Flipped Lecture Concepts

Week 6

Ananda Guna

10.16.14

Plan today

- 2-3 trees
- Red-black trees
- Hashing
- worksheet

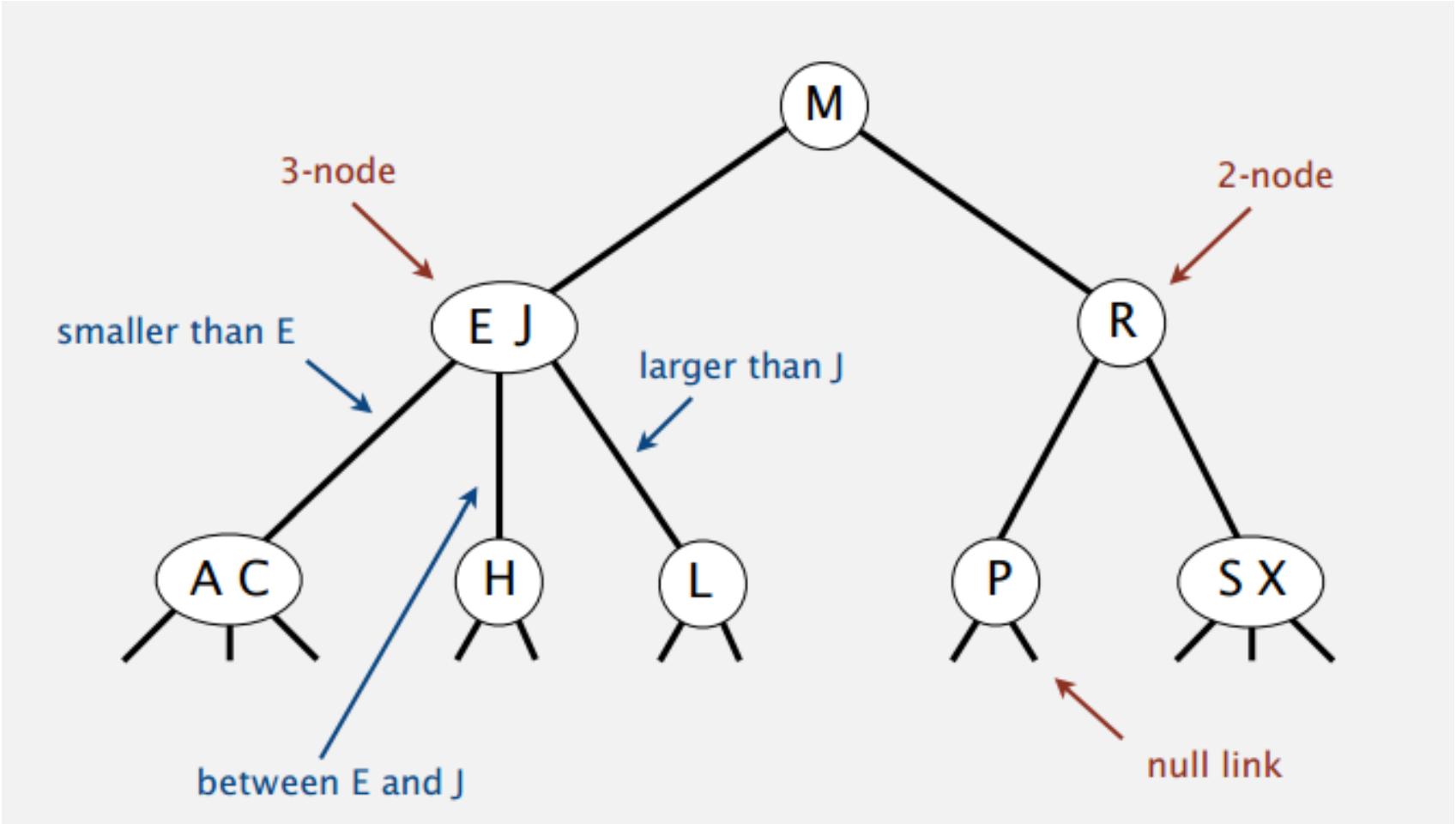
Balanced Trees

- The goal is to create a well balanced BST regardless of the input order.
- Why don’t we just randomize the elements and build a BST?
- Desired performance for BST
  - Insert in log N
  - Delete in log N
  - Find in log N

