Coding+Theory 3: CS2233

18th October, 2024

Kindly adhere to the following instructions.

- Please write a C/C++ program corresponding to each problem. Your code should be well commented on, and variable names should be appropriately chosen.
- Create a folder and put all the code files and name all the files as the question number (i.e. 1.cpp/1.c), add the scanned copy of the theory assignment solution to it, give a name to the folder as "yourRollNo", zip the folder and submit it to the Google Classroom portal. Also, submit the hard copy of the theory assignment to the class.
- Strictly follow the input and output format for each problem.
- Any code that does not follow the input-output criteria won't be evaluated and will get **ZERO**.
- Your code will also be checked against plagiarism (both from web and peer).
- Any form of plagiarism (web/chatGPT/with peers) will be severely penalised and will result in an F grade.
- The submission (strict) timeline is 5th November, Tuesday, 2:30 PM.

Max Marks 45

1 Coding Problems:

1. 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, which is given as input. 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

Input format

- First line contains three space-separated integers n, MAX, MIN, which indicates the number of elements to be inserted into the tree, minimum and maximum number of elements in a node, respectively.
- Second line contains n one-space-separated integers, which are tree elements, where you should insert the elements according to the given order.
- Next line contains k, which indicates number of queries.
- Next k line contains two space-separated integers where the first integer denotes the query and the second integer denotes the input value corresponding to this query.
- Queries - Each line contains 2 integers, separated by space character.
  1 for **Search**.
  2 for **Insert**.
  3 for **Delete**.
  4 for **Displaying the tree in level order**.

- See below for example queries:

- 1 112 $\implies$ search for 112 in the tree. Output - "112 is present" / "112 is not present" if found / not-found
- 2 100 $\implies$ insert 100 into the tree. If the element 100 is already present, then print "100 is already present". Otherwise, it should be inserted and output print "100 is inserted".
- 3 100 $\implies$ delete 100 if present and output print "100 is deleted". Otherwise, print "100 is 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. Elements of the same node should be separated by ',' and a space should separate every node.

Example:
Input and Output:
17 3 1
9 8 5 4 99 78 31 34 89 90 21 23 45 77 88 112 32
8
1 56
56 is not present
(Output for this query. From now on, here, blue text represents the expected output)
2 21
21 is already present
2 56
56 is inserted
2 90
90 is already present
3 51
51 is not present. So it can not be deleted
1 32
32 is present
3 32
32 is deleted
4

```
9,34
5   23  77,89
4  8   21  31  45,56   78,88   90,99,112
```

2. Implement a red-black tree with the following operations:
   (a) Insertion of a node with a given value while maintaining the red-black tree properties.
   (b) Deletion of a node with a given value while maintaining the red-black tree properties.

(c) Find the k^{th} smallest element in the tree, where k is an integer provided as input (1-based indexing).

(d) Find the rank of a given value in the tree (i.e., how many elements are smaller than the given value).

$2.5 \times 4 = 10$ Marks

Input format

- First line contains n, which indicates the number of elements to be inserted into the tree.
- Second line contains n one-space-separated integers, which are tree elements, where you should insert the elements according to the given order.
- Next line contains k, which indicates number of queries.
- Next k line contains two space-separated integers where the first integer denotes the query and the second integer denotes the input value corresponding to this query.
- Queries - Each line contains 2 integers, separated by space character.
  1 for **Insert**.
  2 for **Delete**.
  3 for **Find the k^{th} smallest element**.
  4 for **Find the rank**.

- See below for example queries:
  - 1 100 $\implies$ insert 100 into the tree. If the element 100 is already present, then print "100 is already present. So no need to insert". Otherwise, it should be inserted and output print "100 is inserted".
  - 2 100 $\implies$ delete 100 if present and output print "100 is deleted". Otherwise, print "100 is not present. So it can not be deleted".
  - 3 5 $\implies$ Print the 5^{th} smallest node.
  - 4 100 $\implies$ Print the rank of node 100.

Example:
Input and Output:
17
9 8 5 4 99 78 31 34 89 90 21 23 45 77 88 112 32
4
1 56
56 is inserted
2 51
51 is not present. So it can not be deleted
3 5

21
4 23
5

3. 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. Add this to the theory assignment part of the assignment.

   `Hint: See Theorem 14.1 of CLRS book.`

   Input format

   - First line contains the n which indicates the number of elements in the tree.
   - Next line contains the n elements of the tree where you should insert the elements according to the given order.
   - Next line contains k which indicates the number of queries.
   - Next k lines contain k *enumerate* queries.

   Output format

   - Your output also should contain k lines, which indicates the output of corresponding k *enumerate* queries.
   - **Note:** Output the keys in sorted order as given in below example.

   Example:
   Input:

```
17
9 8 5 4 99 78 31 34 89 90 21 23 45 77 88 112 32
5
34 88
9 78
23 112
77 99
4 34
```

   Output:

```
34 45 77 78 88
9 21 23 31 32 34 45 77 78
23 31 32 34 45 77 78 88 89 90 99 112
77 78 88 89 90 99
4 5 8 9 21 23 31 32 34
```

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

2 Theory questions:

4. 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

5. 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. 5 Marks

6. Let T be a $(2, 4)$ tree. Consider the SPLIT operator that takes the root of a $(2, 4)$ tree T, and a node $x \in T$ as input, and creates a $(2, 4)$ tree T_1 consisting of all elements in $T - \{x\}$ whose keys are less than $key[x]$, and another $(2, 4)$ tree T_2 that consist of all elements in $T - \{x\}$ whose keys are greater than $key[x]$.

   Given a $O(\log n)$ time algorithm for SPLIT operator. Also, write a theorem which proves that your algorithm is correct (that is, it outputs the desired result) and its running time is $O(\log n)$. 2+3=5 Marks

   Instructions on theory problems: Please neatly write the solution using pen-paper (a) to give a proof of the algorithm in Question 3 in Section 1, and (b) Questions 4, 5, and 6 of this section; submit the scan copy in the google classroom page; submit the hard copy to the class.