[Graph]

1. DFS, BFS traversing

Write a program to build an **undirected** graph by giving edge list.

Giving a start vertex, your task is showing the **DFS** (Depth First Search) and the **BFS** (Breadth First Traversal) traversing of the **undirected** graph.

For example,

● Giving the edge list that used to represent undirected graph as follow:
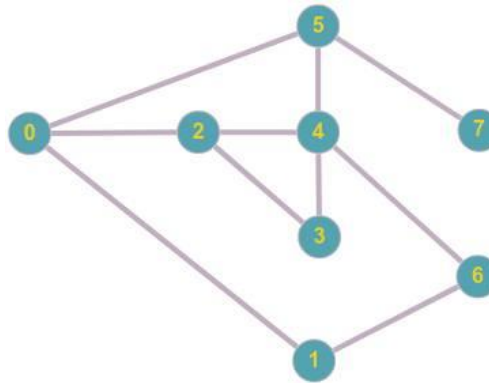
```
8 10 0
0 1
0 2
0 5
1 6
2 3
2 4
3 4
4 5
4 6
5 7
```


Figure 1. The undirected graph that created by giving adjacency matrix

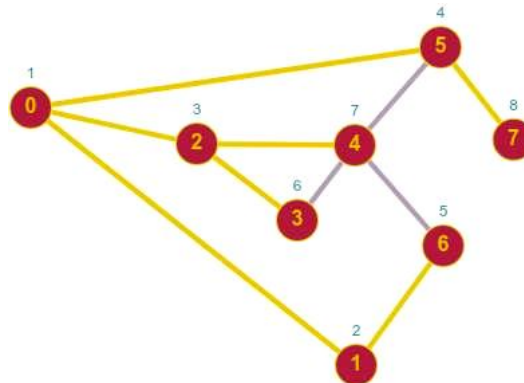● The BFS traversing of the graph is: 0, 1, 2, 5, 6, 3, 4, 7

Figure 2. The BFS traversing of the graph is: 0, 1, 2, 5, 6, 3, 4, 7

The input: are stored in the *bfs_input.txt* text file:
● The first line contains a 3 integers including:

- N ($1 \leq N \leq 30$) is the number of vertex.

- M ($1 \leq M \leq 435$) is the number of edge.

- S ($0 \leq S \leq N$-1) is the starting vertex.

● The next M line, each line contains 2 integers u and v that represent the edge (u, v) of the graph.

The output: the results need to be saved to the *bfs_output.txt* text file:
The first line contains the list of numbers representing the DFS traversing of the graph. Each number separated by one comma.

[Graph]

The second line contains the list of numbers representing the BFS traversing of the graph. Each number separated by one comma.

Sample Input 1	Sample Output 1
8 10 0	0,1,6,4,2,3,5,7
0 1	0,1,2,5,6,3,4,7
0 2	
0 5	
1 6	
2 3	
2 4	
3 4	
4 5	
4 6	
5 7	

Sample Input 2	Sample Output 2
6 7 3	3,2,0,1,5,4
0 1	3,2,4,0,5,1
0 2	
0 5	
2 3	
2 4	
3 4	
4 5	

[Graph]

2. Minimum Span Tree (MST)

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is finding and showing all edge of the MST of the given undirected graph using **Prim** and **Kruskal** algorithms.

