UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
CURSO BACHARELADO EM SISTEMAS DE INFORMAÇÃO

APLICATIVO DE ROTEAMENTO DO MENOR CAMINHO DOS PONTOS TURÍSTICOS DE OEIRAS PARA DISPOSITIVOS MÓVEIS.

LAURENTINO DE MOURA SOUSA JÚNIOR

PICOS - PI
2015

LAURENTINO DE MOURA SOUSA JÚNIOR

APLICATIVO DE ROTEAMENTO DO MENOR CAMINHO DOS PONTOS TURÍSTICOS DE OEIRAS PARA DISPOSITIVOS MÓVEIS.

Monografia submetida ao Curso de Bacharelado de Sistemas de Informação como requisito parcial para obtenção de grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Patrícia Medyna Lauritzen de Lucena Drumond.

PICOS - PI

2015

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

S725a Sousa Júnior, Laurentino de Moura.
Aplicativo do roteamento do menor caminho dos pontos turísticos de Oeiras para dispositivos móveis / Laurentino de Moura Sousa Júnior . – 2015.
CD-ROM : il.; 4 ¾ pol. (53 f.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí, Picos, 2015.
Orientador(A): Profª Me. Patrícia Medyna Lauritzen de Lucena Drumond

1. Sistema Android. 2. Grafo. 3.Dispositivos Móveis.
2. I. Título.

CDD 005

APLICATIVO DE ROTEAMENTO DO MENOR CAMINHO DOS PONTOS TURÍSTICOS DE OEIRAS PARA DISPOSITIVOS MÓVEIS

LAURENTINO DE MOURA SOUSA JÚNIOR

Monografia aprovada como exigência parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Data de Aprovação

Picos – PI. 25 de junho de 2015

Profª Me. Patrícia Medyna Lauritzen de Lucena Drumond
Orientadora

Profª Me. Alcilene Dalília de Sousa
Membro

Prof. Me. Romuere Rodrigues Veloso e Silva
Membro

Dedico este trabalho a minha família, em especial aos meus pais, Laurentino de Moura Sousa e Ronilda Ibiapino Veras Moura, por sempre acreditarem no meu potencial, e não medirem esforços para que eu chegasse até esta etapa da minha vida.

AGRADECIMENTOS

Agradeço a Deus por permitir que eu chegasse a essa etapa da vida com capacidade de poder apreciar os frutos que este trabalho poderá me proporcionar, pela oportunidade e pelo privilégio que nos foram dados em compartilhar tamanha experiência.

À minha família pelo apoio incondicional, em especial a meus pais, Laurentino Moura e Ronilda Ibiapino, pelo suor derramado em prol da formação profissional de todos os filhos. A minhas irmãs Lelia Ibiapino e Amália Ibiapino, pelos conselhos e exemplos que sempre me guiaram pelo caminho certo. A minha sobrinha Giovana Ibiapino, pelos momentos de lazer proporcionados.

A minha orientadora, Patrícia Medyna, por sempre estar disposta a ajudar e partilhar seu conhecimento. Agradeço pela confiança depositada em meu trabalho e em minhas decisões, me proporcionando cada vez mais segurança para seguir com todas as atividades necessárias para minha formação. Agradeço também pela paciência e dedicação ao curso e a todos os seus alunos, buscando a melhor forma possível para atender a todos.

A todos os professores do curso de Sistemas de Informação. Em especial a professora Juliana Oliveira, por repassar de forma clara e objetiva técnicas de programação que foram de grande relevância para o desenvolvimento do sistema.

Ao meu amigo Carlos Alberto por, mesmo distante, participar dessa etapa da minha vida sempre dando conselhos que me fizeram crescer como pessoa e como profissional. Ao meu amigo Antônio Carlos Moura, por despertar em mim maior interesse na área da tecnologia. Aos amigos Flávio Escobar e Marcelo Damasceno, por sempre estarem perto, e por sempre me escutarem e ajudarem quando necessário. E a todos os colegas e amigos que sempre me apoiaram nessa jornada.

A todos vocês muito obrigado!

"O sucesso nasce do querer. Sempre que o homem aplicar a determinação e a persistência para um objetivo, ele vencerá os obstáculos e, se não atingir o alvo, pelo menos fará coisas admiráveis."

(José de Alencar)

RESUMO

O presente trabalho tem como objetivo desenvolver um aplicativo para dispositivos móveis utilizando *ANDROID* para verificar os pontos turísticos da cidade de Oeiras e a melhor rota para percorrer os pontos desejados pelo usuário. A relevância social do trabalho se dá pela necessidade das pessoas que visitam o município pela primeira vez poderem conhecer a importância histórica do município utilizando seu *smartphone* e a partir de sua localização encontrar as rotas para conhecer os pontos turísticos. Inicialmente, foram realizados estudos sobre problemas de roteamento utilizando grafos e algoritmos de otimização além de estudos sobre sistema operacional *ANDROID* e a linguagem de programação adequada ao desenvolvimento do aplicativo. Após a implementação, o aplicativo foi testado em vários tipos de *smartphones* para comparação do desempenho.

Palavras-chave: Dispositivos Móveis, Aplicativo *ANDROID*, Otimização Combinatória.

ABSTRACT

This study aims to develop an application for mobile devices using ANDROID to check the sights of the city of Oeiras and the best route to go the desired points by the user. The social relevance of the work is given by the need of the people visiting the city for the first time be able to meet the historical significance of the city using your smartphone and from your location to find the routes to see the sights. Initially, studies were performed on routing problems using graphs and optimization algorithms as well as studies on ANDROID operating system and the appropriate programming language to develop the application. After implementation, the application has been tested on various types of smartphones for performance comparison.

Keywords: Mobile Devices, Application ANDROID, Combinatorial Optimization.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

OPSF	*Open Shortest Path First*
PCVA	Problema do Caixeiro Viajante Assimétrico
PCV	Problema do Caixeiro Viajante
CMC	Caminho Mais Curto
AGM	Árvore Geradora Mínima
M³S	Árvore Min-Max-Min-Sum
IDE	*Integrated Development Environment*
SDK	*Software Development Kit*
ADT	*Android Development Tools*
AVD	*Android Virtual Devices*
API	*Application Programming Interface*
GPS	*Global Positioning System*

SUMÁRIO

1 INTRODUÇÃO

A cidade de Oeiras foi a primeira capital do Piauí e conta sua história por meio de um preservado patrimônio cultural. Possui boa parte de sua economia baseada no turismo histórico e religioso, trazendo de diversas regiões do Brasil pessoas com intuito de aprender mais sobre a história de seu país. As pessoas que visitam a cidade necessitam se localizar para percorrer os diversos pontos turísticos sem perder tempo e utilizando um bom percurso. Além disso, a tecnologia está presente cada vez mais na vida das pessoas que deixaram de ter um simples dispositivo móvel para comunicação e passaram a utilizar os *smartphones* mais modernos gerenciados pelo sistema operacional *Android*, sendo considerada uma das mais comuns e crescentes tecnologias de comunicação atual.

É visando auxiliar o turista na busca pelo conhecimento histórico da região com tecnologias modernas, que o presente trabalho expõe o desenvolvimento de um aplicativo para facilitar a interação do usuário informando o caminho mais curto do ponto de localização atual aos pontos turísticos da cidade que deseja visitar. Além disso, o aplicativo possui todas as informações históricas, culturais, políticas e sociais do município.

Na implementação do aplicativo foi necessária a modelagem do problema real de localização e roteamento através de grafos e de um problema clássico de otimização combinatória, como por exemplo, o caminho mais curto que é bastante utilizado em pesquisas com aplicações em roteamento. Os Problemas de caminhos mínimos são clássicos das áreas de teoria dos grafos e otimização combinatória. De maneira geral, encontrar o caminho mais curto em um grafo significa encontrar o caminho que une dois vértices e que minimiza uma determinada função objetivo. Na *Internet*, por exemplo, um dos protocolos que podem ser usados para o roteamento de pacotes é o OPSF (*Open Shortest Path First*), que procura o menor caminho disponível para o envio das informações (BURIOL *et al*., 2005).

Em sistemas de navegação, que têm se tornado comuns em carros populares, caminhos mais curtos ajudam tanto a planejar como a otimizar os recursos disponíveis. A Otimização Combinatória possui aplicações em diversas áreas, tais como, problemas de planejamento e programação (scheduling) da produção, roteamento de veículos, redes de telecomunicações, problemas de localização e vários outros (JASZKIEWICZ, 2002).

1.1 Objetivo

O presente trabalho teve como objetivo principal desenvolver uma aplicação para *Android* que calcule a melhor rota possível em grafos para expor pontos turísticos da cidade de Oeiras-PI. A aplicação também deve informar ao turista dados históricos do local selecionado.

1.2 Organização do Documento

Este trabalho está organizado em cinco capítulos. No capítulo 2, encontra-se todo embasamento teórico necessário para o desenvolvimento do projeto, são apresentados os conceitos básicos relacionados a grafos, bem como suas representações, tipos e problemas e os conceitos de *Android,* suas formas de implementação e estrutura básica de desenvolvimento.

No capítulo 3, é apresentada as principais funcionalidades que o sistema apresenta, os diagramas produzidos para o entendimento do problema, o desenvolvimento do sistema e seus recursos de *layout*. Os resultados e discussões estão presentes no capítulo 4. O Capitulo 5 traz as considerações finais e as indicações de trabalhos futuros.

2 ESTADO DA ARTE

Neste capítulo, são apresentados os conceitos básicos que alicerçam o desenvolvimento deste trabalho, tais como grafos, e seus modelos de representação e problemas com grafos. Para conhecimento sobre o aplicativo que foi desenvolvido é necessário também o conhecimento da tecnologia de dispositivos móveis e o sistema operacional *Android*.

2.1 Grafo

Um grafo é um modelo matemático representativo definido por um conjunto finito de vértices e de arestas. Cada aresta associa dois vértices quaisquer, inclusive há a possibilidade de um vértice estar conectado a ele mesmo criando um laço. Segundo Goldbarg (2012), grafo é um modelo gráfico abstrato de representação e solução de um problema em diversas áreas. Consiste em definir um conjunto para nós ou vértices, representados graficamente por círculos ou quadrados, e um conjunto para arestas ou arcos, representados graficamente por uma linha direcionada ou não, conforme Figura 1.

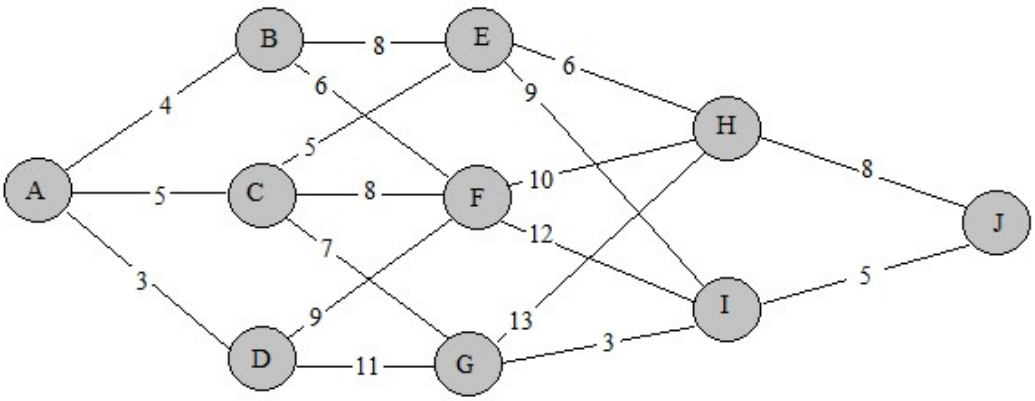


Figura 1 - Exemplo de grafo, (GOLDBARG, 2005).

Existem diversos tipos de grafos que são categorizados de acordo com características próprias tais como, ser direcionado ou não, ponderado ou não. Há também classificações especiais quanto aos caminhos contidos em um grafo, podendo ele estar inserido em mais de uma categoria. Um grafo pode representar de forma simplificada um modelo de um problema

do mundo real, por exemplo, o roteamento de veículos pode ser modelado por um grafo, onde os vértices do grafo significam os cruzamentos das rodovias e as arestas são as rodovias.

2.1.1 Representação de Grafos

Os grafos podem ser representados através do gráfico utilizando círculos e linhas como apresentado anteriormente na Figura 1, ou ainda, utilizando uma matriz que contenha todos os vértices e o peso[1] das arestas caso haja conexão.

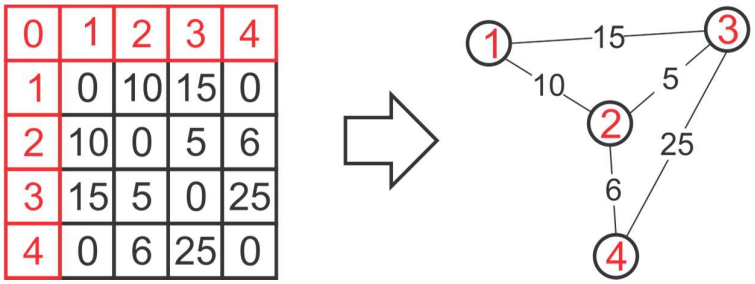


Figura 2 – Exemplo de grafo por meio de matriz (o Autor)

A Figura 2 apresenta um grafo em forma de matriz, onde toda a linha zero e toda a coluna zero indicam os vértices do grafo exceto a posição Matriz [0] [0] que indica um valor nulo. Os demais valores indicam o peso de cada aresta, o peso zero indica que não existe ligação.

