UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDUARDO AUGUSTO SOBRAL JÚNIOR

HERMES: CÁLCULO DE ROTAS DE TRÂNSITO EM DISPOSITIVOS MÓVEIS

RIO DE JANEIRO
2019

EDUARDO AUGUSTO SOBRAL JÚNIOR

HERMES: CÁLCULO DE ROTAS DE TRÂNSITO EM DISPOSITIVOS MÓVEIS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Jonice Oliveira

RIO DE JANEIRO

2019

CIP - Catalogação na Publicação

S677h

Sobral Júnior, Eduardo Augusto
Hermes: cálculo de rotas de trânsito em dispositivos móveis / Eduardo Augusto Sobral Júnior. -- Rio de Janeiro, 2019.
49 f.

Orientador: Jonice Oliveira.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2019.

1. Tráfego. 2. Rotas. 3. Mapas. 4. Android. I. Oliveira, Jonice, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

EDUARDO AUGUSTO SOBRAL JÚNIOR

HERMES: CÁLCULO DE ROTAS DE TRÂNSITO EM DISPOSITIVOS MÓVEIS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Jonice Oliveira
(PPGI/UFRJ)

Danilo Silva Carvalho
(PPGI/UFRJ)

Rafael Elias de Lima Escalfoni
(CEFET-Friburgo)

Dedico este trabalho às memórias do meu pai, Eduardo Sobral e da minha tia, "Glorinha", que foram essenciais à minha formação. Agradeço meus amigos que ajudaram e incentivaram, Danielle Caled, Julio Inácio.

RESUMO

Este trabalho tem como objetivo encontrar soluções que melhorem o trânsito nas grandes cidades. Este trabalho apresenta a arquitetura do Hermes – um aplicativo de mobilidade urbana – e a arquitetura do módulo de cálculo de rotas. A performance do cálculo local (realizado pelo próprio dispositivo) da rota foi analisado comparando os resultados obtidos entre os algoritmos implementados. Este trabalho implementa dois algoritmos de busca de menores caminhos – Dijkstra e A* - no sistema para dispositivos móveis "Android". No caso do algoritmo A*, este trabalho utiliza três heurísticas diferentes. Este trabalho considerou cinco variáveis para analisar a performance do aplicativo: o par algoritmo e heurística, a quantidade de memória utilizada, tempo de execução , e distância entre os pontos inicial e final da rota, com o objetivo de responder qual o melhor algoritmo a ser utilizado e qual a quantidade de memória apropriada.

Palavras-chave: tráfego. rotas. mapas. android.

ABSTRACT

This work aims to find solutions that improve traffic in big cities. This work presents the Hermes architecture - an urban mobility application - and the architecture of the route calculation module. The performance of the local calculation (performed by the device itself) of the route was analyzed comparing the results obtained among the algorithms implemented. This work implements two algorithms of search of smaller paths - Dijkstra and A * - in the system for mobile devices "Android". In the case of the algorithm A *, this work uses three different heuristics. This work uses five variables to analyze the performance of the application: the algorithm and heuristic pair, the amount of memory used, execution time, and distance between the start and end points of the route, in order to answer which algorithm to use and how much memory is appropriate.

Keywords: traffic. route. map. android.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de Grafo 14
Figura 2 – Execução do Algoritmo de Dijskstra 18
Figura 3 – Execução do Algoritmo A* 19
Figura 4 – Geóide 22
Figura 5 – Técnica de tecelagem 23
Figura 6 – Processo de carregamento dos vértices 28
Figura 7 – Atividades do aplicativo, com setas indicando as transições de telas . . 31
Figura 8 – Tela da atividade principal 32
Figura 9 – Caminho x Percurso x Algoritmo 34
Figura 10 – Caminho x Tempo x Cache para Dijkstra 35
Figura 11 – Caminho x Tempo x Cache para A1 36
Figura 12 – Caminho x Tempo x Cache para A2 37
Figura 13 – Caminho x Tempo x Cache para A3 38
Figura 14 – Comparação entre algoritmos 40

LISTA DE TABELAS

Tabela 1 – Análise do cache para Dijkstra. 34
Tabela 2 – Análise do cache para A1 36
Tabela 3 – Análise do cache para A2. 37
Tabela 4 – Análise do cache para A3. 38
Tabela 5 – Comparação entre os métodos. 39
Tabela 6 – Tempo de Resposta (em segundos) 43
Tabela 7 – Coordenadas dos pontos dos percursos (em microgrаus) 48

LISTA DE QUADROS

Quadro 1 – Exemplo de Lista de Adjacências 15
Quadro 2 – Rotas Usadas na Comparação entre apps 43
Quadro 3 – Comparação entre aplicativos 44
Quadro 4 – Valores padrão do atributo *oneway* 48

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
NA	Não Aplicável
GIS	Geographical Information System (Sistema de informação Geográfica)
WGS84	World Geodetic System 1984
ODBL	Open Database License
CSV	Comma Separated Values
MOBAC	Mobile Atlas Creator
OSM	OpenStreetMaps
Pç	Praça
Compos	Compositor

SUMÁRIO

1 INTRODUÇÃO 12
1.1 APRESENTAÇÃO 12
1.2 MOTIVAÇÃO 12
1.3 OBJETIVO 12
1.4 ESTRUTURA DO TRABALHO 12
1.5 CONCLUSÃO 13

2 ARCABOUÇO CONCEITUAL 14
2.1 GRAFOS 14
2.1.1 Algoritmos de Busca 15
2.1.2 Operações Auxiliares 15
2.1.3 Dijkstra 17
2.1.4 A* 18
2.2 OPENSTREETMAPS(OSM) 20
2.3 SISTEMAS DE INFORMAÇÃO GEOGRÁFICA(GIS) 21
2.3.1 Representação da Terra 21
2.3.2 Técnica de Tecelagem 22
2.4 SISTEMA OPERACIONAL ANDROID 22
2.5 BIBLIOTECAS E FERRAMENTAS 24
2.5.1 PostgreSQL 24
2.5.2 Osmdroid 24
2.5.3 Osmosis 24
2.5.4 MOBAC 24
2.6 CONCLUSÃO 25

3 IMPLEMENTAÇÃO DO APLICATIVO 26
3.1 BANCO DE DADOS 26
3.1.1 Origem dos Dados 26
3.1.2 Transformações 26
3.1.3 Formato Final 26
3.1.4 Código de Construção do Banco 27
3.2 GRAFO 27
3.3 ALGORITMOS 29
3.3.1 Análise de Complexidade 29
3.4 APLICATIVO 30
3.4.1 *MapActivity* 30

3.4.2 ***SettingsActivity*** **31**
3.4.3 **Conclusão** **32**

4 **ANÁLISE DE PERFORMANCE** **33**
4.1 PARÂMETROS DE TESTES 33
4.2 VALIDAÇÃO DE HEURÍSTICA 33
4.3 ANÁLISE DO CACHE 33
4.3.1 **Dijkstra** **34**
4.3.2 **A* com Heurística A1** **35**
4.3.3 **A* com Heurística A2** **36**
4.3.4 **A* com Heurística A3** **37**
4.3.5 **Comparação dos resultados** **39**
4.3.6 **Conclusão** **39**

5 **COMPARATIVO COM OUTROS TRABALHOS** **41**
5.1 CRITÉRIOS DE COMPARAÇÃO 41
5.2 DESCRIÇÃO DOS APLICATIVOS 41
5.2.1 **GPSNavigation** **41**
5.2.2 **Magic Earth** **42**
5.2.3 **Waze** **42**
5.2.4 **GoogleMaps** **42**
5.2.5 **Hermes** **42**
5.3 COMPARAÇÃO 43
5.4 CONCLUSÃO 43

6 **CONSIDERAÇÕES FINAIS** **45**
6.1 DIFICULDADES 45
6.2 MELHORIAS 45
6.3 CONCLUSÃO 46

REFERÊNCIAS **47**

APÊNDICE A – QUADROS EXTRAS **48**

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Este trabalho mostra o funcionamento do aplicativo Hermes, desenvolvido para o sistema operacional Android, que busca de rotas de trânsito para automóveis.Dois algoritmos tradicionais de busca foram aplicados, Dijkstra e A*, em um modelo da malha rodoviária da cidade do Rio de Janeiro. Os resultados da performance em relação ao tempo foram comparados uns com os outros, a fim de determinar qual o melhor a ser usado. Duas outras variáveis também foram analisadas com relação à performance: memória e heurística utilizada no algoritmo A*.

1.2 MOTIVAÇÃO

Nos grandes centros urbanos, problemas com congestionamentos se tornam cada vez maiores, gerando diversos transtornos e impactando a qualidade de vida da população. Sistemas de navegação foram desenvolvidos para auxiliar as pessoas a se movimentar pela cidade. Podem ser citados: Google Maps ou Bing Maps e Waze. Algumas dessas soluções funcionam apenas utilizando internet, outras já oferecem também a busca sem rede. Hermes procura resolver este problema buscando as rotas de forma local, utilizando dados abertos.

1.3 OBJETIVO

O objetivo principal deste trabalho é desenvolver um aplicativo para dispositivos móveis que indique qual o melhor caminho (considerando apenas a distância total do percurso) a ser seguido, sem a necessidade de utilizar planos de dados (internet móvel), utilizando dados abertos. Um objetivo secundário é que este aplicativo possua performance (considerando tempo para encontrar rotas) suficiente para ser utilizado em situações reais.

1.4 ESTRUTURA DO TRABALHO

O trabalho está organizado da seguinte forma:

a) no Capítulo 1 é apresentado um resumo sobre o trabalho e sua estrutura, além de suas motivações;

b) no Capítulo 2 é apresentado uma revisão da literatura e conceitos utilizados na elaboração do trabalho;

c) o Capítulo 3 descreve a estrutura do aplicativo e suas funcionalidades;

d) no Capítulo 4 são descritas análises sobre a performance dos diferentes algoritmos e configurações, a fim de escolher a melhor para ser utilizada;

e) no Capítulo 5 será realizada uma comparação do Hermes com outros aplicativos existentes;

f) no Capítulo 6 descreve as conclusões obtidas no estudo.

1.5 CONCLUSÃO

Neste capítulo foi apresentado um resumo do trabalho, suas motivações e sua estruturação. No próximo capítulo serão abordados os conceitos utilizados para o desenvolvimento e implementação do aplicativo.

2 ARCABOUÇO CONCEITUAL

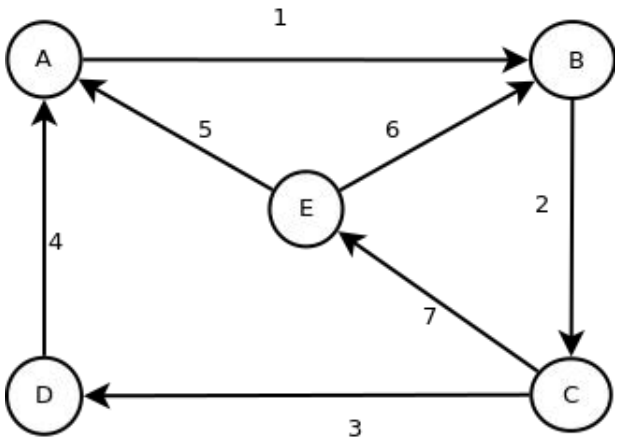
Este capítulo descreve os conceitos utilizados na elaboração deste trabalho. Primeiro, é apresentado o conceito de grafos e descrição dos algoritmos de busca. Após, serão apresentados conceitos de análise geográfica. Por último, será apresentado o sistema Android, sistema operacional utilizado em diversos dispositivos móveis, escolhido como a plataforma para o desenvolvimento do aplicativo.

