Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 09

Lecture 44

Welcome to the 44th lecture of the second-level algorithm course. In the last lecture, we saw a high-level overview of Edmond's blossom algorithm. In this lecture, let us understand that algorithm with a schematic diagram and define the missing pieces, okay? So, let us begin. First, let us define what a flower is.

So, a flower is a structure like this, where the first part is an alternating path. I am using black color to denote the edges of the matching M and blue color for the other edges. So, the first part, which is called a stem, is an alternating path. That means the edges of this path alternate from out of matching to within matching and so on. Alternating path starting with an unmatched vertex.

So, the first vertex is an unmatched vertex. And the end vertex, of course, is a matched vertex. Starting with an unmatched vertex, every other vertex of the alternating path is matched. So, this structure is called a stem, and the other part is called a blossom, which is an odd cycle of length k with $k/2$ floor of $k/2$ matching edges. Such a structure is called a blossom. So, edges along the cycle must alternate except at one vertex, which is why it must be an odd cycle, ok? So, such a structure is called a flower. So, now, let us see a schematic diagram of the execution of Edmond's blossom algorithm.

The Edmond's Blossom algorithm starts with marking all unmatched vertices as even. So, all these vertices are marked even, and they are unmatched. They serve as the roots of the breadth-first search forest that we will be building. We first enqueue all these unmatched marked vertices after marking them as even. In every iteration, we dequeue them.

So, and then we see all its neighbors, ok? So, if any of the vertices has a... So, this is a vertex, say u, and this vertex u has any neighbor which is marked even in the first iteration itself, then this edge—the both endpoints of this edge—are unmatched vertices. Both endpoints of the edge are unmatched vertices, and hence this edge itself is an

augmenting path. So, even in the first iteration, it can happen that u has a neighbor v, which is marked even and belongs to some other tree, which was the third case in our 4 cases of Edmond's Blossom algorithm and in this case we claim that we will find an m augmenting path in this case this age itself is an m augmenting path. So, if I get an m augmenting path the algorithm stops and returns the m augmenting path recall we are only focusing on the job of finding an m augmenting path or concluding correctly that the current matching is a maximum matching. So, assume that in the first iteration there is no such edge because if there is some such edge then we have we are done with our task. all its neighbors are unmarked, we mark them odd because they are at odd levels of the 3, the level of roots are level 0, the next vertices are at level 1 and so on.

So, now, you recall if we discover an unmarked vertex vertices, if v is unmarked when we are processing the edge uv, then v is marked odd. So, suppose this is v. is marked odd and I do not we do not enqueue v, we do not enqueue any vertex marked odd, we look at their matched vertices, mark them even right and then enqueue these even vertices. Similarly, maybe for from this tree this is how the structure look like. So, now suppose we are now exploring this vertex let us call it u, let us remove the previous labels and now suppose I encounter an edge which is already marked even.

So, these are odd, these are again even. Then we can see that we have found an M augmenting path which I am marking indicating using green This is an alternating path edges alternate from within matching and outside matching and both the endpoints are unmatched vertices. Such an path is called an aim augmenting path. So, if the other vertex v belongs to an different tree of the BFS forest that we are building and marked even, then we have found an aim augmenting path.

The other case is if the vertex v is marked even, but it is in the same tree of the BFS tree where u belongs. Then, you see, we have obtained what we called a flower: this is the stem, and this is the blossom. So, in this case, we have got a flower. Another important point that I did not mention in the pseudocode: if the queue, which holds even vertices, becomes empty— then the algorithm outputs that the current matching is a maximum cardinality matching, OK.

So, now, let us discuss how the algorithm should proceed if we get hold of a flower. has discovered a flower, which is a stem union blossom. Think of S and B as the sets of edges: S is the set of edges of the stem, and B is the set of edges of the blossom. We

replace M with M' equal to M symmetric difference S. Let us understand this pictorially. Suppose this is the stem; let us draw a stem. This is how a stem looks like.

It starts with an unmatched vertex and is attached with a blossom, which is an odd cycle. Edges alternate between inside matching and outside matching, except at one vertex because it is an odd cycle. So, this is the stem. So, this is our stem, and I want to get rid of it. So, what do I do?

I drop the matching edges from the stem from the matching and include the alternating edges in the matching. you see that in the stem the number of matching edges is the same as the number of edges which do not belong to the matching. So, this operation is nothing, but $M \Delta S$ and we have cardinality M' is the same as cardinality M. So, both m and M' has the same cardinality and instead of trying to find an M-augmenting path we will now find an M' augmenting path and if we get an M'-augmenting path we will use it to get another matching whose size is 1 more than the size of M' which is the same as 1 more than the size of m that is how we make progress ok.

So, and this is the blossom. b. So, now, when we get a flower we change m to get tree of stem and hence without loss we have we can assume that we have discovered a flower with empty stem. This implies that the blossom B contains an unmatched vertex. Indeed you see after changing the matching M with M' this particular vertex of the blossom is unmatched in M' ok. Now, the algorithm proceeds by recursively calling it on the graph $G/_B$ with the matching $M'/_B$.

Let us recall our algorithm takes two things as input: one is a graph G and a matching on that graph. So, we recursively call it. So, let us understand what the graph $G/_B$ is and how it looks. So, suppose this is my input graph G and this is my blossom B; then, in G mod B, this entire blossom acts as a single vertex, and the edges—whichever edges are incident on any vertex of this blossom B. So, suppose there is an edge between x and y, where y is part of a blossom B in G; then, in $G/_B$, we have a vertex corresponding to the blossom, and for every edge x y where x does not belong to the blossom and y belongs to the blossom, we add an edge between x and the vertex b corresponding to the blossom B. So, this is how we construct $G/_B$. The same applies to $M/_B$. Another way to view it is you simply look at those matching edges in M which are not part of B. So, that matching is M mod B. So, in the graph $G/_B$, we have a super vertex—I am just calling it a super vertex for the sake of understanding, but it is actually a normal vertex in the graph $G/_B$. Let us call the vertex v. Then, for every edge (x, y) of the graph G, with x not in the blossom B

and y in the blossom B, we have an edge between x and the super vertex corresponding to the blossom in the contracted graph. Also, observe that the number of vertices of the contracted graph is the number of vertices of the original graph minus the number of vertices in the blossom plus 1. Now, a blossom is an odd cycle with size at least 3. So, then this is strictly less than the cardinality of B, since the cardinality of B is greater than equal to 3 ok. So, in the recursive call the number of vertices is strictly decreasing and hence we are making progress towards the base case of the recursive call ok. And what is the M/B? M/B is simply $M \setminus B$ ok. You remove the edges of the matching which are part of the blossom and the remaining edges are M/B ok.

Now, let us see why it is enough to run the algorithm on the contracted graph that is because of this important lemma. M is a maximum cardinality matching in G if and only if M/B is a maximum cardinality matching in G/B. So, if the recursive call tells us that M/B is a maximum cardinality matching in G/B because of this lemma we can conclude that M also was an maximum cardinality matching of G. Moreover given an M/B augmenting path in G/B, we can construct an M augmenting path in G in polynomial time. if the recursive call returns an M/B augmenting path in G/B, thereby proving that M/B was not a maximum cardinality matching in G/B, we can use that path to compute an M-augmenting path in G in polynomial time and return that. This justifies the recursive call. So, we will prove this lemma in the next class which will prove the correctness of the algorithm. So, let us stop here.

Thank you.