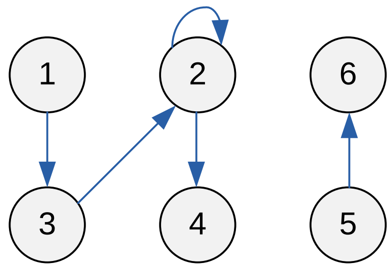
Graphs

In this lecture we introduce the graph data structure and consider three ways of its representation:

- adjacency matrix;
- adjacency list;
- list of edges;

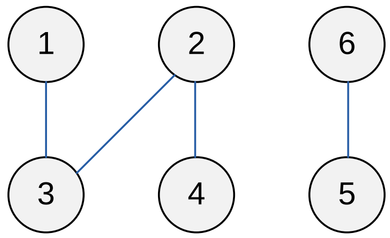
We consider such notations as directed / undirected graphs, loops, degrees of the vertices, regular graphs, complete graphs, hanging vertices.

Directed graph is a pair G = (V, E), where V is a finite set of vertices, E is a set of edges, which is defined as a binary relation on V: $E \subseteq V \times V$. A directed graph is called a **digraph**. Edges - loops connect a vertex to itself.



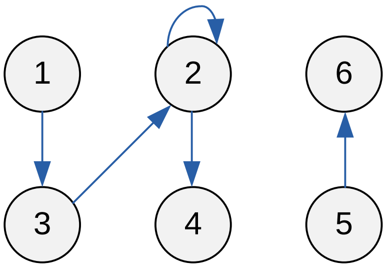
Directed graph G = {V, E}, where
V = {1, 2, 3, 4, 5, 6},
E = {{1, 3}, {3, 2}, {2, 2}, {2, 4}, {5, 6}}

In an ***undirected graph*** the set of edges E is unordered pairs of vertices. An edge (u, v) in an undirected graph is ***incident*** to the vertices u and v. If the graph G contains an edge (u, v), we say that the vertex u is ***adjacent*** to v. For an undirected graph adjacency relation is symmetric.

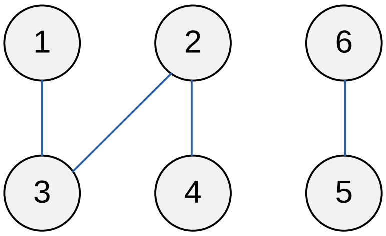


Definition. *The adjacency matrix* of the graph G (V, E), |V| = n, is defined to be a boolean matrix A $n \times n$, that A[i][j] = 1 if and only if between verteces i and j, there is an edge. In case of a weighted graph adjacency matrix is represented two-dimensional numerical matrix, wherein the A[i][j] equals the edge weight unless between nodes i and j, if exists, and A[i][j] = 0 otherwise.

Below given the examples of adjacency matrix:



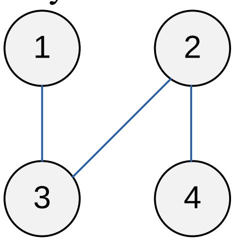
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



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

Check of the presence of edges (v_i, v_j) using the adjacency matrix takes time O(1). Finding all the vertices adjacent to v_i, requires O(n) time (it is enough to look through the i - th line of adjacency matrix).

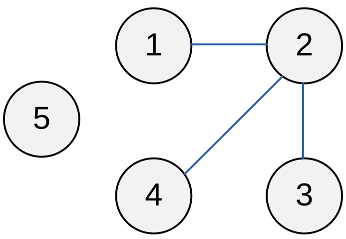
Statement. The adjacency matrix of an undirected graph is symmetric.



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

Undirected graph and its adjacency matrix

E-OLYMP **992. Cities and roads** Galaxy contanis n cities, some of them are connected with two-way roads. Given adjacency matrix of the graph. Find the number of edges in it.

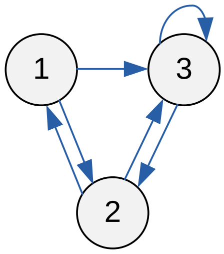


$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

► Graph is undirected. If g is an adjacency matrix, then g[i][j] = g[j][i] for any vertices i and j. For each edge (i, j) we have g[i][j] = g[j][i] = 1. So the number of edges equals to the number of 1's in adjacency matrix, divided by 2.

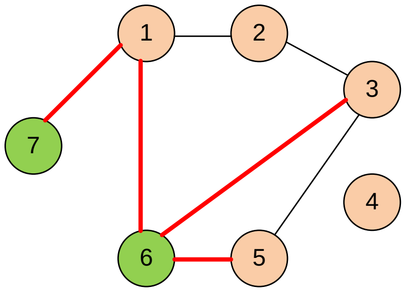
Sample adjacency matrix has six ones, so the number of edges equals to 6 / 2 = 3.

