Binary Search Trees

Today continue discussion of Data Structures

Operations: Insert, Delete, Search

Introduce: Linked List (§ 10.2 of text CLRS)

example

head → [13|] → [3|] → [11|] → [7|/]

Insert onto head; assumes have pointer & don't search

Data Structures	Insert	Delete	Search	Model Elements
Linked List	$O(1)$	$O(1)$	$O(n)$	exact match
Sorted Array	$O(n)$	$O(n)$	$O(\lg n)$	comparison $(<, >, =)$
Hash Table	$O(1)_{\text{expect}}$	$O(1)_{\text{expect}}$	$O(1)_{\text{expect}}$	hash function

move data

binary search: Divide & conquer with middle element

Since we have achieved constant time with hashing, why would we need any other type of data structure?

It turns out that there are hundreds of data structures and this is a very rich research field. Reason is that different types of structures better suited to different tasks. The 3 operations (insert, delete, search) are not the only ones we may be interested in.

→ Simple: minimum (easy for sorted, hard for linked list or hash table)

→ Useful: nearest match

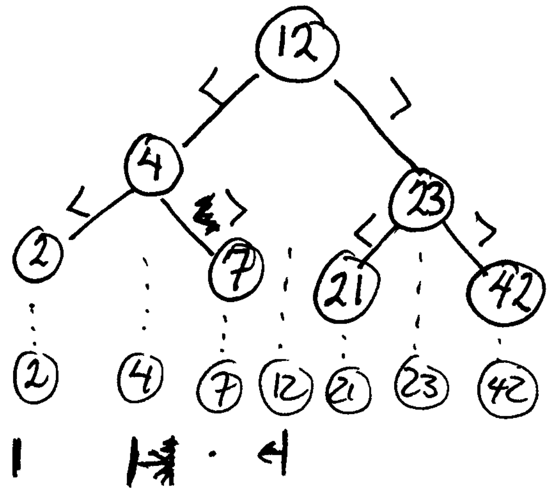
Examples: (1) Need fast search (hash) and fast min (sorted array) $O(1)$ in best

- imagining running both structures in parallel
- once built, do search in hash ($O(1)$) - min in sorted array ($O(1)$)
- problem: to maintain each data structure's integrity, need to insert/delete into each $O(n)$ for sorted array
- does it help to maintain pointers between corresponding elements of each data structure?

② Inexact Searches

- "Interduction to Algirithims"
  - Introd. to Algor., 2nd ed
  - Introd. to Algor.
  - ~~Introductio Algorithms~~
  - Intros. to Algalods
  - Introduction to Logarithms
  - ⋮
- nearest (lexical) match
- hashing doesn't do this; in fact, hash function often chosen so similar keys do not end up in nearby locations (uniformity)
- sorted array with binary search will find where element would be in array $O(\lg n)$, but require $O(n)$ to build/maintain.

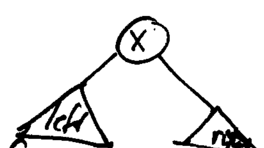
TODAY WE INTRODUCE <u>BINARY SEARCH TREES (BST)</u>



Binary Search

looks like sorted array built into a tree.

Let's imagine searching for the key 7 in this tree and in the sorted array.



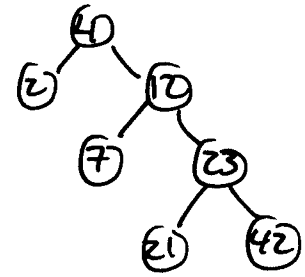
BST Property

① all nodes in left subtree of x have key $\leq$ key[x]
② all nodes in right subtree of x have key $\geq$ key[x]

BST traits

- binary tree (2 branches per node, can be nil ~~at most 2 branches per node~~), left/right, rooted
- each node contains:
  - ⓐ key[x]
  - ⓑ left[x] } can be NIL
  - ⓒ right[x]
  - ⓓ parent[x] = p[x] ← NIL only for root
  - ⓔ satellite_data[x]

To ~~[illegible]~~ See How Tree Works,
Let's ~~[illegible]~~ Look at How would
print Sorted Array from BST




```
Inorder [x]
    Inorder ( left[x] )  unless  left[x] = nil
    Print  key[x]
    Inorder ( right[x] )  unless  right[x] = nil
```

Look at our example

Recursively finds leftmost element and prints. then prints parent and right, etc.

