**Techno India NJR Institute of Technology**



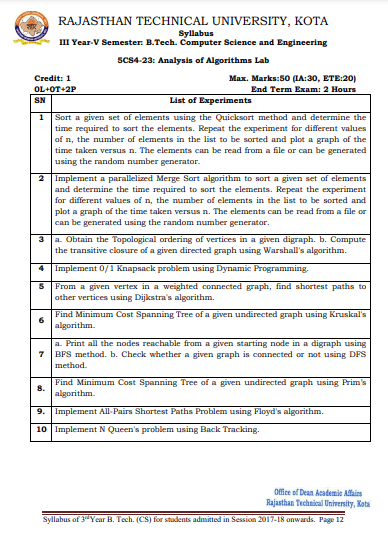
**Lab Course File**

**Analysis of Algorithm Lab (5CS4-22)**

Kirti Dashora

(Assistant Professor)

**Department of CSE**

****

**Course Overview :**

Learn how to analyze a problem and design the solution for the problem.

I. Design and implement efficient algorithms for a specified application.

II. Strengthen the ability to identify and apply the suitable algorithm for the given real world problem.

**Course Outcomes:**

|  |  |  |
| --- | --- | --- |
| **CO.NO.** | **Cognitive Level** | **Course Outcome** |
| 1 | **Comprehension** | Develop an ability to understand basic techniques for designing algorithms, including the techniques of recursion, divide-and-conquer, and greedy |
| 2 | **Application** | Apply the knowledge of algorithms to analyze the programming problems and find the optimal solution to solve a problem |
| 3 | **Synthesis** | To create efficient computer algorithms |

**Prerequisites:**

1. Fundamentals knowledge of Data Structures.
2. Fundamentals knowledge of C and C++.

**Course Outcome Mapping with Program Outcome:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Outcome** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | 3 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **CO2** | 3 | 3 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| **CO3** | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |

**Mapping Justification**

|  |  |  |
| --- | --- | --- |
| **CO** | **PO** | **Justification** |
| CO1,  CO2,  CO3, | **PO1** | To design new algorithms using divide and conquer, greedy technique .that’s why CO1,CO2 and CO3 are highly mapped. |
| **PO2** | To analyze the time and space complexity of divide and conquer algorithms ,dynamic programming etc. that ‘s why CO1,CO2,CO3 are highly mapped. |
| **PO3** | To apply the concept of divide and conquer technique in searching and sorting etc. problem types and concept of greedy technique in finding minimum spanning tree in case of ecommerce applications .that’s why CO1,CO2 are moderately and CO3 are highly corelated |
| **PO4** | To apply the object oriented concepts in real world problems such as multithreading. . that’s why CO2 and CO3 are moderately mapped and CO1 and CO2 are mapped at low level. |

**Course Coverage Module Wise:**

|  |  |
| --- | --- |
| **Sr No** | **Experiment Name** |
| **1** | To perform Binary Search |
| **2** | To find Optimal solution for a Knap Sack Problem using Greedy Method |
| **3** | To find Adjacency matrix for a given graph |
| **4** | To implement Depth First Search |
| **5** | To implement Breadth First Search |
| **6** | To find minimum cost of spanning tree using Prims Algorithm |
| **7** | To write a program to implement Dijkstras algorithm |
| **8** | To write a program to perform All pairs shortest path problem |
| **9** | To perform Matrix Chain Multiplication |
| **10** | To write a program to solve Eight Queens problem using Back Tracking Technique |

**Program No** 1

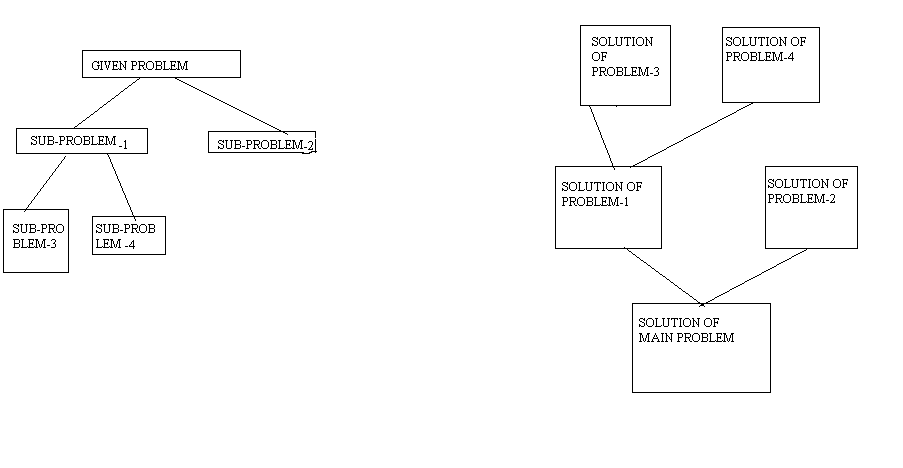
**Aim**:

To perform Binary Search

**Description**:

In Binary Search we use Divide & Conquer principle.