Segundo Ziviani (2005), o uso de matrizes para representação de grafos está habitualmente associado à realização de cálculos envolvendo os dados de um grafo. A matriz de adjacência é a mais utilizada, onde uma matriz da mesma ordem do grafo associa para cada linha e cada coluna um vértice. Os dados estruturais correspondem a valores nulos, caso não exista ligações, e valores não nulos para representação das arestas. Em caso de grafos não valorados caso haja ligação entre os vértices o valor da estrutura M [i] [j] costuma receber o valor um, em grafos valorados o valor dessa estrutura é igual ao peso da aresta conforme Figura 2.

Trata-se de uma matriz figurativa aquela em que os vértices, ou ligações, são representados por cadeias de caracteres ao invés de números são utilizados problemas de enumeração combinatória em grafos, podendo conter também como estrutura da matriz os

[1] Peso é o mesmo que o valor de custo de uma aresta.

rótulos de vértices e arestas. Os exemplos mais comuns são as matrizes latinas que são matrizes quadradas conforme apresentado na Figura 3. (BOAVENTURA NETTO, 2006).

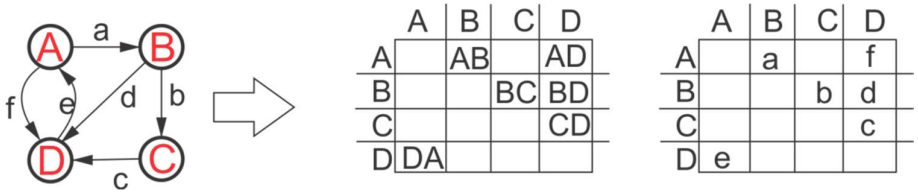


Figura 3 - Grafo representado por matriz figurativa (o Autor)

Segundo Boaventura Netto (2006), utilizando uma matriz que possua dimensões n x m, onde cada linha corresponde a um vértice e cada coluna corresponde a uma ligação, é possível representar um grafo. Esse método é chamado de matriz de incidência e está representado na Figura 4. Observe que no vértice 0 e ligação 1, tem valor positivo (+1), porque no vértice 0 ocorre uma incidência de valor 1 saindo deste vértice enquanto que o valor no mesmo vértice 0 e ligação 5, o valor é negativo (-1), indicando que há uma ligação de chegada no vértice 0.

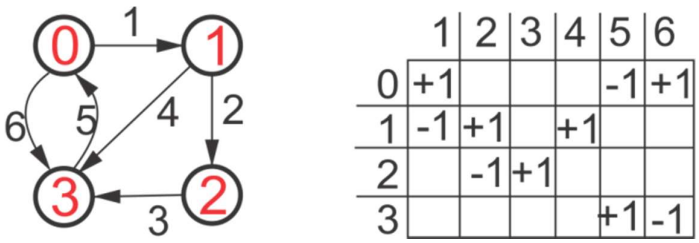


Figura 4 - Grafo representado por matriz de incidência (o Autor)

Segundo Boaventura Netto (2006), a forma mais adequada para a entrada de dados, devido à economia e simplicidade, é a lista de adjacência ou dicionário. Consiste em construir um conjunto de listas de vértices, cada lista é formada por um vértice e pelo conjunto de vértices que possui ligação com ele. Em caso de grafos direcionados ou orientados, existe a opção de formar listas com o vértice que envia um arco ao vértice inicial, desse modo são criadas duas listas de adjacências conforme Figura 5.

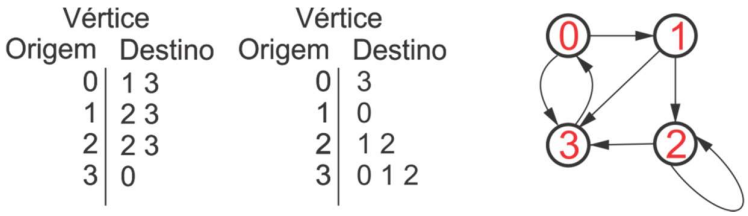


Figura 5 - Grafo orientado representado por lista de adjacência (o Autor)

2.1.2 Tipos de Grafos

Por ser um modelo representativo muito abrangente, grafos são classificados de acordo com suas características individuais. Essas características que definem o que eles representam e o contexto em que estão inseridos.

Segundo Boaventura Netto (2006), um grafo é uma estrutura G = (V, A) onde V é o conjunto dos vértices e A é uma família cujos elementos (não vazios) são delimitados em função de V.

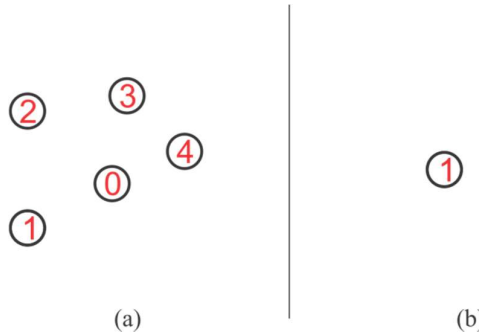


Figura 6 - (a) Grafo vazio. (b) Grafo trivial (o Autor)

Para Goldbarg (2012), o grafo exposto na Figura 6(a) é caracterizado como grafo vazio, pois contém unicamente vértices. Caso o conjunto V contenha apenas um único elemento e o conjunto E for vazio, o grafo G é conhecido como grafo trivial, conforme exposto na Figura 6(b). Existe ainda a possibilidade de um grafo G = (V, A), onde o conjunto de vértices V esteja vazio e o grafo é tratado como nulo.

Há grafos que possuem arestas diferentes para conectar o mesmo par de vértices, e arestas que conectam um vértice a ele mesmo. Esses tipos de arestas recebem notação especial de arestas paralelas e laço ou *self-loops,* respectivamente. Quando se trata de grafo, qualquer informação que se possa extrair torna-se relevante para o desenvolvimento de alguma atividade, por exemplo, o grau de incidência em um vértice pode incluir o grafo em outra categoria. (ZIVIANI, 2005).

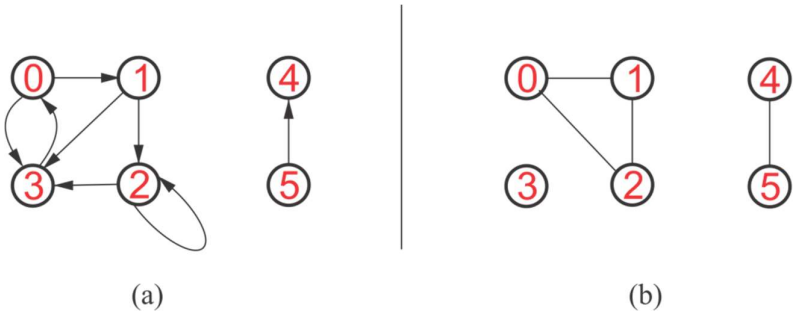
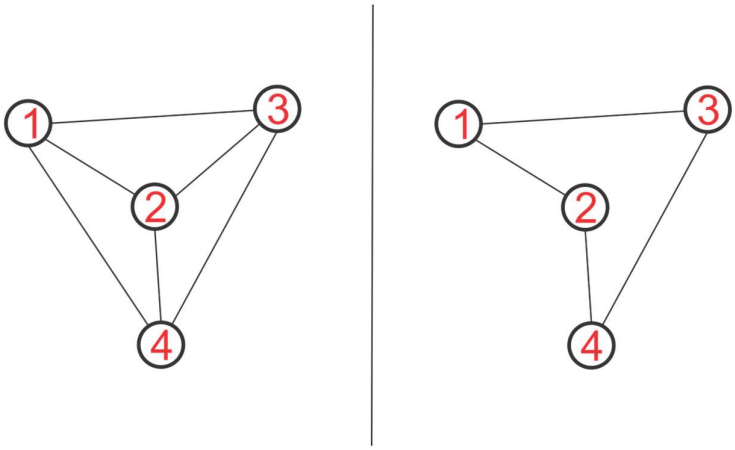


Figura 7 - (a) Grafo direcionado. (b) Grafo não direcionado, (ZIVIANI, 2005)

Na Figura 7 (a) é um exemplo de grafo direcionado em que o vértice 2 possui uma aresta A = (2, 2), exibindo um exemplo de *self-loop,* e os vértices 0 e 1 possuem arestas paralelas.

Segundo Ziviani (2005), um grafo G = (V, A) é tido como direcionado quando o conjunto de arestas A for de relação binária com o conjunto de vértices V, desse modo às arestas são ordenadas e são representadas por setas conforme exibido na Figura 7 (a). Na Figura 7 (b) o grafo G = (V, A) o conjunto de arestas A é constituído por pares de vértices não ordenados representando assim um exemplo de grafo não direcionado, desse modo o vértice 2, por exemplo, tem incidência no vértice 1, o que não ocorre no exemplo do grafo direcionado na Figura 7 (a).

Como citado anteriormente, em grafos cada detalhe serve como índice classificatório, o número de conexões entre os vértices de um grafo e o grau de incidência em cada vértice geram uma nova categoria. Segundo Goldbarg (2012), um grafo é dito completo quando existe uma aresta associada a cada par de vértices e no caso de grafos direcionais, é necessário que haja um arco para cada par de vértices, desse modo, um grafos onde qualquer vértice esta conectados a todos os demais, são chamados de grafos completos como exposto na Figura 8 (a).



(a) (b)

Figura 8 - (a) Grafo completo. (b) Grafo Regular (o Autor)

Diferente do grafo da Figura 8 (a), o grafo da Figura 8 (b) não é completo e retrata um exemplo de grafo regular, outra categoria onde o grau de incidência tem que ser o mesmo para todos os vértices, esse grau de incidência é tido pelo número de arestas que incidem sobre o vértice.

Segundo Boaventura Netto (2006), um grafo é tido como subgrafo de outro, quando o seu conjunto de vértices V' estiver contido no conjunto de vértices V do grafo inicial, e o seu conjunto de arestas A' estiver contido no conjunto de arestas A do grafo inicial assim como exposto na Figura 8. O grafo representado na Figura 8 (b) é subgrafo ou uma sub-estrutura do grafo apresentado na Figura 8 (a), pois todos os seus vértices e suas arestas estão contidos no grafo principal.

Dentre todos os tipos de grafos especiais, existem aqueles em que seu conjunto de vértices pode ser dividido em dois não havendo conexão dentro de um mesmo conjunto apenas de conjunto para conjunto, esse tipo de grafo é denominado de bipartido. Segundo Goldbarg (2012), um grafo G = (N, M) é denominado bipartido quando seu conjunto de vértices N puder ser dividido em dois conjuntos N1 e N2 onde a intercessão entre eles seja vazia e a união seja igual a N, existindo apenas arestas ligando algum vértice de N1 para N2 e vice-versa conforme apresentado na Figura 9.

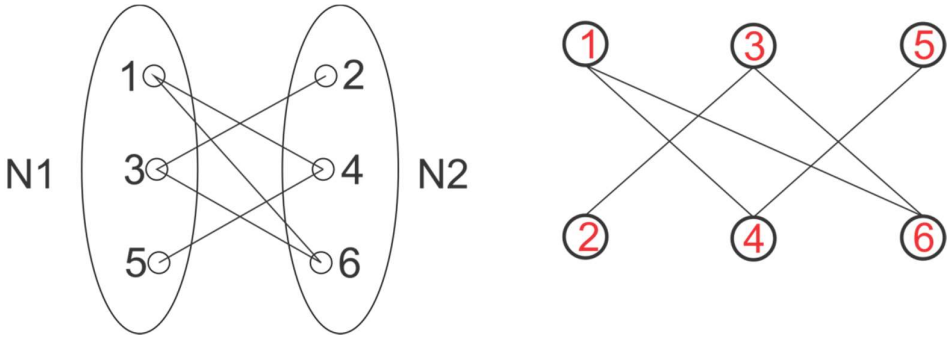


Figura 9 - Grafo bipartido, (Goldbarg, 2012)

É de extrema importância considerar a intensidade da possibilidade de ligações em um grafo, podendo assim classifica-los como conexos ou desconexos. Um grafo é considerado conexo se existir pelo menos uma caminho entre qualquer par de vértices, se o grafo for direcionado, então é dito conexo quando seu grafo subjacente é conexo (GOLDBARG; GOLDBARG, 2012).

