Graph Theory

LAB FILE

MC-405

Submitted to:

Prof Sangita Kansal and Mr Ajay Yadav

Dept. of Applied Mathematics

Submitted by:

Apurv Agarwal

2K16/MC/019

INDEX

S.No.	TOPIC	DATE	TEACHER'S SIGNATURE
1.	Write a program to find the number of vertices , even vertices , odd vertices and the number of edges in a graph.		
2.	Write a program to find UNION, INTERSECTION and RING SUM of two graphs.		
3.	Write a program to find minimum spannin tree of a graph using Prim's Algorithm.		
4.	Write a program to find minimum spanning tree of a graph using Kruskal's Algorithm.		
5.	Write a program to find distance between 2 vertices in a graph using Disjkstra's Algorithm.		
6.	Write a program to find distance between every pair of vertices in a graph using Floyd Warshall's algorithm.		
7.	Write a program to find shortest path between every pair of vertices in a graph using Bellman Ford's algorithm.		
8.	Write a program to find maximum matching in a bipartite graph.		
9.	Write a program to find maximum matching for general graph.		
10.	Write a program to find maximum flow from source node to sink using Ford-Fulkerson algorithm.		

Q1. Write a program to find the number of vertices , even vertices , odd vertices and the number of edges in a graph.

```
#include<iostream>

using namespace std;

#define MAX 10

void degrees();

int G[MAX][MAX];

int n=0;

void create()

{  int i,j;

   cout<<"\nEnter no of vertices : ";

   cin>>n;

   cout<<"\nEnter the adjacency matrix of graph : ";

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

      cin>>G[i][j];}

void edges()

{  int edge=0;

   int matsum=0;

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

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

       {if (i==j)

          {matsum=matsum+(2*G[i][i]);}

         else

          {matsum=matsum+G[i][j];}}}

   edge=matsum/2;

   cout<<endl<<"No. of Edges: "<<edge<<endl;}

int main()

{  create();
```

```
   degrees();

   edges();

   return 0;}

void degrees()

{  int degree,i,j,deg[10],e=0,o=0;

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

   {  degree=0;

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

      { if(i!=j){ if(G[i][j]!=0)

         degree++; } else if(i==j){ if(G[i][j]!=0) degree=degree+2;}}

    deg[i]=degree; }

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

   { if(deg[i]%2==0)

      e++; else o++;}

   cout<<"\nNumber of even vertices: "<<e;

   cout<<"\nNumber of odd vertices: “<<o;}
```

OUTPUT:

```
Enter no of vertices : 3

Enter the adjacency matrix of graph : 0
1
0
1
0
1
0
1
0

Number of even vertices: 1
Number of odd vertices: 2
No. of Edges: 2
Program ended with exit code: 0
```

Q2.Write a program to find UNION, INTERSECTION and RING SUM of two graphs.

//UNION

```
#include<stdio.h>

#include<iostream>

#include<conio.h>

using namespace std;

int printUnion(int arr1[], int arr2[], int m, int n) {

    int i = 0, j = 0;

    while (i < m && j < n) {

        if (arr1[i] < arr2[j])

            printf(" %d ", arr1[i++]);

        else if (arr2[j] < arr1[i]) printf(" %d ", arr2[j++]);

        else {

            printf(" %d ", arr2[j++]);

            i++;

        }

    }

    while (i < m)

        printf(" %d ", arr1[i++]);

    while (j < n)

        printf(" %d ", arr2[j++]);

}

int main() {

    int V1[] = {0,1};

    int V2[] = {0,1,2};

    int m = sizeof(V1) / sizeof(V1[0]);

    int n = sizeof(V2) / sizeof(V2[0]);
```

```
int E1[m][m], E2[n][n], E3[m + n][m + n];
int i, j, k;
printf("Enter the adjacency matrix(symmetric) for graph G1:\n");
for (i = 0; i < m; i++) {
   for (j = 0; j < m; j++)
   {
      printf("E1[%d][%d]=", i, j);
      scanf("%d", & E1[i][j]);
   }
}
printf("Enter the adjacency matrix(symmetric) for graph G2:\n");
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++)
   {
      printf("E2[%d][%d]=", i, j);
      scanf("%d", & E2[i][j]);
   }
}
printf("\nSet of vertices in union of the graphs G1 and G2 is:\n");
printUnion(V1, V2, m, n);
printf("\n");
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++)
   {
      if (E1[i][j] > E2[i][j] && i < m && j < m)
         E3[i][j] = E1[i][j];
      else if (E1[i][j] < E2[i][j] && i < m && j < m) E3[i][j] = E2[i][j];
      else
```

```
                E3[i][j] = E2[i][j];
        }
    }
    printf("Adjacency matrix of union of graphs G1 and G2 is:\n\t");
    for (i = 0; i < n; i++) {
        printf("%d\t", i);
    }
    printf("\n\t");
    for (i = 0; i < n; i++) {
        printf("     ");
    }
    for (i = 0; i < n; i++)
    {
        printf("\n%dl\t", i);
        for (j = 0; j < n; j++)
        {
            printf("%d\t", E3[i][j]);
        }
    }
    getch();
}
```

