CSE 326: Data Structures

Topic #5: Binary Search Trees

Ashish Sabharwal
Autumn, 2003

Today's Outline

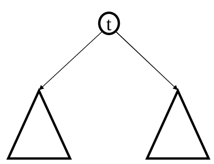
- Admin: Written homework #1 is out!

- Quick Tree Review
- Binary Trees
- **Dictionary ADT / Search ADT**
- **Binary Search Trees**

Tree Calculations

Recall: height is max number of edges from root to a leaf

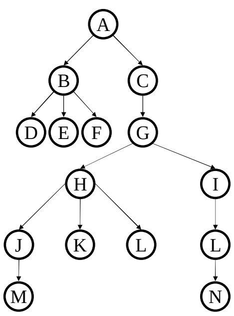
Find the height of the tree...



runtime:

Tree Calculations Example

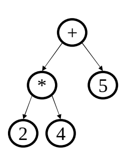
How high is this tree?



More Recursive Tree Calculations: Tree Traversals

A *traversal* is an order for visiting all the nodes of a tree

Three types:
- Pre-order: Root, left subtree, right subtree
- In-order: Left subtree, root, right subtree
- Post-order: Left subtree, right subtree, root

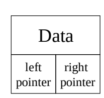


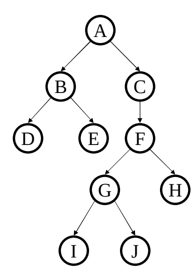
(an expression tree)

Binary Trees

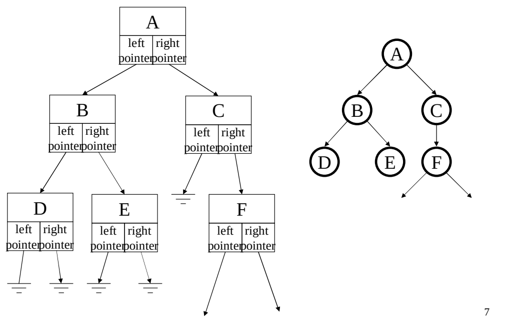
- Binary tree is
  - a root
  - left subtree *(maybe empty)*
  - right subtree *(maybe empty)*

- Representation:



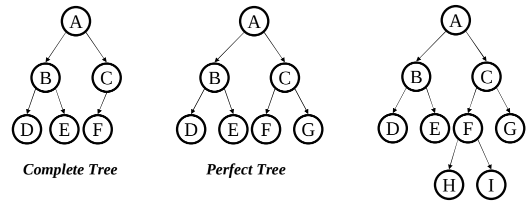


Binary Tree: Representation



7

Binary Tree: Special Cases



Complete Tree

Perfect Tree

Full Tree

8

Binary Tree: Some Numbers!

For binary tree of height *h*:
- max # of leaves:
- max # of nodes:
- min # of leaves:
- min # of nodes:

What's the average tree height for *n* nodes, assuming all distinct trees of *n* nodes are equally likely?

9

ADTs Seen So Far

- Stack
  - Push
  - Pop
- Queue
  - Enqueue
  - Dequeue
- List
  - Insert
  - Remove
  - Find
- Priority Queue
  - Insert
  - DeleteMin

Remember decreaseKey?

10

New! The Search ADT

- Data:
  - unique user-specified *keys*
  - Or: a set of keys
- Operations:
  - Insert (key)
  - Find (key)
    - Checks for membership
  - Remove (key)

insert(Grill) →

- Fry
- Simmer
- Puree
- Braise
- Poach
- Sear
- Stirfry
- Roast

← find(Bake)
NOT FOUND

*The Search ADT is sometimes called the "**Set ADT**"*

11

Also New! The Dictionary ADT

- Data:
  - *values* mapped to user-specified *keys*
  - Or: a set of (key, value) pairs
- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)

insert(ashish…) →

- ashish@cs
  Ashish Sabharwal,
  instructor for CSE326,
  hoping to graduate soon!
- ethanpg@cs
  Ethan Phelps-Goodman,
  TA for CSE326,
  new grad student
- awong@cs
  Albert Jongkit Wong,
  TA for CSE326,
  5th year undergrad

← find(ethanpg)
• ethan
Ethan…, new grad…

An easy extension of the Search ADT!

*The Dictionary ADT is sometimes called the "**Map ADT**"*

12

A Modest Few Uses

- Sets
- Dictionaries
- Networks : Router tables
- Operating systems : Page tables
- Compilers : Symbol tables

Probably the most widely used ADT!

13

Naïve Implementations

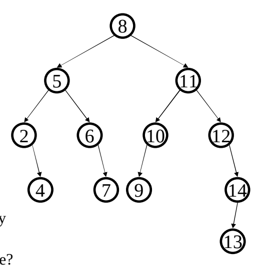
	insert	find	delete
• Unsorted Linked-list			
• Unsorted array			
• Sorted array			

What limits the performance?

14

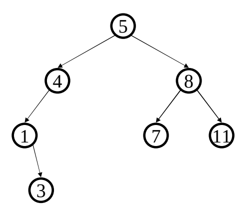
Binary Search Tree Data Structure

- Structural property
  - each node has ≤ 2 children
  - result:
    - storage is small
    - operations are simple
    - average depth is small
- Order property
  - all keys in left subtree smaller than root's key
  - all keys in right subtree larger than root's key
  - result: easy to find any given key
- What must I know about what I store?

