Uma Visão Acerca do uso da Teoria dos Grafos para a Resolução de Labirintos

Aaron Rodrigues[1], **Felipe Maciel**[2], **Lucas Mendonça**[3]

[1]IESB – Instituto de Educação Superior de Brasília (IESB) – Brasília – DF – Brazil

[2]Departamento de Ciência da Computação – Instituto de Educação Superior de Brasília (IESB) Brasília, Distrito Federal

aaron.rodrigues@iesb.edu.br, felipe.maciel@iesb.edu.br, lucas.s.mendonca@iesb.edu.br

Abstract. *This article aims one possibility on the use of Graph Theory, allied to the study of computer algorithms, to the solution of unknown mazes. During the research, it was about the use of APIs, Depth First Search (DFS) and Breadt First Search (BFS) algorithms, the algorithm developed to solve the problem shown in next steps and the study of Python programming language; besides the methodology adopted, the theoretical references obtained and the conclusions acquired from the results of this research.*

Resumo. *O presente artigo trata de uma das possibilidades do uso da Teoria dos Grafos, aliada aos estudos de algoritmos de computador, para a resolução de labirintos desconhecidos. Durante o estudo, tratou-se de chamadas de APIs, dos algoritmos de Busca em Profundidade DFS Depth First Search e Busca em Largura BFS (Breadt First Search), da aplicação utilizada para solucionar o problema evidenciado nas próximas etapas da pesquisa e do estudo da linguagem de programação Python; além da metodologia utilizada, dos referenciais teóricos obtidos e das conclusões adquiridas a partir dos resultados da pesquisa.*

1. Introdução

Sabe-se que labirintos são estruturas que possuem passagens e cruzamentos, criados pelo ser humano com o intuito de confundir quem os adentre, em que sua origem é marcada há mais de 4.000 anos e que eram utilizados, especialmente, como forma de armadilha, devido à sua complexidade.

Além disso, segundo Araldi (2020), a Teoria dos Grafos é um ramo dos estudos da Matemática e da Computação que analisa os relacionamentos entre objetos em um determinado conjunto.

De acordo com Pamplona (2008, p.27), uma possível aplicação da Teoria dos Grafos se encontra na solução de labirintos, em que podem ser usados algoritmos de busca para estabelecer um método que encotre a saída de um labirinto, previamente desconhecido, a partir de uma entrada qualquer.

Portanto, o presente artigo tem como intuito, diante do contexto histórico apresentado e, motivado pelo estudo da Teoria dos Grafos e pelas famosas competições chamadas *Micromouse Competition*, em que mini-robôs são utilizados em desafios para

encontrarem a saída de labirintos de forma rápida, as etapas de construção de um algoritmo que mostre o caminho mais otimizado possível para a superação de um labirinto, assim como a metodologia proposta, o referencial teórico que a envolve e as conclusões obtidas a partir da pesquisa em questão.

2. Objetivos

2.1. Objetivo Geral

Dentre os objetivos do artigo em foco, o principal é construir um aprendizado sobre algoritmos, Teoria dos Grafos e a linguagem de programação Python, além de como eles, juntos, podem ser utilizados para superar os desafios apresentados anteriormente que envolvem, especificamente, a resolução de uma série de labirintos, desconhecidos pelo algoritmo desenvolvido, com o menor número possível de movimentações feitas pelo programa, motivados pelas competições de robôs citadas anteriormente.

2.2. Objetivos Específicos

Dentre os objetivos específicos do artigo em evidência estão, principalmente:

- A construção de um algoritmo na linguagem de programação Python que seja capaz de chamar uma API de forma eficiente para gerar labirintos que serão utilzados para a testagem e avaliação do algoritmo submetido;

- Desenvolver algoritmos de busca em profundidade (DFS) e em largura (BFS) na linguagem de programação Python que sejam capazes de mapear todo o grafo disponibilizado para, então, possibilitar a tomada de decisão pelo caminho mais curto entre uma entrada qualquer e a saída do labirinto;

- Testar, com ajuda da API disponibilizada, o algoritmo desenvolvido para garantir sua eficácia e desempenho;

- Expor conclusões tomadas a partir dos resultados expostos durante a pesquisa em foco, acerca do uso da Teoria dos Grafos para a solução de labirintos previamente desconhecidos pelo algoritmo implementado.

