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

**ST-3 (SET-VI)**

**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. A stack-based algorithm is used to evaluate postfix expressions because it eliminates the need for:
   1. Parentheses
   2. Arithmetic operations
   3. **Operator precedence rules**
   4. Operand values
2. The postfix expression "3 4 + 5 \*" evaluates to:
   1. 27
   2. 23
   3. **35**
   4. 60
3. A binary tree is threaded by:
   1. Adding extra nodes to the tree
   2. **Linking the null pointers of leaf nodes to other nodes**
   3. Creating a circular linked list from the tree nodes
   4. Replacing each null pointer with a pointer to the previous node
4. The Morris Traversal algorithm is used to perform which type of traversal?
   1. **Inorder**
   2. Preorder
   3. Postorder
   4. Level order
5. Which collision resolution technique aims to minimize clustering by spreading out the probing sequence?
   1. Separate Chaining
   2. Linear Probing
   3. Quadratic Probing
   4. **Double Hashing**
6. What is the output of a level-order traversal of the binary tree below?

7

/ \

4 9

* 1. **7 4 9**
  2. 4 7 9
  3. 7 9 4
  4. 4 9 7

1. Which of the following operations can unbalance an AVL tree?
   1. Insertion
   2. Deletion
   3. **Both insertion and deletion**
   4. Searching
2. In AVL tree deletion, if a node has two children, what strategy is used to find its successor?
   1. **Inorder successor**
   2. Preorder successor
   3. Postorder successor
   4. Random successor
3. A \_\_\_\_\_\_\_\_ is a special Tree-based data structure in which the tree is a complete binary tree.?
   1. Graph
   2. **Heap**
   3. List
   4. Stack
4. In a binary tree, a node with at least one child is called:
   1. Root node
   2. **Internal node**
   3. Leaf node
   4. Sibling node

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

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

11) Consider the following operation performed on a stack of size 5.

Push(1);

Pop();

Push(2);

Push(3);

Pop();

Push(4);

Pop();

Pop();

Push(5);

After the completion of all operation, the number of elements present in stack is?

1. **1**
2. 2
3. 3
4. 4

12) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findMax(Node\* root) {

if (root == nullptr) return INT\_MIN;

int leftMax = findMax(root->left);

int rightMax = findMax(root->right);

return max(root->data, max(leftMax, rightMax));

}

int main() {

Node\* root = new Node(5);

root->left = new Node(3);

root->right = new Node(8);

root->left->left = new Node(9);

root->left->right = new Node(4);

root->right->left = new Node(6);

cout << findMax(root);

return 0;

}

a) 6

b) 8

**c) 9**

d) 5

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

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

int main() {

Node\* root = nullptr;

root = new Node{10, nullptr, nullptr};

root->left = new Node{5, nullptr, nullptr};

root->right = new Node{15, nullptr, nullptr};

root->left->left = new Node{3, nullptr, nullptr};

root->left->right = new Node{8, nullptr, nullptr};

cout << "Root key: " << root->key << endl;

cout << "Left child key: " << root->left->key << endl;

cout << "Right child key: " << root->right->key << endl;

cout << "Left grandchild key: " << root->left->left->key << endl;

cout << "Right grandchild key: " << root->left->right->key << endl;

return 0;

}

1. **Root key: 10, Left child key: 5, Right child key: 15, Left grandchild key: 3, Right grandchild key: 8**
2. Root key: 5, Left child key: 10, Right child key: 15, Left grandchild key: 3, Right grandchild key: 8
3. Root key: 15, Left child key: 10, Right child key: 5, Left grandchild key: 8, Right grandchild key: 3
4. Root key: 10, Left child key: 15, Right child key: 5, Left grandchild key: 3, Right grandchild key: 8

14) What will be the output of the following AVL tree deletion operation?

AVLTree tree;

tree.insert(10);

tree.insert(5);

tree.insert(15);

tree.insert(3);

tree.deleteNode(10);

tree.inOrderTraversal();

**a) 3 5 15**

b) 3 5

c) 3 15

d) 5 15

15) What are the values of h1(k) and h2(k) in the hash function?

a)

h1(k) = m mod k

h2(k) = 1+ (m’ mod k)

b)

h1(k) = 1 + (m mod k)

h2(k) = m’ mod k

c)

h1(k) = 1+ (k mod m)

h2(k) = k mod m

**d)**

**h1(k) = k mod m**

**h2(k) = 1+ (k mod m’)**

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

Q16) Evaluate the value of an arithmetic expression in Reverse Polish Notation (postfix notation).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | ["2", "1", "+", "3", "\*"] | ["6", "3", "2", "4", "+","-","\*"] | ["6", "3","/"] |
| **Output** | 9 | -18 | 2 |

Solution :

