National Institute of Technology, Calicut

Department of Computer Science and Engineering

CS2094 – Data Structures Lab

Assignment-4 (Advanced Batch)

Policies for Submission and Evaluation

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Athena server. During evaluation your uploaded programs will be checked in Athena server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program.

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 regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz). The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>.zip (For example: ASSG4_BxxyyyyCS_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>_<FIRSTNAME>_<PROGRAM-NUMBER>.<extension> (For example: ASSG4_BxxyyyyCS_LAXMAN_1.c). If there is a part *a* and a part *b* for a particular question, then name the source files for each part separately as in ASSG4_BxxyyyyCS_LAXMAN_1b.c.

If you do not conform to the above naming conventions, your submission might not be recognized by some automated tools, and hence will lead to a score of 0 for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

Violations 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: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf.

Assignment Questions

1. Write a C program to implement hash table data structure to store student's information with roll number as key. The Hash table should support the following operations :-

- **hashTable(int m)** - create a hash table of size m
- **insert(int k)** - insert element into hash table having key value as k
- **search(int k)** - find whether element with key 'k' is present in hash table or not
- **delete(int k)** - delete the element with key 'k'

Input Format:

The first line contains a character from { 'a', 'b', 'c', 'd'} denoting

- **a** - Collision resolution by Linear probing with hash function

  $h(k,i) = (h_1 (k) + i) \bmod m$ where $h_1(k) = k \bmod m$
- **b** - Collision resolution by Quadratic probing with hash function

  $h(k,i) = (h_1(k) + i^2) \bmod m$ where $h_1(k) = k \bmod m$
- **c** - Collision resolution by Double Hashing with hash functions

  $h(k,i) = (h_1(k) + i*h_2(k)) \bmod m$

  where,

  $h_1(k) = k \bmod m$,

  $h_2(k) = R - (k \bmod R)$ { R = Prime number just smaller than size of table }
- **d** - Collision resolution by Chaining with hash function

$$h(k) = k \bmod m$$

Where,

k = Key,

m = size of hash table

Next line contains an integer, m, denoting the size of hash table.

Next lines contains a character from { ‘i’, ‘s’, ‘d’, ‘p’, ‘t’ } followed by zero or one integer.

- i x - insert the element with key x into hash table
- s x - search the element with key x in hash table. **Print 1 if present otherwise print -1**
- d x - delete the element with key x from hash table.
- p - **print the hash table in “index (key values)” pattern**. See sample output for explanation
- t - terminate the program

Note

- In case of Linear probing, quadratic probing and Double hashing, total elements (n) to be inserted into hash table will be lesser than or equal to size of hash table (m) i.e. **n<=m**
- deletion operation will always be a valid operation
- While printing the hash, multiple key values must be separated by a single white space.

Sample Input

a

7

i 76

i 93

i 40

i 47

i 10

i 55

p

s 35

s 47

d 47

s 55

t

Sample Output

0 (47)

1 (55)

2 (93)

3 (10)

4 ()

5 (40)

6 (76)

-1

1

1

Sample Input

b

7

i 76

i 40

i 48

i 5

s 5

i 55

p

s 62

d 55

s 40

s 55

t

Sample Output

1

0 (47)

1 ()

2 (5)

3 (55)

4 ()

5 (40)

6 (76)

-1

1

-1

Sample Input

c

7

i 76

i 93

i 40

i 47

i 10

i 55

p

d 40

s 47

s 76

s 40

t

Sample Output

0 ()

1 (47)

2 (93)

3 (10)

4 (55)

5 (40)

6 (76)

1

1

-1

Sample Input:

d

5

i 25

i 11

i 10

i 21

i 15

p

s 26

s 15

d 10

s 10

t

Sample Output

0 (25 10 15)

1 (11 21)

2 ()

3 ()

4 ()

-1

1

-1

2. Create a Binary Search Tree (BST) which supports the following operations:

- insert(tree, element) – adds the node specified by element (which contains the data) into the BST specified by tree.
- search(tree, key) – searches for the data specified by key in the BST specified by tree.
- findMin(tree) – retrieves the smallest data in the BST specified by tree.
- findMax(tree) – retrieves the largest data in the BST specified by tree.
- predecessor(tree, element) – retrieves the inorder-predecessor of the node specified by element in the BST specified by tree.
- successor(tree, element) – retrieves the inorder-successor of the node specified by element in the BST specified by tree.
- delete(tree, element) – removes the node specified by element from the BST specified by tree.
- inorder(tree) – To do a recursive inorder traversal of the BST.
- preorder(tree) – To do a recursive preorder traversal of the BST.
- postorder(tree) – To do a recursive postorder traversal of the BST.

Input format

The input consists of multiple lines, each containing a four letter string followed by zero or one integer. The meaning of each line is given below:

- String “stop” means stop the program.
- String “insr” means, create a node which contains the next integer(>= 0) from the input as the data part, and then, insert this node into the BST. In this case, the data is given on the same line as the string “insr”, separated by a space.
- String “srch” means, search for the key specified by the next integer(>= 0) from the input, in the BST. In this case, the key is given on the same line as the string “srch”, separated by a space. If the search is successful, output “FOUND”. Otherwise, output “NOT FOUND”.
- String “minm” means find and output the minimum value in the BST. Print “NIL” if the BST is empty.
- String “maxm” means find and output the maximum value in the BST. Print “NIL” if the BST is empty.
- String “pred” means find and output the inorder-predecessor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string “pred”, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-predecessor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- String “succ” means find and output the inorder-successor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string “succ”, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-successor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- String “delt” means delete the data specified by the next integer(>= 0) in the input from the BST, if present. In this case, the data is given on the same line as the string “delt”, separated by a space. (Here, the data to be deleted is guaranteed to be present in the BST).
- String “inor” means output the in-order traversal of the BST.
- String “prer” means output the pre-order traversal of the BST.
- String “post” means output the post-order traversal of the BST.

