

**M.Sc. (Five Year Integrated) in Computer Science  
(Artificial Intelligence & Data Science)**

**Third Semester**

**Laboratory Record**

**21-805-0306: ALGORITHMS LAB**

*Submitted in partial fulfillment  
of the requirements for the award of degree in  
Master of Science (Five Year Integrated)  
in Computer Science (Artificial Intelligence & Data Science) of  
Cochin University of Science and Technology (CUSAT)  
Kochi*



*Submitted by*

**MARIYA BENNY  
(80521020)**

**DEPARTMENT OF COMPUTER SCIENCE  
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)  
KOCHI-682022**

**JANUARY 2023**

**DEPARTMENT OF COMPUTER SCIENCE**  
**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)**  
**KOCHI, KERALA-682022**



*This is to certify that the software laboratory record for **21-805-0306: Algorithms Lab** is a record of work carried out by **MARIYA BENNY (80521020)** in partial fulfillment of the requirements for the award of degree in **Master of Science (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)** of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the second semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

**Faculty Member in-charge**

Mrs. Raheena Salihin  
Guest Faculty  
Department of Computer Science  
CUSAT

Dr. Philip Samuel  
Professor and Head  
Department of Computer Science  
CUSAT

**Table of Contents**

<b>Sl.No.</b>	<b>Program</b>	<b>Pg.No.</b>
1	Program to use Quick sort to sort elements	01
2	Program to use Breadth First Search Tree	02
3	Program to implement Dijkstra's Algorithm	03
4	Program to implement Bellmann Ford Algorithm	04
5	Program to implement Floyd Warshall Algorithm	05
6	Program to implement Kruskal's Algorithm	06
7	Program to implement Prim's Algorithm	08
8	Program to implement Topological Sorting	10
9	Program to implement Matrix Multiplication Chain Rule	12
10	Program to implement Knapsack Problem	15
11	Program to implement Huffman Code	19
12	Program to implement Traveling Salesman Problem	20

## QUICK SORT

### AIM

To sort elements using quick sort.

### PROGRAM

```
#include<iostream>
#include <ctime>
#include <iomanip>
#include<cstdlib>
#include<chrono>
using namespace std;
using namespace std::chrono;
int Partition(int *A,int LB,int UB)
{
    int pivot = A[LB];
    int START = LB;
    int END = UB;
    while(START < END)
    {
        while(A[START] <= pivot)
        {
            START++;
        }
        while(A[END] > pivot)
        {
            END--;
        }
        if(START < END)
        {
            int temp = A[START];
            A[START] = A[END];
            A[END] = temp;
        }
    }
    int t1 = A[LB];
    A[LB] = A[END];
    A[END] = t1;
    return END;
}
```

```
}  
void QuickSort(int *A,int LB,int UB)  
{  
    if (LB < UB)  
    {  
        int LOC = Partition(A,LB,UB);  
        QuickSort(A,LB,LOC-1);  
        QuickSort(A,LOC+1,UB);  
    }  
}  
void display(int *A, int n)  
{  
    cout<<"The sorted list is : "<<" ";  
    for(int i = 0; i<n; i++)  
    {  
        cout<<A[i]<<" ";  
    }  
}  
  
int main()  
{  
    int n;  
    char choice;  
    do  
    {  
        cout<<"Enter the number of elements : "<<" ";  
        cin>>n;  
        int A[n];  
        int endpt;  
        cout<<"Enter the end point : "<<" ";  
        cin>>endpt;  
        for(int i = 0; i<n; i++)  
        {  
            A[i] = 1+rand()%endpt;  
        }  
        cout<<"The array is : "<<" ";  
        for(int i = 0; i<n; i++)  
        {  
            cout<<A[i]<<" ";  
        }  
        cout<<endl;
```

```
    int LB = 0;
    int UB = n;
    auto start = high_resolution_clock::now();
    QuickSort(A, LB, UB);
    auto stop = high_resolution_clock::now();
    auto doneTime = duration_cast<microseconds>(stop-start);
    //cout<< " You took " <<doneTime.count() << setprecision(8) << " nanoseconds\n";
    cout<< " You took " <<doneTime.count() << " nanoseconds\n";
    display(A,n);
    cout<<endl;
    cout<<"Do you want to continue(y/n)? : "<<" ";
    cin>>choice;
} while (choice!='n');
}
```

