UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

ROGÉRIO DE CARVALHO SILVA

EXTRAÇÃO DE DADOS GEOESPACIAIS DO OPENSTREETMAP PARA GERAÇÃO DE REDES DE RUAS

QUIXADÁ

2016

ROGÉRIO DE CARVALHO SILVA

EXTRAÇÃO DE DADOS GEOESPACIAIS DO OPENSTREETMAP PARA GERAÇÃO DE REDES DE RUAS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel. Área de concentração: Computação.

Orientador: Prof. Msc. Regis Pires Magalhães

QUIXADÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S583e Silva, Rogério de Carvalho.
Extração de dados geoespaciais do Openstreetmap para geração de redes de ruas / Rogério de Carvalho Silva. – 2016.
54 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2016.
Orientação: Prof. Me. Regis Pires Magalhães.

1. Ruas. 2. Mapas. 3. Algoritmos. I. Título.

CDD 005.1

ROGÉRIO DE CARVALHO SILVA

EXTRAÇÃO DE DADOS GEOESPACIAIS DO OPENSTREETMAP PARA GERAÇÃO DE REDES DE RUAS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel. Área de concentração: Computação.

Aprovado em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Msc. Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

_____
Prof.ª Msc. Lívia Almada Cruz
Universidade Federal do Ceará (UFC)

_____
Prof.ª Msc. Ticiana Linhares Coelho da Silva
Universidade Federal do Ceará (UFC)

A minha família.

A todos aqueles que me deram suporte durante esse período de minha vida.

AGRADECIMENTOS

A minha mãe Antônia, ao meu pai Antônio, a minha irmã Regiana, ao meu irmão Rômulo, por acreditarem e me apoiarem durante toda minha vida.

A Dona Iracema, a Sr. Branco, a Valdinha, por me ajudarem nesse período em que me mudei e morei em Quixadá.

A Júnior Holanda, a Holanda Freires, a Karina Holanda, a Aila Tavares, por terem me ajudado quando cheguei no Ceará, e pelos momentos compartilhados.

A Leuson Mário, companheiro de curso e de muitos momentos ao longo de vários anos, que me ajudou em diversas situações na vida acadêmica e pessoal.

A Giulia Catarine, pessoa com a qual compartilhei parte da minha vida, e que esteve comigo nos momentos bons e ruins nesse período.

A todos os companheiros com os quais morei: Júnior Holanda, João Marcos, Brendo Colaço, Jonas Souza, Vau Catanha, Matheus Souza e Lucas Silva, por terem feito parte dessa jornada tão importante.

Aos meus colegas da turma de 2011.1: João Carlos, Sérgio Filho, Adail Carvalho, João Lucas, Laísa Morais, Rafael Ribeiro, Thiago Pereira, Guylherme Tabosa, Marcilio Valois, Edgar de Almeida, Cleiton Brito, Thiago Levi, Romário Farias, Erick Bhrener, Samuel Carvalho, Jefferson Silva, Christyan Moraes, Luis Paulo, Natasha Silveira e Ingredi Silva, que tive o prazer de conhecer.

Às colegas Hinessa Caminha, Mariana Silva, por todos os momentos compartilhados. A Rafael Costa, companheiro de ideologias politicas e sociais.

A todos os servidores da UFC de Quixadá, pelo excelente trabalho, especialmente aos servidores da CTI: Armando Maia, Lucas Sousa, Matheus Medeiros e Ricardo Ferreira, pessoas com as quais aprendi muito.

Ao meu orientador Prof. MSc. Regis Pires Magalhães pelo apoio e paciência ao longo da construção deste trabalho.

A Samy Sá, meu tutor no PET – conexões de saberes, pela orientação nesse período e pelo apoio durante a graduação.

“Und die die tanzten wurden für verrückt gehalten, von denen, die die Musik nicht hören konnten.” *, (Friedrich Nietzsche)*.

RESUMO

Com o crescimento tecnológico, mapas, que antes eram representados unicamente em papel, passaram também a serem representados em formato digital, possibilitando a criação de uma série de aplicações, como por exemplo, aplicações de rotas de trânsito e logística, visto que mapas em formato digital permitem a representação de diversos tipos de informações em um só lugar. Este trabalho criou um serviço de importação de dados geoespaciais a partir do *Openstreetmap* para criação de redes de ruas que serão utilizadas pelo *framework Graphast* para fazer roteamento de veículos. Para elaboração do trabalho foram analisados os formatos de dados disponibilizados pelo *Openstreetmap*. Os elementos OSM foram mapeados para elementos do grafo que o *Graphast* utiliza, logo em seguida, foi modelado e criado um algoritmo para transformar esses dados em um grafo. Para validação do algoritmo, um estudo de caso foi realizado, onde algumas regiões do planeta foram processadas pelo algoritmo criado neste trabalho e os resultados foram comparados aos obtidos através do uso do *framework GraphHopper*.

Palavras-chave: Serviço de importação. Redes de ruas. *Graphast*. *Openstreemap.*

ABSTRACT

With technological advance, maps that were represented only on paper, also started to be represented in digital form, allowing the creation of a series of applications, such as traffic routes applications and logistics applications, since digital maps allow the representation of different types of information in one place. This work has implemented a geospatial data import service from OpenStreetMap that creates a road network to be used by the Graphast framework for vehicle routing. To elaborate this paper the data formats provided by OpenStreetMap were analyzed. The OpenStreetMap elements were mapped to graph elements that Graphast uses, then an algorithm was modeled and created to transform the OpenStreetMap data into a graph. To validate the algorithm, a case study was implemented, where some maps of some regions of the world were processed by the algorithm created in this paper and the results were compared to those obtained by using the GraphHopper framework.

Keywords: Import service. Road networks. Graphast. OpenStreetMap.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

OSM	OpenStreetMap
MB	Megabyte

Sumário

1 INTRODUÇÃO

Uma das necessidades encontradas pelo ser humano é o anseio por conhecer o ambiente no qual está inserido. Uma das formas de suprir essa necessidade é por meio do conhecimento geográfico do ambiente no qual ele convive, que pode ser atingido com o uso de mapas. Com o avanço da tecnologia, mapas, que antes eram representados em papel, passaram também a serem representados em formato digital, abrindo caminho para uma nova gama de possibilidades; como por exemplo, aplicações de rotas de trânsito e logística, visto que mapas em formato digital permitem a representação de diversos tipos de informações em um só lugar.

Essa capacidade de representação de informações gera um grande volume de dados geoespaciais, o que traz novos desafios; como por exemplo, a extração de forma eficiente dessas informações para a criação de uma estrutura otimizada na qual consultas podem ser realizadas.

Ferramentas como Google Maps[1] fornecem serviços de roteamento, localização de endereços, geolocalização, entre outros serviços. Apesar de estar disponível de forma gratuita para uso, a ferramenta é privada e seus dados não são abertos para *download*, além de ter restrições legais[2] de uso.

Uma alternativa para mapas digitais é o OpenStreetMap (OSM), ferramenta colaborativa iniciada em 2004 na University College London (UCL) (HAKLAY; WEBER, 2008), que provê mapas digitais gerados a partir da colaboração voluntária de usuários. Todos os dados gerados pela ferramenta são públicos e podem ser encontrados *online*, juntamente com uma vasta documentação.

O OpenStreetMap é um projeto que possibilita a criação e distribuição de dados geográficos gratuitamente, e é uma alternativa para desenvolvedores que buscam criar aplicações que utilizem mapas, sem se preocupar com restrições legais de uso.

Os mapas disponibilizados pelo OSM contêm diversas informações geoespaciais

1 https://www.google.com.br/maps
2 https://developers.google.com/maps/terms

representadas abstratamente por: nós, caminhos, relações e rótulos. O desenvolvimento de aplicações que trabalham com dados geográficos depende da extração e manipulação desses dados. Porém, o formato e a manipulação dos dados fornecidos pelo OSM são complexos. Alternativamente, é possível extrair as informações realmente relevantes para um determinado domínio de aplicação e realizar consultas sobre as mesmas através da importação dos dados para um banco de dados relacional, usando ferramentas como OSM2PGSQL (wiki-osm), que trabalham com essa abordagem.

Porém, mesmo importando para um banco de dados, ainda existem consultas que são complexas de serem realizadas e que, em certos casos, precisam ser executadas em um curto tempo de execução para dar uma resposta rápida ao usuário; como por exemplo, a busca pelo caminho mais curto (COSTA et al., 2014), kNN (COSTA et al., 2014) (CRUZ; NASCIMENTO; MACÊDO, 2012) e *Optimal Sequenced Route* (OSR) (SHARIFZADEH; KOLAHDOUZAN; SHAHABI, 2008).

Para a construção de ferramentas que utilizem mapas é importante que se possa trabalhar com diversas fontes de dados geoespaciais existentes na literatura, inclusive o OSM. Porém, algoritmos não manipulam os mapas diretamente, mas sim estruturas criadas a partir deles. Este trabalho pretende modelar um algoritmo para geração de um grafo de redes de ruas a partir do OSM e criar um serviço de importação de dados que gere uma estrutura dos dados disponibilizados pelo OSM, no qual informações relevantes podem ser filtradas de forma eficaz e ainda permitir que a importação e a exportação para outros formatos seja possível.

Entretanto, vale ressaltar que nem todas as informações, disponibilizadas pelo OSM, serão consideradas para prover o serviço. Por exemplo, o horário de criação do nó e o id do usuário, responsável pela criação desse nó, não são informações necessárias para as consultas citadas, além de que a não extração dessas informações reduz o volume de dados processados, e posteriormente, reduz o tempo de processamento das consultas realizadas. Vale ressaltar que a primeira versão do serviço desenvolvido gera um grafo rede de ruas onde o roteamento é destinado a veículos, carros e caminhões, visto que o OSM provê dados de caminhos acessíveis apenas para pedestres ou ciclistas; esses tipos de caminhos não serão o

foco desse trabalho.

Os dados utilizados serão importados para um modelo de grafo. Pretende-se ao final do trabalho integrar o serviço gerado ao *framework* Graphast (MAGALHAES et al., 2015). Graphast é um *framework* que permite construir rapidamente aplicações em redes dependentes de tempo (CRUZ; NASCIMENTO; MACÊDO, 2012); esse *framework* está disponível via API (Interface de Programação de Aplicação) pública e dessa forma permite que novos algoritmos sejam incorporados, aperfeiçoando seus serviços.

É importante salientar que apesar deste trabalho não extrair informações dependentes de tempo, ele permitirá a criação de uma rede que pode ser enriquecida com informações de dependência de tempo a partir de outros dados.

