CÓDIGO FONTE

EDITAR & ENVIAR

VISUALIZE O CÓDIGO FONTE DE SUAS SUBMISSÕES, JUNTO COM ALGUNS DETALHES EXTRAS.

SUBMISSÃO # 42925939

PROBLEMA:	1774 - Roteadores
RESPOSTA:	**Accepted**
LINGUAGEM:	Python 3.9 (Python 3.9.4) [+1s]
TEMPO:	0.052s
TAMANHO:	2,52 KB
MEMÓRIA:	-
SUBMISSÃO:	22/12/2024 12:13:25

CÓDIGO FONTE

```python
# -*- coding: utf-8 -*-

# Estrutura de dados para representar o grafo
class Grafo:
    def __init__(self, vertices):
        self.vertices = vertices  # Número de vértices
        self.grafo = []  # Lista de arestas (tupla: (peso, vertice1, vertice2))

    def adicionar_aresta(self, u, v, peso):
        self.grafo.append((peso, u, v))


# Algoritmo de Kruskal
class UnionFind:
    def __init__(self, n):
        self.pai = list(range(n))  # Representação dos pais de cada vértice
        self.rank = [0] * n  # Rank para otimizar a união

    def find(self, u):
        if self.pai[u] != u:
            self.pai[u] = self.find(self.pai[u])  # Caminho comprimido
        return self.pai[u]

    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)

        if root_u != root_v:
            # União por rank (menor árvore se torna subárvore da maior)
            if self.rank[root_u] > self.rank[root_v]:
                self.pai[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.pai[root_u] = root_v
            else:
                self.pai[root_v] = root_u
                self.rank[root_u] += 1
            return True
        return False


def kruskal(grafo):
    # Ordena as arestas por peso
    grafo.grafo.sort()

    mst = []  # Armazenará a árvore geradora mínima
    uf = UnionFind(grafo.vertices)

    for peso, u, v in grafo.grafo:
        if uf.union(u, v):
            mst.append((u, v, peso))

    return mst


def run_challenge():
    input_data_lst = list(input().split())
    if len(input_data_lst) != 2:
        return

    amount_routers, amount_cables = int(input_data_lst[0]), int(input_data_lst[1])
    if amount_routers < 3 or amount_routers > 60:
        return

    if amount_cables < amount_routers or amount_cables > 200:
        return

```

```
        data_cable_lst = list(map(int, input().split()))
        begin_id_router, end_id_router, price_cable = data_cable_lst[0], data_cable_lst[1], data_c
        router_graph.adicionar_aresta(begin_id_router-1, end_id_router-1, price_cable)

    mst = kruskal(router_graph)

    smallest_weight = 0
    for begin_id_router, end_id_router, price_cable in mst:
        smallest_weight += price_cable

    print(str(smallest_weight))
    # print(path_between_first_to_final_router)


if __name__ == '__main__':
    run_challenge()
```