National Institute of Technology Calicut
Department of Computer Science and Engineering
Fourth Semester B. Tech.(CSE)-Winter 2022-23
CS2094D Data Structures Laboratory
Assignment #2

Submission deadline (on or before): 17.02.2023, 2:00 PM

Policies for Submission and Evaluation:

- You must submit your assignment in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute without errors using gcc compiler.
- During the evaluation, failure to execute programs without compilation errors may lead to zero marks for that evaluation.
- Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. Detection of ANY malpractice related to the lab course can lead to awarding an F grade in the course.

Naming Conventions for Submission

- Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

`ASSG<NUMBER>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>.zip`

(Example: $ASSG2\_BxxyyyyCS\_CS01\_LAXMAN.zip$). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

- The source codes must be named as

`ASSG<NUMBER>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

(For example: $ASSG2\_BxxyyyyCS\_CS01\_LAXMAN\_1.c$). If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools, and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: https://minerva.nitc.ac.in/?q=node/650.

QUESTIONS

1. Our institute decided to split the students into four groups to conduct an event. The procedure for splitting is as follows:

   h(A) = (Sum of ASCII value of the characters in the first name of 'A' + age of 'A') % 4, where 'A' represents a student.

   Eg:- h(Veena) = (86+101+101+110+97+19) % 4 = 2; where, 86 - ASCII(V), 101 - ASCII(e), 110 - ASCII(n), 97 - ASCII(a), 19 - age(Veena).

   If h(A) =0, the student is placed in group 0; if h(A) = 1, the student is placed in group 1; if h(A) = 2, the student is placed in group 2, and if h(A) = 3, the student is placed in group 3.

   Write a program to perform the operations *count the number of students in each group*, *the student list in a group as per the order of insertion*, and *the student list who belongs to the same branch in a group.*

   Note: *Assume all the students are from CS, EC, EE, or CE branches.*

   Input format:

   - First line of the input contains an integer 'n' $\in [1, 10^3]$, the total number of students who are participating in the event.
   - Next 'n' consecutive line contains: *first_name*, *roll_number*, and *age*; separated by single space.
   - The input contains a character 'c' followed by an integer 'k' $\in [0, 3]$ to display the count and student list of the group 'k'.
   - The input contains an integer 'm' $\in [0, 3]$, representing the group number, followed by two characters representing the branch name (both uppercase and lowercase are considered the same branch).
   - The input contains a character 'e' to represent the end of the input.

   Output format:

   - The output (if any) of each command should be printed on a separate line.
   - For input lines starting with the character 'c' followed by an integer 'k' $\in [0, 3]$, prints an integer 'x' followed by x number of strings separated by a space.
   - For input lines starting with an integer 'm' $\in [0, 3]$ followed by two characters, prints the strings (first_name) separated by a space, if any student exit in the group 'm'. Otherwise, print -1.

   Sample Input:

   ```
   4
   Veena B220016EC 19
   Abu B210051CS 21
   Ishan B190016CE 22
   Aleena B200036EE 21
   c 0
   1 CS
   c 1
   c 2
   c 3
   3 EC
   2 ec
   e
   ```

   Sample Output:

   ```
   0
   ```

Abu
2 Abu Ishan
1 Veena
1 Aleena
-1
Veena

2. As part of MNTIP scholarship programme, first name, last name, gender, date_of_bith, department and CGPA of fourth semester students of NITC are gathered. The scholarship is for four semesters from fifth semester onwards. The record information is organized using a hash table with the help of a separate chaining technique. In separate chaining, the size of the hash table is 26, such that to insert a record into the hash table, ASCII value of the first character of first name starts with A points to index 0, ASCII value of the first character of first name starts with B points to index 1 and so on. The resulting location contains a pointer to a Binary Search Tree(BST), in this case the root of a binary search tree (BST), where all elements with the same index are stored. Since a BST requires that its nodes be comparable to each other, we need to define an order among the records being stored in the BST lexiographically. The key value of each node in BST is student's (firstname, lastname) pair, assuming it is unique to a student. The update function updates the CGPA of the corresponding key. The insert/update/delete function should return the number of nodes touched in the BST (except the current node) while performing the given operation. The location function returns index-bstsequence, where index represents the index of the key in the hash table, and bstsequence represents the sequence of L/R steps taken on the binary search tree of all records whose index is the same.