**General principle of Divide & Conquer:** If a problem is given we divide it into no. of sub-problems if we get solution of each part then we stop at that point. Otherwise we still divide the problem as we solve all individual sub-problems at last combine all these solution which gives solution of main problem.



**Algorithm:**

Step 1: if (low > high) then return -1

Step 2: if (low < high) the mid=(low + high)/2

Step 3: X be a key. If a[mid] = X then return mid

Step 4: If a[mid] > X then search for X from a[low] to a[mid-1]

Step 5: If a[mid] < X then search for X from a[mid + 1] to a[high]

**Source Code**:

#include<stdio.h>

#include<conio.h>

void main()

{

int a[10],n,i,j,temp;

int beg,end,mid,target;

clrscr();

printf("Enter the total numbers:");

scanf("%d",&n);

printf("Enter the array elements:" );

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

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

beg=0;

end=n;

mid=(beg+end)/2;

printf("\nEnter the number to be searched:");

scanf("%d",&target);

while(beg<=end && a[mid]!=target)

{

if(target<a[mid])

end=mid-1;

else

beg=mid+1;

mid=(beg+end)/2;

}

if(a[mid]==target)

{

printf("\nThe number is found at position %2d",mid);

}

else

{

printf("\nThe number is not found");

}

getch();

}

**OutPut**:

Enter the total numbers:5

Enter the array elements:1 2 3 4 5

Enter the number to be searched: 5

The number is found at position 4

**Viva –Voce Questions**

1. What is the advantage of recursive approach than an iterative approach?

2. What is the worst case complexity of binary search using recursion?

3.  Which is not an application of binary search?

4. Binary Search can be categorized into which design technique?

5. What is the time complexity of uniform binary search?

**Program No 2**

**Aim**:

To find Optimal solution for a Knap Sack Problem using Greedy Method

**Description**:

Greedy method or technique is used to solve Optimization problems. A solution that can be maximized or minimized is called Optimal Solution.

**Knapsack problem:** In Knapsack problem we are given:1) n objects 2) Knapsack with capacity m, 3) An object i is associated with profit Wi , 4) An object i is associated with profit Pi , 5) when an object i is placed in knapsack we get profit Pi Xi .

Here objects can be broken into pieces (Xi Values)

The Objective of Knapsack problem is to maximize the profit .

**Algorithm:**

Algorithm GreedyKnapsack(m,n)

//p[1:n] and w[1:n] contain the profits and weights respectively

//of the n objects ordered such that p[i] / w[i] > = p[i+1] / w[i+1]

//m is the knapsack size and x[1:n] is the solution vector

{

for i := 1 to n do x[i] := 0.0; //Initialize x.

U :=m;

for i :=1 to n do

{

if (w[i] > U) the break;

x[i] := 1.0; U := U – w[i];

}

if (i<=n) then x[i] := U/w[i];

**Source Code**:

#include<stdio.h>

#include<conio.h>

main()

{

clrscr();

int n,m,i,u;

int p[20],w[20];

float x[20];

float optimal=0.0;

printf("Enter number of objects:");

scanf("%d",&n);

printf("Enter capacity of KnapSack:");

scanf("%d",&m);

printf("Enter profits in decreasing order of Pi/Wi:");

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

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

printf("Enter Weights in decreasing order of Pi/Wi:");

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

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

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

x[i]=0.0;

u=m;

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

{

if(w[i]>u)

break;

else

x[i]=1.0;

u=u-w[i];

}

if(i<=n)

x[i]=(float)u/w[i];

printf("The x values are\n");

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

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

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

optimal=optimal+p[i]\*x[i];

printf("\nOptimal Solution is %f",optimal);

getch();

}

**OutPut**:

Enter the number of objects: 3

Enter the capacity of Knapscak:20

Enter profits in decreasing order of pi/wi : 16 15 14

Enter weights in decreasing order of pi/wi : 10 10 10

The x-values are :

1.000000 1.000000 0.000000

Optimal Solution is : 31.000000

**Viva-Voce Question**

1.  What is the objective of the knapsack problem?

2. What is time complexity of fractional knapsack problem?

3.  The main time taking step in fractional knapsack problem is

4.  Which algorithms is the best approach for solving Huffman codes?

**Program No** 3

**Aim**:

To find Adjacency matrix for a given graph

**Algorithm:**

GraphRepresentation:

– A graph *G* = (*V, E*) consists of a set of vertices

*V* = {*vi* | *i = 1, 2, …, n*} and a set of edges

*E* = {*ej* | *j = 1, 2, …, m*} that connect the vertices.