O público-alvo deste trabalho são desenvolvedores que irão usar o *framework Graphast* para construção de aplicações ou pretendem construir seus próprios modelos de grafos de rede de ruas a partir de dados do OSM, oferecendo-lhes não somente uma ferramenta, mas também liberdade de escolha e uma fonte de pesquisa.

Este trabalho está dividido em 7 capítulos, uma introdução, capítulo 1, onde o trabalho é apresentado, o capítulo 2 apresenta os trabalhos relacionados, o capítulo 3 apresenta a fundamentação teórica, onde conceitos básicos para o entendimento do trabalho são descritos. O passos para a execução do trabalho foram detalhados na metodologia, capítulo 4 e artefato gerado foi validado através de um estudo de caso no capítulo 5. São apresentados também uma conclusão e considerações sobre o escopo geral do trabalho no capítulo 6. O último capítulo apresenta as referências utilizadas.

2 TRABALHOS RELACIONADOS

Luxen e Vetter (2011) descrevem o roteamento *off-line* de dados extraídos do OpenStreetMap. Os autores trazem uma abordagem concisa no tratamento de rotas. Foram demonstrados serviços que trabalhavam tanto com processamento do lado do servidor, como com dispositivos móveis. As aplicações extraem o caminho mais curto em redes de ruas em tempo real, ou seja, a consulta de caminho mais curto realizada pelo usuário da aplicação é respondida em tempo real.

Este trabalho se assemelha a Luxen e Vetter (2011) pelo fato de utilizar os formatos disponibilizados pelo OSM como base para geração de redes de ruas a partir da filtragem e processamento dos mesmos, gerando nós e arestas para traçar rotas. Neste trabalho também serão representadas redes de ruas através de grafos, porém o foco será na geração de grafos, enquanto as consultas relacionadas a roteamento, como por exemplo o caminho mais curto, será realizada pelo *framework* Graphast.

Bahrdt (2013) implementa uma busca textual sobre dados do OpenStreetMap em *smartphones* e servidores. Ele apresenta um *framework* que permite buscas complexas sobre os dados do OSM, suportando também a busca por pontos de interesse e geo-localizações.

As informações são extraídas do OSM e armazenadas em um banco de dados relacional, onde serão processadas para a realização de pesquisas. O autor também explica que inicialmente utilizou o formato XML para extração de dados, porém, por questões de desempenho, escolheu abordar o formato PBF.

Uma das relações deste trabalho com o de Bahrtd (2013) é a extração dos dados OSM, porém nesse trabalho os dados são extraídos para um grafo armazenado em memória e nem todas as informações derivadas a partir dos dados geoespaciais foram consideradas, visto que existem muitas informações, e que para este trabalho é definido um modelo de dados filtrando apenas as informações a serem trabalhadas.

Outra ferramenta em que este trabalho se baseia é o GraphHopper[3], uma biblioteca *open source* para roteamento de ruas, escrita em Java e que utiliza dados do OpenStreetMap. Atualmente a biblioteca implementa mecanismos de importação de dados a

3https://github.com/graphhopper/graphhopper/

partir de arquivos OSM em formatos XML e PBF, utilizando mecanismos de paralelização para ganho de desempenho. Além disso, fornece interfaces de importação genéricas para o usuário. Seguindo a abordagem do GraphHopper, este trabalho busca uma implementação eficiente, implementando mecanismos de paralelização de processamento, para extrair os dados. Porém, existe uma preocupação em documentar um modelo de geração de grafo que possa ser usado por outros desenvolvedores para criar seus próprios grafos baseados no OSM.

O quadro 01 mostra um comparativo entre os trabalhos relacionados. O trabalho de Luxen e Vetter gera um grafo de rede ruas, provê roteamento entre pontos, utiliza os formatos PBF e XML para geração do grafo, sendo escrito em C++. O trabalho de Bahrdt é focado apenas em extrair pontos de interesse, utilizando C++ e extraindo os dados a partir do dos formatos XML e PBF, assim como no trabalho de Luxen e Vetter. O GraphHopper tem aspectos semelhantes ao trabalho de Luxen e Vetter, como por exemplo a geração de um grafo de redes de ruas e o roteamento entre pontos, porém a ferramenta é implementada em java.

Quadro 1 - Comparativo entre os trabalhos relacionados

	Gera um grafo de redes de ruas	Extrai pontos de interesse	Provê roteamento	Utiliza os formatos XML/PBF	Escrita em C++	Escrita em Java
Luxen e Vetter (2011)	X		X	X	X	
Bahrdt (2013)		X		X	X	
GraphHopper	X		X	X		X

Fonte: Elaborado pelo autor

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos usados na elaboração deste trabalho. Os elementos do modelo de dados do *Openstreetmap* são apresentados logo no início do capítulo, seguido pela explanação dos formatos de armazenamento de dados disponibilizados pelo OSM. A definição de redes de ruas também é apresentada, seguida pela descrição do *framework Graphast.*

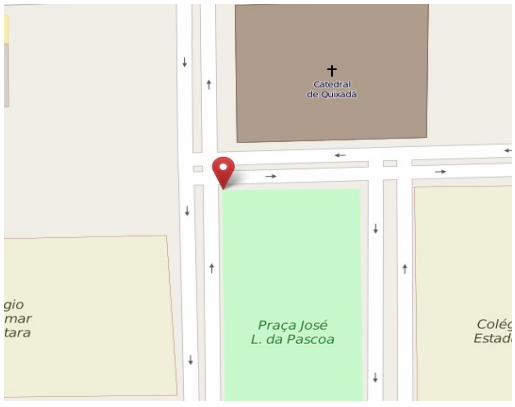
3.1 Elementos do modelo de dados OSM

Nessa subseção, são descritos os principais elementos do modelos de dados do *OpenStreetMap*, que são usados para mapear componentes do mundo físico para o meio digital.

3.1.1 Nó

Um nó representa um ponto único na superfície da terra definido através de uma latitude e uma longitude, que é identificado através de um identificador único[4] (*id*). O nó pode possuir também um rótulo (*tag*), utilizado para representá-lo através de uma chave e um valor, que representam o significado semântico do nó. Por exemplo, um nó poderia representar um banco em um parque ou até mesmo uma fonte de água. A Figura 01 ilustra a representação de um nó no mapa, onde o marcador vermelho representa a localização do nó em relação a elementos reais do mundo.

Figura 1 - Exemplo de um nó no mapa



Fonte: Openstreetmap[5]

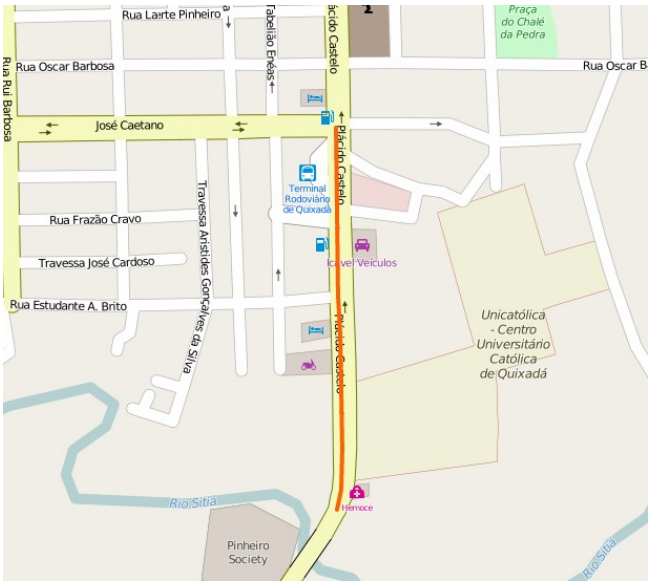
4 https://wiki.openstreetmap.org/wiki/Node

5 www.openstreetmap.org

3.1.2 Caminho

Um caminho é uma lista ordenada de nós com um tamanho, variando de 2 a 2000[6] nós. É usado para representar ruas, rodovias, elementos poligonais, dentre outros. A Figura 2 apresenta uma linha alaranjada no mapa, que tem como finalidade representar graficamente um caminho, representando um trecho em uma avenida. O caminho possui também uma direção, que indica o sentido do trafego naquela região. Geralmente um caminho permite tráfego na duas direções, sendo chamado de bidirecional. Caso a direção seja única, a direção é chamada de unidirecional, onde o tráfego flui do primeiro nó em direção ao último nó do caminho; caso essa direção seja invertida, quando o ultimo nó indica o sentido do tráfego, a direção é chamada de reversa.

Figura 2 - Exemplo de um caminho no mapa



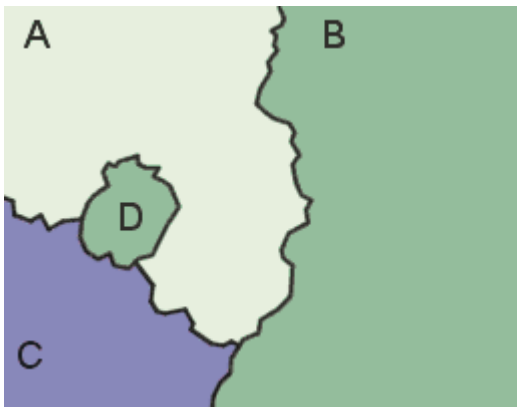
Fonte: OpenstrretMap

3.1.3 Relações

Uma relação é uma estrutura de dados de múltiplos propósitos que documenta a relação entre dois ou mais elementos[7], como nós, caminhos e/ou outras relações. É usada para modelar a relação lógica ou geográfica entre objetos. Um exemplo de uso de relação é na delimitação de fronteiras, onde existe enclave, um território com distinções políticas, sociais e/ou culturais cujas fronteiras geográficas ficam inteiramente dentro dos limites de um outro território[8] , como mostrado na Figura 3, a seguir.

6 https://wiki.openstreetmap.org/wiki/Way
7 https://wiki.openstreetmap.org/wiki/Relation
8 http://www.merriam-webster.com/dictionary/enclave

Figura 3 - Exemplo de Relação de fronteira



Fonte: wiki-osm[9]

A Figura 3 mostra uma relação entre área D e a área B, onde D está contido em A e faz fronteira com C, sendo A e D geograficamente relacionados, mas politicamente separados.

