Exercícios-Problemas (EP)

Semana 5:

https://www.facom.ufu.br/~albertini/grafos/

EP4: UVa 00908 - Re-Connecting Computer Sites

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The input is organized as follows:

- A line containing the number N of computer sites, with 1 ≤ N ≤ 1000000, and where each computer site is referred by a number i, 1 ≤ i ≤ N .
- The set T of previously chosen high-speed lines, consisting of N − 1 lines, each describing a high-speed line, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.
- A line containing the number K of new additional lines, 1 ≤ K ≤ 10.
- K lines, each describing a new high-speed line, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.
- A line containing the number M of originally available high-speed lines, with N − 1 ≤ M ≤ N (N − 1)/2.
- M lines, each describing one of the originally available high-speed lines, and containing the numbers of the two computer sites the line connects and the monthly cost of using this line. All costs are integers.

EP4: UVa 00908 - Re-Connecting Computer Sites

Output

- For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.
- The output file must have one line containing the original cost of connecting the N computer sites with M high-speed lines and another line containing the new cost of connecting the N computer sites with M + K high-speed lines. If the new cost equals the original cost, the same value is written twice.

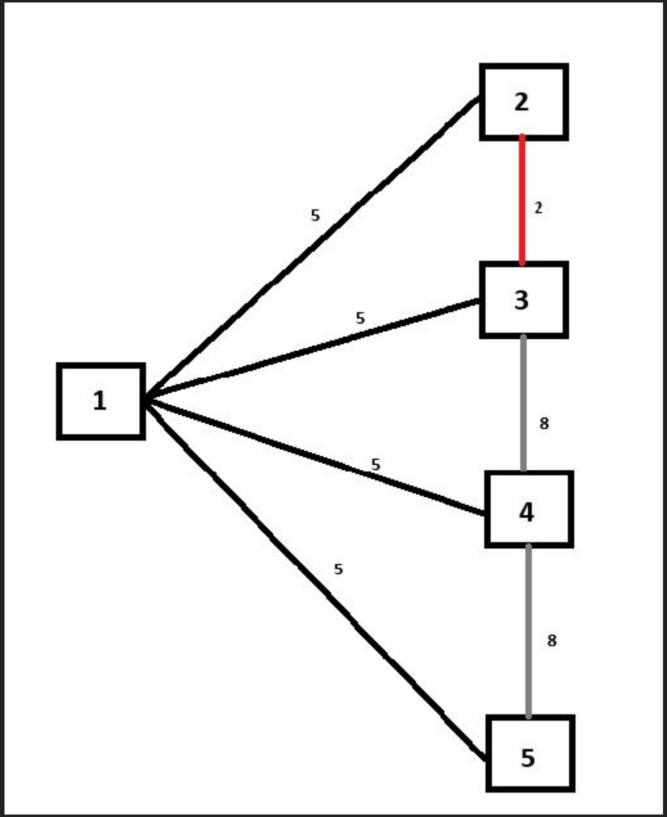
Sample Input

```
5
1 2 5
1 3 5
1 4 5
1 5 5
1
2 3 2
6
1 2 5
1 3 5
1 4 5
1 5 5
3 4 8
4 5 8
```

