UNIVERSIDADE ESTADUAL DO MARANHÃO
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE ENG. DA COMPUTAÇÃO

DENIS MORAES RÊGO CAMPOS

BIBLIOTECA DE CRIAÇÃO DE ROTAS ENTRE COORDENADAS EM AMBIENTES MARÍTIMOS NO SISTEMA BIO DATA.

São Luís
2017

DENIS MORAES RÊGO CAMPOS

BIBLIOTECA DE CRIAÇÃO DE ROTAS ENTRE COORDENADAS EM AMBIENTES MARÍTIMOS NO SISTEMA BIO DATA.

Projeto de Pesquisa vinculada aos Departamentos de Engenharia da Computação da Universidade Estadual do Maranhão com o objetivo qualificação.

Orientador: Prof. Dr. Fernando Jorge Cutrim Demetrio

São Luís

2017

Dissertação de Mestrado em Desenvolvimento de Sistemas e Aplicações.

Orientador: Fernando Jorge Cuntrim Demétrio.

Título em inglês: LIBRARY OF CREATION OF ROUTE BETWEEN COORDINATES IN MARITIME ENVIRONMENTS IN THE BIO DATA SYSTEM.

SUBTÍTULO

Dissertação apresentada como requisito para obtenção do título de Mestre em Desenvolvimento de Sistemas e Aplicações

Aprovação: ____/____/_____

Prof. Dr. _____

Prof. Dr. _____

Prof. Dr. _____

AGRADECIMENTOS

RESUMO

Este trabalho tem como objetivo apresentar a biblioteca de criação de rotas em ambientes aquáticos como mar, rios, lagos, entre outros. Sendo feito totalmente em javascript visando a total comunicação com a API do Google Maps e o framework Bootstrap, a biblioteca utiliza o algoritmo de busca A* heurística para criar três tipos de rotas (por peso, grupo e individual) com n pontos de partida para n pontos destinos, uso de polígonos para representar obstáculos e áreas de pesquisa que serão identificados pela API e pelo algorítmico de busca e identificação da melhor rota para acessar determinada coordenada. Para obtenção dos resultados, a biblioteca foi implementada no Sistema Bio Data que tem como função auxiliar o monitoramento do Biota Aquático, Pluma de Sedimentos, Hídricos e Sedimentos na Baía de São Marcos e Baia do Arraial com os pontos de monitoramento disponibilizados pela EMAP.

Palavras-chave: Rotas. Maps. Busca.

ABSTRACT

This work aims to present the library of route creation in aquatic environments such as sea, rivers, lakes, among others. Being made entirely in javascript for full communication with the Google Maps API and the Bootstrap framework, the library uses the heuristic search algorithm A* to create three types of routes (by weight, group and individual) with n starting points for n points, polygons to represent obstacles and areas of research that will be identified by the API and algorithm of search and identification of the best route to access a given coordinate. To obtain the results, the library was implemented in the Bio Data System whose function is to monitor the Maritime Life, Sediment Plume, Water and Sediments in the Bay of São Marcos and Bay do Arraial with the monitoring points provided by EMAP.

Keywords: Routes. Maps. Search.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API = Application Programming Interface

BD = Datebase / Banco de dados

CSS = Cascading Style Sheets

DARPA = Agência de Projetos de Pesquisa Avançada dos Estados Unidos

EMAP = Empresa Maranhense de Administração Portuária

E/S = Entrada / Saída

HTML5 = HyperText Markup Language 5

HTTPS = Hyper Text Transfer Secure

IBM = International Business Machines

IP = Protocolo de Internet

PHP = Personal Home Page/ Hypertext Preprocessor

POO = Programa orientado ao objeto

SGBDR = Sistema de Gerenciamento de Dados Racional

SQL = Structure Query Language

TCP = Protocolo de controle de Transmissão

UML = Unified Modeling Language

www = Wolrd Wide Web

ZF = Zend Framework

LISTA DE SÍMBOLOS

Φ = latitude

λ = longitude

π = pi

SUMÁRIO

1 INTRODUÇÃO

O porto do Itaqui, integra o segundo maior complexo portuário em movimentação de carga do país, envolvendo cargas como alumínio, soja, fluoreto, cobre, carvão, ferro-gusa, entre outros, a fim de garantir o atendimento de novos mercados, a expansão da infraestrutura portuária torna-se necessário. Devido à grande ativação do porto e planejamentos de extensões na infraestrutura, ancorados pela valorização ambiental, a Empresa Maranhense de Administração Portuária – EMAP visa minimizar qualquer dano a natureza ocasionada pelas ações do porto do Itaqui, para isso, são realizadas análises referentes a amostras coletadas em áreas determinadas para pesquisa.

No intuito de agilizar, acessibilizar, organizar os dados adquiridos na coleta, adjunto na apresentação dos resultados, motivados pelo interesse partilhado da EMAP em preservar a natureza, foi desenvolvido o sistema Bio Data. A ideia de agilizar as pesquisas referentes ao monitoramento da água vem do princípio de criar medidas ou novas políticas, onde os dados provam a poluição do mesmo, pois trata-se de problema que assola todo o território Brasileiro, conforme pesquisas realizadas pela InfoAmazonia, Fundação SOS Mata Atlântica e pela Secretaria de Estado do Meio Ambiente e Recursos Naturais, nos quais soluções deverão ser gerados a partir detalhados dos dados.

O sistema Bio Data utiliza o framework bootstrap para melhorias na experiência do usuário, a fim de adaptar-sxe ao tamanho de tela do aparelho e browser utilizado. Feito principalmente em PHP e javascript, o sistema visa a melhor performance nos aparelhos mobiles, utilizando métodos como utilização do javascript para a comunicação entre páginas, recarregamento parciais na página, ocasionando maior fluidez e evitando excessos de processamento.

O Sistema foi desenvolvida a fim de auxiliar as pesquisas realizadas por empresas e pesquisadores que visam a sustentabilidade do meio ambiente com ênfase a dados hídricos e sedimentos, biota aquática e dispersão da pluma de sedimentos, acompanhando todas as etapas necessárias para o monitoramento, desde o preparativo para a pesquisa de campo à emissão de relatório. Neste trabalho será apresentado algumas funcionalidades do sistema, destacando o algoritmo de rotas desenvolvida especificamente para áreas marítimas, visando

diminuir os custos com embarcações, gerando a distâncias entre pontos de coletas com mais precisão e melhores rotas para acessar o máximo de pontos em uma única viagem, para isso foi utilizado o algoritmo A* em conjunto com algumas funcionalidades do Google Maps.

2 COMPARAÇÕES COM TRABALHOS RELACIONADOS

Este capítulo será apresentado trabalhos que serviram como base e/ou comparação para o desenvolvimento do algoritmo de busca, serão apresentados 5 artigos científicos de diferentes fontes de pesquisa.

2.1 UM SISTEMA WEB DE CONSULTA DE TRAJETOS DE TRANSPORTE PÚBLICO

Sistema web cliente-servidor que utilizado para gerar informações ao usuário sobre o transporte público utilizando algoritmo A*, tendo como base a localização inicial e um destino final, com isso o sistema calcula as linhas de ônibus que o usuário deverá utilizar.

Apesar do sistema assemelha-se com algumas características do sistema Bio Data, como a base cliente-servidor, o uso do algoritmo A*, todavia se divergência em sua aplicação, pois no sistema batizado de Antares possui um facilitador que são caminhos já definidos, uma vez que ruas limitam a área e direção para onde a rota pode passar, enquanto a biblioteca colocada no Bio Data, não possui tal facilitador, todos os obstáculos, caminhos e direção são descobertos na criação das rotas, vale ressaltar que o Google Maps (até o ano 2017) já possui ferramentas semelhantes que criam rotas de ônibus e a pé tendo as mesmas necessidades da localização inicial e o destino, todavia limita-se em áreas terrestres, não conseguindo realizar se as localizações se encontrarem em lagos, mar, lagoa, retornando sempre uma rota por terra.

2.2 APLICAÇÃO DE UM ALGORITMO GENÉTICO PARA O PROBLEMA DO CARTEIRO CHINÊS EM UMA SITUAÇÃO REAL DE COBERTURA DE ARCOS.

Trabalho feito para solucionar o problema do Carteiro Chinês que é um problema de otimização com objetivo de cobrir todos os arcos de um grafo, minimizando a distância total, com isso este trabalho serviu como fonte de pesquisa para solucionar a criação de rotas marítimas, pois um dos objetos é traçar uma rota de classificação boa no menor tempo possível, passando pelas coordenadas uma única vez, integrando a ideia de grafos e possibilitando a comparação entre algoritmos genéticos e algoritmos de busca para o desenvolvimento da biblioteca

conforme os resultados adquiridos na análise dos trabalhos pesquisados, todavia foi utilizado um ambiente onde os vértices do grafo já estavam ligados e cada aresta já possuía o seu peso.

2.3 NAVEGAÇÃO AUTÔNOMA DE UM AGENTE INTELIGENTE: UM ESTUDO COMPARATIVO USANDO LÓGICA FUZZY E ALGORITMO DE BUSCA A*.

Trabalho feito para comparar resultados acerca de duas técnicas de inteligência artificial no processo de tomada de decisão no mesmo ambiente de criação de rotas, tendo como resultado, o algoritmo de busca A* sendo mais eficiente, conseguindo o menor custo para solucionar o problema com medias de 6.1, enquanto Fuzzy atingiu os 7.3, levando em consideração as mesmas regras. Este trabalho serviu como base de estudo na implementação do algoritmo de busca A*, diferenciando-se do ambiente utilizado, uma vez que os vértices não possuem ligação com todos os vértices.

2.4 UN NUEVO ALGORITMO HEURÍSTICO PARA LA CREACIÓN DE RUTAS TURÍSTICAS PERSONALIZADAS

Artigo com o propósito de apresentar um algoritmo heurístico para criar rotas em pontos turísticos, foi utilizado como base de estudo, pois o ambiente de pesquisa assemelha-se nas seguintes características: 1 – o algoritmo não conhece onde serão adicionados os pontos, 2 – uma vez adicionado ele verificará todos os pontos adicionados e criar uma rota entre eles, 3 – o algoritmo deverá passar uma única vez em cada ponto e no menor caminho mais curto, utilizando a análise do problema do caixeiro viajante, para isso o artigo mostra duas formas de criação semelhantes à biblioteca desenvolvida, que é por peso, tendo como principal critério o peso de cada ponto e a segunda forma é agrupar os pontos para trabalhar por partes ao invés do todo, apesar de calcular também o tempo médio (trabalho futuro para a biblioteca) o ambiente tem o facilitador que é os caminhos direcionado graças as ruas e avenidas, algo não presente no ambiente onde a biblioteca trabalha.

2.5 AN IMPROVED DIFFERENTIAL EVOLUTION ALGORITHM FOR MARITIME COLLISION AVOIDANCE ROUTE PLANNING

Artigo que mostra resultados de um algoritmo que visa evitar colisões em ambientes marítimos, utilizando o MATLAB para criar um algoritmo evolutivo diferencial focado exclusivamente em criar uma rota para evitar a colisão do navio com algum obstáculo, mostrou as dificuldades que foram confirmadas no desenvolvimento da biblioteca que é a escala do mapa, pois quanto maior a escala maior o processamento, todavia, melhor é a precisão no resultado, com isso é necessário criar o meio termo entre as duas variantes, servindo como base para o desenvolvimento da biblioteca.

2.6 TRAJECTORY PLANNING WITH NEGOTIATION FOR MARITIME COLLISION AVOIDANC

Neste artigo é representado um algoritmo capaz de prevenir colisões entre navios calculando a trajetória que o navio e os outros navios poderão pegar com base a trajetória atual e velocidade, utiliza o algoritmo A* para realizar o desvio de colisão, com isso, este artigo foi utilizado como estudo para a biblioteca desenvolvida, pois a ideia utilizada para realizar o desvio é semelhante à desejada quando a biblioteca estava na etapa de levantamento de requisitos, pois utiliza a verificação das células criadas na rota futura que os navios pegarão, onde caso uma célula gerada na rota pertence a uma ou mais navio, o algoritmo desviará um dos navios para uma célula de apoio que não pertence à rota original, no caso da biblioteca em vez de criar uma rota futura, ela é substituída pelos polígonos de colisão e verifica se é encontrado pela rota, caso seja a célula de apoio será o perímetro que forma o polígono.

2.7 DESENVOLVIMENTO E INTEGRAÇÃO DE MAPAS DINÂMICOS GEORREFERENCIADOS PARA O GERENCIAMENTO E VIGILÂNCIA EM SAÚDE

Neste artigo foi mostrado uma arquitetura de um sistema georreferenciado utilizando o Google Maps API que serviram como estudo para a adaptação da biblioteca desenvolvida no sistema Bio Data, fazendo-o se comunicar

com o banco de dados do sistema, realizando cadastros e edições dos pontos de coleta, polígonos e áreas de pesquisa

2.8 Avaliação de Heurísticas de Melhoramento e da Metaheurística Busca Tabu para Solução de PRV

Artigo utilizado como base de pesquisa para tentativa de solucionar o problema do retorno da rota ao polígono e comparações com os resultados obtidos com o algoritmo A* utilizado na biblioteca com o algoritmo de busca tabu para desenvolvimento de rotas heurística implementado no artigo, uma vez que ambos utilizando o princípio do “vizinho mais próximo”, buscando o vértices mais próximo do atual, seguindo aos mais distantes a fim de comparação de resultado.

A lista tabu serviu como inspiração para complementar o algoritmo A* na biblioteca, uma vez que ela conseguiu resolver o problema do retorno da rota ao polígono que será mostrado no decorrer deste trabalho.

2.9 COORDINATED ROAD-NETWORK SEARCH ROUTE PLANNING BY A TEAM OF UAVS

Utilizando como base o problema do carteiro chinês e caixeiro viajante, este artigo foi utilizado como estudo da criação de rotas utilizando a teoria de Dubins visando o menor poder computacional e melhores resultados para os múltiplos veículos aéreos não tripulados consigam visitar todos os pontos, utilizando como ênfase as rotas rodoviárias, para isso foi necessário mapear determina região, pegando todas as ruas, avenidas e estradas, nesse ponto esse algoritmo possui um facilitador em comparação ao ambiente que a biblioteca desenvolvida é aplica, pois a existência de ruas limitam a área que a rota passará, seguindo apenas o desenho feito entre elas.

2.10 CRIAÇÃO DE UM MÓDULO DE BASE GEOGRÁFICA PARA A GESTÃO COMERCIAL DE CLIENTES USANDO SERVIÇOS GOOGLE MAPS

Artigo mostra a utilização do Google Maps API em uma aplicação web

cliente-servidor, foi utilizado como estudo das classes contidas na API, entre elas a utilização dos polígonos, marcadores e linhas, as mesmas utilizadas pela biblioteca desenvolvida, auxiliando na estrutura da API, dificuldades encontradas, hierarquia das classes e modificações e funções permitidas em cada classe, todavia a biblioteca desenvolvida necessitou de mais modificações não conseguindo aproveitar todas as funcionalidades da API, isso ocorreu porque a API não consegue realizar as rotas marítimas automaticamente, ela sempre buscará uma rota por terra, ou retornando valores nulos.

3 METODOLOGIA E FERRAMENTAS UTILIZADAS

Neste Capítulo abordaremos a fundamentação teórica utilizado para o desenvolvimento do projeto, destacando uma parte de sua história e características, seguindo a seguinte ordem: engenharia de *software*, programação *web*, linguagem de programação e *database*.

3.1 Engenharia de Software

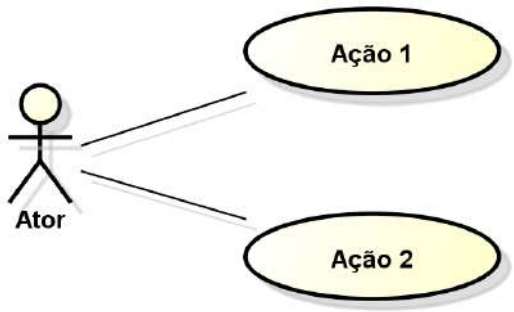
Em todo o desenvolvimento de um *software*, é necessário desenvolver em paralelo um conjunto de documentos referentes ao *software*, descrevendo todas as funcionalidades, especificações, dentre elas encontramos os Diagramas capaz de representar visualmente a estrutura do sistema. Existem vários tipos de Diagramas, neste projeto destacaremos o Diagrama de Caso de Uso, Classe e Sequência.