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
8
0 5 3 0 0 11 0 0
5 0 5 0 0 0 7 11
3 5 0 11 1 7 0 0
0 0 11 0 11 0 11 0
0 0 1 11 0 1 5 7
11 0 7 0 1 0 0 42
0 7 0 11 5 0 0 42
0 11 0 0 7 42 42 0
```
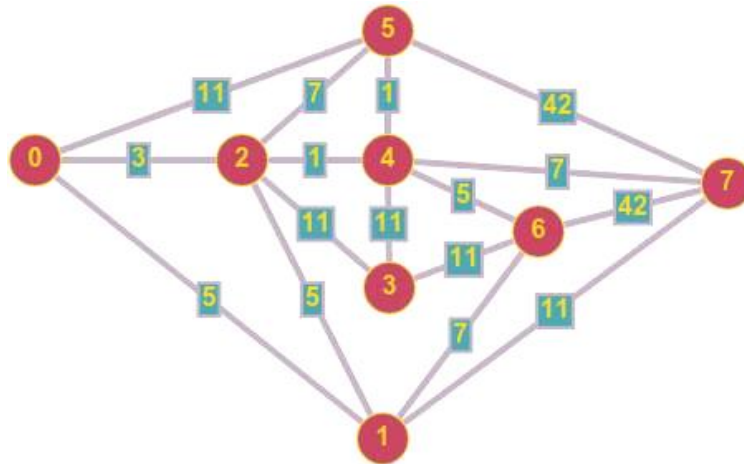

Figure 3. The undirected graph that created by giving adjacency matrix

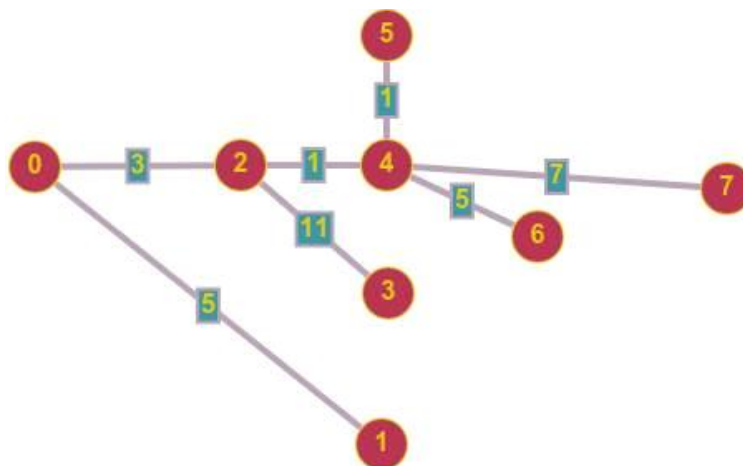● The MST is shown as figure 4 and the weight of minimum spanning tree is 33:

Figure 4. The weight of minimum spanning tree is 33

The input: are stored in the *mst_input.txt* text file:

The first line contains a positive integer N ($1 \leq N \leq 100$) which is the number of vertex of undirected graph.

The next N line, each line containing N integers that represent the adjacency matrix.

The output: the results need to be saved to the *mst_output.txt* text file:

The output contains the weight of the minimum spanning tree and the edges.

[Graph]

Sample Input 1	Sample Output 1
8	Prime:
0 5 3 0 0 11 0 0	33
5 0 5 0 0 0 7 11	0 0 0
3 5 0 11 1 7 0 0	0 1 5
0 0 11 0 11 0 11 0	0 2 3
0 0 1 11 0 1 5 7	2 3 11
11 0 7 0 1 0 0 42	2 4 1
0 7 0 11 5 0 0 42	4 5 1
0 11 0 0 7 42 42 0	4 6 5
	4 7 7
	Kruskal:
	33
	2 4 1
	4 5 1
	0 2 3
	0 1 5
	4 6 5
	4 7 7
	2 3 11

Sample Input 2	Sample Output 2
6	Prime:
0 7 5 0 0 1	18
7 0 0 0 0 0	0 0 0
5 0 0 2 3 0	0 1 7
0 0 2 0 7 0	0 2 5
0 0 3 7 0 9	2 3 2
1 0 0 0 9 0	2 4 3

[Graph]

| | ```
0 5 1
Kruskal:
18
0 5 1
2 3 2
2 4 3
0 2 5
0 1 7
``` |

[Graph]

3. Isolated vertices counting

Write a program to build an **undirected** graph by giving adjacency matrix.

An isolated vertex is a vertex with degree zero; that is, a vertex that is not an endpoint of any edge.

Your task is showing all isolated vertices of the given graph.

