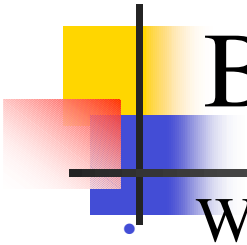
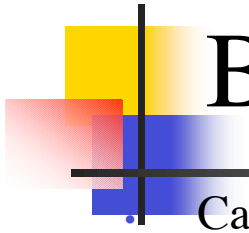
Efficient Representations of Maps and Sets

Corky Cartwright
Department of Computer Science
Rice University



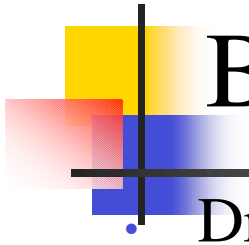
Binary Search Trees

- What is typical run-time cost of a get or put operation on a TreeMap implemented as a binary tree.
  - What is the formula for the number of nodes T(k) in a fully populated binary tree of depth k. Easy to prove by induction that
    $T(k) = 2^k - 1$
  - If we lookup a key in a tree with N keys that is reasonably balanced, the depth of the tree is a small constant times $\log_2 N$. Hence, searching this tree takes on the order of $\log_2 N$ steps. For a million keys, the average tree depth is a little greater than 20.
- What it the worst case? The binary tree degenerates into a list (a vine where every left/right subtree is empty. Then the cost of searching the tree is proportional to N (on average we have to look at half of the nodes). For a million keys, the tree depth is a million.



Balanced Binary Search Trees

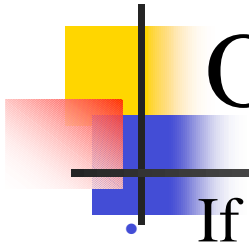
- Can we guarantee log N search behavior? Yes! If we ensure that the tree is balanced.
- Balancing schemes:
  - *Depth matching*: AVL trees. Two extra bits in each node are required to keep track of whether a tree is Heavy Right, Balanced, or Heavy Left (three codes). Every time we insert or delete a node, we must rebalance the tree. Every subtree is only allowed a difference of 1 in depth of right and left subtrees. The case analysis involved in rebalancing is rather ugly. The key idea is the the joints of the tree can be rotated to rebalanced. Think of nodes as rings and the links as cables connecting the rings. Rotating a subtree changes which ring of a subtree is considered the root.
  - *Equal path lengths with varied branching:* (B-trees, Red-Black Trees, 2-3-4 Trees, 2-3 Trees, …) Red-Black is now the most common form taught in Comp Sci courses. B-trees are widely used in industrial practice. Fundamental idea: every path from the root to a terminal node has the same length, but the branching factor varies. When trees grow, they ultimately add a new root! New nodes are first inserted at bottom of tree. In Red-Black and 2-3-4 trees, each branching joint has arity 2, 3, or 4. The trick in Red-Black trees is representing a 2-3-4 tree as a binary tree by inserting auxiliary binary branching nodes. The auxiliary nodes are red and always appear immediately below a black node that they augment. At most two red nodes can augment a black node (one as the left child and one as the right child.) Every path from the root to a terminal contains the same number of black nodes. See Okasaki's paper referenced in the course schedule.



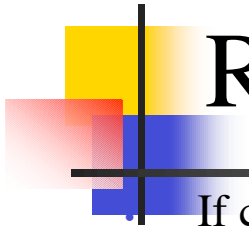
Binary Search Trees in Java

- DrJava includes data structures that are binary search trees. Does DrJava contain an implementation of binary search trees? No!
- The Java collections library (`java.util`) contains the classes `TreeMap` and `TreeSet` that are red-black tree implementations of the `SortedMap` and `SortedSet` interfaces, respectively.
- Comment: *OO programming encourages library use* because OO code organization forces the use of abstract interfaces. An object is a black box accessed via its members. Unless a bug attributable to library code arises, the DrJava developers never need to look at the code implementing those boxes. Even when a bug arises in a library, the usual recourse is to work around it rather than fixing it. Developers generally avoid maintaining their own versions of libraries developed by other organizations.

