Árvores em Estruturas de Dados

LUZ, Jederilson S.[1]

[1] Jederilson Sousa Luz
UFPI - CSHNB/Estruturas de Dados II
Jederilson@gmail.com

Resumo - Este texto apresenta os resultados obtidos no desenvolvimento da atividade sobre grafos da matéria de Estruturas de Dados II na Universidade Federal do Piauí. Serão abordados grafos de estados, ponderados e dígrafos, os métodos de busca em largura e profundidade, e os testes de desempenho e comparações entre os métodos de busca em um dos grafos deste trabalho no contexto dos exercícios propostos da matéria.

(Palavras-chave: grafos, grafo de estados, grafo ponderado, dígrafo, busca em profundidade, busca em largura)

Introdução

As estruturas de dados chamadas de grafos compõem um dos assuntos mais diversos na área da computação e estão presentes em vários cenários distintos. A *teoria dos grafos* é um ramo da matemática que estuda a relação entre objetos de um determinado conjunto. Ao conjunto desses objetos se dá o nome de **grafo,** $G(V, E)$, onde V é um conjunto não vazio de objetos chamados de **vértices** e E é um conjunto de pares não ordenados de V chamado de **arestas** (do inglês, *Edges).*

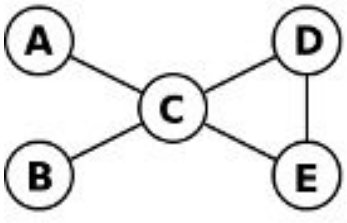

Figura 1 - Representação de um grafo.

Podem ser encontrados em qualquer lugar onde existe uma relação entre objetos, como em redes de computadores (fundamental para os algoritmos de roteamento), em algoritmos de recomendações de amigos dentro de redes sociais como o Facebook, ou de propagandas personalizadas como as do Google. Podem ser aplicados junto à tecnologia de GPS para montar rotas entre cidades (cidades seriam os vértices e ruas as arestas) ou para ajudar uma empresa de correios a organizar o fluxo dos pacotes entre seus estabelecimentos e a casa dos clientes. Também podem ser aplicados para entender todas as possibilidades possíveis dentro de um jogo e as relações entre um estado e outro.

Para representar esse tipo de estrutura computacionalmente, uma das formas é usar a chamada **lista de adjacência**, um vetor N, onde N é a quantidade de vértices do grafo. Em cada posição do vetor haverá uma lista de todos os vértices adjacentes ao vértice correspondente a aquela posição.

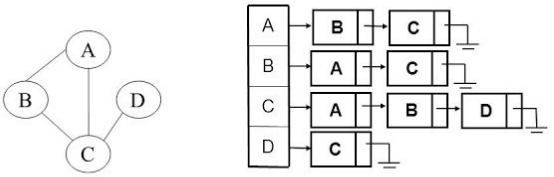

Figura 2 - Representação de uma lista de adjacência para um grafo de 4 vértices.

Neste trabalho serão abordados três cenários diferentes: Torre de Hanoi, Tubo de Formigas e a Viagem de Arthur.

Torre de Hanói

Torre de Hanói é um jogo bem simples onde temos um conjunto de três pinos e N discos, cada disco com um tamanhos diferente. Nesse jogo o objetivo é levar todos os discos inicialmente posicionados no pino A

para o pino C, como mostra a **Figura 3** logo abaixo**.**

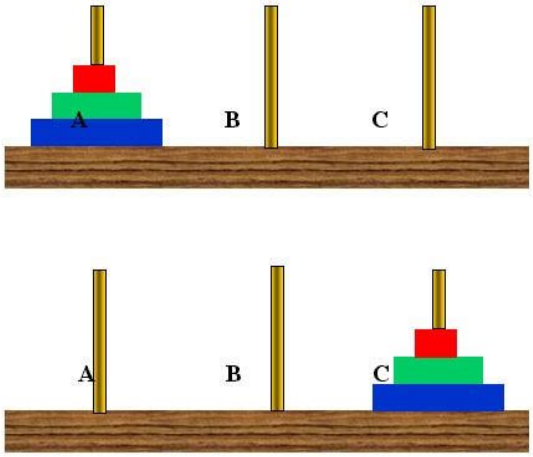

Figura 3 - Representação de uma Torre de Hanoi.