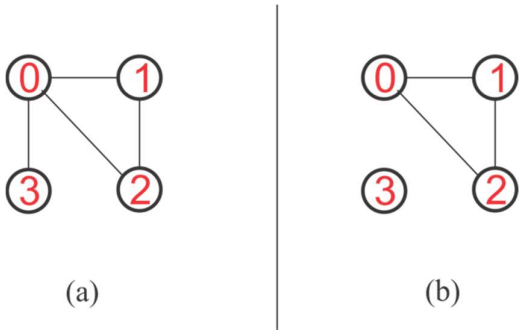


Figura 10 - (a) Grafo conexo. (b) Grafo desconexo (o Autor)

A Figura 10 (a) mostra um exemplo de um grafo conexo. Percebe-se que existe pelo menos um caminho para conectar qualquer par de vértices. O grafo exposto na Figura 10 (b) é dito desconexo porque nem todos os pares de vértices estão ligados por um caminho. Por exemplo, partindo do vértice 3 é impossível se chegar a qualquer outro vértice do grafo. Segundo Bulat (2004), um grafo é classificado como isomorfo quando for possível re-rotular os vértices de G para serem rótulos de G' mantendo as arestas de forma correspondente, ou seja dois grafos G = (V, A) e G' = (V', A') são isomorfos quando existe uma bijeção f: V → V' de modo que (u, v) pertencente ao conjunto A se somente se (f(u),f(v)) for pertencente a A', conforme retratado na Figura 11.

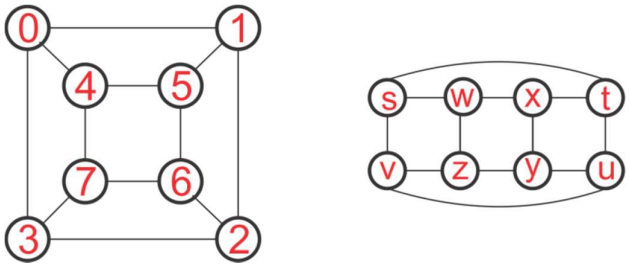


Figura 11 - Dois grafos isomorfos (ZIVIANI, 2004).

A grande maioria dos grafos requer valores numéricos associados às suas arestas, e um rótulo de identificação para vértices e arestas, esses valores são denominados pesos e representam o custo entre um par de vértices ou o custo entre um vértice e ele mesmo em caso de *self-loop,* grafos assim são denominados ponderados e rotulados. Para Boaventura Netto (2006), um grafo é tido como rotulado se houver atribuições associadas as suas arestas ou vértices podendo ser numéricas ou alfabéticas e um grafo ponderado é aquele que existe valores numéricos (pesos) relacionados às suas arestas ou vértices conforme exposto na Figura 12 (a).

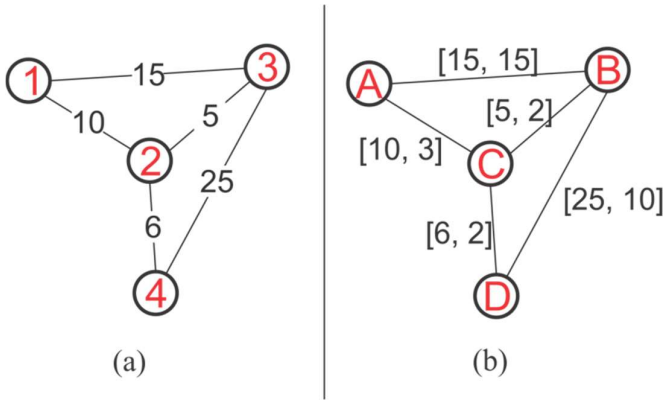


Figura 12 - Grafo rotulado e ponderado. (o Autor)

O grafo da Figura 12(a) mostra um exemplo de grafo ponderado e rotulado, nele os vértices são rotulados por V = {A, B, C, D}, e as arestas possuem um peso, por exemplo, a aresta A = (A, B), possui peso 15. Essas informações podem conter um maior sentido associando o grafo a um problema real, por exemplo, se cada vértice for bairros em uma cidade, e o peso da aresta for a distância entre eles em quilômetros, associando dessa forma é possível obter resultados para diversos problemas como, calcular a menor rota entre dois bairros. Existem também grafos mais complexos onde a forma em que estão ponderados pode conter duas ou mais informações como o grafo da Figura 12 (b). Direcionando para a forma prática com o exemplo anterior, além da distância entre os bairros é encontrado também o tempo necessário para percorrer toda a rua. Esse tipo de grafo requer uma atenção especial

devido à complexidade para solução de seus problemas uma vez que será levado em conta não apenas uma variável. Em grafos existem uma série de problemas relacionado aos caminhos que ele possui. Os caminhos hamiltonianos e cadeias eulerianas são os mais tradicionais. Segundo Jordán (2013), um ciclo hamiltoniano é aquele que contém todos os vértices de um grafo, e um grafo é dito hamiltoniano quando possui um ciclo hamiltoniano.

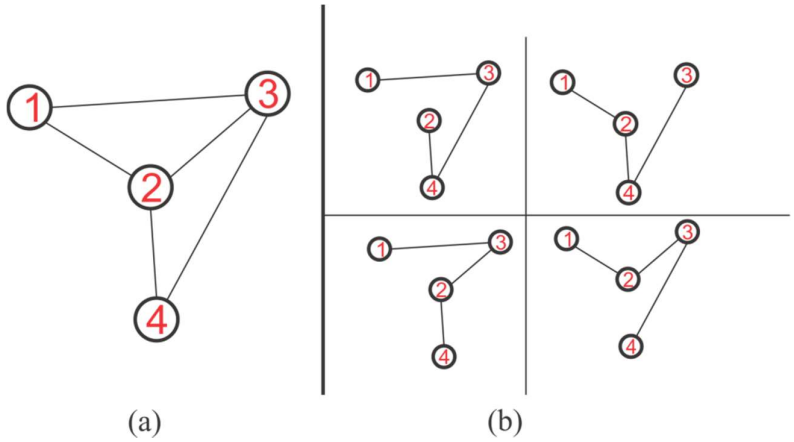


Figura 13 - (a) Grafo hamiltoniano. (b) Caminhos hamiltonianos (o Autor)

Para Goldbarg (2012), um grafo G que contém um caminho que percorra todos os vértices passando por eles apenas uma vez, é tido como grafo hamiltoniano conforme apresentado na Figura 13 (a). A Figura 13 (b) apresenta alguns caminhos que possibilitam a classificação do grafo nessa categoria.

Outra categoria tradicional para se classificar um grafo, é a verificação da existência de cadeias eulerianas. Para Recuero (1994), uma cadeia euleriana é uma cadeia que passa por todas as arestas de um grafo apenas uma vez, e um grafo que possui uma cadeia euleriana é tido como um grafo euleriano conforme mostrado na Figura 14 (a). A Figura 14 (b) retrata uma cadeia euleriana do grafo a esquerda, é importante ressaltar a ordem na qual às arestas devem ser percorridas.

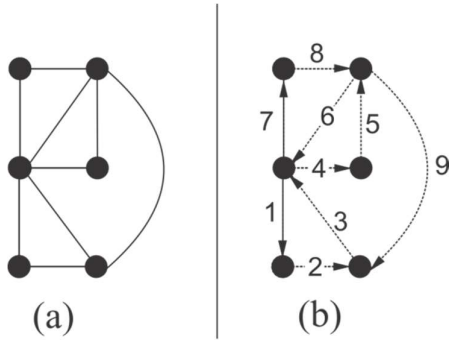


Figura 14 - (a) Grafo euleriano. (b) Caminho euleriano (GOLDBARG, 2012).

2.1.3 Problemas com grafos

Os Problemas clássicos resolvidos por grafos disponibilizam soluções alternativas para problemas atuais, sendo possível a aplicação de vários algoritmos que apresentam soluções heurísticas que satisfazem a necessidade do usuário.

O problema do caixeiro viajante assimétrico (PCVA), por exemplo, define a rota pelo qual o caixeiro deve percorrer para que possa visitar todas as cidades sem ter que passar pela mesma cidade mais de uma vez retornando ao ponto inicial. Observando o modelo representativo de um grafo, cada cidade representa um vértice e as estradas que ligam as cidades são vistas como arestas, desse modo pode ser dado um peso para as arestas tornando possível traçar a melhor rota hamiltoniana, ou seja, a rota que termina no ponto onde começa visitando todos os vértices e formando um ciclo. (BOAVENTURA NETTO, 2006).

O problema do caixeiro-viajante (PCV) é um clássico dos problemas da otimização combinatória onde dado um grafo ponderado g= (N, M), em que o conjunto de vértices é representado por N={1,...,n} e o conjunto de arestas é representado por M={1,...,m}, é extraído um subgrafo hamiltoniano de menor custo. (GOLDBARG; GOLDBARG, 2012).

Segundo Goldbarg (2012), foram desenvolvidos vários algoritmos heurísticos que se aplicam ao (PCV), esses algoritmos possuem complexidades diferentes e alcançam seu objetivo utilizando técnicas diferentes conforme alguns exemplos abaixo:

- A heurística do "Vizinho mais próximo" é uma das mais simples heurísticas que trabalham de forma construtiva. O algoritmo de Bellmore & Nemhauser promove solução para esta heurística, tal algoritmo possui complexidade $O(n^2)$ e está representado no Quadro 1.

Quadro 1 - Algoritmo de Bellmore & Nemhauser (GOLDBARG, 2012)

```
    Algoritmo de Bellmore & Nemhauser
1   Ler G = (N, M)
2   Escolher um vértice inicial h1 e inclui-lo em H ← ( h1)
3   Enquanto |H| < n Fazer
4       Encontrar o vértice hk  não pertencente a H mais próximo de h1
5       Insere o vértice hk após h1  em H (o seu vizinho mais próximo)
6           i ← k
7   Fim_Enquanto
```

Consiste em delimitar um vértice inicial e em seguida verificar qual o seu vizinho mais próximo que não esteja contido na lista de vértices que indica o caminho, uma vez encontrado, o vizinho é inserido na lista e o valor do vértice inicial passa a ser o vértice do vizinho encontrado.

- A heurística de "Inserção de Vértices", segundo Goldbarg (2012), alguns tipos de heurística que trabalham de forma construtiva possuem processos mais elaborados para a tomada de decisão, diferente de algumas estratégias gulosas. A heurística de inserção possui critérios para definir os vértices que serão atribuídos como caminho, dentre os critérios, os mais utilizados são:
    - ✓ Inserção do vértice mais próximo (associado a uma aresta de menor custo).
    - ✓ Inserção do vértice mais distante (associado a aresta de maior custo).
    - ✓ Inserção do vértice (aresta) que conduz a um ciclo de menor custo (inserção mais barata). Esse critério promove um algoritmo de complexidade $O(mn^2)$, diferente dos critérios anteriores que proporcionam complexidade $O(mn)$.
    - ✓ Inserção aleatória.
- Heurística de "Inserção de Arestas", segundo Goldbarg (2012), essa heurística consiste em permutar arestas em um ciclo formado.
- Heurística de Inserção e Deslocamento de Vértice – Shift, é uma forma bastante eficiente e de fácil implementação. Possui complexidade $O(n^2)$ e consiste em inserir um vértice entre dois outros em meio a um ciclo, deslocando todos os demais vértices desse ciclo em uma posição, esse deslocamento ocorre para todos os demais vértices do ciclo. (GOLDBARG; GOLDBARG, 2012).
- Heurística Exchange, para Goldbarg (2012), essa também é uma heurística simples e de fácil implementação, em sua forma mais simples possui complexidade $O(n)$ e consiste em permutar um vértice ao longo de um vetor de representação do ciclo, ocupando todas as posições de permutação possíveis – que são **n**.
- Heurística de Christofides, possue uma etapa de aperfeiçoamento do critério para seleção das arestas, que permita encontrar um ciclo euleriano e transforma-lo em um ciclo hamiltoniano. A heurística é limitada pela complexidade que é $O(n^3)$.

O problema do caminho mais curto é outro problema clássico muito difundido na literatura e consiste em, dado um grafo G= (N, A) e dois vértices desse grafo *u* e *v*, o caminho mais curto entre *u* e *v* é uma sequência de arestas que, passando por vértices distintos, liga *u* a *v* de forma a acumular o menor comprimento, ou distância. Para que se possa determinar o caminho mais curto entre dois vértices, é indispensável que exista uma conexão entre os vértices.

Segundo Goldbarg e Luna (2005), o problema do caminho mais curto está contextualizado dentro do problema de maior percurso em grafos, envolvendo caminhos mais curtos, caminhos mais longos, e percursos hamiltonianos e eulerianos.

Para Boaventura Netto (2006), Existem dois tipos de problemas de caminho mínimo, os que precisam da definição de um vértice para determinar o caminho, e os que determinam o caminho unindo todos os pares de vértices do grafo. Existem algoritmos criados para solucionar ambos os casos, embora, naturalmente, os algoritmos criados para resolver o primeiro caso, onde é definido o vértice para determinação do caminho, pode ser aplicado *n* vezes de modo que também possa resolver o problema descrito no segundo caso, onde determina o caminho unindo todos os pares de vértices de um grafo.

Além disso, existem problemas de otimização que se deve considerar mais de um objetivo conflitantes para minimizar, por exemplo, quando um motorista viaja de uma cidade a outra, não pensa somente no tempo, mas também no trânsito, na conservação das pistas e até no relevo do terreno, devido a seu efeito no consumo de combustível. Isto significa que dificilmente haverá apenas uma solução que consiga minimizar todos os objetivos simultaneamente, pode-se encontrar um conjunto de soluções de tal forma que não pode se comparar uma com a outra para saber qual é a melhor. Uma pode ter o menor caminho e o pior trânsito, enquanto que outra pode ter o caminho mais longo e o melhor trânsito, a tomada de decisão depende do objetivo mais importante a minimizar.

