Percorrendo Túneis

1st Eduardo Augusto Franciscon Reis
Ciência da Computação
IESB
Brasília, Brasil
eduardo.reis@iesb.edu.br

2nd Humberto Pereira Bravo
Ciência da Computação
IESB
Brasília, Brasil
humberto.bravo@iesb.edu.br

3rd Manoel Matheus Correia dos Anjos
Ciência da Computação
IESB
Brasília, Brasil
manoel.matheus@iesb.edu.br

Resumo **Este artigo tem como objetivo a utilização de Teoria dos Grafos para solucionar uma situação hipotética.A situação problema consiste em realizar a saída de um labirinto utilizando linguagem Java, para cumprir esse objetivo o artigo aborda o conceito de DFS *Depth First Search*, consumo de API (Interface de Programação de Aplicações) para a simulação de diferentes tipos de labirintos, bem como a avaliação dos resultados obtidos e das estratégias utilizadas.**

Abstract **This article aims to use Graph Theory to solve a hypothetical situation, the problem situation consists of exiting a maze using Java language, to fulfill this objective the article addresses the concept of DFS *Depth First Search*, consumption API (Application Programming Interface) for simulating different types of mazes, as well as evaluating the results obtained and the strategies used.**

***Index Terms*—Java, DFS, API, Mazes**

I. Introdução

Em uma situação hipotética na qual a mineradora Rio Tinto possui os direitos sobre uma jazida mineral localizada no estado de ”Minas Gerais” onde uma operação de extração de minérios de ferro é feita utilizando o método lavra subterrânea.A mina é composta por múltiplos túneis entrelaçados formando um tipo de labirinto, no local onde um túnel se conecta com o outro, há a presença de sensores, na qual cada sensor possui um identificador.De acordo com a integridade desse equipamento é possível confirmar se a passagem naquela região está ou não bloqueada.

Devido a um desabamento causado por uma falha na operação alguns túneis ficaram comprometidos, fazendo com que os operários ficassem presos durante a operação de extração.

II. Objetivo Geral

Diante do problema proposto, que neste caso é a evacuação dos mineradores, um algoritmo será acionado como medida de segurança.Esse algoritmo utiliza conceitos de Teoria dos Grafos e tem como finalidade explorar os possíveis caminhos e retornar todo o trajeto desde a posição inicial até a saída da mina, ressaltando que ponto de partida é a posição onde o operário se encontra.

III. Objetivo específico

Para realizar a solução da situação problema:

- O desenvolvimento de um algoritmo de busca utilizando o conceito de DFS (Depth First Search).
- Simular diferentes situações utilizando labirintos que serão providos por uma API (Interface de Programação de Aplicações).
- Averiguar os resultados obtidos e concluir se as estratégias utilizadas foram bem bem-sucedidas.

IV. Metodologia

Utilizaremos conceitos de Teoria dos Grafos com o objetivo de resolver a situação proposta.

Para desenvolver o algoritmo foi escolhido a linguagem de programação Java devido a Máquina Virtual (JVM) que permite a independência de plataforma, a linguagem possui um sistema de conexão HTTP já incluída sem a necessidade de dependência externa e o ”Garbage Collector” que faz uma limpeza na memória virtual.

Utilizaremos os diferentes tipos de labirinto serão disponibilizado por uma API (Interface de Programação de Aplicações) com o intuito de simular o funcionamento do algoritmo. Para realizar a chamada de API utilizaremos de ferramentas presente na linguagem Java para fazer a conexão.

Utilizaremos o algoritmo DFS *Depth First Search*, com a finalidade de explorar o labirinto e mostrar o caminho percorrido até a primeira saída encontrada.

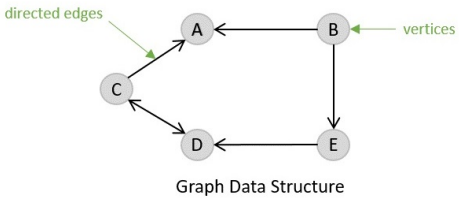
Elaborar uma tabela com os resultados encontrados, sendo os elementos dessa tabela o tipo do labirinto e o tempo gasto para construir um caminho do ponto inicial até a saída.