   Input Format:

   - For insertion
     - The first line of the input should be the character 'i' where 'i' denotes insertion.
     - The second line of the input is a string representing the first name followed by a string representing the last name followed by a single character either 'M' or 'F' representing the gender followed by a string in the format DD-MM-YYYY representing the date of birth followed by a string of length four representing the department and a floating point number representing the CGPA separated by single space in between them.
   - For updation
     - The first line of the input should be the character 'u' where 'u' denotes updation.
     - The second line of the input is a string representing the first name, a string representing the last name and a floating point number representing the updated CGPA separated by single space in between them.
   - For location and deletion
     - The first line of the input should be the character either 'l', or 'd' where 'l' denotes location and 'd' denotes deletion.
     - The second line of the input is a string representing the first name and a string representing the last name separated by single space in between them.

   Input **Output Format:**

   - The output (if any) of each command should be printed on a separate line.
   - The insert/update/delete function should return the number of nodes touched in the BST (except the current node) while performing the given operation.
   - The update/delete function returns –1, if key is not present.
   - The location function returns index-bstsequence, index represents the index of the key in the hash table, and bstsequence represents the sequence of L/R steps taken on the binary search tree of all records whose index is the same. The location function returns –1, if key is not present.

Sample Input:

i
Ram Kumar F 18-05-2022 CSED 6.3
i
Ram Charan F 19-07-2022 CSED 6.3
u
Ram Charan 6.9
u
Ram Kiran 7.3
l
Ram Charan
d
Ram Charan

Sample Output:

0
1
1
-1
17-L
1

3. Given two Arrays check whether they are similar or not using Hashing (Do not use sorting). The arrays are of same size(i.e n).
Note: Similar means both arrays contain same elements, with similar frequency(number of times the element is repeated) in it but the sequence of elements may be different.
Input Format:

- First line contain the size of the array i.e. value of n.
- Second and third line contains the elements of the first and second array respectively.

Output Format:

- Output will be binary value i.e. **1** - if arrays are similar, **0** - if arrays are different.

Sample Input 1:

6
1 2 1 3 2 1
2 2 3 1 1 1

Sample Output 1:

1

4. Open addressing is a method for handling collisions in hashing. The three different methods for open addressing are linear probing, quadratic probing, and double hashing. A brief description of the three methods is given below:
In linear probing, the function used to calculate the next location during collision is: $h'(k) = (h(k) + i) \mod m, i = 1, 2 \ldots$.
In quadratic probing, the function used to calculate the next location during collision is: $h'(k) = (h(k) + i^2) \mod m, i = 1, 2 \ldots$
In double hashing scheme, the primary hash function is, $h1(k) = k \mod N$, where N is the table size. The secondary hash function is, $h2(k) = R - (key \mod R)$ where R is the maximum prime number less than the table size. Double hashing can be done using: $(h1(key) + i * h2(key)) \mod N, i = 0, 1, 2 \ldots$.

Given a set of keys and the table size, write a program to print the locations at which the keys are stored using the above-mentioned three methods and also print the total number of collisions that occur during mapping for each of the three methods.

Input format:

- First line of the input contains an integer, the table size.
- Second line contains space-separated(single space) integer numbers, the keys to be inserted.

Output format:

- First line of the output contains space-separated (single space) integers, the locations obtained using linear probing.
- Second line contains an integer, the total number of collisions that occurred during linear probing.
- Third line of the output contains space-separated (single space) integers, the locations obtained using quadratic probing.
- Fourth line contains an integer, the total number of collisions that occurred during quadratic probing.
- Fifth line of the output contains space-separated (single space) integers, the locations obtained using double hashing.
- Sixth line contains an integer, the total number of collisions that occurred during double hashing.

