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

FARHA T A
(80521009)

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* ***FARHA T A (80521009)****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 third semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

Faculty Member in-charge

Mrs. Raheena Salihin	Dr. Philip Samuel
Guest Faculty	Professor and Head
Department of Computer Science	Department of Computer Science
CUSAT	CUSAT

Table of Contents

QUICK SORT

AIM

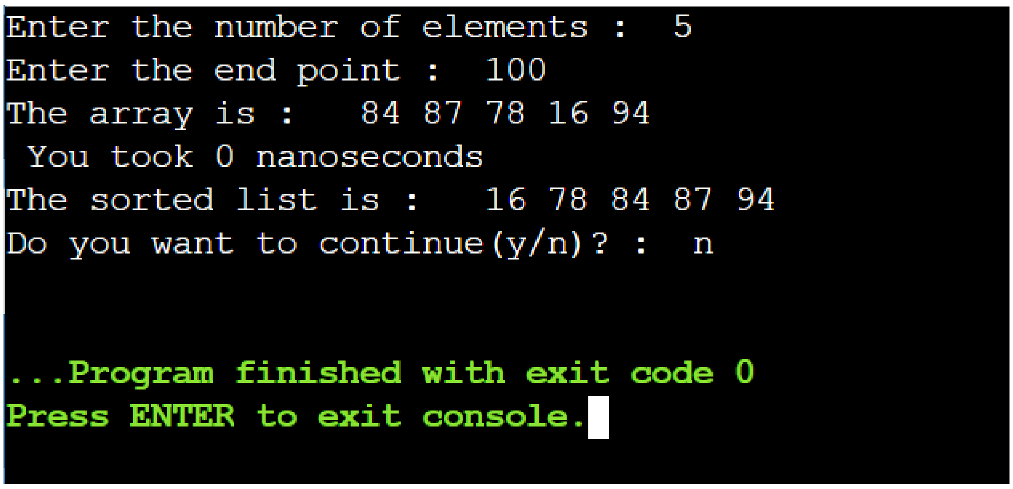
To write a program to sort elements using quick sort.

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() << " 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 :  5
Enter the end point :  100
The array is :   84 87 78 16 94
 You took 0 nanoseconds
The sorted list is :   16 78 84 87 94
Do you want to continue(y/n)? :  n


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

BREADTH FIRST SEARCH

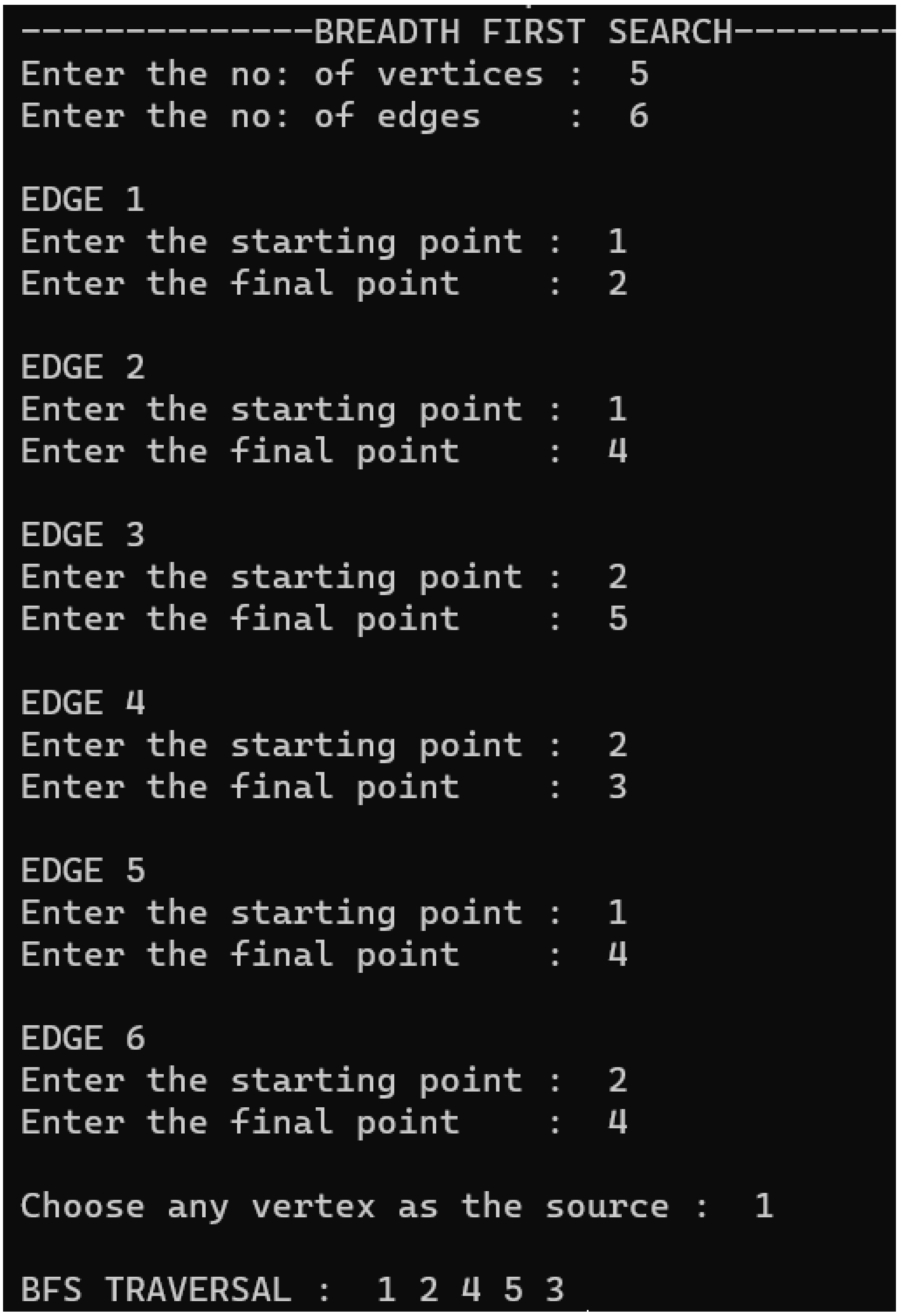
AIM

To write a program to implement Breadth First Search.