Existem três restrições para poder mover os disco entre os pinos:

- A cada jogada apenas um disco pode ser movimentado.
- Apenas o disco do topo da pilha pode ser movimentado.
- Um disco só poder ser posicionado sobre a base de um pino ou sobre um disco maior que ele.

Para facilitar a comunicação sobre as configurações do jogo, será adotada a seguinte notação:

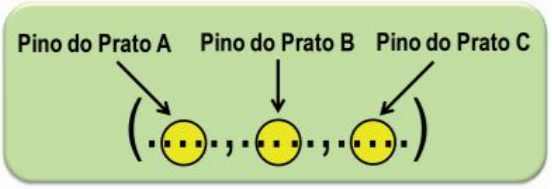

Figura 4 - Notação de um estado da Torre de Hanoi.

Um vetor com N posição, onde N é a quantidade de discos. O valor em cada posição informa em qual pino está aquele disco (os maiores discos ficam mais à esquerda do vetor e os menores mais à direita). A primeira configuração apresentada na **Figura 1** seria (1,1,1) e a segunda (3, 3, 3).

A 1ª questão deste trabalho pedia que fosse montado um grafo representando todas as configurações possíveis dentro do jogo com 3 pinos e 4 discos. Após isso, dada uma configuração inicial, o programa deveria informar a maior e a menor rota da configuração inicial até a final do jogo. O programa também deveria desconsiderar configurações antecedentes a atual. Exemplo: começando de (1,1,2), a rota deve desconsiderar (1,1,1), pois esta é uma configuração anterior da atual.

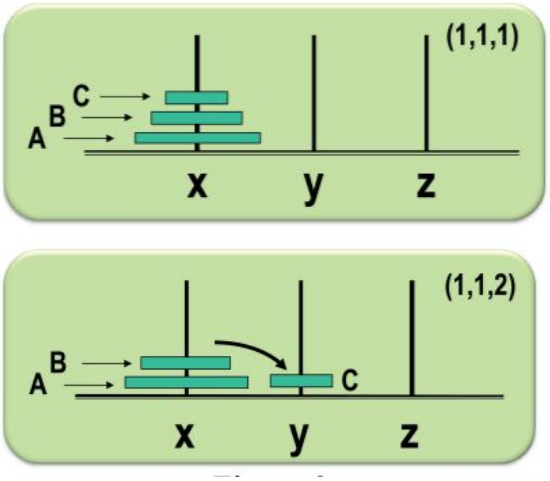

Figura 5 - Exemplo de movimentação entre estados.

Para entender a resolução, primeiro é preciso entender o conceito de **grafo de estados**. Os vértices de um grafo nos exemplos anteriores foram associados a uma informação simples como um número ou uma letra, mas um vértice pode representar uma informação mais complexa como o nome de uma cidade com arestas para as cidades adjacentes, informações de uma pessoa dentro de uma empresa com ligações para outras pessoas que são seus subordinados ou superiores, ou para representar situações dentro de um contexto específico como em um jogo. A informação dentro de cada vértice pode variar muito dependendo do problema que se procura resolver. Em casos onde temos várias configurações de um cenário, o grafo de estados pode ajudar a entender como ocorrem as relações entre cada configuração do cenário. A notação apresentada anteriormente para comunicar uma configuração da torre de hanói pode ser a informação em um vértice do grafo de estados. Assim um grafo da torre de hanoi poderia representar qualquer configuração do jogo e quais outros estados é possível alcançar saindo de um estado específico.

