Unidade 2 Trabalho 3 (U2T3)

Algoritmos Clássicos (Dijkstra e Kruskal)

E a complexidade (dijkstra)?

$O(V^2+E)$ time
$O(V)$ space

1. **Encontrando o vértice com a menor distância:**

Em cada iteração principal, o algoritmo procura pelo nó não visitado com a menor distância conhecida. Essa busca percorre todos os vértices (ainda não visitados) e pode levar até O(V) tempo por iteração, onde V é o número total de vértices.

2. **Repetindo a iteração principal para todos os vértices:**

Como é necessário selecionar o vértice de menor distância V vezes (uma vez para cada vértice), você repete essa busca de tempo O(V) V vezes, resultando em $O(V)* O(V) = O(V^2)$.

3. **Relaxando as arestas:**

Após selecionar um vértice, você itera pelas suas arestas de saída para possivelmente atualizar as distâncias dos seus vizinhos. Durante toda a execução do algoritmo, cada aresta é considerada pelo menos uma vez, somando um custo de O(E), onde E é o número total de arestas.

Como podemos melhorar(dijkstra)?

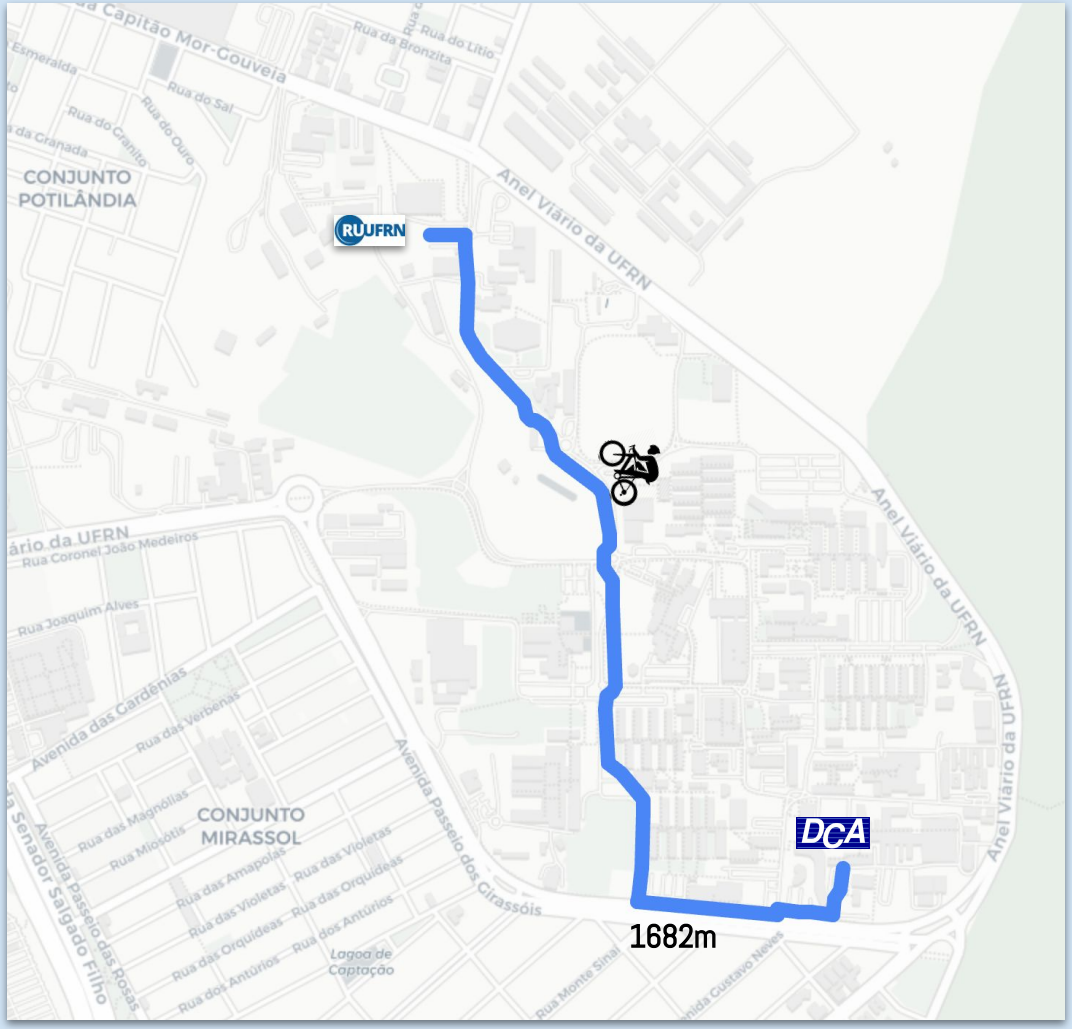
O(V+E)*log(V) time
O(V) space

dijsktra_min_heap.ipynb

Já avaliamos em trabalhos anteriores encontrar o caminho mais curto entre dois pontos usando o OSMnx.

```
1. Importar as bibliotecas necessárias:
   Import osmnx
   Import networkx

2. Definir coordenadas geográficas dos pontos:
   ponto_inicial = (latitude_inicial, longitude_inicial)
   ponto_final = (latitude_final, longitude_final)

3. Baixar o grafo da área:
   grafo = osmnx.graph_from_place("Nome da cidade ou coordenadas",
                                  network_type="walk" ou "drive")

4. Encontrar os nós mais próximos:
   no_inicial = osmnx.nearest_nodes(grafo, longitude_inicial,
                                    latitude_inicial)
   no_final = osmnx.nearest_nodes(grafo,
                                  longitude_final, latitude_final)

5. Calcular o caminho mais curto:
   caminho = networkx.shortest_path(grafo,
                                    source=no_inicial,
                                    target=no_final, weight="length")

6. Visualizar o caminho no mapa:
   figura, eixo = osmnx.plot_graph_route(grafo, caminho)
```

OSMnx



```
1. Importar as bibliotecas necessárias:
   Import osmnx
   Import networkx

2. Definir coordenadas geográficas dos pontos:
   ponto_inicial = (latitude_inicial, longitude_inicial)
