1. A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Bubble Sort to arrange the following IDs:

[5, 4, 3, 2, 1]

#include <iostream>

using namespace std;

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

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

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

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

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int package\_ids[] = {5, 4, 3, 2, 1};

int n = sizeof(package\_ids) / sizeof(package\_ids[0]);

cout << "Before sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

// Call the bubble sort function

bubbleSort(package\_ids, n);

cout << "After sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

return 0;

}

1. A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Insertion Sort to arrange the following IDs:

[5,4,3,2,1]

#include <iostream>

using namespace std;

void insertionSort(int arr[], int n) {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key; // Insert the key at the correct position

}

}

int main() {

int package\_ids[] = {5, 4, 3, 2, 1};

int n = sizeof(package\_ids) / sizeof(package\_ids[0]);

cout << "Before sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

insertionSort(package\_ids, n);

cout << "After sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

return 0;

}

1. A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order. Write a program using Selection Sort to arrange the following IDs:

[5,4,3,2,1]

#include <iostream>

using namespace std;

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

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

int min\_index = i;

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

if (arr[j] < arr[min\_index]) {

min\_index = j;

}

}

int temp = arr[i];

arr[i] = arr[min\_index];

arr[min\_index] = temp;

}

}

int main() {

int package\_ids[] = {5, 4, 3, 2, 1};

int n = sizeof(package\_ids) / sizeof(package\_ids[0]);

cout << "Before sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

// Sort using Selection Sort

selectionSort(package\_ids, n);

cout << "After sorting: ";

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

cout << package\_ids[i] << " ";

}

cout << endl;

return 0;

}

1. A hospital management system stores patient IDs in a linked list to maintain their admission order. You are given the following sequence of patient IDs:

111 → 123 → 124 → NULL

Write a program to create and display this linked list.

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* createNode(int value) {

Node\* newNode = new Node();

newNode->data = value;

newNode->next = nullptr;

return newNode;

}

void displayList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

cout << current->data;

if (current->next != nullptr)

cout << " → ";

current = current->next;

} cout << " → NULL" << endl;

}

int main() {

Node\* head = createNode(111);

head->next = createNode(123);

head->next->next = createNode(124);

cout << "Patient ID Linked List:" << endl;

displayList(head);

return 0;

}

1. A social networking app wants to represent user connections as a graph, where each user is a node and friendships are edges between them. Given a graph showing user connections, create the adjacency list representation for it. Create the adjacency matrix for the given graph.

#include <iostream>

#include <vector>

using namespace std;

const int NUM\_NODES = 6;

void createGraph(vector<int> adjList[]) {

// Based on the graph image:

adjList[0] = {1, 4, 5}; // Node 1 connected to 2, 5, 6

adjList[1] = {0, 2, 4}; // Node 2 connected to 1, 3, 5

adjList[2] = {1, 3, 4}; // Node 3 connected to 2, 4, 5

adjList[3] = {2, 4}; // Node 4 connected to 3, 5

adjList[4] = {0, 1, 2, 3, 5}; // Node 5 connected to 1,2,3,4,6

adjList[5] = {0, 4}; // Node 6 connected to 1, 5

}

void displayAdjList(const vector<int> adjList[]) {

cout << "Adjacency List:\n";

for (int i = 0; i < NUM\_NODES; ++i) {

cout << i + 1 << ": ";

for (int neighbor : adjList[i]) {

cout << neighbor + 1 << " ";

}

cout << endl;

}

}

void createAndDisplayAdjMatrix(const vector<int> adjList[]) {

int matrix[NUM\_NODES][NUM\_NODES] = {0};

// Fill the adjacency matrix

for (int i = 0; i < NUM\_NODES; ++i) {

for (int neighbor : adjList[i]) {

matrix[i][neighbor] = 1;

}

}

// Display the matrix

cout << "\nAdjacency Matrix:\n ";

for (int i = 0; i < NUM\_NODES; ++i) {

cout << i + 1 << " ";

}

cout << "\n";

for (int i = 0; i < NUM\_NODES; ++i) {

cout << i + 1 << ": ";

for (int j = 0; j < NUM\_NODES; ++j) {

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

}

cout << endl;

}

}

int main() {

vector<int> adjList[NUM\_NODES];

createGraph(adjList);

displayAdjList(adjList);

createAndDisplayAdjMatrix(adjList);

return 0;

}

1. A university’s examination system stores student roll numbers in a binary tree for efficient searching. Given the structure of the tree, implement and display the binary tree. 50

/ \

30 70

/ \ /

20 40 60

#include <iostream>

using namespace std;

// Define structure for a tree node

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

// Insert node into BST

Node\* insert(Node\* root, int value) {

if (root == nullptr) {

return new Node(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

} else {

root->right = insert(root->right, value);