3. Metodologia

No artigo em questão, tendo como base o estudo da Teoria dos Grafos e de algoritmos, além dos trabalhos correlatos utilizados para embasar a pesquisa, será proposta uma solução computacional para o problema em foco, evidenciado anteriormente.

Em relação à criação dos labirintos, tanto para testagem, quanto para avaliação do algoritmo submetido, será utilizada uma API (Interface de Programação de Aplicações) externa disponibilizada pelo avaliador por meio da plataforma GitHub.

Foi escolhida a linguagem de programação Python, por conta de sua forte tipagem e sua qualidade multiparadigma, para o desenvolvimento da aplicação em questão, que envolve a chamada da API para criação do grafo a ser percorrido e o desenvolvimento de um algoritmo de busca para o percorrer e auxiliar o usuário a encontrar a sua saída.

Além disso, serão criados diversos labirintos com base na API em questão, com a finalidade de testar os algoritmos de busca o maior número de vezes possíveis para

garantir o funcionamento da aplicação como um todo e, assim, assegurar sua performence e exatidão.

Em síntese, a metodologia escolhida para a solução do problema em evidência se baseia, primeiramente, na verificação de trabalhos correlatos para melhor embasamento da pesquisa e, após isso, na escolha da linguagem de programação para o desenvolvimento do algoritmo que, por sua vez, envolve o consumo de uma API e a implementação de algoritmos de busca e, então, na ampla testagem da aplicação desenvolvida para garantir a sua acurácia e performance.

4. Referencial Teórico

4.1. Entendendo a Teoria dos Grafos

A Teoria dos Grafos teve sua origem situada na história da Matemática, por volta do século XVIII, com a resolução do matemático Euler do problema das Pontes de Königsberg, publicada em 1736. Contudo, a teoria formal acerca de grafos foi desenvolvida, de fato, por volta do século XX (Melo, 2014, p. viii).

Entende-se por grafo qualquer estrutura que possua um conjunto finito e não vazio de construções denominadas vértices interligados por um outro conjunto de pares não ordenados de vértices, denominados arestas. Ou seja, um grafo qualquer pode ser visto como um conjunto finito de pontos, denominados vértices, unidos por ”pontes” denominadas arestas.

Grafos são, usualmente, representados por diagramas, em que os vértices correspondem a pontos em um plano e as arestas equivalem aos arcos que ligam os vértices equivalentes. A imagem resultante gerada por essas uniões não possui algum significado geométrico, uma vez que seu propósito é, exclusivamente, representar relações de adjacência entre os vértices do grafo em questão (Melo, 2014, p. 1).

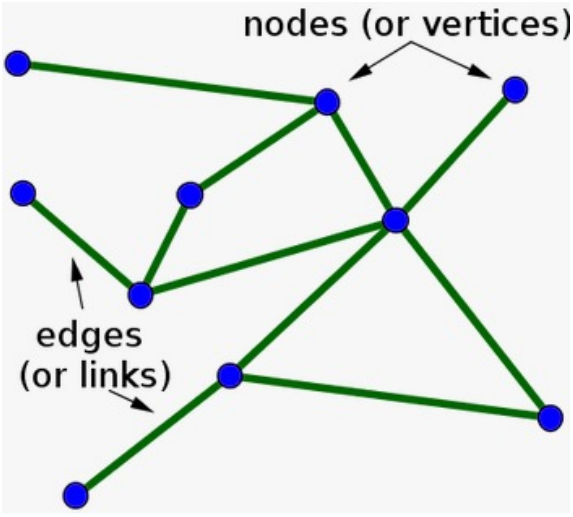

Figure 1. Representação visual de um grafo. Fonte: https://manoelcamilo.wixsite.com/analise/single-post/2017/06/25/desenho-de-grafos-fazendo-nuvens-de-dados-ganharem-significado-visual