• An edge *e* may be represented by the pair (*u*, *v*) where *u* and *v* are the vertices being connected by *e*.If an edge is present between two vertices then that position is set to 1 in the matrix i.e adjacency marix else zero.In undirected graph if (u, v) be the vertex then both the positions of (u,v) and (v,u) is set to 1 but in the case of the directed graph only the position of the (u, v) is set to one.

**Directed and undirected graphs :**

• An undirected graph, *G* = <*V,E*> is the pair:

– *V* = set of distinct vertices

– *E* = set of edges; each member is a set of 2 vertices

Example:

*V* = {*t, u, v, w, x, y, z*}

*E* = {{*u, v*}, {*u, w*}, {*v, w*}, {*v, y*}, {*x, z*}}

• A directed graph, *G* = <*V,E*>:

– Is the same as an undirected graph except E is a set of

ordered pairs.

Example:

*E* = {<*a, b*>, <*a, c*>, <*c, b*>, <*b, e*>, <*e, b>}*

**Description**:

Let G= (V, E) be a graph with n vertices n>=1.The adjacency matrix of G is a two-dimensional n\*n array, say a, with the property that a [i, j] =1 iff the edge (i,j) (<i,j>) for a directed graph)is in E(G).The element a[i,j]=0 if there is no such edge in G.The adjacency matrix for an undirected graph is symmetric, as the edge (i, j) is in E (G).The adjacency for a directed graph may not be symmetric. The space needed to represent a graph using its adjacency matrix is n^2 bits. About half the space can be saved in the case of an undirected graph by storing only the upper or lower of the triangle of the matrix. From the adjacency matrix, we can readily determine whether there is an edge connecting any two vertices i and j.For an undirected graph the degree of any vertex i is its row sum: For a directed graph the row sum is the out-degree, and the column sum is the in-degree.

**Source Code**:

**a)Directed**

#include<stdio.h>

#include<conio.h>

main()

{

clrscr();

int num;

int me;

int j=0,k;

int beg,des;

int adj[20][20]={0};

printf("Directed Graph\n");

printf("Enter number of nodes:");

scanf("%d",&num);

me=num\*(num-1)/2;

while(j<me)

{

printf("Enter the Edges:");

scanf("%d%d",&beg,&des);

if(beg==0 && des==0)

{

break;

}

if(beg>num || des>num || beg<=0 || des<=0)

{

printf("Invalid Edges");

}

else

{

adj[beg][des]=1;

j++;

}

}

printf("Adjacency Matrix is\n");

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

{

for(k=1;k<=num;k++)

{

printf("%d\t",adj[j][k]);

}

printf("\n");

}

getch();

}

**OutPut**:

Directed Graph

Enter number of nodes : 4

Enter the edges : 1 2

Enter the edges : 2 3

Enter the edges : 3 4

Enter the edges : 4 1

Enter the edges : 0 0

Adjacency matrix is:

1 0 0 0

0 0 1 0

0 0 0 1

1 0 0 0

**b)Undirected**

#include<stdio.h>

#include<conio.h>

main()

{

clrscr();

int num;

int me;

int j=0,k;

int beg,des;

int adj[20][20]={0};

printf("Enter number of nodes:");

scanf("%d",&num);

me=num\*(num-1);

while(j<me)

{

printf("Enter the Edges:");

scanf("%d%d",&beg,&des);

if(beg==0 && des==0)

{

break;

}

if(beg>num || des>num || beg<=0 || des<=0)

{

printf("Invalid Edges");

}

else

{

adj[beg][des]=1;

adj[des][beg]=1;

j++;

}

}

printf("Adjacency Matrix is\n");

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

{

for(k=1;k<=num;k++)

{

printf("%d\t",adj[j][k]);

}

printf("\n");

}

getch();

}

**OutPut**:

Enter number of nodes : 4

Enter the edges : 1 2

Enter the edges : 2 3

Enter the edges : 3 4

Enter the edges : 4 1

Enter the edges : 0 0

Adjacency matrix is:

1 0 1

1 0 1 0

0 1 0 1

1 0 1 0

**Viva-Voce Questions**

1. Find number of elements in the adjacency matrix of a graph having 7 vertices ?

2.  What is the time complexity to calculate the number of edges in a graph whose information in stored in form of an adjacency matrix ?

3. What is the maximum number of possible non zero values in an adjacency matrix of a simple graph with n vertices?

**Program No 4**

**Aim**:

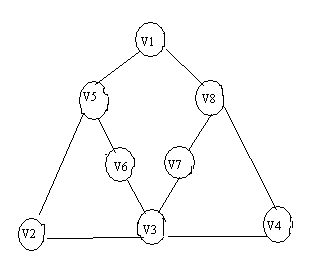
To implement Depth First Search

**Description**:

**Graph Traversal :**It is the visiting of nodes in a graph. It is of two types: 1.Depth First Search (DFS), 2.Breadth First Search (BFS)

**Depth First Search(DFS) :**This method starts at any node in a graph as visited & recursively traverse remaining adjacent nodes.

