Balanced Search Trees

These lecture slides have been adapted from:

- *Algorithms in C*, 3rd Edition, Robert Sedgewick.

Symbol Table Review

Symbol table, dictionary.

- Set of items with keys.
- INSERT a new item.
- SEARCH for an existing item with a 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.
- Red-black trees.
- B-trees.

Splay Trees

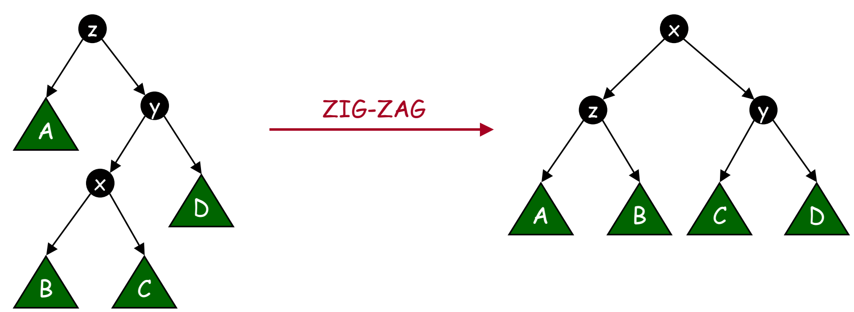
Splay trees.

- Self-adjusting BST.
  - tree automatically reorganizes itself after each op
  - when insert or search 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 Trees

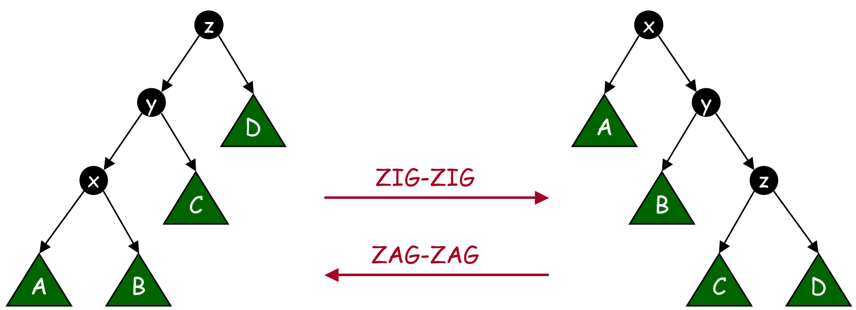
Splay.

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



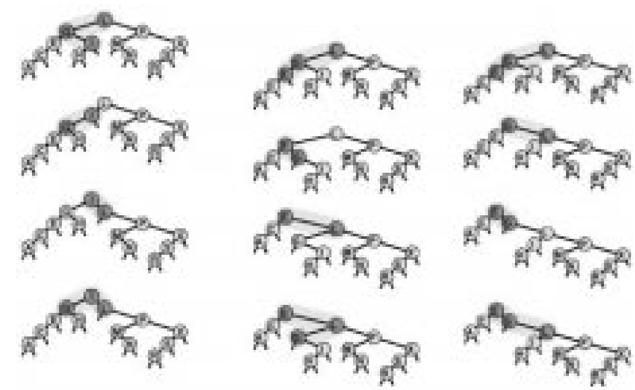
Splay Trees

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 Trees

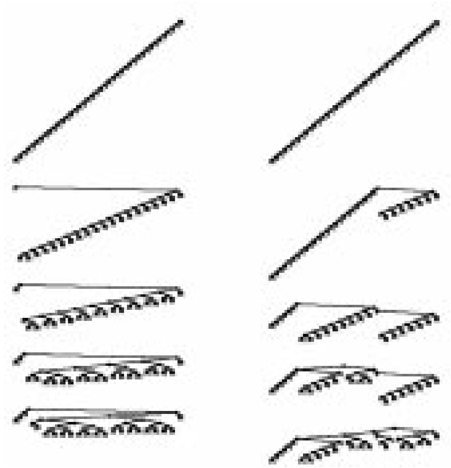
Splay.
- Check two links above current node.
- ZIG-ZAG: if orientations differ, same as root insertion.
- ZIG-ZIG: if orientations match, do top rotation first.



Root = Splay   Root Insertion   Splay Insertion

Splay Trees

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



insert 1, 2, …, 40 | insert 1, 2, …, 40

search 1 | search for random key

search 2

search 3

search 4

Symbol Table: Implementations Cost Summary

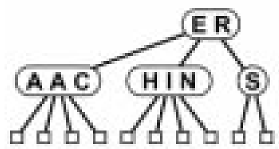
	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete *
Unsorted array	N	1	1	N/2	1	1
Sorted array	log N	N	N	log N	N/2	N/2
BST	N	N	N	log N	log N	sqrt(N) †
Randomized	log N ‡	log N ‡	log N ‡	log N	log N	log N
Splay	log N §	log N §	log N §	log N §	log N §	log N §

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

Splay: sequence of any N ops in O(N log N) time.
Ahead: Can we do all ops in log N time?

2-3-4 Trees

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.



