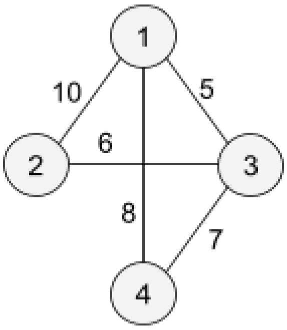
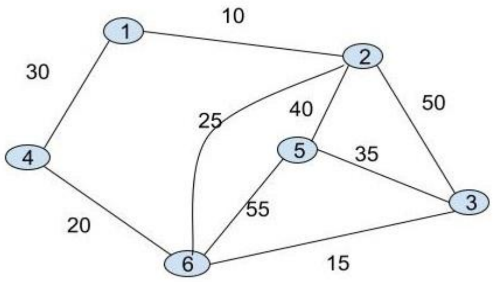
USN □□□□□□□□□□

CELEBRATING 30 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 2 – July 2024

Sub:	**ANALYSIS & DESIGN OF ALGORITHMS**				Sub Code:	**BCS401**	Branch:	**AIML/CSEAIML**
Date:		Duration:	**90 min**	Max Marks:	**50**	Sem/Sec:	**IV -A, B, C**	**OBE**

Answer any FIVE FULL Questions			**MARKS**	**CO**	**RBT**
1	a)	What is Dynamic Programming? What are the various problems that can be solved using dynamic programming?	5	CO3	L1
	b)	Mention the differences between Divide & Conquer and Dynamic Programming.	5	CO3	L1
2		Explain the following with examples: a) Complete Graph b) Directed Acyclic Graph c) Connected Graph Write the adjacency matrix & the cost adjacency matrix of the following graph:  	10	CO3	L2
3		What is Transitive Closure? Apply Warshall's algorithm to compute transitive closure of the digraph defined by the following adjacency matrix: $\begin{array}{\|c\|c\|c\|c\|}\hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 \\ \hline\end{array}$	10	CO3	L3
4		Write a C program to implement Floyd's algorithm.	10	CO3	L2
5		Write four spanning trees for the following graph. What is the cost of a Minimum Spanning Tree and obtain using Prim's algorithm?  1 2 3 4 5 6 10 30 25 40 50 35 55 20 15	10	CO4	L3
6		Write a C program to implement Dijkstra's Algorithm to obtain single source shortest paths to all other vertices.	10	CO4	L2

CI **CCI** **HOD**

---All the Best---

USN □□□□□□□□□□

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 2 – July 2024

Sub:	**ANALYSIS & DESIGN OF ALGORITHMS**				Sub Code:	**BCS401**	Branch:	**AIML/CSEAIML**
Date:	**11/07/24**	Duration:	**90 min**	Max Marks: **50**	Sem/Sec:	**IV -A, B, C**		**OBE**

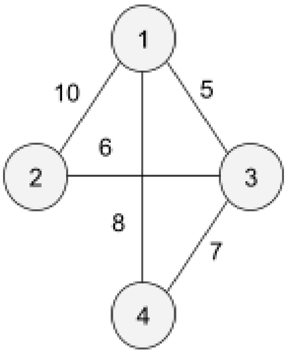
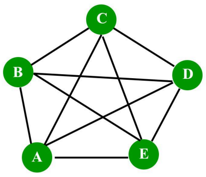
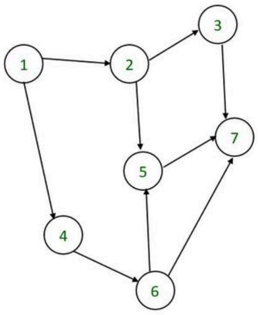
<table>
<tr><th colspan="3"><u>Answer any FIVE FULL Questions</u></th><th>MARKS</th><th>CO</th><th>RBT</th></tr>
<tr><td>1</td><td>a)</td><td>What is Dynamic Programming? What are the various problems that can be solved using dynamic programming?

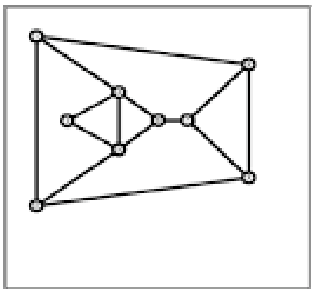
Ans: Dynamic programming is a method of solving the problem with overlapping subproblems. The method works by dividing the problem into subproblems and finding the solution for the same. The solution of the subproblems are used to obtain the solution for the problem. Once the sub-problem is solved, the answer is stored in a table and is never recalculated. When an instance of the sub-problem occurs, the answer is retrieved from the table, saving time.