3.1.1Caso de Uso

De acordo com Gilleanes Guedes (2011), o Diagrama de Caso de Uso é aplicado para visualização de ações que cada ator pode realizar, sendo possível seu uso para representar sistemas, usuários, empresas, entre outros. Um caso de uso é a descrição de um conjunto de ações, incluindo variantes, que o sistema realiza para chegar a um resultado de valor observável para um ator.

Sua utilização está ligada ao mapeamento dos papéis de cada ator, quais atividades possíveis poderão ser realizadas por eles, desta forma podemos compreender como o sistema trabalha e analisar a importância dos atores, conforme a Figura 1

Figura 1. Representação do diagrama de caso de uso feito no Astah

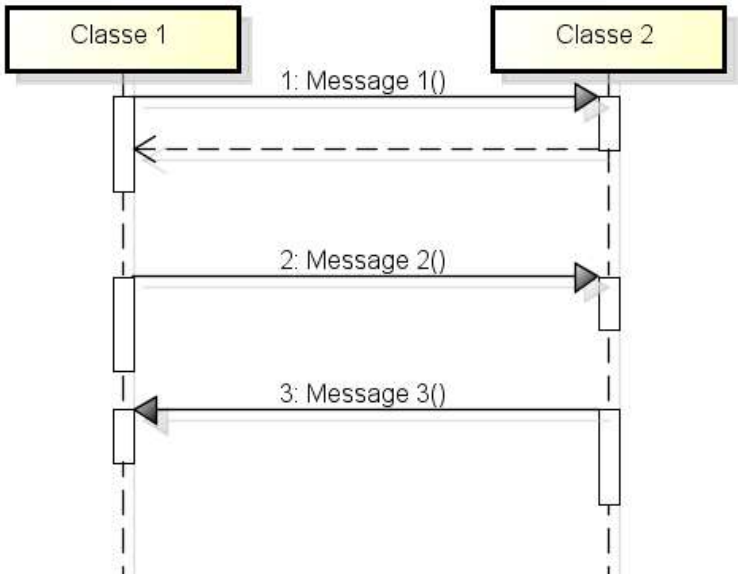


Fonte: Campos, 2017

3.1.2 Sequência

De acordo com Gilleanes Guedes (2011) este Diagrama representa a sequência de comunicação entre duas ou mais instâncias de classes, componentes, subsistemas, entre outros. Sua prioridade está na ordenação das mensagens, organizando o tempo de cima para baixo, ou seja, quando mais acima no Diagrama menor é o tempo de espera para sua execução, dessa forma podemos verificar como determinada ação será realizada pelo Sistema, conforme a figura 2.

Figura 2. Representação do diagrama de sequência feito no Astah

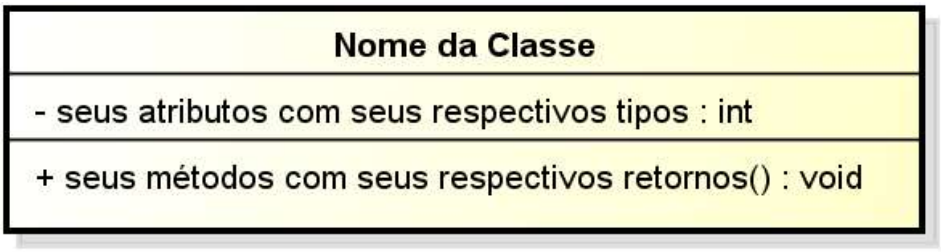


Fonte: Campos, 2017

3.1.3 Classe

De acordo com Gilleanes Guedes (2011) afirma que o diagrama de Classe é provavelmente o mais utilizado e um dos mais importante, pois serve como apoio aos demais diagramas. O diagrama de Classe define a estrutura das classes (seu nome, atributos e métodos) encontradas no sistema e como elas se relacionam, a representação das classes assemelha-se a quadros, cada quadro é dividido em 3 partes, seguindo essa ordem: na parte de cima o nome da classe, seu meio com os atributos e na parte mais inferior os métodos, conforme a Figura 3.

Figura 3. Representação da classe no diagrama de classe feito no Astah



Fonte: Campos, 2017

3.2 Sistema Georreferenciado

Sistemas utilizados para mapear determinadas áreas utilizando imagens, mapas, ou qualquer outra forma de informação geográfica, utilizando coordenadas geográficas como base para realizar ações como marcações, monitoramento, criações de rotas, entre outros, podemos citar alguns sistemas que possuem algumas dessas funcionalidades como o aplicativo do Uber, Facebook, Twitter, Google Maps, Correios, normalmente os sistemas georreferenciados possuem algum tipo de conexão com a internet e/ou servidores, principalmente para estabelecer o compartilhamento das informações e atualizações, como exemplo, um algoritmo de busca de rotas que se encontra no aplicativo do Uber deverá ter todas as ruas e avenidas atualizadas, pois se isso não ocorrer, poderá prejudicar tanto a própria empresa como os motoristas e usuários do aplicativo.

Esses sistemas podem utilizar as seguintes ferramentas de localização: GPS (sistema de posicionamento global, utilizando a posição dos satélites para determinar a localização), A-GPS (GPS assistencial, usando torres de telefones celulares e postes públicos, além dos satélites para determinar a localização do usuário), hotspots Wi-fi, endereços IP conhecidos, entre outros, que são utilizados conforme a necessidade da aplicação e o suporte do aparelho utilizado, essas tecnologias são normalmente utilizados em conjunto com uma API(Application Programming Interface) georreferenciada como exemplo o Google Maps API e o SAPO.

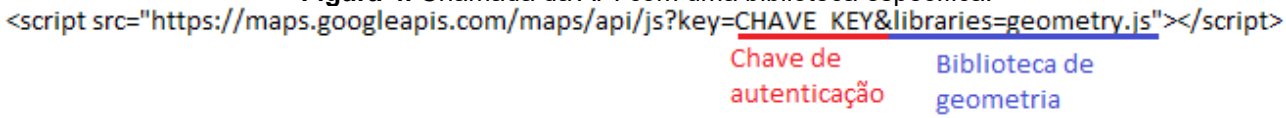
3.3 Google Maps API

Lançado pela Google em 2005, é uma API voltada para mapas, incluindo imagens por satélites que servem como base para os mapas. Sendo compatível com

diversos Sistemas operacionais como OS, Android e para navegadores utilizando o javascript.

Dentro da API, encontramos ferramentas capazes de atender necessidades dos usuários, como a criação de rotas entre endereços, função de GPS(Global Positioning System), monitoramento do tráfico de veículos, comentários e qualificações de lugares feitos pelos usuários, também possibilita que desenvolvedores utilizem as bibliotecas contidas na API a partir de uma obtenção de uma chave, método que a Google adotou para autenticar e liberar na aplicação, a chave é passada no momento que é feito a chamada das bibliotecas junto a API, conforme a Figura 4, cada biblioteca possui sua própria classe, permitindo todas as funcionalidades descritas anteriormente e personalizá-las de maneira que os desenvolvedores desejarem.

Figura 4. Chamada da API com uma biblioteca específica.



Fonte: Campos, 2017

Exemplos as classes utilizadas nesse trabalho:

1 – google.maps.Map: utilizado para carregar a API em determinado local da página, por padrão da Google, em uma tag div do HTML(HyperText Markup Language) adicionando uma id de nome “map”, o objeto criado a partir dessa classe, será setado em classes como google.maps.Marker, google.maps.Polyline e google.maps.Polygon. Possui parâmetros que altera a forma de exibir os mapas, como exemplo, zoom (é a aproximação da câmera em uma determinada localidade, center(localidade determinada por valores de latitude e longitude que a câmera focará assim que a API for totalmente carregada), mapTypeId (tipo de mapa que será carregado, como imagem por satélite, mapa de calor, mapas, entre outros).

Figura 5. Criação do objeto da classe google.maps.Map.

```
map = new google.maps.Map(document.getElementById('map'), {
    zoom: 14,
    center: {lat:-2.5680909, lng: -44.3812118},
    mapTypeId: google.maps.MapTypeId.SATELLITE

});
```

Fonte: Campos, 2017

2 – google.maps.LatLng: utilizado para transformar coordenadas geográficas, latitude (distância ao Equador medida em graus ao longo do meriano de Greenwich, variando entre -90 e 90) e longitude (medida em graus ao longo do Equador entre um ponto e o Meridiano de Greenwich, variando entre -180 e 180) em um objeto legível em outras classes da API, por padrão a latitude é representado pela coordenada “y” ou abreviação “lat” e será sempre escrita primeira, enquanto a longitude é representado por “x” ou abreviação “lng”;

Figura 6. Criação do objeto da classe google.maps.LatLng.

```
var point = new google.maps.LatLng(lat, lng);
```

Fonte: Campos, 2017

3 – google.maps.Marker: utilizado para a criação dos marcadores dentro da API e suas características como sua posição (geolocalização em latitude e longitude), imagem de representação (por padrão terá um marcador da cor vermelha), id de identificação, objeto google.maps.Map (este atributo insere o marcador na API), entre outros, que poderão ser inseridos no objeto criado com esta classe, como visto na figura 7;

Figura 7. Google.maps.Marker.

Fonte: Campos, 2017

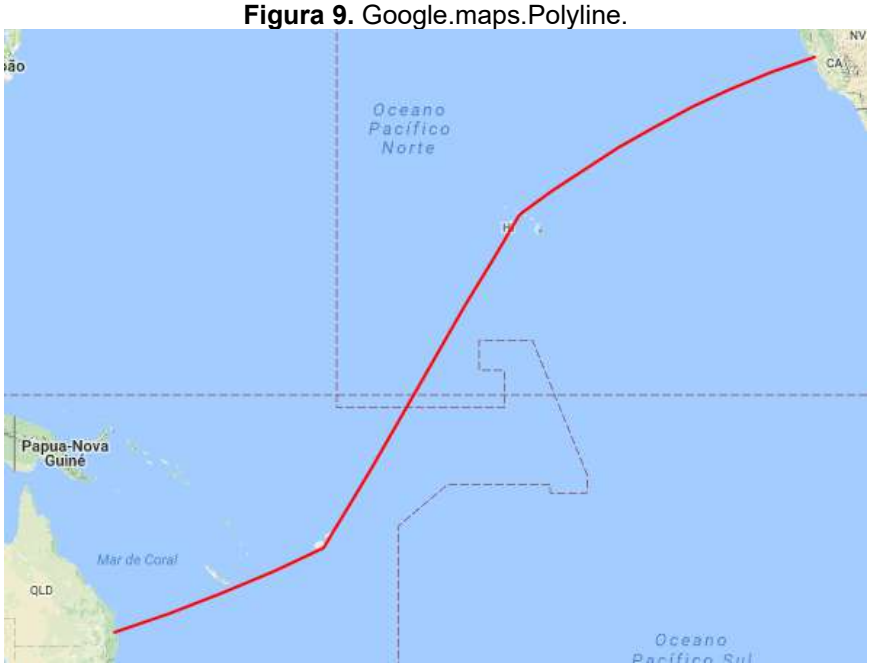
4 – google.maps.InfoWindow: seu objeto é utilizado para a criação de janelas de informações referente a determinado objeto que ela foi inserida, como apresentado na figura 8, um objeto InfoWindow setado em um objeto do tipo Marker;

Figura 8. Google.maps.InfoWindow.

Fonte: Campos, 2017

5 – google.maps.Polyline: seu o objeto é utilizado para criar linhas entre segmentos de uma ou mais coordenadas, podendo representar rotas, distâncias, entre outros, assim como na google.maps.Marker, é possível setar características ao seu objeto, como a grossura da linha, cor, opacidade, ações, objeto google.maps.Map, entre outras, porém para o objeto desta classe seja visto pela API, é necessário setar seu path que é um vetor de objetos google.maps.LatLng que serão conectados em uma ordem sequencial, ou seja, o path traça o caminho que o objeto Polyline realizará, como representado na figura 9;

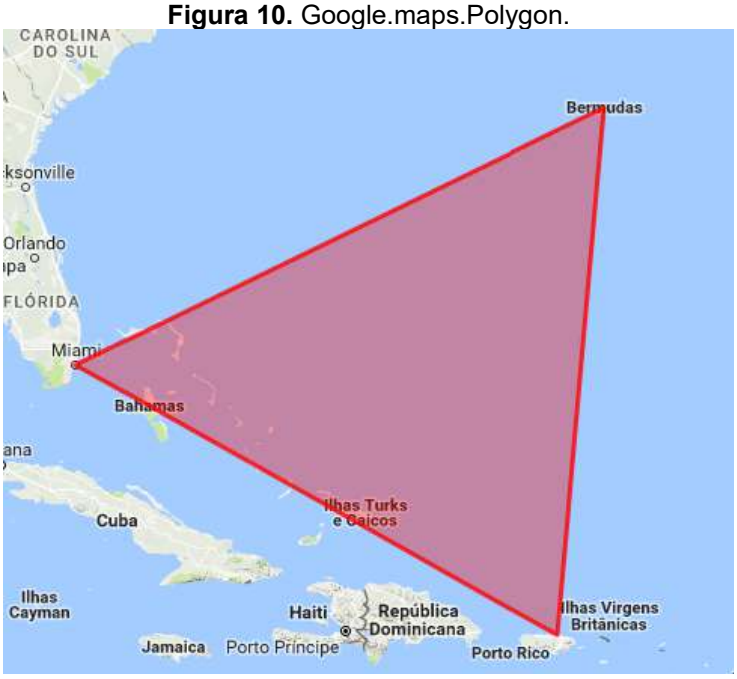
Figura 9. Google.maps.Polyline.



Fonte: Google, 2017

6 – google.maps.Polygon: o objeto desta classe é muito semelhante ao objeto Polyline, diferenciando-se em alguns aspectos, como exemplo, em seu path, sempre conectará o primeiro e o último elemento do vetor, formando um circuito fechado, definindo uma região que se tornará um polígono conforme a figura 10, com isso é possível trabalhar utilizando a área criada, utilizando funções como: google.maps.geometry.poly.containsLocation (verifica se determinada coordenada pertence ao polígono) e google.maps.geometry.poly.isLocationOnEdge (verifica se o ponto perdente ou está em uma distância de 10^{-9} graus de tolerância ao perímetro do polígono).

Figura 10. Google.maps.Polygon.



Fonte: Google, 2017

3.4 Algoritmo de Busca

Algoritmo de Busca são técnicas de inteligência artificial utilizados para resolver problemas de alta complexidade que podem apresentar diferentes caminhos para a solução desejada, como exemplo a melhor rota para chegar em um determinado endereço. Os caminhos partem de um determinado estado de inicial, passando por sucessões de estados que levam até o estado objetivo.

Russel e Norving destacam quatro elementos que pode definir um problema de busca (estado inicial, sucessores, custo do caminho e teste objetivo). Estado inicial: representa o ponto de partida, o estado inicial; Sucessores: conforme

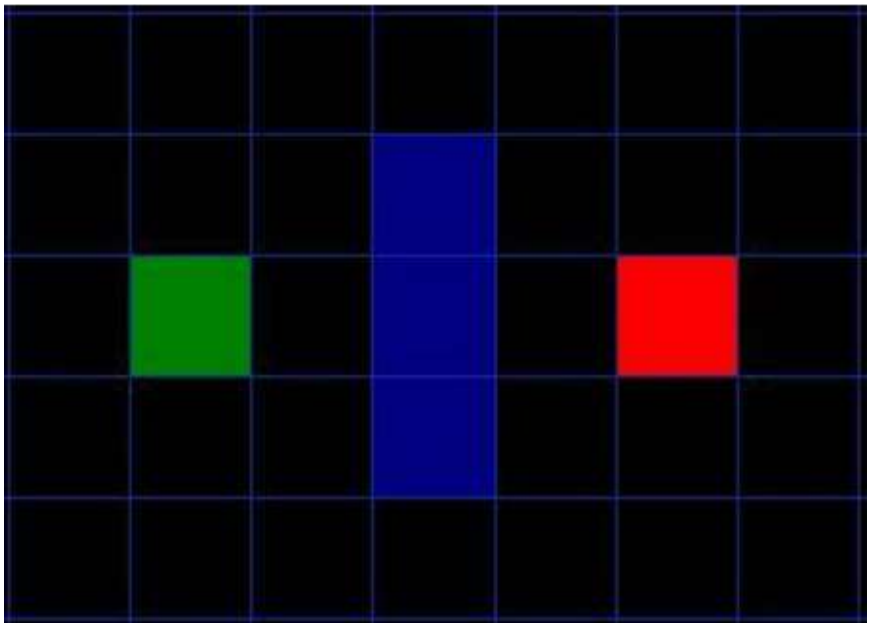
avançamos do nosso estado inicial, passamos por diversos outros espaços, por exemplo, quando saímos de casa(estado inicial) para ir para a faculdade(estado final), é necessário passar por ruas, avenidas, entre outros, cada elemento citado corresponde a um sucessor do nosso estado inicial; Custo do caminho: seguindo o exemplo, cada sucessor possui um custo/medida de desempenho para poder acessá-lo, por exemplo, em nosso ponto inicial podemos pegar o caminho “A” ou “B”, cuja a distância é de tês e cinco quilômetros respectivamente, ambos possuem a mesma distância para a faculdade, o algoritmo deverá processar e considerar qual caminho é o mais adequado para se tomar, visando nosso ponto final, nesse caso como ambos os caminhos podem ser usado para chegar na faculdade, o algoritmo escolheria o caminho “A”, por ter um peso menor que “B”.

