MATRICES FOR GRAPH REPRESENTATION

- GRAPH DATA STRUCTURE
- GRAPH TERMINOLOGY
- GRAPH APPLICATIONS
- GRAPH REPRESENTATION
- INCIDENCE MATRIX
- ADJACENCY MATRIX
- PRACTICE: TASK SEQUENCE PROBLEM

Author: prof. Yevhenii Borodavka

GRAPH DATA STRUCTURE

A graph data structure is a collection of nodes that have data and are connected to other nodes.

Let's try to understand this through an example. On Facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note... anything that has data is a node.

Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.

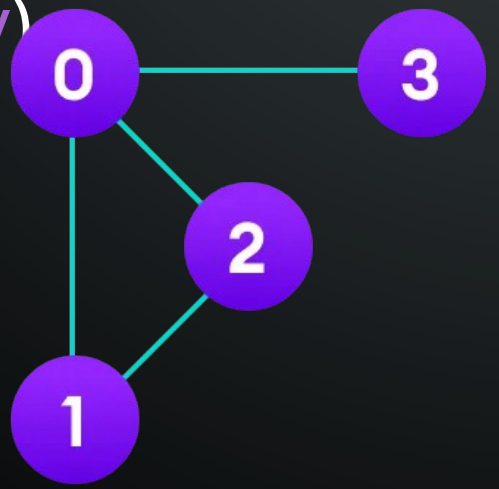
GRAPH DATA STRUCTURE

All of Facebook is then a collection of these nodes and edges. This is because Facebook uses a graph data structure to store its data.

GRAPH DATA STRUCTURE

More precisely, a graph is a data structure (**V**, **E**) that consists of

- A collection of vertices **V**
- A collection of edges **E**, represented as ordered pairs of vertices (u, v)



In the graph,

V = {0, 1, 2, 3}

E = {(0,1), (0,2), (0,3), (1,2)}

G = {**V**, **E**}

GRAPH TERMINOLOGY

Adjacency: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.

Path: A sequence of edges that allows you to go from vertex **A** to vertex **B** is called a path. 0-1, 1-2 and 0-2 are paths from vertex **0** to vertex **2**.

Undirected Graph: A graph in which the direction of the edge is not defined.

Directed Graph: A graph in which an edge (u, v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge

GRAPH APPLICATIONS

Graph is a data structure which is used extensively in our real-life.

- Social Network: Each user is represented as a node and all their activities, suggestion and friend list are represented as an edge between the nodes.
- Google Maps: Various locations are represented as vertices or nodes and the roads are represented as edges and graph theory is used to find shortest path between two nodes.
- Recommendations on e-commerce websites: The “Recommendations for you” section on various e-commerce websites uses graph theory to recommend items of similar type to user’s choice.

GRAPH REPRESENTATION

Graphs are commonly represented in two ways:

- Matrices: Incidence and Adjacency
- Adjacency list

Depends on graph type the information in matrices and list nodes can be different.

We will consider matrix representations only in this lecture.

INCIDENCE MATRIX

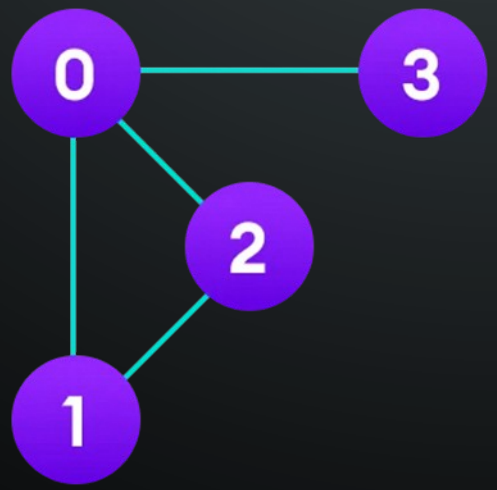
Incidence matrix is that matrix which represents the graph such that with the help of that matrix we can draw a graph. As in every matrix, there are also rows and columns in incidence matrix.

The rows of the matrix represent the number of nodes and the column of the matrix represent the number of edges in the given graph. If there are **N** number of rows in a given incidence matrix, that means in a graph there are **N** number of nodes. Similarly, if there are **M** number of columns in that given incidence matrix, that means in that graph there are **M** number of edges.