### SAMPLE INPUT-OUTPUT

```
Enter the number of elements : 10
Enter the end point : 100
The array is : 84 87 78 16 94 36 87 93 50 22
You took 0 nanoseconds
The sorted list is : 16 22 36 50 78 84 87 87 93 94
Do you want to continue(y/n)? : n

...Program finished with exit code 0
Press ENTER to exit console.
```

## BREADTH FIRST SEARCH TREE

### AIM

### PROGRAM

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
void add_edge(vector<int>adj[],int u,int v)
{
    adj[u].push_back(v);
}
void bfs(int source,vector<int>adj[],bool visited[])
{
    queue<int>q;
    q.push(source);
    visited[source] = true;
    while(!q.empty())
    {
        int u = q.front();
        cout<<u<<" ";
        q.pop();
        //Traversal
        for(int i = 0;i<adj[u].size();i++)
        {
            if(!visited[adj[u][i]])
            {
                q.push(adj[u][i]);
                visited[adj[u][i]] = true;
            }
        }
    }
}
int main()
{
    cout<<"-----BREADTH FIRST SEARCH-----"<<endl;
    int n,e;
    cout<<"Enter the no: of vertices : "<<" ";
    cin>>n;
    vector<int>adj[n];
```

```
bool visited[n];
for(int i = 0; i<5; i++)
{
    visited[i] = false;
}
cout<<"Enter the no: of edges    : "<<" ";
cin>>e;
int a,b,s;
for(int i = 0 ; i<e;i++)
{
    cout<<endl;
    cout<<"EDGE "<<i+1<<endl;
    cout<<"Enter the starting point : "<<" ";
    cin>>a;
    cout<<"Enter the final point    : "<<" ";
    cin>>b;
    add_edge(adj,a,b);
}
cout<<endl;
cout<<"Choose any vertex as the source : "<<" ";
cin>>s;
cout<<endl;
cout<<"BFS TRAVERSAL : "<<" ";
bfs(s,adj,visited);
cout<<endl;
}
```

### **SAMPLE INPUT-OUTPUT**



```
-----BREADTH FIRST SEARCH-----
Enter the no: of vertices : 5
Enter the no: of edges    : 5

EDGE 1
Enter the starting point : 0
Enter the final point    : 1

EDGE 2
Enter the starting point : 1
Enter the final point    : 3

EDGE 3
Enter the starting point : 3
Enter the final point    : 4

EDGE 4
Enter the starting point : 4
Enter the final point    : 2

EDGE 5
Enter the starting point : 2
Enter the final point    : 0

Choose any vertex as the source : 0

BFS TRAVERSAL : 0 1 3 4 2
```

## DJKSTRA'S ALGORITHM

### AIM

### PROGRAM

```
#include<iostream>
#include<stdio.h>
using namespace std;
#define INF 9999
#define V 5

void dijkstra(int G[V][V],int num,int start)
{
    int cost[V][V];
    int distance[V],pred[V];
    int visited[V],count,min_dist,next,i,j;
    for(i=0;i<num;i++)          //Assigning the values (initialisation)
    {
        for(j=0;j<num;j++)
        {
            if(G[i][j]==0)
            {
                cost[i][j]=INF;
            }
            else
            {
                cost[i][j]=G[i][j];
            }
        }
    }
    for(i=0;i<num;i++)
    {
        distance[i]=cost[start][i];
        pred[i]=start;
        visited[i] = 0;
    }
    distance[start] = 0;
    visited[start] = 1;
    count = 1;
    while(count < num-1)
    {
```

```
    min_dist=INF;
    for(i=0;i<num;i++)
    {
        if(distance[i]<min_dist && !visited[i])    //Initial checking for shortest path
        {
            min_dist=distance[i];
            next = i;
        }
    }
    visited[next] = 1;
    for(i=0;i<num;i++)
    {
        if(!visited[i])
        {
            if(min_dist+cost[next][i]<distance[i])    //Relax function
            {
                distance[i]=min_dist+cost[next][i];
                pred[i]=next;
            }
        }
    }
    count++;
}
cout<<endl;
cout<<"Vertex"<<"    "<<"Distance"<<endl<<endl;
for(i=0;i<num;i++)
{
    //if(i!=start)
    {
        cout<<i<<"    "<<distance[i]<<endl;
        cout<<endl;
    }
}
}

int main()
{
    int G[V][V];
    int source;
    for(int i = 0; i < V; i++)
    {
        cout<<"Enter the distance from vertex "<< i <<" to each vertex : "<<" ";
```