Seguindo os estudos de Russel e Norving, é necessário realizar medições de desempenho do algoritmo de busca, enfatizando sua completeza, otimização, complexidade de tempo e espaço. Completeza: o algoritmo garante em solucionar o problema; Otimização: a solução encontra é considerada ótima ou a melhor possível (a escolha do caminho “A” para chegar na faculdade, como mostrado anteriormente); Complexidade de tempo e espaço: refere-se ao tempo gasto pelo algoritmo para solucionar o problema e a quantidade de memória utilizada em sua execução.

3.4.1 Algoritmo de Busca A*

Iniciado em 1964 por Nils Nilsson com algoritmos heurísticos (algoritmos que fornecem soluções sem um limite formal de qualidade) visando o melhoramento do algorítimo de Dijkstra, em seguida melhorado em 1967 por Peter Hart e em 1968 por Bertram Raphel novamente aperfeiçoado e batizado de A* ou estrela, é um dos métodos mais populares para encontrar o caminho mais curto entre dois ou mais locais em uma área mapeada, onde poderá existir obstáculos entre as localidades como representado na figura 11, onde o quadrado em verde representa a localidade de onde partimos, em vermelho ponto de chegada e em azul representa um obstáculo entre eles.

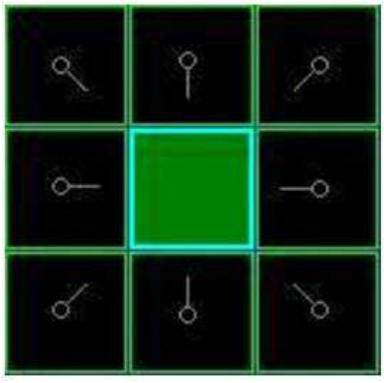
Figura 11. Situação para o algoritmo A*



Fonte: Patrick Lester, 2004

O algoritmo trabalha com nós abertos e fechados, onde os nós abertos são aqueles que o algoritmo tem o conhecimento de sua existência, ou seja, aqueles que foram encontrados pela função heurística, porém ainda não fazem parte da lista que forma o caminho percorrido, enquanto os nos fechados são aqueles que já fazem parte da lista, como apresentado na figura 12, onde o quadrado em verde representa um nó adicionado na lista, enquanto os demais são nós descobertos pelo algoritmo.

Figura 12. Nós visitados e não visitados



Fonte: Patrick Lester, 2004

Para a criação das rotas, o algoritmo buscará o menor custo entre os nós não visitados conforme apresentado na figura 12, em seguida o nó será adicionado

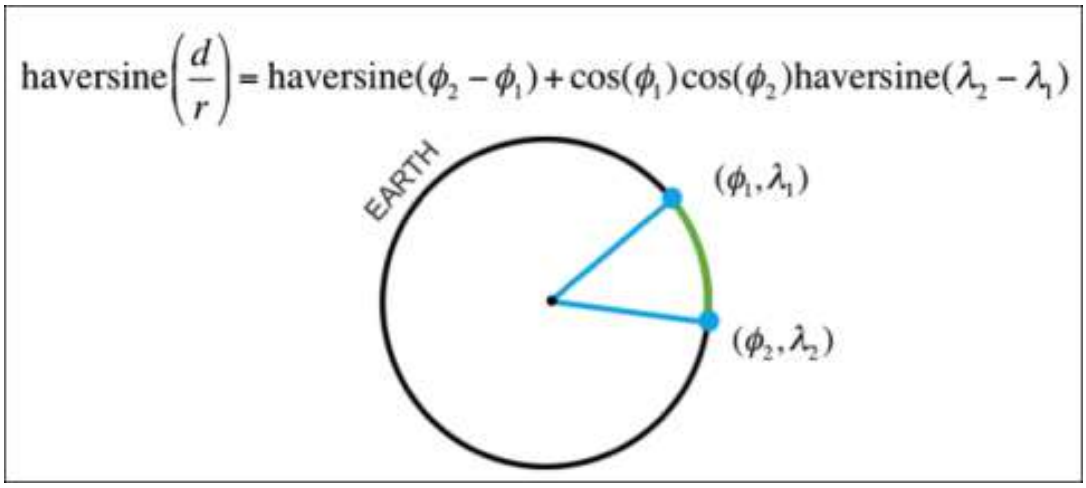
na lista e marcado como visitado, então o custo total é recalculado e é alterando o nó atual (último nó adicionado na lista, também representa a localização atual) que passa a ser o nó adicionado, com a nova localização do nó atual, novos nós serão encontrados, enquanto algoritmo não adicionar o nó destino na lista ou se existir nós não visitados, essas operações serão realizadas, por fim o algoritmo retornará a menor rota ao destino ou um fracasso caso seja impossível chegar ao nó destino.

O algoritmo de busca A* tem como representação a função f(n) = g(n) + h(n), onde f(n) é custo estimado da solução de custo mais baixo passado por n, g(n) é o custo para alcançar cada nó (custo percorrido) e h(n) o custo para ir do nó atual ao nó destino (custo estimado), também chamada de “função heurística” que utiliza conhecimentos específicos do problema a fim de satisfazê-las, porém para que uma heurística seja admissível, é necessário que ela não superestime o custo real, ou seja, o erro deverá ser para baixo, por exemplo, na figura 11, o melhor caminho entre os dois pontos é uma linha reta (custo estimado), porém como existe um obstáculo que impede uma rota em linha reta, o algoritmo é obrigando a buscar novos caminhos, aumentando o custo estimado.

3.5 Fórmula de Haversine

Muito popular e frequentemente usada para desenvolver aplicações GIS(Sistema de informações Geográficas), é uma fórmula que liga dois pontos geográficos a partir de suas coordenadas (latitude e longitude), onde seus valores são dados em radianos, representada na seguinte figura 13, onde Φ é a latitude e λ a longitude dos pontos.

Figura 13. Fórmula de Haversine



Fonte: Daniel Ballinger e Alex Tennant, 2015

Em sua fórmula, é possível obter d utilizando a aplicação da Haversine inversa, função arco seno e arco tangente, substituindo a função haversine (d/r) por h, conforme a figura 14. Caso h aproxima-se de 1, considera-se que os pontos são antipodais, ou seja, lados opostos da esfera.

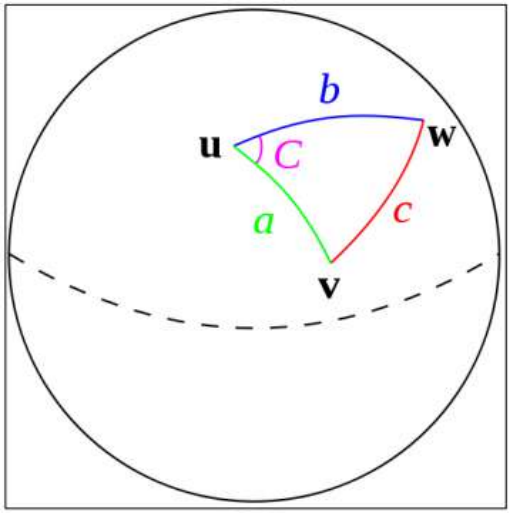
Figura 14. Fórmula de Haversine

$$d = R\ \text{haversin}^{-1}(h) = 2R \arcsin\left(\sqrt{h}\right)$$

Fonte: Daniel Ballinger e Alex Tennant, 2015

A fórmula de Haversine deriva da lei dos Haversines, onde é criado um “triângo” na superfície de uma esfera representada pelos pontos u, v e w, conforme a figura 15, onde o comprimento entre u até v seja a, u até w seja b e w até v seja c, e que o ângulo oposto a c seja C, então a lei estabelece haversin(c) = haversin(a-b) + seno(a) * seno(b) * haversin(C). Seja u o polo norte, enquanto v e w as coordenadas que desejamos saber a distância, a e b valem π/2 – Φ, C a diferença entre as λ (Δλ) e c é d/r, como seno(π/2 – Φ) = cos(Φ), então a fórmula de Haversine é determinada, normalmente o valor do r é 6371 quilômetros (corresponde a media do raio da Terra).

Figura 15. Fórmula de Haversine



Fonte: https://en.wikipedia.org/wiki/Talk%3AHaversine_formula/Alt, 2017

4 ESTUDO DE CASO

O algoritmo de rotas passará por três etapas, o cadastro dos pontos de coleta, o cadastro dos polígonos e a seleção de pontos de partida, para melhor entendimento será apresentado diagramas e explicações de cada etapa, em todas etapas foi utilizado a API do Google Maps, logo foi necessário criar uma chave da API para a utilização de suas ferramentas.

4.1 Arquitetura do Sistema

Neste capítulo ilustraremos o funcionamento do Sistema e em conjunto a biblioteca desenvolvida, para isso será explicado algumas funcionalidades que serão representados com o auxílio de diagramas e imagens.

4.1.1 Caso de Uso

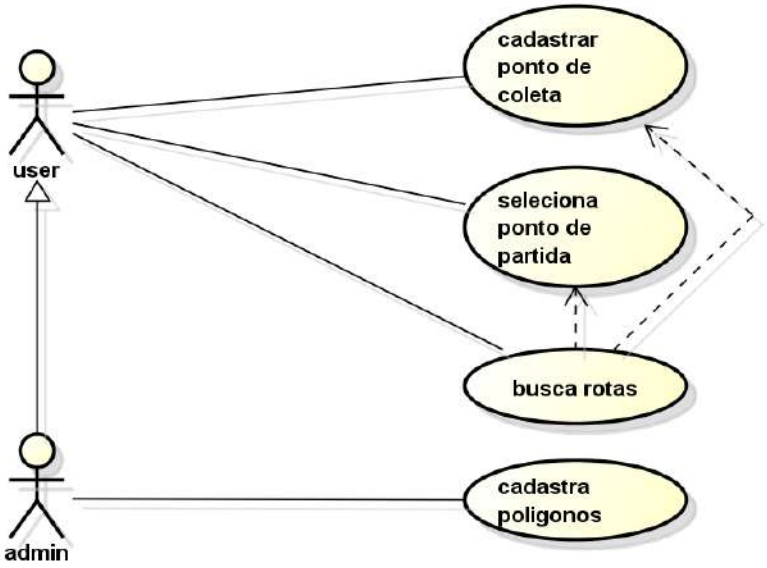
Para a ferramenta de rotas seja executada, será necessário a ação do usuário e administradores do sistema:

1 – Usuário, será capaz de cadastrar os pontos de coleta, pontos de partida (onde partirão para ir aos pontos de coleta) e gerar rotas entre esses pontos;

2 – Os administradores deverão criar os polígonos, que representarão as áreas de estudo e áreas de colisão que impedir[a que as rotas passem por elas e impossibilitará o cadastro de pontos de coleta na área;

As funções de cada usuário está representada na Figura 16.

Figura 16. Diagrama de Uso feito no Astah

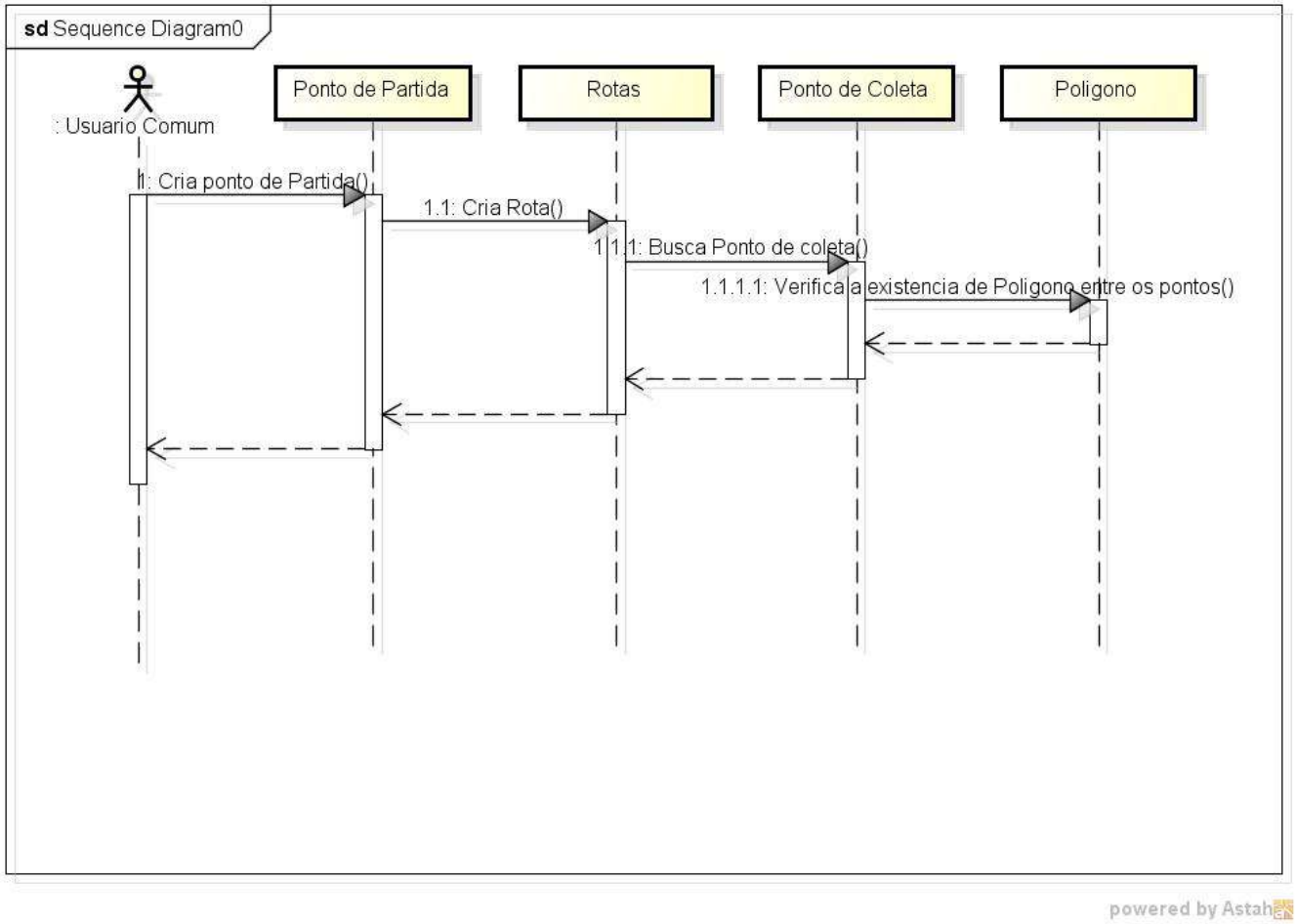


Fonte: Campos, 2016

4.1.2 Diagrama de Sequência

Para o melhor compreendimento da funcionalidade do algoritmo, foi desenvolvido o diagrama de sequência representado pela figura 17. Primeiramente o usuário deverá selecionar os pontos de partidas na API do Google Maps, esses pontos representarão as localidades onde o algoritmo partirá em busca dos pontos de coleta que fazem parte da mesma área de pesquisa que o ponto de partida foi inserido, uma vez encontrado, ele verificará a existência de algum polígono de colisão entre os pontos, caso exista a rota deverá ser desviada, contornando o perímetro do polígono e caso ele consiga chegar ao destino, a rota é validada, se não houver nem um polígono, a rota é validada imediatamente. Para cada ponto de partida e coleta, o algoritmo realiza essa verificação, criando um loop na ação “1.1.1 Busca Ponto de Coleta()”, até que todos os pontos de coleta e de partida sejam visitados pelo menos uma única vez, uma vez visitados as rotas estarão prontas e serão enviadas para o usuário.

