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. 2-3-4 trees, red-black trees, B-trees.

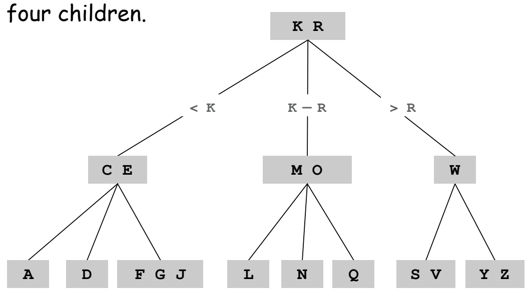
2-3-4 Trees

2-3-4 Tree

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

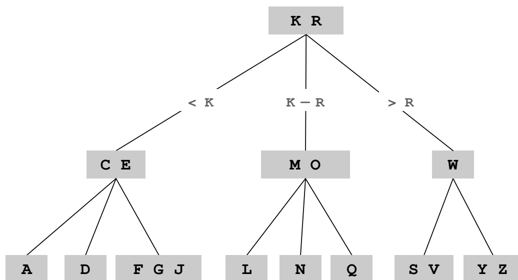
Perfect balance. Every path from root to leaf has same length.

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.



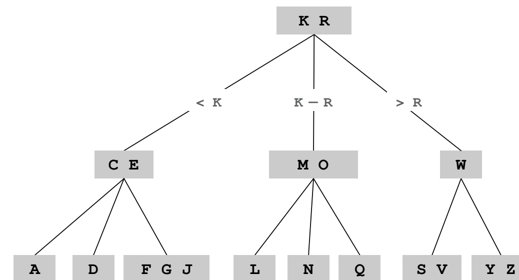
2-3-4 Tree: Search

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



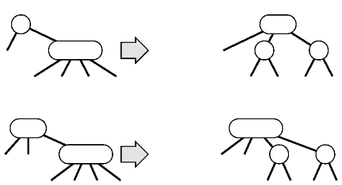
2-3-4 Tree: Insert

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 Tree: Splitting Four Nodes

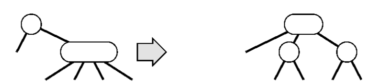
Transform tree on the way down.
- Ensures last node is not a 4-node.
- Local transformation to split 4-nodes:

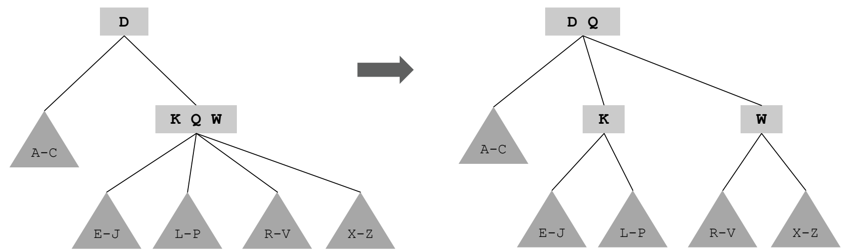


Invariant. Current node is not a 4-node.
Consequence. Insertion at bottom is easy since it's not a 4-node.

2-3-4 Tree: Splitting a Four Node

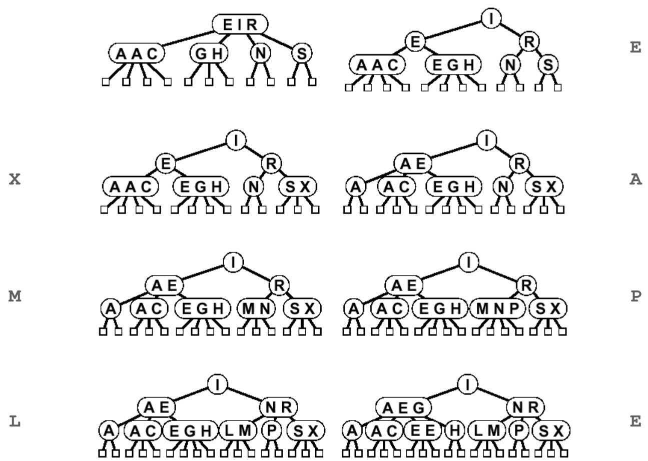
Ex. To split a four node, move middle key up.





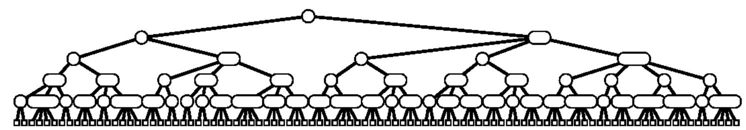
2-3-4 Tree

Tree grows up from the bottom.