```
        for(int j = 0;j < V; j++)
        {
            cin>>G[i][j];
        }
    }
    cout<<endl<<endl;
    cout<<"Choose any vertex as source : "<<" ";
    cin>>source;
    dijkstra(G,V,source);
    return 0;
}
```

### SAMPLE INPUT-OUTPUT

```
Enter the distance from vertex 0 to each vertex : 0 1 2 4 2 3
Enter the distance from vertex 1 to each vertex : 0 4 5 7 1 3
Enter the distance from vertex 2 to each vertex : 1 3 2 4 5 0
Enter the distance from vertex 3 to each vertex : 0 2 4 0 1 5
Enter the distance from vertex 4 to each vertex : 2 1 3 4 5 0
```

```
Choose any vertex as source :
```

```
Vertex      Distance
```

```
0           3
```

```
1           0
```

```
2           4
```

```
3           5
```

```
4           5
```

## LAB CYCLE 3

**BELLMAN FORD****AIM****PROGRAM**

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

// Struct for the edges of the graph
struct Edge {
    int u; //start vertex of the edge
    int v; //end vertex of the edge
    int w; //w of the edge (u,v)
};

// Graph - it consists of edges
struct Graph {
    int V; // Total number of vertices in the graph
    int E; // Total number of edges in the graph
    struct Edge* edge; // Array of edges
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E) {
    struct Graph* graph = new Graph;
    graph->V = V; // Total Vertices
    graph->E = E; // Total edges

    // Array of edges for graph
    graph->edge = new Edge[E];
    return graph;
}

// Printing the solution
void printArr(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```
}

void BellmanFord(struct Graph* graph, int u) {
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    // Step 1: fill the distance array and predecessor array
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;

    // Mark the source vertex
    dist[u] = 0;

    // Step 2: relax edges |V| - 1 times
    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            // Get the edge data
            int u = graph->edge[j].u;
            int v = graph->edge[j].v;
            int w = graph->edge[j].w;
            if (dist[u] != INT_MAX && dist[u] + w < dist[v])
                dist[v] = dist[u] + w;
        }
    }

    // Step 3: detect negative cycle
    // if value changes then we have a negative cycle in the graph
    // and we cannot find the shortest distances
    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].u;
        int v = graph->edge[i].v;
        int w = graph->edge[i].w;
        if (dist[u] != INT_MAX && dist[u] + w < dist[v]) {
            printf("Graph contains negative w cycle");
            return;
        }
    }

    // No negative weight cycle found!
    // Print the distance and predecessor array
```

```
    printArr(dist, V);

    return;
}

int main() {
    // Create a graph
    int V = 5; // Total vertices
    int E = 8; // Total edges

    // Array of edges for graph
    struct Graph* graph = createGraph(V, E);

    //----- adding the edges of the graph
    /*
edge(u, v)
where u = start vertex of the edge (u,v)
v = end vertex of the edge (u,v)

w is the weight of the edge (u,v)
*/

    //edge 0 --> 1
    graph->edge[0].u = 0;
    graph->edge[0].v = 1;
    graph->edge[0].w = 5;

    //edge 0 --> 2
    graph->edge[1].u = 0;
    graph->edge[1].v = 2;
    graph->edge[1].w = 4;

    //edge 1 --> 3
    graph->edge[2].u = 1;
    graph->edge[2].v = 3;
    graph->edge[2].w = 3;

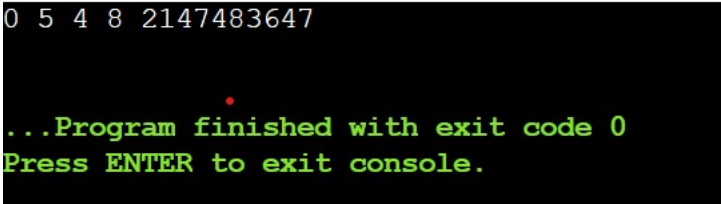
    //edge 2 --> 1
    graph->edge[3].u = 2;
    graph->edge[3].v = 1;
    graph->edge[3].w = 6;
```

```
//edge 3 --> 2
graph->edge[4].u = 3;
graph->edge[4].v = 2;
graph->edge[4].w = 2;

BellmanFord(graph, 0); //0 is the source vertex

return 0;
}
```

