4.4 Balanced Trees

Reference: Chapter 13, Algorithms in Java, 3rd Edition, Robert Sedgewick.

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.

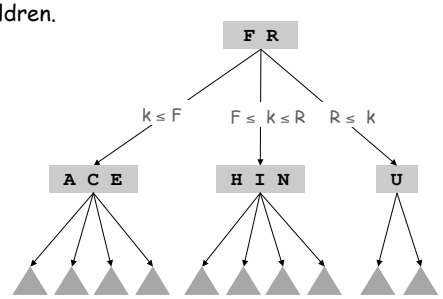
Randomized BST.
- O(log N) time per op. ← unless you get ridiculously unlucky
- Store subtree count in each node.
- Generate random numbers for each insert/delete op.

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

2-3-4 Trees

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

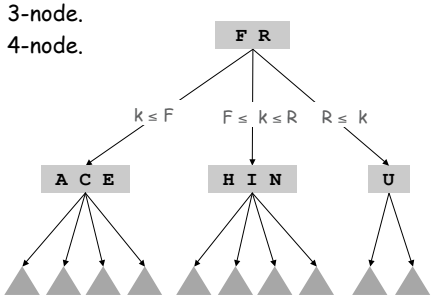
Allow 1, 2, or 3 keys per node.
- 2-node: one key, two children.
- 3-node: two keys, three children.
- 4-node: three keys, four children.



2-3-4 Trees: Search and Insert

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

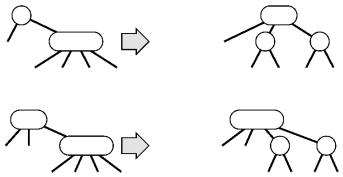
Insert.
- Search to bottom for key.
- 2-node at bottom: convert to 3-node.
- 3-node at bottom: convert to 4-node.
- 4-node at bottom: ??



2-3-4 Trees: Splitting Four Nodes

Transform tree on the way down.

- Ensures last node is not a 4-node.
- Local transformation to split 4-nodes:

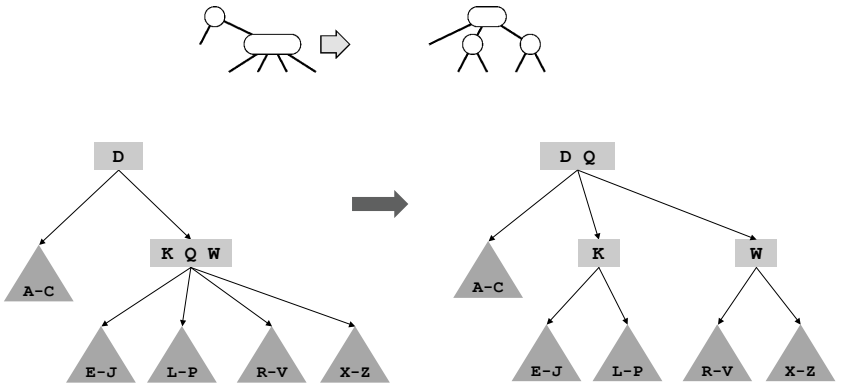


Invariant: current node is not a 4-node.

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

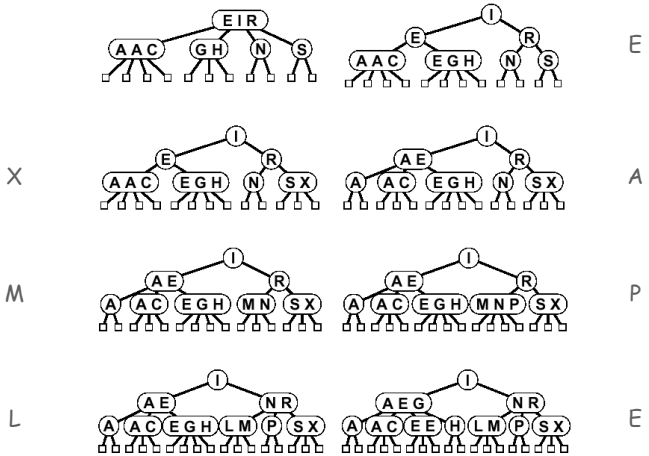
2-3-4 Trees: Splitting a Four Node

Splitting a four node: move middle key up.



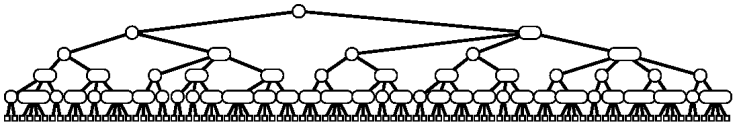
2-3-4 Trees

Tree grows up from the bottom.