Por ser um modelo gráfico que representa situações reais, problemas envolvendo grafos podem ser solucionados e mais bem visualizados quando expressados em fórmulas matemáticas. O problema do caminho mais curto pode ser formulado conforme a fórmula 1, onde os vértices *o* e *d* representam os vértices de início e término do caminho. Para essa formulação, a matriz de incidência é totalmente unimodular, permitindo que a exigência de integridade seja relaxada sem qualquer prejuízo para a solução interna, se a regra de Cramer para obter a solução desse sistema for usada (GOLDBARG, 2005).

$$\text{(CMC) Minimizar } z = \sum_{(i,j)\in A} C_{ij}X_{ij}$$

$$\text{Sujeito a:}$$

$$\sum_{(i,j)\in A} X_{ij} - \sum_{(k,i)\in A} X_{ki} = \begin{cases} -1 \; se \; i = o \\ 0 \; se \; i \neq o \; \text{e} \; i \neq d \\ +1 \; se \; i = d \end{cases}$$

$$X_{ij} \in \{0,1\} \qquad (i,j) \in A$$

Fórmula 1 - Representação matemática do problema do caminho mais curto.

Diversos algoritmos foram criados com o intuito de solucionar o problema do caminho mais curto, os mais eficientes são disponíveis através de abordagem em grafos (GOLDBARG; LUNA, 2005).

O algoritmo de Dijkstra é um algoritmo simples com complexidade $O(n^2)$ onde n representa o número de vértices que servem para encontrar o caminho mais curto. Este algoritmo consiste em definir em um grafo, onde os arcos possuem um peso positivo, o menor caminho entre os vértices *u* e *v* somando o peso das arestas, ao final, o caminho com menor valor é assumido como resultado (TORRUBIA; TERRAZAS, 2010).

Outro problema clássico gerado a partir de um grafo são as árvores geradoras mínimas (AGM). Segundo Goldbarg e Luna (2005), a importância da estrutura de uma árvore é enorme tanto na parte pratica, envolvendo problemas de otimização, quanto na parte de solução de vários outros problemas, tendo como principais aplicações pesquisas relacionadas a problemas de conexões e comunicação.

Segundo Ziviani (2005), uma árvore de um grafo G = (N, A) é um subgrafo conexo e acíclico T, e recebe o nome de arvore geradora, uma vez que T gera um novo grafo. No caso da arvore geradora T tiver comprimento mínimo entre as arestas, T recebe o nome de arvore geradora mínima (AGM). A Figura 15 apresenta um exemplo de AGM.

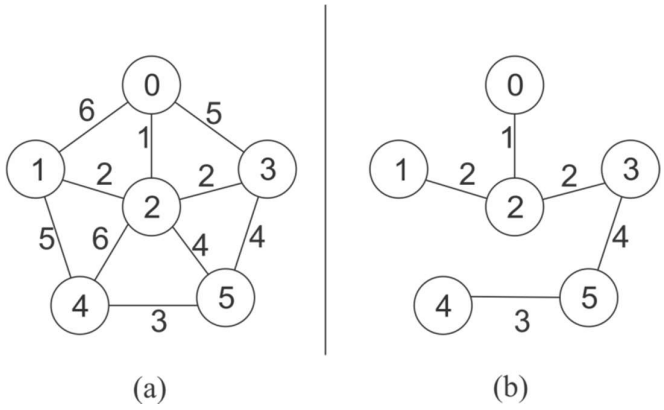


Figura 15 - (a) Grafo G. (b) Arvore geradora mínima T de peso 12 (ZIVIANI, 2005).

A Figura 15 (a) mostra um exemplo de um grafo não direcionado G com os pesos mostrados ao lado de cada aresta, a Figura 15 (b) mostra a árvore geradora mínima T cujo peso total é 12. Para Goldbarg (2005), as conexões em árvores comportam mais modelos de otimização, e dois deles são diferentes da estrutura de AGM e serão expostos nos tópicos abaixo:

- Árvore Min-Max, esse problema também é conhecido como otimização de gargalo e tem como objetivo determinar a árvore que possua a menor aresta máxima.

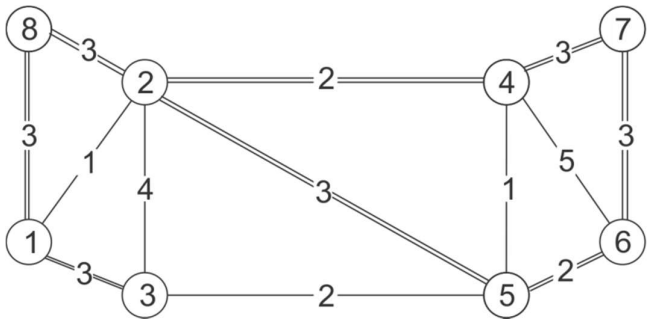


Figura 16 - Exemplo de árvore geradora MinMax. (GOLDBARG, 2005)

A Figura 16 apresenta um exemplo de árvore geradora MinMax onde o gargalo é igual a 3, uma vez que esse seja o menor valor das arestas que incidem nos vértices 7 e 8.

- Árvore Min-Max-Min-Sum (M^3S), é a composição de dois dos problemas de otimização combinatória mais conhecidos, são eles, o problema de *bottleneck* ou gargalo, onde o objetivo é minimizar o custo máximo dentre todas as variáveis que compõem a solução, e o problema de custo linear, onde o objetivo é minimizar a soma dos custos dessas variáveis. A M^3S tem como objetivo minimizar a soma algébrica das funções objetivo dos problemas mencionados.

Segundo Ziviani (2005), existem dois algoritmos que utilizam estratégias gulosas para obtenção da AGM, o algoritmo de Prim (1957) e o algoritmo de Kruskal (1956), ambos, por serem gulosos devem fazer, a cada passo, uma escolha dentre várias possibilidades sendo ela a melhor possível.

- Algoritmo de Prim, segundo Ziviani (2005), apresenta derivações do algoritmo genérico apresentado no Quadro 2.

Quadro 2 - Algoritmo genérico para obtenção da AGM. (ZIVIANI, 2005).

```
     Algoritmo genérico para obtenção da AGM
     Procedure GenericoAGM;
1    S:=0;
2    While S não constitui uma arvore geredora mínima do
3          (u, v) := seleciona (A);
4        If aresta (u, v) é segura para S then S:= S + {(u, v)}
5    return S;
```

Ele utiliza uma regra especifica para determinar uma aresta segura, portanto o subconjunto S forma uma única árvore, e a aresta segura adicionada a S é sempre a aresta de menor custo. A execução do algoritmo de Prim está exemplificada na Figura 17.

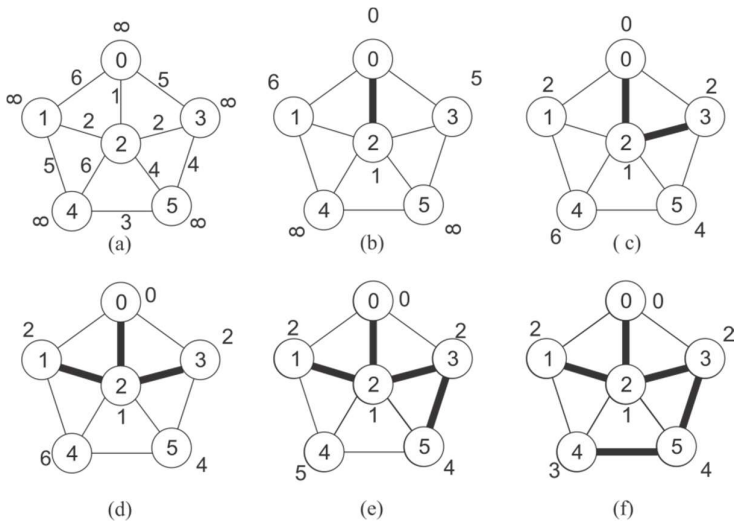


Figura 17 - Execução do algoritmo de Prim. (ZIVIANI, 2005).

- Algoritmo de Kruskal, segundo Ziviani (2005), esse algoritmo também possui aspectos semelhantes ao algoritmo generico apresentado no Quadro 2. Nesse algoritmo o conjunto S é uma floresta e a aresta segura é sempre a aresta de menor peso que conecta dois componentes distintos assim como apresentado na Figura 18.

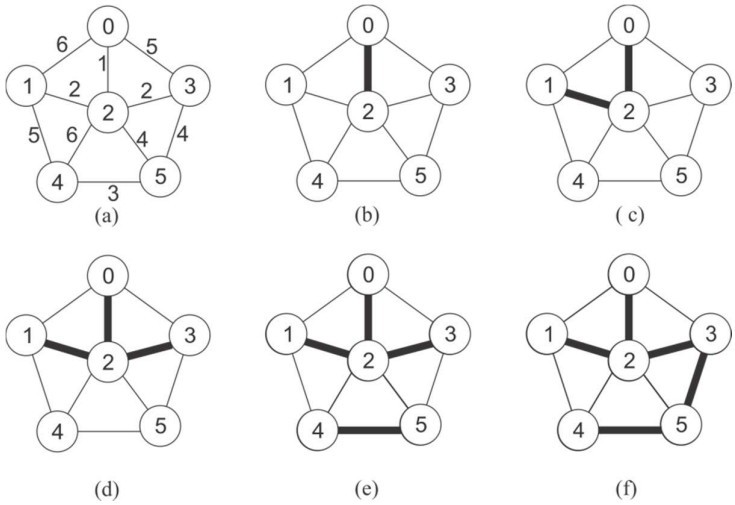


Figura 18 - Execução do algoritmo de Kruskal. (ZIVIANI, 2005).

Na próxima seção são apresentados os conceitos de *Android* por se tratar do sistema operacional utilizado nos dispositivos móveis para o qual o aplicativo foi desenvolvido.

2.2 Android

O sistema operacional *Android* foi desenvolvido para gerenciar recursos de dispositivos móveis em e teve a primeira geração lançada em 2008. Os dados a seguir exibem o sucesso da venda de dispositivos com *Android*, e a superioridade em relação aos concorrentes, tal superioridade e dada por ser mais acessível e possuir uma interface bastante agradável e usual.

Devido ao grande número de usuários, programadores de todo o mundo voltaram os olhos para dispositivos com *Android* visando desenvolver aplicações que promovessem eficiência em tarefas diárias, comunicação, e entretenimento para os usuários. Segundo Deitel *et. al.* (2013), o que possibilita a criação de aplicações é a fraqueza (ou grau de abertura) de uma plataforma. Isso acaba sendo uma vantagem do sistema operacional *Android*, pelo fato de possuir código-fonte aberto e gratuito, permitindo uma melhor interação com o sistema ao disponibilizar a forma como seus recursos são implementados.

Linguagens de programação de alto nível, são linguagens desenvolvidas para facilitar a implementação de programas, tendo em vista que o homem teria maior dificuldade de implementar em linguagem de máquina. Tais linguagens atuam de forma diferente, pois não são compreendidas diretamente pelo computador sendo necessária que haja uma tradução para a linguagem adequada. A tradução é feita por compiladores e requer algum tempo para

conversão, que é compensado pela facilidade de escrita do código já que utiliza palavras reservadas mais facilmente assimiladas pelo homem.

Segundo Deitel (2010), a linguagem de programação Java é de longe a mais utilizada no mundo. Desenvolvida por James Gosling em 1991, a linguagem baseada em C++ é orientada a objetos, e seus programas são organizados em classes que contém métodos que realizam tarefas podendo ou não retornar valores, tais recursos podem ser utilizados em forma de objetos, evitando assim a redundância de códigos e proporciona uma simplificação na escrita do programa.

Por ser gratuita e ter código-fonte aberto, a linguagem Java foi escolhida para desenvolvimento de aplicações para a plataforma *Android*, proporcionando um desenvolvimento sem nenhum código interno desse sistema operacional, sendo assim, programadores Java rapidamente dominam a técnica de desenvolvimento utilizando as APIs (interfaces de programação de aplicativos) também conhecidas como bibliotecas de classes Java, que são classes já existentes que podem ser utilizadas no decorrer do código (DEITEL, 2013). Além disso, o *Android* está disponível em dispositivos de vários fabricantes de equipamentos original (OEM – *original equipment manufacters*) em 48 países, por meio de 59 empresas de telefonia. Esse elevado número de empresas OEMs proporciona uma intensa concorrência beneficiando os clientes que buscam por qualidade e melhores preços.

Alguns ambientes de desenvolvimento integrados foram criados para facilitar o trabalho de programadores, nesses ambientes, também chamados de IDEs, a aplicação é desenvolvida de modo eficiente pois proporciona ao programador uma visão geral do código exibindo de forma enumerada cada linha de comando além de expor possíveis erros e sugerir formas corretas de implementação.

Deitel (2013) diz que, para desenvolver aplicações para *Android* de maneira eficiente é necessário uma boa IDE, a instalação do SDK do *Android*, e um plugin de comunicação. O SDK (*Software Development Kit*) do *Android* oferece as ferramentas necessárias para desenvolver, testar e depurar aplicativos, e se comunica com a IDE por meio de plugin específico, a IDE Eclipse, por exemplo, utiliza o Plugin ADT (*Android Development Tools*) para ter acesso aos recursos do SDK.