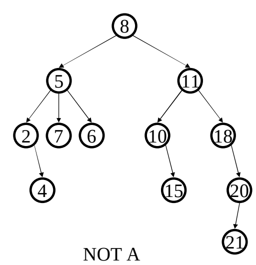


15

Example and Counter-Example



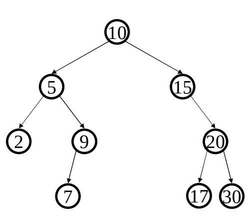
BINARY SEARCH TREE



NOT A
BINARY SEARCH TREE

16

Find in BST, Recursive




```
Node Find(Object key,
            Node root) {
  if (root == NULL)
    return NULL;

  if (key < root.key)
    return Find(key,
                root.left);
  else if (key > root.key)
    return Find(key,
                root.right);
  else
    return root;
}
```

Runtime:

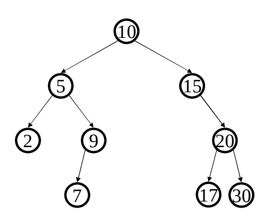
17

Find in BST, Iterative

```
Node Find(Object key,
            Node root) {

  while (root != NULL &&
         root.key != key) {
    if (key < root.key)
      root = root.left;
    else
      root = root.right;
  }

  return root;
}
```



Runtime:

18

Binary Search vs. Binary Search Tree

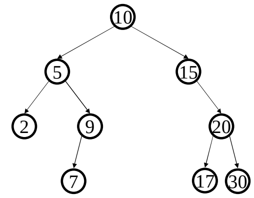
2	5	7	9	10	15	17	20	30

find(9)
find(2)
find(20)
find(15)

A well balanced binary search tree allows O(log *n*) time binary search!

19

Insert in BST



Insert(13)
Insert(8)
Insert(31)

Insertions happen only at the leaves – easy!

Runtime:

20

BuildTree for BST

- Suppose keys 1, 2, 3, 4, 5, 6, 7, 8, 9 are inserted into an initially empty BST.

Runtime depends on the order!

  – in given order

  – in reverse order

  – median first, then left median, right median, etc.

21

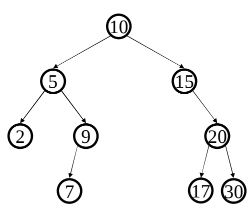
Analysis of BuildTree

- Worst case: $O(n^2)$ as we've seen
- Average case assuming all orderings equally likely:
  – Sum of all depths:
    - $D(n) = D(i) + D(n - i - 1) + (n - 1)$
      $=$
  – Average depth of a node:
  – Total runtime:

22

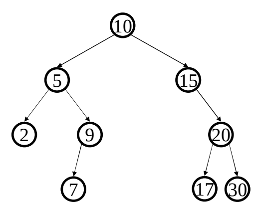
Bonus: FindMin/FindMax

- Find minimum
- Find maximum



23

Deletion in BST



Why might deletion be harder than insertion?

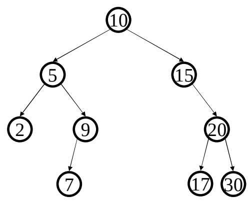
24

Lazy Deletion

Instead of physically deleting nodes, just mark them as deleted

+ simpler
+ physical deletions done in batches
+ some adds just flip deleted flag

– extra memory for deleted flag
– many lazy deletions slow finds
– some operations may have to be modified (e.g., min and max)



25

Lazy Deletion

Delete(17)

Delete(15)

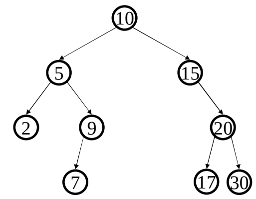
Delete(5)

Find(9)

Find(16)

Insert(5)

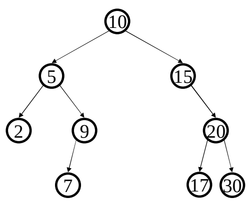
Find(17)



26

Non-lazy Deletion – The Leaf Case

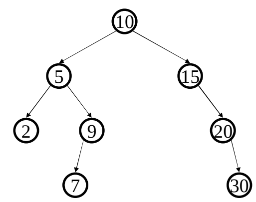
Delete(17)



27

Deletion – The One Child Case

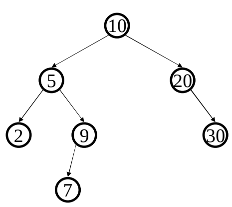
Delete(15)



28

Deletion – The Two Child Case

Delete(5)



What can we replace 5 with?

What happens to that other node?

29

Deletion – The Two Child Case

Idea: Replace the deleted node with a value guaranteed to be between the two child subtrees!

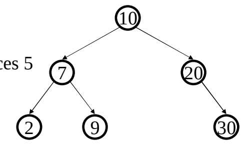
Options:
- *succ* from right subtree: findMin(t.right)
- *pred* from left subtree : findMax(t.left)

Now delete the original node containing *succ* or *pred*
- Leaf or one child case – easy!

30

Finally…

7 replaces 5



Original node containing
7 gets deleted

31

To Do

- Start Homework #1
  - Somewhat long but easy
  - Will get you hands on practice with Math background and heaps
- Read chapter 4 in the book

32