The various problems that uses dynamic programming are:
- Fibonacci series
- Computing binomial coefficient
- Warshal’s algorithm
- Knapsack problem</td><td>5</td><td>CO3</td><td>L1</td></tr>
<tr><td></td><td>b)</td><td>Mention the differences between Divide & Conquer and Dynamic Programming.

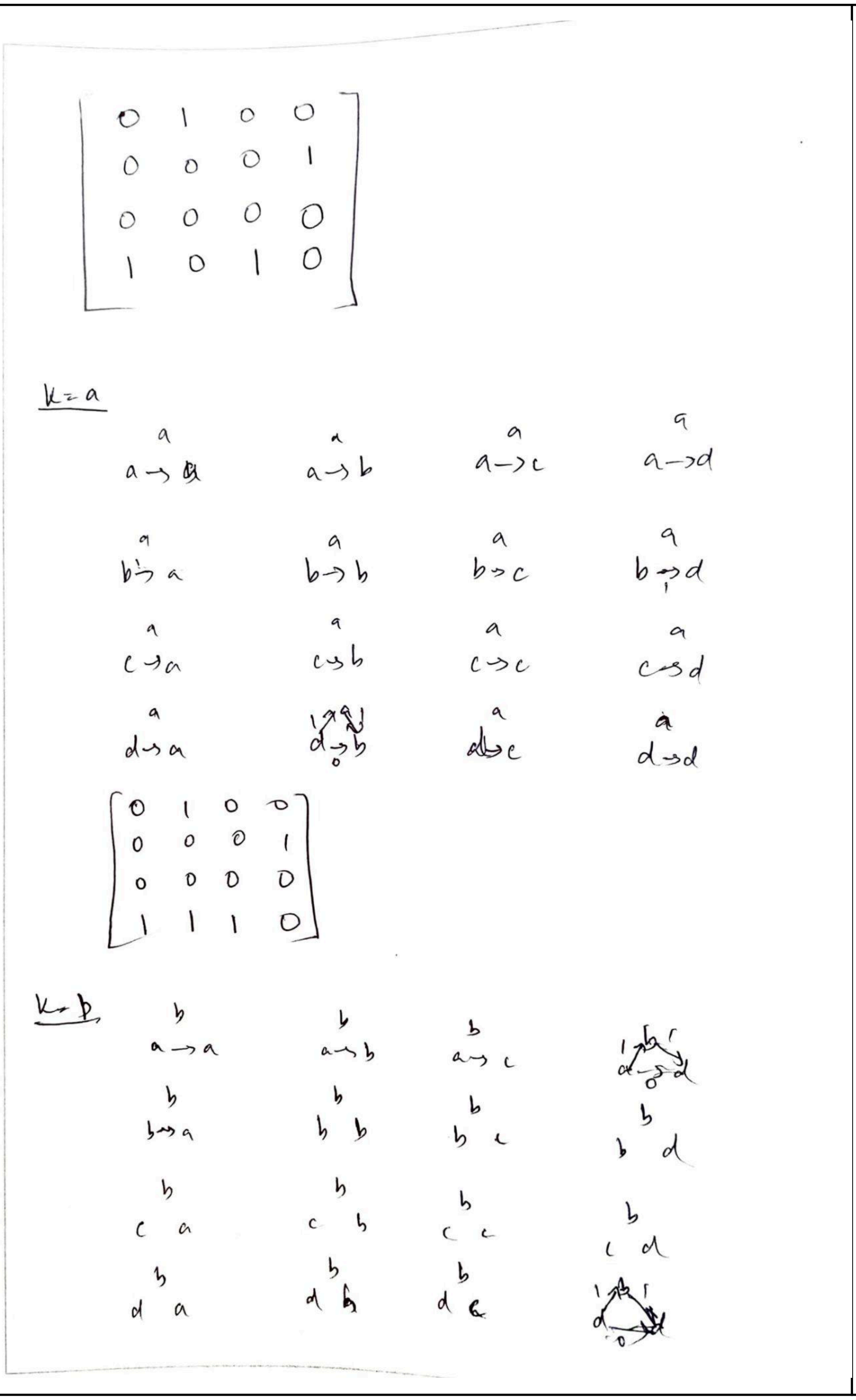
Ans:
<table><tr><th>DIVIDE & CONQUER</th><th>DYNAMIC PROGRAMMING</th></tr><tr><td>Applicable when subproblems are independent.</td><td>Applicable when subproblems are not independent.</td></tr><tr><td>Subproblems are solved separately and combined to get the solution of the original problem.</td><td>The original problem is solved by using the results of previous subproblems.</td></tr><tr><td>Every instance of the subproblem is recalculated.</td><td>Only one instance of the subproblem is computed and stored.</td></tr><tr><td>Uses Top-Down approach.</td><td>Uses Bottom-Up approach.</td></tr></table></td><td>5</td><td>CO3</td><td>L1</td></tr>
</table>

