Algorithms

DFS Homework 4

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

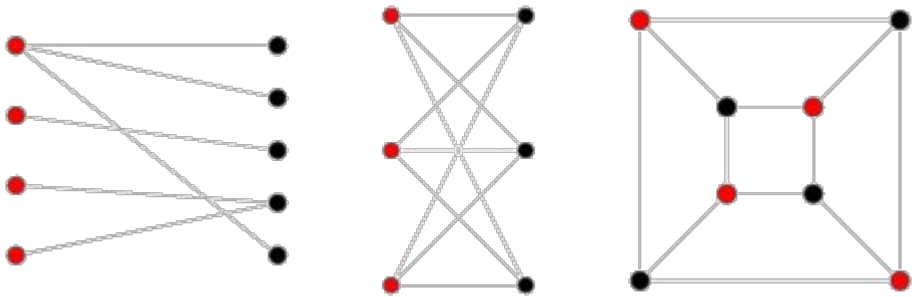
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem #1: LeetCode 785 - Is Graph Bipartite?

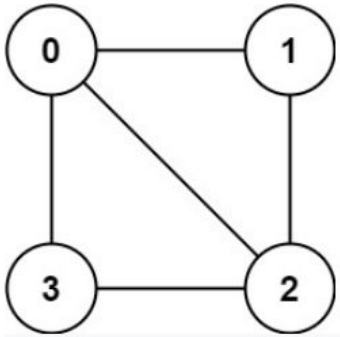
- Given undirected graph, return true if it is bipartite
  - No self-edges or multiple edges
  - Graph is given as an adjacency list (as we used to build)
  - The graph may be multiple components
- A **bipartite graph** is a graph whose vertices can be divided into **two disjoint** groups so that **every edge** connects two vertices from **different** groups
  - In other words, there are no edges which connect vertices from the same groups
- bool isBipartite(vector<vector<int>> &graph)

Bipartite Graphs

- Observe: edges only from group 1 to group 2
  - NO edges between the same group!

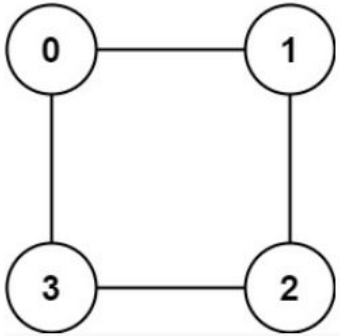


Img src



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.

Example 2:

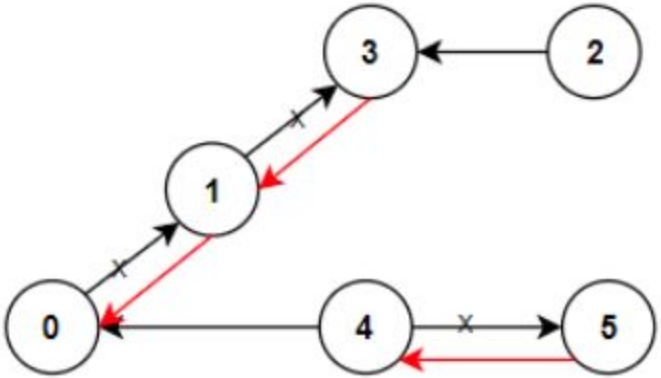


Input: graph = [[1,3],[0,2],[1,3],[0,2]]
Output: true
Explanation: We can partition the nodes into two sets: {0, 2} and {1, 3}.

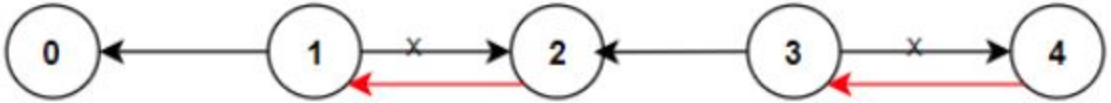
Problem #2: LeetCode 1466 - Reorder Routes to Make All Paths Lead to the City Zero

- Assume we have undirected tree (single path from any node to another).
- Then someone selected a specific direction for edge (now directed graph).
- Your task is **re**orienting the **minimum number of edges** edges such that there is a path from any node to node (0).
  - Return the number of reoriented edges.
  - Note: New graph is still a directed graph. Just we flipped the direction of some edges
- int minReorder(int nodes, vector<vector<int>> &connections)
  - Each connection vector<int>: is the current **directed** edge
  - connections.length = n - 1 (like a tree)
  - 2 <= nodes <= $5 * 10^4$

Example 1:



Input: n = 6, connections = [[0,1],[1,3],[2,3],[4,0],[4,5]]
Output: 3
Explanation: Change the direction of edges show in red such that each node can reach the node 0 (capital).




```
Input: n = 5, connections = [[1,0],[1,2],[3,2],[3,4]]
Output: 2
Explanation: Change the direction of edges show in red such that each node can reach the node
0 (capital).
```