Figura 17. Diagrama de Sequência, coleta de dados feito no Astah



Fonte: Campos, 2016

4.1.2 Diagrama de Classe

Para o cenário onde será utilizado o algoritmo de busca, é necessário utilizar as seguintes classes encontradas na API:

1 – google.maps.Map, essa classe chamará a API do google maps, como atributos foi colocado o zoom (distância focal), center (coordenada que o zoom forcará), mapTypeId (modelo do map), a partir do objeto criado nessa classe, ele será setado nas demais classes;

2 – google.maps.Marker, utilizada para criar os marcadores que representarão os pontos de partida e o conjunto de coordenadas que formam os polígonos, como atributos, a classe possui position (objeto da classe google.maps.LatLng, representando posição que o objeto terá), map (objeto criado pela classe google.maps.Map), id (sua identificação) e icon (imagem que representará o objeto Marker dentro da API), como operação, possui a função

removeMarker (apenas para o ponto de partida, essa operação excluirá o ponto de partida na criação da rota);

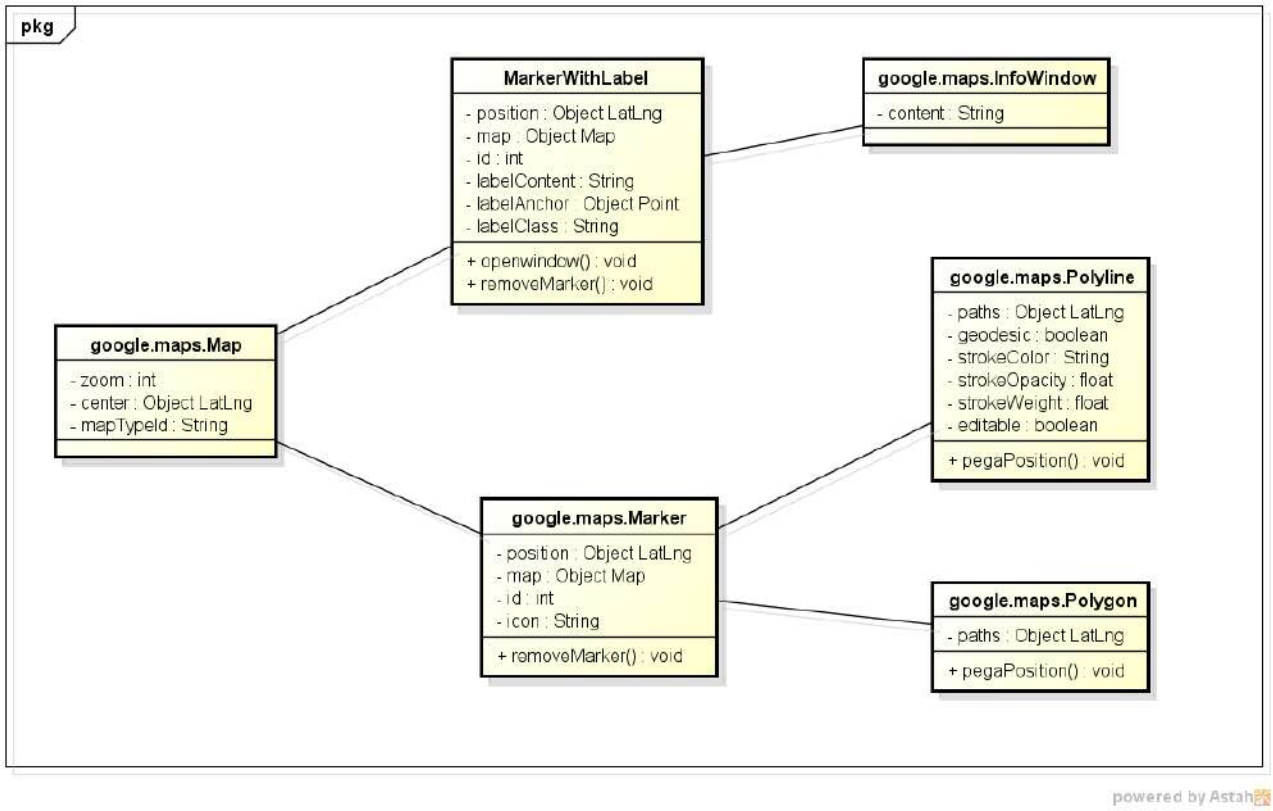
3 – MarkerWithLabel, assemelha-se a classe google.maps.Marker, todavia é utilizado para representar os pontos de coleta, possui mais atributos como labelContent (identificação do ponto de coleta que foi criado no cadastro dos pontos de coleta), labelAnchor (por usar a classe google.maps.Map como base, esse atributo receberá o objeto Point da API que é a imagem padrão dos marcadores utilizada), labelClass (css com as características do painel que mostrará o atributo labelContent), como operação, a classe possui o openwindow (abrirá uma janela com os dados do marcador);

4 – google.maps.InfoWindow, utilizado na operação da classe MarkerWitchLabel, possui apenas um atributo do tipo string com os dados do marcador;

5 – google.maps.Polyline, classe responsável em criar a visualização das rotas, possui os atributos como path (receberá um conjunto de objetos do tipo LatLng), geodesic (conjunto de funções capaz de calcular a distância entre os valores encontrados no path), strokeColor (cor que as linhas terão), strokeOpacity (nível de transparência das linhas representando as rotas), strokeWeight (largura das linhas), editable (função que possibilita a alteração das rotas após serem setados no objeto Map).

6 – google.maps.Polygon, classe responsável em criar os polígonos, possui apenas um atributo paths que receberá um conjunto de objetos do tipo LatLng vindo do objeto Maker.

Figura 18. Diagrama de Classe feito no Astah



Fonte: Campos, 2016

4.2 Cadastro dos Pontos de Coleta

O cadastro dos pontos de coleta será representado por objetos da classe MarkerWithLabel, pois esta classe, é possível adicionar uma label totalmente editável utilizando CSS(Cascading Style Sheets), onde será colocado a identificação do marcador. Em cada marcador será adicionar 2 funcionalidades, uma é realizada utilizando um objeto da classe google.maps.InfoWindow que será setado no objeto MarkerWithLabel, com isso, ao clicar ou tocar no marcador, abrirá uma janela informando seus dados (latitude, longitude e opção de seleção para começar a coleta de dados), já a segunda funcionalidade foi colocado no botão direito do mouse, possibilitando a exclusão do marcador, os objetos MarkerWithLabel também portarão de um id que servirá como identificação de sua localização dentro de uma lista que ele será inserido com os demais pontos de coletas cadastrados, esse id visa especificamente no momento de excluir o ponto, onde não será necessário percorrer a lista, pois já será enviado sua localização.

Figura 19. Cadastro dos Pontos de Coleta

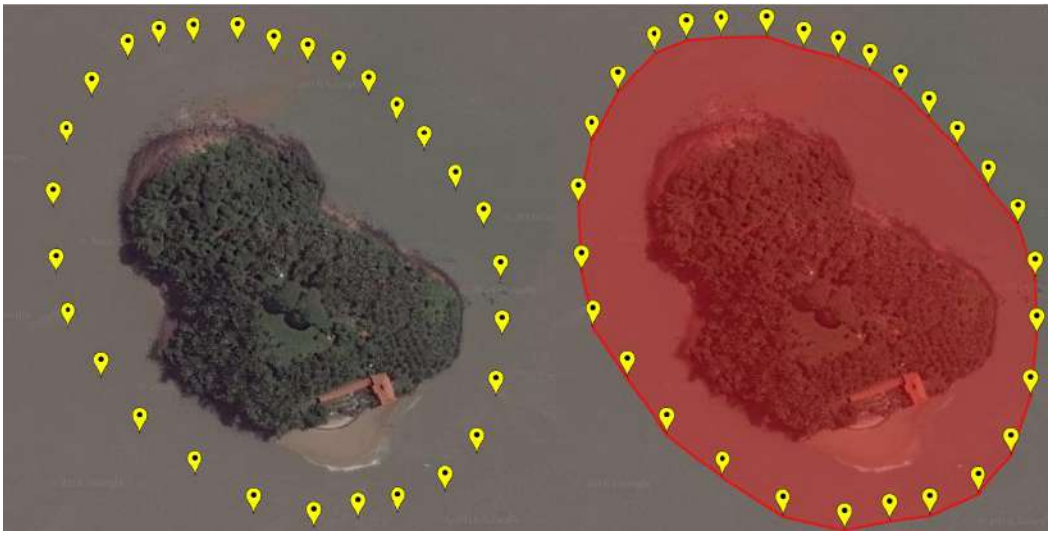
Fonte: Campos, 2017

4.3 Cadastro dos Polígonos

O cadastro dos Polígonos é realizada na área de administradores do Sistema Bio Data, os polígonos são cadastrados em dois momentos distintos e com diferentes funcionalidade para a biblioteca desenvolvida.

Foram utilizados as classes google.maps.Marker e google.maps.Polygon. O polígono é formado por um conjunto de coordenadas que limita a área que o polígono terá na API do Google Maps, em cada coordenada será representado por um o objeto Marker (eles formaram o formato do polígono) no qual deverão ser adicionados em sequência, ou seja, não será possível adicionar um objeto Marker no meio de 2 outros objetos Marker, pois todos os objetos serão adicionados em uma fila que será setado ao objeto Polygon por ordem de cadastro (padrão utilizado pela API), entretanto caso seja adicionado entre 2 Marker, causará uma deformação no polígono. A Figura 20 representa a criação do polígono.

Figura 20. Cadastro de Polígonos

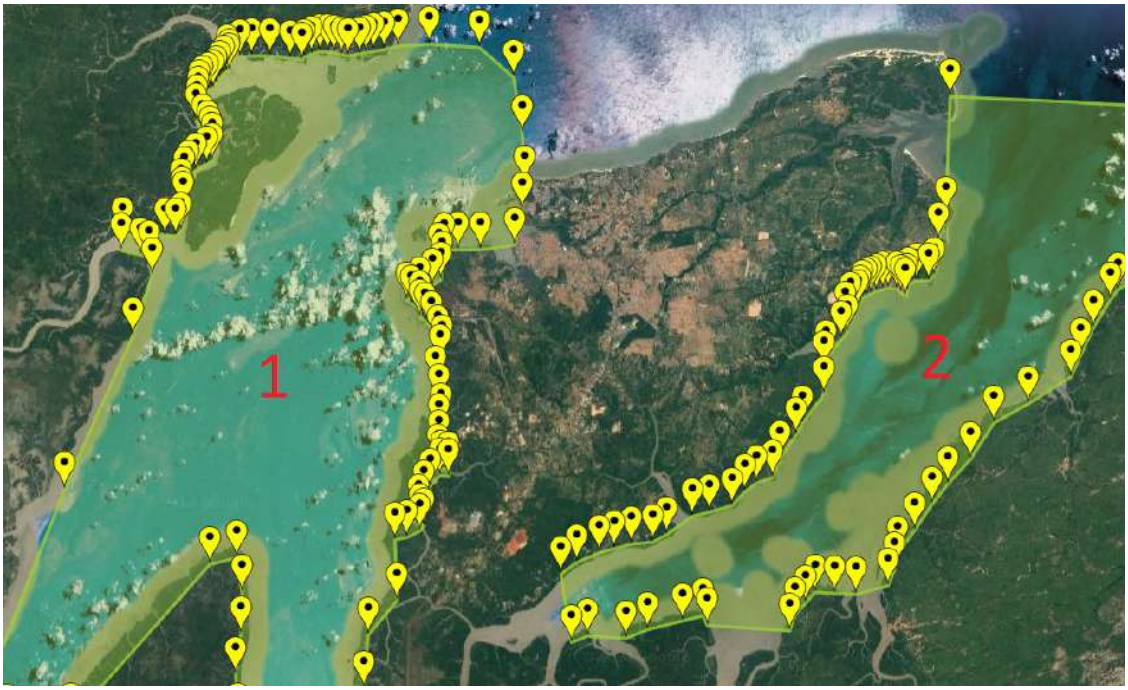


Fonte: Campos, 2017

Devido ao uso do Sistema Bio Data como caso de uso da biblioteca desenvolvida, os polígonos não serão pré definidos, os administradores poderão cadastrar e expandir as áreas conforme o avanço de suas pesquisas, cada polígono terá seus objetos Marker e sua identificação gravada em um banco de dados.

O primeiro momento do cadastro dos polígonos será realizado na zona de pesquisa, que serão áreas que receberão as marcações dos pontos onde será feito a coleta de matérias, cada ponto receberá a identificação da zona que foi inserida, o cadastro da zona visa limitar a visão dos pontos de partidas para apenas a zona que ele está inserido, ou seja, um ponto de partida da zona 1 não poderá chegar nos pontos de coleta da zona 2, organizando a visualização das rotas e evitando dois problemas, um é o processamento excessivo, pois a biblioteca desenvolvida busca uma rota que passe por todos os pontos de coleta, e o outro é o excesso de chamadas das bibliotecas contidas na API, pois a chave da versão free (como usado neste trabalho) possui uma limitação de acessos e visualizações, que por algumas vezes retornaram erro ao tentar percorrer toda a ilha do Upaon-açu quando a biblioteca tentou passar da Baia do Arraial para a Baia de São Marcos ou vice-versa. Para este trabalho foi criado duas zonas de pesquisa, zona 1 representando a Baia de São Marcos e zona 2 representando a Baia do Arraial conforme a figura 21.

Figura 21. Zonas



Fonte: Campos, 2017

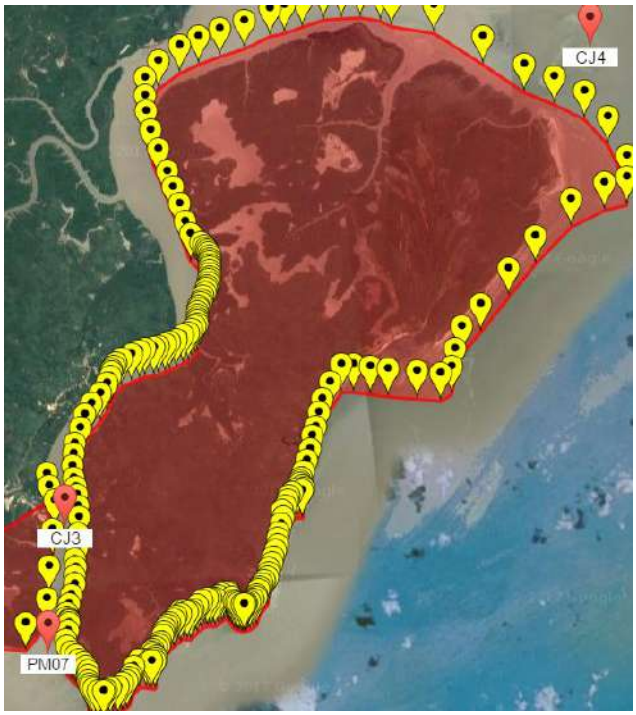
O segundo momento está na identificação de obstáculos (ilhas, dumas de areia, entre outros) dentro das zonas de pesquisas, que terão como função de impedir que a rota passe em determinadas áreas na zona, por exemplo, dentro da zona 1, quando tentamos ir do ponto CJ4 para CJ3 teremos a seguinte rota representada na figura 22.

Figura 22. CJ4 para CJ3

Fonte: Campos, 2017

Visualmente o polígono terá o mesmo formato do obstáculo que desejamos contornar, conforme a figura 23.

Figura 23. Polígono



Fonte: Campos, 2017

4.4 Rotas

Para o Algoritmo de rotas desenvolvido neste trabalho, foi utilizado as

seguintes classes da API, google.maps.Marker, google.maps.InfoWindow, google.maps.LatLng, google.maps.Polyline e google.maps.Polygon. Inicialmente foram criados cinco vetores, markers(armazenará os objetos referentes aos pontos de partida), markers_banco(armazenará os objetos referentes aos pontos de coletas cadastrados anteriormente), coordenadas_aux(armazenará temporariamente os objetos referentes as coordenadas de cada do ponto do polígono cadastrado anteriormente, ou seja, o path) e poligonos_array(armazenará os objetos Polygon que representarão os obstáculos), e por fim areas_array(armazenará os polígonos das zonas de pesquisa), o objetivo desse algoritmo é passar uma única vez em cada ponto de coleta partindo dos pontos de partida.

Primeiramente é feito o carregamento dos pontos de coleta e dos polígonos (os polígonos não serão setados no objeto Map, apenas terão seus objetos criados), armazenando-os em seus respectivos vetores, em seguida, o algoritmo A* esperará a criação dos pontos de partida (receberão a mesma estrutura dos objetos Marker utilizados para criar os polígonos, com dois diferenciais que são o adicionamento do atributo área, que corresponde ao ID da zona de pesquisa que ele foi inserido e o ícone de apresentação que será verde, conforme a figura 24) por parte do usuário.

