Problem Understanding

In the "Maximal Network Rank" problem, we are given an integer n representing the number of cities and a list of roads connecting these cities. Each road connects two distinct cities and is bidirectional. The task is to compute the maximal network rank of the infrastructure, which is defined as the maximum network rank of all pairs of different cities. The network rank of two different cities is the total number of directly connected roads to either city.

For instance, given n = 4 and roads = [[0,1],[0,3],[1,2],[1,3]], the output should be 4. This is because cities 0 and 1 have a combined network rank of 4.

Approach 1/2: Iterative Pairwise Comparison

- **Strategy**: This approach calculates the degree of each city and then iterates over every possible pair of cities to compute their combined network rank.
- **Key Data Structures**:
    - Degree List: To store the number of roads connected to each city.
    - Road Set: For quick lookup of roads.
- **Major Steps**:
    - Calculate the degree of each city.
    - Iterate over all possible pairs of cities.
    - For each pair, compute the network rank and update the max rank.
- **Complexity**:
    - **Time**: $O(n2)O(n^2)O(n2)$ due to iterating over all pairs of cities.
    - **Space**: $O(n+len(roads))O(n + \text{len(roads)})O(n+len(roads))$. The degree list occupies $O(n)O(n)O(n)$ and the road set occupies $O(len(roads))O(\text{len(roads)})O(len(roads))$.

Approach 2/2: Degree Counting with Direct Connection Check

- **Strategy**: Instead of exhaustive pairwise comparison, this method identifies cities with the highest and second-highest degrees, and then restricts checks to a smaller subset of cities.
- **Key Data Structures**:
    - Degree List: To store the number of roads connected to each city.
    - Road Set: For quick $O(1)$ lookup of roads.
- **Major Steps**:
    - Calculate the degree of each city.
    - Identify the highest and second-highest degree cities.
    - Check direct connections among these cities to compute the maximal rank.
- **Complexity**:
    - **Time**: $O(n+\text{len(roads)})$ derived from counting degrees and checking direct connections among a few cities.
    - **Space**: $O(n+\text{len(roads)})$. The degree list occupies $O(n)$ and the road set occupies $O(\text{len(roads)})$.

Differences:

- **Strategy**: The first approach involves a brute-force method of iterating through all pairs of cities. In contrast, the second approach is more optimized, focusing on cities with the highest and second-highest degrees.
- **Efficiency**: The second approach is generally more efficient as it avoids unnecessary computations by restricting the checks to a subset of city pairs.

- **Complexity**: The time complexity of the first approach is quadratic ($O(n2)O(n^2)O(n2)$), while the second approach has a linear time complexity ($O(n+len(roads))O(n + \text{len(roads)})O(n+len(roads))$), making the latter more scalable for larger inputs.

By understanding the differences between these two approaches, one can appreciate the importance of optimizing algorithms and the benefits it brings in terms of efficiency and scalability.

Approach 1/2: Iterative Pairwise Comparison

To solve the "Maximal Network Rank" problem, we first compute the degree of each city (number of roads connected to it). Then, we iterate over every possible pair of cities to compute their combined network rank.

Key Data Structures:

- **Degree List**: A list that stores the number of roads connected to each city.
- **Road Set**: A set that stores the roads for quick look-up.

Enhanced Breakdown:

1. **Initialization**:
    - Initialize a list degree of size n with zeros.
    - Initialize an empty set road_set.
2. **Processing Each Road**:
    - For each road connecting cities a and b :
        - Increment the degree of both a and b.
        - Add the road to road_set.
3. **Iterate Over All City Pairs**:
    - For each pair of distinct cities i and j:
        - Compute their network rank as the sum of their degrees.
        - If they are directly connected, subtract 1 from the rank.
        - Update the max_rank with the current rank if it's higher.

4. **Wrap-up**:
    - Return the computed max_rank.

Example:

Based on the breakdown, here's how the algorithm processes the given example:

1. **Input**:
    - Number of cities n = 4
    - Roads: [[0,1], [0,3], [1,2], [1,3]]
2. **Degree Calculation**:
   After processing all the roads, the degree of connection for each city is as follows:
    - City 0 is connected to 2 other cities.
    - City 1 is connected to 3 other cities.
    - City 2 is connected to 1 other city.
    - City 3 is connected to 2 other cities.

Hence, the degree list is: [2, 3, 1, 2].

3. **Network Rank Calculation for Each Pair**:
   For each unique pair of cities, we calculate their network rank:
    - Pair (0, 1): Network Rank = 4
    - Pair (0, 2): Network Rank = 3
    - Pair (0, 3): Network Rank = 3
    - Pair (1, 2): Network Rank = 3
    - Pair (1, 3): Network Rank = 4
    - Pair (2, 3): Network Rank = 3
4. **Result**:
   The maximal network rank among all pairs of cities is 4.

Summary:

Given the input n = 4 and roads [[0,1], [0,3], [1,2], [1,3]]:

After processing all roads, the degree list is [2, 3, 1, 2]. Iterating over city pairs, the pairs (0, 1) and (1, 3) both give a network rank of 4, which is the maximum. Thus, the output is 4.

Complexity 1/2:

Time Complexity:

- The solution has a time complexity of O(n^2). This is because we iterate over all pairs of cities, resulting in O(n^2) complexity.

Space Complexity:

- The space complexity is O(n+len(roads)). The degree list takes O(*n*) space, and the road_set takes O(len(roads)) space.