**Example:**



V1, V5, V2 , V3, V4, V8 ,V7, V6 .

**Algorithm:**

// Given an undirected(directed) graph G(V,E) with

//n vertices and a n array visited[] initially set

//to zero ,this algorithm visits all vertices

//reachable from v. G and visited[] are global.

{

visited[v]:=1;

for each vertex w adjacent from v do

{

if ( visited[w] = 0) then DFS(w);

}  
}

**Source Code**:

#include<stdio.h>

#include<conio.h>

int i,j,k,n;

int graph[20][20];

int visited[20];

void dfs(int);

void main()

{

clrscr();

printf("Enter number of nodes:");

scanf("%d",&n);

printf("Enter the adjacency matrix\n");

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

{

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

{

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

}

visited[i]=0;

}

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

{

if(visited[i]==0)

{

dfs(i);

}

}

getch();

}

void dfs(int l)

{

visited[l]=1;

printf("Node visited is %d\n",l);

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

{

if(visited[k]==0 && graph[l][k]==1)

{

dfs(k);

}

}

}

**OutPut**:

Enter number of nodes : 3

Enter the Adjacency matrix :

0 1 1

1 0 0

1 0 0

Node Visited is : 1

Node Visited is : 2

Node Visited is : 3

**Viva-Voce Questions**

1. Depth First Search is equivalent to which of the traversal in the Binary Trees?

2. What is time Complexity of DFS ?

3. Which data structure is used in standard implementation of Breadth First Search?

4. A person wants to visit some places. He starts from a vertex and then wants to visit every vertex till it finishes from one vertex, backtracks and then explore other vertex from same vertex. What algorithm he should use?

5. When the Depth First Search of a graph is unique?

**Program No 5**

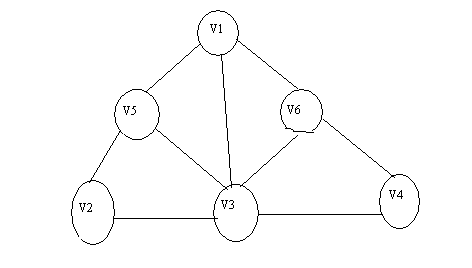
**Aim**:

To implement Breadth First Search

**Description**:

**Breadth First Search(BFS) :**BFS algorithm begins at root node & visits all the neighbouring nodes then for each of these nearest nodes it visits their un visited neighbour nodes . This process terminates when all nodes visited.

**Example:**



V1, V3, V5, V6, V2, V4.

We implement BFS by using queue data structure where as DFS by using stack

**Algorithm:**

// A breadth first search of G is carried out beginning

//at vertex v. For any node i visited[i]=1 if i has

//already been visited. The Graph G and array visited[]

//are global; visited[] is initialized to zero.

{

u:=v; //q is a queue of unexplored vertices.

visited[v]:=1;

repeat

{

for all vertices w adjacent from u do

{

if(visited[w] = 0) then

{

Add w to q; //w is unexplored

visited[w]:=1 ;

}

}

if q is empty then return //no unexplored vertex

Delete u from q; //Get first unexplored vertex.

}until(false);

}

**Source Code**:

#include<stdio.h>

#include<conio.h>

#define max 10

int n,adj[max][max],visited[max];

void bfs();

void readmatrix();

main()

{

int source;

clrscr();

printf("Enter the source node:");

scanf("%d",&source);

readmatrix();

bfs(source);

getch();

}

void readmatrix()

{

int i,j;

printf("Enter number of nodes:");

scanf("%d",&n);

printf("Enter adjacency matrix\n");

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

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

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

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

visited[i]=0;

}

void bfs(int source)

{

int queue[max];

int i,front,rear,root;

printf("The Order is\n");

front=rear=0;

visited[source]=1;

queue[rear++]=source;

printf("%d ",source);

while(front!=rear)

{

root=queue[front];

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

if(adj[root][i] && ! visited[i])

{

visited[i]=1;

queue[rear++]=i;

printf("%d ",i);

}

front++;

}}

**OutPut**:

Enter the Source node:1

Enter number of nodes : 6

Enter Adjacency matrix:

0 0 1 0 1 1

0 0 1 0 1 0

1 1 0 1 1 1

0 0 1 0 0 1

1 1 1 0 0 0

1 0 1 1 0 0

**The order is**

1 3 5 6 2 4

**Viva-Voce Questions**

**1.** Which data structure is used in standard implementation of Breadth First Search ?

2. What is time Complexity of Breadth First Search?

3. The Breadth First Search traversal of a graph will result into?

4.  What are applications of Breadth First Search?

**Program No 6**

**Aim**:

To find minimum cost of spanning tree using Prims Algorithm

**Description**:

Let G(V,E) be an undirected graph.

