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

ABHIN P.T
(80521002)

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* **ABHIN P T (80521002)***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

```cpp
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;

int partition(int* A,int p,int r){
    int x=A[r];
    int i=p-1;

    for(int j=p;j<=r-1;j++){
        if(A[j]<=x){
            i=i+1;
            int temp=A[i];
            A[i]=A[j];
            A[j]=temp;
        }
    }
    int temp=A[i+1];
    A[i+1]=A[r];
    A[r]=temp;
    return i+1;
}
void quick_sort(int A[],int p,int r){
    if(p<r){
        int q=partition(A,p,r);
        quick_sort(A,p,q-1);
        quick_sort(A,q+1,r);
    }

}

int main(){
    int n;
```

```cpp
    cout<<"Emter the number of elemnts : ";
    cin>>n;
    int* A = new int[n];
    for(int i=0;i<n;i++){
        A[i]=rand()%500;
    }
    for(int i=0;i<n;i++)
        cout<<A[i]<<" ";
    cout<<"\n";

    auto start = high_resolution_clock::now();
    quick_sort(A,0,n-1);
    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop - start);
    cout<<"\n\nTime taken to sort : "<<duration.count()<<" microseconds"<<endl;

    auto st = high_resolution_clock::now();
    quick_sort(A,0,n-1);
    auto sp = high_resolution_clock::now();



    auto drtn = duration_cast<microseconds>(sp - st);
    cout<<"\n\nTime taken to sort : "<<drtn.count()<<" microseconds"<<endl;

    return 0;
}
```
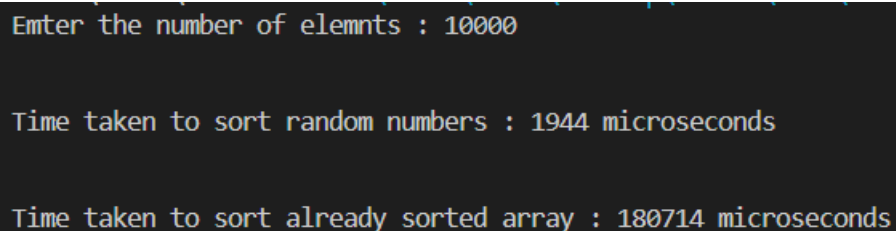
SAMPLE INPUT-OUTPUT


```
Emter the number of elemnts : 10000


Time taken to sort random numbers : 1944 microseconds


Time taken to sort already sorted array : 180714 microseconds
```

BREADTH FIRST SEARCH

AIM

To write a program to implement Breadth First Search.
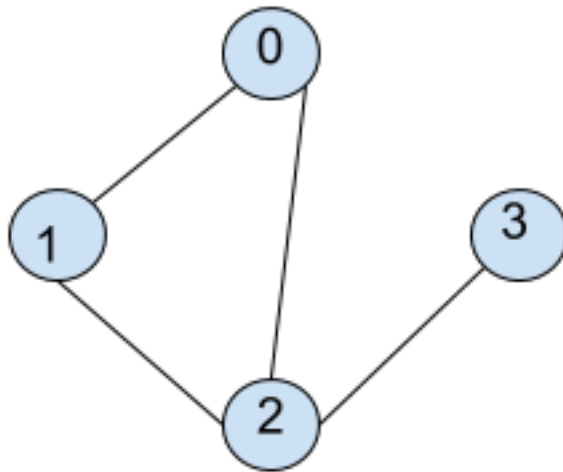
PROGRAM

```cpp
#include<iostream>
using namespace std;
int visited[10];
void bfs(int n,int a[10][10],int source)
{
    int i,q[10],u,front=1,rear=1;
    visited[source]=1;
    q[rear]=source;
    while(front<=rear)
    {
        u=q[front];
        front++;
        for(i=1;i<=n;i++)
            if(a[u][i]==1 && visited[i]==0)
            {
                rear++;
                q[rear]=i;
                visited[i]=1;
            }

    }
}
int main()
{
    int n,a[10][10],i,j,source;
    cout<<"Enter the number of nodes : ";
    cin>>n;
    cout<<"Enter the adjacency matrix :\n";
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];
    cout<<"Enter the source : ";
    cin>>source;
    for(i=1;i<=n;i++)
```

```
        visited[i]=0;
    bfs(n,a,source);
    for(i=1;i<=n;i++)
    {
        if(visited[i]==0)
            cout<<"The node "<<i<<" is NOT reachable.\n";
        else
            cout<<"The node "<<i<<" is reachable.\n";


    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

DJKSTRA'S ALGORITHM

AIM

To write a program to implement Djkstra's Algorithm.

PROGRAM

