CS 457, Data Structures and Algorithms I
Third Problem Set
Renju Radhakrishnan

November 12, 2016

1. **9.3-7**
   Describe $O(n)$-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$, determines the k numbers in S that are closest to the median of S

   Answer
   Given a set S of n distinct numbers, we select two values in between the k value and compare the values in S with those values. We can recursively call the algorithm until it finds the selected value. An easier algorithm is to find the median by using the median of median algorithm, then subtract that median from all values. Afterwards, take the absolute value of all the integers and find the smallest kth value in the array. Thus, completing in $O(n)$ time.

2. **9.2 Weighted Median**

For n distinct elements $x_1, x_2, \ldots, x_n$ with positive weights $w_1, w_2, \ldots, w_n$ such that

$$\sum_{x_i < x_k} w_i < 1/2$$

and

$$\sum_{x_i > x_k} w_i < 1/2$$

For example, if the elements are 0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2 and each element equals its weight (that is, $w_i = x_i$ for $i = 1, 2, \ldots, 7$), then the median is 0.1, but the weighted median is 0.2.

a. Argue that the median of $x_1, x_2, \ldots, x_n$ is the weighted median of the x_i with weights $w_i = 1/n$ for $i = 1, 2, \ldots, n$.

Answer

Given x_k as the median of $x_1, x_2, \ldots, x_n$, by the definition of medians, x_k is bigger than $(n+1)/2-1$ elements as seen from $\sum_{x_i < x_k} w_i < 1/2$. On the other side of the median, x_k is smaller than $(n - (n+1)/2$ elements, as shown from $\sum_{x_i > x_k} w_i < 1/2$. Thus, by definition of weighted medians x_k is a weighted median.

b. Show how to compute the weighted median of n elements in $O(nlgn)$ worst-case time using sorting.

Answer

In order to compute the weighted median of n elements, we sort the list then add the weights of the medians until the median is found.

```
def weighted-median(Set)
   int counter = 0;
   int weight = 0;
   while(weight + Set[i].weight < $1/2$) {
      weight = weight + Set[i].weight;
      counter++;
   }
   return weight;
```

c. Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm such as SELECT from Section 9.3

Answer

We can compute the weighted median using a $\Theta(n)$ median algorithm through calculating the median then recursively searching on each half of the list to obtain the weighted median as shown below.

```
weighted-median(Set) {
   int median = set.getMedian();
   List list1, list2;
   for(int i = 0; i < set.size(); i++) {
      if(Set[i] < median) {
         list1.append(Set[i]);
      } else {
         list2.append(Set[i]);
      }
   }
   if(list1.totalWeight > $1/2$) {
      weighted-median(list1);
   } else {
      weighted-median(list2);
   }
}
```

The ***post-office location problem*** is defined as follows. We are given n points $p_1, p_2, \ldots, p_n$ with associated weights $w_1, w_2, \ldots, w_n$. We wish to find a point p (not necessarily one of the input points) that minimizes the sum $\sum_{i=1}^{n} w_i d(p, p_i)$, where $d(a, b)$ is the distance between points a and b.

d. Argue that the weighted median is a a best solution for the 1-dimensional post-office location problem, in which points are simply real numbers and the distance between points a and b is $d(a, b) = |a - b|$.

Answer

We can show that the weighted median is the best solution for the 1-dimensional post-office location problem by choosing a p to minimize the cost, rewriting it as a function of $\text{cost}(p) = \sum_{i=1}^{n} w_i d(p, p_i)$. We can rewrite the cost function as $\text{cost}(p) = \sum_{p_i<p} w_i d(p - p_i) + \sum_{i=1}^{n} w_i d(p_i - p)$. Then we find the minimum through taking the derivative of cost with respect to p, thus $dc/dp = (\sum_{p_i<p} w_i)$ - $(\sum_{p_i>p} w_i)$. Looking at the derivative, we can conclude that the function is greater than 0.

e. Find the best solution for the 2-dimensional post-office location problem, in which the points are (x, y) coordinate pairs and the distance between points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is the ***Manhattan distance*** given by $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$

This is the same problem as the 1-dimensional post-office location problem except for solving it separately for each dimension. Thus we can use the Manhattan distance to create a cost function where $p = (p_x, p_y)$ and $\text{cost}(p) = (\sum_{i=1}^{n} w_i|x_i - p_x|) + (\sum_{i=1}^{n} w_i|y_i - p_y|)$. We can follow the same steps for the 1-dimensional post-office location problem by taking the minimum value through derivation.

3. **12-3** ***Average node depth in a randomly built binary search tree***

In this problem, we prove that the average depth of a node in a randomly built binary search tree with n nodes is $O(lgn)$. Although this result is weaker than that of Theorem 12.4, the technique we shall use reveals a surprising similarity between the building of a binary search tree and the execution of a RANDOMIZED-QUICKSORT from Section 7.3.

We define the **total path length** $P(T)$ of a binary tree T as the sum, over all nodes x in T, of the depth of node x, which we denote by $d(x, T)$.

a. Argue that the average depth of a node in T is

$$1/n \sum_{x \in T} d(x, T) = 1/nP(T)$$

Thus, we wish to show that the expected value of $P(T)$ is $O(nlgn)$.

4. (10 pts) Solve Exercise 12.2-8 on Page 294 of your textbook.

5. (20 pts) Solve Problem 13-3 on Page 333 of your textbook.

6. (15 pts) You are given a red-black tree T with 15 internal nodes (nodes that hold key values) that form a *full* binary tree of height 3 (i.e., a full binary tree of height 4 if you include the NIL leaves). Can you assign colors to the nodes so that a call to RB-INSERT(T, z) for *any* new key value $z.key$ will cause RB-INSERT-FIXUP(T, z) to change the color of the root to red before switching it back to black? The initial assignment of colors needs to obey the red-black properties. If such a color assignment exists, then provide a sequence of 15 numbers whose insertion (in that order) would lead to such a tree, along with a figure of the resulting tree. If not, then explain why such an assignment cannot exist, using the fact that the tree needs to satisfy the red-black properties.

7. (12 pts) Provide an algorithm that, given a graph $G = (V, E)$ as input, decides whether G is bipartite. Also, provide a tight bound regarding the worst-case running time of the algorithm (show how you got this bound), and explain why your algorithm is correct.