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

**ST-3 (SET-II)**

**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. Assume that the operators +, -, × are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, x , +, -. The postfix expression corresponding to the infix expression a + b × c - d ^ e ^ f is
2. **abc × + def ^ ^ -**
3. abc × + de ^ f ^ -
4. ab + c × d - e ^ f ^
5. - + a × bc ^ ^ def
6. In a binary search tree (BST), the right subtree of a node contains:
   1. Nodes with smaller keys
   2. **Nodes with greater keys**
   3. Nodes with equal keys
   4. Nodes with random keys
7. What is the output of an in-order traversal of the binary tree below?

a

/ \

b c

* 1. a b c
  2. **b a c**
  3. b c a
  4. c b a

1. A Binary Search Tree is converted into a Doubly Linked List using which traversal?
2. **Inorder**
3. Preorder
4. Postorder
5. Level-order
6. Which type of Binary Search Tree allows duplicate elements?
7. AVL Tree
8. Red-Black Tree
9. Splay Tree
10. **Multiset**
11. In a B-Tree, the minimum degree specifies:
12. **The number of children each internal node can have**
13. The number of keys each internal node can have
14. The number of keys each leaf node can have
15. The maximum height of the tree
16. Which of the following is NOT a self-balancing Binary Search Tree?
    1. Splay Tree
    2. B-Tree
    3. AVL Tree
    4. **Binary Heap**
17. If the initial min-heap is {1, 3, 5, 7}, what will be the min-heap after inserting the element 2?
    1. {1, 3, 5, 7, 2}
    2. {1, 2, 5, 7, 3}
    3. {1, 2, 5, 3, 7}
    4. **{1, 2, 3, 5, 7}**
18. In an array-based representation of a binary min-heap, the index of the parent of element 'i' is given by:
19. **(i - 1) / 2**
20. (i + 1) / 2
21. i / 2
22. 2i
23. What is the primary advantage of using a Red-Black Tree over an AVL Tree?
24. **Faster insertion and deletion operations**
25. Smaller memory footprint
26. Better support for multithreading
27. Simpler implementation

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

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

11) Consider the following pseudocode that uses a stack

declare a stack of characters

while ( there are more characters in the word to read )

{

read a character

push the character on the stack

}

while ( the stack is not empty )

{

pop a character off the stack

write the character to the screen

}

What is output for input "abc"?

a) acb

b) bca

**c) cba**

d) abc

12) What is 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) {}

};

void printPostorder(Node\* root) {

if (root == nullptr) return;

printPostorder(root->left);

printPostorder(root->right);

cout << root->data << " ";

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

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

root->left->right = new Node(5);

printPostorder(root);

return 0;

}

**a) 4 5 2 3 1**

b) 5 4 2 3 1

c) 4 5 1 2 3

d) 5 4 1 3 2

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

bool isAVL(Node\* root) {

if (root == nullptr)

return true;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

int balanceFactor = abs(leftHeight - rightHeight);

return (balanceFactor <= 1) && isAVL(root->left) && isAVL(root->right);

}

int main() {

Node\* root = nullptr;

insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 3);

insert(root, 8);

if (isAVL(root))

cout << "The tree is an AVL tree." << endl;

else

cout << "The tree is not an AVL tree." << endl;

return 0;

}

**a) The tree is an AVL tree.**

b) The tree is not an AVL tree.

c) The program will not compile due to an error.

d) The program will run into an infinite loop.

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

Node\* root = newNode(4);

root->left = newNode(2);

root->right = newNode(6);

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

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

cout << root->left->right->data << endl;

a) 1

b) 2

**c) 3**

d) 4

15) What will be returned by give function in hashing?

float func()

{

return (float)(this->numOfElements + 1) / (float)(this->capacity);

}

a) Number of elements in hash table

b) Capacity of hash table

**c) Load factor**

d) Hash key

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

Q16) Given a stack of integers, sort it in ascending order using another temporary stack.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | [4, 3, 1, 8] | [9,13,8,67,24] | [5,15,3,41,0] |
| **Output** | [1, 3, 4, 8] | [8,9,13,24,67] | [0,3,5,15,41] |

Solution :