2-3-4 Tree: Balance

Property. All paths from root to leaf have 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.

2-3-4 Tree: Implementation?

Direct implementation. Complicated because of:

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

```
private void insert(Key key, Val val) {
   Node x = root;
   while (x.getChild(key) != 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

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 §
2-3-4	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

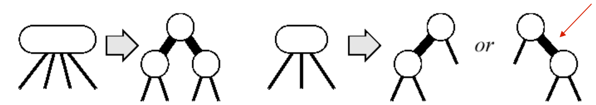
Note. Comparison within nodes not accounted for.

Red-Black Trees

Red-Black Tree

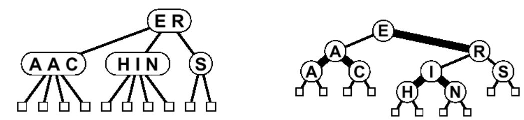
Represent 2-3-4 tree as a BST.

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



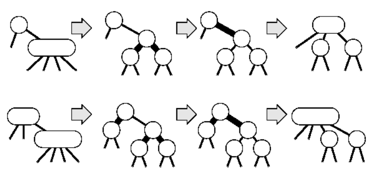
not 1-1 because 3-nodes swing either way.

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



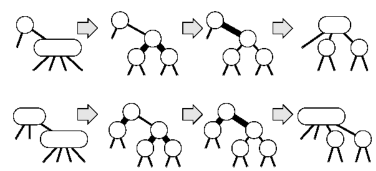
Red-Black Tree: Splitting Nodes

Two easy cases. Switch colors.

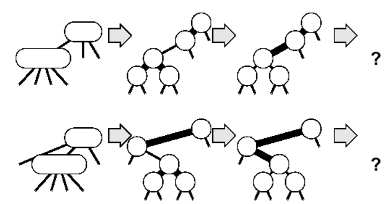


Red-Black Tree: Splitting Nodes

Two easy cases. Switch colors.



Two hard cases. Use rotations.



