**FINAL EXAM (with ANSWERS)**
INSTRUCTIONS: Please answer all questions. This is a closed book exam. You may refer to two sheets of 8"×11" notes, plus appendix A of textbook.

**Problem 1** (AVL Trees) [10+10 points]

Let $T$ be the AVL tree in figure 1. (a) Suppose you insert key 8 into $T$. Show the

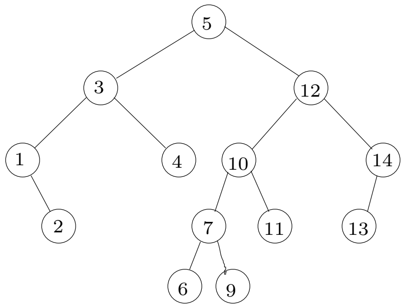


Figure 1:

resulting AVL tree
(b) Suppose you delete key 1 from $T$. Show the resulting AVL tree.

ANSWER:
(a) INSERT 8 as usual. The unbalance is at node Now rotate(10).
(b) DELETE 1 as usual. Now rot(rot(10)).

**Problem 2** (Probability) [10+15 points]

(a) Multiple Choice Question. If you pick a correct choice, you get 10 points; if wrongly, you get -5 points; if you do not pick no choice or more than one choice, you get 0 points. If $X$ and $Y$ are random variables, then

**(i)** $\mathtt{E}[X+Y] = \mathtt{E}[X] + \mathtt{E}[Y]$

**(ii)** $\mathtt{E}[X+Y] = \mathtt{E}[X] + \mathtt{E}[Y] - \mathtt{E}[XY]$

**(iii)** $\mathtt{E}[X+Y] < \mathtt{E}[X] + \mathtt{E}[Y]$

**(iv)** $\mathtt{E}[X+Y] < \mathtt{E}[X] + \mathtt{E}[Y] - \mathtt{E}[XY]$

(b) Here is a dice game. You toss two dice and your reward is the product of the two dice rolls. E.g. if you get a 4 and a 2, then you get 8 dollars. So you can win between 1 and 36 dollars per game. It costs you 20 dollars to play the game. Compute your expected win (or loss) per game. Please show your working to get full credit.

ANSWER:
(a) Correct choice is (i).
(b) Let $X$ be the value of the first die, $Y$ be the value of the second die. Since they are independent, $\mathtt{E}[XY] = \mathtt{E}[X]\mathtt{E}[Y]$. Since $\mathtt{E}[X] = 7/2$, $\mathtt{E}[XY] = 49/4 = 12.25$. [You can also do this (slowly) by computing the probability of all possible values between 1 and 36.]

**Problem 3** (Hashing) [10+20 Points]

(a) Suppose each key is a string of length at most 30. The symbols in a string are capital letters (A-Z) or spaces. Jack Quick pointed out that we can view any key $k$ as an integer $h(k)$ written in base 27 notation. E.g. h("A")=1, h("Z")=26, h(" ")= 0, h("BC")=(2*27)+3=57, h("A B")=(1*27*27)+2=678, etc. What is wrong with Jack's suggestion to use the function $h$ as a hash function? Say how you will choose a hash function (assume the chaining method is used for collision resolution).

(b) Compare linear probing with quadratic probing. (i) Give a reason why linear probing is better than quadratic probing. (ii) Give a reason why quadratic probing is better than linear probing.

ANSWER:

(a) The problem with Jack's idea is that the hash values can be as large as $27^{30}$, and we need a table at least that size. For any string $w$, let $J(w)$ be the integer value proposed by Jack. If I have an estimate $n$ on the number of keys to be hashed, then I would pick a prime number $p$ just larger than $n$. Then I will use the hash function $h(w) = J(w) \bmod p$.

(b) (i) Linear probing is simpler to implement, and it is also guaranteed to cycle through all the empty slots. (ii) Quadratic avoids primary clustering which can greatly slow down the hashing operations of LookUp, Insert, Delete.

**Problem 4** (Graph Traversal) [10+10+10+10+15+15 Points]

Consider traversing the following undirected graph $G$ starting at node 1.

```
1 --- 2 --- 3
|   / |   /
| /   | /
4 --- 5
```

(a) Give a BFS tree for $G$ (be sure to include all the the cross edges).

(b) Give two different DFS trees for $G$ (be sure to include all the back edges).

(c) What is the underlying data structure (ADT) used by (i) the BFS algorithm, and (ii) the DFS algorithm, to search a graph? [NOTE: we are not talking about the graph representation here.]

(d) Use a proof-by-contradiction argument to show: *the DFS tree for a connected undirected graph is spanning.* You may assume the output is a tree.

(e) Give a brief proof of this fact: *in any DFS of a connected, undirected graph, the number of back edges is always the same.* You must show where the "connectedness" hypothesis is used. You may use any facts you know about trees.