**// C++ program to sort a stack using an auxiliary stack.**

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

**using namespace std;**

**// This function returns the sorted stack**

**stack<int> sortStack(stack<int> &input) {**

**stack<int> tmpStack;**

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

**// Pop out the first element**

**int tmp = input.top();**

**input.pop();**

**// While the temporary stack is not empty and the top**

**// of the stack is lesser than 'tmp'**

**while (!tmpStack.empty() && tmpStack.top() < tmp) {**

**// Pop from the temporary stack and push**

**// it to the input stack**

**input.push(tmpStack.top());**

**tmpStack.pop();**

**}**

**// Push 'tmp' into the temporary stack**

**tmpStack.push(tmp);**

**}**

**return tmpStack;**

**}**

**// Main function**

**int main() {**

**stack<int> input;**

**input.push(34);**

**input.push(3);**

**input.push(31);**

**input.push(98);**

**input.push(92);**

**input.push(23);**

**// Use the sortStack function to sort the 'input' stack**

**stack<int> tmpStack = sortStack(input);**

**cout << "Sorted numbers are:\n";**

**// Print the sorted numbers from the temporary stack**

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

**cout << tmpStack.top() << " ";**

**tmpStack.pop();**

**}**

**return 0;**

**}**

Q17) Given an array arr[] containing n elements. The problem is to find the maximum number of distinct elements (non-repeating) after removing k elements from the array.

Note: 1 <= k <= n.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr[] = {5, 7, 5, 5, 1, 2, 2}, k = 3 | arr[] = {1, 2, 3, 4, 5, 6, 7}, k = 5 | arr[] = {1, 2, 2, 2}, k = 1 |
| **Output** | 4 | 2 | 1 |

Solution :

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

**using namespace std;**

**// function to find maximum distinct elements after removing k elements**

**int maxDistinctNum(int a[], int n, int k)**

**{**

**int i;**

**multiset<int> s;**

**// making multiset from given array**

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

**{**

**if(s.find(a[i])==s.end()||k==0)**

**s.insert(a[i]);**

**else**

**{**

**k--;**

**}**

**}**

**if(k!=0)**

**return s.size()-k;**

**else**

**{**

**set<int> st;**

**for(auto it:s)**

**{**

**st.insert(it);**

**}**

**return st.size();**

**}**

**}**

**int main()**

**{**

**int arr[] = { 5, 7, 5, 5, 1, 2, 2 };**

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

**int k = 3;**

**// Function Call**

**cout << "Maximum distinct elements = "**

**<< maxDistinctNum(arr, n, k);**

**return 0;**

**}**

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

Q18) Given a binary tree and an array, the task is to find if the given array sequence is present as a root-to-leaf path in given tree.

Tree:

5

/ \

2 3

/ \

1 4

/ \

6 8

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr[] = {5, 2, 4, 8} for above tree | arr[] = {5, 3, 4, 9} for above tree | arr[] = {5, 3,} for above tree |
| **Output** | "Path Exist" | "Path does not Exist" | "Path Exist" |

Solution :

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

**using namespace std;**

**/\* A binary tree node has data, pointer to left child**

**and a pointer to right child \*/**

**struct Node**

**{**

**int data;**

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

**};**

**/\* utility that allocates a new node with the**

**given data and NULL left and right pointers. \*/**

**struct Node\* newnode(int data)**

**{**

**struct Node\* node = new Node;**

**node->data = data;**

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

**return (node);**

**}**

**// Util function**

**bool existPathUtil(struct Node \*root, int arr[], int n, int index)**

**{**

**// If root is NULL or reached end of the array**

**if(root == NULL or index==n)**

**return false;**

**// If current node is leaf**

**if (root->left == NULL && root->right == NULL)**

**{**

**if((root->data == arr[index]) && (index == n-1))**

**return true;**

**return false;**

**}**

**// If current node is equal to arr[index] this means that till this level path has been matched and**

**// remaining path can be either in left subtree or right subtree.**

**return ((index < n) && (root->data == arr[index]) &&**

**(existPathUtil(root->left, arr, n, index+1) ||**

**existPathUtil(root->right, arr, n, index+1) ));**

**}**

**// Function to check given sequence of root to leaf path exist in tree or not.**

**// index represents current element in sequence of root to leaf path**

**bool existPath(struct Node \*root, int arr[], int n, int index)**

**{**

**if(!root)**

**return (n==0);**

**return existPathUtil(root, arr, n, 0);**

**}**

**int main()**

**{**

**// arr[] --> sequence of root to leaf path**

**int arr[] = {5, 3};**

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

**struct Node \*root = newnode(5);**

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

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

**root->left->left = newnode(1);**

**root->left->right = newnode(4);**

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

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

**existPath(root, arr, n, 0)? cout << "Path Exists" :cout << "Path does not Exist";**

**return 0;**

**}**