Output:

```
Enter the adjacency matrix(symmetric) for graph G1:
E1[0][0]=0
E1[0][1]=1
E1[1][0]=1
E1[1][1]=1
Enter the adjacency matrix(symmetric) for graph G2:
E2[0][0]=0
E2[0][1]=0
E2[0][2]=1
E2[1][0]=0
E2[1][1]=0
E2[1][2]=1
E2[2][0]=1
E2[2][1]=1
E2[2][2]=0

Set of vertices in union of the graphs G1 and G2 is:
 0  1  2
Adjacency matrix of union of graphs G1 and G2 is:
        0       1       2
        _____________________
0|      0       1       1
1|      1       1       1
2|      1       1       0
```

//Intersection.

```
#include<stdio.h>

#include<iostream>

#include<conio.h>

using namespace std;

int printIntersection(int arr1[], int arr2[], int m, int n)

{

  int i = 0, j = 0;

  while (i < m && j < n)

  {

    if (arr1[i] < arr2[j]) i++;

    else if (arr2[j] < arr1[i]) j++;

    else /* if arr1[i] == arr2[j] */

    {

      printf(" %d ", arr2[j++]);

      i++;

    }

  }
```

```
}
int main()
{
    int V1[] = {0,1};
    int V2[] = {0,1,2};
    int m = sizeof(V1) / sizeof(V1[0]);
    int n = sizeof(V2) / sizeof(V2[0]);
    int E1[m][m], E2[n][n], E3[m + n][m + n];
    int i, j, k;
    printf("Enter the adjacency matrix(symmetric) for graph G1:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < m; j++)
        {
            printf("E1[%d][%d]=", i, j);
            scanf("%d", & E1[i][j]);
        }
    }
    printf("Enter the adjacency matrix(symmetric) for graph G2:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
        {
            printf("E2[%d][%d]=", i, j);
            scanf("%d", & E2[i][j]);
        }
    }
    printf("\nSet of vertices in intersection of the graphs G1 and G2 is:\n");
    printIntersection(V1, V2, m, n);
    printf("\n");
```

```
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < m; j++)
        {
            if (E1[i][j] > E2[i][j])
                E3[i][j] = E1[i][j];
            else
                E3[i][j] = E2[i][j];
        }
    }
    printf("Adjacency matrix of intersection of graphs G1 and G2 is:\n\t");
    for (i = 0; i < m; i++) {
        printf("%d\t", i);
    }
    printf("\n\t");
    for (i = 0; i < m; i++) {
        printf("     ");
    }
    for (i = 0; i < m; i++)
    {
        printf("\n%dl\t", i);
        for (j = 0; j < m; j++)
        {
            printf("%d\t", E3[i][j]);
        }
    }
    getch();
}
```

Output:

```
Enter the adjacency matrix(symmetric) for graph G1:
E1[0][0]=1
E1[0][1]=1
E1[1][0]=1
E1[1][1]=1
Enter the adjacency matrix(symmetric) for graph G2:
E2[0][0]=1
E2[0][1]=1
E2[0][2]=1
E2[1][0]=0
E2[1][1]=0
E2[1][2]=0
E2[2][0]=1
E2[2][1]=2
E2[2][2]=1

Set of vertices in intersection of the graphs G1 and G2 is:
 0  1
Adjacency matrix of intersection of graphs G1 and G2 is:
        0       1
        _______________
0|      1       1
1|      1       1
```