Not efficient.	More efficient.

2	Explain the following with examples: a) Complete Graph b) Directed Acyclic Graph c) Connected Graph Write the adjacency matrix & the cost adjacency matrix of the following graph:  Ans: a) Complete Graph A complete graph is a graph in which each vertex is connected to every other vertex. That is, a complete graph is an undirected graph where every pair of distinct vertices is connected by a unique edge.  Complete Graph b) Directed Acyclic Graph A DAG is a Directed Acyclic Graph, a type of graph whose nodes are directionally related to each other and don't form a directional closed loop.  c) Connected Graph A graph is a connected graph if, for each pair of vertices, there exists at least one single path which joins them.	10	CO3	L2

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>What is Transitive Closure? Apply Warshall’s algorithm to compute transitive closure of the digraph defined by the following adjacency matrix:
<table>
<tr><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td></tr>
</table>

Ans:
The transitive closure G* of a directed graph G is a graph that has an edge (u, v) whenever G has a directed path from u to v.
Let G = (V, E) be a simple graph where V is the set of vertices and E is the set of edges. Let N be the no. of vertices in graph G. The matrix P whose elements are given by
$P[i, j] \ = \ 1 \, if \ there \ is \ a \ path \ from \ vertex \ i \ to \ vertex \ j$
$P[i, j] \ = \ 0 \ Otherwise$
is called path matrix or Transitive closure.</td>
<td>10</td>
<td>CO3</td>
<td>L3</td>
</tr>
</table>