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    :  6

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

EDGE 2
Enter the starting point :  1
Enter the final point    :  4

EDGE 3
Enter the starting point :  2
Enter the final point    :  5

EDGE 4
Enter the starting point :  2
Enter the final point    :  3

EDGE 5
Enter the starting point :  1
Enter the final point    :  4

EDGE 6
Enter the starting point :  2
Enter the final point    :  4

Choose any vertex as the source :  1

BFS TRAVERSAL :  1 2 4 5 3
```

DIJKSTRA'S ALGORITHM

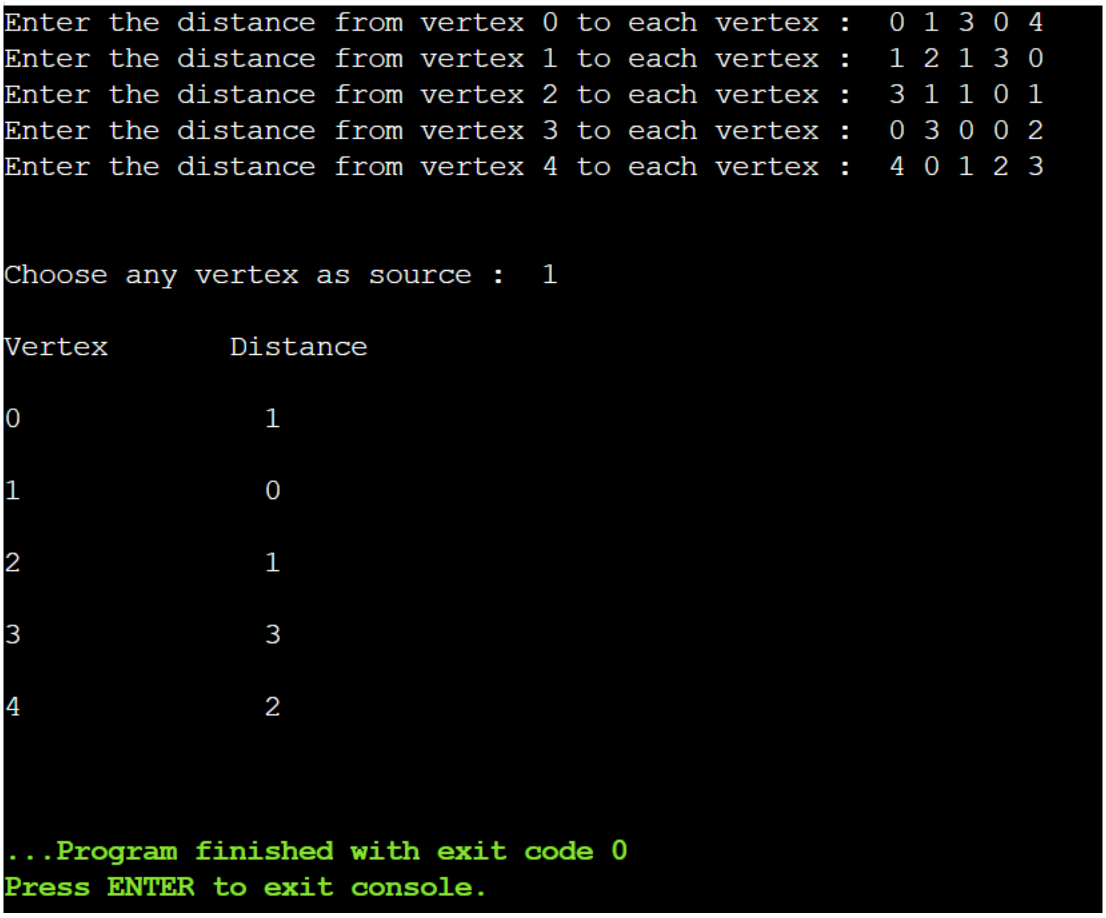
AIM

To write a program to implement Dijkstra's Algorithm.

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++)
    {
        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])
            {
                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])
                {
                    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 3 0 4
Enter the distance from vertex 1 to each vertex :  1 2 1 3 0
Enter the distance from vertex 2 to each vertex :  3 1 1 0 1
Enter the distance from vertex 3 to each vertex :  0 3 0 0 2
Enter the distance from vertex 4 to each vertex :  4 0 1 2 3


Choose any vertex as source :  1

Vertex       Distance

0              1

1              0

2              1

3              3

4              2



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

BELLMAN FORD

AIM

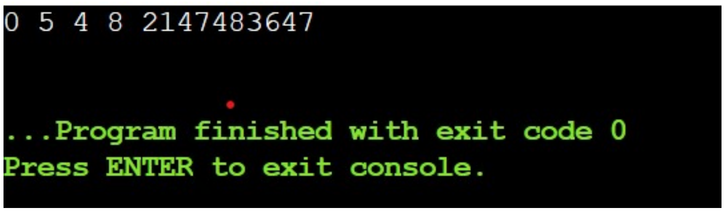
To write a program to implement Bellman Ford Algorithm.

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;
}
\newpage
```