Figura 24. Ponto de Coleta e Partida

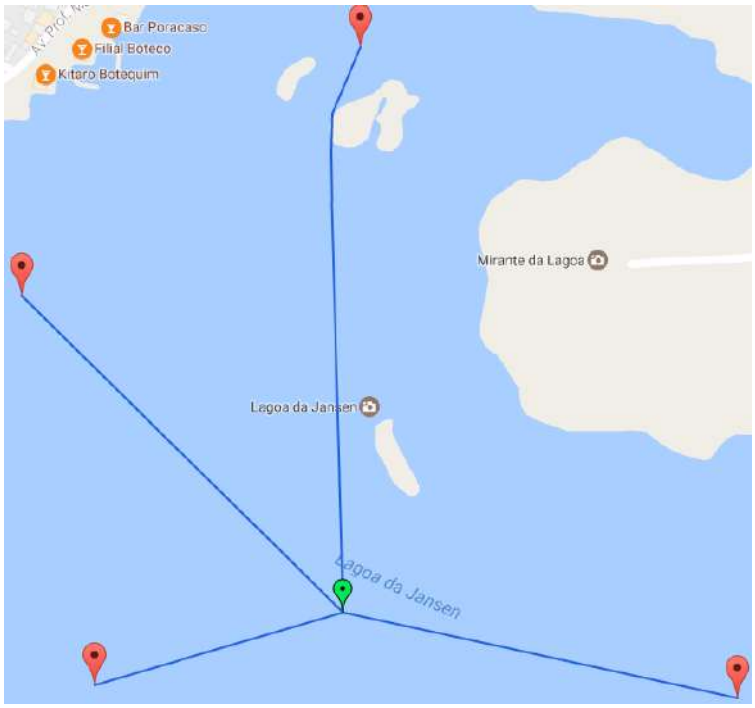
Fonte: Campos, 2017

Uma vez adicionado todos os pontos de partida, o algoritmo A* terá todos os pré requisitos para a criação das rotas. Existem três diferentes tipos de criação das rotas:

1 – Cada ponto de coleta terá uma rota para cada ponto de partida de sua respectiva zona de pesquisa, ou seja, se a zona 1 tiver 2 pontos de partida e 5 pontos de coleta, cada ponto de partida terá 5 rotas distintas, e não existirá rotas que ligue os pontos de coletas, utilizado para medir à distância de cada ponto de

coleta em relação ao ponto de partida.

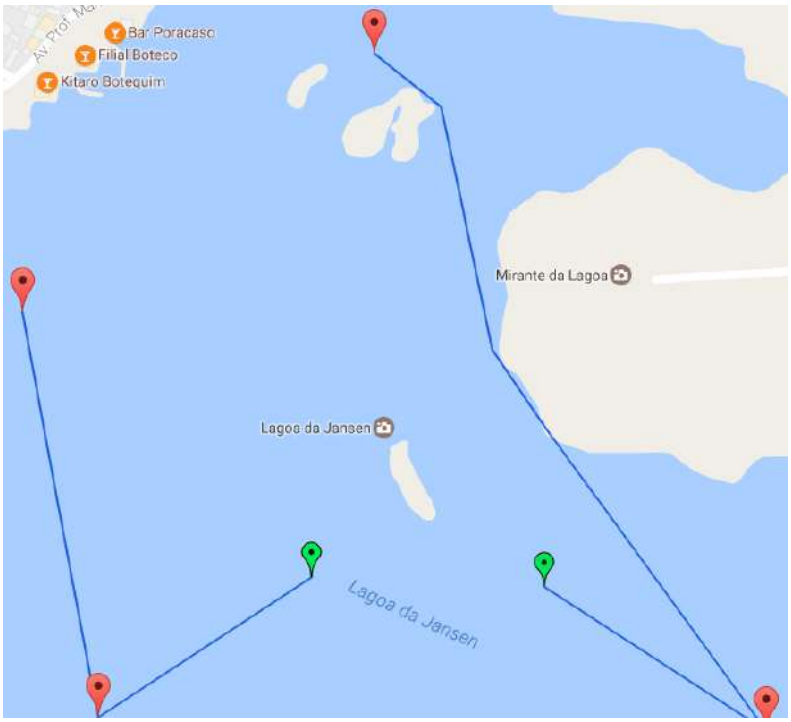
Figura 25. Primeira forma.



Fonte: Campos, 2017

2 – As rotas são feitas tendo com base o custo da rota. Conforme os pontos de coletas forem sendo adicionados na rota de um determinado ponto de partida, o seu custo será incrementado, assim o algoritmo A* selecionará a rota ou rotas (caso estiverem com o mesmo peso) de menor custo para receber o ponto de coleta selecionado por ele (no caso, se existir mais de uma rota, ele levará em consideração o menor peso da rota até o novo ponto), utilizado para criar rotas que tenham pesos semelhantes.

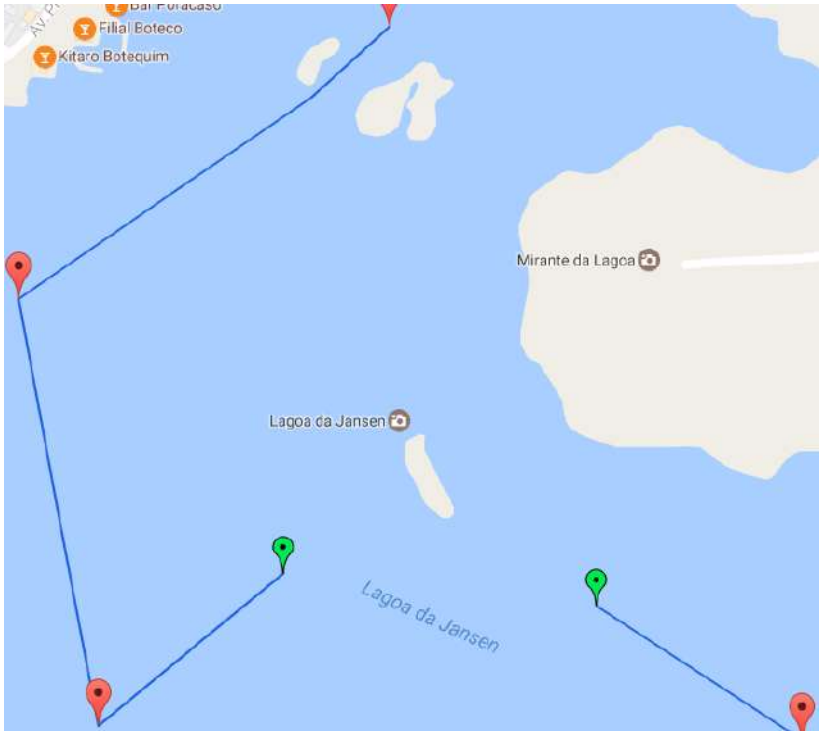
Figura 26. Segunda forma.



Fonte: Campos, 2017

3 – Cria rotas dentro de cada grupo de pontos. Utilizando a primeira forma de criação das rotas, esta forma agrupa os pontos de coleta em pequenos grafos, ou seja, ele trabalhará por grupo e não pelo todo, cada ponto é designado ao grupo do ponto de partida mais próximo de sua localização, não levando em consideração o peso total dos demais grupos, as rotas são criadas em sequência, após o término da rota de um grupo o algoritmo A* partirá para outro grupo, semelha-se a ideia das zonas de pesquisas, um ponto de um grupo não conhece os pontos de outros grupos, liga todos os pontos do grupo criando uma lista, utilizado para criar rotas com os pontos mais próximos do ponto de partida.

Figura 27. Terceira forma.

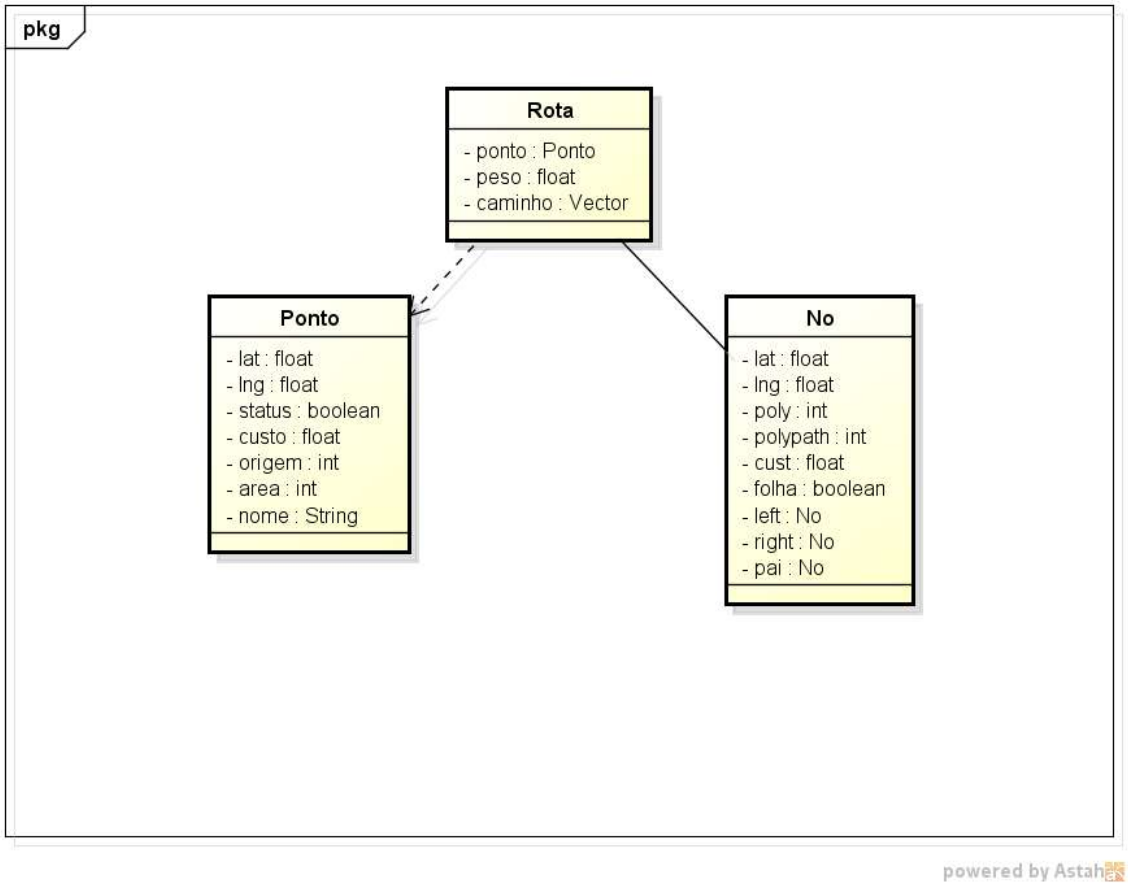


Fonte: Campos, 2017

Apesar de existir três formas de criar as rotas, é utilizado praticamente o mesmo algoritmo, alterando a ordem de chamada das funções ou verificando uma ou outra variável (como exemplo a identificação do grupo), todavia, a base continua a mesma, sendo formado por três classes e um conjunto de descesses funções. O algoritmo A* trabalhará com a classe Ponto que será os pontos de partida e de coleta que serão adicionados na rota, possuindo como atributos, lat representando a latitude, lng a longitude, status para identificar se o ponto já foi visitado, custo que é a distância em metros, origem para identificar o grupo que ele faz parte, área para identificar a zona de pesquisa pertencente e o nome do ponto caso seja um ponto de coleta; classe No, serão as coordenadas auxiliárias, o caminho percorrido entre dois pontos caso exista um obstáculo entre eles, assim como a classe Ponto, terão a latitude e longitude, id do polígono que ele pertence, id do seu endereço no path do polígono, folha para identificar se ele é uma folha que conseguiu chegar ao destino esperado, e os vizinhos left, right e o pai, todos da mesma classe No; e por fim a classe Rota, que representará todas as rotas encontradas, tendo o ponto como o objeto da classe Ponto adicionado na rota, peso que é o total dos custos de cada ponto na rota e o caminho, array de objetos da classe No, para melhor visualização,

a figura 28 mostra a estrutura das classes.

Figura 28. Ponto de Coleta e Partida



Fonte: Campos, 2017

O algoritmo A* passará seis fases até completar a rota, cada fase, possui sua respectiva função que será abordada com o decorrer deste trabalho, a fim de melhor entendimento do funcionamento, é necessário compreender o resumo das seguintes fases:

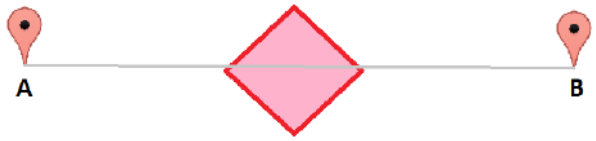
a) Situação: temos o ponto A como origem e o ponto B como destino;

Figura 29. Ponta A e B

Fonte: Campos, 2017

b) Fase 1: o algoritmo A* cria uma pré rota em linha reta entre eles, logo A→B;

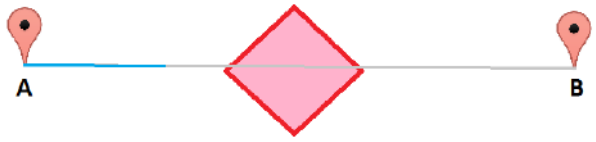
Figura 30. Fase 1



Fonte: Campos, 2017

c) Fase 2: o algoritmo A* divide a pré rota em várias partições que serão adicionados aos poucos;

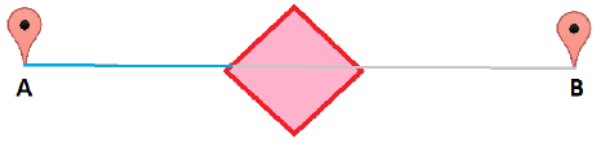
Figura 31. Fase 2



Fonte: Campos, 2017

d) Fase 3: identifica a colisão em um polígono;

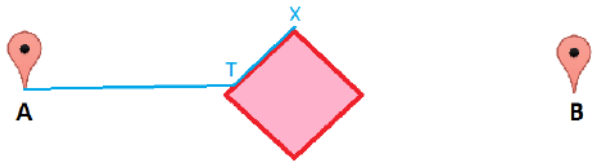
Figura 32. Fase 3



Fonte: Campos, 2017

e) Fase 4: cancela a pré rota criada na Fase 1, busca o ponto mais próximo que forma o perímetro do polígono, adiciona-o a rota, realiza a busca e o incremento na rota enquanto não achar um ponto no perímetro capaz de ir até B, neste exemplo ele achou todos os pontos de T até X (T, U, V, W e X);

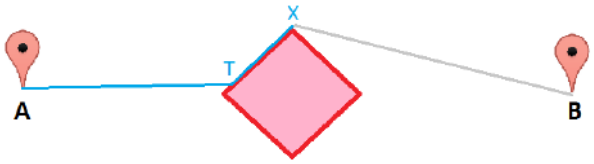
Figura 33. Fase 4



Fonte: Campos, 2017

f) Fase 5: cria outra pré rota entre X e B, em seguida realiza as fases 2, 3, 4;

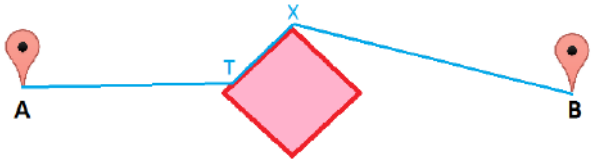
Figura 34. Fase 5



Fonte: Campos, 2017

g) Fase 6: se não existir colisão (Fase 3), a rota de X até B é liberando, com isso nossa rota que antes era A→B, ficou A→T→U→V→W→X→B;

Figura 35. Fase 6



Fonte: Campos, 2017

No fim do algoritmo A* teremos um resultado semelhante a Figura 36.

Figura 36. Resultado

Fonte: Campos, 2017

As rotas são representadas no código por uma matriz, onde o primeiro elemento sempre será o ponto de partida e os demais pontos de coletas adicionados de forma ordenada, ou seja, se for colocado três pontos de partidas, teremos uma matriz com três linhas porém com o número de colunas distintas pois o número de colunas depende do número de pontos de coleta em um determinado grupo e zona

de pesquisa.