Sample Output

```
20
17
```

EP4: UVa 00908 - Re-Connecting Computer Sites



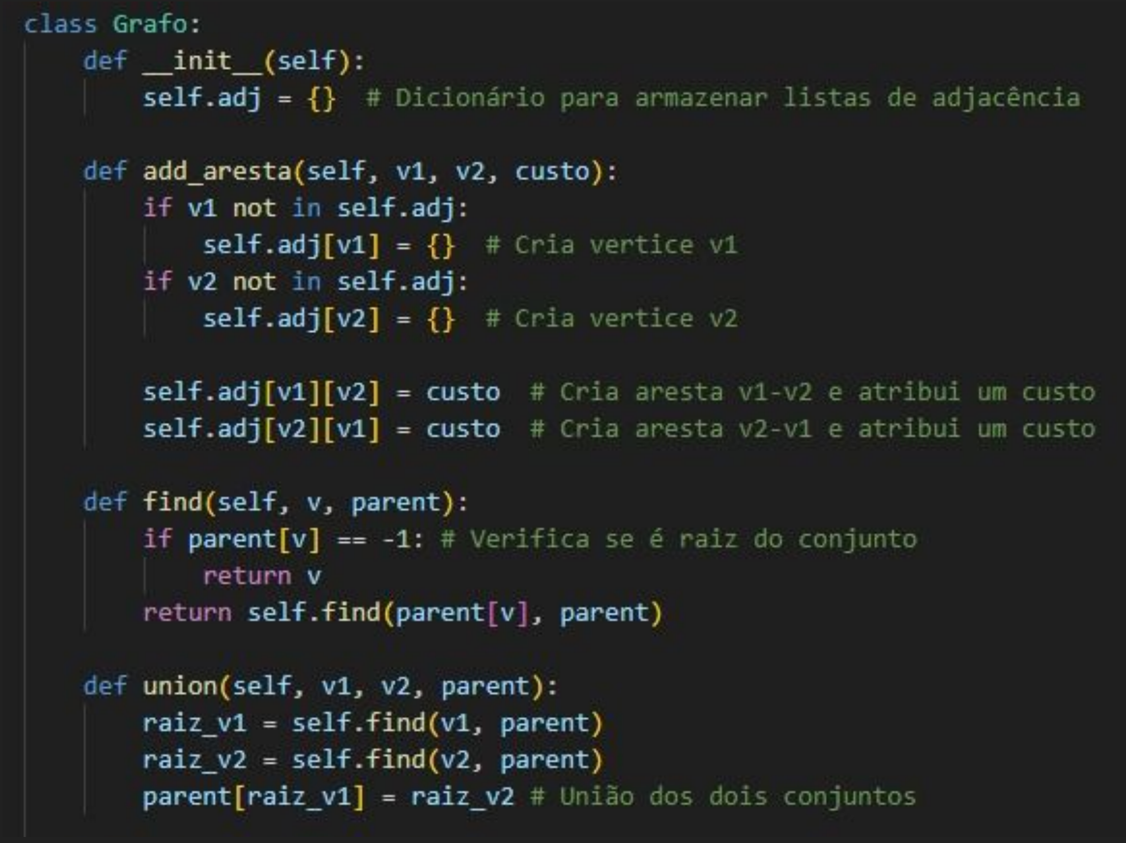
Sample Input

```
5
1 2 5
1 3 5
1 4 5
1 5 5
1
2 3 2
6
1 2 5
1 3 5
1 4 5
1 5 5
3 4 8
4 5 8
```