SAMPLE INPUT-OUTPUT



```
0 5 4 8 2147483647


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

FLOYD WARSHALL ALGORITHM

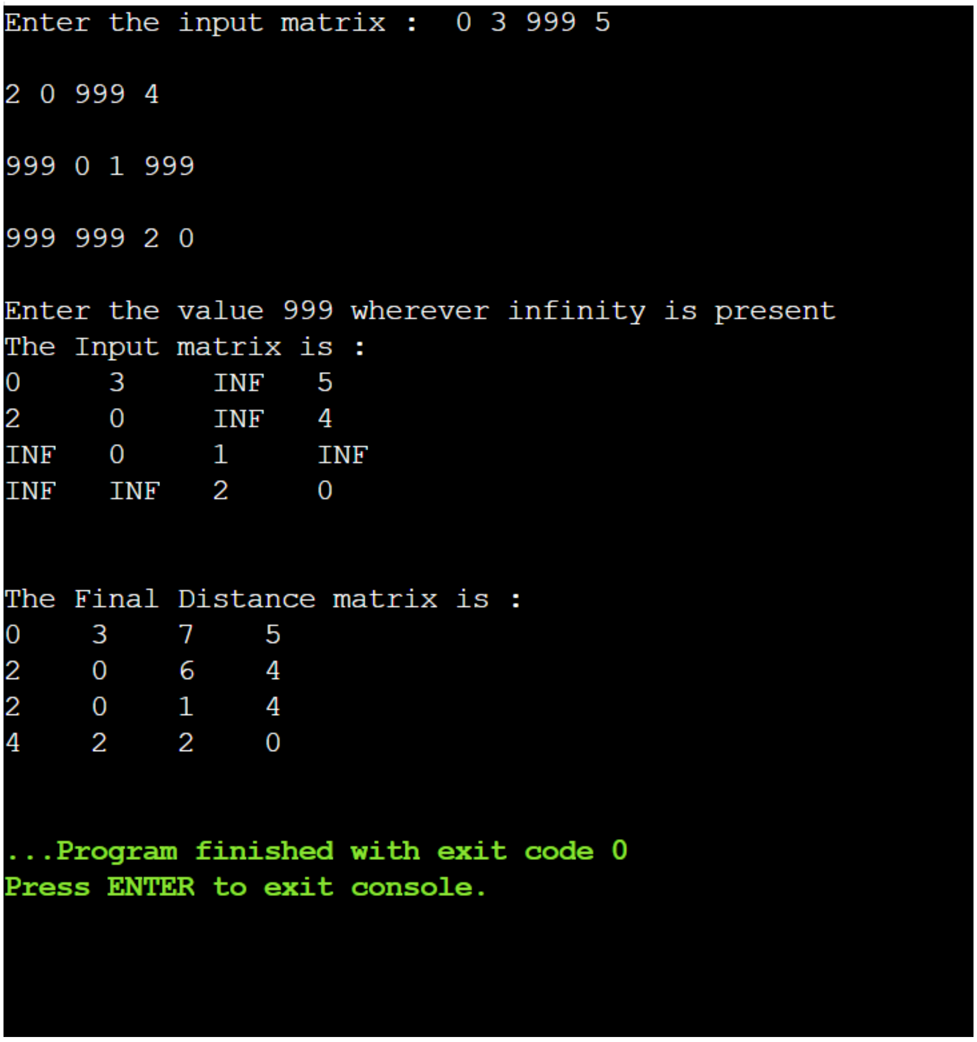
AIM

To write a program to implement Floyd Warshall Algorithm.

PROGRAM

```
#include<iostream>
using namespace std;
#define INF 999
#define num 4
void floyd_warshall(int A[][num])
{
    int i,j,k;
    for(k = 0;k<num;k++)
    {
        for(i= 0;i<num;i++)
        {
            for(j=0;j<num;j++)
            {
                if(A[i][j] > (A[i][k] + A[k][j]) && A[k][j]!= INF && A[i][k] != INF)
                {
                    A[i][j] = A[i][k] + A[k][j];

                }
            }
        }
    }
}

int main()
{
    int i,j;
    cout<<"Enter the input matrix : "<<" ";
    int M[num][num];
    for(i=0;i<num;i++)
    {
        for(j=0;j<num;j++)
        {
            cin>>M[i][j];
        }
        cout<<endl;
    }
    cout<<"Enter the value 999 wherever infinity is present "<<endl;
    cout<<"The Input matrix is : "<<endl;
    for(i=0;i<num;i++)
    {
        for(j=0;j<num;j++)
        {
            if (M[i][j] == INF)
            {
                cout<<"INF"<<"   ";
            }
            else
            {
                cout<< M[i][j]<<"     ";
            }
        }
        cout<<endl;
    }
    floyd_warshall(M);
    cout<<endl<<endl;
    cout<<"The Final Distance matrix is : "<<endl;
    for(i=0;i<num;i++)
    {
        for(j=0;j<num;j++)
        {
            if (M[i][j] == INF)
            {
                cout<<"INF"<<"   ";
            }
            else
            {
                cout<< M[i][j]<<"    ";
            }
        }
        cout<<endl;
    }
    return(0);
}
```

SAMPLE INPUT-OUTPUT



```
Enter the input matrix :  0 3 999 5

2 0 999 4

999 0 1 999

999 999 2 0

Enter the value 999 wherever infinity is present
The Input matrix is :
0      3      INF    5
2      0      INF    4
INF    0      1      INF
INF    INF    2      0


The Final Distance matrix is :
0     3     7     5
2     0     6     4
2     0     1     4
4     2     2     0