do single rotation

do double rotation

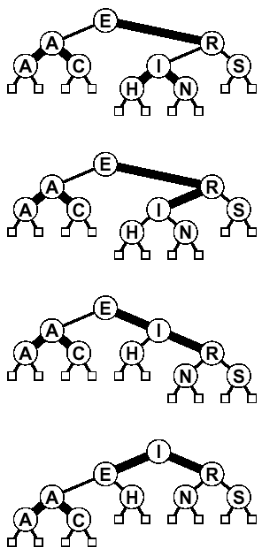
Red-Black Tree: Splitting Nodes

inserting G

change colors

right rotate R →

left rotate E →



Red-Black Tree: Insertion

E

X

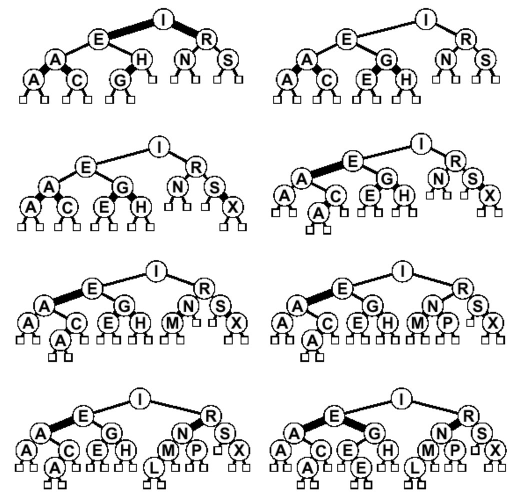
A

M

P

L

E

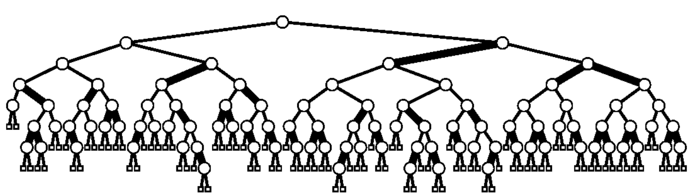


Red-Black Tree: Balance

Property A. Every path from root to leaf has same number of black links.

Property B. At most one red link in-a-row.

Property C. Height of tree is less than 2 lg N + 2.



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

Note. Comparison within nodes are accounted for.

Red-Black Trees: 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.

In the wild. Red-black trees are widely used as system symbol tables.
- Java: `java.util.TreeMap`, `java.util.TreeSet`.
- C++ STL: map, multimap, multiset.
- Linux kernel: `linux/rbtree.h`.

B-Trees

B-Tree

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

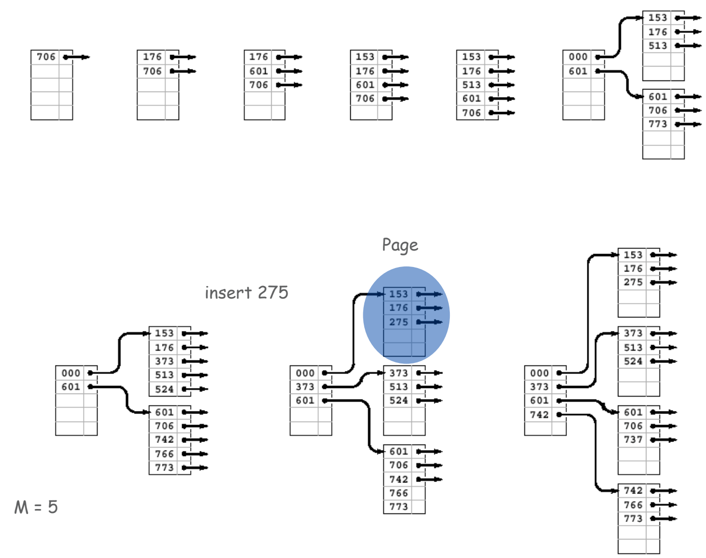
Main application: file systems.
- Reading a page into memory from disk is expensive.
- Accessing info on a page in memory is free.
