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 1

In the ancient kingdom of Nandoria, nestled among mountains and rivers, there were `v` villages, each labeled uniquely from 0 to v - 1. These villages were linked by a network of stone-paved routes traveled by traders, farmers, and the king's couriers. Each route connected a specific pair of villages, represented by a stone marker at each village gate. These markers, inscribed by the kingdom's scholars, documented the routes as pairs [ai, bi] and ensured no village had a path to itself. Additionally, no two villages shared more than one direct route, making the network both simple and functional. Yet, because the kingdom had grown vast, some routes remained a mystery to even the kingdom's wisest scholars.

One day, the young prince of Nandoria decided he wanted to explore the villages and understand his kingdom better. Setting his sights on reaching a distant village, he pondered whether it was possible to journey from his home village, `home`, to a remote destination, `target`, using only the established routes. However, the prince knew his kingdom's network of villages was complex, and he wasn't certain if the routes would always connect him to his desired destination. To avoid becoming lost in unknown lands, the prince sought the help of Nandoria's scholars, who set out to determine whether a reliable route could connect his home to target.

The scholars diligently began their analysis, poring over every stone marker and every route. They looked at all possible routes, checking each route in the 2D integer array routes to find any connection between the prince's home and his intended destination. The scholars knew that each route could be used in either direction, but only once, and they aimed to find if there was indeed a way to reach the distant village without venturing into uncharted or isolated regions. After examining every possibility, the scholars would deliver an answer to the prince, `true`, if he could reach the target and `false` if there is no possible path.

Constraints

1. 1 <= v <= 10^3
2. 0 <= routes.length <= 2 * 10^3
3. routes[i].length = 2
4. 0 <= ai, bi <= v - 1
5. ai != bi

6. 0 <= home, target <= v - 1
7. There are no duplicate routes.
8. There are no self-loop routes.

Example:

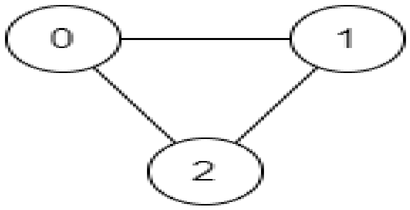

Figure 1: image

Input: `v = 3, routes = [[0, 1], [1, 2], [2, 0]], home = 0, target = 2`
Output: `true`
Explanation: There are two valid paths from village 0 to village 2:
$0 \rightarrow 1 \rightarrow 2$
$0 \rightarrow 2$

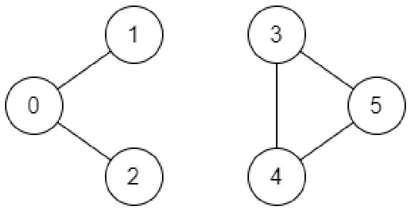

Figure 2: image

Input: `v = 6, routes = [[0, 1], [0, 2], [3, 5], [5, 4], [4, 3]], home = 0, target = 4`
Output: `false`
Explanation: There is no path from village 0 to village 4.

Function

```python
def princeJourney(v, routes, home, target):
    return ans
```

Question 2 - Constrained Network Navigation

You are navigating through a network with n checkpoints and m paths. Each checkpoint is numbered from 1 to n, and each path between checkpoints has a specific difficulty level.

Objective: Your goal is to find the minimum difficulty journey from checkpoint 1 to checkpoint n. However, there's a restriction: as you proceed along your journey, you can only choose paths that are of equal or lower difficulty than the last path you took.

Input:

- You are given the number of checkpoints and the number of paths.

- You are also given a network made up of tuples, each of which has three integers a, b, and d, representing a path between checkpoints a and b with difficulty d.

Output:

- Return minimum difficulty of the journey from checkpoint 1 to checkpoint n if such a journey is possible, or -1 if there is no way to reach checkpoint n from checkpoint 1 under the given restriction.