Implementing ST

implementation	guarantee			average case			ordered ops?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N	N	N		`equals()`
binary search (ordered array)	$\log N$	N	N	$\log N$	N	N	✔	`compareTo()`
BST	N	N	N	$\log N$	$\log N$	$\sqrt{N}$	✔	`compareTo()`
goal	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	✔	`compareTo()`

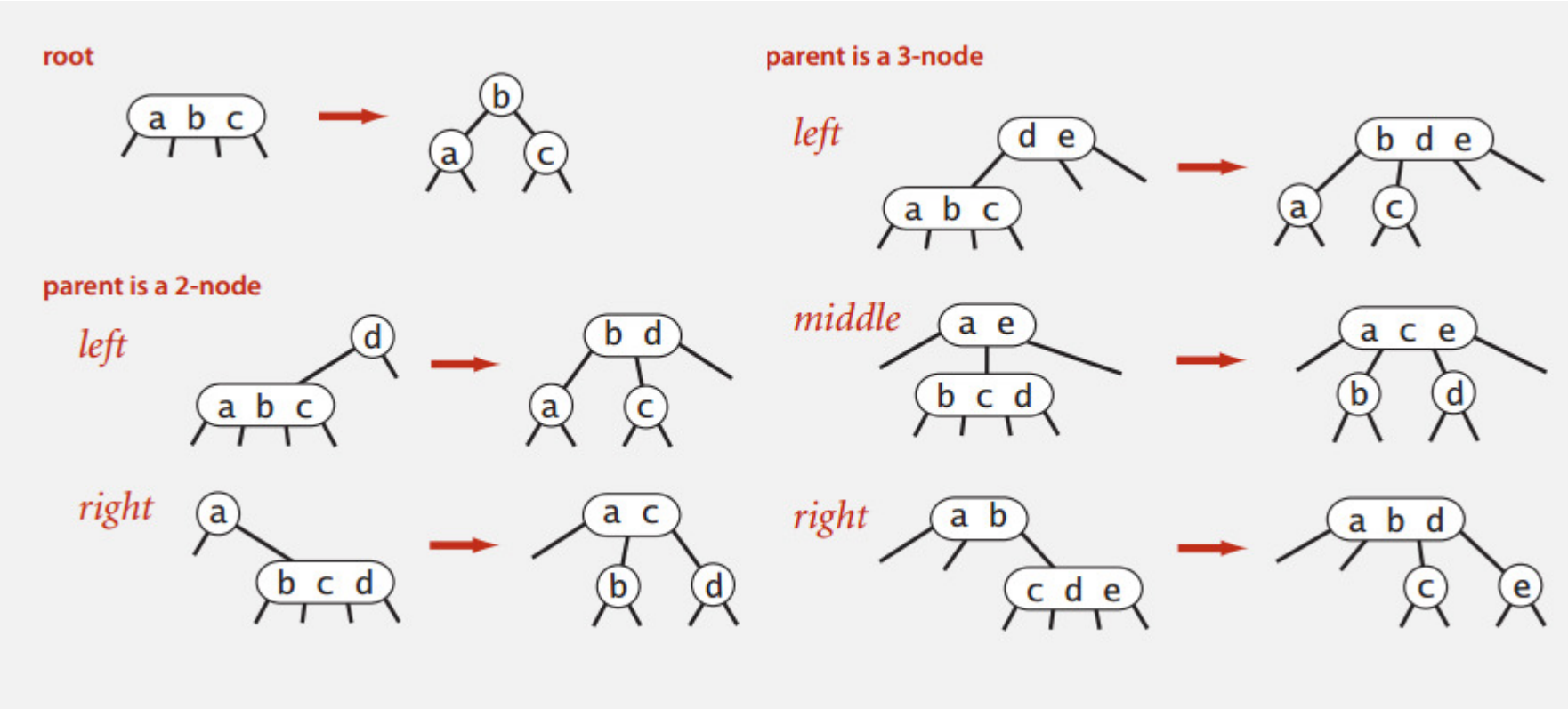
2-3 Trees



Two invariants

- Balance invariant – each path from root to leaf nodes have the same length
- Order invariant – an inorder traversal of the tree produces an ordered sequence

2-3 Tree operations



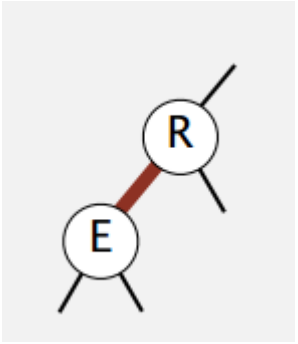
Exercise: Insert 6, 10, 15, 8 , 9 , 20, 30, 40, 50, 25

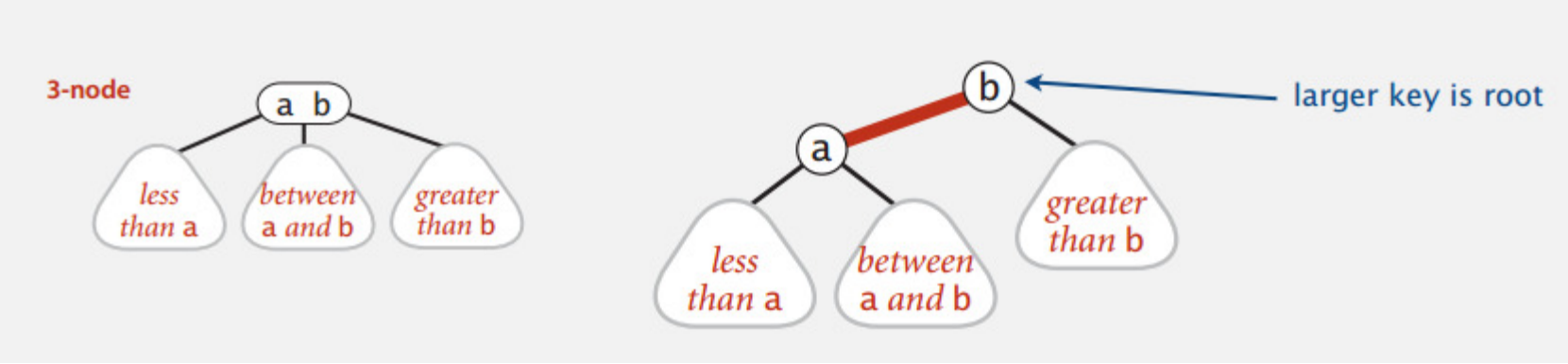
2-3 Trees Facts

- Worst case - $\log_2 N$
- Best Case – $\log_3 N$
- Direct implementation is complicated
  - Maintaining multiple node types is cumbersome.
  - Need multiple compares to move down tree.
  - Need to move back up the tree to split 4-nodes.
  - Large number of cases for splitting.
- Not practical to implement

