COL106 Lab-Graphs

April 2025

1 Loud and Rich

There is a group of n people labeled from 0 to $n-1$ where each person has a different amount of money and a different level of quietness.
You are given an array `richer` where `richer[i]` $= [a_i, b_i]$ indicates that person a_i has more money than person b_i, and an integer array `quiet` where `quiet[i]` is the quietness of the ith person. All the given data in `richer` are logically correct (i.e., the data will not lead you to a situation where x is richer than y and y is richer than x at the same time).
Return an integer array `answer` where `answer[x]` $= y$ if y is the least quiet person (i.e., the person y with the smallest value of `quiet[y]`) among all people who definitely have equal to or more money than person x.

- **Input:** `richer = [[1, 0], [2, 1], [3, 1], [3, 7], [4, 3], [5, 3], [6, 3]]`, `quiet = [3, 2, 5, 4, 6, 1, 7, 0]`
- **Output:** `[5, 5, 2, 5, 4, 5, 6, 7]`

2 * Possible Bipartition

We want to split a group of n people (labeled from 1 to n) into two groups of any size. Each person may dislike some other people, and they should not be placed in the same group.
Given the integer n and the array `dislikes`, where `dislikes[i]` $= [a_i, b_i]$ indicates that the person labeled a_i does not like the person labeled b_i, return `true` if it is possible to split everyone into two groups in this way, and `false` otherwise.

- **Input:** `n = 4`, `dislikes = [[1, 2], [1, 3], [2, 4]]`
- **Output:** `true`

3 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]]`

4 * Graph Valid Tree

You have a graph of n nodes labeled from 0 to $n-1$. You are given an integer n and a list of edges, where `edges[i]` $= [a_i, b_i]$ indicates that there is an undirected edge between nodes a_i and b_i in the graph.
Return `true` if the edges of the given graph make up a valid tree, and `false` otherwise.

- **Input:** `n = 5`, `edges = [[0, 1], [0, 2], [0, 3], [1, 4]]`
- **Output:** `true`

5 * Least Congested City

There are n cities numbered from 0 to $n-1$. You are given the array `edges`, where `edges[i]` $= [from_i, to_i, weight_i]$ represents a bidirectional and weighted edge between cities $from_i$ and to_i.
You are also given an integer `distanceThreshold`.
Return the city with the smallest number of cities that are reachable through some path and whose distance is at most `distanceThreshold`. If there are multiple such cities, return the city with the greatest number.
Note: The distance of a path connecting cities i and j is equal to the sum of the edges' weights along that path.

- **Input:** `n = 4`, `edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]`, `distanceThreshold = 4`
- **Output:** `3`

6 Word Search in a Grid

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cells, where "adjacent" cells are those horizontally or vertically neighboring. The cell itself does not count as an adjacent cell.

The same letter cell may be used more than once.

- **Input:**
  - `board` =
    $$ABCE, SFCS, ADEE$$
  - `word` = "ABCCED", "SEE", "ABCB", "ABFSAB", "ABCD"
- **Output:**
  - For `word` = "ABCCED", returns 1
  - For `word` = "SEE", returns 1
  - For `word` = "ABCB", returns 1
  - For `word` = "ABFSAB", returns 1
  - For `word` = "ABCD", returns 0

Note: Here, 1 corresponds to true, and 0 corresponds to false.

7 * Cycle Detection in a Directed Graph

You are given a directed graph with A nodes. The graph is described using a matrix `B` of size $M \times 2$, where each row `B[i]` represents a directed edge from node `B[i][0]` to node `B[i][1]`.

Your task is to determine whether the graph contains a cycle. Return 1 if there is a cycle, and 0 otherwise.

- **Input:**
  - An integer A, representing the number of nodes.
  - A matrix `B` of size $M \times 2$, where each row represents a directed edge.
- **Output:**
  - Return 1 if the graph contains a cycle.
  - Return 0 if the graph does not contain a cycle.

Constraints and Notes:

1. A cycle must involve at least two nodes.
2. The graph does not contain self-loops (edges from a node to itself).
3. There are no multiple edges between any pair of nodes.
4. The graph may or may not be connected.
5. Nodes are numbered from 1 to A.

8 Minimum Rolls in Snakes and Ladders

Rishabh is playing Snakes and Ladders and wonders: "If I could choose the number I roll on the die, what would be the minimum number of rolls required to reach the destination square?"

Rules of the Game:

- The game uses a cubic die with 6 faces numbered from 1 to 6.
- The player starts at square 1 and must reach square 100 by rolling the die. The final roll must land exactly on square 100. If a roll would move the player beyond square 100, no move is made.
- If the player lands at the base of a ladder, they must climb to its top. Ladders always go up.
- If the player lands at the mouth of a snake, they must slide down to its tail. Snakes always go down.

Board Description:

- The board is a 10×10 grid with squares numbered from 1 to 100.
- The board contains N ladders, represented as a matrix `A` of size $N \times 2$, where each ladder is described as:

  $$(A[i][0], A[i][1])$$

  Here, $A[i][0]$ is the base of the ladder, and $A[i][1]$ is its top.
- The board contains M snakes, represented as a matrix `B` of size $M \times 2$, where each snake is described as:

  $$(B[i][0], B[i][1])$$

  Here, $B[i][0]$ is the mouth of the snake, and $B[i][1]$ is its tail.

Your task is to determine the minimum number of rolls required to reach square 100, assuming you can choose any number between 1 and 6 for each roll.

9 Find Course Schedule

There are a total of `numCourses` courses you have to take, labeled from 0 to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] =`

$$a_i, b_i$$

indicates that you must take course b_i first if you want to take course a_i.

For example, the pair $[0, 1]$, indicates that to take course 0 you have to first take course 1. Return the ordering of courses you should take to finish all courses. If there are many valid answers, return any of them. If it is impossible to finish all courses, return an empty array.

Examples

Example 1:

Input: `numCourses = 2`, `prerequisites = [[1,0]]`
Output: `[0,1]`
Explanation: There are a total of 2 courses to take. To take course 1, you should have finished course 0. So the correct course order is `[0,1]`.

Example 2:

Input: `numCourses = 4`, `prerequisites = [[1,0],[2,0],[3,1],[3,2]]`
Output: `[0,2,1,3]`
Explanation: There are a total of 4 courses to take. To take course 3, you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is `[0,1,2,3]`. Another correct ordering is `[0,2,1,3]`.

Example 3:

Input: `numCourses = 1`, `prerequisites = []`
Output: `[0]`

10 Find Optimal Time

You are given a network of n nodes, labeled from 1 to n. You are also given `times`, a list of travel times as directed edges `times[i] =` $(u_i, v_i, w_i), where u_i$ `is the source node,` v_i `is the target node, and` w_i `is the time it takes for a signal to travel from source to target.`

`We will send a signal from a given node` k. `Return the minimum time it takes for all the` n `nodes to receive the signal. If it is impossible for all the` n `nodes to receive the signal, return` -1.

Examples

Example 1:

`Input: times = [[2,1,1],[2,3,1],[3,4,1]],` $n = 4$, $k = 2$
`Output:` 2

Example 2:

`Input: times = [[1,2,1]],` $n = 2$, $k = 1$
`Output:` 1

Example 3:

Input: `times = [[1,2,1]]`, $n = 2$, $k = 2$
Output: -1