Relatório – Trabalho Final "Not Waze"

Estrutura de Dados e Algoritmos – 2021 – Prof. Jorge Poco

Daniel Csillag
Rafael Portácio
Wellington José

16 de novembro de 2021

1 Introdução

Criamos um mini-Waze para a cidade do Rio de Janeiro usando diferentes algoritmos de busca para encontrar o menor caminho entre dois pontos dados, segue o link para o nosso repositório: link para o repositório e o link para o video.

2 Coleta e armazenamento dos dados

Assim como foi especificado na proposta do projeto, extraímos os dados usando a biblioteca OSMnx. Com eles criamos um arquivo .JSON (de tamanho 56M) com as informações dos nós (id, latitude e longitude) e das arestas (nó de saída, nó de chegada, comprimento e tempo estimado de chegada [ETA]). Fazemos uma leitura desse .JSON precisamos tirar os espaços do **json** para poder trabalhar com ele usando o **sed** para em código C++ criar o objeto grafo, que contém objetos arestas e nós. Organizamos os nós num HASH MAP. Para as arestas, fizemos uma lista de adjacências (adjacency list) já que temos um grafo bem esparso (pois os graus dos nós são pequenos em relação à quantidade de nós total). Dessa forma, teríamos que uma matriz de adjacências seria desnecessariamente grande pois teria dimensões extremamente altas e a grande maioria dos elementos dela seriam iguais a zero.

3 Backend

Aqui usamos os algoritmos Dijsktra, A* com heurística euclidiana e A* com heurística manhattan. Usando uma MIN HEAP que implementamos para gerenciar os custos dos caminhos.

A principal diferença do dijkstra para o A* é a adição da heurística que está implementada como uma função que é passada como argumento para o algoritmo, sua a utilidade é dar prioridade para os nós próximos do destino final, levando assim menos passos. Conforme os gráficos abaixo, o Dijsktra ainda conseguiu ser mais consistente na velocidade por não se preocupar com o cálculo da heurística.

