Daa lab 1 solutions

Roll number 55

Reg no. 230957162

L0 - Prerequisite Lab Exercise Questions (Related to Data structures): 1. Given a sorted array, write a program to implement binary search and return the index of the target element. If the target is not found, return -1. 2. Implement a stack using an array. Include functions to perform the following operations: • Push an element onto the stack • Pop an element from the stack • Display the top element • Check if the stack is empty 3. Implement a queue using an array. Include functions for the following operations: • Enqueue an element • Dequeue an element • Display the front element • Check if the queue is empty. 4. Implement a doubly linked list and perform the following operations: • Insert a node at the beginning and end • Delete a node from the beginning and end • Traverse the list in forward and backward directions 5. Implement a graph using an adjacency matrix. Write functions for: • Adding an edge • Removing an edge • Displaying the graph 6. Implement a binary search tree (BST) and write functions for: • Inserting a node • Deleting a node • Searching for a node

#include <iostream>

using namespace std;

// 1. Binary Search Implementation

int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid; // Target found at index mid

}

if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1; // Target not found

}

// 2. Stack Implementation Using Array

class Stack {

private:

int arr[100]; // Array to hold stack elements

int top; // Index of the top element

public:

Stack() : top(-1) {}

void push(int x) {

if (top < 99) {

arr[++top] = x;

} else {

cout << "Stack overflow" << endl;

}

}

void pop() {

if (top >= 0) {

top--;

} else {

cout << "Stack underflow" << endl;

}

}

int peek() {

if (top >= 0) {

return arr[top];

} else {

cout << "Stack is empty" << endl;

return -1;

}

}

bool isEmpty() {

return top == -1;

}

};

// 3. Queue Implementation Using Array

class Queue {

private:

int arr[100]; // Array to hold queue elements

int front, rear; // Front and rear pointers

public:

Queue() : front(0), rear(0) {}

void enqueue(int x) {

if (rear < 100) {

arr[rear++] = x;

} else {

cout << "Queue is full" << endl;

}

}

void dequeue() {

if (front < rear) {

front++;

} else {

cout << "Queue is empty" << endl;

}

}

int frontElement() {

if (front < rear) {

return arr[front];

} else {

cout << "Queue is empty" << endl;

return -1;

}

}

bool isEmpty() {

return front == rear;

}

};

// 4. Doubly Linked List Implementation

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int val) : data(val), next(nullptr), prev(nullptr) {}

};

class DoublyLinkedList {

private:

Node\* head;

Node\* tail;

public:

DoublyLinkedList() : head(nullptr), tail(nullptr) {}

void insertAtBeginning(int val) {

Node\* newNode = new Node(val);

if (head == nullptr) {

head = tail = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

void insertAtEnd(int val) {

Node\* newNode = new Node(val);

if (tail == nullptr) {

head = tail = newNode;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

void deleteFromBeginning() {

if (head == nullptr) {

cout << "List is empty" << endl;

return;

}

Node\* temp = head;

head = head->next;

if (head) head->prev = nullptr;

delete temp;

}

void deleteFromEnd() {

if (tail == nullptr) {

cout << "List is empty" << endl;

return;

}

Node\* temp = tail;

tail = tail->prev;

if (tail) tail->next = nullptr;

delete temp;

}

void traverseForward() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

void traverseBackward() {

Node\* temp = tail;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->prev;

}

cout << endl;

}

};

// 5. Graph Implementation Using Adjacency Matrix

class Graph {

private:

int adjMatrix[5][5];

int vertices;

public:

Graph(int v) : vertices(v) {

for (int i = 0; i < vertices; i++) {

for (int j = 0; j < vertices; j++) {

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int u, int v) {

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1;

}

void removeEdge(int u, int v) {

adjMatrix[u][v] = 0;

adjMatrix[v][u] = 0;

}

void displayGraph() {

for (int i = 0; i < vertices; i++) {

for (int j = 0; j < vertices; j++) {

cout << adjMatrix[i][j] << " ";

}

cout << endl;

}

}

};

// 6. Binary Search Tree Implementation

class BSTNode {

public:

int data;

BSTNode\* left;

BSTNode\* right;

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

};

class BST {

private:

BSTNode\* root;

public:

BST() : root(nullptr) {}

BSTNode\* insert(BSTNode\* node, int val) {

if (node == nullptr) {

return new BSTNode(val);

}

if (val < node->data) {

node->left = insert(node->left, val);

} else {

node->right = insert(node->right, val);

}

return node;

}

BSTNode\* deleteNode(BSTNode\* root, int key) {

if (root == nullptr) return root;

if (key < root->data) {

root->left = deleteNode(root->left, key);

} else if (key > root->data) {

root->right = deleteNode(root->right, key);

} else {

if (root->left == nullptr) {

BSTNode\* temp = root->right;

delete root;

return temp;

} else if (root->right == nullptr) {

BSTNode\* temp = root->left;

delete root;

return temp;

}

BSTNode\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

BSTNode\* findMin(BSTNode\* node) {

while (node && node->left != nullptr) {

node = node->left;

}

return node;

