Relatório EP2 - MAC121

Leonardo Lana Violin Oliveira

26 de Agosto de 2016

Simplificações

A única simplificação implementada no meu programa, foi o fato de que se o número de peças é maior que o numero de buracos, isto torna o tabuleiro impossível, a função *pegSolitaire* imprime "Impossivel".

Explicação do fluxo do EP

Assim que o programa acaba de ler a matriz, que será o tabuleiro, ele gera uma matriz onde os 1's viram -1's e vice-versa, a qual corresponde ao tabuleiro esperado ao final da simulação do jogo. Enquanto a função *finalMatrix* está varrendo o tabuleiro, ela conta quantos buracos e peças há no tabuleiro.

Após gerar o tabuleiro reverso, o programa chama a função *pegSolitaire* que por sua vez inicia cinco flags *eql* (equal), *bck* (backtracking), *end*, *ok*, *middle*, estas flags seram explicados conforme seu uso. Além das flags, gera a pilha *hist* (history) e um struct do tipo pos *crt* (current).

A pilha é criada com o tamanho $3 * (pieces - holes)$, pois $pieces - holes$ é o número máximo de movimentos que posso fazer até inverter o tabuleiro e isto é multiplicado por 3, pois ele empilha a struct *crt*, ou seja, a cada empilhamento, ele empilha três *short int*'s.

O laço mais externo checa se as matrizes são iguais através da flag *end*, os laços intermediário e interno varrem a matriz até acharem uma peça que possui um movimento válido. Após acharem uma peça com movimento válido, o programa faz o movimento com a flag *middle* = -1, pois a posição do meio da tripla precisa virar um buraco, empilha a posição final da peça e o movimento feito e reinicia a busca da matriz, volta para a posição (0,0).

Caso a busca consiga chegar no final da matriz, é porquê não há movimentos para serem feitos. Ele desempilha a posição que a peça parou após fazer o movimento, e o

movimento que ela fez para chegar lá. Usa a flag *middle* para desfazer o movimento, pois quando estou desfazendo, a posição do meio da tripla tem que virar uma peça.

Ao chegar no backtracking, a flag *bck* é acesa fazendo com que a busca da matriz comece da posição desempilhada com o próximo movimento em relação ao desempilhado.

Caso tente desempilhar de uma pilha vazia, então todos os movimentos possíveis foram tentados com todas as peças e a matriz não ficou igual à matriz final, portanto, não há solução.

Caso chegue ao final, com a matriz atual sendo igual a matriz final, então a função *pegSolitaire* chama a função *printPath* que imprime os movimentos, através da pilha, pois ela tem todos os movimentos que foram feitos feitos. Usando os movimentos desempilhados, o programa calcula a posição de origem da posição desempilhada, chamando a função *originPos*.

Após fazer as impressões necessárias, seja para casos possíveis ou impossíveis, libera a memória alocada e finaliza o programa.

Observações

Um fato interessante a ser mencionado é a disparidade no tempo de execução se o programa procurar buracos e movimentá-los ao invés de fazer isto com as peças. Com buracos o programa pode ser executado por diversas horas e não encontrará solução para o resta um padrão, porém fazendo os movimentos com peças, o programa encontra a solução em segundos.

Dependendo do caso testado, a prioridade dos movimentos influência muito no tempo de execução.

Há diversas simplificações para o jogo, como a função Pagoda que determina se o tabuleiro é possível ou não rapidamente. E mover as peças em pacotes, o que agiliza os movimentos e evita buscas desnecessárias. Nenhuma dessas simplificações foram implementadas no meu EP.

Casos testados

Teste 1 –> tempo: **4,19**s

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	-1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

Teste 2 –> tempo: **3,76**s

0	0	1	1	1	0	0
0	0	1	1	1	0	0
1	1	1	1	1	1	1
1	1	1	-1	1	1	1
1	1	1	1	1	1	1
0	0	1	1	1	0	0
0	0	1	1	1	0	0

Teste 3 –> tempo: **1,04**s

1	1	1	1
1	1	1	1
1	1	1	1
1	1	-1	1
1	1	1	1
1	1	1	1