Problem Pattern

1. We have to connect all the buildings by the path which will cost us the least.
2. If we treat each building as a node and all the path as edges then we get a graph.
3. **Cost of each path can be calculated using the length of the path and cost to construct one unit(given by p)**
4. Now we just need to find the path in the graph which connects all the cities and is of minimum cost.
5. The way to do that is to find out the Minimum Spanning Tree(MST) of the graph which can be found using prim's algorithm or Kruskal algorithm.

Problem Approach

1. Create a graph using the given edge list. We will use prim's algorithm to find the minimum spanning tree.
2. Initialize the cost of all vertices as infinite.
3. Create an empty priority queue PQ. Every item of PQ is a pair (cost, vertex). Make sure to compare based on cost.
4. Initialize all vertices as not part of MST yet. We use a boolean array `visited` for this purpose. This array is required to make sure that an already considered vertex is not included in PQ again. This is where Prim's implementation differs from Dijkstra. In Dijkstra's algorithm, we didn't need this array as distances always increase. We require this array here because the cost of a processed vertex may decrease if not checked.
5. Insert source vertex into PQ and make its cost as 0.
6. Keep a variable `totalCost` = 0
7. While either PQ doesn't become empty
    1. Extract minimum key vertex from PQ. Let the extracted vertex be u.
    2. Include u in MST using visited[u] = true.
    3. totalCost += cost[u]
    4. Loop through all adjacent of u and do the following for every vertex v. // If the cost of edge (u,v) is smaller than the cost of v and v is not already in MST

If visited[v] = false && cost[v] > weight(u, v)
Update key of v, i.e., do cost[v] = weight(u, v)
Insert v into the PQ

8. Return `totalCost`