Assignment 05

Instructions

1. Write your functions according to the signature provided below in a python(.py) file and submit them to the autograder for evaluation.
2. The autograder has a limited number of attempts. Hence, test it thoroughly before submitting it for evaluation.
3. Save your python file as text(.txt) file and submit it to the canvas.
4. If the assignment has theoretical questions, upload those answers as a pdf file on the canvas.
5. Submission on the canvas is mandatory and the canvas submitted text file should be the same as that of the final submission of the autograder. If not, marks will be deducted.
6. No submission on canvas or no submission on autograder then marks will be deducted.
7. Access the auto grader at https://c200.luddy.indiana.edu.

Question 01

In the mysterious world of Bitland, a dangerous virus has infected its society, turning some residents into imposters. The 'grid' is a map of Bitland, showing where each resident is. The virus can only move up, down, left, or right every minute. In the map, infected people are marked as '2', safe people are marked as '1' and '0' to indicate no person. The government has asked for your help by creating a function called findSafeResidents(grid). Your function needs to carefully look at the map and find the coordinates of residents who are not affected by the virus. The output of your function will be a list of coordinates, showing where those residents are who have avoided getting sick. Your algorithm is like a beacon of hope, showing the way through the darkness of the virus.

Examples

Example 1:

```
Input: grid = [[0,1],[2,0]]
Output:[[0,1]]
```

Explanation: As the virus could not travel througgh the diagonal the resident at (0,1) is safe.

Example 2:

```
Input: [[[0, 2, 0], [0, 0, 0], [0, 0, 1]]]
Output: [[2,2]]
```

Explanation: here the resident at index (2,2) remains unaffected from the infected person at index (0,1).

Constraints

- Solve using BFS traversal (level order traversal)

Function

```
def findSafeResidents(grid)->list[list[int]]:
  return [[0,0]]
```

Question 02

Alice stumbled upon an unusual network of roads connecting N cities. Each road is directed and connects two cities in one direction only. The cities are numbered from 1 to N. Curious about the network, Alice ponders, "What is the minimum number of roads he needs to reverse in order to have at least one route from city 1 to city N?"

Hint: Can you use Dijkstra's single source shortest path algorithm?

Examples

Example 1:

```
Input: N = 4, roads = [[2,1], [3,2], [3,4], [4,1], [2,4]]
Output: 1
```

Explanation: Alice can reverse road [1,2] to find a route from city 1 to city 4.

Example 2:

```
Input: N = 3, roads = [[1,2],[2,3],[3,1]]
Output: 0
```

Explanation: Alice can directly travel from city 1 to city 3 without reversing any roads.

Example 3:

```
Input: N = 5, roads = [[1,2], [2,3], [3,4], [5,4], [5,1], [5,2]]
Output: 1
```

Explanation: Alice can reverse road [5,1] or [5,2] or [5,4] to find a route from city 1 to city 5.

Constraints

- Time complexity should be atmost $O(V + E)$ where V is number of cities and E is number of roads.

Function

```
def reverse_roads(n, roads):
  return 0
```

Question 03

You have information about n different recipes. You are given a string array recipes and a 2D string array ingredients. The ith recipe has the name recipes[i], and you can create it if you have all the needed ingredients from ingredients[i]. Ingredients to a recipe may need to be created from other recipes, i.e., ingredients[i] may contain a string that is in recipes.

You are also given a string array supplies containing all the ingredients that you initially have, and you have an infinite supply of all of them.

Return a list of all the recipes that you can create. You may return the answer in any order.

Note that two recipes may contain each other in their ingredients.

Hint: Topological sort, Khans BFS algorithm

Examples

```
Example 1:
Input: recipes = ["bread"],
ingredients = [["yeast","flour"]],
supplies = ["yeast","flour","corn"]

Output: ["bread"]
```

Explanation: We can create "bread" since we have the ingredients "yeast" and "flour".

Example 2:

```
Input: recipes = ["bread","sandwich"],
ingredients = [["yeast","flour"],["bread","meat"]],
supplies = ["yeast","flour","meat"]

Output: ["bread","sandwich"]
```

Explanation: We can create "bread" since we have the ingredients "yeast" and "flour". We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread".

