Universidade Federal do ABC

Técnicas Avançadas de Programação – 2018.Q1

Profs. Daniel M. Martin e Francisco Isidro Massetto

Projeto 1: O Jogo dos Peões

Valor: 10 pontos. Entrega: 07/04

1 Descrição

Usando apenas 3 colunas de um tabuleiro de Xadrez, 3 peões pretos e 3 peões brancos, dois jogadores movimentam esses peões alternadamente até que um deles consiga colocar os seus peões nas casas inicialmente ocupadas pelos peões do adversário. A posição inicial do tabuleiro encontra-se na Figura 1 abaixo.

Figura 1: posição inicial das peças no tabuleiro.

Cada jogador, na sua vez, pode mover apenas um peão e este deve permanecer na mesma coluna. Só são permitidas jogadas em que o peão movimentado avança em direção ao campo do adversário. Um peão pode avançar um número arbitrário de casas livres. Se, no momento da jogada, ele estiver sendo bloqueado pelo peão do adversário, então ele pode "pular", mas deve parar na casa livre logo em seguida.

Pode acontecer que um jogador fique sem movimentos válidos na sua vez de jogar. Nesse caso, ele "passa" a vez para o adversário.

2 Tarefa

Sua tarefa é fazer um *bot* capaz de jogar este jogo, ou seja, você deve fazer um programa que serve como um jogador virtual que seja capaz de ganhar o jogo dos peões sempre que existir uma estratégia vencedora. Seu programa **não poderá ter interface gráfica**, mas deve ser feito para interagir com o usuário através da entrada e da saída padrões[1].

[1]Se você conseguiu a proeza de chegar a este curso sem saber o que é *entrada padrão* ou *saída padrão*, pesquise sobre isso e aprenda esses conceitos antes de fazer qualquer outra coisa! Dica: digite *standard input output* no Google.

3 Conceitos abordados

A resolução deste problema envolve conhecimentos de combinatória, teoria dos grafos, recursão e também o conceito de pré-processamento.

3.1 Combinatória

Em primeiro lugar, para estimar o número de estados desse jogo, é necessário conhecer o princípio fundamental da contagem.

3.2 Teoria de grafos

De cada uma das possíveis configurações do jogo, e dependendo de quem é a vez de jogar, é possível efetuar um conjunto de movimentos válidos que pode ser vazio. Portanto, é natural modelar o jogo como um grafo dirigido, onde os vértices do grafo são todas as possíveis configurações do tabuleiro e os arcos desse grafo possuem duas cores: os arcos brancos representam os movimentos válidos para o jogador dos peões brancos e os arcos pretos têm função análoga.

3.3 Pré-processamento

A eficiência do cálculo da estratégia ótima pode ser aumentada com o uso de tabelas pré-computadas com os valores de funções que são muito usadas.

3.4 Recursão

O algoritmo que encontra a estratégia vencedora (e o jogador que a possui) é essencialmente uma busca em profundidade no grafo de estados e, portanto, possui natureza recursiva.

4 Especificações

4.1 Entrada e saída

Como foi dito anteriormente, seu jogo deverá interagir com o usuário pela entrada e saída padrões. Portanto, é necessário estabelecer uma linguagem que o programa e o usuário deverão usar para que essa comunicação seja possível.

Toda linha da entrada (e também da saída) consistirá de dois números separados por espaço em branco: o primeiro número indica a coluna em que o jogador escolheu fazer seu movimento, e o segundo, quantas casas ele movimentou o peão daquela coluna em direção ao campo do adversário.

Vejamos a entrada e a saída correspondentes ao exemplo da Figura 2.

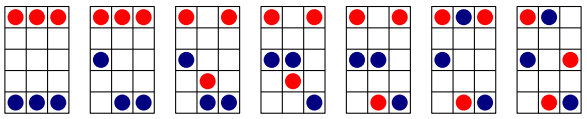

Figura 2: um exemplo de abertura do jogo.

De acordo com a especificação, se o computador (seu programa) está jogando com as peças vermelhas e o usuário com as peças azuis, então a troca de mensagens entre o usuário e o programa ficarão assim.

