Balanced BST

- **Balanced BSTs** guarantee $O(\log N)$ performance at all times
  - the height or left and right sub-trees are about the same
  - simple BST are $O(N)$ in the worst case

- **Categories of BSTs**
  - AVL, SPLAY trees: dynamic environment
  - optimal trees: static environment
AVL Trees

- **AVL** (Adelson, Lelskii, Landis): the height of the left and right subtrees differ by at most 1
  - the same for every subtree
- **Number of comparisons** for membership operations
  - best case: completely balanced
  - worst case: $1.44 \log(N+2)$
  - expected case: $\log N + .25 \leq \left\lceil \log(N + 1) \right\rceil$
AVL Trees

- “—” : completely balanced sub-tree
- “/” : the left sub-tree is 1 higher
- “\” : the right sub-tree is 1 higher
AVL Trees
Non AVL Trees

"/\" : the left sub-tree is 2 higher
"\\" : the right sub-tree is 2 higher
Single Right Rotations

- Insertions or deletions may result in non AVL trees => apply rotations to balance the tree
Single Left Rotations
insert 1

single right rotation

insert 9

single left rotation
Double Left Rotation

- Composition of two single rotations (one right and one left rotation)
Example of Double Left Rotation

Critical node

insert 6
Double Right Rotation

- Composition of two single rotations (one left and one right rotation)
Insertion (deletion) in AVL

1. Follow the search path to verify that the key is not already there
2. Insert (delete) the key
3. Retreat along the search path and check the balance factor
4. Rebalance if necessary (see next)
Rebalancing

- For every node reached coming up from its left sub-tree after insertion readjust balance factor
  - ‘=’ becomes ‘/’ => no operation
  - ‘\’ becomes ‘=’ => no operation
  - ‘/’ becomes ‘//’ => must be rebalanced!!
- The “//” node becomes a “critical node”
- Only the path from the critical node to the leaf has to be rebalanced!!
- Rotation is applied only at the critical node!
The balance factor of the critical node determines what rotation is to take place
- single or double
- If the child and the grand child (inserted node) of the critical node are on the same direction (both “/”) => single rotation
- Else => double rotation
- Rebalance similarly if coming up from the right sub-tree (opposite signs)
Performance

- Performance of membership operations on AVL trees:
  - easy for the worst case!
- An AVL tree will never be more than 45% higher than its perfectly balanced counterpart (Adelson, Velskii, Landis):
  - \( \log(N+1) \leq h_b(N) \leq 1.4404 \log(N+2) - 0.302 \)
Worst case AVL

- Sparse tree => each sub-tree has minimum number of nodes
Fibonacci Trees

- $T_h$: tree of height $h$
- $T_h$ has two sub-trees, one with height $h-1$ and one with height $h-2$
  - else it wouldn’t have minimum number of nodes
- $T_0$ is the empty sub-tree (also Fibonacci)
- $T_1$ is the sub-tree with 1 node (also Fibonacci)
Fibonacci Trees (cont.)

- **Average height**
  - \( N_h \) number of nodes of \( T_h \)
  - \( N_h = N_{h-1} + N_{h-2} + 1 \)
  - \( N_0 = 1 \)
  - \( N_1 = 2 \)
  - \( N_h = \frac{1}{\sqrt{5}} \left( \frac{1 + \sqrt{5}}{2} \right)^{h+2} - \frac{1}{\sqrt{5}} \left( \frac{1 - \sqrt{5}}{2} \right)^{h+2} - 1 \)
  - From which \( h \leq 1.44 \log(N+1) \)
More Examples

single rotation

insert 9
Examples (cont.)

double rotation

insert 7
Examples (cont.)

double rotation

insert 7
Examples (cont.)

single rotation

delete 6
Examples (cont.)

single rotation

delete 5
Examples (cont.)

double rotation

delete 5
General Deletions
General Deletions (cont.)
General Deletions (cont.)

```
delete 5
```

```
  3
 / \
2   10
/     /
1  7   11
      /
      9
```
Self Organizing Search

- **Splay Trees**: adapt to query patterns
  - move to root (front): whenever a node is accessed move it to the root using rotations
  - equivalent to move-to-front in arrays
- **current node**
  - insertions: inserted node
  - deletions: father of deleted node or null if this is the root
- **membership**: the last accessed node
Search(10)

first rotation

second rotation

third rotation
Splay Cases

- If the current node $q$ has no grandfather but it has father $p \Rightarrow$ only one single rotation
  - two symmetric cases: $L$, $R$

$$\begin{array}{c}
\text{p} \\
\text{a} \\
\text{q} \\
\text{b} \quad \text{c}
\end{array} \quad \rightarrow \quad \begin{array}{c}
\text{p} \\
\text{q} \\
\text{c} \\
\text{a} \quad \text{b}
\end{array}$$
- If \( p \) has also grandfather \( qp \) => 4 cases

\[
\begin{align*}
\text{LL symmetric of RR} \\
\text{RL symmetric of RL}
\end{align*}
\]
a, b, c ... are sub-trees
Splay Performance