E-OLYMP **5072. Count number of edges** Given adjacency matrix of the ***directed*** graph. Find the number of edges in it.



► Graph is directed. Number of edges equals to the number of ones in adjacency matrix.

E-OLYMP [994. Colored rain](#) Vertices of the graph are colored with three colors. Find the number of edges that connect vertices of different colors.

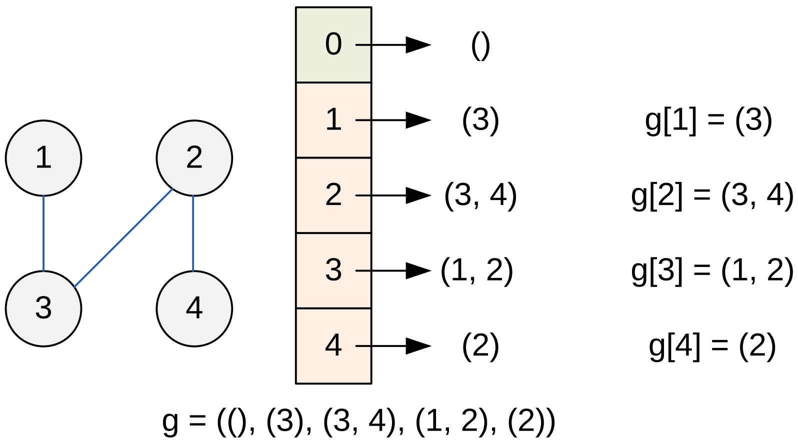


► Read the adjacency matrix. Read the array of colors: col[i] contains the color of the i-th vertex. Count the number of edges (i, j) for which col[i] $\neq$ col[j].

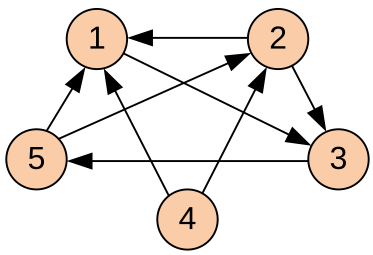
If the graph contains n elements, for storage of the adjacency matrix we use n^2 memory elements. If the graph is sparse (contains a small number of edges), the storage of information in the adjacency matrix is not effective. To do this, use the adjacency list.

Adjacency list contains for each vertex $v \in$ V list of vertices adjacent to it. The number of memory cells required to represent the graph using the adjacency list has order of |V| + |E|. Adjacency list can be declared like

```
vector<vector<int> > g;
```



E-OLYMP [3981. From adjacency matrix to adjacency list](#) Given adjacency matrix of the ***directed*** graph. Print its adjacency list. In the *i*-th line first print the number of edges outgoing from the *i*-th vertex.

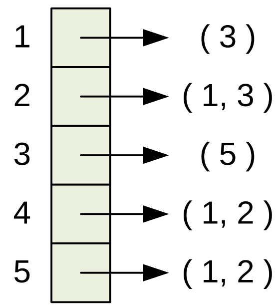


Input

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

Output

```
1 3
2 1 3
1 5
2 1 2
2 1 2
```



► Declare adjacency list:

```
vector<vector<int> > g;
```

Read *n* – the number of vertices in the graph.

```
scanf("%d", &n);
```

Vertices are numbered from 1 to *n*. Resize the vector g.

```
g.resize(n + 1);
```

Read adjacency matrix. For each directed edge (*i*, *j*) add value of *j* to the end of array g[*i*].

```
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
{
  scanf("%d", &val);
  if (val == 1) g[i].push_back(j);
}
```

Print adjacency list.

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

Print the size of g[i] first – the number of edges adjacent to the *i*-th vertex.

```
  printf("%d", g[i].size());
```

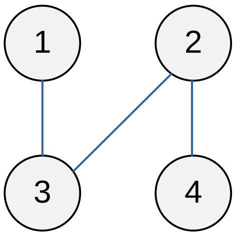
Print the vertices, adjacent to the *i*-th vertex: g[*i*][0], g[*i*][1], … .

```
  for (j = 0; j < g[i].size(); j++)
    printf(" %d", g[i][j]);
  printf("\n");
}
```

E-OLYMP [3982. From adjacency list to adjacency matrix](#) Given adjacency list of the ***directed*** graph. Print its adjacency matrix.

► Read adjacency list and construct adjacency matrix.

List of edges is a list of pairs, where each pair represents two vertices connected with an edge. First line usually contains number of vertices n (sometimes it can contain number e of edges also). Pairs of vertices starts from the second line.



List of edges

4
1 3
2 3
2 4

g[1][3] = g[3][1] = 1
g[2][3] = g[3][2] = 1
g[2][4] = g[4][2] = 1

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

E-OLYMP [4763. From list of edges to adjacency matrix](#) Given list of edges of not directed graph. Print its adjacency matrix.