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

KRUSKAL'S ALGORITHM

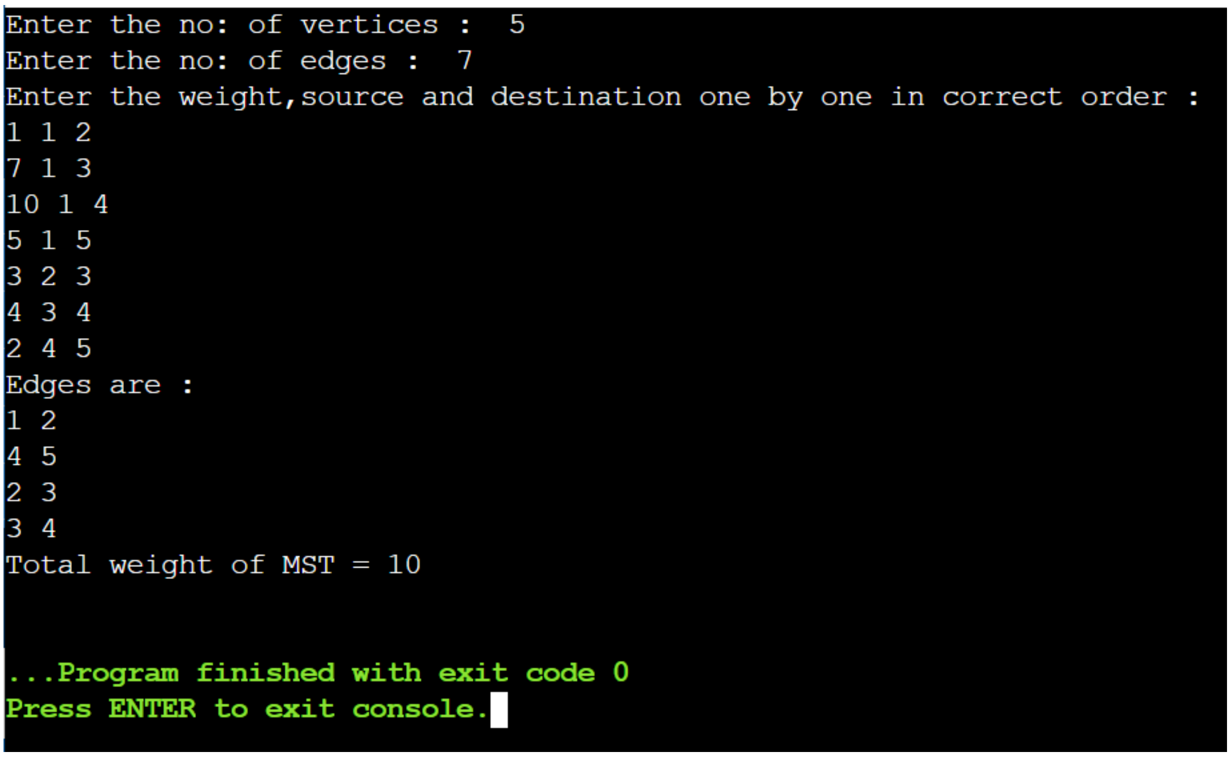
AIM

To write a program to implement Kruskal's Algorithm.

PROGRAM

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
const int MAX = 99999;
int parent[MAX];
int find(int a)
{
    while (parent[a] != a)
    {
        parent[a] = parent[parent[a]];
        a = parent[a];
    }
    return a;
}

void add(int a, int b)
{

    int d = find(a);
    int e = find(b);
    parent[d] = parent[e];
}

int main()
{
    int V, E;
    cout << "Enter the no: of vertices : "
         << " ";
    cin >> V;
    cout << "Enter the no: of edges : "
         << " ";
    cin >> E;
    vector<pair<int, pair<int, int>>> adj;
    cout << "Enter the weight,source and destination one by one in correct order : "
    << endl;
    for (int i = 0; i < E; i++)
    {
        int weight;
        int src, destination;
        cin >> weight >> src >> destination;
        adj.push_back({weight, {src, destination}});
    }
    sort(adj.begin(), adj.end());
    for (int i = 0; i < MAX; i++)
    {
        parent[i] = i;
    }
    vector<pair<int, int>> tree_edges;
    int totalweight = 0;
    for (auto x : adj)
    {
        int a = x.second.first;
        int b = x.second.second;
        int cost = x.first;
        if (find(a) != find(b))
        {
            totalweight += cost;
            add(a, b);
            tree_edges.push_back({a, b});
        }
    }
    cout << "Edges are : " << endl;
    for (auto x : tree_edges)
    {
        cout << x.first << " " << x.second << endl;
    }
    cout << "Total weight of MST = ";
    cout << totalweight << endl;

    return (0);
}
```

SAMPLE INPUT-OUTPUT



```
Enter the no: of vertices :  5
Enter the no: of edges :  7
Enter the weight,source and destination one by one in correct order :
1 1 2
7 1 3
10 1 4
5 1 5
3 2 3
4 3 4
2 4 5
Edges are :
1 2
4 5
2 3
3 4
Total weight of MST = 10