Sample Input:

```
7
76 93 40 47 10 55
```

Sample Output:

```
6 2 5 0 3 1
4
6 2 5 0 3 1
6
6 2 5 1 3 4
2
```

5. Write a program to create an AVL Tree A and perform the operations *insertion*, *deletion*, *search* and *traversal*. Assume that the AVL Tree A does not contain duplicate values. Your program should contain the following functions.

   - Insert(A, k) – Inserts a new node with key 'k' into the tree A.
   - Search(A, k) - Searches for a node with key 'k' in A, and returns a pointer to the node with key k if one exists; otherwise, it returns NIL.
   - DeleteNode(A, k) – Deletes a node with the key 'k' from the tree A.
   - GetBalance(A, k) – Prints the balance factor of the node with 'k' as key in the tree A.
     Note:- Balance factor is an integer which is calculated for each node as:

     $$B\_factor = height(left\_subtree) - height(right\_subtree)$$

   - LeftRotate(A, k) – Perform left rotation in the tree A, with respect to the node with key 'k'.
   - RightRotate(A, k) – Perform right rotation in the tree A, with respect to node with key 'k'.

- PrintTree(A) – Prints the tree given by A in the parenthesis format as: (t (left-subtree)(right-subtree)),t represents root node of tree A. Empty parenthesis ( ) represents a null tree.

  Note: After each insertion on an AVL Tree, it may result in increasing the height of the tree. Similarly, after each deletion on an AVL Tree, it may result in decreasing the height of the tree. To maintain height balanced property of AVL tree, we may need to call rotation functions.
  Tree A should be an AVL Tree after INSERT AND DELETE NODE operations.

Input Format:

- Each line contains a character from '*i*', '*d*', '*s*', '*b*', '*p*' and '*e*' followed by at most one integer. The integers, if given, are in the range $[-10^6, 10^6]$.
- Character '*i*' is followed by an integer separated by space; a node with this integer as key is created and inserted into A.
- Character '*d*' is followed by an integer separated by space; the node with this integer as key is deleted from A and the deleted node's key is printed.
- Character *'s'* is followed by an integer separated by space; find the node with this integer as key in A.
- Character '*b*' is followed by an integer separated by space; find the balance factor of the node with this integer as key in A and the print the balance-factor.
- Character '*p*' is to print the Parenthesis Representation of the tree A.
- Character *'e'* is to 'exit' from the program.

Output Format:

- The output (if any) of each command should be printed on a separate line.
- For option '*d*', print the deleted node's key. If a node with the input key is not present in A, then print FALSE.
- For option '*s*', if the key is present in A, then print TRUE. If key is not present in A, then print FALSE.
- For option '*b*', if the key k is present in A, then print the balance factor of the node with k as key. If key is not present in A, then print FALSE.
- For option *'p'*, print the Parenthesis Representation of the tree A.

Sample Input:

```
i 4
i 6
i 3
i 2
i 1
s 2
p
b 4
d 3
p
e
```

Sample Output:

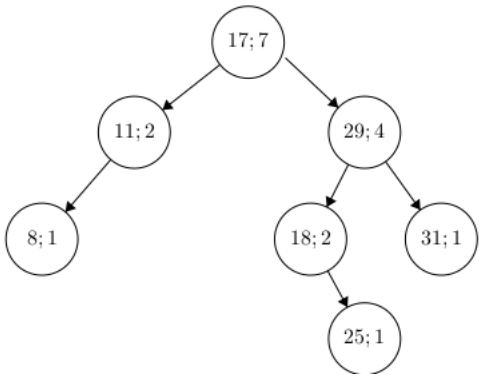
```
TRUE
( 4 ( 2 ( 1 ( ) ( ) ) ( 3 ( ) ( ) ) ) ( 6 ( ) ( ) ) )
1
3
( 4 ( 2 ( 1 ( ) ( ) ) ( ) ) ( 6 ( ) ( ) ) )
```

6. Write a program to compute the median element of the set of elements stored in the AVL tree. The median of a set of n numbers is the element that appears in the n/2 th position, when the set is written in sorted order. When n is even, n/2 and when n is odd, (n + 1)/2 is the position for the median element. For example, if the set is 3, 2, 1, 4, 6 then the set in sorted order is 1, 2, 3, 4, 6, and the median is 3.