Voltando para o problema da torre de hanoi, a questão pedia que fosse criado um grafo para um jogo de torre de hanoi com 3 pinos e 4 discos. Para criar esse grafo foi

utilizada uma lista de adjacência 81x3, pois para 4 discos e três pinos, por exemplo (3,1,3,2), temos 3^4 estados diferentes, mas nenhum vértice tem mais que três arestas. Isso ocorre porque qualquer que seja a situação, no máximo será possível escolher entre dois pinos para mover o menor disco e uma movimentação para o segundo menor disco disponível. Para gerar as arestas dessa matriz foi necessário criar uma regra capaz de generalizar as comparações, ou seja, dado dois estados, um estado atual A e uma tentativa de movimento para um estado B, ser capaz de dizer se é possível executar o movimento. Em termos mais simples, dizer se existe uma aresta entre os dois vértices A e B. A lógica desenvolvida é estruturada em 4 condições. Caso todas sejam verdadeira (e nesta ordem), a ligação existe:

- Os estados contem $N-1$ números iguais, onde N é o número de discos.
- A diferença entre os estado deve ocorrer na mesma posição, ou seja: $estado\_1[N-X] \; != \; estado\_2[N-X]$.
- Não podem haver discos acima do que se pretende movimentar, logo, $estado\_1[N-X] \; != \; estado\_1[N-X+i]$ para $0 < i < N-X$.
- Da mesma forma não podem haver discos menores do que o movimentado no pino alvo, logo, $estado\_2[N-X] \; != \; estado\_2[N-X+i]$ para $0 < i < N-X$.

Note que os discos menores estão representados mais à direita do vetor, por essa razão a verificação ocorre apenas da posição da alteração em diante.

Outro conceito importante a seguir para a resolução desta questão é o de buscas. É muito comum a necessidade de encontrar a melhor estrutura de decisões ou caminhos em um grafo para chegar em um destino ou estado final. No problema da torre de hanói é pedido que encontremos o caminho com o maior e menor número de vértices. Para resolver esse tipo de problema existem duas soluções principais: **busca em largura** e **busca em profundidade**.

A busca em largura recebe um ponto de origem e agenda uma visita com todos os vértices adjacentes ao atual em uma fila, depois navega para o próximo item da fila. Esse passo é repetido até que não haja mais vértices agendados na fila. É importante marcar vértices que já foram analisados para não entrar em loops. Para prevenir esse tipo de problema, a abordagem adotada neste trabalho utiliza cores como representação: vértices **pretos** já foram visitados, **cinzas** estão agendados para serem visitados e **brancos** nem foram visitados e nem estão agendados ainda. A questão do agendamento é necessária porque na busca em largura primeiro todos os vértices adjacentes são agendados e só depois um deles é visitado. É importante que estes não sejam agendados novamente por outro vértice adjacente a eles.

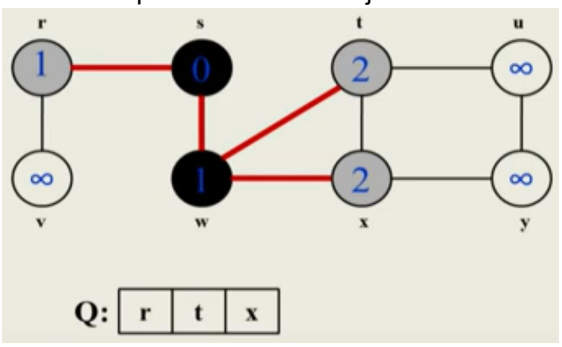

Figura 6 - Representação de uma busca em largura.

