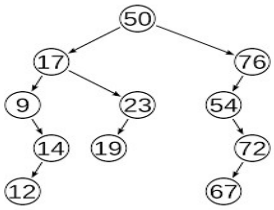
Binary Search Trees



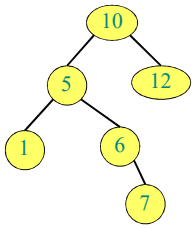
6.006 Lecture 5

Image from Wikipedia.org

Overview

http://izismile.com/tags/Gibraltar/

- A problem: Runway reservations
- A new datastructure:
  Binary Search Trees

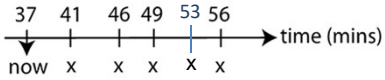


Runway reservation system

- Maintain reservations for set of landings on single runway
  - Legality: in future + at least 3 unscheduled minutes before/after each landing
- Operations:
  - *Add* new request to land at time *t* (if legal)
  - *Find next* landing and remove from set

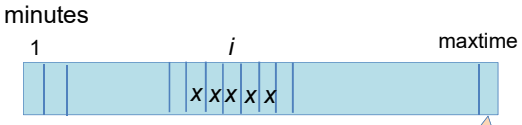
Runway reservation system

- Example



  - *R* = (41, 46, 49, 56)
  - requests for time:
    - 44 => reject (46 in *R*)
    - 53 => ok, *R* is now (41, 46, 49, 53, 56)
    - 20 => not allowed (already past)
- Ideas for efficient implementation ?

A possibility?



Insert x,*i*

Add *O(1)*

Find next *O(maxtime)*

Numbers arbitrarily large

Data structures that we know:

- Represent R as *(sorted)* linked-list
  - Add *O(n)*
  - Find next *O(1)*
- Represent R as unsorted array
  - Add *O(n)* *(test legality? Insert O(1))*
  - Find next *O(n)*
- Represent R as sorted array
  - Add *O(n)* *(insert? Test legality O(log n))*
  - Find next *O(1)*
- Represent R as min heap
  - Add *O(n)* *(test legality?)*
  - Find next *O(1)*

Is this the best we can do?

Let's apply our 6.006 data structure superpowers again!

Binary Search Trees (BSTs)

A tree …

…where each node *x* has:

a *key[x]*

three pointers:

left[x] : points to left child

right[x] : points to right child

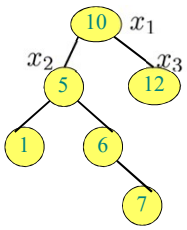
p[x] : points to parent

E.g. $key[x_1]=10$

$left[x_1]=x_2$

$p[x_2]=x_1$

$p[x_1]=NIL$

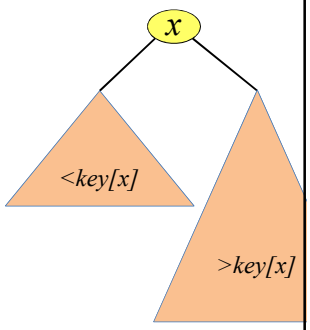


Binary Search Trees (BSTs)

BST property :

for any node *x*:

- for all nodes *y* in left subtree of x:
  key[y] ≤ key[x]
- for all nodes *y* in right subtree of *x*:
  key[y] ≥ key[x]

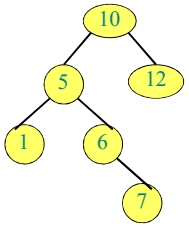


Binary Search Trees (BSTs)

BST property :

for any node *x*:

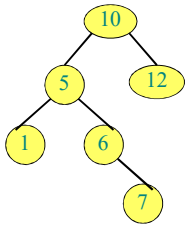
- for all nodes *y* in the left subtree of x:
  key[y] ≤ key[x]
- for all nodes *y* in the right subtree of *x*:
  key[y] ≥ key[x]



Sorting given BST

Inorder-tree-walk(x):

- If $x \neq$ Null
  - Inorder-tree-walk(left[x])
  - Output key[x]
  - Inorder-tree-walk(right[x])



O(n) time!

Question:

- Given a set of keys, is there a unique BST?
- No!