Red-black trees

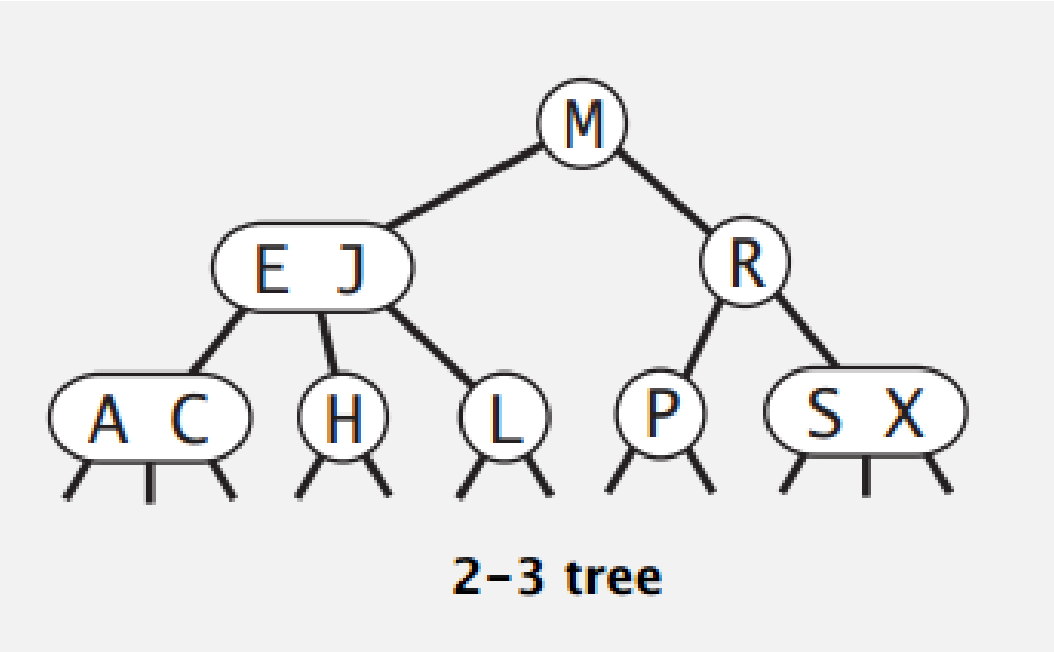
- How to represent 3-nodes?
  - Regular BST with red "glue" links.



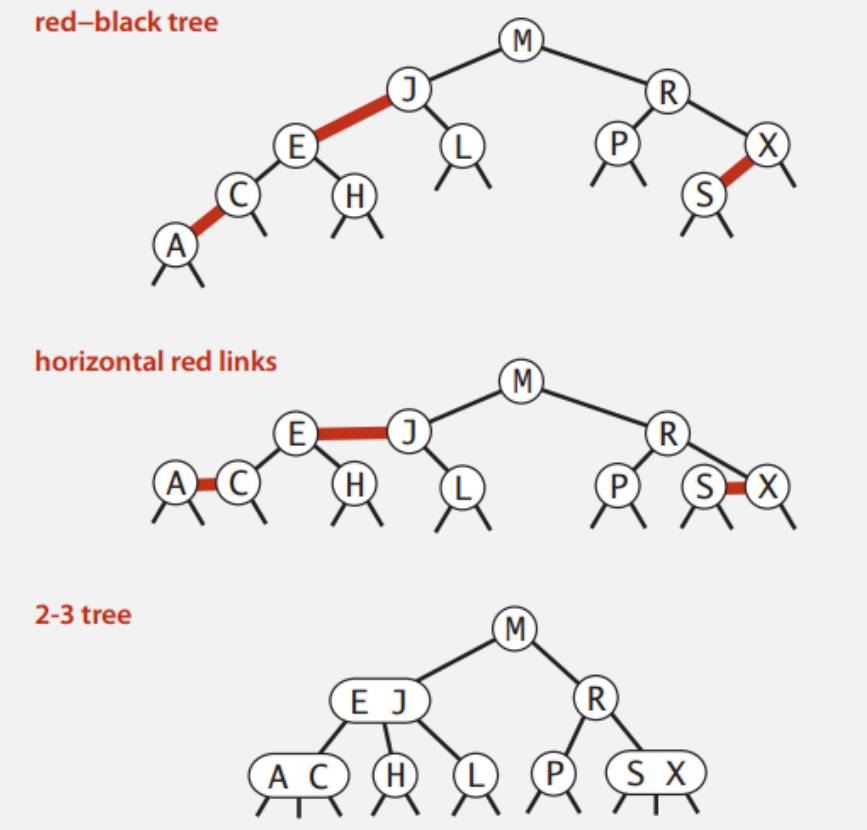


Exercise

- Convert to a red-black tree



2-3 Trees and red-black trees



Question: How do you convert a red-black tree to a 2-3 tree?