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
9
0 1 1 0 0 1 0 0 0
1 0 0 0 0 0 0 1 0
1 0 0 1 1 0 0 0 0
0 0 1 0 1 0 0 0 0
0 0 1 1 0 1 0 1 0
1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0
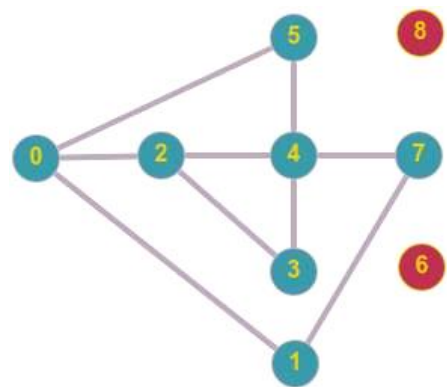0 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0
```


Figure 5. The undirected graph that created by giving adjacency matrix

● All isolated vertices of the matrix are 6 and 8.

The input: are stored in the *isolatedVertices_input.txt* text file:

● The first line contains a positive integer N ($1 \leq N \leq 100$) which is the number of vertex of undirected graph.
● The next N line, each line containing N integers that represent the adjacency matrix.

The output: the results need to be saved to the *isolatedVertices_output.txt* text file:
● One line contains the list of isolated vertices of the undirected graph, each number separated by one comma.
● If the undirected graph doesn't have any isolated vertex, the result file will contain the message "There is a connected graph".

Sample Input 1	Sample Output 1
9 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	6,8

[Graph]

Sample Input 2	Sample Output 2
6 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1 0	There is a connected graph

[Graph]

4. Eulerian cycle & path

Write a program to build an **undirected** graph by giving edge list.

Giving a start vertex, your task is finding the Eulerian cycle and path of the **undirected** graph.

For example,

- Giving the adjacency matrix that used to represent undirected graph as follow:

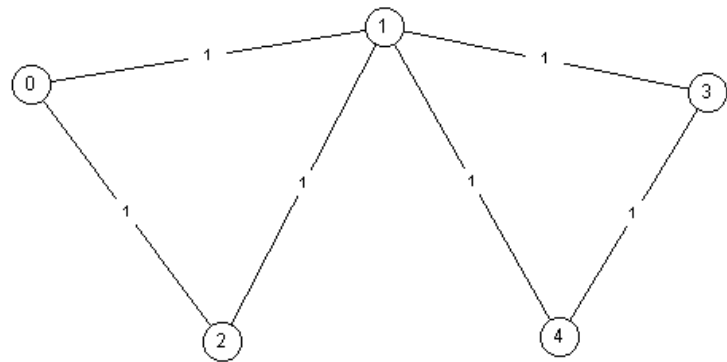```
5 0
0 1 1 0 0
1 0 1 1 1
1 1 0 0 0
0 1 0 0 1
0 1 0 1 0
```

Figure 1. The undirected graph that created by giving adjacency matrix

- Eulerian cycle: 0,2,1,4,3,1,0
- Eulerian path: 0,2,1,4,3,1

The input: are stored in the *euler_input.txt* text file:
- The first line contains a 2 integers including:

   - N ($1 \le N \le 30$) is the number of vertex.

   - S ($0 \le S \le N$-1) is the starting vertex.
- The next N line, each line containing N integers that represent the adjacency matrix.

The output: the results need to be saved to the *euler_output.txt* text file:
The first line contains the list of numbers representing eulerian cycle of the graph. Each number separated by one comma.
The second line contains the list of numbers representing eulerian path of the graph. Each number separated by one comma.

Sample Input 1	Sample Output 1
5 0 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 1	0,2,1,4,3,1,0

[Graph]

0 1 0 1 0	

Sample Input 2	*Sample Output 2*
5 0 0 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0	No eulerian cycle! 4,3,0,2,1,0

[Graph]

5. Connected component

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is to count number of connected components in an undirected graph.