```cpp
#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;
```

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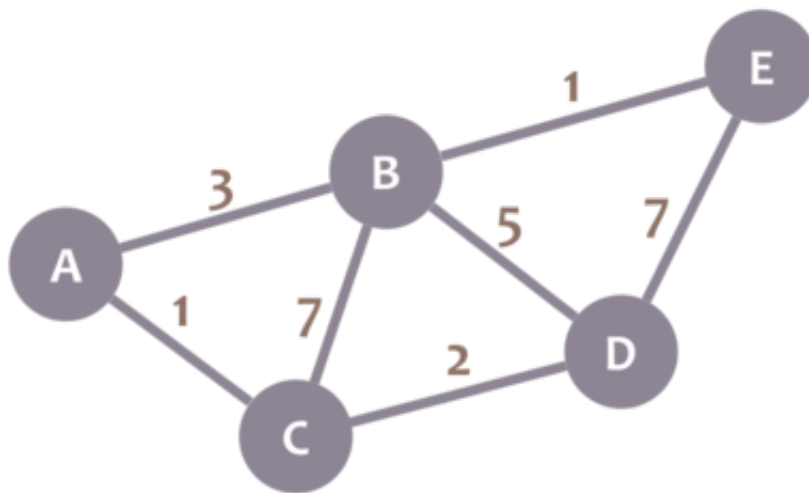
```
    int G[V][V];
    int source;
    for(int i = 0; i <  V; i++)
    {
        cout<<"Enter the distance from vertex "
        << i <<" to each vertex : "<<" ";
        for(int j = 0;j < V; j++)
        {
            cin>>G[i][j];
        }
    }
    cout<<endl<<endl;
    cout<<"Choose any vertex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

SAMPLE INPUT-OUTPUT

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


Choose any vertex as source :  0

Vertex        Distance

0                0

1                3

2                1

3                3

4                4
```

BELLMAN FORD

AIM

To write a program to implement Bellman Ford Algorithm.

PROGRAM

```
#include<iostream>
#define MAX 10
using namespace std;
typedef struct edge
{
  int src;
  int dest;
  int wt;
}edge;
void bellman_ford(int nv,edge e[],int src_graph,int ne)
{
  int u,v,weight,i,j=0;
  int dis[MAX];

  /* initializing array 'dis' with 999. 999 denotes infinite distance */
  for(i=0;i<nv;i++)
  {
    dis[i]=999;
  }

  /* distance of source vertex from source vertex is o */
  dis[src_graph]=0;

  /* relaxing all the edges nv - 1 times */
  for(i=0;i<nv-1;i++)
  {
    for(j=0;j<ne;j++)
    {
      u=e[j].src;
      v=e[j].dest;
      weight=e[j].wt;

      if(dis[u]!=999 && dis[u]+weight < dis[v])
      {
```

```
          dis[v]=dis[u]+weight;
        }
      }


  }


  /* checking if negative cycle is present */
  for(j=0;j<ne;j++)
  {
    u=e[j].src;
    v=e[j].dest;
    weight=e[j].wt;

    if(dis[u]+weight < dis[v])
    {
      cout<<"\n\nNEGATIVE CYCLE PRESENT..!!\n";
      return;
    }
  }

  cout<<"\nVertex"<<"  Distance from source";
  for(i=1;i<=nv;i++)
  {
    cout<<"\n"<<i<<"\t"<<dis[i];
  }
}
int main()
{
  int nv,ne,src_graph;
  edge e[MAX];

  cout<<"Enter the number of vertices: ";
  cin>>nv;

