**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“ Jnana Sangama ”, Belgaum -590014, Karnataka.**

****

## LAB REPORT

**on**

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

***Submitted by***

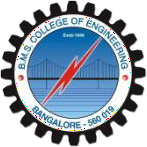
## Bhoomika B G (1BM23CS067)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

## COMPUTER SCIENCE AND ENGINEERING

****

**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

## BENGALURU-560019

**Feb-2025 to June-2025**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

****

**CERTIFICATE**

This is to certify that the Lab work entitled “ANALYSIS AND DESIGN OF ALGORITHMS” carried out by **Bhoomika B G (1BM23CS067),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)** work prescribed for the said degree.

**Prof. Anusha S Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

# Index Sheet

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1. | Write program to obtain the Topological ordering of vertices in a given digraph. | 5-6 |
| 2. | Implement Johnson Trotter algorithm to generate permutations. | 7-9 |
| 3. | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort. | 10 |
| 4. | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 11-12 |
| 5. | Sort a given set of N integer elements using Heap Sort technique and compute its time taken | 13-14 |
| 6. | Implement 0/1 Knapsack problem using dynamic programming. | 15 |
| 7. | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 16 |
| 8. | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.  Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm. | 17-20 |
| 9. | Implement Fractional Knapsack using Greedy technique. | 21-22 |
| 10. | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 23-24 |
| 11. | Implement “N-Queens Problem” using Backtracking. | 25-26 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems |

**Lab program 1:**

**Write program to obtain the Topological ordering of vertices in a given digraph.**

## CODE:

#include <stdio.h>

void main() { int n;

printf("Enter the number of vertices: "); scanf("%d", &n);

int adj[n][n], indegree[n], queue[n], front = 0, rear = -1, topological\_order[n], index = 0;

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

}

printf("Enter the Adjacency matrix: \n"); for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) { scanf("%d", &adj[i][j]); if (adj[i][j] == 1) {

indegree[j]++;

}

}

}

for (int i = 0; i < n; i++) { if (indegree[i] == 0) {

queue[++rear] = i;

}

}

while (front <= rear) {

int vertex = queue[front++]; topological\_order[index++] = vertex;

for (int i = 0; i < n; i++) { if (adj[vertex][i] == 1) {

indegree[i]--;

if (indegree[i] == 0) { queue[++rear] = i;

}

}

}

}

if (index != n) {

printf("There exists a cycle in the graph. Topological sorting is not possible.\n");

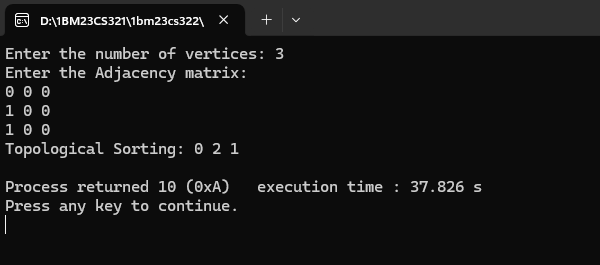
} else {

printf("Topological Sorting: "); for (int i = n-1; i >= 0; i--) {

printf("%d ", topological\_order[i]);

}

printf("\n");

}

**Lab program 2: Implement Johnson Trotter algorithm to generate permutations.**

## CODE:

## #include <stdio.h>

## #include <stdlib.h>

int co = 0;

int getMobile(int perm[], int dir[], int n) { int mobile = -1, mobileIndex = -1;

for (int i = 0; i < n; i++) { int swapWith = i + dir[i];

if (swapWith >= 0 && swapWith < n && perm[i] > perm[swapWith]) { if (perm[i] > mobile) {

mobile = perm[i]; mobileIndex = i;

}

}

}

return mobileIndex;

}

void printPermutations(int n) { int values[n], perm[n], dir[n];

printf("Enter numbers: "); for (int i = 0; i < n; i++) {

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

}

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

if (values[i] > values[j]) {

int temp = values[i]; values[i] = values[j]; values[j] = temp;

}

}

}

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

dir[i] = -1;