Test Cases

```
Input: checkpoints = 5, paths = 6, network = [(1, 2, 3), (2, 3, 2), (3, 5, 1), (1, 4, 4),
 (4, 5, 3), (2, 4, 1)]
Output: 6
Explanation: In this case, we are given a graph with 5 nodes and 6 edges,where
each edge has an assigned difficulty. The task is to find the shortest path from
Node 1 to Node 5 under the constraint that each subsequent edge traversed must
have a difficulty no greater than the previous edge. Starting from Node 1,
we explore paths that satisfy this difficulty constraint. One valid path is
1 → 2 → 3 → 5 which traverses edges with difficulties 3, 2, and 1, respectively,
meeting the non-increasing difficulty condition. The total cost of this path
is  3 + 2 + 1 = 6. Another possible path, 1 → 4 → 5, has edges with difficulties
4 and 3, resulting in a higher total cost of 4 + 3 = 7. Since the
1 → 2 → 3 → 5 path offers the minimum cost while satisfying the constraints,
the output is 6.
```

```
Input: checkpoints = 5, paths = 3, network = [(1, 2, 1), (2, 3, 2), (4, 5, 3)]
Output: -1
Explanation: In this case, we have a graph with 5 nodes and 3 edges: (1, 2)
with difficulty 1, (2, 3) with difficulty 2, and (4, 5) with difficulty 3. Our
goal is to find the shortest path from Node 1 to Node 5 under the condition
that each subsequent edge used must have a difficulty no greater than the
previous edge. However, we notice that the graph is disconnected; there is
no path connecting the component with nodes 1, 2, and 3 to the
component with nodes 4 and 5. Since Node 5 is isolated from Node 1, it is
impossible to reach Node 5 from Node 1, making any path between them infeasible.
As a result, the output should be -1, indicating that no valid path exists
from Node 1 to Node 5.
```

Constraints

- 1 <= checkpoints <= 10
- 0 <= paths <= 9
- 1 <= difficulty <= 10^9

Function Signature

```python
def constrainedNetwork(checkpoints: int, paths: int, network: list[tuple(int, int, int)])
    pass
```

Question 3 - Diwali Celebrations

Jack and John are organizing a Diwali tour across cities in Indiana, with Bloomington as their base. Each city has its own tradition for Diwali: some celebrate with grand festivals, while others remain quieter, marking Diwali only in smaller gatherings or not at all. Whether a city has a Diwali celebration depends on the number of Indians residing there—some cities have vibrant celebrations (di = 1), while others have none or very little (di = 0).

Jack and John also know, there are n-1 highways connecting n cities across Indiana, allowing them to travel between any two cities.

Their goal? For each city, they want to find out the maximum possible difference between cities that celebrate Diwali and those that do not in any connected group of cities including that city. Specifically, they want to calculate and return this difference for each city, allowing them to understand where they can experience the highest balance of celebrations versus non-celebrations.

In other words, for each city, they want to calculate the maximum difference between the number of cities with celebrations and non-celebrations within any connected group of cities that includes that city. This difference will help them pinpoint the most festive experience possible across Indiana.

Constraints

2 <= n <= 200000
0 <= di <= 1

Example:

Input 1:
n = 4
highways = [(1,2), (1, 3), (1, 4)]
IsDiwali = [0, 0, 1, 0]
Output: [0, -1, 1, -1]
Explanation:
Given number of cities = 4, and the highways are connecting from city 1 to city 2, city 1 to city 3, city 1 to city 4. Also, Diwali is only celebrated in city 3. With this information, if we draw the network of cities,
For city 1: Network connecting city 1 and city 3 is the best network and this results in in maximum difference between celebrations and non celebrations cities = (1 -1) = 0
For city 2: Any Network of cities results in maximum difference of only -1, whether it can be only city 2, or city 2, 1, 3.
For city 3: Including only itself in the network of connected cities, will result in a maximum difference of (1 - 0) = 1
For city 4: Similar to city 2 = -1

Input 2:
n = 5
highways = [(1, 2), (2, 3), (3, 4), (3, 5)]
IsDiwali = [0, 1, 0, 1, 0]
Output: [0, 1, 1, 1, 0]

Function

```
def MostFestiveExperience(n: int, highways: List[Tuple[int,int]], IsDiwali: List[int]) -> 
    pass
```

Question 4

In the vast, rugged lands of Westeros, alliances among the noble houses are as critical as they are fragile. Castles are bound through marriage, pacts, or oaths of loyalty, but many alliances remain hidden or forgotten. Westeros can be divided into **Dominions**—groups of interconnected castles and houses that share direct or indirect bonds. These Dominions signify regions of power and influence that stand isolated.