2.1 GRAFOS

Segundo Bondy e Murty (1976, p. 1), o conceito de grafo surge de abstrações matemáticas que utilizam a ideia de pontos (vértices) e linhas ligando pontos (arestas), simbolizando uma conexão entre esses pontos. Neste trabalho foi utilizado dados de redes de ruas, onde os pontos representam as interseções (ou, algumas vezes, marcos subdividindo uma rua em segmentos menores) e as linhas, as ruas.

Cormen et al. (2009, p. 589) indica como grafos podem ser implementados: dados um grafo G, uma coleção de vértices V, e arestas (pares ordenados de vértices) E, uma implementação por listas de adjacências do grafo G consiste de um vetor de listas de vértices indexado pelos vértices. Uma aresta (u, v) é representada pela existência do vértice v na lista de adjacências do vértice u. Em casos em que o grafo possui custos associados a arestas, o custo de cada aresta é guardado junto com o vértice v na lista de adjacências do vértice u. A Figura 1 contém um exemplo de grafo enquanto o Quadro 1 contém a representação em lista de adjacências para este grafo.

Figura 1 – Exemplo de Grafo



Quadro 1 – Exemplo de Lista de Adjacências

Vértice	**Adjacências**
A	B
B	C
C	D, E
D	A
E	A,B

2.1.1 Algoritmos de Busca

O problema que este trabalho resolve consiste em encontrar um caminho entre dois pontos na cidade do Rio de Janeiro, utilizando a rede de ruas da cidade. Considerando essa rede um grafo, o problema inicial pode ser traduzido em encontrar o caminho de menor custo entre dois pontos em um grafo. Diversos algoritmos foram desenvolvidos com o objetivo de resolver este problema. Este trabalho, entretanto, foca em dois: Dijkstra e A*.

2.1.2 Operações Auxiliares

Na descrição dos algoritmos, serão utilizadas algumas operações auxiliares, para facilitar o entendimento dos mesmos. A operação de relaxamento (relax) foi adaptada de Cormen et al. (2009), para manter consistência de estilo com as outras operações. A seguir elas serão listadas e definidas:

a) `Custo(v)`

   Função que retorna o menor custo conhecido para se chegar a ao vértice v;

b) `AtualizaCusto(v, c)`

   Função que atualiza o menor custo conhecido para se chegar a v para o valor c;

c) `Pai(v)`

   Função que retorna o vértice imediatamente anterior no caminho conhecido de menor custo (pai) do vértice v;

d) `AtualizaPai(v, p)`

   Função que atualiza o pai do vértice v para p;

e) `CustoE(a, b)`

   Função que retorna o custo da aresta que liga o vértice a ao vértice b;

f) `Heuristica(v)`

   Função que retorna uma estimativa do custo do caminho de menor custo entre v e o vértice final;

g) `Relax(a, b)`

Pseudocódigo:

```
Caso Custo(b) > Custo(a) + CustoE(a, b) {
    AtualizaCusto(b, Custo(a) + CustoE(a, b))
    AtualizaPai(b, a)
}
```

Função que relaxa a aresta que liga o vértice a ao b;

h) `CustoH(v)`

Função que retorna o custo estimado do caminho entre os vértices de inicial e final que passa pelo vértice v;

i) `AtualizaCustoH(v, c)`

Função que atualiza o custo estimado do caminho entre os vértices de inicial e final que passa por v para c;

j) `RelaxH(a, b)`

Pseudocódigo:

```
Caso CustoH(b) > Custo(a) + CustoE(a, b) + Heuristica(b) {
    AtualizaCusto(a, Custo(a)+CustoE(a, b))
    AtualizaPai(b, a)
    AtualizaCustoH(Custo(b) + Heuristica(b))
}
```

Relaxa uma aresta, quando uma heurística é utilizada;

k) `Caminho(I,F)`

Pseudocódigo:

```
Se I for igual a F {
    retornar sequencia vazia
}
senão {
    retornar Caminho(I, Pai(F)) + F
}
```

Encontra o caminho como uma sequência de vértices entre um vértice inicial e um final, desde que tal caminho exista.

2.1.3 Dijkstra

Conforme Cormen et al. (2009), dado um grafo G, um vértice inicial I, e um vértice final D, um conjunto de arestas E com pesos não-negativos, o algoritmo de Dijkstra sempre calcula o caminho de menor custo entre I e D. O pseudocódigo a seguir é baseado no pseudocódigo apresentado por Cormen et al. (2009), com as seguintes modificações:

a) O algoritmo original procura o menor caminho entre o vértice inicial para todos os vértices. Como, neste trabalho, o interesse é encontrar o menor caminho para um vértice específico, o algoritmo para e retorna ao encontrá-lo;

b) A coleção Fronteira (Q na versão original) é dinâmica, sendo construída durante a execução do algoritmo. Na versão original, todos os vértices pertencem a ela no início do algoritmo. Entretanto, o tamanho dessa coleção possui influência na performance do algoritmo (essa questão será discutida na seção 3.3). Construí-la aos poucos permite mantê-la o menor possível e diminuir o impacto que operações (adicionar e remover mínimo) realizadas sobre a mesma causam ao algoritmo.

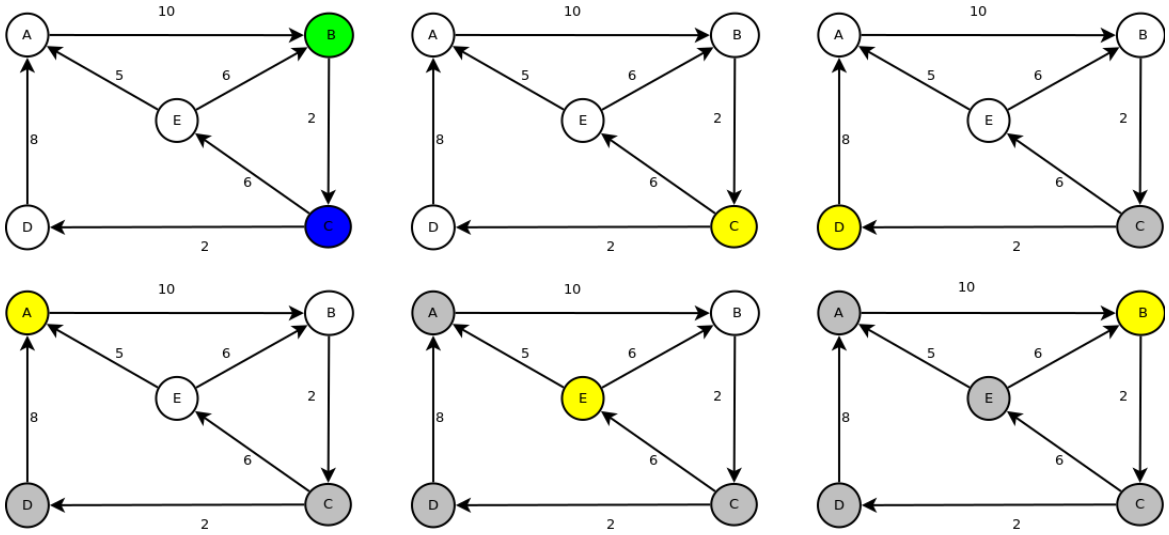
2.1.3.1 Pseudocódigo Dijkstra

```
1  - Iniciar coleções de vértices vazias: Visitados e Fronteira.
2  - Guardar os vértices I, inicial, e F, final.
3  - AtualizaCusto(V, infinito), em todos os vértices.
4  - AtualizaCusto(I,0).
5  - Adicionar I à Fronteira.
6  - Enquanto a Fronteira não for vazia {
7  -   Encontrar VA, vértice da Fronteira cujo Custo(VA) seja o menor.
8  -     Remover VA da Fronteira.
9  -     Se VA for igual a F, parar.
10 -     Adicionar VA à coleção Visitados.
11 -     Para cada vértice (VV) na adjacência de VA {
12 -         Caso VV já estiver em Visitados {
13 -             pular este vértice e continuar no próximo.
14 -         }
15 -         Relax(VA, VV).
16 -         Caso VV não estiver presente na fronteira {
17 -             Adicionar VV à Fronteira.
18 -         }
19 -     }
20 - }
21 - Retornar Caminho(I, F) e Custo(F).
```

2.1.3.2 Exemplo Dijkstra

Um exemplo de execução do algoritmo pode ser observado na Figura 2. Os vértices azul e verde são, respectivamente, os vértices inicial e final. Os vértices amarelos são aqueles que estão sendo visitados em cada etapa. Os vértices cinzas são aqueles que já foram visitados.

Figura 2 – Execução do Algoritmo de Dijskstra



2.1.4 A*

De acordo com Russell e Norvig (1995), A* é um algoritmo de busca informada, que utiliza heurísticas pra diminuir o espaço de busca e encontrar uma solução em menos passos, se comparado com algoritmos não-informados. Russell e Norvig (1995, p. 96) apresentam também o conceito de heurísticas admissíveis, definidas como aquelas nas quais o custo estimado de se chegar ao vértice final é sempre menor ou igual ao custo real. No caso de uma heurística ser admissível, é garantido que a solução encontrada é o menor caminho possível e uma demonstração pode ser encontrada em Russell e Norvig (1995, p. 99).

2.1.4.1 Pseudocódigo

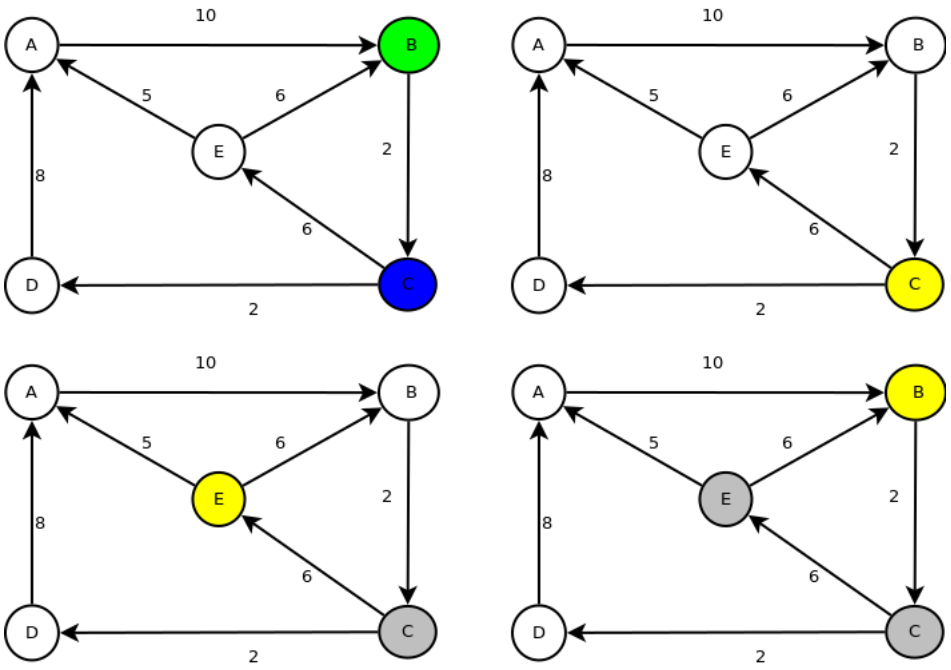
```
1  - Iniciar coleções de vértices vazias: Visitados e Fronteira.
2  - Guardar os vértices I, inicial, e F, final.
3  - AtualizaCusto(V, infinito), em todos os vértices.
4  - AtualizaCusto(I, 0).
5  - AtualizaCustoH(I, Heuristica(I)).
6  - Adicionar I à Fronteira.
```

```
7  - Enquanto a Fronteira não for vazia {
8  -     Encontrar VA, vértice da Fronteira cujo CustoH(VA) seja o menor.
9  -     Remover VA da Fronteira.
10 -     Se VA for igual a F, parar.
11 -     Adicionar VA à coleção Visitados.
12 -     Para cada vértice (VV) na adjacência de VA {
13 -         Caso VV já estiver em Visitados {
14 -             pular este vértice e continuar no próximo.
15 -         }
16 -         RelaxH(VA, VV).
17 -         Caso VV não estiver presente na fronteira {
18 -             Adicionar VV à Fronteira.
19 -         }
20 -     }
21 - }
22 - Retornar Caminho(I, F) e Custo(F).
```

2.1.4.2 Exemplo A*

A Figura 3 apresenta um exemplo de execução do algoritmo A*. Os vértices azul e verde são, respectivamente, os vértices inicial e final. Os vértices amarelos são aqueles que estão sendo visitados em cada etapa. Os vértices cinzas são aqueles que já foram visitados.