3.1.4 Tag

Uma *tag* (ou rótulo) descreve o significado de um elemento no qual está associada, como um nó, caminho ou relação. É composta de dois elementos: uma chave e um valor[10]. A chave descreve uma classe ampla de uma característica, como a velocidade máxima de uma rodovia. O valor detalha essa característica, que nesse exemplo seria o valor numérico representando a velocidade (maxspeed = 50). Vale ressaltar que elementos OSM podem possuir muitas ou nenhuma *tag(s)*. Um exemplo de *tag* bastante utilizada neste trabalho é a *tag highway,* que indica que um caminho é trafegável, seja por pessoas, bicicletas, barcos, trens ou veículos.

3.2 Formatos de armazenamento de dados OSM

O Quadro 1 descreve os principais formatos utilizados para armazenar dados no *OpenStreetMap*: XML, PBF, o5m e Shapefiles[11], juntamente com uma sucinta descrição dos mesmos. Na seção 4.1.2 é apresentado um quadro comparativo entre os formatos, mostrando

9http://wiki.openstreetmap.org/wiki/Relation:boundary
10https://wiki.openstreetmap.org/wiki/Tags
11http://wiki.openstreetmap.org/wiki/OSM_file_formats

as vantagens e desvantagens de cada um.

Quadro 2 - Principais formatos de dados disponibilizados pelo OSM

Formato	Descrição
OSM XML	Arquivos no formato XML
PBF Format	Altamente compactado, formato binário
o5m	Utiliza mesma codificação do PBF, mesma estrutura do XML
Shapefiles	Descreve as geometrias(nós,caminhos e caminhos fechados) encontradas no mapa

Fonte: wiki OpenStreetMap[12] (com adaptações)

3.2.1 OSM XML

O XML, abreviação de *eXtensible Markup Language*, surgiu como um padrão para estruturação e troca de dados pela Web e pode ser usada para fornecer informações sobre a estrutura e o significado de dados. O Openstreetmap disponibiliza publicamente[13] os dados geoespaciais das diversas regiões do planeta, inclusive um arquivo com os dados de todo o planeta (planet.osm), em formato XML.

3.2.2 PBF Format

Para se entender o formato é necessário primeiro a apresentação de alguns conceitos básicos que foram utilizados para construir o mesmo. A subseção 3.2.2.1 apresenta os conceitos de serialização e desserialização e a subseção 3.2.2.2 apresenta o conceito de *Protocol Buffers.*

3.2.2.1 Serialização e Desserialização

O processo de serialização é definido como um processo de transformação dos dados, de um objeto, em uma matriz de *bytes* serial (AIHKISALO; PAASO, 2011). O processo inverso, a transformação de uma matriz de *bytes* serial em objeto, é chamado de desserialização. Alguns formatos de dados disponibilizados pelo OSM estão serializados e para sua manipulação é preciso desserializá-los.

12 http://wiki.openstreetmap.org/wiki/OSM_file_formats

13 http://www.geofabrik.de/

3.2.2.2 Protocol Buffers

Protocol Buffer é um mecanismo de serialização de dados estruturados, independente de linguagem e plataforma, para uso em protocolos de comunicação, armazenamento de dados, entre outros (KAUR; FUAD, 2010). Essa técnica foi criada pelo Google (2015) para representar os dados na sua infraestrutura de intranet, e foi disponibilizada ao público em 2008 (KAUR; FUAD, 2010).

Para representar a estrutura dos dados, o *protocol buffers* utiliza o conceito de mensagens, onde cada mensagem pode ser considerada a estrutura do objeto que se deseja representar. Essas estruturas são descritas em arquivos com a extensão ".proto" , que são a base para a compilação que serve de base para uma posterior serialização dos dados. É importante ressaltar que para desserializar tais dados é necessário conhecer o arquivo de definição de estrutura (.proto). O OSM implementa o *protocol buffers* para serializar informações geoespaciais e disponibilizá-las para o público.

Um exemplo de mensagem é apresentado na Figura 4, que representa a estrutura de um nó definida pelo OSM. O nó é composto por identificador (id), latitude (lat), longitude (lon), chaves (keys), valores (vals), e informações (info). O id representa um número único que identifica o nó, a latitude e longitude representam a posição geográfica, as chaves e valores são utilizadas para representar um rótulo, ou seja, representar o que aquele nó significa. As informações representam a estrutura que será usada para decodificar o arquivo binário; informações adicionais sobre o protocolo podem se encontradas na documentação[14].

Figura 4 - Exemplo de Mensagem

```
message Node {
   required sint64 id = 1;
   required sint64 lat = 7;
   required sint64 lon = 8;
   repeated uint32 keys = 9 [packed = true]; // Denote strings
   repeated uint32 vals = 10 [packed = true];// Denote strings
   optional Info info = 11; // Contains metadata
}
```

Fonte: wiki OpenstreetMap[15]

14 https://developers.google.com/protocol-buffers/docs/proto
15 https://wiki.openstreetmap.org/wiki/PBF_Format

3.2.2.3 O Formato PBF

O formato PBF (*Protocolbuffer Binary Format*) é uma alternativa para o XML, e foi baseado na implementação de *Protocol Buffers* (subseção 2.4.2.1). Foi projetado para que os arquivos gerados fossem menores que arquivos XML e até 6 vezes mais rápido para escrever e 5 vezes mais rápido para ler[16], quando comparado à leitura e escrita de um arquivo compactado com gzip[17].

3.2.3 O5M

Foi projetado para implementar características do formato XML e do formato PBF, tentando combinar as vantagens de cada um. O numero 5 no nome significa[18] "5 vezes mais rápido que o .OSM". Os principais objetivos do formato são obter arquivos de tamanho pequeno e rápido processamento, alinhado a uma hierarquia bem definida sendo possível unir dois ou mais aquivos. O formato deixa também o usuário livre para escolher o método de compressão que irá aplicar no arquivo.

3.2.4 Shapefiles

O formato foi desenvolvido e regulamentado pelo ESRI[19] (*Environmental Systems Research Institute*) e tem como finalidade descrever geometrias, tais como pontos, linhas ou polígonos. *Shapefiles* são divididos em diferentes arquivos, cada qual com uma finalidade específica.

As três extensões obrigatórias, que são definidas na especificação do ESRI (2015), são ".shp", que define a própria geometria, o ".shx", que representa os índices das geometrias definidas no ".shp" , e por ultimo o ".dbf", que define os atributos contidos nas geometrias. Esse mesmo padrão é aplicado no OSM e em órgãos governamentais como IBGE (Instituto Brasileiro de Geografia e Estatística) e DNIT (Departamento Nacional de Infraestrutura de Transportes), e os dados são disponibilizados abertamente.

---

16 http://wiki.openstreetmap.org/wiki/PBF_Format
17 http://www.gzip.org/
18 http://wiki.openstreetmap.org/wiki/O5m
19 http://www.esri.com/

3.3 Redes de ruas

Segundo White, Bernstein e Kornhauser (2000), um grafo pode representar uma rede de ruas. As interseções entre as ruas representam os vértices do grafo e as arestas são representadas pelas conexões entre os vértices da rede de ruas. As arestas também indicam o sentido de tráfego da rua. Uma aresta que representa uma rua de tráfego em sentido único é chamada de unidirecional, já uma aresta representando uma rua de tráfego em dois sentidos é chamada de bidirecional.

Arestas possuem também pesos ou custos que podem, por exemplo, representar a distância entre dois vértices ou o tempo necessário para trafegar de um vértice a outro. A Figura 5 mostra como uma rede de ruas é representada por um grafo. As arestas, linhas azuis, representam as ruas, e os vértices, pontos vermelhos, representam as intersecções entre elas.

Figura 5 - Grafo representado redes de ruas.

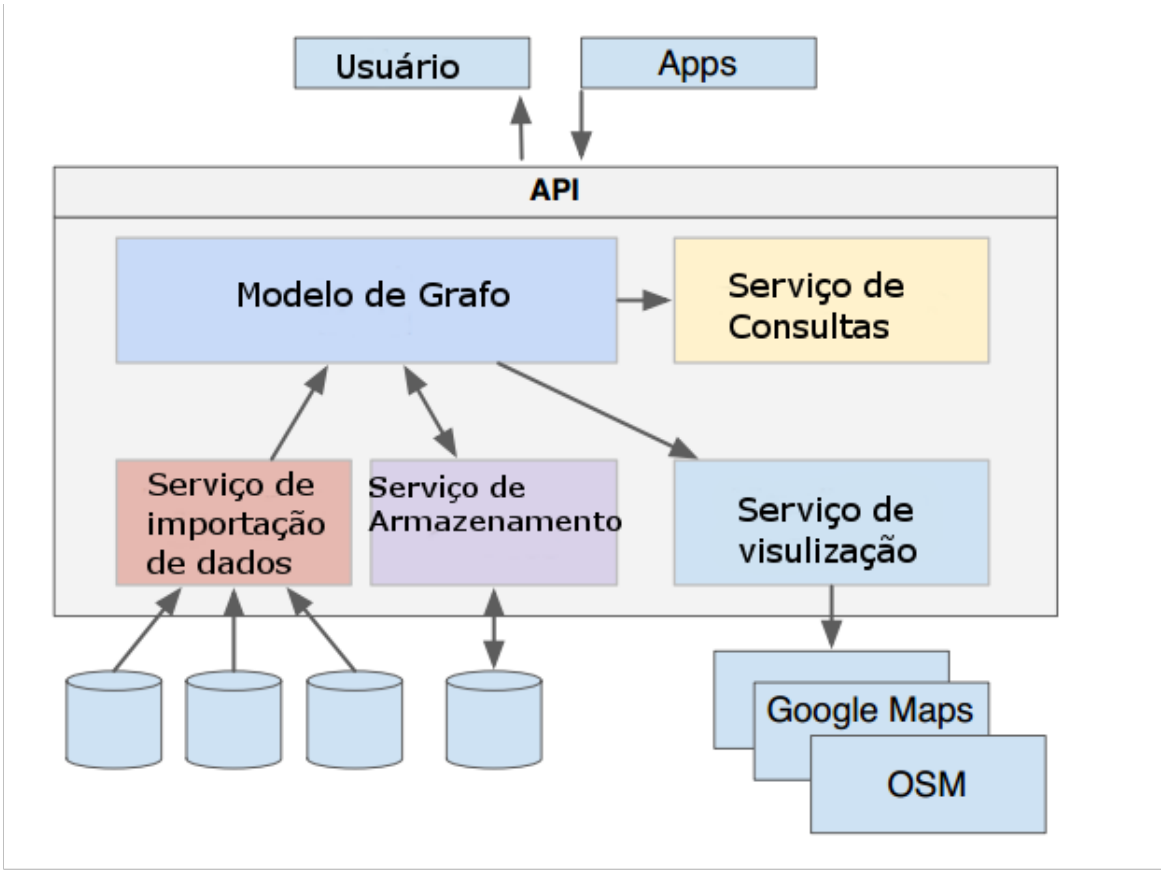
Fonte: Adaptada pelo autor a partir do OSM.