Balance in 2-3-4 Trees

Property. 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.

Note. Comparison within nodes not accounted for.

2-3-4 Trees: Implementation?

Direct implementation complicated because of:

- Maintaining multiple node types.
- Implementation of `getChild`.
- Large number of cases for `split`.

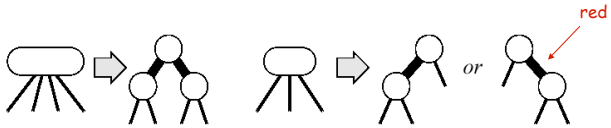
```
private Node insert(Node h, Key key, Value val) {
   Node x = h;
   while (x != null) {
      x = x.getChild(key);
      if (x.is4Node()) x.split();
   }
   if      (x.is2Node()) x.make3Node(key, val);
   else if (x.is3Node()) x.make4Node(key, val);
}
```

fantasy code

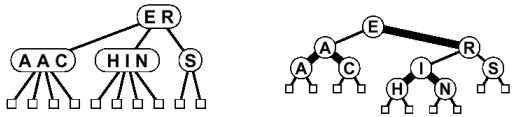
Red-Black Trees

Represent 2-3-4 trees as binary trees.

- Use "internal" edges for 3- and 4- nodes.



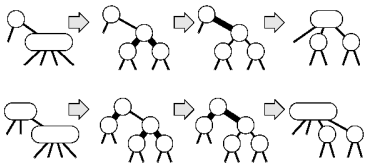
- Correspondence between 2-3-4 trees and red-black trees.



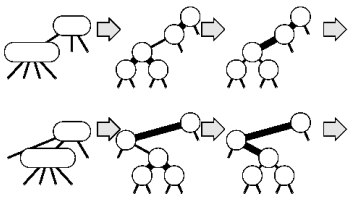
- Not 1-1 because 3-nodes swing either way.

Splitting Nodes in Red-Black Trees

Two easy cases: switch colors.



Two hard cases: use rotations.

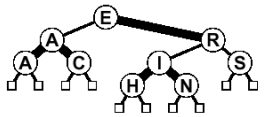


do single rotation

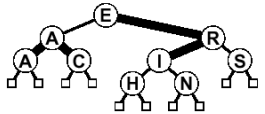
do double rotation

Red-Black Tree Node Split Example

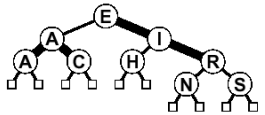
inserting G



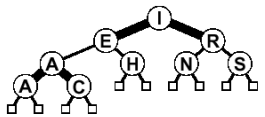
change colors



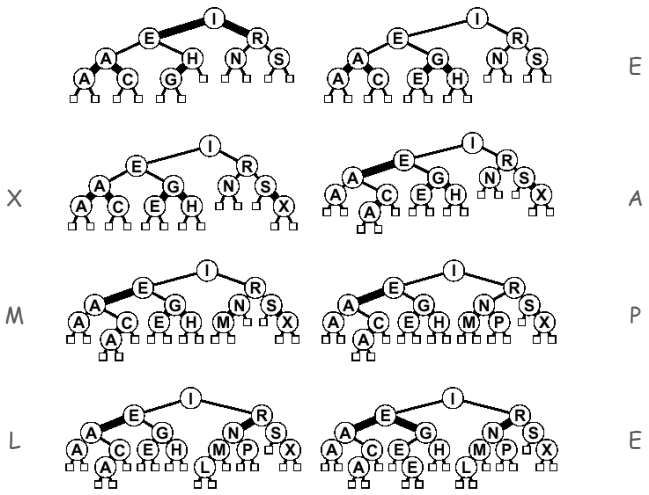
right rotate R →



left rotate E →

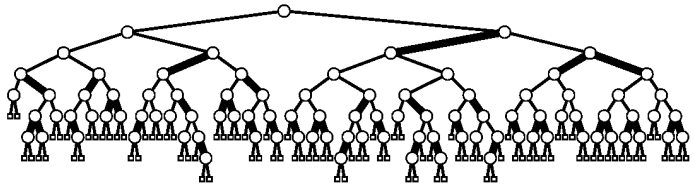


Red-Black Tree Construction



Balance in Red-Black Trees

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



Tree height. Worst case: 2 lg N.

Note. Comparison within nodes are counted.

