Assignment 07

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 picturesque land of Veridia, there are n cities scattered across the rolling hills and valleys, each with its own unique charm and character. The cities are connected by a network of roads, bridges, and pathways, forming a web of connections that brings the inhabitants closer together.

The study involved examining the array 'edges', which represented the connections between cities. Each element in the 'edges' array, `edges[i] = [from_i, to_i, weight_i]`, described a bidirectional and weighted edge between two cities, `from_i` and `to_i`, with a corresponding distance or weight, `weight_i`. These edges represent the roads or brdiges or pathwasy connecting the cities.

The people of Veridia also set a threshold, `distanceThreshold`, which defined the maximum allowable distance for a city to be considered reachable. They were interested in identifying the city with the smallest number of cities that could be reached through some path while staying within the given distance threshold. If multiple cities met this criterion, the people of Veridia wanted to choose the city with the greatest city number, giving preference to the larger cities as special hub.

Your task was to solve this problem for the people of Veridia. You needed to find that city, which, will be chosen as the special hub.

Examples

Example 1:

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

Output: 3

Explanation:
The neighboring cities at a distanceThreshold = 4 for each city are:
City 0 -> [City 1, City 2]
City 1 -> [City 0, City 2, City 3]
```

```
City 2 -> [City 0, City 1, City 3]
City 3 -> [City 1, City 2]
Cities 0 and 3 have 2 neighboring cities at a distanceThreshold = 4,
but we have to return city 3 since it has the greatest number.
```

Example 2:

```
Input: n = 5,
edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]],
distanceThreshold = 2

Output: 0

Explanation:
The neighboring cities at a distanceThreshold = 2 for each city are:
City 0 -> [City 1]
City 1 -> [City 0, City 4]
City 2 -> [City 3, City 4]
City 3 -> [City 2, City 4]
City 4 -> [City 1, City 2, City 3]
The city 0 has 1 neighboring city at a distanceThreshold = 2.
```

Function

```
def specialHub(n: int, edges: List[List[int]], distanceThreshold: int):
  #return int
  return cityNumber
```

Question 2

In the vibrant realm of Harmonia, there lay n cities, each with its unique charm. The King desired unity, tasking engineers to connect them. The challenge: find the minimum cost to link every city, ensuring a network that reach all the cities. With an array of connections indicating the cost to unite them, the kingdom's future hinged on this mission. The cost of connecting two cities was represented by an array of connections, where connections[i] = [xi, yi, costi], and it indicated the expense of creating a bidirectional link between city xi and city yi. But the solution was elusive; harmony required the most cost-effective path. The engineers embarked on a quest to weave the cities together. In the end, the answer to the King's challenge was a number representing the minimum cost to connect all the cities. However, there was a possibility that it might not be feasible to connect all cities. In such cases, where some cities remained isolated, the answer was -1, signaling the inability to achieve complete connectivity.

Examples

Example 1:

```
Input: n = 3, connections = [[1,2,5],[1,3,6],[2,3,1]]
Output: 6
```

Explanation: Choosing any 2 edges will connect all cities so we choose the minimum 2.

Example 2:

```
Input: n = 4, connections = [[1,2,3],[3,4,4]]
Output: -1
```

Explanation: There is no way to connect all cities even if all edges are used.

Function

```
def challenge(n: int, connections: List[List[int]]):
  #return int
  return minimumCost
```

Question 3

In the peaceful village of Willowville, there are n houses scattered across the lush landscape. The villagers have been facing an increasing need for a reliable water supply. To address this issue, they have decided to undertake a project to ensure that each house receives clean and safe water.

The project involves two main options for providing water to the houses:

1. **Building Wells**: For each house `i`, the villagers have the choice to construct a well within the property directly. The cost of building a well inside each house is given by the array `wells`, where `wells[i - 1]` represents the cost for house `i` to have its own well. Each villager values their well for its convenience and self-sufficiency.
2. **Laying Pipes**: Alternatively, they can lay pipes connecting houses to supply water. The cost of laying pipes between houses is represented by the array `pipes`, where each `pipes[j] = [house1_j, house2_j, cost_j]` specifies the cost of connecting `house1_j` and `house2_j` using a pipe. These connections are bidirectional, and there could be multiple valid connections between the same two houses with different costs.

The village council is seeking your expertise to minimize the total cost of providing water to all houses while ensuring that each house has a reliable water source. What is the minimum total cost to supply water to all houses while considering both well construction and pipe laying?

Examples

Example 1:

