

Chapter 12 B

Advanced Implementations of Tables

Red-Black Trees

- A 2-3-4 tree
 - Advantages
 - It is balanced
 - Its insertion and deletion operations use only one pass from root to leaf
 - Disadvantage
 - Requires more storage than a binary search tree
- A red-black tree
 - A special binary search tree
 - Used to represent a 2-3-4 tree
 - Has the advantages of a 2-3-4 tree, without the storage overhead

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-2

Red-Black Trees

- Basic idea
 - Represent each 3-node and 4-node in a 2-3-4 tree as an equivalent binary tree
- Red and black children references
 - Used to distinguish between 2-nodes that appeared in the original 2-3-4 tree and 2-nodes that are generated from 3-nodes and 4-nodes
 - Black references are used for child references in the original 2-3-4 tree
 - Red references are used to link the 2-nodes that result from the split 3-nodes and 4-nodes

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-3

Red-Black Trees

Figure 12.31

Red-black
representation of a 4-
node

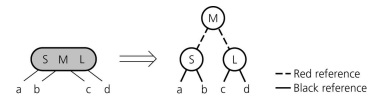
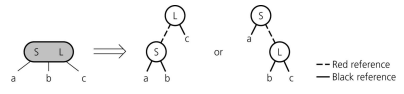


Figure 12.32

Red-black
representation of a 3-
node



© 2004 Pearson Addison-Wesley. All rights reserved

12 B-4

Red-Black Trees: Searching and Traversing a Red-Black Tree

- A red-black tree is a binary search tree
- The algorithms for a binary search tree can be used to search and traverse a red-black tree

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-5

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

- Insertion algorithm
 - The 2-3-4 insertion algorithm can be adjusted to accommodate the red-black representation
 - The process of splitting 4-nodes that are encountered during a search must be reformulated in terms of the red-black representation
 - In a red-black tree, splitting the equivalent of a 4-node requires only simple color changes
 - Rotation: a reference change that results in a shorter tree
- Deletion algorithm
 - Derived from the 2-3-4 deletion algorithm

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-6

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

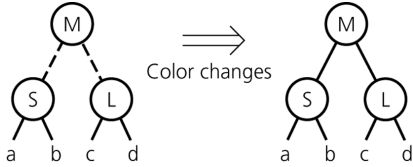


Figure 12.34
Splitting a red-black representation of a 4-node that is the root

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-7

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

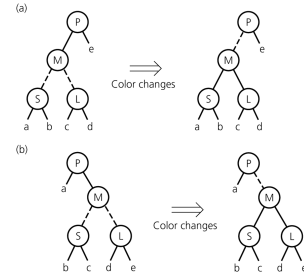


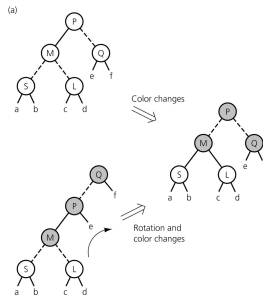
Figure 12.35
Splitting a red-black representation of a 4-node whose parent is a 2-node

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-8

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

Figure 12.36a
Splitting a red-black representation of a 4-node whose parent is a 3-node

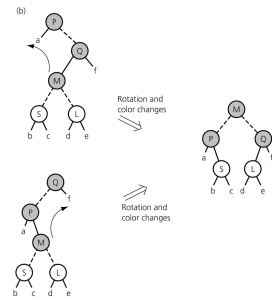


© 2004 Pearson Addison-Wesley. All rights reserved

12 B-9

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

Figure 12.36b
Splitting a red-black representation of a 4-node whose parent is a 3-node

