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

Assignment-3

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-NUMBER>.<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 C program to implement the operations Insertion, Deletion, Searching and Traversal. on an AVL tree, A:

Your program should include the following functions.

insert(*A, data*) – Inserts a new node with the 'data' into the tree A. The function should support duplicate entries of same 'data'. The rule to follow for inserting a new node is 'Go left if the data is less than root, Go right if the data is greater or equal root'.

search(*A, data)* - displays 'TRUE' if the data is present in the tree A else displays 'FALSE'.
If the 'data' is present the function should return the node containing the 'data' else return 'NULL'

deleteNode(*A,data*) – First call search(A, data) and if 'data' is present removes that node (ie nodeToBeDeleted) containing 'data'.
The rule for deleting the node
If nodeToBeDeleted is the leaf node remove it.
If nodetobeDeleted has one child exchange the contents of nodeToBeDeleted with that of child and then remove the child.
If nodeToBeDeleted has two children find the inorder successor of the nodeToBeDeleted.
If 'data' is not present in A print 'Deletion Impossible'

getBalance(*A,k*) – prints the balance factor of the node k in the tree A. Balance factor will be an integer, which is calculated for each node as 'height(left subtree) - height(right subtree)'.

leftRotate(*A,k)* – Perform left rotation in the tree A, with respect to node k.

rightRotate(*A,k*) – Perform right rotation in the tree A, with respect to node k.

isAVL(A) – checks if the tree pointed by A is an AVL tree or not.

printTree(*A*) – prints the tree given by A in the parenthesis format as Tree t is represented as (t(left-subtree)(right-subtree)). Empty parentheses ( ) represents a null tree.

(Note:When insertion is performed it may result in increasing the height of the tree and when deletion is performed 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 of the input contains one of the following string:

1. The string “insr” followed by data of integer i : Call function insert(A, i)
2. The string “delt” followed by data of integer i : Call function deleteNode(A,i)
3. The string “pbal” followed by data of integer i : Call function getBalance(A,i)
4. The string “disp” : Call function printTree(A)
5. The string “avlc” : Call function isAVL(A)
6. The string “srch” followed by data of integer i : Call function Search(A,i)

Output Format

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

Sample Input:

insr 4
insr 6
insr 3
insr 2
insr 1
srch 2
disp
delt 3
disp

Sample Output:

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

2. Write a program to check whether a tree is AVL tree or not. You are given a string that represents a tree. First construct a tree from the given input string and each node in the tree consists of a key and two pointers (left and right).

```
struct node {
        int key;
        struct node* left;
```

```
    struct node* right;
}
```

Do not alter the structure of the tree (Should use only one tree). If there are n nodes in the tree, your function must run in O(n) time. Your program should include the following functions.

isAVL(struct node* *root*) : returns 1 if the tree is an AVL tree otherwise 0.

Input format:
A single line containing a string representing parenthesized representation of a tree.

Output Format:
Print 1 if the tree is AVL tree otherwise 0

Sample Input and Output

Input1:
(10 (15 ( ) ( )) (20 ( ) ( )))
Output1:
0

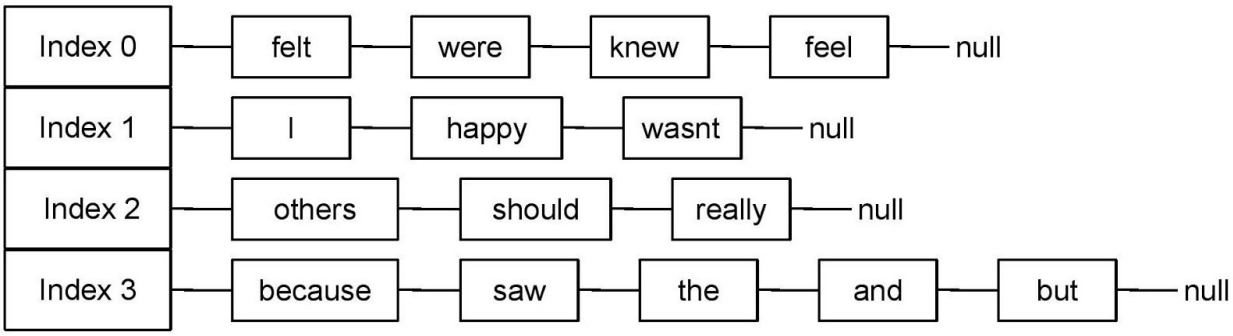
Input2:
(12 (8 (5 (4 ( ) ( )) ( )) (11 ( ) ( ))) (18 (17 ( ) ( )) ( )))
Output2:
1

3. Write a program to group the words (ignore non-alphabetic characters) according to their lengths from a given sentence (maximum of 500 characters) and a given capacity using Hashtable with separate chaining.