A sub-graph T=(V,E1) is said to be a spanning tree of G if T is a tree.

A Minimum cost spanning tree is a spanning tree with minimum weight.

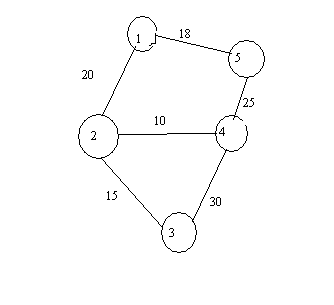
**Prim’s method:**

The Greedy method to obtain MCST builds the tree edge by edge.

Choose the edges with minimum cost

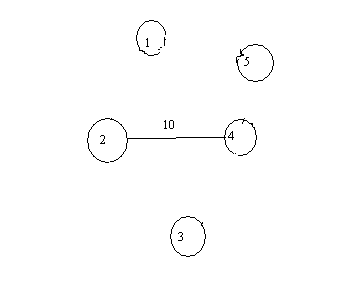
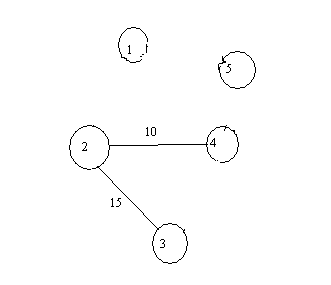
Cycles should be avoided.

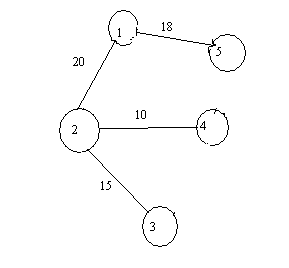
**Graph:**

****

**TOTAL COST:118**

**Stages are:**



TOTAL COST :63

**Algorithm:**

Prim(E,cost,n,t)

//E is the set of edges in G.cost [1:n,1:n] is the cost

//adjacency matrix of an n vertex graph such that cost[i,j] is

//either a positive real number or infinity if no edge of (i,j) exists

//a minimum spanning tree is computed and stored as a set of

//edges in the array t[1:n-1,1:2] . (t[i,1],t[i,2] is an edge in the minimum cost spanning

//tree .The final cost is returned.

{

Let (k,l) be an edge of minimum cost in Ei

mincost :=cost[k,l];

t[1,1]:=k;t[1,2]:=l;

for i:=1 to n do //initialize near

if(cost[i,l]<cost[i,k]) then near[i] :=l;

else

near[i]:=k;

near[k]:=near [l]:=0;

for i:=2 to n-1 do

{

// Find n-2 additional edges for t

Let j be an index such that near[j]!=0and

Cost[j,near[j]] is minimum

t[i,1]:=j;t[i,2]:=near[j];

mincost :=mincost+cost[j,near[j]];

near[j]:=0;

for k:=1 to n do //Upnear[].

if((near[k]!=0) and (cost[k,near[k]]>cost[k,j]))

then near[k]:=j;

}

return mincost;

}

**Source Code**:

#include<stdio.h>

#include<conio.h>

int g[20][20],d[20],visited[20],p[20];

int v,e;

int i,j;

void creategraph();

void prim();

void main()

{

clrscr();

creategraph();

prim();

getch();

}

void creategraph()

{

int a,b,w;

printf("Enter number of vertices:");

scanf("%d",&v);

printf("Enter number of edges:");

scanf("%d",&e);

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

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

g[i][j]=0;

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

p[i]=0;

visited[i]=0;

d[i]=32767;

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

{

printf("Enter edges a,b and w:");

scanf("%d%d%d",&a,&b,&w);

g[a][b]=g[b][a]=w;

}

}

void prim()

{

int current=1,totalvisited=1,min,mincost=0;

visited[current]=1;

d[current]=0;

while(totalvisited!=v)

{

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

{

if(g[current][i]!=0)

if(visited[i]==0)

if(d[i]>g[current][i])

{

d[i]=g[current][i];

p[i]=current;

}

}

min=32767;

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

{

if(visited[i]==0)

if(d[i]<min)

{

min=d[i];

current=i;

}

}

visited[current]=1;

totalvisited++;

}

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

mincost=mincost+d[i];

printf("Minimum cost of the spanning tree is %d",mincost);

}

**OutPut**:

Enter number of vertices : 3

Enter number of edges : 3

Enter edges a , b &w: 1 2 10

Enter edges a , b &w: 2 3 20

Enter edges a , b &w: 3 1 30

Minimum cost of the spanning tree is 30

**Viva-Voce Question**

1. What is the worst case time complexity of Prim’s algorithm if adjacency matrix is used?

2. Prim’s algorithm belongs to which design technique?

3. Prim’s algorithm can be efficiently implemented using Which search for graphs with greater density.

4.What are applications of Prim’s algorithm.

