**Roll No…………….. Total No. of Pages:……**

**ST-3 (SET-I)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. The minimum number of stacks needed to implement a queue is:
2. 1
3. **2**
4. 3
5. 4
6. In a binary tree, the maximum number of nodes with height 'h' is:
   1. 2^h
   2. **2^(h+1) - 1**
   3. h^2
   4. h \* log2(h)
7. A complete binary tree can have all its levels filled except the last, and the last level is filled left to right.
   1. **True**
   2. False
8. The balance factor of a node in a binary tree is defined as \_\_\_\_\_
   1. addition of heights of left and right subtrees
   2. height of right subtree minus height of left subtree
   3. **height of left subtree minus height of right subtree**
   4. height of right subtree minus one
9. In a self-balancing Binary Search Tree, what is the maximum height for a tree with n nodes to guarantee O(log n) search complexity?
   1. n/2
   2. 2 \* log(n)
   3. **log(n)**
   4. n
10. Which operation is NOT typically performed using a Binary Search Tree?
    1. Finding the maximum element
    2. Finding the minimum element
    3. **Finding the median element**
    4. Sorting a random list of elements
11. In a binary max-heap, the new element to be inserted is placed at the \_\_\_\_\_ of the heap.
    1. Root
    2. Leftmost leaf
    3. Rightmost leaf
    4. **Bottom-right position**
12. Which property of a good hash function helps distribute keys evenly across the hash table?
    1. Collisions
    2. Clustering
    3. **Uniformity**
    4. Indexing
13. Which of the following is NOT a property of a Binary Search Tree?
    1. The left subtree of a node contains elements less than the node's key
    2. The right subtree of a node contains elements greater than the node's key
    3. **The left and right subtrees are balanced**
    4. There are no duplicate elements
14. The B-Tree is optimized for:
15. Searching in internal memory
16. Sorting elements
17. Storing elements in a queue
18. **Searching in external storage**

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

11) Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.

void fun(int n)

{

Stack S; // Say it creates an empty stack S

while (n > 0)

{

// This line pushes the value of n%2 to stack S

push(&S, n%2);

n = n/2;

}

// Run while Stack S is not empty

while (!isEmpty(&S))

printf("%d ", pop(&S)); // pop an element from S and print it

}

What does the above function do in general?

1. Prints binary representation of n in reverse order
2. **Prints binary representation of n**
3. Prints the value of Logn
4. Prints the value of Logn in reverse order

12) What is the function of following code snippet?

int fun(Node\* root) {

if (root == nullptr) return 0;

return 1 + fun(root->left) + fun(root->right);

}

a) Print inorder

b) Calculate height

**c) Calculate number of nodes**

d) Calculate

13) What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

void insert(Node\*& root, int key) {

if (root == nullptr) {

root = new Node{key, nullptr, nullptr};

return;

}

if (key < root->key)

insert(root->left, key);

else

insert(root->right, key);

}

int getHeight(Node\* root) {

if (root == nullptr)

return -1;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = nullptr;

insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 3);

insert(root, 8);

cout << "Height of the AVL tree: " << getHeight(root) << endl;

return 0;

}

a) Height of the AVL tree: 2

**b) Height of the AVL tree: 3**

c) Height of the AVL tree: 4

d) Height of the AVL tree: 5

14) What will be the output of the following code snippet?

int findHeight(Node\* root) {

if (root == NULL)

return -1;

int leftHeight = findHeight(root->left);

int rightHeight = findHeight(root->right);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = newNode(8);

root->left = newNode(3);

root->right = newNode(10);

root->left->left = newNode(1);

root->left->right = newNode(6);

root->left->right->left = newNode(4);

root->left->right->right = newNode(7);

cout << findHeight(root) << endl;

return 0;

}

a) 2

**b) 3**

c) 4

d) 5

15) What will be the function of below code snippet?

int bucketIndex = this->hashFunction(key);

node<T> \*bucketHead = this->arr[bucketIndex];

while (bucketHead != NULL)

{

if (bucketHead->key == key)

{

return bucketHead->value;

}

bucketHead = bucketHead->next;

}

return -1;

a) Insert key in hash table

b) Delete key from hash table

c) Generate hash value for key

**d) Search key in hash table**

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Given two expressions in the form of strings. The task is to compare them and check if they are similar.

Expressions consist of lowercase alphabets, ‘+’, ‘-‘ and ‘( )’.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | exp1 = "-(c+b+a)"  exp2 = "-c-b-a" | exp1 = "a-b-(c-d)"  exp2 = "a-b-c-d" | exp1 = "-(c+b-a)"  exp2 = "-c-b-a" |
| **Output** | Yes | No | No |

Solution :