Por ser desenvolvida para operar em computadores, a IDE Eclipse se tornaria pouco eficiente se cada teste tivesse que ser realizado em um dispositivo com *Android*, porém no SDK do *Android* está incluso uma ferramenta AVD (*Android Virtual Devices*) que permite simulação de um dispositivo com *Android* no computador. Segundo Deitel (2013), antes de executar um aplicativo no emulador é necessário criar um AVD e definir as características do

dispositivo que deseja fazer os testes, incluindo o tamanho da tela em pixels, a densidade de pixels, o tamanho físico da tela, o tamanho do SD para armazenamento dos dados, dentre outras características.

O sistema de posicionamento global (GPS) tem sua origem a partir de um sistema de posicionamento antigo utilizado pela marinha dos Estados Unidos chamado “TRANSIT”, esse sistema utilizava satélites para localizar geograficamente submarinos, misseis e embarcações militares. Com o passar do tempo essa tecnologia foi implantada em diversas áreas, beneficiando toda a população mundial, está presente até mesmo em dispositivos móveis (FALLAS, 2002).

A plataforma *Androi*d, aproveita diversos recursos implantados no dispositivo como placas wireless e GPS. Tais recursos para serem explorados, necessitam de uma interface de programação de aplicativos (API) que são rotinas e padrões de software necessária para utilização de um recurso específico. Conforme Lecheta (2013), existem duas versões de API de Mapas sendo que em 2012 a versão 1 foi descontinuada pela Google e portanto ficou obsoleta, a versão 2 no entanto, continua sendo utilizada para integrar aplicações aos recursos como Google Maps e até mesmo utilizando recursos de localização por GPS .

3 APLICATIVO OEIRASTOUR

O aplicativo OeirasTour é um sistema desenvolvido para orientar o turista na cidade de Oeiras-PI a encontrar o caminho mais curto entre os pontos turísticos. O aplicativo tem como objetivo calcular, a partir da localização do usuário, uma rota para visitar todos os pontos escolhidos, além de mostrar detalhes dos pontos turísticos selecionados, tornando o passeio do usuário mais agradável e interativo.

Nesse capítulo são abordadas a modelagem do sistema implementado, a lógica do aplicativo e a organização do *layout* para que o mesmo se adapte aos demais tipos de tela.

3.1 Modelagem do Sistema

A modelagem do aplicativo foi realizada por meio do diagrama de caso de uso, sendo possível entender o funcionamento geral do sistema por fornecer uma visão de todas as suas funções. O sistema em questão não possui a necessidade de um administrador, pois suas funções são totalmente voltadas ao usuário comum e o diagrama de caso de uso possui apenas um ator "Usuário", que pode efetuar todas as ações disponíveis no sistema, como mostrado na Figura 19.

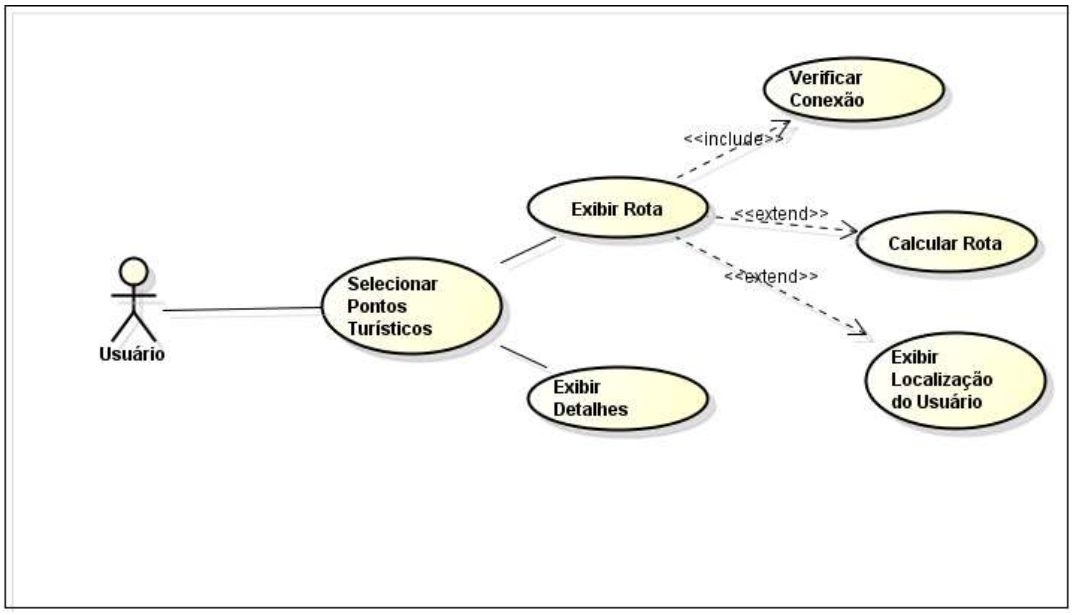


Figura 19 - Diagrama de caso de uso.

A primeira ação é a seleção dos pontos turísticos desejados, que serve de base para realização das demais ações. Em seguida, o usuário pode exibir a rota ou exibir os detalhes dos pontos selecionados. Caso escolha exibir os detalhes, outra ação tem início expondo os detalhes dos pontos selecionados. Caso escolha exibir rota, é iniciada uma ação que possui uma inclusão para verificar a conexão do usuário com a *internet* e com o GPS.A ação de exibir rota também possui mais duas extensões, uma responsável para calcular a rota, marcar o mapa, e direcionar a visão para a cidade, e a outra responsável apenas para direcionar a visão localização do usuário no mapa.

O diagrama de classe apresentado na Figura 20 mostra as informações que o sistema necessita, expostas em atributos e métodos pertencentes às classes, e a forma como elas estão relacionam entre si.

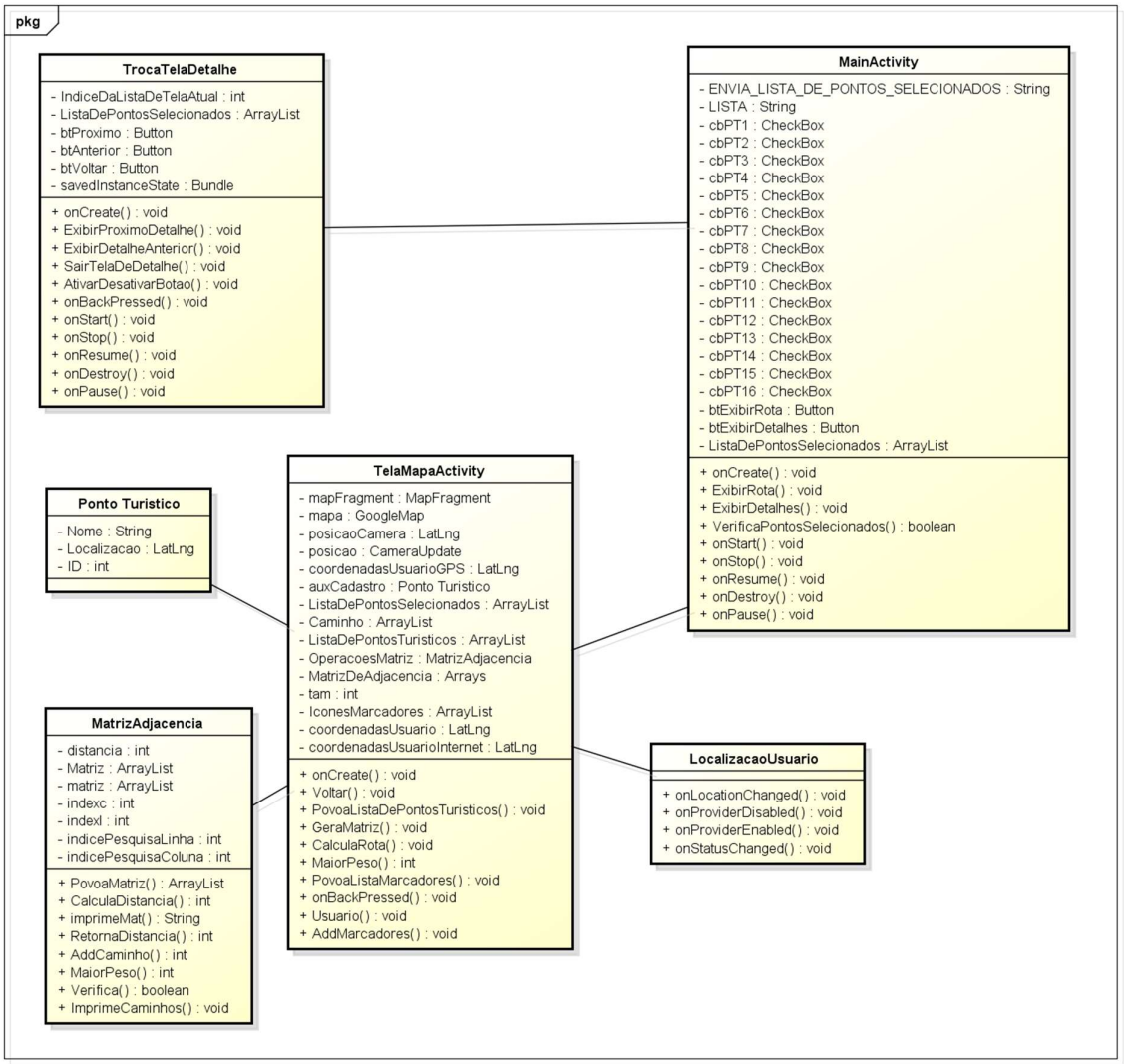


Figura 20 - Diagrama de classe.

A classe "*Main Activity*" é a classe principal da aplicação. Por meio dessa classe é possível realizar a primeira ação do diagrama de caso de uso apresentado na Figura 19, fornecendo base para as ações seguintes. Após selecionar os pontos turísticos desejados a classe "TocaTelaDetalhe" é a responsável para fazer a transição entre os layouts das telas dos pontos escolhidos, e a classe "TelaMapaActivity" fica responsável para a exibição do mapa na tela, calcular rota, adicionar marcadores no mapa, e direcionar a visão para o usuário ou para o centro do conjunto de pontos turísticos. Esta classe possui dois objetos um para representar um ponto turístico e o outro representa uma matriz de adjacência com todas as operações necessárias para cálculo da rota. Esta classe também possui uma relação com outra classe "LocalizacaoUsuario" que é responsável por retornar as coordenadas do dispositivo móvel podendo ser um retorno pela internet ou por GPS.

3.2 Lógica do Aplicativo

Em *Android* quando se pretende realizar uma atividade é necessário criar uma *Activity,* representada por uma classe java identificada como *activity* no *AndroidManifest* da aplicação. O *AndroidManifest* de uma aplicação é um recurso de extrema importância pois nele é dado as definições das atividades do sistema, permissões para a utilização de alguns recursos do dispositivo como por exemplo, a conexão com a *internet*, o uso do GPS, ou o uso da câmera do aparelho. A Figura 21 apresenta o ciclo de vida de uma *activity* para facilitar o entendimento da forma como esse recurso funciona.

Assim que uma *activity* é iniciada é executado seu método *onCreate.* A partir disso, o ciclo acontece como exposto na Figura 21. Uma *activity* pode sofrer algum fator que influencie em seu estado atual, por exemplo, uma ligação recebida pode por uma *activity* em espera chamando seu método *onStop*. A navegação entre as telas de uma aplicação pode por uma *activity* em pausa chamando o método *onPause*. Ao executar os métodos *onResume* e *onStart* pode ser reestabelecida a comunicação com a activity, seguindo o ciclo da Figura 21 até a *activity s*er destruída.

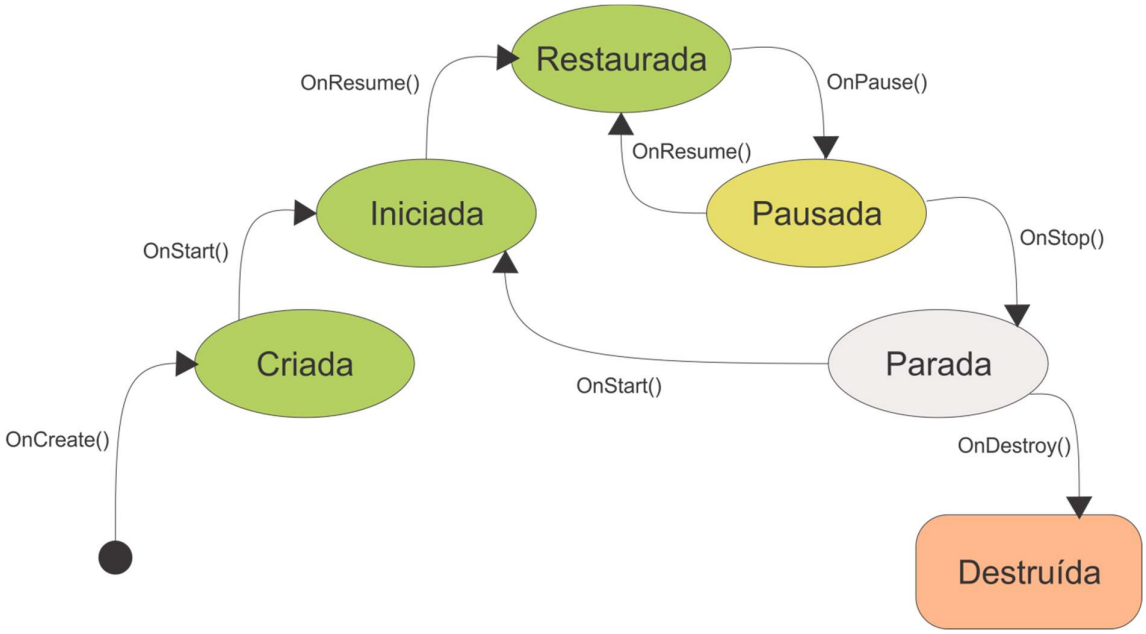


Figura 21 - Ciclo de vida de uma *activity* (developer.android.com)

A aplicação em questão foi criada com três atividades, uma para cada tela, e cada uma possui um objetivo individual. Na *activity* da tela inicial, além de exibir o *layout* inicial, possui uma função para preencher uma lista de inteiros com os valores dos identificadores de cada tela de detalhe. Sabendo que foi criada uma tela para cada ponto turístico, esses identificadores servem também para identificar um ponto turístico específico e são encontrados na classe "R", uma classe gerada e implementada automaticamente que possibilita a interação do código java com os demais recursos da aplicação.