Red-black tree properties

- A BST such that
  - No node has two red links connected to it
  - Every path from root to null link has the same number of **black links**
  - Red links lean left.

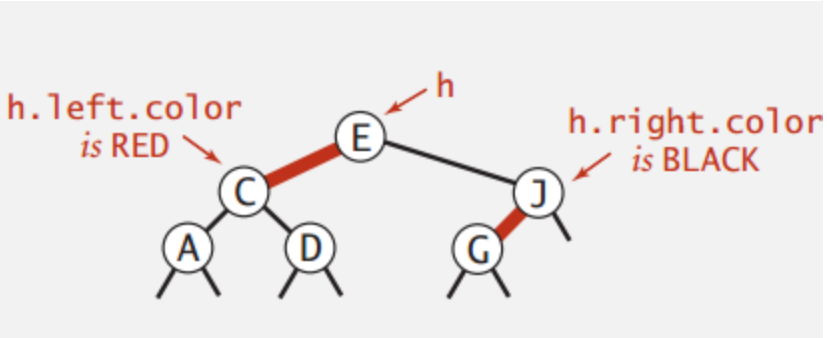
Red-black tree implementation

```
private static final boolean RED   = true;
private static final boolean BLACK = false;

private class Node
{
   Key key;
   Value val;
   Node left, right;
   boolean color;   // color of parent link
}

private boolean isRed(Node x)
{
   if (x == null) return false;
   return x.color == RED;
}
```