As tensions rise and shadows lengthen, Arya Stark is trying to identify these isolated Dominions of power. Equipped with her keen instincts, Arya uncovers the ties that bind each castle in the Seven Kingdoms. If **Castle A** is directly allied with **Castle B**, and **Castle B** shares a tie with **Castle C**, then all three belong to the same Dominion. Arya's task is to reveal the number of these Dominions across Westeros, uncovering the hidden alliances that could shift the balance of power.

You are given a matrix where **allianceMatrix[i][j] = 1** shows that **House i** and **House j** are directly allied, while **allianceMatrix[i][j] = 0** means no direct connection. Help Arya count the distinct Dominions to reveal the true network of alliances and the independent powers lurking within Westeros. ### Constraints

-1 <= n <= 200 -n == dominions.length -n == dominions[i].length -dominions[i][j] is 1 or 0. -dominions[i][i] == 1 -dominions[i][j] == dominions[j][i] **Hint:** Use a recursive approach to explore all possible partitions of the string.

Example 1:

Input: dominion = [[1,1,0],[1,1,0],[0,0,1]]

Output: 2

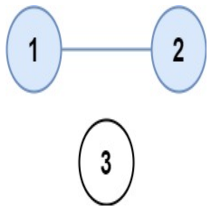

Figure 3: image

Example 2:

Input: dominion = [[1,0,0],[0,1,0],[0,0,1]]

Output: 3

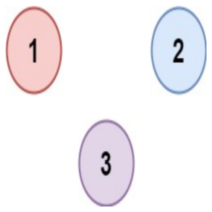

Figure 4: image

Function

```python
def findDominion(dominion: List[List[int]]) -> int:
    #Your code goes here.
```

Question 5 - A Diwali Tale of Rangipur

In the vibrant village of Rangipur, nestled between lush green fields and a sparkling river, the preparations for the grand festival of Diwali were in full swing. Houses were adorned with diyas (oil lamps), and colorful rangoli patterns decorated every doorstep. The entire village buzzed with excitement, but this year, there was an unusual challenge that threatened the celebration.

Rangipur was known for its closely-knit community, where every family shared a special bond with others. However, a playful feud had begun among the villagers. This year, they decided to split into two groups, **Team Rangoli** and **Team Diyas**, for the Diwali celebrations. The idea was to make the festival more competitive and enjoyable. But there was a catch: each family had strong preferences about which families they could or couldn't interact with, and this made the division a challenging puzzle.

The village was represented as a large network of houses, with each house numbered from `0` to `n - 1`. The relationships between the houses were described as an undirected graph, where an edge between house `u` and house `v` meant that the two families were closely associated and must end up in opposite teams. In other words, every edge needed to connect one family from Team Rangoli and one from Team Diyas.

The elders gathered around the banyan tree to discuss the matter and decide if such a partition was even possible. The question loomed over the entire village:

"Can we divide the houses into two teams for the Diwali celebration without any conflicts?"

The elders needed to ensure that the villagers could be split into two groups such that every connection linked a family from Team Rangoli to a family from Team Diyas.

Example 1

```
Input: graph = [[1, 2, 3], [0, 2], [0, 1, 3], [0, 2]]
Output: false
```

Explanation:
- House `0` is connected to houses `1`, `2`, and `3`. - House `1` is connected to houses `0` and `2`, and so on. - When the elders tried to split the village into Team Rangoli and Team Diyas, they found themselves

in a dilemma. There was no way to divide the houses without causing a conflict. Therefore, the answer was **false**.

Example 2

```
Input: graph = [[1, 3], [0, 2], [1, 3], [0, 2]]
Output: true
```

Explanation:
- The elders found a way to split the houses: Team Rangoli could include houses `0` and `2`, while Team Diyas could include houses `1` and `3`. - Every connection between the houses linked families from opposite teams, and everyone was happy to proceed with the celebrations. Hence, the answer was **true**.

Constraints

1. `1 <= n <= 100` (The village has between 1 and 100 houses.)
2. `0 <= graph[u].length < n` (Each house is connected to a few other houses, but not all.)
3. `0 <= graph[u][i] <= n - 1` (The connections are between valid house numbers.)
4. `graph[u]` does not contain `u`. (A house cannot be connected to itself.)
5. All values in `graph[u]` are unique. (No duplicate connections between the same houses.)
6. If `graph[u]` contains `v`, then `graph[v]` contains `u`. (The relationships are mutual.)