$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

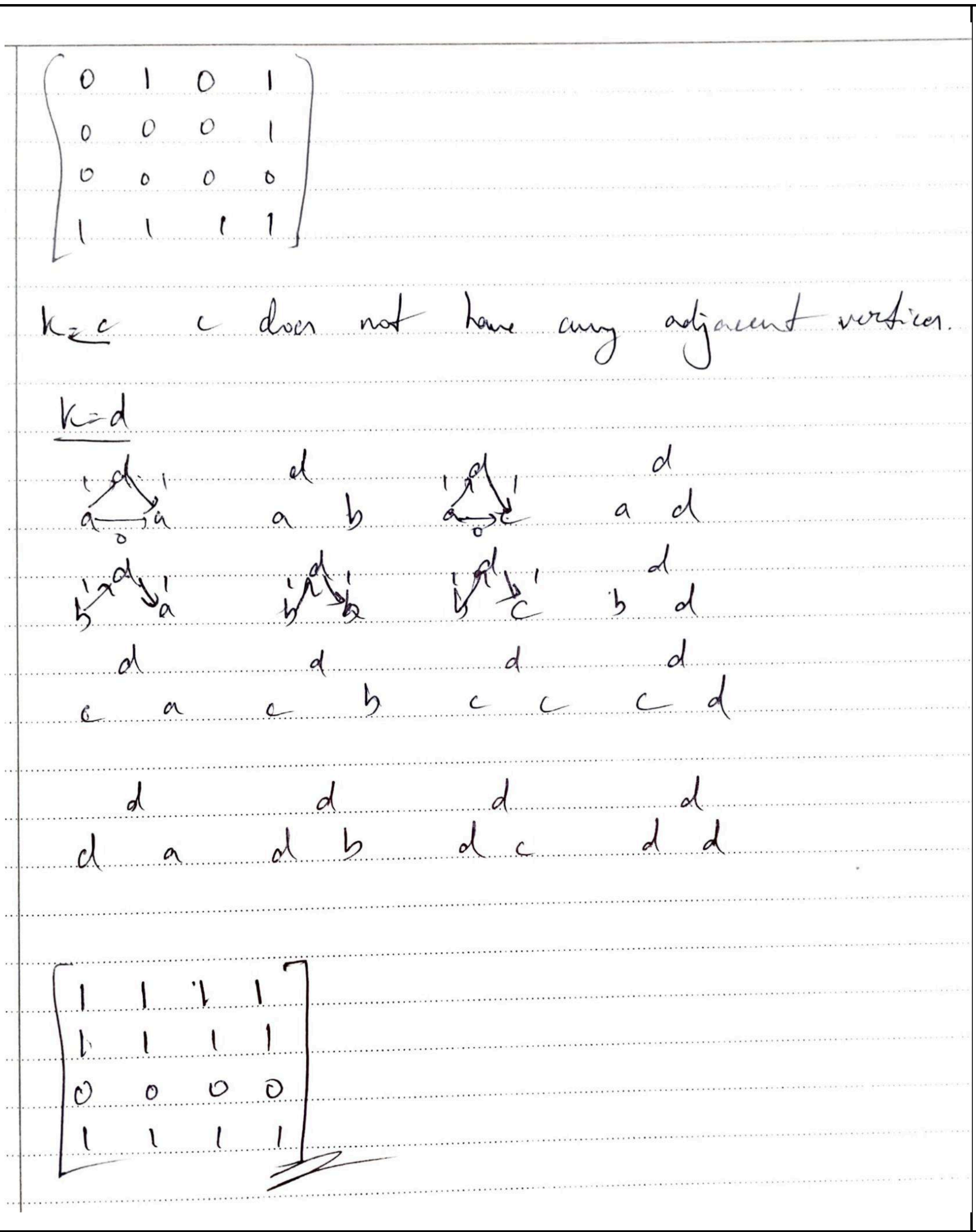
k=a

$\overset{a}{a \to a}$	$\overset{a}{a \to b}$	$\overset{a}{a \to c}$	$\overset{a}{a \to d}$
$\overset{a}{b \to a}$	$\overset{a}{b \to b}$	$\overset{a}{b \to c}$	$\overset{a}{b \to d}$
$\overset{a}{c \to a}$	$\overset{a}{c \to b}$	$\overset{a}{c \to c}$	$\overset{a}{c \to d}$
$\overset{a}{d \to a}$	$\overset{1\ a}{\underset{0}{d \to b}}$	$\overset{a}{d \to c}$	$\overset{a}{d \to d}$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

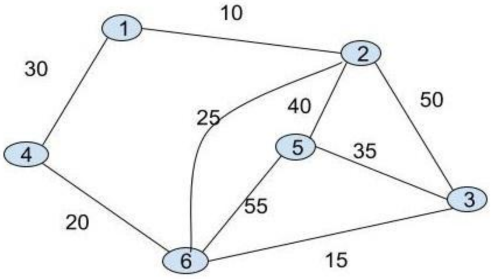
k=b

$\overset{b}{a \to a}$	$\overset{b}{a \to b}$	$\overset{b}{a \to c}$	$\overset{1\ b\ 1}{\underset{0}{a \to d}}$
$\overset{b}{b \to a}$	$\overset{b}{b\ \ b}$	$\overset{b}{b\ \ c}$	$\overset{b}{b\ \ d}$
$\overset{b}{c\ \ a}$	$\overset{b}{c\ \ b}$	$\overset{b}{c\ \ c}$	$\overset{b}{c\ \ d}$
$\overset{b}{d\ \ a}$	$\overset{b}{d\ \ b}$	$\overset{b}{d\ \ c}$	$\overset{1\ b\ 1}{\underset{0}{d\ \ d}}$

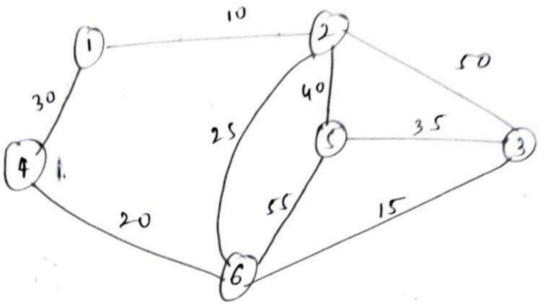
	 $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ k=c c does not have any adjacent vertices. k=d d a a d a b d a c d a d d b a d b b d b c d b d d c a d c b d c c d c d d d a d d b d d c d d d $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$			

