CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya



Why study algorithms and performance?

- Algorithms help us to understand *scalability*.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a *language* for talking about program behavior.
- Performance is the *currency* of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Searching

- Given a list of numbers: L = {1,4,5,...}
 - L can be implemented as linked list or array.
- Given another number: x
- Return whether x is present in L
 - Variant return the any / first instance of x in L
- Linear Search:
 - For each element y in L: if x==y return y
 - Return null



Running time

- The running time depends on the input:
 - if x appears early in L, running time is low.
- Parameterize the running time by the size of the input, say length of L.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.



Kinds of analyses

Worst-case: (usually)

• T(n) = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- *T*(*n*) = expected time of algorithm over all inputs of size *n*.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

• Cheat with a slow algorithm that works fast on *some* input.



Machine-independent time

What is linear search's worst-case time?

- It depends on the speed of our computer:
 - relative speed (on the same machine),
 - absolute speed (on different machines).

BIG IDEA:

- Ignore machine-dependent constants.
- Look at *growth* of T(n) as $n \to \infty$.

"Asymptotic Analysis"



Θ -notation

Math: $\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and}$ $n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 5n + 6046 = \Theta(n^3)$
- Big-O notation: f(n) = O(T(n)) if $f(n) \le cT(n)$ for some c, and $n \ge n_0$.



Asymptotic performance

When *n* gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

Running time for Linear Search

- T(n) = O(n)
- Worst case running time is also $\Theta(n)$.
- $T(n) = O(n^2)$
- Can we do better ?
- Also, can maintain L as a dynamic datastructure ?

Binary Search Trees

- *Binary Search Trees* (BSTs) are an important data structure for dynamic sets
- Comprises of a number of linked nodes / items / elements.
- In addition to satellite data, nodes have:
 - *key*: an identifying field inducing a total ordering
 - *left*: pointer to a left child (may be NULL)
 - *right*: pointer to a right child (may be NULL)
 - p: pointer to a parent node (NULL for root)

Binary Search Trees

• BST property:

key[leftSubtree(x)] ≤ key[x] ≤ key[rightSubtree(x)]

• Example:



Inorder Tree Walk

- What does the following code do? TreeWalk(x) TreeWalk(left[x]); print(x); TreeWalk(right[x]);
- A: prints elements in sorted (increasing) order
- This is called an *inorder tree walk*

- *Preorder tree walk*: print root, then left, then right

Postorder tree walk: print left, then right, then root

Inorder Tree Walk



- How long will a tree walk take?
- Prove that inorder walk prints in monotonically increasing order

Operations on BSTs: Search

• Given a key and a pointer to a node, returns an element with that key or NULL:

```
TreeSearch(x, k)
if (x = NULL or k = key[x])
    return x;
if (k < key[x])
    return TreeSearch(left[x], k);
else
    return TreeSearch(right[x], k);</pre>
```

BST Search: Example

• Search for *D* and *C*:



Operations on BSTs: Search

• Here's another function that does the same:

```
TreeSearch(x, k)
while (x != NULL and k != key[x])
    if (k < key[x])
        x = left[x];
    else
        x = right[x];
return x;</pre>
```

• Which of these two functions is more efficient?

Operations of BSTs: Insert

- Adds an element x to the tree so that the binary search tree property continues to hold
- The basic algorithm
 - Like the search procedure above
 - Insert x in place of NULL
 - Use a "trailing pointer" to keep track of where you came from (like inserting into singly linked list)

BST Insert: Example

• Example: Insert C



BST Search/Insert: Running Time

- What is the running time of TreeSearch() or TreeInsert()?
- A: O(h), where h = height of tree
- What is the height of a binary search tree?
- A: worst case: h = O(n) when tree is just a linear string of left or right children

– We'll keep all analysis in terms of h for now

- Later we'll see how to maintain $h = O(\lg n)$

More BST Operations

- BSTs are good for more than searching. For example, can implement a priority queue
- What operations must a priority queue have?
 - Insert
 - Minimum
 - Extract-Min

BST Operations: Minimum

- How can we implement a Minimum() query?
- What is the running time?

BST Operations: Successor

- Two cases:
 - x has a right subtree: successor is minimum node in right subtree
 - x has no right subtree: successor is first ancestor of x whose left child is also ancestor of x
 - Intuition: As long as you move to the left up the tree, you're visiting smaller nodes.
- Predecessor: similar algorithm

BST Operations: Successor

TREE-SUCCESSOR(x)

1 **if** $x.right \neq NIL$ 2 **return** TREE-MINIMUM(x.right) 3 y = x.p4 **while** $y \neq NIL$ and x == y.right5 x = y6 y = y.p7 **return** y

BST Operations: Delete

- Deletion is a bit tricky
 3 cases:

 x has no children:
 Remove x
 A

 D H

 Example: delete K or H or B
 - x has two children:
 - Swap x with successor
 - Perform case 1 or 2 to delete it

BST Operations: Delete

- Why will case 2 always go to case 0 or case 1?
- A: because when x has 2 children, its successor is the minimum in its right subtree
- Could we swap x with predecessor instead of successor?
- A: yes. Would it be a good idea?
- A: might be good to alternate