CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 15

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we will look at Heaps and Heap Sort.

2 Heaps

A heap is a binary tree whose keys satisfy the following property:

Heap Property
The key of every node is greater than or equal to the key of any of its children.

Heaps are useful to implement priority queues, an abstract data type defined by the following two operations:

1. Insert(x): insert a key x into the data structure.
2. Max-Remove(): remove the largest key from the data structure and return it.

Priority queues are like queues but when we dequeue the highest priority element is retrieved first.

3 Representation of Trees

In general, trees can be represented explicitly or implicitly.

Explicitly: A node with k children is a record containing an array of k pointers and one pointer to the parent. Alternatively, we can keep two pointers: first to the first child and second to the next sibling.

Implicitly: We use an array. Consider a binary tree. The root is stored in $A[1]$, the children in $A[2], A[3]$ and $A[4]$ and in general inductively the left child of a node v stored in $A[i]$ is stored in $A[2i]$ and the right child is stored in $A[2i+1]$. For ternary trees for a node stored in $A[i]$ we store its three children at $A[3i-1], A[3i]$ and $A[3i+1]$. The advantage is that no pointers are needed which saves storage, however if the tree is unbalanced, i.e., some leaves are much farther away from the root than the others, then many non-existing nodes must be represented. For example, Figure 1 shows that an array of size 30 may be needed for a tree on 8 nodes.

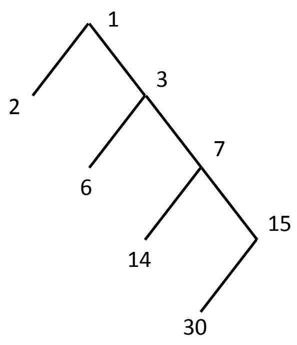

Figure 1:

4 Heaps Revisited

Heaps can be represented using Explicit or Implicit tree representation, however since we can ensure that heaps will be balanced we can assume the array is $A[1, 2, \ldots, k]$ where k is an upper bound in the max number of elements the heap will ever contain.

Remove Operation: By the heap property, the node with the largest key in a heap is the root $A[1]$. We remove it, take the leaf $x = A[n]$, delete it and put it in the place of the root, i.e., $A[1] := A[n]$ and $n := n-1$. We have two separate heaps and x at the root. We now propagate x down the tree until it reaches a subtree for which it is a maximum. First we find the max of children and if it is larger than x, then swap it with x. Inductively continue until either x becomes max for a subtree or when it reaches a leaf. The maximum number of comparisons is thus $2\lceil \log_2 n \rceil$.

Insert Operation: It is bottom-up as compared to Remove which is top-down. We increment n by one and insert x as the new leaf $A[n]$. We then compare the new leaf with its parent, and swap if the new leaf is larger than its parent. We continue inductively (for correctness) this process, promoting the new key up the tree until the new key is not larger than its parent or it reaches the root, The maximum number of comparisons is thus $\lceil \log_2 n \rceil$.

Overall we can insert and remove in time $O(\log n)$ per operation for heaps. However heaps are not useful for some operations like search.

5 Heap Sort

Heap Sort is another fast sorting algorithm although it is not as fast as quicksort in practice (though its worst case needs $O(n \log n)$ comparisons). Unlike merge sort, heap sort is an in-place sort.

First building a heap: Given an array $A[1, 2, \ldots, n]$ of elements in an arbitrary order, rearrange the elements so that the array satisfies the heap property. There are two ways: top-down and bottom-up, but we say only top-down which is simpler and more efficient. Consider scanning the array from left to right. The induction hypothesis is that the array $A[1, 2, \ldots, i]$ is a heap. The base case is trivial as $A[1]$ is a heap. The main part is to incorporate $A[i+1]$ into the heap $A[1, 2, \ldots, i]$ which is exactly the same as inserting $A[i+1]$ into the heap. The number of comparisons in the worst case is $\lfloor \log_2(i+1) \rfloor$. The total number of comparisons is thus $\sum_{i=1}^{n} \lfloor \log_2 i \rfloor = \theta(n \log n)$.

The rest of the sort: Since A is a heap, we know $A[1]$ is the max element. Thus we remove from A and put it at $A[n]$ and continue with $A[1, 2, \ldots, n-1]$. The overall running time is $\sum_{i=1}^{n} 2\lceil \log_2(n-i+1) \rceil = \sum_{i=1}^{n} \lceil \log i \rceil = \theta(n \log n)$. So the overall running time is $O(n \log n)$.

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*