**#include <iostream>**

**#include <stack>**

**#include <vector>**

**using namespace std;**

**// Function to evaluate Reverse Polish Notation (RPN) expression**

**int evalRPN(vector<string>& tokens) {**

**stack<int> st; // Initialize a stack to hold operands**

**// Iterate through each token in the expression**

**for (string token : tokens) {**

**if (token == "+" || token == "-" || token == "\*" || token == "/") {**

**// If the token is an operator, pop the top two elements from the stack**

**int num2 = st.top(); st.pop();**

**int num1 = st.top(); st.pop();**

**// Perform the operation based on the operator**

**if (token == "+") st.push(num1 + num2);**

**else if (token == "-") st.push(num1 - num2);**

**else if (token == "\*") st.push(num1 \* num2);**

**else if (token == "/") st.push(num1 / num2);**

**} else {**

**// If the token is an operand, convert it to an integer and push it onto the stack**

**st.push(stoi(token));**

**}**

**}**

**// The final result will be left on the top of the stack**

**return st.top();**

**}**

**int main() {**

**vector<string> tokens = {"6", "3", "2", "4", "+","-","\*"};**

**cout << "Result: " << evalRPN(tokens) << endl; // Evaluate the RPN expression and display the result**

**return 0;**

**}**

Q17) Write a CPP program to print the top view of binary tree. Top view of a binary tree is the set of nodes visible when the tree is viewed from the top.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 10  / \  20 30  / \ / \  40 60 90 100 | 1  / \  2 3  \  4  \  5  \  6 | 10  / \  20 30 |
| **Output** | 40 20 10 30 100 | 2 1 3 6 | 20 10 30 |

Solution :

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

**using namespace std;**

**// Structure to represent a tree node**

**struct Node {**

**int data;**

**Node\* left;**

**Node\* right;**

**Node(int val) {**

**data = val;**

**left = right = NULL;**

**}**

**};**

**// Function to print the top view of a binary tree**

**void topView(Node\* root) {**

**if (!root)**

**return;**

**map<int, int> verticalMap; // Map to store vertical level and node data**

**queue<pair<Node\*, int>> q; // Queue for BFS traversal**

**q.push({root, 0});**

**while (!q.empty()) {**

**Node\* node = q.front().first;**

**int verticalLevel = q.front().second;**

**q.pop();**

**// Insert the node's data if not present in the map**

**if (verticalMap.find(verticalLevel) == verticalMap.end()) {**

**verticalMap[verticalLevel] = node->data;**

**}**

**// Push left child with decreased vertical level**

**if (node->left) {**

**q.push({node->left, verticalLevel - 1});**

**}**

**// Push right child with increased vertical level**

**if (node->right) {**

**q.push({node->right, verticalLevel + 1});**

**}**

**}**

**// Print the nodes in the top view**

**for (const auto& entry : verticalMap) {**

**cout << entry.second << " ";**

**}**

**}**

**int main() {**

**Node\* root = new Node(10);**

**root->left = new Node(20);**

**root->right = new Node(30);**

**root->left->right = new Node(60);**

**root->left->left = new Node(40);**

**root->right->left = new Node(90);**

**root->right->right = new Node(100);**

**cout << "Top view of the binary tree: ";**

**topView(root);**

**return 0;**

**}**

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

Q18) Write a C++ program to insert new element to MAX-Heap.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | heap={ 10, 5, 3, 2, 4 }  key=15 | heap={ 12,6,4,3,5 }  key=21 | heap={ 30,15,19 }  key=2 |
| **Output** | 15 5 10 2 4 3 | 21 6 12 3 5 4 | 30 15 19 2 |

Solution :

**#include <iostream>**

**using namespace std;**

**#define MAX 1000 // Max size of Heap**

**// Function to heapify ith node in a Heap**

**// of size n following a Bottom-up approach**

**void heapify(int arr[], int n, int i)**

**{**

**// Find parent**

**int parent = (i - 1) / 2;**

**if (arr[parent] > 0) {**

**// For Max-Heap**

**// If current node is greater than its parent**

**// Swap both of them and call heapify again**

**// for the parent**

**if (arr[i] > arr[parent]) {**

**swap(arr[i], arr[parent]);**

**// Recursively heapify the parent node**

**heapify(arr, n, parent);**

**}**

**}**

**}**

**// Function to insert a new node to the Heap**

**void insertNode(int arr[], int& n, int Key)**

**{**

**// Increase the size of Heap by 1**

**n = n + 1;**

**// Insert the element at end of Heap**

**arr[n - 1] = Key;**

**// Heapify the new node following a**

**// Bottom-up approach**

**heapify(arr, n, n - 1);**

**}**

**// A utility function to print array of size n**

**void printArray(int arr[], int n)**

**{**

**for (int i = 0; i < n; ++i)**

**cout << arr[i] << " ";**

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

**}**

**int main()**

**{**

**// Array representation of Max-Heap**

**// 10**

**// / \**

**// 5 3**

**// / \**

**// 2 4**

**int arr[MAX] = { 10, 5, 3, 2, 4 };**

**int n = 5;**

**int key = 15;**

**insertNode(arr, n, key);**

**printArray(arr, n);**

**// Final Heap will be:**

**// 15**

**// / \**

**// 5 10**

**// / \ /**

**// 2 4 3**

**return 0;**

**}**