6.006 - Introduction to Algorithms

Lecture 3

Prof. Piotr Indyk
Overview

- Runway reservation system:
  - Definition
  - How to solve with lists

- Binary Search Trees
  - Operations

Readings: CLRS 10, 12.1-3
Runway reservation system

• Problem definition:
  – Single (busy) runway
  – Reservations for landings
    • maintain a set of future landing times
    • a new request to land at time $t$
    • add $t$ to the set if no other landings are scheduled within $< 3$ minutes from $t$
    • when a plane lands, removed from the set
Runway reservation system

• Example

- $R = (41, 46, 49.1, 56)$
- requests for time:
  - $44 \Rightarrow$ reject (46 in R)
  - $53 \Rightarrow$ ok
  - $20 \Rightarrow$ not allowed (already past)

• Ideas for efficient implementation?
Some options:

• Keep $R$ as an unsorted list
  – Bad: takes linear time to search for collisions
  – Good: can insert $t$ in $O(1)$ time

• Keep $R$ as a sorted array
  (resort after each insertion)
  – Bad: takes “a lot of” time to insert elements
  – Good: 3 minute check can be done in $O(\log n)$ time:
    – Using binary search, find* the smallest $i$ such that $R[i] \geq t$ (next larger element)
    – Compare $t$ to $R[i]$ and $R[i-1]$

Need: fast insertion into sorted list
(sort of)
Binary Search Trees

- Simple and natural data structures
- Building blocks for

(a,b) tree, 2-3 tree, 2-3-4 tree, AA tree, AVL tree, B tree, B+ tree, B* tree, Cartesian tree, Dancing tree, H tree, Leftist tree, Red-black tree, Scapegoat tree, Splay tree, T tree, Tango tree, Top tree, UB tree,
Binary Search Trees (BSTs)

• Each node $x$ has:
  – $\text{key}[x]$
  – Pointers:
    • $\text{left}[x]$
    • $\text{right}[x]$
    • $\text{p}[x]$
Binary Search Trees (BSTs)

• Property: for any node $x$:
  – For all nodes $y$ in the left subtree of $x$:
    \[ \text{key}[y] \leq \text{key}[x] \]
  – For all nodes $y$ in the right subtree of $x$:
    \[ \text{key}[y] \geq \text{key}[x] \]

• How are BSTs made?
Growing BSTs

- Insert 10
- Insert 12
- Insert 5
- Insert 1
- Insert 6
- Insert 7
BST as a data structure

- Operations:
  - `insert(k)`: inserts key `k`
  - `search(k)`: finds the node containing key `k` (if it exists)
  - `next-larger(x)`: finds the next element after element `x`
  - `findmin(x)`: finds the minimum of the tree rooted at `x`
  - `delete(x)`: deletes node `x`
Search

Search(k):

• Recurse left or right until you find k, or get NIL
Next-larger

next-larger(x):
- If right[x] ≠ NIL then
  return minimum(right[x])
- Otherwise
  y ← p[x]
  While y ≠ NIL and x = right[y] do
    • x ← y
    • y ← p[y]
  Return y

next-larger(5)
next-larger(7)
Minimum

Minimum( x )
- While left[x]≠NIL do
  x ← left[x]
- Return x
Analysis

- We have seen insertion, search, minimum, etc.
- How much time does any of this take?
- Worst case: $O(\text{height})$
  $\Rightarrow$ height really important
- After we insert $n$ elements, what is the worst possible BST height?
Analysis

• n-1

• So, still $O(n)$ for the runway reservation system operations

• Next lecture: balanced BSTs

• Readings: CLRS 13.1-2