4	Write a C program to implement Floyd's algorithm.
	
	Ans:
	// C Program for Floyd Warshall Algorithm
	#include <stdio.h>
	
	// Number of vertices in the graph
	#define V 4
	
	/* Define Infinite as a large enough
	  value. This value will be used
	  for vertices not connected to each other */
	#define INF 99999
	
	// A function to print the solution matrix
	void printSolution(int dist[][V]);
	
	// Solves the all-pairs shortest path
	// problem using Floyd Warshall algorithm
		10	CO3	L2

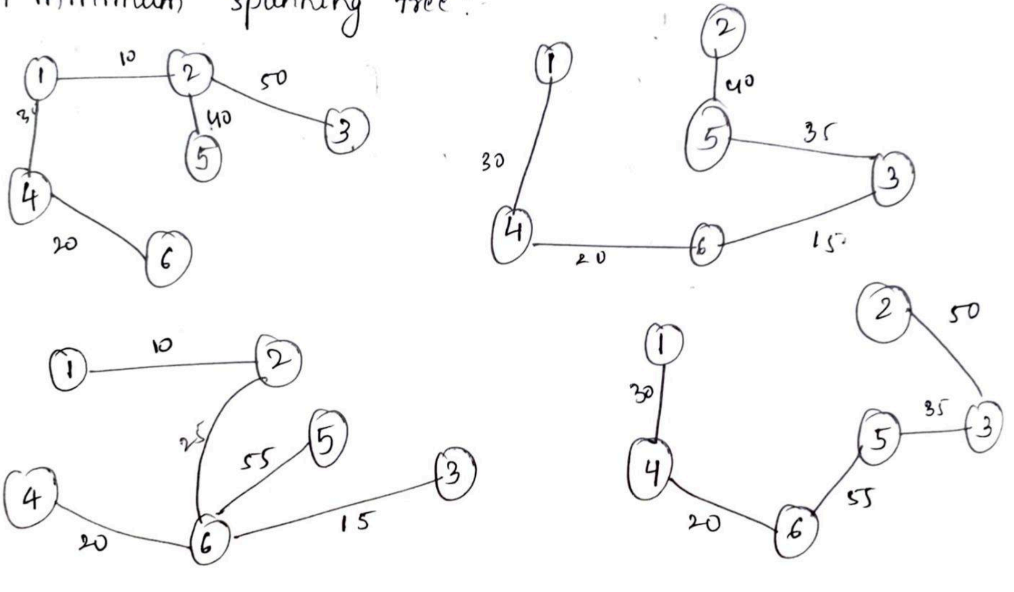
```
void floydWarshall(int dist[][V])
{
   int i, j, k;

   /* Add all vertices one by one to
     the set of intermediate vertices.
     ---> Before start of an iteration, we
     have shortest distances between all
     pairs of vertices such that the shortest
     distances consider only the
     vertices in set {0, 1, 2, .. k-1} as
     intermediate vertices.
     ----> After the end of an iteration,
     vertex no. k is added to the set of
     intermediate vertices and the set
     becomes {0, 1, 2, .. k} */
   for (k = 0; k < V; k++) {
      // Pick all vertices as source one by one
      for (i = 0; i < V; i++) {
         // Pick all vertices as destination for the
         // above picked source
         for (j = 0; j < V; j++) {
            // If vertex k is on the shortest path from
            // i to j, then update the value of
            // dist[i][j]
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
         }
      }
   }

   // Print the shortest distance matrix
   printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
   printf(
      "The following matrix shows the shortest distances"
      " between every pair of vertices \n");
   for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
         if (dist[i][j] == INF)
            printf("%7s", "INF");
         else
            printf("%7d", dist[i][j]);
      }
      printf("\n");
   }
}

// driver's code
int main()
{
  /* Let us create the following weighted graph
          10
     (0)------->(3)
      |         /|\
    5 |          |
      |          | 1
     \|/         |
     (1)------->(2)
          3           */
  int graph[V][V] = { { 0, 5, INF, 10 },
                      { INF, 0, 3, INF },
                      { INF, INF, 0, 1 },
                      { INF, INF, INF, 0 } };

  // Function call
  floydWarshall(graph);
  return 0;
}
```