Ao clicar no botão para exibir a rota na tela inicial, a função de preencher a lista com os valores selecionados é executada. Caso nenhum ponto turístico for selecionado, uma mensagem é exibida ao usuário solicitando pelo menos uma seleção. Caso haja seleção, uma mensagem é exibida informando a necessidade de uma conexão com a *internet* e da ativação do GPS. Em seguida a atividade atual será destruída e a próxima atividade é iniciada tendo como parâmetro a lista de valores de identificadores assim a nova atividade possui conhecimento de quais pontos foram selecionados.

Ao clicar no botão para exibir os detalhes dos pontos turísticos, a função para preencher a lista também é executada, porém não precisa de uma mensagem informando a necessidade de conexão com a internet e ativação do GPS. Caso haja seleção a atividade atual é destruída e se inicia uma nova atividade tendo como parâmetros a lista de identificadores para tomar conhecimento dos pontos selecionados.

Quando a *activity* responsável por mostrar o mapa na tela é iniciada, além de exibir o *layout* da tela ela gera uma matriz de adjacência a partir dos pontos turísticos recebidos por parâmetros, e realiza o cálculo da rota utilizando essa matriz. Para gerar a matriz de adjacência um objeto contendo o grafo dos pontos turísticos foi criado para poder resgatar as distâncias de um ponto a outro. Esse objeto também conta com um método que verifica a distância entre coordenadas geográficas para poder medir a distância entre a localização do usuário e dos pontos turísticos.

A tela de exibição do mapa contém três botões, um para retornar a tela anterior destruindo a *activity* atual e iniciando a *activity* da tela principal, um botão para exibir a localização do usuário direcionando o mapa para as coordenadas obtidas por GPS ou pela *internet* e adiciona um marcador, e um botão para cálculo de rota que preenche uma lista de objetos do tipo ponto turístico, que contém uma variável para nome e uma variável para latitude e longitude, calcula a rota e adiciona os marcadores certos em ordem para serem visitados. A lista de pontos turísticos é preenchida a partir da lista de identificadores recebida por parâmetro e é utilizada para calcular a rota e adicionar os marcadores no mapa.

Para ser calculada a rota é necessária uma matriz de adjacência que inclua a localização do usuário como um nó do grafo, para isso é preenchida a matriz apenas relacionando os pontos turísticos desejados com base na matriz de adjacência geral disponível no APÊNDICE A. Seguidamente, o restante da matriz é preenchido utilizando o método para medir distâncias entre coordenadas. Gerada a matriz de adjacência, aplica-se a ela o algoritmo de calcular rota. Esse algoritmo retorna uma lista onde os elementos são identificadores dos pontos turísticos e estão organizados de modo que a primeira posição seja o primeiro ponto a ser visitado e a segunda posição seja o segundo ponto a ser visitado e assim por diante.

O algoritmo desenvolvido para encontrar o menor caminho foi baseado no algoritmo de Bellmore e Nemhauser que resolve a heurística do “Vizinho mais próximo”. Consiste em delimitar um caminho que resolva o PCV analisando o peso das arestas de um grafo. Partindo de um ponto inicial, o algoritmo verifica qual vértice possui o menor custo para ser alcançado e o adiciona em uma lista que representa o caminho. Em seguida, o vértice inicial passa a ser o vértice adicionado na lista e é feita novamente verificação do vértice com menor custo para ser alcançado e realizando as demais ações até o fim do grafo, lembrando que um vértice só pode estar contido apenas uma vez na lista de caminhos. Como no caso do aplicativo a localização do usuário é o último vértice adicionado no grafo, o ponto inicial é o justamente o último elemento da matriz de adjacência, assim como exemplificado na Figura 22.

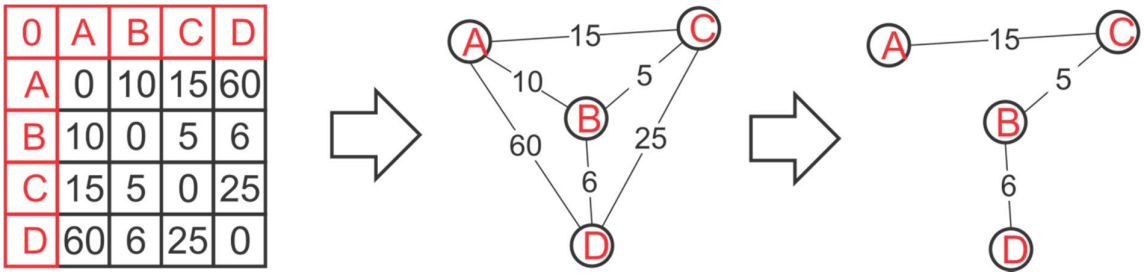


Figura 22 - Execução do algoritmo de Bellmore e Nemhauser

Na matriz de adjacência exposta na Figura 22, toda a linha zero e toda a coluna zero em destaque representam os vértices do grafo. A execução do algoritmo é iniciada partindo do vértice D que seria a localização do usuário e o último vértice adicionado, é feita a verificação dos valores de cada linha da coluna D. Em seguida, o vértice que fez a menor ligação é adicionado na lista de caminhos e é definido como ponto de partida. Observe que, o vértice B é o que possui menor peso de ligação com o vértice D, desse modo a lista que representa o caminho estará preenchida com os vértices {D, B}. Como o custo zero representa a ausência de aresta, o vértice mais próximo de B seria o vértice C com custo da aresta igual a 5, assim C é adicionado a lista de caminho e é feita novamente verificação da menor aresta. É notado que a menor aresta seria a aresta CB com custo 5, porém B já está contido na lista de caminho. Nesse caso o vértice adicionado é o vértice A por possuir o segundo menor peso dentre as arestas de C e a lista de caminhos estará completa seguindo a ordem {D, B, C, A}. Como todos os vértices estão incluídos na lista de caminhos, a execução do algoritmo é encerrada e a lista será retornada. No aplicativo, os vértices do grafo são os pontos turísticos além do ponto inicial que é o ponto de localização do usuário.

Pode-se observar na Figura 23 que o aplicativo utiliza um GPS para localização do usuário e dos pontos turísticos. A Figura 23(a) apresenta um mapa de marcação da localização do usuário e a Figura 23(b) mostra um mapa de localização dos pontos turísticos escolhidos pelo usuário. Com a lista de caminhos definida é necessário adicionar os marcadores. Para isso é feita uma limpeza no mapa excluindo quaisquer marcadores anteriores e adicionando os novos. Isso é feito para não sobrepor os marcadores economizando memória. Um ícone de marcação especial foi criado para marcar o usuário e, os ícones de marcação dos pontos turísticos são numerados para mostrar a ordem que se deve seguir e quando clicados exibem o nome do ponto turístico, eles são incluídos seguindo a mesma ordem da lista de caminho conforme mostrado na Figura 23.

(a) (b)

Figura 23 - (a) Mapa com marcador de usuário. (b) Mapa com marcadores de pontos turísticos.

Para a exibição dos detalhes dos pontos turísticos foi criada uma *activity* que recebe por parâmetro a lista de inteiros que representa os identificadores das telas de *layout* dos pontos turísticos. Por receber esse parâmetro, a primeira tela que é exibida corresponde ao primeiro *layout* da lista, caso só se tenha feito seleção de um único ponto turístico, então os botões "Próximo" e "Anterior" serão desabilitados, caso contrário ao clicar em "Proximo" um contador é incrementado com mais 1, e em seguida a *activity* é destruída e novamente criada só que a tela que será exibida é a correspondente ao contador na lista de *layouts*, do mesmo modo acontece caso seja clicado no botão "Anterior" porém, o contador será decrementado em menos 1. Desse modo não se acumula exibição de *layout*, economizando memória durante a navegação nas telas de detalhes dos pontos turísticos.

3.3 Organização do *Layout* do Aplicativo

Atualmente existem milhares de dispositivos móveis com sistema operacional *Android*, por conta disso, existem também vários tipos de telas com tamanho e resoluções diferentes tornando mais importante a forma como foi criado o *layout* da aplicação de modo que o mesmo se adapte aos diferentes dispositivos.

A tela inicial foi desenvolvida a partir de um *Linear Layout*, permitindo que todos os recursos de *layout* da página sejam dispostos um abaixo do outro, ou um ao lado do outro, dependendo da orientação definida. A tela inicial contém todos os pontos disponíveis para cálculo de rota e para exibição dos detalhes descritos cada um com um *Check Box* dentro de um *Scroll View e um Horizontal Scroll View.* O *Check Box* permite que o usuário marque ou desmarque um ponto turístico. O fato de todos os pontos turísticos estarem dentro de dois *Scrolls* é dada para listagem, tendo em vista que o aplicativo contém dezesseis pontos fica inviável exibir todos os *Check Box* na tela de uma só vez sendo necessário que seja habilitada a função de rolagem utilizando o *Scroll View*.

Em alguns casos, o nome do ponto turístico pode ser grande demais para o comprimento da tela, não sendo exibido completamente, porém um *Horizontal Scroll View* permite rolagem para os lados permitindo assim uma visualização completa. Além disso, foi observado que mesmo após a utilização dos recursos de rolagem vertical e rolagem horizontal proporcionada pelo *Scroll View* e pelo *Horizontal Scroll View* respectivamente, os botões presentes no fim da tela permanecem imóveis. Isso ocorre pela forma como o *layout* foi organizado e possibilita ao usuário calcular a rota ou exibir o detalhe dos pontos selecionados de forma mais eficiente.

Quadro 3 - Organização do layout da tela inicial.

Organização dos recursos de layout da tela inicial do aplicativo.

```
1  <Linear Layout>
2     <Text View/>
3     <Scroll View>
4        <Linear Layout>
5           <Horizontal Scroll View>
6              <Linear Layout>
7                 <Check Box></Check Box>
8                 <Check Box></Check Box> ...
9              <Linear Layout>
10          </Horizontal Scroll View>
11       </Linear Layout>
12    </Scroll View>
13    <Button/>
14    <Button/>
15 </Linear Layout>
```

O Quadro 3 mostra a forma como foram organizados os recursos de *layout* presentes na tela inicial do aplicativo, onde fica claro que os botões estão fora do *Scroll View* e fora do *Horizontal Scroll View*. Vale ressaltar que após a declaração de um *Scroll View* ou de um *Horizontal Scroll View*, foi necessário definir outro tipo de *layout* para tornar possível a inclusão de novos elementos, no caso exposto pelo Quadro 3, o *layout* definido foi o *Linear Layout* para organizar os elementos de forma vertical, exibindo um logo abaixo do outro. É possível observar na linha 2 a utilização de um recurso chamado *Text View* que tem como função exibir na tela um texto qualquer, no caso da aplicação o texto possui o seguinte valor, "Selecione um ponto turístico". Na linha 8, após a declaração do *Check Box* foi utilizado reticências para simbolizar a existência de mais desses recursos tendo em vista que foi criado um para cada ponto turístico totalizando dezesseis *Check Box*.

Todos os recursos de *layout* necessitam de atributos que definem sua aparência e a forma como são referenciados dentro da aplicação. Para o *layout* da tela inicial foi importante definir a localização de todos os componentes como, por exemplo, os botões, que mesmo com a variação no tamanho das telas eles possuem sempre altura que suporta o texto de dentro, e um comprimento que sempre é igual ao comprimento da tela. Para isso o atributo altura foi definido como "*wrap_content*" tornando a altura ideal para conter os elementos existentes dentro do botão e o comprimento foi definido como "*fill_parent*" permitindo que se estenda por todo o espaço livre da tela.

Os atributos do *Scroll View* são necessários para que o mesmo não venha a sobrepor os botões localizados ao fim da tela. Para isso o atributo peso foi modificado para um valor menor do que o peso dos botões, desse modo, o *Scroll View* inicia após o *Text View*, como propõe o *Linear Layout* vertical, então o *Scroll View* irá o máximo de altura até que ainda seja possível exibir os dois botões, fazendo sempre os botões se localizarem no final da tela, a menos que a tela seja grande o suficiente para exibir todos os dezesseis pontos turísticos e ainda sobrar espaço. Nesse caso, os botões ficariam ao fim do *Scroll View* e todo o espaço abaixo deles seria preenchido pelo layout base da tela inicial, um *Linear Layout* que teve altura e comprimento definidos como "*fill_parent*" preenchendo todo o espaço disponível na tela.