\

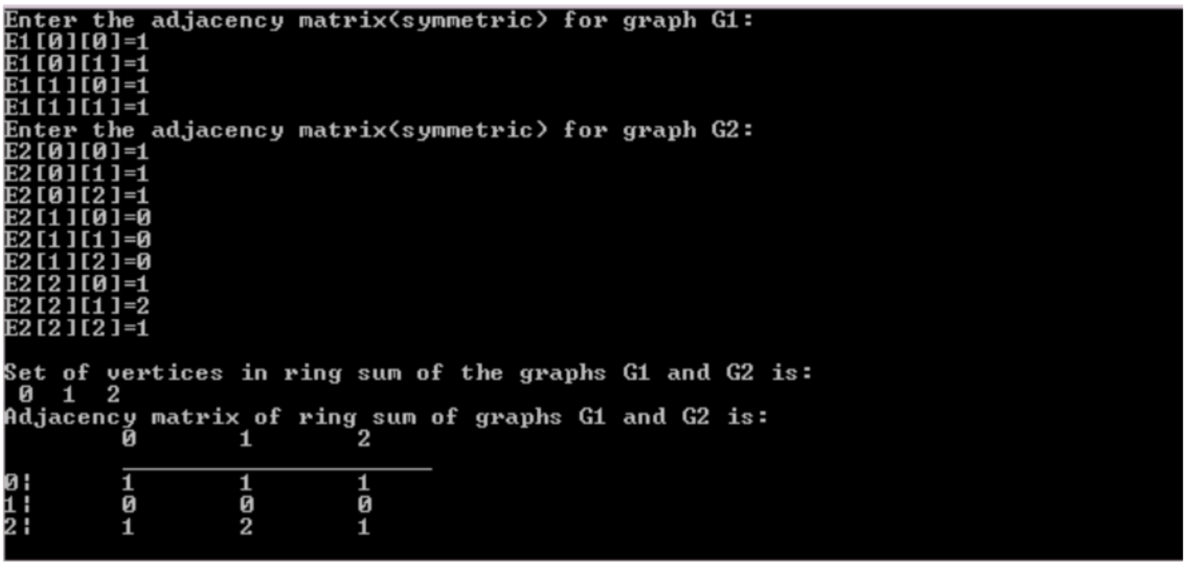
//RING SUM.

```
#include<stdio.h>
#include<iostream>
#include<conio.h> using namespace std;
int printUnion(int arr1[], int arr2[], int m, int n)
{
   int i = 0, j = 0;
   while (i < m && j < n)
   {
      if (arr1[i] < arr2[j])
         printf(" %d ", arr1[i++]);
      else if (arr2[j] < arr1[i]) printf(" %d ", arr2[j++]);
      else
      {
         printf(" %d ", arr2[j++]);
         i++;
      }
   }
   while (i < m)
      printf(" %d ", arr1[i++]);
   while (j < n)
      printf(" %d ", arr2[j++]);
```

```
}
int main()
{
    int V1[] = {0,1};
    int V2[] = {0,1,2};
    int m = sizeof(V1) / sizeof(V1[0]);
    int n = sizeof(V2) / sizeof(V2[0]);
    int E1[m][m], E2[n][n], E3[m + n][m + n];
    int i, j, k;
    printf("Enter the adjacency matrix(symmetric) for graph G1:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < m; j++)
        {
            printf("E1[%d][%d]=", i, j);
            scanf("%d", & E1[i][j]);
        }
    }
    printf("Enter the adjacency matrix(symmetric) for graph G2:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
        {
            printf("E2[%d][%d]=", i, j);
            scanf("%d", & E2[i][j]);
        }
    }
    printf("\nSet of vertices in ring sum of the graphs G1 and G2 is:\n");
    printUnion(V1, V2, m, n);
    printf("\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
        {
            if (E1[i][j] == E2[i][j] && i < m && j < m) E3[i][j] = 0;
            else if (E1[i][j] < E2[i][j] && i < m && j < m) E3[i][j] = E2[i][j];
            if (E1[i][j] < E2[i][j] && i < m && j < m)
                E3[i][j] = E1[i][j];
```

```
            else
                E3[i][j] = E2[i][j];
        }
    }
    printf("Adjacency matrix of ring sum of graphs G1 and G2 is:\n\t");
    for (i = 0; i < n; i++) {
        printf("%d\t", i);
    }
    printf("\n\t");
    for (i = 0; i < n; i++) {
        printf("        ");
    }
    for (i = 0; i < n; i++) {
        printf("\n%dl\t", i);
        for (j = 0; j < n; j++)
        {
            printf("%d\t", E3[i][j]);
        }
    }
    getch();
}
```

