MAC323 - Estruturas de Dados - Exercício Programa Otelo

Siang - 1999

Otelo, também conhecido por Riversi, é um jogo que usa um tabuleiros, com dois jogadores. Um jogador (vou chamar de Branco) joga com pedras brancas (denotadas aqui por "o") e outro chamado Preto joga com pedras pretas (denotadas por "x").

(Uma maneira prática é usar pedras coloridas brancas num lado e pretas noutro. Virando a pedra muda a sua cor.)

O tabuleiro é composto por 8 linhas e 8 colunas, totalizando 64 casas:

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

O jogo inicia-se com a seguinte disposição, com duas pedras de cada cor no centro:

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . o x . . .
. . . x o . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Preto começa primeiro e depois os dois adversários alternam suas jogadas, até que nenhum lado tenha uma jogada válida, ou até o tabuleiro estar totalmente preenchido. Então o jogador com mais pedras de sua cor ganha o jogo, podendo haver empate.

O que é uma jogada válida?

É colocar uma pedra no tabuleiro numa posição

1. que deve estar vazia antes da jogada,

2. ao colocar a pedra deve cercar uma ou mais pedras consecutivas do adversário, em linha horizontal, vertical ou diagonal.

Cercar significa ter no meio de duas pedras de mesma cor uma ou mais pedras consecutivas de outra cor. Note-se que uma das duas pedras *cercantes* deve ser a recentemente colocada através da jogada.

O resultado da jogada é mudar a cor das pedras cercadas para a sua cor.

Por exemplo, Preto pode jogar na posição abaixo, pois cerca uma pedra branca entre duas pedras pretas.

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . x o x . . .
. . . x o . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

resultando no seguinte:

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . x x x . . .
. . . x o . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Branco pode jogar agora por exemplo assim:

```
. . . . . . . .
. . . . . . . .
. . o . . . . .
. . x x x . . .
. . . x o . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

e com isso resulta em:

```
. . . . . . . .
. . . . . . . .
. . O . . . . .
. . X O X . . .
. . . X O . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Quando um jogador não consegue fazer uma jogada válida, então perde a vez e passa ao adversário.

O que faz o Exercício-Programa

O trabalho é escrever um programa para ser um dos jogadores do Otelo, digamos o Branco.

Para simplicar, dado o pouco tempo que sobra até o final do semestre, podem simplificar bem a parte visual, usando a mesma notação como neste enunciado.

Suponha que o adversário inicia a primeira jogada. O programa basicamente repete o seguinte até terminar o jogo:

1. exibe a configuação do tabuleiro,
2. recebe a jogada do adversário (em coordenadas),
3. exibe a configuação do tabuleiro,
4. escolhe a sua jogada (ver abaixo como fazer isso),
5. exibe a configuação do tabuleiro,
6. exibe a contagem das pedras de cada lado.

Para ficar mais simples, podem usar o método mini-max explicado abaixo.

(O método mais sofisticado é o alfa-beta, que dou uma breve explicação mas a sua implementação necessitaria mais estudos. Um artigo está referenciado abaixo.)

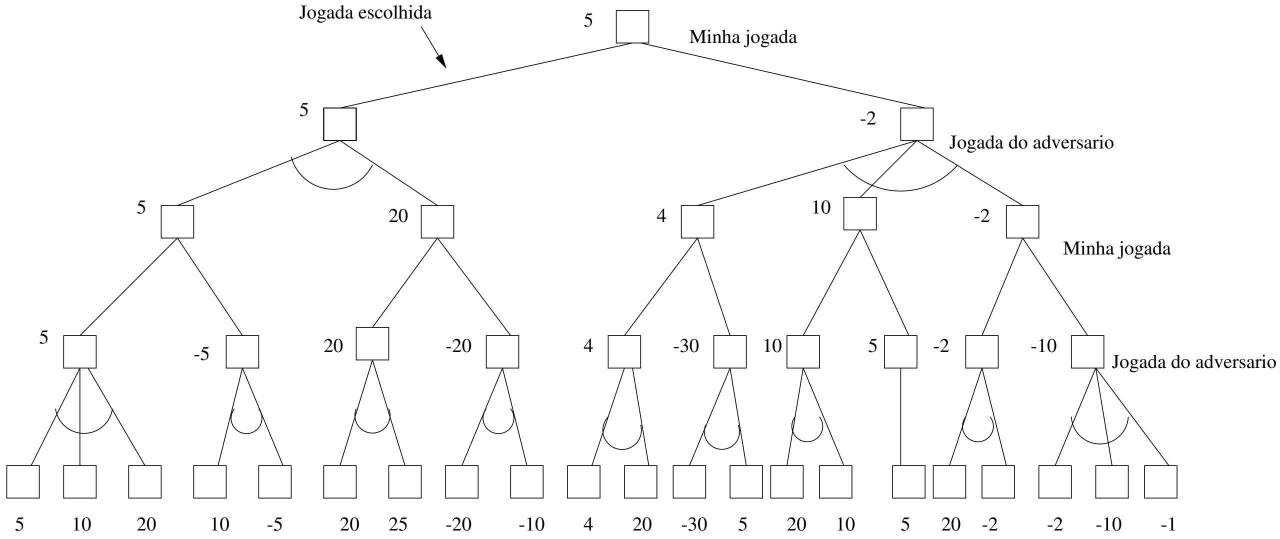
Uma sugestão é decompor o programa escrevendo vários procedimentos, alguns dos quais podem ser:

- Dado uma configuração de tabuleiro, exibe-a na tela.
- Dada uma posição de tabuleiro, verifica se a jogada com uma pedra da dada cor nesta posição é uma jogada válida.

- Dada uma posição correspondente a uma jogada válida de uma determinada cor, realiza a jogada. Isto quer dizer atualizar as peças do tabuleiro de acordo com a jogada.
- Dada uma configuração do tabuleiro, obtém o valor (nota) determinado pela função de avaliação.

Método Mini-max

Dada uma configuração do tabuleiro, vamos discutir como o programa pode decidir qual a jogada a fazer. Na figura abaixo "minha jogada" significa a jogada do computador.



(Nota: essa figura não é visível para quem está usando html, tem que usar postscript para ver a figura, sorry...)

A raiz (primeiro nível da figura) da árvore de jogo acima representa uma dada configuração do tabuleiro. As raízes de suas subárvores (segunda nível) representam todas as novas configurações que podem ser alcançadas com a minha jogada. Por sua vez, com a jogada do adversário, podem ser geradas as configurações representadas pelas nós do terceiro nível, e assim sucessivamente. O arco que une as ramificações representa jogada do adversário. É preciso entender, como é óbvio, que eu tenho controle sobre a minha jogada, mas não sobre como o adversário vai jogar. As jogadas do adversário são portanto marcadas com o arco para distinguir das minhas jogadas.

No método mini-max, dada uma configuração do tabuleiro (a raiz), geram-se as possíveis jogadas até uma certa profundidade, parando sempre com as configurações geradas pelo adversário. A geração dos nós (configurações do tabuleiro) segue a ordem por largura, ou seja nível por nível da esquerda para a direita (*breadth-first*).

Para decidir qual a melhor jogada minha a partir da raiz vou fazer o seguinte.

Para cada nó terminal (isto é, os nós do último nível) da árvore assim gerada, procuro atribuir um valor ou uma nota através de uma função de avaliação f.

f (configuração) = nota

Tanto maior a nota, mais favorável é a configuração para mim; ao passo que tanto menor é a nota melhor é para o meu adversário. Em outras palavras, eu procuro maximizar a nota, enquanto que o meu adversário procura minimizar a nota.

É muito importante saber como definir essa função de avaliação. Ela tem a ver com a estratégia do jogo. A escolha da função pode muitas vezes acabar determinando se o programa vai jogar bem ou mal.

No caso do Otelo, uma função simples é ditada pela estratégia *gulosa*, como se segue:

f (configuração) = número das minhas peças - número das peças do adversário.

Suponha que já atribuímos notas a todos os nós terminais através da função de avaliação. Vamos agora proceder de baixo para cima em direção à raiz da árvore. Repare os 3 nós terminais mais a esquera, com notas 5, 10 e 20. Para escolher uma jogada entre as 3 que geram essas notas, o adversário naturalmente iria escolher a mais esquerda, isto é, aquela jogada que dê a menor nota (5). Com isso obtenho as notas do nível acima dos nós terminais, usando sempre o mínimo das notas. Subindo mais um nível, como aí é a minha jogada, posso escolher sempre aquela que maximiza a nota. Assim sucessivamente, usando alternadamente mínimo e máximo, chego finalmente ao nível das nós gerados pela raiz. Aí é a minha vez de jogar, então escolho aquela jogada que dê a maior nota (no caso 5). Essa jogada está indicada pela seta da figura. A decisão da jogada está assim feita.