**#include <bits/stdc++.h>**

**using namespace std;**

**const int MAX\_CHAR = 26;**

**// Return local sign of the operand.**

**bool adjSign(string s, int i) {**

**if (i == 0)**

**return true;**

**if (s[i - 1] == '-')**

**return false;**

**return true;**

**}**

**// Evaluate expressions into the count vector**

**void eval(string s, vector<int>& v, bool add) {**

**stack<bool> stk;**

**stk.push(true); // Global sign initially positive**

**int i = 0;**

**while (s[i] != '\0') {**

**if (s[i] == '+' || s[i] == '-') {**

**i++;**

**continue;**

**}**

**if (s[i] == '(') {**

**if (adjSign(s, i))**

**stk.push(stk.top());**

**else**

**stk.push(!stk.top());**

**} else if (s[i] == ')')**

**stk.pop();**

**else {**

**if (stk.top())**

**v[s[i] - 'a'] += (adjSign(s, i) ? add ? 1 : -1 : add ? -1 : 1);**

**else**

**v[s[i] - 'a'] += (adjSign(s, i) ? add ? -1 : 1 : add ? 1 : -1);**

**}**

**i++;**

**}**

**}**

**// Returns true if expr1 and expr2 represent the same expressions**

**bool areSame(string expr1, string expr2) {**

**vector<int> v(MAX\_CHAR, 0); // Initialize vector for all operands**

**eval(expr1, v, true); // Put signs of all operands in expr1**

**eval(expr2, v, false); // Subtract signs of operands in expr2**

**// If expressions are same, the vector must be all 0.**

**for (int i = 0; i < MAX\_CHAR; i++)**

**if (v[i] != 0)**

**return false;**

**return true;**

**}**

**int main() {**

**string expr1 = "-(a+b-c)", expr2 = "-a-b-c";**

**if (areSame(expr1, expr2))**

**cout << "Yes\n";**

**else**

**cout << "No\n";**

**return 0;**

**}**

Q17) Given a binary tree, print the vertical sum of the tree.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 1  / \  2 3  / \ / \  4 5 6 7  \  8  \  9 | 1  / \  2 3 | 1  / \  2 3  / \ / \  4 5 6 7 |
| **Output** | Following are the values of vertical sums with the positions of the columns with respect to root  -2: 4  -1: 2  0: 12  1: 11  2: 7  3: 9 | Following are the values of vertical sums with the positions of the columns with respect to root  -1: 2  0: 1  1: 3 | Following are the values of vertical sums with the positions of the columns with respect to root  -2: 4  -1: 2  0: 12  1: 3  2: 7 |

Solution :

**#include<bits/stdc++.h>**

**using namespace std;**

**struct Node**

**{**

**int data;**

**struct Node \*left, \*right;**

**};**

**// A utility function to create a new Binary Tree node**

**Node\* newNode(int data)**

**{**

**Node \*temp = new Node;**

**temp->data = data;**

**temp->left = temp->right = NULL;**

**return temp;**

**}**

**// Traverses the tree in in-order form and populates a hashMap that contains the vertical sum**

**void verticalSumUtil(Node \*node, int hd,**

**map<int, int> &Map)**

**{**

**// Base case**

**if (node == NULL) return;**

**// Recur for left subtree**

**verticalSumUtil(node->left, hd-1, Map);**

**// Add val of current node to map entry of corresponding hd**

**Map[hd] += node->data;**

**// Recur for right subtree**

**verticalSumUtil(node->right, hd+1, Map);**

**}**

**// Function to find vertical sum**

**void verticalSum(Node \*root)**

**{**

**// a map to store sum of nodes for each horizontal distance**

**map < int, int> Map;**

**map < int, int> :: iterator it;**

**// populate the map**

**verticalSumUtil(root, 0, Map);**

**// Prints the values stored by VerticalSumUtil()**

**for (it = Map.begin(); it != Map.end(); ++it)**

**{**

**cout << it->first << ": "**

**<< it->second << endl;**

**}**

**}**

**int main()**

**{**

**// Create binary tree as shown in above figure**

**Node \*root = newNode(1);**

**root->left = newNode(2);**

**root->right = newNode(3);**

**root->left->left = newNode(4);**

**root->left->right = newNode(5);**

**root->right->left = newNode(6);**

**root->right->right = newNode(7);**

**root->right->left->right = newNode(8);**

**root->right->right->right = newNode(9);**

**cout << "Following are the values of vertical sums with the positions of the columns with respect to root\n";**

**verticalSum(root);**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) Write a C++ program to find med in stream of running integers

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 5 10 15 | 1, 2, 3, 4 | 5, 15, 10, 20, 3 |
| **Output** | 5, 7.5, 10 | 1, 1.5, 2, 2.5 | 5 10 10 12.5 10 |

Solution :

**#include<bits/stdc++.h>**

**using namespace std;**

**// function to calculate med of stream**

**void printMedians(double arr[], int n)**

**{**

**// max heap to store the smaller half elements**

**priority\_queue<double> s;**

**// min heap to store the greater half elements**

**priority\_queue<double,vector<double>,greater<double> > g;**

**double med = arr[0];**

**s.push(arr[0]);**

**cout << med << endl;**

**// reading elements of stream one by one**

**/\* At any time we try to make heaps balanced and**

**their sizes differ by at-most 1. If heaps are**

**balanced,then we declare median as average of**

**min\_heap\_right.top() and max\_heap\_left.top()**

**If heaps are unbalanced,then median is defined**

**as the top element of heap of larger size \*/**

**for (int i=1; i < n; i++)**

**{**

**double x = arr[i];**

**// case1(left side heap has more elements)**

**if (s.size() > g.size())**

**{**

**if (x < med)**

**{**

**g.push(s.top());**

**s.pop();**

**s.push(x);**

**}**

**else**

**g.push(x);**

**med = (s.top() + g.top())/2.0;**

**}**

**// case2(both heaps are balanced)**

**else if (s.size()==g.size())**

**{**

**if (x < med)**

**{**

**s.push(x);**

**med = (double)s.top();**

**}**

**else**

**{**

**g.push(x);**

**med = (double)g.top();**

**}**

**}**

**// case3(right side heap has more elements)**

**else**

**{**

**if (x > med)**

**{**

**s.push(g.top());**

**g.pop();**

**g.push(x);**

**}**

**else**

**s.push(x);**

**med = (s.top() + g.top())/2.0;**

**}**

**cout << med << endl;**

**}**

**}**

**int main()**

**{**

**// stream of integers**

**double arr[] = {5, 15, 10, 20, 3};**

**int n = sizeof(arr)/sizeof(arr[0]);**

**printMedians(arr, n);**

**return 0;**

**}**