Building

Problem Statement

You have just opened a dish cable business and you want to connect your dish lines in your area. In your area there are **N** buldings and **E** roads. The roads are two way obviously. In each road there is a cost of connecting the cables. You want to connect all buldings in such a way that there is connection from any building to another, not necessary to be directly.

As you are a businessman, you want the total cost to be minimum. Can you tell the minimum total cost to do the work?

Note: There can be multiple road from one building to another. If it is not possible to connect all the building, print **-1**.

Input Format

- First line will contain **N** and **E**.
- Next E lines will contain **A**, **B** and **W** which means there is a connection in between A and B where the cost for connecting the cable is W.

Constraints

1. 1 <= **N, E** <= 10^5
2. 1 <= **A, B** <= N
3. 1 <= **W** <= 10^9

Output Format

- Output the minimum cost.

Sample Input 0

```
5 7
1 2 10
1 3 5
3 2 4
2 4 1
3 4 2
4 5 3
1 5 2
```

Sample Output 0

```
8
```

Sample Input 1

```
3 2
1 2 10
2 1 2
```

Sample Output 1

```
-1
```

Submissions: 99
Max Score: 25
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

C++20

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

#define pii pair<int, pair<int, int>>
int n, m;
vector<pii> ans;

// DSU_UNION_BY_SIZE
const int N = 1e5 + 5;
int parent[N];
int parentSize[N];

void dsu_set(int n)
{
    for (int i = 1; i <= n; i++)
    {
        parent[i] = i;
        parentSize[i] = 1;
    }
}

int dsu_find(int node)
{
    while (parent[node] != node)
    {
        node = parent[node];
    }
    return node;
}

// This is union by size
void dsu_union(int a, int b)
{
    int leaderA = dsu_find(a);
    int leaderB = dsu_find(b);
    if (leaderA != leaderB)
    {
        if (parentSize[leaderA] > parentSize[leaderB])
        {
            // Leader is 'A'
            parent[leaderB] = leaderA;
            parentSize[leaderA] += parentSize[leaderB];
        }

        else
        {
            // Leader is 'B'
            parent[leaderA] = leaderB;
            parentSize[leaderB] += parentSize[leaderA];
        }
    }
}

int main()
{
    cin >> n >> m;

    priority_queue<pii, vector<pii>, greater<pii>> pq;

    dsu_set(n);

    for (int i = 1; i <= m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        pq.push({w, {u, v}});
    }

    // Kruskal's Algorithm
    while (!pq.empty())
    {
        auto el = pq.top();
        pq.pop();

        int w = el.first;
        int u = el.second.first;
        int v = el.second.second;

        int leaderU = dsu_find(u);
        int leaderV = dsu_find(v);
        if (leaderU == leaderV)
            continue;
        ans.push_back({w, {u, v}});

        dsu_union(u, v);
    }

    long long int cost = 0;
    for (auto e : ans)
    {
        // cout << e.second.first << " " << e.second.second << " " << e.first << endl;
        cost+= (long long int)(e.first);
    }
    if(ans.size()==n-1)
    {
        cout<<cost<<endl;
    }
    else
    cout<<"-1"<<endl;



    return 0;
}
```

Line: 1 Col: 1

Upload Code as File   Test against custom input   Run Code   Submit Code