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
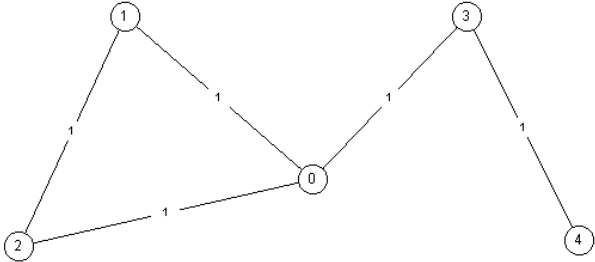10
0 1 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0
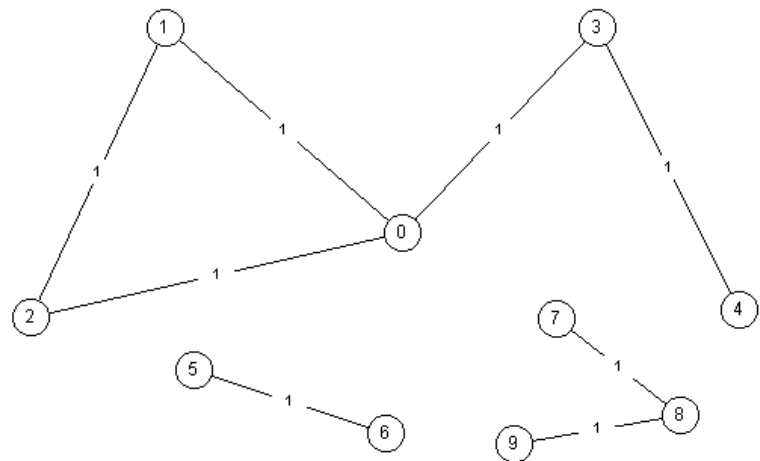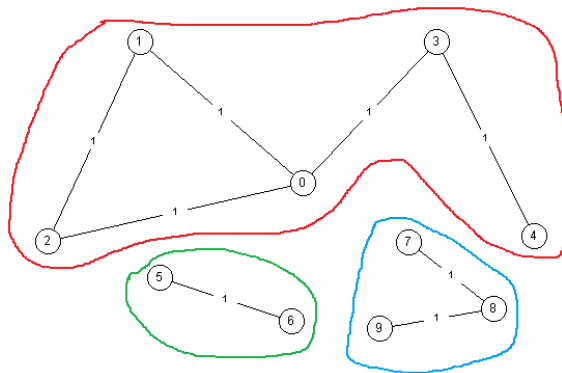0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 1 0
```


Figure 3. The undirected graph that created by giving adjacency matrix

● The number of connected components is 3:

The input: are stored in the ***cc_input.txt*** text file:

The first line contains a positive integer *N* (1 ≤ *N* ≤ 100) which is the number of vertex of undirected graph.

The next *N* line, each line containing *N* integers that represent the adjacency matrix.

The output: the results need to be saved to the ***cc_output.txt*** text file:

The first line contains number of connected components.

The next number of connected components line, each line contains the list of numbers representing the DFS traversing of each connected component. Each number separated by one comma.

[Graph]

Sample Input 1	Sample Output 1
10	3
0 1 1 1 0 0 0 0 0 0	0,1,2,3,4
1 0 1 0 0 0 0 0 0 0	5,6
1 1 0 0 0 0 0 0 0 0	7,8,9
1 0 0 0 1 0 0 0 0 0	
0 0 0 1 0 0 0 0 0 0	
0 0 0 0 0 0 1 0 0 0	
0 0 0 0 0 1 0 0 0 0	
0 0 0 0 0 0 0 0 1 0	
0 0 0 0 0 0 0 1 0 1	
0 0 0 0 0 0 0 0 1 0	

Sample Input 2	Sample Output 2
5 0	1
0 1 1 0 0	
1 0 1 1 1	0,2,1,3,4
1 1 0 0 0	
0 1 0 0 1	
0 1 0 1 0	

[Graph]

6. Shortest Path

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is to show the shortest path between a starting vertex and an ending vertex of the graph.

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
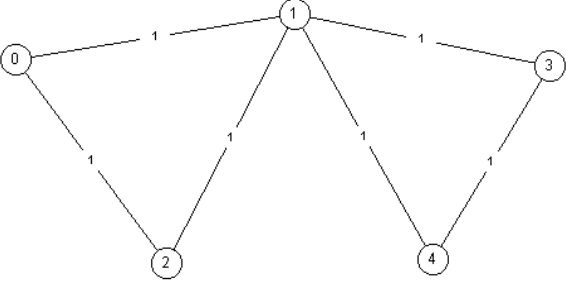8 0 7
0 5 3 0 0 11 0 0
5 0 5 0 0 0 7 11
3 5 0 11 1 7 0 0
0 0 11 0 11 0 11 0
0 0 1 11 0 1 5 7
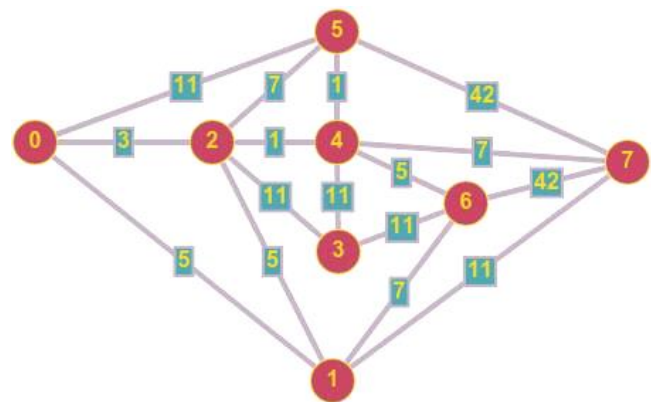11 0 7 0 1 0 0 42
0 7 0 11 5 0 0 42
0 11 0 0 7 42 42 0
```