INCIDENCE MATRIX

For undirected graphs incidence matrix consists of values $\mathbf{B}_{ij}$:

- 1 – if vertex v_i is incidence with edge e_j
- 0 – otherwise

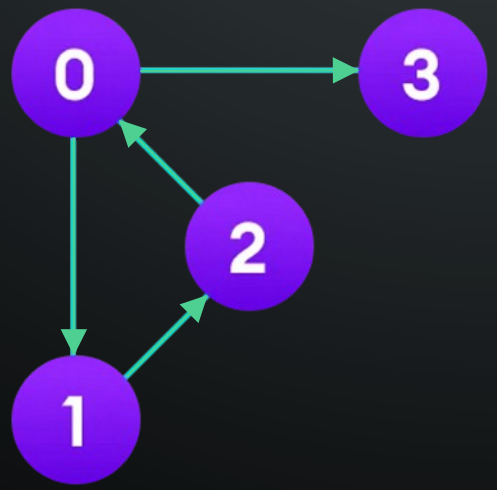


M	0-1	0-2	0-3	1-2
0	1	1	1	0
1	1	0	0	1
2	0	1	0	1
3	0	0	1	0

INCIDENCE MATRIX

For directed graphs incidence matrix consists of values $\mathbf{B}_{ij}$:

- -1 – if edge e_j leaves vertex v_i
- 1 – if edge e_j enters vertex v_i
- 0 – otherwise

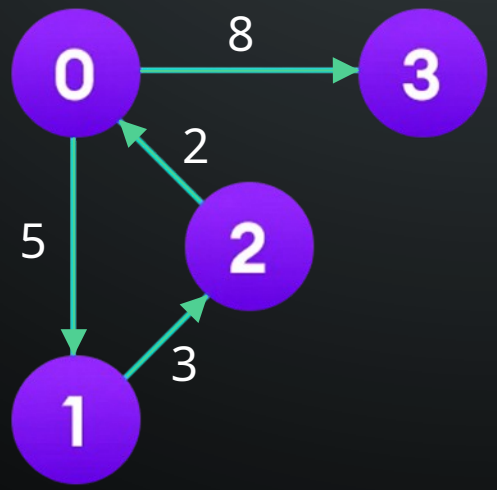


M	0-1	0-2	0-3	1-2
0	-1	1	-1	0
1	1	0	0	-1
2	0	-1	0	1
3	0	0	1	0

INCIDENCE MATRIX

A weighted graph can be represented using the weight of the edge in place of a 1.

For example, the incidence matrix of the graph to the left is:



M	0-1	0-2	0-3	1-2
0	-5	2	-8	0
1	5	0	0	-3
2	0	-2	0	3
3	0	0	8	0

INCIDENCE MATRIX

Advantages:

- Good for sparse graphs
- Constant-time vertex lookups – **O**(1)

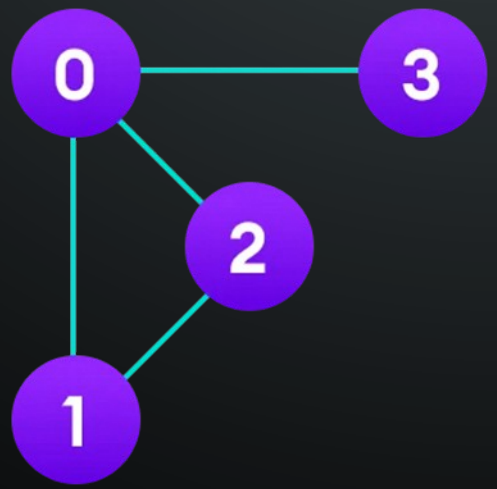
Disadvantages:

- Takes up a lot of space for dense graphs
- More complex to implement and use in computer programs
- Slow to iterate over all edges

ADJACENCY MATRIX

An adjacency matrix is a 2D array of **V** x **V** vertices. Each row and column represent a vertex.

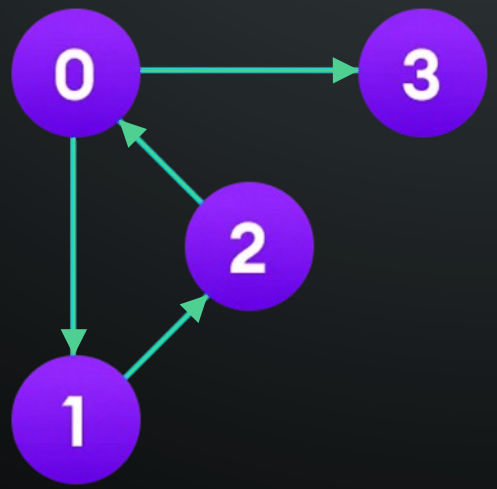
For undirected graph if the value of any matrix element $\mathbf{B}_{ij}$ is **1**, it represents that there is an edge connecting vertex i and vertex j.