► For each input not directed edge (a, b) we must assign g[a][b] = g[b][a] = 1, where g is an adjacency matrix.

Declare adjacency matrix.

```
#define MAX 110
int g[MAX][MAX];
```

Read the number of vertices n and edges m.

```
scanf("%d %d", &n, &m);
```

Initialize adjacency matrix g with 0.

```
memset(g, 0, sizeof(g));
```

Read m edges. For each edge (a, b) assign g[a][b] = g[b][a] = 1.

```
for (i = 0; i < m; i++)
{
  scanf("%d %d", &a, &b);
  g[a][b] = g[b][a] = 1;
}
```

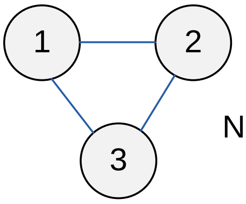
Print the resulting adjacency matrix.

```
for (i = 1; i <= n; i++)
{
  for (j = 1; j <= n; j++)
    printf("%d ", g[i][j]);
  printf("\n");
}
```

An undirected graph is called ***simple*** if it has no loops and an arbitrary pair of vertices is connected by no more than one edge.

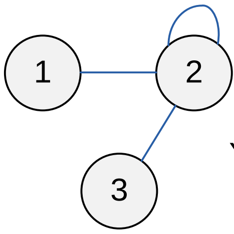
E-OLYMP [4761. Loops](#) Graph is given with an adjacency matrix. Determine whether it contains loops.

```
0 1 1
1 0 1
1 1 0
```



NO

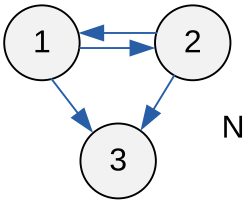
```
0 1 0
1 1 1
0 1 0
```



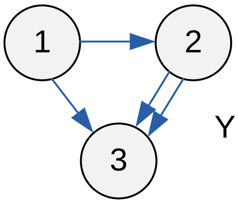
YES

► Graph contains loops if there exists such i for which g[i][i] = 1. If the diagonal of the adjacency matrix contains at least one 1, then answer to the problem is "YES".

E-OLYMP [5073. Multiedges](#) Directed graph is given with a list of edges. Check whether it contains multiedges.



NO



YES

► Let g be an adjacency matrix.

```
int g[101][101];
```

Read the number of vertices n and the number of edges m.

```
scanf("%d %d", &n, &m);
```

Let *flag* = 1 if multiedges exists and *flag* = 0 otherwise. For each input edge (a, b) increase the value of g[a][b] by 1.

```
flag = 0;
for (i = 0; i < m; i++)
{
  scanf("%d %d", &a, &b);
  g[a][b]++;
```

If for some values a and b the value g[a][b] is greater than 1, there exists more than one edge (a, b).

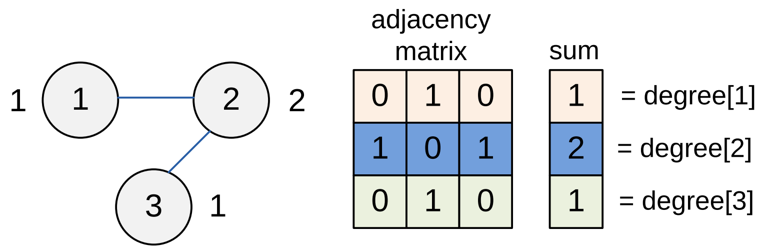
```
  if (g[a][b] > 1) flag = 1;
}
```

Print the answer.

```
if (flag)
  puts("YES");
else
  puts("NO");
```

Degree of a vertex in an undirected graph is the number of incident edges. For directed graphs there is distinguished ***input*** and ***output*** vertices; the sum of the input and output ***powers*** is called the ***degree*** of a vertex.

E-OLYMP **[4764. Degrees of vertices](#)** Graph is given with its adjacency matrix. Find the degrees of all its vertices.



► Let's declare integer array int deg[101], where deg[*i*] equals to the degree of *i*-th vertex. First we need to read adjacency matrix.

```
scanf("%d", &n);
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
  scanf("%d", &g[i][j]);
```

