National Institute of Technology, Calicut
Department of Computer Science and Engineering
Monsoon 2019-20
CS2094D – Data Structures Lab

Assignment-4

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, inputfiles, etc.) except your source code, into the zip archive. The source codes must be named as ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>_<PROGRAM-NO>.<extension>. (For example: ASSG4_BxxyyyyCS_LAXMAN_1.c). If there are multiple parts 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 an F grade in the course. The department policy on academic integrity can be found at:

http://minerva.nitc.ac.in/cse/sites/default/files/attachments/news/Academic-Integrity_new.pdf .

1. Write a program to check whether a tree is a Red-Black tree or not. Red-Black Tree is a self-balancing Binary Search Tree (BST) where every node obeys the following rules.

1) Every node is colored either red or black.

2) Root of the tree is always black.

3) There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4) For each node, the number of black nodes from itself to any leaf in the subtree rooted at that node is the same.

Your program should include the following functions and should run in linear time.

isRedBlack*(*struct node* *root)* : returns 1 if the tree is an Red-Black tree otherwise 0.

Input format:
A single line containing a string containing the parenthesis representation of a tree. Note that, in addition to the normal parenthesis representation, the *key* is followed by one of the characters R/B specifying the color of the node specified after the key, followed by a single space.

Output Format:
Print 1 if the tree is a Red-Black tree; otherwise, print 0.

Sample Input and Output

Input1:
```
( 33 B ( 13 R ( 11 B ( ) ( ) ) ( 21 B ( 15 R ( ) ( ) ) ( 31 R (
) ( ) ) ) ) ( 53 B ( 41 B ( ) ( ) ) ( 61 R ( ) ( ) )  ) )
```

Output1:
```
0
```

Input2:
```
( 33 B ( 13 R ( 11 B ( ) ( ) ) ( 21 B ( 15 R ( ) ( ) ) ( 31 R (
) ( ) ) ) ) ( 53 B ( 41 R ( ) ( ) ) ( 61 R ( ) ( ) )  ) )
```

Output2:
```
1
```

2. 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:
Every line of the input contains a positive integer "key" : Call function insertRedBlack(root, key)

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	**Output**
`25`	`( 25 B ( ) ( ) )`
`18`	`( 25 B ( 18 R ( ) ( ) ) ( ) )`
`34`	`( 25 B ( 18 R ( ) ( ) ) ( 34 R ( ) ( ) ) )`
`30`	`( 25 B ( 18 B ( ) ( ) ) ( 34 B ( 30 R ( ) ( ) ) ( ) ) )`
`36`	`( 25 B ( 18 B ( ) ( ) ) ( 34 B ( 30 R ( ) ( ) ) ( 36 R ( ) ( ) ) ) )`
`28`	`( 25 B ( 18 B ( ) ( ) ) ( 34 R ( 30 B ( 28 R ( ) ( ) ) ( ) ) ( 36 B ( ) ( ) ) ) )`
`29`	`( 25 B ( 18 B ( ) ( ) ) ( 34 R ( 29 B ( 28 R ( ) ( ) ) ( 30 R ( ) ( ) ) ) ( 36 B ( ) ( ) ) ) )`

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 resulting 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. If x is found, output the value of the representative of the set containing the element x, 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. If either x or y isn’t present in the disjoint set , print “ERROR”, without terminating. If find(x) = find(y), print find(x) itself. Otherwise, print the representative of the merged set. The representative of the merged 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

```
m 1
m 2
m 3
m 4
m 5
m 6
m 7
m 8
m 9
u 1 2
u 3 4
u 5 6
u 7 8
u 9 8
u 6 8
u 4 8
u 2 8
f 9
m 10
u 10 9
s
```

Sample Output

```
1
2
3
4
5
6
7
8
9
1 1 1 1
3 3 3 3
5 5 5 5
7 7 7 7
9 7 9 7
5 5 5 5
3 5 3 5
1 5 1 5
1 5 1 5
10
10 5 10 5
38 32 33 30
```

4. Write a menu driven program to implement the strict version of binomial min-heap; that is, after any operation, no two trees in the heap should have the same rank. Your program must implement the following functions.

MAKE-HEAP(): Create and return a new heap containing no elements.

INSERT(*H, x*): Insert node *x* into heap *H*.

MINIMUM(*H*): Return the value of the smallest key in the heap *H*. The function should run in O(1) time, in the worst case.

UNION(*H1, H2*): Create and return a new heap that contains all the nodes of heaps *H1* and *H2*. Heaps *H1* and *H2* are "destroyed" by this operation.

EXTRACT-MIN(*H*): Remove the node from heap *H* whose key is minimum and return a pointer to the node.

DECREASE-KEY(*H, x, k*): If node *x*'s key is at least *k*, decrease the value of its key by *k*. Otherwise, print -1.

DELETE(*H, x*): Delete node *x* from heap *H*.

Additionally, the program should maintain an array *A* (in the main function) to hold pointers to the nodes in the heap, as detailed in the **input format** section below.

Input Format

The first line of the input contains a single integer *n*, the size of the array *A*.

Each of the following lines contains a string from the set {'insr', 'min', 'extr', 'decr', 'del', 'tc', 'stop'}, followed by zero, one or two integers, as specified below.

`insr j k`	- If *A[j]* is not pointing to any node in the heap, then insert a new node with key 'k' into the heap and add a pointer from *A[j]* to the new node.
`min`	- Print the value of the smallest key in the heap.
`extr`	- Print the value of the smallest key in the heap and delete the node containing this key.
`decr j k`	- If *A[j]* is pointing to a node with key at least *k*, decrease the key by *k*.
`del j`	- Print the key of the node pointed to by *A[j]* and delete the node.
`tc`	- Print the number of trees currently in the heap.
`stop`	- Terminate the program.

Output Format

If an operation cannot be performed, print -1.

Sample Input and Output

Input:

```
10
insr 0 10
insr 2 20
insr 5 30
insr 1 40
insr 3 50
extr
insr 0 60
decr 3 45
extr
tc
min
del 5
```

Output:

```
10
5
1
20
30
```