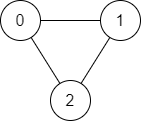
**1971. Find if Path Exists in Graph**

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to n - 1 (**inclusive**). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [ui, vi] denotes a bi-directional edge between vertex ui and vertex vi. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex source to vertex destination.

Given edges and the integers n, source, and destination, return true*if there is a****valid path****from*source*to*destination*, or*false*otherwise.*



**Input:** n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2

**Output:** true

**Explanation:** There are two paths from vertex 0 to vertex 2:

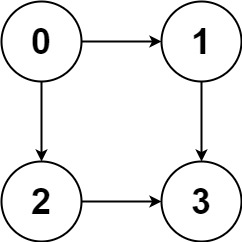
- 0 → 1 → 2

- 0 → 2

**797. All Paths From Source to Target**

Given a directed acyclic graph (**DAG**) of n nodes labeled from 0 to n - 1, find all possible paths from node 0 to node n - 1 and return them in **any order**.

The graph is given as follows: graph[i] is a list of all nodes you can visit from node i (i.e., there is a directed edge from node i to node graph[i][j]).



**Input:** graph = [[1,2],[3],[3],[]]

**Output:** [[0,1,3],[0,2,3]]

**Explanation:** There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.

**841. Keys and Rooms**

There are n rooms labeled from 0 to n - 1 and all the rooms are locked except for room 0. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of **distinct keys** in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array rooms where rooms[i] is the set of keys that you can obtain if you visited room i, return true *if you can visit****all****the rooms, or* false *otherwise*.

**Input:** rooms = [[1],[2],[3],[]]

**Output:** true

**Explanation:**

We visit room 0 and pick up key 1.

We then visit room 1 and pick up key 2.

We then visit room 2 and pick up key 3.

We then visit room 3.

Since we were able to visit every room, we return true.

**200. Number of Islands**

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Input:** grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

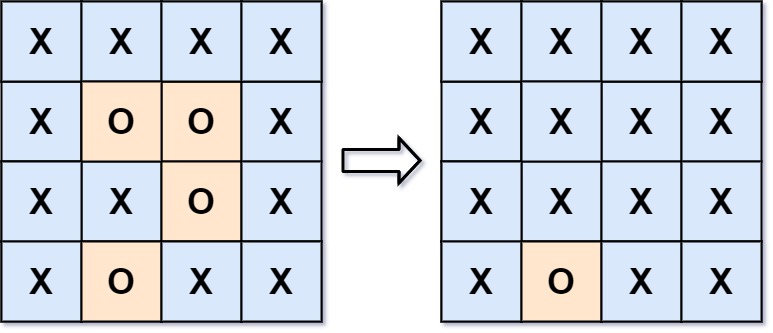
]

**Output:** 3

**130. Surrounded Regions**

Given an m x n matrix board containing 'X' and 'O', *capture all regions that are 4-directionally surrounded by* 'X'.

A region is **captured** by flipping all 'O's into 'X's in that surrounded region.



**Input:** board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]

**Output:** [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]

**Explanation:** Notice that an 'O' should not be flipped if:

- It is on the border, or

- It is adjacent to an 'O' that should not be flipped.

The bottom 'O' is on the border, so it is not flipped.

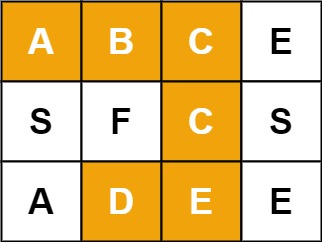
The other three 'O' form a surrounded region, so they are flipped.

**133. Clone Graph**

**79. Word Search**

Given an m x n grid of characters board and a string word, return true *if* word *exists in the grid*.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.



