Graph Theory

File by:

Ashish Gupta

2k16/MC/023

Department of Applied Mathematics

Mathematics and Computing

Submitted to:

VISION AND MISSION OF THE UNIVERSITY

Vision :

" To be a world class university through education , innovation and research for the service of humanity "

Mission :

1. To establish centres of excellence in emerging areas of science, engineering, technology, management and allied areas.
2. To foster an ecosystem for incubation, product development, transfer of technology and entrepreneurship.
3. To create environment of collaboration, experimentation, imagination and creativity.
4. To develop human potential with analytical abilities, ethics and integrity.
5. To provide environment friendly, reasonable and sustainable solutions for local & global needs.

VISION AND MISSION OF THE DEPARTMENT

Vision:

"To emerge as a centre of excellence and eminence by imparting futuristic technical education with solid mathematical background in keeping with global standards, making our students technologically and mathematically competent and ethically strong so that they can readily contribute to the rapid advancement of society and mankind."

Mission:

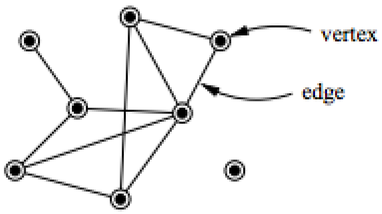
1. To achieve academic excellence through innovative teaching and learning practices.
2. To improve the research competence to address social needs.
3. To inculcate a culture that supports and reinforces ethical, professional behaviours for a harmonious and prosperous society.
4. Strive to make students to understand, appreciate and gain mathematical skills and develop logic, so that they are able to contribute intelligently in decision making which characterizes our scientific and technological age.

Content

SNo.	Practical Name / Description	Date	Remarks
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)	between 2 vertices in a graph using Disjkstra's Algorithm.		
6)	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 max flow from source node to sink using Ford- Fulkerson algorithm.		

Practical 1:

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



URL to Code: https://ide.geeksforgeeks.org/XlvtxaYHHN

Code:

```
#include<bits/stdc++.h>
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
class Graph
{
	int V;
	list < int >* adj;
public:
	Graph(int V)
	{
		this->V = V;
		adj = new list<int>[V];
	}
	void addEdge(int u, int v);
	int countEdges();
	void count_evenOdd();
};
void Graph::addEdge(int u, int v)
{
	adj[u].push_back(v);
	adj[v].push_back(u);
}
int Graph::countEdges()
{
	int sum = 0;
	for (int i = 0; i < V; i++)
		sum += adj[i].size();
	return sum / 2;
}
void Graph::count_evenOdd() {
	int even_degree = 0, odd_degree = 0;
	for (int i = 0; i < V; i++) {
		int degree = adj[i].size();
		if (degree % 2 == 0)
			even_degree++;
		else
```

```
                    odd_degree++;
        }
        cout << "No. of even vertices:" << even_degree << endl;
        cout << "No. of odd vertices:" << odd_degree << endl;
}
int main()
{
        int V, E, u, v, w;
        // cout<<"No. of Vertices:";
        cin >> V;
        // cout<<"No. of Edges:";
        cin >> E;
        Graph g(V);
        for (int i = 0; i < E; i++) {
                cin >> u;
                cin >> v;
                g.addEdge(u, v);
        }
        cout << "No.of vertices : " << V << endl;
        g.count_evenOdd();
        cout << "No.of edges : " << g.countEdges() << endl;
        return 0;
}
```

Output:

```
9
14
01
07
1 2
1 7
2 3
2 8
2 5
```

Copy

Run | Run+URL (Generates URL as well)

Generated URL:

Copy

https://ide.geeksforgeeks.org/XlvtxaYHHN

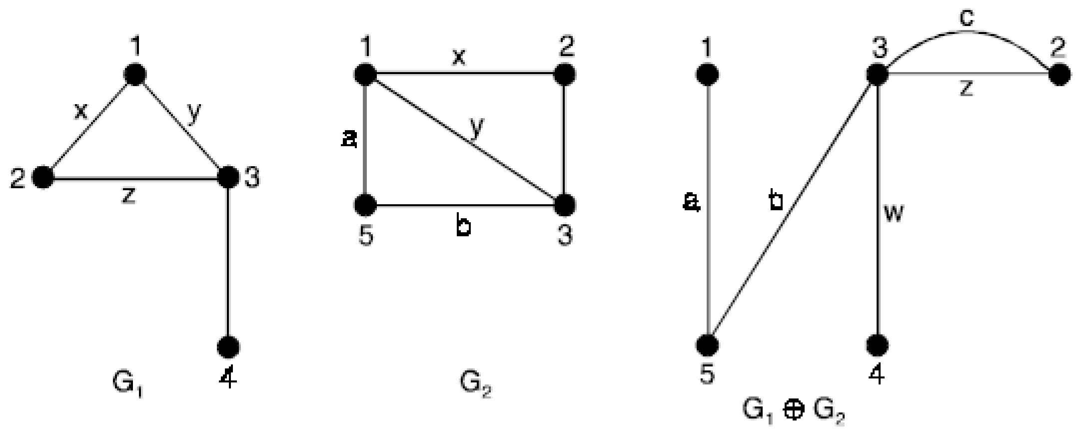
Output:

Copy

```
No.of vertices : 9
No. of even vertices:5
No. of odd vertices:4
No.of edges : 14
```

Practical 2:

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



URL to Code: https://ide.geeksforgeeks.org/L1GNIw3tuu

Code:

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

using namespace std;
int unionPrint(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 intersectionPrint(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
```

```
                {
                        printf(" %d ", arr2[j++]);
                        i++;
                }
        }
}
int main()
{
        int m, n, i, j, k;
        cout << "Enter the number of vertices in G1 & G2 : ";
        cin >> m >> n;
        int V1[m], V2[n];
        for (i = 0; i < m; i++)
        {
                V1[i] = i;
        }
        for (i = 0; i < n; i++) {
                V2[i] = i;
        }
        int E1[m][m], E2[n][n], E3[m + n][m + n];
        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");
        unionPrint(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++)
        {
                Cout<<"%d\t"<< i;
        }
        printf("\n\t");
        for (i = 0; i < n; i++)
        {
                printf("_______");
```

```c
}
for (i = 0; i < n; i++)
{
	printf("\n%d|\t", i);
	for (j = 0; j < n; j++)
	{
		printf("%d\t", E3[i][j]);
	}
}
printf("\nSet of vertices in intersection of the graphs G1 and G2 is:\n");
intersectionPrint(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%d|\t", i);
	for (j = 0; j < m; j++)
	{
		printf("%d\t", E3[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++)
        {
                Cout<<"\n%d|\t"<<i
                for (j = 0; j < n; j++)
                {
                        Cout<<"%d\t"<< E3[i][j];
                }
        }
        return 0;
}
```

Output:

```
2 3
0
1
1
1
0
0
1
0
```

Copy

Run

Run+URL (Generates URL as well)

Generated URL:

Copy

https://ide.geeksforgeeks.org/L1GNIw3tuu

Output:

Copy

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

Practical 3:

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

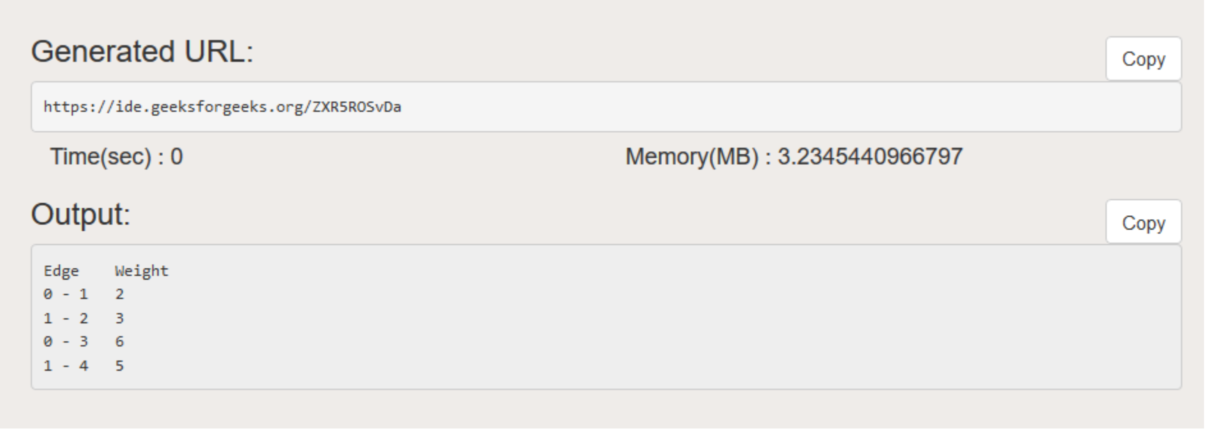
URL to Code: https://ide.geeksforgeeks.org/ZXR5ROSvDa

Code:

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

// Number of vertices in the graph
#define V 5


int minKey(int key[], bool mstSet[])
{
	// Initialize min value
	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;

	return min_index;
}


void printMST(int parent[], int graph[V][V])
{
	cout << "Edge \tWeight\n";
	for (int i = 1; i < V; i++)
		cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << " 
\n";
}


void primMST(int graph[V][V])
{
	// Array to store constructed MST
	int parent[V];

	// Key values used to pick minimum weight edge in cut
	int key[V];

	// To represent set of vertices not yet included in MST
	bool mstSet[V];

	// Initialize all keys as INFINITE
	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;

	// Always include first 1st vertex in MST.
	// Make key 0 so that this vertex is picked as first vertex.
	key[0] = 0;
```