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

PRIM'S ALGORITHM

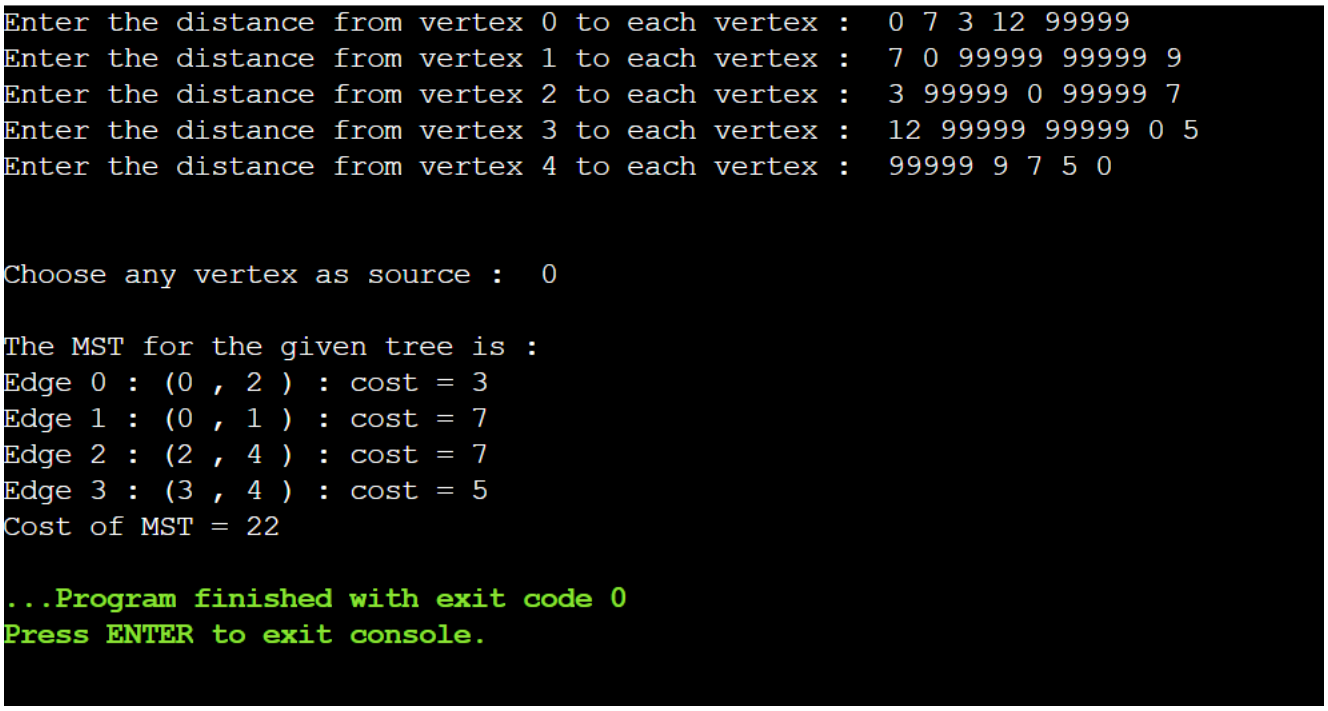
AIM

To write a program to implement Prim's Algorithm.

PROGRAM

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
const int MAX = 99999;
#define V 5
bool createsMST(int u, int v, vector<bool> V_MST)
{
    if (u == v)
    {
        return false;
    }
    if (V_MST[u] == false && V_MST[v] == false)
    {
        return false;
    }
    else if (V_MST[u] == true && V_MST[v] == true)
    {
        return false;
    }
    return true;
}
void MST_display(int cost[][V])
{
    vector<bool> V_MST(V, false);
    V_MST[0] = true;
    int edgeNo = 0, MSTcost = 0;
    while (edgeNo < V - 1)
    {
        int min = MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                if (cost[i][j] < min)
                {
                    if (createsMST(i, j, V_MST))
                    {
                        min = cost[i][j];
                        a = i;
                        b = j;
                    }
                }
            }
        }
        if (a != -1 && b != -1)
        {
            cout << "Edge " << edgeNo++ << " : (" << a << " , " << b << " ) :
            cost = " << min << endl;
            MSTcost += min;
            V_MST[b] = V_MST[a] = true;
        }
    }
    cout << "Cost of MST = " << MSTcost;
}
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];
        }
    }

    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (G[i][j] == -1)
            {
                G[i][j] = MAX;
            }
            else
            {
                G[i][j] = G[i][j];
            }
        }
    }
    cout << endl<< endl;
    cout << "Choose any vertex as source : "<< " ";
    cin >> source;
    cout << endl;
    cout << "The MST for the given tree is :\n";
    MST_display(G);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the distance from vertex 0 to each vertex :  0 7 3 12 99999
Enter the distance from vertex 1 to each vertex :  7 0 99999 99999 9
Enter the distance from vertex 2 to each vertex :  3 99999 0 99999 7
Enter the distance from vertex 3 to each vertex :  12 99999 99999 0 5
Enter the distance from vertex 4 to each vertex :  99999 9 7 5 0


Choose any vertex as source :  0

The MST for the given tree is :
Edge 0 : (0 , 2 ) : cost = 3
Edge 1 : (0 , 1 ) : cost = 7
Edge 2 : (2 , 4 ) : cost = 7
Edge 3 : (3 , 4 ) : cost = 5
Cost of MST = 22

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

TOPOLOGICAL SORTING

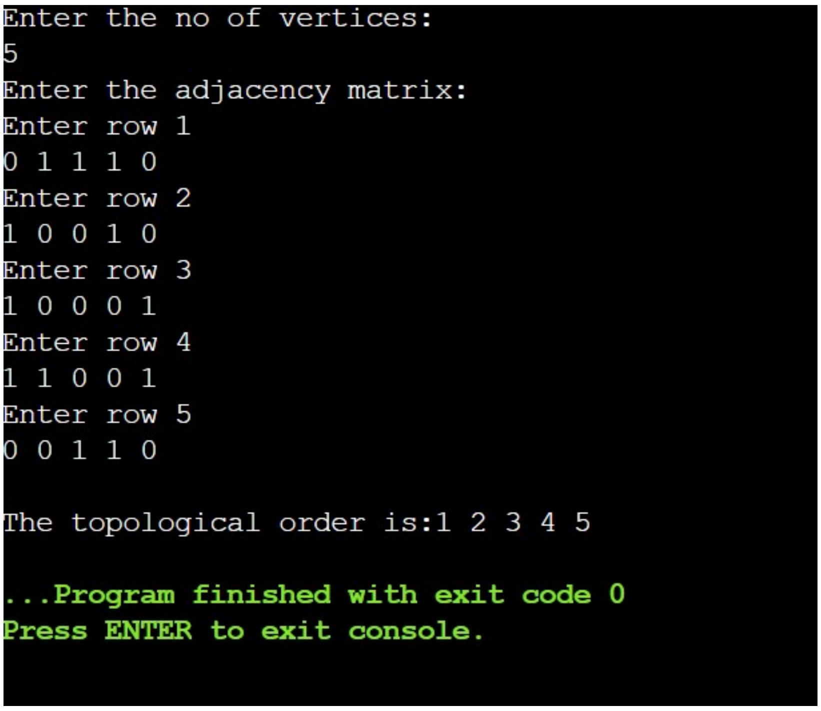
AIM

To write a program to implement Topological Sorting.