   ponto_final = (latitude_final, longitude_final)

3. Baixar o grafo da área:
   grafo = osmnx.graph_from_place("Nome da cidade ou coordenadas",
                                  network_type="walk" ou "drive")

4. Encontrar os nós mais próximos:
   no_inicial = osmnx.nearest_nodes(grafo, longitude_inicial,
                                    latitude_inicial)
   no_final = osmnx.nearest_nodes(grafo,
                                  longitude_final, latitude_final)

5. Calcular o caminho mais curto:
   caminho = networkx.shortest_path(grafo,
                                    source=no_inicial,
                                    target=no_final, weight="length")

6. Visualizar o caminho no mapa:
   figura, eixo = osmnx.plot_graph_route(grafo, caminho)
```

OSMnx

Problema #1

Avaliar o algoritmo de dijkstra compartilhado no arquivo dijsktra_min_heap.ipynb com a solução presente no networkx e visualizar o resultado no OSMnx. Ajustes no código podem ser necessários.

Requisito #1

- Escolher 10 pares (origem,destino) como pontos de interesse da **cidade de Natal-RN** e avaliar o algoritmo de dijkstra (networkx e min heap) comparando visualmente as soluções no OSMnx.

```
1. Importar as bibliotecas necessárias:
   Import osmnx
   Import networkx

2. Definir coordenadas geográficas dos pontos:
   ponto_inicial = (latitude_inicial, longitude_inicial)
   ponto_final = (latitude_final, longitude_final)

3. Baixar o grafo da área:
   grafo = osmnx.graph_from_place("Nome da cidade ou coordenadas",
                                  network_type="walk" ou "drive")

4. Encontrar os nós mais próximos:
   no_inicial = osmnx.nearest_nodes(grafo, longitude_inicial,
                                    latitude_inicial)
   no_final = osmnx.nearest_nodes(grafo,
                                  longitude_final, latitude_final)

5. Calcular o caminho mais curto:
   caminho = networkx.shortest_path(grafo,
                                    source=no_inicial,
                                    target=no_final, weight="length")

6. Visualizar o caminho no mapa:
   figura, eixo = osmnx.plot_graph_route(grafo, caminho)
```

OSMnx

Problema #1

Avaliar o algoritmo de dijkstra compartilhado no arquivo dijsktra_min_heap.ipynb com a solução presente no networkx e visualizar o resultado no OSMnx. Ajustes no código podem ser necessários.