Sample Output

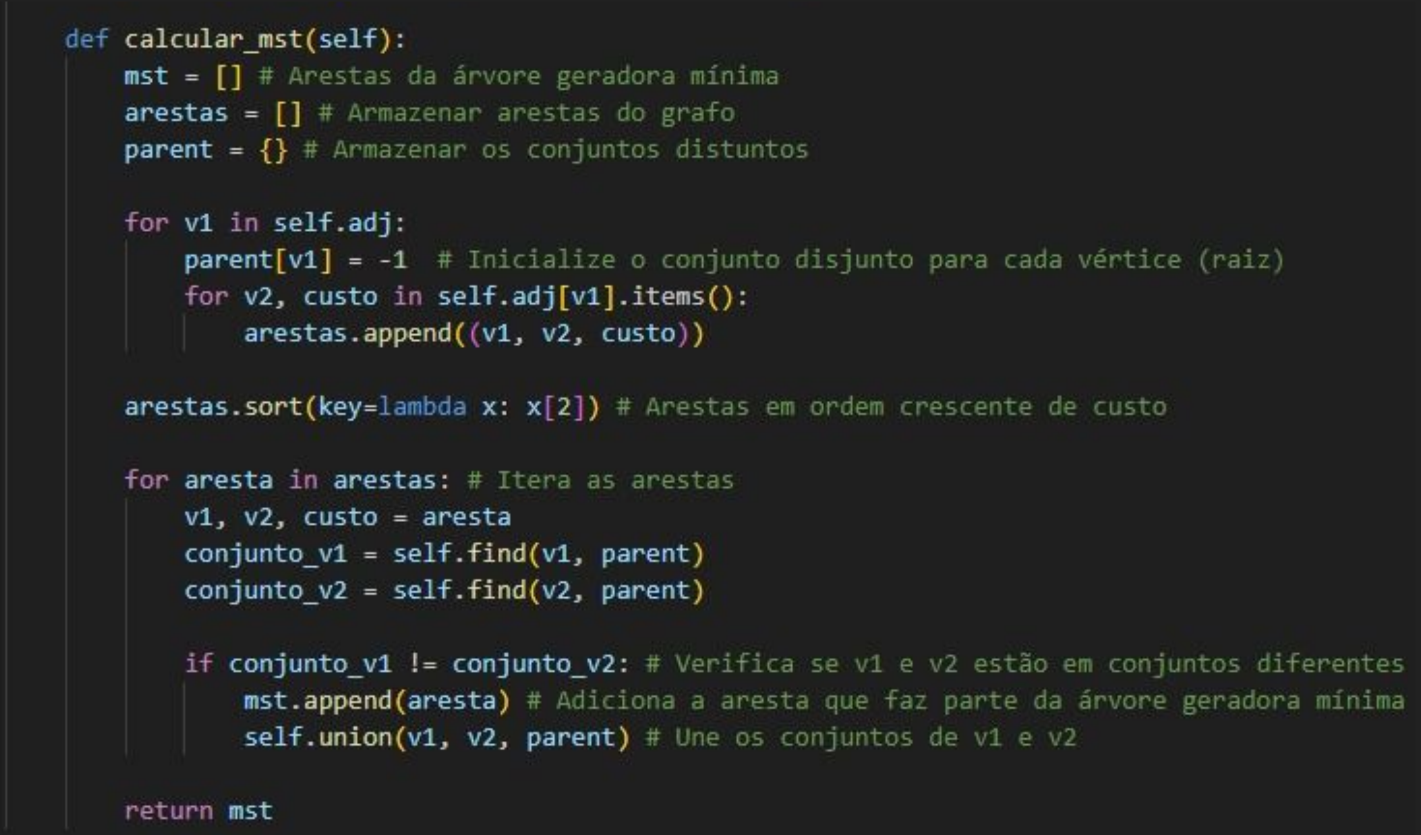
```
20
17
```

EP4: UVa 00908 - Re-Connecting Computer Sites



```python
class Grafo:
    def __init__(self):
        self.adj = {}  # Dicionário para armazenar listas de adjacência

    def add_aresta(self, v1, v2, custo):
        if v1 not in self.adj:
            self.adj[v1] = {}  # Cria vertice v1
        if v2 not in self.adj:
            self.adj[v2] = {}  # Cria vertice v2

        self.adj[v1][v2] = custo  # Cria aresta v1-v2 e atribui um custo
        self.adj[v2][v1] = custo  # Cria aresta v2-v1 e atribui um custo

    def find(self, v, parent):
        if parent[v] == -1: # Verifica se é raiz do conjunto
            return v
        return self.find(parent[v], parent)

    def union(self, v1, v2, parent):
        raiz_v1 = self.find(v1, parent)
        raiz_v2 = self.find(v2, parent)
        parent[raiz_v1] = raiz_v2 # União dos dois conjuntos
```