2 4 7 12 21 23 42

That is, Inorder tree walk visits each node in sorted key order. It really just visits each node once, and through recursive calls keeps "placeholders" throughout the tree.

Time: $O(n)$ where n = # nodes in tree

Now, Imagine using the binary search tree as a data structure for information retrieval

This is iterative version

```
Search(T, k)
    x ← root[T]
    while x ≠ nil and key[x] ≠ k
    |  do if k < key[x]
    |       then  x ← left[x]
    |       else  x ← right[x]
    |← return x
```

Start at root node. If find key, done. Otherwise, use BST property to decide which subtree to follow. If reach nil, then key does not exist

Like binary search, but are not guaranteed of subdividing in half each subsearch. If tree is reasonably well balanced (proportional) will still be efficient.

Do examples ←
- successful
- unsuccessful

Analysis of Search

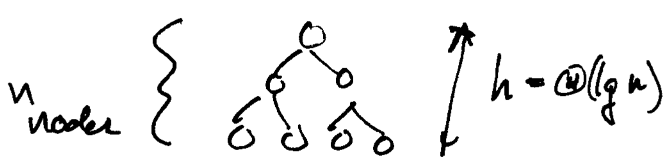
① Just form of comparison model (like binary search)

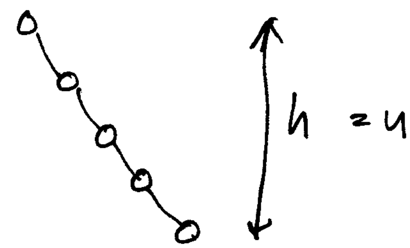
② Correctness (idea)

if $k < key[x]$, then by BST property, $k <$ all keys in right subtree (and can safely ignore)

③ Running Time

- To successfully find node $x \rightarrow \Theta(\text{depth}(x))$
  where depth = distance from root down to x
- Worst case $\rightarrow \Theta(\text{height}(T)) \equiv \Theta(h)$
- Worst case for any tree of n nodes $= \Theta(n)$





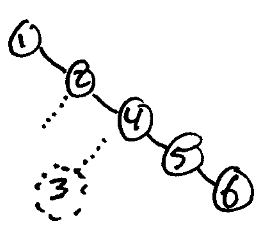
What about nearest matches?

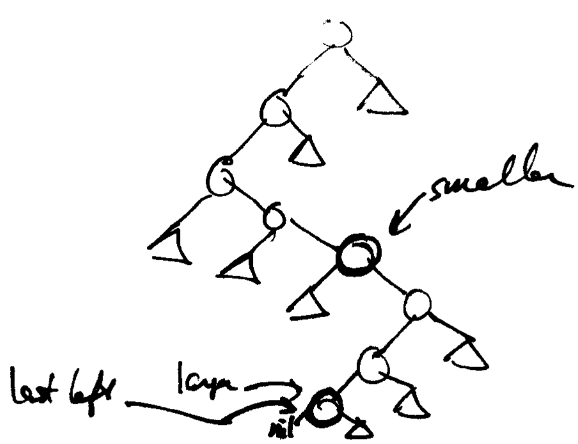
- Unsuccessful searches identify location where key would fit.

i.e. if last move was left

- next larger element is last node visited
- next smaller element is last time moved right

(& symmetric statements if last move was right)





Dynamic BST

Insert (T, k)

- run search until reach nil (pretend $left[x] = nil$)
- $left[x] \leftarrow$ new node with $key = k$
  - $left = nil$
  - $right = nil$
  - $parent = x$