   Consider a modified AVL tree in which each parent node stores both a key and the total no_of_nodes in the right and left subtree + 1 .

   ```
   struct node {
   int key ;
   int no_of_elements ;
   struct node  left;
   struct node  right ;
   }
   ```

   Here's an example of how the modified AVL tree would look (the number after a semicolon indicates the total no_of_elements in the right and left subtree + 1).



   As the number of nodes below a given node does not remain the same, the no_of_elements value at needs to be computed after each iteration. For example, after inserting 21 , the no_of_elements values at the root node increased and updated as 8, similarly at node '18' it's get decreased and updated as 1.

   Do not alter the structure of the tree (should use only one tree). Your program should include the following functions.
   •**getMedian(struct node* root):** returns median element of the AVL tree.

   Input Format:
   • Each line contains a character 'i' followed by an integer separated by a space; a node with this integer as key is created and inserted into the AVL.
   • Character 'g', is to get the median of the AVL tree.
   • Character 't' is to 'terminate' the program.

   Output Format:
   • Print median of the AVL tree.

   Constraints:
   1 <= n <= 1000

Sample Input 1:
i 17
i 11
i 29
i 8
i 18
i 31
i 25
g
t

Sample Output 1:
18

Sample Input 2:
i 37
i 21
i 80
i 81
g
t

Sample Output 2:
37

7. Find the number of nodes with value greater than a given positive integer K in a given AVL tree without using any extra space. Create a function countNodesGreaterThanK(currentNode, K) which will count the number of nodes with values greater than K in the tree rooted at the current node.
NOTE: A node with value K may or may not be present in the given tree.
Input Format:

- Each line contains a character 'i' followed by an integer separated by a space; a node with this integer as key is created and inserted into the AVL tree.
- Character 'c' followed by an integer 'K' separated by a space, is to count the nodes with value greater than K in the AVL tree.
- Character 'e' is to 'exit' the program.

Output Format:

- First line contains the number of nodes with value greater than the given element, K.
- Second line contains all the values greater than K, in sorted order, separated by space.

Constraints: $1 \leq$ node.value ≤ 1000
$1 \leq K <$ max(node.value), where max(node.value) is the maximum value present in the AVL tree.

Sample Input:
i 30
i 20
i 40
i 10
i 25
c 20
i 50
c 20

e

Sample Output:
3
25 30 40
4
25 30 40 50

8. A Red-Black tree is a self-balancing binary search tree where every node obeys the following rules.
   (a) Every node is either red or black
   (b) The root is always black
   (c) There are no two adjacent red nodes (A red node cannot have a red parent or red child)
   (d) All paths from a node to descendant nodes contain the same number of black nodes

   Write a program to create a Red Black Tree from the given input. Your program should include the following function

   - InsertRedBlack(struct node* root, key) : Inserts a new node with the 'key' into the tree and prints parenthesized representation (with corresponding colors) of the created red-black tree.

   Input Format:

   - Each line of the input contains a positive integer 'key' or a character 't'. If the input is a positive integer then Call function InsertRedBlack(root, key). If 't' is encountered, terminate the program.

   Output Format:

   - For each line of the input, the corresponding line of the output should contain the Parenthesis Representation (key value followed by color) of the current tree.

   Sample Input:
   25
   18
   50
   80
   12
   100
   34
   t
   Sample Output:
   (25 B ( ) ( ))
   (25 B (18 R ( ) ( )) ( ))
   (25 B (18 R ( ) ( )) (50 R ( ) ( )))
   (25 B (18 B ( ) ( )) (50 B ( ) (80 R ( ) ( ))))
   (25 B (18 B (12 R ( ) ( )) ( )) (50 B ( ) (80 R ( ) ( ))))
   (25 B (18 B (12 R ( ) ( )) ( )) (80 B (50 R ( ) ( )) (100 R ( ) ( ))))
   (25 B (18 B (12 R ( ) ( )) ( )) (80 R (50 B (34 R ( ) ( )) ( )) (100 B ( ) ( ))))

   Note: For further queries contact
   BINUJOSE A binujose_p200050cs@nitc.ac.in