Report for Programming Problem 1 - 2048

Team: 2018297934_2018298731

Student ID: 2018297934 Name Luis Carlos Lopes Loureiro

Student ID: 2018298731 Name João Pedro Pacheco Silva

Descrição do Algoritmo:

O algoritmo usado para solucionar o problema é um algoritmo recursivo que adota a estratégia de backtracking em que se para a recursão de soluções parciais cuja extensão levaria a uma solução invalida ou pior. Em cada passo recursivo o algoritmo altera o estado do tabuleiro de 4 formas diferentes, aplicando cada uma das diferentes jogadas.

Para aplicar cada jogada, são mantidos dois ponteiros, um para a posição onde vamos tentar colocar a peça e outro para a peça que vamos tentar mover. O segundo ponteiro, começa na segunda posição e vai se deslocando até encontrar uma peça. Sempre que encontrar uma, vemos se a posição para a qual o queremos mover está ocupada. Se não estiver move-se a peça para essa posição e se estiver vemos se é possível fazer merge. Caso não seja possível fazer merge, movemos o primeiro ponteiro para a próxima posição. Esta processo é aplicado a cada uma das linhas/colunas do tabuleiro.

Para cada um dos novos tabuleiros é chamada a função recursiva e o processo repete-se até que o tabuleiro tenha apenas uma peça. A solução final é guardada numa variável local, que vai sendo atualizada à medida que se encontram soluções melhores.

Condições de corte:

- Parar a recursão se o número de jogadas ultrapassar o limite estabelecido.

A partir do momento em que o limite de jogadas especificado no input é ultrapassado a recursão poderá ser cortada pois já se sabe que todos os ramos dessa subárvore recursiva irão levar a uma resposta invalida.

Melhorias:

1- Caso o algoritmo já tenha encontrado uma solução, ramos da árvore de recursão em que o número de jogadas ultrapasse a solução previa também podem ser cortados, pois passa a ser impossível encontrar uma solução melhor nos passos recursivos seguintes. Portanto o limite de jogadas pode ser atualizado à medida que se vão encontrando soluções melhores.

2- É possível estabelecer um limite inferior para poder cortar a recuperação mais cedo, isto é, um número abaixo do qual se torna impossível esvaziar o tabuleiro. Segundo o enunciado, uma peça que resulte de um merge não pode fazer merge com outras peças na mesma jogada, portanto no melhor caso uma jogada pode reduzir o número de peças para metade (quanto todas as peças dão origem a

um merge) ou metade mais 1 no caso de haver um número ímpar de peças. Assim para 2 peças o número mínimo de jogadas possível será 1 (i.e. Reduzir para metade 1 vez), para 3 e 4 será 2 (i.e. Reduzir para metade 2 vezes), para 5, 6, 7 e 8 será 3 e genericamente para n peças será log2(n) (com o resultado aproximado ao inteiro imediatamente acima). Concluindo, se o menor número possível de jogadas para esvaziar o tabuleiro não for suficiente mente pequeno pra impedir que se ultrapasse o limite de jogadas, não há necessidade de estender a recursão. Condição de corte: if moves + limite inferior ≥ limite superior

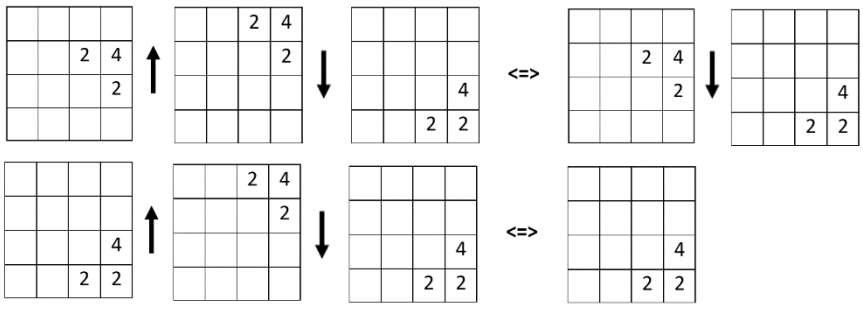
- Parar a recursão caso a jogada não altere o tabuleiro.