3.4 *Framework Graphast*

Graphast é um *framework* que dá suporte a construção de aplicações com redes dependentes de tempo (MAGALHAES et al., 2015). O principal objetivo é permitir que desenvolvedores criem rapidamente soluções para diferentes tipos de problemas relacionados a redes dependentes de tempo.

A Figura 6 ilustra a organização arquitetural do *framework*, mostrando como os serviços estão organizados, e os principais componentes do mesmo. É importante destacar que este trabalho será responsável por implementar um serviço de importação de dados que será integrado ao *framework.*

Figura 6 - Arquitetura do *framework* Graphast



Fonte: Adaptado de MAGALHAES et. al, 2015.

O *framework* é composto por um modelo de grafo que representa tanto redes independentes e dependentes de tempo. O serviço de consultas é responsável por executar

algoritmos de consulta sobre o modelo de grafo. O serviço de visualização enfatiza os resultados dos algorítimos de consulta possibilitando a iteração entre o usuário e o grafo. O serviço de importação de dados do Graphast é responsável pela importação de dados de diferentes fontes. O serviço utiliza atualmente o *framework Graphhopper* para importação dos dados, porém não possui nenhuma implementação própria implementada. O principal objetivo deste trabalho é criar um algoritmo genérico para geração de grafos a partir do OSM, este trabalho também pretende criar uma implementação que possa ser integrada ao serviço de importação de dados do Graphast.

3.4.1 Modelo de Dados

Graphast ainda implementa um modelo de grafo, composto basicamente por nós e arestas, que é armazenado em memória para que operações de leitura e escrita possam ser realizadas de forma eficiente. A preocupação deste trabalho é criar um algoritmo que possibilite a geração de um grafo a partir do OSM e usar esse algoritmo para criar um serviço de importação de dados que seja utilizado no *framework*.

O modelo de grafo é composto por nós e arestas; o Quadro 2 apresenta os elementos que compõe o modelo.

Quadro 3 - Modelo de Grafo

	ID	Latitude	Longitude	Categoria	Rótulo	Geometria	FC	Distância	Nó de Origem	Nó de Destino
Nó	X	X	X	X	X		X			
Aresta	X				X	X	X	X	X	X

Fonte: Elaborado pelo autor

O nó é composto por um identificador único, uma latitude, uma longitude, uma categoria que representa um ponto de interesse, por exemplo, uma farmácia ou um restaurante. FC representa uma função de custo que pode ser usada, por exemplo, para representar o horário de funcionamento de um local. Vale ressaltar que a função de custo não é uma informação provida pelo OSM, mas que pode ser usada pelo *framework* para representar redes dependentes de tempo.

A aresta é representada também por um identificador único, uma função de custo. Possui também um nó de origem e destino que representam a direção da aresta. Contém

também uma geometria, que é uma lista de pontos que junto com o nó de origem e nó de destino compõem um caminho; por exemplo, para traçar um caminho do ponto A (nó origem) ao ponto B (nó destino) é preciso passar por vários pontos. Esses pontos representam a geometria. O atributo distância, presente na aresta, representa a distância entre os dois nós que compõem uma aresta.

Esse modelo de grafo será utilizado para receber dados a partir do OSM, onde somente a função de custo temporal não é obtida a partir dos dados do OSM. No entanto, este dado pode ser obtido a partir de outras fontes de dados para enriquecer o grafo e possibilitar a realização de consultas dependentes do tempo.

Neste capítulo foram apresentados os elementos do modelo de dados do *Openstreetmap* e os formatos de dados que esses elementos estão disponíveis. Foi apresentado também a definição de rede de ruas. O *framework Graphast* também foi detalhado. Os conceitos apresentados serão utilizados ao longo do trabalho. O próximo capítulo apresenta a metodologia utilizada no trabalho.

4 METODOLOGIA

Para a geração de um grafo de redes de ruas a partir do OSM foi preciso seguir algumas etapas, onde a primeira delas é escolher, dentre os formatos de dados disponibilizados pelo OSM, qual formato mais adequado para se utilizar na implementação. Em seguida, os elementos OSM foram mapeados para elementos do grafo que representam a rede de ruas; para que o grafo fosse gerado corretamente, alguns filtros foram definidos o que permitiu então a modelagem de um algoritmo para a criação do grafo; tal algoritmo foi implementado e verificado para que fosse integrado ao *framework Graphast*.

4.1 Escolher formato de dados OSM

O OSM, como já dito na subseção 3.2, disponibiliza seus dados em uma série de formatos, cada um com suas vantagens e desvantagens. Como cada formado exige implementações diferentes, o objetivo deste passo foi analisar os formatos disponibilizados pelo OSM, listar suas características e escolher um formato para utilizar no desenvolvimento do projeto.

4.1.2 Comparativo entre os formatos

O Quadro 3 lista um conjunto de características que podem ou não estar presentes nos formatos de armazenamento de geodados que o OSM oferece suporte. São avaliados quatro (4) formatos que o OSM disponibiliza: OSM XML, Formato PBF, O5M e SHAPEFILES. As informações foram coletadas a partir da documentação do OSM[20]. As características de formato que não se pôde afirmar estão marcadas como N/A.

A primeira característica listada é a capacidade que o homem tem de conseguir ler e interpretar as informações contidas no arquivo. Isso é importante inicialmente para facilitar o aprendizado da organização estrutural dos dados e identificar as informações contidas no mesmo.

A segunda característica avalia o quão fácil é editar um ou mais atributos de um arquivo. Devido a forma que foi projetado, o XML apresenta a capacidade de ser editado facilmente pois o formato é suportado por diversos editores de textos simples, a exemplo do *Notepad do Windows e Gedit do Linux.*

20 https://wiki.openstreetmap.org/wiki/OSM_file_formats

O terceiro item do Quadro 3 avalia se o formato já possui uma compressão nativa. A compressão possibilita uma redução no tamanho dos arquivos, facilitando a transferência via rede. De todos os formatos analisados apenas o XML não é um formato com uma compressão nativa, acarretando em arquivos com tamanho superior aos arquivos nos demais formatos analisados.

A quarta característica é a capacidade de *merge* de cada formato, ou seja, a possibilidade agregar mais dados de outros arquivos de mesmo formato a um arquivo já existente. Baseado na documentação do OSM foi possível afirmar a presença dessa característica em apenas dois formatos: XML e O5M.

A quinta característica aborda o quão difundido é o formato atualmente. O formato XML tem muitos anos de existência e é usando amplamente no mercado, desde *Web Services* até aplicações simples. A documentação do OSM cita quatro (4) ferramentas de conversão de formato: Osmosis[21], Osmconvert[22], Osmium[23] e Osm4j[24], que implementam o formato XML e PBF. Outro exemplo de ferramenta que dá suporte ao formato PBF é *framework GraphHoper*, citado nos trabalhos relacionados. O O5M é o formato mais recente criado pelo OSM e não foram encontradas uma gama abrangente de ferramentas implementadas na linguagem Java que oferece suporte ao formato. O formato *shapefiles*, apesar da documentação não citar ferramentas, é um padrão criado na década de 90 e é utilizado por ferramentas como Arcgis[25], Qgis[26] e órgãos governamentais como IBGE e DNIT, sendo considerado amplamente utilizado.

A sexta característica listada é a velocidade com que os formatos podem ser processados. O XML é um formato de texto que o torna lento para ser processado, se comparado aos outros formatos analisados, que são formatos binários.

O sétimo item da lista aborda o suporte ao acesso randômico, ou seja, a

21 http://wiki.openstreetmap.org/wiki/Osmosis
22 http://wiki.openstreetmap.org/wiki/Osmconvert
23 http://wiki.openstreetmap.org/wiki/Osmium
24 http://wiki.openstreetmap.org/wiki/Osm4j
25 https://www.arcgis.com/features/
26 http://www.qgis.org/en/site/

capacidade de acessar um item em qualquer posição de uma determinada lista de elementos. Essa característica pode influenciar na forma como dados são consultados. Dos formatos analisados, o XML é o único que dá suporte a essa característica, sendo que a partir da documentação não foi possível inferir ou negar essa característica no formato *shapefiles*.

A oitava característica se refere a forma como os arquivos estão divididos, sendo que o único formato que necessita ser dividido em mais de um arquivo é o *shapefiles*, pois apenas um arquivo não representa a informação completa. Isso é importante, pois pode deixar a implementação mais complexa, visto que seria necessário trabalhar com vários arquivos e índices para se ter uma informação completa.

A nona característica avalia a capacidade de encontrar os geodados para *donwload* em diferentes formatos. Foram avaliados os formatos disponibilizados pela Geofabrik[27], um companhia alemã que tem parceria com o *OpenStreetMap Foundation*[28] e que disponibiliza os dados coletados pelo OSM. A Geofabrik disponibiliza os dados em três formatos: XML (arquivos compactados), PBF e *shapefiles*. Essa característica é importante pois faz com que um maior número de pessoas tenham acesso aos dados do OSM. O próprio OSM também tem a opção de exportar regiões em formato XML.

27 http://www.geofabrik.de/

28 https://wiki.osmfoundation.org

Quadro 4 - Comparativo entre os formatos de armazenamento de arquivos OSM

Característica/Formato	OSM XML	Formato PBF	O5M	SHAPEFILES
1 - Legível ao homem	Sim	Não	Não	Não
2 - Fácil edição	Sim	Não	Não	Não
3 - Compressão	Não	Formato já compresso	Já compresso, mas forma de compressão pode ser escolhida pelo usuário	Não compresso
4 - Possibilita merge de arquivos	Sim	N/A	Sim	N/A
5 - Utilizado por diversas ferramentas	Sim	Sim	Não	Sim
6 - Processamento	Lento	Rápido	Rápido	N/A
7 - Acesso randômico	Sim	Não	Não	N/A
8 - Dados divididos em vários arquivos	Não	Não	Não	Sim
9 - Disponibilizado diretamente para download	Sim	Sim	Necessário conversão a partir de outros formatos	Sim

Fonte: Elaborado pelo autor

4.1.3 Escolha do formato

Para a escolha do formato padrão do OSM a ser usado na importação de dados proposta neste trabalho foram levadas em consideração algumas características previamente avaliadas: a compressão, as ferramentas para trabalhar com o formato, a velocidade de processamento e a disponibilidade desses arquivos online. Dentre os formatos, o que melhor atende esses requisitos é o formato PBF pois possui uma boa taxa de compressão, seu processamento é eficiente e geodados nesse formato podem ser encontrados facilmente online.

