CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U. nº 198, de 14/10/2016
AELBRA EDUCAÇÃO SUPERIOR - GRADUAÇÃO E PÓS-GRADUAÇÃO S.A.

Lucas Costa da Silva

DESENVOLVIMENTO DE UM APLICATIVO MOBILE PARA AUXÍLIO À LOCALIZAÇÃO NO AMBIENTE INTERNO DO CEULP/ULBRA

Palmas – TO

2020

Lucas Costa da Silva

DESENVOLVIMENTO DE UM APLICATIVO MOBILE PARA AUXÍLIO À LOCALIZAÇÃO NO AMBIENTE INTERNO DO CEULP/ULBRA

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. Esp. Fábio Castro Araújo

Palmas – TO

2020

Lucas Costa da Silva

DESENVOLVIMENTO DE UM APLICATIVO MOBILE PARA AUXÍLIO À LOCALIZAÇÃO NO AMBIENTE INTERNO DO CEULP/ULBRA

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. Esp. Fábio Castro Araújo

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Esp. Fábio Castro Araújo

Orientador

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Jackson Gomes de Souza

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Madianita Bogo Marioti

Centro Universitário Luterano de Palmas – CEULP

Palmas – TO

2020

Aos meus pais, professores, amigos e colegas de curso e a todos aqueles que de certa forma contribuíram para a realização deste trabalho.

AGRADECIMENTOS

Primeiramente agradeço a Deus pelo dom da vida e por ter me guiado ao longo desse percurso.

A minha família, pelo apoio e incentivos que me concederam em todos os momentos, especialmente aos meus pais, Jeorge e Maria e irmã Eliana por todo apoio, paciência e compreensão.

Ao meu orientador, Fábio Castro, pelo seu apoio, incentivos nos momentos de dificuldade e por todo conhecimento compartilhado durante o processo de pesquisa e desenvolvimento deste trabalho.

A todos os colegas e amigos de curso, especialmente ao amigo Thiago Aparecido que me acompanhou durante essa jornada.

Agradeço também, a todos os professores pelos ensinamentos e valores passados a mim durante o curso.

Enfim, a todos que fizeram parte dessa grande etapa em minha vida.

RESUMO

SILVA, Lucas Costa. **Desenvolvimento de um aplicativo mobile para auxílio à localização no ambiente interno do CEULP/ULBRA**. 2020. 39 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas/TO, 2020.

Ferramentas de mobilidade urbana estão cada vez mais presentes no cotidiano das pessoas, facilitam o deslocamento de pedestres e veículos em lugares desconhecidos apenas com a utilização de um dispositivo móvel. Os mapas digitais são projetados principalmente para perímetros urbanos e rodovias, ajudam os usuários a encontrarem um percurso otimizado até um determinado destino, seja no trânsito, ciclismo, caminhadas e em outras atividades ao ar livre. Apesar de existir certa facilidade na exploração de diversas regiões ao redor do mundo, proporcionada pelo mapeamento de áreas externas, ainda assim, pode haver dificuldades para pessoas encontrem seus destinos em ambientes internos como *shoppings*, hospitais, fábricas, estádios, aeroportos entre outros. As universidades são um exemplo de ambiente interno, contêm áreas extensas com diversos setores e edifícios, que dificilmente uma pessoa que não conhece o local irá encontrar com facilidade um ponto de interesse sem a ajuda de um instrutor ou com auxílio de um mapa interno da área. Dito isso, este trabalho descreve o desenvolvimento de um aplicativo *mobile* que calcula e informa o menor caminho entre pontos de interesses localizados no ambiente interno do Centro Universitário Luterano de Palmas. Para criação do mapa digital, são criadas imagens do ambiente interno do campus no formato SVG, os pontos de interesse são modelados em um grafo que possui informações das coordenadas de cada local, no qual é aplicado o algoritmo Dijkstra para determinar o caminho mínimo. O resultado é aplicativo *mobile* para auxiliar alunos, funcionários e visitantes da instituição a encontrarem pontos de interesse na área interna do campus, permitindo definir uma origem e um destino para calcular o caminho de menor distância entre eles.

Palavras chave: Grafos, algoritmo de Dijkstra, SVG.

LISTA DE FIGURAS

Figura 1 - As sete pontes de Königsberg 10
Figura 2 - Diagrama criado por Euler 11
Figura 3 - Grafo ciclo de amizade 12
Figura 4 - Grafo com vértices rotulados por grau 12
Figura 5 - Representação de distância entre capitais 13
Figura 6 - Grafo ciclo 14
Figura 7 - Pré-processamento algoritmo de Dijkstra 16
Figura 8 - Pseudocódigo do algoritmo de Dijkstra 16
Figura 9 - Processo de “relaxamento” 17
Figura 10 - Grafo ilustrativo com 5 vértices 18
Figura 11 - Etapas do desenvolvimento do aplicativo 22
Figura 12 – Representação da infraestrutura interna do CEULP/ULBRA 24
Figura 13 – Desenho do mapa 25
Figura 14 - Desenho do Grafo 26
Figura 15 - Estrutura do arquivo JSON 26
Figura 16 – Protótipo 27
Figura 17 – Mapa renderizado no aplicativo 28
Figura 18 – Telas de pesquisa 29
Figura 19 – Node Model 30
Figura 20 - Algoritmo de Dijkstra 32
Figura 21 - Serviço de desenho 34
Figura 22 – Caminho entre o “Hall” e a “Sala 314” 35
Figura 23 – Caminho entre a “Sala 311” e a “Sala 219” 36

LISTA DE TABELAS

Tabela 1 - Passos da aplicação do algoritmo de Dijkstra 19

LISTA DE ABREVIATURAS E SIGLAS

API - Application Programming Interface

CEULP - Centro Universitário Luterano de Palmas

GPS - Global positioning system

PDF - Portable Document Format

SDK - Software development kit

JSON - JavaScript Object Notation

SMIL - Synchronized Multimedia Integration Language

SVG - Scalable Vector Graphics

ULBRA - Universidade Luterana do Brasil

W3C - World Wide Web Consortium

WEB - World Wide Web

XML - Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO **7**
2 REFERENCIAL TEÓRICO **10**
2.1 GRAFOS 10
2.2 ALGORITMO DE DIJKSTRA 15
3 METODOLOGIA **20**
3.1 MATERIAIS 20
3.1.1 Dart **20**
3.1.2 Flutter **20**
3.1.2 SVG **21**
3.2 MÉTODOS 22
4 RESULTADOS E DISCUSSÃO **24**
4.1 MAPEAMENTO 24
4.2 CRIAÇÃO DO GRAFO 25
4.3 CONSTRUÇÃO DO PROTÓTIPO 27
4.4 DESENVOLVIMENTO DA INTERFACE 28
4.5 APLICAÇÃO DO ALGORITMO DE DIJKSTRA 30
4.6 DESENHO DO CAMINHO MÍNIMO 33
5 CONSIDERAÇÕES FINAIS **36**
REFERÊNCIAS **38**

1 INTRODUÇÃO

Com a evolução tecnológica e a popularização dos dispositivos móveis, surgem frequentemente novas aplicações para diferentes finalidades, que interferem diretamente na cultura e nos hábitos da sociedade. Em vista disso, empreendedores exploram diversas possibilidades a serem desenvolvidas, desde um simples aplicativo para organização de tarefas, a aplicativos avançados como os de *delivery* que utilizam mapas digitais complexos para representar perímetros urbanos.

Os mapas digitais são recursos que estão cada vez mais presentes na rotina das pessoas. Segundo Barbalho (2017), os mapas físicos são estáticos e limitados, restritos a informações por página visualizada e com uma escala fixa, já os mapas digitais apresentam um formato dinâmico, podendo ser navegado com independência, possibilitado a ampliação e redução da escala de visualização.

Através do desenvolvimento de aplicações que facilitam a roteirização e localização sem a necessidade de mapas físicos ou informações dadas por pessoas da localidade, é possível explorar o mundo apenas utilizando um dispositivo móvel na palma mão. Essas aplicações são orientadas através do Sistema de Posicionamento Global (GPS) que informa a um dispositivo receptor móvel sua posição atual de qualquer lugar ou momento (TAROUCO, 2013).

A maioria dessas aplicações contém mapas digitais com o intuito de exibir o menor percurso até um determinado destino. O conceito por trás desse propósito denomina-se roteirização e tem a finalidade de criar rotas otimizadas, ou seja, busca minimizar a distância de trajetos, normalmente com o objetivo de reduzir o tempo, fadiga ou recursos na rota percorrida. Junior (2019) afirma que a roteirização é uma metodologia fundamental para a redução de custos e garante aperfeiçoamento na prestação de serviços, ao reduzir o tempo de percurso e melhorar o desempenho operacional.

O *Google Maps* é um serviço de visualização de mapas digitais, desenvolvido e fornecido gratuitamente pela Google. Possui diversos recursos disponíveis, como visualização de ruas na função *Street View*, bem como a função de planejamento de rotas, que permite ao usuário verificar o caminho exato para um determinado local, informações sobre o transporte público, condição do trânsito em tempo real, além de permitir o salvamento dos mapas para utilização no modo *offline* (TAROUCO, 2013).

Apesar de existir certa facilidade proporcionada pelo mapeamento de diversas regiões ao redor do mundo, ainda assim pode haver dificuldades para pessoas encontrem seus destinos em ambientes internos, em locais com diversos setores, como uma universidade por exemplo. Pensando nisso, foi desenvolvida uma ferramenta para integrar ao *Google Maps,* denominada

Google Indoor Maps, que busca facilitar a localização em ambientes internos, como *shoppings centers*, estádios de futebol e aeroportos. Os mapas internos (*Indoor Maps*) representam a área interna de um ou mais edifícios e incluem estruturas como estradas, corredores, elevadores, salas, portas, escadas, entre outros (LEE, 2014).

Uma das principais funcionalidades dos mapas digitais é a opção de ampliação e redução da escala de exibição, criada para os usuários visualizarem maiores detalhes de uma determinada área do mapa. Para que essa funcionalidade execute correntemente é necessário que a área que esteja sendo visualizada não sofra efeitos de *pixelização*.

O *Scalable Vector Graphics* (SVG) é um formato XML (*Extensible Markup Language*) utilizado para criação de gráficos bidimensionais, elaboração de mapas digitais e animações. Diferentemente das ilustrações *bitmap* (*raster*), os dados que compõem imagens vetoriais não são *pixels*, este formato utiliza fórmulas matemáticas para desenhar as curvas e linhas das imagens (CABRAL, 2005).

De acordo com Júnior (2004), as imagens construídas no formato SVG possuem recursos de *zoom* e são escalonáveis, isto significa que os usuários podem ampliar ou reduzir uma área específica de uma ilustração, como um mapa por exemplo, sem perda de definição ou diminuição dos detalhes da imagem. Consequentemente, as imagens no formato SVG por serem escalonáveis são de alta qualidade em qualquer resolução.

É bastante comum ver pessoas perdidas ou confusas ao tentar encontrar algum ponto de interesse em um lugar desconhecido, principalmente quando a área desse local é ampla. O Centro Universitário Luterano possui uma infraestrutura de 268.233,32 m² de área total e 51.166,00 m² de área construída, com mais de 150 salas de estudo, além de escritórios modelos, laboratórios e clínicas, divididos em blocos e setores (CEULP, 2020).