Output format

- The output (if any) of each command should be printed on a separate line.

Note

- **Strictly follow the output format**. It should be NIL, NOT FOUND, FOUND

Sample Input	**Sample Output**
srch 25	NOT FOUND
minm	NIL
maxm	NIL
pred 25	NOT FOUND
succ 25	NOT FOUND
insr 25	
srch 25	FOUND
minm	25
maxm	25
pred 25	NIL
succ 25	NIL
insr 13	
insr 50	
insr 45	
insr 55	
insr 18	
srch 10	NOT FOUND
srch 13	FOUND
srch 35	NOT FOUND
srch 55	FOUND
srch 80	NOT FOUND

inor	13 18 25 45 50 55
prer	25 13 18 50 45 55
post	18 13 45 55 50 25
minm	13
maxm	55
pred 13	NIL
succ 13	18
pred 18	13
succ 18	25
pred 25	18
succ 25	45
pred 45	25
succ 45	50
pred 50	45
succ 50	55
pred 55	50
succ 55	NIL
delt 55	
delt 13	
delt 25	
minm	18
maxm	50
pred 18	NIL
succ 18	45
pred 45	18
succ 45	50
pred 50	45
succ 50	NIL
delt 45	

delt 50	
delt 18	
minm	NIL
maxm	NIL
insr 90	
minm	90
maxm	90
stop	

3. Write a program that implements the disjoint-set data structure using rooted forests. Also,write functions to implement the *ranked union* and *path compression* heuristics on your data structure, and compute the efficiency of the disjoint set data structure **find** operation by applying neither, either or both of the heuristics, by counting the total number of data accesses performed over the course of the program.

Your program must support the following functions:

- **makeset(x)** creates a singleton set with element **x**.
- **find(x)** finds the representative of the set containing the element **x**.
- **union(x,y)** merges the sets containing elements **x** and **y** into a single set. The representative of the resultant set is assigned with **find(x)**, unless the ranked union heuristic is used and the ranks of both **find(x)** and **find(y)** are different. Otherwise,the representative is assigned in accordance with the ranked union heuristic.

Note that looking up an element in the data structure must be done in O(1) time.

Input format

The input consists of multiple lines, each one containing a character from {'m', 'f', 'u', 's'}followed by zero, one or two integers. The integer(s), if given, is in the range 0 to 10000.

- Call the function **makeset(x)** if the input line contains the character 'm' followed by an integer **x**. Print “PRESENT” if **x** is already present in some set, and the value of **x**, otherwise.
- Call the function **find(x)** if the input line contains the character 'f' followed by an integer **x**. Output the value of **find(x)** if **x** is found, and “NOT FOUND”, otherwise.
- Call the function **union(x,y)** if the input line contains the character 'u' followed by space separated integers **x** and **y**. Print “ERROR”, without terminating, if either **x** or **y** isn’t present in the disjoint set. Print **find(x)** itself if **find(x)=find(y)**. Otherwise, print the representative of the resultant set. The representative of the resultant set is assigned with **find(x)**, unless the ranked union heuristic is used and the ranks of both **find(x)** and **find(y)** are different. Otherwise, the representative is assigned in accordance with the ranked union heuristic.
- If the input line contains the character 's', print the number of data accesses performed by the **find** commands by each of the data structures over the course of the program and terminate.

Output format

The output consists of multiple lines of space-separated columns. The columns correspond to the following disjoint-set data structures:

- a. with neither ranked union nor path compression applied.
- b. with only ranked union applied.
- c. with only path compression applied.
- d. with both ranked union and path compression applied.

Each line in the output contains the output of the corresponding line in the input, after applying to the respective data structures. The final line of the output contains the number of data accesses performed by the **find** commands by each of the data structures over the course of the program.

Sample Input	Sample Output
m 1	1
m 2	2
m 3	3
m 4	4
m 5	5
m 6	6
m 7	7
m 8	8
m 9	9
u 1 2	1 1 1 1
u 3 4	3 3 3 3
u 5 6	5 5 5 5
u 7 8	7 7 7 7
u 9 8	9 7 9 7
u 6 8	5 5 5 5
u 4 8	3 5 3 5
u 2 8	1 5 1 5
f 9	1 5 1 5
m 10	10
u 10 9	10 5 10 5
s	38 32 33 30

4. Given the preorder and inorder traversal of a binary tree with unique keys, design and implement an algorithm to create the binary tree and print it in a list (scheme like) format.

Input format

- The first line contains a positive integer, representing the number of nodes in the tree.

- The second line contains *n* space separated integers, representing a valid preorder traversal of a tree.
- The third line contains *n* space separated integers, representing a valid inorder traversal of the tree.

Output format

- Single line representing the binary tree in a scheme like format separated by a single white space

Sample Input

10

43 15 8 30 20 35 60 50 82 70

8 15 20 30 35 43 50 60 70 82

Sample Output

(43 (15 (8 ( ) ( )) (30 (20 ( ) ( )) (35 ( ) ( )))) (60 (50 ( ) ( )) (82 (70 ( ) ( )) ( ))))

Sample Input

11

55 28 74 96 20 59 42 13 52 82 44

96 74 28 59 20 42 55 13 82 52 44

Sample Output

(55 (28 (74 (96 ( ) ( )) ( )) (20 (59 ( ) ( )) (42 ( ) ( )))) (13 ( ) (52 (82 ( ) ( )) (44 ( ) ( )))))