Function

```python
def Rangipur_Network(graph: List[List[int]]) -> bool:
    # Your Code Here...
```

Question 6

Erwin and Levi are tasked with setting up a network of scouting outposts across a dangerous, titan-infested area. Each outpost is represented by a pair of coordinates on their scouting map. To stay connected, the outposts need to be linked with signal lines. The cost of laying down a signal line between any two outposts is based on the Manhattan distance between them.

The Manhattan distance between two outposts at [xi, yi] and [xj, yj] is calculated as |xi - xj| + |yi - yj|

Levi is keen on minimizing the cost and the safety of his team while maintaining an efficient network, and Erwin is focused on ensuring that the strategic outposts remain connected.

Their goal is to ensure that all the outposts are connected with the minimum cost, creating a network where there is exactly one simple path between any two outposts (no loops, just direct connections). What is the minimum cost for them to complete the network?

Constraints

- 1 <= points.length <= 1000

- 10000 <= xi, yi <= 10000
- All pairs (xi, yi) are distinct.

Example:

```
Input: points = [[3,12],[-2,5],[-4,1]]
Output: 18
```

```
Input: points = [[0,0],[2,2],[3,10],[5,2],[7,0]]
Output: 20
```

Function

```
def minCostConnectPoints(points: List[List[int]]) -> int:
    return result
```

Question 7 - Cheapest Land Transport

You're working on a travel planner for land transport, where you have multiple routes between cities. Each route has a source city, destination city, travel cost, and the mode of transport (e.g., bus, train, etc.). Given this information, find the **cheapest cost** to travel from a starting city to a destination city with **at most K intermediate stops**.

If there is no valid route to reach the destination within K stops, return -1.

Input: 1. `n` - Total number of cities, labeled from 0 to n-1. 2. `routes` - A list of arrays, where each array represents a route in the format `[src, dest, cost, mode]`. 3. `src` - The starting city. 4. `dst` - The destination city. 5. `K` - Maximum number of stops allowed between `src` and `dst`.

Output: - Return the **minimum travel cost** to reach `dst` from `src` within `K` stops. Return -1 if it's not possible.

Constraints

- 1 <= n <= 100 (Number of cities)
- 0 <= cost <= 10^4 (Travel cost on each route)
- 0 <= src, dest < n
- 0 <= K < n (Maximum number of intermediate stops)

Time Complexity

Aim for a solution with time complexity O(n + E log n), where E is the number of routes.

Test Cases

```
Input:
n = 4
routes = [[0, 1, 100, 'bus'], [1, 2, 100, 'train'], [2, 3, 100, 'train'],
[0, 3, 500, 'bus']]
src = 0
dst = 3
```

```
K = 1

Output:
500

Explanation: Direct route from 0 to 3 costs 500, which is cheaper than going
through
stops with only 1 stop allowed.
```

```
Input:
n = 3
routes = [[0, 1, 300, 'bus'], [1, 2, 300, 'bus'], [0, 2, 700, 'train']]
src = 0
dst = 2
K = 1

Output:
600

Explanation: Route through city 1 costs 600, which is cheaper than the direct route.
```

```
Input:
n = 3
routes = [[0, 1, 100, 'train'], [1, 0, 100, 'bus']]
src = 0
dst = 2
K = 1

Output:
-1

Explanation: There is no path from city 0 to city 2.
```

Function Definition

```python
def findCheapestLandTransport(n, routes, src, dst, K):
    # Your implementation here
```

Question 8 - Network of Influencers

You're working as a data scientist for a cutting-edge tech company developing a recommendation system based on a network of influencers. The platform tracks connections between influencers and their followers. Each influencer can directly or indirectly influence others through these connections, and the company wants to understand how influence spreads.

The platform has a total of n influencers, numbered from 0 to n - 1. The relationships between them are captured in a data structure where each connection between two influencers is represented by a one-way link. This link means one influencer can influence the other. These connections are stored in a 2D list, where each entry [A, B] means that influencer A has a direct influence over influencer B.

Your task is to develop an algorithm that, for each influencer, finds a list of all other influencers who can influence them, either directly or indirectly through a chain of connections. The result for each influencer should be sorted in ascending order by their ID number.

Your goal is to map the entire network of influence, helping the platform better understand the hierarchy and relationships between influencers.

Constraints

- 1 <= n <= 1000
- 0 <= connections.length <= min(2000, n * (n - 1) / 2)
- connections[i].length == 2
- 0 <= A, B <= n - 1
- A != B
- There are no duplicate edges
- The Network is directed and acyclic.