EP4: UVa 00908 - Re-Connecting Computer Sites



```
def calcular_mst(self):
    mst = [] # Arestas da árvore geradora mínima
    arestas = [] # Armazenar arestas do grafo
    parent = {} # Armazenar os conjuntos distuntos

    for v1 in self.adj:
        parent[v1] = -1  # Inicialize o conjunto disjunto para cada vértice (raiz)
        for v2, custo in self.adj[v1].items():
            arestas.append((v1, v2, custo))

    arestas.sort(key=lambda x: x[2]) # Arestas em ordem crescente de custo

    for aresta in arestas: # Itera as arestas
        v1, v2, custo = aresta
        conjunto_v1 = self.find(v1, parent)
        conjunto_v2 = self.find(v2, parent)

        if conjunto_v1 != conjunto_v2: # Verifica se v1 e v2 estão em conjuntos diferentes
            mst.append(aresta) # Adiciona a aresta que faz parte da árvore geradora mínima
            self.union(v1, v2, parent) # Une os conjuntos de v1 e v2

    return mst
```

EP9: Kattis communications satellite

Input

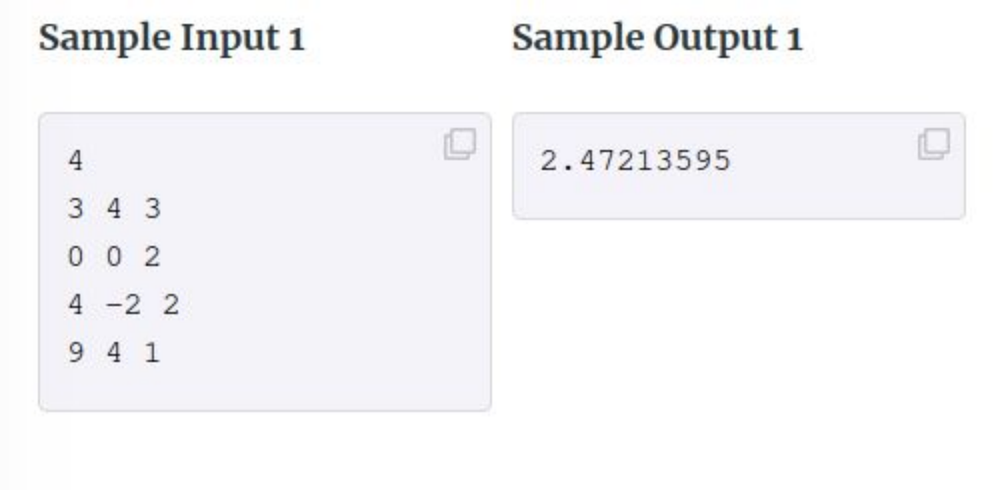
The first line of the input contains a single integer 1 <= N <= 2000, the number of dish antennas attached to the satellite.

Then follow N lines, each of which contains three integers X, Y, and Z specifying the location and radius of one dish antenna. These integers satisfy the bounds -1000 <= X, Y <= 1000 and 1 <= R <= 100.

Output

Compute the satellite design that minimizes the sum of beam lengths, while obeying the above specifications, and print that sum. The answer is considered correct if the absolute or relative error is less than 10^{-6}.

