Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 03

Lecture 13

Welcome to the 13th lecture of the second-level algorithms course. We have been studying Fibonacci Heap for the last couple of lectures, and in this lecture, we will conclude our study of Fibonacci Heap. We have shown in the last class that for any tree, if you look at a Fibonacci Heap node with degree k and ask what is the number of nodes in the subtree rooted at that node. then that is at least f. There, we have assumed one claim which we will prove now, thereby finishing that incomplete proof.

So, let us begin. So, the missing piece of last lecture's proof was this claim. that f is greater than or equal to f for every integer k greater than or equal to 0. Recall what was s_k ?

It is the minimum number of nodes in a subtree rooted at a node of degree k in any Fibonacci Heap. So, we will again prove this using induction on k. So, the base case for k equal to 0 and 1, it holds obviously because LHS is 1. is greater than or equal to 1, and RHS is less than or equal to 1 for k equal to 0 and 1. Sorry, for k equal to 0, LHS is equal to 1, and RHS is also equal to 1. So, let me fix it.

For k equal to 1, if the degree is 1, that means it has 1 child, and so LHS is greater than or equal to 2, and RHS is 2. So, we have LHS greater than or equal to RHS. Why greater than or equal to 2 for LHS? Because the root node has degree 1, but the child node may also have degree 1, and so on.

So, the number of nodes in a subtree rooted at a node whose degree is 1 is at least 2. So, the induction hypothesis assumes for all integers from 0 to $k-1$. Now, the inductive step Now, here is a node of degree at least k. So here is $y_1, y_2, \ldots, y_k$.

So, s_k is the minimum number of nodes in such a subtree. So, again, let us count the root node and the y_1 separately. So, we write s_k is greater than or equal to 2 plus the number

of nodes in the subtree rooted at y_i for $i=2,\ldots,k$. The degree of y_i is at least $i-2$, as we have proved in the lemma.

So, now use we use induction hypothesis. So, this is greater than equal to 2 plus now we use that f_1 is 1 and f_0 is 0. So, this is 1 plus this sum. Now, this we have proved before that this is f_{k+2}.

So, this concludes the proof of this claim. Now, recall what was our main thing that we need to prove, we have to prove that the maximum degree of any node in any n node Fibonacci heap is $O(\log n)$. Now, let us prove that final piece how proving this lemma prove that important result that we need. So, here is the that we get as a corollary the maximum degree of any node in an n node Fibonacci heap is $O(\log n)$, proof. Let x be any node in an n node Fibonacci heap okay we will show that x dot degree is $O(\log n)$. So, the number of nodes in the subtree rooted at x is at most n. So, what we have is n is greater than equal to the number of nodes in the sub-tree rooted at x and this number is greater than equal to s of $x.degree$.

Recall, s_i was the minimum number of nodes in any subtree rooted at a node of degree i. So, this number, the number of nodes in the subtree rooted at x, is at least s of $x.degree$. Now, we have shown that s of $x.degree$ or s_k is greater than or equal to f_{k+2}. So, this is greater than or equal to, so let us call this number k. So this is basically s_k, and s_k is greater than or equal to f_{k+2}, which is greater than or equal to ϕ^k. What we have is n is greater than or equal to ϕ^k. It is $k \leq \log_\phi n$.

Hence, k is $O(\log n)$. So, this concludes the proof of our claim that the degree of every node in an n-node Fibonacci heap is $O(\log n)$. So, this concludes our discussion on Fibonacci heaps. So, our next topic is maximum flow in a network. So, let us begin that. Maximum flow in a network.

So, what is the scenario here? We have a directed graph G equal to (V, E). There is a special vertex called the source, and there is another special vertex called the sink, t. Every node or every edge $e \in E$ has a non-negative capacity c_e. Our goal

is to send maximum amount of flow from s to t. So, let us see an example. So, suppose I have this graph. Two special nodes source and sink. Source is denoted by s, sink is denoted by t. Later we will see that we can assume without loss of generality that there is no incoming edge to source and there is no outgoing edge from sink.

Each edge has a capacity which is some non-negative integer. So, this is how an input to a flow problem looks like. What are the motivations? There are lots of applications of maximum flow and we will see many of them. For a real world motivation, we can think of this network as a road network, where each node is a city and the edges are the highways.

The capacity is maybe the number of trucks in crores or in in in thousands or lakhs number of tracks trucks that can go from one end point of the edge to the other end point in say one hour that could be the capacity of each edge and we want to send the maximum amount of trucks from source is to destination t destination or sink okay so this is source and this is sink. Of course, we cannot send any amount of some amount of truck more along and along a road more than its capacity. So, one possible way is to send say suppose

units of truck from S to A, then those 5 units can be forwarded to A to B and B to T. So, this particular thing is called a flow and you see because of this flow the capacity of every edge is now reduced. They are called residual capacity. For example, the edge from S to A originally had the capacity of 5, now it is carrying a 5 amount of flow, so the residual capacity, the capacity remaining, the more capacity that this edge can handle is 5. On the other hand, the edge A to B.

Its capacity is 5 and it is currently carrying 5 units of flow. So, it cannot carry any more amount of flow. This edge is called is saturated by this flow. Okay, so let's see, can we send some more amount of flow? Yes, of course, there is another path.

S to C you send 2 units of flow, C to D you send 2 units of flow and D to T you send 2 more units of flow. This way you are now sending a total 7 units of flow from S to T. You again see that because of the second flow the edge C to D has now become saturated, it does not have any residual capacity left. Can we send more amount of flow? Of course, yes.

So, the edge to C has a residual capacity of 18. So, we can send 3 more units of flow. So, make this 5. Then, this extra 3 units of flow, we can route it from C to D, then B to D. And then, d to t these three. So initially, it was carrying two units of flow; three more get added, so now its total capacity is five.

And it is not clear whether we will be able to send more units of flow. Later, we will see techniques to verify whether this is the maximum amount of flow or not. In this particular

example, this indeed is the maximum flow. So, we are sending 10 units of flow from S to T, which is indeed the maximum

amount of flow that can be sent from S to T. So, this is the intuitive idea of this problem. To study it formally, we need to define the notion of flow formally. So, let us do that. A flow is a function from edges of the graph to non-negative real numbers, satisfying the following properties. The first property is the capacity constraint for every edge.

In the graph, the flow that this edge is carrying—let us denote it by f(e)—is some number between 0 and the capacity of that edge, c(e). The second one is flow. conservation. It says that at every node except S and T, the total amount of flow into the node must be equal to the total amount of flow going out of the node. For every node V in Vg, the total flow in—which is the sum over all edges that go from some node into V—must be equal to the total flow out of

V, which is the sum of the flows of all edges leaving V. Okay. This is for all vertices in V except S and T, the source and sink. So, this is the flow, and now we have to define the value of the flow, which would capture the amount of flow sent from S to T. So, the value of a flow F is the total flow going out of the source node S, which is the summation Now, here is a believable lemma which says that the total flow going out of source S must be the same as the total flow going into sink T. Okay, so this is the proof. I leave it as homework. Let me just give you a hint. Proof hint: consider the flow into vertex V minus the flow going out of vertex V. Now, construct this sum and compute it in two ways.

One is for every vertex on the left-hand side; on the right-hand side, you consider for every how much it is contributing to this sum, to this expression, and then from that, you will get that the total flow going out of the source node S is the same as the total flow going into the sink node. So, then this value of the flow, which we aim to maximize, can also be defined as the sum of the flow values of all the edges going into T. So, with this, let us stop here. We will continue our study of flow in the next couple of lectures.

Thank you.