}

bool search(BSTNode\* root, int key) {

if (root == nullptr) return false;

if (key == root->data) return true;

if (key < root->data) return search(root->left, key);

return search(root->right, key);

}

void insert(int val) {

root = insert(root, val);

}

void deleteNode(int key) {

root = deleteNode(root, key);

}

bool search(int key) {

return search(root, key);

}

};

// Main Program to test everything

int main() {

// 1. Binary Search Example

int arr[] = {1, 3, 5, 7, 9, 11};

int size = sizeof(arr) / sizeof(arr[0]);

int target = 7;

int result = binarySearch(arr, size, target);

cout << "Element found at index: " << result << endl;

// 2. Stack Example

Stack s;

s.push(10);

s.push(20);

cout << "Top element: " << s.peek() << endl;

s.pop();

cout << "Top element after pop: " << s.peek() << endl;

// 3. Queue Example

Queue q;

q.enqueue(5);

q.enqueue(10);

cout << "Front element: " << q.frontElement() << endl;

q.dequeue();

cout << "Front element after dequeue: " << q.frontElement() << endl;

// 4. Doubly Linked List Example

DoublyLinkedList dll;

dll.insertAtBeginning(10);

dll.insertAtEnd(20);

dll.insertAtBeginning(5);

dll.traverseForward();

dll.traverseBackward();

// 5. Graph Example

Graph g(5);

g.addEdge(0, 1);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.displayGraph();

// 6. BST Example

BST tree;

tree.insert(50);

tree.insert(30);

tree.insert(70);

tree.deleteNode(30);

cout << "Search 70 in BST: " << (tree.search(70) ? "Found" : "Not Found") << endl;

return 0;

}

Element found at index: 3

Top element: 20

Top element after pop: 10

Front element: 5

Front element after dequeue: 10

10 5 20

20 5 10

0 1 0 0 0

1 0 0 0 0

0 1 0 0 0

0 0 0 1 0

0 0 0 0 0

Search 70 in BST: Found

#include <iostream>

#include <string>

using namespace std;

week1

Lab Exercise 1: 1) Write an algorithm for finding the Greatest Common Divisor (GCD) of two numbers using Euclid’s algorithm. Determine the time and space complexity of the algorithm. Write a program to implement the Euclid’s algorithm. 2) Write a program to find GCD using consecutive integer checking method and analyze its time efficiency. 3) Write a program to sort a set of integers using selection sort and bubble sort algorithms and analyze their time efficiencies. 4) Write a program to implement brute-force string matching. Analyse its time efficiency.

// 1) Euclid's Algorithm for GCD

int gcdEuclid(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

// 2) GCD using Consecutive Integer Checking

int gcdConsecutive(int a, int b) {

int minVal = (a < b) ? a : b;

for (int i = minVal; i > 0; --i) {

if (a % i == 0 && b % i == 0) {

return i;

}

}

return 1;

}

// 3) Selection Sort

void selectionSort(int arr[], int size) {

for (int i = 0; i < size - 1; i++) {

int minIdx = i;

for (int j = i + 1; j < size; j++) {

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

swap(arr[i], arr[minIdx]);

}

}

// 4) Bubble Sort

void bubbleSort(int arr[], int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

// 5) Brute-Force String Matching

int bruteForceStringMatch(string text, string pattern) {

int m = text.length();

int n = pattern.length();

for (int i = 0; i <= m - n; i++) {

int j = 0;

while (j < n && text[i + j] == pattern[j]) {

j++;

}

if (j == n) {

return i; // Match found at index i

}

}

return -1; // No match found

}

int main() {

// GCD using Euclid's Algorithm

int a = 56, b = 98;

cout << "1) GCD of " << a << " and " << b << " using Euclid's Algorithm: " << gcdEuclid(a, b) << endl;

// GCD using Consecutive Integer Checking

cout << "2) GCD of " << a << " and " << b << " using Consecutive Integer Checking: " << gcdConsecutive(a, b) << endl;

// Selection Sort Example

int arr1[] = {64, 25, 12, 22, 11};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

selectionSort(arr1, size1);

cout << "3) Selection Sort: ";

for (int i = 0; i < size1; i++) {

cout << arr1[i] << " ";

}

cout << endl;

// Bubble Sort Example

int arr2[] = {64, 25, 12, 22, 11};

int size2 = sizeof(arr2) / sizeof(arr2[0]);

bubbleSort(arr2, size2);

cout << "4) Bubble Sort: ";

for (int i = 0; i < size2; i++) {

cout << arr2[i] << " ";

}

cout << endl;

// Brute-Force String Matching

string text = "ABABDABACDABABCABAB";

string pattern = "ABABCABAB";

int matchIndex = bruteForceStringMatch(text, pattern);

if (matchIndex != -1) {

cout << "5) Pattern found at index: " << matchIndex << endl;

} else {

cout << "5) Pattern not found" << endl;

}

return 0;

}

1) GCD of 56 and 98 using Euclid's Algorithm: 14

2) GCD of 56 and 98 using Consecutive Integer Checking: 14

3) Selection Sort: 11 12 22 25 64

4) Bubble Sort: 11 12 22 25 64

5) Pattern found at index: 10