Example 3:

```
Input: recipes = ["bread","sandwich","burger"],
ingredients = [["yeast","flour"],["bread","meat"],["sandwich","meat","bread"]],
supplies = ["yeast","flour","meat"]

Output: ["bread","sandwich","burger"]
```

Explanation: We can create "bread" since we have the ingredients "yeast" and "flour". We can create "sandwich" since we have the ingredient "meat" and can create the ingredient "bread". We can create "burger" since we have the ingredient "meat" and can create the ingredients "bread" and "sandwich".

Constraints

- Time complexity should be atmost $O(recipes.length+supplies.length+sum(ingredients[i].size))$

Function

```python
def findAllRecipes(recipes, ingredients, supplies):
  # Expected output type
  return 0
```

Question 04

Alexander, the ruler of Bitland, has conquered the kingdom of Byteland. He discovers that the kingdom has an abundance of roads, many of which are redundant. To reduce management expenses, he aims to remove roads that would result in cycles. In other words, if road A connects

to road B, road B connects to road C, and road A connects to road C, he wants to replace these connections with the minimum cost of maintaining two roads such that there are no cycles. To achieve this, he seeks your assistance in completing the function connectInMinCost(connections), which should return the minimum cost required to maintain the roads. Each element in the connection array represents a road connection in the format [source, destination, maintenance cost].

Examples

Example 1:

```
Input: connections = [[0,1,100],[0,2,300],[1,2,370]]
Output: 400
```

Explanation: connect the cities 0 and 1 with cost of 100 and connect cities 0 and 2 with cost of 300. Example 2:

```
Input: connections = [[0,1,100],[1,2,300]]
Output: 400
```

Explanation: We need to connect cities 0 and 1 with 100 and connect cities 1 and 2 with 300.

Constraints

- Assume the graph is not disjoint i.e there always exists a path connecting the city.
- Solve the problem using the concept of minimum spanning tree.

Function

```
def connectInMinCost(connections)->int:
  # Expected output type
  return minCost
```

Question 05

You are the manager of Bloomington Transit, a bus company operating in a city with a total of numRoutes bus routes, labeled from 0 to numRoutes - 1. Your buses traverse these routes to provide transportation services to passengers. However, there are certain dependencies between routes that must be adhered to.You are provided with a list of route dependencies, routeDependencies, where each entry routeDependencies[i] = [ai, bi] indicates that buses on route bi must be taken before buses on route ai.For instance, the pair [1, 0] implies that buses on route 0 must be taken before buses on route 1. Your task is to ascertain whether it's feasible to schedule the bus routes in such a way that all routes can be covered without violating the dependencies. Return True if it's possible to cover all routes while adhering to the dependencies, and False otherwise.

Examples

Example 1:

```
Input: numRoutes = 3 dependencies = [[0, 1], [1, 2], [2, 0]],
Output: False
```

Explanation: Both routes cannot be interdependent.

Example 2:

```
Input: points = [[-1,2],[3,-2],[0,0],[4,3],[-5,7]]
Output: 23
```

Explanation: Connect (0, 0) with (3, -2), (0, 0) with (-1, 2), (-1, 2) with (-5, 7), (3, -2) with (4, 3).

Example 3:

```
Input: points = [[0,1],[1,0],[0,-1],[-1,0]]
Output: 6
```

Function

```
def busRouter(numRoutes,dependencies):
  # Expected output type
  return 0
```

Question 6

When you play the Game of Thrones, you win, or you die. There is no middle ground. There's a group of nobles, each one numbered from 1 to n. Each noble has a certain level of power corresponding to their number. Among these nobles, there are connections called "friendships." If you see [a, b] in the list of friendships, it means noble 'a' and noble 'b' are friends.

A noble is considered vulnerable if two things are true: they have at least one friend, and all their friends are more powerful than them.

You'll need to deal with three types of 'queries':

- [1, a, b] - Add a friendship between nobles 'a' and 'b'.
- [2, a, b] - Remove a friendship between nobles 'a' and 'b'.
- [3] - Figure out what happens when all vulnerable nobles are killed, along with their friendships, and see if any new nobles become vulnerable. Keep doing this until there are no more vulnerable nobles. Then, count how many nobles are left. It's important to note that each time you do this process, it starts with all nobles alive and no vulnerabilities carried over from previous times.