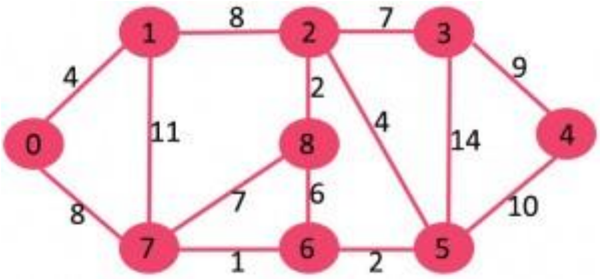
```
        parent[0] = -1; // First node is always root of MST

        // The MST will have V vertices
        for (int count = 0; count < V - 1; count++)
        {
                // Pick the minimum key vertex from the
                // set of vertices not yet included in MST
                int u = minKey(key, mstSet);

                // Add the picked vertex to the MST Set
                mstSet[u] = true;

                // Update key value and parent index of
                // the adjacent vertices of the picked vertex.
                // Consider only those vertices which are not
                // yet included in MST
                for (int v = 0; v < V; v++)

                        // graph[u][v] is non zero only for adjacent vertices of m
                        // mstSet[v] is false for vertices not yet included in MST
                        // Update the key only if graph[u][v] is smaller than key[v]
                        if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                                parent[v] = u, key[v] = graph[u][v];
        }

        // print the constructed MST
        printMST(parent, graph);
}

// Driver code
int main()
{

        int graph[V][V] = { { 0, 2, 0, 6, 0 },
                                                { 2, 0, 3, 8, 5 },
                                                { 0, 3, 0, 0, 7 },
                                                { 6, 8, 0, 0, 9 },
                                                { 0, 5, 7, 9, 0 } };

        // Print the solution
        primMST(graph);

        return 0;
}
```

Output:



Practical 4:

Aim: Write a program to find minimum spanning tree of a graph using Kruskal's Algorithm.



URL to Code: https://ide.geeksforgeeks.org/TvR0AjbQIR

Code:

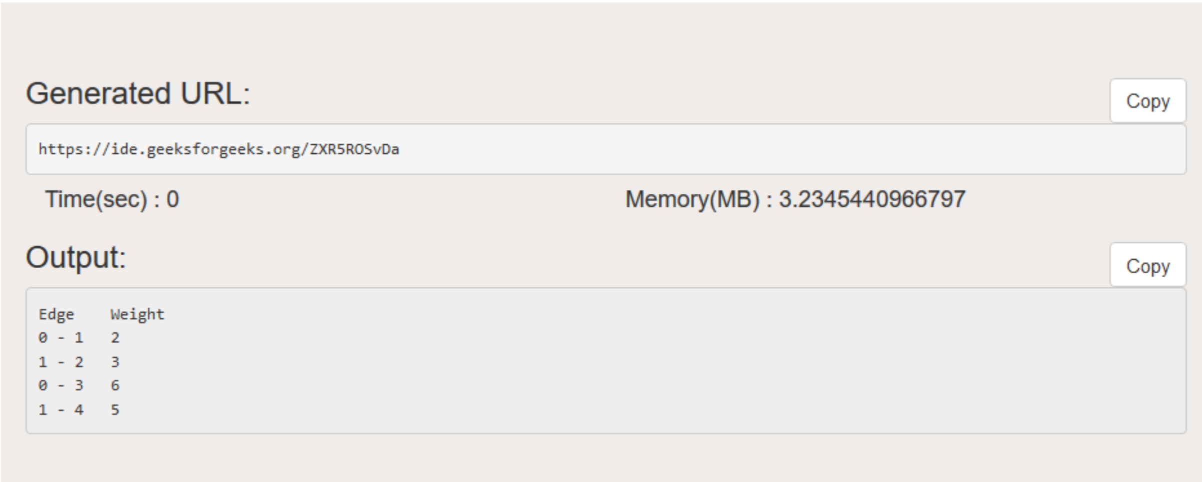
```
#include <bits/stdc++.h>
using namespace std;

// a structure to represent a weighted edge in graph
class Edge
{
public:
	int src, dest, weight;
};

// a structure to represent a connected, undirected
// and weighted graph
class Graph
{
public:
	// V-> Number of vertices, E-> Number of edges
	int V, E;


	Edge* edge;
};

// Creates a graph with V vertices and E edges
Graph* createGraph(int V, int E)
{
	Graph* graph = new Graph;
	graph->V = V;
	graph->E = E;

	graph->edge = new Edge[E];

	return graph;
}

// A structure to represent a subset for union-find
class subset
{
public:
```

```
	int parent;
	int rank;
};

// A utility function to find set of an element i
// (uses path compression technique)
int find(subset subsets[], int i)
{
	// find root and make root as parent of i
	// (path compression)
	if (subsets[i].parent != i)
		subsets[i].parent = find(subsets, subsets[i].parent);

	return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(subset subsets[], int x, int y)
{
	int xroot = find(subsets, x);
	int yroot = find(subsets, y);

	// Attach smaller rank tree under root of high
	// rank tree (Union by Rank)
	if (subsets[xroot].rank < subsets[yroot].rank)
		subsets[xroot].parent = yroot;
	else if (subsets[xroot].rank > subsets[yroot].rank)
		subsets[yroot].parent = xroot;

	// If ranks are same, then make one as root and
	// increment its rank by one
	else
	{
		subsets[yroot].parent = xroot;
		subsets[xroot].rank++;
	}
}

// Compare two edges according to their weights.
// Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
	Edge* a1 = (Edge*)a;
	Edge* b1 = (Edge*)b;
	return a1->weight > b1->weight;
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST(Graph* graph)
{
	int V = graph->V;
	Edge result[V]; // Tnis will store the resultant MST
	int e = 0; // An index variable, used for result[]
	int i = 0; // An index variable, used for sorted edges

	// Step 1: Sort all the edges in non-decreasing order

	qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

	// Allocate memory for creating V ssubsets
	subset* subsets = new subset[(V * sizeof(subset))];
```

```
	// Create V subsets with single elements
	for (int v = 0; v < V; ++v)
	{
		subsets[v].parent = v;
		subsets[v].rank = 0;
	}

	// Number of edges to be taken is equal to V-1
	while (e < V - 1 && i < graph->E)
	{
		// Step 2: Pick the smallest edge. And increment
		// the index for next iteration
		Edge next_edge = graph->edge[i++];

		int x = find(subsets, next_edge.src);
		int y = find(subsets, next_edge.dest);

		// If including this edge does't cause cycle,
		// include it in result and increment the index
		// of result for next edge
		if (x != y)
		{
			result[e++] = next_edge;
			Union(subsets, x, y);
		}
		// Else discard the next_edge
	}

	// print the contents of result[] to display the
	// built MST
	cout << "Following are the edges in the constructed MST\n";
	for (i = 0; i < e; ++i)
		cout << result[i].src << " -- " << result[i].dest << " == " <<
result[i].weight << endl;
	return;
}

// Driver code
int main()
{

	int V = 4; // Number of vertices in graph
	int E = 5; // Number of edges in graph
	Graph* graph = createGraph(V, E);


	// add edge 0-1
	graph->edge[0].src = 0;
	graph->edge[0].dest = 1;
	graph->edge[0].weight = 10;

	// add edge 0-2
	graph->edge[1].src = 0;
	graph->edge[1].dest = 2;
	graph->edge[1].weight = 6;

	// add edge 0-3
	graph->edge[2].src = 0;
	graph->edge[2].dest = 3;
	graph->edge[2].weight = 5;
```