  /* if you enter no of vertices: 5 then vertices will be 1,2,3,4,5. so while giving input
  printf("Enter the source vertex of the graph: ");
  cin>>src_graph;

  cout<<"\nEnter no. of edges: ";
  cin>>ne;
```

```
  for(int i=0;i<ne;i++)
  {
     cout<<"\nFor edge "<<i+1<<"=>";
     cout<<"\nEnter source vertex :";
     cin>>e[i].src;
     cout<<"Enter destination vertex :";
     cin>>e[i].dest;
     cout<<"Enter weight :";
     cin>>e[i].wt;
  }


  bellman_ford(nv,e,src_graph,ne);


  return 0;
}
```

SAMPLE INPUT-OUTPUT

```
C:\Users\91892\Desktop\Public\sem3\DAA>a.exe
Enter the number of vertices: 4
Enter the source vertex of the graph: 1

Enter no. of edges: 4

For edge 1=>
Enter source vertex :1
Enter destination vertex :2
Enter weight :4

For edge 2=>
Enter source vertex :1
Enter destination vertex :3
Enter weight :3

For edge 3=>
Enter source vertex :2
Enter destination vertex :4
Enter weight :7

For edge 4=>
Enter source vertex :3
Enter destination vertex :4
Enter weight :-2

Vertex   Distance from source
1        0
2        4
3        3
4        1
```

FLOYD WARSHALL ALGORITHM

AIM

To write a program to implement Floyd Warshall Algorithm.

PROGRAM

```cpp
#include<iostream>
using namespace std;
#define INF 99999
#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,no_of_vertices;

    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 99999 wherever infinity is present "<<endl<<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

```
PS C:\Users\91892\Desktop\Public\sem3\DAA> cd "c:\Users\91892\D
Enter the input matrix :  0 9999 3 9999

2 0 9999 9999

9999 7 0 1

6 9999 9999 0

Enter the value 99999 wherever infinity is present

The Input matrix is :
0       9999    3       9999
2       0       9999    9999
9999    7       0       1
6       9999    9999    0


The Final Distance matrix is :
0    10    3    4
2    0     5    6
7    7     0    1
6    16    9    0
PS C:\Users\91892\Desktop\Public\sem3\DAA>
```

KRUSKAL'S ALGORITHM

AIM

To write a program to implement Kruskal's Algorithm.

PROGRAM

```cpp
#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; // Storing the edges of MST
    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)) // checking whether it forms a cycle
        {
        // if the two vertices are in different subsets, merge them into one
            totalweight += cost;
            add(a, b);
            tree_edges.push_back({a, b});
        }
    }
    cout << "Edges of Minimum Spanning Tree : " << endl;
    for (auto x : tree_edges) // MST edges
    {
        cout << x.first << " " << x.second << endl;
    }
    cout << "Total weight of MST = ";
    cout << totalweight << endl;

    return (0);
}
```

SAMPLE INPUT-OUTPUT

```
AA> cd "c:\Users\91892\Deskto
p\Public\sem3\DAA\" ; if ($?) { g++ Kruskals.cpp -o Kruskals } ; if
($?) { .\Kruskals }
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
3 2 3
7 1 3
10 1 4
4 3 4
2 4 5
5 1 5
Edges are :
1 2
4 5
2 3
3 4
Total weight of MST = 10
PS C:\Users\91892\Desktop\Public\sem3\DAA>
```

PRIM'S ALGORITHM

AIM

To write a program to implement Prim's Algorithm.