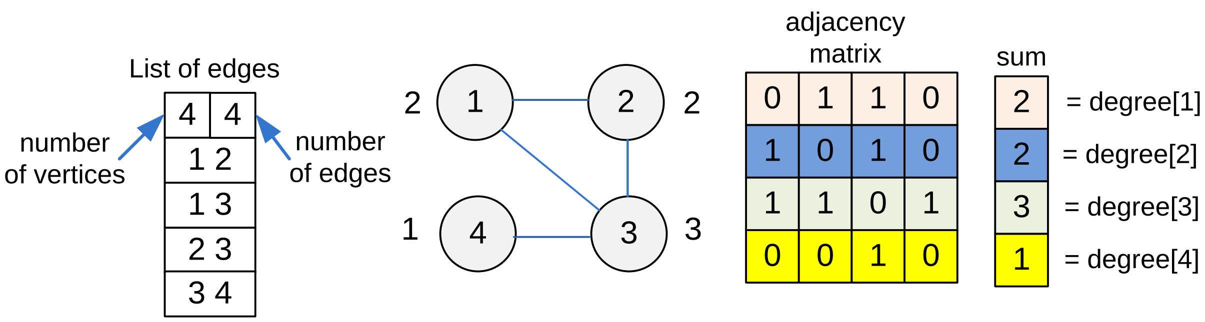
Degree of the *i*-th vertex equals to the sum of elements of *i*-th row in the matrix.

```
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
  deg[i] += g[i][j];
```

Print the degrees of the vertices.

```
for (i = 1; i <= n; i++)
  printf("%d\n", deg[i]);
```

E-OLYMP **[5074. Degrees of vertices by a list of edges](#)** Undirected graph is given with a list of edges. Find the degrees of all its vertices.



► Let's declare integer array int deg[101], where deg[*i*] equals to the degree of *i*-th vertex. First we need to read the number of vertices and the number of edges.

```
scanf("%d %d", &n, &m);
```

For each input edge (*a*, *b*) we need to increase the degree of vertices *a* and *b*.

```
for (i = 0; i < m; i++)
{
```

```
  scanf("%d %d", &a, &b);
  deg[a]++; deg[b]++;
}
```

Print the degrees of the vertices.

```
for (i = 1; i <= n; i++)
  printf("%d\n", deg[i]);
```

E-OLYMP **[993. Traffic lights]** There are m tunnels and n junctions, each tunnel connects two crossroads. There is a traffic light in every tunnel before every intersection. Find the number of traffic lights at each intersection.

► Problem is similar to **5074** *Degrees of vertices by a list of edge*.

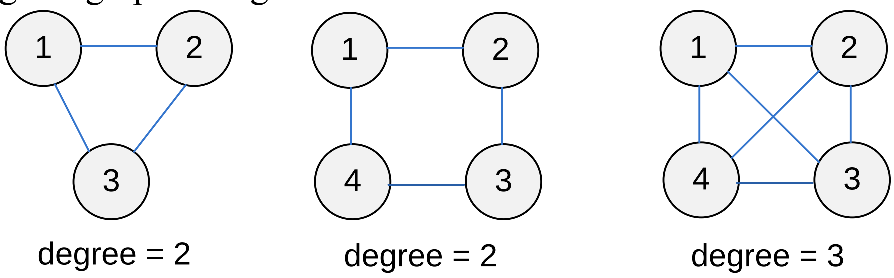
E-OLYMP **[5080. Number of hanging vertices 1]** Given an undirected graph with an adjacency matrix. Count the number of hanging vertices in it. The vertex is hanging, if its degree is 1.

► Find the degree of each vertex. Count the number of vertices with degree 1.

E-OLYMP **[5088. Number of hanging vertices 2]** Given an undirected graph with a list of edges. Count the number of hanging vertices in it. The vertex is hanging, if its degree is 1.

► Find the degree of each vertex. Count the number of vertices with degree 1.

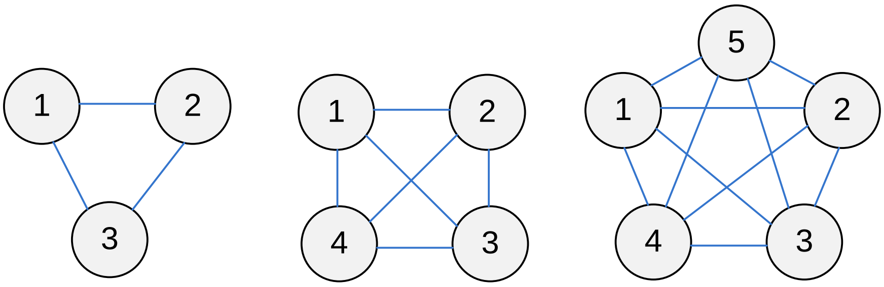
E-OLYMP **[5076. Regular graph]** Undirected graph is called **regular**, if all its vertices have the same degree. Graph is given with a list of edges. Check, is it regular. Samples of regular graphs are given below:



► Find the degree of each vertex in *deg* array. Graph will be **regular**, if all elements in *deg* array are the same.

A simple graph is called **complete** if every pair of vertices is connected by an edge. This graph contains C_n^2 edges.

E-OLYMP **[3987. Complete graph]** Undirected graph is given with a list of edges. Check, is it ***complete***. Samples of complete graphs are given below:



► Construct the adjacency matrix. Graph is complete if g[i][j] = 1 for any $i \neq j$. Adjacency matrix must contain 1 in all positions (except the main diagonal).