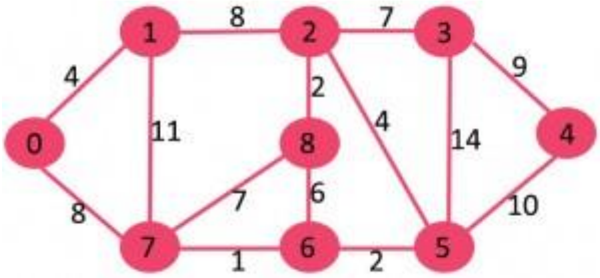
```
        // add edge 1-3
        graph->edge[3].src = 1;
        graph->edge[3].dest = 3;
        graph->edge[3].weight = 15;

        // add edge 2-3
        graph->edge[4].src = 2;
        graph->edge[4].dest = 3;
        graph->edge[4].weight = 4;

        KruskalMST(graph);

        return 0;
}
```

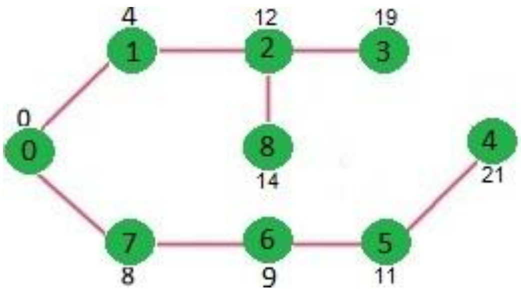
Output:



Practical 5:

Aim: Write a program to find shortest path between 2 vertices in a graph using Dijkstra's Algorithm.





URL to Code: https://ide.geeksforgeeks.org/TvR0AjbQIR

Code:

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


#define V 9


int minDistance(int dist[], bool sptSet[])
{
	// Initialize min value
	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (sptSet[v] == false && dist[v] <= min)
			min = dist[v], min_index = v;

	return min_index;
}


int printSolution(int dist[])
{
	printf("Vertex \t\t Distance from Source\n");
	for (int i = 0; i < V; i++)
		printf("%d \t\t %d\n", i, dist[i]);
```

```
}

void dijkstra(int graph[V][V], int src)
{
	int dist[V]; // The output array.  dist[i] will hold the shortest
	// distance from src to i

	bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
	// path tree or shortest distance from src to i is finalized

	// Initialize all distances as INFINITE and stpSet[] as false
	for (int i = 0; i < V; i++)
		dist[i] = INT_MAX, sptSet[i] = false;

	// Distance of source vertex from itself is always 0
	dist[src] = 0;

	// Find shortest path for all vertices
	for (int count = 0; count < V - 1; count++) {
		// Pick the minimum distance vertex from the set of vertices not
		// yet processed. u is always equal to src in the first iteration.
		int u = minDistance(dist, sptSet);

		// Mark the picked vertex as processed
		sptSet[u] = true;

		// Update dist value of the adjacent vertices of the picked vertex.
		for (int v = 0; v < V; v++)