- Splay trees adapt to unknown or changing probability distributions
- Splay trees do not guarantee logarithmic cost for each access
  - AVL trees do!
  - asymptotic cost close to the cost of the optimal BST for unknown probability distributions
- It can be shown that the “cost of $m$ operations on an initially empty splay tree, where $n$ are insertions is $O(m \log n)$ in the worst case”
Optimal BST

- Static environment: no insertions or deletions
- Keys are accessed with various frequencies
- Have the most frequently accessed keys near the root
- Application: a symbol table in main memory
Searching

- Given symbols $a_1 < a_2 < ... < a_n$ and their probabilities: $p_1, p_2, ... p_n$ minimize cost
- Successful search $cost = \sum_{i=1}^{n} p_i level(a_i)$
- Transform unsuccessful to successful
  - consider new symbols $E_1, E_2, ... E_n$

- $\infty ... a_1 ... a_2 ... a_i ... a_{i+1} ... a_n ... a_{n+1}$

- $E_0 \quad E_1 \quad E_2 \quad E_i \quad ... \quad E_n$

- $E_i = (a_i, a_{i+1}) \quad E_0 = (-\infty, a_1) \quad E_n = (a_n, \infty)$
Unsuccessful Search

An unsuccessful search for all values in $E_i$ terminates on the same failure node (in fact, one node higher)
Example

\((a_1, a_2, a_3) = (\text{do}, \text{if}, \text{read})\)

\[ p_i = q_i = \frac{1}{7} \]

\[ \text{cost} = \frac{15}{7} \]

Optimal BST

Trees in Main Memory

E.G.M. Petrakis
Read

Do

If

Cost = 15/7

Do

Read

If

Cost = 15/7
Search Cost

- If $p_i$ is the probability to search for $a_i$ and $q_i$ is the probability to search in $E_i$ then

$$\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} q_i = 1$$

$$\text{cost} = \sum_{i=1}^{n} p_i \cdot \text{level}(a_i) + \sum_{i=1}^{n} q_i \cdot \{\text{level}(E_i) - 1\}$$

- successful search
- unsuccessful search
Observation 1

- In a BST, a subtree has nodes that are consecutive in a sorted sequence of keys (e.g. [5,26])
Observation 2

- If $T_{ij}$ is a sub-tree of an optimal BST holding keys from $i$ to $j$ then
  - $T_{ij}$ must be optimal among all possible BSTs that store the same keys
  - optimality lemma: all sub-trees of an optimal BST are also optimal
Optimal BST Construction

1) Construct all possible trees:
   NP-hard, there are \( \frac{1}{n+1} \binom{2n}{n} \) trees!!

2) Dynamic programming solves the problem in polynomial time \( O(n^3) \)
   - at each step, the algorithm finds and stores the optimal tree in each range of key values
   - increases the size of the range at each step until the whole range is obtained
Example (successful search only):

keys
probabilities
1 10 20 40
0.3 0.2 0.1 0.4

1) BSTs with 1 node

range 1 10 20 40
cost= 0.3 0.2 0.1 0.4

2) BSTs with 2 nodes

k=1-10  k=10-20  k=20-20

range 1-10
1 10

cost=0.3 1+0.2 2=0.7

type= E.G.M. Petrakis

range 10-20
10 20

cost=0.2+0.1 2=0.4

range 20-40
20 40

cost=0.1+0.8=0.9

optimal

Trees in Main Memory

optimal

optimal

cost=0.4+0.2=0.6
3) BSTs with 3 nodes

- For $k=1-20$ and range 1-20:
  - Cost: $0.1 + 2(0.3 + 0.1) = 1.3$
  - Optimal tree:

- For $k=10-40$ and range 10-40:
  - Cost: $0.2 + 2(0.4 + 0.1) = 1.1$
  - Optimal tree:
4) BSTs with 4 nodes

range 1-40

1

40

10

20

cost = 0.3 + 2 \cdot 0.4 + 3 \cdot 0.2 + 4 \cdot 0.1 = 2.1

10

40

20

OPTIMAL BST

cost = 0.2 + 2 \cdot (0.3 + 0.4) + 3 \cdot 0.1 = 1.9

1

40

20

20

10

20

40

1

10

1

40

1

10

40

1

10

20

cost = 0.1 + 2 \cdot 0.3 + 3 \cdot 0.2 = 2.1

cost = 0.4 + 2 \cdot 0.3 + 3 \cdot 0.2 + 4 \cdot 0.1 = 2
Compute all optimal BSTs for all $C_{ij}$, $i,j = 1,2..n$

Let $m = j - i$: number of keys in range $C_{ij}$

$n - m - 1$ $C_{ij}$'s must be computed

The one with the minimum cost must be found, this takes $O(m(n-m-1))$ time

For all $C_{ij}$'s it takes $\sum_{1 \leq m \leq n} (nm - m^2) = O(n^3)$

There is a better $O(n^2)$ algorithm by Knuth

There is also a $O(n)$ greedy algorithm
Optimal BSTs

- High probability keys should be near the root
- But, the value of a key is also a factor
- It may not be desirable to put the smallest or largest key close to the root => this may result in skinny trees (e.g., lists)