Symbol Table: Implementations Cost Summary

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete
Sorted array	log N	N	N	log N	N	N
Unsorted list	N	1	1	N	1	1
Hashing	N	1	N	1*	1*	1*
BST	N	N	N	log N †	log N †	log N †
Randomized BST	log N ‡	log N ‡	log N ‡	log N	log N	log N
Splay	log N §	log N §	log N §	log N §	log N §	log N §
Red-Black	log N	log N	log N	log N	log N	log N

* assumes hash map is random for all keys
† N is the number of nodes ever inserted
‡ probabilistic guarantee
§ amortized guarantee

Red-Black Trees in Practice

Red-black trees vs. splay trees.

- Fewer rotations than splay trees. ← at most 2 per insertion
- 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.

Red Black Tree: Java Library

Java has built-in libraries for symbol tables.

- `TreeMap` = red black tree implementation.

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

Duplicate policy.

- Java `HashMap` allows `null` values.
- Our implementations forbid `null` values.

B-Trees

B-Tree. Generalizes 2-3-4 trees by allowing up to M links per node.

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.

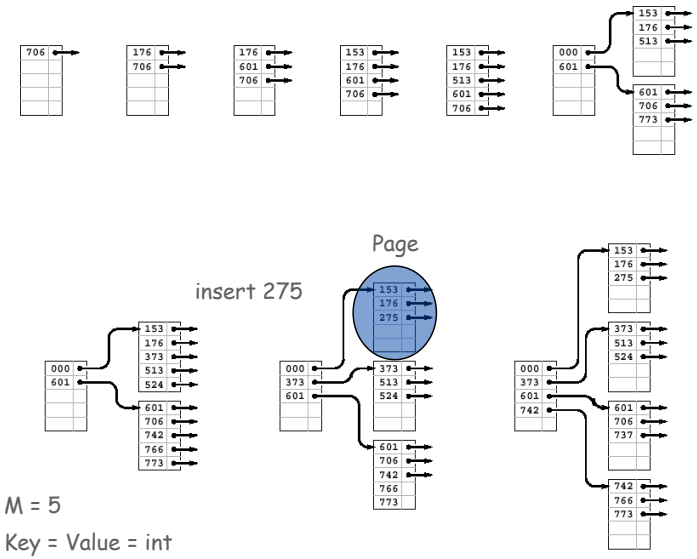
Space-time tradeoff.

- M large ⇒ only a few levels in tree.
- M small ⇒ less wasted space.
- Typical M = 1000, N < 1 trillion.

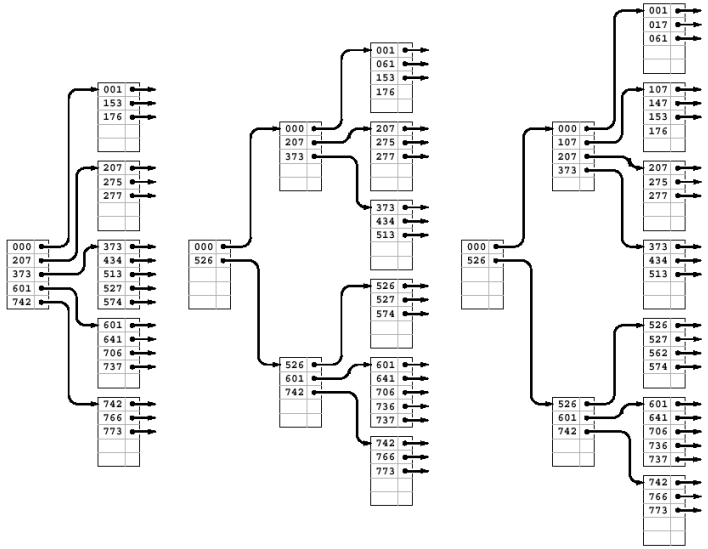
Bottom line: number of page accesses is $\log_M N$ per op.

- 3 or 4 in practice!

B-Tree Example



B-Tree Example (cont)



Symbol Table: Implementations Cost Summary

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete
Sorted array	log N	N	N	log N	N / 2	N / 2
Unsorted list	N	N	N	N	N	N
Hashing	N	1	N	1*	1*	1*
BST	N	N	N	log N †	log N †	log N †
Randomized BST	log N ‡	log N ‡	log N ‡	log N	log N	log N
Splay	log N §	log N §	log N §	log N §	log N §	log N §
Red-Black	log N	log N	log N	log N	log N	log N
B-Tree	1	1	1	1	1	1

page accesses

B-Tree: Number of PAGE accesses is $\log_M N$ per op.

B-Trees in the Wild

File systems.
- Windows: HPFS.
- Mac: HFS, HFS+.
- 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.
- Worst-case: red-black tree.
- Algorithms are variations on a theme: rotations when inserting.

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