}

while (1) {

for (int i = 0; i < n; i++) printf("%d ", values[perm[i]]);

printf("\n"); co++;

int mobileIndex = getMobile(perm, dir, n); if (mobileIndex == -1) break;

int swapWith = mobileIndex + dir[mobileIndex];

int temp = perm[mobileIndex]; perm[mobileIndex] = perm[swapWith]; perm[swapWith] = temp;

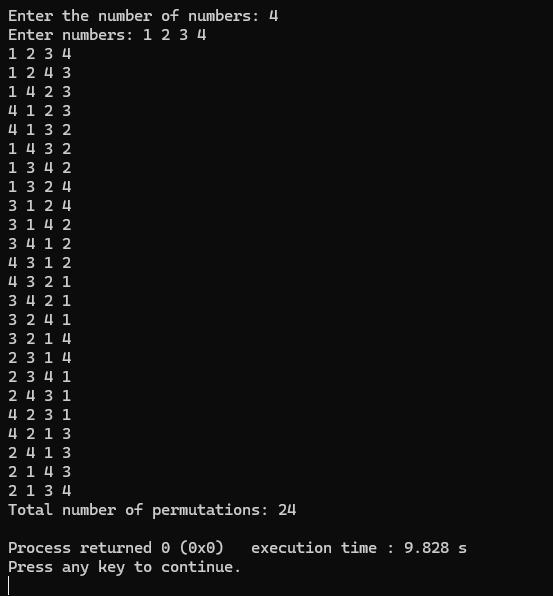
int tempDir = dir[mobileIndex]; dir[mobileIndex] = dir[swapWith]; dir[swapWith] = tempDir;

mobileIndex = swapWith;

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

if (perm[i] > perm[mobileIndex]) dir[i] \*= -1;

}

}

}

int main() {

int n;

printf("Enter the number of numbers: "); scanf("%d", &n);

printPermutations(n);

printf("Total number of permutations: %d\n", co); return 0;

}

**Lab program 3: Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.**

## CODE:

#include <stdio.h> #include <time.h>

void simplesort(int a[], int low, int mid, int high); void mergesort(int a[], int low, int high);

int main() {

clock\_t start, end;

int n;

printf("Enter the number of elements: "); scanf("%d", &n);

int a[n];

/\*printf("Enter the elements: "); for (int i = 0; i < n; i++) {

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

}

start = clock(); mergesort(a, 0, n - 1); end = clock();

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Sorted array:\n");

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

{ printf("%d ", a[i]);}

printf("\nTime taken: %lf seconds\n", time\_taken);

return 0;

}

void mergesort(int a[], int low, int high) { if (low < high) {

int mid = (low + high) / 2; mergesort(a, low, mid); mergesort(a, mid + 1, high); simplesort(a, low, mid, high);

}}

void simplesort(int a[], int low, int mid, int high) { int i = low, j = mid + 1, k = 0;

int temp[high - low + 1];

while (i <= mid && j <= high) { if (a[i] < a[j]) {

temp[k++] = a[i++];

} else {

temp[k++] = a[j++];

}}

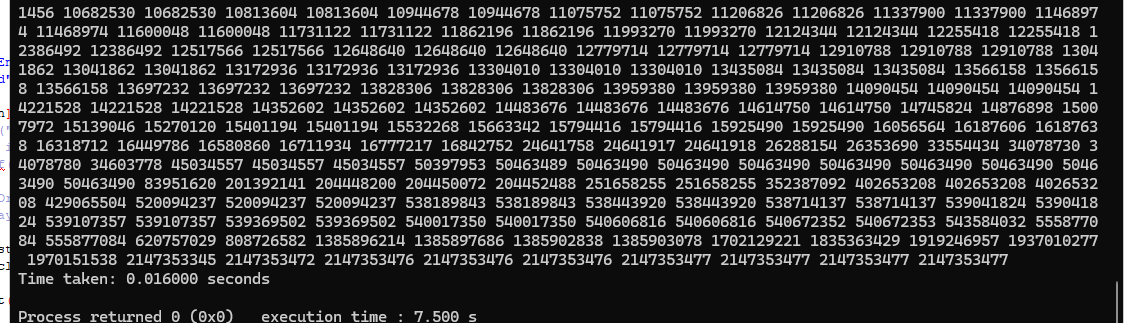
while (i <= mid) { temp[k++] = a[i++];}

while (j <= high) { temp[k++] = a[j++];}

for (i = low, k = 0; i <= high; i++, k++)