**Input:** board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"

**Output:** true

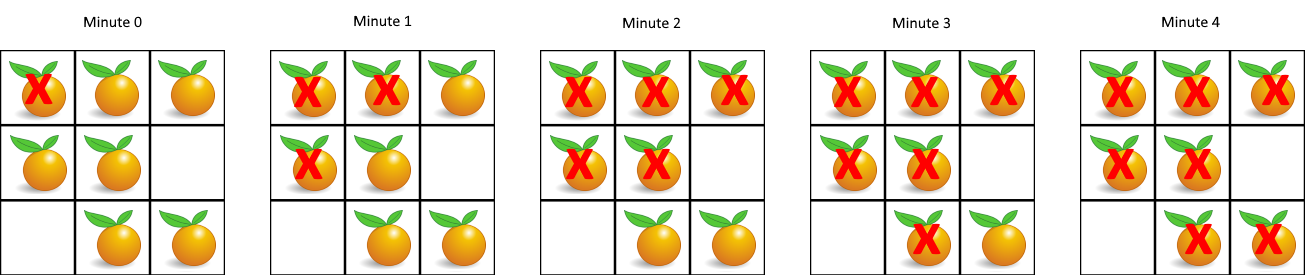
**994. Rotting Oranges**

You are given an m x n grid where each cell can have one of three values:

* 0 representing an empty cell,
* 1 representing a fresh orange, or
* 2 representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange*. If *this is impossible, return* -1.



**Input:** grid = [[2,1,1],[1,1,0],[0,1,1]]

**Output:** 4

**688. Knight Probability in Chessboard**

On an n x n chessboard, a knight starts at the cell (row, column) and attempts to make exactly k moves. The rows and columns are **0-indexed**, so the top-left cell is (0, 0), and the bottom-right cell is (n - 1, n - 1).

A chess knight has eight possible moves it can make, as illustrated below. Each move is two cells in a cardinal direction, then one cell in an orthogonal direction.



Each time the knight is to move, it chooses one of eight possible moves uniformly at random (even if the piece would go off the chessboard) and moves there.

The knight continues moving until it has made exactly k moves or has moved off the chessboard.

Return *the probability that the knight remains on the board after it has stopped moving*.

**Input:** n = 3, k = 2, row = 0, column = 0

**Output:** 0.06250

**Explanation:** There are two moves (to (1,2), (2,1)) that will keep the knight on the board.

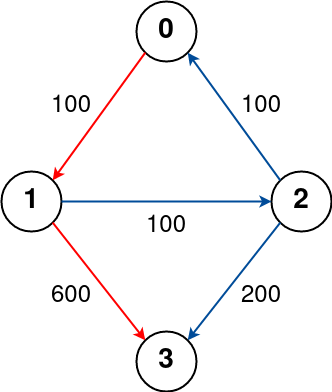
From each of those positions, there are also two moves that will keep the knight on the board.

The total probability the knight stays on the board is 0.0625.

**787. Cheapest Flights Within K Stops**

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, pricei] indicates that there is a flight from city fromi to city toi with cost pricei.

You are also given three integers src, dst, and k, return ***the cheapest price****from*src*to*dst*with at most*k*stops.*If there is no such route, return-1.



**Input:** n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1

**Output:** 700

**Explanation:**

The graph is shown above.

The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.

Note that the path through cities [0,1,2,3] is cheaper but is invalid because it uses 2 stops.

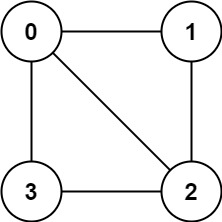
**785. Is Graph Bipartite?**

There is an **undirected** graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v. The graph has the following properties:

* There are no self-edges (graph[u] does not contain u).
* There are no parallel edges (graph[u] does not contain duplicate values).
* If v is in graph[u], then u is in graph[v] (the graph is undirected).
* The graph may not be connected, meaning there may be two nodes u and v such that there is no path between them.

A graph is **bipartite** if the nodes can be partitioned into two independent sets A and B such that **every** edge in the graph connects a node in set A and a node in set B.

Return true*if and only if it is****bipartite***.



**Input:** graph = [[1,2,3],[0,2],[0,1,3],[0,2]]

**Output:** false

**Explanation:** There is no way to partition the nodes into two independent sets such that every edge connects a node in one and a node in the other.