1584. Min Cost to Connect All Points - Médio

Problema:

Você é dado um array points representando coordenadas inteiras de alguns pontos em um plano 2D, onde points[i] = [xi, yi]. O custo de conectar dois pontos [xi, yi] e [xj, yj] é a distância de Manhattan entre eles:

Distância de Manhattan = |xi-xj| + |yi-yj|

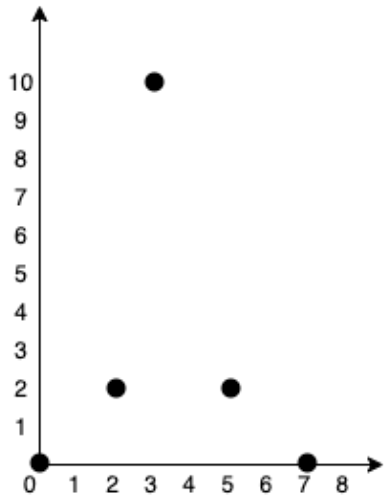
Retorne o custo mínimo para conectar todos os pontos, de forma que haja exatamente um caminho simples entre qualquer par de pontos.

Restrições:

- 1 <= points.length <= 1000;
- -10^6 <= xi,yi <= 10^6;
- Todos os pares (xi,yi) são distintos.

Exemplos:

Exemplo 1:



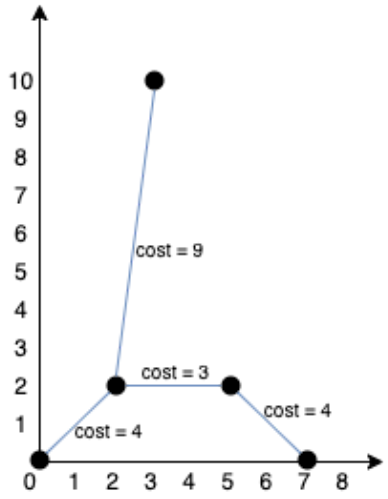
Entrada:

```
points = [[0,0],[2,2],[3,10],[5,2],[7,0]]
```

Saída:

```
20
```

Explicação: Os pontos podem ser conectados como mostrado na imagem, resultando em um custo total de 20. Note que há um caminho único entre qualquer par de pontos.



Exemplo 2:

Entrada:

```
points = [[3,12],[-2,5],[-4,1]]
```

Saída:

```
18
```

Explicação:

A distância mínima para conectar todos os pontos é 18, calculada da seguinte forma: conectamos o ponto [−4,1] ao ponto [−2,5] com custo |−4−(−2)|+|1−5|=6, e depois conectamos [−2,5] ao ponto [3,12] com custo |−2−3|+|5−12|=12. A soma desses custos é 6+12=18. Essa configuração forma a árvore geradora mínima, pois é o caminho que conecta todos os pontos com o menor custo possível.

Solução proposta:

A solução para este problema pode ser obtida utilizando o algoritmo de Prim para construir uma Árvore Geradora Mínima (MST). O algoritmo funciona selecionando arestas de menor custo iterativamente, enquanto conecta todos os pontos sem formar ciclos.

```
import heapq

class Solution(object):
    def minCostConnectPoints(self, points):
```

```
        """
        :type points: List[List[int]]
        :rtype: int
        """
        # Número de pontos
        n = len(points)

        # Função para calcular a distância de Manhattan entre dois pontos
        def distancia_manhattan(p1, p2):
            return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])

        # Fila de prioridade para gerenciar arestas
        min_heap = []

        # Nós já incluídos na MST
        MST = [False] * n

        # Começamos com o nó 0
        min_heap.append((0, 0))

        custo_total = 0
        arestas_usadas = 0

        while arestas_usadas < n:
            # Pegar a aresta de menor custo
            custo, curr = heapq.heappop(min_heap)

            # Se o nó já está na MST, ignorar
            if MST[curr]:
                continue

            # Adiciona o nó à MST
            MST[curr] = True
            custo_total += custo
            arestas_usadas += 1

            # Explorar os vizinhos do nó atual
            for prox_no in range(n):
                if not MST[prox_no]:
                    prox_custo = distancia_manhattan(points[curr],
points[prox_no])
                    heapq.heappush(min_heap, (prox_custo, prox_no))

        return custo_total

# Testes locais
if __name__ == "__main__":
    solution = Solution()

    # Exemplo 1
    pontos1 = [[0, 0], [2, 2], [3, 10], [5, 2], [7, 0]]
    print("Exemplo 1:", solution.minCostConnectPoints(pontos1))

    # Exemplo 2
```

```
    pontos2 = [[3, 12], [-2, 5], [-4, 1]]
    print("Exemplo 2:", solution.minCostConnectPoints(pontos2))
```