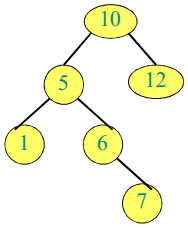
Some BST operations:

- *insert(k)*: insert a node with key k at the appropriate location of the tree
- *find(k)*: finds the node containing key *k* (if it exists)
- *delete(k)*: delete the node containing key *k*, if such a node exists
- *findmin(x) (findmax(x))*: finds the minimum (maximum) of the tree rooted at *x*
- *deletemin()*: finds the minimum of the tree and deletes it
- *successor(x)*: finds the node containing the key that is the immediate next of *key[x]*

BST-Sort

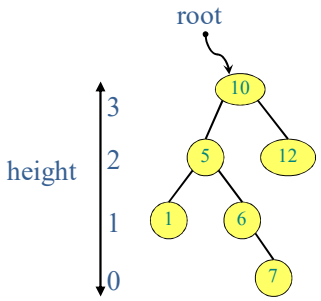
- Create BST from input
- Inorder-tree-walk(root)

10,12,5,6,1,7



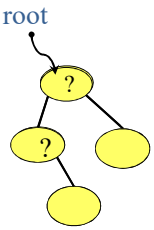
Growing BSTs

- Insert 10
- Insert 12
- Insert 5
- Insert 1
- Insert 6
- Insert 7



BST Insert

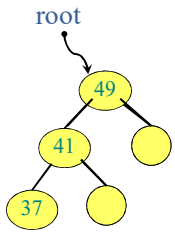
insert(k): insert a node with key *k* at the appropriate location of the tree



e.g. insert(37)

BST insert

insert(k): insert a node with key *k* at the appropriate location of the tree



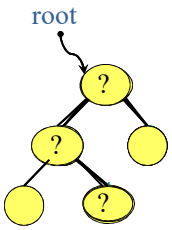
e.g. insert(37)

Running time?

Aside: Can do the "within 3" check for reservation system during insertion.

BST find

find(*k*): finds the node containing key k (if it exists)

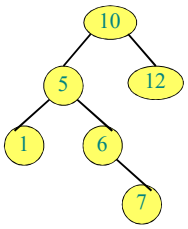


e.g. find(46)

Running time?

BST findmin

findmin: finds the node containing min valued key k

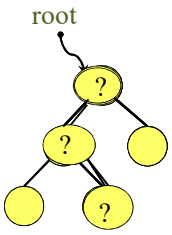


"Go left young man"
-- Horace Greeley

Running time?

BST delete

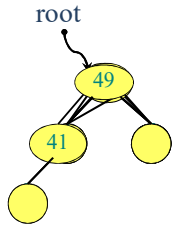
delete(k): delete the node containing key *k*, if such a node exists



e.g. delete(46)

BST delete

delete(k): delete the node containing key *k*, if such a node exists



e.g. delete(46)

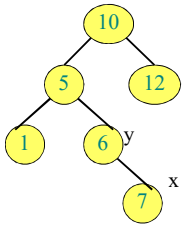
Running time?

Question: What if we have to delete a node that is internal? How do we fill in the hole? A: next lecture.

Successor

Go up tree until find a "left child"

successor(x):
- If right[x] ≠ NIL then return findmin(right[x])
- Otherwise
  y ← p[x]
  While y≠NIL and x=right[y] do
    - x ← y
    - y ← p[y]
  Return y

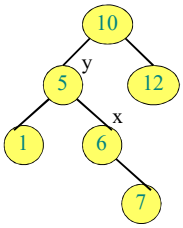


successor(5) = 6
successor(7)

Successor

Go up tree until find a "left child"

successor(x):
- If right[x] ≠ NIL then return findmin(right[x])
- Otherwise
  y ← p[x]
  While y≠NIL and x=right[y] do
    - x ← y
    - y ← p[y]
  Return y

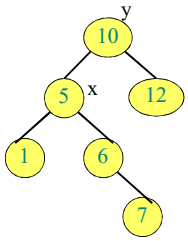


successor(5) = 6
successor(7)

Successor

Go up tree until find a "left child"

successor(x):
- If right[x] ≠ NIL then return findmin(right[x])
- Otherwise
  y ← p[x]
  While y≠NIL and x=right[y] do
    - x ← y
    - y ← p[y]
  Return y

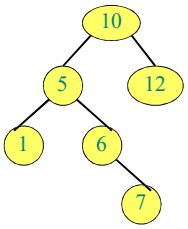


successor(5) = 6
successor(7) = 10

Analysis

- How much time do operations take ?
  - Worst case: *O(height)*

  => height really important
- After we insert *n* elements, what is the worst possible BST height?



Analysis

- Height can be *n-1*
  - Still *O(n)* for the runway reservation system operations?
- Next lecture:

Balanced BSTs