Figura 37. Resultado

Partida	Coleta	Coleta	Coleta	Coleta
Partida	Coleta			
Partida	Coleta	Coleta		

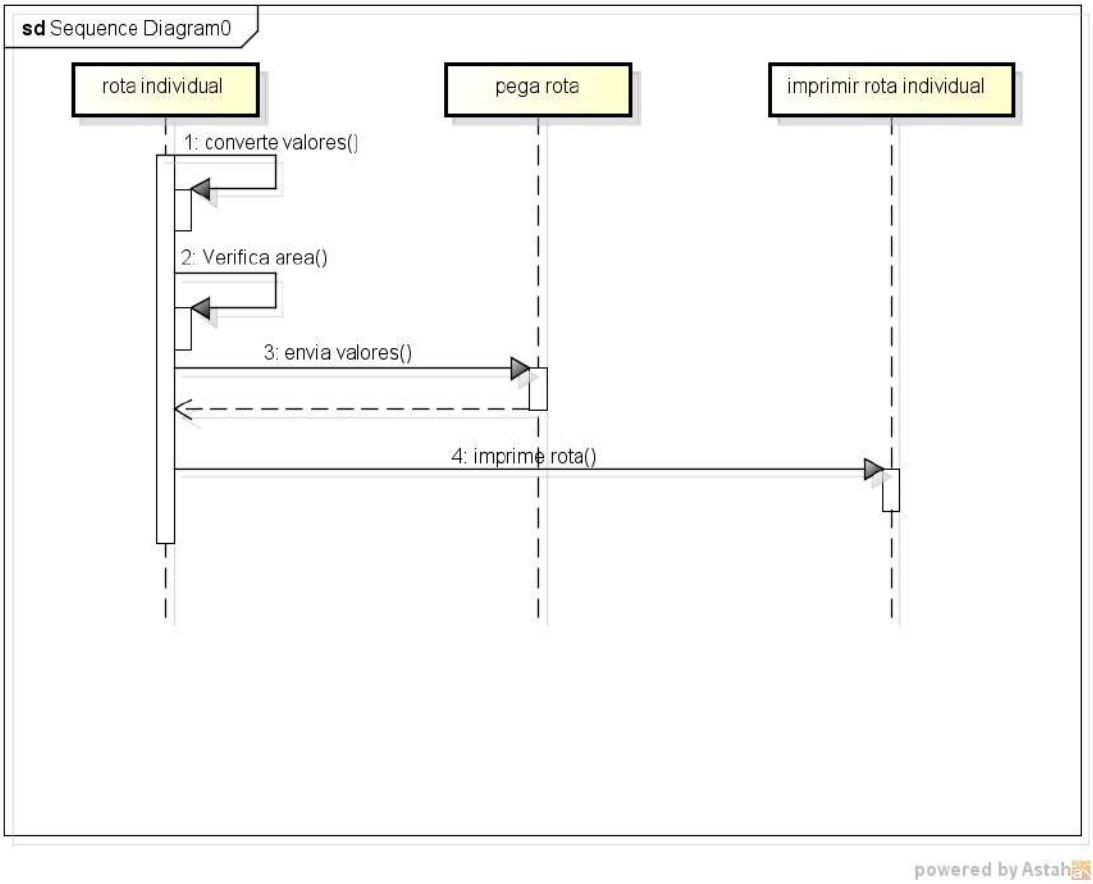
Fonte: Campos, 2017

4.4.1 Função “rota_individual()”

Esta função tem como responsabilidade, desenvolver a primeira forma de criação de rotas citada anteriormente. Primeiramente a função pegará os pontos de partida e coleta, convertendo-os para a classe Ponto que serão colocados conforme a rota for se formando na classe Rota, cada objeto criado nessa classe será armazenado em um vetor de tamanho correspondente a quantidade de pontos de partida, cada parte do vetor representará as rotas desenvolvidas de forma ordenada.

Para o desenvolvimento da rota, é necessário percorrer tanto os pontos de partida como de coleta, logo é necessário criar dois loop, como para essa forma de criação não é necessário realizar comparações entre rotas entre os pontos de coleta, é realizado apenas a chamada da função “pega_rota” (será explicada com mais detalhes no decorrer do trabalho) que retornará o objeto Rota pronto para ser adicionado no vetor de rotas, por fim, ao percorrer todos os pontos, é chamada a função “imprimir_rotas_individual”.

Figura 38. Diagrama rota_individual



Fonte: Campos, 2017

4.4.2 Função “imprimir_rotas_individual()”

Utilizado para imprimir as rotas criadas pela função “rota_individual”, como o resultado final é uma matriz, é necessário criar dois loop para percorrer os todos dados, entretanto em cada ponto adicionado na rota, deverá ser feito uma verificação no atributo caminho, encontrado no objeto, esse atributo armazena todas as coordenadas que o algoritmo A* precisou utilizar para chegar até o ponto que devem ser adicionadas antes.

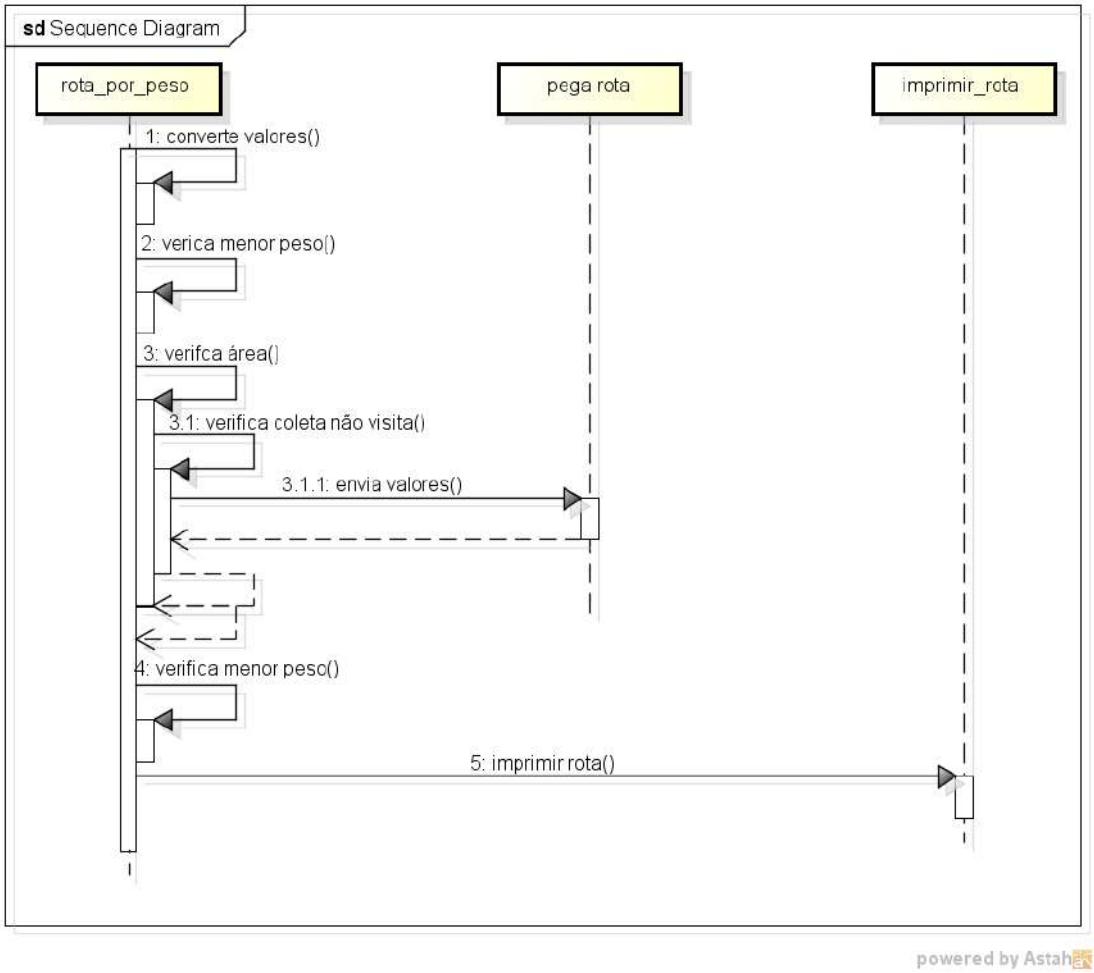
Como as rotas dessa forma são feitas sempre saindo do primeiro elemento da rota (ponto de partida), não será possível criar um vetor para armazenar todos os pontos e enviar para a função que cria as polyline, para isso, foi necessário criar uma função que receba o ponto de partida e o ponto destino (“polyline_individual”), a fim de criar separadamente cada rota.

Todas as funções de impressão das rotas, também mostram os valores fora da API em uma div identificada como resultado, todavia sua estrutura só é compatível com sistemas que utilizam o framework bootstrap.

4.4.3 Função “rota_por_peso()”

Função utilizada para criar a segunda forma de rotas, onde o principal critério é o peso total de cada rota, assim como a “rota_individual”, é necessário converter para as classes utilizadas (Ponto e Rota), nesta forma, é adicionado uma verificação dos pesos antes de adicionar qualquer novo ponto a rota, comparando o atributo peso do objeto Rota localizado no último elemento de cada linha da matriz, o menor valor terá o endereço da rota adicionado em um vetor de permissão, por exemplo, se o menor peso foi encontrado na segunda linha da matriz, logo a rota dois será liberada para receber uma nova coordenada, todavia caso exista pesos iguais, ambas serão liberadas, mas apenas uma receberá uma nova coordenada, a escolha dependerá do menor peso para o novo ponto, diferente da função “rota_individual”, a “rota_por_peso” mede o último elemento de cada linha da matriz com os pontos de coleta, o menor peso retornado pela função “pega_rota” terá o objeto salvo em uma variável temporária, junto com a localização do ponto de coleta no vetor e o endereço linha da matriz que foi feito a medição. Após percorrer todos pontos de coletas não visitados, o objeto salvo anteriormente é adicionado na linha da matriz que teve seu endereço salvo, em seguida é alterado o atributo status do ponto de coleta selecionado para 1, ou seja, o ponto de coleta é marcado como visitado, quando não existir mais pontos de coletas não visitados, é chamada a função “imprimir_rotas”.

Figura 39. Diagrama rotas por_peso



Fonte: Campos, 2017

4.4.4 Função “imprimir_rotas()”

Responsável em deixar visíveis as rotas criadas pelas funções “rota_por_peso” e “rota_por_grupo”, sua estrutura assemelha-se a função “imprimir_rotas_individual”, todavia em vez de criar cada rota separadamente, essa função pegará toda uma linha da matriz e enviará para a função “polyline”.

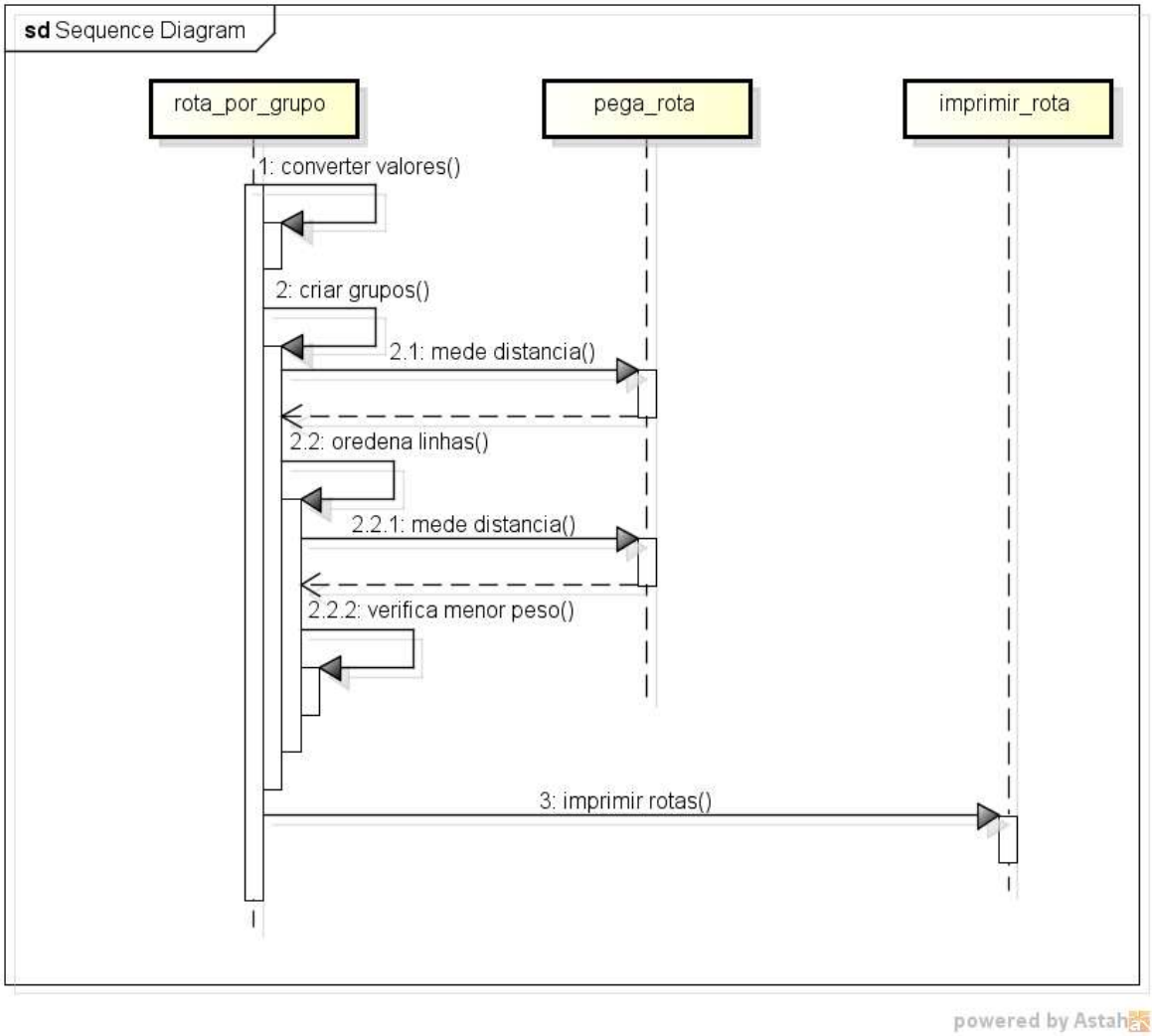
4.4.5 Função “rota_por_grupo()”

Responsável em criar a terceira forma de rotas, tem como critério a criação de grupos, esta função mistura um pouco das outras formas, como exemplo a necessidade de converter os valores, utilização do algoritmo que mede a rota individualmente para separar os pontos de coleta conforme a medição sendo devolvida pela função “pega_rota”, cada ponto de coleta terá o atributo origem recebendo o endereço do ponto de partida de menor peso até ele, isso identificará

de qual grupo cada ponto de coleta pertence.

Uma vez que foi atribuído o grupo de cada ponto de coleta, é realizado o desenvolvimento das rotas semelhantes à função “rota_por_peso”, todavia não existirá a comparação do peso entre linhas diferentes, uma vez que já foi determinado para qual linha irá o ponto de coleta na criação dos grupos, a comparação será feita apenas nos pontos que pertencem naquele determinado grupo, ordenando-os na sequência da rota, toda vez que um ponto é colocado na ordem certa, é marcado como visitado. Após a ordenação de cada linha da matriz, elas serão enviadas para a função “imprimir_rotas”.