Atualmente, as informações sobre as salas de aulas do campus são disponibilizadas aos alunos e atualizadas semestralmente no portal acadêmico da instituição, na página “Ache sua sala”. As informações que são apresentadas na página web são separadas por cursos e turmas institucionais, cada opção possui uma lista de disciplinas, que inclui dados como o nome da disciplina, o horário das aulas, a identificação da turma e o número da sala.

Em vista disso, é correto afirmar que existe uma página *web* que cita a localização da sala de aula, mas não existe um *software* que informe a localização e a rota até as salas. Além disso, no início de todo o semestre na instituição, durante a primeira semana de aula é designada uma equipe para ajudar os calouros a encontrarem suas salas. Mas, a equipe somente informa o bloco ou prédio em que está situada determinada sala, não acompanha o calouro até o local, sendo que a maioria desses acadêmicos estão em sua primeira visita ao campus.

Consequentemente, esses alunos podem encontrar dificuldades para localizar seus destinos, ainda mais quando se conhece o local, bloco ou setor onde está localizada a sala. Por esse motivo, os alunos e visitantes solicitam informações para pessoas próximas ou procuram informações em placas informativas. Porém, na maioria das vezes essas informações são incompreensíveis ou desatualizadas.

Portanto, é justificável o desenvolvimento de um aplicativo *mobile* que auxilie na localização dos alunos e visitantes no ambiente interno do campus, sem a necessidade de estar acompanhado de um colaborador ou um mapa físico da instituição. O aplicativo ajudará os usuários a encontrarem pontos de interesses mais facilmente, auxiliando através de rotas definidas pelo algoritmo de cálculo de caminho de custo mínimo algoritmo de *Dijkstra*.

Neste contexto, o objetivo principal deste trabalho foi desenvolver um aplicativo *mobile* para calcular e informar o menor caminho entre um ponto de origem e destino localizados no ambiente interno do CEULP/ULBRA. Para isso, foram utilizados os conceitos de grafos, visando organizar as informações de localização das salas, e o algoritmo de Dijkstra para determinar o menor caminho entre um dado ponto de origem e destino.

Para atingir esse objetivo foi necessário mapear as coordenadas dos pontos de interesse e as rotas entre as coordenadas, criar um grafo com as informações da localização e distância entre os locais, implementar o algoritmo *Dijkstra* na linguagem de programação Dart, representar a área interna do campus através de imagens vetorizadas no formato SVG e desenvolver uma interface *mobile* com o *framework* Flutter para apresentar o menor caminho entre a origem e o destino.

Este documento está estruturado em cinco capítulos. O segundo capítulo apresenta uma fundamentação teórica sobre grafos e o algoritmo de *Dijkstra*, que serviu de base para o desenvolvimento deste trabalho. O terceiro capítulo descreve os materiais utilizados no desenvolvimento do aplicativo *mobile*, além da metodologia adotada. O quarto capítulo expõe os resultados obtidos e discussões sobre esses resultados. Por fim, o quinto capítulo apresenta as considerações finais e trabalhos futuros.

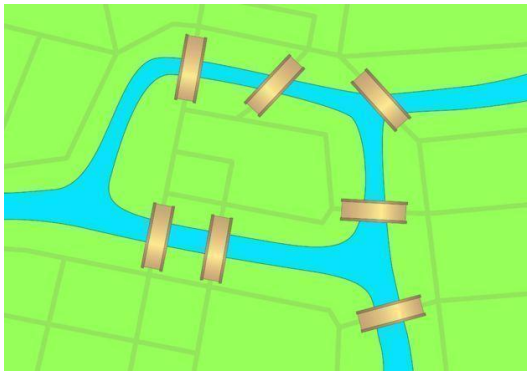
2 REFERENCIAL TEÓRICO

Esta seção apresentará os principais conceitos e definições relacionados a Teoria dos Grafos e Algoritmo de *Dijkstra*. O entendimento destes conceitos foi fundamental para o desenvolvimento do projeto proposto neste trabalho, que se resume em desenvolver um aplicativo *mobile* para calcular e informar o caminho de menor distância entre pontos localizados no ambiente interno do CEULP/ULBRA.

2.1 GRAFOS

A teoria dos grafos foi introduzida no século XVIII pelo matemático e físico suíço Leonhard Euler, criou o conceito ao solucionar o enigma conhecido como “o problema das sete pontes de Königsberg” (NEGRI, 2017). Königsberg, capital da Prússia, hoje sob o nome de Kaliningrado na Rússia, possuía uma ilha chamada de Kneiphof, onde existiam sete pontes, como está ilustrado na Figura 1.

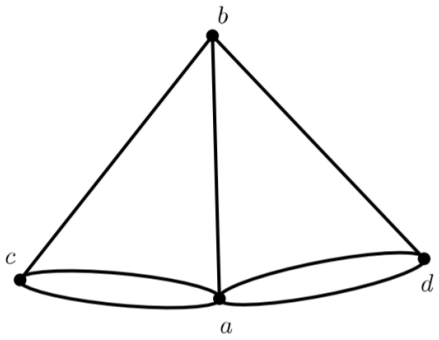
Figura 1 - As sete pontes de Königsberg



Fonte: SAUTOY (2018)

As sete pontes atravessam o rio Pregel interligavam diferentes pontos da cidade e da ilha. Um passatempo comum entre os habitantes da região, era tentar encontrar um caminho que passasse pelas setes pontes iniciando e finalizando no lugar de origem, cruzando cada ponte apenas uma vez. Em 1736, Euler provou que não existia um caminho que possibilitasse esse feito, para chegar a esse resultado representou o problema em um diagrama conforme é ilustrado na Figura 2, cada parte da terra é representado por um vértice e cada ponte é uma aresta (PRESTES, 2016).

Figura 2 - Diagrama criado por Euler



Fonte: COSTA (2017)

Para resolver o problema das sete pontes foi necessário traçar um caminho fechado compondo todas as ilhas e pontes. Para tal fim, Euler criou um diagrama conforme está sendo apresentado na Figura 2, onde os pontos *a, b, c, d* representam as ilhas e as sete linhas que interligam esses pontos são equivalentes às pontes.

Como mencionado anteriormente, Euler encontrou um resultado negativo ao tentar solucionar o enigma das sete pontes, percebeu que para percorrer cada ponto, são necessárias duas linhas, uma para entrar e outra para sair. Isto significa que ao entrar na ilha, são feitas duas travessias, uma para chegar e outra para voltar. Em consequência disso, Euler concluiu que cada ponto deve ter um número par de conexões para percorrer todo o grafo, passando uma única vez por cada ponte. Pelo fato de o grafo construído possuir um número ímpar de conexões, o problema não tem uma solução (JURKIEWICZ, 2009).

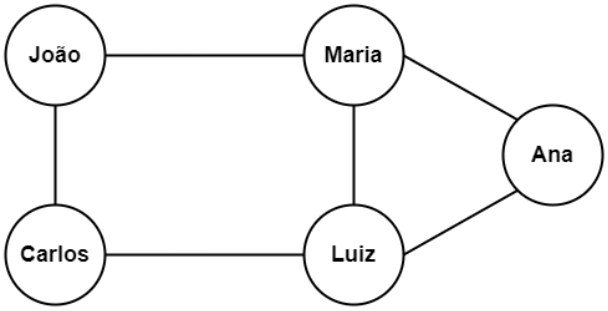
Como a maioria dos conceitos matemáticos, a teoria dos grafos também possui diversos termos e nomenclaturas. Negri (2017, p. 27) traz a seguinte definição matemática para grafos:

> Um grafo G consiste em um conjunto finito e não vazio $V(G)$ de objetos chamados vértices, juntamente com um conjunto $E(G)$ de pares não ordenados de vértices cujos elementos são chamados de arestas. Pode-se representá-lo por $G = (V; E)$, onde $V = V(G)$ e $E = E(G)$. Seja $G(V; E)$ um grafo e u, v dois de seus vértices, diz-se que u e v são adjacentes se existe uma aresta que incida nestes dois vértices. Pode-se denotar a aresta por $\{u, v\}$ ou simplesmente uv quando não houver perigo de confusão.

Os Grafos são habitualmente demonstrados através de diagramas, é desenhado um círculo ou ponto para representar cada vértice (nó), e para cada aresta é desenhado um arco ou uma linha para conectar vértices equivalentes. Com a utilização dos grafos é possível descrever o relacionamento entre pares de objetos como pessoas, lugares, computadores, ao conectar esses objetos são criados relacionamentos, identificados como amizade, caminho, conexão, entre

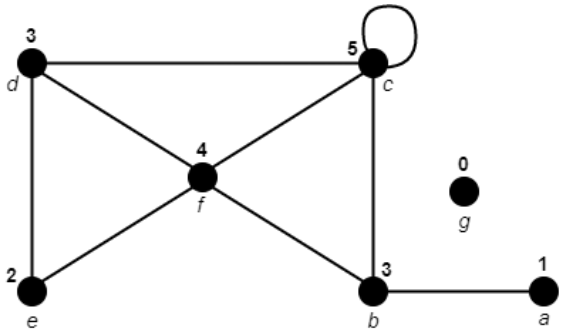
outros. Os objetos são os vértices do grafo e os relacionamentos são as arestas, dessa maneira várias situações são possíveis de serem representadas através da aplicação dos conceitos de grafos.

Figura 3 - Grafo ciclo de amizade



O exemplo apresentado na Figura 3 é um grafo simples, visto que não possui laços e arestas paralelas, ou seja, não contém uma aresta ligando as extremidades ao mesmo vértice e não contém mais de uma aresta ligando dois vértices. Diferentemente do grafo da Figura 2, nos vértices c e a possuem duas arestas conectando os dois vértices, isso é denominado como multigrafo (NEGRI, 2017). O grafo da Figura 3 representa um grupo de pessoas conectadas socialmente, com conjunto de vértices $V = \{João, Maria, Ana, Luiz, Carlos\}$ e um conjunto de arestas $E = \{\{João, Maria\}, \{Maria, Ana\}, \{Ana, Luiz\}, \{Luiz, Maria\}, \{Luiz, Carlos\}, \{Carlos, João\}\}$, o exemplo declara que a amizade entre essas pessoas é "simétrica", em outro termo: se x é amigo de y então y é amigo de x.

Figura 4 - Grafo com vértices rotulados por grau

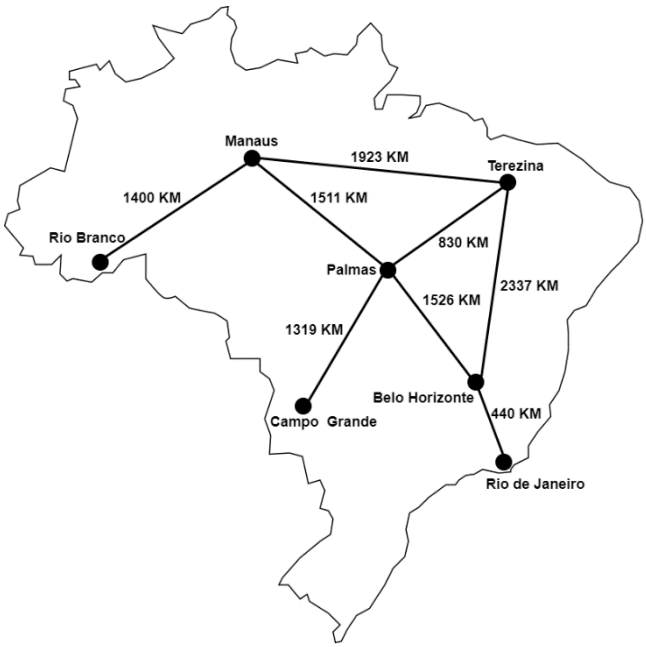


Para cada vértice v de um grafo G, define-se grau (valência) como a quantidade de arestas que incidem em cada vértice v (COSTA, 2011). No grafo exemplo da Figura 4, os vértices b e d possuem grau três, f tem grau quatro, e grau dois, a grau um, c possui um laço

nesse caso ao invés de somar uma é somado duas incidências, então c tem grau cinco e g não contém arestas, nesse caso o vértice g tem grau zero. Um teorema interessante sobre graus, é que "a soma dos graus dos vértices de um grafo é sempre o dobro do número de arestas" (SOUZA, 2014, pg. 7).