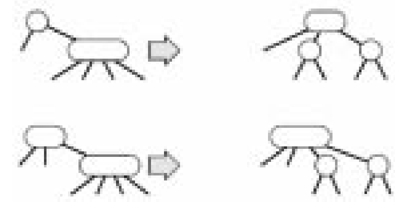
SEARCH.
- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).

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

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

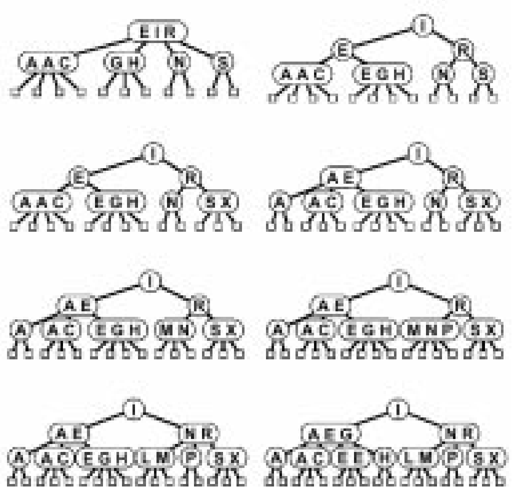
Local transformation to split 4-nodes:



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

Tree grows up from the bottom.



E

X

A

M

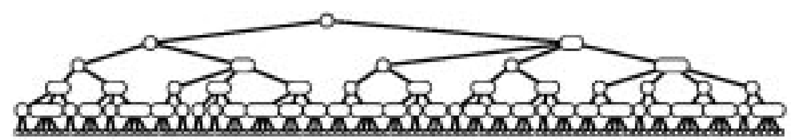
P

L

E

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: $\log_4 N = \frac{1}{2} \lg N$ 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:

- therightlink().
- Maintaining multiple node types.
- Large number of cases for split().

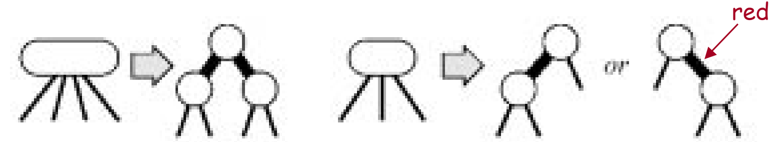
Fantasy Code

```
link insertR(link h, Item item) {
   Key v = ITEMkey(item);
   link x = h;
   while (x != NULL) {
      x = therightlink(x, v);
      if fournode(x) then split(x);
   }
   if twonode(x) then makethree(x, v);
   else if threenode(x) then makefour(x, v);
}
return head;
```

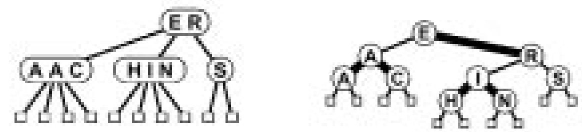

Red-Black Trees

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.

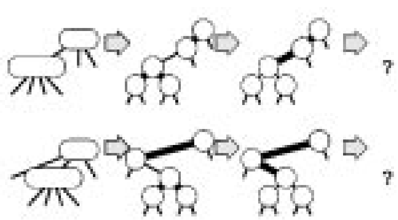


- Not 1-1 because 3-nodes swing either way.

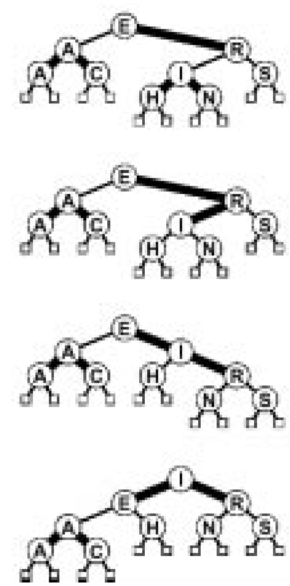
Splitting Nodes in Red-Black Trees

Two cases are easy: switch colors.

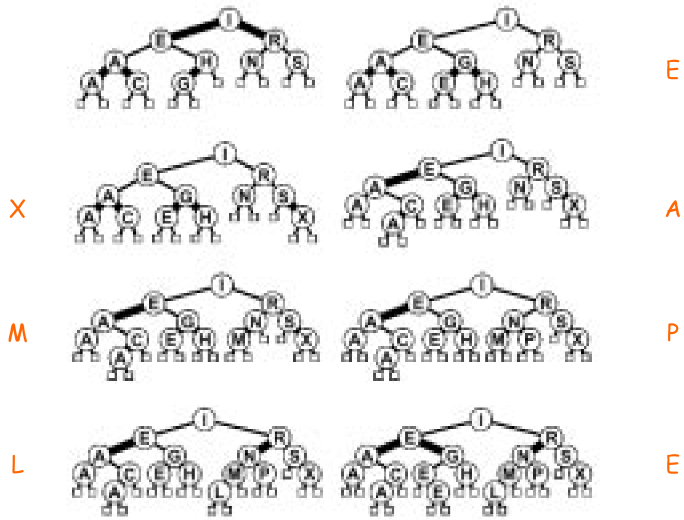
Two cases require ROTATIONS.