null links are black

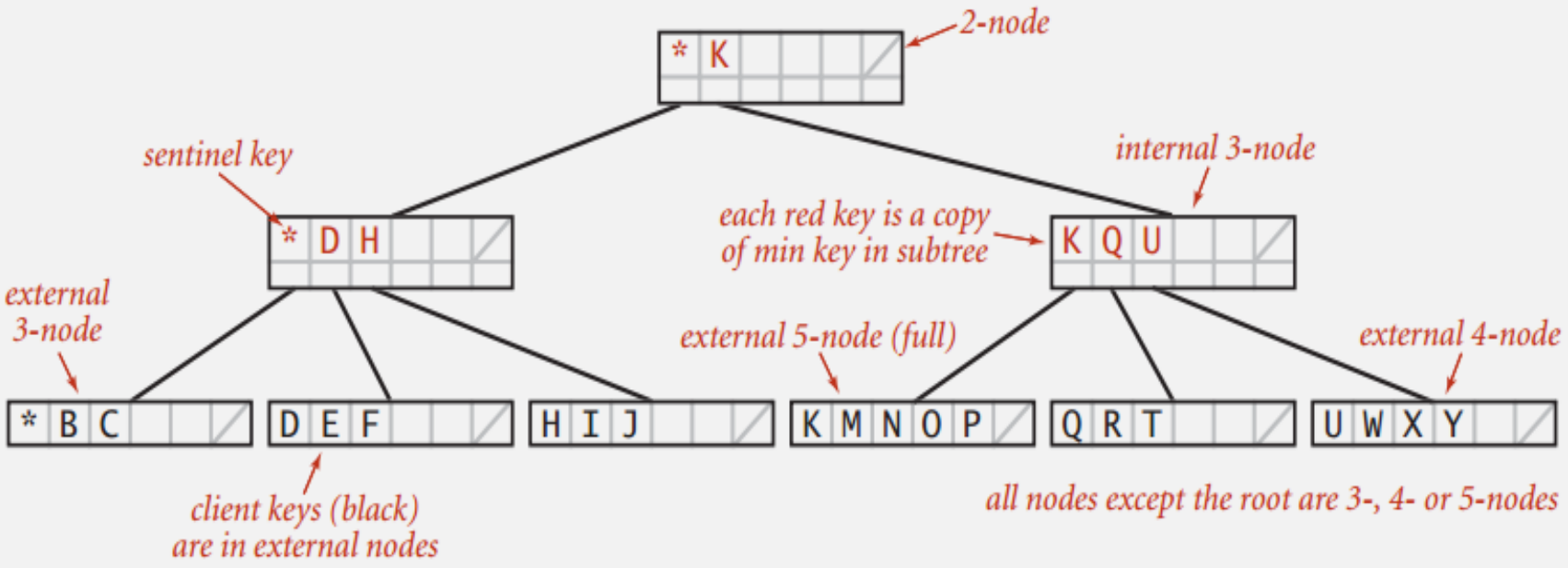


B-Trees

B-tree. Generalize 2-3 trees by allowing up to $M - 1$ key-link pairs per node.

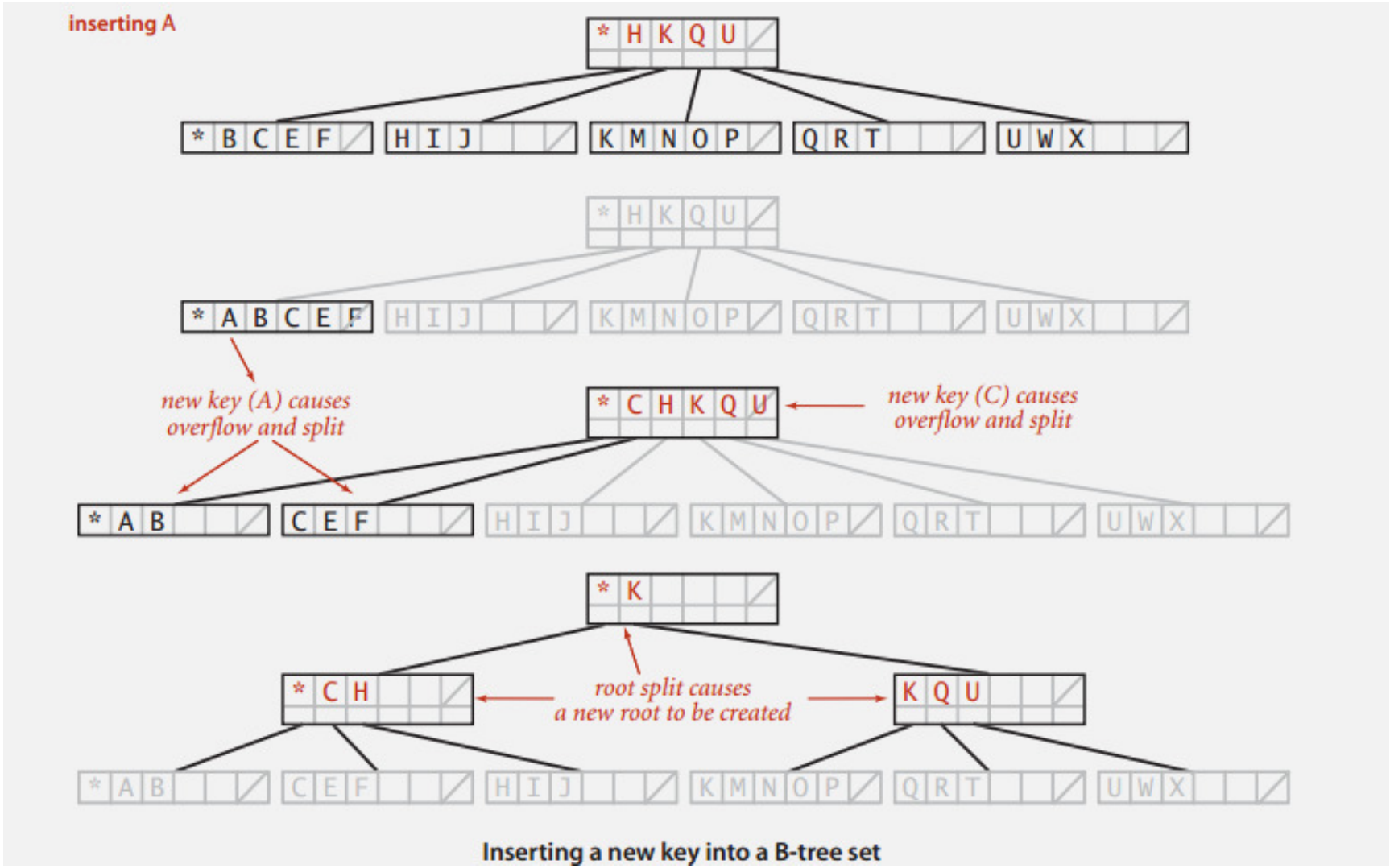
- At least 2 key-link pairs at root.
- At least $M / 2$ key-link pairs in other nodes.
- External nodes contain client keys.
- Internal nodes contain copies of keys to guide search.