Figura 40. Diagrama rotas por_peso



Fonte: Campos, 2017

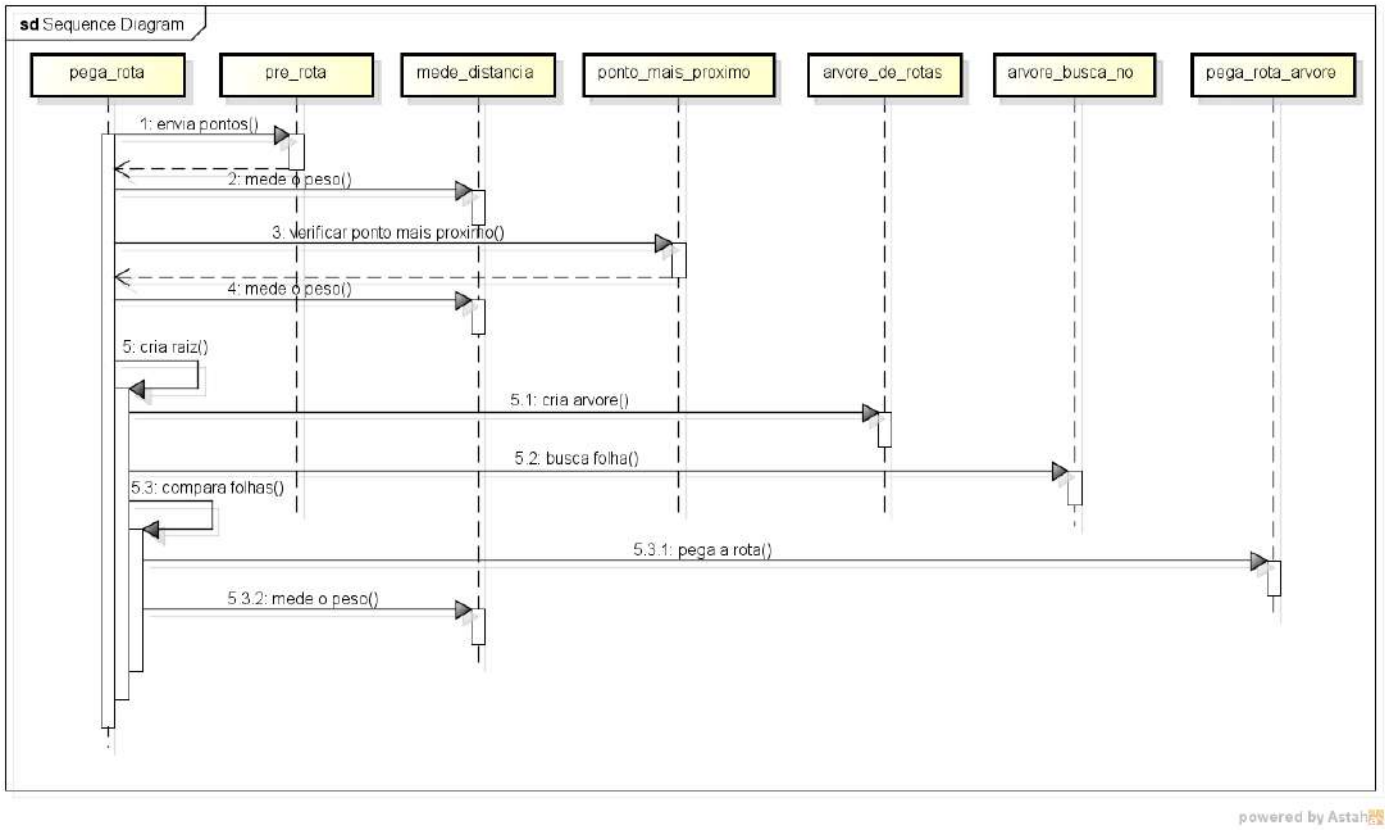
4.4.5 Função “pega_rota()”

Função que organiza as necessidades para a criação da rota entre dois objetos Ponto que são passados no parâmetro da função, realizando as chamadas

das funções necessárias para que o algoritmo A* passe por todas as fases.

A fase 1 é iniciada na chamada da função “pre_rota” que retornará dois valores que dependem da fase 2 que ocorre nela, caso a fase dois seja verdadeiro, entra-se direto na fase 6, onde um objeto da classe Rota é retornando com os seguintes dados: atributo ponto com o destino; o peso com a medida da rota vindo da função “mede_distancia”; e o caminho com valor nulo, caso falso, o algoritmo entrará na fase 3, recebendo da função “pre_rota” a identificação do polígono que ocorreu a colisão e qual a coordenada que isso aconteceu, que serão passados no parâmetro da chamada da função “ponto_mais_proximo”, entrando na fase 4, após estabelecer o ponto mais próximo, é realizado a chamada da função “mede_distancia” para pegar o peso da rota até o ponto encontrado, em seguida cria-se o um objeto da classe No, recebendo a latitude, longitude, id do polígono, endereço do path do ponto no polígono, vindo da função “ponto_mais_proximo”, o peso gerado anteriormente e o valor 0 ao atributo folha, com isso, é iniciado a fase 5 chamando da função “arvore_de_rotas” passando como parâmetro o objeto No, o ponto destino, o valor 1 e o valor nulo (está função será explicada ao decorrer do trabalho), em seguida é chamada a função “arvore_busca_no”, para encontrar os nos que conseguiram acesso ao destino, comparando-os e selecionando a de menor peso, esse nó será passado na função “pega_rota_arvore” que pegará todos os nós pertencentes a rota, armazenando-os no atributo caminho do objeto da classe Rota que será criada.

Figura 41. Diagrama rotas pega_rota



Fonte: Campos, 2017

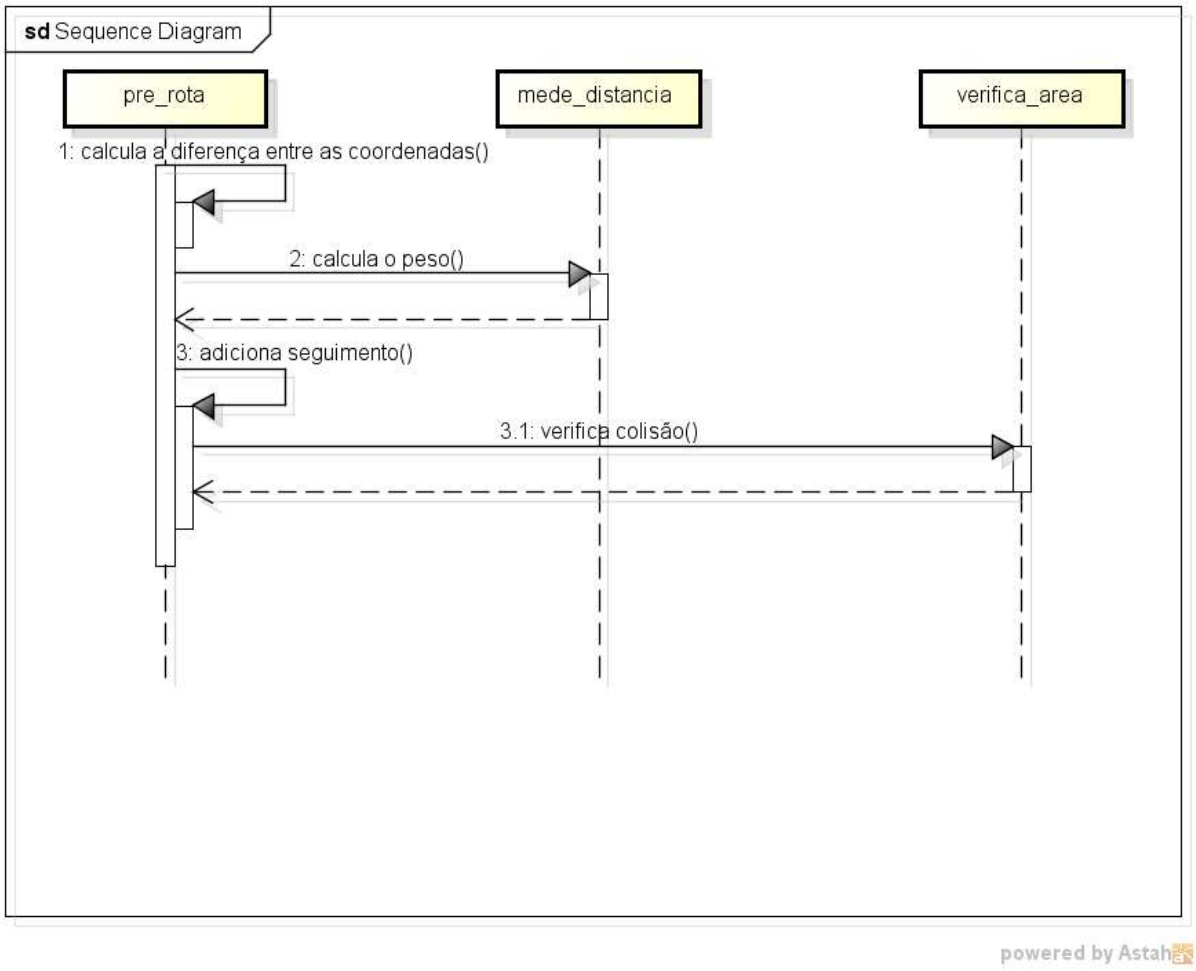
4.4.5 Função “pre_rota()”

Utilizada para realizar a segunda fase do algoritmo, esta função receberá dois objetos da classe Ponto (um sendo a partida e o outro o destino) e tentará criar a melhor rota possível (uma linha reta) entre eles, todavia a rota será particionada em seguimentos de tamanhos equivalentes, que serão adicionados um de cada vez, isso ocorre visando o tratamento de colisões com os polígonos, pois caso um segmento colida, teremos a posição mais precisa, ao contrário da ferramenta encontrada na API que retorna apenas se um polyline pertence ou não em um polygon, mas não mostra onde.

Para a criação dos seguimentos, a função extrairá a latitude e longitude de ambos objetos, e verificará a diferença entre eles, em seguida é calculado o peso entre os objetos chamando a função “mede_distancia”, com isso será usado a seguinte fórmula $x = x + (pdx / n)$ onde “x” é a latitude atual, “pdx” é a diferença entre a latitude do destino com a latitude da partida e “n” o peso entre os dois objetos e maior que 10, o mesmo raciocínio é utilizado para a longitude que é representado

por “y” ($y = y + (pdy / n)$), a cada novo seguimento adicionado na rota, é chamada a função “verificar_area” que realiza a terceira fase verificando se ocorreu colisão, se não ocorreu, o seguimento é validado e a função “pre_rota” continuará adicionando, caso ocorra, a rota é cancelada e a função retorna o seguimento e a identificação do polígono colidido.

Figura 42. Diagrama rotas pre_rota



Fonte: Campos, 2017

4.4.6 Função “mede_distancia()”

Utilizada para medir a distância em quilômetros entre duas coordenadas, utilizando a fórmula de haversine, recebe dois objetos da classe Ponto, em seguida extrai a latitude e longitude de ambos para realizar o calculo, retornando o valor.

4.4.7 Função “verificar_area()” e “verificar_area_ponto()”

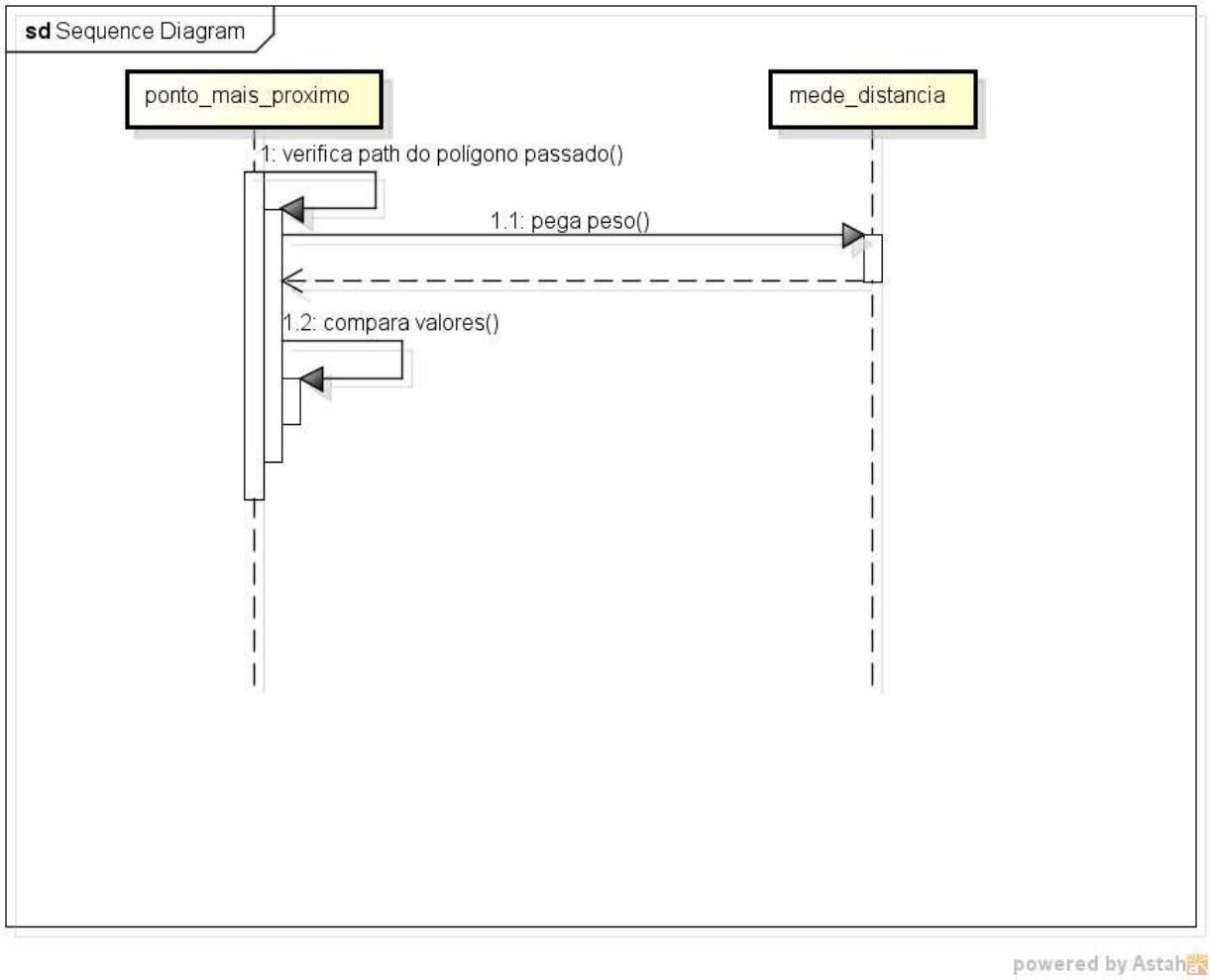
Função responsável em verificar se determinada coordenada faz parte do conjunto de coordenadas de determinado polígono, para isso ela recebe a latitude e

longitude em seguida, cria um objeto LatLng para ser reconhecido pela função “google.maps.geometry.poly.containsLocation” da API que retornará um valor booleano.

4.4.8 Função “ponto_mais_proximo()”

Toda vez que a função “pre-rota” retornar uma colisão com um novo polígono, será necessário realizar a chamada da função “ponto_mais_proximo” passando os dados da colisão, esta função tem como responsabilidade, identificar qual o ponto que forma o polígono mais próximo da colisão, liberando-o para ser visto pelo algoritmo A*, retornando os seus dados.

Figura 43. Diagrama rotas ponto_mais_proximo



Fonte: Campos, 2017

4.4.9 Função “arvore_de_rotas()”

Função responsável em desenvolver as rotas, criando os caminhos utilizando objetos da classe Nó, ou seja, ela é a fase 4 do algoritmo, é chamada toda

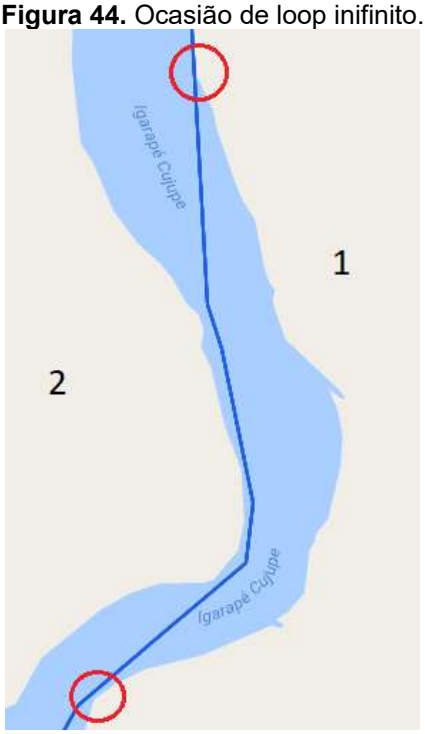
vez que ocorre uma colisão, tem como parâmetros um objeto Nó(nó atual), o destino (objeto Ponto que queremos chegar), tipo (tipo de colisão), nó pai (objeto Nó anterior) e uma lista tabu (uma lista que verifica a quantidade de vezes que uma rota foi utilizada), com isso é formulado uma árvore que representará todas as rotas possíveis, utilizando os seguintes passos.

1 – Cria-se o nó raiz na “função pega_rota” a partir do ponto liberado na primeira colisão, em seguida o nó é passado junto com o destino, tipo de colisão igual a 1, porém os demais valores nulos.

2 – Verifica se o valor do nó pai é diferente de NULL, se for o nó é adicionado como pai do nó atual.

3 – Verifica a lista tabu, cada rota tem cinco tentativas de serem completadas, com isso é possível evitar loops infinitos que podem ocorrer caso a rota passe por um polígono em seguida retorna a ele, conforme a Figura 44 em nos testes, o loop ocorre devido ao tipo 1 de colisões. A lista tabu será sempre alimentada toda vez que um objeto nó for criado.

Figura 44. Ocasião de loop inifinito.



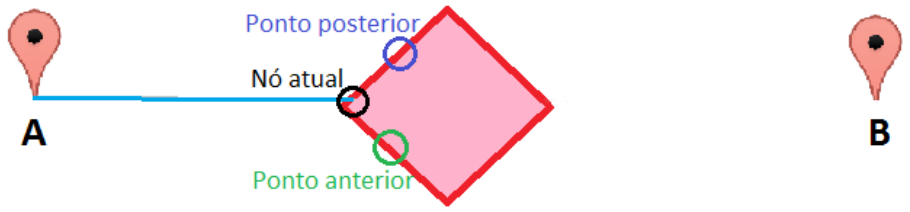
Fonte: Campos, 2017

4 – Após a liberação do objeto no pela lista tabu, o algoritmo A* cria uma pré rota ao destino, caso consiga, a pré rota é liberada e a rota é finalizada, caso não, é verificado em que tipo a colisão entrará.