Figure 5. The undirected graph that created by giving adjacency matrix

The input: are stored in the *shortestPath_input.txt* text file:
● The first line contains a 2 integers including:

  ▪ **N** (1 ≤ **N** ≤ 30) is the number of vertex.

  ▪ **S** (0 ≤ **S** ≤ **N**-1) is the starting vertex.

  ▪ **E** (0 ≤ **S** ≤ **N**-1) is the ending vertex.

● The next **N** line, each line containing **N** integers that represent the adjacency matrix.

The output: the results need to be saved to the *shortestPath_output.txt* text file:
● One line contains the shortest path from the starting vertex to the ending vertiex in the graph, each number separated by one comma.

Sample Input 1	Sample Output 1
8 0 7 0 5 3 0 0 11 0 0 5 0 5 0 0 0 7 11 3 5 0 11 1 7 0 0 0 0 11 0 11 0 11 0 0 0 1 11 0 1 5 7 11 0 7 0 1 0 0 42 0 7 0 11 5 0 0 42 0 11 0 0 7 42 42 0	0->2->4->7: 11

[Graph]

Sample Input 2	Sample Output 2
7 0 4 0 7 5 0 0 4 0 7 0 0 0 0 0 0 5 0 0 2 7 0 0 0 0 2 0 0 0 3 0 0 7 0 0 9 1 4 0 0 0 9 0 0 0 0 0 3 1 0 0	0->2->3->6->4: 11

[Graph]

7. Shortest Paths

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is to show the shortest path between a starting vertex, and the rest of the graph.