{ a[i] = temp[k];}

}



**Lab program 4: Sort a given set of N integer elements using Quick Sort technique and compute its time taken.**

## CODE:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap(int \*a, int \*b)

{ int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high]; int i = low - 1;

for (int j = low; j < high; j++) { if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}}

swap(&arr[i + 1], &arr[high]); return (i + 1);}

void quickSort(int arr[], int low, int high) { if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1); quickSort(arr, pi + 1, high);

}}

void printArray(int arr[], int size) { for (int i = 0; i < size; i++)

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

}

int main() { int n;

printf("Enter number of elements: "); scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n); for (int i = 0; i < n; i++)

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

printf("Original array: "); printArray(arr, n);

clock\_t start, end; start = clock();

quickSort(arr, 0, n - 1);

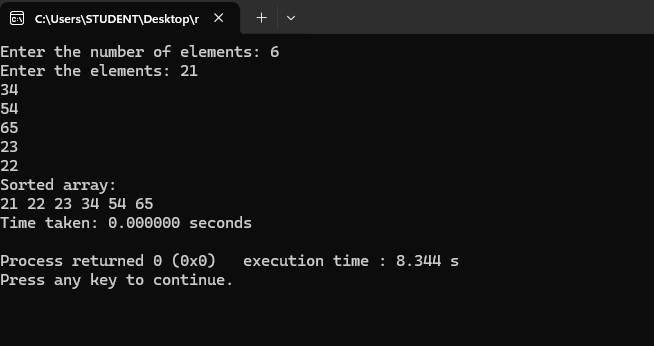
end = clock();

double time\_taken = ((double)(end - start)) / CLK\_TCK;

printf("Sorted array: "); printArray(arr, n);

printf("Time taken: %f seconds\n", time\_taken); return 0;

}



**Lab program 5: Sort a given set of N integer elements using Heap Sort technique and compute its time taken.**

## CODE;

#include <stdio.h> #include <stdlib.h> #include <time.h>

// Function to swap two elements void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Function to heapify the subtree rooted at index i void heapify(int arr[], int n, int i) {

int largest = i; // Initialize largest as root int left = 2 \* i + 1; // left = 2\*i + 1

int right = 2 \* i + 2; // right = 2\*i + 2

// If left child is larger than root

if (left < n && arr[left] > arr[largest]) { largest = left;

}

// If right child is larger than largest so far if (right < n && arr[right] > arr[largest]) {

largest = right;

}

// If largest is not root if (largest != i) {

swap(&arr[i], &arr[largest]);

// Recursively heapify the affected sub-tree heapify(arr, n, largest);

}

}

heapSort(int arr[], int n) {

// Build max heap

for (int i = n / 2 - 1; i >= 0; i--) { heapify(arr, n, i);

}

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

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

int main() { int N;

printf("Enter the number of elements: "); scanf("%d", &N);

int arr[N];

printf("Enter the elements:\n");

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

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

}

clock\_t start = clock();

heapSort(arr, N);

clock\_t end = clock();

double time\_taken = ((double)(end - start))

printf("Time taken for Heap Sort: %f seconds\n", time\_taken);

printf("Sorted array: "); for (int i = 0; i < N; i++) {

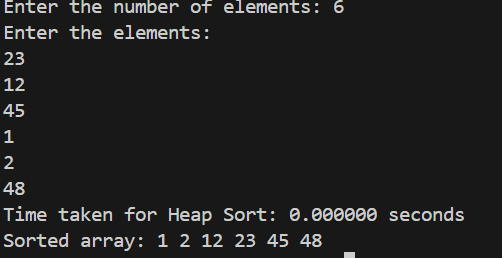
printf("%d ", arr[i]);

