UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Cecília Regina Oliveira de Assis

Reconhecimento de Naves

Trabalho Prático - Grafos

Minas Gerais, Brasil

Maio de 2019

Sumário

1 Introdução

1.1 Descrição do problema

O problema a ser modelado e resolvido, utilizando grafos, no trabalho se dividiu em duas etapas:

- A primeira foi a fase de **reconhecimento das naves**, onde uma frota de naves inimigas deveria ser reconhecida quanto a sua categoria, a partir de sua estrutura. Para cada uma das naves foi disposto um conjunto de entrada que enumerava um arranjo de postos de combate e suas conexões, que por sua vez caracterizavam as possibilidades de teleporte entre posições.
- A segunda foi a etapa de cálculo do limite inferior do **tempo de vantagem**. Aqui, a pergunta a ser respondida era: quão rápido toda a tripulação de qualquer nave consegue chegar em seu posto de combate?

1.2 Modelagem do problema

A modelagem do problema empregando a condição de grafos foi bastante intuitiva. Sendo $G = (V, E)$ a frota inimiga, os postos de combate representaram o conjunto $V[G]$ e as conexões entre eles $E[G]$, utilizando a conotação adotada por Cormen et al. (2009).

2 Desenvolvimento

O trabalho foi desenvolvido em `C++` utilizando, de modo geral, as estruturas convencionais dispostas pela linguagem, e.g. *vetores*, *filas* e *structs*.

O grafo foi construído através de *vetores de vetores*, sendo cada índice do mesmo um posto de combate e cada posição sua lista de adjacências. Já as naves foram representadas como *structs* nomeadas `Ship`, com informações quanto suas arestas, vértices, número de folhas, tipo e identificação (Cód. 2.1).

```
struct Ship {
    int id;
    int type;
    int paths = 0;
    int leaves = 0;
    vector<int> posts;
};
```

Exemplo de código 2.1 – Estrutura de representação das naves

2.1 Visão geral do algoritmo

Para cada seção do algoritmo uma função foi criada a fim de organização, de forma que o arranjo da função principal foi a seguinte, onde o nome de cada função se encontra entre parênteses:

1. Leitura do número de vértices e arestas
2. Criação do grafo (`make_graphs()`)
3. Reconhecimento das naves (`recognize_ships()`)
4. Armazenamento das trocas de posto (`store_changes()`)
5. Cálculo do limite inferior do tempo de vantagem (`their_advantage()`)
6. Exposição da frota, i.e, a saída do algoritmo (`expose_fleet()`)

É importante salientar que o código foi complementante escrito em inglês, conforme é possível observar a partir da nomenclatura das funções.

2.2 Reconhecimento das naves

Para resolver essa etapa, cada nave foi analisada enquanto uma componente conexa e o algoritmo DFS empregado.

Ambos os algoritmos de busca em grafos poderiam ter sido adotados, porém o DFS foi o escolhido devido sua propriedade de visitar um vértice no momento em que ele é descoberto, facilitando então o caminhar pelas naves de forma direta.

Já o conceito de componentes conexas adveio do fato de, trivialmente, existirem caminhos entre quaisquer pares de postos da mesma embarcação.

2.2.1 Critérios de classificação

Cada nave foi analisada e então classificada de forma individual, sendo empregados quatro critérios mutualmente exclusivos:

1. Caso a nave apresentasse duas folhas, a mesma era então classificada enquanto nave de **reconhecimento**;
2. Ao passo que se possuísse o mesmo número de vértices e arestas, o veículo era então **transportadora**;
3. Se exibisse uma quantidade de vértices menor que seu montante de arestas, tal nave era **bombardeiro**;
4. Não se encaixando em nenhuma das mencionadas possibilidades, **frigata**.

2.3 Tempo de vantagem

De forma similar a etapa de reconhecimento, o cálculo do tempo de vantagem também foi operado de forma distinta para cada nave:

- **Reconhecimento** seguindo basicamente a estrutura de uma fila encadeada, ao apresentar nós sequenciais, o cálculo do tempo de vantagem desta nave obedeceu o valor exibido pelo módulo da diferença entre os vértices da mudança de postos.
- **Frigata** por ser uma árvore, o algoritmo LCA (Lowest Common Ancestral), que busca pelo menor ancestral em comum entre os vértices (SIAUDZIONIS, 2016) foi empregado.
- **Transportadora** esta nave possuía em sua estrutura um ciclo, logo, para o cálculo do tempo de vantagem neste veículo duas possibilidades foram avaliadas e a melhor

delas escolhida, a primeira abordando o uso dos vértices do ciclo e a segunda a situação contrária.

- **Bomberdeiro** sendo um grafo bipartido completo, após a coloração de cada vértices, também duas possibilidade eram analisadas. Caso as posições da troca fizessem parte da mesma partição da nave, então a menor distância entre eles apresentava duas unidades, caso contrário somente uma.

Por fim, após a avaliação de todas as naves, o menor tempo de vantagem conhecido foi divido por dois, posto que efetuar o teleporte de um integrante da tripulação encurtava em uma unidade a distância de outro do seu posto de destino.

3 Discussão

3.1 Análise de complexidade

Por tarefa da resolução, têm-se:

Tabela 1 – Análise de complexidade de Tempo e Espaço do algoritmo

	Tempo	**Espaço**
Construção do Gráfico	$O(\|V\| + \|E\|)$	$O(\|V\| + \|E\|)$
Reconhecimento das naves	$O(\|V\| + \|E\|)$	$O(\|V\| + \|V\| + \|V\| + \|V\| + \|E\|)$
Tempo de vantagem: Reconhecimento	$O(\|V\|)$	$O(1)$
Tempo de vantagem: Frigata	$O(\|V\|^2)$	$O(1)$
Tempo de vantagem: Transportadora	$O(\|V\|)$	$O(1)$
Tempo de vantagem: Bomberdeiro	$O(\|V\|)$	$O(1)$
Total	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$

Para o reconhecimento das naves foram necessários três vetores para armazenamento das partições, dos níveis e dos pais de cada vértices e mais quatro, que em seu total ocuparam espaço $O(|V| + |E|)$, para armazenamento das naves.

O tempo de vantagem ocupou espaço $O(1)$, dado que todo seu pré-processamento já havia sido realizado, de forma que somente a soma das distâncias precisou ser alocada. Tal fase, para a *frigata*, foi aquele que representou maior tempo de execução, em virtude da complexidade $O(|V|)$ para cada chamada do algoritmo LCA.

Finalmente, a complexidade total de execução foi $O(|V|^2)$ em função do cálculo do tempo de vantagem da *frigata* e em espaço $O(|V| + |E|)$ por essa figurar os termos de maior ordem.

3.2 Análise experimental

Para discussão dessa seção, a tabela 2 apresenta a quantidade de vértices e arestas de cada entrada dos casos de teste:

Tabela 2 – Metadados dos casos de teste

	# Vértices	**# Arestas**
1.in	165	546
2.in	21	20
3.in	999	992
4.in	2435	10000
5.in	10000	9524
6.in	11823	100000
7.in	42758	100000
8.in	22436	100000
9.in	36407	1000000
10.in	100000	95626
pdf1.in	15	14
pdf2.in	19	18

O software Valgrind (2000-2019) foi adotado para cálculo da alocação de memória para cada um dos arquivos acima, enquanto um gráfico de com cinquenta execuções explicita o tempo decorrido na operação de caso.

Conforme apresentam as figuras 1, 2, sendo para o gráfico de tempo cada linha/cor uma execução, o caso "`9.in`" foi o mais custo tanto no tempo quanto na alocação de memória, tendo este a maior quantidade de aresta dentre os demais.

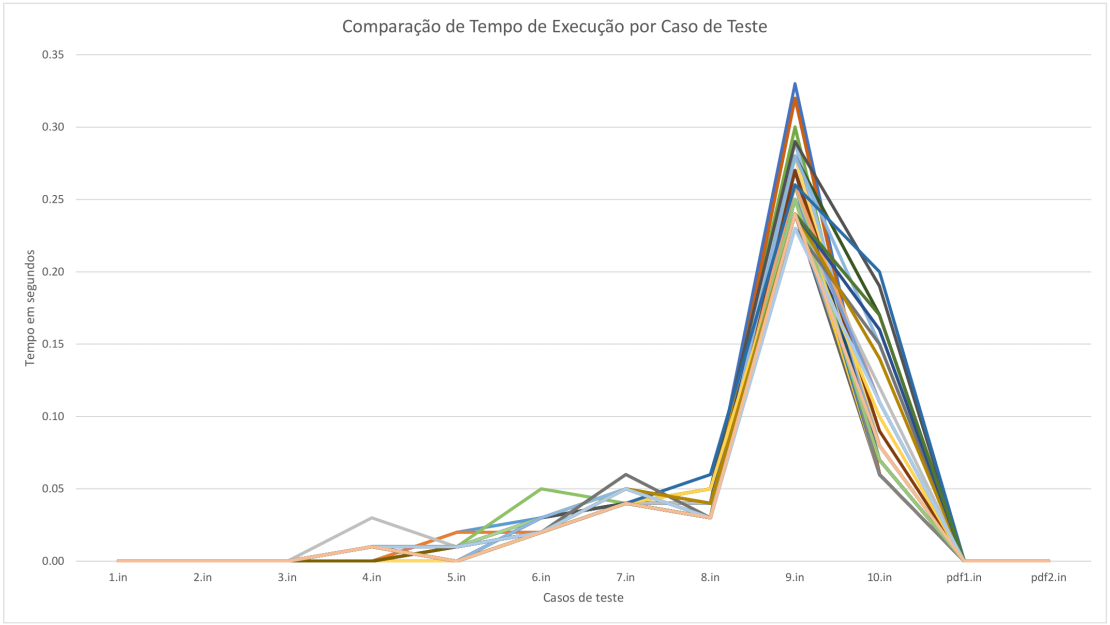

Figura 1 – Comparação de Tempo de Execução por Caso de Teste

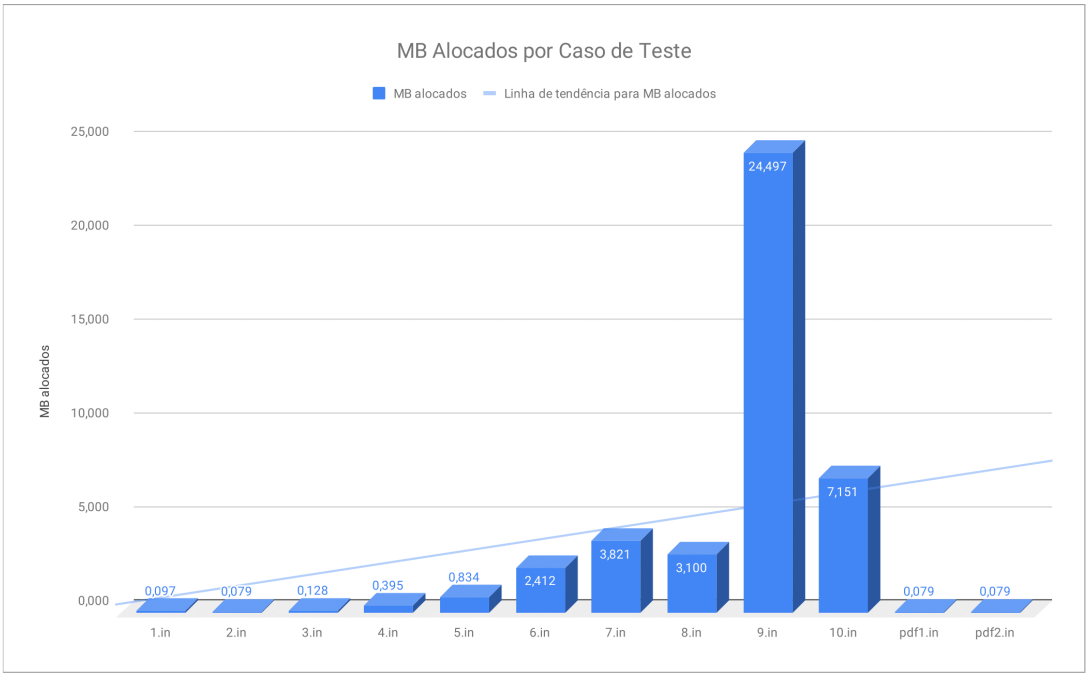

Figura 2 – Comparação de Alocação de Memória por Caso de Teste

Referências

CORMEN, T. H. et al. *Introduction to algorithms*. [S.l.]: MIT press, 2009. Citado na página 2.

SIAUDZIONIS, L. Menor ancestral comum. 2016. (Accessed on 05/12/2019). Citado na página 4.

Valgrind. *Valgrind*. 2000–2019. Disponível em: <http://valgrind.org>. Citado na página 7.