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.
```

MATRIX MULTIPLICATION CHAIN RULE

AIM

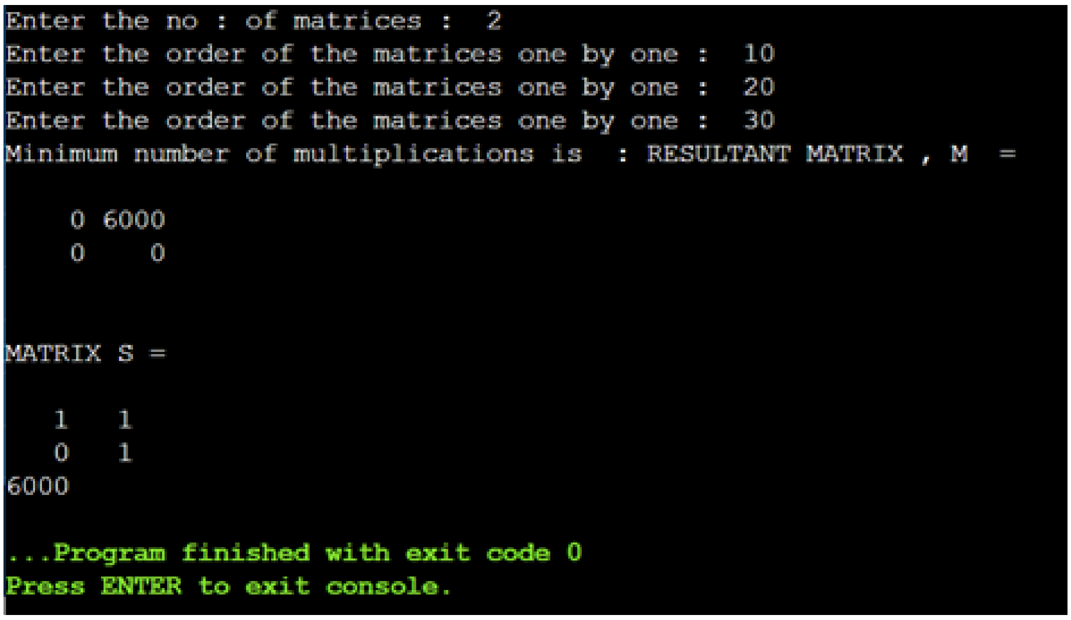
To write a program to implement Matrix Multiplication Chain Rule.

PROGRAM

```
#include <bits/stdc++.h>
#include <iostream>
#include <iomanip>
using namespace std;
int MatrixChainOrder(int p[], int n)
{
    int m[n][n];
    int s[n - 1][n - 1]; // Stores the value of k
    int i, j, k, L, q;

    for (i = 1; i < n; i++)
    {
        for (int j = 1; j < n; j++)
        {
            m[i][j] = 0;
        }
    }
    for (i = 1; i < n; i++)
    {
        for (int j = 1; j < n; j++)
        {
            if (i > j)
            {
                s[i][j] = 0;
            }
            else
            {
                s[i][j] = 1;
            }
        }
    }

    // L = chain length
    for (L = 2; L < n; L++)
    {
        for (i = 1; i < n - L + 1; i++)
        {
            j = i + L - 1;
            m[i][j] = INT_MAX;
            for (k = i; k <= j - 1; k++)
            {
                // q = cost/scalar multiplications
                q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                {
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
    cout << "RESULTANT MATRIX , M  =  " << endl
         << endl;
    for (int i = 1; i < n; i++)
    {
        for (int j = 1; j < n; j++)
        {
            cout << setw(5) << m[i][j] << setw(4);
        }
        cout << endl;
    }
    cout << endl
         << endl;
    cout << "MATRIX S =  " << endl
         << endl;
    for (int i = 1; i < n; i++)
    {
        for (int j = 1; j < n; j++)
        {
            cout << setw(4) << s[i][j] << setw(4);
        }
        cout << endl;
    }
    return m[1][n - 1];
}
```

```
int main()
{
    // int A[] = {5, 4, 6, 2, 7};
    int num;
    cout << "Enter the no : of matrices : "
         << " ";
    cin >> num;
    int A[num + 1];
    for (int i = 0; i < num + 1; i++)
    {
        cout << "Enter the order of the matrices one by one : "
             << " ";
        cin >> A[i];
    }
    int size = sizeof(A) / sizeof(A[0]);
    cout << "Minimum number of multiplications is  : " << MatrixChainOrder(A, size);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the no : of matrices :  2
Enter the order of the matrices one by one :  10
Enter the order of the matrices one by one :  20
Enter the order of the matrices one by one :  30
Minimum number of multiplications is  : RESULTANT MATRIX , M  =