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
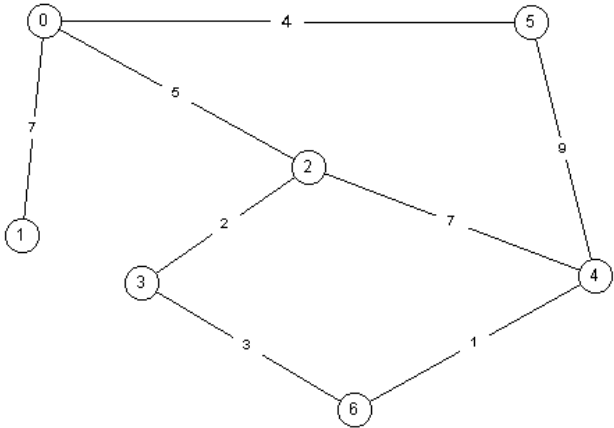8 0
0 5 3 0 0 11 0 0
5 0 5 0 0 0 7 11
3 5 0 11 1 7 0 0
0 0 11 0 11 0 11 0
0 0 1 11 0 1 5 7
11 0 7 0 1 0 0 42
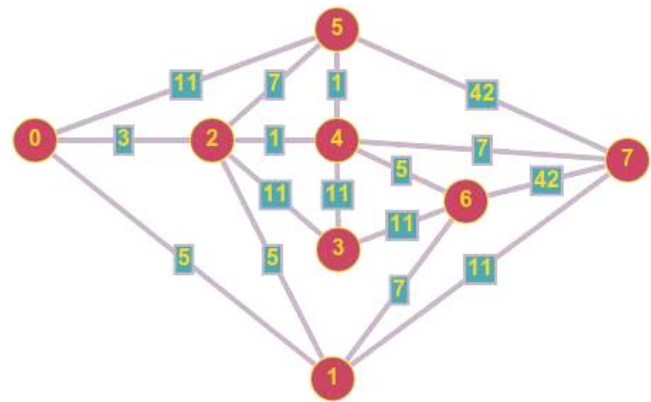0 7 0 11 5 0 0 42
0 11 0 0 7 42 42 0
```


Figure 5. The undirected graph that created by giving adjacency matrix

The input: are stored in the *shortestPath_input.txt* text file:
● The first line contains a 2 integers including:
  ▪ *N* (1 ≤ *N* ≤ 30) is the number of vertex.
  ▪ **S** (0 ≤ *S* ≤ *N*-1) is the starting vertex.
● The next *N* line, each line containing *N* integers that represent the adjacency matrix.

The output: the results need to be saved to the *shortestPath_output.txt* text file:
● Each line contains the shortest path from the starting vertex to the rest vertices in the graph, each number separated by one comma.

Sample Input 1	Sample Output 1

`8 0`
`0 5 3 0 0 11 0 0`
`5 0 5 0 0 0 7 11`
`3 5 0 11 1 7 0 0`
`0 0 11 0 11 0 11 0`
`0 0 1 11 0 1 5 7`
`11 0 7 0 1 0 0 42`
`0 7 0 11 5 0 0 42`
`0 11 0 0 7 42 42 0`
	`0: 0`
	
	`0->1: 5`
	
	`0->2: 3`
	
	`0->2->3: 14`
	
	`0->2->4: 4`
	
	`0->2->4->5: 5`
	
	`0->2->4->6: 9`
	
	`0->2->4->7: 11`

[Graph]

Sample Input 2	Sample Output 2
6 0 0 7 5 0 0 1 7 0 0 0 0 0 5 0 0 2 3 0 0 0 2 0 7 0 0 0 3 7 0 9 1 0 0 0 9 0	0: 0 0->1: 7 0->2: 5 0->2->3: 7 0->2->4: 8 0->5: 1

[Graph]

8. Cut Vertex (or Articulation Point)

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is to find all the cut vertices in the given graph. *When we remove a vertex from a graph then graph will break into two or more graphs. This vertex is called a cut vertex.*

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
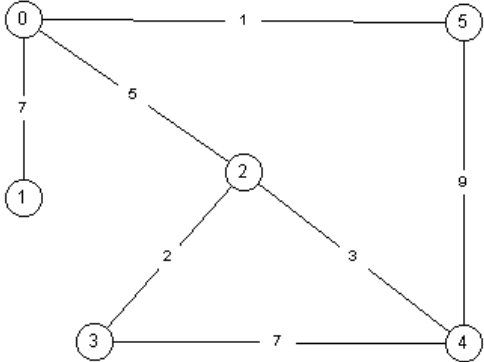5
0 1 1 1 0
1 0 1 0 0
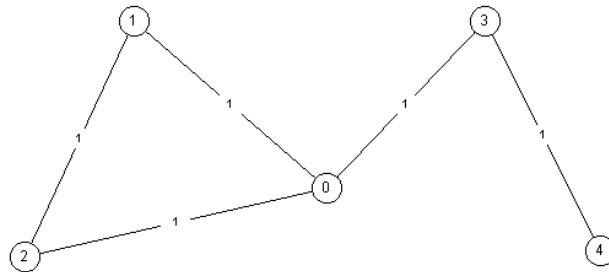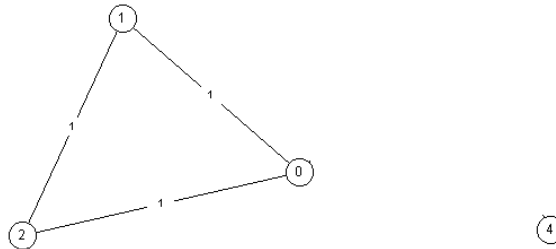1 1 0 0 0
1 0 0 0 1
0 0 0 1 0
```


Figure 5. The undirected graph that created by giving adjacency matrix

Cut vertex: 0

Cut vertex: 3

The input: are stored in the *cutVertex_input.txt* text file:
● The first line contains a positive integer N (1 ≤ N ≤ 100) which is the number of vertex of undirected graph.
● The next *N* line, each line containing *N* integers that represent the adjacency matrix.

