Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 10

Lecture 48

Welcome to the 48th lecture of the second-level algorithm course. In the last class, we finished our discussion of Edmond's blossom algorithm, then we started wrapping up the discussion using the pseudocode of Edmond's blossom algorithm. We have seen the iterative procedure which calls for the subroutine to find an augmenting path or conclude the current matching M is a maximum cardinality matching. In this lecture, let us see the pseudocode of the augmenting procedure, OK?

So, let us begin. Pseudocode of augment subroutine. So, it is a recursive procedure. So, we begin with marking all unmatched vertices even and enqueue them while the queue is non-empty, dequeue a vertex Now, for all edges incident on v, we perform this task. Let us call this vertex u, then for all edges u v, we perform the following task. If v is marked even and belongs to are different trees in the BFS forest, then the tree where u belongs, then we have found an augmenting path, and we return that. Then we return m and the augmenting path found. Else if v is marked even and belongs to the same tree in the modified breadth-first search forest. This should be then where u belongs Then we have found a blossom. With say, we have found a flower with say stem S and blossom B. Then we first replace M with the symmetric difference of M and S. Then we call the augment subroutine recursively on the graph $G\,B$ with the matching mod B. If the above recursive call. returns an $M\,B$ augmenting path P, then we compute an M-augmenting path Q from P in polynomial time and return Q, return M comma Q. Else, the recursive call returns that $M\,B$ is maximum cardinality matching in $G\,B$. Else, that is in this case the recursive call returns that $M\,B$ is a maximum cardinality matching in $G\,B$, then we return that M is a maximum cardinality matching in G. Ok.

So, then this is what we do in this else if case else if in this else if ladder. So, if v is marked even and belong to the different tree in a BFS forest that we are building then we have found an augmenting path that we return if it is in the same tree and marked even

then we have found a blossom we perform a recursive call and based on that we perform. The third option is else if v is unmarked then what do we do? if this is the case then we mark v odd let $M(v)$ be the with which v is matched then we add the edge $(v, M(v))$ in the BFS tree where v belongs ok. And we enqueue $M(v)$ and mark $M(v)$ even. The other case when is marked odd that we ignore as we have discussed before. So, this is how we process the edge u v and this finishes the for loop and this also finishes the while loop. So, this is end of else if this is end of for loop and this is end of while loop. Observe that in the for loop or while loop.

The algorithm may find an augmenting path and return it, or it may discover that M is a maximum cardinality matching because of the recursive call and return. So, if the execution comes outside the while loop, that is when the queue is empty, then as we have already proved, if the queue becomes empty. Then our current matching M must be a maximum cardinality matching. Then return that M is a maximum cardinality. Matching in G. So, this is the augment procedure.

So, this finishes our discussion of Edmond's blossom algorithm for finding a maximum cardinality matching. Next, we start another important topic, which is called order statistics. So, what is the input here? Input is an array of n integers. And an integer k, and output is the k-th smallest integer.

In the array A, OK. So, let us first see what nicely we can do and some special cases. So, we can compute. The minimum integer of A with $n-1$. Comparisons, and this is optimal.

It can be proved that this is optimal $n-1$ comparisons are necessary in the worst case or not only worst case in any case is necessary to compute the minimum by any algorithm. So, now let us find out the case for k equal to 2. I want to compute the minimum and second minimum.

The naive approach is to find the minimum with $n-1$ comparisons. So, for simplicity let us count only the number of comparisons as our cost of the algorithm. So, the an obvious approach is to find the minimum first with $n-1$ comparisons and then find the minimum of the remaining $n-1$ integers using $n-2$ comparisons. This makes $2n-3$ comparisons in total can you do better. and the answer is yes. Here is a nice approach we pair up the elements the n integers with into $n/2$ pairs and compare every pair of elements. So, by $n/2$ comparisons, we get rid of the $n/2$ integers in the first iteration. Then, in the next iteration, we again pair up these elements and continue this process. We again pair up the $n/2$ ceiling. I strongly recommend that, to get the main idea, you assume that n is a power

of 2. So, that we do not have to worry about the ceiling and floor of $n/2$. Once you understand the algorithm completely, then the use of floor and ceiling will be clear to you. So, you again pair up these integers into pairs and repeat the process.

Pictorially, if these are the elements, we first compare a_1 and a_2. And whichever is the maximum, let us call it the winner, that goes to the second level. a3 and a4, the winner of a_3 and a_4, which is the maximum of a_3 and a_4, goes to the second level, and they are compared in the next layer. This is how a binary tree is binary search tree is performed or, ah, becomes apparent. So, the height of the search tree, let us call this a search tree, is $\log_2 n$, where height is the number of edges in the root-to-leaf path. All the leaf nodes are at the same level.

The root node is the maximum. You can check that the total number of nodes in this tree, the total number of non-leaf nodes in the tree, is $n-1$, which is the number of comparisons. The number of comparisons is the number of non-leaf nodes in the search tree, which is $n-1$, since every non-leaf node has exactly 2 children, and the number of leaf nodes is n, ok. So, take it as a nice exercise in graph theory that in any tree, or any binary tree, a rooted binary tree where every non-leaf node has exactly 2 children, the number of internal nodes (non-leaf nodes) is 1 less than the number of leaf nodes. So, using this, with $n-1$ comparisons, we have found the maximum, which is not a surprise because the $n-1$ bound is a tight bound, but in the next picture, we will see how. Using this search tree, we can find the second maximum with far fewer comparisons than the $n-2$ number of comparisons we needed in our naive approach, ok.

So, let us stop here. Thank you.