Example:

Given the capacity: 4
Given a sentence: I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn't really happy.

Hashtable of the sentence:



Note: Hash table is implemented as an array in which each entry contains a head pointer to a linkedlist which contains the words of the same group. Words can be grouped using the following formula.
Index number = length of a word % capacity

Words generating same index number belong to the same linked list. Duplicate words are not allowed in the list. Each node of the linkedlist is of the following type.

```
 struct node{
char *word;     // word to be store
struct node * next;      //pointer to the next node
 }
```

Input format:
First line contains a single integer representing the capacity of the hash table.
Next line contains a stream of strings separated by spaces representing the sentence.

Output format:
Print all the lists starting from index 0, then index 1 and so on, separated by newlines.
Words on each list is separated by a single space. Print 'NULL' if any of the list is empty

Program includes following functions:

hashCode(char[] ***word)***: returns the index number of the word using the above mentioned formula.

groupWords(char ***sentence[]*****, int** ***capacity*****):** creates the hashtable and prints all the list.

main(): reads the inputs and calls groupWords().

Sample input and Output

Input 1:
4
I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn't really happy.

Output 1:
felt were new feel
I happy wasnt
Others should really
because saw the and but

Input 2:
10
So I said yes to Thomas Clinton and later thought that I had said yes to God and later still realized I had said yes only to Thomas Clinton.

Output 2:
NULL
I
So to
yes and had God
said that only
later still
Thomas
Clinton thought
realized
NULL

4. A Needle in the Haystack

Our hacker, Little Stuart lately has been fascinated by ancient puzzles. One day going through some really old books he finds something scribbled on the corner of a page. Now Little Stuart

believes that the scribbled text is more mysterious than it originally looks, so he decides to find every occurrence of all the permutations of the scribbled text in the entire book. Since this is a huge task, Little Stuart needs your help, he needs you to only figure out if any permutation of the scribbled text exists in the given text string, so he can save time and analyze only those text strings where a valid permutation is present (Use hashtable for writing your program).

Input Format:

First line contains the number of test cases T. Each test case contains two lines ,first line contains pattern and next line contains a text string. All characters in both the strings are in lowercase only [a-z].

Output Format:
For each test case print "YES" or "NO" (quotes is used only for clarity) depending on whether any permutation of the pattern exists in the text string.
Constraints:
$1 \leq T \leq 100$
$1 \leq |Pattern| \leq 1000$
$1 \leq |Text\ String| \leq 100000$

Sample Input and Output

Input 1:
3
hack
indiahacks
code
eddy
coder
Iamredoc

Output 1:

YES
NO
YES

5. Write a C program to implement hash table data structure using open addressing to store student's information with roll number as key. Your program should contain the following functions:-

- **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

$h(k, i) = (h_1(k) + c_1 i + c_2 i^2) \ mod \ m$

where $h_1(k) = k \bmod m$, c_1 and c_2 are positive auxiliary constants, i = 0,1,...m-1

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 the size of table }

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

In case of quadratic probing only (option b) Next line contains the constants c_1 and c_2 separated by space

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 the size of the hash table (m) i.e. n<=m

a. deletion operation will always be a valid operation

b. While printing the hash, multiple key values must be separated by a single white space.)

Output File Format:

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

Sample Input and Output

Input 1:

```
a
7
i 76
i 93
i 40
i 47
i 10
i 55
p
s 35
s 47
d 47
s 55
T
```

Output 1:

```
0 (47)
1 (55)
2 (93)
3 (10)
4 ()
5 (40)
6 (76)
-1
1
1
```

Input 2:

```
b
7
0 1
i 76
i 40
i 47
i 5
s 5
```

i 55
p
s 62
d 55

Output 2:
1
0 (5)
1 ()
2 (47)
3 (55)
4 ()
5 (40)
6 (76)
-1

Input 3:
c
7
i 76
i 93
i 40
i 47
i 10
i 55
p
d 40
s 47
s 76
s 40
T

Output 3:
0 ()
1 (47)
2 (93)
3 (10)

4 (55)
5 (40)
6 (76)
1
1
-1

6. Write a program to sort an array of integers with many repetitions using AVL tree. A basic sorting algorithm like MergeSort, HeapSort would take O(nLogn) time where n is the number of elements. A better Solution is to use Self-Balancing Binary Search Tree like AVL tree to sort in O(n Log m) time where m is the number of distinct elements. 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:
First line containing the number of elements in the array.
Second line containing space separated integers of the array.

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

Sample Input and Output:
Input 1:
12
100 12 100 1 1 12 100 1 12 100 1 1
Output 1:
1 1 1 1 1 12 12 12 100 100 100 100