The output: the results need to be saved to the *shortestPath_output.txt* text file:
● The first line contains the number of the cut vertex.
● The second line contains the list of the cut vertex, each number separated by one comma.

[Graph]

Sample Input 1	Sample Output 1
5 0 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0	2 0,3

Sample Input 2	Sample Output 2
7 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 	4 1,2,4,5

[Graph]

9. Cut Edge (or Bridge)

Write a program to build an **undirected** graph by giving adjacency matrix.

Your task is to find all cut edges in the given graph. ***When we remove an edge from a graph then graph will break into two or more graphs. This removal edge is called a cut edge or bridge.***

For example,

● Giving the adjacency matrix that used to represent undirected graph as follow:

```
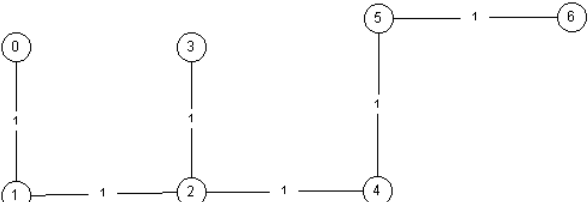5
0 1 1 1 0
1 0 1 0 0
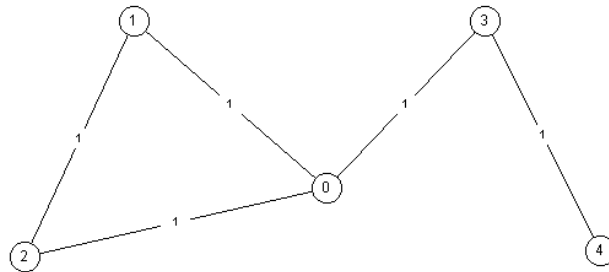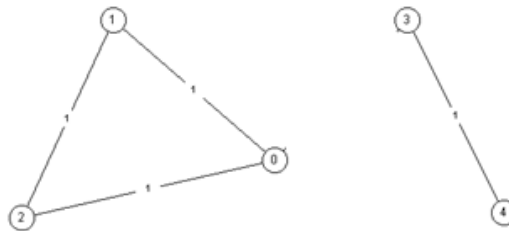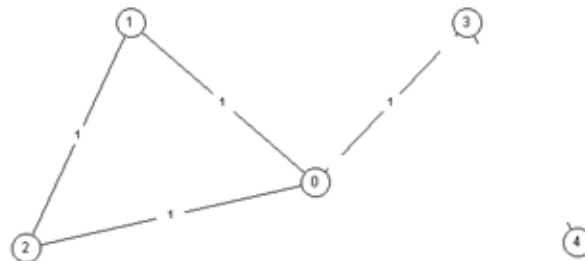1 1 0 0 0
1 0 0 0 1
0 0 0 1 0
```


Figure 5. The undirected graph that created by giving adjacency matrix

Cut edge: (0, 3)

Cut edge: (3, 4)

The input: are stored in the ***cutEdge_input.txt*** text file:
● The first line contains a positive integer N (1 ≤ N ≤ 100) which is the number of vertex of undirected graph.
● The next *N* line, each line containing *N* integers that represent the adjacency matrix.

The output: the results need to be saved to the ***shortestPath_output.txt*** text file:
● The first line contains the number of the cut edge.
● The second line contains the list of the cut edge, each number separated by one comma.

[Graph]

Sample Input 1	Sample Output 1
5 0 1 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0	2 0 3 3 4

Sample Input 2	Sample Output 2
7 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 	6 0 1 1 2 2 3 2 4 4 5 5 6
6 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 	1 2 3