```
Input: n = 3, wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]
Output: 3
```

Explanation: The best strategy is to build a well in the first house with cost 1 and connect the other houses to it with cost 2 so the total cost is 3.

Example 2:

```
Input: n = 2, wells = [1,1], pipes = [[1,2,1],[1,2,2]]
Output: 2
```

Explanation: There are multiple possible strategies that result in minimal cost. One of the options is - build a well inside house 1 with cost 1 and build a well inside house2 with cost 1.

Function

```
def min_cost_to_supply_water(n, wells, pipes):
  #write your code here
  return min_cost
```

Question 4

You have a source city represented as `s` and a 2D list called `prices`. In this list, `prices[c1][c2]` signifies the total fare for a plane ticket from city 'c1' to city 'c2'. If `prices[c1][c2] == -1`, it means there is no direct plane route connecting cities 'c1' and 'c2'. As a tourist guide, your task is to generate a list named `min_prices` that contains the minimum fares to reach all accessible cities from the source city `s`. If a city is not reachable, denote the price as `-1`.

Examples

```
Input:
prices = [[0,-1,4,1,-1],[-1,0,-1,-1,-1],[4,-1,0,-1,2],[1,-1,-1,0,3],[-1,-1,2,3,0]],
and s = 0

output: [0,-1,4,1,4]
```

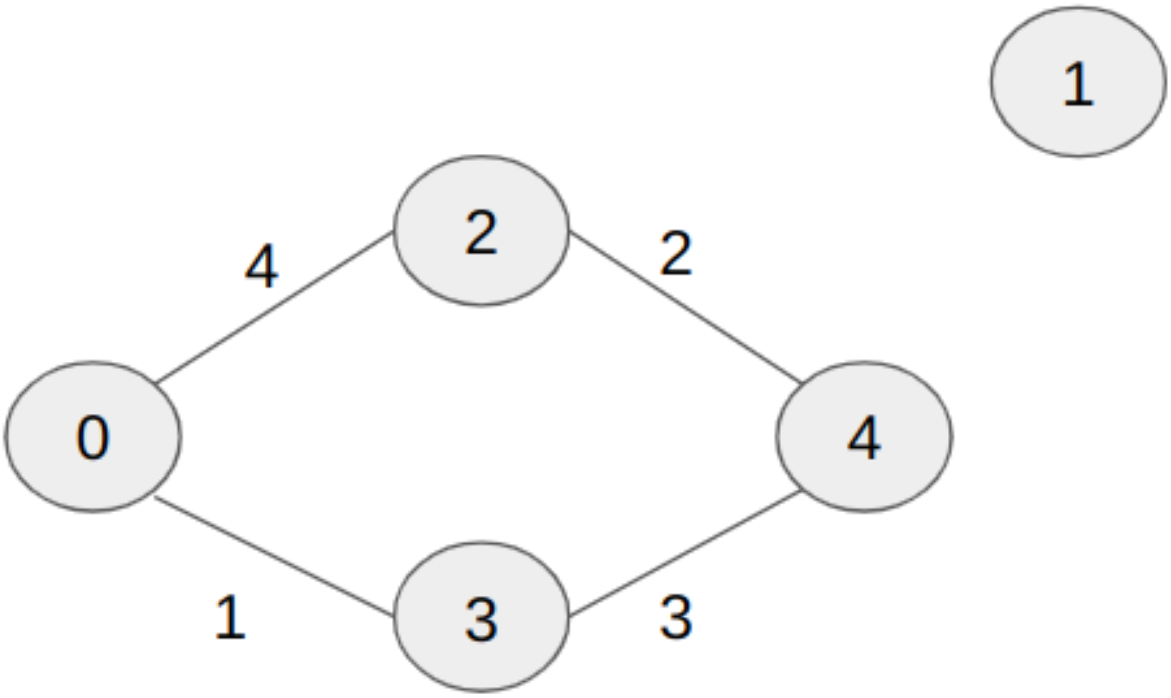

Figure 1: Graph

Function