4.2 Mapear elementos OSM para elementos do grafo

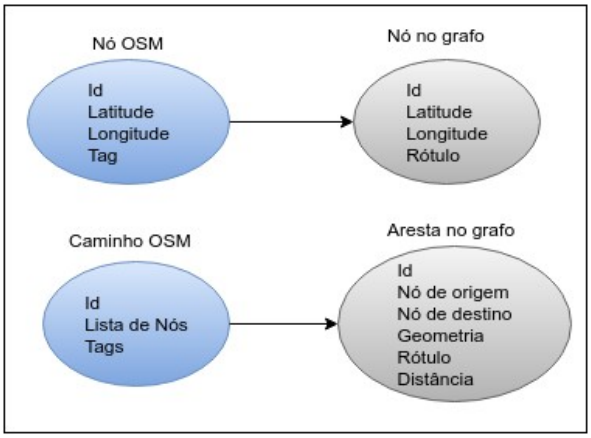
Nesta etapa do projeto foram analisados os elementos de dados do OSM apresentados na subseção 3.1 e o modelo de dados apresentado na subseção 3.4.1, para que fosse possível mapear elementos do OSM para componentes no grafo.

A Figura 7 mostra como os elementos do mapa são traduzidos para itens no grafo, como, por exemplo, um nó no mapa é composto de um id, latitude, longitude e *tags,* sendo convertido para um nó no grafo que é formado por um id, uma latitude e uma longitude e um rótulo que indica algum possível ponto de interesse que esse nó representa; por exemplo uma farmácia ou um restaurante.

A imagem também apresenta a tradução de um caminho no mapa para uma aresta no grafo, sendo que o caminho no OSM é constituído por um id, uma lista de nós e *tags* e

pode ser transformado em um ou mais arestas no grafo. O vértice gerado é composto por um id, nó de origem, nó de destino, uma geometria, um rótulo e a distância.

Figura 7 – Mapeamentos de elementos OSM para o modelo de dados.



Fonte: Elaborada pelo autor

4.3 Definir filtros para Geração do grafo

Para a geração de um grafo de redes de ruas reais é preciso definir que tipo de meio de transporte o grafo oferecerá suporte; por exemplo, navegação para carros ou bicicletas. Tendo em vista que no mundo real diferentes tipos de meio de transporte navegam por diferentes tipos de caminhos e que este trabalho é focado em roteamento para automóveis. Definiu-se um filtro para determinar se dado caminho no mapa é candidato ou não a se transformar em vértices no grafo.

A definição de um filtro é uma tarefa que exige conhecimento dos diversos cenários possíveis para algum tipo de roteamento, fazendo com que a criação de um filtro não seja uma tarefa trivial para leigos no domínio. O Quadro 4 apresenta o resultado de uma intensiva leitura na documentação[29] do OSM mostrando as condições necessárias para que um caminho seja ou não considerado como trafegável para veículos.

Antes de abordar detalhes do Quadro 4, é necessário entender o significado de algumas *tags* do OSM, como por exemplo a *tag highway,* que indica se o caminho é trafegável, seja por veículos, bicicletas ou pedestres. A *tag highway* pode assumir diferentes valores, como os valores apresentados no Quadro 4. Esse valores indicam o tipo da *tag,* sendo importante apenas saber se a *tag* indica ou não um caminho trafegável.

O Quadro 4 mostra que elementos que possuem a *tag highway* com um valor "*motorway"* ou "*motorway_link"* ou "*motorway_junction"* ou "*trunk"* ou "*trunk_link"* ou "*primary_link"* ou "*primary"* ou "*secondary"* ou "*tertiary"* ou "*unclassified"* ou

29 http://wiki.openstreetmap.org/

“*unsurfaced”* ou “*residential”* ou “*living_street”* permitem navegação por veículos, desde que a *tag access* não possua valores iguais a “*private”* ou “*no”* ou que a *tag “impassable”* não seja igual a “*yes”* ou a *tag status* não apresente valor “*impassable”* ou que *tag motor_vehicle* não seja igual a “*no”* ou a *tag surface* não tenha valor igual a “*ground”* e por último que a *tag motorcar* não tenha valores iguais a “*no”* ou “*private”* ou “*privative”* ou “*permissive”* ou qualquer outro valor desconhecido. Vale ressaltar que as *tags* citadas podem ou não aparecer, e para um algoritmo mais preciso é necessário checar todas as possibilidades apresentadas.

O Quadro 4 sugere também que caminhos com a *tag “highway”,* com valor “*service”,* e que também possuem a *tag “tunnel”,* com valor “*passage”,* não são trafegáveis para veículos. Isso ocorre pois essas áreas representam áreas de serviço em túneis.

Outro ponto ressaltado no Quadro 4 são elementos com *tag “highway”,* com valores “*bridleway”* ou “*cycleway”* ou “*footway”* ou “*pedestrian”* ou “*bus_guideway”* ou “*path”,* e com a *tag “motorcar”,* com valor igual “*no”,* indicando que esses elementos não são candidatos a compor o grafo.

Caso o caminho não possua a *tag “highway”* ele não é considerado no grafo. Porém o Quadro 4 apresenta uma exceção a essa regra; caso o caminho possua a *tag “route”,* com valor igual a “*shuttle_train”* ou “*ferry”,* e possua a *tag “motorcar”* ou “*motor_vehicle”,* igual a “*yes”,* o caminho é considerado no grafo. Isso ocorre pois em algumas situações é preciso fazer alguma travessia, como, por exemplo, em um rio onde a travessia é feita por balsas.

Quadro 5 - filtros de rotas para veículos

Chave – Valor		Chave – Valor		Permitido?
highway	motorway ou motorway_link ou motorway_junction ou trunk ou trunk_link ou Primary_link ou primary ou secondary ou tertiary ou unclassified ou Unsurfaced ou Residential ou living_street	**motorcar**	'yes' ou vazio	SIM
		access	'private' ou 'no'	NÃO
		impassable	'yes'	
		status	'impassable'	
		motor_vehicle	'no'	
		surface	'ground'	
		motorcar	'no' 'private' ou 'permissive' ou valor desconhecido	
	service	**tunnel**	'passage'	
	bridleway ou cycleway ou footway ou pedestrian ou bus_guideway ou Path	**motorcar**	'no' ou vazio	
	track	**tracktype**	diferente de 'grade1' ou 'grade2' ou 'grade3'	
route	shuttle_train Ferry	motocar	'yes'	SIM
		motor_vehicle		

Fonte: Elaborado pelo autor

4.4 Modelar algoritmo para criação do grafo

Baseado nas informações obtidas no mapeamento dos elementos OSM e na definição dos filtros, um algoritmo para criação do grafo foi modelado. O algoritmo é dividido em duas etapas, descritas nas Figura 8 e 9, respectivamente.

O algoritmo recebe como entrada um arquivo PBF, mas pode ser adaptado para oferecer suporte aos outros formatos disponibilizados pelo OSM, citados na subseção 3.2, bastando apenas implementar módulos de leitura para os diferentes formatos.

4.4.1 Primeira etapa do algoritmo

A Figura 8 descreve um pseudocódigo da primeira etapa do algoritmo, onde

ocorre o pré-processamento dos elementos. O primeiro passo é receber um arquivo e iterar sobre todos os elementos desse arquivo, linha 1. Para cada elemento verifica-se seu tipo, linha 3. Se o elemento for um nó, então esse elemento é armazenado em uma estrutura de dados; caso o elemento seja um caminho, valida-se esse caminho, linha 7, baseando-se nos filtros explicados no passo anterior. Cria-se então outro laço para iterar sobre todos os nós que compõem esse caminho.

Na linha 9 é verificado se o nó já está armazenado no grafo; caso esteja, esse nó é marcado como principal, caso contrário, o nó é marcado como secundário. Nós principais são nós que aparecem em mais de um caminho e representam interseções entre ruas, como descrito na subseção 3.3, logo são candidatos a se tornarem vértices no grafo. Nós secundários são nós que aparecem apenas uma vez em todo o conjunto de caminhos e são os nós que aparecem entre dois vértices, constituindo assim, a geometria de uma aresta.

Figura 8 - Primeira etapa do algoritmo

```
1
2  while(exist((elementsOSM)){
3      if(elementOSM == NODE){
4          addNodeToStructure(elementOSM)
5      }else if(elementOSM == CAMINHO){
6
7          if(valido(elementOSM)){
8              for(i =0; i < elementOSM.NODES(); i++){
9                  if(findNoNaEstrutura(elementoOSM.getNODE(i).id){
10                     marcarNoComoPrincipal(elementOSM.getNODE(i).id);
11                 }else{
12                     marcarNoComoSecundario(elmentOSM.getNODE(i).id)
13                 }
14
15             }
16         }
17
18     }
19 }
```

Fonte: Elaborada pelo autor.

4.4.2 Segunda etapa do algoritmo

Na segunda etapa do algoritmo, descrita na Figura 9, por meio do pseudocódigo, é preciso iterar novamente por todos os elementos do mapa para que o grafo seja construído.

4.4.2.1 Iterar sobre elementos OSM

Assim como na primeira etapa, o primeiro passo é receber um arquivo e iterar sobre todos os elementos desse arquivo, linha 2, verificando-se seu tipo. Se o elemento não for um caminho, esse elemento é ignorado e a iteração continua. Caso o elemento seja um caminho, valida-se esse caminho, linha 5, baseando-se nos filtros explicados no passo anterior.

4.4.2.2 Verificar existência de barreiras no caminho

Caso o caminho seja válido é necessário checar se entre os nós que compõem o caminho, existe algum nó que possui alguma barreira, linha 6; como por exemplo, um portão ou algo que impeça a passagem de veículos. Isso é possível através da identificação da *tag barrier* no nó. Caso exista alguma barreira é necessário redimensionar o caminho, linha 7, fazendo com que o nó que representa a barreira seja o nó final do caminho, pois assume-se que não seja possível transitar além da barreira.