Red-Black Tree Node Split Example

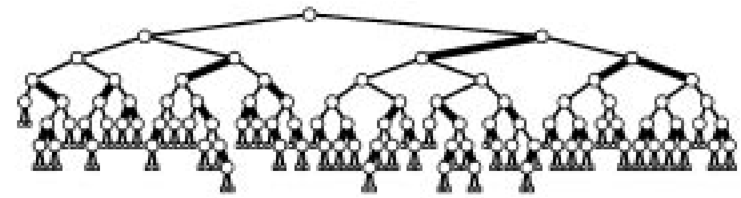


Red-Black Tree Construction



Balance in Red-Black Trees

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



Tree height.

- Worst case: 2 lg N.

Comparison within nodes ARE counted.

Symbol Table: Implementations Cost Summary

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete *
Unsorted array	N	1	1	N/2	1	1
Sorted array	log N	N	N	log N	N/2	N/2
BST	N	N	N	log N	log N	sqrt(N) †
Randomized	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 we know location of node to be deleted
† if delete allowed, insert/search become sqrt(N)
‡ probabilistic guarantee
§ amortized guarantee

Red-Black Trees in Practice

Efficient.

- Fewer rotations than splay trees.
- Can even eliminate the 1 bit of storage needed for color.

Flexible.

- Interval trees.
- Order statistic trees.

Widely used as system symbol table.

- Java: TreeMap, TreeSet.
- C++ STL: map, multimap, multiset.

B-Trees

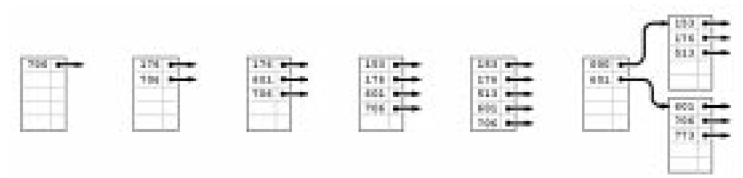
B-Tree generalize 2-3-4 trees by allowing up to M links per node.
- Split full nodes on the way down.

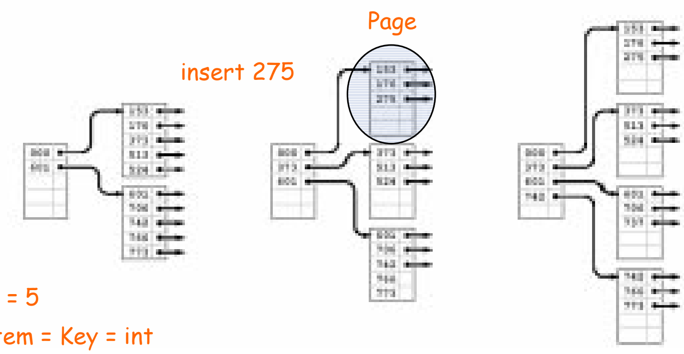
Main application: file systems.
- Reading a page from disk is expensive.
- Accessing info on a page is free.
- Goal: minimize # page accesses.
- Node size M = page size.

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

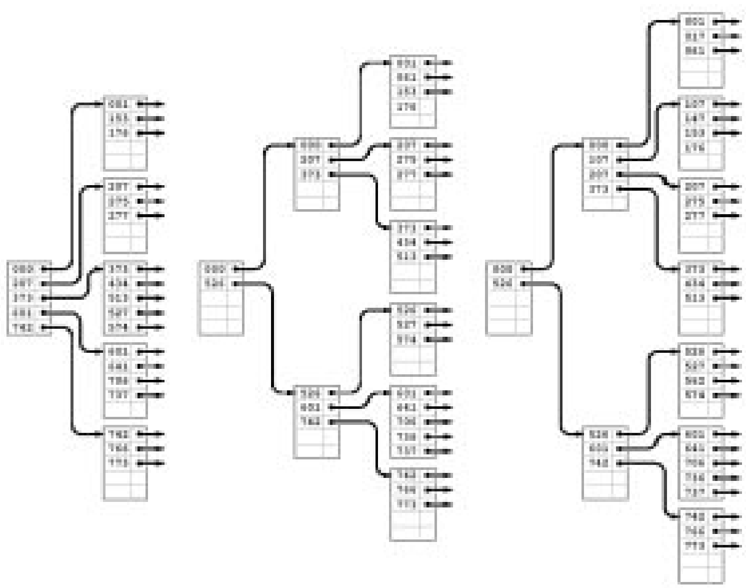




M = 5

Item = Key = int

B-Tree Example (cont)



Symbol Table: Implementations Cost Summary

	Worst Case			Average Case		
Implementation	Search	Insert	Delete	Search	Insert	Delete *
Unsorted array	N	1	1	N/2	1	1
Sorted array	log N	N	N	log N	N/2	N/2
BST	N	N	N	log N	log N	sqrt(N) †
Randomized	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.

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.