M	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

ADJACENCY MATRIX

For directed graphs the value of adjacency matrix $\mathbf{B}_{ij}$ is **1** if there is a edge with direction from vertex i to vertex j.

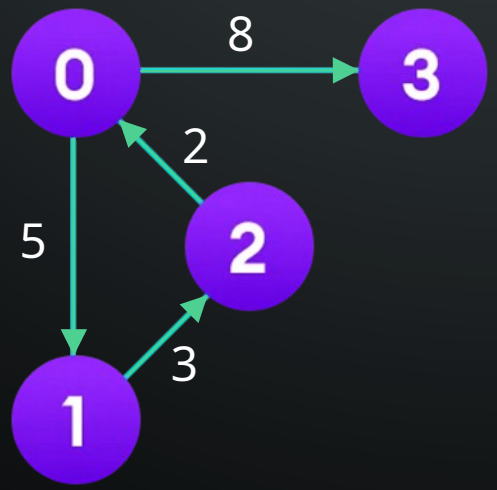


M	0	1	2	3
0	0	1	0	1
1	0	0	1	0
2	1	0	0	0
3	0	0	0	0

ADJACENCY MATRIX

A weighted graph can be represented using the weight of the edge in place of a 1.

For example, the adjacency matrix of the graph to the left is:



M	0	1	2	3
0	0	5	0	8
1	0	0	3	0
2	2	0	0	0
3	0	0	0	0

ADJACENCY MATRIX

Advantages:

- Easy to implement and use in computer programs
- Good for dense graphs (graphs with many edges)
- Constant-time edge lookups – **O**(1)

Disadvantages:

- Takes up a lot of space for sparse graphs (graphs with few edges)
- Slow to iterate over all edges or vertices

PRACTICE: TASK SEQUENCE PROBLEM

Problem. You have **N** tasks (1<=**N**<=100) you need to do. You also have a **K** pair of the tasks (1<=**K**<=100) which describes the task dependency – the second task must be done after the first one. You need to sort the tasks in the correct order for execution.

Example. You have **N**=9 tasks and **K**=9 pairs: (4,1) (1,2) (2,3) (2,7) (5,6) (7,6) (1,5) (8,5) (8,9). Correct orders are 8,9,4,1,5,2,3,7,6 or 4,1,2,3,8,5,7,6,9 or 4,8,9,1,2,3,5,7,6. But this order 4,1,5,2,3,7,6,8,9 is incorrect because the task 5 can't be done before the task 8.

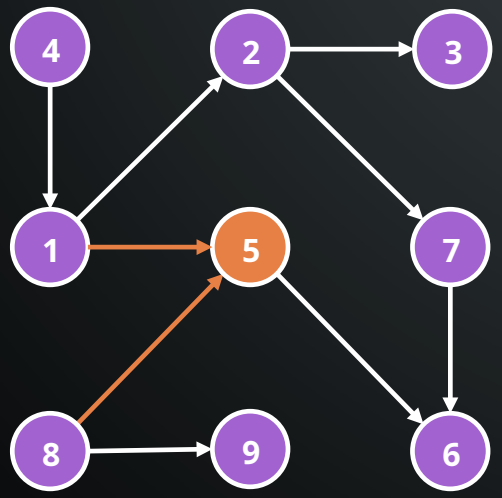
Task. Create a program using C/C++/Python to solve this problem.

Input. Two strings: the first one with two numbers **N** and **K** divided by space and the second one with **K** pairs of the tasks indexes divided by spaces.

Output. The correct order of the task execution (tasks indexes divided by

PRACTICE: TASK SEQUENCE PROBLEM

Graph from example and its adjacency matrix **M**. The **D** is ‘done’ vector.



M	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0

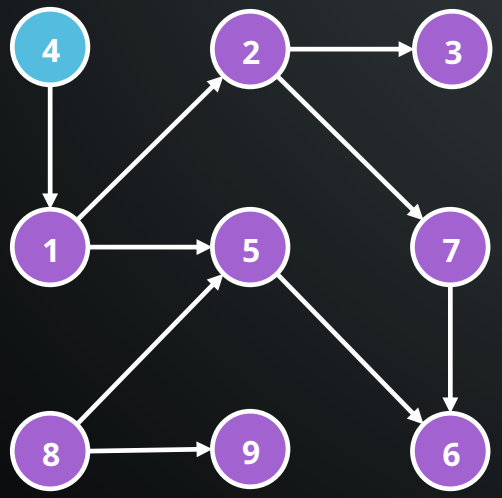
D
0
0
0
0
0
0
0
0
0

A									

The **A** is output result.