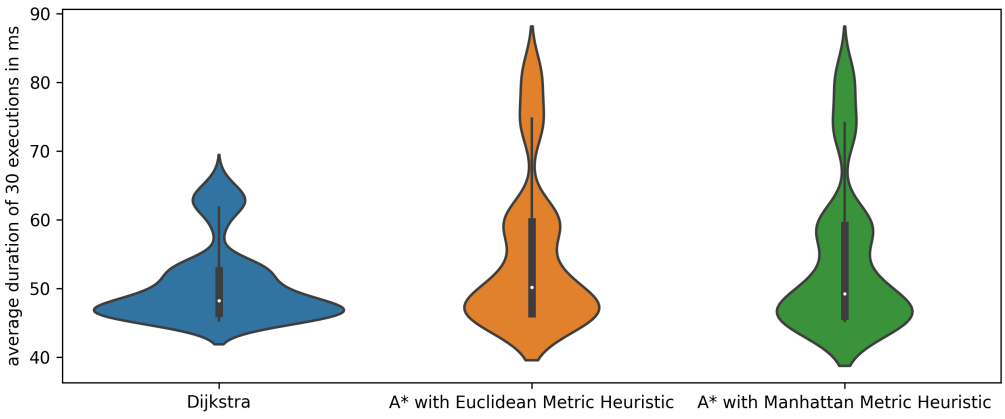

Figura 1: Comparação dos tempos de execução dos algoritmos

4 Frontend

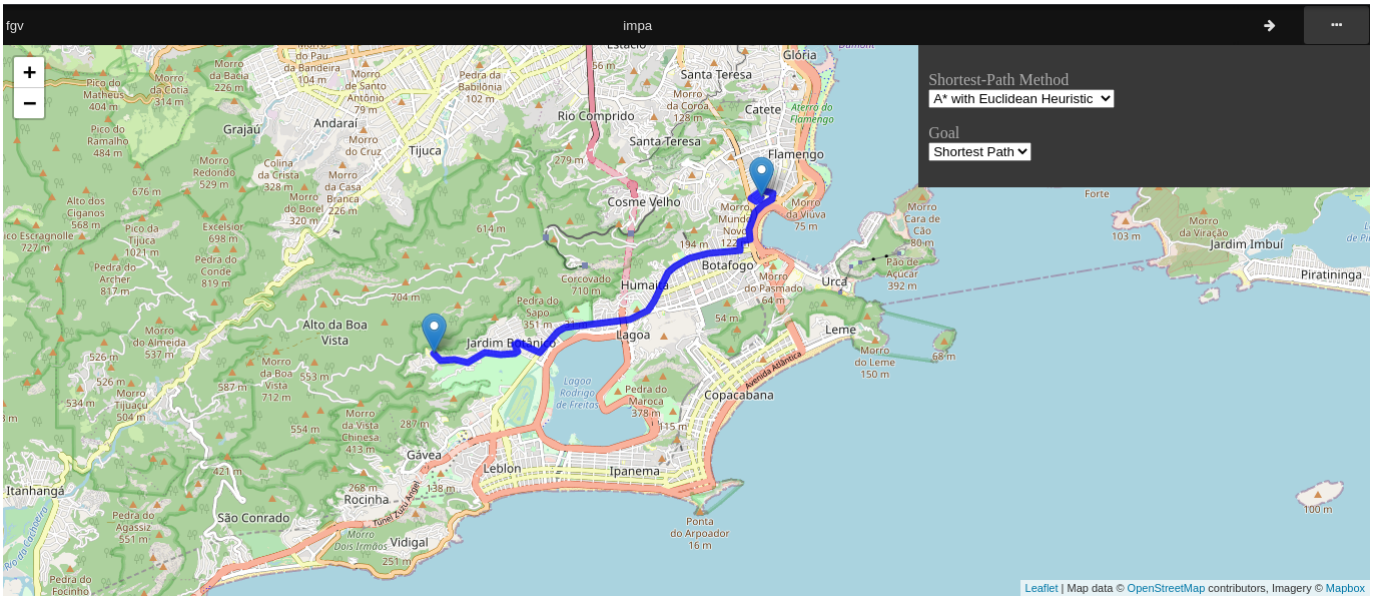

Figura 2: Interface web toda

No Frontend, usamos a biblioteca Leaflet.js para fazer o mapa, colocamos algumas features, tais como:

- permitir a escolha do algoritmo a ser utilizado a partir de uma dropdown.
- poder selecionar os pontos de início e fim a partir da função de geocoding do nominatim, assim permitindo escrever endereços para obter os pontos.
- poder mudar os pontos de início e fim ao arrastar seus respectivos markers pelo mapa.
- armazenar na url as coordenadas de início e fim, assim como o algoritmo utilizado, para assim poder obter o mesmo trajeto ao copiar a url para outra aba.
- exibir o comprimento, o tempo estimado de chegada e o tempo de execução do algoritmo utilizado ao posicionar o mouse sobre um dado caminho.
- podemos calcular não só o menor caminho como também o mais rápido a partir de uma dropdown. Logo abaixo temos os 2 casos:

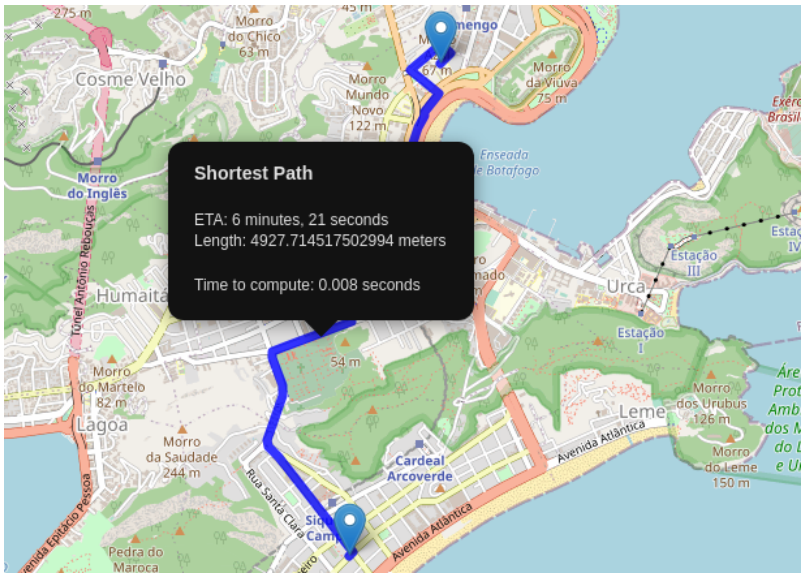

Figura 3: Caminho com distancia

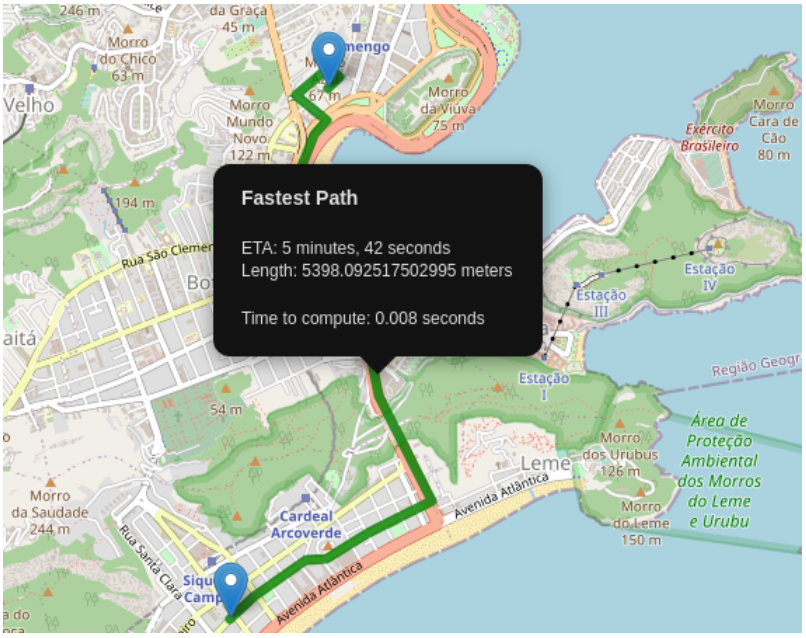

Figura 4: Caminho mais rápido

5 Problemas encontrados

Conforme testamos, encontramos problemas no Waze que tínhamos:

5.1 Lookup pelo nó mais próximo

No início, nossa implementação encontrava o nó mais próximo dos pontos de início e de fim, mas nisso encontramos problemas, pois assim só podíamos traçar caminhos conectando nós e por mais que adicionássemos uma conexão do ponto de início ao primeiro nó e do ponto de final ao segundo nó, essa conexão poderia ser estranha ou até mesmo um trajeto contramão.

Visando resolver isso, nós implementamos uma busca pela aresta mais próxima e projetamos o ponto nela, e com essa aresta podemos fazer a conexão do ponto de início a um nó que é adequado para ele se conectar.

5.1.1 Tomando frações de arestas

No caso, para conectar o ponto de início a um nó, pode ser que precisemos fazer um trajeto equivalente a uma fração de uma aresta, e dessa forma adicionamos isso no trajeto, usando para isso um peso (weight) igual à porcentagem de fração tomada vezes o peso total da aresta, sendo assim o peso proporcional à fração. Permitindo assim posicionar pontos no meio das ruas e não somente nos nós. (Conforme imagem abaixo)