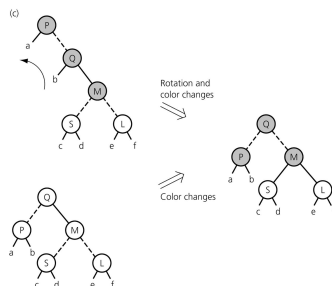


© 2004 Pearson Addison-Wesley. All rights reserved

12 B-10

Red-Black Trees: Inserting and Deleting From a Red-Black Tree

Figure 12.36c
Splitting a red-black representation of a 4-node whose parent is a 3-node



© 2004 Pearson Addison-Wesley. All rights reserved

12 B-11

AVL Trees

- An AVL tree
 - A balanced binary search tree
 - Can be searched almost as efficiently as a minimum-height binary search tree
 - Maintains a height close to the minimum
 - Requires far less work than would be necessary to keep the height exactly equal to the minimum
- Basic strategy of the AVL method
 - After each insertion or deletion
 - Check whether the tree is still balanced
 - If the tree is unbalanced, restore the balance

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-12

AVL Trees

- Rotations
 - Restore the balance of a tree
 - Two types
 - Single rotation
 - Double rotation

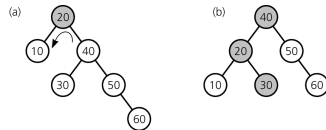


Figure 12.38
a) An unbalanced binary search tree; b) a balanced tree after a single left rotation

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-13

AVL Trees

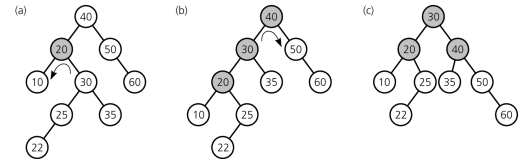


Figure 12.42
a) Before; b) during; and c) after a double rotation

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-14

AVL Trees

- Advantage
 - Height of an AVL tree with n nodes is always very close to the theoretical minimum
- Disadvantage
 - An AVL tree implementation of a table is more difficult than other implementations

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-15

Hashing

- Hashing
 - Enables access to table items in time that is relatively constant and independent of the items
- Hash function
 - Maps the search key of a table item into a location that will contain the item
- Hash table
 - An array that contains the table items, as assigned by a hash function

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-16

Hashing

- A perfect hash function
 - Maps each search key into a unique location of the hash table
 - Possible if all the search keys are known
- Collisions
 - Occur when the hash function maps more than one item into the same array location
- Collision-resolution schemes
 - Assign locations in the hash table to items with different search keys when the items are involved in a collision
- Requirements for a hash function
 - Be easy and fast to compute
 - Place items evenly throughout the hash table

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-17

Hash Functions

- It is sufficient for hash functions to operate on integers
- Simple hash functions that operate on positive integers
 - Selecting digits
 - Folding
 - Module arithmetic
- Converting a character string to an integer
 - If the search key is a character string, it can be converted into an integer before the hash function is applied

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-18

Resolving Collisions

- Two approaches to collision resolution
 - Approach 1: Open addressing
 - A category of collision resolution schemes that probe for an empty, or open, location in the hash table
 - The sequence of locations that are examined is the probe sequence
 - Linear probing
 - Searches the hash table sequentially, starting from the original location specified by the hash function
 - Possible problem
 - » Primary clustering

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-19

Resolving Collisions

- Approach 1: Open addressing (Continued)
 - Quadratic probing
 - Searches the hash table beginning with the original location that the hash function specifies and continues at increments of 1^2 , 2^2 , 3^2 , and so on
 - Possible problem
 - Secondary clustering
 - Double hashing
 - Uses two hash functions
 - Searches the hash table starting from the location that one hash function determines and considers every n^{th} location, where n is determined from a second hash function
- Increasing the size of the hash table
 - The hash function must be applied to every item in the old hash table before the item is placed into the new hash table

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-20

Resolving Collisions

- Approach 2: Restructuring the hash table
 - Changes the structure of the hash table so that it can accommodate more than one item in the same location
 - Buckets
 - Each location in the hash table is itself an array called a bucket
 - Separate chaining
 - Each hash table location is a linked list