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **1**.



M	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0

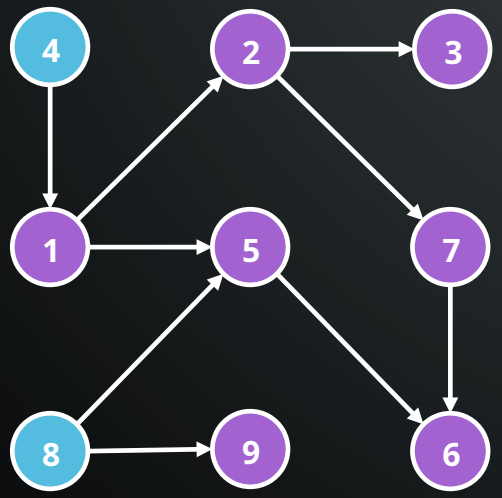
D
0
0
0
1
0
0
0
0
0

The **A** is output result.

A	4								

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **2**.



M	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0

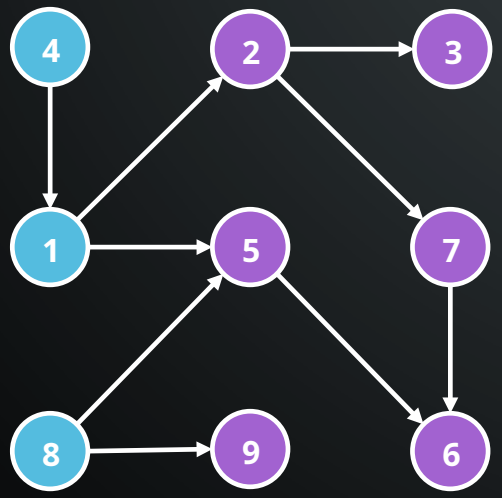
D
0
0
0
1
0
0
0
1
0

The **A** is output result.

A	4	8							

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **3**.



M	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0

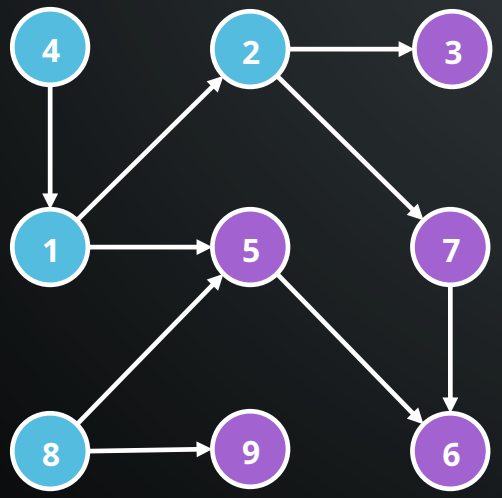
D
1
0
0
1
0
0
0
0
1
0

The **A** is output result.

A	4	8	1						

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **4**.



M	1	2	3	4	5	6	7	8	9
1	0	1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	0

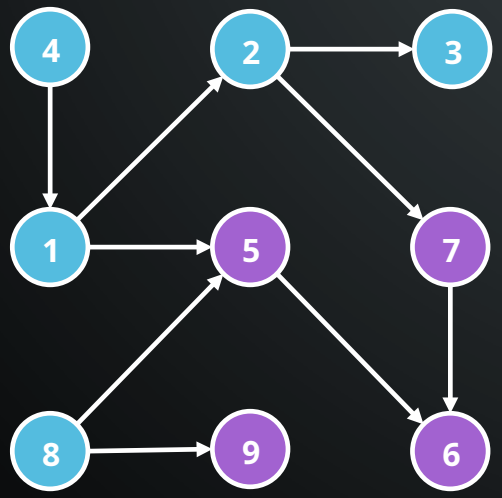
D
1
1
0
1
0
0
0
1
0

The **A** is output result.

A	4	8	1	2					

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **5**.