Requisito #2

- Organize um repositório no Github com tudo o que foi gerado, incluindo arquivo README, códigos, figuras, explicações, etc.
- Faça um vídeo de até 5min (sugestão, Loom) explicando a solução e os resultados. O link deverá estar do vídeo estar no repositório.

```
1. Importar as bibliotecas necessárias:
   Import osmnx
   Import networkx

2. Definir coordenadas geográficas dos pontos:
   ponto_inicial = (latitude_inicial, longitude_inicial)
   ponto_final = (latitude_final, longitude_final)

3. Baixar o grafo da área:
   grafo = osmnx.graph_from_place("Nome da cidade ou coordenadas",
                                  network_type="walk" ou "drive")

4. Encontrar os nós mais próximos:
   no_inicial = osmnx.nearest_nodes(grafo, longitude_inicial,
                                    latitude_inicial)
   no_final = osmnx.nearest_nodes(grafo,
                                  longitude_final, latitude_final)

5. Calcular o caminho mais curto:
   caminho = networkx.shortest_path(grafo,
                                    source=no_inicial,
                                    target=no_final, weight="length")

6. Visualizar o caminho no mapa:
   figura, eixo = osmnx.plot_graph_route(grafo, caminho)
```

OSMnx

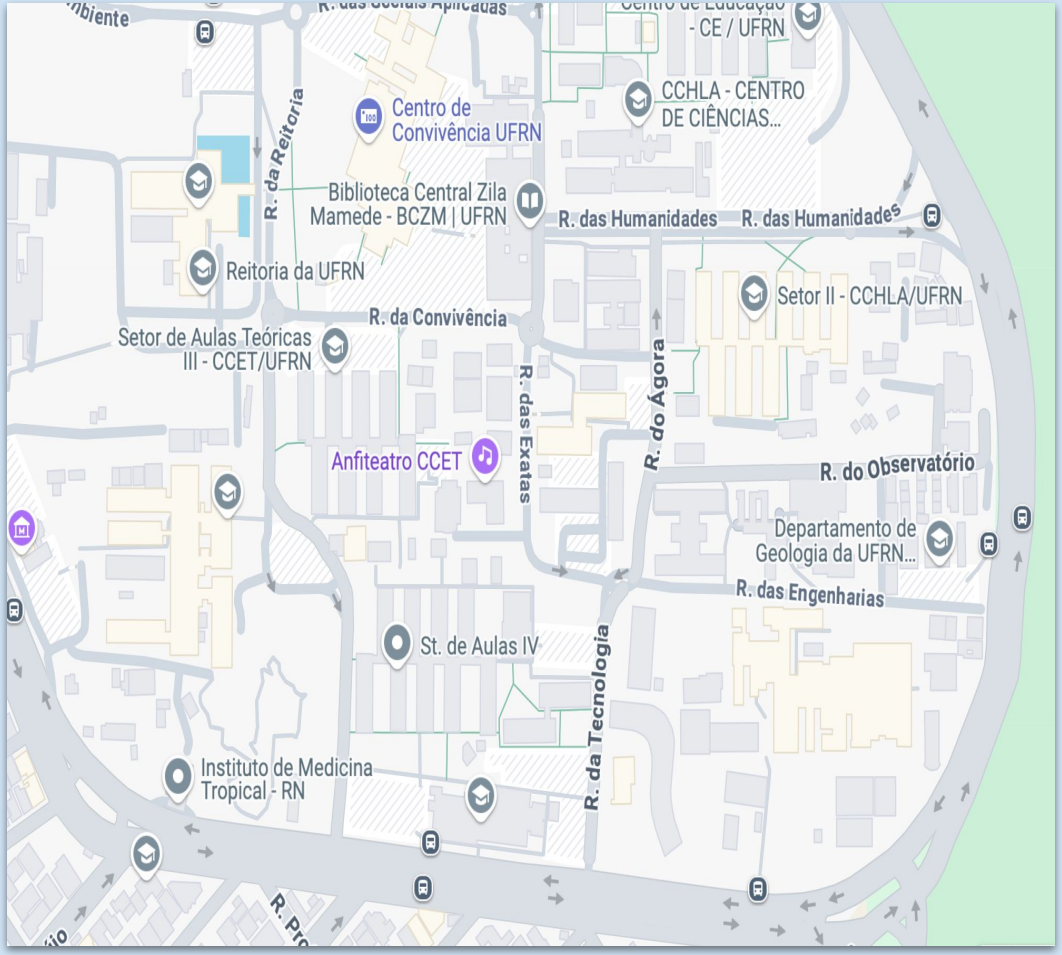
Problema #1

Avaliar o algoritmo de dijkstra compartilhado no arquivo dijsktra_min_heap.ipynb com a solução presente no networkx e visualizar o resultado no OSMnx. Ajustes no código podem ser necessários.

Requisito #3

- Trabalho individual
- Nota: 2,5 pontos na unidade 2.

E sobre o Kruskal (mst)?



Sugestões de problemas

Tirol, Natal-RN

Otimização de Infraestrutura:
Use o Algoritmo de Kruskal para projetar o layout mais econômico para infraestruturas como fibras ópticas, tubulações de água ou redes elétricas, minimizando o comprimento total das conexões necessárias.

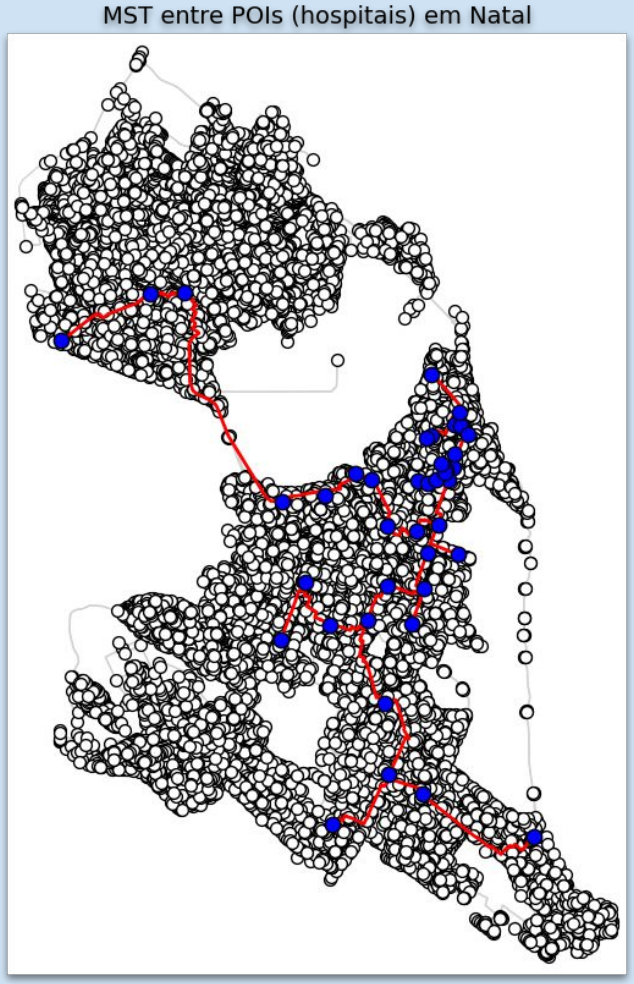
Planejamento de Transporte:
Conecte grandes centros de transporte (por exemplo, terminais de ônibus, estações de trem e aeroportos) utilizando uma Árvore Geradora Mínima (MST) para identificar as rotas mais curtas e reduzir as distâncias de viagem.

