Road Reparation

There are n cities and m roads between them.

Unfortunately, the condition of the roads is so poor that they cannot be used. Your task is to repair some of the roads so that there will be a decent route between any two cities.

For each road, you know its reparation cost, and you should find a solution where the total cost is as small as possible.

Input
The first input line has two integers n and m: the number of cities and roads.
The cities are numbered 1,2,...,n.

Then, there are m lines describing the roads.
Each line has three integers a, b and c: there is a road between cities a and b, and its reparation cost is c.
All roads are two-way roads.

Every road is between two different cities, and there is at most one road between two cities.

Output
Print one integer: the minimum total reparation cost. However, if there are no solutions, print "IMPOSSIBLE".

Constraints
1 <= n <= 10^5
1 <= m <= 2.10^5
1 <= a,b <= n
1 <= c <= 10^9

Example
Input:
5 6
1 2 3
2 3 5
2 4 2
3 4 8
5 1 7
5 4 4

Output:
14

Problem Breakdown

Lets understand the problem first

You are given:

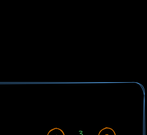
- n cities (nodes)
- m bidirectional roads (edges), each with a repair cost

Your task:

Reconnect all cities using some of the roads such that:

- Every city is reachable from every other
- Total cost is minimized

If it's not possible to connect all cities → print "IMPOSSIBLE"

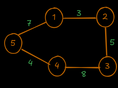


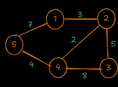
Spanning Tree: What Is It?

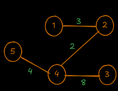
A spanning tree of a graph is a subgraph that:

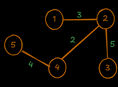
- Includes all nodes
- Is connected
- Has no cycles

For n nodes, a spanning tree has exactly n - 1 edges.









What Is a Minimum Spanning Tree (MST)?

A Minimum Spanning Tree is a spanning tree with the least total edge cost.

How Do We Build an MST?

There are two classic algorithms:

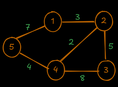
Prim's Algorithm

- Start from any node
- Greedily pick the cheapest edge connecting to an unvisited node
- Repeat until all nodes are included

Kruskal's Algorithm

- Sort all edges by weight
- Use a Disjoint Set (Union Find) to avoid cycles
- Greedily pick smallest edges that don't create a cycle

Understanding Prim's Algorithm



Algorithm

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n,m;
    cin >> n >> m;

    vector<vector<pair<int, int>>> graph(n+1);
    vector<int> visited(n+1, false);

    for(int i=0; i<m; ++i) {
        int u, v, wt;
        cin >> u >> v >> wt;
        graph[u].push_back({v, wt});
        graph[v].push_back({u, wt});
    }
```

Reading input and constructing an adjacency list graph

```
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    pq.push({0, 1});
```

Priority queue because we need to get the next smallest edges everytime we are greedy for that

```
    long long res = 0;

    while(!pq.empty()) {
        auto [cost, node] = pq.top();
        pq.pop();

        if(visited[node]) continue;
        visited[node] = true;

        res += cost;

        for(auto [nbr, wt] : graph[node]) {
            if(!visited[nbr]) {
                pq.push({wt, nbr});
            }
        }
    }
```

Start Prim's from node 1 with cost 0.

Take the smallest available edge

Mark node as visited

Add cost to the result

Add all neighboring edges to the queue if neighbor is unvisited

```
    for(int i=1; i<=n; ++i) {
        if(!visited[i]) {
            cout << "IMPOSSIBLE" << endl;
            return 0;
        }
    }
```

After Prim's finishes, if any node is unvisited → graph is disconnected

```
    cout << res << endl;
    return 0;
}
```

Time Complexity

Prim's (with heap) O((n + m) log n)

Space Complexity

O(n + m)