**Conectividade:** Há um caminho possível entre cada par de vértices.

**Algoritmo de Componentes-Fortemente-Conexas:**

* Encontra os conjuntos máximos.
  + Conjunto máximo: visa formar um conjunto conexo com a maior quantidade de vértices possíveis.
    - Todos os vértices de um conjunto conexo devem alcançar todos os outros vértices do conjunto conexo.
* Usam busca em profundidade repetidas vezes.
* Entrada: grafo (preferencialmente não-conexo) dirigido não-ponderado G.
* Saída: vetor de visitados C, vetor de tempo do começo da vista T, vetor tempo do término da visita F, vetor de ancestrais A.
  + O vetor A é o vetor que define os componentes fortemente conexos.
  + Complexidade: O(|V|+|E|).

**Algoritmo DFS para Ordenação Topológica:**

* Encontra a lista contendo a ordenação topológica que contém os vértices mais dependentes ao final dessa lista (mais à direita).
* Entrada: grafo dirigido acíclico não-ponderado G.
* Saída: vetor de visitados C, vetor de tempo do começo da vista T, vetor tempo do término da visita F, ordenação topológica O.
  + O vetor O é o vetor que define a ordenação topológica.
  + Complexidade: O(|V|+|A|).

**Kruskal ou Prim:**

* Encontra a árvore geradora mínima.
  + Árvore geradora mínima: é uma árvore (portanto não contém ciclo) que conecta todos os vértices no qual a soma de todas as arestas possui custo mínimo.
* Aresta segura/leve: aresta de menor custo que não gera um ciclo.
* Entrada: grafo conexo não-dirigido e ponderado G.
* Saída:
  + Kruskal: lista de conjuntos de vértices S, lista de conjunto de arestas ordenadas pelo menor custo E’, vetor contendo as arestas da árvore geradora mínima A, menor custo w(A).
    - O vetor A é o vetor que define a árvore geradora mínima.
    - Complexidade: O(|E|log2|E|).
  + Prim: estrutura de prioridade Q que possui os vértices V e suas respectivas arestas leves K, vetor, que junto com V, contém as arestas da árvore geradora mínima A, menor custo w(A).
    - O vetor A junto ao vetor V é o que define a árvore geradora mínima.
    - Complexidade: O(|E|log2|V|), podendo ser aprimorada usando Fibonacci Heaps para O(|E|+log2|V|).

a) Podemos representar o problema usando um grafo não-dirigido e ponderado. Não-dirigido pois a ideia de direção entre uma área e outra não faz sentido. Ponderado pois existe uma distância entre uma área e outra.

Criamos o grafo G = (V, E, g, w). Onde V é o conjunto de áreas (vértices); E é o conjunto de caminhos entre áreas (arestas) no qual cada aresta (ci, cj) pertencente a E representa um caminho entre ci e cj; g é a função A -> R2 que representa a coordenada de uma área; w é a função (R2, R2) -> R+ que representa a distância euclidiana entre dois pontos.

O problema pode ser resolvido com uma versão modificada do algoritmo de Kruskal (algoritmo 21). A escolha desse algoritmo ocorre pois queremos conectar as áreas com caminhos utilizando metragem mínima (distância mínima). Precisamos realizar uma modificação no algoritmo pois precisamos calcular a distância (dada pela distância euclidiana) entre cada par de áreas.

Nosso input é G = (V, g), nosso output é a árvore geradora mínima A. Precisamos montar o conjunto de caminhos (arestas). Para criar o conjunto de arestas, podemos criar uma aresta entre cada vértice considerando a coordenada do vértice (dado pela função g). Sabemos que na linha 4 do algoritmo temos a ordenação das arestas, sendo assim, dentro da ordenação das arestas, devemos usar a função w (distância euclidiana) como a função que determina o peso das arestas. Não entrarei em detalhes sobre a função da distância euclidiana, visto que w está dentro da linha 4 do algoritmo e tal linha não está sendo detalhada no algoritmo. Há algumas funções já implementadas em, por exemplo, bibliotecas de Python, que retorna a distância euclidiana entre dois duas coordenadas. Tais bibliotecas podem ser numpy ou scipy.

Modificações:

No algoritmo 21, após a linha 3:

4. E = {}

5. for i in range(len(V)):

6.            for j in range(len(V)):

7.                            if j != i:

8.                                            E.add((g(V[i]), g(V[j]))

b) A complexidade do algoritmo é a mesma que a complexidade do algoritmo de Kruskal O(|E| log2|E|) mais o custo para calcular o as arestas O(|V|^2). Portanto O(|E|log2|E| + |V|^2), ou simplesmente O(|V|^2).