Otimização de Rota Turística:
Gere uma MST conectando principais atrações turísticas (por exemplo, museus, pontos de referência, praias) para criar rotas de turismo eficientes que minimizem a distância percorrida.

Planejamento de Expansão Urbana:
Use uma MST para conectar bairros em desenvolvimento ou áreas em construção à rede urbana existente com custos mínimos de infraestrutura.

Análise de Infraestrutura Crítica:
Identifique estradas ou interseções críticas comparando a MST com a rede viária original, destacando as vias essenciais para manter a conectividade.

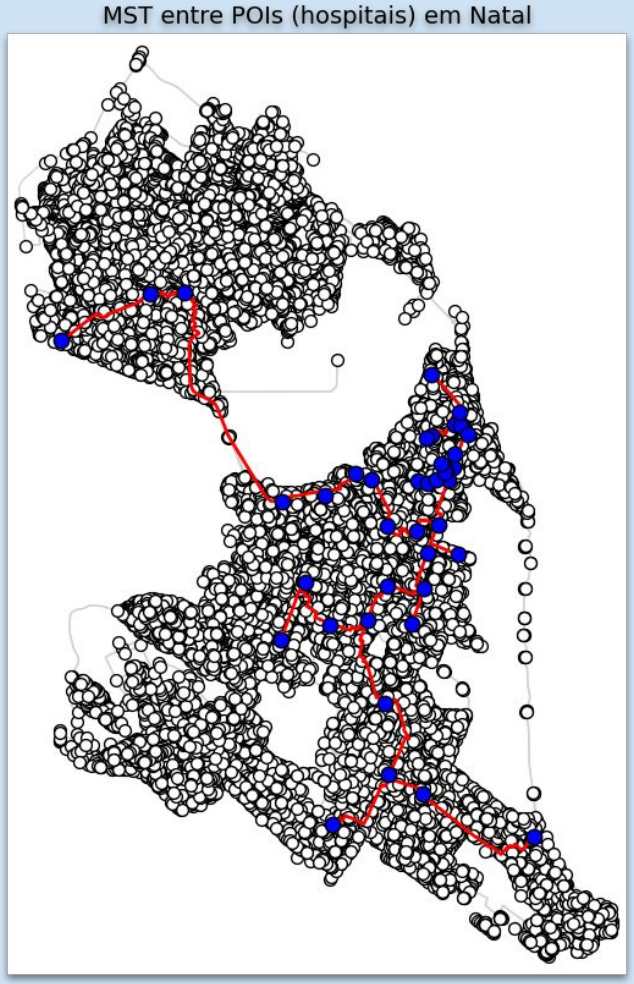
jupyter kruskal_natal.ipynb



Requisito #01

Dado um dos problemas anteriores, escolha alguns pontos de interesse (PoI - points of interest) e calcule a MST para esses pontos. Use o notebook kruskal_natal.ipynb como referência.