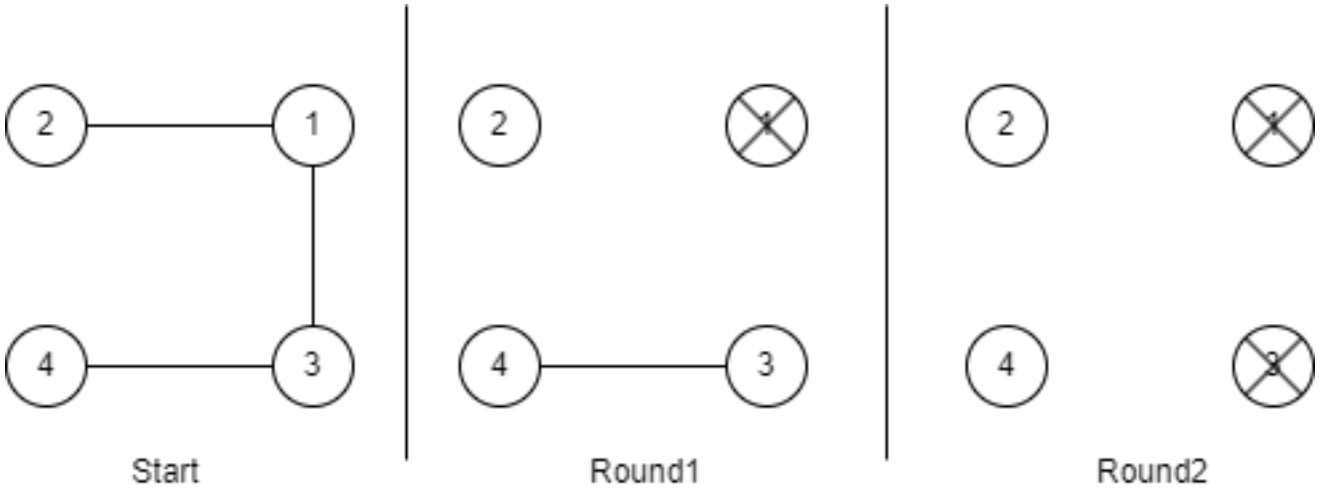
For each query type 3, calculate the number of remaining nobles. Put all these numbers in a list and return the list as the output.

Examples

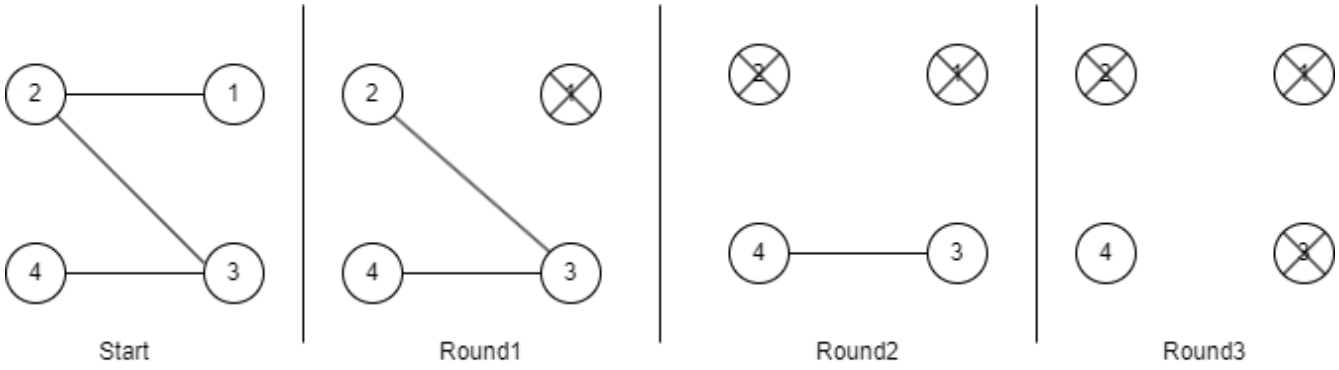
Example 1:

```
Input: n = 4, friendships = [[2, 1], [1, 3], [3, 4]],
queries = [[3], [1, 2, 3], [2, 3, 1], [3], [1, 2, 4], [2, 2, 1], [3]]
Output: [2, 1, 2]
Explanation: See the below diagrams,
- For the first type 3 query, the remaining nobles are 2 and 4 (count 2).
```