Example 1:

```
Input: n = 8, connections = [[0,3],[0,4],[1,3],[2,4],[2,7],[3,5],[3,6],[3,7],[4,6]]

Output: [[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]]
Explanation:
Influencers 0, 1, and 2 are not inflcuenced by anyone.
- Influencer 3 is influenced by 0 and 1.
- Influencer 4 is influenced by 0 and 2.
- Influencer 5 is influenced by 0, 1, and 3.
- Influencer 6 is influenced by 0, 1, 2, 3, and 4.
- Influencer 7 is influenced by 0, 1, 2, and 3.
```

Example 2:

```
Input: n = 5, connections = [[0,1],[0,2],[0,3],[0,4],[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]
Output: [[],[0],[0,1],[0,1,2],[0,1,2,3]]
Explanation:
Influencer 0 is not influenced by anyone.
- Influencer 1 is influenced by 0.
- Influencer 2 is influenced 0 and 1.
- Influencer 3 is influenced 0, 1, and 2.
- Influencer 4 is influenced 0, 1, 2, and 3.
```

Function

```python
def influencer_network(n:int, connections:list[list[int]]) -> list[list[int]]:
    # Write your code here
    pass
```

Question 9 - The Curious Case of the Parentheses Graph in Gokuldham

One sweltering summer afternoon in Gokuldham Society, Jethalal decided to take a break from his usual complaints about his shop and the age of inflation. He sat down in his favorite armchair, with a cold drink, and thought, "What would be the perfect gift for my friend Taarak Mehta?"

So, he handed Taarak… a balanced string of parentheses, , of length `2n`! But it wasn't just Taarak who was intrigued. Tapu, Jethalal's bright and curious son, couldn't resist the allure of an unsolved puzzle. As he watched Taarak study the string in fascination, Tapu's imagination ran wild. What if these parentheses could be turned into… a graph?

And so, with the help of the Tapu Sena, he decided to transform the sequence of `2n` parentheses into a graph with `2n` vertices. But how? They made a rule: For any two distinct vertices i and j, an edge would be drawn between these vertices only if the parentheses between positions i and j in the string `s` formed a balanced sequence of parentheses on their own.

As Tapu and the Tapu Sena added more and more edges, they were thrilled to see a complex structure emerge. This wasn't just any graph—it was a mystery waiting to be solved! The question was: how many connected clusters, or connected components, would there be in the final graph?

Now, dear students, help Tapu Sena to find connected components in the final graph.

Example:

Example 1:

```
Input: n = 1, s = ()
Output: 1
Explanation: The final graph contains one connected component with one edge between
1 and 2.
```

Example 2:

```
Input: n = 3, s = ()(())
Output: 2
Explanation: Here is what the final graph would look like, which contains two
connected components.
```

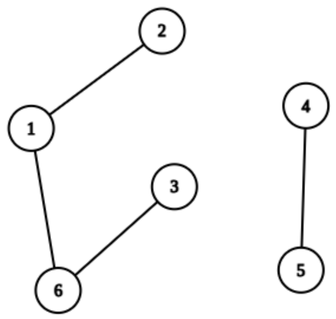

Figure 5: image

Constraints:

- **Solve the question in O(n) time complexity**
- String contains only these two characters: '(', ')'

Function

```
def GraphInGokuldham(n: int, s: str) -> int:
    pass
```

Question 10 - Diwali Potluck and Halloween Walk

A group of international students who came to the U.S. for their master's program wanted to celebrate Diwali together. They decided to organize a potluck at one of their friends' homes, cooking and sharing traditional dishes from India. Since it was also Halloween night in the U.S., they thought it would be exciting to go for a walk around the neighborhood after their meal to see how the locals had decorated their houses.

However, most of the students had assignments due that night, so they agreed they would only go if they could quickly return home after seeing a few decorations. They also didn't want to pass by the same decorations twice, hoping to enjoy a continuous route with new sights along the way. This meant that the path had to take them through some decorated houses and eventually bring them back to their starting point without repeating any streets.

The neighborhood has n houses connected by various streets, represented as pairs [ui, vi] in an array. Each pair denotes a street between two houses, ui and vi, where each house has a unique label from 0 to n-1. Some houses are adorned with elaborate Halloween displays, while others barely show any festive touch.

