Coding+Theory 3: CS2233

28th September, 2023

Instructions: Strictly follow the input and output format for each problem. **Your code should run in an infinite loop expecting for query number**.

Input format

- First line will contain n, which indicates the number of elements to be inserted into the tree.
- Second line will contain n one-space-separated integers, which are your tree elements. Assume this to be an array of n integers.
- Queries - Each line contains 2 integers separated by space character. First integer for type of query. 1 for search. 2 for insert. 3 for delete. 4 for displaying the tree in level order. For example.,
- See below for queries:
  - 1 112 $\implies$ search for 112 in the tree. Output - "112 present" / "112 not present" if found / not-found
  - 2 100 $\implies$ insert 100 into the tree. If the element 100 is already present, then print(100 already present. So no need to insert). Otherwise, it should be inserted and output print(100 inserted).
  - 3 100 $\implies$ delete 100 if present and output print(100 deleted). Otherwise, print(100 not present. So it can not be deleted).
  - 4 $\implies$ Print the tree level by level. Each level must be printed in each new line.

Example:
Input and Output:
17
9 8 5 4 99 78 31 34 89 90 21 23 45 77 88 112 32
1 56
(Output for this query. From now on, here, blue text represents the expected output)56 not present
2 21

21 already present. So no need to insert.
2 56
56 inserted
2 90
90 already present.So no need to insert
3 51
51 not present. So it can not be deleted
1 32
32 not present
3 32
32 not present. So it can not be deleted
4
" display the elements in level order"
Note:- Code should run in infinite loop expecting the query.

Max Marks 60

1 Coding Problems:

1. Construct an AVL tree.

   The AVL tree should be constructed by calling the `insert (root, key)` listed below.

   Each node of the tree should use the following `struct` data type:

```
struct node
{
int data;
struct node *left;
struct node *right;
};
```

   You can assume that you have stored the pointer to the `root` node. Please, write the functions for:

   (a) `search (root, key)` – this function takes the pointer to the `root` node, and `key` as input, and returns the pointer to the node where `key` is present. If `key` is not present in the AVL tree, then the code should output an error message.

   (b) `insert (root, key)` – this function takes the pointer to the `root` node, and key as input, and inserts the node at the appropriate position.

   (c) `delete (root, key)` – this function takes the pointer to the `root` node, and the key as input, and deletes the corresponding node.

(d) In points b, and c, the output should be the tree obtained after node insertion/deletion. Please output the tree by printing the nodes level-by-level.

2+3+3+2=10 Marks

2. Construct an `red-black` tree.

   The `red-black` tree should be constructed by calling the `insert (root, key)` listed below.

   Each node of the tree should use the following `struct` data type:

```
struct node
{
int data;
bool color; /* stores the color of the current node */
struct node *left;
struct node *right;
};
```

   You can assume that you have stored the pointer to the `root` node. Please, write the functions for:

   (a) `search (root, key)` – this function takes the pointer to the `root` node, and `key` as input, and returns the pointer to the node where `key` is present. If `key` is not present in the `red-black` tree, then the code should output an error message.

   (b) `insert (root, key)` – this function takes the pointer to the `root` node, and key as input, and inserts the node at the appropriate position.

   (c) `delete (root, key)` – this function takes the pointer to the `root` node, and the key as input, and deletes the corresponding node.

   (d) In points b, and c, the output should be the tree obtained after node insertion/deletion. Please output the tree by printing the nodes level-by-level.

   2+3+3+2=10 Marks

3. Construct an `(2-4)`-tree.

   The `(2-4)`-tree should be constructed by calling the `insert (root, key)` listed below. Each node of the tree should use the following `struct` data type:

```
struct node
{
int data[3];
```

```
struct node *childern[4];
};
```

You can assume that you have stored the pointer to the `root` node. Please, write the functions for:

(a) `search (root, key)` – this function takes the pointer to the `root` node, and `key` as input, and returns the pointer to the node where `key` is present. If `key` is not present in the `(2-4)`-tree, then the code should output an error message.

(b) `insert (root, key)` – this function takes the pointer to the `root` node, and key as input, and inserts the node at the appropriate position.

(c) `delete (root, key)` – this function takes the pointer to the `root` node, and the key as input, and deletes the corresponding node.

(d) In points *b*, and *c*, the output should be the tree obtained after node insertion/deletion. Please output the tree by printing the nodes level-by-level.

2+3+3+2=10 Marks

4. Construct an `B`-tree.

The `B`-tree should be constructed by calling the `insert (root, key)` listed below. Each node of the tree should use the following `struct` data type:

```
struct BTreeNode
{
int key[MAX + 1], count;
/* count stores the number of keys in the current node */
struct BTreeNode *children[MAX + 1];
};
```

Additionally, there are two variables `MAX` and `MIN` denoting the minimum and maximum number of elements in a node. You can assume that you have stored the pointer to the `root` node. Please, write the functions for:

(a) `search (root, key)` – this function takes the pointer to the `root` node, and `key` as input, and returns the pointer to the node where `key` is present. If `key` is not present in the `B`-tree, then the code should output an error message.

(b) `insert (root, key)` – this function takes the pointer to the `root` node, and key as input, and inserts the node at the appropriate position.

(c) `delete (root, key)` – this function takes the pointer to the `root` node, and the key as input, and deletes the corresponding node.

(d) In points b, and c, the output should be the tree obtained after node insertion/deletion. Please output the tree by printing the nodes level-by-level.

2+3+3+2=10 Marks

5. Consider the problem of augmenting `red-black`-trees with an operation $RB-ENUMERATE(x, a, b)$ that output all keys k such that $a \leq k \leq b$ in a `red-black` tree rooted x. Write an algorithm that implements $RB-ENUMERATE(x, a, b)$ in $\Theta(m+\log n)$ time, where m is the number of keys that are outputted, and n is the number of internal nodes in the tree.

   Also, give a write to prove the correctness and efficiency of your algorithm.

   `Hint: See Theorem 14.1 of CLRS book.`

   7 (Coding) +3 (theory)=10 Marks

2 Theory questions:

1. Let us define a relaxed `red-black` tree as a binary search tree that satisfies the following properties. In other words, the `root` may be either red or black. Consider a relaxed `red-black` tree T whose root is `red`. If we color the `root` of T black but make no other changes to `T`, is the resulting tree a `red-black` tree?

   (a) Every node is either `red` or `black`.

   (b) Every `leaf` (NIL) is `black`.

   (c) If a node is `red`, then both its children are `black`.

   (d) For each node, all simple paths from the node to descendant leaves contain the same number of `black` nodes.

   5 Marks

2. Show that the longest simple path from a node x in a `red-black` tree to a descendant leaf has a length at most twice that of the shortest simple path from node x to a descendant leaf.

   Instructions on theory problems: Please neatly write the solution using pen-paper 1) to give a proof of the algorithm in Question 5 in Section 1, and 2) Questions 1 and 2 of this section; and submit the scan copy in the google classroom page.

   5 Marks