    0 6000
    0    0


MATRIX S =

   1   1
   0   1
6000

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

KNAPSACK PROBLEM

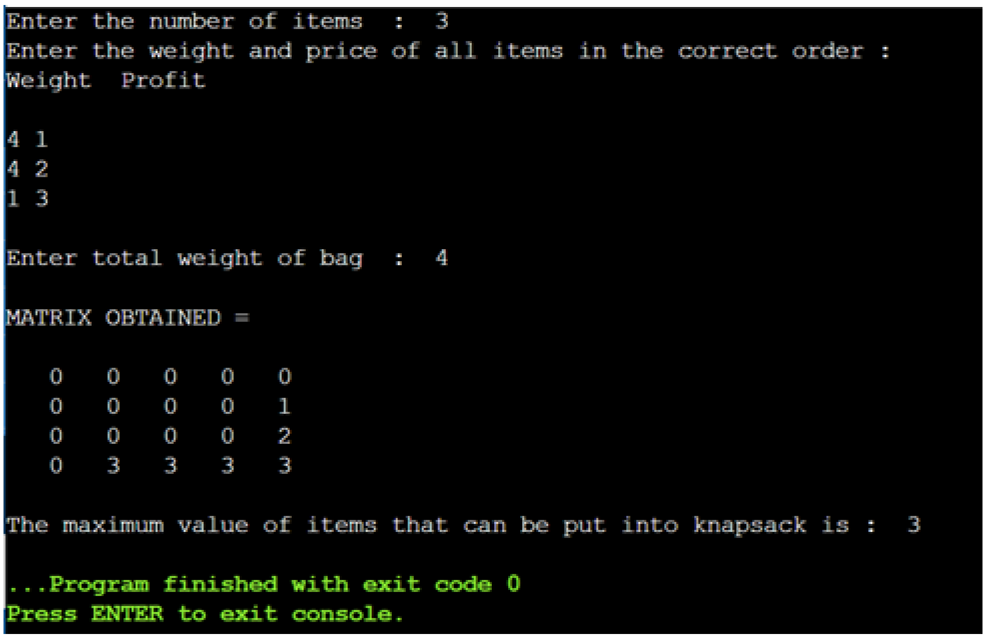
AIM

To write a program to implement Knapsack Problem.

PROGRAM

```
#include <bits/stdc++.h>
using namespace std;

int knapsack_dp(int n, int M, int w[], int p[])
{
    int i, j;
    int c[n + 1][M + 1];
    for (i = 0; i <= n + 1; i++)
    {
        for (j = 0; j <= M + 1; j++)
        {
            c[i][j] = 0;
        }
    }
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= M; j++)
        {
            if (w[i - 1] <= j)
            {
                c[i][j] = max(c[i - 1][j], p[i - 1] + c[i - 1][j - w[i - 1]]);
            }
            else
            {
                c[i][j] = c[i - 1][j];
            }
        }
    }
    cout << "MATRIX OBTAINED =  " << endl
         << endl;
    for (int i = 0; i < n + 1; i++)
    {
        for (int j = 0; j < M + 1; j++)
        {
            cout << setw(4) << c[i][j] << setw(4);
        }
        cout << endl;
    }
    return c[n][M];
}

int main()
{
    int i, j;
    int n; // number of items
    int M; // capacity of knapsack (total weight of bag)
    cout << "-------------------0/1 KNAPSACK PROBLEM-------------------" << endl;
    cout << "Enter the no. of items  : "
         << " ";
    cin >> n;

    int w[n]; // weight of items
    int p[n]; // profit of items

    cout << "Enter the weight and price of all items in the correct order : " << endl;
    cout << "Weight"
         << "  "
         << "Profit" << endl
         << endl;
    for (i = 0; i < n; i++)
    {
        cin >> w[i] >> p[i];
    }
    cout << endl;
    cout << "Enter the capacity of knapsack  :  ";
    cin >> M; // total weight of the bag
    cout << endl;
    int result = knapsack_dp(n, M, w, p);
    cout << endl;
    cout << "The maximum value of items that can be put into knapsack is :  " << result;
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of items  :  3
Enter the weight and price of all items in the correct order :
Weight  Profit

4 1
4 2
1 3

Enter total weight of bag  :  4

MATRIX OBTAINED =

   0   0   0   0   0
   0   0   0   0   1
   0   0   0   0   2
   0   3   3   3   3

The maximum value of items that can be put into knapsack is :  3

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

HUFFMAN CODE

AIM

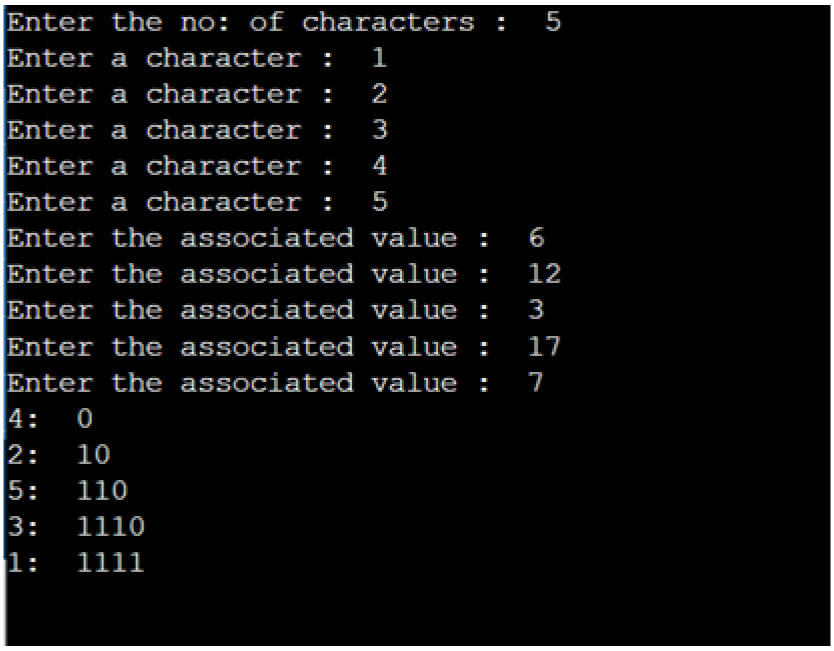
To write a program to implement Huffman Code.

PROGRAM

