National Institute of Technology Calicut
Department of Computer Science and Engineering
Fourth Semester B. Tech.(CSE)-Winter 2021
CS2094D Data Structures Laboratory
Assignment #2

Submission deadline (on or before): 01.03.2021, 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: $ASSG1\_BxxyyyyCS\_LAXMAN.zip$). 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: $ASSG1\_BxxyyyyCS\_LAXMAN\_1.c$). 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 implement a HASH TABLE data structure to store the product details with *product_id* as key. Your program should contain the following functions.

   - HASHTABLE(m): Creates a hash table T of size m.
   - INSERT(T,k): Inserts an element into hash table T having key value k.
   - SEARCH(T,k): Checks whether an element with key 'k' is present in hash table Ts or not.
   - DELETE(T,k): Deletes the element with key 'k' from hash table.
     Note: Assume that the deletion operation will always be a valid operation. i.e. the element to be deleted is present in the hash table.

   Input Format:

   - The first line contains a character from { 'a', 'b'}:
     - Character 'a' denotes collision resolution by Quadratic Probing with hash function
       $$h(k,i) = (h_1(k) + c_1 i + c_2 i^2)\ mod\ m$$
       where $h_1(k) = k\ mod\ m$, c_1 and c_2 are positive auxiliary constants and $i \in [0, m-1]$.
     - Character 'b' denotes collision resolution by Double Hashing with hash function
       $$h(k,i) = (h_1(k) + i * h_2(k))\ mod\ m$$
       where $h_1(k) = k\ mod\ m$, $h_2(k) = R - (k\ mod\ R)$ { R = Prime number just smaller than the size of table } and $i \in [0, m-1]$.
   - Second line contains an integer $m \in [1, 100]$, denotes the size of the hash table.
   - In case of quadratic probing only (option a), next line contains two integers $c1$ and $c2$ separated by space.
   - Subsequent lines may contain 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. If the key is present in hash table, then print 1. Otherwise, print -1.
     - d x: delete the element with key x from hash table.
     - p: print the hash table in "index (key values)" pattern. If no key values are present in an index, then print "()" after "index" (Refer sample output for explanation).
     - t: terminate the program

     Note: Total number of elements n to be inserted into hash table will be lesser than or equal to size of hash table m i.e., $n \leq m$.

   Output Format:

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

   Sample Input 1:

   ```
   a
   7
   0 1
   i 76
   i 40
   i 48
   i 5
   s 5
   i 55
   p
   s 62
   d 55
   ```

t

Sample Output 1:

1
0 (48)
1 ()
2 (5)
3 (55)
4 ()
5 (40)
6 (76)
-1

Sample Input 2:

b
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 2:

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

2. Write a program to group the words according to their lengths from a given string S using a HASH TABLE of size k with separate chaining. Assume that only alphabets are present in the string S and maximum size of the string is 500. The words of the string are grouped using the following formula.

$$Index\_No = (length\_of\_word * length\_of\_word)\%k$$

where % is the modulo operation and k is the size of hash table. If the string contains multiple occurrences of a word w, then it should not be added again in a group. Only the first occurrence of w is added to the group.

Note: Hash table is implemented as an array in which each entry contains a head pointer to a linked list which contains the words of the same group. Words generating same Index_No belong to the same linked list (refer sample output for explanation). Duplicate words are not allowed in the list. Each node of the linked list 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 of the input contains an integer 'k', the size of the hash table.
- Second line of the input contains a string/sentence of words.

Output Format:

- Each line of the output should print the index number and words in it, separated by a colon(:).
- The words inside a group are separated by minus sign(-).
- If no words are present in the group then print 'null' in place of words.

Sample Input 1:

```
3
Write a program to create a hash table
```

Sample Output 1:

```
0:create
1:Write-a-program-to-hash-table
2:null
```

Sample Input 2:

```
5
This program is a program to create a hash table
```

Sample Output 2:

```
0:table
1:This-a-create-hash
2:null
3:null
4:program-is-to
```

3. Two strings are said to be *special anagrams*, if their *special power value* is same. The special power of a string is calculated to be the summation of 2 raised to the powers of the lexicographical index of each **unique character** in the string. The lexicographical index of 'a' is 0, of 'b' is 1, and so on. For example, for the string 'abcac', special power value = $2^0 + 2^1 + 2^2 = 7$. Hence, it will be considered as a special anagram of the string 'cab' whose special power value is 7 as well.

   Write a program to classify the given input string into groups of special anagrams using a HASH TABLE with separate chaining. The size of hash table is 255. For P anagrams having the same special power value, print the P strings in one line separated from each other by a space in **lexicographical order**.

   Note: The special power values of anagrams are considered as the index number in the hash table. Hash table is implemented as an array in which each entry contains a head pointer to a linked list which contains the anagrams with same special power value. The anagrams generating same special power value belong to the same linked list (refer sample output for explanation). Each node of the linked list is of the following type.

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

   Input Format:

   - The first line of the input contains an integer $n \in [1, 100]$, the number of strings in the input.

- The next n lines contain the input strings to be classified.
- Each input string S contains alphabets from $\{a, b, c, d, e, f, g, h\}$. $1 \leq |S| \leq 50$ where $|S|$ denotes the length of any input string.

Output Format:

- The output contains k lines in which anagrams with lower special power value should be printed first.
- In each of the k lines, print the words which belong to the same special anagram group in their **lexicographical order**. Each word in a line should be separated by a space.