choose M as large as possible so that M links fit in a page, e.g., M = 1024



Anatomy of a B-tree set (M = 6)

Inserting to a B-tree



Inserting a new key into a B-tree set

B-Tree Facts

Insertion in a B-tree

- Search for new key.
- Insert at bottom.
- Split nodes with M key-link pairs on the way up the tree.

Proposition. A search or an insertion in a B-tree of order M with N keys requires between $\log_{M-1} N$ and $\log_{M/2} N$ probes.

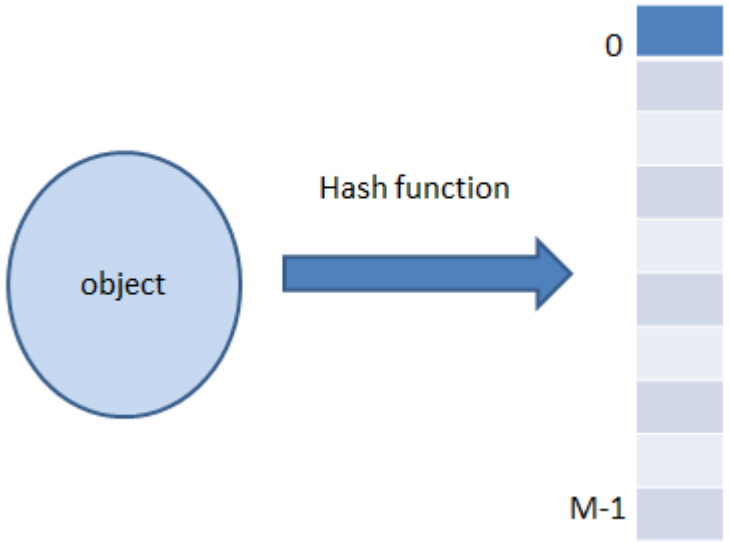
hashing

Hash Table

- A better way (almost constant time) to support
  - put and find operations
- Ideal goal
  - Distribute keys evenly in a table
  - Need a function (we call hash function) to compute a table index
  - The hash function must perform well for given keys

Hash Function

- Easy to compute
- Avoid collisions
- Maps the object to a table of size M



Finding a hash function

- What would be a good hash function for following key types?
  - Social Security number
  - Phone number
  - B’day
  - Strings
  - Integer
  - Double
  - Just Java objects

Java hashCode()

Requirement. If `x.equals(y)`, then `(x.hashCode() == y.hashCode())`.

Default implementation. Memory address of `x`.

Legal (but poor) implementation. Always return `17`.

Customized implementations. `Integer`, `Double`, `String`, `File`, `URL`, `Date`, ...

User-defined types. Users are on their own.

