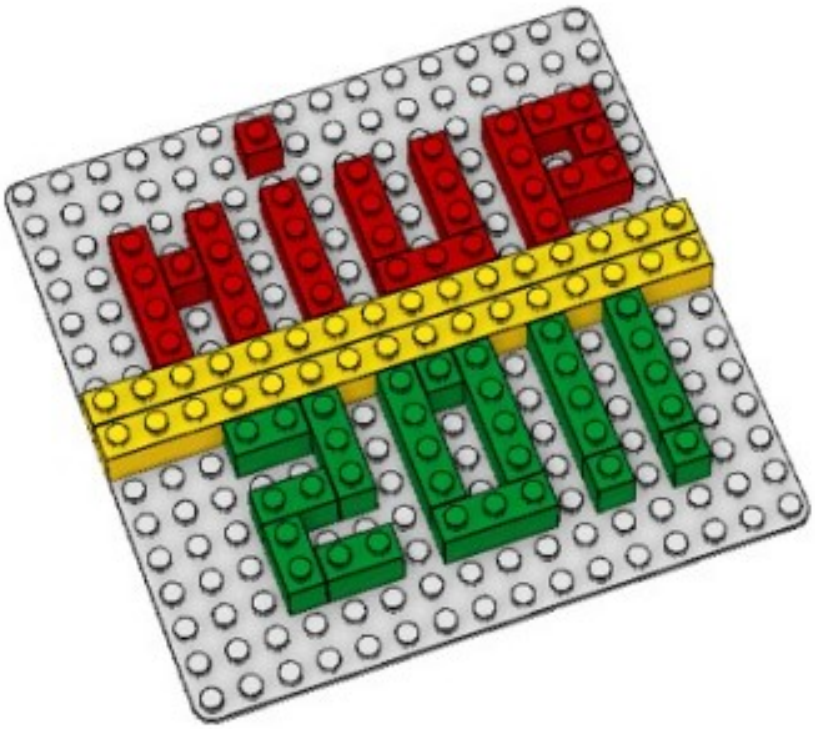
UNIVERSIDADE DE ÉVORA

Problem: Mosaics



Março 2022

Rafael Silva 45813

Vicente Romão 45467

(Grupo 106)

1 Introdução

Na presença de várias peças de lego de tamanhos diferentes (1×1, 1×2, 1×3, 1×4, 1×6, 1×8, 1×10, 1×12 e 1×16) e uma base, é possível criar construções de várias maneiras possíveis. O objetivo deste trabalho é através dessas construções, calcular o número de maneiras diferentes de forma a conseguir construir esse mosaico.

A resolução deste problema, baseia-se na **programação dinâmica** com o objetivo de chegar a uma solução ótima do problema.

2 Desenvolvimento do Algoritmo

2.1 *Input*

O nosso algoritmo começa por ler a primeira linha do input, com o auxilio da classe **BufferedReader** e **System.in.**

A primeira linha do input contem dois inteiros separados por uma "espaço", através do método **split** são guardados num *array* de *Strings* e a partir daí define-se que a primeira variável corresponde ao número de linhas (**lines**) e a segunda variável corresponde ao número de colunas (**columns**) existente na base onde irá ser criada a construção de peças lego. É importante referir que o máximo de linhas e colunas aceitável pelo algoritmo é 1000.

A partir da segunda linha até ao fim do input, o utilizador coloca o tabuleiro com a construção em que se pretende saber o número de combinações possíveis com as peças de lego que possui.

2.2 Cálculo do número de possibilidades de cada peça

De forma a otimizar o processo do algoritmo foi criado um método com base no exercício "Troco de moedas" realizado numa das aulas práticas com o objetivo de calcular o número de possibilidades de cada peça. Este método é denominado **getArrayPieces**.

O método recebe como argumento o número de colunas (**bPiece**). Depois define-se no array **Pieces** todos os tamanhos de peças existentes no algoritmo. O array long no final vai guardar o número de possibilidades de cada peça e possui o tamanho bPiece+1. O caso base será na posição 0 do array e tem o valor de 1. Por exemplo para calcular a possibilidade num espaço de tamanho 4, como existe uma peça única de tamanho 4 no array Pieces, sabemos a partida que o espaço fica preenchido com pelo menos 1 peça de lego. A partir daqui, o método faz um

loop para preencher todas as posições do array até bPiece. Cada posição deste array corresponde ao número de possibilidades do índice do array.

2.3 Cálculo do número de maneiras diferentes de forma a conseguir construir um mosaico

Depois de calcular o número de possibilidades para cada peça, começamos por criar um loop que itera todas as linhas do mosaico de forma a verificar se é um espaço vazio ou uma peça. Caso encontre uma peça é verificado o tamanho da mesma e com o array retornado pelo método **getArrayPieces**, consegue-se obter de forma eficiente o número de possibilidades. Tendo esse número de possibilidades, para chegar a um valor total do mosaico, o algoritmo multiplica as possibilidades de cada peça, apresentando no output o mesmo.

2.3.1 Recursiva

Para implementar o método **getArrayPieces**, tivemos como base a seguinte função recursiva:

$$m(q)=\begin{cases} 1 \text{ se } q=0 \\ \forall\, X\in\{1,2,3,4,6,8,10,12,16\},\ m(q-X[i]) \text{ se } q-X[i]>0 \end{cases}$$

3 Análise de Complexidades

	Temporal	Espacial
getArrayPieces	O(n*)	O(n)

*n é o número de colunas

No pior caso, este método poderá ter uma complexidade de O(1000).

4 Conclusão

Em conclusão, neste trabalho tivemos como objetivo determinar a forma como o método **getArrayPieces** deveria encontrar o número de combinações possíveis para cada tamanho, começamos por desenvolver a lógica em papel seguindo em exemplo o exercício "Troco em Moedas" e também desenvolvemos este pensamento em pseudocódigo de modo a facilitar a implementação do método. Tendo este método a funcionar corretamente, apenas restava saber quantas peças existiam no input e o tamanho das mesmas para saber o número total de possibilidades de construir o mosaico.

Foram realizadas todas as tarefas propostas pelo enunciado do trabalho e não encontramos nenhum problema ao implementar este algoritmo.