Figura 3 – Execução do Algoritmo A*



Comparando os resultados, pode ser observado que enquanto o algoritmo de Dijkstra precisou de 6 passos para encontrar a resposta correta, o algoritmo A* precisou de apenas 4. Essa redução no número de vértices visitados se traduz em uma redução no tempo de execução do algoritmo.

2.2 OPENSTREETMAPS(OSM)

Segundo a página do próprio projeto Openstreetmap Foundation (2016), o OpenStreetMaps é um projeto colaborativo de mapeamento mundial, possuindo informações disponibilizadas por grandes empresas, como a Microsoft. Os dados são disponibilizados de acordo com a licença ODBL, disponível em Openstreetmap Foundation (2018). Este projeto foi de importância essencial para este trabalho pois todos os dados foram obtidos a partir dele. As informações são armazenadas em três tipos de dados:

a) nós - são compostos pelo menos por um id e um par latitude/longitude. Representam um ponto específico na superfície da Terra. Podem ser usados para representar características geográficas isoladas (uma fonte) ou ser membros de um grupo (sendo o grupo um caminho ou uma relação);

b) caminhos - são listas ordenadas que possuem entre 2 e 2000 nós que definem uma linha com vários segmentos. Podem ser abertas e representar características como rios, estradas, ruas. Podem ser fechadas e representar áreas contidas num polígono, como floresta, campos e construções;

c) relações - uma estrutura com diversos propósitos que estabelece uma relação entre outros elementos (sejam nós, caminhos ou outras relações). Podem ser usadas para representar rotas (de ônibus, por exemplo), áreas com mais de 2000 nós ou que possuam buracos.

Cada exemplar de um dos tipos de dados pode possuir atributos que expandem seu significado e adicionam informações relevantes. Para cada tipo de dado, foram considerados os seguintes atributos:

a) nós :
   - id – Identificador único de cada nó;
   - latitude – latitude do nó;
   - longitude – longitude do nó.

b) caminhos :
   - id – Identificador único de cada caminho;
   - *oneway* - Indica o sentido da via. Caso seja igual a *yes*, indica que a via segue na ordem da lista do caminho. Caso seja igual a *no*, indica que a via segue nos dois sentidos. Caso seja igual a -1, indica que a via segue apenas no sentido inverso ao da lista;

– *highway* – Este atributo está presente para indicar que o caminho é uma via de trânsito. O valor do atributo indica o tipo de via (rua, estrada, via expressa, rua principal, rua secundária, ...). Caso o atributo *oneway* não esteja presente, o atributo *highway* indica um valor padrão para *oneway*. O Quadro 4 contém uma tabela para valores do atributo *highway* utilizados e o valor padrão correspondente para o atributo *oneway*.

No contexto deste trabalho, os vértices de um grafo são representados pelos nós e as arestas podem ser inferidas pelos caminhos. Por exemplo: nós A, B, C e caminho: A, B, C produz as arestas: AB e BC. Caso *oneway* tenha valor *no*, as arestas CB e BA também fazem parte do grafo.

2.3 SISTEMAS DE INFORMAÇÃO GEOGRÁFICA(GIS)

Huisman e de By (2009, p. 32)apresentam uma definição na qual, para ser considerado GIS, é necessário que um sistema possua quatro funcionalidades ao gerir informação georreferenciada: captura e preparação de dados, gerenciamento de dados (incluindo armazenamento e manutenção), manipulação e análise, e apresentação dos dados.

Apesar de possuir elementos de um GIS, falta ao aplicativo funções que permitam que seja considerado um GIS de acordo Huisman e de By (2009), devido ao seu escopo limitado (considerando as características implícitas citadas por eles, como suporte à entrada e transformações entre sistemas de coordenadas). Contudo, este trabalho necessita utilizar técnicas e dados normalmente encontrados em GIS, como ao trabalhar com latitudes e longitudes, e em especial com relação ao banco de dados.

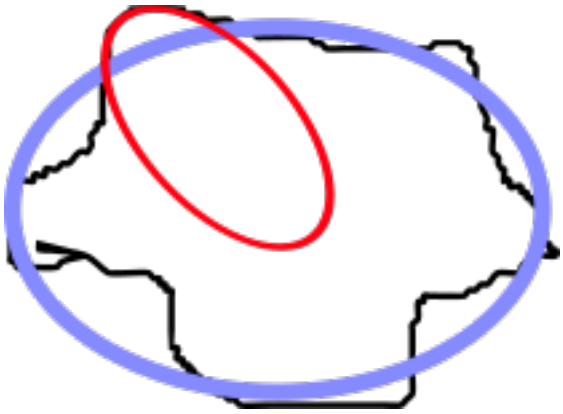
2.3.1 Representação da Terra

Um dos maiores desafios ao se trabalhar com informações georreferenciadas está em como representar a superfície da Terra. Apesar de seu formato elipsóide, esse formato não é perfeito, havendo irregularidades que provocam distorções nas tentativas de modelá-la, como afirmado por Ordnance Survey (2018) e Huisman e de By (2009). Devido a essas distorções, diversos sistemas de coordenadas foram propostos e muitos ainda são utilizados, dependendo da região e problema que se deseja resolver, aumentando a adequação do modelo à realidade em uma região ao custo de causar detrimento em outra. A Figura 4, adaptada de Ordnance Survey (2018), ilustra um caso, onde o elipsóide azul oferece uma melhor aproximação global e o elipsóide vermelho oferece uma melhor aproximação local.

Ordnance Survey (2018) exemplifica alguns mitos (que acabam gerando dificuldades) ao se trabalhar com dados geográficos:

a) um ponto possui uma latitude e longitude únicos – Devido aos diferentes sistemas utilizados, cada um deles pode representar uma mesma localidade com coordenadas diferentes;

Figura 4 – Geóide



b) as coordenadas de um ponto não mudam – Devido ao movimento das placas tectônicas, pontos podem se mover até 10 centímetros por ano (ORDNANCE SURVEY, 2018), o que, acumulado, podem provocar um deslocamento de alguns metros em décadas, mesmo dentro do mesmo sistema;

c) existem fórmulas matemáticas exatas para mudar entre sistemas de coordenadas – A dificuldade de conversão é tamanha que aproximações são usadas para a conversão.

Devido a esses fatores, deve-se ter cuidado ao se trabalhar com dados geográficos, a fim de evitar confusões e garantir que os dados estejam corretos.

2.3.2 Técnica de Tecelagem

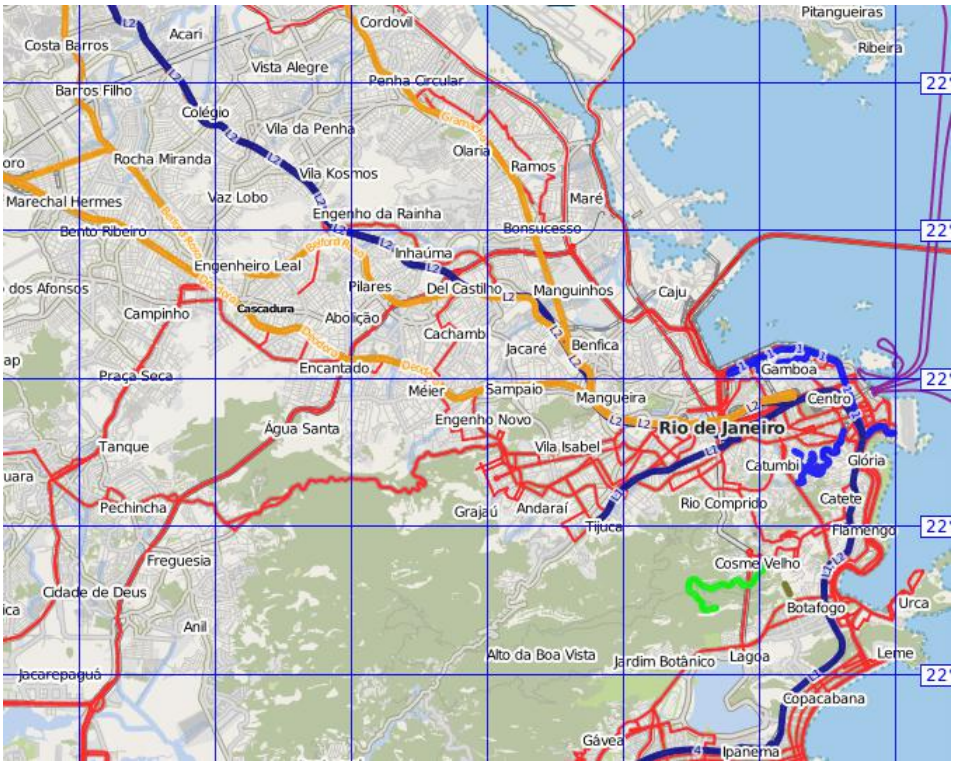
"Tecelagem é uma partição do espaço em células mutuamente exclusivas que, juntas, compõem completamente o espaço de estudo"(HUISMAN; de By, 2009, p. 85, tradução do autor). [1] Essa técnica é utilizada no aplicativo tanto para a exibição do mapa e quanto no seu banco de dados, permitindo trabalhar com apenas um recorte dos dados de cada vez e limitando o uso de memória. Uma representação visual da técnica pode ser observada na Figura 5. As linhas verticais e horizontais azuis representam os limites entre cada célula.

2.4 SISTEMA OPERACIONAL ANDROID

Android é um sistema operacional que atualmente é desenvolvido pela Google com o objetivo de ser utilizado em dispositivos móveis, sendo o sistema escolhido para o desenvolvimento do aplicativo. Segundo Android Developers (2018), página com a documentação oficial, o sistema possui quatro tipos de componentes fundamentais:

[1] A tessellation (or tiling) is a partitioning of space into mutually exclusive cells that together make up the complete study space.

Figura 5 – Técnica de tecelagem



a) Atividades – Representam a interface com o usuário. Cada atividade representa uma única tela. Por exemplo, um aplicativo pode ter uma tela de menu, uma tela de configuração, e várias telas diferentes, cada uma relacionada a uma função do aplicativo;

b) Serviços – São componentes que representam operações longas ou processamento de trabalhos remotos. Não apresentam interface (telas), nem requerem interação com o usuário, podendo continuar sua execução mesmo se o usuário passar a utilizar outro aplicativo;

c) Provedores de Conteúdo – São componentes que gerenciam um conjunto de dados persistente do aplicativo. Servem como interface padrão para que outros componentes, inclusive de outros aplicativos (se o provedor permitir), possam acessar (ou modificar, se permitido) esses dados;

d) Receptores de Transmissão – São utilizados para a transmissão e recepção de eventos globais do sistema (exemplos: início de chamada, telefone com pouca bateria, localização obtida, ou eventos definidos outros aplicativos), não se limitando ao aplicativo.

Este trabalho utilizou apenas atividades em sua produção. A busca de rotas, apesar de poder ser delegada a um serviço, foi, no entanto, delegada uma *thread*. Esta escolha foi devido à simplicidade de utilização da mesma em comparação com um serviço,

considerando a necessidade da aplicação.

O grafo representante da malha rodoviária poderia ser representado com um provedor de conteúdo, entretanto esta decisão poderia ocasionar processamento desnecessário que impactaria negativamente o desempenho do aplicativo. Novas versões, no entanto, poderiam utilizá-lo, e comparar com os resultados obtidos neste trabalho para determinar de forma objetiva a melhor alternativa.