			// Update dist[v] only if is not in sptSet, there is an edge from
			// u to v, and total weight of path from src to  v through u is
			// smaller than current value of dist[v]
			if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
				&& dist[u] + graph[u][v] < dist[v])
				dist[v] = dist[u] + graph[u][v];
	}

	// print the constructed distance array
	printSolution(dist);
}

// driver program to test above function
int main()
{

	int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
						{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },
						{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },
						{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },
						{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },
						{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },
						{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },
						{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },
						{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

	dijkstra(graph, 0);

	return 0;
}
```

Output:

Generated URL:

Copy

```
https://ide.geeksforgeeks.org/dXD1gYHTyw
```

Time(sec) : 0

Memory(MB) : 1.4835064990234

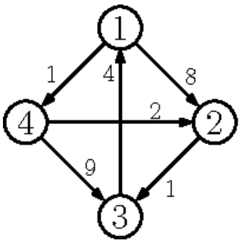
Output:

Copy

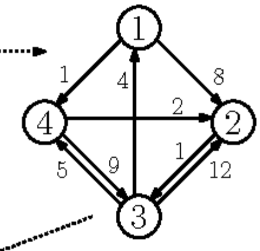
```
Vertex          Distance from Source
0               0
1               4
2               12
3               19
4               21
5               11
6               9
7               8
8               14
```

Practical 6:

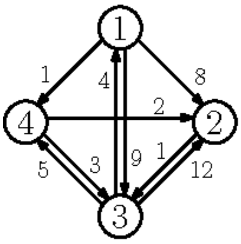
Aim: Write a program to find shortest path between every pair of vertices in a graph using Floyd Warshall's algorithm.



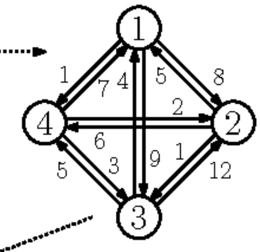
$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$



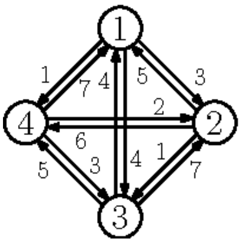
$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$



$$d^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

$$\text{final} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

URL to Code: https://ide.geeksforgeeks.org/LJBGuaqtQm

Code:

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

// Number of vertices in the graph
#define V 4

#define INF 99999


void printSolution(int dist[][V]);


void floydWarshall(int graph[][V])
{

       int dist[V][V], i, j, k;
```

```
	for (i = 0; i < V; i++)
		for (j = 0; j < V; j++)
			dist[i][j] = graph[i][j];


	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])
{
	cout << "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)
				cout << "INF" << "     ";
			else
				cout << dist[i][j] << "     ";
		}
		cout << endl;
	}
}

// Driver code
int main()
{

	int graph[V][V] = { {0, 5, INF, 10},
						{INF, 0, 3, INF},
						{INF, INF, 0, 1},
						{INF, INF, INF, 0}
	};

	// Print the solution
	floydWarshall(graph);
	return 0;
}
```

Output:

Generated URL:

Copy

```
https://ide.geeksforgeeks.org/LJBGuaqtQm
```

Time(sec) : 0   Memory(MB) : 3.1798970120239

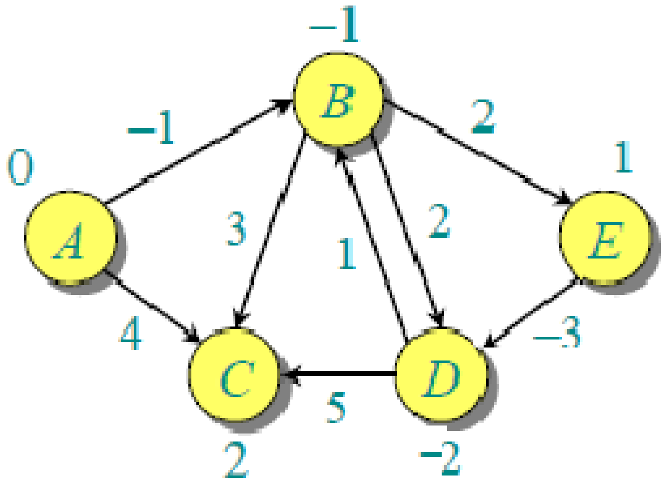
Output:

Copy

```
The following matrix shows the shortest distances between every pair of vertices
0      5      8      9
INF      0      3      4
INF      INF      0      1
INF      INF      INF      0
```

Practical 7:

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



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

URL to Code: https://ide.geeksforgeeks.org/qNjvnG5y5i

Code:

```
#include <bits/stdc++.h>

struct Edge {
	int src, dest, weight;
};

struct Graph {
	int V, E;

	struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
	struct Graph* graph = new Graph;
	graph->V = V;
	graph->E = E;
	graph->edge = new Edge[E];
	return graph;
}

// A utility function used to print the solution
void printArr(int dist[], int n)
{
```

```
	printf("Vertex   Distance from Source\n");
	for (int i = 0; i < n; ++i)
		printf("%d \t\t %d\n", i, dist[i]);
}

void BellmanFord(struct Graph* graph, int src)
{
	int V = graph->V;
	int E = graph->E;
	int dist[V];

	for (int i = 0; i < V; i++)
		dist[i] = INT_MAX;
	dist[src] = 0;

	for (int i = 1; i <= V - 1; i++) {
		for (int j = 0; j < E; j++) {
			int u = graph->edge[j].src;
			int v = graph->edge[j].dest;
			int weight = graph->edge[j].weight;
			if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
				dist[v] = dist[u] + weight;
		}
	}

	for (int i = 0; i < E; i++) {
		int u = graph->edge[i].src;
		int v = graph->edge[i].dest;
		int weight = graph->edge[i].weight;
		if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
			printf("Graph contains negative weight cycle");
			return; // If negative cycle is detected, simply return
		}
	}

	printArr(dist, V);

	return;
}

// Driver program to test above functions
int main()
{
	int V = 5; // Number of vertices in graph
	int E = 8; // Number of edges in graph
	struct Graph* graph = createGraph(V, E);

	// add edge 0-1 (or A-B in above figure)
	graph->edge[0].src = 0;
	graph->edge[0].dest = 1;
	graph->edge[0].weight = -1;

	// add edge 0-2 (or A-C in above figure)
	graph->edge[1].src = 0;
	graph->edge[1].dest = 2;
	graph->edge[1].weight = 4;

	// add edge 1-2 (or B-C in above figure)
	graph->edge[2].src = 1;
	graph->edge[2].dest = 2;
	graph->edge[2].weight = 3;
```

```
    // add edge 1-3 (or B-D in above figure)
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    // add edge 1-4 (or A-E in above figure)
    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)
    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)
    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;

    // add edge 4-3 (or E-D in above figure)
    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);

    return 0;
}
```

Output:

Generated URL: Copy

https://ide.geeksforgeeks.org/qNjvnG5y5i

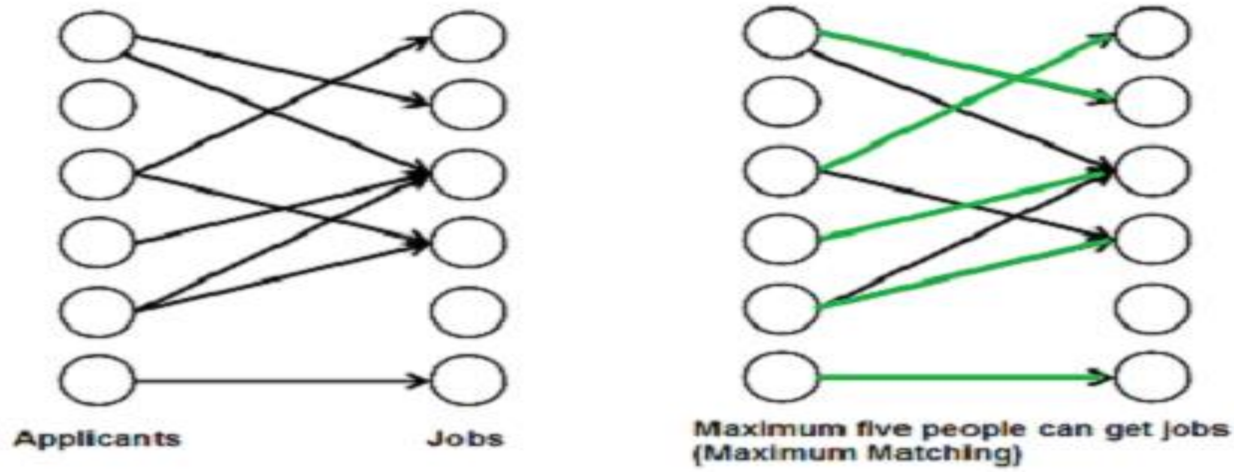
Time(sec) : 0 Memory(MB) : 3.4341827597046

Output: Copy

```
Vertex   Distance from Source
0            0
1            -1
2            2
3            -2
4            1
```

Practical 8:

Aim: Write a program to find maximum matching in a bipartite graph.



URL to Code: https://ide.geeksforgeeks.org/JHnbrDtIFH

Code:

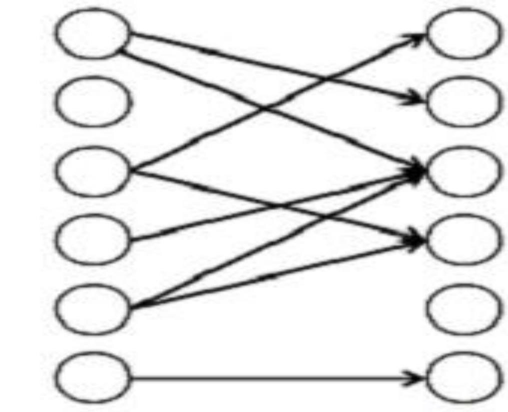
```
#include <iostream>
#include <string.h>
using namespace std;

#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])
                {
                        // Mark v as visited
                        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])
```