ANSWER:

(a) Assume we start at node 1 in our BFS. The cross edges are indicated by double lines below:

```
     1
   /   \
 2 ===== 4
/ \    //
```

```
 /   \  //
3===== 5
```

(b)

```
1 ----> 2 ---->3
^^    >> ^^    /
||  //  !!   /
|| //   !!  /
4 <---- 5 <

1 ----> 2 ---> 3
^^    >> |    >>
||  //  |   //
|| //   V  //
4 <---- 5
```

(c) (i) BFS algorithm uses a FIFO Queue while (ii) DFS algorithm uses a Stack (or LIFO Queue).

(d) Proof. Suppose there is a node $v$ that is not in the DFS tree. Let $r$ be the root of the DFS tree. Take any path $(u_0, u_1, \ldots, u_k)$ in the graph from $r$ to $v$ (we can do this because the graph is assumed to be connected). Thus, $u_0 = r$ and $u_k = v$. We claim by induction on $i = 0, 1, \ldots, k$ that every node $u_i$ on this path is visited. In particular, this means $u_k = v$ is visited, a contradiction. It is clear from the DFS algorithm that $u_0$ is visited. Suppose $u_{i-1}$ is visited, then it is also clear from the DFS algorithm that $u_i$ would be visited. This is because when we visit $u_{i-1}$, we check all the neighbors of $u_{i-1}$ to see if they are visited. If not, we would have visited it. In particular, $u_i$ would have been visited. This completes the proof.

(e) Since the graph is connected, previous question shows that the tree is a spanning tree. Such a tree has n-1 vertices (n is the total number of vertices). All non-tree edges must be a back edge. There are $|E| - n + 1$ of these, and this number does not depend on any particular properties of the DFS tree.

**Problem 5** (Sorting) [5+5 Points. -3 points if wrong, 0 points if omitted.]

(a) (True/False) If quicksort makes poor choices for the "pivot" elements, then it can be slower in big-Oh speed than mergesort.

(b) (True/False) Although bucket sort can realize some efficiencies from restrictions on the range of keys to be sorted, it can never do better than the big-Omega(n log n) lower bound on comparison based sorting that has been proven.

ANSWER:

(a) True. [Note: it can take $\Omega(n^2)$ time in the worst case.]

(b) False. [Note: it does not fall under the comparison-based model because it is not restricted to only comparing two keys in its decision making steps.]

(c) True. [Note: this is because $n \lg n$ is faster than $n^2$ in the big-Oh sense.]

**Problem 6** (Analysis) [15 points]

Consider the following pseudo-code:

```
for i := 1 to n do begin
      DOTHIS(i);
      j := 1;
      while (j < i) do begin
            j := 2 * j;
            DOTHAT(j);
      end;
end;
```

Suppose DOTHIS($i$) takes $O(\log i)$ time and DOTHAT($j$) takes $O(j)$ time. Determine the running time of this code.

ANSWER: The DOTHIS($i$) operation is executed $n$ times, for a total of $\Theta(n \log n)$ time. For each $i$, the DOTHAT($j$) operation in the WHILE loop is executed $< \ln i$. The total cose for the WHILE loop is $\sum_{k=1}^{\ln i} 2^k = \Theta(i)$. Summing over all $i$, the total cost is $\Theta(n^2)$. Thus the whole algorithm is $\Theta(n^2)$.

**Problem 7** (Binary Search Tree) [20 points]

The **rank** of a key $k$ in a set $S$ is the number of keys in $S$ that are less than or equal to $k$. We wish to modify the binary search tree (BST) structure and algorithms so that for any key $k$, you can determine its rank among the keys in the tree. Briefly describe (a) how to modify the BST structure, (b) your (new) rank algorithm, modifications to the usual (c) insertion and (d) deletion algorithms. Assume that keys are unique. All the algorithm must run in time $\mathcal{O}(h)$ where $h$ is the height of the tree.

ANSWER:
(a) At each node, we now store the SIZE of that node. The SIZE of a node is defined to be the number of nodes in the subtree rooted at that node. So SIZE of a node is 1 if and only if the node is a leaf.
(b) The rank algorithm is as follows: suppose we want to find the rank of a key $k$. Basically, we do the usual LookUp (or Search) on $k$. We initialize a variable, CurrentRank, to 0. At each node $u$ that we visit, if $u.KEY \leq k$, then we increase CurrentRank by $1 + SIZE(u.LEFT)$. When our search finally stops, we return CurrentRank.
(c) After we insert in the usual way, we must retrace our step up the path from the inserted node. For each node we encounter on the way up, we increment its SIZE.
(d) After we delete in the usual way, if $u$ is the node that we physically removed, we must retrace our step up the path from $u$. For each node we encounter on the way up, we decrement its SIZE.