LAB PROGRAMS: (SHARMILA.N – 192311330)

**1.Write a program in C to read n number of values in an array and display them in reverse order.**

**Test Data :**

**Input the number of elements to store in the array :3**

**The values store into the array are : 2 5 7**

**The values store into the array in reverse are : 7 5 2**

PROGRAM:

#include <stdio.h>

int main() {

int n;

printf("Input the number of elements to store in the array: ");

scanf("%d", &n);

int arr[n];

printf("Input %d number of elements in the array:\n", n);

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

printf("element - %d : ", i);

scanf("%d", &arr[i]);

}

printf("The values stored into the array are:\n");

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

printf("%d ", arr[i]);

}

printf("\n");

printf("The values stored into the array in reverse are:\n");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**2. Implement a C Program for AVL tree and perform Insertion and Deletion of Nodes**

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int key;

struct Node \*left;

struct Node \*right;

int height;

} Node;

int height(Node \*node) {

if (node == NULL)

return 0;

return node->height;

}

int max(int a, int b) {

return (a > b) ? a : b;

}

Node \*newNode(int key) {

Node \*node = (Node \*)malloc(sizeof(Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return node;

}

Node \*rightRotate(Node \*y) {

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node \*leftRotate(Node \*x) {

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

int getBalance(Node \*node) {

if (node == NULL)

return 0;

return height(node->left) - height(node->right);

}

Node \*insert(Node \*node, int key) {

if (node == NULL)

return newNode(key);

if (key < node->key)

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

else if (key > node->key)

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

else

return node;

node->height = 1 + max(height(node->left), height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rightRotate(node);

if (balance < -1 && key > node->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

Node \*minValueNode(Node \*node) {

Node \*current = node;

while (current->left != NULL)

current = current->left;

return current;

}

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

if (root == NULL)

return root;

if (key < root->key)

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

else if (key > root->key)

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

else {

if ((root->left == NULL) || (root->right == NULL)) {

Node \*temp = root->left ? root->left : root->right;

if (temp == NULL) {

temp = root;

root = NULL;

} else

\*root = \*temp;

free(temp);

} else {

Node \*temp = minValueNode(root->right);

root->key = temp->key;

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

}

}

if (root == NULL)

return root;

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void inOrder(Node \*root) {

if (root != NULL) {

inOrder(root->left);

printf("%d ", root->key);

inOrder(root->right);

}

}

int main() {

Node \*root = NULL;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

printf("Inorder traversal of the AVL tree:\n");

inOrder(root);

printf("\n");

root = deleteNode(root, 30);

printf("Inorder traversal after deletion:\n");

inOrder(root);

printf("\n");

return 0;

}

**3.Implement a C Program to Check for a valid String**

PROGRAM:

#include <stdio.h>

#include <string.h>

int isAlpha(char c) {

return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');

}

int isValidString(const char \*str) {

if (str == NULL || strlen(str) == 0)

return 0;

for (int i = 0; str[i] != '\0'; i++) {

if (!isAlpha(str[i]))

return 0;

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

size\_t len = strlen(str);

if (len > 0 && str[len - 1] == '\n') {

str[len - 1] = '\0';

}

if (isValidString(str))

printf("The string is valid.\n");

else

printf("The string is not valid.\n");

return 0;

}

**4. Implement a C Program whether it is a Valid stack**

**Input: pushed = { 1, 2, 3, 4, 5 }, popped = { 4, 5, 3, 2, 1 }**

**Output: True**

PROGRAM:

#include <stdio.h>

#define MAX 100

int validateStackSequences(int pushed[], int pushedSize, int popped[], int poppedSize) {

int stack[MAX];

int top = -1;

int popIndex = 0;

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

stack[++top] = pushed[i];

while (top >= 0 && stack[top] == popped[popIndex]) {

top--;

popIndex++;

}

}

return top == -1 ? 1 : 0;

}

int main() {

int pushed[] = {1, 2, 3, 4, 5};

int popped[] = {4, 5, 3, 2, 1};

int pushedSize = 5;

int poppedSize = 5;

if (validateStackSequences(pushed, pushedSize, popped, poppedSize))

printf("Output: True\n");

else

printf("Output: False\n");

return 0;

}

**5. Implement a C Program to Merge two Arrays**

**Input:**

**arr1 = [1, 2, 3, 4, 5]**

**arr2 = [6, 7, 8, 9, 10]**

**Output:**

**arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

PROGRAM:

#include <stdio.h>

#define MAX 100

void mergeArrays(int arr1[], int size1, int arr2[], int size2, int arr3[]) {

int i, j;

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

arr3[i] = arr1[i];

}

for (j = 0; j < size2; j++) {

arr3[i++] = arr2[j];