Levando em consideração que um grafo G, possui dois vértices em cada uma das arestas, ao calcular a quantidade de arestas incidentes em cada vértice, cada aresta será acrescentada duas vezes na somatória, em vista disso, o resultado final da soma dos graus de todos os vértices sempre será o dobro do número de arestas. O grafo da Figura 4, possui 9 arestas, com respectivos graus para cada vértice: $a = 1, b = 3, c = 5, d = 3, e = 2$ e $f = 4$, a soma total dos graus dos vértices é 18, ou seja, o dobro da quantidade de arestas. O vértice g é desconsiderando por ser um grafo trivial, ou seja, um grafo com apenas um único vértice e sem arestas (NEGRI, 2017).

Figura 5 - Representação de distância entre capitais[1]



A maioria dos contextos que envolvem grafos, é atribuído um valor para cada aresta, que identifica uma informação de um vértice a outro, esse valor pode representar um custo, um peso, uma distância e assim por diante. O grafo de exemplo da Figura 5 apresenta esse conceito, os valores das arestas representam a distância em quilômetros entre algumas capitais do Brasil

[1] As posições das capitais e as distâncias entre as mesmas apresentadas no mapa é meramente ilustrativa, não condiz com exatidão à realidade.

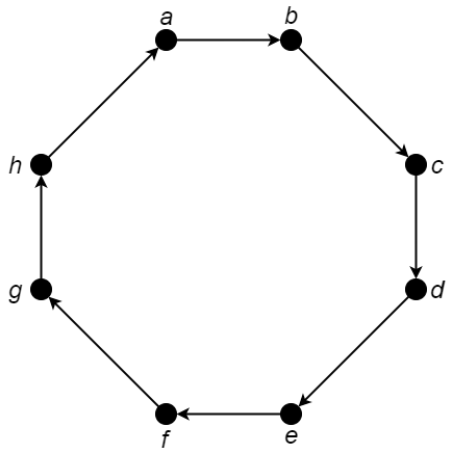
representadas pelos vértices. De acordo Souza (2014), esse conceito é definido como grafo valorado (rotulado ou ponderado), que consiste em um grafo $G = (V; E; T)$, em que V é o conjunto de vértices do grafo, E é o conjunto de arestas do grafo e T é o conjunto de números reais que são associados a cada uma das arestas dos grafos ou aos vértices do grafo.

Um passeio em um grafo G é uma sequência (lista) finita de arestas $(v_0, v_1, \dots, v_{x-1}, v_x)$, onde x representa o comprimento do passeio (SOUZA, 2014). O comprimento de um passeio é dado pelo seu número de arestas.

Um caminho em um grafo G é um passeio entre vértices v_0 e v_x, ou seja, é o conjunto de arestas que conectam v_0 a v_x (COSTA, 2011). A sequência de arestas que conectam o vértice $Manaus$ ao vértice $Rio\ de\ Janeiro$ é um exemplo de caminho, levando em consideração o grafo da Figura 5. A origem é o primeiro vértice de um caminho, e o destino é seu último vértice. Considerando que um caminho tem a origem no vértice v_x e destino em v_y, expressamos que o caminho inicia em v_x e termina em v_y.

De acordo com Costa (2011), se todas as arestas de um passeio são distintas, denomina-se o próprio como trilha, caso contrário se $v_0 = v_x$, o passeio é um circuito ou uma trilha fechada. Um circuito em que todos os vértices são distintos, com exceção do primeiro e do último, declara-se como ciclo ou caminho fechado, em outras palavras, é um grafo G com conjunto de vértices $V = \{v_0, v_1, \dots, v_{x-1}, v_x\}$ e que com conjunto de arestas $E = \{\{v_0, v_1\}, \dots, \{v_{x-1}, v_x\}, \{v_x, v_0\}\}$.

Figura 6 - Grafo ciclo



A Figura 6 apresenta um exemplo de grafo $G = (V; E)$ com ciclo, com um conjunto de arestas $E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{d, e\}, \{e, f\}, \{f, g\}, \{g, h\}, \{h, a\}\}$. O grafo contém um ciclo visto que não existe aresta repetida no conjunto e os vértices inicial e final são idênticos. A

sequência de vértices $V = \{a, b, c, d, e, f, g, h, a\}$, é um exemplo de circuito. Nota-se que as arestas deste grafo são direcionadas, ou seja, há sinalizações de direção entre os vértices.

Segundo Souza (2014), um grafo direcionado (digrafo) é um par ordenado $G = (V;\ E)$, que há uma relação de um vértice para outro, determinado a direção a ser seguida. Esse conceito é útil, por exemplo, em um grafo G em que as arestas representam rotas, como em cidade que contém vias de mão única, essa via há uma direção específica em que os veículos devem seguir.

Para determinar o menor caminho entre dois locais, sendo que existem diversos caminhos com distâncias diferentes entre esses lugares, torna-se viável a criação de um grafo que contenha as coordenadas desses pontos, para estruturar caminhos entre essas coordenadas. Com a utilização de grafos é possível conectar os pontos formando "caminhos", possibilitando encontrar o que possui menor custo, isso pode ser calculado utilizando o algoritmo de Dijkstra, explicado a seguir.

2.2 ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra, criado no século XVI pelo cientista da computação holandês Edsger Wybe Dijkstra, soluciona o problema do caminho mínimo entre um dado vértice a qualquer outro ponto de um grafo, ou melhor, serve para encontrar o caminho com menor tempo, distância ou custo entre dois vértices de um grafo (MENDONCA, 2017).

O problema do caminho mínimo pode ser empregado em vários contextos, ele aparece na maioria das situações do dia a dia, como por exemplo, no transporte, em redes de comunicação, na coleta de lixo, entre outros, em que seja necessário encontrar o caminho mais rápido, acessível ou seguro entre dois pontos. Em outras palavras, esse problema baseia-se em descobrir a rota entre uma origem e um destino, reduzindo o custo do caminho entre um par vértices de um grafo, o custo corresponde à soma de cada aresta percorrida no percurso.

A aplicação do algoritmo ocorre apenas em grafos $G = (V;\ E)$, em que o conjunto E inclua somente arestas com pesos não negativos (NEGRI, 2017). Esta restrição aplica-se em contexto como exemplo, em aplicações financeiras ou transporte, onde as arestas geralmente representam uma quantia em moeda ou tempo médio de um percurso, em situações como essas caso os valores atribuídos às arestas forem negativos, o algoritmo não funcionará corretamente.

De acordo com Cormen (2012), se o grafo G possui um ciclo de peso negativo que inclua a origem s, para todo $v \in V$, o peso do caminho mínimo é definido como $d(s, v) = -\infty$. Isto significa que nenhum caminho que parte da origem s a outro vértice do ciclo será o caminho mínimo, visto que sempre existirá um caminho de menor custo ao percorrer o ciclo de peso negativo.

Figura 7 - Pré-processamento algoritmo de Dijkstra

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for cada vértice v ∈ V[G]
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

Fonte: CORMEN (2012)

O método de inicialização do algoritmo de Dijkstra é denominado *INITIALIZE-SINGLE-SOURCE*, conforme apresentado na Figura 7. O método recebe dois parâmetros, sendo eles o grafo G e o vértice de origem s, esse procedimento consiste em atribuir para todos os vértices do grafo: o caminho de custo mínimo igual a nulo (*NIL*) e uma estimativa de distância mínima como infinito, com exceção do vértice de origem s, que é definido como $zero$ (OSTA, 2019). O valor atribuído a esses vértices deve ser extremamente grande ou infinito, com a finalidade de quando ocorrer a verificação do custo mínimo, esse peso seja maior que qualquer outro presente no grafo.

Figura 8 - Pseudocódigo do algoritmo de Dijkstra

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = V[G]
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for cada vértice v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

Fonte: CORMEN (2012)

A Figura 7 apresenta o pseudocódigo do algoritmo de Dijkstra, o algoritmo recebe três parâmetros, sendo G equivalente a um grafo com n vértices, w uma função que retorna o peso das arestas entre dois vértices, o terceiro parâmetro s é o vértice de origem (OSTA, 2019). Como explicado anteriormente, na primeira linha o método INITIALIZE-SINGLE-SOURCE, é um pré-processamento que atribui um custo à cada vértice do grafo.

Segundo Osta (2019) na linha dois e três é inicializado duas variáveis: um conjunto S vazio e uma fila de prioridade Q, contendo os vértices, o atributo de distância $v.d$ de cada vértice é utilizado como parâmetro de comparação. Em seguida da quarta à sexta linha, ocorre uma interação sobre Q enquanto existir vértices adjacentes na fila, o vértice u é removido, sendo u a distância mínima até o vértice de origem s contido em Q, logo após, u é adicionado em S. O algoritmo opera com esses dois conjuntos de vértices, S contém os vértices cuja o menor peso até a origem já foi descoberto e Q uma fila de prioridade mínima que inclui os vértices que o peso ainda é uma estimativa.

O algoritmo de Dijkstra utiliza a técnica de relaxamento aresta, baseia-se em procurar alternativas para melhorar o resultado do caminho mais curto até um determinado vértice. Essa técnica é destinada para cada vértice $v \in V$, mantém um atributo $v.d$, que é uma estimativa de caminho mais curto desde o vértice de origem s até v, ao encontrar um caminho mais curto para v o atributo $v.d$ é atualizado (CORMEN, 2012).