```
def cheapestRoutes(s, prices):
    #Write your code
    return min_prices
```

Constraints

- -1 <= Prices[c1][c2]

Question 5

In the bustling city of Movietown, the rivalry between the hero and the villain has always been a captivating spectacle for the residents. The hero, a symbol of justice and valor, had a dedicated group of supporters who cheered for him, while the enigmatic and cunning villain had his own loyal followers.

Although the battles between the hero and the villain were legendary, there were a few wise individuals who believed in harmony rather than conflict. These peace-loving residents decided to assign every supporter to one house in the city, ensuring that connected houses do not belong to the same group to prevent group meetings.

The city was abuzz with this unique initiative. As they worked tirelessly to create this ordering of houses, they wondered if it was possible to achieve a harmonious balance. The only question that remained was: could the residents of Movietown be assigned to houses such that no two connected houses belong to the same group? It was a puzzle that demanded a solution. Help them achieve their goal.

Examples

Example 1:

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

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

Function

```
def isDividePossible(n, connected_houses):
  #write your code here
  return boolean_value
```

Question 6

As the lead planner for the city's new rapid delivery service, you're asked to optimize the layout of delivery hubs. The city's grid-like road system dictates that drones can only travel in straight lines along the avenues and streets, incurring costs proportional to the Manhattan distance between hubs. Your challenge is to connect all hubs while minimizing the infrastructure cost, ensuring that any hub can dispatch a drone to any other hub in the network.

Given a list of proposed locations for the drone stations, represented as integer coordinates on a 2D city grid, determine the minimum cost required to connect all the stations into a network. The company wants to minimize the initial investment on the infrastructure while ensuring full connectivity.

Examples

Example 1:

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

Explanation:

We can connect the points in the following way with the Manhattan distance:

-   Connect [0, 0] to [2, 2] with a cost of |0 - 2| + |0 - 2| = 4.
-   Connect [2, 2] to [5, 2] with a cost of |2 - 5| + |2 - 2| = 3.
-   Connect [5, 2] to [7, 0] with a cost of |5 - 7| + |2 - 0| = 4.
-   Connect [2, 2] to [3, 10] with a cost of |2 - 3| + |2 - 10| = 9.
```

The total cost is $4 + 3 + 4 + 9 = 20$. This is the minimum cost to connect all points with one simple path between any two points.

Note that there could be other ways to connect the points, but the total cost will not be less than 20. The solution ensures that every point is connected directly or indirectly to every other point with the minimum possible total cost.

Function

```python
def minCostToConnectHubs(hubs):
  #write your code here
  return min_cost
```

Question 7

As a data routing specialist for a global communications company, your task is to ensure that messages are transmitted across a network of satellites with the highest reliability. Each satellite link has a known probability of successful transmission. Your goal is to find the most reliable route from the control center (start_node) to a distant outpost (end_node) across this network.

Given n satellites (nodes- 0-indexed), a list of direct communication links (edges - edges[i] = [a, b] is an undirected edge connecting the nodes a and b), and their corresponding probabilities of successful transmission (prob), write a function to compute the path with the highest success probability from the control center to the outpost. If no path exists, the function should return 0.