### SAMPLE INPUT-OUTPUT



A screenshot of a terminal window with a black background and green text. The first line shows the output of a program: "0 5 4 8 2147483647". The second line shows a red dot cursor. The third line says "...Program finished with exit code 0". The fourth line says "Press ENTER to exit console."

```
0 5 4 8 2147483647
...Program finished with exit code 0
Press ENTER to exit console.
```



## FLOYD WARSHALL ALGORITHM

### AIM

### PROGRAM

```
#include <iostream>
#include <conio.h>
using namespace std;
void floyds(int b[][7])
{
    int i, j, k;
    for (k = 0; k < 7; k++)
    {
        for (i = 0; i < 7; i++)
        {
            for (j = 0; j < 7; j++)
            {
                if ((b[i][k] * b[k][j] != 0) && (i != j))
                {
                    if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))
                    {
                        b[i][j] = b[i][k] + b[k][j];
                    }
                }
            }
        }
    }

    for (i = 0; i < 7; i++)
    {
        cout<<"\nMinimum Cost With Respect to Node:"<<i<<endl;
        for (j = 0; j < 7; j++)
        {
            cout<<b[i][j]<<"\t";
        }

        cout<<endl;
    }
}

int main()
{
    int b[7][7];
    cout<<"ENTER VALUES OF ADJACENCY MATRIX\n\n";
```

```
for (int i = 0; i < 7; i++)
{
    cout<<"enter values for "<<(i+1)<<" row"<<endl;
    for (int j = 0; j < 7; j++)
    {
        cin>>b[i][j];
    }
}
floyds(b);
getch();
}
```