}

printf("\n");

return 0;

}



**Lab program 6: Implement 0/1 Knapsack problem using dynamic programming.**

#include <stdio.h>

int max(int a, int b) { return (a > b) ? a : b;}

int knapsack(int capacity, int weights[], int values[], int n) { int dp[n + 1][capacity + 1];

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

for (int w = 0; w <= capacity; w++) { if (i == 0 || w == 0)

dp[i][w] = 0;

else if (weights[i - 1] <= w)

dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]); else

dp[i][w] = dp[i - 1][w];}}

return dp[n][capacity];}

void main() {

int n, capacity;

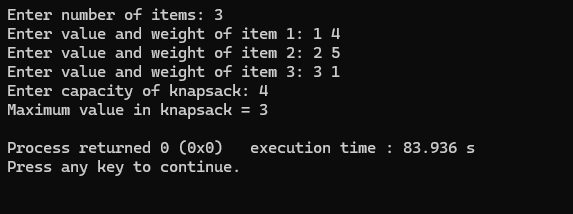
printf("Enter number of items: "); scanf("%d", &n);

int values[n], weights[n]; for (int i = 0; i < n; i++) {

printf("Enter value and weight of item %d: ", i + 1); scanf("%d %d", &values[i], &weights[i]);}

printf("Enter capacity of knapsack: "); scanf("%d", &capacity);

int maxValue = knapsack(capacity, weights, values, n); printf("Maximum value in knapsack = %d\n", maxValue);}



**Lab program 7: Implement All Pair Shortest paths problem using Floyd’s**

**algorithm.**

#include <stdio.h> #define INF 99999

void main(){ int V;

printf("Enter number of vertices: "); scanf("%d", &V);

int graph[V][V], dist[V][V];

printf("Enter the adjacency matrix (use %d for INF):\n", INF); for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++) scanf("%d",&graph[i][j];

for (int i = 0; i < V; i++) for (int j = 0; j < V;j++)

dist[i][j] = graph[i][j];

for (int k = 0; k < V; k++) for (int i = 0; i < V; i++)

for (int j = 0; j < V; j++)

if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

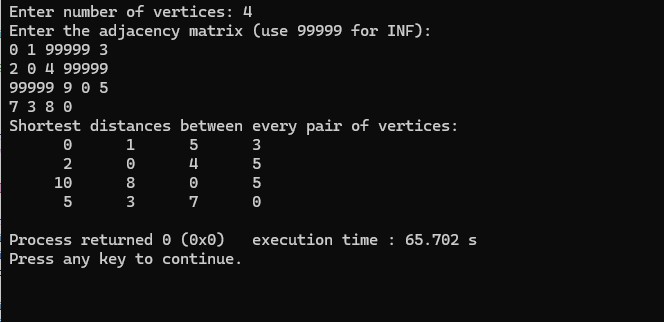
printf("Shortest distances between every pair of vertices:\n"); for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) { if (dist[i][j] == INF)

printf("%7s", "INF"); else

printf("%7d", dist[i][j]);}

printf("\n");}}



**Lab program 8:**

**Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s**

**algorithm.**

#include <stdio.h> #include <limits.h>

#define MAX\_V 100

int V;

int minKey(int key[], int mstSet[]) { int min = INT\_MAX, min\_index =1;

for (int v = 0; v < V; v++)

if (!mstSet[v] && key[v] < min) min = key[v], min\_index = v;

return min\_index;

}

void primMST(int graph[MAX\_V][MAX\_V]) {

int parent[MAX\_V], key[MAX\_V], mstSet[MAX\_V];

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = 0;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) { int u = minKey(key, mstSet);

mstSet[u] = 1;

for (int v = 0; v < V; v++) {

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) { parent[v] = u;

key[v] = graph[u][v];

}

}

}

int totalWeight = 0; printf("Edge \tWeight\n"); for (int i = 1; i < V; i++) {

printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);

totalWeight += graph[i][parent[i]];