2.5 BIBLIOTECAS E FERRAMENTAS

2.5.1 PostgreSQL

PostgreSQL é um banco de dados relacional. Possui uma extensão chamada PostGIS, que facilita a manipulação de dados geográficos. Foi utilizada para realizar um pré-processamento dos dados a fim de remover dados desnecessários e organizar o banco para o formato final utilizado.

2.5.2 Osmdroid

Osmdroid é uma biblioteca substituta para a API de mapas do *Google* para o Android. Foi escolhida por possuir um sistema para exibir mapas, com suporte para desenho de ícones e formas geométricas sobre as imagens dos mapas. Ela permite escolher onde armazenar as imagens utilizadas nos mapas, podendo estas estarem armazenados localmente ou na rede. Sua escolha em detrimento à API oficial se deve à flexibilidade e poder que a mesma oferece ao desenvolvedor, sendo que este projeto seria mais difícil de implementar utilizando a API do *Google*.

2.5.3 Osmosis

Osmosis é uma aplicação para o processamento de informações de arquivos OSM. Essa ferramenta foi utilizada para realizar a importação do banco de dados em seu formato original (arquivo OSM) para um banco de dados relacional, onde a informação pôde ser processada.

2.5.4 MOBAC

MOBAC é uma ferramenta para geração de imagens de mapas, compatível com o osmdroid. Foi a ferramenta utilizada para gerar as imagens utilizadas nos mapas do aplicativo.

2.6 CONCLUSÃO

Neste capítulo foi abordado a base teórica e a apresentação de conceitos que serão desenvolvidos no próximo capítulo, mostrando sua aplicação em um caso real.

3 IMPLEMENTAÇÃO DO APLICATIVO

Neste capítulo será abordado detalhes de implementação do aplicativo, sendo consideradas escolhas realizadas durante o processo de construção do mesmo.

3.1 BANCO DE DADOS

3.1.1 Origem dos Dados

Para poder encontrar rotas de trânsito na cidade do Rio de Janeiro, foi necessário obter os dados da malha rodoviária da cidade. Foi utilizado um arquivo OSM obtido no endereço <http://mapzen.com/data/metro-extracts> como base para geração do banco de dados utilizado no aplicativo (obtido em 16 de janeiro de 2016).

3.1.2 Transformações

O arquivo OSM foi importado para um banco de dados PostgreSQL com a extensão postgis (para processamento de dados geográficos) utilizando a ferramenta osmosis. Os nós foram filtrados para que somente fossem utilizados aqueles cujas latitudes estavam entre: $-23,75$ e $-22,77$, e longitude entre: $-43,79$ e $-43,15$ (região que compreende o município do Rio de Janeiro). Cada nó foi mapeado como um vértice do grafo. Cada vértice guardou as informações de latitude, longitude e id do nó.

A primeira etapa consistiu em selecionar os caminhos que seriam utilizados. Caminhos com a etiqueta *highway* com os valores: *motorway, motorway_link, trunk, trunk_link, primary, primary_link, secondary, secondary_link, tertiary, tertiary_link, residential, unclassified e living_street* foram utilizados para criar o grafo.

Para cada caminho, foi verificada a existência da etiqueta *oneway*. Caso ela não estivesse presente, ela foi criada com o valor padrão presente no Tabela 7 para o tipo de caminho (de acordo com a etiqueta *highway*).

Para aqueles caminhos cujo atributo *oneway* indicava mão única, as arestas foram criadas ligando o vértice ao seu seguinte imediato. Naqueles que indicavam mão dupla foi criada uma aresta adicional ligando cada vértice ao seu anterior imediato no caminho. Nos casos em que o atributo *oneway* possuía valor igual a -1, como indicado na documentação, as arestas foram criadas apenas no sentido inverso (ligando cada vértice ao seu anterior). Vértices que não foram utilizados em nenhum caminho foram excluídos.

3.1.3 Formato Final

Os vértices foram particionados de duas formas: por células do mapa, guardados no diretório *database*, e por *id*, guardados no diretório *id_database* .

As células foram particionadas utilizando intervalos de $0,02°$ para latitudes e $0,01°$ para longitudes, com a primeira célula começando no ponto de latitude $-90°$ e longitude $-180°$ (a Figura 5 ilustra uma partição, mas com intervalos maiores). Na pasta database, cada arquivo representa uma célula, cada registro no arquivo um vértice, e é nomeado de acordo com a célula que representa, da seguinte forma: considerando o ponto de menor latitude e longitude da célula, o nome do arquivo é $(latitude/0,02°)\_(longitude/0,01°)$. Dessa forma, é fácil descobrir em qual arquivo um vértice está presente, dados sua latitude e longitude, Cada arquivo está em um formato CSV (com colunas separadas por espaço), sendo, a primeira coluna, o id de cada vértice, a segunda, a latitude e, a terceira, a longitude. As colunas, a partir da quarta, representam os ids dos vértices adjacentes, como numa lista de adjacência.

Cada arquivo representa uma partição dos vértices do grafo, com relação aos ids. Cada vértice está registrado em um arquivo cujo nome é igual ao id do vértice sem os dois últimos dígitos (vértice de id 1234 estaria no arquivo 12). Cada arquivo está no formato CSV no qual as colunas são, respectivamente, o *id*, a latitude, e a longitude. Um arquivo com os dados pode ser obtido no endereço <https://s3.amazonaws.com/tcchermes/roaddb.zip>.

3.1.4 Código de Construção do Banco

Códigos auxiliares que realizam os processos descritos acima estão disponíveis em <https://s3.amazonaws.com/tcchermes/code.zip>. Há quatro arquivos:

a) *create_db* – Um *shell script* que cria um banco de dados postgresql, instala a extensão postgis e por último executa o programa osmosis para importação do banco de dados;

b) *db.py* – Um *script python* que cria o banco de dados no formato final;

c) *id_db.py* – Um script python que cria o banco de dados de ids, no formato final;

d) *uniform_db.sql* – Um script sql que uniformiza o banco de dados, adicionando o atributo *oneway* conforme especificado anteriormente, elimina nós que não serão usados, e transforma entre o formato interno em um formado intermediário, utilizado por db.py e id_db.py para a criação do banco de dados final.

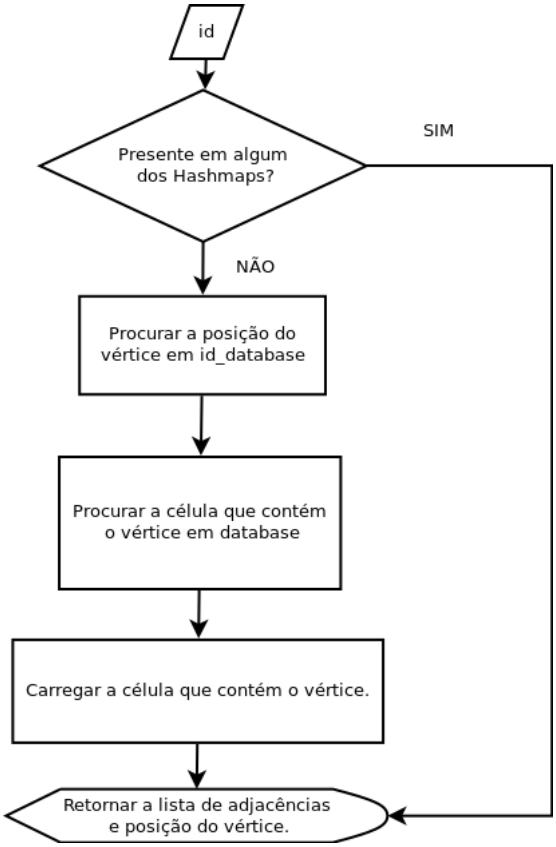
3.2 GRAFO

Graph é uma classe implementada para representar a abstração de um grafo. A implementação é similar a uma lista de adjacências, onde um vértice pode ser encontrado pelo seu id, e sua posição junto com uma lista é retornada. Um detalhe de implementação importante é que vértices podem ser carregados do armazenamento de longo prazo ou eliminados da memória RAM de forma transparente ao utilizador. Possui também a opção de procurar um vértice cuja posição seja a mais próxima de uma coordenada geográfica.

O custo de uma aresta é a distância entre os pontos (em metros), sendo calculado considerando a latitude e longitude do vértice inicial e final. O cálculo é realizado utilizando uma função da biblioteca *osmdroid*, que utiliza a fórmula de haversine para encontrar a distância entre os pontos, considerando a Terra como uma circuferência com raio de 6378137 metros. Este trabalho considerou que o erro entre a realidade e o modelo seria pequeno o suficiente para ser ignorado.

Apresenta um vetor de *HashMaps*, cada elemento representando uma célula do mapa e cada *HashMap* contendo os vértices de uma célula. Quando um vértice é buscado, primeiro é verificado se ele está presente no vetor. Caso contrário, a posição do vértice é procurada em id_database, e depois usada para carregar a célula onde o vértice se encontra. Por fim, o vértice e sua lista de adjacências é retornada. A Figura 6 ilustra esse processo.

Figura 6 – Processo de carregamento dos vértices



Ao carregar uma célula, uma função de espalhamento é utilizada para determinar onde no vetor de células ela será carregada, substituindo a célula antiga. Para poucas células, a troca constante torna-se um gargalo, tornando a execução mais lenta. Para muitas células, a procura no vetor se torna onerosa, e um dos objetivos deste trabalho é justamente encontrar um meio termo que permita diminuir o tempo de execução da busca.

3.3 ALGORITMOS

Nesta seção serão discutidas algumas questões relacionadas à implementação dos algoritmos e como as decisões tomadas influenciaram a performance. As classes *PriorityQueue* e *HashMap*, parte do *Java Collections Framework* e são que implementações, respectivamente, de uma fila de prioridade e de uma tabela hash, foram utilizadas para a implementação dos algoritmos.

3.3.1 Análise de Complexidade

Na implementação são utilizadas implementações contidas no *Java Collections Framework*. O algoritmo implementado utiliza uma coleção *nodeInfo*, que é implementada utilizando a classe *HashMap* (na qual adicionar e recuperar itens possui tempo amortizado $O(1)$) sendo preenchida conforme o algoritmo encontre novos vértices, enquanto a coleção Fronteira é implementada utilizando a classe *PriorityQueue* (operações de adicionar elemento e remover mínimo possui tempo $O(log(n))$, operação de remover elemento arbitrário possui $O(n)$).

A coleção *nodeInfo* é indexada pelo id do vértice, guardando não apenas o mesmo, como também o custo atual conhecido de visitação ao vértice, o *pai* no caminho de menor custo, se o vértice pertence à coleção Visitados e se o vértice ainda está na Fronteira. Isso permite que as operações *Custo(v)*, *Pai(v)* e *AtualizaPai(v, p)* e os testes nas linhas 12 e 16 (caso Djkistra) ou 13 e 17 (caso A*) possam ser realizados com tempo $O(1)$, além de permitir que a linha 3 não precise ser executada(vertices que não estejam presentes em *nodeInfo* possuem tanto custo quanto custo estimado infinito e não possuem vértice pai).

As funções *AtualizaCusto(v, c)* e *AtualizaCustoH(v, c)* são executadas em tempo $O(n)$, pois apesar da atualização do custo nas informações ter custo $O(1)$, é necessário atualizar a fila de prioridade (reordenar) com o custo atual do vértice, sendo necessário remover e reinserir o vértice. Como indicado anteriormente, inserir o vértice possui custo $O(log(n))$, contudo a operação de remover um vértice arbitrário tem custo $O(n)$.