Outro *layout* desenvolvido para o aplicativo foi o responsável pela exibição da tela que contém o mapa para exibição da rota. Esse *layout* é composto por três botões e um *Fragment*, componente utilizado para exibição do mapa, os botões estão organizados um ao lado do outro na parte superior da tela dentro de um *Linear Layout* horizontal, fora desse *Linear Layout* encontra-se o *Fragment* ocupando todo o restante da tela. Todo esse conjunto

está contido dentro de um *Linear Layout* vertical, que organiza os componentes de forma vertical e ocupa todo o espaço disponível na tela do dispositivo.

Para se adequar aos diferentes tipos de tela, os botões superiores foram definidos com altura necessária para mostrar o conteúdo interno e comprimento que ocupa todo o espaço livre da tela. Para um botão não sobrepor o outro, o atributo peso foi definido com o mesmo valor para todos.

O recurso *Fragment*, responsável por exibir o mapa, foi declarado com altura máxima permitida, e comprimento máximo permitido, não sobrepondo os botões, pois a forma como o *layout* se organiza define que o *Fragment* se inicie após o *Linear Layout* que contém os botões assim como exposto no Quadro 4.

Quadro 4. Organização do layout da tela do mapa.

Organização dos recursos de layout do mapa do aplicativo.

```
1  <Linear Layout>
2      <Linear Layout>
3          <Button/>
4          <Button/>
5      </Linear Layout>
6      <Fragmente/>
7  </Linear Layout>
```

Para atender a outra função do aplicativo foi desenvolvido um modelo de *layout* específico para cada ponto turístico com o objetivo de exibir os detalhes dos pontos selecionados pelo usuário. Esse novo modelo de *layout* consiste em um sistema que expõe imagens do ponto turístico, bem como um texto que descreve o local.

No desenvolvimento do *layout* da tela de detalhes foi utilizado um *Scroll View* dentro de um *Linear Layout* vertical, e dentro desse *Scroll View* existe um *Horizontal Scroll View* contendo os *Image View*, componente responsável para exibição das imagens do ponto turístico, sendo necessário um para cada imagem. Após o *Horizontal Scroll View* existe um *Text View* para mostrar o texto detalhando o ponto turístico e os botões de navegação e visualizar os detalhes do ponto seguinte ou do ponto anterior além de poder retornar a tela inicial.

A organização desses *layouts* em alguns dispositivos com telas maiores, os botões podem ser vistos mesmo sem o usuário realizar a rolagem vertical, dependendo apenas do tamanho do texto que descreve o ponto turístico. Em telas de menor tamanho, a necessidade

de rolagem é maior, tendo em vista que possui menos espaço para exibição do texto. Diferente da tela inicial, a tela de detalhes não possui os botões imóveis, para conseguir isso o *layout* deve estar organizado de modo diferente assim como apresentado no Quadro 5.

Quadro 5. Organização dos recursos de layout da tela de detalhes.

Organização dos recursos de layout da tela de detalhes dos pontos turísticos.

```
1  <Linear Layout>
2     <Scroll View>
3        <Linear Layout>
4           <Text View/>
5           <Horizontal Scroll View/>
6              <Linear Layout>
7                 <Image View/>
8                 <Image View/>
9              </Linear Layout>
10          </Horizontal Scroll View>
11          <Text View/>
12          <Linear Layout>
13             <Button/>
14             <Button/>
15          </Linear Layout>
16          <Button/>
17       </Linear Layout>
18    </Scroll View>
19 </Linear Layout/>
```

No Quadro 5 na linha 12 foi criado um *Linear Layout* para organizar os dois botões de maneira horizontal definindo mesmo valor de atributo peso nos botões, desse modo eles ocupam o mesmo espaço em tela. No próximo capítulo, são apresentadas as telas do aplicativo como resultado do trabalho.

4 RESULTADOS E DISCUSSÕES

A tela inicial do aplicativo OeirasTour contém dois botões, um para calcular a rota e o outro para exibir os detalhes, ambos encontram-se ao fim da tela e estão fora dos *Scrolls* permitindo assim uma visualização constante. Todos os recursos de layout da tela inicial são facilmente observados na Figura 24 que exibe capturas de tela de um dispositivo com tela de 2.7 polegadas.

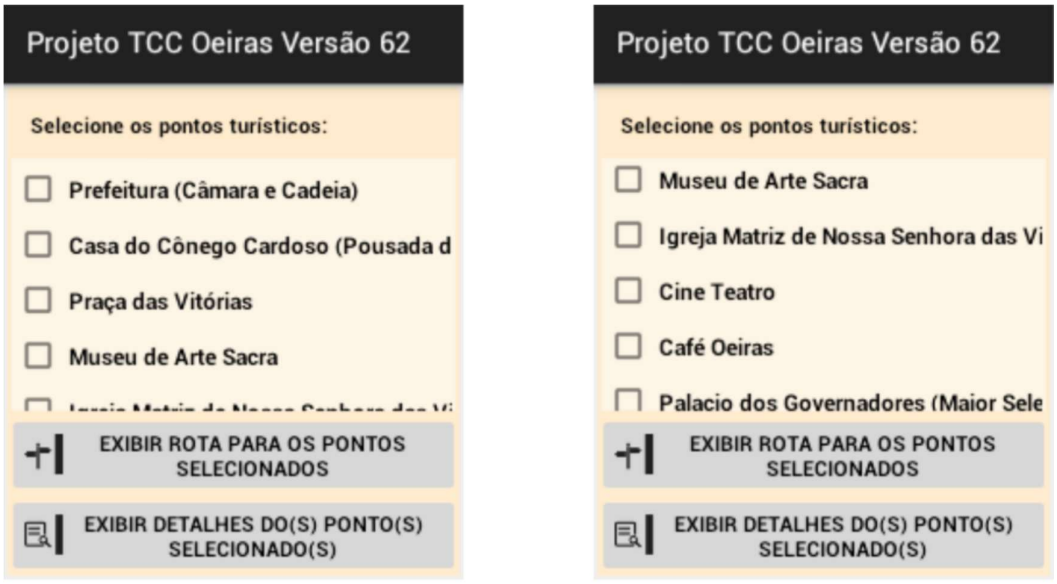


Figura 24 - Capturas de telas do aplicativo em dispositivos com 2.7 polegadas.

Na Figura 24 são expostas capturas de telas feitas em um dispositivo com tela de 2.7 polegadas, nota-se que a captura da Figura 24(a) mostra apenas quatro *Check Box* diferentes em um total de dezesseis *Check Box* sendo um para cada ponto turístico. A exibição de apenas quatro pontos se dá pelo fato da tela ser relativamente pequena, para permitir o acesso aos demais *Check Box* do layout, então foi habilitada a função de "rolagem" utilizando um *Scroll View*. Pode–se notar que na Figura 24(a), o "Museu de Arte Sacra" é o último ponto turístico representado na tela e na Figura 24(b) esse ponto turístico é exibido em primeiro e segue com os demais pontos disponíveis, representando assim que foi utilizada a função de "rolagem" presente na tela inicial. Ainda pode ser observado que mesmo rolando a tela para exibição dos demais pontos turísticos, os botões abaixo ficam imóveis.

Dentre todos os recursos de *layout* apresentados na tela inicial do aplicativo, o recurso *Horizontal Scroll View* soluciona o problema que ocorre em telas pequenas, o nome de algum ponto turístico não fica totalmente visível para o usuário, como é o caso do ponto "Casa do Cônego Cardoso (Pousada do Cônego)", havendo necessidade também de uma rolagem horizontal conforme apresentado na Figura 25.

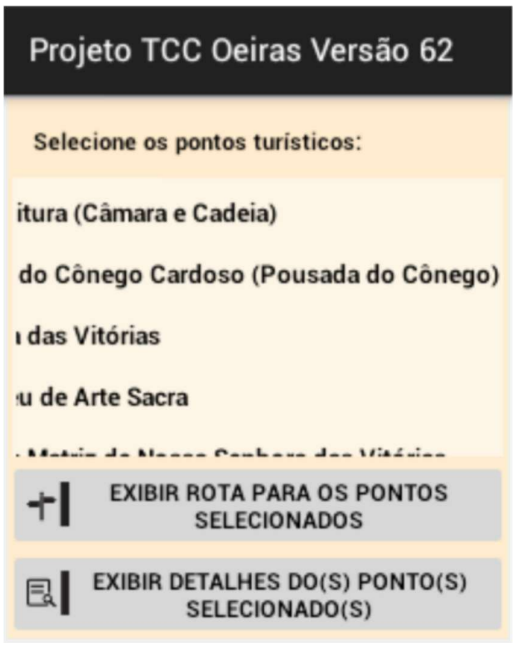


Figura 25 - Tela inicial do aplicativo após rolagem horizontal.

A Figura 25 mostra a utilização do *Horizontal Scroll View* na tela inicial do aplicativo, podendo então visualizar a continuação do nome do ponto turístico após uma rolagem horizontal. Desta forma é possível saber o nome completo do ponto turístico por mais que a tela do dispositivo seja pequena. A utilização desse recurso permite uma listagem dos pontos sem haver quebra de linha em seus nomes.

Na Figura 26 é exibida a tela inicial do aplicativo em um dispositivo com tela de 4.7 polegadas, sendo possível observar que em telas relativamente grandes é exibida maior quantidade de pontos turísticos e em alguns casos a rolagem horizontal não é necessária.

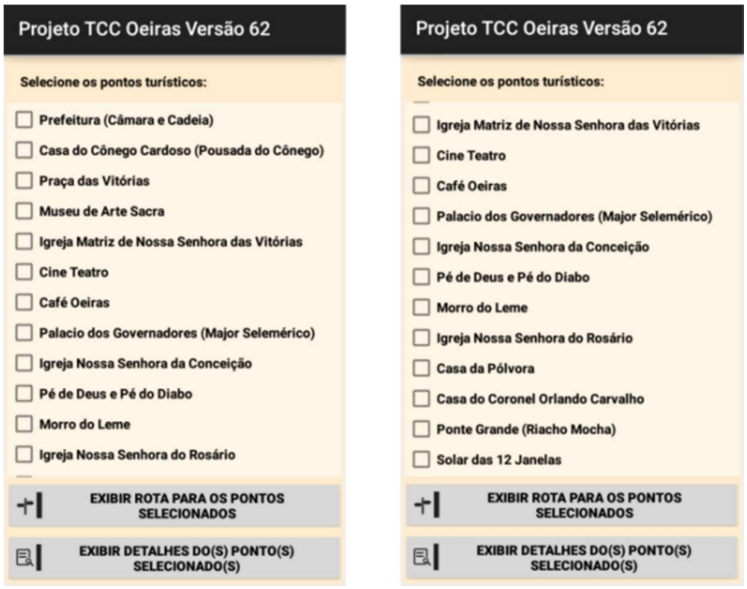


Figura 26 - Tela inicial em dispositivo com tela de 4.7 polegadas.

A tela do mapa da aplicação com imagem capturada de um dispositivo móvel de tela de 2.7 polegadas é exibida na Figura 27. Na parte superior apresenta botões de voltar, calcular rota e de localizar o usuário.

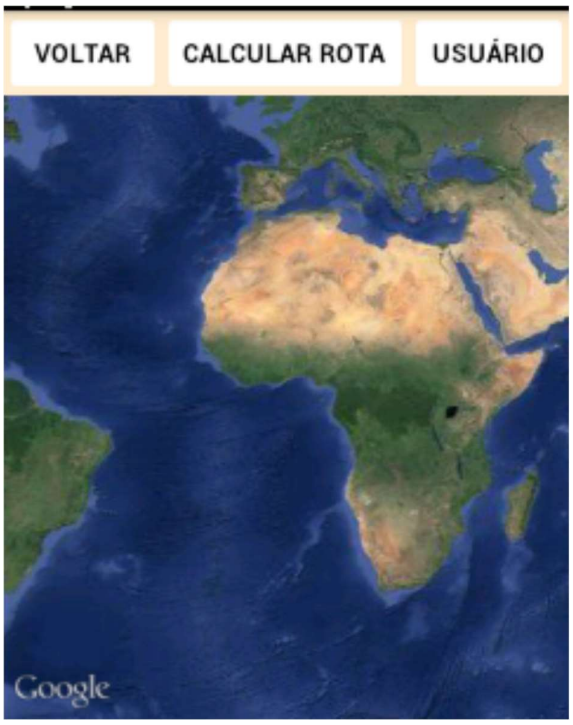


Figura 27 - Tela para exibição do mapa em dispositivo com 2.7 polegadas.

Na Figura 28, é possível observar como os pontos turísticos estão dispostos em tela. Na parte superior, está o nome do ponto turístico seguido da foto e sua descrição. A imagem à direita representa apenas a continuação da imagem à esquerda e nela existe um destaque em dois botões, “Anterior” e “Próximo”, mostrando que estão desabilitados porque não foi selecionado mais de um ponto turístico.