5	Write four spanning trees for the following graph. What is the cost of a Minimum Spanning Tree and obtain using Prim's algorithm?  1 2 3 4 5 6 10 30 25 40 50 35 55 20 15 Ans:	10	CO4	L3

given G (V,E) :



4 minimum spanning tree :



Prim's algorithm:-

S1 : Sellct an arbitary vertex (v)

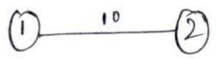
S2 : Find all the adjacent vertixes of v

S3 : Select the vertex with minimum cost and consturct the MSP, if it does not form cycle

S4 : Repeat these steps ~~and~~ until all nodes are explored.

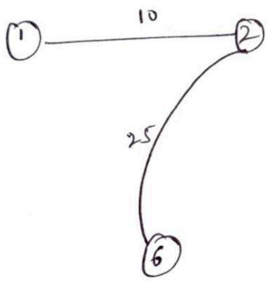
S1 :- arbitaary vertex, v = 1

adjacent vertices, $\boxed{1 \to 2 = 10}$, $1 \to 4 = 30$



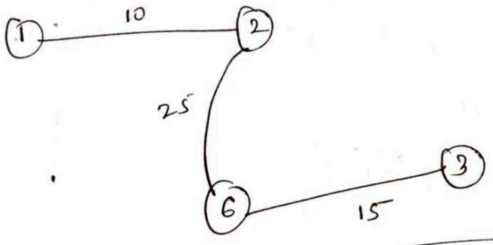
$v = 2$, $1 \to 4 = 30$

$2 \to 3 = 50$ $2 \to 5 = 40$ $\boxed{2 \to 6 = 25}$



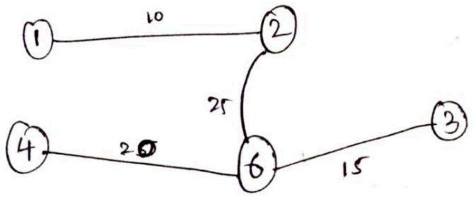
$1 \to 4 = 30$, $2 \to 3 = 50$, $2 \to 5 = 40$

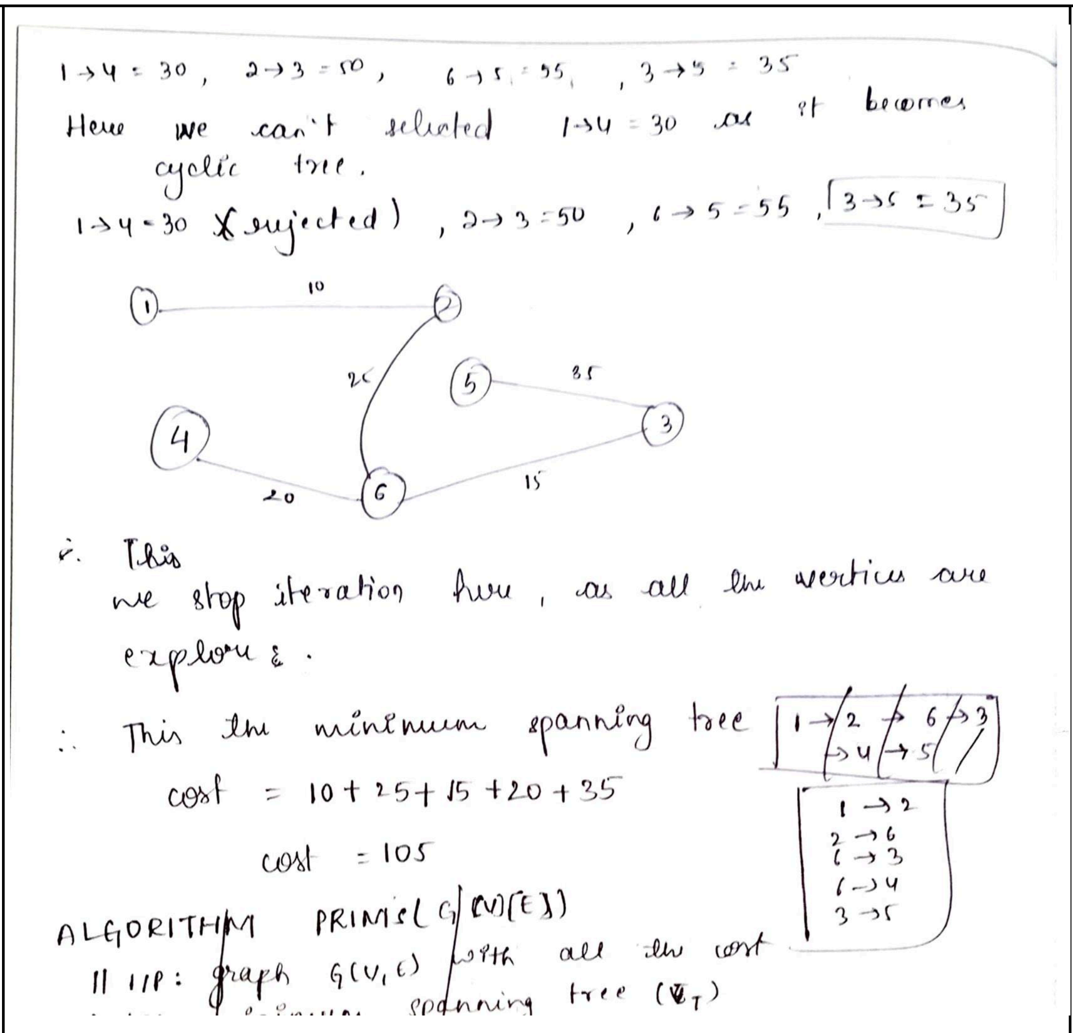
$v = 6$, $6 \to 4 = 20$, $6 \to 5 = 55$, $\boxed{6 \to 3 = 15}$