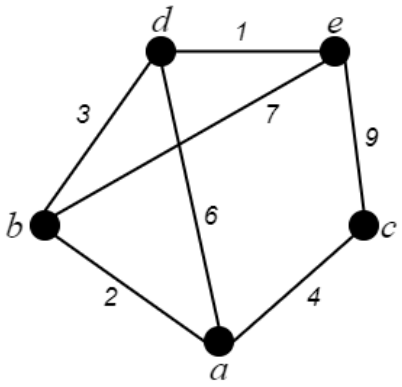
Figura 9 - Processo de "relaxamento"

```
RELAX(u, v, w)
1   if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

Fonte: CORMEN (2012)

Por último na linha sete da Figura 8, todos os vértices adjacentes a u são visitados, e para cada vértice é o método "RELAX" é iniciado, conforme apresentado na Figura 9. O método recebe três parâmetros: o vértice u, o adjacente v e a função de distância w, logo abaixo é feito a atualização das estimativas de distância mínima. Ao finalizar a execução do método, tem como resultado a distância de custo mínimo, partindo do vértice de origem s para todos os outros vértices do grafo, sendo que a estimativa de distância mínima está presente no atributo $v.d$ (OSTA, 2019).

Figura 10 - Grafo ilustrativo com 5 vértices



Considerando o grafo G apresentando na Figura 10, composto por um conjunto de vértices $V = \{a,\ b,\ c,\ d,\ e\}$ e interligados por um conjunto de arestas $E = \{\{a, b\}, \{b, d\}, \{d, e\}, \{e, c\}, \{c, a\}, \{a, d\}, \{b, e\}\}$, com seus respectivos pesos. O algoritmo de Dijkstra será utilizado para obter o menor caminho do vértice de origem a para todos os outros vértices do grafo. Para descrever os caminhos será utilizado a seguinte representação geométrica $\left(\{x_0,\ x_1,\ x_2, x_3\},\ \{\{x_0, x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}\right) = n$, constituída por dois conjuntos, o primeiro corresponde aos vértices do caminho e o segundo representa as arestas que ligam os vértices equivalentes, n condiz ao custo do caminho.

Como primeiro passo, o algoritmo de Dijkstra atribui o valor zero para a estimativa custo mínimo do vértice inicial a e ∞ às demais estimativas. Em seguida, o conjunto S é criado $S = \emptyset$, e todos vértices do grafo G são adicionados ao conjunto Q, logo $Q\ = \{a, b, c, d, e\}$. As distâncias dos os vértices inclusos no conjunto Q são comparadas, em seguida, como vértice a possui a menor estimativa de custo mínimo é adicionado ao conjunto S e removido de Q. Nota-se na Figura 10 que os elementos incluídos no conjunto $G.Adj[a] = \{b,\ c,\ d\}$, são vértices adjacentes ao a.

Posteriormente, o algoritmo de Dijkstra inicia o processo de "relaxamento" para cada vértice adjacente de a, para esse fim o algoritmo realiza uma indagação, se é possível melhorar a estimativa de custo mínimo para b, c e d através do vértice a. As distâncias desses vértices adjacentes são calculadas, a como é o vértice de origem possui peso 0, o cálculo é feito somando esse valor com peso das arestas que os unem.

Considerando o peso 0 do vértice a e o peso da aresta $\{a,\ b\} = 2$, ao somar esses pesos, é dito que o caminho que parte a com destino ao vértice b tem peso 2, e assim sucessivamente, o caminho $(\{a,\ c\}, \{\{a,\ c\}\}) = 4$ e $(\{a,\ d\}, \{\{a,\ d\}\}) = 6$, ao finalizar os cálculos realiza-se uma comparação com a estimativa de cada vértice, caso o custo encontrado seja menor, o atributo é atualizado. Como o vértice a não contém uma ligação direta para o vértice e, ou seja,

o vértice e não é adjacente de a, nesse caso a estimativa de do caminho de custo mínimo de e não é atualizada, ou seja, permanece como ∞.

Ao finalizar os cálculos de custo mínimo dos vértices adjacentes à a, um novo ciclo é iniciado, as distâncias dos vértices inclusos no conjunto Q são comparadas, e o vértice com menor o custo é adicionado ao conjunto S. Esse ciclo é repetido até que todos os elementos incluídos no conjunto Q sejam visitados e as distâncias mínimas partindo do vértice origem para todos os outros vértices do grafo estejam incluídas no conjunto S. A Tabela 1 exibe o passo a passo da aplicação do algoritmo de Dijkstra no grafo apresentado na Figura 10.

Tabela 1 - Passos da aplicação do algoritmo de Dijkstra

Vértice	Passo 1	Passo 2	Passo 3	Passo 4	Passo 5
a	$\mathbf{(0, a)}$	*	*	*	*
b	$(2, a)$	$\mathbf{(2, a)}$	*	*	*
c	$(4, a)$	$(4, a)$	$\mathbf{(4, a)}$	*	*
d	$(6, a)$	$(5, b)$	$(5, b)$	$\mathbf{(5, b)}$	*
e	∞	$(9, b)$	$(9, b)$	$(6, d)$	$\mathbf{(6, d)}$

Os elementos que estão destacados na Tabela 1, indica a menor distância de cada vértice em relação a origem a. Por exemplo, o caminho de custo mínimo do vértice e para a origem a é o seguinte percurso: $(\{a,\ b,\ d, e\},\ \{\{a,\ b\}, \{b, d\}, \{d, e\}\}) = 6$. Observa-se na Tabela 1 é apresentado o mesmo percurso, porém, o inverso, no passo 5 temos o elemento "(6, d)", simboliza que para chegar ao vértice e foi necessário passar pelo o vértice d, e assim sucessivamente, para d passou-se através de b, por fim, para chegar à b foi necessário partir do vértice a.

Com o intuito de calcular e informar o menor caminho entre dois lugares, dado que as coordenadas desses pontos e a distância entre eles estejam armazenadas em um grafo, torna-se viável a implantação do algoritmo Dijkstra para determinar o caminho mínimo entre um dado local de origem e um local de destino.

3 METODOLOGIA

Nessa seção são apresentados os materiais e os métodos utilizados no processo de desenvolvimento do aplicativo *mobile*.

3.1 MATERIAIS

Os materiais que foram utilizados no desenvolvimento deste trabalho foram Dart, Flutter e SVG, conforme descritos a seguir.

3.1.1 Dart

Dart é uma linguagem de programação criada originalmente para o desenvolvimento de *scripts* para páginas web, desenvolvido pela *Google* em um projeto criado pelos desenvolvedores Lars Bark e Kasper Lund, anunciada oficialmente na conferência GOTO em outubro de 2011 em Aarhus, na Dinamarca (DART, 2020).

O Dart é uma linguagem *open source*, multiparadigma e fortemente tipada, apesar das declarações de tipo serem opcionais, além disso sua sintaxe é baseada na linguagem C. Os projetos criados em Dart são estruturados de maneira modular, em unidades chamadas *libraries*, um projeto compõe um ou mais *library* que integram a base de um código. Um código implementado em Dart pode ser compilado em *ahead-of-time* (AOT) que transforma o código implementado em código de máquina e *just-in-time* (JIT) que permite a compilação em tempo de execução (ALMEIDA, 2019).

Embora o Dart tenha sido projetado inicialmente para o desenvolvimento de *scripts* para *web*, geralmente conhecemos essa linguagem graças ao *framework* Flutter, criado para desenvolvimento de aplicações multiplataforma, com ele é possível desenvolver aplicações *mobile, web e desktop*. Para este projeto a linguagem de programação Dart foi utilizada no código fonte da aplicação, dando suporte às regras de negócio, ou seja, especificando as características das funcionalidades do aplicativo.

3.1.2 Flutter

O Flutter é um *UI toolkit* (kit de desenvolvimento de interface de usuário) *open source* criado pela *Google*, para o desenvolvimento de aplicativos *Android, iOS, web* e *desktop* a partir de uma única base de código. Para a implementação dos códigos das aplicações, é utilizado o Dart como linguagem de programação padrão. O propósito do *framework* é permitir que desenvolvedores criem aplicativos de alta performance e fluídos com experiência nativa em todas as plataformas (FLUTTER, 2020).

No Flutter o fluxo de desenvolvimento é orientado ao design, todos componentes do são *widgets*, como exemplo os elementos estruturais como botões e menus, de estilo como fontes e cores, de *layout* como margens, espaçamento, além de possuir *widgets* com design particular para o Android (*Material Componets*) e iOS (*Cupertino*). Os *widgets* são estruturados em uma árvore de *widgets*, em uma hierarquia pai e filho, sendo responsável em formar o *layout* que é renderizado na tela (ALMEIDA, 2019).

Neste projeto o Flutter será utilizado para desenvolver a interface do aplicativo *mobile*. Entre os principais pontos que levaram a escolha dessa tecnologia para o desenvolvimento deste trabalho são: facilidade no aprendizado do Dart e Flutter, velocidade no desenvolvimento (com poucas linhas de código já é possível ver os resultados em tela), e a performance nativa.

3.1.2 SVG

O *Scalable Vector Graphics* (Gráficos Vetoriais Escalonáveis), conhecido popularmente pela sigla SVG, é utiliza a sintaxe do XML para reproduzir gráficos baseados em vetores compostos por elementos como caminhos, textos, formas, cores, entre outros. Existem várias vantagens na utilização do formato SVG, mas a mais reconhecida é sua tecnologia que evita distorções. As ilustrações construídas neste formato não sofrem efeitos de *pixelização*[2] ao ampliar ou reduzir o tamanho, isto é, a qualidade da imagem não sofre perda em relação ao conteúdo original (KAUAN, 2015).

O SVG é um padrão recomendado pela W3C[3], para a geração de gráficos bidimensionais, mapas digitais e animações. Com esse modelo é possível criar páginas *web* de alta resolução gráfica, integrando vários elementos como declives, fontes embutidas, transparências e filtros. Além disso, proporciona a representação de três categorias de objetos gráficos, sendo formas vetoriais, imagens e texto. Os dados compõem esses objetos vetoriais não *pixels*, ao invés disso, são utilizadas expressões matemáticas para representar formas geométricas das ilustrações (CABRAL, 2005).

Os elementos básicos para criação de formas geométricas no SVG são representados pelas *tags*: *line*, *rectangle*, *polygon*, *circle*, *polyline*, *rect* e *ellipse*. Todas essas formas básicas são efetivamente instâncias básicas do elemento *path*, criado para especificar um caminho ou uma sequência de linhas e curvas. Com o elemento *path* é possível desenhar todas as formas,

[2] Efeito ocorrido quando é ampliando uma determinada área de uma imagem *raster* (*bitmap*), o que leva a exibição dos *pixels* individuais que compõem a imagem.
[3] Consórcio que regulamenta diretrizes e protocolos para a *web*.

sua sintaxe possui comandos como *moveto*, *lineto*, e *closepath*, utilizados para definir as formas geométricas e os caminhos (EISENBERG, 2002).

As imagens construídas no formato SVG suportam diversos tipos de *display*, são redimensionadas de acordo com o tamanho da tela sem distorcer e diminuir a qualidade da ilustração. Uma das principais características de um mapa digital é a opção de ampliar ou reduzir o *zoom* de uma área específica, mas é essencial que a visualização dessa área seja nítida, o SVG é uma opção de tecnologia que torna essa funcionalidade possível. Por essa razão, as imagens do ambiente interno do CEULP/ULBRA foram criadas no formato SVG e em seguida integradas no mapa.

3.2 MÉTODOS

Nessa seção são apresentadas as etapas que foram executadas no processo de desenvolvimento de um aplicativo *mobile* para calcular e informar o menor caminho entre pontos localizados no ambiente interno do CEULP/ULBRA, conforme é ilustrado na Figura 11.

Figura 11 - Etapas do desenvolvimento do aplicativo

A primeira etapa consistiu em mapear a área interna do CEULP/ULBRA, para criação das imagens no formato SVG para serem utilizadas no mapa digital, o mapeamento foi realizado

através do estudo de plantas baixas da infraestrutura do campus. Além disso, foi necessário adquirir as coordenadas das salas, com intuito identificar a localização e a distância entre cada local seguido as medidas disponíveis nas plantas baixas.

Já na segunda etapa foi realizada a criação do grafo, as informações de cada salas foram adicionadas nos vértices do grafo, os quais detêm dados únicos sobre cada local, como a coordenadas e a identificação do local. As arestas do grafo servem de conexão entre os vértices, nelas ficam armazenadas as distâncias e os relacionamentos entre cada local.

Na terceira etapa foi desenvolvido um protótipo da estrutura do aplicativo. A elaboração de um protótipo tem o objetivo de definir a interação entre as funcionalidades do *software*, além de facilitar o desenvolvimento. O protótipo desenvolvido é de média fidelidade, ou seja, sem a implementação de código, oferecendo apenas o layout com navegação entre as funcionalidades, suficiente para evidenciar perspectivas das interações do usuário.

A quarta etapa envolveu o desenvolvimento da interface do aplicativo *mobile* utilizando o Flutter, seguindo todo o planejamento das fases antecedentes. As imagens criadas no formato SVG foram posicionadas no aplicativo e foi desenvolvido uma tela de pesquisa para que seja possível o usuário definir uma origem e destino, exibir no mapa a menor rota de menor distância entre esses pontos.

Já a quinta etapa correspondeu à aplicação do algoritmo do Algoritmo *Dijkstra* no grafo construído na segunda etapa. O algoritmo foi implementado com linguagem Dart, tendo como parâmetros o ponto origem, o destino e o grafo completo, e retorna o caminho mínimo entre pontos escolhidos pelo usuário.

A etapa final consistiu em desenhar o menor caminho sobre o mapa, utilizando as coordenadas dos vértices que compõem o caminho mínimo entre os pontos de interesse obtidos na etapa anterior com a aplicação do algoritmo *Dijkstra*. O caminho foi desenhado manipulando a árvore de objetos da sintaxe XML da ilustração do campus no formato SVG.

4 RESULTADOS E DISCUSSÃO

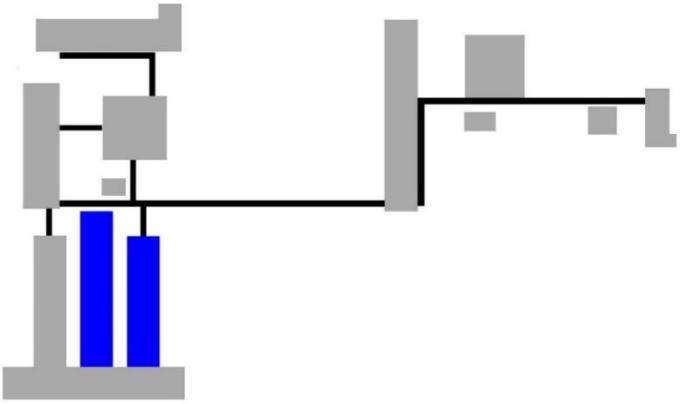
Nesta seção é descrito o processo de execução de cada etapa da solução proposta. Após o estudo detalhado sobre grafos, Algoritmo de Dijkstra e as tecnologias a serem utilizadas, essa seção traz consigo as justificativas para cada tomada de decisão realizada e os resultados obtidos na execução das etapas.

4.1 MAPEAMENTO

Para o desenvolvimento do mapa digital foi necessário estruturar a área interna do campus. Para essa finalidade, foi solicitado as plantas baixas à Fábrica de *Software* da instituição. As plantas baixas foram disponibilizadas no formato PDF (*Portable Document Format*), sendo elas de toda a infraestrutura interna do campus.

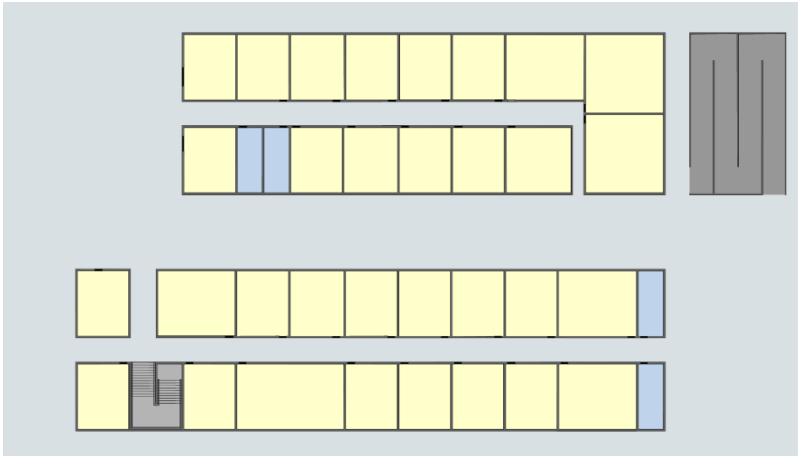
Como citado anteriormente, a área total da infraestrutura é de 268.233,32 m², sendo 51.166,00 m² de área construída, com mais de 150 salas de estudo, incluindo escritórios modelos, laboratórios e clínicas, divididos em vários blocos e setores. Com o intuito de representar essas construções no mapa, com as proporções equivalentes, as imagens no formato SVG foram desenhadas seguindo as dimensões reais da instituição, de acordo com as medidas disponíveis nas plantas baixas.

Figura 12 – Representação da infraestrutura interna do CEULP/ULBRA



A Figura 12 apresenta uma visão geral dos blocos do campus, a representação foi criada de acordo com as estruturas das construções disponíveis nas plantas baixas. Com base no estudo da infraestrutura, inicialmente foram escolhidos dois prédios para o desenvolvimento do mapa, sendo eles os blocos dois e três, localizados em frente à entrada do campus, conforme estão destacados em azul na Figura 12.

Figura 13 – Desenho do mapa



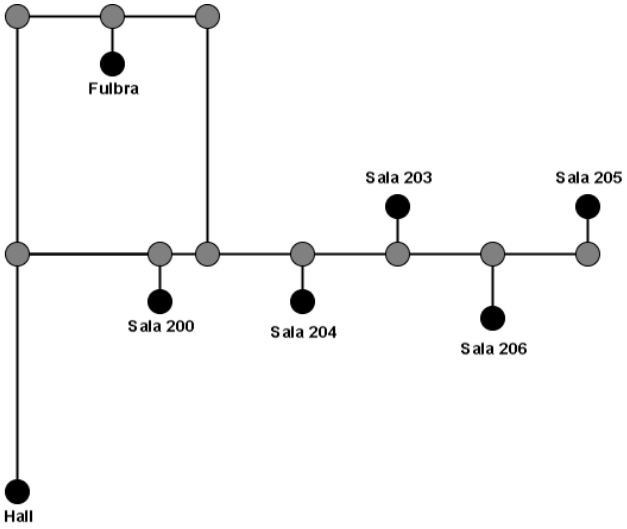
O desenho dos blocos foi realizado utilizando a ferramenta para edição de imagens vetoriais *Adobe Illustrator*, a Figura 13 apresenta o resultado do desenho dos prédios dois e três. A ilustração foi criada importando as plantas baixas no formato PDF específicas de cada bloco para a ferramenta. Assim, a estrutura serviu como base para o contorno das construções de cada bloco. Após finalizar o desenho foi então exportado como uma imagem do formato SVG.

4.2 CRIAÇÃO DO GRAFO

Os problemas para encontrar um local específico envolve diversas variáveis como tempo, distância, custo, recursos entre outros, mas a variável principal é a coordenada que identifica onde um determinado ponto está localizado. Informações como essas podem ser representadas através de grafos, onde os vértices podem conter dados específicos sobre uma determinada localização e as arestas representam os relacionamentos e o custo para se deslocar até outro local.

O grafo construído neste trabalho serviu para armazenar e estruturar as informações de localizações específicas do campus, sendo um grafo valorado não direcionado. Valorado porque todas as arestas possuem um determinado valor que representa a distância para locomover-se para outro vértice, e não direcionado porque os relacionamentos entre os vértices são simétricos, ou seja, o caminho utilizado para chegar até um destino é o mesmo para voltar à origem.

Figura 14 - Desenho do Grafo



Para fins de desenvolvimento e visualização foi criado um diagrama do grafo utilizando a ferramenta *web* Draw.io que é apresentado na Figura 14. A ilustração apresenta um grafo em que os pontos de cor cinza são vértices auxiliares que representam o acesso ao corredor, e os demais são correspondentes aos pontos de interesse. A distância de uma sala para um vértice auxiliar é o mesmo das suas vizinhas, isso foi estabelecido em razão de que o valor da distância entre a entrada de cada sala a um vértice auxiliar é o mesmo para todas as salas.

Para armazenar e organizar os dados dos locais a serem mapeados, foi criado um arquivo JSON (*JavaScript Object Notation*), as informações sobre cada local, como nome, distância e relacionamentos foram inseridas manualmente no arquivo.

Figura 15 - Estrutura do arquivo JSON