Na **Figura 6**, o vértices **s** e **w** (pretos) já foram visitados; **r**, **t** e **x** (cinzas) estão agendados e os demais (brancos) ainda não foram nem visitados e nem agendados. Para agendar os vértices, assim que são encontrados nas adjacências do vértice preto, são adicionados em uma fila. O processo de visitar, marcar e navegar na ordem correta é gerenciado por essa fila de vértices. Sempre que o processo for executado em um vértice, navega-se para o próximo item da fila, o remove e adiciona no final da fila seus vértices adjacentes ainda não agendados. Quando a fila ficar vazia a busca acaba. Os vértices guardam durante o agendamento a informação de quem é seu antecessor e qual é a sua distância da origem, que é a distância do seu antecessor até a origem +1;

Esse tipo de busca foi utilizado para mapear a distância de cada estado da torre de hanói até o estado inicial (1,1,1,1). Dessa forma é possível buscar rotas de qualquer ponto até o estado (3,3,3,3) sem refazer jogadas, pois as jogadas anteriores terão uma distância menor da origem, possibilitando ao algoritmo navegar apenas para estado com distância maior ou igual a do vértice atual.

O outro tipo de busca, a busca em profundidade, funciona da seguinte forma:

Dado um vértice origem, é escolhido o primeiro de seus vértices adjacentes e navega-se até ele. Esse processo é repetido no próximo vértice até que não seja mais possível continuar, seja por não haver mais vértices adjacentes ou por apenas ter opção de vértices já visitados (é importante guardar essa informação aqui também), ou por ter chegado no destino desejado.

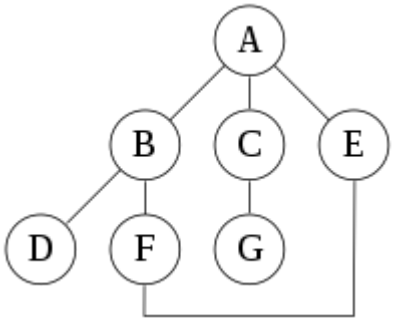

Figura 7 - Grafo para exemplificar busca em profundidade.

Uma busca em profundidade no grafo anterior executaria os seguintes caminhos:

- **A, B, D;** Retorna para **B** por não haver mais caminhos;
- **A, B, F, E**; Retorna para **A** por não haver mais caminhos possíveis (**A** já foi visitado neste caminho);
- **A, C, G;** Retorna para **A** por não haver mais caminhos;
- **A, E, F, B. D;** Retorna para **A** por não haver mais caminhos;
- A busca é encerrada por já ter visitados todas as rotas.

Observe que a busca em profundidade cria uma árvore de decisões englobando todas as rotas possíveis. Foi por meio desse tipo de busca que a maior e menor rota de um estado qualquer do grafo da torre de hanói até o estado (3,3,3,3) foram alcançadas, levando em conta o número de vértices visitados e a distância de cada vértice até a origem para não “desfazer” jogadas já feitas.

Os detalhes da implementação e explicações das funções estão no anexo 1.

Tubo de Formigas

Na 2ª questão foi apresentado o problema do tubo de formigas, onde são posicionadas em um tubo “4 formigas de forma que fiquem equidistantes entre si. As formigas somente podem caminhar pelo fundo do tubo. Elas caminham sempre na direção em que sua cabeça enxerga (caminha para frente) e todas com a mesma velocidade. Uma formiga nunca para de caminhar e jamais passa por cima de outra dentro do tubo. Quando duas formigas se encontram, ambas inverte a direção da caminhada. Elas são lentas para inverter o movimento, de forma que gastam o tempo de percorrer meio tubo para girar o corpo e inverter a direção do movimento. Quando uma formiga alcança o fim do tubo, ela vai embora”.

Neste problema foi pedido que um grafo de estados fosse montado representando todos os estados possíveis do problema e então que, dado um estado inicial, o programa fosse capaz de mostrar o menor caminho entre os estados para sair do tubo até que não restasse nenhuma formiga. Para montar esse grafo primeiro precisamos especificar uma notação para representar cada estado. Seguiremos a seguinte notação:

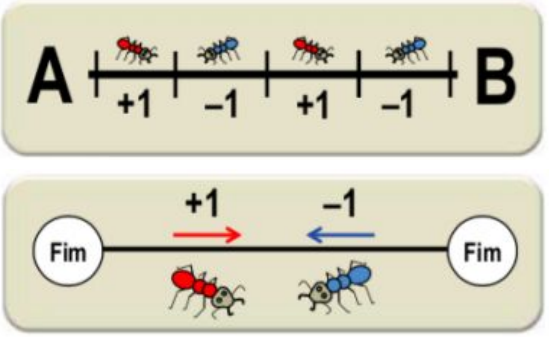

Figura 8 - Representação de um estado do tubo de formigas.