O uso da Teoria dos Grafos auxilia o entendimento de situações habituais como: redes de computadores ou de comunicações; árvores genealógicas; situações da Química

Orgânica; etc (Melo, 2014, p. viii). Além disso, é possível, com o uso da Teoria dos Grafos, a resolução de problemas computacionais, como o previsto na pesquisa em questão, que a utiliza aliada ao uso de algoritmos de computador para encontrar a saída de um labirinto desconhecido de forma eficiente.

4.2. A linguagem de programação Python

A linguagem de programação Python, escolhida para o desenvolvimento do algoritmo para a resolução do problema em questão, foi criada pelo matemático e programador holandês Guido Van Rossum no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI), por volta de 1990. A linguagem foi criada com base em outra já existente na época, denominada ABC, e tinha como foco, primeiramente, acadêmicos e pesquisadores das áreas de Física e Engenharia. Atualmente, está em sua terceira versão, lançada em 2008, e é uma das linguagens de programação mais usadas pelo mundo (Borges, 2014, p. 15).

A escolha dessa linguagem de programação se deu por sua forte e dinâmica tipagem e sua característica multiparadigma e multiplataforma. Ou seja, sua tipagem forte e dinâmica se dá devido às suas variáveis poderem armazenar qualquer tipo de dados (inteiro, float, etc.); sua característica multiparadigma se deve ao fato de sustentar diversos paradigmas do desenvolvimento (orientado a objetos, imperativo, interpretado, etc.); e sua qualidade multiplataforma diz respeito à sua plena operação em qualquer sistema operacional sem que sejam necessárias alterações no código-fonte da plataforma (Andrade, 2020).

Dentro da linguagem de programação escolhida, existem bibliotecas, também chamadas de pacotes, criadas para facilitar o desenvolvimento de novos algoritmos. Essas bibliotecas são, de acordo com Andrade (2023), arquivos simples de código, encapsulados, que são repetidos com frequência considerável para serem usados futuramente em diferentes aplicações. Com isso, é possível ao usuário criar suas próprias bibliotecas, utilizar as bibliotecas padrão da linguagem ou bibliotecas de terceiros.

Durante o desenvolvimento do algoritmo que solucionará o problema em questão, foram utilizadas algumas bibliotecas presentes na linguagem Python, dentre elas estão:

- Requests: simplifica a interlocução com a internet. Com ajuda dela, é possível enviar e receber informações da rede, além da interação com APIs de forma prática, por exemplo;

- Urllib3: ajuda a lidar com redirecionamentos HTTP e a validar a API utilziada no projeto;

- Collections(deque): utilizada em generalizações de pilhas e filas. Oferece suporte para inserções e retiradas de elementos dessas estruturas de dados de forma segura;

4.3. API - *Application Programming Interface*

De acordo com Jacobson, Brail e Woods (2012, p.4), considera-se que uma API (*Application Programming Interface* - Interface de Programação de Aplicações) é uma interface que possibilita a comunicação entre diferentes plataformas por meio de protocolos distintos para a construção de diferentes aplicações.

Uma API funciona como uma espécie de ”correio”, que recebe requisições de um determinado usuário e as transmite a um servidor e, após isso, são processadas por ele e devolvidas ao cliente. É comunmente utilzada por profissionais da tecnologia da informação para que sejam realizadas integrações entre diversas plataformas e novas tecnologias sejam criadas (Krieger, 2021).