V. Referencial Teórico

A. Termos

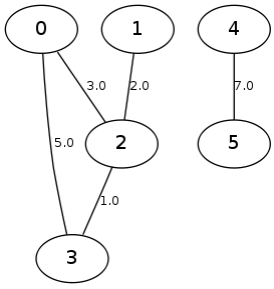
- Teoria dos Grafos: Tem sua origem na área da matemática por volta do século XVIII com o matemático Euler.Um grafo pode ser definido por um conjunto finito e não vazio de objetos chamados vértices, juntamente com um conjunto de pares não ordenados de vértices,chamado de arestas.Ou seja, grafos geralmente são representados por diagramas, onde os elementos chamados de vértices são conectados por uma aresta a um outro vértice,desse modo, formando uma ligação entre ambos. A Adjacência ocorre quando um par de vértices pertencem a uma

aresta em comum.Desse modo esses vértices podem ser considerados vizinhos. [3]



Demonstração de um grafo. [4].

- Grafo ponderado: grafo é ponderado quando suas arestas possuem um peso.Nesse caso, o peso seria o custo presente na aresta de partir de um vértice até o outro. [5]



Demonstração de um grafo ponderado. [5].

- Algoritmo: É um conjunto de regras que possui uma sequência ordenada e finita de ações executáveis (passos) que tem como finalidade resolver uma situação problema, no caso da Ciência da Computação, executar uma tarefa. [6]
- DFS: Em um algoritmo de busca em profundidade DFS *Depth First Search*, seu objetivo consiste em realizar uma busca o mais fundo possível no grafo, as arestas são exploradas a partir do vértice ”x” mais recentemente descoberto que ainda possui vértices adjacentes não explorados.Quando todas arestas adjacentes a ”x” tiverem sido exploradas, a busca “anda para trás” (do inglês backtrack) para explorar vértices do qual ”x” foi descoberto.O processo continua até que todos os vértices que são alcançáveis a partir do vértice inicial sejam descobertos. [7]
- BFS: em um algoritmo de busca em largura BFS *breadth-first search*. A busca em largura começa por um vértice ”x”, especificado pelo usuário. O algoritmo visita ”x”, depois visita todos os vizinhos de ”x”, depois todos os vizinhos dos vizinhos, e assim por diante.O algoritmo numera os vértices, em sequência, na ordem em que eles são descobertos (ou seja, visitados pela primeira vez). [8]
- dijkstra: O Algoritmo de Dijkstra é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. [9]
- A*: O algoritmo A* é um algoritmo de busca utilizado em grafos ponderados, tem como objetivo encontrar encontra o menor caminho entre dois vértices em um grafo. Nesse algoritmo é avaliado o custo do início até o destino desejado. [10]
- API: É um uma interface de comunicação que um sistema oferece para que outros acessem suas funções, dados ou recursos sem que o software ou plataforma externa precise saber como foi implementados.Ou seja, é um conjunto de padrões utilizado para facilitar a integração entre software.APIs utiliza um formato pré-definido de dados para compartilhar informações entre os sistemas, como o ”JSON” utilizado nas aplicações web. [11]
- Json: É um arquivo que contém uma série de dados estruturados em formato texto sendo este utilizado para transferir informações entre sistemas.Tem como origem a linguagem ”JavaScript”, mas pode ser utilizado em diversas linguagens de programação e sistemas.Os dados contidos em um arquivo JSON são estruturados em chave e valor. Chave: corresponde ao identificador do conteúdo. Valor: representa o conteúdo correspondente e pode conter os seguintes tipos de dados:string, array, object, number, boolean ou null. [12]
- Java: Linguagem de programação, tem sua origem no ano de 1995 dentro dos laboratórios da empresa Sun Microsystem, e tem como Características:
  - Uma linguagem de programação de alto nível orientada a objetos.
  - Máquina Virtual (Java Virtual Machine ou JVM), que garante independência de plataforma, pois o código executa na máquina virtual e essa pode ser portada para outras plataformas como Windows ou Linux.
  - Java Runtime Environment (JRE): Agrupa a máquina virtual e alguns recursos para a execução de aplicações Java.
  - Java Development Kit (JDK): Um conjunto de utilitários que oferece suporte ao desenvolvimento de aplicações.
  - No Java, os programas são escritos em um arquivo com a extensão .java, que em um processo posterior serão compilados para arquivos com a extensão .class. Esses, por sua vez, contêm os códigos a serem executados na máquina virtual. [13]
- Array: É uma estrutura de dados que tem como objetivo armazenar espaço na memória agrupando uma coleção de elementos em uma única variável.Essas informações podem ser acessadas por meio de um índice da posição pretendida. [14]
- Pilha: É uma estrutura de dados que possui uma lista de elementos empilhados, cada elemento novo fica alocado no topo da pilha.A Pilha é caracterizada como LIFO (Last In First Out), o O último elemento a entrar é o primeiro a sair. [15]
- Garbage Collector: realiza o gerenciamento automático

