**1. Fractional Knapsack Problem**

**CODE**

# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity)

{

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

{

if (weight[i] > u)

break;

else

{

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if (i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is: ");

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

printf("%f\t", x[i]);

printf("\nMaximum profit is: %f", tp);

}

int main()

{

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

printf("\nEnter the no. of objects: ");

scanf("%d", &num);

printf("\nEnter the wts and profits of each object: ");

for (i = 0; i < num; i++)

scanf("%f %f", &weight[i], &profit[i]);

printf("\nEnter the capacity of knapsack: ");

scanf("%f", &capacity);

for (i = 0; i < num; i++)

ratio[i] = profit[i] / weight[i];

for (i = 0; i < num; i++)

{

for (j = i + 1; j < num; j++)

{

if (ratio[i] < ratio[j])

{

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

}

knapsack(num, weight, profit, capacity);

return(0);

}

**OUTPUT**

Enter the no. of objects: 7

Enter the wts and profits of each object:

2 10

3 5

5 15

7 7

1 6

4 18

1 3

Enter the capacity of knapsack: 15

The result vector is: 1.000000 1.000000 1.000000 1.000000 1.000000 0.666667 0.000000

Maximum profit is: 55.333332

**2. Job Sequencing Problem**

**CODE**

#include<iostream>

#include<algorithm>

using namespace std;

struct Job

{

char id;

int dead;

int profit;

};

bool comparison(Job a, Job b)

{

return (a.profit > b.profit);

}

void printJobScheduling(Job arr[], int n)

{

sort(arr, arr+n, comparison);

int result[n];

bool slot[n];

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

slot[i] = false;

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

{

for (int j=min(n, arr[i].dead)-1; j>=0; j--)

{

if (slot[j]==false)

{

result[j] = i;

slot[j] = true;

break;

}

}

}

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

if (slot[i])

cout << arr[result[i]].id << " ";

}

int main()

{

Job arr[] = { {'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27},

{'d', 1, 25}, {'e', 3, 15}};

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

cout << "Following is maximum profit sequence of jobs\n";

printJobScheduling(arr, n);

return 0;

}

**OUTPUT**

Following is maximum profit sequence of jobs

c a e

**CONCLUSION**

In this lab, we got familiar with greedy algorithms: fractional knapsack and job sequencing problems and implemented them using the C and C++ programming language.

**1. Kruskal’s Algorithm**

**CODE**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

printf("\nImplementation of Kruskal's Algorithm\n");

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("\nThe edges of Minimum Cost Spanning Tree are\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

{

for(j=1;j <= n;j++)

{

if(cost[i][j] < min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimum cost = %d\n",mincost);

getch();

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

return 1;

}

return 0;

}

**OUTPUT**

Implementation of Kruskal's Algorithm

Enter the no. of vertices:5

Enter the cost adjacency matrix:

0 7 3 4 0

7 0 6 8 1

3 6 0 0 5

4 8 0 0 2

0 1 5 2 0

The edges of Minimum Cost Spanning Tree are

1 edge (2,5) =1

2 edge (4,5) =2

3 edge (1,3) =3

4 edge (1,4) =4

Minimum cost = 10

**2. Prim’s Algorithm**

**CODE**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

printf("\nEnter the number of nodes:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

}

visited[1]=1;

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

for(j=1;j<=n;j++)

if(cost[i][j]< min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\nMinimum cost=%d",mincost);

getch();

}

**OUTPUT**

Enter the number of nodes:5

Enter the adjacency matrix:

0 7 3 4 0

7 0 6 8 1

3 6 0 0 5

4 8 0 0 2

0 1 5 2 0

Edge 1:(1 3) cost:3

Edge 2:(1 4) cost:4

Edge 3:(4 5) cost:2

Edge 4:(5 2) cost:1

Minimum cost=10

**3. Dijkstra’s Algorithm**

**CODE**

#include<iostream>

#include<climits>

using namespace std;

int minimumDist(int dist[], bool Tset[])

{

int min=INT\_MAX,index;

for(int i=0;i<6;i++)

{

if(Tset[i]==false && dist[i]<=min)

{

min=dist[i];

index=i;

}

}

return index;

}

void Dijkstra(int graph[6][6],int src)

{

int dist[6];

bool Tset[6];

for(int i = 0; i<6; i++)