CustoE(v) é calculado utilizando uma fórmula de haversine, a partir dos dados do vértice, possuindo tempo $O(1)$. A função Heuristica(v) pode ser representada por três funções diferentes, dependendo da heurística utilizada. Sendo D(A,B) a distância dada pela fórmula de haversine dois vértices, sendo V o vértice considerado, I o vértice inicial e F o vértice final, as heurísticas podem ser:

a) A1, sendo $A1(V) = D(V, F)$;

b) A2, sendo $A2(V) = D(V, F)^2$;

c) A3, sendo $A3(V) = D(V, F)^2/D(I, F)$.

Dessa forma, todas as três heurísticas possuem tempo $O(1)$ para serem calculadas.

Com isso, podemos calcular a complexidade de ambos os algoritmos. No caso de dijkstra, temos: linhas 1 a 5 são executadas apenas 1 vez, possuindo $O(1)$. A linha 6 cria um bloco de repetição que é executado até n (número de vértices) vezes. Linhas 7, 9, 10 tem $n * O(1) = O(n)$. Linha 8 tem $n * O(log(n))$. Linha 11 abre um bloco de repetição que é executado até m (número de arestas) vezes. Linha 15 tem $m * O(n) = O(m * n)$. Linhas 13 e 16 tem $m * O(1) = O(m)$. Linha 17 tem $m * O(log(n)$. Linha 21 tem $O(n)$.

Somando tudo, temos $O(1)+O(n)+n*O(log(n))+m*O(n)+O(m)+m*O(log(n) = O(m * n)$

No caso do A*, temos: linhas 1 a 6 são executadas apenas 1 vez, possuindo $O(1)$. Linha 7 cria um bloco de repetição que é executado até n (número de vértices) vezes. Linhas 8, 10, 11 tem $n * O(1) = O(n)$. Linha 8 tem $n * O(log(n))$. Linha 12 abre um bloco de repetição que é executado até m (número de arestas) vezes. Linha 16 tem $m * O(n) = O(m * n)$. Linhas 14 e 17 tem $m * O(1) = O(m)$. Linha 18 tem $m * O(log(n)$. Linha 22 tem $O(n)$.

Somando tudo, temos $O(1)+O(n)+n*O(log(n))+m*O(n)+O(m)+m*O(log(n) = O(m * n)$

3.4 APLICATIVO

Nesta seção serão descritas as atividades do aplicativo, da perspectiva do usuário, indicando quais ações são possíveis de serem realizadas em cada atividade.

3.4.1 *MapActivity*

MapActivity é a principal atividade do aplicativo, sendo a primeira a ser apresentada ao iniciar o aplicativo. Em sua primeira execução, procura realizar o *download* de um arquivo contendo as imagens do mapa geradas pelo MOBAC. Caso não seja possível, tenta novamente sempre que for iniciado. Apresenta o mapa do Rio de Janeiro, mostrando o local onde o utilizador está presente, como pode ser observado na Figura 8. Utilizada para visualização do mapa e da rota encontrada. Ações possíveis:

a) buscar Rota – é necessário tocar a tela. O primeiro toque seleciona um ponto de origem para busca. O segundo toque seleciona um ponto de destino e inicia a busca;

b) mover o mapa – tocar em um posto e arrastar;

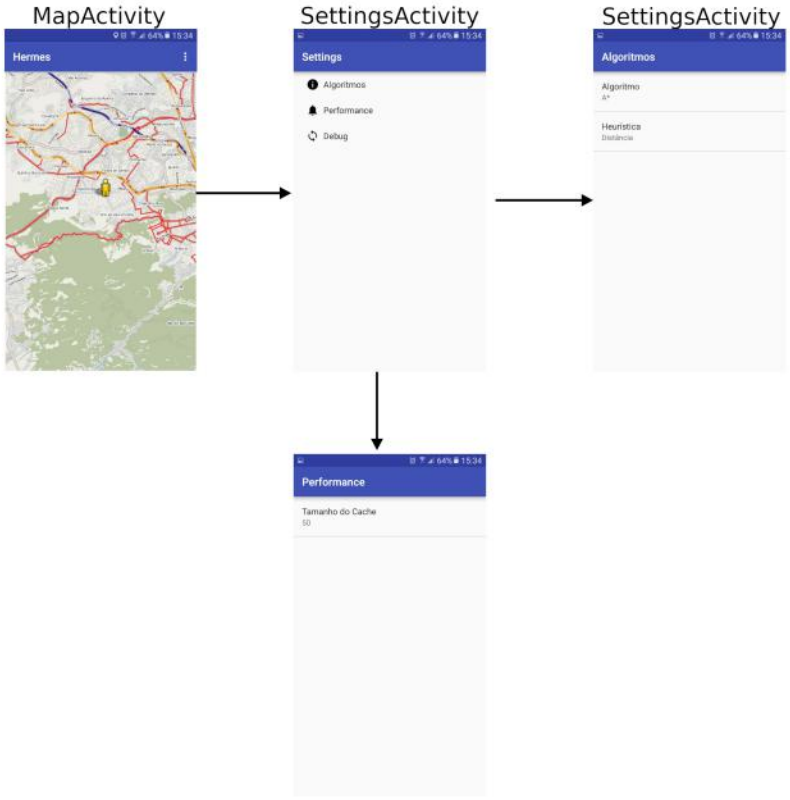
c) aproximar o mapa (*zoomin*) – tocar em dois pontos e afastar os dedos;

d) afastar o mapa (*zoomout*) – tocar em dois pontos diferentes e aproximar os dedos;

e) no menu -

   – *Settings* - ir para a *SettingsActivity*;

Figura 7 – Atividades do aplicativo, com setas indicando as transições de telas



– *FindPath* - procura a menor rota utilizando as posições configuradas na área *Debug* em *SettingsActivity*;
– *Credits* - Uma atividade na qual os créditos pelas bibliotecas e ferramentas utilizadas no aplicativo é reconhecido.

3.4.2 *SettingsActivity*

Atividade que permite ajustar as configurações de execução da busca. Suas telas podem ser vistas na Figura 7 As opções de configuração são:

a) Algoritmo
   – algoritmo – Permite escolher entre o algoritmo de Djikistra ou A* ao calcular a rota;
   – heurística – Determina qual heurística a ser utilizada no algoritmo A*. Há três heurísticas disponíveis. Distância, distância ao quadrado (D2), e distância ao quadrado dividido pela distância entre os pontos inicial e final (D2M).

Figura 8 – Tela da atividade principal

b) Performance - Tamanho do cache são quantas células do mapa são mantidas em memória, no máximo. Um numero maior usa mais memória, porém há menor acesso ao disco;

c) Debug - Contém opções auxiliares, para ajudar na verificação dos resultados;

- *Show Results Data* – se marcado, ao terminar de procurar a rota, exibe: distância entre os pontos origem e destino, a distância percorrida ao utilizar a rota encontrada e quantos vértices foram visitados;
- *Result Save File* – Caminho para um arquivo onde os dados das buscas (algoritmo e heurística utilizados, tempo de execução, tamanho do caminho) podem ser armazenados para posterior análise;
- *Start Latitude, Start Longitude* – Latitude e longitude do ponto de origem, em microgaus;
- *End Latitude, End Longitude* – Latitude e Longitude do ponto de destino, em microgaus.

3.4.3 Conclusão

Este capítulo apresentou a implementação do aplicativo, desde a criação do banco de dados, detalhes de implementação dos algoritmos até a interação com o usuário. A seguir será realizada uma avaliação do aplicativo, com o objetivo de encontrar os melhores parâmetros para sua utilização.

4 ANÁLISE DE PERFORMANCE

4.1 PARÂMETROS DE TESTES

Os testes foram realizados utilizando um aparelho da fabricante Alcatel, modelo Pixi4 4034E, com 1Gb de memória RAM, processador 1.3 GHz Quad Core.

Para realizar os testes, foram criados 8 grupos de rotas, dependendo da distância entre os pontos inicial e final de cada rota. Esses grupos foram divididos em intervalos de 1km, iniciando em 0.5km até 8.5km. Foram selecionados 3 rotas de cada grupo para a realização dos testes, gerando um total de 24 percursos a serem analisados.

Hermes calculou a rota 60 vezes para cada percurso, sendo cinco repetições para cada combinação de método de cálculo (Dijkstra, e A* com cada heurística) e tamanho de cache (25, 50 ou 100).

4.2 VALIDAÇÃO DE HEURÍSTICA

O algoritmo A* utilizou três heurísticas: distância entre os pontos, distância elevada ao quadrado, e distância elevada ao quadrado dividida pela distância entre os pontos inicial e final, que serão chamadas, respectivamente de A1, A2 e A3. Apesar de A2 e A3 não serem admissíveis, caso os resultados práticos ao se utilizá-las forem satisfatórios (aumento no percurso menor que 10% e tempo médio de execução menor), torna-se interessante utilizá-las.

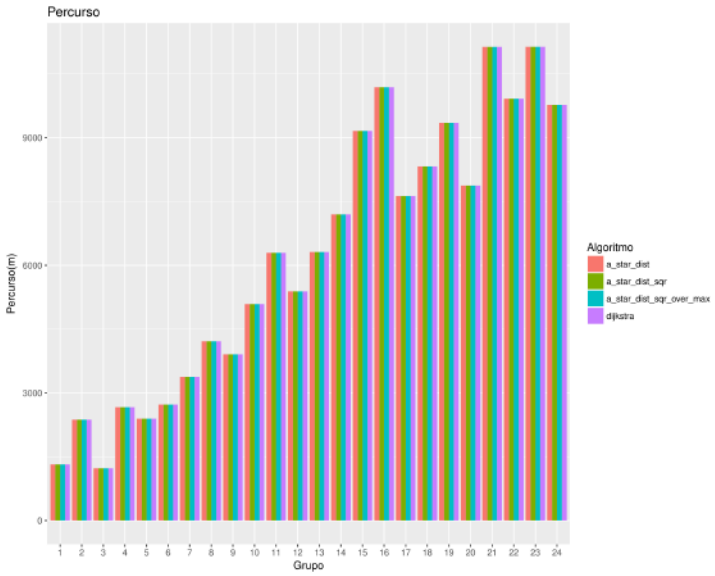
A Figura 9 mostra o resultado da comparação entre o tamanho do percurso obtido por algoritmo para cada par de pontos. Todos algoritmos apresentaram o mesmo tamanho de percurso para cada par de pontos.

4.3 ANÁLISE DO CACHE

Esta seção tem como objetivo determinar qual a melhor configuração de algoritmo e cache para utilizar, considerando o menor tempo de execução. Para cada método(combinação de algoritmo e heurística), O impacto do tamanho do cache foi analisado, com o objetivo de escolher o tamanho que apresenta o menor tempo de execução para cada método. Em seguida, o melhor resultado de cada algoritmo foi comparado uns com os outros, para determinar o melhor método.

O tempo de cada percurso foi calculado como a média entre o tempo de execução entre as cinco repetições de cada combinação. O melhor valor para o cache foi escolhido considerando sendo o menor entre a média dos tempos de execução para todos os percursos.

Figura 9 – Caminho x Percurso x Algoritmo



4.3.1 Dijkstra

A Tabela 1 contém a média para cada valor de cache usando o algoritmo de Dijkstra. A Figura 10 apresenta o resultado para cada percurso, com intervalo de confiança de 95%.

