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

**ST-3 (SET-V)**

**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) Which of the following algorithms uses a stack to keep track of the parentheses in an expression?

1. Depth-first search
2. Breadth-first search
3. Dijkstra's algorithm
4. **Parentheses balancing algorithm**

2) A stack-based algorithm is used to convert infix expressions to postfix expressions because it helps manage:

1. Operator associativity
2. Operand values
3. **Parentheses**
4. Operator precedence

3) In a binary tree, the depth of a node is defined as:

1. The number of subtrees it has
2. The number of children it has
3. The number of ancestors it has
4. **The distance from the root node**

4) Given a binary tree, write a function to determine whether it is a valid binary search tree (BST).

1. Check if it is a complete binary tree
2. Check if it is a full binary tree
3. **Perform an inorder traversal and check if the values are in ascending order**
4. Check if the root node is greater than its left child and smaller than its right child

5) In a binary tree, a node with no children is called:

1. Root node
2. Internal node
3. **Leaf node**
4. Sibling node

6) What is the primary disadvantage of using open addressing for collision resolution?

1. It requires extra memory for linked lists.
2. **It leads to clustering.**
3. It cannot handle large datasets.
4. It may require frequent resizing.

7) What is the output of a post-order traversal of the binary tree below?

2

/ \

1 3

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

8)For a binary search tree to be an AVL tree, the balance factor of each node must be in the range of \_\_\_\_\_\_.

1. **-1 to 1**
2. -2 to 2
3. 0 to 1
4. -1 to 0

9)When deleting a node in an AVL tree, how many recursive calls are made to reach the target node?

1. One recursive call
2. Two recursive calls
3. Three recursive calls
4. **It depends on the tree's height**

10) What is the worst-case time complexity to insert 'k' elements consecutively into a binary heap of size 'n' using the standard heap insertion algorithm?

1. **O(k log n)**
2. O(n log k)
3. O(k + n)
4. O(n^2)

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

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

11) A binary tree can be represented by an array by using the following formula to find the indices of left and right children of a node at index 'i':

Left child index: 2i + 1

Right child index: 2i + 2

**a) True**

b) False

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 printInorder(Node\* root) {

if (root == nullptr) return;

printInorder(root->left);

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

printInorder(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

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

printInorder(root);

return 0;

}

**a) 4 2 1 3**

b) 1 2 4 3

c) 4 2 3 1

d) 1 4 2 3

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 main() {

Node\* root = nullptr;

insert(root, 12);

insert(root, 7);

insert(root, 17);

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

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

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

return 0;

}

**a) Root key: 12, Left child key: 7, Right child key: 17**

b) Root key: 7, Left child key: 17, Right child key: 12

c) Root key: 12, Left child key: 17, Right child key: 7

d) Root key: 17, Left child key: 12, Right child key: 7

14) Consider the pseudo code:

int avl(binarysearchtree root):

if(not root)

return 0

left\_tree\_height = avl(left\_of\_root)

if(left\_tree\_height== -1)

return left\_tree\_height

right\_tree\_height= avl(right\_of\_root)

if(right\_tree\_height==-1)

return right\_tree\_height

Does the above code can check if a binary search tree is an AVL tree?

**a) yes**

b) no

15) Where does the number 14 get inserted in the following table?

Index Key

0

1 79

2

3

4 69

5 98

6

7 72

8

9

10

11 50

12

a) 2

**b) 9**

c) 4

d) 8

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

Q16) Given an array, find the next greater element for each element. If there is no greater element, output -1 for that element.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | [4, 5, 2, 10] | [9,8,13,17,4] | [9,12,6,61,91] |
| **Output** | [5, 10, 10, -1] | [13 13 17 -1 -1] | [12 61 61 91 -1] |

Solution :

