Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 09

Lecture 43

Hello, welcome to the 43rd lecture of the second-level algorithm course. In the last lecture, we started discussing the maximum matching problem in general graphs and also stated the Tutte-Berge theorem. In this lecture, we will see the Edmond's blossom algorithm to find a maximum cardinality matching in an arbitrary graph. in polynomial time, thereby proving the Tutte-Berge theorem, OK. So, let us begin.

So, in the Edmond's blossom algorithm, it is an iterative algorithm, as I said. So, we start with an empty matching M and iteratively compute either a matching M' of cardinality same as M and an M'-augmenting path P or a matching M' with cardinality M' being the same as the cardinality of M and a subset of vertices. with cardinality M', which is the same as cardinality M, being at least the Tutte-Berge So, if we find an augmenting path P, an M'-augmenting path, then we use P to get another matching of cardinality one more than the cardinality of M, and we repeat the process. Otherwise, we get hold of a subset which proves that we have a maximum cardinality matching. The current matching M is a maximum cardinality matching.

Since the cardinality of any maximum cardinality matching is at most $n/2$, where n is the number of vertices in the graph. The above algorithm iterates for at most $n/2$ times, ok.

So, all we need to do is perform this task of finding an augmenting or a subset which proves that the current matching is optimal. That needs to be done in polynomial time, and that is why the Edmond's Blossom algorithm comes into the picture. So, this is the task. How do we accomplish it? So, now, our modified goal—this is our modified goal.

That means we are given a matching M and either come up with a matching M' (which can be the same as M) and an M' augmenting path P, or you come up with a subset U proving that the cardinality of M is the cardinality of any maximum cardinality matching. So, for that, we perform a modified version of BFS (Breadth-First Search). Because we

are looking for an augmenting path, and the endpoints of the augmenting paths are unmatched vertices, it makes sense to start our modified BFS from unmatched vertices. To perform that task systematically, we mark all unmatched vertices as even and enqueue them, ok.

In every iteration we dequeue a vertex which must be marked, which must be already marked even. So, throughout the run of the algorithm, we will mark some more vertices even, and those vertices will be at even distance from the root of the breadth-first search tree. We will maintain the following invariant: vertices marked even are at even distances from the root of the tree. in the breadth-first search forest that we are building Okay, and we will also mark some vertices odd; they will be at odd distance from the root. Vertices marked odd are at odd distances from the root of the tree in the breadth first search forest that we are building. So, this is the invariant that we will maintain. Let u be the vertex dequeued. Okay, and let us explore an edge from u, and u v be any edge incident on u. Then, depending on what v is, we will perform certain tasks.

The first one is if the vertex v is already marked. Odd, then we simply ignore the edge uv. That is, such an edge uv will not be a part of the BFS forest. BFS stands for breadth-first search. The second one, the second possibility, is if the vertex is unmarked, then observe that V must be a matched vertex. Since we have marked all unmatched vertices even at the beginning of our modified breadth-first search. Why modified? For example, look at the first step in breadth-first search. First of all, no vertices are marked even or odd. Second, we do not ignore any such edge here. We see we ignore the edge uv if v is an odd vertex labeled odd. So, now, you see if the vertex v is unmarked, then it must be matched. Let M(v) be the mate of v in m, that means the vertex with which v is matched. In this case, we mark v odd, M(v) even, and add the edges u v and vM(v) to our modified breadth-first search forest. So, we will see this with a pictorial diagram, but let us first see the exact algorithm. This is like a pseudo code.

The third one is the vertex v is already marked. Even and v belongs to some different BFS tree, than the BFS tree where u belongs. In this case, the path from the root of the BFS tree where u belongs to u followed by the edge u v followed by the path from v to the root of the BFS tree where v belongs forms an M-augmenting path. The fourth case is v is already marked even and v and u belong to the same BFS tree. In this case, we obtain a structure called a flower. We will see what a flower is. But a flower has two parts.

A blossom. And an optional stem; the stem is optional and it may not have one. We modify M suitably to get rid of the stem, if any, and thus we are left with a blossom B. Now, the idea is to recursively find an augmenting path in the graph G/B, OK? So, this recursive call either gives an augmenting path in the graph G/B or it gives a subset of vertices in G/B which shows that the matching is an optimal one. So, we use the output of the recursive call to either find an augmenting path or declare that the current matching is an optimal one. So, this is the high-level pseudocode of the algorithm. In the next lecture, we will see the working of the algorithm with a schematic diagram. We will define a flower, what its blossom is, what its stem is, and how to modify the current matching suitably to get rid of the optional stem. Why or how we can use the output of the recursive call to either output an augmenting path or declare that it is a maximum matching, OK?

So, let us stop here. Thank you.