6.006 - Introduction to Algorithms

Lecture 3

Prof. Patrick Jaillet
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

  - $R = (41, 46, 49, 56)$
  - requests for time:
    - $44 \Rightarrow$ reject (46 in $R$)
    - $53 \Rightarrow$ ok
    - $20 \Rightarrow$ not allowed (already past)

- Ideas for efficient implementation?
Proposed algorithm

• (keep R as a sorted list)

init: R = [ ]
req(t): if t < now: return "error"
for i in range (len(R)):
  if abs(t-R[i]) < 3: return "error"
  R.append(t)
R = sorted(R)
land: t = R[0]
if (t != now) return error
R = R[1: ] (drop R[0] from R)

• Complexity?
• Can we do better?
Some options:

• Keep R as a sorted list:
  – takes linear time to insert element in proper place
  – a 3 minute check can then be done in O(1)

• Keep R as a sorted array:
  – takes O(log n) to find a place to insert new time ...
  – but still requires linear time to actually insert
    (requires shifting of elements)

• Keep R in unsorted order
  – takes linear time to search for collisions

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

• Each node $x$ has:
  – $\text{key}[x]$
  – Pointers:
    • $\text{left}[x]$
    • $\text{right}[x]$
    • $p[x]$
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 Diagram]

Root

Height: 3

Height: 2

Height: 1

Height: 0
**BST as a data structure**

- **Operations:**
  - `insert(k)` (note: can do the “within 3” check for reservation during insertion)
  - `find(k)`: finds the node containing key `k` (if it exists)
  - `findmin(x)`: finds the minimum of the tree rooted at `x`
  - `deletemin()`: finds the minimum of the tree and delete it
  - `next-larger(x)`: finds the next element after element `x`
next-larger(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
Back to runway reservation system

- New requirement: How many planes are scheduled to land at times $\leq t$?

- Augment the BST structure by keeping track of size of subtrees:

  - Walk down tree to find desired time
    - Add in nodes that are smaller
    - Add in subtree sizes to the left
Analysis

- We have seen insertion, deletion, search, findmin, 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