CIS 313

week of Jan 28

fourth week of the term
binary search trees

• chapter 12
• we will look at
  • definitions
  • properties
  • operations: insert, delete, search
  • traversals: inorder, postorder, preorder, level order
  • worst case behavior
  • average case behavior
• then move onto self-balancing BSTs: red-black, 2-3, 2-3-4, ...
various trees

• free tree
• rooted tree
• ordered tree
• binary tree
• binary search tree
  • (search property) let x be a node in a BST. If y is a node in the left subtree of x, then $y.key \leq x.key$. If y is in the right subtree of x, then $y.key \geq x.key$
assorted facts and definitions

• any tree with n nodes has n-1 edges
• a binary tree with left/right pointers and n nodes has n+1 null pointers
• a full binary tree with n internal nodes has n+1 external nodes
• full binary tree: all nodes have either 2 children (the internal nodes) or 0 children (external)
• a binary tree of n nodes has height at least $\log n$ and at most n-1
• height = distance of node from bottom, depth = distance from top
facts, defs cont’d

• internal path length (I): sum of the depths of all the nodes
• external path length (E): sum of the depths of the nulls (externals)
• fact: $E = I + 2n$ (nice exercise)
• I corresponds to successful search in BST, average search time is $1 + \frac{I}{n}$
• E corresponds to unsuccessful search, average failed search time is $\frac{E}{n+1}$
• worst case tree: skew tree (every node has just one child)
sample BST
BST operations

• find(x)
• insert(x): find a null and put it there
• successor(x)
  • successor(10)=11, successor(15)=17
  • algorithm?
    • if x has right child, go right once, then left until end
    • otherwise, follow parent links until “right” turn
• delete(x): how?
  • if 0 children, remove
  • if 1 child, splice out
  • if 2 children, replace with successor value, then remove successor node
walks

• inorder
  • 1 3 4 5 7 8 9 10 11 12 13 15 17 18 20 23

• preorder
  • 12 10 5 3 1 4 8 7 9 11 17 13 15 20 18 23

• postorder
  • 1 4 3 7 9 8 5 11 10 15 13 18 23 20 17 12
randomly built BST

• we have n values and will insert them one-by-one into a BST
• what will that BST look like?
• there are n! permutations of the input
  • we assume each one equally likely
• how many BST shapes can there be?
  • Catalan number, which is \( \frac{1}{n+1} \binom{2n}{n} = \Omega\left(\frac{4^n}{n^2}\right) \)
  • (hard!)
Counting permutations for a tree

• given a tree shape T we can determine the number of permutations which, if inserted into empty BST, would end up with that tree
• build up number bottom up
• at node x, suppose left subtree of x has n nodes and is generated by r permutations, and
• right subtree has m nodes and is generated by s permutations
• the the subtree rooted at x
  • has n+m+1 nodes
  • is generated by \( \binom{n+m}{n} \cdot r \cdot s \) permutations
example

- left side generated by 1 permutation: 13 15
- right side by two
  - 20 18 23
  - 20 23 18
- for full tree, pick one permutation each for the left and right sides
- permutation for the whole tree must start with 17 followed by \( n+m = 2+3 = 5 \) spaces
  - 17 __ __ __ __ __
- choose two for them for the left tree, which can be done in \( \binom{5}{2} = 10 \) ways
- example: 2\textsuperscript{nd} and 5\textsuperscript{th} positions
  - 17 __ 13 __ __ 15
- either of the two remaining perms can go in remaining three slots
  - 17 20 13 18 23 15
  - 17 20 13 23 18 15
- total number of permutations for whole tree:
  \[ 1 \cdot 2 \cdot \binom{5}{2} = 20 \]

intuition: balanced trees more “likely”
back to sorting theme

• we can build an abstract sort method based on BST
• given unsorted list, insert all values into empty BST
• perform inorder walk

```
BST SORT
** input list a=(a_1,a_2,...,a_n)
create BST T

for i=1 to n
    T.insert(a_i)

perform T.inorder
    when visiting a node, store value in list b

return b
```

this part is O(n)
expected behavior

- if list a is chosen randomly from among all n! permutations
- how long does “for i=1 to n T.insert(a_i)” take?
- worst case: O(n^2)
- want to argue: on average O(n \lg n)

- main fact: expected search time (1+1/n) in BST built from randomly chosen permutation is
  \[2 \cdot \ln(n + 1) + O(1) \approx 1.38 \log_2 n + O(1)\]
describe a binary search tree on n nodes such that the average depth of a node in the tree is $\Theta(\lg n)$ but the height of the tree is $\omega(\lg n)$