- Goal: minimize # page accesses.
- Node size M = page size.

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

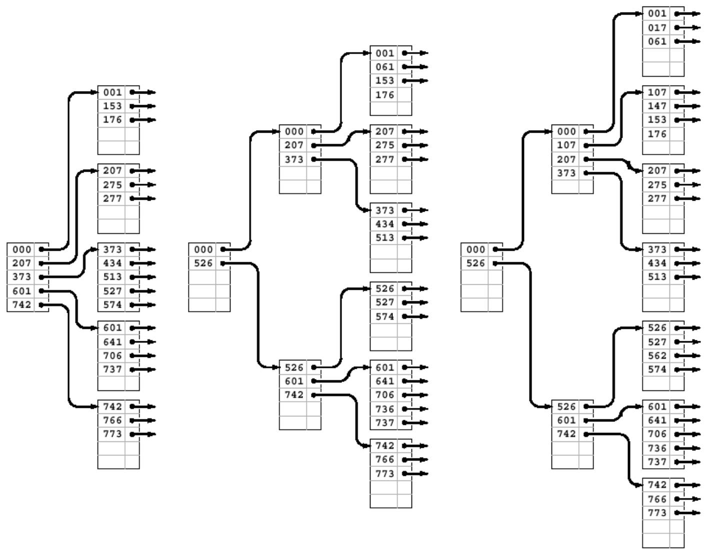
Bottom line. Number of page accesses is $\log_M N$ per op.

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.

effectively a constant

B-Trees in the Wild

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.

File systems.
- Windows: HPFS.
- Mac: HFS, HFS+.
- Linux: ReiserFS, XFS, Ext3FS, JFS.

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

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.

Splay Trees

Splay Trees

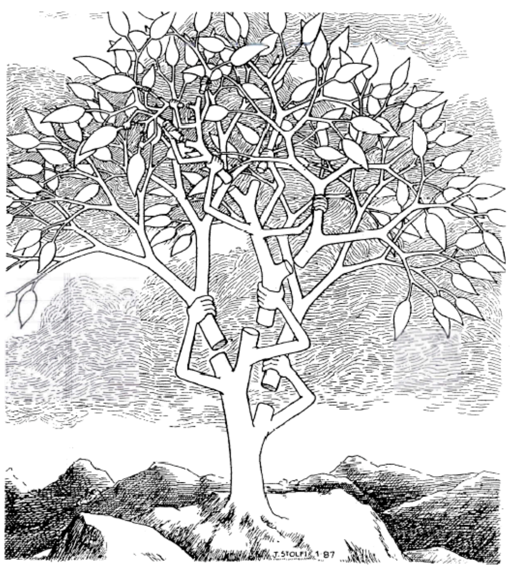
Splay trees = self-adjusting BST.

- 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, starting from empty splay tree, takes O(N log N) time.

- Height of tree can be N.
- Individual op can take linear time.

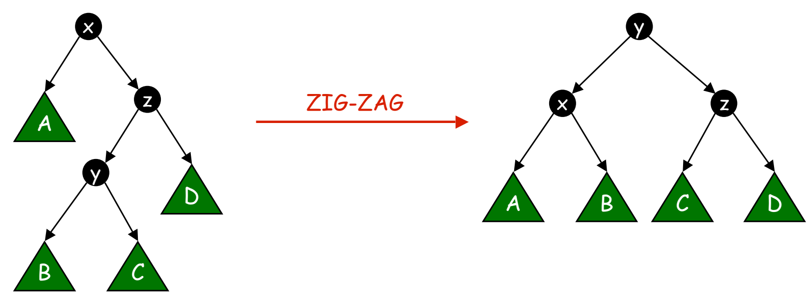
A Self-Adjusting Tree



Splay Trees

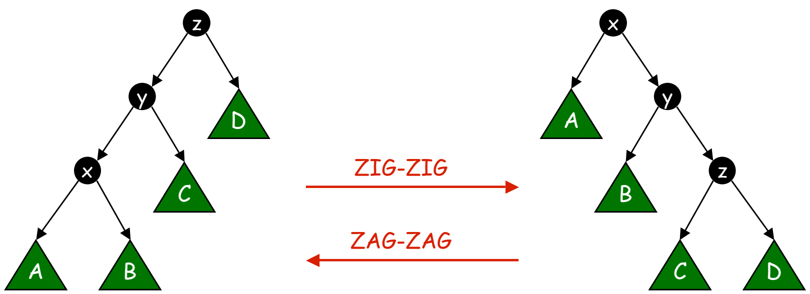
Splay.

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



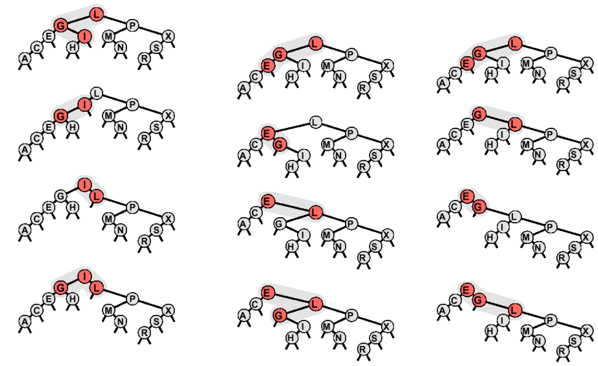
Splay Trees

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

