National Institute of Technology Calicut
Department of Computer Science and Engineering
Fourth Semester B. Tech.(CSE)-Winter 2021
CS2094D Data Structures Laboratory
Assignment #3

Submission deadline (on or before): 05.04.2021, 09:00 AM

Policies for Submission and Evaluation:

- Programs should be written in C language and compiled using C compiler in Linux platform.
- Ensure that your programs will compile and execute without errors in the Linux platform.
- 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>_<FIRST-NAME>.zip`

  (Example: *ASSG*1_*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>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: *ASSG*1_*BxxyyyyCS_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: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity_new.pdf.

QUESTIONS

1. 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.
     Note: The caller of this function is assumed to invoke SEARCH() function to locate the node x.
   - 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 k.
   - RIGHTROTATE(A,k) – Perform right rotation in the tree A, with respect to node k.
   - ISAVL(A) – Checks whether the tree pointed by A is an AVL tree or not.
   - PRINTTREE(A) – Prints the tree given by A in the paranthesis format as: (t (left-subtree)(right-subtree)). Empty parentheses ( ) 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 need to implement rotation functions.

   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 space-separated 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

Sample Output:

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

2. Given an array of n integers with any of these integers appearing any number of times. Write a program to sort these n integers in $\mathcal{O}(nlogm)$ time, where m is number of distinct elements in array.

   Hint: Use AVL tree. The idea is to extend tree node to have count of keys also. Each node in the tree is of the following type.

```
Struct node{
        int key;
        int count;              //number of times a key appears in the array
        int height;
        struct node *left;
        struct node *right;
        }
```

   Input Format:

   - The first line of the input contains an integer $n \in [1, 100]$, number of elements in the array.
   - Second line containing space separated integers of the array. The integers are in the range $[-10^6, 10^6]$

   Output Format:

   - Single line containing space separated integers of the given input array in non-decreasing order.

   Sample Input:

   12
   100 12 100 1 1 12 100 1 12 100 1 1

   Sample Output:

   1 1 1 1 1 12 12 12 100 100 100 100

3. 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:

- 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 ( ) ( )) ( ))
50	(25 B (18 R ( ) ( )) (50 R ( ) ( )))
80	(25 B (18 B ( ) ( )) (50 B ( ) (80 R ( ) ( ))))
12	(25 B (18 B (12 R ( ) ( )) ( )) (50 B ( ) (80 R ( ) ( ))))
100	(25 B (18 B (12 R ( ) ( )) ( )) (80 B (50 R ( ) ( )) (100 R ( ) ( ))))
34	(25 B (18 B (12 R ( ) ( )) ( )) (80 R (50 B (34 R ( ) ( )) ( )) (100 B ( ) ( ))))

4. Write a program to implement a Binomial Heap and perform the operations *insertion*, *deletion*, *extract_minimum* and *union*. Your program should contain the following functions:

- MakeHeap() - Creates and returns a new heap H containing no elements.
- Insert(H, x) – Inserts a new node with key 'x' into the heap H.
- Minimum(H) – Return the value of the smallest key in the heap H.
- ExtractMin(H) – Deletes the node with minimum key value from heap H.
- DecreaseKey(H, x, k) – If the node of H with key 'x' is at least 'k', then decreases the value of node with key 'x' by 'k'. Otherwise, it print -1.
- Delete(H, x) - Deletes the node with key 'x' from the heap H. If node is not present, then print -1.
- Union(H_1, H_2) - Create and return a new heap H that contains all the nodes of heaps H_1 and H_2. Heaps H_1 and H_2 are "destroyed" by this operation.

Input Format:

- Each line contains a character from 'i', 'm', 'x', 'r', 'd' and 'e' followed by at most one integer. The integers, if given, are in the range $[-10^6, 10^6]$.
- i k - inserts k into the heap
- d k - deletes node k from the heap
- p - prints the binomial heap
- m - prints the minimum element in the binomial heap (Note:- In print function, level order traversal is to be used).
- x - extracts the minimum element from the heap
- r y z - decreases the value of node with key y by z.
- e - 'exit' from the program.

Output Format:

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

Sample Input:

```
i 10
i 20
i 30
i 40
i 50
p
m
x
p
r 50 4
p
r 70 5
```

Sample Output:

```
50 10 30 20 40
10
10
20 30 40 50
46
20 30 40 46
-1
```

5. Write a program to implement a FIBONACCI HEAP and perform the operations *insertion*, *deletion*, *extract_minimum*, *decrease_key* and *union*. Your program should contain the following functions:

   - MAKEHEAP() - Creates and returns a new heap H containing no elements.
   - INSERT(H, x) – Inserts a new node with key 'x' into the heap H.
   - MINIMUM(H) – Returns a pointer to the node in heap H whose key is minimum.
   - EXTRACTMIN(H) – Deletes the node with minimum key value from heap H.
   - DECREASEKEY(H, x, k) – Decreases the value of node 'x' of the heap H by 'k', If node x's key is at least 'k'. Otherwise, it returns NIL.
   - DELETE(H, x) - Deletes the node with key 'x' from the heap H. (If node not present, it returns NIL)
   - UNION(H_1, H_2) - Create and return a new heap H that contains all the nodes of heaps H_1 and H_2. Heaps H_1 and H_2 are "destroyed" by this operation.

   Input Format:

   - Each line contains a character from 'i', 'm', 'x', 'r', 'd' and 'e' followed by at most one integer. The integers, if given, are in the range $[-10^6, 10^6]$.
   - i k - inserts k into the heap
   - d k - deletes node k from the heap
   - p - prints the Fibonacci heap (Note:- In print function, level order traversal is to be used).
   - m - prints the minimum element in the Fibonacci heap
   - x - extracts the minimum element from the heap
   - r y z - decreases the value of node with key y by z.
   - e - 'exit' from the program.

   Output Format:

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

Sample Input:

i 10
i 20
i 30
i 40
i 50
m
x
p
r 50 15
p

Sample Output:

10
10
20 30 40 50
35
20 30 40 35

6. 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 -1 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 -1, otherwise.
     - Call the function union(x,y) if the input line contains the character 'u' followed by space separated integers x and y . Print -1, 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 last 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
```