de memória.Tem como finalidade remover os objetos não utilizados e compactá-los após a liberação de espaço, em Em Java os objetos são criados quando necessário, e quando não estão mais em uso, a JVM através do Garbage Collector remove automaticamente os objetos e libera memória. [16]
- Memória virtual: A ideia básica é que cada programa tem seu próprio espaço de endereçamento, o qual é dividido em blocos chamados de páginas. Cada página é uma série contígua de endereços.Elas são mapeadas na memória física, mas nem todas precisam estar na memória física ao mesmo tempo para executar o programa. [17]
- jazida mineral: são concentrações em volume específico de material de ocorrência natural onde podem ser extraídos minerais de valor econômico com lucro. [18]
- lavra subterrânea: método usado para retirar os minérios encontrados abaixo do solo. Os locais são mais profundos, e as rochas podem ser mais sólidas. Nesse tipo de extração, as substâncias são delimitadas via sondas. [18]

B. Unidades

- Complexidade Temporal: Se refere ao tempo levado para um algoritmo executar uma operação dado uma cadeia de entrada [19].
- Big O: Notação para descrever o limite de um quando quando o número de valores de entrada tende ao infinito. Essa notação ignora constantes e leva somente o maior crescimento em conta, logo, um algoritmo que realiza $5n^3 + 21n + \log n$ operações terá uma complexidade $O(n^3)$ [19].
- $O(n)$: Para cada item de entrada ocorrerá uma operação.
- $O(\log n)$: O número de operações segue uma curva logarítmica.
- $O(1)$: Independente da quantidade de valores de entrada essa estrutura realizará o mesmo número de operações.
- $O(1)*$ ou $O(1)$ Amortizado: Se refere a pagar uma dívida adiantado, que nesse caso se refere a alocação de memória necessária para inserir todos os valores em uma tabela hash.

VI. Desenvolvimento

Nossa solução trata os sensores como vértices em um grafo, e a distância entre eles como a aresta.

O motivo de tratarmos o caminho na mina como um grafo se dá pelo fato que os túneis se entrelaçam, e o grafo permite representar esses cruzamentos.

O programa vai mapear o estado do túneis da mina se conectando a um sensor por vez, podendo se mover para um dos sensores adjacentes ao que está conectado.

A comunicação com os sensores se dará por meio de uma conexão *HTTP* onde o sensor receberá um valor contendo um número representando qual dos sensores adjacentes queremos analisar.

1) **Nó** ⇒ Representa uma posição no grafo, tendo um atributo contendo sua posição, um atributo cujo valor é um array contendo todos os vizinhos daquela posição, um atributo *boolean* dizendo se a posição é o ponto inicial e um atributo *boolean* dizendo se a posição é uma saída.
2) **Iniciar** ⇒ Recebe o nome do usuário e o nome do local em que ele trabalha e retorna um nó.
3) **Movimentar** ⇒ Recebe o nome do usuário, o nome do local em que ele trabalha e o próximo movimento a ser feito no grafo no formato *json*.
   Caso seja um movimento válido retornará um nó.
4) **Valida_caminho** ⇒ Recebe um array e verifica se é possível que uma pessoa percorra o caminho enviado, retornando *true* caso seja possível, *false* caso seja impossível e o número de movimentos para chegar a saída.

