Relatório de EA

2048 clean up!

2020-2021

André Cristóvão Ferreira da Silva 2018277921
João Alexandre Santos Cruz 2018288464

Algorithm description

Para resolver o problema usou-se a seguinte abordagem:

```
function Solver2048(board, max, movesDone, movesWitouthSum) :

if movesDone = max or movesDone = minimum or movesWithoutSum=5 then
  // rejection case
  return;
if lLastMove ≠ 'R' or lastMove ≠ 'L' or sumlLast then
  boardR, sum, done =moveRight(board); if done then // base case
    if movesDone < minimum then
      minimum = movesDone;
    return;
  if changes then // recursive step
    Solver2048(boardR,max,movesDone + 1, movesWithoutSum + sum);
else if lLastMove ≠ 'U' or lastMove ≠ 'D' or sumlLast then
  boardR, sum, done =moveUp(board); if done then // base case
    if movesDone < minimum then
      minimum = movesDone;
    return;
  if changes then // recursive step
    Solver2048(boardR,max,movesDone + 1, movesWithoutSum + sum);
else if lLastMove ≠ 'L' or lastMove ≠ 'R' or sumlLast then
  boardR, sum, done = moveLeft(board); if done then // base case
    if movesDone < minimum then
      minimum = movesDone;
    return;
  if changes then // recursive step
    Solver2048(boardR,max,movesDone + 1, movesWithoutSum + sum);
else if lLastMove ≠ 'D' or lastMove ≠ 'U' or sumlLast then
  boardR, sum, done = moveDown(board); if done then // base case
    if movesDone < minimum then
      minimum = movesDone;
    return;
  if changes then // recursive step
    Solver2048(boardR,max,movesDone + 1, movesWithoutSum + sum);
```

Para aumentar a velocidade do algoritmo foram utilizadas as seguintes medidas:

- Durante a leitura do *input* é efetuado um *check* para verificar se o *board* é possível ou impossível;
- Verificar se existiu mudanças durante o movimento e, caso não haja, cortar;

- Guardar os 2 últimos movimentos para saber se existiu alguma soma durante esses;
- Se durante 5 movimentos seguidos não houver soma, cortar;
- Mudar a ordem dos movimentos para *Right Up Left Down* de maneira a que os movimentos laterais e horizontais se alternem entre si.

Data structures

Para as *data structures* foram usados vetores e vetores de vetores (matriz).

Correctness

O primeiro passo que o algoritmo faz é verificar se o *board* dado é possível ou não, isto é feito com base no uso do logaritmo de base 2, pois foi verificado que um *board* só é possível se a soma de todos os números for logaritmo de base 2.

Durante os movimentos é verificado se existiu algum movimento ou alguma soma. Caso não tenha existido nenhum movimento então é feito um corte, uma vez que não faz sentido continuar por um lado que não gera mudanças.

São guardados o último e penúltimo movimento, assim é possível não executar um movimento que não gera qualquer efeito no *board*. Por exemplo, se tivermos feito Direita-Esquerda e não tiver existido nenhuma soma então não faz sentido efetuar outra vez Direita uma vez que o board vai voltar à posição inicial.

Foi adicionado também um número máximo de movimentos que se pode executar sem qualquer soma efetuada, este número é 5 uma vez que com os testes feitos verificou-se que sejam quais forem os movimentos e seja qual for a ordem se após 5 movimentos não tiver existido nenhuma soma então é porque o *board* é impossível.

Desta maneira é possível confirmar que o algoritmo vai passar por todos os casos cortando o mais cedo possível para se obter os melhores tempos.

Algorithm Analysis

Na complexidade espacial teremos de ter em conta todas as chamadas da função recursiva. A cada nível n teremos 4^n chamadas de função e em cada chamada o maior gasto de memória vem do armazenamento do movimento para realização do *backtracking* que neste caso terá complexidade $O(n^2)$. As outras variáveis que são precisas são ou constantes ou *arrays* e portanto não terão impacto na complexidade final. Desta forma teremos com k constantes e j *arrays* temos:

$S(n)=4^n * n^2 + 4^n * n * k + 4^n * 1 * j \in O(4^n * n^2)$

Quanto à complexidade temporal, se k é o número máximo de movimentos então o tempo no pior caso seria:

$T(k) = 4T(k-1) + Tq$

$= 4(4T(k-2)+ Tq) + Tq = 16T(k-2) + 5Tq$

$= 16(4T(k-3) + Tq) + 5Tq = 64T(k-3) + 21Tq$

$= 64(4T(k-4) + Tq) + 21Tq = 256T(k-3) + 85Tq$

...

$= 4^i * T(k-i) + (4^i-1)/3 * Tq$

$= 4^kT(0) + (4^k - 1)/3Tq \in O(4^n)$

References

https://youtu.be/JSn-DJU8qf0