Output:



Q3. Write a program to find minimum spanning tree of a graph using Prim's Algorithm.

```
#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];

int main() {

    printf("\n Enter the number of nodes:");

    scanf("%d", & n);

    printf("\n Enter the weighted 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;

    printf("\n");

    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("\n Edge %d:(%d %d) cost:%d", ne++, a, b, min);

        mincost += min;

        visited[b] = 1;

      }

      cost[a]

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

    }

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

    getch();

}
```

Output:

```
 Enter the number of nodes:3

 Enter the weighted matrix:
1
2
3
3
2
1
4
5
6


 Edge 1:(1 2) cost:2
 Edge 2:(2 3) cost:1
 Minimun cost=3
```

Q4. Write a program to find minimum spanning tree of a graph using Kruskal’s Algorithm.

```
//krushkal.

#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);

int main() {

   printf("\n\n\tImplementation of Kruskal's algorithm\n\n");

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

   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\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("\n%d edge (%d,%d) =%d\n", ne++, a, b, min);
        mincost += min;
    }
    cost[a][b] = cost[b][a] = 999;
}
printf("\n\tMinimum 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;
    }
    getch();
}
```

Output:

```
         Implementation of Kruskal's algorithm


Enter the no. of vertices
3

Enter the cost adjacency matrix
0
1
1
1
0
1
1
1
0

The edges of Minimum Cost Spanning Tree are


1 edge (1,2) =1

2 edge (1,3) =1

         Minimum cost = 2
```

Q5. Write a program to find shortest path between 2 vertices in a graph using Disjkstra's Algorithm.

```
#include "stdio.h"

#define infinity 999

void dij(int n,int v,int cost[10][10],int dist[])

{ int i,u,count,w,flag[10],min; for(i=1;i<=n;i++)

   {  flag[i]=0;

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

      count=2;}

  while(count<=n)

   {  min=99;

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

      {  if(dist[w]<min && !flag[w])

         {  min=dist[w];

            u=w;}}

       flag[u]=1;

       count++;

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

          if((dist[u]+cost[u][w]<dist[w]) && !flag[w])

          {dist[w]=dist[u]+cost[u][w];}}}

int main()

{  int n,v,i,j,cost[10][10],dist[10];

   printf("\n Enter the number of nodes:");

   scanf("%d",&n);

   printf("\n Enter the cost 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]=infinity;}}
    printf("\n Enter the source :");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n Shortest path:\n");
    for(i=1;i<=n;i++)
    {  if(i!=v)
       printf("%d->%d,cost=%d\n",v,i,dist[i]);}}
```

OUTPUT:

```
 Enter the number of nodes:3

 Enter the cost matrix:
0
1
0
1
0
1
0
1
0

 Enter the source :1

 Shortest path:
1->2,cost=1
1->3,cost=2
Program ended with exit code: 0
```

Q6.Write a program to find shortest path between every pair of vertices in a graph using Floyd warshall's algorithm.

```
#include<iostream>

#include<conio.h>

 using namespace std;

void floyds(int b[][7], int n) {

    int i, j, k;

    for (k = 0; k < n; k++) {

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

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

                if ((b[i][k] * b[k][j] != 0) && (i != j)) {

                    if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0)) {

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

                    }

                }

            }

        }

    }

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

        cout << "\nMinimum Cost With Respect to Node:" << i << endl;

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

            cout << b[i][j] << "\t";

        }

    }

}

int main() {

    int b[7][7], n;

    cout << "\n Enter the number of nodes:";
```

```
    cin >> n;

    cout << "ENTER VALUES OF ADJACENCY MATRIX\n\n";

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

        cout << "enter values for " << (i + 1) << " row" << endl;

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

            cin >> b[i][j];

        }

    }

    floyds(b, n);

    getch();

}
```

Output:

```
 Enter the number of nodes:3
ENTER VALUES OF ADJACENCY MATRIX

enter values for 1 row
1
1
1
enter values for 2 row
0
1
1
enter values for 3 row
1
2
3

Minimum Cost With Respect to Node:0
1       1       1
Minimum Cost With Respect to Node:1
2       1       1
Minimum Cost With Respect to Node:2
1       2       3
Process returned 0 (0x0)   execution time : 47.650 s
Press any key to continue.
```