$1 \to 4 = 30$ $2 \to 3 = 50$, $2 \to 5 = 40$, $\boxed{6 \to 4 = 20}$, $6 \to 5 = 55$

$v = 3$, $3 \to 5 = \cancel{40}\,35$





1→4 = 30 , 2→3 = 50 , 6→5 = 55 , 3→5 = 35

Here we can't selected 1→4 = 30 as it becomes cyclic tree.

1→4 = 30 ✗(rejected) , 2→3 = 50 , 6→5 = 55 , $\boxed{3\to5 = 35}$

1
2
3
4
5
6
10
25
35
20
15

∴ This we stop iteration here , as all the vertices are explored .

∴ This the minimum spanning tree ~~1 → 2 → 6 → 3 →4 →5~~

cost = 10 + 25 + 15 + 20 + 35

cost = 105

1 → 2
2 → 6
6 → 3
6 → 4
3 → 5

ALGORITHM PRIMS(G[V][E])

// I/P : graph G(V, E) with all the cost

spanning tree (V_T)

6	Write a C program to implement Dijkstra's Algorithm to obtain single source shortest paths to all other vertices.	10	CO4	L2

Ans:

```
#include<stdio.h>
#include<stdlib.h>

void dijkstra(int n, int cost[n][n], int source, int parent[], int visited[], int dist[])
{
    int min, u, v;
    for(int i=0;i<n;i++){
        if(i!=source){
            min = 99;
            u = -1;
            for(int j=0;j<n;j++){
                if(visited[j]==0){
                    if(dist[j]<min){
                        min = dist[j];
                        u = j;
                    }
                }
            }
            if(u == -1)
                return;
            visited[u] = 1;
            for(v=0;v<n;v++){
                if(visited[v]==0){
                    if((dist[u]+cost[u][v]<dist[v]) && (cost[u][v]!=0)){
                        dist[v] = dist[u]+cost[u][v];
                        parent[v] = u;
                    }
                }
            }
        }
    }
    for(int i=0;i<n;i++){
        printf("\n%d -> %d with parent[%d] = %d\n", source, i, parent[i], dist[i]);
    }
}

void main(){
    int n;
    printf("Enter the no. of nodes: ");
    scanf("%d", &n);
    int cost[n][n];
    int dist[n];
    int parent[n];
    int visited[n];
    int source;
    printf("\nEnter the cost adjacency matrix:\n");
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%d", &cost[i][j]);
    printf("\nSelect Source man: ");
    scanf("%d", &source);
    for(int i=0;i<n;i++)
    {
        visited[i] = 0;
        parent[i] = source;
        dist[i] = cost[source][i];
    }
    visited[source] = 1;
    dijkstra(n, cost, source, parent, visited, dist);
}
```

CI **CCI** **HOD**

--All the Best--