Examples

Example 1:

```
Input: n = 3,
edges = [[0,1],[1,2],[0,2]],
prob = [0.5,0.5,0.2],
start_node = 0, end_node = 2

Output: 0.25
```

Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.

Example 2:

```
Input: n = 3,
edges = [[0,1]],
prob = [0.5], start_node = 0, end_node = 2

Output: 0.0
```

Explanation: There is no path between 0 and 2.

Function

```
def findMaxSuccessPath (n, edges, prob, start_node, end_node):
  #write your code here
  return max_succ_path
```

Question 8

Once upon a time, in a realm governed by the sands of time, there existed a grand "grid", a labyrinth of temporal locks, spread out in an m x n matrix. Each cell within this mystical grid was bound by a time spell, signifying the earliest moment one could step into its bounds. The matrix was laid out such that "grid[row][col]" held the secret of the temporal lock, the minimum time required before the cell (row, col) would welcome a visitor. This grid was not merely a pattern of numbers but a challenge set by the ancients, a puzzle stretching across dimensions of time and space.

In the upper left corner stood our intrepid traveler, poised at the grid's entrance at the very stroke of the 0th second, eyes alight with determination. The journey ahead was fraught with the essence of time; the traveler could only move to an adjacent cell each second, venturing up, down, left, or right, but never diagonally, with each step synced to the relentless ticking of the cosmic clock. The ultimate goal lay diagonally opposite, the bottom-right cell of the matrix, a destination that seemed to whisper promises of victory if reached in time. The challenge was clear: navigate this temporal maze with the swiftness of thought, overcoming the time-locked cells to reach the end. Yet, the question hung in the air, perfumed with the scent of mystery — what is the minimum time needed to conquer this grid, or was it an impossible task, destined to end in the solemnity of a "-1," a journey never to be completed?

Examples

Example 1:

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

Explanation:

One of the paths that we can take is the following:
- at t = 0, we are on the cell (0,0).
- at t = 1, we move to the cell (0,1). It is possible because grid[0][1] <= 1.
- at t = 2, we move to the cell (1,1). It is possible because grid[1][1] <= 2.
- at t = 3, we move to the cell (1,2). It is possible because grid[1][2] <= 3.
- at t = 4, we move to the cell (1,1). It is possible because grid[1][1] <= 4.
- at t = 5, we move to the cell (1,2). It is possible because grid[1][2] <= 5.
- at t = 6, we move to the cell (1,3). It is possible because grid[1][3] <= 6.
- at t = 7, we move to the cell (2,3). It is possible because grid[2][3] <= 7.
The final time is 7. It can be shown that it is the minimum time possible.
```

Example 2:

```
Input: grid = [[0,2,4],[3,2,1],[1,0,4]]
Output: -1

Explanation:
- at t = 0, we are on the cell (0,0).
- at t=1, we cannot move to (0,1) or (1,0) because the required time
to reach both the cells is greater than 1( t value)
```

Function

```python
def minimumTimeToVisit(grid):
  #write your code here
  return min_time
```

Question 9

In the realm of the interconnected isles of Graphonia, each island is known by its own unique number, ranging from 0 to n-1 and is linked to others by magical bridges. The power of these bridges, however, comes with a peculiarity: they can only bear a certain amount of weight, beyond which they become impassable. An ancient scroll, the edgeList, contains records of all the bridges [ui, vi, wt], where "ui" and "vi" represent the islands connected by the bridge, and "wt" is the maximum weight it can hold.

The islanders have now come up with a mystical contraption known as the "WeightLimitedPathsExist". Once invoked with the number of islands and the edgeList,and the list of journeys, it should determine if a traveler, carrying a weight "w", can journey from one island "p" to another "q" without breaking any of the bridges. It should return a list of solution 'True' or 'False' for the given list of journeys.

Examples

Example 1:

```
Input: n=6,
bridges=[[0, 2, 4], [0, 3, 2], [1, 2, 3], [2, 3, 1], [4, 5, 5]],
querylist=[[2, 3, 1], [1, 3, 3], [2, 0, 3], [0, 5, 6]]
Output: [True, False, True, False]