Como o programa será usado em situações de emergência, teremos de gerar um caminho em tempo real, sem manter o mapa pré carregado em memória.

A. Exploração do Grafo

Por não ser um grafo ponderado, a escolha de algoritmos para o processo de mapeamento foi limitada ao BFS, *Breadth First Search*, e ao DFS, *Depth First Search*, pois ambos ignoram possíveis variações de distância entre pontos adjacentes, diferente de algoritmos como *Djikstra* ou *A**.

B. API

Os possíveis movimentos que o minerador pode executar no labirinto está relacionado com a posição em que ele se encontra, caso um movimento que não é permito ocorra ele é considerado como movimento inválido.

A API implementa 4 métodos de comunicação

- *https://gtm.delary.dev/**labirintos*** : Não recebe nenhum valor e retorna um array de strings contendo nomes válidos de labirintos para teste.
- *https://gtm.delary.dev/**iniciar*** : Recebe duas strings, uma contendo o nome do labirinto a ser explorado e outra contendo o nome da sessão. Retorna uma string json contendo um inteiro contendo a posição inicial, dois booleanos, um mostrando se esta é a posição inicial e outra mostrando se esta é a posição final, e um array contendo todos os movimentos possíveis naquele ponto.
- *https://gtm.delary.dev/**movimentar*** : Recebe duas strings, uma contendo o nome do labirinto a ser explorado e outra contendo o nome da sessão, e um inteiro contendo a próxima posição para se mover. Caso seja possível realizar o movimento, será retornado uma string json com o mesmo formato recebida pelo iniciar, só que com os valores do novo objeto.
- *https://gtm.delary.dev/**validar_caminho***: Recebe duas strings, uma contendo o nome do labirinto a ser explorado e outra contendo o nome da sessão, e um array contendo todos os movimentos necessários para ir do começo ao fim do labirinto. Retorna uma string json contendo um booleano dizendo se o caminho é válido e um inteiro contendo o número de movimentos passados.

C. Parser

O Parser é um algoritmo que transforma json em classes Java. Foi utilizada a biblioteca Gson para traduzir os valores recebidos e os valores enviados em json. As classes criadas para enviar e receber mensagens da API ficarão na pasta "comunicação" no código do projeto.

D. Comunicação

Essas classes modelam os valores definidos na sessão API como respostas recebidas ou mensagens enviadas.

Todos eles são definidos como *records*, introduzidos no Java 14 e finalizados no Java 17, pois todos os seus valores são definidos no momento da declaração e acessados somente no momento do envio.

- Inicio : Valores a serem enviados para a API inicio.
- Movimento : Valores a serem enviados para a API movimentar.
- Node : Valores recebidos da API inicio e movimentar.
- CaminhoParaValidar : Valores a serem enviados para a API validar_caminho.
- CaminhoValidado : Valores recebidos da API validar_caminho.

E. Algoritmo

O algoritmo possui duas classes principais. *ChamoAPI* e *DFS*.

1) ChamoAPI: Possui um objeto HttpClient que permite a comunicação com a API e metodos para interagir com a API. Métodos Públicos

- **ChamoAPI()** → **ChamoAPI** : Construtor. Inicializa o cliente HTTP e o Gson.
- **GetNomes()** → **String** : Retorna os nomes válidos dos labirintos.
- **inicio()** → **String** : Gera uma nova sessão na API, retorna a resposta de inicio na API. O processo de transformar esta String json em uma classe Java é responsabilidade do local que a chama.
- **proxMovimento(Integer posicao)** → **String** : Recebe um inteiro contendo uma posição no grafo, retorna a resposta de movimentar na API.
- **validaCaminho(ArrayList<Integer> caminho)** → **CaminhoValidado** : Recebe um array com um caminho indo do começo a saída do grafo. Gera um CaminhoParaValidar e envia para o endereço validar_caminho. Após receber a resposta, transforma o JSON em um objeto CaminhoValidado. Caso o caminho seja válido, move o minerador do primeiro item até o último. Retorna o o objeto CaminhoValidado.