Como o valor de 100 para o cache foi o menor entre os analisados, será utilizado como base para comparações. Considerando as seguintes hipóteses:

a) H_1 O valor do cache 25 apresenta média do tempo de execução igual ou menor que a média para o valor 100;

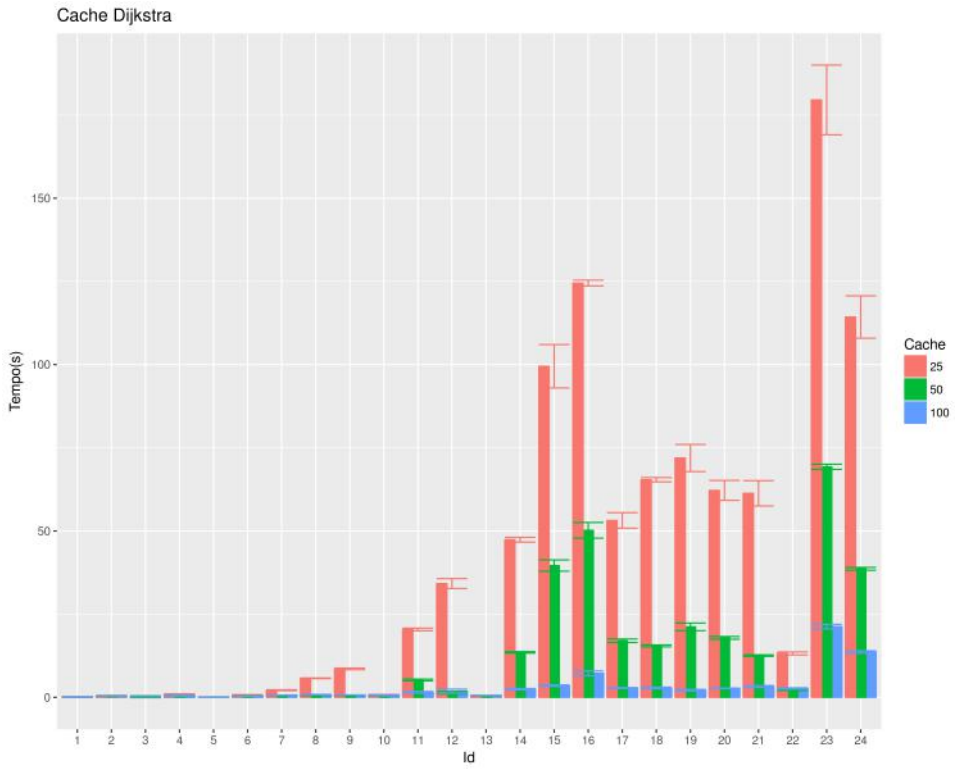
b) H_2 O valor do cache 50 apresenta média do tempo de execução igual ou menor que a média para o valor 100.

Podemos testá-las contra H_3 - O valor do cache alternativo apresenta média do tempo de execução maior que a média para o valor 100. Testando as hipóteses H_1 e H_2, uma de cada vez, e considerando-as hipóteses nulas, para serem aceitas o valor da média correspondente deve ser menor que 12.75. Sendo maiores em ambos os casos , as duas hipóteses são rejeitadas, indicando o valor 100 como o melhor entre os valores testados.

Tabela 1 – Análise do cache para Dijkstra.

Cache	Média do Tempo	Desvio padrão
25	40,29	49,23
50	12,85	18,75
100	3,04	4,85

Figura 10 – Caminho x Tempo x Cache para Dijkstra



4.3.2 A* com Heurística A1

A Tabela 2 contém a média para cada valor de cache usando o método A1. A Figura 11 apresenta o resultado para cada percurso, com seu respectivo intervalo de confiança (95%).

Como o valor de 50 para o cache foi o menor entre os analisados, será utilizado como base para comparações. Considerando as seguintes hipóteses:

a) H_1 O valor do cache 25 apresenta média do tempo de execução igual ou menor que a média para o valor 50;

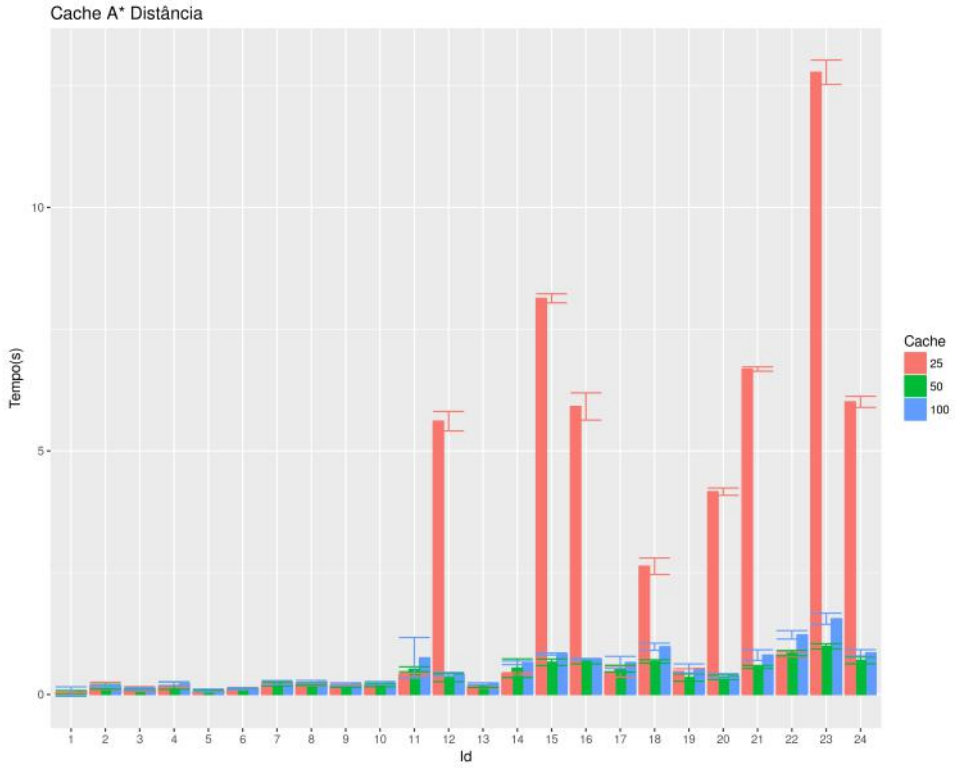
b) H_2 O valor do cache 100 apresenta média do tempo de execução igual ou menor que a média para o valor 50.

Podemos testá-las contra H_3 - O valor do cache alternativo apresenta média do tempo de execução maior que a média para o valor 50. Testando as hipóteses H_1 e H_2, uma de cada vez, e considerando-as hipóteses nulas, para serem aceitas o valor da média correspondente deve ser menor que $0,93$. Como no caso de $H1$ a média do tempo está fora do intervalo de aceitação ela deve ser rejeitada. No caso da hipótese H_2, apesar de a média ser maior, ainda está dentro do intervalo, e o resultado é inconclusivo, e não se pode afirmar que o valor 100 é pior que 50, para o nível de confiança estabelecido.

Tabela 2 – Análise do cache para A1

Cache	Média do Tempo	Desvio padrão
25	2,34	3,43
50	0,39	0,27
100	0,51	0,4

Figura 11 – Caminho x Tempo x Cache para A1



4.3.3 A* com Heurística A2

A Tabela 3 contém a média para cada valor de cache usando o método A2. A Figura 12 apresenta o resultado para cada percurso, com seu respectivo intervalo de confiança (95%).

Como o valor de 50 para o cache foi o menor entre os analisados, será utilizado como base para comparações. Considerando as seguintes hipóteses:

a) H_1 O valor do cache 25 apresenta média do tempo de execução igual ou menor que a média para o valor 50;

b) H_2 O valor do cache 100 apresenta média do tempo de execução igual ou menor que a média para o valor 50.

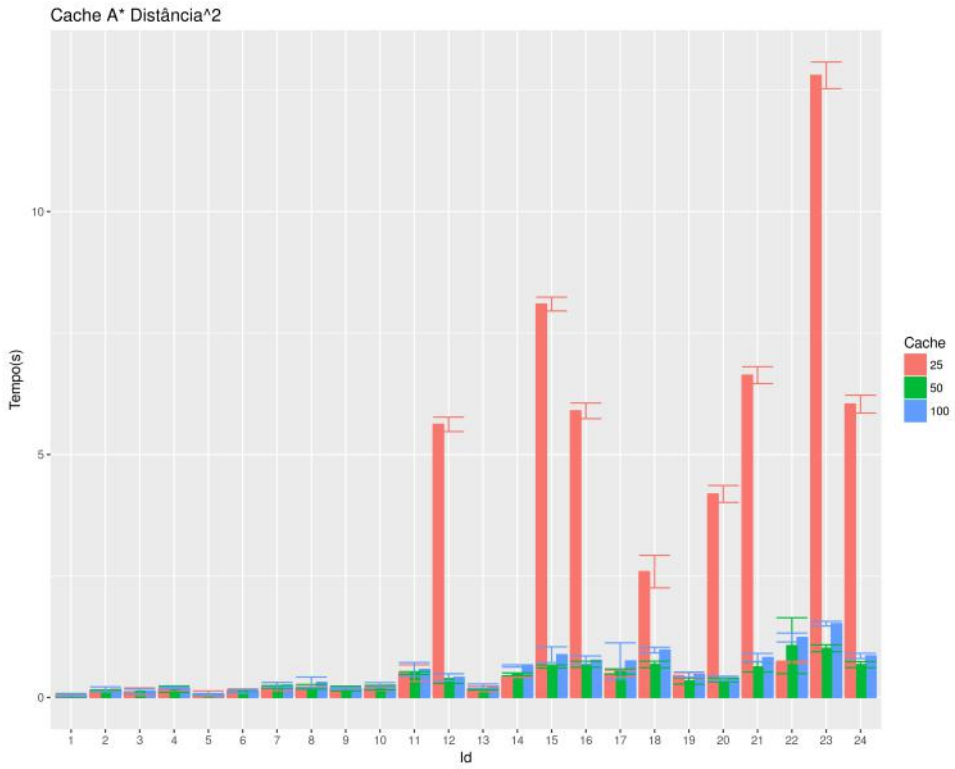
Podemos testá-las contra H_3 - O valor do cache alternativo apresenta média do tempo de execução maior que a média para o valor 50. Testando as hipóteses H_1 e H_2, uma de cada vez, e considerando-as hipóteses nulas, para serem aceitas o valor da média correspondente deve ser menor que $0,98$.

Como no caso de $H1$ a média do tempo está fora do intervalo de aceitação ela deve ser rejeitada. No caso da hipótese H_2, apesar de a média ser maior, ainda está dentro do intervalo, e o resultado é inconclusivo, e não se pode afirmar que o valor 100 é pior que 50, para o nível de confiança estabelecido.

Tabela 3 – Análise do cache para A2.

Cache	Média do Tempo	Desvio padrão
25	2,34	3,43
50	0,40	0,29
100	0,51	0,39

Figura 12 – Caminho x Tempo x Cache para A2



4.3.4 A* com Heurística A3

A Tabela 4 contém a média para cada valor de cache usando o método A3. A Figura 13 apresenta o resultado para cada percurso, com seu respectivo intervalo de confiança (95%).

Como o valor de 50 para o cache foi o menor entre os analisados, será utilizado como base para comparações. Considerando as seguintes hipóteses:

a) H_1 O valor do cache 25 apresenta média do tempo de execução igual ou menor que a média para o valor 50;

b) H_2 O valor do cache 100 apresenta média do tempo de execução igual ou menor que a média para o valor 50.