```
{
  "floors": [
    {
      "floor": 0,
      "nodes": [
        {
          "name":"Hall",
          "type":"poi",
          "coordinate":{
            "x": 1450,
            "y": 2225
          },
          "edges":[
            {"name": "BC2EntranceHelper","distance": 9.22}
            ]
        }
      ]
    }
  ]
}
```

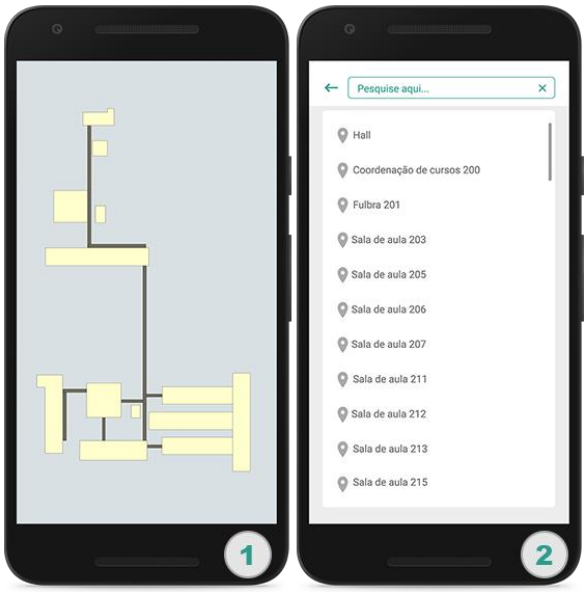
Os dados dos locais foram organizados conforme apresenta a Figura 15, seguindo a seguinte estrutura hierárquica, o campus possui vários andares e em cada andar possui diversos pontos de interesse (POI) e pontos auxiliares (OHRT, 2013). Em um andar contém vários nós, sendo que cada nó possui informações sobre um ponto específico e um número atribuído que se refere a ordem dos andares, por exemplo, o andar 0 é relacionado ao térreo. Um nó representa um ponto auxiliar ou de interesse, sendo obrigatório ter um nome único para que seja possível fazer referências.

Cada ponto possui uma coordenada contendo os atributos (x, y), sendo que x identifica a posição no eixo horizontal e y a posição no eixo vertical da imagem SVG. Além disso, possui um conjunto de arestas que referenciam os relacionamentos e as distâncias em metros para outros nós. As metragens das distâncias entre as salas foram retiradas das plantas baixas do campus.

4.3 CONSTRUÇÃO DO PROTÓTIPO

Após o mapeamento do campus e a criação do grafo, foi criado um protótipo do aplicativo a ser desenvolvido. O protótipo desenvolvido é de média fidelidade e sem implementação de código-fonte, suficiente para simular as interações dos usuários, definir a paleta de cores da aplicação e o posicionamento dos componentes nas telas. O software utilizado para criação do protótipo foi o *Figma*, uma ferramenta para prototipagem de projetos de *design* de interface.

Figura 16 – Protótipo



A Figura 16 apresenta o protótipo das principais telas do aplicativo *mobile*, a tela principal (1) será responsável em exibir o mapa digital criado no formato SVG a partir das plantas baixas do campus, que possibilita a ampliação, redução e arrastar uma área específica do campus. O mapa exibido na ilustração representa toda infraestrutura da intuição, demonstra como ficará a visualização do mapa quando todos os blocos do campus forem mapeados.

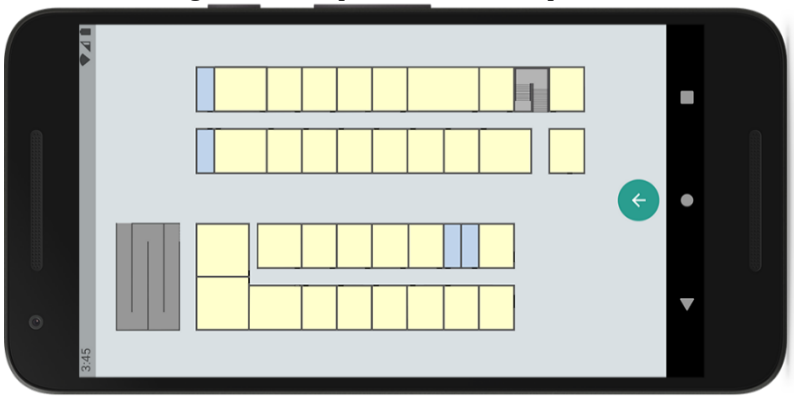
A tela de pesquisa (2) permitirá que os usuários realizem buscas e escolham os locais de origem e destino a serem alcançados, para assim ser calculado o caminho com a menor distância. Após as escolhas, o usuário será redirecionado para a tela principal, onde será exibido o caminho mínimo calculado pelo algoritmo de *Dijkstra* sobre a representação do espaço interno da instituição.

4.4 DESENVOLVIMENTO DA INTERFACE

A interface do aplicativo *mobile* para a exibição do mapa digital e do caminho mínimo foi desenvolvida utilizando o Flutter, um kit de desenvolvimento de interface de usuário para aplicações multiplataforma que utiliza o Dart como principal linguagem de programação. A tela principal do aplicativo é responsável por apresentar o mapa interno do campus construído no formato SVG. Para essa finalidade foi necessário importar o arquivo da ilustração para a aplicação, para isso foi utilizado o *flutter_svg*, uma biblioteca nativa do Dart que permite pintar e exibir ilustrações no formato SVG.

O formato SVG descreve o desenho de forma vetorial através da sintaxe XML, dito isso, ao invés de passar o arquivo bruto como parâmetro de renderização, as informações incluídas no arquivo foram carregadas na aplicação no formato XML. Foi escolhida essa configuração porque o caminho mínimo será desenhado sobre o mapa digital manipulando a árvore de elementos da estrutura XML.

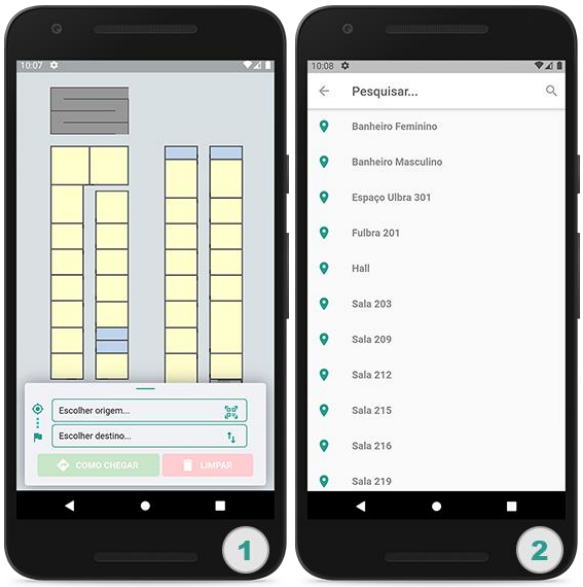
Figura 17 – Mapa renderizado no aplicativo



A Figura 17 apresenta a tela principal do aplicativo *mobile* com a imagem dos blocos dois e três do CEULP/ULBRA, desenhada no formato SVG e compilada na aplicação utilizando a biblioteca *flutter_svg*. Para permitir a interatividade do usuário com o mapa, foi utilizado o *photo_view*, uma biblioteca desenvolvida pela comunidade *open source*, que trasforma imagens estáticas em interativas.

Como citado anteriormente no Flutter tudo são *widgets*, todos elementos da interface do usuário, inclusive o aplicativo em si é um *widget.* A biblioteca *photo_view* foi usada para transformar a imagem renderizada no formato SVG em um *widget* sensível aos gestos dos usuários, permitindo redimensionar, girar e arrastar a ilustração. Com isso, os usuários que utilizarem o aplicativo poderão visualizar uma determinada área com nitidez, sem problemas de distorção ao ampliar ou reduzir o *zoom* do mapa digital.

Figura 18 – Telas de pesquisa



Para permitir a interatividade do usuário no aplicativo, foi desenvolvido um menu deslizante para ser exibido acima do mapa do campus, conforme apresenta a primeira tela da Figura 18. Esse menu é composto por dois campos de textos e dois botões de ação, sendo “Como chegar” e “Limpar”. Ao interagir com qualquer campo de texto o usuário é redirecionado para uma tela de pesquisa conforme a segunda tela da Figura 18. Nessa tela estão todos os pontos de interesse disponíveis, após pesquisar e selecionar um ponto o usuário é redirecionado para tela principal.

O primeiro campo do menu deslizante é responsável pela escolha do ponto de origem e segundo pelo destino. Assim que os dois pontos forem escolhidos, o botão “Como chegar” é

habilitado, ao ser pressionado o caminho mínimo entre esses pontos é exibido no mapa. Com o caminho mínimo sendo exibido, o botão “Como chegar” é desabilitado e o “Limpar” é habilitado, um novo caminho só pode ser calculado somente após o caminho anterior estiver sido limpo. Ambos campos de texto possuem um ícone de ação, sendo para respectivamente selecionar um ponto de origem a partir da leitura de um código QR e um para alternar as posições dos pontos de interesse, ou seja, ao pressionar as informações dos campos de texto origem e destino são trocados de posição.

As principais diferenças entre a localização em ambientes abertos, e a localização em ambientes internos estão associadas às distinções das tecnologias de posicionamentos, o modo em que os dados são modelados e os conceitos básicos de cognição espacial. O GPS é o sistema de posicionamento utilizado em ambientes externos, mas sua precisão em ambientes internos é de 5 a 10 metros, os sinais de satélites em locais fechados são atenuados e sofrem interferências de outras conexões, que impossibilita a troca de informações em tempo real, tornando inviável para posicionamento em ambientes fechados (LEANDRO, 2009).

Um recurso que pode solucionar esse problema são os marcadores visuais, que utilizam a câmera do dispositivo para decodificar o posicionamento, permitindo que o software estime a coordenada através da referência do marcador. O QR-Code (Código de resposta rápida) é um código de barras que armazena diversos tipos de dados entre unidades e valores, com uma capacidade de leitura de alta velocidade (PRESA, 2015). O código QR é um tipo de marcador visual que pode ser facilmente escaneado pela câmera de um dispositivo móvel. Além disso, sua implementação não necessita de equipamentos adicionais, apenas do receptor de leitura e da imagem do código.

Pela facilidade de uso e praticamente ter custo de implantação, a tecnologia de posicionamento escolhida para ser utilizada neste trabalho foi o QR-Code. Os usuários do aplicativo poderão usar a câmera dos seus dispositivos para escanear ponto de origens posicionados em pontos estratégicos do campus.

Para gerar os códigos QR dos pontos de interesse foi utilizado o site QR-Code Generator, que disponibiliza uma plataforma gratuita para a geração e personalização de códigos QR. Os códigos utilizados para estipular a localização em que o usuário está posicionado. O *qrscan* foi a biblioteca Flutter utilizada para adicionar o *scanner* no aplicativo mobile. Através do escaneamento é retornado para o aplicativo o nome do ponto de interesse em que o usuário está posicionado, o parâmetro preenche o campo de origem do menu deslizando, possibilitando que seja escolhido o ponto de destino para o cálculo do caminho mínimo.

4.5 APLICAÇÃO DO ALGORITMO DE DIJKSTRA

O formato JSON foi desenvolvido para ser um padrão de troca de dados legível para humanos e fácil para os computadores compilar e analisar (JSON, 2020). Um formato de texto completamente independente de outras linguagens, sua estrutura é composta por coleções objetos com um conjunto de pares de nomes e valores. Sua estrutura é universal, praticamente todas as linguagens de programação modernas suportam a compilação.

Figura 19 – Node Model