Termos um vetor de tamanho N, onde N é o número de formigas no tubo. Esse tamanho diminui conforme as formigas deixam o tubo. A movimentação das formigas segue as seguintes regras:

- Quando duas formigas se encontram, seus valores de movimentos são multiplicados por –1.
- Quando uma formiga com movimento –1 alcança a extremidade A do tubo, ou uma formiga com movimento +1 alcança a extremidade B do tubo, a configuração é reduzida em uma posição.

Observe que não é possível para as formigas retornar para as configurações

anteriores. Uma vez que o estado muda, não há mais nenhum caminho de retorno. A **Figura 8** logo abaixo exemplifica uma rota a partir de um estado específico até que o tubo fique vazio.

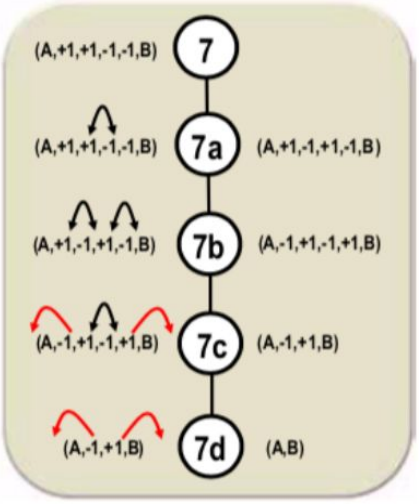

Figura 9 - Exemplo de troca de estados com a caminhada das formigas.

.

Para representar esse tipo de grafo temos o **dígrafo**, ou grafo direcionado. Neste tipo de grafo uma ligação de A para B não implica necessariamente em uma ligação de B para A, como foi visto no exemplo da **Figura 1**.

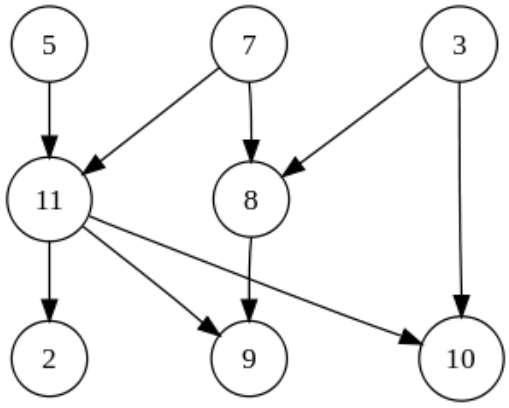

Figura 10 - Representação de um dígrafo.

As setas especificam que a aresta só é válida de 5 para 11, ou de 3 para 8, por exemplo. Caso houvesse a ligação inversa, isso teria que ser especificado explicitamente, diferente dos grafos normais, onde sempre existe a ligação para os dois lados.

O tubo de formigas é dígrafo. Outra observação importante é que cada estado só tem uma aresta apontando para outro estado. Cada estado desse grafo, dadas as regras de movimentação das formigas, sempre irão caminhar para um único outro estado específico. Essa característica nos permite simplificar a lista de adjacência, pois não é necessário ter uma lista de lista, apenas um vetor que guarda a adjacência correta na posição correspondente a determinado vértice.

Para encontrar a rota do estado inicial informado pelo usuário até o estado final (⊘), foi utilizada a busca em profundidade. Como sempre só haverá uma rota, a busca em profundidade sempre encontra um único caminho.

Detalhes de implementação podem ser encontrado no anexo 1.