### SAMPLE INPUT-OUTPUT

ENTER VALUES OF ADJACENCY MATRIX

enter values for 1 row

0 3 6 0 0 0 0

enter values for 2 row

3 0 2 4 0 0 0

enter values for 3 row

6 2 0 1 4 2 0

enter values for 4 row

0 4 1 0 2 0 4

enter values for 5 row

0 0 4 2 0 2 1

enter values for 6 row

0 0 2 0 2 0 1

enter values for 7 row

0 0 0 4 1 1 00 0

Minimum Cost With Respect to Node:0

0	3	5	6	8	7	8
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:1

3	0	2	3	5	4	5
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:2

5	2	0	1	3	2	3
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:3

6	3	1	0	2	3	3
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:4

8	5	3	2	0	2	1
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:5

7	4	2	3	2	0	1
---	---	---	---	---	---	---

Minimum Cost With Respect to Node:6

8	5	3	3	1	1	0
---	---	---	---	---	---	---

## KRUSKAL'S ALGORITHM

### AIM

### PROGRAM

```
// Kruskal's algorithm in C++

#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

#define edge pair<int, int>

class Graph {
    private:
        vector<pair<int, edge> > G; // graph
        vector<pair<int, edge> > T; // mst
        int *parent;
        int V; // number of vertices/nodes in graph
    public:
        Graph(int V);
        void AddWeightedEdge(int u, int v, int w);
        int find_set(int i);
        void union_set(int u, int v);
        void kruskal();
        void print();
};

Graph::Graph(int V) {
    parent = new int[V];

    //i 0 1 2 3 4 5
    //parent[i] 0 1 2 3 4 5
    for (int i = 0; i < V; i++)
        parent[i] = i;

    G.clear();
    T.clear();
}

void Graph::AddWeightedEdge(int u, int v, int w) {
    G.push_back(make_pair(w, edge(u, v)));
```

```
}

int Graph::find_set(int i) {
    // If i is the parent of itself
    if (i == parent[i])
        return i;
    else
        // Else if i is not the parent of itself
        // Then i is not the representative of his set,
        // so we recursively call Find on its parent
        return find_set(parent[i]);
}

void Graph::union_set(int u, int v) {
    parent[u] = parent[v];
}

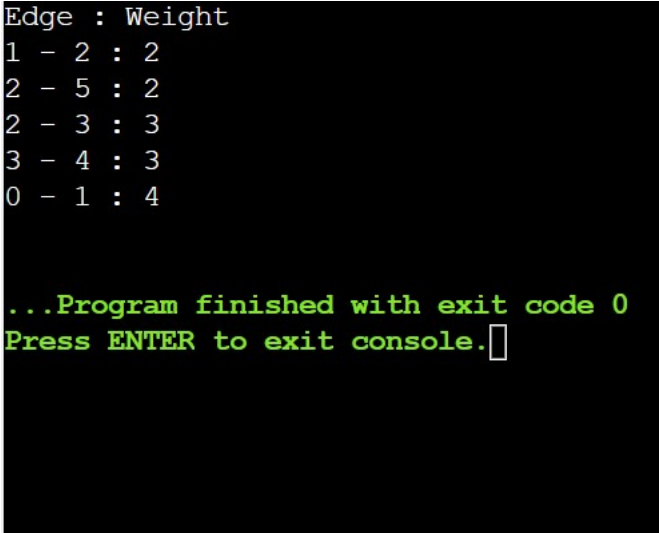
void Graph::kruskal() {
    int i, uRep, vRep;
    sort(G.begin(), G.end()); // increasing weight
    for (i = 0; i < G.size(); i++) {
        uRep = find_set(G[i].second.first);
        vRep = find_set(G[i].second.second);
        if (uRep != vRep) {
            T.push_back(G[i]); // add to tree
            union_set(uRep, vRep);
        }
    }
}

void Graph::print() {
    cout << "Edge : "
        << " Weight" << endl;
    for (int i = 0; i < T.size(); i++) {
        cout << T[i].second.first << " - " << T[i].second.second << " : "
            << T[i].first;
        cout << endl;
    }
}

int main() {
    Graph g(6);
    g.AddWeightedEdge(0, 1, 4);
    g.AddWeightedEdge(0, 2, 4);
    g.AddWeightedEdge(1, 2, 2);
}
```

```
g.AddWeightedEdge(1, 0, 4);
g.AddWeightedEdge(2, 0, 4);
g.AddWeightedEdge(2, 1, 2);
g.AddWeightedEdge(2, 3, 3);
g.AddWeightedEdge(2, 5, 2);
g.AddWeightedEdge(2, 4, 4);
g.AddWeightedEdge(3, 2, 3);
g.AddWeightedEdge(3, 4, 3);
g.AddWeightedEdge(4, 2, 4);
g.AddWeightedEdge(4, 3, 3);
g.AddWeightedEdge(5, 2, 2);
g.AddWeightedEdge(5, 4, 3);
g.kruskal();
g.print();
return 0;
}
```

#### SAMPLE INPUT-OUTPUT



```
Edge : Weight
1 - 2 : 2
2 - 5 : 2
2 - 3 : 3
3 - 4 : 3
0 - 1 : 4

...Program finished with exit code 0
Press ENTER to exit console.[]
```