Existem, de acordo com Krieger (2021) basicamente, 4 tipos de APIs:

- APIs públicas: livres para todos os indivíduos que quiserem acessar uma determinada aplicação;

- APIs privadas: possuem o seu uso limitado a apenas usuários previamente autorizados pelo dono da aplicação;

- APIs de parceiros: utilizadas, majoritariamente, para facilitar a integração e comunicação entre usuários;

- APIs compostas: comunmente utilizadas em arquiteturas de microsserviços, em que são dados ao usuário diferentes dados para a execução de determinados encargos.

Desse modo, empregar APIs para o desenvolvimento de aplicações computacionais é vantajoso, pois provém uma maior flexibilidade de fornecimento de conteúdos ao usuário e a melhora da arquitetura do sistema (Jacobson, Brail e Woods, 2012, p. 15). Assim, entende-se que o emprego dessa interface, por parte do avaliador, foi apoiada em critérios técnicos e objetivos, o que possibilitou um direcionamento mais eficiente no desenvolvimento da aplicação que solucionará o problema evidenciado.

4.4. DFS - *Depth First Search*

Em um algoritmo de busca em profundidade (DFS - *Depth First Search*), o objetivo principal é visitar todos os vértices do grafo e computá-los na ordem em que são acessados e, assim, buscar o seu nó mais ”fundo”. O algoritmo em foco ajuda o usuário a entender o grafo em que está trabalhando, reunindo informações importantes, como: seu ”formato”, denotado pelo arranjo de seus vértices e arestas; e conhecimentos que podem ser gerados a partir da enumeração de seus vértices (Feofiloff, 2019).

Nesse algoritmo, a exploração começa em um vértice inicial qualquer e, caso posua vértices não explorados em sua lista de adjacências (ligados a ele por meio de arestas), o algoritmo os visita e parte para os próximos que não foram visitados, seguindo a mesma regra e armazenando os nós visitados em uma estrutura de dados denominada Pilha LIFO (*Last In, First Out*). Caso o algoritmo pare em algum vértice em que todos os que estão em sua lista de adjacências já foram visitados, ele ”anda para trás”, removendo o último elemento alocado em sua pilha de recursividade e voltando ao vértice alocado anteriormente até chegar a algum que possua um elemento não visitado em sua lista de adjacências para, então, prosseguir com a exploração. Assim, o processo continua até que todos os vértices alcançáveis tenham sido visitados pelo algoritmo (Brandão, 2009, p. 8).

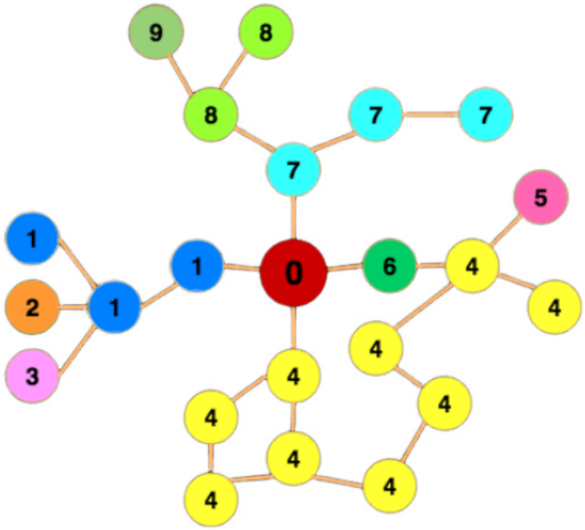

Figure 2. Representação visual de um DFS em operação. Fonte: https://blog.pantuza.com/artigos/busca-em-profundidade.

4.5. BFS - *Breadth First Search*

Em um algoritmo de busca em largura BFS (*Breadth First Search*), o objetivo permanece o mesmo do DFS. Contudo, seu uso é vantajoso, principalmente, quando se deseja obter o menor caminho, em quantidade de vértices percorridos, entre dois vértices quaisquer do grafo em questão (Desenvolvendo Software, 2023).