Example 3:

```
Input: n = 3, connections = [[1,0],[2,0]]
Output: 0
```

Problem #3: LeetCode 1631. Path With Minimum Effort

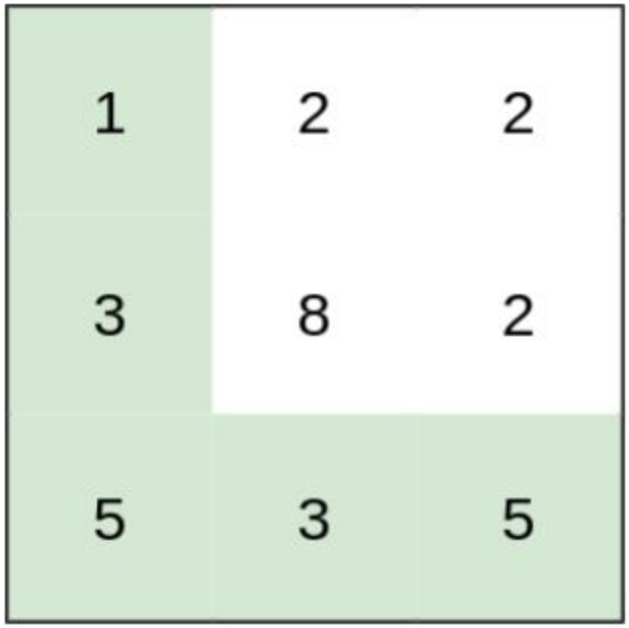
You are a hiker preparing for an upcoming hike. You are given `heights`, a 2D array of size `rows x columns`, where `heights[row][col]` represents the height of cell `(row, col)`. You are situated in the top-left cell, `(0, 0)`, and you hope to travel to the bottom-right cell, `(rows-1, columns-1)` (i.e., **0-indexed**). You can move **up**, **down**, **left**, or **right**, and you wish to find a route that requires the minimum **effort**.

A route's **effort** is the **maximum absolute difference** in heights between two consecutive cells of the route.

Return *the minimum **effort** required to travel from the top-left cell to the bottom-right cell.*

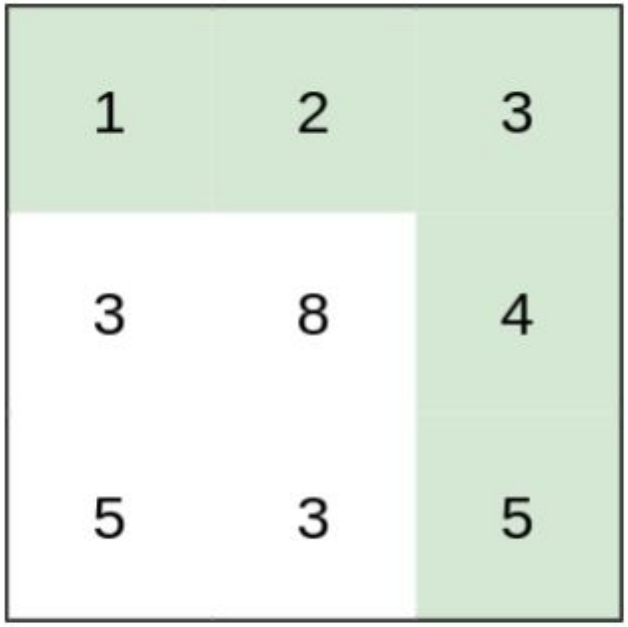
- int minimumEffortPath(vector<vector<int>> &heights)
  - 1 <= rows, columns <= 100
  - 1 <= heights[i][j] <= 10^6

Example 1:



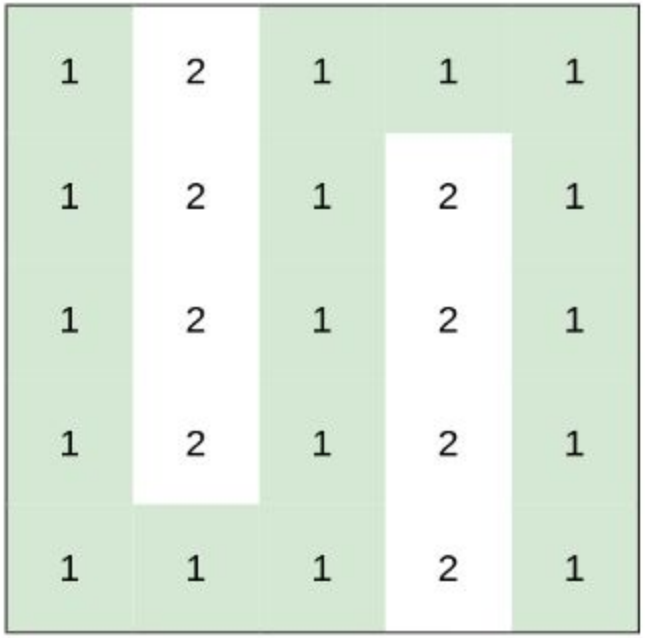
Input: heights = [[1,2,2],[3,8,2],[5,3,5]]
Output: 2
Explanation: The route of [1,3,5,3,5] has a maximum absolute difference of 2 in consecutive cells.
This is better than the route of [1,2,2,2,5], where the maximum absolute difference is 3.

Example 2:



Input: heights = [[1,2,3],[3,8,4],[5,3,5]]
Output: 1
Explanation: The route of [1,2,3,4,5] has a maximum absolute difference of 1 in consecutive cells, which is better than route [1,3,5,3,5].

Example 3:




```
Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]
Output: 0
Explanation: This route does not require any effort.
```

Problem #4: O(V) Cycle Detection in **undirected**

- Give an **undirected** graph, return true if it contains a **cycle**
  - No self-loops or parallel edges in the graph
  - Given an example and analysis, why your approach will fail for a **directed** graph
    - You can use only 3 nodes!
    - Don’t code how to check cycle existence in a directed graph
- bool has_cycle_undirected(GRAPH &graph_adj_list)
  - This function is part of your code must be O(|V|). **Prove it**

Problem #4: O(V) Cycle Detection in **undirected**

- Start your program by reading number of test cases.
- Then for each case read 2 numbers (N nodes and M edges).
- Then read M lines, each line has two **0-based indices** for an undirected edge
- For each test case, print a single line
  - Yes if there is a cycle
  - No otherwise
- Develop your own test cases.
- Use my input/output files and compare

```
12
2
34 3
40 1
51 2
62 3
7
86 6
90 1
100 2
111 3
121 4
132 5
141 5
```

```
No
Yes
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."