4.4.2.3 Iterar sobre nós do caminho

O próximo passo é iterar sobre os nós do caminho. Para cada nó é verificado se o nó é principal ou secundário. Isso é possível graças a etapa 1 do algoritmo, onde nós são marcados como principais ou secundários. Caso o nó seja secundário, o nó é ignorado e a iteração continua. Caso o nó seja principal, linha 10, identifica-se o nó de origem e o nó de destino. O nó de origem é identificado como o primeiro nó principal que aparece no caminho e o nó de destino é o próximo nó principal que aparece no caminho. Tanto o nó de origem quanto o nó de destino se transformam em vértices no grafo e a geometria são os pontos obtidos entre esses nós. Após a identificação, tais vértices são adicionados no grafo, linha 14, caso ainda não tenham sido.

4.4.2.4 Verificar direção do caminho

O próximo passo é a identificação da direção do caminho, ou seja, identificar qual o sentido do tráfego, explicado na subseção 3.1.2, para que isso seja atribuído na aresta, linha 16. O Quadro 5 mostra como obter a direção do caminho. Assume-se que um caminho é bidirecional, porém como mostrado no Quadro 5, caso o caminho possua a *tag oneway* com valor igual a "*yes"* ou "*true"* ou "1", ele é considerado como unidirecional. Outra possibilidade é que a *tag oneway* possua valor igual a "*-1"*. Nesse caso o caminho possui direção reversa. Caso o caminho possua as *tags "vehicle:backward"* ou "*motor_vehicle:backward"* com valor igual a "*no"* a direção é unidirecional; caso possua as *tags "vehicle:forward"* ou "*motor_vehicle:foward"* igual a "*no"* a direção é reversa.

Quadro 6 - direção de um caminho

Chave	Valor	Direção
oneway	yes	unidirecional
	true	
	1	
	-1	reverso
vehicle:backward	no	unidirecional
motor_vehicle:backward	no	
vehicle:forward	no	reverso
motor_vehicle:forward	no	
outros		bidirecional

Fonte: Elaborado pelo autor.

4.4.2.5 Criar e adicionar aresta no grafo

O penúltimo passo, linha 18, é criação da aresta a partir dos vértices criados e da direção identificada, e o último passo é a adição da aresta ao grafo

Figura 9 - Segunda etapa do algoritmo

```
1
2 while(exist((elementsOSM)){
3    if(elementOSM == CAMINHO){
4
5         if(valido(elementOSM)){
6             if(caminhoPossuirBarreiras(elementOSM)){
7                 elementOSM = redimensionarCaminho(elementOSM);
8             }
9             for(i =0; i < elementOSM.NODES(); i++){
10                 if(isPrincipalNode(elementOSM.getNode(i))){
11                    Node noDeOrigem = getNodeOrigem(elementOSM);
12                    Node noDeDestino = getNodeDestino(elementOSM);
13
14                    adicionarNosNografo(noDeOrigem, noDeDestino);
15
16                    Direcao direcaoCaminho = getDirecao(elementOSM);
17
18                    Aresta aresta(direcaoCaminho, noDeOrigem, noDeDestino);
19                    adicionarArestaGrafo(aresta);
20                }
21             }
22         }
23
24     }
25 }
26
```

Fonte: Elaborada pelo autor.

4.5 Implementar algoritmo

Nessa etapa do projeto o algoritmo foi implementado seguindo o modelo do passo anterior. Todo o código se encontra disponível *online*[30]. Para o desenvolvimento do projeto, foi utilizada a linguagem *Java* juntamente com a IDE (ambiente de desenvolvimento integrado) eclipse[31].

A Figura 10 mostra a estrutura do projeto, que foi divido em módulos. Para ser mais amplamente difundida a língua inglesa foi escolhida atribuir nomes aos artefatos. Cada módulo possui uma finalidade específica. O pacote *model* é responsável por representar os elementos do OSM, como, por exemplo, nós e caminhos, onde cada classe implementa métodos de manipulação dos elementos.

O módulo *Reader* contém duas classes, *PbfReaderExample* e *Reader*. A classe *PbfReaderExample,* como o nome sugere, é apenas um exemplo de como funciona a leitura de um arquivo PBF. A classe *Reader* é a principal classe do projeto, pois nela se encontra a implementação do algoritmo de geração do grafo. Essa classe recebe como atributo um arquivo PBF e o transforma em um grafo.

Para que o módulo *Reader* pudesse operar corretamente foi necessário criar um módulo auxiliar, *readerBasedOnOsmosis,* que é responsável por processar o arquivo PBF utilizando mecanismos de paralelização e transformando os dados binários em elementos OSM que estão modelados no pacote *model.* Vale ressaltar que esse pacote foi baseado na biblioteca OSMOSIS[32] criada pelo OSM, sofrendo modificações para operar no contexto do projeto.

Outro módulo importante é o *repository,* pois implementa classes que são responsáveis por manipular e armazenar estruturas que o módulo *reader* utiliza para auxiliar na geração do grafo. Por exemplo, a classe *NodeRepository* é responsável por armazenar os *ids* dos nós preprocessados na primeira etapa do algoritmo.

Para testar a implementação foi criado um módulo de teste, *test,* onde a classe *TestImport* recebe o caminho do arquivo PBF e instancia a classe *Reader invocando* logo em seguida o método para a geração do grafo.

30 https://github.com/carvalho7976/PbfToGraphastProject
31 https://eclipse.org/
32 https://github.com/openstreetmap/osmosis

Figura 10 - Estrutura geral do projeto

Fonte: Elaborada pelo autor.

4.6 Verificar resultados obtidos pelo algoritmo

Para verificar a conformidade do algoritmo a implementação foi testada em 14 mapas: Mônaco, Andorra, Azores, Montenegro, Albania, Moldova, New Hampshire, Serbia, Bulgaria, Croatia, Portugal, Slovakia, Denmark e Brazil. O resultado é apresentado em um estudo de caso elaborado no capítulo 5.

4.7 Integrar ao *framework*

A próxima etapa, após a implementação e verificação de conformidades dos resultados do projeto, é a integração com o *framework Graphast* . O projeto foi enviado à equipe de desenvolvimento para que a integração seja realizada.

5 ESTUDO DE CASO

Após o estudo e a implementação do algoritmo, o projeto foi testado em 14 regiões: Mônaco, Andorra, Azores, Montenegro, Albania, Moldova, New Hampshire, Serbia, Bulgaria, Croatia, Portugal, Slovakia, Denmark e Brazil. Tais regiões foram escolhidos devido aos diferentes tamanhos do arquivo PBF de seus mapas, pois facilitam a análise dos resultados e correções de possíveis erros. Os resultados foram comparados aos resultados obtidos pelo *framework GraphHopper,* citado no capítulo 2.

Os Quadros 6, 7, 8 , 9 e 10 mostram os nós obtidos através da execução do algoritmo em uma máquina com 6GB de memória RAM disponível e um processador core i7 2.5 GHz.

Quadro 7 - Resultado Mônaco

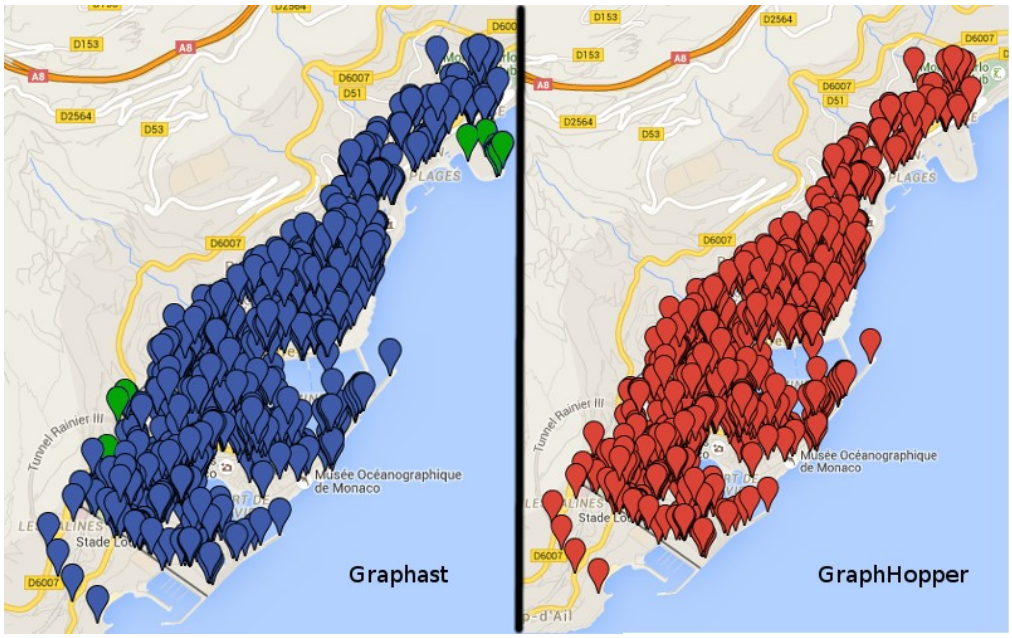
Framework	**Graphast**	**Graphhopper**
Mapa	Monaco	Monaco
Tamanho	320,2KB	320,2KB
Nº de Arestas	1337	1343
Nº de Nós	786	772
Memória	130.50 MB	130 MB
Tempo(segundos)	0,548	0,593

Fonte: Elaborado pelo autor.

O Quadro 6 mostra o resultado da geração do grafo utilizando o mapa de Mônaco. Com o serviço criado para o *framework Graphast f*oram geradas 1337 arestas e 786 nós , consumindo 130,5 MB de memória RAM em um tempo de 0,548 segundos. Através do *framework GraphHopper* foram gerados 1343 arestas e 786 nós, consumindo 130 MB de memória RAM em um tempo de 0,593 segundos. Neste teste ambos os algoritmos tiveram o consumo de memória e tempo de execução semelhantes.

A Figura 11 apresenta os nós obtidos em ambos os algoritmos para o mapa de Mônaco. Os pontos vermelhos indicam os nós obtidos pelo *framework GraphHopper*. Os pontos azuis indicam os nós que aparecem no algoritmo criado pelo autor. Os pontos verdes representam as disparidades encontradas, que são explanadas na subseção 5.2.

Figura 11 - Nós gerados em Mônaco pelo *Graphast* (azul) e pelo *GraphHopper* (vermelho)



Fonte: Elaborada pelo autor.

O Quadro 7 apresenta os resultados para o mapa de Andorra, onde o G*raphHopper* obteve um consumo de memória menor e a quantidade de nós e arestas geradas também foi menor.