**Program No 7**

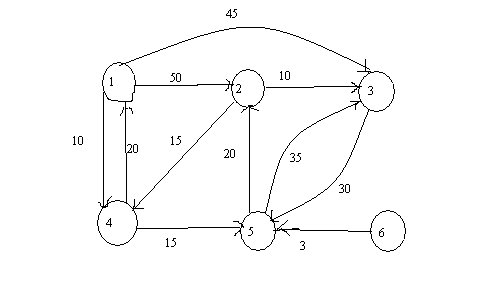
**Aim**:

To write a program to implement Dijkstras algorithm

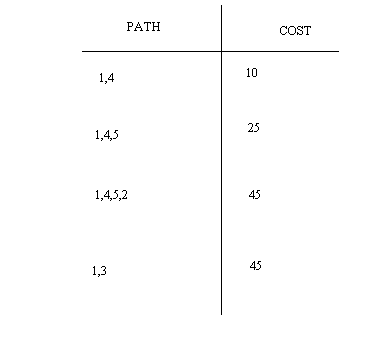
**Description**:

Let G (V,E) be a graph , V-Vertices & E-Edges .We have to choose one vertex as source , the problem aim is to find out minimum distance between source node and all remaining nodes.

This problem is the case of ordered paradigm in Greedy method.



This problem can be implemented by an algorithm called Dijkastra’s algorithm.



**Algorithm**:

//G be a graph

//Cost matrix [1:n,1:n] for the graph G

//S={set of vertices that path already generated}

//Let V be source vertex

//dist[j];1<=j<=n denotes distance between V and j

void main()

{

for i:=1 to n do

{

s[i]=false; // initialize s with n

dist[i]=cost[v,i]; //define distance

}

s[v]=true; //put v in s

dist[v]=0.0; //Distance between v and v is 0

for num:=2 to n-1 do

{

paths from v //choose u from among those vertices not in S such that dist[u]=min;

s[u]=true;

for(each w adjascent to u with s[w]=false)

if(dist[w]>dist[u]+cost[u,w])

then

dist[w]=dist[u]+cost[u,w]; //upthe distance

}

}

**Source Code**:

#include<stdio.h>

#include<conio.h>

#define infinity 32767

int cost[20][20],n,dist[20],s[20],a[20][20];

void setdata();

void getdata();

void path(int);

void setdata()

{

int i,j,k;

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

scanf("%d",&n);

printf("Enter Adjacency Matrix: ");

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

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

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

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

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

{

if(i==j)

cost[i][i]=0;

else if (a[i][j]!=0)

{

printf("\nEnter cost from %d to %d: ",i,j);

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

}

else

cost[i][j]=infinity;

}

}

void getdata()

{

int i;

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

if(dist[i]==32767)

printf("not reachable");

else

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

}

void path(int v)

{

int i,j,min,u;

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

{

s[i]=0;

dist[i]=cost[v][i];

}

s[v]=1;

dist[v]=0;

for(i=2;i<=n;i++)

{

min=32767;

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

if(s[j]==0 && dist[j]<min)

u=j;

s[u]=1;

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

if(s[j]==0 && a[u][j]==1)

if(dist[j]>dist[u]+cost[u][j])

dist[j]=dist[u]+cost[u][j];

}

}

void main()

{

int v;

clrscr();

setdata();

printf("\nEnter the source vertex: ");

scanf("%d",&v);

path(v);

printf("\nShortest paths " );

getdata();

getch();

}

**Output**:

Enter number of nodes: 6

Enter Adjacency Matrix: 0 1 1 1 0 0

0 0 1 1 0 0

0 0 0 0 1 0

1 0 0 0 1 0

0 1 1 0 0 0

0 0 0 0 1 0

Enter cost from 1 to 2: 50

Enter cost from 1 to 3: 45

Enter cost from 1 to 4: 10

Enter cost from 2 to 3: 10

Enter cost from 2 to 4: 15

Enter cost from 3 to 5: 30

Enter cost from 4 to 1: 20

Enter cost from 4 to 5: 15

Enter cost from 5 to 2: 20

Enter cost from 5 to 3: 35

Enter cost from 6 to 5: 3

Enter the source vertex: 1

Shortest paths 0 45 45 10 25not reachable

**Viva-Voce Questions**

1. Which is the most commonly used data structure for implementing Dijkstra’s Algorithm?

2. What is the time complexity of Dijikstra’s algorithm?

3. How many priority queue operations are involved in Dijkstra’s Algorithm?

4. How many times the insert and extract min operations are invoked per vertex?

5. What is running time of Dijkstra’s algorithm using Binary min- heap method?

**Program No 8**

**Aim**:

To write a program to perform All pairs shortest path problem

**Description**:

Let G(V,E) be a directed graph ,with V-Vertices & E-Edges

The objective of all pairs shortest path problem is to find out matrix A(i,j) where i,j is the shortest path from i to j