```dart
import 'package:ceulp_indoor/app/models/coordinate.dart';

class Node {
  String name;
  String type;
  double distance;
  Coordinate coordinate;
  Map<Node, double> edges = {};
  Node previous;

  Node(this.name, [this.distance = double.infinity]);

  void setEdge(Node node, double distance) => edges[node] = distance;
}
```

Os dados do grafo foram armazenados em um arquivo JSON, para carregá-los na aplicação foi criado um modelo para os vértices conforme apresenta a Figura 19, esse modelo foi criado para facilitar a manipulação dos dados, sendo responsável leitura, escrita e validação das informações inseridas no grafo. O modelo (*model*) é uma camada da arquitetura MVC (*Model-View-Controller*), este padrão busca dividir as responsabilidades da aplicação, separando os códigos referentes à interface do usuário das regras de negócios, resultando em muitas vantagens relacionadas à reusabilidade, manutenibilidade e compreensão do código fonte.

Para importar os dados relacionados as arestas e vértices do grafo, presentes no arquivo JSON, tornando as informações do mapeamento do campus disponíveis para a aplicação do algoritmo de Dijkstra, foi utilizado um serviço nativo do Flutter denominado *rootBundle* que permite carregar recursos empacotados no projeto. Para o recurso ser reconhecido é necessário especificá-lo no diretório de ativos, no arquivo *pubspec.yaml*, localizado na raiz do projeto

Flutter. O processo de carga é realizado através de um objeto *rootBundle* que identifica o caminho do recurso.

O carregamento dos dados do arquivo JSON foi realizado utilizando o método *loadString()* do objeto *rootBundle* que retorna as informações do ativo em formato de texto. As informações contidas no texto retornado foram convertidas em objetos do modelo vértice. Após a criação dos objetos foi realizada uma segunda interação na *string* no formato JSON, com objetivo de popular as arestas do grafo.

Conforme apresentado na Figura 19, cada objeto que representa um vértice contém um conjunto de arestas, esse conjunto é composto por elementos que contém a referência e distância para os vértices adjacentes. Por esses elementos serem referências foi necessário realizar uma segunda iteração para criar as arestas, para essa finalidade foi utilizado o método *setEdge* da classe modelo vértice, que adiciona as referências e distâncias para os vértices vizinhos. Assim, após a conclusão deste procedimento é retornado um grafo com vários objetos manipuláveis, sendo cada instância, um vértice com suas arestas inclusas.

Figura 20 - Algoritmo de Dijkstra

```dart
import 'dart:collection';
import 'package:collection/collection.dart';
import '../models/node.dart';

class DijkstraService {
  PriorityQueue<Node> _pq;

  List<Node> shortest(Map<String, Node> _graph, String origin, String destiny) {
    _pq = PriorityQueue<Node>();

    _graph[origin].distance = 0;

    _graph.forEach((key, node) {
      if (key != origin) {
        node.distance = double.infinity;
      }

      _pq.add(node);
    });

    while (_pq.isNotEmpty) {
      Node u = _pq.removeFirst();
      u.edges.forEach((v, distance) {
        double newDistance = _graph[u.name].distance + u.edges[v];
        if (newDistance < v.distance) {
          v.distance = newDistance;
          v.previous = u;
          _graph[v.name] = v;
          _pq.add(v);
        }
      });
    }