Viagem de Arthur

A 3ª questão apresenta um problema onde um personagem chamado Arthur junto com sua família desejam saber qual a melhor rota para um passeio em um país dado um grafo que represente todas as cidades do país, as estradas existentes ligando as cidades e de qual cidade a família começa o passeio. As estradas têm pedágio fixo em ambas as direções. O pai de Arthur tem uma quantidade P de dinheiro que pode gastar e deseja saber qual a rota em que pode visitar mais cidades com esta quantia.

O grafo da cidade é um **grafo ponderado**, um tipo de grafo que possui pesos nas arestas. No contexto da questão, o peso das arestas (estradas) entre dois vértices (cidades) é o valor do pedágio. Para usar a aresta é necessário levar em conta o seu custo, no caso, se há dinheiro disponível para a viagem. Levando isso em conta, pede-se que seja dada a rota com mais vértices visitados usando a busca em profundidade.

Na questão seguinte pede-se que o mesmo problema seja resolvido utilizando busca em largura. Os algoritmos utilizados aqui são essencialmente os mesmo utilizados nas questões anteriores, apenas com a diferença que aqui não existe um destino. O objetivo é percorrer o maior número de cidades possível com a quantidade P de dinheiro.

Após programar as duas buscas, a 5ª questão pede que sejam feitos testes com 10

grafos diferentes, comparar os tempos das buscas e responder a duas questões:

- É possível determinar em qual das soluções eles visitará um maior número de cidades?
- É possível determinar qual solução eles conseguiram economizar mais?

Todos os 10 grafos utilizados estarão listados no anexo 1. Aqui nos interessa apenas saber dos resultados dos testes. Os dados brutos dos testes estão disponíveis no anexo 2.

Quanto aos dados compilados, temos dois gráficos para explorar as perguntas citadas anteriormente:

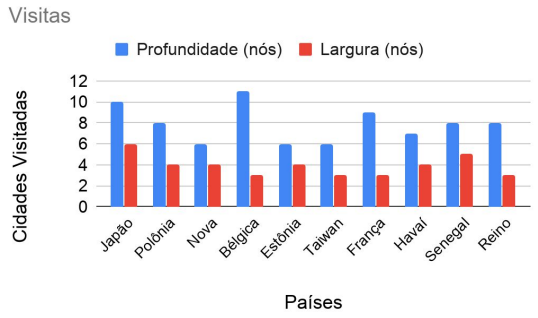

Gráfico 1 - Número de cidades visitadas por cada método de busca.

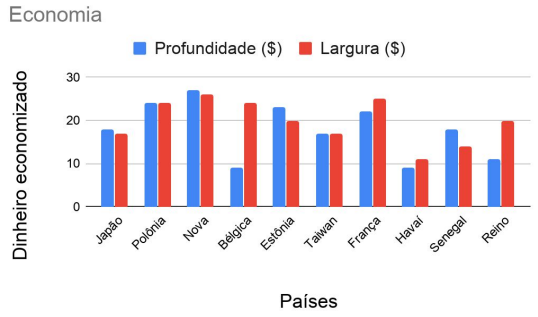

Gráfico 2 - Dinheiro não utilizado pela melhor rota encontrada de cada método.

Os grafos elaborados são representações de vários países diferentes para testar os dois códigos em diversas situações e também para aproximar o contexto da questão para o de uma viagem real. Algumas convenções adotadas nos testes:

- A origem da busca sempre será a capital do país;
- A capital do país é sempre o vértice 1;
- Todas as viagens utilizaram o valor fixo de $P = 40$.

Os resultados obtidos no **Gráfico 1** são bem claros em informar que a busca em profundidade na maioria dos casos terá rotas maiores que as da busca em largura, nos testes apresentado aqui, em todos os casos. Isso ocorre porque a busca em profundidade testa todas as combinações possíveis e retorna a com mais visitas. A busca em largura elabora diversas rotas diferentes a partir da origem ao mesmo tempo e não compartilha nós entre cada uma, o que acaba com várias outras das possibilidades de rotas exploradas pela busca em profundidade.

