Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 02

Lecture 08

Welcome to the eighth lecture of second level algorithms course. In the last class, we have started studying Fibonacci Hips and we have seen examples of insertion and extract min. So, today let us begin with seeing the pseudocode of insertion and extract min. So let's call this procedure fibHipInsert. This procedure takes the Fibonacci Hip and a key called x to be inserted.

So what does h store? Recall h store the pointer to the minimum node in the root list. So, first with the key I create a Fibonacci heap node. So, $x.degree$ is 0. Let us recall how do we insert a key?

Insert it in the circular doubly linked list of the root list. So, I will create a singleton node with no child with key x and insert it in the head list or root list which is a circular doubly linked list. So, I am creating a degree 0 node with key x. So, $x.degree$ 0 its parent is null, its child is also null because it does not have any children. $x.mark$ is 0 or false and I will insert it in the root list and update the min pointer. So, I will check if h dot min

is null then in this case the Fibonacci heap is currently empty and h should create to the h should h dot min should point to the node containing x. So, then create a root list for each containing just So, here x is a node which is already created and it $x.key$ contains the key to be inserted and in this fib heap insert we are updating the other fields of the structure x and $h.min$ and think of H is also a structure which stores information like pointer to min and number of nodes in the current Fibonacci heap and so on. So, this one is instead of H we write $h.min$.

then $h.min$ is equal to x. Else the Fibonacci heap is currently non-empty. So, then we insert x in the root list. and then we update the min if the key of x is less than $h.min$. If $x.key$ is less than $h.min$ which is a pointer to the minimum node dot key, then $h.min$ is

equal to x. And finally, I update the number of nodes $h.n$ by adding it by incrementing it by 1.

So, this is the insertion procedure. This we have seen in the last class using examples. Now, let us move on to the pseudocode of extract mean procedure. So let us call the procedure FibHipExtractMin. which takes as argument the Fibonacci heap node which stores all the metadata and pointer to the minimum node and this procedure should delete the minimum key from the Fibonacci heap and return the minimum key.

So, let us first store the minimum node in a temporary variable which will be used to return it. So, if z is equal to null then the Fibonacci heap is empty. and in this case we do not need to return anything. So, we will proceed if $f.min$ which is z is not equal to null.

So, in this case what should we do? Let us recall we will remove z and all the children of z we will insert it in the root list first. So, for each child x of z, we add x to the root list of h and make the parent pointer of x null.

So, after doing this we remove z from the root list of h. ok and now we check if z is the only node in the Fibonacci heap then we need to update some metadata that if z is same as $z.\text{right}$ or $z.\text{left}$ then $h.min$ is null because z is deleted else we make $h.min$ point to some other node in the root list which we will update subsequently. say $z.\text{right}$ and now at this point $h.min$ is not pointing to the minimum node. So, we will update it.

We do consolidate. So, we will write another procedure called consolidate edge. We will make this update, the compression, and finally, we increase the number of decrease, not increase, because we are deleting.

So, decrease the number of nodes in the Fibonacci heap by 1. So, this is the procedure. This is the end of if, and then we return z. So, in the procedure consolidate, we will merge those subtrees or those trees in the root list which have the root node of trees with the same degree. So, that is the consolidate procedure.

So, let us see the pseudocode of it. So, let us call it consolidate. So, let me write pseudocode of Consolidate. This takes the Fibonacci heap as

The input first creates a temporary array of size equal to the maximum possible degree of any node in a Fibonacci heap with n nodes. So, let $A[0,\ldots,d]$ be $H.n$. $H.n$ is the number of nodes in the Fibonacci heap, and $d(H.n)$ is the maximum possible degree of any node in a Fibonacci heap of size n. We will prove later that $d(H.n)$ is $O(\log n)$. So,

once we know this bound, we will replace $d(H.n)$ with $O(\log n)$. For now, let us write $d(H.n)$, but when implementing the code, we will replace $d(H.n)$ with $O(\log n)$. This is a new array.

I initialize all entries of this array to null. This array will store pointers to nodes, specifically pointers to roots of trees where the degree of the root node matches the array index. Now, for each node W in the root list of H, we do the following. We will perform the following steps. We will check if there is any node with the same degree as W, and if so, we will merge them.

For this, we use a temporary variable x, which stores a pointer to w, and d is $x.degree$. Now, while A[d] is not null, set y equal to A[d]. Here, d is the degree of w, and we check if there is any other node of degree d that we have already encountered and is currently in the root list. If such a node exists, its pointer will be stored in A[d]. If A[d] is null, it means we have not observed any node in the root list with degree d. If it is not null, it means there exists a tree in the root list where the root node has degree d. So, let me write that as another node with the same degree as x is present in the root list. So, if that is the case then you recall we merge them. So, let us merge them. So, first what we do that if $x.key$ is more than $y.key$, I will exchange the key so that I can assume without loss of generality that $x.key$ is less than equal to $y.key$.

So, then you exchange $x.key$ and $y.key$. And, then what will you do? So, the merging procedure let us perform this using a procedure or let us write another function let us call it fib hip link H Y X. So, it basically makes y a child of x. So, after this ad is null because we have merged two nodes of degree d and now we have one node of degree $d+1$. So, now I will check is there any node of degree $d+1$ in the next iteration that is why I have this while loop instead of an if condition.

So, it can happen that I have encountered a node of say degree 10 and I observe in that array A that I have already encountered another node of degree 10 then I merge them the degree of the merged node is 11. Now, it can be possible that I have already encountered another node of degree 11 and so on this can continue that is why I have a while loop instead of a if condition. So, this is the end of while loop and when that ends. So, A D is x and then so, this is the merging process.

So, after this, it is ensured that all, and this is the end of the end of the for loop also. So, ad is x. So, this is the end of the for loop. So, here now the root list has nodes of distinct degrees only.

So, next, what I will do, you see, you recall that. Before calling the consolidate procedure, $h.min$ was not updated properly. It was updated to something else. So, we need to update $h.min$, and how will we do it? We will scan the root list, which has at most dn nodes, and find out the minimum and make $h.min$ point to the minimum.

So, that is what we will do. So, let us start with $h.min$. to be, say, null. Now, I will scan the root list for i = 0 ,…, $d(h.n)$. First, I check if there is a node of degree i in the root list. So, if there is a node, then the pointer to that node is stored at index i in the array a. So, if a[i] is not equal to null, then if $h.min$ is null, create a

Root list for H containing just $a[i]$. OK, and then make $h.min$ point to $a[i]$ Else, that means $h.min$ is already pointing to something; the root list has some value. So, else we insert $a[i]$ into H's root list, OK, and then we update h dot min if needed if $a[i].key$ is less than $h.min.key$, then $h.min$ is $a[i]$, OK.

So, this is the end of else; this is the end of form, OK. Now, the only missing piece is this Fibonacci heap link, which simply makes Y the child of X. So, let us do that: Fib heap link H Y X. First, you remove Y from the root list of H, then make Y a child of X. For every node in the Fibonacci heap, we also store the degree of the node in that structure.

So, because now Y has one more child, I will increase the degree of Y by 1: increase Y dot degree by 1. OK, and mark of Y is false. Recall, mark is turned to true if a node has lost a child from the time it has been made a child of another node. So, in this process, Y has been made the child of X just now.

So, y dot mark is false. So, the use of this marking we will see in the next class when we discuss the decrease key operation. So, let us stop here. Thank you.