Em um BFS, um vértice pai qualquer é primeiramente acessado, em seguida todos os vértices que estão em sua lista de adjacências também são e, após isso, todos os vértices vizinhos dos seus vizinhos são igualmente visitados, e assim em diante. O algoritmo aloca os vértices percorridos em sequência, seguindo a ordem em que são acessados (Feofiloff, 2019). Para garantir o pleno funcionamento dessa operação, a aplicação utiliza a estrutura de dados Fila FiFo (*First in, First out*) para armazenar os vértices visitados, em que o primeiro elemento a ser inserido à fila é, sempre, o primeiro a ser retirado dela. Dessa maneira, o grafo é percorrido do início ao final e cada vértice é percorrido no máximo uma vez (Sidinei, Gomes e Santos).

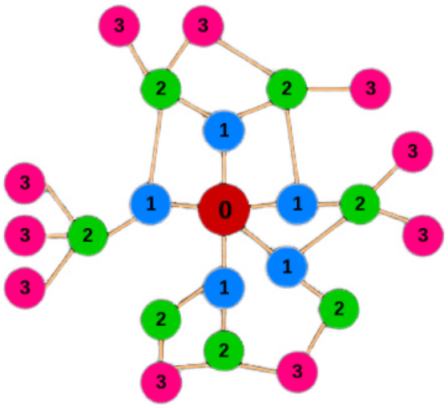

Figure 3. Representação visual de um BFS em operação. Fonte: https://blog.pantuza.com/artigos/busca-em-largura.

5. Desenvolvimento

5.1. A chamada da API

O projeto foi executado com base na API disponibilizada no link (https://gtm.delary.dev/) para viabilizar a compreensão do grafo e as movimentações possíveis dentro do labirinto em questão, que será previamente desconhecido pelo algoritmo de busca utilizado. A API fornecerá três *endpoints*:

- /iniciar: possibilita ao usuário começar a exploração do labirinto;

- /movimentar: disponibiliza ao usuário as movimentações possíveis dentro do labirinto e o movimenta entre os nós do grafo;

- /valida_caminho: imprime na tela do usuário a sequência de movimentos executada, verifica se efetivamente vai do início ao fim do labirinto e a valida.

5.2. O desenvolvimento do algoritmo

O programa faz uso de três bibliotecas: *requests* pra realizar a chamada da API; *urllib3* para a validação da API e retirada do aviso de "insecureRequestWarning (Unverified HTTPS request is being made to host 'gtm.delary.dev')", em que é adicionado um certificado de verificação ao algoritmo; e *Collections (deque)*, com a finalidade de aprimorar a segurança das operações de inserção e retirada de elementos de estruturas de pilhas e filas.

Em geral, foram criadas 4 funções durante o desenvolvimento da aplicação:

- Obter_labirinto(): realiza uma chamada para API com "requests.get('https://gtm.delary.dev/labirintos', verify=False)", que obtém o nome dos labirintos disponiveis na API, verifica sua validação e retorna um "arquivo.json()", que exibe o nome de todos os labirintos;

- Iniciar_labirinto(nome_labirinto, id_jogador): inicia o labirinto. Os parâmetros passados precisam ser: o nome do labirinto que será percorrido e um ID qualquer para o jogador. Essa função envia o nome do labirinto e o ID selecionado para a URL que inicia o jogo, faz a sua verificação e retorna o "iniciar.json()", e a resposta devolvida pelo algoritmo será composta da sua posição atual, uma flag de início marcada como verdadeira, outra de final marcada como falso e os possíveis movimentos a partir daquela posição em específico.

- Movimentar_labirinto(nome_labirinto,id_jogador, nova_posicao): sua função é mover o usuário dentro do labirinto, em que os parâmetros passados são: o nome do labirinto, o ID do jogador e o número da nova posição no labirinto. Ele envia as requisições, verifica se elas estão válidas e retorna um "movimentar.json()", que exibe o número da nova posição com a flag "posição atual", se o usuário está no início ou no final do grafo e os movimentos possíveis para as próximas posições.