}

printf("Total weight of MST: %d\n", totalWeight);

}

int main() {

int graph[MAX\_V][MAX\_V] = {0}; int E;

printf("Enter number of vertices: "); scanf("%d", &V);

printf("Enter number of edges: "); scanf("%d", &E);

printf("Enter edges as: source, destination, weight\n"); for (int i = 0; i < E; i++) {

int u, v, w;

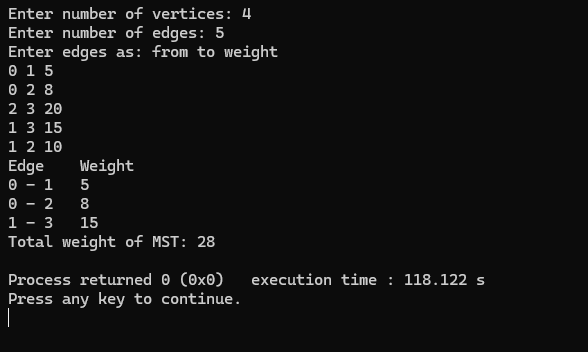
scanf("%d %d %d", &u, &v, &w); graph[u][v] = w;

graph[v][u] = w;

}

primMST(graph);

return 0;



#include <stdio.h> #include <stdlib.h>

struct Edge {

int src, dest, weight;

};

int \*parent;

int find(int i) {

while (i != parent[i]) i = parent[i];

return i;}

void unionSet(int x, int y) { int xset = find(x);

int yset = find(y); parent[xset] = yset;}

int compare(const void\* a, const void\* b) {

return ((struct Edge\*)a)->weight - ((struct Edge\*)b)->weight;}

int main() {

int V, E, totalWeight = 0;

printf("Enter the number of vertices and edges: "); scanf("%d %d", &V, &E);

struct Edge edges[E];

parent = (int\*)malloc(V \* sizeof(int));

printf("Enter the source edge, destination edge and the weight:\n");

for (int i = 0; i < E; i++)

scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);

for (int i = 0; i < V; i++) parent[i] = i;

qsort(edges, E, sizeof(edges[0]), compare);

printf("Edges in the MST:\n");

for (int i = 0, e = 0; e < V - 1 && i < E; i++) { int x = find(edges[i].src);

int y = find(edges[i].dest);

if (x != y) {

printf("%d - %d\t%d\n", edges[i].src, edges[i].dest, edges[i].weight); unionSet(x, y);

totalWeight += edges[i].weight; e++;

}

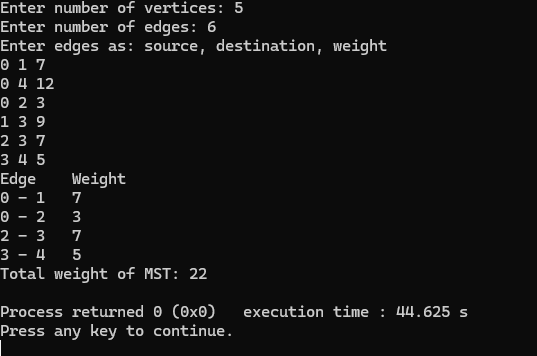
}

printf("Total weight of MST: %d\n", totalWeight);

free(parent);

return 0;

}



**Lab program 9:**

**Implement Fractional Knapsack using Greedy technique. CODE**

#include<stdio.h> #include <stdlib.h>

struct Item { int value; int weight;

};

int compare(const void \*a, const void \*b) {

double r1 = (double)((struct Item \*)a)->value / ((struct Item \*)a)->weight; double r2 = (double)((struct Item \*)b)->value / ((struct Item \*)b)->weight; return (r1 < r2) ? 1 : -1;}

double fractionalKnapsack(int capacity, struct Item items[], int n) { qsort(items, n, sizeof(struct Item), compare);

double totalValue = 0.0;

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

if (capacity >= items[i].weight) { capacity -= items[i].weight; totalValue += items[i].value;