Other Binary Search Techniques



- If the set of items we need to search rarely changes, there are simpler O(log N) complexity searching schemes than binary trees. All that we need to do is use an array of entries (raw keys for sets, key-value pairs for maps) sorted by the key. Then we can perform binary search, where we repeatedly slice the array in half ensuring the the entry we are looking for is in the retained half. This process takes $O(\log_2 N)$ steps. The code is very simple to write, but you can write an infinite loop if you are not careful. You *must* make sure that each search iteration reduces the size of the array slice.
- Why use binary search instead of binary search trees.
- Question which takes more storage for a fixed size table sorted by key: a binary search array or a binary tree?

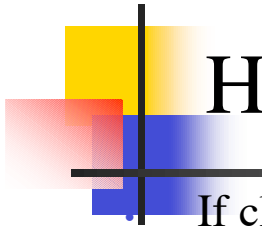


Really Fast Search Techniques

- If client only asks if specific keys are present in the table, there are better data structures and algorithms available than (binary) search trees or binary table search (of ordered arrays).
- Search trees and binary table search enable the client to quickly (O ($\log_2$ N)) perform inexact queries such as "find the smallest key greater than some threshold value". But …
- Constant time searching for a specific key is possible using a data structure called a hash table. There are two different forms of hash tables: *direct chaining* (informally called bucket hashing) and *open addressing* tables. The latter are generally covered in data structures courses but are rarely used in practice. In some contexts (with short entries and low table load factors), they have a modest space advantage but they behave badly under heavy load factors and do not support key deletion.
- A direct chaining hash table is an array of lists of entries (keys or key-value pairs) called buckets. Lists are typically implemented as linked lists but other representations are possible. Ideally each bucket should contain at most one item, but in the worst case, all of the entries in a table can theoretically fall into the same bucket. Every item is assigned to the bucket determined by its hash code.

  Object -> hashCode -> bucket index (last step typically a mod operation)

  usWithin a bucket, linear search (brute force) is used to find an entry.
- Choice of hash function is very important. Prime table sizes work well. Use empiricism.

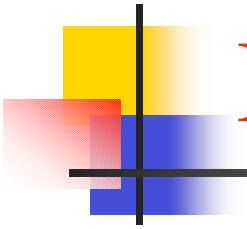


HW 11 Support: Rice Game Tree Environment

- If client only asks if specific keys are present in the table, there are better data structures and algorithms available than (binary) search trees or binary table search (of ordered arrays).
- Search trees and binary table search enable the client to quickly (O ($\log_2$ N)) perform inexact queries such as "find the smallest key greater than some threshold value". But …
- Constant time searching for a specific key is possible using a data structure called a hash table. There are two different forms of hash tables: *direct chaining* (informally called bucket hashing) and *open addressing* tables. The latter are generally covered in data structures courses but are rarely used in practice. In some contexts (with short entries and low table load factors), they have a modest space advantage but they behave badly under heavy load factors and do not support key deletion.
- A direct chaining hash table is an array of lists of entries (keys or key-value pairs) called buckets. Lists are typically implemented as linked lists but other representations are possible. Ideally each bucket should contain at most one item, but in the worst case, all of the entries in a table can theoretically fall into the same bucket. Every item is assigned to the bucket determined by its hash code.

  Object -> hashCode -> bucket index (last step typically a mod operation)

  usWithin a bucket, linear search (brute force) is used to find an entry.
- Choice of hash function is very important. Prime table sizes work well. Use empiricism.



For Next Class

- Exam over OO material will be distributed on Friday, April 10, and due by 11:59PM on Friday, April 17 in my office.
- Tic-tac-toe homework due Wednesday. Like the last assignment, you only have to write a modest part of the total application. Have fun.
- DrJava makes it easy to practice writing code fragments/exercises. Do it! Don't be afraid to experiment. The interactions pane makes it easy.