Illinois Institute of Technology
Department of Computer Science

Homework 5 Solutions

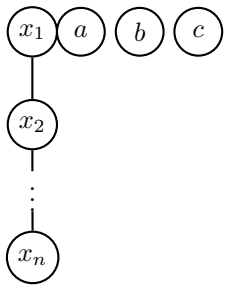
CS 430 Introduction to Algorithms
Fall Semester, 2010

1. **Problem 19.4-1 on page 526**.

   Solution: We can use induction to show that it is possible to create a Fibonacci heap consisting of one tree that is a linear chain of n nodes.

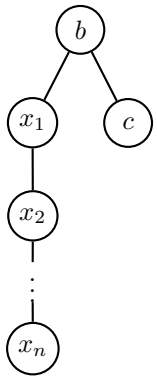
   Basis step: Suppose the heap is initially empty. If we insert a node into the heap we will now have a Fibonacci heap of one node that consists of just one tree that is a linear chain of one node.

   Inductive step: Suppose we are able to construct an n-node Fibonacci heap that consists of a single linear chain of n nodes, where $n \geq 1$. After applying the following series of operations, we can produce a $n+1$-node Fibonacci heap that is a linear chain of $n+1$ nodes.[1] First we insert keys a, b, and c into the heap such that $a < b < c < x_1$, where x_1 is the root of our n-node linear chain. This now gives us the following Fibonacci heap.

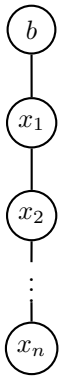


   Next we perform an FIB-HEAP-EXTRACT-MIN operation which deletes a, the current minimum key, and performs a CONSOLIDATE operation. This links nodes with keys b and c and then links that tree with the one whose root key is x_1, since they both have one child and therefore they both have a degree of one. x_1 will be a child of b since we already assumed $b < x_1$. So we now have the following Fibonacci heap.

[1] Alternatively, we could assume that we have a chain of $n-1$ nodes, where $n-1 \geq 1$ then prove that we can build a chain of length n. All we need to do in the inductive step is show that we can grow an existing linear chain by one. By starting with the basis step and repeating the inductive step we can create a linear chain of any length.



Finally, we perform a FIB-HEAP-DELETE operation on c, which removes it from the heap and leaves us with an $n+1$ node heap that consists of just a single linear chain of $n+1$ nodes.



2. **Problem 19-1(a) on page 527**.

   Solution: Line 7 cannot be performed in $O(1)$ actual time, since by Corollary 19.5 (page 526) we know that a node x can have $O(\log n)$ children. Even though the child list may be a doubly linked list and could be added to root list in $O(1)$ time, each child has a parent pointer that needs to be updated which may take $O(\log n)$ time total.

3. **Problem 19.2(e) on page 529**.

   Solution: Here we are looking for the worst-case actual (not amortized) running time. The worst case running times for MAKE-HEAP and MINIMUM are still $O(1)$. Since INSERT, UNION, and EXTRACT-MIN all consolidate the root list, the root list is always $O(\log n)$ in length. The worst case situation for INSERT, UNION, and EXTRACT-MIN happens when the new root list resulting from an insert/merge contains two nodes with degree 0 and a node with each possible degree from 1 to $D(n)$. This results in a runtime of $O(\log n)$ as CONSOLIDATE will have to merge all of the trees in the new root list. Hence the worst case runtime for all three of these operations that call CONSOLIDATE is $O(\log n)$.