Mesmo montando rotas menores, a busca em largura não conseguiu economizar mais que a em profundidade, como apresentado no **Gráfico 2**. Apenas no caso da Bélgica houve uma diferença grande de economia em favor da largura, mas nesse caso a busca em profundidade visitou quase 4 vezes mais cidades. A diferença de gasto é justificável. Nos testes da França, Havaí e Reino Unido a largura também guardou mais dinheiro, mas a profundidade também visitou pelo menos o dobro de cidades nesses testes. Levando em conta a relação $CF = custo/cidades$, onde CF é o custo benefício da rota, a busca em profundidade ganha com $CF \approx 2,866$ de custo em média para visitar cada cidade, enquanto a busca em largura tem $CF = 5,345$ em média. A real vantagem da busca em largura sobre a em profundidade é a de encontrar um destino. Por montar várias rotas em paralelo, consegue encontrar o destino com mais facilidade que a profundidade em grafos muito grandes. Basicamente a busca em largura jogou fora de casa nesses testes.

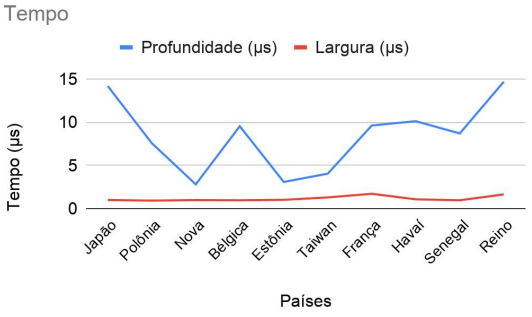

Gráfico 3 - Médias do tempo gasto por cada busca nos grafos elaborados..

Quanto ao tempo, a busca em profundidade indiscutivelmente gasta mais tempo para retornar a rota, pois testa cada

combinação possível de rota a partir da origem, gerando uma árvore de decisão enorme. A busca em largura visita cada nó apenas uma vez. É o esperado que seja mais rápida.

Os detalhes de implementação das funções utilizadas na resolução desse problema estão disponíveis no anexo 1.

Conclusão

O objetivo deste trabalho foi alcançado. Como resultado agora é possível fazer comparações entre os algoritmos de busca em largura e profundidade. Muito já foi dito no texto, mas para compilar as conclusões encontradas, estão listadas aqui:

- Busca em profundidade consegue encontrar as melhores rotas.
- Busca em largura executa mais rapidamente.

Anexo

1. Detalhamento da implementação das funções utilizadas na resolução dos exercícios estão disponíveis em: https://docs.google.com/document/d/1i8YMQC_aRKwQXnID3jAczq715WUREot2j7lOU7q7CHY/edit?usp=sharing
2. Dados brutos dos testes: https://docs.google.com/spreadsheets/d/1R_fN5S8KAKazFpbmd05Nfs3aznBxUm6zLzk4KH9g86g/edit?usp=sharing
3. Repositório dos códigos disponível em: https://github.com/Jejinketsu/TrabalhoGrafos

Referências

4. VILLAS, Marcos V. *et al*. **Estruturas de Dados**. N.º 12. Local: Rio de Janeiro, Elsevier: Campus, 1993.
5. PREISS, Bruno R. **Estruturas de Dados e Algoritmos**. N.º 5. Local: Rio de Janeiro, Elsevier: Campus, 2000.
6. CORMEN, Thomas H. *et al*. **Algoritmos**. N.º 2. Local: Rio de Janeiro, Elsevier: Campus, 2002.
7. TENEMBAUM, Aaron M.; LANGSAM, Yedidyah; SOUZA, AUGENSTEIN, Moshe J. **Estruturas de Dados Usando C**. Local: São Paulo, MAKRON Books, 1995.