    Queue<Node> inOrder = Queue<Node>();
    inOrder.addFirst(_graph[destiny]);
    bool stop = false;
    while (!stop) {
      if (_graph[destiny].previous == null) {
        stop = true;
      } else {
        inOrder.addFirst(_graph[destiny].previous);
        destiny = _graph[destiny].previous?.name;
        stop = destiny == origin;
      }
    }
    return inOrder.toList();
  }
}
```

Com o objetivo de encontrar o menor caminho entre a origem e o destino escolhidos pelo usuário, o algoritmo de *Dijkstra* foi implementado na linguagem de programação Dart, conforme apresenta a Figura 20. O algoritmo recebe como parâmetros o grafo criado com as informações dos pontos de interesse carregadas do arquivo JSON, o nome da origem e de destino a ser alcançado.

A estrutura do algoritmo é dividida em três partes, a inicialização, entre as linhas 9 a 19, o bloco de código realiza as seguintes instruções: cria uma fila de prioridade contendo todos os vértices do grafo e adiciona o valor zero para a distância do vértice de origem e infinito para os demais vértices. Entre as linhas 21 a 45 é realizado a etapa de interação, enquanto existir

vértices na fila de prioridades, o primeiro vértice da fila é removido e examinado se a distância mínima partindo da origem é menor que estimativa atual, se viável, o custo mínimo para alcançar esse vértice é atualizado.

Ao final desse procedimento, temos os caminhos mínimos partindo da origem para todos os outros vértices do grafo. Para selecionar apenas o caminho da origem até o destino passado pelo usuário, é realizada uma segunda interação, cada vértice guarda uma variável *previous* que especifica qual foi o vértice utilizado para alcançá-lo. O procedimento realiza uma varredura pelos elementos antecedentes e adiciona cada vértice em uma fila, ao finalizar o procedimento a fila será composta por todos os vértices que foram visitados para chegar ao destino a partir da origem. A finalização, linha 46, retorna uma lista de vértices que compõem o caminho mínimo partindo da origem até o destino.

4.6 DESENHO DO CAMINHO MÍNIMO

Após a aplicação do algoritmo de Dijkstra no grafo construído com as informações dos pontos de interesses localizados no ambiente interno do CEULP/ULBRA. O próximo passo foi desenhar o caminho mínimo através dos vértices e arestas obtidos como resposta. Para isso foi implementado um serviço para desenhar esse caminho no mapa digital. O serviço de desenho utiliza o DOM (*Document Object Model*) para manipular a árvore de objetos da estrutura XML da imagem do campus no formato SVG.

O DOM é uma API (*Application Programming Interface*) independente de plataforma e linguagem de programação, permite que programas e *scripts* manipulem dinamicamente o conteúdo, a estrutura e o estilo de documentos HTML (*HyperText Markup Language*) e XML (W3C, 2020). No DOM, esses documentos são manipulados como uma coleção de nós organizados em uma hierarquia, permitindo a navegação através de uma árvore de objetos, para adicionar ou manipular uma informação específica.

Figura 21 - Serviço de desenho

```dart
import 'package:ceulp_indoor/app/models/node.dart';
import 'package:xml/xml.dart' as xml;

class DrawService {
  static String parse(String documentFile, List<Node> nodes) {
    final document = xml.XmlDocument.parse(documentFile);
    final List<xml.XmlNode> textGroup =
        document.findAllElements('rect').first.parent.children;

    for (int i = 0; i < nodes.length - 1; i++) {
      addPath(textGroup, nodes[i].coordinate.x, nodes[i].coordinate.y,
          nodes[i + 1].coordinate.x, nodes[i + 1].coordinate.y);
    }

    addOriginIcon(textGroup, nodes.first.coordinate.x, nodes.first.coordinate.y);

    addDestinyIcon(textGroup, nodes.last.coordinate.x, nodes.last.coordinate.y);