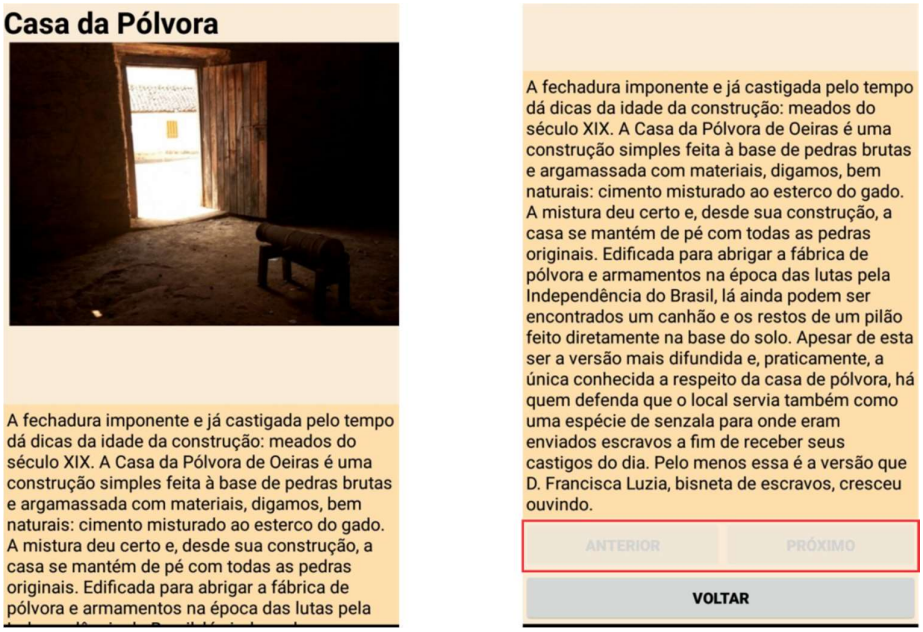


Figura 28 - Tela de detalhes de um ponto turístico por um dispositivo de 4.5 polegadas de tela

Desse modo não existe uma próxima tela ou uma tela anterior, caso contrário, os botões de navegação estariam habilitados assim como o botão “Voltar” presente ao fim da tela. No caso do *layout* dessa tela, foi utilizado um *Horizontal Scroll View* para expor as imagens, desse modo, o usuário consegue ver todas as imagens disponíveis rolando a tela horizontalmente e ao rolar a tela verticalmente poderá ler o texto que descreve ponto turístico, essa função pode ser facilmente observada na Figura 28.

Na Figura 29, é possível observar que ao rolar a tela verticalmente o restante do texto e os botões são exibidos e, ao efetuar a rolagem horizontal apenas na parte superior da tela onde se encontra a imagem do ponto turístico, é possível visualizar as demais fotos do ponto em questão.

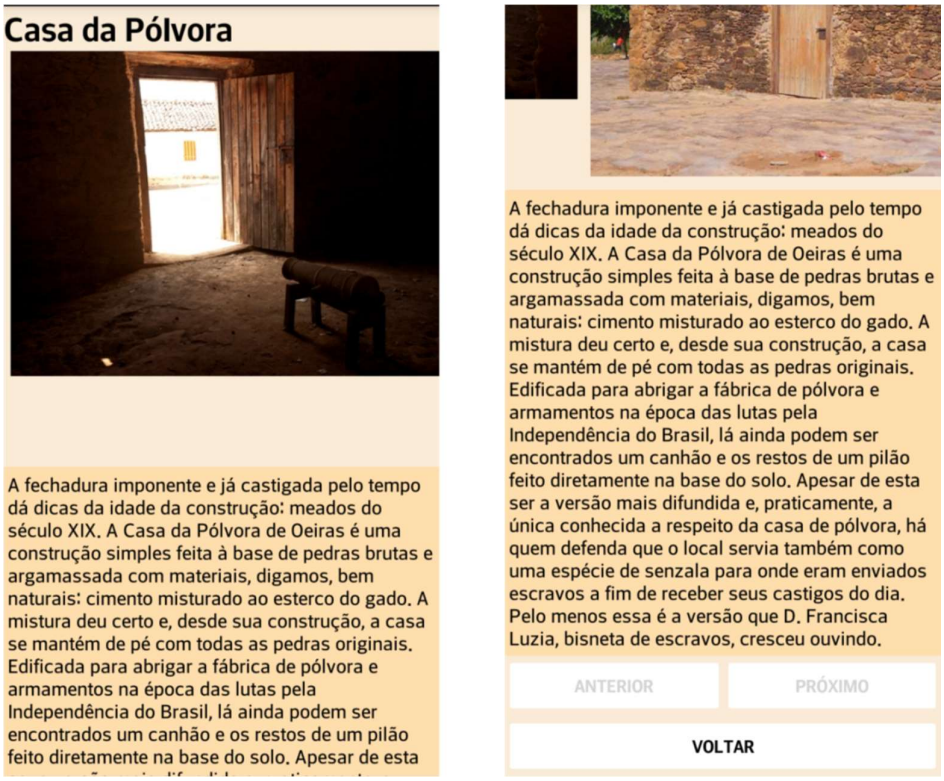


Figura 29 - Tela de detalhes obtida por dispositivo com 4.7 polegadas de tela.

O aplicativo foi testado com alguns dispositivos móveis de tamanhos variados para averiguação do comportamento do *layout* em diferentes resoluções. Dispositivos com 2.7, 4.5, e 4.7 polegadas de tela, foram utilizados para realização dos testes e apresentaram resultados satisfatórios, tendo em vista que o *layout* se adaptou aos diferentes dispositivos.

5 CONSIDERAÇÕES FINAIS

O objetivo do trabalho é apresentar um aplicativo que calcula uma rota para percorrer os principais pontos turísticos da cidade de Oeiras de maneira eficiente. Além disso, o aplicativo fornece informações detalhadas sobre os pontos que forem selecionados pelo usuário do sistema. Esse aplicativo foi desenvolvido para a plataforma *Android* voltado a dispositivos móveis, permitindo ao usuário utilizá-lo de qualquer local, levando em consideração que haja conexão com a *internet*, seja possível estabelecer uma comunicação com o GPS.

No decorrer do desenvolvimento do aplicativo, foi feita uma análise da necessidade e dos benefícios que o mesmo traria a toda comunidade interessada em seus serviços propostos, além de serem testadas diversas formas de se calcular a rota partindo de um ponto específico em uma matriz de adjacência, utilizando o algoritmo do menor caminho.

Os resultados dos testes realizados em todos os diferentes tamanhos de dispositivos móveis foram satisfatórios, pois os *layouts* das telas foram desenvolvidos para serem responsivos.

Como trabalhos futuros é sugerida, a utilização de heurísticas diferentes para cálculo da rota, implementação da mesma ideia em um aplicativo voltado para áreas diferentes como saúde ou educação, um estudo mais abrangente em complexidade de algoritmos para dispositivos móveis tendo em vista as limitações de processamento dos dispositivos atuais.

6 REFERÊNCIAS BIBLIOGRÁFICAS

BOAVENTURA NETTO, P. O. **Grafos:** Teorias, Modelos, Algoritmos. 4. Ed. São Paulo: E. Blucher, 2006.

BOSCO – MONTEIRO, J. **Google *Android*:** Crie aplicações para celulares e *tablets*. São Paulo: Casa do Código, 2013.

BULAT, M. **Isomorfismo de grafo y de funciones lógicas con algunas aplicaciones**. Revista de Matemática: Teoría y Aplicaciones 1998 5(2) : 87–112, 2004

BURIOL, L. S. *et al*. **A hibrid genetic algorithm for the weight setting problem in ospf/is-is routing**. Networks, N.Y. Print, New York, v. 46, n. 1, p. 3656, 2005.

DEITEL, P.; DEITEL, A.; DEITEL, H.; MORGANO, M. ***Android* para Programadores:** Uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2013.

DEITEL, H. M.; DEITEL, P. J. **Java**: Como Programar. 8. Ed. São Paulo: Pearson Prentice Hall, 2010.

FALLAS, J. **Sistema de posicionamiento global**. 2002. Universidad Nacional. Heredia. Costa Rica.

GOLDBARG, M.; GOLDBARG, E. **Grafos:** Conceitos, Algoritmos, Aplicações. Rio de Janeiro: Elsevier, 2012.

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória Programação Linear***: Modelos e Algoritmos*. 2. Ed. Rio de Janeiro: Elsevier, 2005.

LLUCH; Cristina e SANABRIA-CODESAL; Esther. **Grafos hamiltonianos en el disseño de viajes**. Modelling in Science Education and Learning Volumen 6(2), No. 11, 2013.

RECUERO; A. **Aplicaciones de la Teoría de Grafos:** Búsqueda de Caminhos em uma red y Análisis de su Conectividad. Informes de la Construcción, Vol. 46, n.° 433, 1994.

SANCHEZ, G. T.; LOZANO, V. M. **Algoritmo de Dijkstra**: Un Tutorial Interactivo. 2001. Universitat de les Illes Balears. Disponível em <http://www.uib.es/ca/>. acesso em 22 jul 2014.

APÊNDICES

APÊNDICE A

Matriz de adjacência geral entre os pontos turísticos.

MATRIZ

0	cbPT1	cbPT2	cbPT3	cbPT4	cbPT5	cbPT6	cbPT7	cbPT8	cbPT9	cbPT10	cbPT11	cbPT12	cbPT13	cbPT14	cbPT15	cbPT16
cbPT1	**0**	**17**	**8**	**120**	**70**	**160**	**260**	**270**	**650**	**720**	**1630**	**700**	**850**	**730**	**350**	**155**
cbPT2	**17**	**0**	**11**	**140**	**80**	**180**	**240**	**250**	**670**	**730**	**1640**	**690**	**840**	**745**	**340**	**145**
cbPT3	**8**	**11**	**0**	**81**	**12**	**56**	**89**	**180**	**630**	**550**	**1620**	**620**	**770**	**705**	**355**	**4**
cbPT4	**120**	**140**	**81**	**0**	**32**	**170**	**265**	**400**	**500**	**700**	**1575**	**780**	**800**	**695**	**450**	**190**
cbPT5	**70**	**80**	**12**	**32**	**0**	**23**	**180**	**390**	**530**	**670**	**1600**	**740**	**760**	**699**	**445**	**150**
cbPT6	**160**	**180**	**56**	**170**	**23**	**0**	**190**	**399**	**600**	**660**	**1605**	**750**	**735**	**740**	**500**	**140**
cbPT7	**260**	**240**	**89**	**265**	**180**	**190**	**0**	**350**	**630**	**690**	**1655**	**730**	**725**	**770**	**560**	**130**
cbPT8	**270**	**250**	**180**	**400**	**390**	**399**	**350**	**0**	**900**	**800**	**1990**	**550**	**950**	**940**	**350**	**250**
cbPT9	**650**	**670**	**630**	**500**	**530**	**600**	**630**	**900**	**0**	**1050**	**200**	**1400**	**1260**	**42**	**650**	**550**
cbPT10	**720**	**730**	**550**	**700**	**670**	**660**	**690**	**800**	**1050**	**0**	**1790**	**400**	**120**	**850**	**1000**	**600**
cbPT11	**1630**	**1640**	**1620**	**1575**	**1600**	**1605**	**1655**	**1990**	**2000**	**1790**	**0**	**1999**	**1990**	**2000**	**1900**	**1700**
cbPT12	**700**	**690**	**620**	**780**	**740**	**750**	**730**	**550**	**1400**	**400**	**1999**	**0**	**400**	**840**	**850**	**600**
cbPT13	**850**	**840**	**770**	**800**	**760**	**735**	**725**	**950**	**1260**	**120**	**1990**	**400**	**0**	**1300**	**1100**	**700**
cbPT14	**730**	**745**	**705**	**695**	**699**	**740**	**770**	**940**	**42**	**850**	**2000**	**ior,1300**	**1300**	**0**	**850**	**760**
cbPT15	**350**	**340**	**355**	**450**	**445**	**500**	**560**	**350**	**650**	**1000**	**1900**	**850**	**1100**	**850**	**0**	**455**
cbPT16	**155**	**145**	**4**	**190**	**150**	**140**	**130**	**250**	**550**	**600**	**1700**	**600**	**700**	**760**	**445**	**0**

TERMO DE AUTORIZAÇÃO PARA PUBLICAÇÃO DIGITAL NA BIBLIOTECA "JOSÉ ALBANO DE MACEDO"

Identificação do Tipo de Documento

( ) Tese

( ) Dissertação

(x) Monografia

( ) Artigo

Eu, Laurentino de Moura Sousa Júnior., autorizo com base na Lei Federal nº 9.610 de 19 de Fevereiro de 1998 e na Lei nº 10.973 de 02 de dezembro de 2004, a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação Aplicativo de roteamento do menor caminho dos pontos turísticos de Oeiras para dispositivos móveis. de minha autoria, em formato PDF, para fins de leitura e/ou impressão, pela internet a título de divulgação da produção científica gerada pela Universidade.

Picos-PI 28 de Abril de 2016.

Laurentino de M. S. Júnior.
Assinatura

Assinatura