Algoritmos 1 - TRABALHO PRÁTICO 1:
O JOGO DO PULO

Letícia da Silva Macedo Alves < leti.ufmg@gmail.com>

Matrícula: 2018054443

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brasil

1. Introdução

O problema especificado refere-se ao Jogo do Pulo, cujo objetivo é, dado um tabuleiro (com valores de pulo em cada casa, que determina quais casas podem ser atingidas a partir de dada casa), jogadores e suas respectivas posições iniciais, quer-se atingir a última casa do tabuleiro, que está na posição (M-1,N-1), sendo M o número de linhas e N o número de colunas. Além disso, se um jogador cai em uma posição cujo valor de pulo é zero, não será possível sair dela. As movimentações podem se feitas apenas na horizontal e vertical e a ordem inicial dos jogadores é dada pela ordem em que são lidos na entrada. O jogo acaba de duas formas: sem vencedores ou com um vencedor.

É proposto então o desenvolvimento de um programa para o Jogo do Pulo. Para um determinado tabuleiro bidimensional, um número de jogadores com posição e ordem inicial, será determinado qual será o jogador vencedor (se houver) do jogo e em qual rodada. Vale ressaltar que o programa é desenvolvido de forma que os jogadores sempre têm sua movimentação de modo a maximizar a sua chance de vitória. A partir da entrada fornecida (número M de linhas, N de colunas do tabuleiro e número K de jogadores), um Jogo é criado. Um Jogo consiste de um tabuleiro (Grafo representado por uma Lista de Adjacência) e um vector de K jogadores.

Para determinar o resultado, a função resultadoJogo() é chamada e serão realizadas as etapas necessárias (como comparações e Buscas em Largura - BFS) detalhadas no decorrer deste documento.

O resultado na saída podem ser dois possívesl: ou serão duas linhas, sendo elas o nome do jogador vencedor (uma letra maiúscula) e em quantas rodadas venceu, ou a mensagem “SEM VENCEDORES”.

Na realização deste projeto, a linguagem utilizada foi C++11 e foram implementadas as classes: Jogo, Grafo e Jogador. A forma de encontrar os caminhos no tabuleiro até a posição de vitória foi o algoritmo de Busca em Largura (BFS), que foi considerada adequada, atendendo perfeitamente ao que se esperava da solução do problema.

2. Implementação

O formato de entrada, como o proposto, é por passagem dos arquivos de teste como parâmetro para o programa via linha de comando e a saída utiliza para imprimir o resultado o padrão de sistema stdout (cout). Além disso para a verificação de alocação e desalocação correta de memória, foi utilizado o Valgrind.

2.1. A classe Jogo

A classe jogo possui os atributos: *vector<Jogador>* jogadores* e *Grafo* tabuleiro*.
Além disso possui o método void *resultadoJogo(ifstream& entrada)*, que recebe como parâmetro o arquivo de entrada e dá início ao processo de definir o vencedor (se houver) e o número de jogadas realizadas para essa vitória. Esse método é chamado na *main()*.

2.1.1 O método resultadoJogo

Este método faz para cada jogador dentre os K: começa lendo do arquivo de entrada os valores (x, y) da posição inicial (convertidos para um único inteiro, que se refere à representação de cada

vértice do tabuleiro através de um vector), atribui ao jogador essa posição, e *vector<int> paramentrosJogador* recebe os três valores de retorno de uma BFS e os coloca nos três atributos restantes do respectivo jogador. Após esse processo ser realizado para todos os jogadores, é o momento de determinar o resultado do jogo. Uma série de comparações é feita sobre todos os atributos dos jogadores, determinando: se houve vencedor (em caso negativo printando SEM VENCEDORES) e, se houve, quem foi e em qual rodada (printando a letra que é o nome desse vencedor e na linha abaixo o valor da jogada).

2.2. A classe Grafo

A classe Grafo é utilizada, no contexto do problema, na representação do tabuleiro, isto é, o tabuleiro do jogo será um grafo (representado como Lista de Adjacência). Possui os atributos: *int M, N, numVertices* (sendo M e N as dimensões do tabuleiro), *vector<list<int>>* listaAdj, vector<int>* valoresPulo* (que guardará os valores de pulo de cada casa do tabuleiro). Dentre seus métodos se destacam: *void criaListaAdj(ifstream& entrada)* (que lerá do arquivo de entrada os valores de pulo de cada casa e adicionará a vértice as demais posições de tabuleiro que por ele podem ser atingidas) e *vector<int> BFS(int casaInicial, int casaFinal)* (que realiza a Busca em Largura na Lista de Adjacência sendo a raiz, isto é, casaInicial, a posição na qual o jogador começou o jogo e a casaFinal a de posição (M-1,N-1) no tabuleiro, na qual um jogador vence o jogo).

2.2.1 O método criaListaAdj

Responsável por criar a Lista de Adjacência (no formato vector de lists) que representa o tabuleiro. Isto é cada uma das M*N posições do vetor representa um vértice/casa do tabuleiro. Cada uma dessas casas possui uma lista, que são as outras casas atingíveis por ela, dado seu valor de pulo.

2.2.1 O método BFS

Realiza uma Busca em Largura dado um vértice de inicio (casaInicial) e o vértice que se pretende atingir (casaFinal). Começa então com todos os vértices ainda não descobertos, com exceção da raiz, que aparece na primeira posição da fila. À fila é adicionado todo vértice alcançável a partir do vértice que está na primeira posição e que ainda não está marcado como descoberto (retirando também da frente da fila este nó atual que está sendo explorado) Para isso, itera-se pela Lista de Adjacência do Grafo. Nessa busca fica também determinado o nó pai (que foi responsável pela descoberta de um dado nó) e o nível/camada ao qual um nó pertecence na busca BFS, sendo que a raiz encontra-se na camada 0. Ao final da busca, é verificado se foi ou não possível atingir a casaFinal e os parametros do jogador correspondentes são retornados (haCaminho, tamCaminho, penultimaCasa).

2.3. A classe Jogador

A classe Jogador possui os seguintes atributos: *int posicaoInicial, int haCaminho* (que receberá 1 se, após a BFS, for constatado que o jogador possui um caminho para a última casa do tabuleiro partindo de sua posicaoInicial*), int tamCaminho* (o tamanho deste caminho até o final, caso existe, isto é, a quantidade de arestas no grafo para se chegar à vitória) e *int penultimaCasa* (isto é, em qual casa o jogador estava duas casas antes de chegar, se possível, ao final). Todos esses atributos são usados para determinar se houve vencedor e qual foi esse vencedor (obedecendo os critérios de desempate, caso mais de um jogador possua um caminho de mesmo tamanho até o final: vence aquele que o penultimaCasa possui menor valor. Caso esse atributo empate também, vence o jogador que jogou primeiro na primeira rodada).

3. Instruções de compilação e execução

A compilação é realizada utilizando o compilador GCC. Como o projeto faz uso de Makefile, o processo de compilação dos códigos-fonte é resumido a um único comando "make". Ou seja, uma possível sequência de comandos para compilação e execução do programa em sistema Unix seria:

"$ make"

"$ ./tp1 jogo_do_pulo.txt"

4. Análise de Complexidade

4.1.Tempo

Primeiramente, vale ressaltar que operações como atribuição, passagem de parâmetro, leitura, chamada de função, comparações, retornos, operações aritméticas simples, combinação de resultados e as demais do programa que não envolvem diretamente loops, são desconsideradas na análise de complexidade por serem O(1) e não serão diretamente citadas.

A função main cria um Jogo e chama seu método *resultadoJogo()*. Este método possui um loop for que é O(K) que realiza comparações entre os K jogadores. O método possui tamém outro loop for O(K). Mas para cada K deste último será chamado um método *BFS()* de Grafo. Em BFS há um loop for O(M*N), isto é, O(n) sendo n o número de vértices do Grafo. Nesse loop ocorrem apenas atribuições de valores a posições de vetores. BFS possui também o algoritmo de Busca em Largura (BFS) que ocorre dentro de um while em O(m+n), já que é percorrida a lista de adjacência, onde m é a quantidade de arestas do Grafo. Sendo assim o algoritmo possui complexidade temporal de O(K) + (O(K)*(O(n)+O(m+n)) = O(K) + O(K*n) + O(K*m + K*n) = O(K*m + K*n).

4.2.Espaço

Consideremos que operações que não envolvem diretamente alocações e manipulações de estruturas como vetores e listas podem ser ignoradas na análise de complexidade, uma vez que são O(1) em memória. A função main aloca um Jogo. A classe Jogo possui os atributos *vector<Jogador>* jogadores*, que é O(K) e *Grafo* tabuleiro*. A classe Grafo possui um *vector<list<int>>* listaAdj* que é O(m+n), um *vector<int>* valoresPulo* que é O(M*N), ou seja, O(n), além de um *vector<int> parametrosJogador* que utiliza memória auxiliar de O(3) no método BFS. Possui também os vetores *camada, descobertos e pais*, todos O(n) e uma *list<int> fila*, que ocupará um espaço O(n).

Logo, o programa de maneira geral possui complexidade de espaço O(K) + O(m+n) + O(n) + O(3) + 3*O(n) + O(n) = O(K+m+n), sendo K o número de jogadores m o número de arestas e n o número de vértices.

5. Análise Experimental

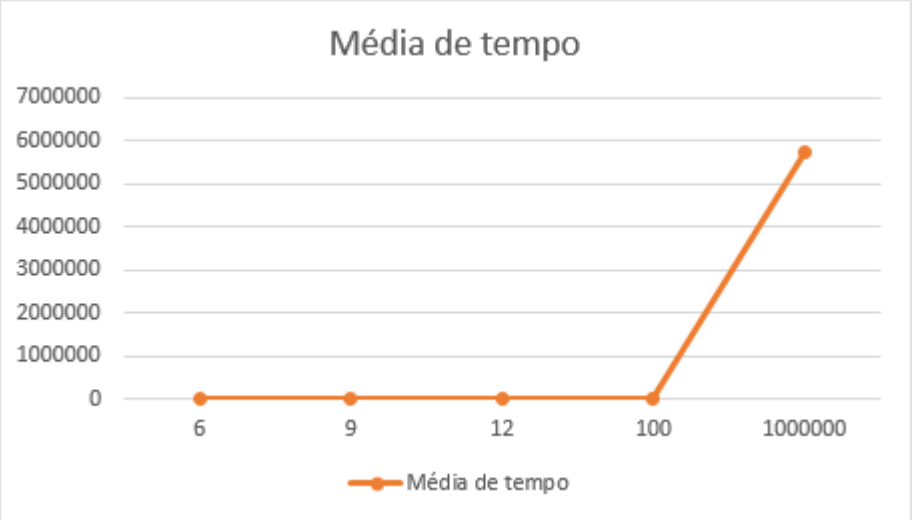


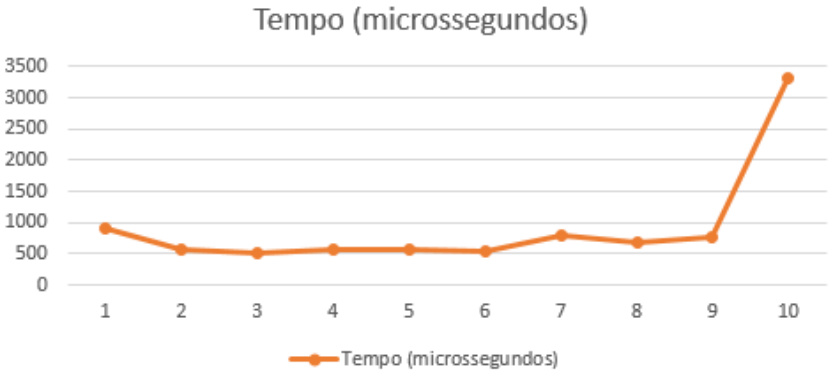
*Gráfico de média de tempo de execução (eixo y) realizada com 10 iterações do programa para cada tamanho de entrada (eixo x, sendo o tamanho de entrada M*N, onde M é a quantidade de linhas do tabuleiro e N a de colunas)

Com a análise experimental, é observável que o programa executa em tempo semelhante para entradas pequenas e médias e aumenta quase linearmente conforme crescem entradas muito grandes. Isso é reflexo de sua complexidade temporal, citada anteriormente. Para esta análise foi desconsiderado o número de jogadores como uma variável, uma vez que foram sempre semelhantes e baixos.

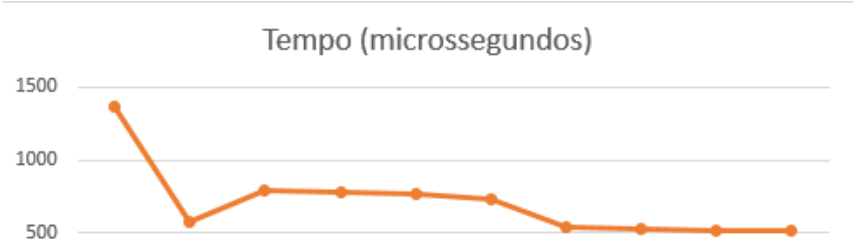
Foi observado também que, dado a solução proposta, o tempo de execução não é influenciado pelo fato de haver ou não vencedor no Jogo do Pulo. Isto é, o tempo para jogos com vencedores é semelhante ao tempo de jogos sem vencedores.

Segue abaixo os gráficos geradores do gráfico acima juntamente com os valores de média e desvio (sendo o eixo x a iteração do algoritmo e o eixo y o tempo de execução em microssegundos):

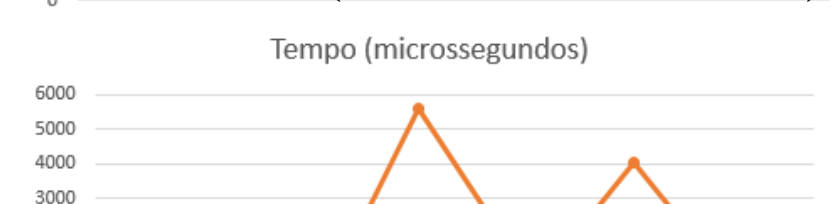
• Tamanho 6 de entrada: (Média = 712,4; Desvio = 257,8795)



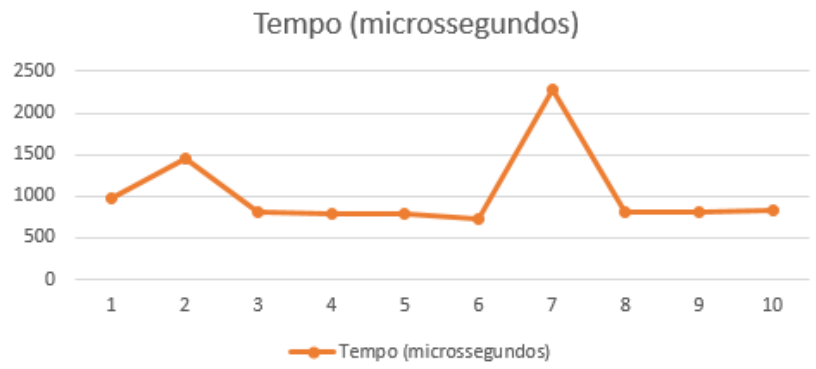
• Tamanho 9 de entrada: (Média = 921,5; Desvio = 852,133)



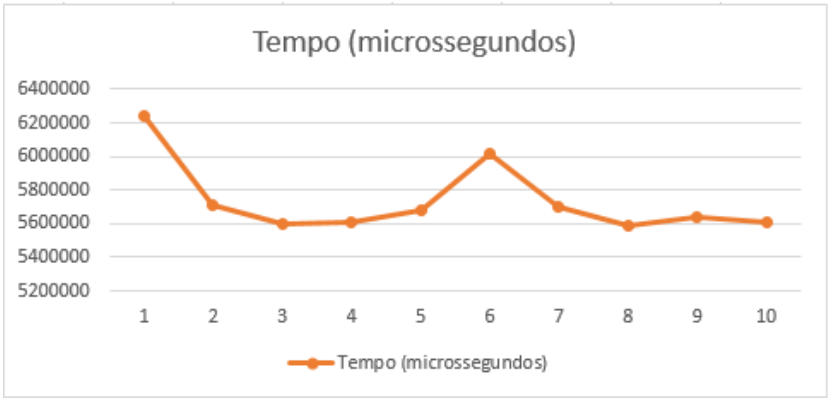
• Tamanho 12 de entrada: (Média = 2257,1; Desvio = 1432,506)



• Tamanho 100 de entrada: (Média = 1029,3; Desvio = 483,952)



• Tamanho 1000000 de entrada: (Média = 5739768,2; Desvio = 216738,369)



6. Conclusão

A representação do tabuleiro como um Grafo e o algoritmo de Busca em Largura mostraram-se adequados no processo determinação dos resultados finais do jogo em termos de tempo e eficiência para atender ao que foi proposto nas especificações. O projeto possibilitou a exploração das diferentes matérias vistas até o momento no curso, como algoritmo de busca percorrendo a estrutura de um grafo, a representação de um grafo como lista de adjacência, bem como modularização, alocação dinâmica, uso de memória e complexidade de algoritmo.

Bibliografia

KLEINBERG, Jon; TARDOS, Éva. (2005) “Algorithm Design”, Addison-Wesley.