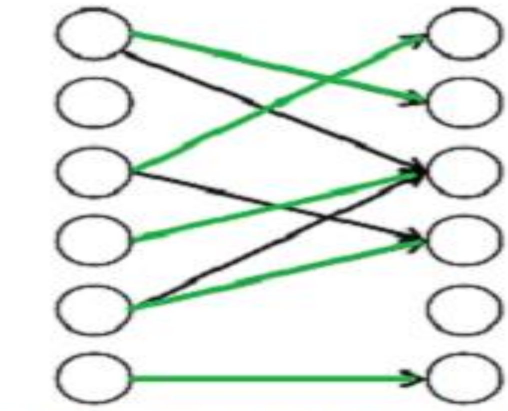
```
{
        int matchR[N];

        // Initially all jobs are available
        memset(matchR, -1, sizeof(matchR));

        // Count of jobs assigned to applicants
        int result = 0;
        for (int u = 0; u < M; u++)
        {
                bool seen[N];
                memset(seen, 0, sizeof(seen));

                // Find if the applicant 'u' can get a job
                if (bpm(bpGraph, u, seen, matchR))
                        result++;
        }
        return result;
}

int main()
{
        bool bpGraph[M][N] = { {0, 1, 1, 0, 0, 0},
                                                {1, 0, 0, 1, 0, 0},
                                                {0, 0, 1, 0, 0, 0},
                                                {0, 0, 1, 1, 0, 0},
                                                {0, 0, 0, 0, 0, 0},
                                                {0, 0, 0, 0, 0, 1} };

        cout << "Maximum number of applicants that can get job is "
                << maxBPM(bpGraph);

        return 0;
}
```

Output:

Practical 9:

Aim: Write a program to find maximum matching for a general graph.





URL to Code: https://ide.geeksforgeeks.org/ZoQagFGJIk

Code:

```
#include <bits/stdc++.h>
using namespace std;
const int M = 500;
struct struct_edge { int v; struct_edge* n; };
typedef struct_edge* edge;
struct_edge pool[M * M * 2];
edge top = pool, adj[M];
int V, E, match[M], qh, qt, q[M], father[M], base[M];
bool inq[M], inb[M], ed[M][M];
void add_edge(int u, int v)
{
	top->v = v, top->n = adj[u], adj[u] = top++;
	top->v = u, top->n = adj[v], adj[v] = top++;
}
int LCA(int root, int u, int v)
{
	static bool inp[M];
	memset(inp, 0, sizeof(inp));
	while (1)
	{
		inp[u = base[u]] = true;
		if (u == root) break;
		u = father[match[u]];
	}
	while (1)
	{
		if (inp[v = base[v]]) return v;
		else v = father[match[v]];
	}
}
```

```
void mark_blossom(int lca, int u)
{
	while (base[u] != lca)
	{
		int v = match[u];
		inb[base[u]] = inb[base[v]] = true;
		u = father[v];
		if (base[u] != lca) father[u] = v;
	}
}
void blossom_contraction(int s, int u, int v)
{
	int lca = LCA(s, u, v);
	memset(inb, 0, sizeof(inb));
	mark_blossom(lca, u);
	mark_blossom(lca, v);
	if (base[u] != lca)
		father[u] = v;
	if (base[v] != lca)
		father[v] = u;
	for (int u = 0; u < V; u++)
		if (inb[base[u]])
		{
			base[u] = lca;
			if (!inq[u])
				inq[q[++qt] = u] = true;
		}
}
int find_augmenting_path(int s)
{
	memset(inq, 0, sizeof(inq));
	memset(father, -1, sizeof(father));
	for (int i = 0; i < V; i++) base[i] = i;
	inq[q[qh = qt = 0] = s] = true;
	while (qh <= qt)
	{
		int u = q[qh++];
		for (edge e = adj[u]; e; e = e->n)
		{
			int v = e->v;
			if (base[u] != base[v] && match[u] != v)
				if ((v == s) || (match[v] != -1 && father[match[v]] != -
1))
					blossom_contraction(s, u, v);
				else if (father[v] == -1)
				{
					father[v] = u;
					if (match[v] == -1)
						return v;
					else if (!inq[match[v]])
						inq[q[++qt] = match[v]] = true;
				}
		}
	}
	return -1;
}
int augment_path(int s, int t)
{
	int u = t, v, w;
	while (u != -1)
	{
		v = father[u];
```

```
            w = match[v];
            match[v] = u;
            match[u] = v;
            u = w;
        }
        return t != -1;
}
int edmonds()
{
        int matchc = 0;
        memset(match, -1, sizeof(match));
        for (int u = 0; u < V; u++)
            if (match[u] == -1)
                matchc += augment_path(u, find_augmenting_path(u));
        return matchc;
}
int main()
{
        int u, v;
        cout << "Enter the number of vertices and edges : ";
        cin >> V >> E;
        cout << "Enter the edges : \n";
        while (E--)
        {
            cin >> u >> v;
            if (!ed[u - 1][v - 1])
            {
                add_edge(u - 1, v - 1);
                ed[u - 1][v - 1] = ed[v - 1][u - 1] = true;
            }
        }
        cout << "Number of matches : " << edmonds() << endl;
        cout << "The matches are : \n";
        for (int i = 0; i < V; i++)
            if (i < match[i])
                cout << i + 1 << " " << match[i] + 1 << endl;
}
```