- Validar_caminho(nome_labirinto, id_jogador, movimentos): Será a requisição final, em que são passados como parâmetro: o nome do labirinto, o ID do jogador e uma lista dos movimentos que foram utilizados pelo algoritmo para percorrer o caminho dentro do grafo. Essa função só é chamada ao final das movimentações, no momento em que a saída é encontrada. Então, será solicitado à API se o caminho percorrido pelo algoritmo é válido.

5.2.1. O emprego do DFS

Para o reconhecimento e análise prévios do labirinto que será percorrido, foi adotado, primeiramente, um algoritmo DFS (*Depth First Search*), que proporciona o reconhecimento do ”formato” do grafo, denotado pela disposição de vértices e arestas em sua estrutura (Feofiloff, 2019). As movimentações do algoritmo serão armazenadas em uma pilha de recursividade e exibidas na tela do usuário com os últimos vértices visitados, que serão armazenadas em uma estrutura de Lista de Adjacências, ideal para armazenar grafos e suas relações de adjacências de forma eficaz (Feofiloff, 2019).

Apesar de sua implementação permitir ao usuário reconhecer grande parte do grafo a ser percorrido, incluindo os vértices de entrada e de saída, com a sua aplicação pura e isolada não é possível encontrar o menor caminho entre eles e, para realizar essa tarefa, será aplicado um algoritmo BFS em seguida, que é capaz de encontrá-lo de forma eficiente.

5.2.2. O emprego do BFS

Com o emprego do algoritmo DFS, foi possível realizar o mapeamento do grafo e descobrir o ”formato” do labirinto em questão, com seus vértices armazenados em uma lista de adjacências, além de encontrar o nó que representa a sua saıda. A partir daí, foi adotado um algoritmo BFS (*Breadt First Search*), executado com base na lista de adjacências obtida como resultado do DFS, com o objetivo de encontrar a saída do grafo com o menor número de movimentações possıveis, uma vez que sua implementação possibilita que isso aconteça (Desenvolvendo Software, 2023).

Assim, suas movimentações serão armazenadas em uma estrutura de dados denominada Fila, em que o nó inicial será inserido primeiramente, após eles os seus vizinhos serão e, em seguida os vizinhos dos vizinhos, até que a saıda seja encontrada e, quando for, é mostrada na tela do usuário o menor caminho entre o inıcio e o final do labirinto. Então, dessa maneira será possıvel, por parte da aplicação desenvolvida, encontrar o caminho mais otimizado entre o vértice de entrada e o vértice de saıda do labirinto.

6. Resultados

Durante o desenvolvimento do projeto em questão, houve dificuldades, por parte da aplicação, em lidar com os labirintos de maiores proporções. Durante o funcionamento do algoritmo DFS, houve um aumento considerável do tempo de execução da aplicação como um todo devido à uma perda de eficiência do algoritmo, gerado por dificuldades em ter que retirar e inserir grandes quantidades de vértices de sua pilha de recursividade em curtos espaços de tempo, em situações de grafos que possuíam ”ramos” com grandes quantidades de vértices que seguiam em uma direção específica.

Já durante a execução do algoritmo BFS, ele se mostrou eficiente em encontrar a saıda dos grafos testados com o menor número de movimentações possıveis, principalmente quando ela se encontrava em um nó próximo à entrada do labirinto, pois sua atuação vinha seguida ao DFS, que já havia mapeado grande parte dos grafos, mas era ineficiente em encontrar o caminho mais otimizado possıvel entre a entrada e a saıda dos labirintos.

Segue, abaixo, tabela com os tempos de execução da aplicação desenvolvida registrados em todos os labirintos de teste disponíveis na API fornecida:

LABIRINTOS	TEMPO DE EXECUÇÃO
Maze-sample	4 segundos
Maze-sample2	5 segundos
Medium-maze	16 segundos
Large-maze	3 minutos e 3 segundos
Very-large-maze	> 10 minutos (timeout)

Figure 4. Tabela de tempo de execução do algoritmo nos diferentes labirintos presentes na API.

