then a forward edge may appear. So, by pushing some flow along the edge uv, the edge in the opposite direction vu

May appear in the residual graph after the current iteration. Even if it was not present in the residual graph after the previous iteration. But because we are pushing flow along the edge u to v, that means that the $h(u)$ must be the $h(v)+1$.

Since the height of u must be the $h(v)+1$, push is only allowed along the downhill edge. The edge v to u goes uphill and thus satisfies the third invariant. So, this shows that if we perform a push operation and if the invariant was true before the beginning of this iteration, the invariant continues to hold true after this iteration. The second case is we perform a relabel operation at vertex v, that is, we increase the height of vertex v by 1. Okay, so now again, invariant 1 and 2 continue to hold because they say that the height of s must be equal to n, the height of t must be equal to 0, which we are maintaining throughout the run of the algorithm. It is only the third invariant that we need to check. So why we perform a relabel operation at v? Since we have performed a relabel operation at v, it did not have any outgoing edge to W such that the height of w is $h(v)-1$. Recall if v has an excess flow and then only we are picking such a vertex of maximum height and first we are checking if there is any downhill neighbor. If there is any downhill neighbor we push as much flow as we can along that along that edge otherwise the vertex v does not have any downhill neighbor that means all its outgoing neighbors has height same as v or above so after increasing the height of v by 1, v cannot have any neighbour w such that $h(w) \leq h(v)-2$ ok. So, we can replace this equal to with less than equal to. Why? Because if it has a downhill neighbor then the height of downhill neighbor can be at most 1 less than and because the invariant does not, because the invariant holds before the beginning of this iteration height of all those neighbors must be at least height of v minus 1. So, this there is no such neighbor with this condition.

But this is exactly invariant three recall What was invariant 3? It says that for every h from v to w, height of w must be at least height of v minus 1, which is a restatement of this statement, which says that we cannot have a neighbor w whose height is at most 2 less than the height of v. So, now we have proved that invariant 3 also holds. So, this concludes the proof of the lemma that irrespective of whether we perform a push operation or reliable operation in the current iteration, if the invariant was true before the beginning of this iteration, invariants continue to hold after the iteration.

Because the invariant was true at the beginning of the while loop after initialization, and this while loop maintains the invariant, the invariant continues to hold true, ok. So, next in the next lecture, we will see how using this, we can bound the number of iterations of the while loop, thereby proving that it cannot be the case that the while loop does not

terminate. If it has to terminate, but if it terminates, then it can terminate only with a flow. And because it is maintaining the invariant, thereby maintaining the optimality condition—that is, t is disconnected from s when the algorithm, the while loop, terminates with a flow—it must be a maximum s-t flow. So, this part, bounding the number of times that the while loop iterates,

we will see in the next lecture. Thank you.