Spaceships

Eduardo Vieira e Sousa
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
eduardoatrcomp@gmail.com

***Abstract*—Este relatório aborda a implementação do desafio Spaceships. Primeiramente, alguns aspectos gerais sobre o projeto são abordados, assim como as estruturas de dados implementadas e os métodos aplicados na resolução do problema. Em seguida, é feita uma análise da complexidade de cada algoritmo em termos de tempo e espaço. Por fim, o tempo de execução e o consumo de memória são analisados em função do número de vértices, arestas e movimentos para cada tipo de grafo, sendo os resultados expostos por meio de tabelas e gráficos.**

I. Introdução

O problema consiste em ler um grafo onde as arestas estão representadas por pares de vértices, assim como os movimentos entre eles. Cada nave é uma componente conexa, que pode ser classificada em 4 tipos diferentes, cada um pertencendo a uma classe de grafos, da seguinte maneira.

- Reconhecimento: caso particular de árvore;
- Frigata: árvore;
- Bombardeiro: bipartido;
- Transportador: ciclo.

Deste modo, para cada arquivo de entrada as naves devem ser identificadas utilizando-se das propriedades da classe a qual elas pertencem. Além disso, é necessário calcular o **Tempo de Vantagem**, que é um limite inferior de tempo para que cada tripulante chegue ao seu posto. Isso pode ser feito calculando a soma das distâncias de cada vértice de partida para seu respectivo vértice de destino. Em seguida, basta dividir o total por dois, considerando que a cada permutação, os tripulantes trocados de lugar ficam mais próximos de seus destinos, já que queremos um limite inferior para o tempo. Por fim, basta escrever o resultado no arquivo de saída fornecido por parâmetro.

A implementação do problema foi feita utilizando a linguagem Java. Nas seções seguintes estão demonstrados os processos de modelagem e testes do programa desenvolvido.

II. Modelagem

Todo o problema pode ser modelado utilizando quatro classes: **PostoDeCombate**, **Radar**, **Relatório** e **Main**. Cada uma delas está explicada a seguir.

A. PostoDeCombate

São as estruturas de dados que representam os vértices do grafo de entrada, onde as arestas são representadas no formato de lista adjacência. Cada **PostoDeCombate** possui os seguintes atributos.

- **numero**: um inteiro que representa o número do vértice, utilizado em sua identificação;
- **nivel**: caso o vértice faça parte de uma nave de reconhecimento ou Frigata, esse valor indica o nível onde ele está localizado dentro da árvore;
- **particao**: esse valor serve para o cálculo da localização dentro das partições em naves do tipo Bombardeiro, ou para o cálculo do **Lowest Common Ancestor** em naves de Reconhecimento ou Frigatas;
- **corBFS e corDFS**: representam as cores dos vértices utilizadas nas buscas em largura e profundidade. Cada vértice possui variáveis diferentes para que uma busca não interfira na outra, já que ambas ocorrem simultaneamente;
- **distanciaBFS e distanciaDFS**: representam as distâncias dos vértices, calculada a partir de um vértice raiz, utilizando o respectivo tipo de busca;
- **posto**: um ponteiro indicando o **PostoDeCombate** para o qual o tripulante do vértice atual deve se deslocar antes de uma batalha;
- **pai**: aponta para o vértice predecessor ao vértice atual após a realização de uma busca;
- **teleports**: representa a lista de adjacência do vértice, ou seja, é uma lista contendo todos os **PostoDeCombate** ligados a ele.

B. Radar

Classe encarregada da leitura do arquivo de entrada e da geração dos vértices do grafo. O método **sondarFrotas** recebe o caminho para o arquivo e retorna uma lista de **PostoDeCombate**, representando uma frota. Além disso, também foram implementados métodos para a impressão do grafo e geração de arquivos de teste com naves dos quatro tipos.

C. Relatório

Possui a função de gerar o relatório, que é dividido em duas partes: identificação das naves e cálculo do **Tempo de Vantagem**. O método **buscar** recebe uma lista de **PostoDeCombate**, que representa o grafo, e começa a iterar sobre os seus vértices, fazendo uma busca em profundidade a partir de todos os nodos em brancos que encontrar, identificando os demais vértices pertencentes à nave e inserindo-os em uma lista. Toda vez que a busca em profundidade termina de percorrer as listas de adjacência, retornando para o laço mais externo, isso significa que uma nave foi completamente

explorada, e o próximo vértice em branco encontrado pertence a uma nova componente. Ao finalizar cada nave, é gerada uma lista de **PostoDeCombate** para representá-la, e também é feita a contagem do seu número de vértices, arestas e também do grau máximo de seus nodos. Esses dados são utilizados na identificação do tipo de nave, pelo método **identificaNave**, da seguinte maneira.

- Transportadores: número de vértices igual ao número de arestas;
- Bombardeiros: número de vértices menor do que o número de arestas;
- Frigatas: número de vértices maior do que o número de arestas e grau máximo maior que 2;
- Reconhecimento: número de vértices maior do que o número de arestas e grau máximo menor ou igual a 2.

Após a identificação da nave, o seu tempo de vantagem é imediatamente calculado utilizando o método **tempoDeVantagem**, que além de receber a lista de **PostoDeCombate** representando a nave, recebe também o seu tipo, e o tempo de vantagem mínimo calculado até o momento. O método utiliza diferentes abordagens para calcular o tempo de cada tipo de nave, explorando as características de cada classe para um cálculo mais eficiente.

- Reconhecimento: Para esse tipo de nave, como todos os vértices estão em sequência, basta encontrar um nodo de uma das extremidades, ou seja, com grau igual a 1, e em seguida fazer uma busca em profundidade a partir dele. Para calcular a distância entre dois vértices basta obter o valor absoluto da diferença das distâncias da raiz até cada um deles, como mostra a figura (1).

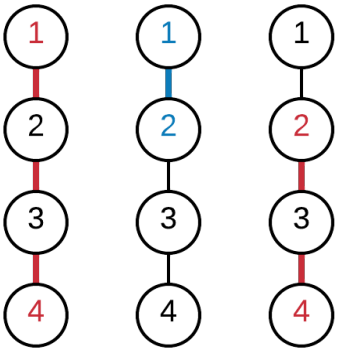


Fig. 1. Caminho de 2 para 4 com DFS partindo de 1

- Frigatas: são grafos do tipo árvore, por isso inicialmente é necessário fazer uma busca em largura, e em seguida calcular o Lowest Common Ancestor (LCA) para os pares de vértices sobre o qual pretende-se fazer o cálculo das distâncias. Em seguida, basta fazer a soma das distâncias do vértice raiz do BFS para os dois nodos, e depois subtrair dessa soma a distância da raiz até o LCA dos vértices iniciais duas vezes, já que essa distância foi somada em dobro, a figura (2) demonstra o funcionamento desse processo;

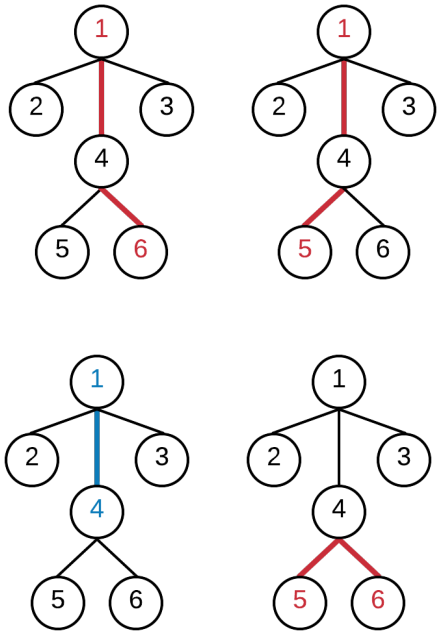


Fig. 2. Caminho de 5 para 6 com BFS partindo de 1 e LCA 4

- Bombardeiros: são representados por grafos bipartidos, sendo que todos os vértices de uma partição estão ligados diretamente com os da outra, mas não podem existir arestas entre nodos da mesma partição. Deste modo, qualquer caminho nesse grafo será de tamanho 1 quando o vértice de chegada não pertencer à mesma partição do vértice de saída, e caso contrário, o tamanho do caminho será igual a 2, como mostra a figura (3). Para calcular a distância basta atribuir o valor 1 para a variável **particao** de todos os vértices da lista de adjacência de um nodo qualquer, já que cada um deles possui aresta para todos os outros da outra partição. Em seguida, basta calcular a distância para todos os nodos do grafo verificando se o vértice de saída e o de chegada estão na mesma partição, utilizando o valor atribuído e fazendo a soma dos tempos corretos;

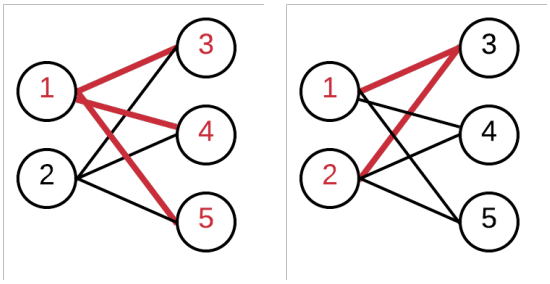


Fig. 3. Caminhos de tamanho 1 para outra partição e tamanho 2 para a mesma

- Transportadores: para calcular o tempo de vantagem de transportadores, basta fazer um DFS de um vértice qualquer, que ao terminar, terá atribuído distâncias de 0 a $V-1$, de maneira sequencial para os demais vértices. Em seguida, para obter a distância entre dois vértices basta calcular o valor absoluto da diferença entre suas distâncias para o vértice raiz obtidas na busca em profundidade. Caso esse valor seja maior que a metade do número de vértices, será então mais vantajoso fazer o caminho inverso no ciclo, e para isso basta subtrair a distância calculada anteriormente do número total de vértices, encontrando deste modo o tamanho do menor caminho, que utiliza as arestas restantes, como demonstrado na figura (4).

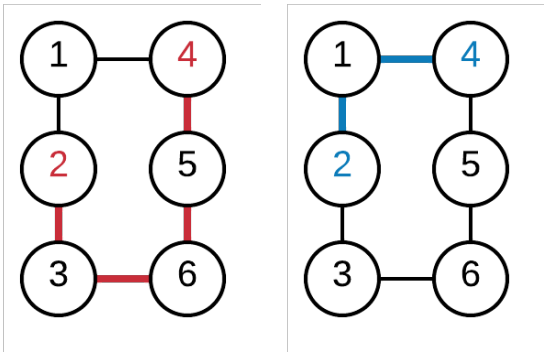


Fig. 4. Caminho de 4 para 2 com DFS partindo de 1

D. Main

Classe principal encarregada de fazer a leitura dos dos arquivos recebido por parâmetro, instanciar as demais classes e invocar os métodos necessários para a resolução do problema.

III. Análise de Complexidade

As abordagens utilizadas no cálculo do **Tempo de Vantagem** possuem diferentes complexidades de tempo, sendo que todas elas individualmente exibem um melhor de desempenho quando comparadas a um método genérico, como por exemplo a aplicação da busca em largura de todos para todos na obtenção das distâncias entre os vértices. A seguir serão analisadas a complexidade em termos de tempo e espaço.

A. Complexidade de Tempo

A identificação se utiliza apenas de um DFS para localizar cada uma das componentes conexas, que representam as naves, além de outras operações que possuem de tempo constante. Deste modo, a complexidade de tempo para identificação é $\mathcal{O}(V+E)$. Para as naves de Reconhecimento, Frigatas e Transportadores, essa complexidade se mantém em $\mathcal{O}(V)$. Já para os transportadores que possuem uma quantidade de arestas equivalente à multiplicação dos nodos de uma partição pela outra, a complexidade pode ser representada por $\mathcal{O}(E)$ no pior caso.

Já o cálculo para os tempos de vantagem utiliza métodos diferentes para cada tipo de grafo, sendo que cada um possui a seguinte complexidade:

- Transportadores: utiliza um DFS para calcular a distância inicial a partir da raiz, e após isso o cálculo das distâncias entre os vértices leva tempo constante, obtendo complexidade de tempo de $\mathcal{O}(V+E) = \mathcal{O}(V)$, já que o número de vértices é igual ao número de arestas;
- Bombardeiros: no pior caso, dependendo da distribuição das partições, ele explora quase toda a lista de vértices da nave para atribuição das partições. Já o cálculo das distâncias é feito em tempo constante, deste modo sua complexidade de tempo é $\mathcal{O}(V)$;
- Frigatas: primeiramente é necessário fazer uma busca em largura para obtenção distâncias a partir de um nodo raiz. Já as informações necessárias para o cálculo do LCA são obtidas no DFS da identificação das naves, deste modo, o cáculo do LCA para cada par de vértices possui complexidade de tempo $\mathcal{O}(\sqrt{V})$. A distância deve ser calculada uma vez para cada vértice do grafo, obtendo-se então uma complexidade de $\mathcal{O}(V\sqrt{V})$, além de mais algumas operações em tempo constante, logo a complexidade para as Frigatas é de $\mathcal{O}(V\sqrt{V})$;
- Reconhecimento: basta executar uma busca em profundidade em tempo $\mathcal{O}(V+E)$, a partir de um vértice da extremidade que leva no máximo $\mathcal{O}(V)$ para se encontrado, obtendo as distâncias, além de e algumas operações em tempo constante. Como o número de vértices da nave de Reconhecimento e maior que o seu número de arestas, podemos dizer que a complexidade de tempo do algoritmo é $\mathcal{O}(V)$.

B. Complexidade de Espaço

Quanto a complexidade de espaço, o **Radar** gera um **PostoDeCombate** para cada vértice no momento da leitura, o que gera uma complexidade em espaço de $\mathcal{O}(V)$. A partir desse momento todos os atributos necessários para a identificação da naves e o cálculo do **Tempo de Vantagem** estão contidos dentro dessa estrutura, fazendo com que o programa não exceda essa complexidade em termos de espaço.

IV. Análise Experimental

Para a análise experimental, algumas entradas foram geradas variando apenas a quantidade de vértices de 10 a 10^5, já que em todos os tipos de naves a quantidade de arestas é dependente do número de postos de combate. Deste modo, foi escolhida uma relação entre as distâncias de combate que desencadeassem no pior caso para cada classe. Em seguida foram analisados o gasto de memória e o tempo de execução, como mostram as tabelas a seguir.

1) Reconhecimento: Para as naves de Reconhecimento, a pior permutação possível é quando os os vértices dos extremos são trocados de lugar, fazendo com que as distâncias máximas possíveis sejam percorridas, como mostra a figura (5).

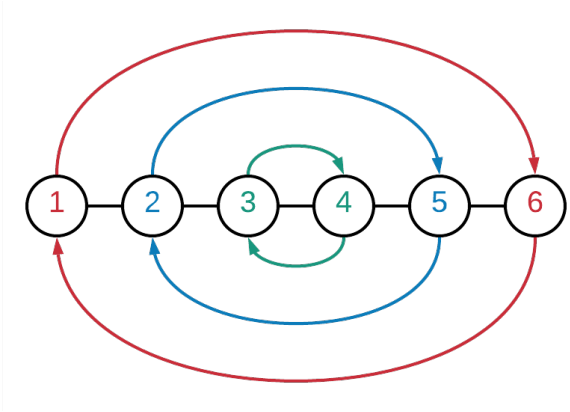


Fig. 5. Combinação que gera as maiores distâncias possíveis no Reconhecimento

TABLE I
COMPARAÇÃO DO TEMPO DE EXECUÇÃO (S) E O CUSTO DE MEMÓRIA (MB) EM FUNÇÃO DO NÚMERO DE VÉRTICES PARA O PIOR CASO DA NAVE DE RECONHECIMENTO

	Reconhecimento				
Vértices	10	100	1000	10000	100000
Tempo	0.002	0.006	0.041	0.089	0.299
Memória	0.999	1.011	1.140	2.356	14.380

2) Frigata: O pior caso da Frigata é o mesmo da nave de Reconhecimento, já que a distância máxima que pode ser obtida entre dois vértices é quando ambos se localizam nos extremos, como mostra a figura (7).

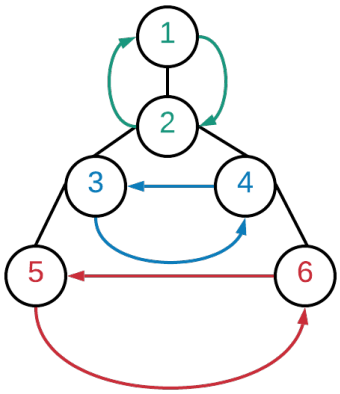


Fig. 6. Combinação que gera as maiores distâncias possíveis na Frigata

TABLE II
COMPARAÇÃO DO TEMPO DE EXECUÇÃO (S) E O CUSTO DE MEMÓRIA (MB) EM FUNÇÃO DO NÚMERO DE VÉRTICES PARA O PIOR CASO DA FRIGATA

	Frigata				
Vértices	10	100	1000	10000	100000
Tempo	0.003	0.006	0.034	0.158	0.528
Memória	1.002	1.015	1.144	2.360	14.384

3) Bombardeiro: No seu pior caso, o Bombardeiro possui os vértices distribuídos de maneira equilibrada entre as duas partições. Deste modo, o número de arestas que é igual a $V_{part1} \times V_{part2}$, totalizando em $(\frac{V}{2})^2$, ou seja, $\mathcal{O}(V^2)$. No entanto essa configuração não aumenta o **Tempo de Vantagem**, apenas sua complexidade de espaço para gerar um numero tão grande arestas. Deste modo, como o limite de arestas é de 10^6, não já necessidade de testar essa configuração, logo os testes serão realizados na configuração que minimiza o total de arestas, mantendo apenas 2 postos em uma partição, e o resto na outra, como mostra a figura (7).

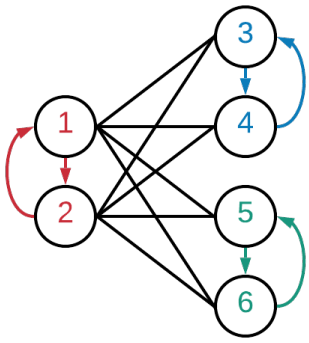


Fig. 7. Combinação que gera as maiores distâncias possíveis no Bombardeiro

TABLE III
COMPARAÇÃO DO TEMPO DE EXECUÇÃO (S) E O CUSTO DE MEMÓRIA (MB) EM FUNÇÃO DO NÚMERO DE VÉRTICES PARA O PIOR CASO DO BOMBARDEIRO

	Bombardeiro				
Vértices	10	100	1000	10000	100000
Tempo	0.003	0.007	0 0.035	0.124	0.374
Memória	1.012	1.026	1.162	2.476	1.5212

4) Transportador: O transportador possui um pior caso quando o vértice é permutado com o seu exato oposto, percorrendo metade do número total de vértices para todos os nodos, de acordo com a figura (7).

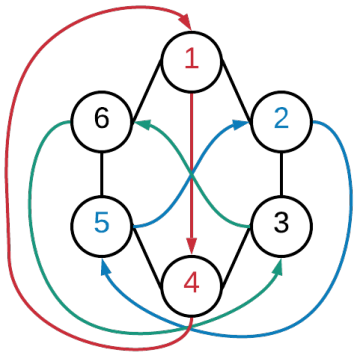
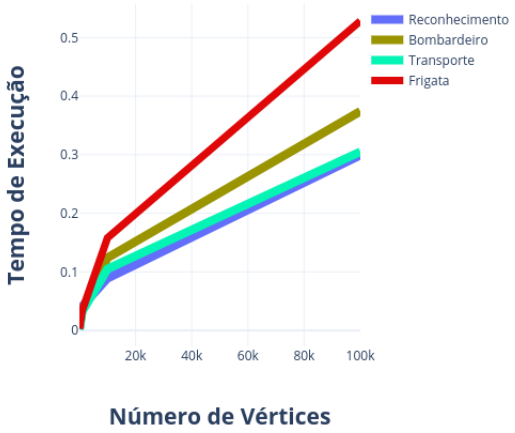


Fig. 8. Combinação que gera as maiores distâncias possíveis no Transportador

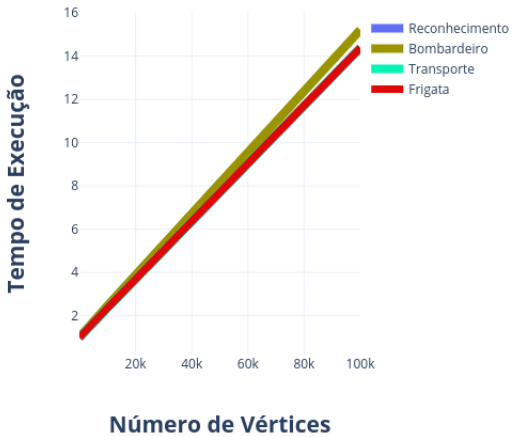
TABLE IV
COMPARAÇÃO DO TEMPO DE EXECUÇÃO (S) E O CUSTO DE MEMÓRIA (MB) EM FUNÇÃO DO NÚMERO DE VÉRTICES PARA O PIOR CASO DO TRANSPORTADOR

	Transportador				
Vértices	10	100	1000	10000	100000
Tempo	0.002	0.006	0.032	0.105	0.305
Memória	1.003	1.015	1.144	2.359	14.384

E por fim, estão apresentados dois gráficos compilando os dados obtidos nas amostragens anteriores. Primeiramente o crescimento do tempo em função do número de vértices.



Os resultados corroboram com a análise de complexidade, que atribuiu o maior custo em termos de tempo ao cálculo da Frigata, os demais estão próximos com uma leve desvantagem para o Bombardeiro.



No segundo gráfico, temos os resultados em termos de espaço, que também estão de acordo com o que foi obtido na análise de complexidade, com os três tipos de naves bem próximos, e uma leve desvantagem para o Bombardeiro, que se deve à sua maior quantidade de arestas quando comparado às demais naves.