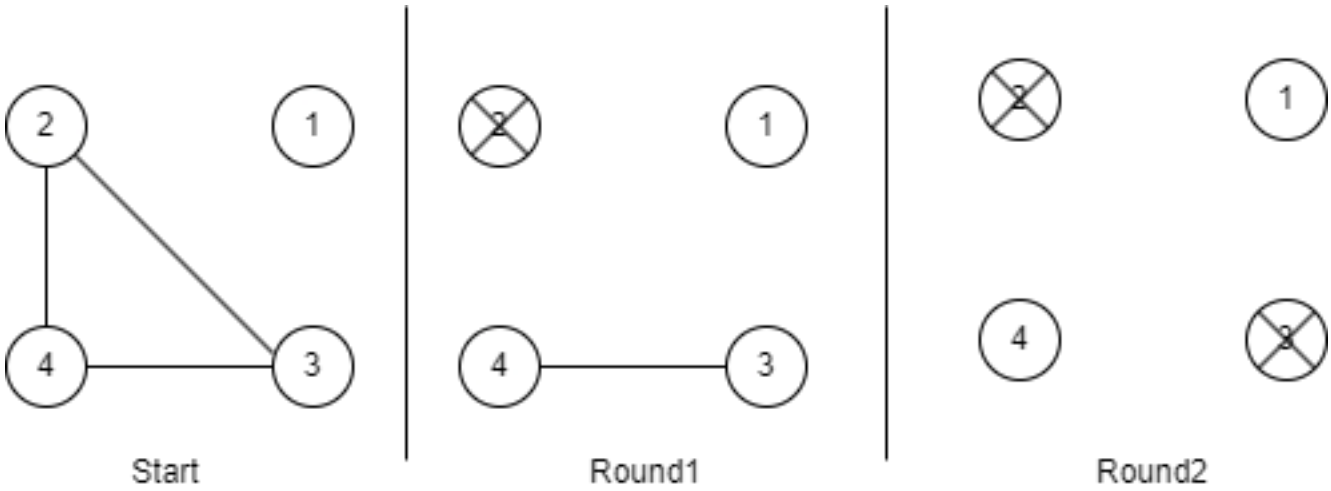
- For the second type 3 query, the remaining nobel is 4 (count 1).
- For the second type 3 query, the remaining nobels are 1 and 4 (count 2).



For First Type 3 Query



For Second Type 3 Query



For Third Type 3 Query

Function

```
def gameOfThrones(n: int, friendships: list[int], queries: list[int]) -> list[int]:
    pass
```

Question 7

You've been handed a network, a complex web of 'n' nodes, each bearing a unique label from 1 to n. Alongside this intricate structure, you've got 'times', an intriguing array detailing travel times in the form of directed edges: times[i] = (ui, vi, wi). Here, ui represents the origin node, vi symbolizes the destination node, and wi denotes the time it takes for a signal to traverse from source to destination.

Now, picture this: we're initiating a signal transmission from a specific node, 'k'. Your mission? Calculate the shortest time required for all n nodes to catch this electrifying signal. But here's the twist: if it's utterly implausible for every single node to grasp this signal, your answer should be -1. So, let the thrilling quest for the optimal signal journey begin!

Examples

Example 1:

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

Explanation: It takes one amount of time to reach nodes 1 and 3. And a total of two amounts of time to reach node 4.

Example 2:

```
Input: times = [[1, 2, 5], [2, 3, 2], [1, 3, 9]], n = 3, k = 2
Output: -1
```

Explanation: From 2, it is impossible to reach node 1.

Constraints

- ui != vi
- 0 <= wi <= 100

Function

```
def optimalSignal(times: list[list[int]], n: int, k: int) -> int:
    pass
```

Question 8

Imagine you're handed an array of points, each point being like a hidden gem with its own set of integer coordinates on a 2D-plane, neatly arranged in the array points[i] = [xi, yi].

Now, here's where the adventure begins: The cost of linking two of these points [xi, yi] and [xj, yj] is calculated using the Manhattan distance between them: |xi - xj| + |yi - yj|, where |val| signifies the absolute value of val.

Your quest? Find the most efficient way to connect all these points while minimizing the cost. And remember, every point must be connected in such a way that there exists precisely one simple path between any pair of points. It's a journey of optimization and connectivity across the mysterious 2D-plane!

Examples

Example 1:

```
Input: points = [[-1,2],[3,-2],[0,0],[4,3],[-5,7]]
Output: 23
```

Explanation: Connect (0, 0) with (3, -2), (0, 0) with (-1, 2), (-1, 2) with (-5, 7), (3, -2) with (4, 3).

Example 2:

```
Input: points = [[0,1],[1,0],[0,-1],[-1,0]]
Output: 6
```

Function

```python
def connectAllPoints(points: list[list[int]]) -> int:
    pass
```