Q7.Write a program to find shortest path between every pair of vertices in a graph using Bellman Ford's algorithm.

```
//bellmanford.
#include<iostream>
#include<stdio.h>
#include<conio.h>
#define INFINITY 999
using namespace std;
struct node
{
    int cost;
    int value;
    int from;
}a[5];
void addEdge(int am[][5], int src, int dest, int cost) {
    am[src][dest] = cost;
    return;}
void bell(int am[][5]) {
    int i, j, k, c = 0, temp;
    a[0].cost = 0;
    a[0].from = 0;
    a[0].value = 0;
    for (i = 1; i < 5; i++) {
        a[i].from = 0;
        a[i].cost = INFINITY;
        a[i].value = 0;}
    while (c < 5) {
```

```
    int min = 999;
    for (i = 0; i < 5; i++) {
        if (min > a[i].cost && a[i].value == 0) {
            min = a[i].cost;
        } else {
            continue;
        }
    }
    for (i = 0; i < 5; i++) {
        if (min == a[i].cost && a[i].value == 0) {
            break;
        } else {
            continue;
        }
    }
    temp = i;
    for (k = 0; k < 5; k++) {
        if (am[temp][k] + a[temp].cost < a[k].cost) {
            a[k].cost = am[temp][k] + a[temp].cost;
            a[k].from = temp;
        } else {
            continue;
        }
    }
    a[temp].value = 1;
    c++;
}
cout << "Cost" << "\t" << "Source Node" << endl;
```

```
    for (j = 0; j < 5; j++) {
        cout << a[j].cost << "\t" << a[j].from << endl;
    }
}
int main() {
    int n, am[5][5], c = 0, i, j, cost;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            am[i][j] = INFINITY;
        }
    }
    while (c < 8) {
        cout << "Enter the source, destination and cost of edge\n";
        cin >> i >> j >> cost;
        addEdge(am, i, j, cost);
        c++;
    }
    bell(am);
    getch();
}
```

Output:

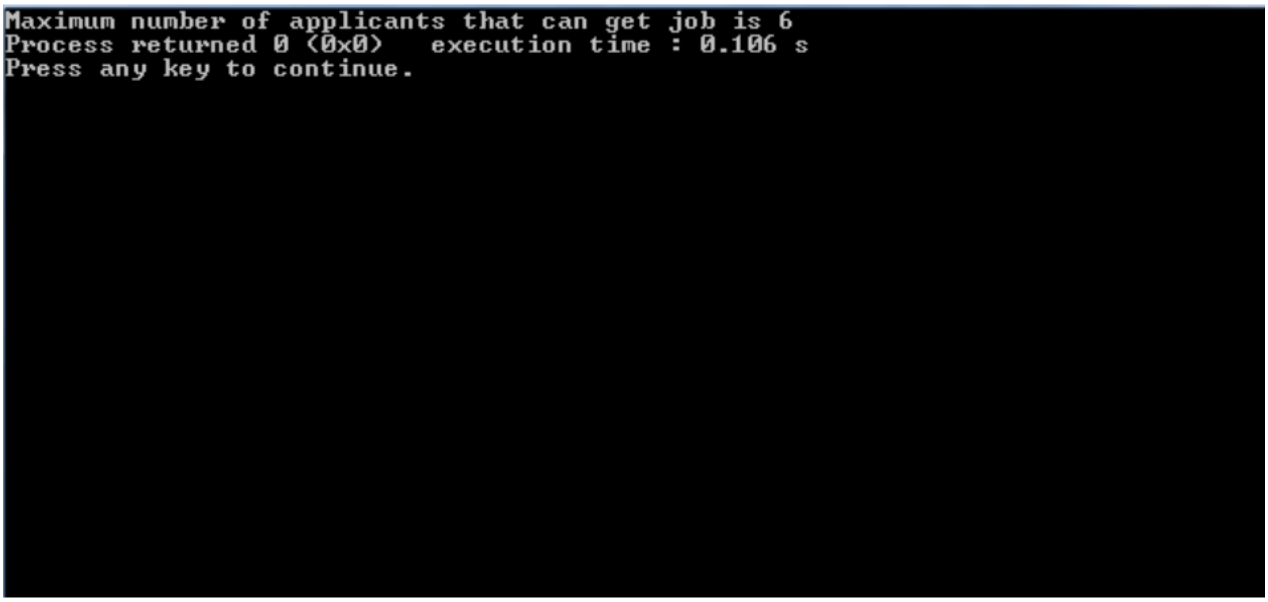
```
2
3
8
Enter the source, destination and cost of edge
2
4
9
Enter the source, destination and cost of edge
4
0
4
Enter the source, destination and cost of edge
4
3
2
Cost    Source Node
0       0
5       0
9       1
8       1
18      2

Process returned 0 (0x0)   execution time : 220.331 s
Press any key to continue.
```

