CS F211 Reading and Exercises

Topics : **Comparison of Sorted Arrays and Hashtables. Ordered Dictionaries - Incremental/On-line Data Structures. Search Trees - Binary Search Trees. Binary Trees - Definitions, Operations and Implementation. Binary Search Trees - Definitions, Implementation, Analysis. Binary Search Tree - Height, Balancing of height. Height Balanced Binary Search Trees - AVL Trees. AVL Trees - Definition, Rotations, Insertion, Deletion, Analysis, Implementation issues and Pragmatics.**

Reading :

Topic	Reference	Remarks
Ordered Dictionaries and Binary Search Trees	T1 Sec 3.1	Required
AVL Trees	T1 Sec 3.2	Required

Review Questions

1. When is a sorted array better than a hashtable?
2. Mention two limitations of hashtables.
3. What is the cost of finding the minimum element of a list stored in a hashtable?
4. What issues arise in resizing hashtables?
5. Write down the representation (i.e. type definitions) for a binary tree in C.
6. Define the height of a tree.
7. What is the time complexity of a find operation in a binary search tree?
8. What is the time complexity of an insert operation in a binary search tree?
9. Where does an insert happen in a binary search tree?
10. What is the worst case height of a binary search tree of N nodes? Show an example.
11. What is the best case height of a binary search tree of N nodes? Show an example.
12. What is a height-balanced tree?
13. What is the intent of a rotation operation in an AVL tree?
14. How many rotations, in the worst case, are required for an insert operation in an

AVL tree?

15. How many rotations, in the worst case, are required for a delete operation in an AVL tree?
16. What are the worst case time complexities of find, insert, and delete operations in an AVL tree?
17. How is the height (balance) information maintained in an AVL tree?
18. What is Locality of Reference?
19. What is a Memory Hierarchy?
20. How does a rotate operation impact locality in a binary tree?
21. Why does a write operation impact the performance of the memory hierarchy?
22. Why does a delete operation in an AVL tree heavily impact the performance of the memory hieararchy?
23. What is a splay tree? Is it a height-balanced tree?
24. How is Locality of Reference leveraged in a splay tree?
25. What are the heuristics used in a splay tree?
26. What are the basic tree transformation operations used in a splay tree?
27. When does a splay tree give a better (expected) performance than an AVL tree?
28. What is the worst case time complexity of a find operation in a splay tree?

Problems / Exercises

LEGEND (for annotations on exercises):

A - Analytical	**D - Design**	**P - Programming**	**E - Experimental**
m - Math required	**r - Reading**	*** - hard**	**W - Writing**

1. **[P]**Implement the operations on a binary tree: getLeft, getRight, getRootVal, compose (i.e. makeBinTree), and isEmpty.

2.
   a. **[P]**Implement the find and insert operations on a binary search trees as recursive procedures.
   b. **[P]** Convert the above procedures to iterative versions.

c. **[P]** Implement the delete operation on a binary search tree.
d. **[P]** Implement the in-order traversal algorithm on a binary tree - in recursive and iterative forms.

3. **[A]** Identify inputs for which a sequence of insertions result in a skewed binary search tree i.e. height is close to N, the number of nodes. Provide a general characterization of such inputs and a comparative description of this behavior with that of QuickSort.

4. **[P]** Use in-order traversal to:

a. Write a procedure to find the minimum element in a BST.
b. Write a procedure to find the maximum element in a BST
c. Write a procedure to find the second smallest element in a BST.
d. Write a procedure to find the k^{th} smallest element in a BST.
e. Write a procedure to find the element closest to a given element in a BST.

5. **[A]**

a. Argue the correctness of the rotation procedure: i.e. it preserves the " (binary) search" property.
b. Argue that at most one invocation of the rotation procedure is enough to height-balance the tree after an insertion, assuming the tree was height-balanced to begin with.
c. Argue that one invocation of the rotation procedure may not be enough to height-balance the tree after a deletion, assuming the tree was height-balanced to begin with. Provide a worst-case example where the number of rotations required is the maximum.

6. **[P]**

a. Create a suitable representation for a node in an AVL tree.

b. Implement the restructure (i.e. rotate) operation in an AVL tree, given nodes X, Y, and Z:
   - Precondition: Y is parent of X and Z is parent of Y
   - Steps:
     i. Let a,b, and c be X, Y, and Z in left-to-right (i.e. sorted) order
     ii. let T0, T1, T2, and T3 in left-to-right order the other sub-trees of X,Y, and Z (i.e. sub trees not rooted at X, and Y).
     iii. let b replace Z.

iv. let a be the left child of b; let T0 and T1 be left and right subtrees of a.
v. let c be the right child of a; let T2 and T3 be left and right subtrees of c.

- Postcondition: The tree is height-balanced and retains the search property.

c. Implement the insert operation in an AVL tree. This should consist of the following steps:
   i. find the point of insertion
   ii. find Z, the first unbalanced node in the path from the point of insertion to the root
   iii. identify Y as the child of Z, and X as the child of Y, both in the same path (mentioned in step (ii))
   iv. call rotate(X,Y,Z).

d. Implement the delete operation in an AVL tree. This should consist of the following steps:

   i. find the node with the given key and delete it (from its parent) and free the node.
   ii. start at the parent of the delete node
   iii. find Z, the first unbalanced node in the path to the root
   iv. identify Y as the child of Z with larger height, and X as the child of Y with larger height.
   v. call rotate(X,Y,Z).
   vi. start with (restructured) Z (i.e. b in rotation procedure) and repeat steps (iii) to (v) until there are no more unbalanced nodes.

e. **[P,E]**
   a. Compare the performance of Simple BSTs, and AVL trees by running them on several input sets. Generate different kinds of input values: random input values and already sorted input values (at the time of insertion), nearly sorted input values, and reverse sorted input values. Generate different mixes and sequences of queries: a high percentages of finds vs. insertions/deletions, roughly same number of insertions/deletions vs. finds; all (or most) insertions at the beginning and finds arrive later, all (or most) deletes are arrive in one cluster, insertions/deletions are frequently interleaved with finds, all operations occur in clusters.
   b. Instrument your code for AVL trees to keep count of the number of tree

transformation operations - i.e. number of rotations in AVL trees.

c. Re-do the comparison in (a) by assuming that operations are done offline (i.e. the input list is given before any element is inserted into the binary search tree and find operations are performed after all insertions are done) but choose the root of each subtree: (i) randomly (ii) by using median-of-medians. Measure the rotations in AVL tree for each case.

f. **[rP,A,E,W]**
   a. Read about at least one other technique for height-balancing search trees and implement that technique.
   b. Compare the performance of the technique you learnt with that used in AVL trees -analytically and experimentally.
   c. Write a brief report on the comparison.