PROGRAM

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
const int MAX = 99999; // INF
#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)
```
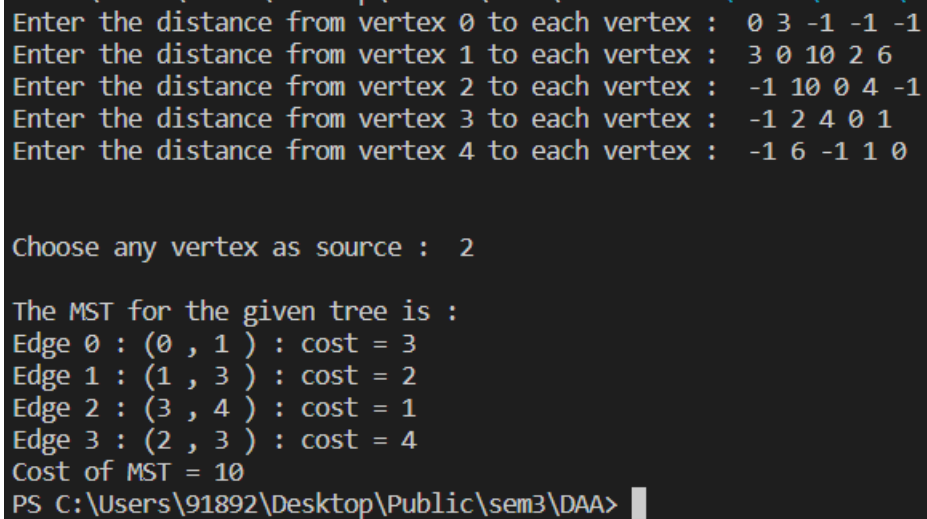
```cpp
{
if (createsMST(i, j, V_MST))
{
min = cost[i][j];
a = i;
b = j;
}
}
}
}
if (a != -1 && b != -1)
{
cout << "Edge " << edgeNo++ << " : (" << a << " , " <<
b << " ) : cost = " << min << endl;
MSTcost += min;
V_MST[b] = V_MST[a] = true;
}
}
cout << "Cost of MST = " << MSTcost;
}
int main()
{
int 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++) // Assigning the values (initialisation)
{
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 3 -1 -1 -1
Enter the distance from vertex 1 to each vertex :  3 0 10 2 6
Enter the distance from vertex 2 to each vertex :  -1 10 0 4 -1
Enter the distance from vertex 3 to each vertex :  -1 2 4 0 1
Enter the distance from vertex 4 to each vertex :  -1 6 -1 1 0


Choose any vertex as source :  2

The MST for the given tree is :
Edge 0 : (0 , 1 ) : cost = 3
Edge 1 : (1 , 3 ) : cost = 2
Edge 2 : (3 , 4 ) : cost = 1
Edge 3 : (2 , 3 ) : cost = 4
Cost of MST = 10
PS C:\Users\91892\Desktop\Public\sem3\DAA>
```

TOPOLOGICAL SORTING

AIM

To write a program to implement Topological Sorting.

PROGRAM

```cpp
#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
```

MATRIX MULTIPLICATION CHAIN RULE

AIM

To write a program to implement Matrix Multiplication Chain Rule.

PROGRAM

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

int MatrixChainOrder(int p[], int i, int j)
{
if (i == j)
return 0;
int k;
int mini = INT_MAX;
int count;



for (k = i; k < j; k++)
{
count = MatrixChainOrder(p, i, k)
+ MatrixChainOrder(p, k + 1, j)
+ p[i - 1] * p[k] * p[j];

mini = min(count, mini);
}



return mini;
}



int main()
{
    int n;
    cout<<"Enter the number of matrices : ";
    cin>>n;
int arr[n+1];
    cout<<"Enter the order of matrices \n";
```
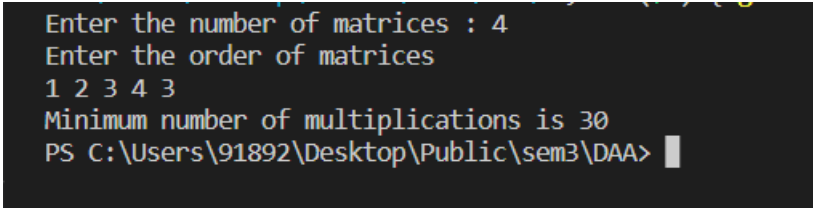
```
    for(int i=0;i<n+1;i++)
        cin>>arr[i];
int N = n+1;


cout << "Minimum number of multiplications is "
<< MatrixChainOrder(arr, 1, N - 1);
return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the number of matrices : 4
Enter the order of matrices
1 2 3 4 3
Minimum number of multiplications is 30
PS C:\Users\91892\Desktop\Public\sem3\DAA>
```