2) DFS: Depth First Search, é um algoritmo que encontra um caminho entre dois pontos em um gráfico após explorar um ramo do grafo inteiramente, após encontrar o final ele explorará o próximo ramo, repetindo este processo até que um caminho seja encontrado ou o grafo inteiro seja explorado.

Decidimos utilizar o *DFS* pois sua implementação recursiva permite o processo de ida e volta no grafo, pois ocorre um movimento para a nova posição, uma chamada recursiva e, ao retornar da chamada recursiva, volta para a posição anterior.

Utilizamos um HashSet para guardar as posições já visitadas. O Set é estrutura de dados impede a inserção de valores repetidos, por isso pode ser usada para garantir que não passamos no mesmo lugar duas vezes. O Hash é uma função que recebe um valor e o transforma em um número pseudo-aleatório, que será usado como índice de inserção no Set [20]. Métodos Públicos

- **DFS()** → **DFS** : Inicializa o parser Gson, inicializa o HashSet sem nenhum elemento visitado e inicializa o ChamoAPI. Há um array interno contendo o caminho do início ao fim, o qual pode ser nulo.
- **processaGrafo()** : Começa a aplicação do DFS no grafo da API. Após encontrar a saída, passa o valor para a variável caminho.
- **gotoFim()** → **CaminhoValidado** : Válida que o caminho não está vazio e passa o caminho para a API. Caso o caminho esteja vazio, gera um erro.
- **getCaminho()** → **ArrayList<Integer>** : Retorna o caminho presente no objeto. Caso esteja vazio, retorna um Optional vazio.

VII. Resultado

Tipo do labirinto	Execução do Programa
maze-sample	2.076ms
maze-sample-2	2.949ms
medium-maze	7.072ms
large-maze	31.363ms
very-large-maze	259.572ms

Estes foram os resultados obtidos da simulação dos diferentes labirintos. A estratégia utilizado com o emprego do DFS como algoritmo de busca foi bem-sucedida para os labirintos "maze sample", "maze sample 2", "medium maze" e "large maze", mas para o "very large maze" o tempo de execução acabou sendo demorado.

O tempo de execução para o "very-large-maze" pode ter ocorrido devido ao fato que o *DFS* explora os pontos mais distantes do ponto inicial primeiro, fazendo com que o processo de encontrar a saída em grafos com grande número de elementos gere movimentos desnecessários caso a saída esteja relativamente próxima do ponto inicial.

VIII. Conclusão

Neste artigo foi apresentado uma maneira de encontrar a saída de uma situação hipotética semelhante a um labirinto utilizando o conceito de busca em profundidade(DFS).O uso do DFS para explorar os caminhos disponíveis e mostrar a primeira saída encontrada apresentou-se bem-sucedida para labirintos curtos e médios.Para os labirintos com uma maior quantidade de elementos o desempenho ficou aquém do esperado gerando uma deterioração no tempo de execução.

REFERENCES

[1] S. Kibler, A. Hauer, D. Giessel, C. Malveaux, and D. Raskovic, "IEEE Micromouse for mechatronics research and education," in 2011 IEEE International Conference on Mechatronics, 2011, pp. 887–892.

[2] S. Mishra and P. Bande, "Maze Solving Algorithms for Micro Mouse,' ' in 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008, pp. 86–93.

[3] G. S. Melo, "Introdução à Teoria dos Grafos," Universidade Federal da Paraíba, 2015. [Online]. Disponível em: https://repositorio.ufpb.br/jspui/bitstream/tede/7549/5/arquivototal.pdf. [Acesso em: 18-Nov-2023].