    return document.toString();
  }
}
```

A Figura 21 apresenta o serviço de desenhar utilizado para inserir o caminho mínimo no mapa digital. O serviço possui uma função *parse*, que tem como parâmetros de entrada o as informações do arquivo SVG em formato de texto e a uma lista de vértices que compõem o caminho mínimo obtido na aplicação do algoritmo de Dijkstra. A função utiliza o DOM para manipular as informações do documento SVG, o resultado desse processamento pode ser compilado ao estado anterior do aplicativo de forma dinâmica.

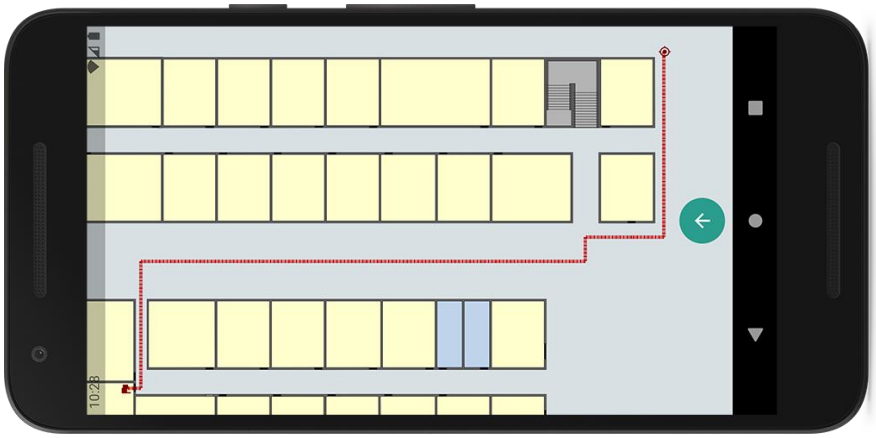
A inicialização desse procedimento, linhas 6 e 7, realiza a leitura do documento SVG e seleciona um grupo de elementos da árvore de objetos XML. A escolha desses elementos é crucial para que o caminho seja exibido acima da última camada de objetos e consequentemente para ser visível pelos os usuários ao interagir com o mapa. A escolha desse conjunto de elementos foi realizada através da análise das camadas da imagem SVG e testes de exibição do caminho no aplicativo.

A iteração do procedimento, linhas 10 a 17, é responsável por adicionar elementos na árvore de objetos. Para desenhar o caminho é utilizado as coordenadas (x, y) de cada vértice percorrido entre a origem e o destino. O procedimento realiza uma iteração sobre a lista de vértices que compõem o caminho mínimo e, para cada conexão é adicionado um elemento *line* no documento. O elemento cria uma linha conectando duas coordenadas, ou seja, para desenhar o caminho é utilizado a coordenada (x_1, y_1) do vértice de partida e a coordenada (x_2, y_2) do vértice de chegada.

Para diferenciar o ponto de origem e o de destino, é adicionado ícones em suas respectivas coordenadas um de foco circular para identificar a origem e uma bandeira para

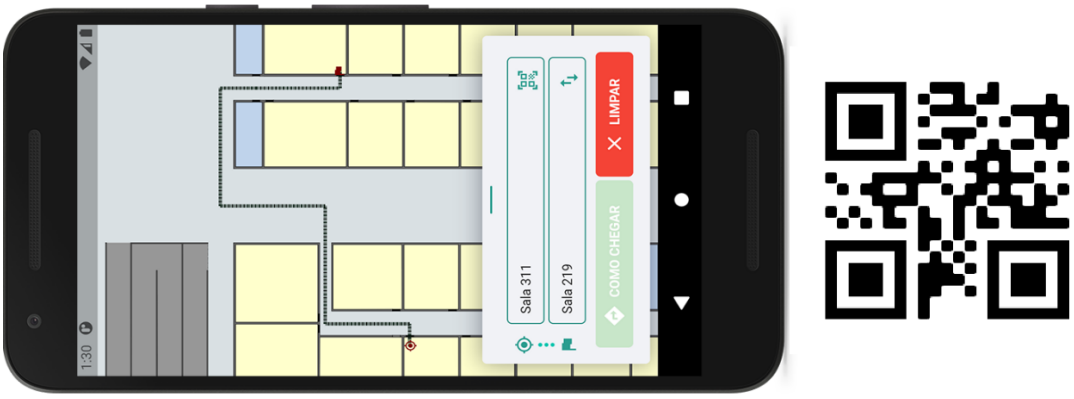
representar o destino. Os ícones foram adicionados na árvore de objetos utilizando o elemento *path* que desenha formas geométricas básicas especificando cada componente dos objetos através de uma sequência de linhas e curvas. A finalização do procedimento, linha 19, retorna um arquivo SVG em formato de texto, com o caminho mínimo desenhado sobre os as construções do campus.

Figura 22 – Caminho entre o “Hall” e a “Sala 314”



A Figura 22 exibe o mapa digital com o caminho de menor distância entre o “Hall” e a “Sala 314”, parâmetros utilizados respectivamente como origem e destino na aplicação do algoritmo de Dijkstra no grafo construído. Sendo que a origem localizada no canto superior direito do mapa, representada por um ícone de foco circular que indica a posição atual do usuário e o destino localizado no canto inferior esquerdo no mapa representado por uma bandeira indica o ponto de interesse a ser alcançado. Por fim, o caminho é uma linha que indica o percurso a ser seguido entre os dois pontos.

Figura 23 – Caminho entre a “Sala 311” e a “Sala 219”



A Figura 23 exibe o mapa digital com o caminho de menor distância entre o ponto de interesse “Sala 311” e “Sala 219”, sendo que a origem foi adicionada a partir de um QR-Code, escaneado pela câmera de um dispositivo *mobile*. Os QR-Code’s ficarão disponíveis em pontos estratégicos do campus, assim facilitando a localização de usuários que desconhecem o ponto atual em que está posicionado na infraestrutura de instituição.

Após a finalização do desenvolvimento do aplicativo *mobile*, foi então realizados testes em todos os caminhos possíveis do mapa digital, a fim de verificar a autenticidade das distâncias que compõem o caminho mínimo. Qualquer peso atribuído incorretamente a uma aresta interfere diretamente no calculo final da distância do caminho percorrido entre uma origem e destino.

Os testes foram realizados manualmente verificando todos os caminhos possíveis a todos pontos de interesse, para isso foram verificadas as distâncias dos percursos, considerando que para um percurso está correto, a sua distância para ir e voltar de uma origem para um destino sejam a mesma. Logo, o valor da aresta atribuída para alcançar um vértice refere ao mesmo valor para retornar ao vértice de origem. Já o percurso entre uma origem e destino que possuem mais de um caminho mínimo com o mesmo peso, foi analisado que o algoritmo de Dijkstra considera o primeiro que foi encontrado na aplicação do cálculo de distância mínima.

5 CONSIDERAÇÕES FINAIS

Este trabalho foi desenvolvido com o propósito de auxiliar alunos, funcionários e visitantes a encontrarem pontos de interesses localizados no ambiente interno do Centro Universitário Luterano de Palmas. Para essa finalidade, foram executadas etapas que envolveram diversos objetivos, desde o mapeamento da infraestrutura interna do campus até o desenho do caminho mínimo sobre um mapa digital.

Através da utilização de conceitos de estruturas de dados, foi possível desenvolver um aplicativo *mobile* que informa o caminho de menor distância entre pontos localizados no ambiente interno do CEULP/ULBRA. Apesar deste trabalho ser realizado no contexto universitário, as técnicas e metodologias aplicadas, com as devidas adaptações podem ser utilizadas em diferentes situações envolvendo ambientes internos.

Os objetivos específicos do trabalho, que eram mapear a área interna do campus, as coordenadas dos pontos de interesse e rotas entre as coordenadas para a criação de um grafo. E desenvolver uma interface *mobile* que informe o menor caminho entre uma origem e destino calculado através da aplicação do algoritmo de Dijkstra em uma representação digital do ambiente interno da instituição foram atingidos.

O objetivo de mapear a área interna do Centro Universitário Luterano de Palmas foi alcançado através do estudo de plantas baixas da infraestrutura do campus. Sendo a etapa que demandou mais atenção na execução da metodologia adotada, porque além de ser essencial seguir as proporções reais da instituição, de acordo com as medidas disponíveis nas plantas baixas. As coordenadas e distâncias dos relacionamentos entre os pontos de interesse foram analisados e testados, dado que qualquer informação mapeada incorretamente influencia diretamente na aplicação do algoritmo de Dijkstra no grafo construído.

O objetivo de desenvolver uma interface *mobile* foi alcançado utilizando o Flutter e a linguagem de programação Dart. Com essas ferramentas torna possível disponibilizar o aplicativo para plataformas iOS e Android com todos os recursos compilados de forma nativa. Além disso, o código-fonte é centralizado em um único projeto otimizando a manutenção e adição de novas funcionalidades na aplicação.

Em vista disso e através dos resultados apresentados na seção anterior, foi o objetivo principal deste trabalho foi atingido através do desenvolvimento de um aplicativo *mobile* utilizado e algoritmo de Dijkstra. Contendo funcionalidades que permite que usuários encontrem seus pontos de interesse mais facilmente selecionando uma origem e destino no aplicativo para exibir o menor caminho sobre um mapa digital construindo com imagem no formato SVG que representa a área interna do CEULP/ULBRA.

Para trabalhos futuros, novas funcionalidades podem ser adicionadas na aplicação, como por exemplo: possibilidade de mudança de piso, um seletor de nível para alternar a visualização dos andares em um prédio. Recursos de acessibilidade para exibir um caminho alternativo para deficientes físicos, por exemplo, calcular a distância mínima apenas nos caminhos que contém rampas, calçadas ou banheiros adaptados. Realizar integração com sistemas da instituição para disponibilizar as matérias matriculadas no semestre atual, para assim, facilitar a localização de uma sala de aula do dia atual.

REFERÊNCIAS

ADAIXO, M; AMADOR, G. N. P.; GOMES, A. J. P. **Algoritmo de Dijkstra com Mapa de Influência de Atratores e Repulsores**, Covilhã, Portugal, 2012. Disponível em: <https://scitecin.isr.uc.pt/Proceedings/Papers/VideoJogos/10.pdf>. Acesso em: 20 abr. 2020.

ALMEIDA, R. R; MOREIRA, J. P. **Tecnologias para o Desenvolvimento de Aplicações Multiplataforma**: Um estudo sobre os frameworks React Native e Flutter, Porto Alegre, Rio Grande do Sul, 2019. Disponível em: <http://raam.alcidesmaya.com.br/index.php/projetos/article/view/54/54>. Acesso em: 29 abr. 2020.

BARBALHO, L. V.; GRAÇA, A. J. S. **A evolução da orientação urbana:** Dos guias de ruas à Geoweb. Rio de Janeiro, 2017. Disponível em: <http://www.cartografia.org.br/cbc/trabalhos/3/750/CT03-91_1506816128.pdf>. Acesso em: 9 jun. 2020.

CABRAL, I. P. S. et al. **Padrão SVG em Aplicações Ambientais**. XII Simpósio Brasileiro de Sensoriamento Remoto, Goiânia, p. 2075-2082, 2005. Disponível em: <http://www.cin.br/~sbgames/proceedings/files/Uma%20Proposta%20de%20Arquitetura%20de%20Software.pdf>. Acesso em: 23 abr. 2020.

CEULP. **Portal Acadêmico do Centro Universitário Luterano de Palmas**, CEULP/ULBRA, 2020. Disponível em: <http://ulbra-to.br/>. Acesso em: 2 mai. 2020.

CORMEN, T. H. et al. **Algoritmos**: Teoria e Prática. Tradução de Arlete Simille Marques. 3º. ed. Rio de Janeiro: Elsevier, 2012. 944 p.

COSTA, P. P. **Teoria dos Grafos e suas Aplicações**. 2011. 77 p. Dissertação (Mestrado em Matemática Universitária) - Universidade Estadual Paulista Júlio de Mesquita Filho, Instituto de Geociências e Ciências Exatas, Rio Claro, 2011. Disponível em: <https://repositorio.unesp.br/bitstream/handle/11449/94358/costa_pp_me_rcla.pdf>. Acesso em: 18 abr. 2020.

COSTA, R.V. **Grafos:** Algumas aplicações a nível médio. 2017. 79 p. Dissertação (Mestrado Profissional em Matemática) – Universidade de Brasília, Brasília, 2017. Disponível em: <https://repositorio.unb.br/bitstream/10482/24847/1/2017_RodrigoVazCosta.pdf>. Acesso em: 28 mar. 2020.

DART. **Dart programming language**: Dart documentation, 2020. Disponível em: <https://dart.dev/>. Acesso em: 29 abr. 2020.

EISENBERG, J. David; Royds, A. B. **SVG Essentials**: Producing Scalable Vector Graphics with XML. O′Reilly, 2002. 368 p.

FLUTTER. **Flutter:** Beautiful native apps in record time, Flutter documentation, 2020. Disponível em: <https://flutter.dev/>. Acesso em: 29 abr. 2020.

HUANG, Haosheng; GARTNER, Georg. **A survey of mobile indoor navigation systems**. In: Cartography in Central and Eastern Europe. Springer, Berlin, Heidelberg, 2009. p. 305-319. Disponível em: <https://www.researchgate.net/publication/226163713_A_Survey_of_Mobile_Indoor_Navigation_Systems>. Acesso em: 26 abr. 2020.

JSON. **JavaScript Object Notation**, 2020. Disponível em: <https://www.json.org/>. Acesso em: 17 nov. 2020.

JUNIOR, J. A. G. A.; PONTES, H. L. J.; ALBERTIN, M. R. **Desenvolvimento de um software educacional para apoio ao ensino de localização e roteirização**. Exacta, v. 17, n. 3, p. 81-99 2019. Disponível em: <https://doi.org/10.5585/exactaep.v17n3.8444>. Acesso em: 9 jun. 2020.

JUNIOR, V. P. S. **Potencial da linguagem SVG Scalable Vector Graphic para visualização de dados espaciais na internet:** Estudo de caso. 2004. 110 p. Dissertação (Mestrado em Engenharia Civil) - Programa de Pós-Graduação em Engenharia Civil - Universidade Federal de Santa Catarina, Florianópolis, 2004. Disponível em: <https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/87447/214402.pdf>. Acesso em: 15 jul. 2020.

JURKIEWICZ, S. **Grafos:** Uma Introdução. São Paulo: OBMEP, 2009. 119 p. Disponível em: <http://www.obmep.org.br/docs/apostila5.pdf >. Acesso em: 16 abr. 2020.

KUAN, J. **Learning Highcharts 4**: Design eye-catching and interactive JavaScript charts for your web page with Highcharts, one of the leading tools in web charting. Packt Publishing, 2015. 478 p.

LEANDRO, Diuliana. **Investigação do posicionamento GPS em ambientes internos com o auxílio do efeito de multicaminho**. 2009. p. 686-687. Dissertação (Mestrado em Ciências Geodésicas) - Programa de Pós-Graduação em Ciências Geodésicas, Setor de Ciências da Terra, Universidade Federal do Paraná, Curitiba, 2009. Disponível em: <https://docplayer.com.br/72030894-Universidade-federal-do-parana-diuliana-leandro-investigacao-do-posicionamento-gps-em-ambientes-internos-com-o-auxilio-do-efeito-de-multicaminho.html>. Acesso em: 15 nov. 2020.

LEE J. et al. **Open Geospatial Consortium**: OGC IndoorGML, Technical report, Open Geospatial Consortium. 2014. Disponível em: <https://oceanbestpractices.net/bitstream/handle/11329/1074/14-005r3_OGC_IndoorGML.pdf>. Acesso em: 27 abr. 2020.

MAPS PLATFORM. **Google Maps Platform:** Documentation, Google Developers, 2020. Disponível em: <https://developers.google.com/maps/documentation/>. Acesso em: 12 jun. 2020.

MENDONCA, P. F.; KESTRING, F. B. F; SILVA, F. P. **Um estudo sobre algoritmos para roteirização.** Revista Eletrônica Científica Inovação e Tecnologia, Edição Especial Cadernos

Matemática, E – 5122, 2017. Disponível em: <https://periodicos.utfpr.edu.br/recit/article/view/e-5122/pdf_1>. Acesso em: 28 mar. 2020.

NEGRI, M. A. S. **Caminhos em um grafo e o algoritmo de Dijkstra**. 2017. 75 p. Dissertação (Mestrado Profissional em Matemática) – Universidade Federal de Santa Catarina, Centro de Ciências Físicas e Matemáticas, Programa de Pós Graduação em Matemática, Florianópolis/SC, 2017. Disponível em: <https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/183409/349816.pdf>. Acesso em: 28 mar. 2020.

OHRT, J.; TURAU, V. **Simple indoor routing on svg maps**. International Conference on Indoor Positioning and Indoor Navigation. IEEE, p. 1-6, 2013. Disponível em: <https://ieeexplore.ieee.org/document/6851432>. Acesso em: 3 set. 2020.

OSTA, C. Z. **Análise comparativa de algoritmos de caminho de custo mínimo aplicado em rede de fibra óptica**. 2019. 47p. Trabalho de Conclusão de Curso (Graduação) – Curso de Sistemas de Informação, Centro Tecnológico da Universidade Federal de Santa Catarina, Florianópolis/SC, 2019. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/202485/Monografia_Christian_Zirke_Osta.pdf>. Acesso em: 3 ago. 2020.

PRESA, Eduardo. **Proposta de um sistema de localização interna para o ambiente universitário**. 2015. 102p. Trabalho de Conclusão de Curso (Graduação) – Curso de Tecnologias da Informação e Comunicação, Universidade Federal de Santa Catarina Campus Araranguá, Araranguá, 2015. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/132187/Eduardo_Presa.pdf >. Acesso em: 15 nov. 2020.

PRESTES, E. **Introdução à Teoria dos Grafos**. Universidade Federal de Santa Catarina, Santa Catarina, Rio Grande do Sul, 2016. Disponível em: <http://www.inf.ufrgs.br/~prestes/Courses/Graph%20Theory/Livro/ParteLivroGrafos.pdf>. Acesso em: 28 mar. 2020.

SAUTOY, M. D. **O enigma resolvido há 300 anos pelo matemático Leonard Euler e que hoje nos permite navegar na internet**. BBC, 2018. Disponível em: <https://www.bbc.com/portuguese/geral-44157282>. Acesso em: 11 abr. 2020.

SOUZA, R. F. **Resolução de problemas via teoria de grafos**. 2014. 48 p. Dissertação (Mestrado Profissional em Matemática) – Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, São Carlos, 2014. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55136/tde-06072015-103319/publico/Dissertacao_RenatoFdeSouza_Revisada.pdf >. Acesso em: 18 jul. 2020.

TAROUCO, F. **A metrópole comunicacional e a popularização dos apps para dispositivos móveis**. Seminário Internacional de Pesquisa em Comunicação –Epistemologia e desafios da pesquisa no campo da comunicação, p. 1-15, 2013. Disponível em: <http://coral.ufsm.br/sipecom/2013/wp-content/uploads/gravity_forms/1-

997169d8a192ed05af1de5bcf3ac7daa/2013/09/A-metropole-comunicacional-o-e-a-popularizacao-dos-apps.pdf>. Acesso em: 9 jun. 2020.

W3C. **World Wide Web Consortium:** Documentation, 2020. Disponível em: <https://www.w3.org/>. Acesso em: 10 nov. 2020.