KNAPSACK PROBLEM

AIM

To write a program to implement Knapsack Problem.

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;
```

```
}
\newpage
```

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
```

HUFFMAN CODE

AIM

To write a program to implement Huffman Code.

PROGRAM

```cpp
// 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);
  }
```

```c
}

// Check if size if 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);
}
```
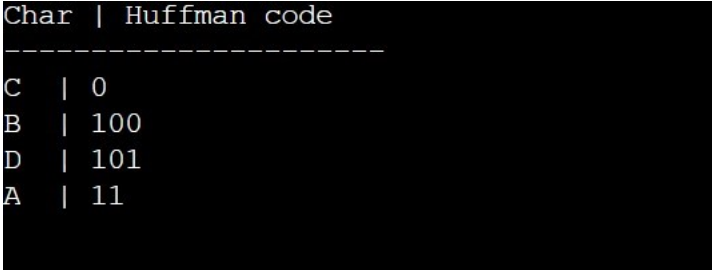
SAMPLE INPUT-OUTPUT

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

TRAVELLING SALESMAN PROBLEM

AIM

Implement any scheme to find the optimal solution for the Traveling Salesman Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.
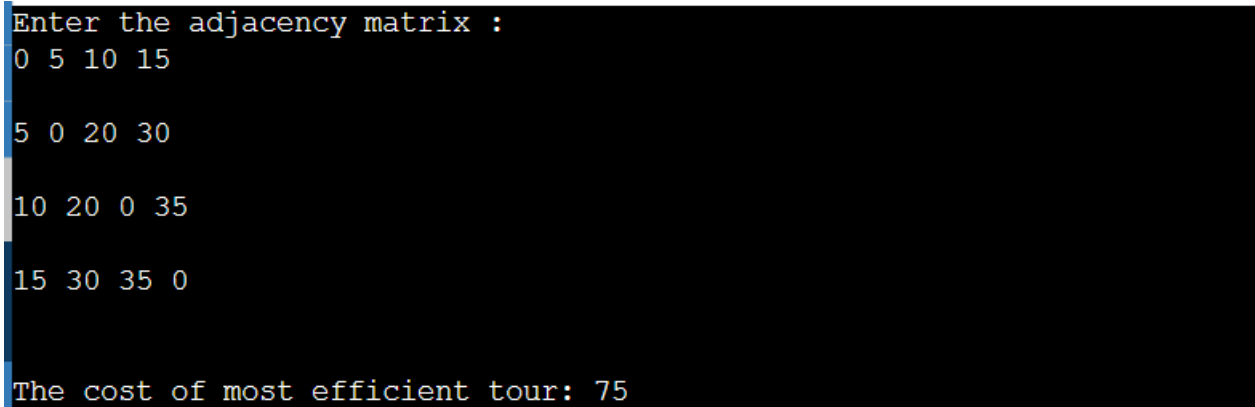
PROGRAM

```cpp
#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()
{
  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<<"\nThe cost of most efficient tour: " << TSP(graph, p) << endl;
  return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the adjacency matrix :
0 5 10 15

5 0 20 30

10 20 0 35

15 30 35 0


The cost of most efficient tour: 75
```