Entrada:

```
1 2
2 2
2 2
```

Saída:

```
2 3
2 1
3 2
```

No terminal fica assim:

```
1 2
2 3
2 2
2 1
2 2
3 2
```

4.1.1 Passando a vez

Se o usuário (ou o seu programa) precisar "passar a vez" por falta de opções de jogada, esse tipo de jogada será indicado por um par de números zero.

4.2 Linha de comando

Seu programa deve aceitar um argumento na linha de comando[2] que é uma de duas possibilidades `primeiro` ou `segundo`. Se o argumento `primeiro` estiver presente, seu programa deve começar jogando. Se o argumento `segundo` estiver presente, seu programa deve esperar (ler da entrada padrão) a jogada do usuário.

4.3 Comportamento

Seu programa deve **ganhar** o jogo sempre que isso for possível. O que significa isso? Significa que, se existir uma estratégia vencedora para o jogador virtual controlado pelo seu programa – **em qualquer dos estados intermediários que o jogo venha a atingir** – seu programa deve ganhar o jogo!

Para que você possa compreender melhor o que foi dito acima, vamos tomar como exemplo o Jogo da Velha. É claro que, jogando com atenção, é sempre possível forçar um empate no Jogo da Velha. Mas, supondo que você (X) esteja jogando com alguém (O) que deixou o jogo chegar no estado

```
X   O
  O
    X
```

e, supondo que é a sua vez de jogar, você não deve perder a chance de ganhar esse jogo! Assim, também, o programa que você fizer, deve se aproveitar de movimentos ruins que o oponente vier a fazer e *ganhar o jogo sempre que possível!*

[2] Com relação à *argumentos de linha de comando*, reitero os mesmos comentários e sugestões na nota de rodapé anterior.

4.4 Scripts para correção/competição

Serão disponibilizados pelo Tidia alguns scripts que você poderá utilizar para testar o seu programa (num ambiente Linux, é claro!).

Um dos scripts permite o jogo dos peões seja jogado por *dois* oponentes virtuais. Isso significa que seu programa poderá competir contra o programa de um colega seu! Ou ainda, que você possa colocar seu programa para jogar contra si mesmo, para ver se a estratégia vencedora está bem implementada. Que vença o melhor!

O segundo script faz uso do script anterior para competir seu programa contra um programa gabarito e testar se a estratégia usada pelo seu programa é uma estratégia vencedora. Esse script irá dar uma nota para o seu programa. A nota que que você receberá pelo projeto será aquela obtida quando o professor rodar esse script no programa que você submeter pelo Tidia.

5 Desafio (bônus)

Quem implementar **corretamente** o desafio proposto nesta seção ganhará mais 10 pontos de bônus na nota deste projeto.

O desafio consiste em fazer a mesma coisa proposta na tarefa original (seção 2), mas para um tabuleiro com 8 linhas e 5 colunas e 5 peões de cada jogador, inicialmente dispostos como na Figura 3 abaixo.

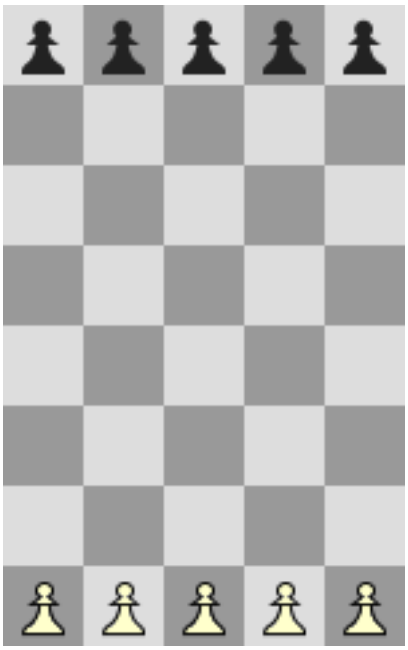

Figura 3: posição inicial da versão desafio.

5.1 Complicações

Provavelmente, se você implementar a mesma solução da tarefa original, você não conseguirá concluir o desafio porque o grafo de transição de estados não caberá na memória. Para conseguir calcular a existência de estratégias vencedoras você precisará tirar vantagem das simetrias do problema e construir um grafo reduzido na memória que, embora não represente todos os estados atingíveis pelo jogo, ele contém toda a informação necessária para a computação das estratégias.

Converse com o professor sobre como seria esse grafo reduzido!