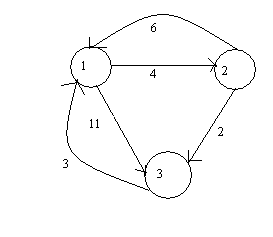
cost(i,j) = cost of edge <i,j> &

cost(i,i)=0

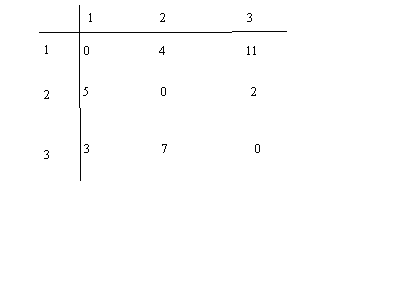
cost(i,j)=infinity //no such edge exists

A0 (i,j) =cost(i,j)

If K is an intermediate vertex then Ak (i,j)=min{Ak-1(i,j),Ak-1(i,k)+Ak-1(k,j)}



Therefore Solution is :



**Algorithm**:

AllPaths(cost,A,n)

//cost[1:n,1:n] is the cost adjacency matrix of a graph with

//n vertices;A[I,j] is the cost of a shortest path from vertex

//i to vertex j.cost[i,i] = 0.0 for 1<=i<=n.

{

for i:= 1 to n do

for j:= 1 to n do

A[i,j]:=cost[i,j]; // Copy cost into A

for k:= 1 to n do

for i:= 1 to n do

for j:= 1 to n do

A[i,j]:=min(A[i,j],A[i,k] + A[k,j]);

}

**Source Code**:

#include<stdio.h>

#include<conio.h>

int cost[20][20],n,a[20][20];

void setdata();

void getdata();

void path();

void setdata()

{

int i,j,k;

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

scanf("%d",&n);

printf("\nEnter cost matrix(32767 for infinity):");

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

{

printf("\nEnter %d row\n",i);

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

{

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

}

}

}

void getdata()

{

int i,j;

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

{

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

{

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

printf("\n");

}

}

}

void path()

{

int i,j,k,l;

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

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

a[i][j]=cost[i][j];

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

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

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

{

l=a[i][k]+a[k][j];

a[i][j]=(a[i][j]>l)?l:a[i][j];

}

}

void main()

{

clrscr();

setdata();

path();

printf("\nMatrix with shortest path is:");

getdata();

getch();

}

**Output**:

Enter the number of nodes:3

Enter cost matrix(32767 for infinity):

Enter 1 row

0 4 11

Enter 2 row

6 0 2

Enter 3 row

3 7 0

Matrix with shortest path is:0

4

6

5

0

2

3

7

0

**Viva-Voce Questions**

1. Which type of problems can be solved using Floyd Warshall’s Algorithm.

2. What is the running time of the Floyd Warshall Algorithm?

3. What approach is being followed in Floyd Warshall Algorithm?

4. What procedure is being followed in Floyd Warshall Algorithm?

5.  What happens when the value of k is 0 in the Floyd Warshall Algorithm?

**Program No** 9

**Aim**:

To perform Matrix Chain Multiplication

**Description**:

Let A,B &C are 3 matrices , since matrix multiplication is associative (AB)C=A(BC).The objective in matrix chain multiplication is to find out the paranthasization with less number of operations.

A=|1 2|

| 3 4|

B=|2 2 1|

|1 1 1|

C=|4 1|

|1 1|

|1 1|

(AB)C

AB= |4 4 3|

|10 10 7|

(AB)C=|23 11|

| 57 27|

A(BC)

BC==|11 5|

| 6 3|

A(BC) =|23 11|

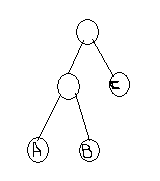
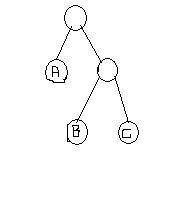
| 57 27|

PARANTHESIZATION COST CALCULATION TOTAL COST

(AB)C (2\*2\*3)+(2\*3\*2) 24

A(BC) (2\*3\*2)+(2\*2\*2) 20

Tree diagram for (AB)C Tree diagram for A(BC)

**Algorithm**:

*// n* is the number of matrices

// l is chain length

// Matrix Ai has the dimension p[i-1] x p[i]

Matrix-Chain-Order(int p[])

{

n = p.length - 1;

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

m[i,i] = 0;

for (l=2; l<=n; l++) {

for (i=1; i<=n-l+1; i++) {

j = i+l-1;

m[i,j] = MAXINT;

for (k=i; k<=j-1; k++) {

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;