Explanation: Graph of given bridges is as follows:
```

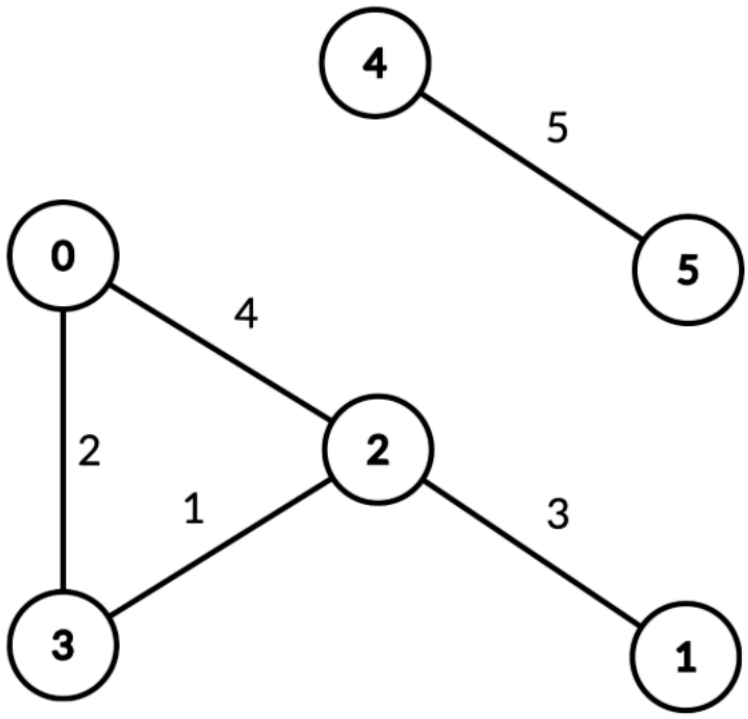

Figure 2: Graph

```
query[2, 3, 1]; return True
  given p=2, q=3, and w=1 and there is a path from 2 to 3
  that can hold weight of 1 unit so the query returns True.
query[1, 3, 3]; return False
  since There is no path to go from 1 to 3 with weight greater than 2.
query[2, 0, 3]; return True.
  There is an edge from 2 to 0 that can hold up to weight 4.
  Since the weight in the query is 3 it returns True.
query[0, 5, 6]; return False.
  There are no paths from 0 to 5.
```

Function

```
def WeightLimitedPathsExist(n,edgelist,querylist):
  #write your code here
  return list
```

Question 10

Bunty is fascinated by the shortest path algorithm and starts applying it everywhere he travels. Once he visits the magical city of London and wishes to travel from the start position to the end position.

The bus fare associated with going from position x to position y is the Manhattan distance.

However, the magical city of London has some magical bridges in between, given in the array 'bridges'. Each bridge in the 'bridges' array can be used multiple times to travel from the start position to the end position, paying the given fare.

Being an average solver, Bunty isn't able to figure out the solution. Help him reach his target spending the least amount of money.

Constraints

Solve it using shortest path algorithm

Examples

Example 1:

```
Input: start = [2,2], end = [4,4], bridges = [[1,1,2,2,2],[2,2,4,3,1]]

Output: 2
```

Explanation: Although, the bus fare without using the magical bridges is 4, Bunty can use the magical bridge from (2,2) until (4,3) and take the bus from (4,3) to (4,4), resulting in the minimum money required to reach the end position as 2.

```
Input: start = [5,5], end = [10,10], bridges = [[3,3,4,4,5],[5,5,7,5,2],[7,5,7,10,6]]
Output: 10
```

Explanation: Without using the magical bridges, Bunty can reach the end position with a minimum money of 10.

Function

```
def shortestFareRoute(start, target, specialRoads):
  #write your code here
  return min_fare
```