Outras estratégias para Otelo

A estratégia gulosa (onde a função de avaliação é simplesmente a diferença entre as quantidades de pedras de cada lado) não é uma boa estratégia. Ou pelo menos ela sozinha não é suficiente para obter um bom programa.

```
C . . . . . . C
. X . . . . X .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. X . . . . X .
C . . . . . . C
```

Um outra estratégia comum é de ponderar as posições. Os quatro cantos do tabuleiro (C) são as posições mais estáveis e devem receber um peso alto. As demais posições da borda do tabuleiro também são boas em termos de estabilidade. Por outro lado a posição marcada com X na figura anterior é altamente não desejável, pois pode permitir o adversário conseguir o canto.

Outras estratégias mais sofisticadas existem. Uma idéia pouco intuitiva é a chamada mobilidade. Mobilidade tem a ver com o número de possíveis jogadas de cada jogador. Tanto maior é o número de possibilidades, melhor para este jogador pois ele tem mais escolha (o que é obvio) e portanto ele não se sujeita a situação em fica forçado a fazer uma jogada numa posição desfavorável.

Sejam $C =$ número de possíveis jogadas minhas (do computador) e $A =$ número de possíveis jogadas do adversário.

A mobilidade pode ser expressa por alguma função envolvendo a diferença $C - A$.

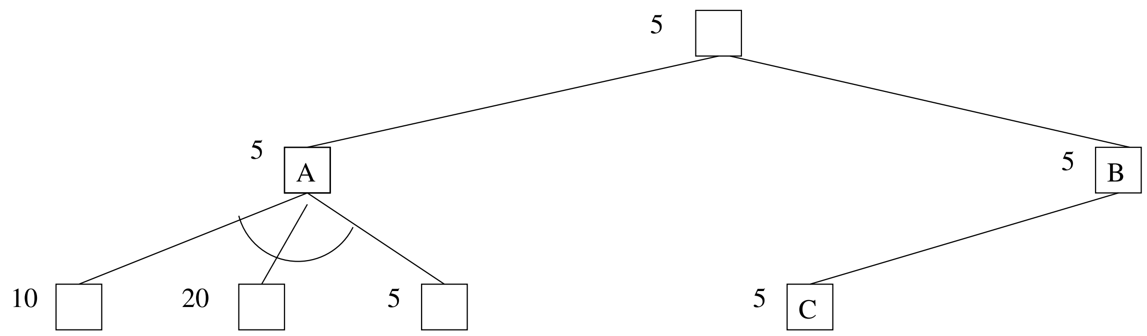
A função de avalição pode ser uma combinação das diversas estratégias.

Método Alfa-beta

No método mini-max, todos os nós terminais até uma certa profundidade são primeiro gerados. Depois todos esses nós terminais são avaliados e os valores obtidos retornados até a raiz da árvore pelo mini-max.

No método alfa-beta, vamos usar a ordem por profundidade (*depth-first*) para gerar os nós da árvore. Uma grande diferença é que, assim que um nó terminal é gerado, o seu valor é obtido através da função de avaliação. Um valor "temporário" é retornado à raiz. Desse modo é possível não ter que gerar todos os nós como no método mini-max, pois aquelas subárvore desnecessárias são podadas. Para a mesma profundidade dos nós terminais, o método alfa-beta pode levar bem menos tempo que o método mini-max, devido a essa podação. Em inglês, esse método é conhecido pelo nome de *alpha-beta pruning*. A redução no número de nós gerados permite muitas vezes aumentar a profundidade da árvore para decidir a melhor jogada.

Considere a seguinte figura, onde os nós são gerados por *depth-first*. A profundiade adotada é apenas 2, para simplificar a ilustração. Nó A já gerou todos os seus nós terminais que são avaliados retornando o valor 5 ao nó A.



O nó raiz pode receber o valor provisório VP de 5. Dependendo de valores retornados pelos outros sucessores da raiz, o valor final da raiz pode ser $\geq$ ao valor VP que é 5 mas *não pode ser menor*.

Agora suponha a geração do primeiro sucessor de B, nó C que tem valor 5. B pode receber o valor provisório 5. De novo, dependendo dos valores dos outros sucessores de B, o valor final de B pode ser $\leq$ o valor VP que é 5 mas *não pode ser maior*.

Podemos então observar que o valor final de B não pode exceder o valor VP da raiz e assim podemos descontinuar a geração de nós abaixo de B. Dizemos que podamos a subárvore abaixo de B. Isso é possível pois garantimos que B não é melhor que A.

Mais detalhes podem ser obtidos no artigo:

D. E. Knuth. An Analysis of Alpha-Beta Pruning. *Artificial Intelligence* 6, 1975, pp. 293-326.