```
#include <bits/stdc++.h>
using namespace std;

struct MinHeapNode
{
    char d;
    unsigned frequency;
    MinHeapNode *lChild, *rChild;

    MinHeapNode(char d, unsigned frequency)

    {

        lChild = rChild = NULL;
        this->d = d;
        this->frequency = frequency;
    }
};

// function to compare
struct compare
{
    bool operator()(MinHeapNode *l, MinHeapNode *r)
    {
        return (l->frequency > r->frequency);
    }
};

void printCodes(struct MinHeapNode *root, string str)
{
    if (!root)
        return;

    if (root->d != '$')
```

```
        cout << root->d << ": " << str << "\n";

    printCodes(root->lChild, str + "0");
    printCodes(root->rChild, str + "1");
}

void HuffmanCodes(char d[], int frequency[], int size)
{
    struct MinHeapNode *lChild, *rChild, *top;

    priority_queue<MinHeapNode *, vector<MinHeapNode *>, compare> minHeap;

    for (int i = 0; i < size; i++)
        minHeap.push(new MinHeapNode(d[i], frequency[i]));

    while (minHeap.size() != 1)
    {
        lChild = minHeap.top();
        minHeap.pop();

        rChild = minHeap.top();
        minHeap.pop();

        top = new MinHeapNode('$', lChild->frequency + rChild->frequency);

        top->lChild = lChild;
        top->rChild = rChild;

        minHeap.push(top);
    }
    printCodes(minHeap.top(), " ");
}

int main()
{
    int num;

    cout << "Enter the no: of characters : "
         << " ";
    cin >> num;
    char A[num];
    int X[num];
    for (int i = 0; i < num; i++)
    {
        cout << "Enter a character : "
             << " ";
        cin >> A[i];
    }
    for (int i = 0; i < num; i++)
    {
        cout << "Enter the associated value : "
             << " ";
        cin >> X[i];
    }

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

    HuffmanCodes(A, X, size);

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the no: of characters :  5
Enter a character :  1
Enter a character :  2
Enter a character :  3
Enter a character :  4
Enter a character :  5
Enter the associated value :  6
Enter the associated value :  12
Enter the associated value :  3
Enter the associated value :  17
Enter the associated value :  7
4:  0
2:  10
5:  110
3:  1110
1:  1111
```

TRAVELING SALESMAN PROBLEM

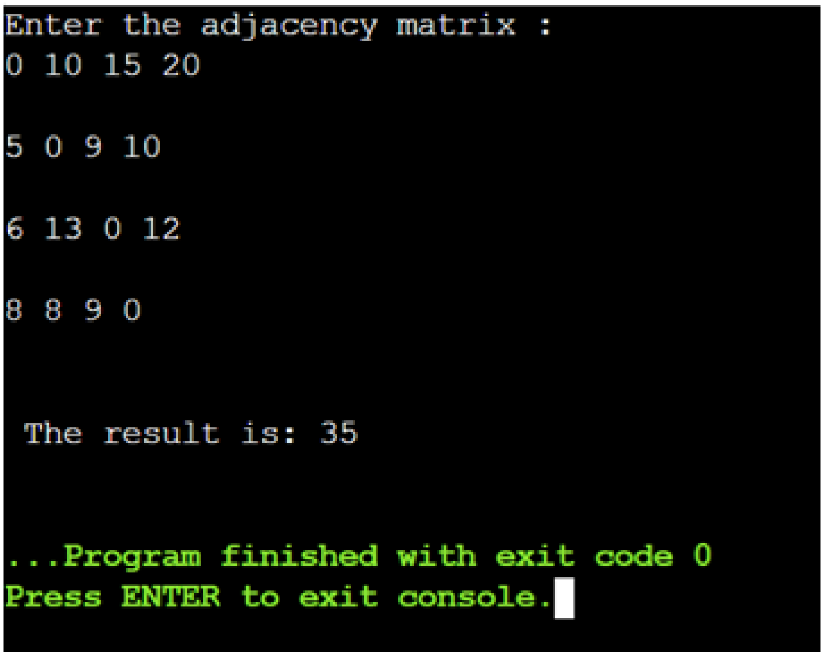
AIM

To write a program to implement Traveling Salesman Problem.

PROGRAM

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
#define vr 4
int TSP(int graph[][vr], int p)
{
    vector<int> ver;
    for (int i = 0; i < vr; i++)
    {
        if (i != p)
            ver.push_back(i);
    }
    int m_p = INT_MAX;
    do
    {
        int current_path = 0;
        int k = p;
        for (int i = 0; i < ver.size(); i++)
        {
            current_path += graph[k][ver[i]];
            k = ver[i];
        }
        current_path += graph[k][p];
        m_p = min(m_p, current_path); // to update the value of minimum weight
    } while (next_permutation(ver.begin(), ver.end()));
    return m_p;
}
int main()
{
    cout << "--------------TRAVELLING SALESMAN PROBLEM----------------" << endl;
    int graph[vr][vr];
    cout << "Enter the adjacency matrix : " << endl;
    for (int i = 0; i < vr; i++)
    {
        for (int j = 0; j < vr; j++)
        {
            cin >> graph[i][j];
        }
        cout << endl;
    }
    int p = 0;
    cout << "\n The result is: " << TSP(graph, p) << endl;
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the adjacency matrix :
0 10 15 20

5 0 9 10

6 13 0 12

8 8 9 0


 The result is: 35


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