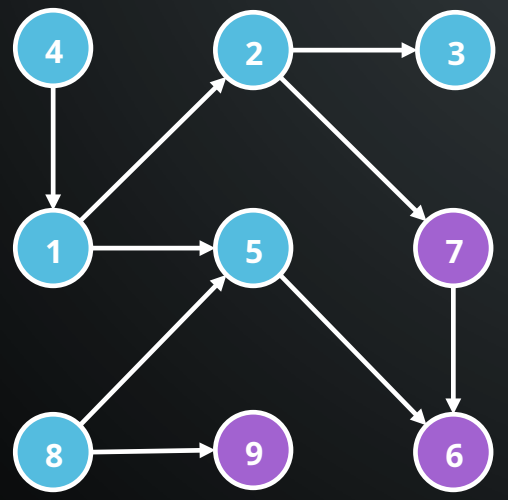
M	1	2	3	4	5	6	7	8	9	D
1	0	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0

The **A** is output result.

A	4	8	1	2	3				

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **6**.



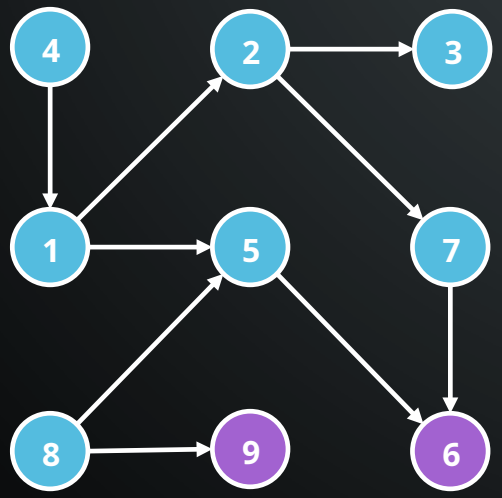
M	1	2	3	4	5	6	7	8	9	D
1	0	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0

The **A** is output result.

A	4	8	1	2	3	5			

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **7**.



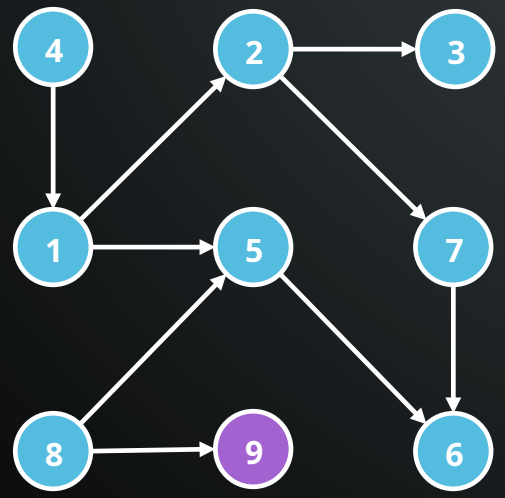
M	1	2	3	4	5	6	7	8	9	D
1	0	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	1
8	0	0	0	0	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0

The **A** is output result.

A	4	8	1	2	3	5	7		

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **8**.



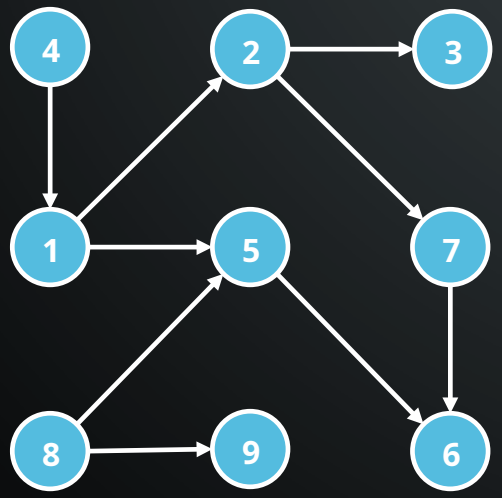
M	1	2	3	4	5	6	7	8	9	D
1	0	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	0	1
8	0	0	0	0	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0

The **A** is output result.

A	4	8	1	2	3	5	7	6	

PRACTICE: TASK SEQUENCE PROBLEM

Traverse matrix **M** by columns **N** times. Step **9**.



M	1	2	3	4	5	6	7	8	9	D
1	0	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	1	0	0	1
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	1
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	0	1
8	0	0	0	0	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	1

The **A** is output result.

A	4	8	1	2	3	5	7	6	9

PRACTICE: TASK SEQUENCE PROBLEM

Code example:

```
#include <iostream>
#define MAX 101
int N, K, A[MAX], M[MAX][MAX];
int main()
{
    cin >> N >> K;
    int f, s;
    for(int i = 0; i < K; i++)
    {
        cin >> f >> s;
        M[f][s] = 1;
    }
    for(int i = 1; i <= N; i++)
    {
        int v = 0;
        // Your code here
        cout << v << " ";
    }
    return 0;
}
```

THANK YOU!