The students hoped that if such a path existed, it would pass through at least one friend's house, allowing them to start and end their walk there. This way, they could organize the potluck at that friend's house, enjoy the decorations along their walk, and return to complete their assignments.

If no such path existed, they would simply organize the potluck at any convenient place and skip the walk entirely, staying in to finish their work. Can you help them determine the length of the shortest possible path that meets these conditions? If no such path exists, return -1.

Constraints

- 2 <= n <= 1000
- 1 <= streets.length <= 1000
- streets[i].length == 2
- 0 <= ui, vi < n
- ui != vi
- There are no repeated streets.

Example:

Input 1:

```
n = 6
streets = [[0, 1], [1, 2], [2, 3], [3, 0], [3, 4], [4, 5]]
```

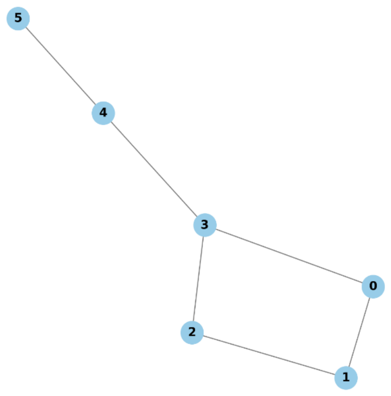

Figure 6: image

```
Output: 4
Explanation: There is a path, that will allow them to return to a point, without
watching the same decorations again. The path with the smallest distance is :
3 -> 0 -> 1 -> 2 -> 3.
```

Input 2:

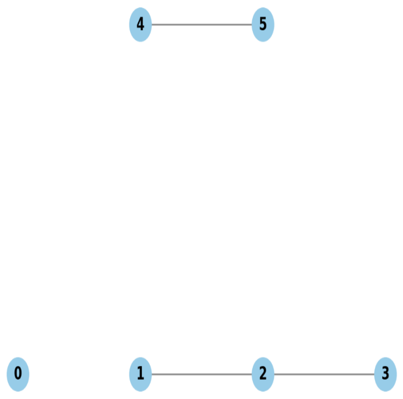

Figure 7: image

```
n = 6
streets = [[1, 2], [2, 3], [4, 5]]
Output: -1
Explanation: There is no path that will help them return home, without watch
the same decoration again.
```

Function

```
def festiveWalkPath(n: int, streets: List[List[int]]) -> int:
  pass
```

Question 11- Luffy's Quest to Unite the Islands

Luffy and the Straw Hat Pirates have discovered a mysterious map that shows `n` islands scattered across the Grand Line. These islands are numbered from `0` to `n - 1`. The map also reveals several sea routes between the islands, depicted as an array of edges. Each route `edges[i] = [a, b]` indicates a direct connection between island `a` and island `b`.

However, the map is incomplete, and not all islands are connected. Luffy wants to know how many distinct groups of islands are fully connected either directly or indirectly. Two islands are in the same group if they are connected through a series of routes, meaning that Luffy can sail between any two islands in the group.

Your task is to help Luffy find the total number of **connected groups** of islands on the map.

Important Restriction: Since navigating the Grand Line is tricky, you cannot use Depth-First Search (DFS) or Breadth-First Search (BFS). You must use the **Union-Find** technique to determine the number of connected island groups.

Constraints

```
* 1 <= n <= 100
* 0 <= edges.length <= n * (n - 1) / 2
```

Example 1:

Input: n = 5, edges = [[0, 1], [1, 2], [3, 4]] Output: 2 Explanation:

Here, we have 5 nodes (numbered from 0 to 4) and 3 edges connecting certain nodes.

Edge [0, 1]: Connects node 0 to node 1. Edge [1, 2]: Connects node 1 to node 2. Edge [3, 4]: Connects node 3 to node 4.

Below graph you can see 2 different set of graphs independent of each other

Example 2:

Input: n=edges=[[0,1], [1,2], [2,3], [4,5]]

Output: 2 Explanation: Here, we have 6 nodes (numbered from 0 to 5) and 4 edges connecting certain nodes.

Edge [0, 1]: Connects node 0 to node 1. Edge [1, 2]: Connects node 1 to node 2. Edge [2, 3]: Connects node 2 to node 3. Edge [4, 5]: Connects node 4 to node 5.

Function

```python
def countComponents(n: int, edges: list[list[int]]) -> int:

    return ans
```