## PRIM'S ALGORITHM

### AIM

### PROGRAM

```
// Prim's Algorithm in C++

#include <cstring>
#include <iostream>
using namespace std;

#define INF 9999999

// number of vertices in grapj
#define V 5

// create a 2d array of size 5x5
//for adjacency matrix to represent graph

int G[V][V] = {
    {0, 9, 75, 0, 0},
    {9, 0, 95, 19, 42},
    {75, 95, 0, 51, 66},
    {0, 19, 51, 0, 31},
    {0, 42, 66, 31, 0}};

int main() {
    int no_edge; // number of edge

    // create a array to track selected vertex
    // selected will become true otherwise false
    int selected[V];

    // set selected false initially
    memset(selected, false, sizeof(selected));

    // set number of edge to 0
    no_edge = 0;

    // the number of egde in minimum spanning tree will be
    // always less than (V -1), where V is number of vertices in
```

```
//graph

// choose 0th vertex and make it true
selected[0] = true;

int x; // row number
int y; // col number

// print for edge and weight
cout << "Edge"
    << " : "
    << "Weight";
cout << endl;
while (no_edge < V - 1) {
    //For every vertex in the set S, find the all adjacent vertices
    // , calculate the distance from the vertex selected at step 1.
    // if the vertex is already in the set S, discard it otherwise
    //choose another vertex nearest to selected vertex at step 1.

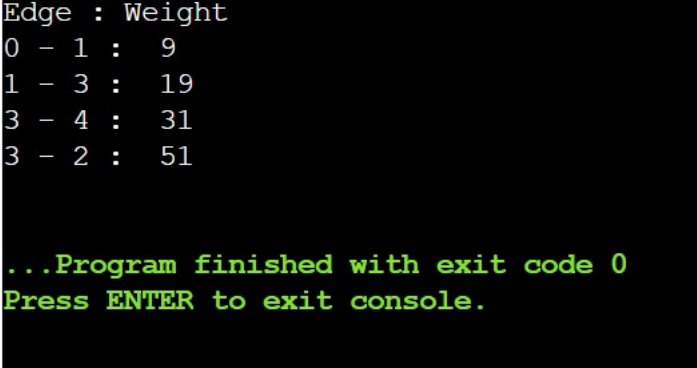
    int min = INF;
    x = 0;
    y = 0;

    for (int i = 0; i < V; i++) {
        if (selected[i]) {
            for (int j = 0; j < V; j++) {
                if (!selected[j] && G[i][j]) { // not in selected and there is an edge
                    if (min > G[i][j]) {
                        min = G[i][j];
                        x = i;
                        y = j;
                    }
                }
            }
        }
    }

    cout << x << " - " << y << " : " << G[x][y];
    cout << endl;
    selected[y] = true;
    no_edge++;
}
```

```
    return 0;  
}
```

### SAMPLE INPUT-OUTPUT



Edge : Weight  
0 - 1 : 9  
1 - 3 : 19  
3 - 4 : 31  
3 - 2 : 51  
  
...Program finished with exit code 0  
Press ENTER to exit console.



## TOPOLOGICAL SORTING

### AIM

### PROGRAM

```
#include<iostream>
using namespace std;

int main(){
int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

cout<<"Enter the no of vertices:\n";
cin>>n;

cout<<"Enter the adjacency matrix:\n";
for(i=0;i<n;i++){
cout<<"Enter row "<<i+1<<"\n";
for(j=0;j<n;j++)
cin>>a[i][j];
}

for(i=0;i<n;i++){
    indeg[i]=0;
    flag[i]=0;
}

for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        indeg[i]=indeg[i]+a[j][i];

cout<<"\nThe topological order is:";

while(count<n){
    for(k=0;k<n;k++){
        if((indeg[k]==0) && (flag[k]==0)){
            cout<<k+1<<" ";
            flag[k]=1;
        }

        for(i=0;i<n;i++){
            if(a[i][k]==1)
```

```
        indeg[k]--;  
    }  
}  
  
    count++;  
}  
  
    return 0;  
}
```

### SAMPLE INPUT-OUTPUT

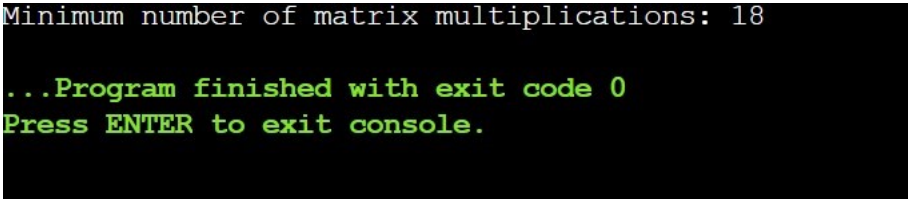
```
Enter the no of vertices:  
5  
Enter the adjacency matrix:  
Enter row 1  
0 1 1 1 0  
Enter row 2  
1 0 0 1 0  
Enter row 3  
1 0 0 0 1  
Enter row 4  
1 1 0 0 1  
Enter row 5  
0 0 1 1 0  
  
The topological order is:1 2 3 4 5  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

## LAB CYCLE 3

**MATRIX MULTIPLICATION CHAIN RULE****AIM****PROGRAM**