**#include <iostream>**

**using namespace std;**

**/\* prints element and NGE pair**

**for all elements of arr[] of size n \*/**

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

**{**

**int next, i, j;**

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

**next = -1;**

**for (j = i + 1; j < n; j++) {**

**if (arr[i] < arr[j]) {**

**next = arr[j];**

**break;**

**}**

**}**

**cout << next << " ";**

**}**

**}**

**int main()**

**{**

**int arr[] = { 11, 13, 21, 3 };**

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

**printNGE(arr, n);**

**return 0;**

**}**

Q17) Implement a program to encode and decode a given text using Huffman coding.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | text= linkedin | text= morya | text=snap |
| **Output** | Character With there Frequencies:  d 010  e 011  i 00  k 111  l 110  n 10  Encoded Huffman data: 11000101110110100010  Decoded Huffman Data: linkedin | Character With there Frequencies:  a 110  m 10  o 111  r 01  y 00  Encoded Huffman data:  101110100110  Decoded Huffman Data:  morya | Character With there Frequencies:  a 00  n 10  p 01  s 11  Encoded Huffman data:  11100001  Decoded Huffman Data:  Snap |

Solution :

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

**#define MAX\_TREE\_HT 256**

**using namespace std;**

**// to map each character its huffman value**

**map<char, string> codes;**

**// To store the frequency of character of the input data**

**map<char, int> freq;**

**// A Huffman tree node**

**struct MinHeapNode {**

**char data; // One of the input characters**

**int freq; // Frequency of the character**

**MinHeapNode \*left, \*right; // Left and right child**

**MinHeapNode(char data, int freq)**

**{**

**left = right = NULL;**

**this->data = data;**

**this->freq = freq;**

**}**

**};**

**// utility function for the priority queue**

**struct compare {**

**bool operator()(MinHeapNode\* l, MinHeapNode\* r)**

**{**

**return (l->freq > r->freq);**

**}**

**};**

**// utility function to print characters along with**

**// there huffman value**

**void printCodes(struct MinHeapNode\* root, string str)**

**{**

**if (!root)**

**return;**

**if (root->data != '$')**

**cout << root->data << ": " << str << "\n";**

**printCodes(root->left, str + "0");**

**printCodes(root->right, str + "1");**

**}**

**// utility function to store characters along with**

**// there huffman value in a hash table, here we**

**// have C++ STL map**

**void storeCodes(struct MinHeapNode\* root, string str)**

**{**

**if (root == NULL)**

**return;**

**if (root->data != '$')**

**codes[root->data] = str;**

**storeCodes(root->left, str + "0");**

**storeCodes(root->right, str + "1");**

**}**

**// STL priority queue to store heap tree, with respect**

**// to their heap root node value**

**priority\_queue<MinHeapNode\*, vector<MinHeapNode\*>, compare>**

**minHeap;**

**// function to build the Huffman tree and store it**

**// in minHeap**

**void HuffmanCodes(int size)**

**{**

**struct MinHeapNode \*left, \*right, \*top;**

**for (map<char, int>::iterator v = freq.begin();**

**v != freq.end(); v++)**

**minHeap.push(new MinHeapNode(v->first, v->second));**

**while (minHeap.size() != 1) {**

**left = minHeap.top();**

**minHeap.pop();**

**right = minHeap.top();**

**minHeap.pop();**

**top = new MinHeapNode('$',**

**left->freq + right->freq);**

**top->left = left;**

**top->right = right;**

**minHeap.push(top);**

**}**

**storeCodes(minHeap.top(), "");**

**}**

**// utility function to store map each character with its**

**// frequency in input string**

**void calcFreq(string str, int n)**

**{**

**for (int i = 0; i < str.size(); i++)**

**freq[str[i]]++;**

**}**

**// function iterates through the encoded string s**

**// if s[i]=='1' then move to node->right**

**// if s[i]=='0' then move to node->left**

**// if leaf node append the node->data to our output string**

**string decode\_file(struct MinHeapNode\* root, string s)**