EP9: Kattis communications satellite



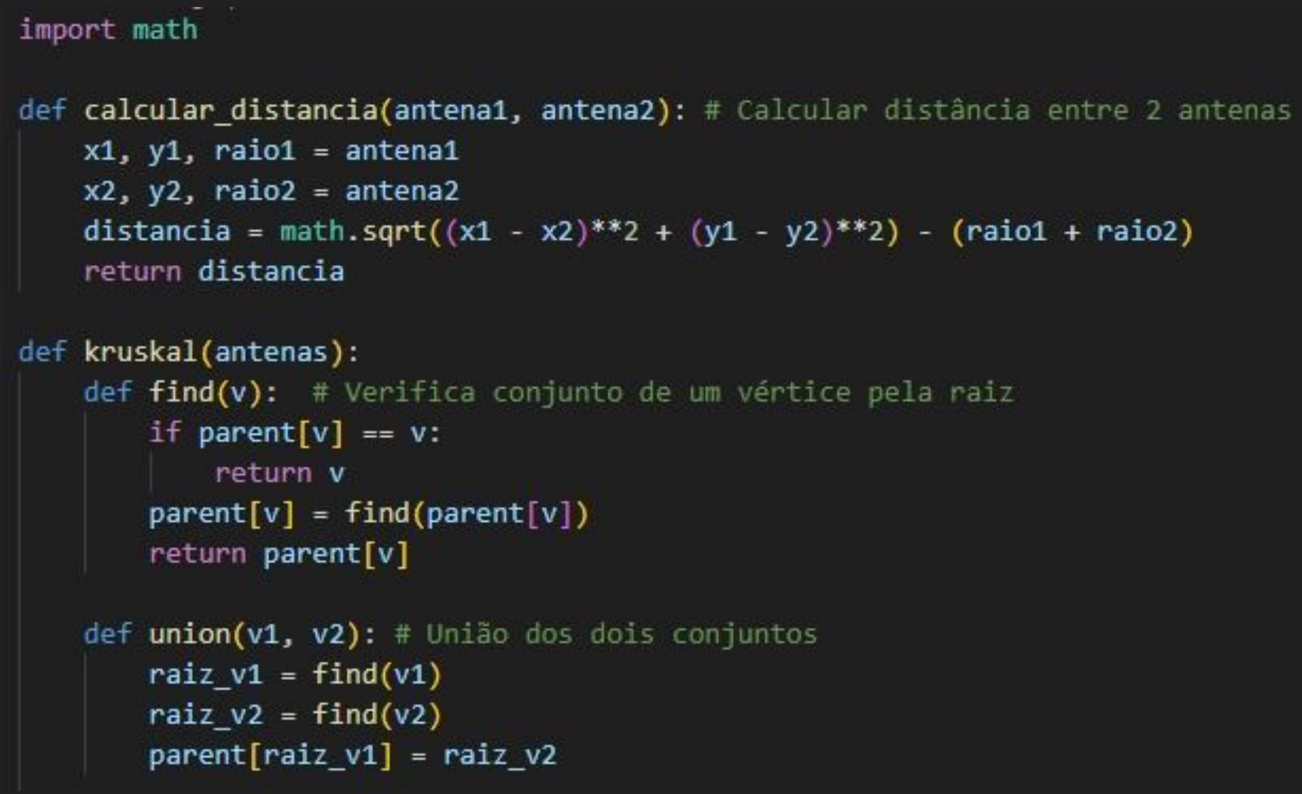
Sample Input 1

```
4
3 4 3
0 0 2
4 -2 2
9 4 1
```