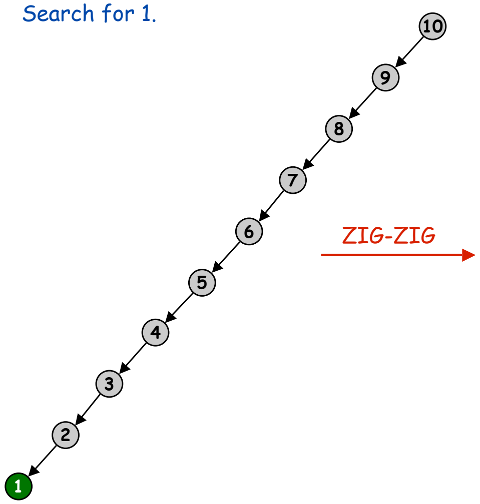
Splay.
- Check two links above current node.
- ZIG-ZAG: if orientations differ, same as root insertion.
- ZIG-ZIG: if orientations match, do top rotation first.



Root = Splay   Root Insertion   Splay Insertion

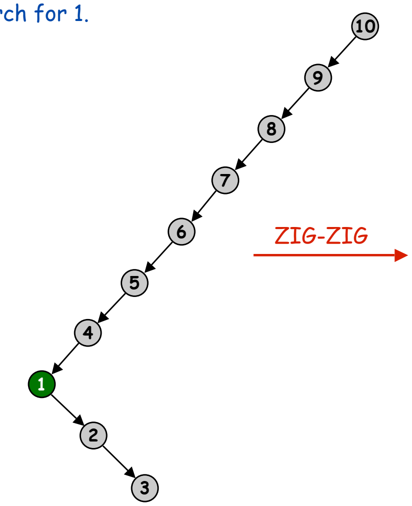
Splay Example

Search for 1.



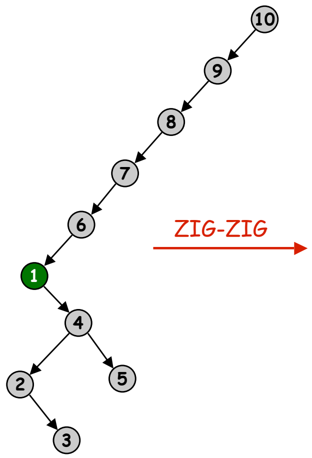
Splay Example

Search for 1.



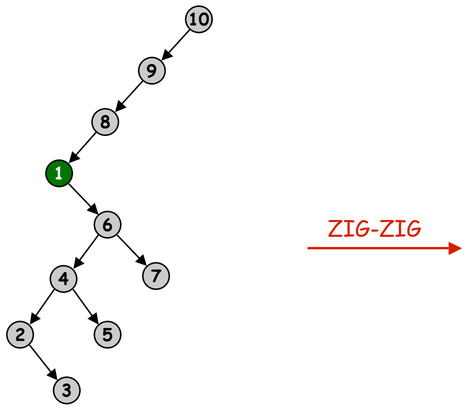
Splay Example

Search for 1.



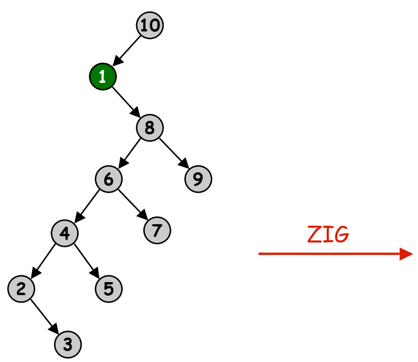
Splay Example

Search for 1.



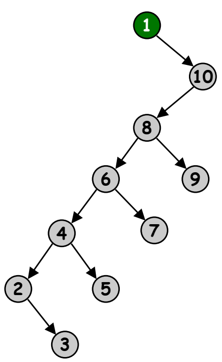
Splay Example

Search for 1.



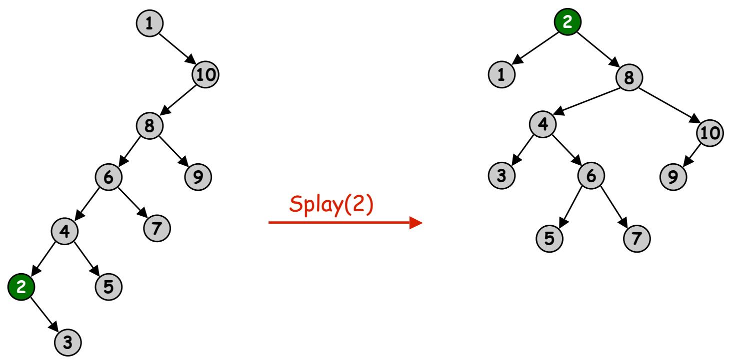
Splay Example

Search for 1.



Splay Example

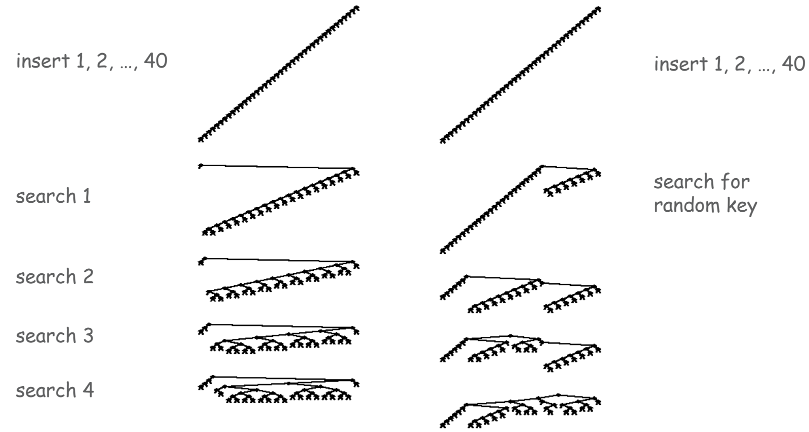
Search for 2.



Splay Trees

Intuition.

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



Symbol Table: Implementations Cost Summary

	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	sqrt(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 §

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

Splay: Sequence of N ops takes linearithmic time.
Ahead: Can we do all ops in log N time guaranteed?