Search Complexity

COMP 215 Lecture 16

Bounds for Searching

- We've seen two search algorithms.
  - Sequential Search: n comparisons in the worst case.
    - Optimal?
  - Binary Search $\lfloor \lg n \rfloor + 1$ comparisons in the worst case.
    - Optimal?
- It will take some work to determine whether it is possible to beat $\lfloor \lg n \rfloor + 1$.

Decision Trees For Searching

- Nodes represent comparisons.
- Each node has three children.
  - <, =, >
- Let's draw the complete decision tree for binary search of 7 items.
- We are really only interested in the comparison nodes.
- The worst case number of comparisons for a searching algorithm is the number of nodes in the longest path from the root to a leaf.
- Equal to $d + 1$, where d is the depth of the binary tree of comparison nodes.

Decision Trees for Searching

- When considering sorting, we started with the fact that the decision tree needed to have $n!$ leaves.
- Here we have a requirement on the total number of nodes: n.
  - If we are searching an array of n items, the tree of comparison nodes must have at least n nodes if it represents a valid search. (Lemma 8.2 from the book).
  - Proof in the book is a bit tricky, because, technically comparisons in the search can be made either with the search key, or between two items in the array.

Proof that Search Tree Must Have *n* Items

- First show that every array position *i* must be included in at least one comparison node.
  - Assume the contrary, then consider two inputs, one in which the search key matches, *i* and one in which it does not. Our tree will reach the same leaf in either case. Therefore the tree does not represent a valid search.

Proof that Search Tree Must Have *n* Items

- A search tree can include every index in a comparison and still have fewer than n nodes *if* some comparisons are between elements in the array.
  - For this to be the case, there must be some index *i* that is compared only with other entries.
  - Consider two inputs that differ only at position i. The search key matches the A[i] in one, but not the other. Further, Assume that A[i] contains the smaller item in the comparison for both inputs. The decision tree reaches the same leaf in either case. Therefore the tree does not represent a valid search.

Minimum Tree Depth

- What is the minimum depth of a tree with n nodes?
- $d \geq \lfloor \lg n \rfloor$
- Proof: $n \leq 1+2+2^2+\cdots+2^d = \sum_{i=0}^{d} 2^i$
  - Because there can be at most 2^i nodes at depth i.

$$\sum_{i=0}^{d} 2^i = 2^{d+1}-1, \quad \text{so} \quad n \leq 2^{d+1}-1$$

$$n < 2^{d+1}$$

$$\lg n < d+1$$

$$\lfloor \lg n \rfloor \leq d$$

Lower Bound for Searching

- Worst case number of comparisons for a given decision tree is the depth of the tree plus one.
- The decision tree must have at least n comparison nodes.
- Therefore any search algorithm must make at least $\lfloor \lg n \rfloor + 1$ comparisons in the worst case.
- Binary search performs exactly $\lfloor \lg n \rfloor + 1$ comparisons in the worst case.
- It is an optimal search algorithm.

Bound on Average Case

- Assuming our search key is in the array,
- Each comparison node in the decision tree is equally likely to lead to the terminal leaf in the search.
- The total node distance (TND) is the total number of comparison nodes traversed from the root to each possible result.
- The TND in the binary search decision tree is:

$$\lfloor \lg n \rfloor - 1 \leq A(n) \leq \lfloor \lg n \rfloor$$

- Taking into account that the item may not be in the array leads to:

$$\lfloor \lg n \rfloor - \frac{1}{2} \leq A(n) \leq \lfloor \lg n \rfloor + \frac{1}{2}$$

Bound on Average Case

- Preceding story applies to binary search.
- It can be shown than the decision tree associated with binary search has the minimum TND among all trees with n nodes.
- Therefore binary search has optimal average case performance.