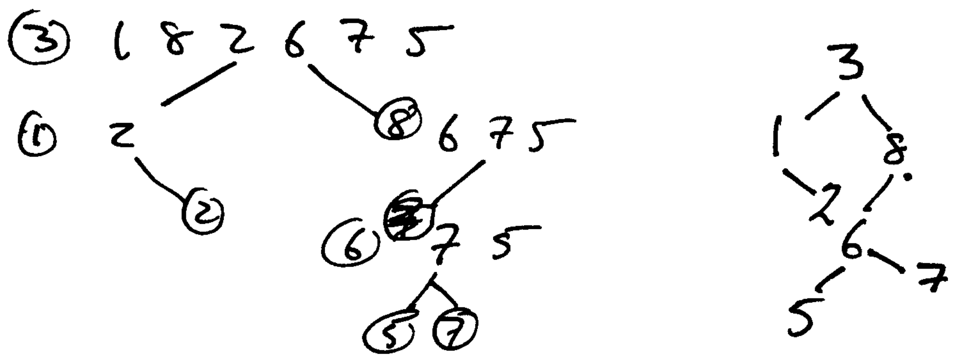
(Example)

Running time $\Theta(h)$

BST Sort (A)

for $i \leftarrow 1$ to n
  do Insert $(A[i])$
Inorder (root)

Analysis: Same comparisons as quicksort, but in different order



Claim $\Theta(n \lg n)$ time in average case (uniformly random permutation)

$\rightarrow$ avg depth of node $= O(\lg n)$

$=$ avg time for insert/search

In fact, max depth of node $= O(\lg n)$ in avg case

$\rightarrow$ recitation

More BST operations

Minimum(x) — minimum key in subtree rooted at x

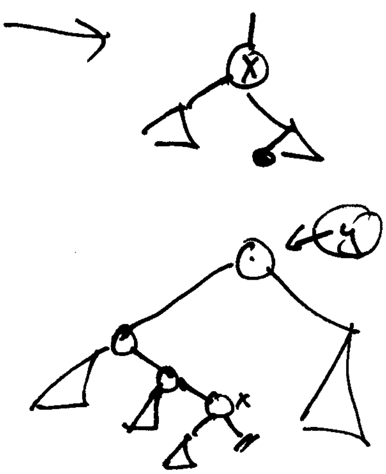
```
while left[x] ≠ nil
    do x ← left[x]
return x
```

Time = $O(h)$

Maximum is simple change

Successor(x) — next higher element in inorder seq

```
if right[x] ≠ nil
    then return minimum(right[x])
    else y ← parent[x]
        while y ≠ nil and x = right[y]
            do x ← y
               y ← parent[y]
        return y
```



Time = $O(h)$

predecessor is simple change

Delete(x) — given pointer to a node

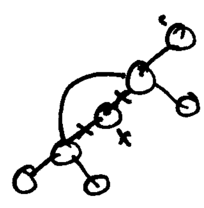
① If x has no children (leaf of tree) then remove x

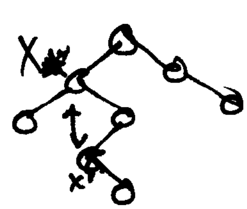
② If x has exactly one child, then splice it out

③ If x has 2 children, then swap x with successor(x)
 — do ① or ② with
 (guaranteed to have 1 or 0 children)

Time = $O(h)$

①


②


③


Problem: Worst Case BST height is $\Theta(n)$

$\Rightarrow$ degenerates to linked list

Solution: Force BST (or related) to stay "balanced"

$\Rightarrow$ have $\Theta(\lg n)$ height

Will get all ops in $\Theta(\lg n)$ time

BST ops

Search
Insert
Delete
Minimum
Maximum
Successor
Predecessor

$\left.\right\} \Theta(h)$

Started discussion with $\Theta(n)$ data problems. Now, with balanced tree, are all $\Theta(\lg n)$