```
#include<iostream>
#include<climits>
using namespace std;
int matOrder(int array[], int n){
    int minMul[n][n]; //holds the number of scalar multiplication needed
    for (int i=1; i<n; i++)
        minMul[i][i] = 0; //for multiplication with 1 matrix, cost is 0
    for (int length=2; length<n; length++){ //find the chain length starting from 2
        for (int i=1; i<n-length+1; i++){
            int j = i+length-1;
            minMul[i][j] = INT_MAX; //set to infinity
            for (int k=i; k<=j-1; k++){
                //store cost per multiplications
                int q = minMul[i][k] + minMul[k+1][j] + array[i-1]*array[k]*array[j];
                if (q < minMul[i][j])
                    minMul[i][j] = q;
            }
        }
    }
    return minMul[1][n-1];
}

int main(){
    int arr[] = {1, 2, 3, 4};
    int size = 4;
    cout << "Minimum number of matrix multiplications: "<<matOrder(arr, size);
}
```

**SAMPLE INPUT-OUTPUT**A screenshot of a terminal window with a black background and green text. The text shows the output of the program: 'Minimum number of matrix multiplications: 18', followed by '...Program finished with exit code 0' and 'Press ENTER to exit console.' on separate lines.

```
Minimum number of matrix multiplications: 18
...Program finished with exit code 0
Press ENTER to exit console.
```

## KNAPSACK PROBLEM

### AIM

### PROGRAM

```
#include <iostream>
using namespace std;
int max(int x, int y) {
    return (x > y) ? x : y;
}
int knapSack(int W, int w[], int v[], int n) {
    int i, wt;
    int K[n + 1][W + 1];
    for (i = 0; i <= n; i++) {
        for (wt = 0; wt <= W; wt++) {
            if (i == 0 || wt == 0)
                K[i][wt] = 0;
            else if (w[i - 1] <= wt)
                K[i][wt] = max(v[i - 1] + K[i - 1][wt - w[i - 1]], K[i - 1][wt]);
            else
                K[i][wt] = K[i - 1][wt];
        }
    }
    return K[n][W];
}
int main() {
    cout << "Enter the number of items in a Knapsack:";
    int n, W;
    cin >> n;
    int v[n], w[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter value and weight for item " << i << ":";
        cin >> v[i];
        cin >> w[i];
    }
    cout << "Enter the capacity of knapsack";
    cin >> W;
    cout << knapSack(W, w, v, n);
    return 0;
}
```

### SAMPLE INPUT-OUTPUT

```
Enter the number of items in a Knapsack:4
Enter value and weight for item 0:10
50
Enter value and weight for item 1:20
60
Enter value and weight for item 2:30
70
Enter value and weight for item 3:40
90
Enter the capacity of knapsack100
40

...Program finished with exit code 0
Press ENTER to exit console.□
```

## HUFFMAN CODE

### AIM

### PROGRAM

```
// Huffman Coding in C++

#include <iostream>
using namespace std;

#define MAX_TREE_HT 50

struct MinHNode {
    unsigned freq;
    char item;
    struct MinHNode *left, *right;
};

struct MinH {
    unsigned size;
    unsigned capacity;
    struct MinHNode **array;
};

// Creating Huffman tree node
struct MinHNode *newNode(char item, unsigned freq) {
    struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));

    temp->left = temp->right = NULL;
    temp->item = item;
    temp->freq = freq;

    return temp;
}

// Create min heap using given capacity
struct MinH *createMinH(unsigned capacity) {
    struct MinH *minHeap = (struct MinH *)malloc(sizeof(struct MinH));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHNode **)malloc(minHeap->capacity * sizeof(struct MinHNode *));
}
```

```
    return minHeap;
}

// Print the array
void printArray(int arr[], int n) {
    int i;
    for (i = 0; i < n; ++i)
        cout << arr[i];

    cout << "\n";
}

// Swap function
void swapMinHNode(struct MinHNode **a, struct MinHNode **b) {
    struct MinHNode *t = *a;
    *a = *b;
    *b = t;
}

// Heapify
void minHeapify(struct MinH *minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx) {
        swapMinHNode(&minHeap->array[smallest],
                    &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

// Check if size is 1
int checkSizeOne(struct MinH *minHeap) {
    return (minHeap->size == 1);
}
```

```
}

// Extract the min
struct MinHNode *extractMin(struct MinH *minHeap) {
    struct MinHNode *temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];

    --minHeap->size;
    minHeapify(minHeap, 0);

    return temp;
}

// Insertion
void insertMinHeap(struct MinH *minHeap, struct MinHNode *minHeapNode) {
    ++minHeap->size;
    int i = minHeap->size - 1;