[4] "Graph Data Structure" Tutorials Point, 2023. [Online]. Disponível em: https://www.tutorialspoint.com/data _structures_algorithms/graph_data_structure.htm. [Acesso em: 18-Nov-2023].

[5] "Explorando a classe ArrayList no Java,' ' DevMedia, 2019. [Online]. Disponível em: https://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/ 24298. [Acesso em: 22-Out-2023].

[6] R. Gogoni, "O que é algoritmo?" Tecnoblog, 2019. [Online]. Disponível em: https://tecnoblog.net/responde/o-que-e-algoritmo/. [Acesso em: 18-Nov-2023].

[7] H. Sandmann, "Teoria dos Grafos," Universidade Federal de Alfenas, 2010. [Online]. Disponível em: https://www.bcc.unifal-mg.edu.br/ humberto/disciplinas/2010_2_grafos/pdf_aulas/aula_04.pdf. [Acesso em: 18-Nov-2023].

[8] P. Feofiloff, "Busca em largura", 2019. [Online]. Disponível em: https://www.ime.usp.br/ pf/algoritmos-para-grafos/aulas/bfs.html. [Acesso em: 19-Nov-2023].

[9] P. Feofiloff, "Algoritmo de Dijkstra", 2019. [Online]. Disponível em: https://www.ime.usp.br/ pf/algoritmos-para-grafos/aulas/dijkstra.html. [Acesso em: 19-Nov-2023].

[10] P. M. S. Souza, "PERSISTENT A*: UMA MELHORIA DO ALGORITMO A* USANDO ÁRVORE DE SEGMENTO PERSISTENTE E HASHING INCREMENTAL", 2022. [Online]. Disponível em: https://repositorio.ufc.br/bitstream/riufc/65339/1/2022_tcc_pmssousa.pdf. [Acesso em: 19-Nov-2023].

[11] D. Melo, "O que é uma API? Guia para iniciantes." Tecnoblog, 2020. [Online]. Disponível em:: https://tecnoblog.net/responde/o-que-e-uma-api-guia-para-iniciantes/. [Acesso em: 18-Nov-2023].

[12] I. Souza, "O que é JSON?" Rock Content, 2020. [Online]. Disponível em: https://rockcontent.com/br/blog/json/. [Acesso em: 18-Nov-2023].

[13] A. Bessa, "Java: o que é, para que serve e como funciona." Alura, 2023. [Online]. Disponível em: https://www.alura.com.br/artigos/java. [Acesso em: 18-Nov-2023].

[14] U. Roveda, "O que é um Array?" Kenzie Academy Brasil, 2022. [Online]. Disponível em: https://kenzie.com.br/blog/o-que-array/. [Acesso em: 18-Nov-2023].

[15] W. Silva, "Pilha: Estrutura de Dados," Growth Code, 2021. [Online]. Disponível em: https://growthcode.com.br/algoritmos/pilha-estrutura-de-dados/. [Acesso em: 18-Nov-2023].

[16] G. B. Zarelli, "Java Garbage Collector — Por que precisamos conhecê-lo?" Medium, 2022. [Online]. Disponível em: https://medium.com/luizalabs/java-garbage-collector-porque-precisamos-conhec%C3%AA-lo-9d26ebb0a6d8. [Acesso em: 18-Nov-2023].

[17] A. S. Tanenbaum, H. Bos, "Sistemas Operacionais Modernos," 4ª edição, 2015. pp. 134-135.

[18] M. Santos, "O Que é Mineração?", 2019. [Online]. Disponível em: https://blog.jazida.com/ o-que-e-mineracao/. [Acesso em: 25-Out-2023].

[19] P. Morin, "Open Data Structures," 2013. [Online]. Disponível em: https://opendatastructures.org/ods-python.pdf. [Acesso em: 26-Out-2023].

[20] "O que é Hashing? Como funcionam os códigos de Hashing com exemplos," FreeCodeCamp, 2022. [Online]. Disponível em: https://www.freecodecamp.org/portuguese/news/o-que-e-hashing-como-funcionam-os-codigos-de-hashing-com-exemplos/. [Acesso em: 18-Nov-2023].