**{**

**string ans = "";**

**struct MinHeapNode\* curr = root;**

**for (int i = 0; i < s.size(); i++) {**

**if (s[i] == '0')**

**curr = curr->left;**

**else**

**curr = curr->right;**

**// reached leaf node**

**if (curr->left == NULL and curr->right == NULL) {**

**ans += curr->data;**

**curr = root;**

**}**

**}**

**// cout<<ans<<endl;**

**return ans + '\0';**

**}**

**int main()**

**{**

**string str = "linkedin";**

**string encodedString, decodedString;**

**calcFreq(str, str.length());**

**HuffmanCodes(str.length());**

**cout << "Character With there Frequencies:\n";**

**for (auto v = codes.begin(); v != codes.end(); v++)**

**cout << v->first << ' ' << v->second << endl;**

**for (auto i : str)**

**encodedString += codes[i];**

**cout << "\nEncoded Huffman data:\n"**

**<< encodedString << endl;**

**decodedString= decode\_file(minHeap.top(), encodedString);**

**cout << "\nDecoded Huffman Data:\n"**

**<< decodedString << endl;**

**return 0;**

**}**

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

Q18) Given an array of N elements and two integers A, B which belong to the given array. Create a Binary Search Tree by inserting elements from arr[0] to arr[n-1]. The task is to find the maximum element in the path from A to B.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr[] = { 18, 36, 9, 6, 12, 10, 1, 8 }  a = 1  b = 10 | arr[] = { 18, 36, 9, 6, 12, 10, 1, 8 }  a = 8  b = 18 | arr[] = { 18, 36, 9, 6, 12, 10, 1, 8 }  a = 8  b = 10 |
| **Output** | 12 | 18 | 12 |

Solution :

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

**using namespace std;**

**struct Node**

**{**

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

**int data;**

**};**

**// Create and return a pointer of new Node.**

**Node \*createNode(int x)**

**{**

**Node \*p = new Node;**

**p -> data = x;**

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

**return p;**

**}**

**// Insert a new Node in Binary Search Tree.**

**void insertNode(struct Node \*root, int x)**

**{**

**Node \*p = root, \*q = NULL;**

**while (p != NULL)**

**{**

**q = p;**

**if (p -> data < x)**

**p = p -> right;**

**else**

**p = p -> left;**

**}**

**if (q == NULL)**

**p = createNode(x);**

**else**

**{**

**if (q -> data < x)**

**q -> right = createNode(x);**

**else**

**q -> left = createNode(x);**

**}**

**}**

**// Return the maximum element between a Node and its given ancestor.**

**int maxelpath(Node \*q, int x)**

**{**

**Node \*p = q;**

**int mx = INT\_MIN;**

**// Traversing the path between ancestor and Node and finding maximum element.**

**while (p -> data != x)**

**{**

**if (p -> data > x)**

**{**

**mx = max(mx, p -> data);**

**p = p -> left;**

**}**

**else**

**{**

**mx = max(mx, p -> data);**

**p = p -> right;**

**}**

**}**

**return max(mx, x);**

**}**

**// Return maximum element in the path between two given Node of BST.**

**int maximumElement(struct Node \*root, int x, int y)**

**{**

**Node \*p = root;**

**// Finding the LCA of Node x and Node y**

**while ((x < p -> data && y < p -> data) ||**

**(x > p -> data && y > p -> data))**

**{**

**// Checking if both the Node lie on the left side of the parent p.**

**if (x < p -> data && y < p -> data)**

**p = p -> left;**

**// Checking if both the Node lie on the right side of the parent p.**

**else if (x > p -> data && y > p -> data)**

**p = p -> right;**

**}**

**// Return the maximum of maximum elements occur in path from ancestor to both Node.**

**return max(maxelpath(p, x), maxelpath(p, y));**

**}**

**int main()**

**{**

**int arr[] = { 18, 36, 9, 6, 12, 10, 1, 8 };**

**int a = 1, b = 10;**

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

**// Creating the root of Binary Search Tree**

**struct Node \*root = createNode(arr[0]);**

**// Inserting Nodes in Binary Search Tree**

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

**insertNode(root, arr[i]);**

**cout << maximumElement(root, a, b) << endl;**

**return 0;**

**}**