**Announcements**

- Don't go to a tutorial for COMP2160 on Thursday
- ANZAC Day is tomorrow
- Your quizzes will be returned next week
- Your assignment is due really soon — (I hope you started early...)
- Test cases supplied this week
- Code will be submitted with WebCT

---

**ANZAC Day**

- ANZAC Day - 25 April - is probably Australia's most important national occasion. It marks the anniversary of the first major military action fought by Australian and New Zealand forces during the First World War. ANZAC stands for Australian and New Zealand Army Corps. The soldiers in those forces quickly became known as ANZACS, and the pride they soon took in that name endures to this day.


---

**2-3-4 Trees**

- Rules for placing data items in the nodes of a 2-3-4 tree
  - A 2-node must contain a single data item whose search keys satisfy the relationships pictured in Figure 13.3a
  - A 3-node must contain two data items whose search keys satisfy the relationships pictured in Figure 13.3b
  - A 4-node must contain three data items whose search keys S, M, and L satisfy the relationship pictured in Figure 13.21
  - A leaf may contain either one, two, or three data items

![Figure 13.21](http://www.awm.gov.au)  
A 4-node in a 2-3-4 tree

---

**2-3-4 Trees: Searching and Traversing a 2-3-4 Tree**

- Search and traversal algorithms for a 2-3-4 tree are simple extensions of the corresponding algorithms for a 2-3 tree
2-3-4 Trees: Inserting into a 2-3-4 Tree

- The insertion algorithm for a 2-3-4 tree
  - Splits a node by moving one of its items up to its parent node
  - Splits 4-nodes as soon as it encounters them on the way down the tree from the root to a leaf
    - Result: when a 4-node is split and an item is moved up to the node’s parent, the parent cannot possibly be a 4-node and can accommodate another item

2-3-4 Trees: Splitting 4-nodes During Insertion

- A 4-node is split as soon as it is encountered during a search from the root to a leaf
- The 4-node that is split will
  - Be the root, or
  - Have a 2-node parent, or
  - Have a 3-node parent

2-3-4 Trees: Deleting from a 2-3-4 Tree

- The deletion algorithm for a 2-3-4 tree
  - Locate the node n that contains the item theItem
  - Find theItem’s inorder successor and swap it with theItem (deletion will always be at a leaf)
  - If that leaf is a 3-node or a 4-node, remove theItem
  - To ensure that theItem does not occur in a 2-node
    - Transform each 2-node encountered into a 3-node or a 4-node

2-3-4 Trees: Remarks

- Advantage of 2-3 and 2-3-4 trees
  - Easy-to-maintain balance
- Insertion and deletion algorithms for a 2-3-4 tree require fewer steps than those for a 2-3 tree
- Allowing nodes with more than four children is counterproductive
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
  - Can be used to represent a 2-3-4 tree
  - Has the advantages of a 2-3-4 tree, without the storage overhead

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
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 deletion algorithm

Figure 13.34
Splitting a red-black representation of a 4-node that is the root

Figure 13.35
Splitting a red-black representation of a 4-node whose parent is a 2-node

Figure 13.36a
Splitting a red-black representation of a 4-node whose parent is a 3-node

Figure 13.36b
Splitting a red-black representation of a 4-node whose parent is a 3-node

Figure 13.36c
Splitting a red-black representation of a 4-node whose parent is a 3-node
Other definitions

• Alternatively if you think of red-black trees without deriving them from 2-3-4 trees, there are corresponding definitions.

Properties

1. A node is either red or black
2. The root is black
3. All leaves are black (they are NULL)
4. Both children of every red node are black
   → red nodes can only have black parents; no red adjacent nodes
5. Every path from root to leaf has the same number of black nodes

These mean the longest root-leaf path can only be up to twice the shortest root-leaf path:
– the tree is roughly balanced (not "nearly balanced" as with AVLs, later)

This in turn means RB trees have good worst-case complexity (unlike BSTs)

Demo

• click here... fingers crossed...
• (and if that doesn’t work, try
   http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/RedBlackTree-Example.html)

AVL Trees

• Invented by G.M. Adelson-Velskii and E.M. Landis, "An algorithm for the organization of information" [1962]
• It is self-balancing: as such it is a self-organizing data structure
  – (can you think of any others?)
• It can be thought of as "nearly balanced" or in the context of AVL trees just "balanced".
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

The balance factor (bf) is a measure of how well balanced each node is:

\[ bf = (\text{height of left subtree}) - (\text{height of right subtree}) \]

(remember the height of a subtree is the length of the longest path from its root to any leaf)

• Nodes with bf = 0 or ±1 are "balanced" in this sense

Rotations

– Restore the balance of a tree
  – Two types
    • Single rotation
    • Double rotation

Advantage

– Height of an AVL tree with n nodes is always very close to the theoretical minimum (so it's good when you need lots of look-up)

Disadvantage

– An AVL tree implementation of a table is more difficult than other implementations (so it's a pain to program)
The ADT Priority Queue: A Variation of the ADT Table

• Pseudocode for the operations of the ADT priority queue

  createPQueue()
  // Creates an empty priority queue.

  pqIsEmpty()
  // Determines whether a priority queue is empty.

  pqInsert(newItem) throws PQueueException
  // Inserts newItem into a priority queue.
  // Throws PQueueException if priority queue is full.

  pqDelete()
  // Retrieves and then deletes the item in a priority queue with the highest priority value.

The ADT Priority Queue: A Variation of the ADT Table

• Pseudocode for the operations of the ADT priority queue (Continued)

  pqInsert(newItem) throws PQueueException
  // Inserts newItem into a priority queue.
  // Throws PQueueException if priority queue is full.

  pqDelete()
  // Retrieves and then deletes the item in a priority queue with the highest priority value.

The ADT Priority Queue: A Variation of the ADT Table

• Possible implementations
  – Sorted linear implementations
    • Appropriate if the number of items in the priority queue is small
  – Array-based implementation
    • Maintains the items sorted in ascending order of priority value
  – Reference-based implementation
    • Maintains the items sorted in descending order of priority value

The ADT Priority Queue: A Variation of the ADT Table

• Possible implementations (Continued)
  – Binary search tree implementation
    • Appropriate for any priority queue

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 (ha!)
• 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