**Problem A**

Given a directed graph, print its adjacency information.

You can use an array of vectors in c++ to store the graph in a adjacency list. Below are a few tips to use it.

**Declaration**:

vector<int> g[100];

(Assuming your nodes are numbered 0 to 99. You need to change the size, if then number of nodes are different)

**Initialization:**

for(i =0;i<N;i++)g[i].clear();

(Assuming you have N nodes in total numbered 0 to N-1. g[i].clear() function empties the i-th vector)

**Adding edge:**

To add a particular edge u -> v: g[u].push\_back(v);

**Traversing the list of edges for a particular node u:**

for(i=0;i<g[u].size();i++){

v = g[u][i]; // so you have an edge from u to v

}

**Input**

First line: N ( 0<N<=100000), number of nodes.

Second line: M (0<M<=100000), number of edges.

Next M lines, each: U V (0<=U, V<N), defines an edge from U to V.

**Output**

Total N lines, each for one node. Print the adjacent nodes in the input order for that node. See sample for more clarification.

|  |  |
| --- | --- |
| 6  6  1 2  1 3  0 1  2 4  4 5  5 1 | 0: 1  1: 2 3  2: 4  3:  4: 5  5: 1 |

**Problem B**

Given an unweighted directed graph, find shortest path from a single source node.

**Input**

First line: N ( 0&lt;N&lt;=100000), number of nodes.

Second line: M (0&lt;M&lt;=200000), number of edges.

Next M lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Total N lines, each for one node. Print the shortest path of each node considering node 0 as the source. See sample for more clarification.

|  |  |
| --- | --- |
| 6  6  1 2  1 3  0 1  2 4  5 4  5 1 | 0: 0  1: 1  2: 2  3: 2  4: 3  5: Not Reachable |

**Problem C**

Given an unweighted undirected graph, run dfs and find the components with discovery time and finishing time. Assume that you are running dfs in increasing order.

**Input**

First line: N ( 0&lt;N&lt;=1000), number of nodes.

Second line: M (0&lt;M&lt;=20000), number of edges.

Next M lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Total N lines, each for one node. Print the component number, discovery time and finishing time each node. See sample for more clarification.

|  |  |
| --- | --- |
| 6  6  1 2  1 3  0 4  2 3  5 4  5 0 | 0: 1 1 6  1: 2 7 12  2: 2 8 11  3: 2 9 10  4: 1 2 5  5: 1 3 4 |

**Problem D**

Given an directed graph, find if the graph has any cycle.

**Input**

First line: N ( 0&lt;N&lt;=100000), number of nodes.

Second line: M (0&lt;M&lt;=200000), number of edges.

Next M lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Print Yes/No. See sample for more clarification.

|  |  |
| --- | --- |
| 6  4  1 2  1 3  0 4  5 4 | No |
| 3  3  1 2  2 0  0 1 | Yes |

**Problem E**

Given an NxM grid, with blocked cells, find the shortest path to go from cell (0,0) to (N-1,M-1). From each cell you can go to four directions: Up, Down, Left and Right. You can never step into a blocked cell.

**Input**

First line: N (0&lt;N&lt;=100), M (0&lt;M&lt;=100).

Next N lines, each contains M characters. ‘.’ indicates empty cell. ‘#’ indicates blocked cells.

**Output**

One line with the shortest path. See sample for more clarification. If it’s not possible print -1.

|  |  |
| --- | --- |
| 5 3  ...  ##.  ...  .##  ... | 10 |
| 5 3  ...  ##.  ...  .#.  ... | 6 |

**Problem F**

Given an unweighted undirected graph, find shortest path from a single source node. Print the path. If there are multiple shortest path, print the lexicographically smallest one.

**Input**

First line: N ( 0&lt;N&lt;=100), number of nodes.

Second line: M (0&lt;M&lt;=2000), number of edges.

Next M lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Total N lines, each for one node. Print the shortest path of each node considering node 0 as the source. See sample for more clarification.

|  |  |
| --- | --- |
| 6  5  1 3  1 2  0 1  3 4  2 4 | 0: 0  1: 0 1  2: 0 1 2  3: 0 1 3  4: 0 1 2 4  5: Not Reachable |

Explanation:

For Node 4, there are 2 shortest paths, 0 1 2 4 and 0 1 3 4. 0 1 2 4 is lexicographically smaller.

**Problem G**

Given an unweighted undirected graph, find if the graph is bipartite. In a bipartite is graph, the vertices can be divided into two sets A and B, such that, there will be no edges between two nodes of set A. Also there will be no edges between two nodes of set B.

**Input**

First line: N ( 0&lt;N&lt;=100), number of nodes.

Second line: M (0&lt;M&lt;=2000), number of edges.

Next M lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Yes/No. See sample for clarification

|  |  |
| --- | --- |
| 6  5  1 3  1 2  0 1  3 4  2 4 | Yes |
| 3  3  0 1  0 2  1 2 | No |

Explanation:

**In the first sample, the sets are {1, 4} and {0, 2, 3}.**

**Problem H**

Given a unweighted undirected unrooted tree, find its diameter. The diameter of a tree is distance between the nodes which are most further apart.

**Input**

First line: N ( 0&lt;N&lt;=100000), number of nodes.

Next N-1 lines, each: U V (0&lt;=U, V&lt;N), defines an edge from U to V.

**Output**

Diameter. See sample for clarification

|  |  |
| --- | --- |
| 6  1 3  1 2  0 1  3 4  2 5  6 7  0 1  0 3  3 4  1 4  1 3  2 4  3 5 | 4 |

Explanation:

**The most distant nodes are 4 and 5 and the distance between them is 4.**