Sample Input 1:

6
eah
hea
hac
ahe
cah
bah

Sample Output 1:

bah
cah hac
ahe eah hea

Sample Input 2:

6
heaaa
hacca
hbaba
ahb
cah
eah

Sample Output 2:

ahb hbaba
cah hacca
eah heaaa

4. 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*', '*c*', '*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 '*c*' is to check whether the tree A is AVL Tree or not.
- 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 '*c*', if the tree A is an AVL Tree, then print TRUE. If tree A is not an AVL Tree, 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 ( ) ( ) ) )
```

5. Given a Binary Search Tree (BST) T in Parenthesis Representation. Write a program that implements the following functions:

   - Main() - Creates the Binary Search Tree T given in the Parenthesis Representation and repeatedly reads a character 'r', 'u', 'i', 'l', 's' or 'e' from the console and calls the sub-functions appropriately until character 'e' is entered.
   - Predecessor(T, k) - Searches for a node with key k in T, and returns a pointer to a node which is predecessor of the node with key k if one exists; otherwise, it returns NIL.
   - Successor(T, k) - Searches for a node with key k in T, and returns a pointer to a node which is successor of the node with key k if one exists; otherwise, it returns NIL.
   - Inorder(T) - Performs recursive inorder traversal of the BST T and prints the data in the nodes of T in inorder.
   - Klargest(T, k)- Returns the K-th largest value in the BST T.
   - Ksmallest(T, k)- Returns the K-th smallest value in the BST T.

   Note: Each node of the BST is of the following type:

```
struct node {
    int data;
    struct node *leftChild;
    struct node *rightChild;
};
```

   Input format:

   - The integers, if given, are in the range $[-10^6, 10^6]$.
   - First line of the input contains space separated Parenthesis Representation of the tree T
   - Subsequent lines may contain a character from 'r', 'u', 'i', 'l' or 's' followed by at most one integer.
     - r x: find the predecessor of the node with the integer x as key in T.
     - u x: find the successor of the node with the integer x as key in T.
     - i: print the inorder traversal of T.
     - l k: find the node with kth largest value in T.
     - s k: find the node with kth smallest value in T.
   - 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 'r', if the key is present in T, then print its predecessor. If key is not present in T, then print -1.
   - For option 'u', if the key is present in T, then print its successor. If key is not present in T, then print -1.
   - For option *'i'*, print the data in the nodes of T obtained from inorder traversal.
   - For option 'l', if the k-th largest value is present in T, then print it. If it is not present in T, then print -1.
   - For option 's', if the k-th smallest value is present in T, then print it. If it is not present in T, then print -1.

   Sample Input :

```
( 25 ( 13 ( ) 18 ( ) ( ) ) ) ( 50 ( 45 ( ) ( ) )( 55 ( ) ( ) ) )
r 25
u 30
```

u 25
i
s 5
l 3
e

Sample Output :

18
-1
45
13 18 25 45 50 55
50
45

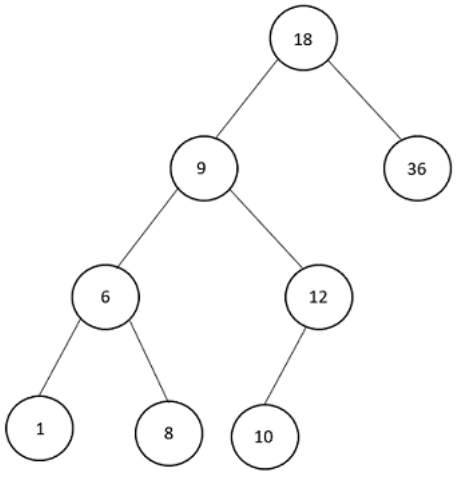
6. Given an array A of n integers and two integers a *and* b which belongs to the given array. Write a program to create a Binary Search Tree T by inserting the elements from $A[0]$ to $A[n-1]$ and find the maximum element in the path from a to b. Assume that the elements a and b are present in T.

Note: Each node of the BST is of the following type:

```
struct node {
      int data;
      struct node *leftChild;
      struct node *rightChild;
};
```

Example:

A= 18, 36, 9, 6, 12, 10, 1, 8 ,
a = 1, b = 10.
The maximum element in the path from 1 to 10 is 12. (Refer figure)



Input Format:

- First line of the input contains n integers separated by a space representing the elements of array. The integers are in the range $[-10^6, 10^6]$.
- Second line of the input contains two integers a and b.

Output Format:

- Single line that contains the maximum element in the path from a to b.

Sample Input:

18 36 9 6 12 10 1 8
1 10

Sample Output:

12

7. Given a list of n numbers sorted in ascending order, develop a program to determine the order in which these elements should be inserted into a BST such that the tree construction should be completed in $O(n \log n)$ time.(ie. the tree should be height balanced). To insert n nodes into a BST you need to make at least $O(\log n)$ comparisons. Hence the time complexity for the construction of BST with n elements will be minimum as $O(n \log n)$. Print the BST created by maintaining this property. Also find the level-wise sum of the elements in the tree, starting from root to leaf.

   Hint: For n elements in the input, make $\lceil n/2 \rceil^{th}$ element as the root and recursively perform the same for the left half and right half.

   Input Format:

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

   Output Format:

   - The output (if any) of each command should be printed on a separate line.
   - First line of the output contains Parenthesis Representation of the tree T as: (t (left-subtree)(right-subtree)). Empty parentheses ( ) represents a null tree.
   - Second line of the output contains level sums starting from root ($\lfloor \log n \rfloor + 1$ entries) separated by a space.

   Sample Input 1:

   7

   18 20 24 30 36 50 51

   Sample Output 1:

   (30 (20 (18 ( ) ( )) (24 ( ) ( ))) (50 (36 ( ) ( )) (51 ( ) ( ))))

   30 70 129