    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }

    minHeap->array[i] = minHeapNode;
}

// BUild min heap
void buildMinHeap(struct MinH *minHeap) {
    int n = minHeap->size - 1;
    int i;

    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

int isLeaf(struct MinHNode *root) {
    return !(root->left) && !(root->right);
}

struct MinH *createAndBuildMinHeap(char item[], int freq[], int size) {
    struct MinH *minHeap = createMinH(size);
```



```
for (int i = 0; i < size; ++i)
    minHeap->array[i] = newNode(item[i], freq[i]);

minHeap->size = size;
buildMinHeap(minHeap);

return minHeap;
}

struct MinHNode *buildHfTree(char item[], int freq[], int size) {
    struct MinHNode *left, *right, *top;
    struct MinH *minHeap = createAndBuildMinHeap(item, freq, size);

    while (!checkSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        top = newNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}

void printHCodes(struct MinHNode *root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printHCodes(root->left, arr, top + 1);
    }

    if (root->right) {
        arr[top] = 1;
        printHCodes(root->right, arr, top + 1);
    }

    if (isLeaf(root)) {
        cout << root->item << "  | ";
        printArray(arr, top);
    }
}
```

```
}

// Wrapper function
void HuffmanCodes(char item[], int freq[], int size) {
    struct MinHNode *root = buildHfTree(item, freq, size);

    int arr[MAX_TREE_HT], top = 0;

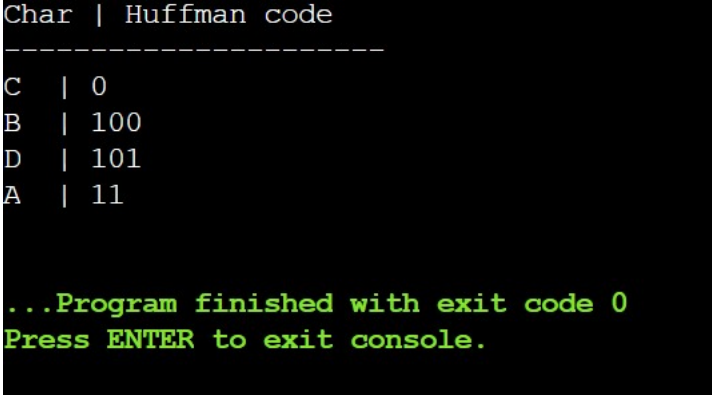
    printHCodes(root, arr, top);
}

int main() {
    char arr[] = {'A', 'B', 'C', 'D'};
    int freq[] = {5, 1, 6, 3};

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

    cout << "Char | Huffman code ";
    cout << "\n-----\n";
    HuffmanCodes(arr, freq, size);
}
```

### SAMPLE INPUT-OUTPUT



```
Char | Huffman code
-----
C   | 0
B   | 100
D   | 101
A   | 11

...Program finished with exit code 0
Press ENTER to exit console.
```

## TRAVELING SALESMAN PROBLEM

### AIM

### PROGRAM

```
#include <bits/stdc++.h>
using namespace std;
#define vr 4
int TSP(int grph[][vr], int p) // implement traveling Salesman Problem. {
    vector<int> ver; //
    for (int i = 0; i < vr; i++)
        if (i != p)
            ver.push_back(i);
    int m_p = INT_MAX; // store minimum weight of a graph
    do {
        int cur_pth = 0;
        int k = p;
        for (int i = 0; i < ver.size(); i++) {
            cur_pth += grph[k][ver[i]];
            k = ver[i];
        }
        cur_pth += grph[k][p];
        m_p = min(m_p, cur_pth); // to update the value of minimum weight
    }
    while (next_permutation(ver.begin(), ver.end()));
    return m_p;
}
int main() {
    int grph[][vr] = { { 0, 5, 10, 15 }, //values of a graph in a form of matrix
        { 5, 0, 20, 30 },
        { 10, 20, 0, 35 },
        { 15, 30, 35, 0 }
    };
    int p = 0;
    cout<< "\n The result is: "<< TSP(grph, p) << endl;
    return 0;
}
```

### SAMPLE INPUT-OUTPUT

```
The result is: 75
```