Sample Output 1

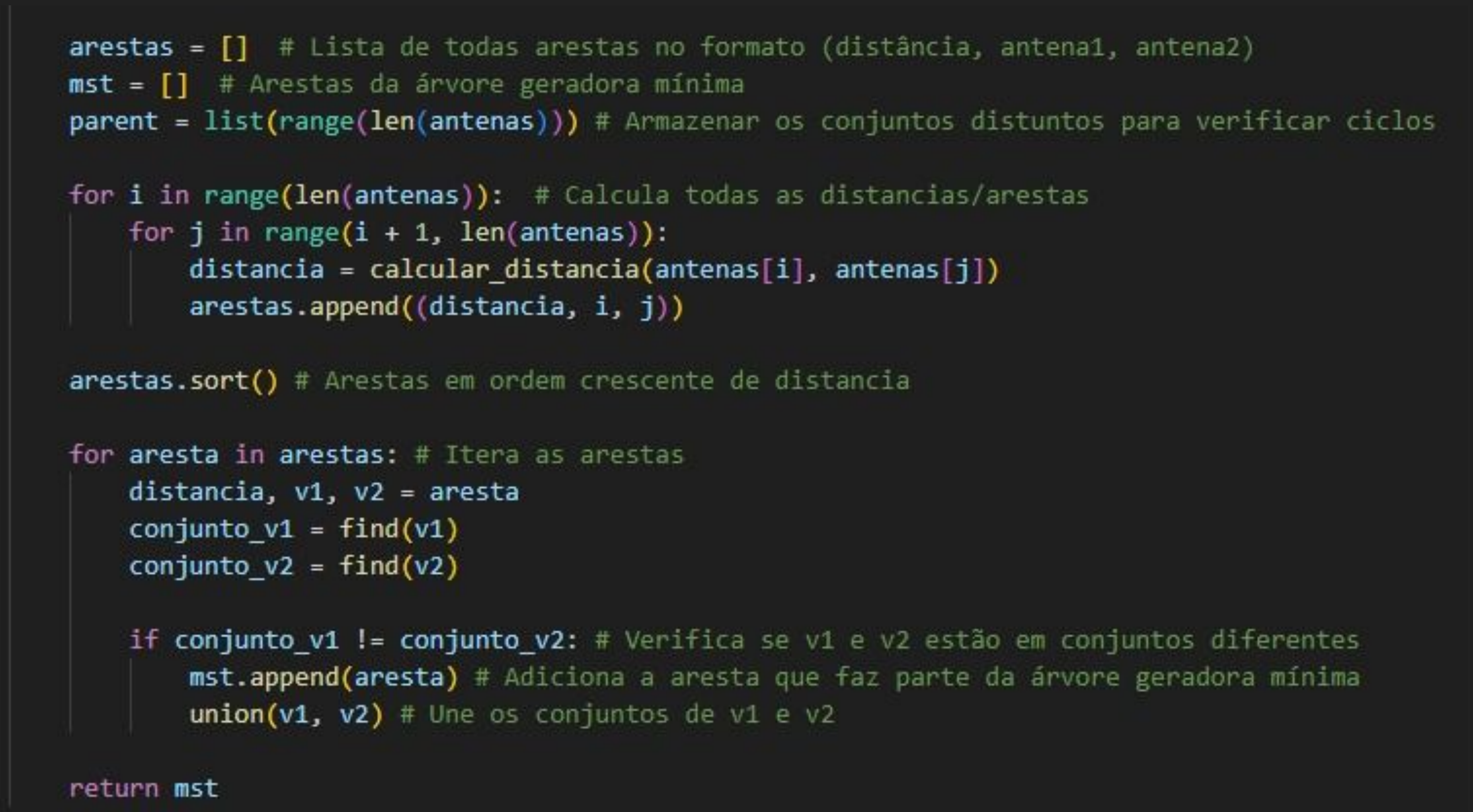
```
2.47213595
```

EP9: Kattis communications satellite



```python
import math

def calcular_distancia(antena1, antena2): # Calcular distância entre 2 antenas
    x1, y1, raio1 = antena1
    x2, y2, raio2 = antena2
    distancia = math.sqrt((x1 - x2)**2 + (y1 - y2)**2) - (raio1 + raio2)
    return distancia

def kruskal(antenas):
    def find(v):  # Verifica conjunto de um vértice pela raiz
        if parent[v] == v:
            return v
        parent[v] = find(parent[v])
        return parent[v]

    def union(v1, v2): # União dos dois conjuntos
        raiz_v1 = find(v1)
        raiz_v2 = find(v2)
        parent[raiz_v1] = raiz_v2
```

EP9: Kattis communications satellite



```
    arestas = []  # Lista de todas arestas no formato (distância, antena1, antena2)
    mst = []  # Arestas da árvore geradora mínima
    parent = list(range(len(antenas))) # Armazenar os conjuntos distuntos para verificar ciclos

    for i in range(len(antenas)):  # Calcula todas as distancias/arestas
        for j in range(i + 1, len(antenas)):
            distancia = calcular_distancia(antenas[i], antenas[j])
            arestas.append((distancia, i, j))

    arestas.sort() # Arestas em ordem crescente de distancia

    for aresta in arestas: # Itera as arestas
        distancia, v1, v2 = aresta
        conjunto_v1 = find(v1)
        conjunto_v2 = find(v2)

        if conjunto_v1 != conjunto_v2: # Verifica se v1 e v2 estão em conjuntos diferentes
            mst.append(aresta) # Adiciona a aresta que faz parte da árvore geradora mínima
            union(v1, v2) # Une os conjuntos de v1 e v2

    return mst
```