Q8.Write a program to find maximum matching in a bipartite graph.

```
#include <iostream>

#include <string.h>

#include<conio.h>

#include<stdio.h>

using namespace std;

#define M 6

#define N 6

bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[]) {

   for (int v = 0; v < N; v++)

   {

      if (bpGraph[u][v] && !seen[v]) {

         seen[v] = true;

      if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen, matchR))

         {

            matchR[v] = u;

            return true;

         }

      }

   }

   return false;

}

int maxBPM(bool bpGraph[M][N])

{

   int matchR[N];

   memset(matchR, -1, sizeof(matchR));

   int result = 0;

   for (int u = 0; u < M; u++)
```

```
    {
       bool seen[N];
        memset(seen, 0, sizeof(seen));
        if (bpm(bpGraph, u, seen, matchR))
        result++;
    }
    return result;
}
int main()
{
    bool bpGraph[M][N] = { {0, 1, 1, 0, 0},{1,0,0,1,0}, {1,0,1,0,1}, {1,0,1,1,0}, {0,1,0,1,},};
cout << "Maximum number of applicants that can get job is " << maxBPM(bpGraph);
getch();
}
```

Output:



Q9.Write a program to find maximum matching for general graph.

```
// C++ implementation of Hopcroft Karp algorithm for
// maximum matching
#include<bits/stdc++.h>
using namespace std;
#define NIL 0
#define INF INT_MAX
// A class to represent Bipartite graph for Hopcroft
// Karp implementation
class BipGraph
{
	// m and n are number of vertices on left
	// and right sides of Bipartite Graph
	int m, n;
	// adj[u] stores adjacents of left side
	// vertex 'u'. The value of u ranges from 1 to m.
	// 0 is used for dummy vertex
	list<int> *adj;
	// These are basically pointers to arrays needed
	// for hopcroftKarp()
	int *pairU, *pairV, *dist;
	public:
	BipGraph(int m, int n); // Constructor
	void addEdge(int u, int v); // To add edge
	// Returns true if there is an augmenting path
	bool bfs();
	// Adds augmenting path if there is one beginning
	// with u
```

```
    bool dfs(int u);
    // Returns size of maximum matcing
    int hopcroftKarp();
};
// Returns size of maximum matching
int BipGraph::hopcroftKarp()
{
    // pairU[u] stores pair of u in matching where u
    // is a vertex on left side of Bipartite Graph.
    // If u doesn't have any pair, then pairU[u] is NIL
    pairU = new int[m+1];
    // pairV[v] stores pair of v in matching. If v
    // doesn't have any pair, then pairU[v] is NIL
    pairV = new int[n+1];
    // dist[u] stores distance of left side vertices
    // dist[u] is one more than dist[u'] if u is next
    // to u'in augmenting path
    dist = new int[m+1];
    // Initialize NIL as pair of all vertices
    for (int u=0; u<m; u++)
        pairU[u] = NIL;
    for (int v=0; v<n; v++)
        pairV[v] = NIL;
    // Initialize result
    int result = 0;
    // Keep updating the result while there is an
    // augmenting path.
    while (bfs())
```

```
	{
		// Find a free vertex
		for (int u=1; u<=m; u++)
			// If current vertex is free and there is
			// an augmenting path from current vertex
			if (pairU[u]==NIL && dfs(u))
				result++;
	}
	return result;
}
// Returns true if there is an augmenting path, else returns
// false
bool BipGraph::bfs()
{
	queue<int> Q; //an integer queue
	// First layer of vertices (set distance as 0)
	for (int u=1; u<=m; u++)
	{
		// If this is a free vertex, add it to queue
		if (pairU[u]==NIL)
		{
			// u is not matched
			dist[u] = 0;
			Q.push(u);
		}

		// Else set distance as infinite so that this vertex
		// is considered next time
```

```
        else dist[u] = INF;
}
// Initialize distance to NIL as infinite
dist[NIL] = INF;
// Q is going to contain vertices of left side only.
while (!Q.empty())
{