jupyter kruskal_natal.ipynb

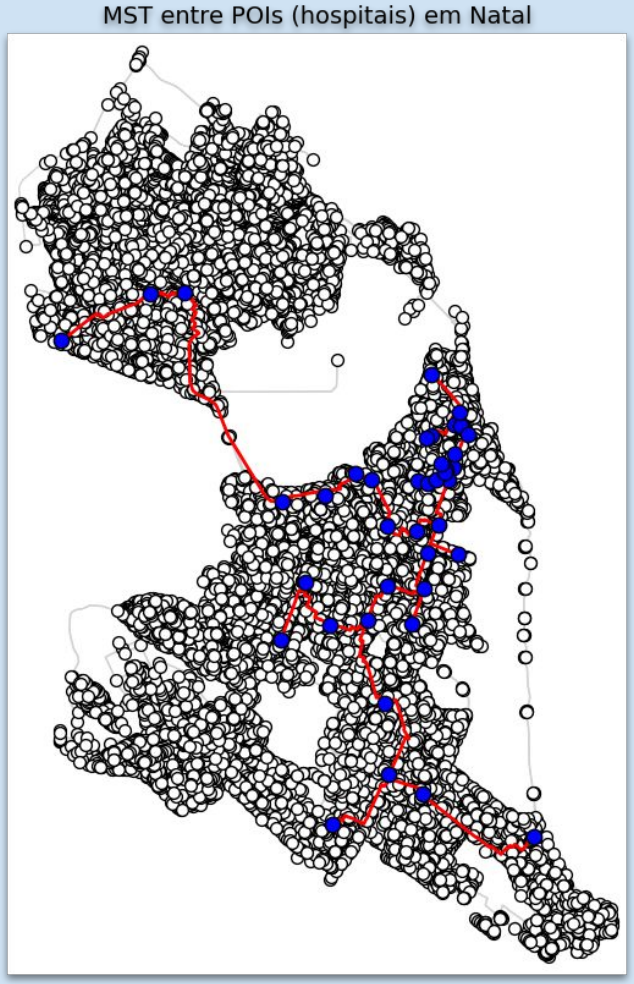


Requisito #2

Organize um repositório no Github com tudo o que foi gerado, incluindo arquivo README, códigos, figuras, explicações, etc.

Faça um vídeo de até 5min (sugestão, Loom) explicando a solução e os resultados. O link deverá estar do vídeo estar no repositório.

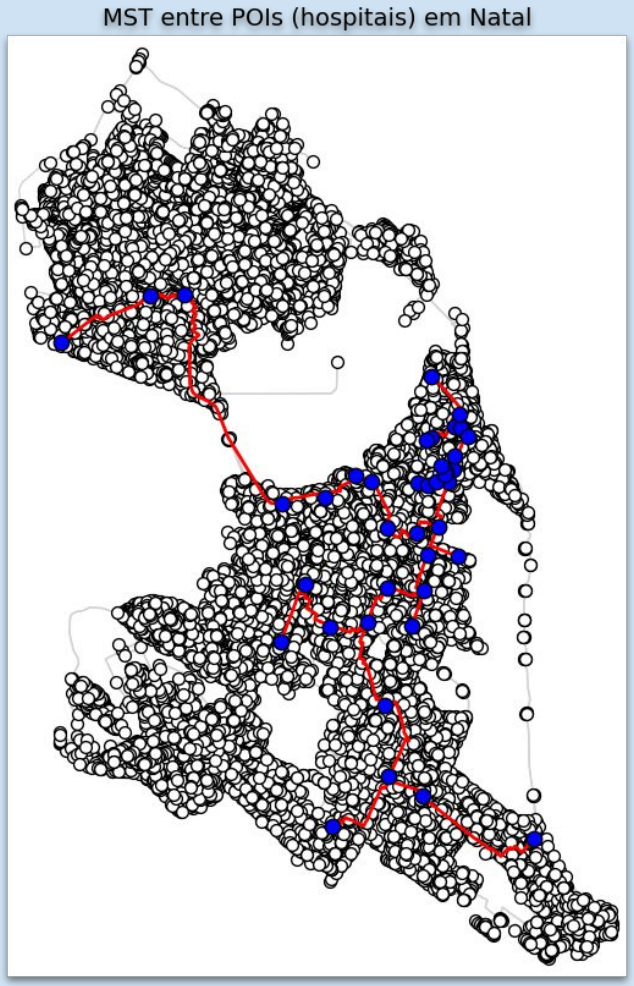
jupyter kruskal_natal.ipynb



Requisito #3

Trabalho individual
Nota: 2,5 pontos na unidade 2.

jupyter kruskal_natal.ipynb



Submissão

Submeter o link do repositório no github referente a tarefa.

Prazo: 13/01/2025 as 23h59

Obs: os arquivos notebooks sugeridos como referência estão no repositório da disciplina no github na semana 12.

jupyter kruskal_natal.ipynb