CS325 Winter 2013: HW 2

Daniel Reichert
Trevor Bramwell
Lance Stringham

January 25, 2013

1

Problem: Given two sorted arrays $a[1, ..., n]$ and $b[1, ..., n]$, given a $O(\log n)$ algorithm to find the median of their combined $2n$ elements. (Hint: use divide and conquer).

Solution:

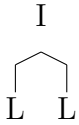
```
med_a = a[n / 2]
med_b = b[n / 2]
i = 4
while (i < n)
    if (med_a > med_b)
        med_a = a[((i - 1) * n) / 2]
        med_b = b[n / i]
    else (med_a < med_b)
        med_a = a[n / i]
        med_b = b[((i - 1) * n) / 2]
    i = 2 * i
    if (i >= n)
        median = (med_a + med_b) / 2
        return median
```

2

Problem: Prove the following statement by induction: In any full binary tree, the number of leaves is exactly one more than the number of internal nodes. Definition: A node is a leaf if it has no children; otherwise, it is an internal node. A full binary tree is a binary tree whose node is either a leaf node or an internal node with exactly two children.

Solution: Proof by induction.

Base case: a full binary tree of height 1.

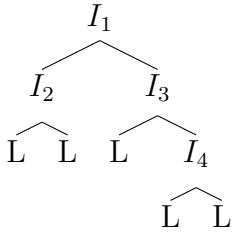


This tree contains two leafs and one internal node, which satisfies the statement $L = I + 1$, and proves our base case.

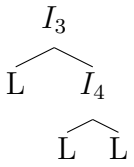
Inductive step: Assume: $L_k = I_k + 1$ for $1 \leq k \leq n$

Prove: $L_{(k+1)} = I_{(k+1)} + 1$

By the definition of a full binary tree we know that any internal node may be cut such that the remaining two branches are also full binary trees. This allows us to express any full binary tree as a set of recursively defined full binary trees.



In the above full binary tree, if the internal node I_3 was cut away from the tree to only be represented as in the tree below, the newly created tree is still a full binary tree.



Further, if the internal node I_4 is cut away from the tree, the newly created full binary tree conforms to the base case.

Thus, by induction, in any full binary tree, the number of leaves is exactly one more than the number of internal nodes. □

3

Problem: Interval scheduling. We are given a set of requests for using a resource. Each request i specifies a starting time $s(i)$ and an end time $f(i)$. The resource can only accommodate one request at a time. If two requests overlap in time, they are incompatible and cannot be both fulfilled. The goal is to identify a maximum subset of compatible requests. One possible greedy strategy is to select at each step the request that is compatible with the maximum number of the remaining requests. Will this greedy strategy lead to an optimal solution? If so, provide a proof. If not, provide a counter example.

Solution: Proof by counter example.
In the following counter example we will demonstrate that the greedy choice of maximal number of remaining requests is not optimal.

```
|--1--||--2--||--3--||--4--|
  |--5--| |--6--| |--7--|
  |--8--|         |--9--|
  |--10-|         |--11-|
```

In this example the optimal solution consists of choosing four intervals numbered 1-4. Intervals 1 and 4 have seven remaining requests. Intervals 2,3,5,7,8,9,10 an 11 have six remaining requests. Interval 6 has a total of eight remaining requests. Because interval 6 has the highest number of requests remaining, it would be selected first. This first choice conflicts with intervals 2 and 3. Thus only allowing a solution with a total of three intervals. Thus, proof by contradiction. □

4

Problem: You and your friends are taking a long hiking trip of L miles, along which there are n camping sites located at distances $x_1, x_2, \ldots, x_n$ respectively from the start of the trip. You can hike at most d miles per day, by the end of which you must stop and camp for the night. You need make a valid trip plan that takes the minimum number of camping stops. The plan should specify which camping sites to use, and it is only valid if any two consecutive stops are no more than d miles apart. Your friend proposed the following strategy: each time you come to a camp site, check whether you can make it to the next site before the end of the day (i.e., before finishing the d miles quota for the day. We assume this can always be determined correctly), if so, keep hiking. If not, stop for the night. This is in fact a greedy algorithm, which simply choose to hike as long as possible each day. Prove that this greedy algorithm achieves the optimal solution, i.e., it uses the minimum number of stops. (Hint: construct a proof that is similar to the interval scheduling proof, which shows that the greedy algorithm stays ahead.)

Solution:

Proof by Induction. Let O be the set of outcomes from the algorithm that gives the optimum solution $O = \{j_1, \ldots, j_n\}$, and A be the set of outcomes of an algorithm that gives the stops chosen by travelling the maximum distance each day $A = \{i_1, \ldots, i_k\}$. We will prove that A is the optimum solution by proving $k = n$.

Assume that for $f(i_r) \leq f(j_r)$ for any arbitrary positive integer r such that $1 \leq r < k$, where $f(\cdot)$ is the function that chooses when to stop.

Base case: Let $r = 1$. Since $f(\cdot)$ chooses to stop at the campsite that maximizes d travelled, then $i = j$ and $f(i_1) = f(j_1)$.

Inductive step: Assume $1 \leq l < r$ is true. For any $r > 1$ the inductive assumption implies that $f(i_{r-1}) \leq f(j_{r-1})$. Thus j_r is compatible with $i_1, \ldots, i_{r-1}$, but since the algorithm selected i_r over j_r, this implies that $f(i_r) \leq f(j_r)$.

□