Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 02

Lecture 10

Welcome to the 10th lecture of the second-level algorithms course. So, till now, we have studied all the procedures of the Fibonacci heap, and in this lecture, we will study the amortized time complexity of those operations, okay? So, let us begin. So, these are the basic operations of the Fibonacci heap. And these are the amortized time complexities, okay?

Extract min and decrease key. We will show that the amortized time complexity of insertion is $\Theta(1)$, extract min is $O(\log n)$, and decrease key is $\Theta(1)$. Let us contrast this with the worst-case time complexity of these procedures. So, insertion is worst-case $\Theta(1)$, extract min is worst-case $\Theta(n)$, and decrease key is worst-case $\Theta(n)$. Proving these worst-case time complexities, I will give you as homework.

So, we will use the accounting method for analyzing the amortized time complexity of these operations. Okay, for that, we will assume this—which we will prove later—assume this fact: the maximum degree of any node in any n-node Fibonacci heap is $O(\log n)$. And actually, in the proof of this fact, we will see the use of Fibonacci numbers, thereby justifying the name of the data structure called Fibonacci, okay?

So, let us assume this and let us prove the amortized time complexity of these operations. So, we will maintain the following invariant recall in the accounting method money is deposited across various places in the data structure which is typically used to pay for future operations and also as you also recall that we can pay for any big O of one time operation with 1 rupee. So, we will ensure that every node in the root list and every mark node has rupee 1 stored with them ok. So, let us first begin with ah the charging procedure. So, we charge insertion rupees to extract min log n. So, let me let us be slightly more concrete and instead of assuming that the maximum degree is $O(\log n)$. let us assume that this is at most some c times log n for some constant c.

So, then this extract mean operation will charge c times log n and then decrease k we will again charge 2 rupees and then let us see how we pay for actual costs. So, let us consider any sequence of n operations where each operation is either insertion or extract mean or decrease key. So, we will assume we will prove that after every operation after paying for the actual cost the invariant still holds.

This is for every i say 0 also. So, we will prove this by induction on i. So, initially the data structure was empty, thus the invariant holds, ok. So, let us assume that the invariant holds at the i minus 1th iteration till the i minus 1th iteration. This is the induction hypothesis.

So, there are 3 choices for the ith operation. We will show that for any of the 3 operations, the invariant still holds after the ith operation. So, case 1: the ith operation is an insertion, ok. Suppose the ith operation is an insertion. Then what I do, you see, let us recall the insertion procedure: we create a node and simply insert it in the root list, update the H dot min node if necessary.

So, the actual cost of insertion is big O of 1, ok. We charge 2 rupees for insertion, we pay the actual cost with 1 rupee and store 1 rupee remaining. with the node, with the newly inserted node, ok. Let us recall the invariant. The invariant says that every node in the root list must have 1 rupee with it and every node in the

Every node that is marked must have 1 rupee on it. In insertion, we do not mark or unmark any node. So, the set of marked nodes remains the same before and after insertion; only one extra node is getting inserted in the root list, and we are storing 1 rupee with it. So, the invariant continues to hold after the i-th operation, ok.

So, now, case The i-th operation is an extract min, ok. Let us recall what the extract min procedure does. It deletes the $H.min$ from the root list, and then we consolidate the list. What does

So, if the root list currently has k nodes, the consolidate procedure takes O(k) time. So, suppose the root list has k nodes, ok. Then the consolidate procedure takes O(k) time.

And after the consolidated procedure let us assume that the root list has at most k' nodes. So, what is the time complexity for consolidation? The actual cost of consolidation which is basically the actual cost of extract means. is O(k) ok.

Now, you see we use the number of nodes in the root list was k and after consolidation it has become k'. That means, $k-k'$ nodes which were root node now has become non root node. So, let us observe that observe that $k-k'$ nodes has have become from being root nodes after (i-1)-th operation to non root nodes after i-th operation. So, we use the money stored in those $k-k'$ nodes plus the C log n amount of money to pay for the exact cost which is $O(k)$. We use the rupees $k-k'+C\log n$ which we charge this extract mean operation to pay for the $O(k)$ time actual cost. This is enough.

what we need to show is that to pay for some operation whose cost is $O(k)$ we must pay at least $k-k'$ rupees. So, we will show that the total amount spent $k-k'+C\log n$ must be at least k. So, this is enough. since what is the relationship between k' and C log n? C log n is the bound C log n is the maximum possible degree of any node on an n node binary n node Fibonacci okay. And after consolidation the root list has unique degrees that the nodes in the root list the degree of every node is different no 2 degrees of node there are no 2 nodes of same degree. So, that means, k' is less than equal to C log n this and hence $k-k'+C\log n$. is greater than equal to k. So, we are paying at least k rupees to pay for O(k) time operation which is enough ok. And you see that we are only using those money stored in the nodes which has become from root to non root. So, the nodes which were root node and now also root node their money is not touched.

Hence the loop invariant is maintained. Note that we have not touched the money stored at though at the nodes which where part of the root after both (i-1)-th and i-th operations. Hence, the invariant continues to hold since the set of marked nodes does not change in extract mean procedure Recall that some nodes are marked or unmarked only in the decrease key operation recall what was the marking scheme. We were we set the mark field of a node to be true if it is the first time it loses a from the time it has been made a child of another node.

So, in the decrease key recall after decreasing the key if the mini property gets violated that means, the value of the new key is less than the value of the parent node then the mini property gets violated and in that case we cut the link between the node and its parent and make this subtree rooted at that node to be part of root list and mark the parent if it is not already marked and if it is already marked then we have done a cascading cut and mark the last one if it is not true. So, in the extract mean the set of nodes which are marked does not change and loop invariant says that every node in the root list and every marked node must have rupees 1 stored in it.

So, because by induction hypothesis because after (i-1)-th operation every node in which are marked every marked node stores rupees 1 with it after i-th operation if it is extract min those nodes continue to store rupees 1 and those are the exactly the set of marked nodes. And, we have made many nodes from root to non root possibly many nodes and the amount stored in that node are used to merge the corresponding trees. So, hence this extract main process is also fully paid off. So, in the next class we will see how charging rupees 2 for decrease key is also enough to pay for the actual cost. So, let us stop here.

Thank you very much.