Se apos aplicar uma jogada, o tabuleiro permanecer igual, então a jogada foi inútil. Ao fazer uma jogada inútil deve cortar-se a recursão, pois ao chamar a função recursiva com os mesmos parâmetros (ou seja, com o mesmo tabuleiro) iram criar-se repetições de ramos da árvore de recursão, ou seja, irá repetir-se trabalho computacional desnecessariamente. Para além disso, o próximo passo recursivo também conterá essa mesma jogada inútil levando à criação de um ciclo que só irá terminar quando chegar à condição de corte anterior.

Para determinar se uma jogada é inútil, na função que aplica a jogada, verifica-se se houve peças que se movimentaram ou se houve merges.

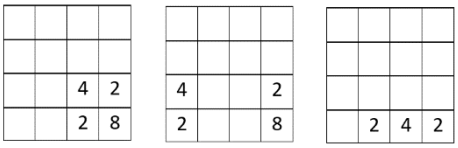
- Parar a recursão caso uma sequência de jogadas seja inútil:

Se após aplicar uma jogada não tiver ocorrido nenhum merge, no próximo passo recursivo deve cortar-se a recursão para a mesma jogada e para a jogada na direção inversa. Não se deve repetir a mesma jogada nessas condições pois já se sabe que será uma jogada inútil e não se deve continuar a recursão para a jogada na direção inversa, pois tornaria a jogada anterior numa jogada inútil. A jogada anterior tornar-se-ia inútil pois teríamos o mesmo resultado caso não a tivéssemos aplicado.



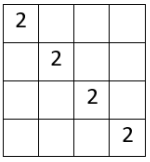
- Parar a recursão quando o tabuleiro se encontra num estado impossível de resolver:

Casos onde a soma dos valores de todas as peças seja diferente de um expoente de 2, são impossíveis e devem ser descartados logo no início. Outros casos onde é impossível esvaziar o tabuleiro devem-se à disposição das peças. Esses casos acontecem apenas quando todas as colunas com peças têm o mesmo número de peças e todas as linhas com peças também têm o mesmo número de peças.



Quando o tabuleiro reúne essas condições, podemos assumir que qualquer jogada sem merges é uma jogada inútil, pois mesmo que as peças se movam a sua disposição relativa umas às outras vai permanecer a mesma.

Esta condição só é aplicável a partir da segunda jogada, onde se tem a certeza de que os blocos estão encostados a uma das paredes.



-> A condição não é aplicável

Todos estas condições de corte são validas, pois, a solução ótima nunca irá conter uma jogada ou sequência de jogadas inútil. Para cada solução com jogadas/ sequências de jogadas inúteis, existe também uma solução equivalente que não aplica essas jogadas inúteis e que, portanto, tem menos movimentos. Se solução ótima tivesse jogadas/sequências de jogadas inúteis teríamos uma contradição, pois existiria uma solução equivalente com menos jogadas.

Estruturas usadas:

Foi usada uma matriz de inteiros para representar o tabuleiro.

Complexidade:

A complexidade espacial em cada passo recursivo e no caso base é de C + n, sendo n o número de casas do tabuleiro e C um número constante que representa o espaço ocupado pelas variáveis locais e parâmetros da função. Considerando o pior caso, a complexidade espacial do algoritmo é igual à complexidade espacial do passo recursivo multiplicada pela profundidade N da arvore de recursão, ou seja, $N*(C + n) \in O(n)$

A complexidade temporal de cada passo recursivo é $Tq(n) = 4*n + c \in O(n)$, sendo n o número de casas do tabuleiro e a complexidade temporal do caso base é $T(0) = c \in O(1)$. Considerando o pior caso, a complexidade temporal do algoritmo é:

$T(N) = 4 * T(N-1) + Tq(n) = 16*T(N-2) + 5*Tq(n) = 4^N * T(N-N) + (4^N - 1) * Tq(n)$

$= 4^N * T(0) + (4^N - 1) * Tq(n) = 4^N * T(0) + (4^{N+1} - 4) * (n+c) \in O(4^{N+1})$, sendo N a profundidade da árvore de recursão.

Referencias:

- Slides das aulas teoricas (“Week01 - T - Introduction to EA”, “Week02 - T - Recursion” e “Week03 - T - Backtracking”).