Balanced Trees

Splay trees
2-3-4 trees
Red-black trees
B-trees


Symbol Table Review

Symbol table: key-value pair abstraction.
- Insert a value with specified key.
- Search for value given key.
- Delete value with given key.

Randomized BST.
- log N time per op (unless you get ridiculously unlucky).
- Store subtree count in each node.
- Generate random numbers for each insert/delete op.

This lecture.
- Splay trees.
- 2-3-4 trees.
- Red-black trees.
- B-trees.

Splay Trees

Splay trees = self-adjusting BST.
- Tree automatically reorganizes itself after each op.
- After inserting x or searching for x, rotate x up to root using double rotations.
- Tree remains "balanced" without explicitly storing any balance information.

Amortized guarantee: any sequence of N ops takes O(N log N) time.
- Height of tree can be N.
- Individual op can take linear time.

Splay.
- Check two links above current node.

⇒ ZIG-ZAG: if orientations differ, same as root insertion.
⇒ ZIG-ZIG: if orientations match, do top rotation first.

Diagram:

- ZIG-ZAG transformation:
  - Before: X to Y, D to Z
  - After: X to Y, C to B

X
  / \
A   2
    /
   Y

Z
  / \
A   B
    /
   C

Diagram:

- ZIG-ZAG transformation:
  - Before: X to Y, D to Z
  - After: X to Y, C to B

X
  / \
A   2
    /
   Y

Z
  / \
A   B
    /
   C

Diagram:

- ZIG-ZAG transformation:
  - Before: X to Y, D to Z
  - After: X to Y, C to B

X
  / \
A   2
    /
   Y

Z
  / \
A   B
    /
   C
Splay Trees

Splay.
- Check two links above current node.
  - ZIG-ZAG: if orientations differ, same as root insertion.
  - ZIG-ZIG: if orientations match, do top rotation first.

Splay Example

Search for 1.

ZIG-ZIG

Root = Splay Root Insertion Splay Insertion
Splay Example

Search for 1.

ZIG-ZIG

Splay Example

Search for 1.

ZIG-ZIG

Splay Example

Search for 1.

ZIG

Splay Example

Search for 1.

ZIG
Splay Example

Search for 2.

Splay trees

Intuition.
- Splay rotations halve search path.
- Reduces length of path for many other nodes in tree.

Symbol Table: Implementations Cost Summary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert†</td>
</tr>
<tr>
<td>Sorted array</td>
<td>log N</td>
<td>N</td>
</tr>
<tr>
<td>Unsorted list</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>Hashing</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>BST</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Randomized BST</td>
<td>log N †</td>
<td>log N †</td>
</tr>
<tr>
<td>Splay</td>
<td>log N ‡</td>
<td>log N ‡</td>
</tr>
</tbody>
</table>

- * assumes we know location of node to be deleted
- † if delete allowed, insert/search become sqrt(N)
- ‡ probabilistic guarantee
- § amortized guarantee

2-3-4 Trees

2-3-4 tree.
- Scheme to keep tree balanced.
- Generalize node to allow multiple keys.

Allow 1, 2, or 3 keys per node.
- 2-node: one key, two children.
- 3-node: two keys, three children.
- 4-node: three keys, four children.

Splay: sequence of any N ops in O(N log N) time.
Ahead: Can we do all ops in log N time guaranteed?
**2-3-4 Trees: Search and Insert**

**SEARCH.**
- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

**INSERT.**
- Search to bottom for key.
- 2-node at bottom: convert to 3-node.
- 3-node at bottom: convert to 4-node.
- 4-node at bottom: ??

**2-3-4 Trees: Splitting Four Nodes**

Transform tree on the way DOWN.
- Ensure that last node is not a 4-node.

**Local transformation to split 4-nodes:**

Invariant: current node is not a 4-node.
- One of two above transformations must apply at next node.
- Insertion at bottom is easy since it’s not a 4-node.

**2-3-4 Trees: Splitting a Four Node**

Splitting a four node: move middle key up.
Balance in 2-3-4 Trees

All paths from top to bottom have exactly the same length.

Tree height.
- Worst case: $\lg N$ all 2-nodes
- Best case: $\log_4 N = 1/2 \lg N$ all 4-nodes
- Between 10 and 20 for a million nodes.
- Between 15 and 30 for a billion nodes.

Comparison within nodes not accounted for.

2-3-4 Trees: Implementation?

Direct implementation complicated because of:
- Maintaining multiple node types.
- Implementation of `getElement`.
- Large number of cases for `split`.

```java
private Node insert(Node h, String key, Object value) {
    Node x = h;
    while (x != null) {
        x = x.getChild(key);
        if (x.is4Node()) x.split();
    }
    if (x.is2Node()) x.make3Node(key, value);
    else if (x.is3Node()) x.make4Node(key, value);
}
```

Fantasy Code

Red-Black Trees

Represent 2-3-4 trees as binary trees.
- Use "internal" edges for 3- and 4- nodes.

- Correspondence between 2-3-4 trees and red-black trees.

- Not 1-1 because 3-nodes swing either way.

Splitting Nodes in Red-Black Trees

Two cases are easy: switch colors.

Two cases require rotations.