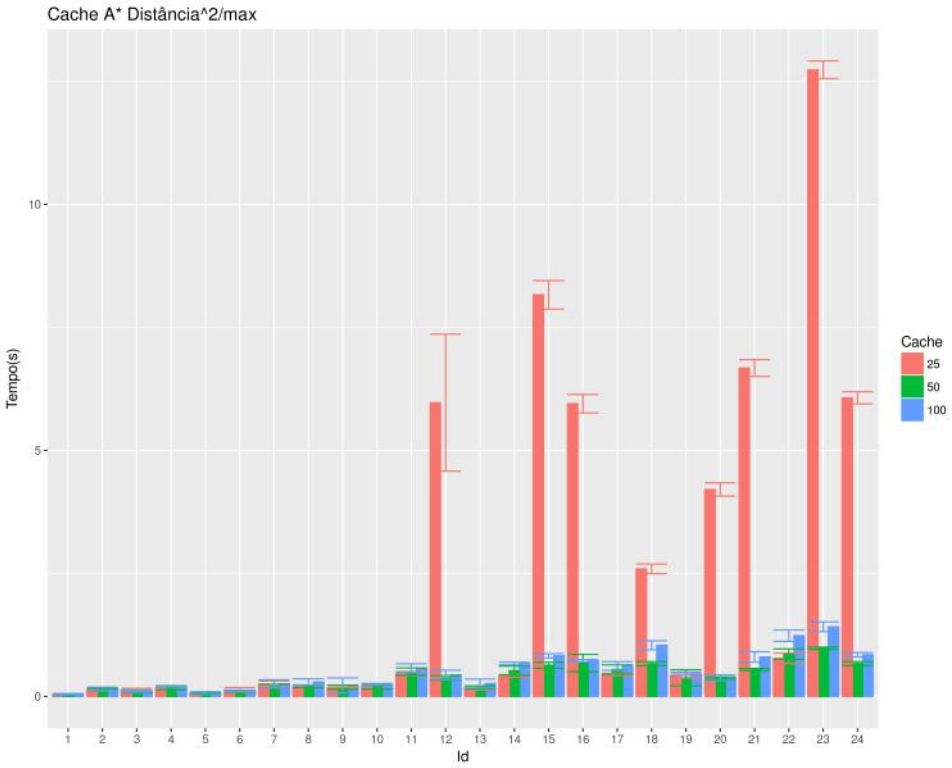
Podemos testá-las contra H_3 - O valor do cache alternativo apresenta média do tempo de execução maior que a média para o valor 50. Testando as hipóteses H_1 e H_2, uma de cada vez, e considerando-as hipóteses nulas, para serem aceitas o valor da média correspondente deve ser menor que $0,93$.

Como no caso de $H1$ a média do tempo está fora do intervalo de aceitação ela deve ser rejeitada. No caso da hipótese H_2, apesar de a média ser maior, ainda está dentro do intervalo, e o resultado é inconclusivo, e não se pode afirmar que o valor 100 é pior que 50, para o nível de confiança estabelecido.

Tabela 4 – Análise do cache para A3.

Cache	Média do Tempo	Desvio padrão
25	2,36	3,45
50	0,39	0,27
100	0,50	0,38

Figura 13 – Caminho x Tempo x Cache para A3



4.3.5 Comparação dos resultados

Nesta seção o melhor resultado de cada método de busca será comparado com o resultado dos outros métodos. A Tabela 5 contém os dados do resultado de cada método, sendo a menor média pertencente ao método A3, que será usado como base para comparação.

Podem ser realizados três testes de hipótese, um para cada método alternativo, cada um com um intervalo de confiança de 95%:

a) H_1 O algoritmo de Dijkstra apresenta menor tempo de execução que o método A3;

b) H_2 O método A1 apresenta menor tempo de execução que o método A3;

c) H_3 O método A2 apresenta menor tempo de execução que o método A3.

Testando as hipóteses H_1, H_2 e H_3, uma de cada vez, e considerando-as hipóteses nulas, para serem aceitas o valor da média correspondente deve ser menor que $0,93$.

Tabela 5 – Comparação entre os métodos.

Método	Média do Tempo	Desvio padrão
Dijkstra	3,04	4,85
A1	0,39	0,27
A2	0,40	0,29
A3	0,39	0,27

A hipótese nula deve ser aceita para o caso dos métodos A1 e A3, não havendo diferença significativa considerando o intervalo de confiança proposto, mas deve ser rejeitada para o caso do algoritmo de Dijkstra. A Figura 14 mostra uma comparação entre o melhor resultado de cada algoritmo para cada caminho.

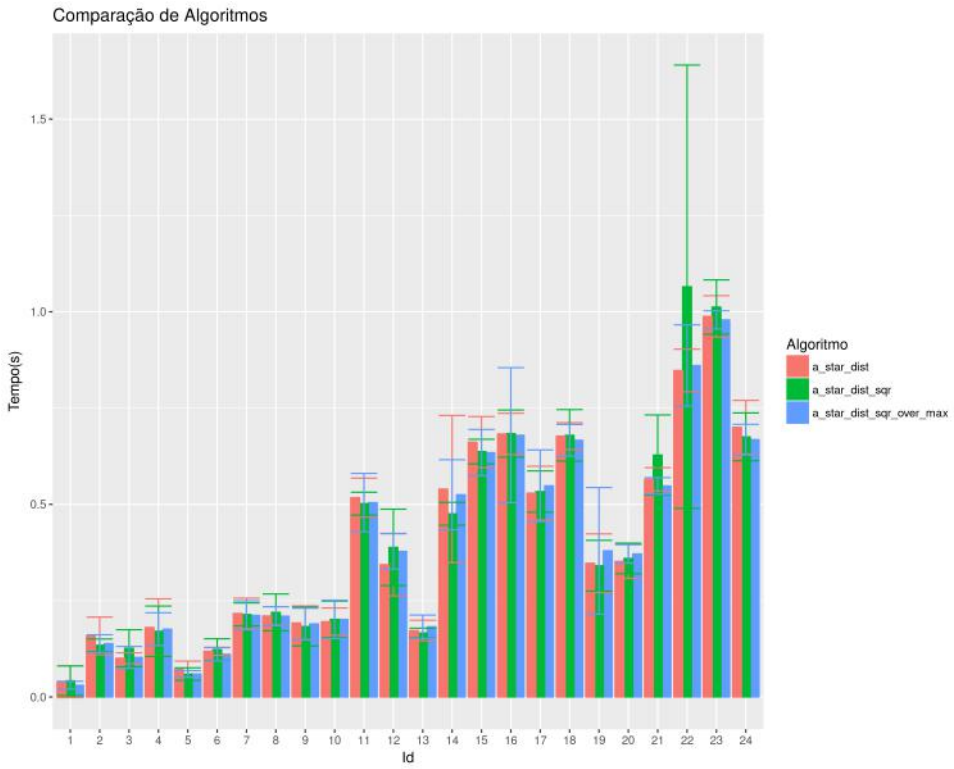
4.3.6 Conclusão

A melhor configuração a ser utilizada pelo aplicativo deve ser a combinação A1 (algoritmo A* utilizando a distância como heurística) com valor 50 para o cache. Apesar de A2 apresentar média menor, não houve diferença significativa em comparação com A1, que apresenta a vantagem de possuir heurística admissível.

O valor do cache, no entanto, permanece inconclusivo. Em determinados percursos, é possível garantir que o valor de 50 é melhor dentro do intervalo de confiança. A variação causada pelos percursos (em oposição a variação causada pela mudança no cache) pode estar influenciando o resultado, e uma outra abordagem talvez consiga estabelecer uma conclusão.

O aplicativo consegue determinar rotas da melhor forma quando configurado com o algoritmo A* utilizando a distância entre os pontos como heurística e cache 50. Se 3 segundos for considerado como tempo máximo aceitável,(tempo máximo recomendado

Figura 14 – Comparação entre algoritmos



pela Google para carregar uma página na internet [1]), Hermes consegue, com sua melhor configuração, encontrar rotas de forma satisfatória entre pontos distantes em até 8,5 km.

Tendo considerado apenas o aplicativo neste capítulo, o próximo irá focar em comparações com outros aplicativos disponíveis no mercado.

[1] <https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/>

5 COMPARATIVO COM OUTROS TRABALHOS

Cinco aplicativos foram inicialmente selecionados para serem comparados neste trabalho (2 deles estão presentes na página do OpenstreetMaps; GoogleMaps e Waze são conhecidos; Hermes é o aplicativo desenvolvido neste trabalho): GPS Navigation, Magic Earth, Waze, GoogleMaps, Hermes. Na Seção 5.1 serão descritos os critérios utilizados para comparar os aplicativos. A Seção 5.2 apresenta uma breve descrição dos aplicativos referidos. E, por fim, a Seção 5.3 apresenta a comparação dos aplicativos, bem como a análise dos resultados.

5.1 CRITÉRIOS DE COMPARAÇÃO

Os seguintes critérios foram utilizados para comparar os aplicativos:

a) **fonte de dados** - de onde vieram as informações dos mapas? Eles são de origem livre? É possível obter os dados dos mapas?;

b) **capacidade de responder** *offline* - o aplicativo consegue responder sem conexão com a internet?;

c) **tempo de resposta** *offline* - demora encontrar uma rota sem internet no aplicativo?;

d) **outras rotas** - o aplicativo mostra mais de uma rota possível?;

e) **outros critérios** - é possível procurar apenas pela distância, ou pode procurar minimizar o tempo ou outros fatores?;

f) **popularidade** - quantos *downloads* o aplicativo possui no GooglePlay (loja de aplicativos da Google)?;

g) **tamanho** - quantos megabytes uma instalação típica consome do celular?(uma instalação típica considera o aplicativo junto com mapas, mesmo que eles precisem ser instalados posteriormente).

5.2 DESCRIÇÃO DOS APLICATIVOS

5.2.1 GPSNavigation

O GPSNavigation[1] é um aplicativo de navegação por GPS *offline* gratuito, baseado no Premium Maps desenvolvido por Maps & GPS Navigation. Possui funcionalidades de mapas 3D, aviso de limite de velocidade e navegação veicular e para pedestres gratuita. Oferece mais funcionalidades, como guia por voz, para usuários pagantes.

[1] <https://play.google.com/store/apps/details?id=cz.aponia.bor3>

5.2.2 Magic Earth

Magic Earth[2] é um aplicativo de mapas e navegação desenvolvido pela General Magic. Oferece uma gama de funcionalidades: mapas 3D ou por satélite, navegação ponto a ponto, e informação de tráfego compartilhada pelos usuários, informação de trânsito, e mapas *offline*.

Dados obtidos do OpenStreetMap.

5.2.3 Waze

O Waze[3], lançado em 2008 pela *start-up* Waze Mobile de Israel, é um aplicativo que além de oferecer ferramentas para auxiliar na navegação, funciona também como rede social. Através deste aplicativo, é possível compartilhar alertas de acidentes, perigos nas rotas e outras informações de tráfego. O Waze ainda permite a interação com amigos, compartilhamento de rotas e localização. Outro recurso interessante para auxiliar a navegação é o uso é o sistema de navegação guiada por voz. O Waze conta com versões para IOS, Android, Windows Phone, Symbian, BlackBerry. Atualmente, o Waze faz a monetização do serviço oferecido através da inserção de propagandas ao longo das rotas dos usuários.

5.2.4 GoogleMaps

GoogleMaps[4] é um serviço oferecido pela Google que permite a visualização de mapas e traçado de rotas entre duas ou mais localizações. Por meio deste serviço, o usuário pode escolher suas rotas (de carro, transportes públicos, trajetos a pé ou de bicicleta), avaliar pontos de interesses e compartilhar os *links* para o caminho traçado. O GoogleMaps também possibilita a utilização do seu serviço através de diferentes plataformas: um usuário pode sincronizar suas pesquisas feitas pela aplicação web com um aplicativo do instalado em seu smartfone Android/iOS, por exemplo. O GoogleMaps conta ainda com uma funcionalidade que permite a sua utilização em modo *offline*, bastando para isto que o usuário faça o *download* de mapas que lhe interessem. Entretanto, esta funcionalidade requer certo espaço de armazenamento disponível no aparelho do usuário. Assim como o Waze, o GoogleMaps disponibiliza a navegação por voz, com informações do trânsito em tempo real, além de notificações de alertas ou atualização de caminho.

5.2.5 Hermes

Hermes[5] é o aplicativo desenvolvido e descrito neste trabalho.

[2] <https://play.google.com/store/apps/details?id=com.generalmagic.magicearth>
[3] <https://www.waze.com/>
[4] <http://maps.google.com/>
[5] <https://play.google.com/store/apps/details?id=nzs.nav.hermes1>

5.3 COMPARAÇÃO

O Quadro 3 apresenta os resultados para cada critério, para cada aplicativo. Enquanto a maioria dos critérios demandou apenas tempo de pesquisa e uso dos aplicativos, o tempo de reposta, de forma diferente, exigiu a execução de testes manuais.

