Relatório 1º projecto ASA 2022/2023

Grupo: AL042/TPyyy

Aluno(s): Mateus Spencer (100032)

Descrição do Problema e da Solução

Para Resolver o Problema proposto no enunciado eu implementei uma solução recursiva que utiliza *memoization*. O tabuleiro é representado como no input (um veto de n entradas em que para cada índice de uma linha indica o número de colunas seguidas desde a esquerda que tem uma peça). A função recebe um vetor que representa o tabuleiro bem como outros parâmetros que ajudam a diminuir os cálculos por chamada e a guardar as soluções. Em cada chamada a função ve qual é a peça na coluna mais à direita que está mais acima, e consoante o número de peças que têm seguidas abaixo dela (k) chama a função k vezes recursivamente, somando o seu resultado ao total de soluções para a configuração atual sem ter retirado nada, em que lhe passa um tabuleiro com a peça k retirada (desde que esta peça não exceda o número de linhas, começando no tal bloco mais à direita e acima (1x1, 2x2, 3x3…). Quando chegar ao caso base (neste caso um tabuleiro só com uma coluna, retorna 1 pois dada a sequência de peças colocadas chegámos a um tabuleiro com solução 1.No final destas chamadas guardamos o total retornado pelas chamadas recursivas e guardamos a solução num hashmap.

Análise Teórica

A complexidade do problema está dominada pela chamada função recursiva que é exponencial dada a natureza do problema. No entanto com a utilização de memoization os problemas até certo ponto são resolvidos com uma complexidade muito mais reduzida.

int solve(recebe o tabuleiro, outros parâmetros){

if(se o mapa só tiver solução com blocos 1x1)){ return 1;} - O(1)

calcular e guardar o hash deste mapa para aceder à possível solução no hashmapO(n)

if (se já tem solução para este tabuleiro){return solução do hashmap;)- O(1)

da coluna mais à direita qual é a peça mais acima - O(n)

fora(até que tamanho de peça posso colocar abaixo, quantas peças tem abaixo seguidas){ max_piece ++;} - O(n)

for(desde a peça 1 até à max_piece que pode ser retirada desta coluna){

fazer uma cópia do mapa para ser passada recursivamente - O(n)

Retirar peça do mapa - O(n)

if(se a peça não exceder o tamanho das linhas){ - O(1)

Chamada Recursiva: total += solve(MAPA SEM A PEÇA RETIRADA, outros parâmetros já calculados); - O(n^m)

}else{terminar função porque as próximas peças também excederiam o tabuleiro O(1)} }

//guardar número acumulado de soluções para este tabuleiro na hash table - O(1)

Relatório 1º projecto ASA 2022/2023

Grupo: AL042/TPyyy

Aluno(s): Mateus Spencer (100032)

---

return total; - O(1)}

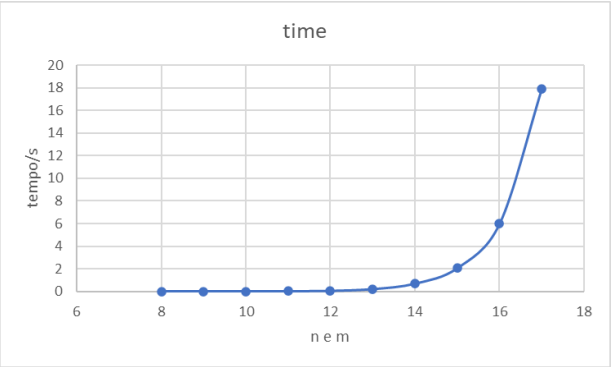
- Leitura dos dados de entrada: O(n)
- Algumas operações dentro da função recursiva são O(1)
- Fazer cópia do mapa e calcular hash para o mapa que dominam a chamada recursiva - O(n)
- Apresentação dos dados. O(1)

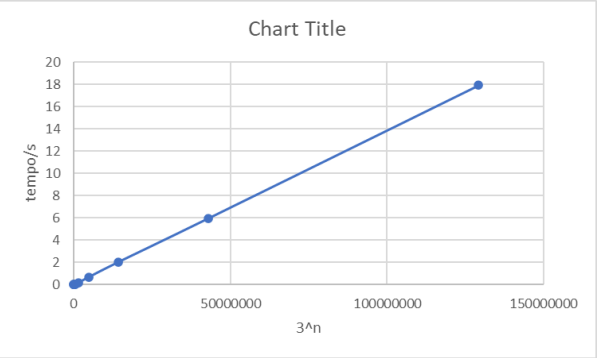
Complexidade global da solução sem memoization: O(n^m)

Complexidade global da solução com memoization: O(k^n)

Avaliação Experimental dos Resultados

Código corrido para diferentes tabuleiros quadrados sem escada. Corrido com o comando unix time antes da instrução. 8x8,10x10…17x17





O gráfico da esquerda demonstra que com o aumento linear do tamanho do tabuleiro o tempo de solução aumenta exponencialmente.

Mas se em função do tempo tivermos um aumento exponencial do tamanho do mapa como a complexidade com memoization prevê, vemos uma correlação linear, o que indica que esta complexidade está aproximadamente certa