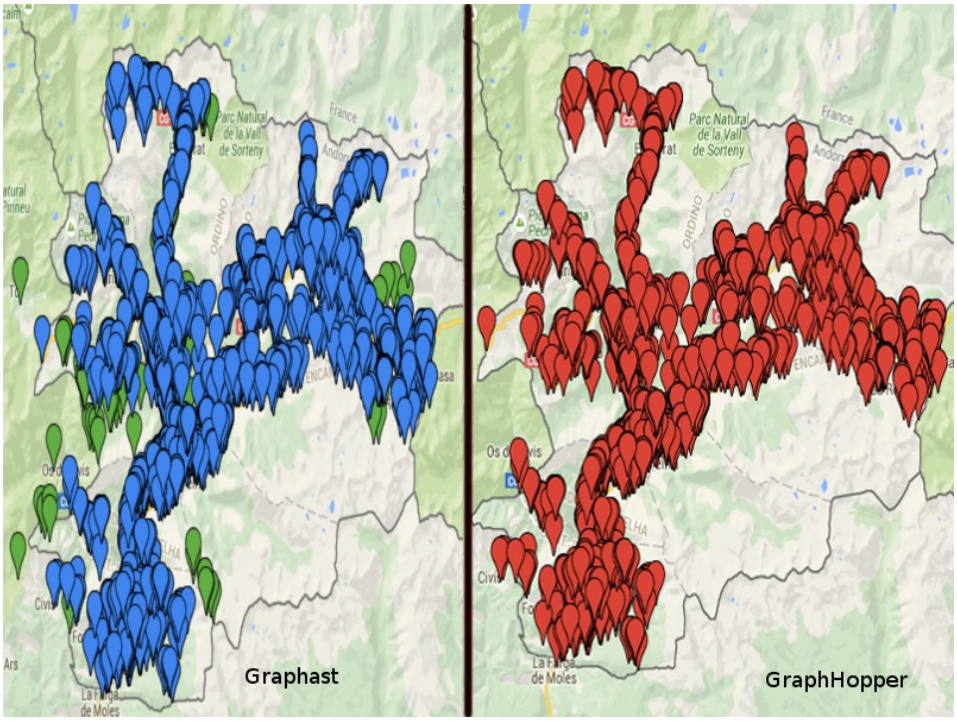
Quadro 8 - Resultado Andorra

Framework	**Graphast**	**Graphhopper**
Mapa	Andorra	Andorra
Tamanho	1.3MB	1.3MB
Nº de Arestas	7149	6817
Nº de Nós	3531	3390
Memória	219.00 MB	184.5 MB
Tempo(segundos)	1,147	1,186

Fonte: Elaborado pelo autor.

A Figura 12 apresenta os nós obtidos em ambos os algoritmos para o mapa de Andorra. Os pontos vermelhos indicam os nós obtidos utilizando o *framework GraphHopper*. Os pontos azuis indicam os nós que aparecem no algoritmo implementado pelo autor. Os pontos verdes representam as disparidades encontradas, que são explanadas na subseção 5.2.

Figura 12 - Nós gerados em Andorra pelo *Graphast* (azul) e pelo *GraphHopper* (vermelho)



Fonte: Elaborada pelo autor.

5.1 Comparação com o *framework GraphHopper*

O Quadro 8 apresenta o resultado da geração do grafo utilizando os mapas de Azores, Montenegro, Albania, Moldova, New Hampshire, Serbia, Bulgaria, Croatia, Portugal e Slovakia. Para cada região é feita uma comparação, entre o *framework GraphHopper e Graphast,* do numero de arestas, numero de nós, memória consumida, e o tempo levado para gerar o grafo.

5.1.1 Numero de nós e arestas

Nota-se que em todas as regiões o numero de nós e arestas, encontradas no *framework Graphast,* é maior em relação aos nós e arestas encontrados no *framework GraphHopper.* Tal disparidade é explicada na subseção 5.2.

Quadro 9 - Comparação de desempenho de tempo, consumo de memória, numero de nós e número de arestas

	Mapa	Tamanho	Nº de Arestas	Nº de Nós	Memória	Tempo (segundos)
Graphast	Azores	4,5 MB	44902	18866	448.00 MB	3,315
Graphhopper			43247	18230	513 MB	3,7
Graphast	Montenegro	14MB	83394	37003	860.50 MB	6,485
Graphhopper			81156	35863	664 MB	7,2
Graphast	Albania	18 MB	182977	79625	1181.00 MB	12,174
Graphhopper			181208	78346	1091 MB	13,416
Graphast	Moldova	23,6 MB	400062	149680	1261 MB	15,1
Graphhopper			396481	149017	1270 MB	15,282
Graphast	New Hampshire	31,7 MB	364437	149552	1583 MB	17,458
Graphhopper			354111	146212	1334 MB	16,576
Graphast	Bulgaria	64,4 MB	1028053	392532	2583.50 MB	44,138
Graphhopper			1014720	386627	2159 MB	31,575
Graphast	Serbia	47,1 MB	497051	204567	1981 MB	27,093
Graphhopper			487054	199151	1808 MB	17,801
Graphast	Croatia	84,8 MB	725432	314665	3305 MB	58,806
Graphhopper			658526	286937	2511 MB	27,347
Graphast	Portugal	135 MB	2144745	888477	4882 MB	352,048
Graphhopper			2117410	878211	4049 MB	72,439
Graphast	Slovakia	170,5 MB	741586	326492	4547.50 MB	182,388
Graphhopper			637215	279354	3716 MB	42,389

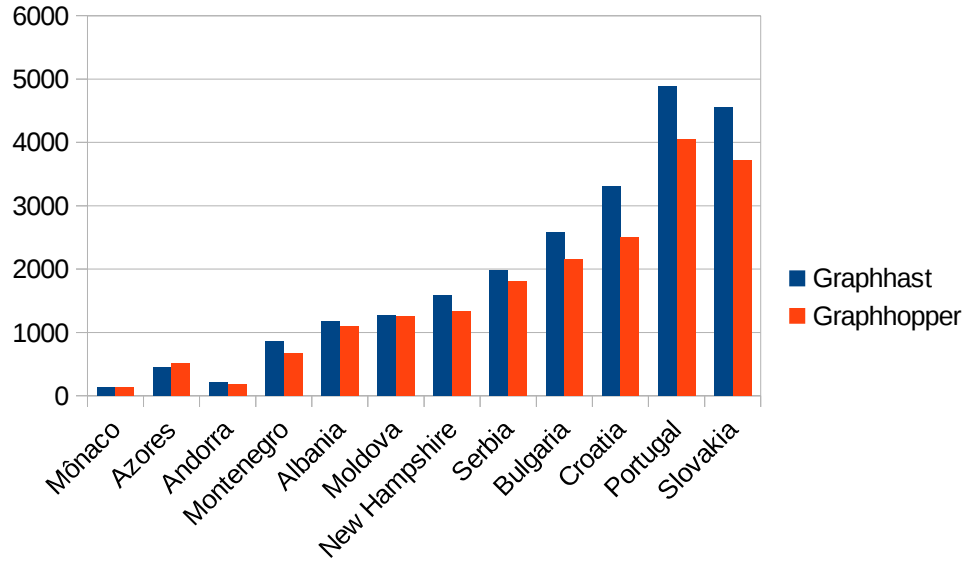
Fonte: Elaborado pelo autor.

5.1.2 Consumo de memória

A partir dos dados coletados um gráfico que compara o consumo de memória entre o *GraphHopper* e o *Graphast* foi gerado. O Gráfico 1 analisou os dados coletados das regiões de Mônaco, Andorra, Azores, Montenegro, Albania, Moldova, New Hampshire, Serbia, Bulgaria, Croatia, Portugal e Slovakia.

O eixo da vertical representa o consumo de memória em MB, o eixo horizontal mostra as regiões utilizadas. É possível perceber que o *framework GraphHopper* obtém um consumo de memória menor na maioria das regiões. Nota-se uma disparidade maior no mapa de Portugal onde a diferença de consumo entre os dois *frameworks* aproxima-se de 1000 MB.

Gráfico 1 - Comparação de consumo de memória entre *GraphHopper e Graphast*



Fonte: Elaborado pelo autor.

5.1.2 Tempo de processamento

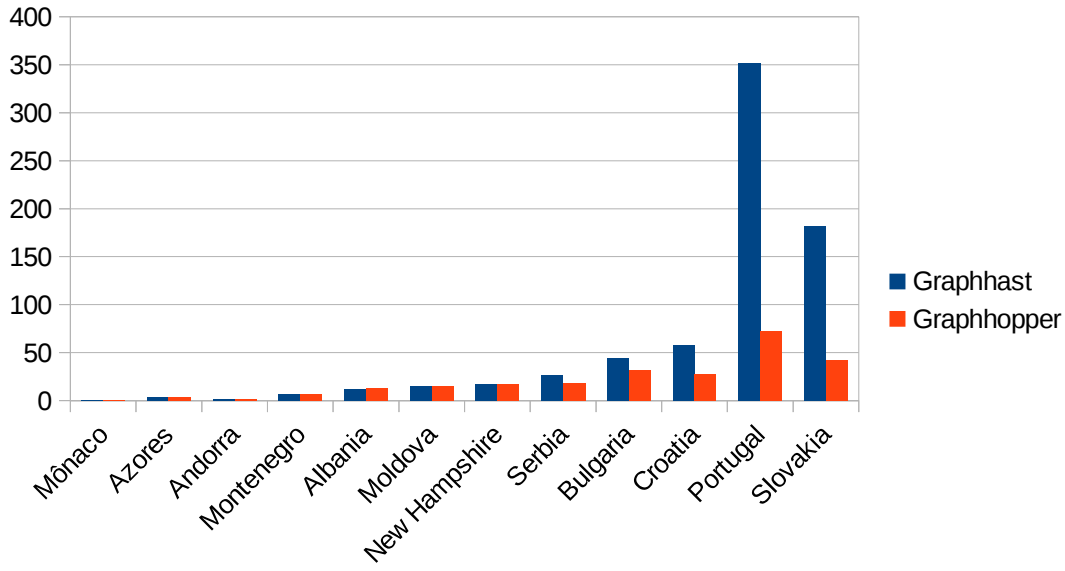
A partir dos dados coletados um gráfico que compara tempo de processamento entre o *GraphHopper* e o *Graphast* foi gerado. O Gráfico 2 analisou os dados coletados das regiões de Mônaco, Andorra, Azores, Montenegro, Albania, Moldova, New Hampshire, Serbia, Bulgaria, Croatia, Portugal e Slovakia.