        // Dequeue a vertex
        int u = Q.front();
        Q.pop();
        // If this node is not NIL and can provide a shorter path to NIL
        if (dist[u] < dist[NIL])
        {
                // Get all adjacent vertices of the dequeued vertex u
                list<int>::iterator i;
                for (i=adj[u].begin(); i!=adj[u].end(); ++i)
                {
                        int v = *i;
                        // If pair of v is not considered so far
                        // (v, pairV[V]) is not yet explored edge.
                        if (dist[pairV[v]] == INF)
                        {
                                // Consider the pair and add it to queue
                                dist[pairV[v]] = dist[u] + 1;
                                Q.push(pairV[v]);
                        }
                }
        }
```

```
	}
	// If we could come back to NIL using alternating path of distinct
	// vertices then there is an augmenting path
	return (dist[NIL] != INF);
}
// Returns true if there is an augmenting path beginning with free vertex u
bool BipGraph::dfs(int u)
{
	if (u != NIL)
	{
		list<int>::iterator i;
		for (i=adj[u].begin(); i!=adj[u].end(); ++i)
		{
			// Adjacent to u
			int v = *i;

			// Follow the distances set by BFS
			if (dist[pairV[v]] == dist[u]+1)
			{
				// If dfs for pair of v also returns
				// true
				if (dfs(pairV[v]) == true)
				{
					pairV[v] = u;
					pairU[u] = v;
					return true;
				}
			}
```

```
        }
        // If there is no augmenting path beginning with u.
        dist[u] = INF;
        return false;
    }
    return true;
}
// Constructor
BipGraph::BipGraph(int m, int n)
{
    this->m = m;
    this->n = n;
    adj = new list<int>[m+1];
}
// To add edge from u to v and v to u
void BipGraph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add u to v's list.
}
// Driver Program
int main()
{
    BipGraph g(4, 4);
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 1);
    g.addEdge(3, 2);
    g.addEdge(4, 2);
```

```
    g.addEdge(4, 4);

    cout << "Size of maximum matching is " << g.hopcroftKarp();

    return 0;

}
```

Output:

```
Size of maximum matching is 4
Program ended with exit code: 0
```

}

Q10.Write a program to find maximum flow from source node to sink using Ford-Fulkerson algorithm.

```
#include <iostream>

#include <string.h>

#include <queue>

using namespace std;

boolbfs(intrGraph[][6], int s, int t, int parent[]) {

   bool visited[6];

   memset(visited, 0, sizeof(visited));

   queue < int > q;

   q.push(s);

   visited[s] = true;

   parent[s] = -1;

   while (!q.empty()) {

      int u = q.front();

      q.pop();

      for (int v = 0; v < 6; v++) {

         if (visited[v] == false && rGraph[u][v] > 0) {

            q.push(v);

            parent[v] = u;

            visited[v] = true;

         }

      }

   }

   return (visited[t] == true);

}

int fordFulkerson(int graph[6][6], int s, int t) {

   int u, v;
```

```
    int rGraph[6][6];
    for (u = 0; u < 6; u++) {
        for (v = 0; v < 6; v++) {
            rGraph[u][v] = graph[u][v];
        }
    }
    int parent[6];
    int max_flow = 0;
    while (bfs(rGraph, s, t, parent)) {
        int path_flow = INT_MAX;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;
}
int main() {
    int graph[6][6] ={ {0, 10, 7, 0, 0, 0}, {0, 0, 10, 9, 0, 0}, {0, 7, 0, 0, 14, 0}, {0, 0, 8, 0, 0, 20}, {0, 0,
0, 3, 0, 6}, {0, 0, 0, 0, 0, 0} };
    cout << "The maximum possible flow is " << fordFulkerson(graph, 0, 5);
    getch();}
```

Output:

```
The maximum possible flow is 17
Process returned 0 (0x0)   execution time : 0.231 s
Press any key to continue.
```