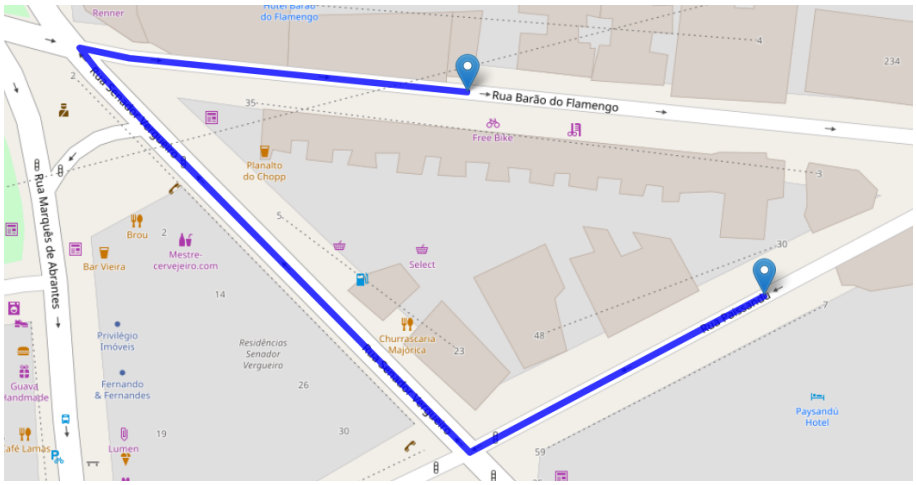

Figura 5: Como podemos ver aqui temos pontos que não são vértices

Sendo também possível fazer uma fração de aresta no meio de uma rua para poder criar um caminho que está dentro de uma mesma aresta. (Conforme abaixo)

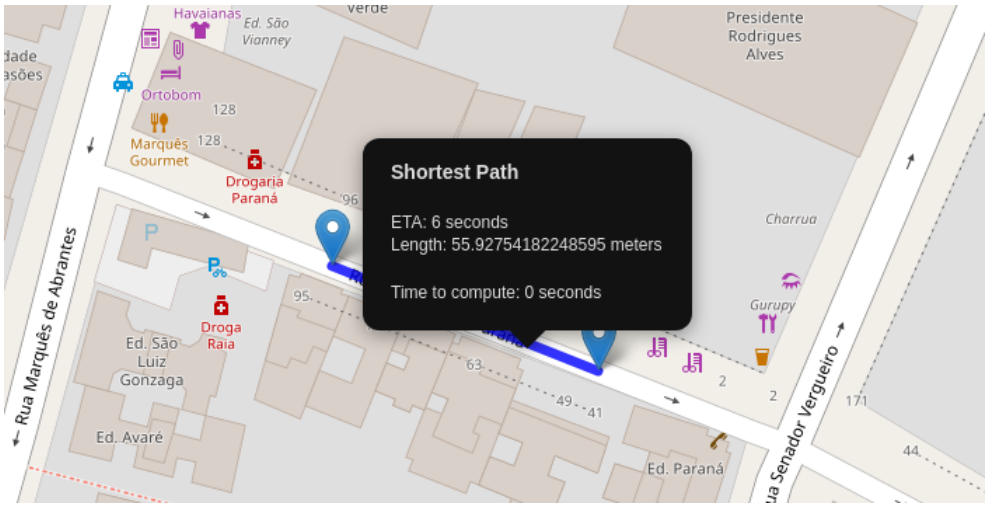

Figura 6: Aqui temos um trajeto que é fração intermediária de uma única aresta.

5.2 Ruas com mão dupla

Quando por exemplo o início está no meio de uma rua de mão dupla, existem duas opções para o nó de início do trajeto e por isso faz-se necessário chamar o algoritmo mais de uma vez caso deseje-se usar o grafo original. Sendo em alguns casos, necessário chamar o algoritmo 4 vezes, caso ambos o início e fim estejam no meio de ruas de mão.
OBS: Uma possibilidade para evitar que o algoritmo seja chamado mais de uma vez, seria usar um grafo alterado G' em que temos os nós do pontos de início e fim adicionados no grafo e fazemos as mudanças necessárias.

Na nossa implementação atual, temos um bug para ruas de mão dupla, por ele acabar percorrendo até o ponto final depois de passar pela mesma rua onde ele está. (Conforme imagem abaixo)

Figura 7: caso de rua com mão dupla, nó de baixo com caminho errado

6 Conclusão

Nosso projeto “Not Waze” é dinâmico com opções de arrastar os pontos de início e fim para onde desejar, permite ao usuário escolher o algoritmo a ser utilizado e disponibiliza o tempo necessário para executar. Sendo assim bastante útil para a análise da eficiência dos diferentes algoritmos e das diferentes heurísticas do A* para os diferentes tipos de caminho.