} else {

totalValue += items[i].value \* ((double)capacity / items[i].weight); break;}}

return totalValue;}

int main() {

int n, capacity;

printf("Enter number of items: "); scanf("%d", &n);

struct Item items[n]; for (int i = 0; i < n; i++) {

printf("Enter value and weight of item %d: ", i + 1);

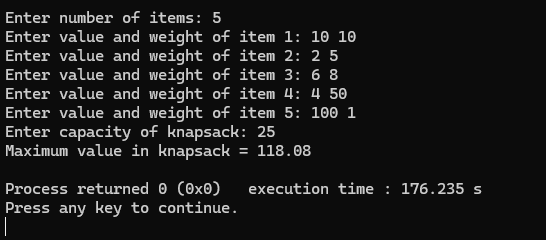
scanf("%d %d", &items[i].value, &items[i].weight);

}

printf("Enter capacity of knapsack: "); scanf("%d", &capacity);

double maxValue = fractionalKnapsack(capacity, items, n); printf("Maximum value in knapsack = %.2f\n", maxValue); return 0;

}



**Lab program 10:**

**From a given vertex in a weighted connected graph, find shortest paths to**

**other vertices using Dijkstra’s algorithm.**

## CODE

#include <stdio.h> #include <limits.h>

#define MAX 100

#define INF 99999

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

int dist[MAX], visited[MAX] = {0};

for (int i = 0; i < n; i++) { dist[i] = INF;

}

dist[start] = 0;

for (int count = 0; count < n - 1; count++) { int min = INF, u;

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

if (!visited[v] && dist[v] <= min) { min = dist[v];

u = v;

}

}

visited[u] = 1;

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

if (!visited[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];}}}

printf("Vertex\tDistance from Source %d\n", start); for (int i = 0; i < n; i++) {

printf("%d\t\t%d\n", i, dist[i]);

}

}

int main() {

int n, graph[MAX][MAX], start;

printf("Enter number of vertices: "); scanf("%d", &n);

printf("Enter the adjacency matrix (0 for no edge):\n"); for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) { scanf("%d", &graph[i][j]);

if (graph[i][j] == 0 && i != j) { graph[i][j] = INF;

}

}

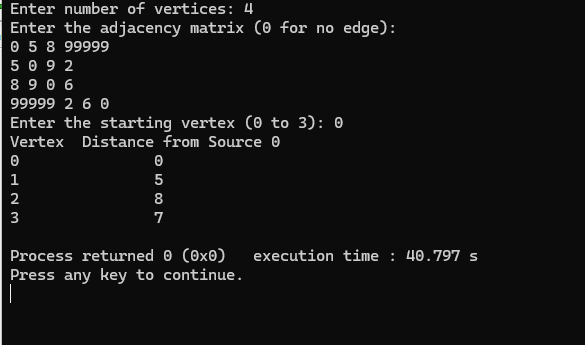
}

printf("Enter the starting vertex (0 to %d): ", n - 1); scanf("%d", &start);

dijkstra(graph, n, start);

return 0;

}



**Lab program 11:**

**implement “N-Queens Problem” using Backtracking.**

## CODE

#include <stdio.h> #include <stdbool.h>

#define N 4

int board[N];

void printSolution() {

for (int i = 0; i < N; i++) { for (int j = 0; j < N; j++) {

if (board[i] == j) printf("Q ");

else

printf(". ");

}

printf("\n");

}

printf("\n");

}

bool isSafe(int row, int col) { for (int i = 0; i < row; i++) {

if (board[i] == col ||)

board[i] - i == col - row ||

board[i] + i == col + row)

return false;

}

return true;

}

bool solveNQueens(int row) { if (row == N) {

printSolution(); // All queens placed successfully return true;

}

bool res = false;

for (int col = 0; col < N; col++) { if (isSafe(row, col)) {

board[row] = col; // Place queen

res = solveNQueens(row + 1) || res; // Recursive call

}

}

return res;

}

int main() {

printf("Solutions to %d-Queens problem:\n\n", N); if (!solveNQueens(0))

printf("No solution exists.\n");

return 0;

}