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-21

The Efficiency of Hashing

- An analysis of the average-case efficiency of hashing involves the load factor
 - Load factor α
 - Ratio of the current number of items in the table to the maximum size of the array table
 - Measures how full a hash table is
 - Should not exceed $2/3$
 - Hashing efficiency for a particular search also depends on whether the search is successful
 - Unsuccessful searches generally require more time than successful searches

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-22

The Efficiency of Hashing

- Linear probing
 - Successful search: $\frac{1}{2}[1 + 1/(1-\alpha)]$
 - Unsuccessful search: $\frac{1}{2}[1 + 1/(1-\alpha)^2]$
- Quadratic probing and double hashing
 - Successful search: $-\log_e(1-\alpha)/\alpha$
 - Unsuccessful search: $1/(1-\alpha)$
- Separate chaining
 - Insertion is $O(1)$
 - Retrievals and deletions
 - Successful search: $1 + (\alpha/2)$
 - Unsuccessful search: α

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-23

The Efficiency of Hashing

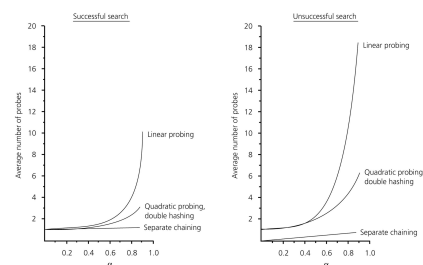


Figure 12.50

The relative efficiency of four collision-resolution methods

© 2004 Pearson Addison-Wesley. All rights reserved

12 B-24

What Constitutes a Good Hash Function?

- A good hash function should
 - Be easy and fast to compute
 - Scatter the data evenly throughout the hash table
- Issues to consider with regard to how evenly a hash function scatters the search keys
 - How well does the hash function scatter random data?
 - How well does the hash function scatter nonrandom data?
- General requirements of a hash function
 - The calculation of the hash function should involve the entire search key
 - If a hash function uses module arithmetic, the base should be prime

© 2004 Pearson Addison-Wesley. All rights reserved.

12 B-25

Table Traversal: An Inefficient Operation Under Hashing

- Hashing as an implementation of the ADT table
 - For many applications, hashing provides the most efficient implementation
 - Hashing is not efficient for
 - Traversal in sorted order
 - Finding the item with the smallest or largest value in its search key
 - Range query
- In external storage, you can simultaneously use
 - A hashing implementation of the `tableRetrieve` operation
 - A search-tree implementation of the ordered operations

© 2004 Pearson Addison-Wesley. All rights reserved.

12 B-26

Data With Multiple Organizations

- Many applications require a data organization that simultaneously supports several different data-management tasks
 - Several independent data structures do not support all operations efficiently
 - Interdependent data structures provide a better way to support a multiple organization of data

© 2004 Pearson Addison-Wesley. All rights reserved.

12 B-27

Summary

- A 2-3 tree and a 2-3-4 tree are variants of a binary search tree in which the balanced is easily maintained
- The insertion and deletion algorithms for a 2-3-4 tree are more efficient than the corresponding algorithms for a 2-3 tree
- A red-black tree is a binary tree representation of a 2-3-4 tree that requires less storage than a 2-3-4 tree
- An AVL tree is a binary search tree that is guaranteed to remain balanced
- Hashing as a table implementation calculates where the data item should be rather than search for it

© 2004 Pearson Addison-Wesley. All rights reserved.

12 B-28

Summary

- A hash function should be extremely easy to compute and should scatter the search keys evenly throughout the hash table
- A collision occurs when two different search keys hash into the same array location
- Hashing does not efficiently support operations that require the table items to be ordered
- Hashing as a table implementation is simpler and faster than balanced search tree implementations when table operations such as traversal are not important to a particular application
- Several independent organizations can be imposed on a given set of data

© 2004 Pearson Addison-Wesley. All rights reserved.

12 B-29