}

}

}

}

**Source Code**:

#include<stdio.h>

#include<conio.h>

#define MAX 15

#define INF 4294967295

int num,p[MAX+1],n;

void print(int [][MAX],int,int);

void matrixchainorder();

void setdata();

void printorder();

void print(int s[MAX][MAX],int i,int j)

{

if(i==j)

printf("A%d",num++);

else

{

printf("(");

print(s,i,s[i][j]);

printf(" x ");

print(s,s[i][j]+1,j);

printf(" ) ");

}

}

void matrixchainorder()

{

unsigned int q;

unsigned long m[MAX][MAX]={0};

int s[MAX][MAX]={0};

int l,j,i,k;

for(l=2;l<=n;l++)

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

{

j=i+l-1;

m[i][j]=INF;

for(k=i;k<j;k++)

{

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;

}

}

}

printf("Number of Multiplications are %d ",m[1][n]);

num=1;

printf("Order of Multiplication is: ");

print(s,1,n);

}

void setdata()

{

int i;

printf("\nEnter number of matrices: ");

scanf("%d",&n);

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

{

printf("Enter %d matrix size:",i);

scanf("%d%d",&p[i-1],&p[i]);

}

}

void printorder()

{

int i,j;

matrixchainorder();

}

void main()

{

setdata();

printorder();

}

**Output**:

Enter number of matrices: 3

Enter 1 matrix size:2 2

Enter 2 matrix size:2 3

Enter 3 matrix size:3 2

Number of Multiplications are 20 Order of Multiplication is: (A1 x (A2 x A3 ) )

**Viva –Voce Question**

1. Which methods can be used to solve the matrix chain multiplication problem?

2. Consider the two matrices P and Q which are 10 x 20 and 20 x 30 matrices respectively. What is the number of multiplications required to multiply the two matrices?

3. What is the time complexity of dynamic programming implementation of the matrix chain problem?

4. Consider the matrices P, Q and R which are 10 x 20, 20 x 30 and 30 x 40 matrices respectively. What is the minimum number of multiplications required to multiply the three matrices?

5. What is space complexity of matrix chain multiplication

**Program No 10**

**Aim**:

To write a program to solve Eight Queens problem using Back Tracking Technique

**Description**:

Many problems which deal with searching for a set of solutions or which ask for an optimal solution satisfying some constraints can be solved using the backtracking formulation.

Constraints in backtracking are of 2 types:

1.Explicit

2.Implicit

In backtracking desired solutions n tuple i.e (x1,x2....xn) where xi are choosen from given a finite set of Si (i.e) finding a vector that maximises (or) minimises a criterion function p(x1,x2...xn).

Explicit constraints:Each Xi should be taken from one of the Wi.

Implicit constraints:we have to avoid multiple instances of same sub-set.

**Algorithm**:

//Recursive algorithm for Backtracking

//Let X[] be an array of objects , n be the number of objects

//Output is to find out solution vector

{

for each X[k] ***E*** p(X[1],X[2],…..X[k-1]) // criterian function

do

{

if Bk(X[1],X[2],…..X[k]) != 0 // bounding function

then

if(X[1],X[2],…..X[k]) is in path then write X[1:k];

else

Back Track

}

}

**Source Code**:

#include <stdio.h>

#include<conio.h>

int row[8],s=0;

int safe(int,int);

void putboard();

void queen(int);

int safe(int x, int y)

{

int i;

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

if( row[y-i]==x || row[y-i]==x-i || row[y-i]==x+i)

return 0;

return 1;

}

void putboard()

{

int x,y;

printf("\nSolution # %d",++s);

printf(":\n---------------------------------\n");

for(y=0;y<8; y++)

{

for (x=0;x<8;x++)

if(x==row[y])

printf("| Q ");

else

printf("| ");

printf("|\n---------------------------------\n");

}

getch();

}

void queen(int y)

{

int x;

for(x=0;x<8;x++)

{

row[y-1]=x;

if( safe(x,y-1) )

if (y<8)

queen(y+1);

else

putboard();

}

}

void main()

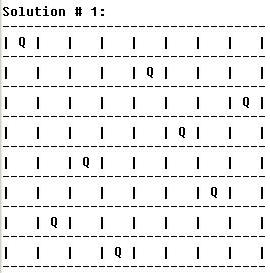
{

clrscr();

queen(1);

}

**Output**:



**Viva-Voce Question**

1. For how many queens was the extended version of Eight Queen Puzzle applicable for n\*n squares?

2.  How many solutions are there for 8 queens on 8\*8 board?

3.  In how many directions do queens attack each other?

4. Placing n-queens so that no two queens attack each other is called?

5.  Where is the n-queens problem implemented?