5 – Verificação do tipo da colisão, existem seis diferentes situações para o

tratamento das colisões, tendo como base a comparação do polígono do nó atual com o polígono retornou da função “pre-rota”, caso mesmo polígono, se na chamada da função “arvore_de_rotas” foi passado o tipo igual 1, significa que o nó atual é a raiz da árvore, logo são liberados os pontos anterior e posterior do endereço do path encontrado no nó atual para serem encontrados pelo algoritmo A*, com isso é criado dois novos objetos nó referente a cada um, com seus respectivos valores de peso, latitude, longitude, endereço do path e identificação da folha igual a 0, ambos terão como pai o nó atual, em seguida serão passados no parâmetro para a função “arvore_de_rotas”, todavia, o nó anterior terá o tipo igual a 2 e o posterior igual a 3.

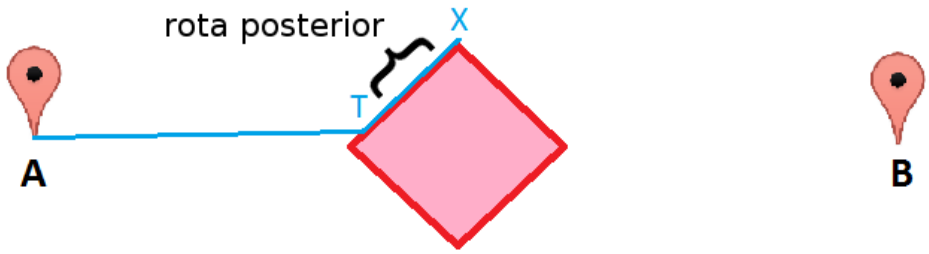
Figura 45. Tipo 1 com raiz



Fonte: Campos, 2017

Os tipos 2 e 3 quando o polígono continua o mesmo, são utilizados para contornar o polígono até um determinado ponto que a rota conseguia sair dele, o tipo 2 contornará os pontos anteriores e o tipo 3 os posteriores de forma recursiva.

Figura 46. Rota do Tipo 3



Fonte: Campos, 2017

Caso o ocorra uma colisão com um polígono diferente, é verificado qual o último tipo passado como parâmetro, caso seja tipo 1, não ocorre mudança nas operações citadas anteriormente, porém no tipo 2 e 3 ao chamar a função “arvore_de_rotas”, será colocado o tipo 1 para liberá os dois pontos.

6 – Verifica se a pre rota conseguiu chegar ao destino, toda vez que uma rota feita pelo passo 5 alcançou o destino, o nó atual é marcado como solução. Apos os passos terminados, a árvore é finalizada com todas as rotas possíveis para chegar ao destino.

4.4.10 Função “arvore_busca_no ()”

Função responsável em agrupar todos os nós que solucionaram o problema, em seguida realiza a comparação entre eles verificando a de menor peso.

4.4.11 Função “pega_rota_arvore ()”

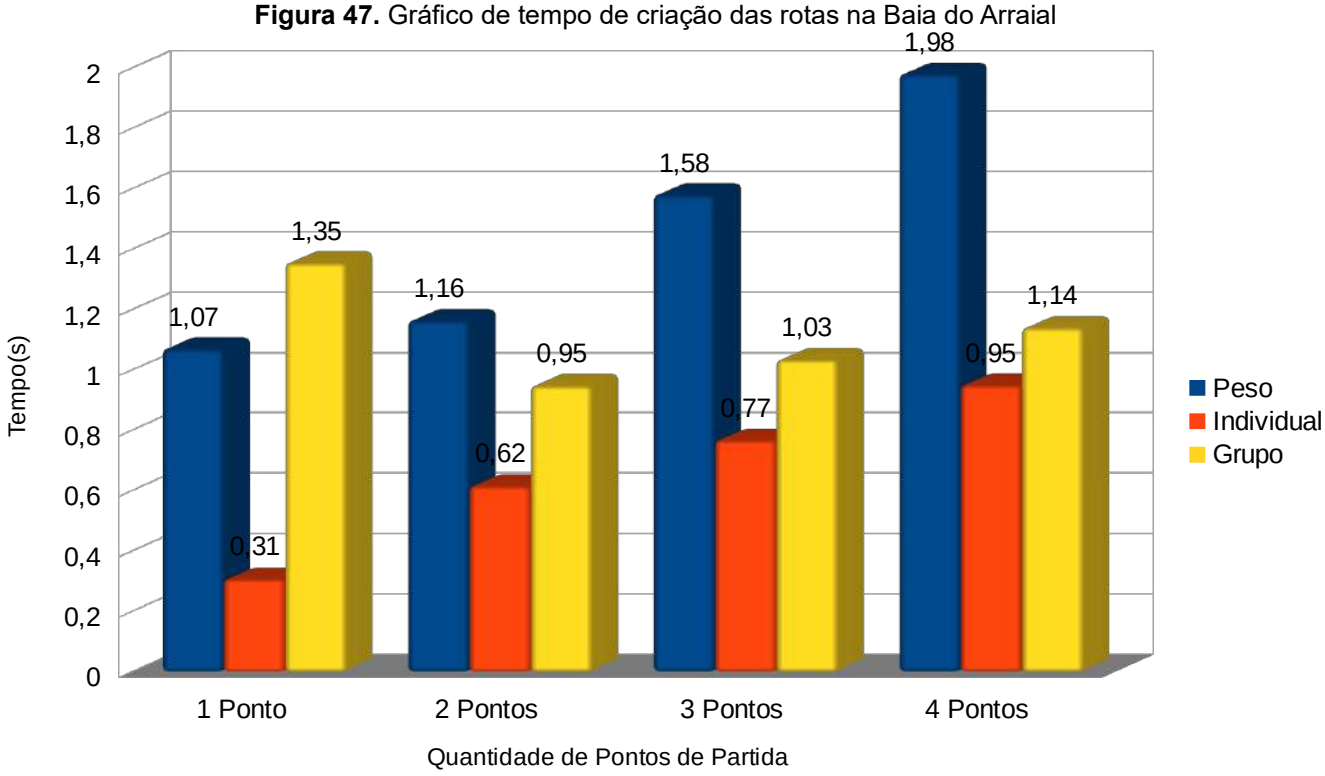
Após a seleção do nó na função “arvore_busca_no”, é realizado o armazenamento dos ancestrais do nó, passando pelos nós pai até chegar na raiz da árvore, esse caminho corresponde a rota de menor peso, em seguida é retornado a função “pega_rota”.

5 RESULTADOS

Neste capítulo será apresentado os resultados obtidos em testes de performance adicionando pontos de partida em uma única área de pesquisa, envolvendo de 6 até 27 pontos, realizando cem testes em cada quantidade de ponto de partida em cada Baia, a configuração utilizada para testes, foi um computador com i7 6700, 16 gb de ram.

Os primeiros resultados de performance referente à Baia do Arraial com um, dois e três pontos de partidas utilizando os três modos de criação de rotas desenvolvidas na biblioteca.

Figura 47. Gráfico de tempo de criação das rotas na Baia do Arraial



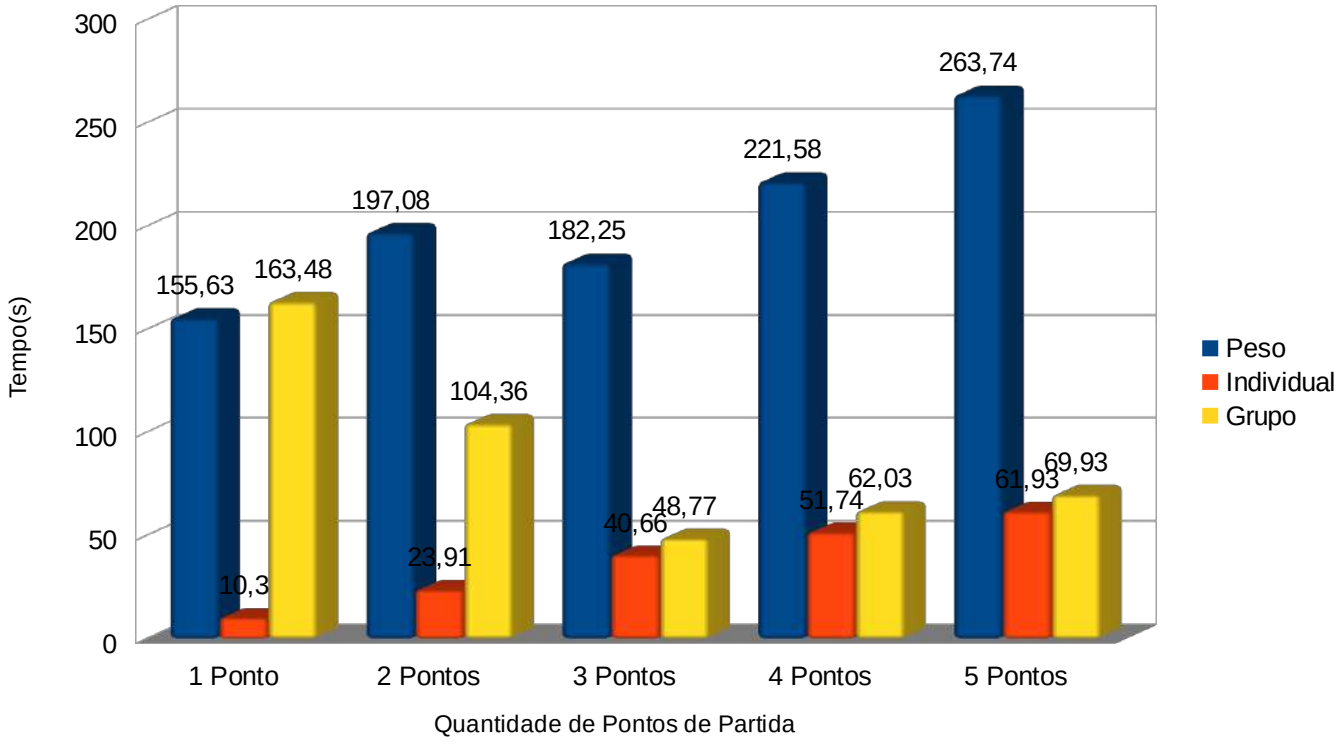
Fonte: Campos, 2017

Ao verificar os resultados encontrados, percebemos uma semelhança no tempo de execução, claramente o tempo da rota individual será mais rápido, pois ela não cria uma rota ligando todos os pontos de coleta, apenas uma única rota para cada ponto de coleta vindo do ponto de partida, todavia a Baia do Arraial não possui muitos obstáculos, facilitando a criação das rotas, mas é possível afirmar, que conforme o número de pontos de partida é acrescentado, a criação de rotas por peso tende a subir mais que os demais, pois é o único que trabalha utilizando todos

os pontos de coleta o tempo todo.

O segundo resultado referente a todo o litoral onde se encontra o Porto do Itaqui, Vale, Ponta da Área, e outras localidades dentro da Baía de São Marcos, envolvendo 27 pontos de coletas.

Figura 48. Gráfico de tempo de criação das rotas na Baia de São Marcos



Fonte: Campos, 2017

Como a Baía de São Marcos possui um número muito maior de pontos de coletas e obstáculos, os resultados apresentados na Figura 50 se patronizarão a partir da inclusão do terceiro ponto de partida, pois os resultados começaram a se assemelhar aos padrões da Baía do Arraial, como o tempo das rotas por grupo se mantendo perto do individual e o aumento do tempo na criação de rotas por peso mais acendente que os demais, a diferença entre as rotas por peso e grupo ocorre, devido a estrutura da rota por grupo, pois ela utiliza tanto a rota individual(para separar em grupos) e o peso para criar as rotas, por isso que nos resultados por usar as duas estruturas ele demora mais quando tem um único ponto de partida, porém começa a ter sua performance melhorada conforme mais pontos de partidas são adicionados, todavia as rotas criadas por ele pode ocorrer uma diferença muito grande no total do peso, ou pode haver ponto de partida sem nem um ponto de coleta ligado a ele.

REFERÊNCIAS

Abraham Silberschatz, Henry Korth e Sudarshan:"Sistema de Banco de Dados 5 Edição", 2006

Ajax. Disponível :http://api.jquery.com/jquery.ajax/. Acesso: julho/2016

Bastos, R. & Jaques, P. "ANTARES: Um sistema Web de consulta de rotas de ônibus como serviço público" Disponível: http://seer.upf.br/index.php/rbca/article/view/650, agosto/2016

Blaich,M.,Rosenfelder M.,Schuster M.,Bittel, O. & Reuter J., "Fast Grid Based Collision Avoidance for Vessels using A* Search Algorithm. In Proc. Of the 17th International Conference on Methods and Models in Automation and Robotics (MMAR)", 2012

Calderón, A. "Desarrollo de una aplicación Web para representación de datos de posicionamiento" Dosponível:https://ddd.uab.cat/pub/trerecpro/2009/hdl_2072_48077/PFC_AnaSierraCalderon.pdf, agosto/2016

Copyright © 2001-2014 The PHP Group. História do PHP. Disponível em: http://www.php.net/manual/pt_BR/history.php.php.

Costa, D. "Avaliação do sistema integrado de gestão ambiental da autoridade Portuária de Itaqui/MA (EMAP) como base para a sustentabilidade das atividades portuárias" Disponível: http://www.ppgse.ufma.br/web/uploads/files/Dairle%20Costa.pdf, agosto/2016

Departament of Mathematics. "Distance between Points on the Earth's Surface". Kansas State University, Diponível: https://www.math.ksu.edu/~dbski/writings/haversine.pdf, agosto/2016

Enovation Solutions – **PHP 5.5 vs PHP 7 performance comparison for Moodle 3.0**. Disponível em: http://www.enovation.ie/php-5-5-vs-php-7-performance-comparison-moodle-3-0/. Acessado em março de 2016

Felipe F., Rodrigo M. & Patrícia J., "UM SISTEMA WEB DE CONSULTA DE TRAJETOS DE TRANSPORTE PÚBLICO", 2011

Francisco E. "Un nuevo algoritmo heurístico para la creación de rutas turísticas personalizadas", dezembro 2014

Gilleanes T. A. Guedes – UML 2 Uma Abordagem Prática, 2009

Goole Maps. Disponível: https://developers.google.com/maps/?hl=pt-br. Acesso: julho/2016.

Hincapié A., Porras R., & GallegoA. “Técnicas heurísticas aplicadas al problema del cartero viajante (TSP). Scientia et Technica, 1(24)”, 2004.

Hornauer S., Hahn A., Blaich M. & Reuter J., “Trajectory Planning with Negotiation for Maritime Collision Avoidance”, Volume 9, numero 3, setembro/2015

MySql. Disponível: http://www.mysql.com/. Acesso: julho/2016.

Neto V. & Chiari N. & Carvalho I. & Pisa. I & Alvez D. “Desenvolvimento e Integração de Mapas Dinâmicos Georreferenciados para o Gerenciamento e Vigilância em Saúde”, Disponível: http://www.jhi-sbis.saude.ws/ojs-jhi/index.php/jhi-sbis/article/view/284

PHP7, Disponível: https://secure.php.net/. Acesso: agosto/2016

Pinheiro E., Costa K., Camila R., Marcos A., Thiago D. & Cleber G. “Navegação autônoma de um agente inteligente: um estudo comparativo usando Lógica Fuzzy e Algoritmo de Busca A*”, volume. 7, número 1, pp. 87-98, 2009

Puente, R. & Cortés, M. "Algorithm for shortest path search in Geographic Information Systems by using reduced graphs". Disponivel: http://springerplus.springeropen.com/articles/10.1186/2193-1801-2-291, agosto/2016

Russel, S. & Norvig, P. "Inteligência artificial". Elsevier, Rio de Janeiro, Brasil.

RUSSEL S. & Norvig P. Inteligência artificial: uma abordagem moderna (3rd ed.). Campus, 2004.

Silva, Maurício Samy – Javascript Guia do Programador, 2011

VOGEL, L. Google Maps Android API v2 – **Tutorial. 2013**. Disponível em: http://www.vogella.com/articles /AndroidGoogleMaps/article.html, acessado em fevereiro de 2016

Yuxin Zhao, Wang Li, Shaojun Feng & Wolfgang Schuster, “An Improved Differential Evolution Algorithm for Maritime Collision Avoidance Route Planning”, novembro/2014