Para a realização dos testes manuais, foram selecionadas três rotas dentro do município do rio de janeiro, cada uma sendo executada uma única vez, cuja distância entre os pontos estivesse entre 10km e 12km. O Quadro 2 indica os pontos inicial e final de cada rota.

Para a medição do tempo, um cronometro foi utilizado. A contagem começa no momento em que o botão que inicia a procura da rota é pressionado (alguns aplicativos iniciavam no momento da confirmação do segundo ponto, portanto esse momento foi o considerado) até a exibição da rota. O aplicativo Waze foi desconsiderado pois não possui rotas *offline*, e a forma para encontrar rotas a partir de pontos que não sejam a posição atual do usuário é realizada em mais de uma etapa, o que a princípio permitiria o cálculo antecipado da rota. Os resultados obtidos podem ser conferidos na Tabela 6.

Quadro 2 – Rotas Usadas na Comparação entre apps

Nome	**Local Inicial**	**Local Final**
A	Shopping Sulacap	Estádio Engenhão
B	Aeroporto Galeão	Maracanã
C	Museu do Amanhã	Mirante do Leblon

Tabela 6 – Tempo de Resposta (em segundos)

Rota	GPSNav	Magic Earth	GoogleMaps	Hermes
A	1,60	2,03	1,80	2,28
B	1,40	2,87	2,10	2,46
C	1,21	2,26	1,80	2,38
Média	1,40	2,39	1,90	2,37

5.4 CONCLUSÃO

Nesta seção apresentamos outros aplicativos do mesmo nicho do Hermes para situá-lo em relação a seus pares. Podemos perceber que apesar de não possuir a função de procurar rotas *offline*, Waze é um dos aplicativos mais populares, perdendo apenas para o GoogleMaps. Considerando o tamanho e popularidade do aplicativo GPSNavigation, o tamanho não parece ser algo muito relevante na escolha dos aplicativos. Contudo, O tamanho dos outros aplicativos parecem indicar que o tamanho ideal é bem menor, entre 150Mb e 250Mb. O desempenho do Hermes foi similar ao do Magic Earth, sendo

Quadro 3 – Comparação entre aplicativos

Critério	**GPSNav**	**Magic E.**	**GoogleMaps**	**Waze**	**Hermes**
Fonte de Dados	Própria	OSM	Própria	Própria	OSM
Offline	sim	sim	sim	não	sim
Tempo de Resposta	1,40	2,39	1,90	NA	2,37
Outras Rotas	sim	não	não	não	não
Outros Critérios	sim	sim	não	não	não
Downloads	50.000.000+	500.000+	1.000.000.000+	100.000.000+	1+
Tamanho	1,04Gb	232Mb	232Mb	161Mb	55Mb

GPSNavigation o mais rápido, seguido pelo GoogleMaps. Entretanto, os testes realizados foram bem simples, e a quantidade de dados é insuficiente para poder ser extrapolada.

6 CONSIDERAÇÕES FINAIS

6.1 DIFICULDADES

Este trabalho, apesar de simples, aborda três áreas bem distintas: algoritmos de busca em grafos, programação para Android, e informação geográfica, o que demandou maior tempo de estudo. Entretanto as maiores dificuldades ocorreram no campo prático.

Duas características do sistema Android impuseram dificuldades no desenvolvimento: sua evolução constante, que por vezes obrigava atualização para suas versões mais novas e a variação existente entre diferentes dispositivos, mesmo considerando a mesma versão. Em determinados casos, o aplicativo fucionava em um dispositivo, mas não em outro.

A implementação do banco de dados foi uma das partes mais difíceis. Uma versão inicial utilizava banco de dados relacionais. Entretanto essa solução não era satisfatória: células com muitos nós podiam terminar o aplicativo pois no número de resultados era maior que o permitido pelo sistema. O modelo atual foi criado para poder ter um controle maior sobre esse processo, e ele ainda pode ser melhorado.

A ferramenta MOBAC, apesar da indiscutível utilizade para o projeto, apresentou alguns problemas em algumas situações. Ela permite escolher o servidor de mapas para gerar o arquivo de imagens para uso *offline*. A dificuldade surgiu quando um dos principais servidores resolveu não mais permitir seu uso, e as imagens deixaram de ser geradas. Nem todos os servidores geravam imagens para o mundo inteiro.

6.2 MELHORIAS

As principais melhorias teriam relação com o desempenho do aplicativo. Nesse sentido, há algumas opções que poderiam ser implementadas:

a) Sanders, Schultes e Vetter (2008) utilizam a técnica de hierarquia de nós para obter tempos de resposta da ordem de milisegundos, além de empregar um modelo de banco de dados que pode ser facilmente adaptado ao sistema para otimizá-lo;

b) Delling et al. (2013) apresenta o algoritmo utilizado no núcleo, na época, do Bing Maps (serviço de busca de rotas online da Microsoft). Algumas das técnicas poderiam ser adaptadas para o cenário *offline* do aplicativo;

Outras melhorias teriam relação com funcionalidades:

a) opção de selecionar outros mapas, além do Rio de Janeiro;

b) obter dados de trânsito em tempo real e adaptar a rota de acordo;

c) adicionar navegação passo a passo;

d) adicionar opções para transporte público;

e) adicionar opções para pedestres;

f) automatizar o processo de construção das imagens do mapa e dos bancos de dados, para mantê-los atualizados com o projeto OSM;

g) utilizar o gps para obter informação do tempo de viagem para compartilhar com outras pessoas.

h) buscar pontos pelo nome da localidade.

6.3 CONCLUSÃO

Este trabalho se ocupou de descrever o aplicativo Hermes, desenvolvido para facilitar a navegação *offline* do usuário. O desenvolvimento abrangeu diversas áreas de conhecimento, e conseguiu apresentar um resultado satisfatório para distâncias curtas. Por isso, é necessário um maior trabalho no sentido de determinar a velocidade do aplicativo em distâncias maiores. Comparando com outros aplicativos no mercado, Hermes conseguiu apresentar desempenho similar, dentro do que foi testado, a aplicativos bem populares. Entretanto falta ao mesmo uma melhor usabilidade para que pudesse tentar ganhar popularidade.

REFERÊNCIAS

ANDROID DEVELOPERS. **Desenvolvedores Android**. 2018. Página Oficial. Disponível em: <https://developer.android.com>. Acesso em: 12 dezembro 2018.

BONDY, J. A.; MURTY, U. **Graph Theory With Applications**. [S.l.]: Elsevier Science Ltd/North-Holland, 1976. ISBN 0444194517.

CORMEN, T. H. et al. **Introduction to Algorithms, 3rd Edition (The MIT Press)**. [S.l.]: The MIT Press, 2009. ISBN 9780262033848.

DELLING, D. et al. Customizable route planning in road networks. Working paper, submitted for publication. 2013. Disponível em: <https://www.microsoft.com/en-us/research/publication/customizable-route-planning-in-road-networks/>.

HUISMAN, O.; de By, R. **Principles of geographic information systems : an introductory textbook**. [S.l.]: ITC, 2009. (ITC Educational Textbook Series). ISBN 978-90-6164-269-5.

OPENSTREETMAP FOUNDATION. **OpenStreetMap Wiki**. 2016. Disponível em: <https://wiki.openstreetmap.org>. Acesso em: 21 janeiro 2016.

OPENSTREETMAP FOUNDATION. **Open Database License (ODbL) v1.0 | Open Data Commons**. 2018. Disponível em: <https://opendatacommons.org/licenses/odbl/1-0/index.html>. Acesso em: 12 dezembro 2018.

ORDNANCE SURVEY. **A Guide to Coordinate Systems in Great Britain**. [S.l.], 2018. Disponível em: <https://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-britain.pdf>. Acesso em: 12 de dezembro de 2018.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice Hall, 1995. ISBN 0131038052.

SANDERS, P.; SCHULTES, D.; VETTER, C. Mobile route planning. In: HALPERIN, D.; MEHLHORN, K. (Ed.). **Algorithms - ESA 2008**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 732–743. ISBN 978-3-540-87744-8.

APÊNDICE A – QUADROS EXTRAS

Quadro 4 – Valores padrão do atributo *oneway*

Valor do atributo *highway*	**Valor padrão do atributo *oneway***
motorway	yes
trunk	yes
primary	no
secondary	no
tertiary	no
residential	no
unclassified	no
living_street	no
motorway_link	yes
trunk_link	yes
primary_link	yes
secondary_link	yes
tertiary_link	yes

Fonte: (OPENSTREETMAP FOUNDATION, 2016)

Tabela 7 – Coordenadas dos pontos dos percursos (em micrograus)

Id Caminho	**Ponto**	**Local**	**Latitude**	**Longitude**
1	Inicial	Praça Nossa Senhora da Paz	-22983060	-43206990
1	Final	Praça General Osório	-22985730	-43197540
2	Inicial	Praça Tiradentes	-22907560	-43183620
2	Final	Praça Candelária	-22901250	-43178130
3	Inicial	Câmara Municipal	-22909570	-43176560
3	Final	Praça Candelária	-22901250	-43178130
4	Inicial	UERJ	-22912930	-43235860
4	Final	Praça Afonso Pena	-22918260	-43217230
5	Inicial	Museu do Forte	-22986090	-43188730
5	Final	Praça Serzedo Correia	-22969550	-43183820
6	Inicial	Engenhão	-22895600	-43291090
6	Final	Igreja N.S.C. Aparecida	-22895060	-43272850
7	Inicial	Praça Marechal Hermes	-22900290	-43202870
7	Final	Catedral de São Sebastião	-22910790	-43179710
8	Inicial	Praça Saes Peña	-22924629	-43232756
8	Final	Praça Nobel	-22927140	-43260250
9	Inicial	Engenhão	-22895600	-43291090

Quadro 4: Continuação

Id Caminho	Ponto	Local	Latitude	Longitude
9	Final	Estação Trem Sampaio	-22902070	-43261763
10	Inicial	Praça General Osório	-22985730	-43197540
10	Final	Pç. Compos. Mauro Duarte	-22954100	-43184060
11	Inicial	Praça Amambaí	-22903460	-43294350
11	Final	Shopping Nova América	-22876060	-43273370
12	Inicial	Largo da Segunda Feira	-22921840	-43221320
12	Final	Praça Malvino Réis	-22919460	-43261610
13	Inicial	Arpoador	-22987840	-43194330
13	Final	Urca	-22953540	-43167270
14	Inicial	Praça Manet	-22884830	-43275180
14	Final	Estação Trem Penha	-22840650	-43278610
15	Inicial	Praça Saens Pena	-22924629	-43232756
15	Final	Jardim Botânico	-22966910	-43218680
16	Inicial	Igreja da Penha	-22842470	-43277930
16	Final	Hospital N. S. do Loreto	-22815330	-43225540
17	Inicial	Praça Seca	-22897090	-43351680
17	Final	Praça Belmiro Gurgel	-22844350	-43345630
18	Inicial	Praça da Apoteose	-22913180	-43197130
18	Final	Praça Pio XI	-22961700	-43217700
19	Inicial	CCMN	-22854950	-43231810
19	Final	Sambódromo	-22911883	-43196664
20	Inicial	Colégio Militar	-22916483	-43226254
20	Final	Engenhão	-22895600	-43291090
21	Inicial	Uruguaiana	-22903060	-43181850
21	Final	CCMN	-22854950	-43231810
22	Inicial	Metrô Glória	-22920620	-43176820
22	Final	Praça Nossa Senhora da Paz	-22983060	-43206990
23	Inicial	Engenhão	-22895600	-43291090
23	Final	COPEAD	-22859600	-43220000
24	Inicial	Praça Amambaí	-22903460	-43294350
24	Final	Praça Afonso Pena	-22918700	-43218330