Approach

Overview

We are given an undirected weighted graph, represented by an array edges, where edges[i] = [u, v, w] indicates an edge between vertices u and v with weight w. Additionally, we are given an array queries, where queries[i] = [s, t] represents a pair of nodes in the graph.

For each query, our task is to determine the minimum *cost* of a *walk* that starts at node s and ends at node t. If no such walk exists, the answer is -1. Let's first define the two key terms involved in this task:

- A *walk* in a graph is a sequence of connected vertices and the edges that connect them. Unlike a path, a walk allows both edges and vertices to be repeated.
- The *cost* of a walk is defined as the bitwise AND of the weights of all edges encountered in the walk.

First, recall that the bitwise AND operation compares the bits of all the numbers involved and keeps a bit as 1 only if it is 1 in every number; otherwise, the bit becomes 0. Now, consider the smallest number in the group. It already has some bits set to 0. Since the AND operation can only turn bits off (changing 1 to 0, but never 0 to 1), the result can never have more 1s than the smallest number. This means the result is always less than or equal to the smallest number.

In this problem, that tells us that adding more edges to a walk can only keep the cost the same or make it smaller. So, to find the minimum cost, we should try to include as many edges as possible in the walk.

Notice that since w AND w = w, revisiting the same edge multiple times does not change the total cost. This can be useful if we need to backtrack to take a different path, in order to visit more edges.

Approach 1: Disjoint-Set (Union-Find)

Intuition

First, let's determine when the answer to a query is -1. This happens when no walk exists between the two nodes, meaning they belong to different connected components.

A connected component in an undirected graph is a group of nodes where there is a path between any pair of nodes.

Now, suppose the two nodes belong to the same connected component. What is the minimum cost of a walk connecting them? As mentioned, the optimal walk includes as many edges as possible. Since revisiting an edge does not affect the total score, we can freely traverse the edges of the component, meaning that we can move back and forth to reach all of them. Therefore, the best way to achieve the lowest cost is to visit every edge in the component.

To efficiently find and process the connected components of the graph, we use the Disjoint Set (Union-Find) data structure. This approach relies on two main operations: Union and Find. Each connected component has a representative node, known as its root, which is returned by the Find operation for any node in the group. When we Union two nodes, we merge their entire groups, as now a path exists between every node in one group and every node in the other. To maintain efficiency, the root of the larger group is chosen as the representative of the merged group. This minimizes the time needed for future Find operations by reducing the number of steps required to reach the current representative.

Disjoint Set (Union-Find): For a more comprehensive understanding of the Disjoint Set data structure, check out the Disjoint Set/Union-Find Explore Card. This resource provides an in-depth look at Union-Find, explaining its key concepts and applications with a variety of problems to solidify understanding of the pattern.

Once the nodes are grouped into connected components, we calculate the total cost for each component as the bitwise AND of all its edge weights. In the end, the minimum cost of a walk between any two nodes in the same component will be the same and equal to the component's total cost.