}

return root;

}

// Inorder traversal (left, root, right)

void inorder(Node\* root) {

if (root == nullptr) return;

inorder(root->left);

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

inorder(root->right);

}

// Optional: Preorder and Postorder

void preorder(Node\* root) {

if (root == nullptr) return;

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

preorder(root->left);

preorder(root->right);

}

void postorder(Node\* root) {

if (root == nullptr) return;

postorder(root->left);

postorder(root->right);

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

}

int main() {

Node\* root = nullptr;

// Insert nodes as per the given structure

root = insert(root, 50);

insert(root, 30);

insert(root, 70);

insert(root, 20);

insert(root, 40);

insert(root, 60);

cout << "Inorder Traversal (Sorted Order): ";

inorder(root);

cout << endl;

cout << "Preorder Traversal: ";

preorder(root);

cout << endl;

cout << "Postorder Traversal: ";

postorder(root);

cout << endl;

return 0;

}

1. An online library wants to store book IDs efficiently using hashing. The hash function used is:

h(key) = key % table\_size

If the book IDs are [1, 2, 3, 4] and the hash table size is 3, insert the keys into the hash table and show the final table representation.

#include <iostream>

using namespace std;

int main() {

int tableSize = 3;

int hashTable[3];

// Initialize hash table with -1 (empty)

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

hashTable[i] = -1;

}

int bookIDs[] = {1, 2, 3, 4};

int n = 4;

cout << "Inserting book IDs using hash function h(key) = key % " << tableSize << endl;

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

int key = bookIDs[i];

int index = key % tableSize;

// Linear probing in case of collision

while (hashTable[index] != -1) {

index = (index + 1) % tableSize;

}

hashTable[index] = key;

}

cout << "\nFinal Hash Table Representation:\n";

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

cout << "Index " << i << " -> ";

if (hashTable[i] == -1)

cout << "Empty";

else

cout << hashTable[i];

cout << endl;

}

return 0;

}

1. A city traffic control system represents road connections between intersections as a graph, where each intersection is a node and roads are edges. Given the graph, create the adjacency matrix representation for it.

#include <iostream>

using namespace std;

int main() {

const int n = 6; // number of nodes

int adj[n+1][n+1] = {0};

// Add edges (undirected graph)

int edges[][2] = {

{1, 2}, {1, 5}, {1, 6},

{2, 3}, {2, 5},

{3, 4}, {3, 5},

{4, 5}, {5, 6}

};

int totalEdges = sizeof(edges) / sizeof(edges[0]);

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

int u = edges[i][0];

int v = edges[i][1];

adj[u][v] = 1;

adj[v][u] = 1; // because undirected

}

// Print adjacency matrix

cout << "Adjacency Matrix Representation:\n";

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

for (int j = 1; j <= n; j++) {

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

}

cout << endl;

}

return 0;

}

Q.1) When a king grew old, he decided to divide his kingdoms and wealth between his two sons. He listed all the kingdoms and their respective values on a piece of paper in a sequence.

He announced that the division would take place pairwise from both ends of the list.

That means, in each round, there will be two kingdoms — one from the left end (value = x) and one from the right end (value = y).

If the elder son chooses x, then the younger son will automatically get y, and vice versa.

Since the king did not want to be unjust but loved his elder son more, he gave him the first choice to select a kingdom in each round.

Now, write a program to determine how the elder son should choose items so that he always remains in maximum benefit.

The program should also display:

The kingdoms chosen by both sons, and

Their total wealth.

#include <iostream>

using namespace std;

int main() {

int n;

cout << "Enter number of kingdoms: ";

cin >> n;

int a[100]; // assuming max 100 kingdoms

cout << "Enter the value of each kingdom:\n";

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

cin >> a[i];

}

int left = 0, right = n - 1;

int elder = 0, younger = 0;

int elderKingdoms[100], youngerKingdoms[100];

int e = 0, y = 0;

while (left <= right) {

if (a[left] >= a[right]) {

elderKingdoms[e++] = a[left];

elder += a[left];

youngerKingdoms[y++] = a[right];

younger += a[right];

left++;

right--;

} else {

elderKingdoms[e++] = a[right];

elder += a[right];

youngerKingdoms[y++] = a[left];

younger += a[left];

left++;

right--;

}

}

cout << "\nElder son's kingdoms: ";

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

cout << elderKingdoms[i] << " ";

}

cout << "\nYounger son's kingdoms: ";

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

cout << youngerKingdoms[i] << " ";

}

cout << "\n\nElder son's total wealth: " << elder;

cout << "\nYounger son's total wealth: " << younger << endl;

return 0;

}