- do single rotation
- do double rotation
Red-Black Tree Node Split Example

- Inserting $G$:
  - Change colors:
  - Right rotate $R$ →
  - Left rotate $E$ →

Red-Black Tree Construction

Balance in Red-Black Trees

Length of longest path is at most twice the length of shortest path.

Tree height.
- Worst case: $2 \log N$.

Comparison within nodes ARE counted.

Symbol Table: Implementations Cost Summary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted array</td>
<td>$\log N$</td>
<td>$N$</td>
<td>$N$</td>
<td>$\log N$</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>Unsorted list</td>
<td>$N$</td>
<td>$1$</td>
<td>$1$</td>
<td>$N$</td>
<td>$1^*$</td>
<td>$1^*$</td>
</tr>
<tr>
<td>Hashing</td>
<td>$N$</td>
<td>$1$</td>
<td>$N$</td>
<td>$1^*$</td>
<td>$1^*$</td>
<td>$1^*$</td>
</tr>
<tr>
<td>BST</td>
<td>$N$</td>
<td>$N$</td>
<td>$N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\sqrt(N)^{\dagger}$</td>
</tr>
<tr>
<td>Randomized BST</td>
<td>$\log N^{\dagger}$</td>
<td>$\log N^{\dagger}$</td>
<td>$\log N^{\dagger}$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
</tr>
<tr>
<td>Splay</td>
<td>$\log N^{\ddagger}$</td>
<td>$\log N^{\ddagger}$</td>
<td>$\log N^{\ddagger}$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
</tr>
<tr>
<td>Red-Black</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
<td>$\log N$</td>
</tr>
</tbody>
</table>

* assumes hash map is random for all keys
† if delete allowed, insert/search become $\sqrt(N)$
‡ probabilistic guarantee
§ amortized guarantee
**Red-Black Trees in Practice**

- Red-black trees vs. splay trees.
  - Fewer rotations than splay trees.
  - One extra bit per node for color. Possible to eliminate.
- Red-black trees vs. hashing.
  - Hashing code is simpler and usually faster.
  - Arithmetic to compute hash vs. comparison.
  - Hashing performance guarantee is weaker.
  - BSTs have more flexibility and can support wider range of ops.

Red-black trees are widely used as system symbol tables.
- Java: TreeMap, TreeSet.
- C++ STL: map, multimap, multiset.

**Symbol Table: Java Libraries**

Java has built-in library for red-black tree symbol table.
- TreeMap = red-black tree implementation.

```java
import java.util.TreeMap;
public class TreeMapDemo {
    public static void main(String[] args) {
        TreeMap st = new TreeMap();
        st.put("www.cs.princeton.edu", "128.112.136.11");
        st.put("www.princeton.edu", "128.112.128.15");
        st.put("www.simpsons.com", "209.052.165.60");
        System.out.println(st.get("www.cs.princeton.edu"));
    }
}
```

Duplicate policy.
- Java TreeMap forbids two elements with the same key.
- Sedgewick implementations allow duplicate keys.

**B-Trees**

B-Tree generalize 2-3-4 trees by allowing up to $M$ links per node.
- Split full nodes on the way down.

**Main application: file systems.**
- Reading a page into memory from disk is expensive.
- Accessing info on a page in memory is free.
- Goal: minimize # page accesses.
- Node size $M =$ page size.

**Space-time tradeoff.**
- $M$ large $\Rightarrow$ only a few levels in tree.
- $M$ small $\Rightarrow$ less wasted space.
- Typical $M = 1000$, $N < 1$ trillion.

Bottom line: number of PAGE accesses is $\log_M N$ per op.
- 3 or 4 in practice!

**B-Tree Example**

$M = 5$

Item = Key = int
B-Tree Example (cont)

Symbol Table: Implementations Cost Summary

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Worst Case</th>
<th>Average Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
</tr>
<tr>
<td>Sorted array</td>
<td>log N</td>
<td>N</td>
</tr>
<tr>
<td>Unsorted list</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>Hashing</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>BST</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Randomized BST</td>
<td>log N</td>
<td>log N</td>
</tr>
<tr>
<td>Splay</td>
<td>log N</td>
<td>log N</td>
</tr>
<tr>
<td>Red-Black</td>
<td>log N</td>
<td>log N</td>
</tr>
</tbody>
</table>

B-Tree in the Wild

File systems.
- Window HPFS (high performance file system).
- Mac HFS (hierarchical file system).
- Linux: ReiserFS, XFS, Ext3FS, JFS. journaling

Databases.
- Most common index type in modern databases.
- ORACLE, DB2, INGRES, SQL, PostgreSQL, . . .

Variants.
- B trees: Bayer-McCreight (1972, Boeing)
- B+ trees: all data in external nodes.
- B* trees: keeps pages at least 2/3 full.
- R-trees for spatial searching: GIS, VLSI.

Summary

Goal: ST implementation with log N guarantee for all ops.
- Probabilistic: randomized BST.
- Amortized: splay tree, hashing.
- Worst-case: red-black tree. from re-doubling.
- Algorithms are variations on a theme: rotations when inserting.

Abstraction extends to give search algorithms for huge files.
- B-tree.