{

dist[i] = INT\_MAX;

Tset[i] = false;

}

dist[src] = 0;

for(int i = 0; i<6; i++)

{

int m=minimumDist(dist,Tset);

Tset[m]=true;

for(int i = 0; i<6; i++)

{

if(!Tset[i] && graph[m][i] && dist[m]!=INT\_MAX &&

dist[m] + graph[m][i] < dist[i])

dist[i]=dist[m]+graph[m][i];

}

}

cout<<"Vertex\t\tDistance from source"<<endl;

for(int i = 0; i<6; i++)

{

char str=65+i;

cout<<str<<"\t\t\t"<<dist[i]<<endl;

}

}

int main()

{

int graph[6][6] ={{0,10,20,0,0,0},{0,0,0,50,10,0},

{0,0,0,20,33,0},{0,0,0,0,20,2},

{0,0,0,0,0,1},{0,0,0,0,0,0}};

Dijkstra(graph,0);

return 0;

}

**OUTPUT**

Vertex Distance from source

A 0

B 10

C 20

D 40

E 20

F 21

**CONCLUSION**

In this lab, we implemented greedy graph algorithms; Kruskal’s and Prim’s to find the minimum spanning tree and Dijkstra’s to find the shortest path from a node to all other nodes in a graph.

**1. Floyd-Warshall Algorithm**

**CODE**

#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix(int A[][nV]);

void floydWarshall(int graph[][nV])

{

int A[nV][nV], i, j, k;

for (i = 0; i < nV; i++)

for (j = 0; j < nV; j++)

A[i][j] = graph[i][j];

for (k = 0; k < nV; k++)

{

for (i = 0; i < nV; i++)

{

for (j = 0; j < nV; j++)

{

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

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

}

}

}

printMatrix(A);

}

void printMatrix(int A[][nV])

{

printf("The shortest path matrix:\n");

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

{

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

{

if (A[i][j] == INF)

printf("%4s", "INF");

else

printf("%4d", A[i][j]);

}

printf("\n");

}

}

int main()

{

int graph[nV][nV] = {{0, 6, INF, 11},

{2, 0, INF, 4},

{INF, 4, 0, INF},

{INF, INF, 5, 0}};

floydWarshall(graph);

}

**OUTPUT**

The shortest path matrix:

0 6 15 10

2 0 9 4

6 4 0 8

11 9 5 0

**2. Travelling Salesman Problem**

**CODE**

#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

void takeInput()

{

int i,j;

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

scanf("%d",&n);

printf("\nEnter the Cost Matrix\n");

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

{

printf("\nEnter Elements of Row: %d\n",i+1);

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

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

completed[i]=0;

}

printf("\n\nThe cost list is:");

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

{

printf("\n");

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

printf("\t%d",ary[i][j]);

}

}

void mincost(int city)

{

int i,ncity;

completed[city]=1;

printf("%d--->",city+1);

ncity=least(city);

if(ncity==999)

{

ncity=0;

printf("%d",ncity+1);

cost+=ary[city][ncity];

return;

}

mincost(ncity);

}

int least(int c)

{

int i,nc=999;

int min=999,kmin;

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

{

if((ary[c][i]!=0)&&(completed[i]==0))

if(ary[c][i]+ary[i][c] < min)

{

min=ary[i][0]+ary[c][i];

kmin=ary[c][i];

nc=i;

}

}

if(min!=999)

cost+=kmin;

return nc;

}

int main()

{

takeInput();

printf("\n\nThe Path is:\n");

mincost(0); //passing 0 because starting vertex

printf("\n\nMinimum cost is %d\n ",cost);

return 0;

}

**OUTPUT**

Enter the number of cities: 4

Enter the Cost Matrix

Enter Elements of Row: 1

1 5 2 3

Enter Elements of Row: 2

2 5 8 2

Enter Elements of Row: 3

4 1 8 4

Enter Elements of Row: 4

1 5 0 3

The cost list is:

1 5 2 3

2 5 8 2

4 1 8 4

1 5 0 3

The Path is:

1--->4--->2--->3--->1

Minimum cost is 20

**CONCLUSION**

In this lab, I implemented Floyd-Warshall algorithm to find the all pairs shortest path in the given graph and implemented the travelling salesman problem to find the shortest path.