Java hash codes

```
public final class Integer
{
   private final int value;
   ...

   public int hashCode()
   {  return value;  }
}
```

```
public final class Boolean
{
   private final boolean value;
   ...

   public int hashCode()
   {
      if (value) return 1231;
      else       return 1237;
   }
}
```

```
public final class Double
{
   private final double value;
   ...

   public int hashCode()
   {
      long bits = doubleToLongBits(value);
      return (int) (bits ^ (bits >>> 32));
   }
}
```

convert to IEEE 64-bit representation;
xor most significant 32-bits
with least significant 32-bits

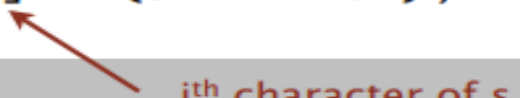
Warning: -0.0 and +0.0 have different hash codes

Java string hashCode()

Java library implementation

```
public final class String
{
   private final char[] s;
   ...

   public int hashCode()
   {
      int hash = 0;
      for (int i = 0; i < length(); i++)
         hash = s[i] + (31 * hash);
      return hash;
   }
}
```



Based on the formula

$$h = s[0] \cdot 31^{L-1} + \ldots + s[L-3] \cdot 31^2 + s[L-2] \cdot 31^1 + s[L-1] \cdot 31^0.$$

Modular Hashing

Hash code. An int between -2^{31} and $2^{31} - 1$.

Hash function. An int between 0 and M - 1 (for use as array index).

typically a prime or power of 2

```
private int hash(Key key)
{  return key.hashCode() % M;  }
```

bug

```
private int hash(Key key)
{  return Math.abs(key.hashCode()) % M;  }
```

1-in-a-billion bug

hashCode() of "polygenelubricants" is -2^{31}

```
private int hash(Key key)
{  return (key.hashCode() & 0x7fffffff) % M;  }
```

correct

Hash code design

Hash code design

"Standard" recipe for user-defined types.

- Combine each significant field using the $31x + y$ rule.
- If field is a primitive type, use wrapper type `hashCode()`.
- If field is `null`, return `0`.
- If field is a reference type, use `hashCode()`. ← applies rule recursively
- If field is an array, apply to each entry. ← or use `Arrays.deepHashCode()`

Uniform hashing

Bins and balls. Throw balls uniformly at random into M bins.

Birthday problem. Expect two balls in the same bin after $\sim \sqrt{\pi M / 2}$ tosses.

Coupon collector. Expect every bin has ≥ 1 ball after $\sim M \ln M$ tosses.

Load balancing. After M tosses, expect most loaded bin has $\Theta (\log M / \log \log M)$ balls.

Implementation

- A hash table of N keys can be implemented using
  - Separate Chaining
    - An array of M Linked Lists
      - Insert to beginning if the key is not present
      - Double when N/M > 8 or halve when N/M < 2
      - Rehash all keys when resizing
  - Linear Probing
    - An Array of size M >= N
      - Open addressing when resolving collisions
      - Linear probing – x, x+1, x+2 etc
      - rehash all keys when resizing.

Deleting in a hash table

- Separate chaining hash table

- Linear probing hash table

Applications of ST

- Sets
- Indexing
- Sparse vectors

Sets

public class SET<Key extends Comparable<Key>>	
SET()	*create an empty set*
void add(Key key)	*add the key to the set*
boolean contains(Key key)	*is the key in the set?*
void remove(Key key)	*remove the key from the set*
int size()	*return the number of keys in the set*
Iterator<Key> iterator()	*iterator through keys in the set*

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards

File Indexing

Query: given a string, find all files that contains that string

Solution. Key = query string; value = set of files containing that string.

```
import java.io.File;
public class FileIndex
{
   public static void main(String[] args)
   {
      ST<String, SET<File>> st = new ST<String, SET<File>>();

      for (String filename : args) {
         File file = new File(filename);
         In in = new In(file);
         while (!in.isEmpty())
         {
            String key = in.readString();
            if (!st.contains(key))
               st.put(word, new SET<File>());
            SET<File> set = st.get(key);
            set.add(file);
         }
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         StdOut.println(st.get(query));
      }
   }
}
```