}

}

int main() {

int arr1[] = {1, 2, 3, 4, 5};

int arr2[] = {6, 7, 8, 9, 10};

int arr3[MAX];

int size1 = 5;

int size2 = 5;

int size3 = size1 + size2;

mergeArrays(arr1, size1, arr2, size2, arr3);

printf("Merged array: ");

for (int i = 0; i < size3; i++) {

printf("%d ", arr3[i]);

}

printf("\n");

return 0;

}

**6. Implement a C Program for Graph to Identify shortest path**

**Input :**

**Enter number of nodes:4**

**Enter weight of all the paths in adjacency matrix form**

**0 10 30 100**

**10 0 10 90**

**30 10 0 30**

**100 90 30 0**

**Enter the source:1**

**Enter the target:4**

**1 to 2 to 3 to 4**

**Output:**

**shortest path is 50**

PROGRAM:

#include <stdio.h>

#include <limits.h>

#define MAX 100

void dijkstra(int graph[MAX][MAX], int numNodes, int start, int end) {

int dist[MAX];

int visited[MAX];

int path[MAX];

int i, j, min, nextNode;

for (i = 0; i < numNodes; i++) {

dist[i] = INT\_MAX;

visited[i] = 0;

path[i] = -1;

}

dist[start - 1] = 0;

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

min = INT\_MAX;

for (j = 0; j < numNodes; j++) {

if (!visited[j] && dist[j] < min) {

min = dist[j];

nextNode = j;

}

}

visited[nextNode] = 1;

for (j = 0; j < numNodes; j++) {

if (!visited[j] && graph[nextNode][j] && dist[nextNode] != INT\_MAX &&

dist[nextNode] + graph[nextNode][j] < dist[j]) {

dist[j] = dist[nextNode] + graph[nextNode][j];

path[j] = nextNode;

}

}

}

printf("Shortest path is %d\n", dist[end - 1]);

printf("Path: ");

int node = end - 1;

int stack[MAX];

int top = -1;

while (node != -1) {

stack[++top] = node;

node = path[node];

}

while (top >= 0) {

printf("%d ", stack[top] + 1);

top--;

}

printf("\n");

}

int main() {

int graph[MAX][MAX];

int numNodes;

int source, target;

printf("Enter number of nodes\n");

scanf("%d", &numNodes);

printf("Enter weight of all the paths in adjacency matrix form\n");

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

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

scanf("%d", &graph[i][j]);

}

}

printf("Enter the source\n");

scanf("%d", &source);

printf("Enter the target\n");

scanf("%d", &target);

dijkstra(graph, numNodes, source, target);

return 0;

}

**7. Write a program in C to count the total number of duplicate elements in an array.**

**Test Data :**

**Input the number of elements to be stored in the array :3**

**Input 3 elements in the array :**

**element - 0 : 5**

**element - 1 : 1**

**element - 2 : 1**

**Expected Output :**

**Total number of duplicate elements found in the array is : 1**

PROGRAM:

#include <stdio.h>

#define MAX 100

int countDuplicates(int arr[], int size) {

int count = 0;

int visited[MAX] = {0};

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

if (visited[i])

continue;

int isDuplicate = 0;

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

if (arr[i] == arr[j]) {

isDuplicate = 1;

visited[j] = 1;

}

}

if (isDuplicate) {

count++;

}

}

return count;

}

int main() {

int arr[MAX];

int size;

printf("Input the number of elements to be stored in the array : ");

scanf("%d", &size);

printf("Input %d elements in the array :\n", size);

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

printf("element - %d : ", i);

scanf("%d", &arr[i]);

}

int duplicateCount = countDuplicates(arr, size);

printf("Total number of duplicate elements found in the array is : %d\n", duplicateCount);

return 0;

}

**8.Implement a C Program Traveling Salesman Problem to Identify shortest path**

**Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.**

PROGRAM:

#include <stdio.h>

#include <limits.h>

int tsp(int currentCity, int count, int cost, int startCity, int n, int dist[][n], int visited[]) {

if (count == n && dist[currentCity][startCity]) {

return cost + dist[currentCity][startCity];

}

int minCost = INT\_MAX;

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

if (!visited[i] && dist[currentCity][i]) {

visited[i] = 1;

int newCost = tsp(i, count + 1, cost + dist[currentCity][i], startCity, n, dist, visited);

minCost = (newCost < minCost) ? newCost : minCost;

visited[i] = 0;

}

}

return minCost;

}

int main() {

int n;

printf("Enter the number of cities: ");

scanf("%d", &n);

int dist[n][n];

int visited[n];

printf("Enter the distance matrix:\n");

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

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

scanf("%d", &dist[i][j]);

}

}

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

visited[i] = 0;

}

visited[0] = 1;

int result = tsp(0, 1, 0, 0, n, dist, visited);

printf("The shortest path has length: %d\n", result);

return 0;

}