Output:

- -
0 1
0 2
1 2
1 3
1 4
3 2
3 1
4 3
2

Copy

> Run

> Run+URL (Generates URL as well)

Generated URL:

Copy

https://ide.geeksforgeeks.org/ZoQagFGJIk

Time(sec) : 0

Memory(MB) : 11.211074171753

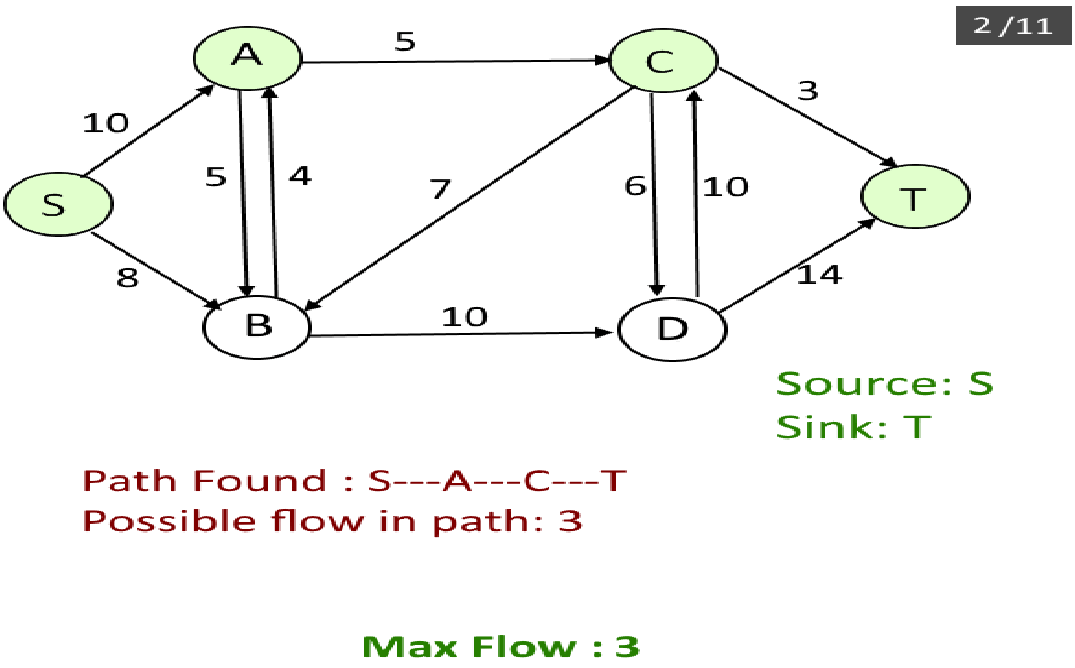
Output:

Copy

```
Enter the number of vertices and edges : Enter the edges :
Number of matches : 2
The matches are :
1 4
2 3
```

Practical 10:

Aim: Write a program to find maximum matching for a general graph.



URL to Code: https://ide.geeksforgeeks.org/ZoQagFGJIk

Code:

```
#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
using namespace std;

#define V 6

bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
	bool visited[V];
	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 < V; 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[V][V], int s, int t)
{
    int u, v;

    int rGraph[V][V]; // Residual graph where rGraph[i][j] indicates
                                    // residual capacity of edge from i to j (if there
                                    // is an edge. If rGraph[i][j] is 0, then there is
not)
    for (u = 0; u < V; u++)
            for (v = 0; v < V; v++)
                    rGraph[u][v] = graph[u][v];

    int parent[V];  // This array is filled by BFS and to store path

    int max_flow = 0;  // There is no flow initially

    // Augment the flow while tere is path from source to sink
    while (bfs(rGraph, s, t, parent))
    {
            // Find minimum residual capacity of the edges along the
            // path filled by BFS. Or we can say find the maximum flow
            // through the path found.
            int path_flow = INT_MAX;
            for (v = t; v != s; v = parent[v])
            {
                    u = parent[v];
                    path_flow = min(path_flow, rGraph[u][v]);
            }

            // update residual capacities of the edges and reverse edges
            // along the path
            for (v = t; v != s; v = parent[v])
            {
                    u = parent[v];
                    rGraph[u][v] -= path_flow;
                    rGraph[v][u] += path_flow;
            }

            // Add path flow to overall flow
            max_flow += path_flow;
    }

    // Return the overall flow
    return max_flow;
}

// Driver program to test above functions
int main()
{
```

```
        // Let us create a graph shown in the above example
        int graph[V][V] = { {0, 16, 13, 0, 0, 0},
                                        {0, 0, 10, 12, 0, 0},
                                        {0, 4, 0, 0, 14, 0},
                                        {0, 0, 9, 0, 0, 20},
                                        {0, 0, 0, 7, 0, 4},
                                        {0, 0, 0, 0, 0, 0}
        };

        cout << "The maximum possible flow is " << fordFulkerson(graph, 0, 5);

        return 0;
}
```

Output: