Primeiro Trabalho Prático de Algoritmos e Estruturas de Dados III

Dilvonei Alves Lacerda Junior, Rafael Cardoso Ferreira

Professor: Leonardo Rocha

Departamento de Ciências da Computação – Universidade Federal de São João del-Rei - UFSJ – Campus Tancredo Neves

São João del-Rei – MG – Brasil

1. Introdução

O jogo de damas foi inventado a mais de 4000 anos atrás e possui inúmeras variações ao redor do mundo. Trata-se de um jogo para dois jogadores, cujo o tabuleiro é quadrado 8x8, pequenos quadrados que são alternadamente coloridos com uma cor clara e uma cor escura (os quadrados de cor escurar é que são as casas). No entanto o jogo de damas do vovô, proposto neste trabalho, difere em alguns pontos, o tabuleiro é retangular com N linhas e M colunas, ou seja, nem sempre é um quadrado, e o número de pequenos quadrados pode mudar a cada novo jogo. No início do jogo cada jogador tem um número de peças posicionadas nas casas mais próximas da borda, que são sempre bordas opostas para cada jogador.

Um dos movimentos do jogo é capturar uma peça do oponente, saltando sobre ela, diagonalmente, para a casa adjacente além da peça, casa esta que deve estar vazia. A peça do oponente é então removida do tabuleiro. As três casas envolvidas na captura (a casa inicial, a que contém a peça do oponente e a vazia, onde sua peça estará após a jogada) devem estar diagonalmente alinhadas e devem ser diagonalmente adjacentes, e não se pode salta sobre uma peça sua. Na Dama do Vovô as peças podem ser capturadas diagonalmente para frente e para trás, também podemos fazer uma captura múltipla com apenas uma peça saltando primeiro em uma direção depois em outra, podendo ser capturado apenas uma peça em cada salto e várias peças com os saltos seguidos.

O presente trabalho tem por objetivo determinar a melhor jogada que se pode fazer na Dama da vovó. Dadas as dimensões do tabuleiro N X M, e uma descrição do estado corrente de um jogo, na sua jogada, determinar computacionalmente o número máximo de peças do seu oponente que podem ser capturadas em um movimento de captura.

Assim, é preciso nesse trabalho implementar duas estratégias para solucionar esse problema:

• Uma estratégia força bruta;

• Uma estratégia alternativa que seja capaz de sempre resolver o problema, tentando o mínimo de movimentos;

1.1. Algoritmos força bruta:

1.1.2 Força bruta:

O método de força bruta é um algoritmo de implementação simples, que consiste em enumerar todas as possibilidades. Entretanto o custo do forçar bruta é proporcional ao tamanho da entrada, que no nosso problema do jogo de damas tende a crescer exponencialmente. O forçar bruta é utilizado quando o tamanho da entrada é limitado, ou não se conhece o um algoritmo mais eficiente.

1.1.3 Estratégia alternativa:

O algoritmo

2. Implementação

A

2.1 Estruturas de Dados

Duas TADs foram criadas: uma chamada tipo_peca e outa chamada tabuleiro. A estrutura tipo_peca possui como atributo dois inteiros que chamamos como “linha e coluna” que guardam a posição daquela peça no tabuleiro, um inteiro chamado “max” que guarda o máximo de jogadas que aquela peça pode realizar e um tamanho, que chamamos de “tam” que é o número de peças que são representadas por 1. A estrutura tabuleiro contem o tamanho dois inteiros que guardas o tamanho desse tabuleiro, NxM, que chamamos de “linha” e “coluna”, um vetor de inteiro que guarda o estado atual desse tabuleiro e o tamanho desse vetor de estado.

2.1.1 Funções.

2.2.1 int cria_tabuleiro(int linhas, int colunas)

Recebe o numero de linhas e colunas (NxM) e aloca todas as linhas e colunas do tabuleiro, NxM , que é uma matriz alocada dinamicamente, a cada entrada diferentes temos uma matriz com um tamanho diferente.

2.2.2 int *ler_arquivo (int *entrada, int *tam_entrada)

Ler o arquivo até o final e guarda todos os dados em um vetor de inteiros que chamamos de “entrada”, ao final retorna o arquivo em um vetor. Portanto para arquivos com uma entrada muito grande nosso condigo apresenta algumas inconsistências, podendo estourar a memória.

2.2.3 tabuleiros* separaJogos(int* ent, in tamanho)

Recebe o vetor entrada gerado na função de 2.2.2 int ler_arquivo e o tamanho do vetor, e separa todos o tamanho de cada jogo e seus respectivos estados atuais, comparando se a entrada é maior igual a 3, é linha e colunas, se não é parte do estado atual do jogo, a função retorna uma lista de tabuleiros.

2.2.4 int** geraTabuleiro(int linha, int coluna)

Recebe numero de linhas e colunas, utiliza a cria_tabuleiro(2.1.1) pra alocar, que é uma matriz do tamanho NxM de acordo entrada, e em todas as posições que são inacessíveis no tabuleiros colocamos ‘-1’ e 0 em todas posições que vão ser utilizadas, preenchemos com 0 que

são casas que podem ser utilizadas no tabuleiro(matriz), e o retorna no fim da função, a complexidade nessa função é de NxM que são os números de linhas e colunas.

2.2.5 tipo_peca *prenche_tabulerio(int linha, int coluna, int tam_estado, int *estado, int **matriz).

Recebe o estado de um jogo, e preenche a matriz (que é o nosso tabuleiro) com esse estado, e retorna um vetor do tipo_peca, para sabemos onde estão todas as peças do garoto no tabuleiro, que são as peças que nos interessa para saber o número máximo de jogadas, tendo sua complexidade de (NxM)/2.

2.2.6 int melhor_jogada(int **matriz, int linha, int coluna, tipo_peca *pecas)

Recebe a matriz com preenchida com um estado de jogo, o número de linhas e colunas do jogo, e onde estão todas as peças do garoto na matriz. Através do vetor do tipo_pecas, dessas forma chama função int testa_caminhos(2.2.7) para testar o máximo de movimentos de cada peça daquele jogo.

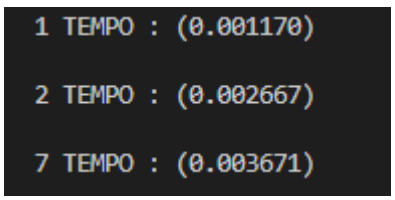
2.2.7 int testa_caminhos(int **jogo, int linha, int coluna, int linha_atual, int coluna_atual, int jogadas_atuais)

Recebe um jogo, o numero de linhas e colunas desse jogo e testa todos os caminhos possíveis para as peças do garoto, no tabuleiro. A posição da peça são os inteiros “linhas atual e coluna atual” recursivamente, testando todas as posições possíveis, respeitando as limitações do jogo, e quando chega no máximo de uma direção ela volta uma posição e verifica se tem caminhos possíveis ainda, até chegar na posição inicial, onde não tem mais possibilidades de movimentos.

3 Análise

O custo no melhor caso, ou seja, quando se tem somente uma peça no tabuleiro que pode se mover somente uma vez, O (1), mas no pior caso é O(n!).

Para as três entradas de exemplos obtivemos os tempos abaixo:



A direita é o melhor jogada da matriz 3x3 , 5x3 e 8x8 e a medida que temos uma entrada maior percebemos que o número de testes também aumenta , assim demorando mais tempo para ser executado.

4 Conclusão

Ao decorrer do trabalho percebemos que a forma mais intuitiva de resolvermos o problema de se obter o maior numero de jogadas é com algoritmo de força bruta, porem não é o melhor para ser implementado, pois consume muita memoria e tempo para testar todas as possiblidades, principalmente quando temos tabuleiros que são muito grandes, podendo estourar a memória.