7. Conclusão

A partir do que foi mostrado durante o curso da pesquisa em questão, especialmente durante as informações reveladas no referencial teórico, embasadas por uma série de referências bibliográficas, e a partir dos resultados obtidos conforme o desenvolvimento do algoritmo em foco, pode-se concluir, primeiramente, que o emprego da API, por parte do avaliador, proporcionou um direcionamento adequado para o progresso da aplicação em foco, além de ter possibilitado uma maior flexibilidade para o seu avanço. Ademais, a seleção da linguagem de programação Python para o desenvolvimento de toda a aplicação de que se trata, por seu dinamismo e sua caracterıstica multiparadigma e de forte tipagem, favoreceu o desenvolvimento do algoritmo em questão, certamente. Além disso, pode-se informar que a utilização da Teoria dos Grafos, traduzida na aplicação dos algoritmos de Busca em Profundidade DFS e Busca em Largura BFS, utilizados respectivamente como partes principais da aplicação para solucionar os labirintos impostos de forma eficiente, é assertiva, apesar das dificuldades enfrentadas para a aplicação do algoritmo DFS em situações especıficas, que poderiam gerar lentidão ou baixa eficiência de sua operação.

References

(2023). Busca em largura. *Desenvolvendo Software*.

(2023). Collections - tipos de dados de contêineres. *Python Software Foundation*.

Alberte Emilie Christensen, Cecilie Jegind Christensen, J. L. G. H. and Otto, N. (2019). Generating and solving mazes. *Roskilde Universitet*.

Andrade, A. P. (2020). O que é python? *Blog Treina Web*.

Andrade, T. (2023). Simplificando o sistema de bibliotecas com python. *Revelo Community*.

Borges, L. E. (2014). *Python para Desenvolvedores*. Novatec Editora.

Brandão, H. (2009). Introdução à busca em grafos. *Departamento de Ciências Exatas da Universidade Federal de Alfenas*.

Conceito.de., E. (2014). Conceito de labirinto. *Conceito.de.*

de Melo, G. S. (2014). Introdução à teoria dos grafos. *Universidade Federal da Paraíba - Centro de Ciências Exatas e da Natureza - Departamento de Matemática*.

Fahmi, I. R. and Suroso, D. J. (2022). A simulation-based study of maze- solving-robot navigation for educational purposes. *Journal of Robotics and Control (JRC)*.

Feofiloff, P. (2019a). Algoritmos para grafos. *www.ime.usp.br/ pf/algoritmos*$_p ara_g rafos/$.

Feofiloff, P. (2019b). Busca em profundidade. *Instituto de Matemática e Estatística da USP*.

Gená, M. (2021). Python e apis: conhecendo a biblioteca requests.

Jacobson, D. (2008). Apis: A strategy guide. *Reilly Media*.

Krieger, D. (2021). O que É api, como funciona e quais os principais tipos de api? *Blog Kenzie Academy*.

Kumar, N. and Kaur, S. (2019). A review of various maze solving algorithms based on graph theory. *IJSRD - International Journal for Scientific Research Development*.

Pamplona, D. C. (2008). Tópicos em teoria dos grafos. *Universidade Federal de Santa Catarina*.

Petrov, A. (2023). Urllib3. *Read the Docs*.

Sidinei, G. and Luis, S. Código de busca em largura e profundidade.

[Jacobson 2008] [Feofiloff 2019a] [Sidinei and Luis] [Alberte Emilie Christensen and Otto 2019] [Kumar and Kaur 2019] [Conceito.de. 2014] [Fahmi and Suroso 2022] [Pamplona 2008] [de Melo 2014] [Krieger 2021] [Borges 2014] [Andrade 2020] [Andrade 2023] [Feofiloff 2019b] [Petrov 2023] [Gená 2021] [Col 2023] [Brandão 2009] [BFS 2023]