O eixo da vertical representa o tempo de processamento, em segundos, o eixo horizontal mostra as regiões utilizadas. É possível perceber que o *framework GraphHopper* obtém um consumo de memória menor na maioria das regiões. Nota-se uma disparidade maior no mapa de Portugal onde o tempo de processamento atingiu os 350 segundos, através do *Graphast* e chegou a 70 segundos utilizando o *GraphHopper.*

Tal diferença está relacionada a quantidade de nós contidas na região, pois a região de Portugal, apesar de pequena em tamanho, possui o maior numero de nós dentre as regiões avaliadas. Tais nós são armazenados durante o processamento, o que acarreta um uso maior de memória e uma maior manipulação de objetos pelo *Graphast.* Outro fator que influencia no processamento é o tamanho do arquivo, pois significa uma maior quantidade de

elementos que são processados. O *GraphHopper* consegue manipular melhor suas estruturas, fazendo com o que o tempo de processamento se mantenha mais estável em comparação ao *Graphast.*

Gráfico 2 - Comparação tempo de processamento entre *GraphHopper e Graphast*



Fonte: Elaborado pelo autor.

5.2 Limitações

Foi verificado que o algoritmo gera o grafo parcialmente correto pois podem aparecer conjuntos de grafos desconexos, ou seja, um conjunto de vértices que não tem ligação com o grafo principal. Por exemplo, um estacionamento que contem vários caminhos internamente, porém é separado da rua através de um portão; nesse caso o roteamento só é possível até o portão, fazendo com que o grafo do estacionamento nunca seja atingido por alguma rota.

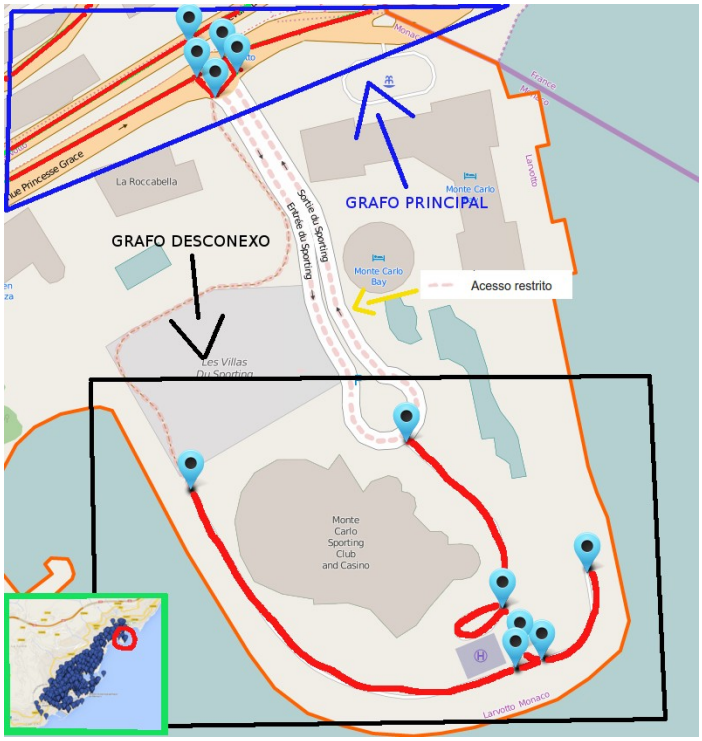
A Figura 13 mostra uma região, delimitada por um retângulo preto, em Mônaco onde existe um grafo desconexo e, portanto, não possui acesso ao grafo principal para seus vértices.

Os pontos azuis representam os vértices do grafo, ligados por arestas em vermelho. A área marcada com um triângulo azul indica vértices e arestas que fazem parte do grafo principal. É possível notar na imagem uma seta amarela que indica um caminho onde o tráfego é restrito e, portanto, não é possível fazer parte do grafo.

Os grafos desconexos ocorrem devido ao modelo de geração do grafo onde todos os caminhos são analisados e o grafo é construído pouco a pouco na ordem em que os caminhos são processados, ou seja, partes do grafo são criadas e unidas a medida que os nós e arestas são adicionados no grafo.

As consequências dos grafos desconexos podem ser vistas na subseção 5.1.1, onde os resultados mostram que o número de nós e arestas geradas pelo algoritmo são sempre maiores que as geradas pelo framework GraphHopper, que implementa mecanismos para remover tais grafos.

Figura 13 - Grafo desconexo



Fonte: Elaborada pelo autor.

Outra limitação encontrada é o consumo de memória RAM. Foi constatado que quanto maior o número de nós e o tamanho de um mapa, maior o consumo de memória. O Quadro 9 mostra o resultado do processamento da região da Dinamarca, que possui um tamanho de 208,4 MB. O *Graphast* não foi capaz de gerar um grafo a partir do arquivo pois o uso de memória excedeu a memória disponível na máquina, 6144 MB. Já o *GraphHopper* Foi capaz de processar o arquivo em um tempo de 69,3 segundos, gerando 809676 nós e 1808290 arestas, consumindo 5256 MB de memória.

Quadro 10 - Análise mapa Denmark

Framework	**Graphast**	**Graphhopper**
Mapa	Denmark	Denmark
Tamanho	208,4 MB	208,4MB
Nº de Arestas		1808290
Nº de Nós		809676
Memória	> 6144 MB	5256 MB
Tempo(segundos)		69,3

Fonte: Elaborado pelo autor.

Apesar de ter um gerenciamento de memória melhor, o *GraphHopper* também possui problemas para processar grandes arquivos. Foi testado o mapa do Brasil, que possui

374,5 MB, e ambos os *frameworks* não obtiveram sucesso ao processar o arquivo, pois a memória requerida para armazenar as estruturas era maior que a memória RAM disponível na máquina, 6144 MB.

Quadro 11 - Análise mapa Brazil

Framework	**Graphast**	**Graphhopper**
Mapa	Brazil	Brazil
Tamanho	374,5 MB	374,5 MB
Nº de Arestas		
Nº de Nós		
Memória	> 6144 MB	> 6144 MB
Tempo(segundos)		

Fonte: Elaborado pelo autor.

5.3 Dificuldades e lições aprendidas

Durante o processo de construção do trabalho foram encontradas algumas dificuldades. A primeira delas foi entender realmente os objetivos do trabalho, pois inicialmente o trabalho focou bastante em analisar os formatos disponibilizados pelo OSM quando na verdade o valor agregado do projeto final estava no processo de geração do grafo.

Outra dificuldade encontrada foi entender o domínio da aplicação através da análise da documentação do o*penstreetmap,* pois a mesma possui uma quantidade vasta de informações mas o processo de aprendizado do domínio é dificultado pela forma com que os dados estão organizados. Isso também acarretou em meses de análise e testes para a criação dos filtros, subseção 4.3.

Apesar dos percalços, foi possível aprender bastante sobre dados geoespaciais, além de um aprofundamento no conhecimento tecnológico e de pesquisa. Diga-se de passagem, é gratificante produzir conhecimento que pode ser utilizado para construção de ferramentas e que pode servir como base para futuras pesquisas.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho analisou os formatos de dados disponibilizados pelo OSM, onde um formato foi escolhido para ser usado na geração de um grafo de redes de ruas. Foi gerado também um modelo para criação de grafos a partir de redes de ruas do OSM. A partir desse modelo foi criado um serviço de importação de dados para um modelo de grafo provido pelo *framework Graphast.*

Como trabalho futuros, pretende-se corrigir as falhas apontadas na subseção 5.2, em que o grafo gerado contém mais nós e arestas que o necessário para roteamento. Através do estudo de caso, realizado no capítulo 5, mostrou-se necessário otimizar as estruturas que armazenam os nós, para que o consumo de memória seja reduzido e o tempo de processamento diminua.

Outro ponto a ser considerado para próximos trabalhos é calcular a distância entre os vértices e extrair os pontos de interesse no mapa, para que seja possível fazer pesquisas de locais, como por exemplos farmácias e restaurantes.

REFERÊNCIAS

AIHKISALO, T; PAASO, T. A performance comparison of web service object marshalling and unmarshalling solutions. *In:* SERVICES, 2011, Washington. **Anais…** Washington: IEEE, 2011, p. 122-129.

BAHRDT, D. Osmfind: fast textual search on OSM data--on sm atphones and servers. *In:* INTERNATIONAL WORKSHOP ON MOBILE GEOGRAPHIC INFORMATION SYSTEMS, 2013, Orlando. **Anais…** Orlando: ACM, 2013. p. 35-42.

COSTA, C. et al. A*-based solutions for knn queries with operating time constraints in time-dependent road networks. *In:* INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT, 15., 2014, Brisbane. **Anais…** Brisbane: IEEE, 2014. p. 23-32.

CRUZ, L; NASCIMENTO, M. A; MACÊDO, J. A. F. K-nearest neighbors queries in time-dependent road networks. **Journal of Information and Data Management**, v. 3, n. 3, p. 211, 2012.

ESRI. **SHAPEFILES.** Disponível em: <https://www.esri.com/library/whitepapers/-pdfs/shapefile.pdf/>. Acesso em: 26 dez. 2015.

GOOGLE. **Protocol Buffers.** Disponível em: <https://developers.google.com/protocol-buffers/>. Acesso em: 07 jun. 2016.

HAKLAY, M; WEBER, P. Openstreetmap: User-generated streep maps. **Pervasive Computing, IEEE,** v. 7, n. 4, p. 12-18. 2008.

KAUR, G; FUAD, M. M. An evaluation of protocol buffer. *In:* SOUTHEASTCON, 2010, North Carolina. **Anais…** North Carolina: IEEE, 2010, p. 459-462.

LUXEN, D; VETTER, C. Real-time rounting with OpenStreetMap data. *In:* INTERNATIONAL CONFERENCE ON ADVANCES IN GEOHRAPHIC INFORMATION SYSTEMS, 19., 2011, Chicago. **Anais…** Chicago: ACM, 2011, p. 513-516.

MAGALHAES, R. et al. Graphast: An extensible framework for building applications on time-dependet networks. *In:* INTERNATIONAL CONFERENCE ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 22., 2014, Dallas. **Anais…** Dallas, 2014, p.

SHARIFZADEN, M; KOLAHDOUZAN, M; SHAHABI, C. The optional sequenced route query. **The VLDB journal**, v. 17, n. 4, p. 765-787. 2008.

WHITE, C. E.; BERNSTEIN, D.; KORNHAUSER, A. L. Some map matching algorithms for personal navigation assistants. **Transportation Research Part C: Emerging Technologies, Elsevier**, v. 8, n. 1, p. 91–108, 2000.