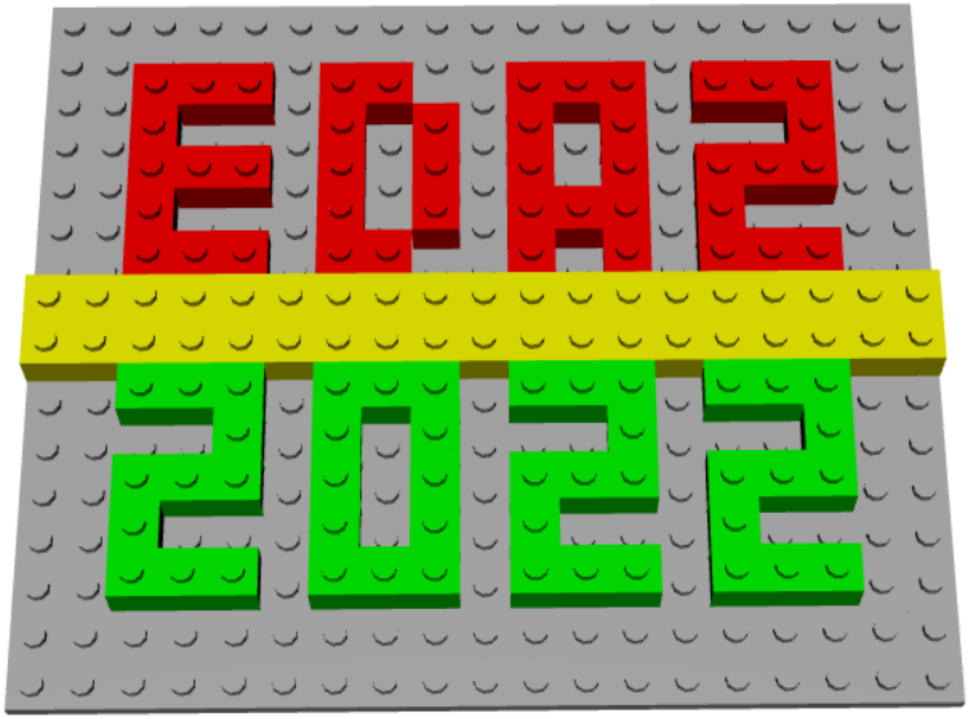

U.C. Estrutura de Dados e Algoritmos II

1º Trabalho-Mosaics

Docente: Vasco Pedro
Grupo: g121
Discentes: André Baião 48092, Gonçalo Barradas 48402

Março 2022

Índice

1 Objetivo

Com o intuito de avaliar os conhecimentos adquiridos pelos alunos foi proposto um trabalho que consiste na resolução do problema "Mosaics". O trabalho consiste na elaboração de um programa, a representação da sua função recursiva e no cálculo das suas complexidades (temporal e espacial).

A resolução deste problema baseia-se na analise das sequências existentes num dado mosaico e no calculo do maior número de combinações possíveis de realizar com as peças permitidas.

2 Descrição do algoritmo

Para resolver este problema, o algoritmo deve conter 2 parte: Na primeira parte recorre-se à análise linha a linha de forma a identificar as sequências de cores, podendo ocorrer um dos seguintes casos:

- No primeiro caso é avaliado o primeiro valor e se este não for ".", então existe uma cor, começando uma sequência.
- No segundo caso se houver uma sequência seguida de um ".", então é calculado o valor das combinações possíveis e reiniciada a sequência.
- No terceiro caso é avaliado o último valor do mosaico, havendo dois pontos de avaliação:
  - Se o valor actual for uma cor e se pertencer à sequência, então calcula-se o valor das combinações possíveis do tamanho da sequência mais um que corresponde à última peça.
  - No caso de ser outro valor é calculado o valor das combinações dessa sequência e reiniciado o valor da mesma.
- No quarto caso, o objectivo é avaliar qualquer posição do mosaico excluindo a primeira e a última posição. No caso de o valor ser igual ao penúltimo valor avaliado adiciona-se um ao tamanho da sequência, em qualquer outro caso é calculado o valor das combinações e reiniciado o valor da sequência.

A segunda parte tem como objectivo calcular as maneiras possíveis de realizar uma sequência utilizando as peças de lego permitidas.

Foi criado o método "possibleCombinations" que, para cada posição da sequência avalia quais as peças que podem ser utilizadas e calcula o número de combinações possíveis a realizar com essas peças. Este ciclo é executado n vezes conforme o número de sequência de cores que existam.

3 Função Recursiva

Dada uma sequência não vazia crescente de inteiros positivos $\mathrm{B} = (\mathrm{B}_1\, B_2 \ldots B_N)$, a função que calcula o número de maneiras diferentes que se pode fazer um determinado tamanho positivo i é:

$$\mathrm{C}_B(i) = \begin{cases} 1 & \text{se } i = 0 \\ \displaystyle\sum_{N=1 \wedge \forall N : B_N \leq i}^{N} C_B(i - B_N) & \text{se } i > 0 \end{cases}$$

A partir desta função, é possível obter um algoritmo iterativo :

```
1  possibleCombinations(i)
2      let C[0 ... i]   //novo array
3      N = |T|              //Numero de peças
4      m[0] = 1             //caso Base
5      for x = 1 to i do
6          j = 1
7          while j <= N and B[j] <= x do
8              C[x] = C[x] + C[x - B[j]]      //Caso recursivo
```

```
9                 j = j + 1
10        return C[i]
```

4 Calculo da Complexidade

4.1 Temporal

Após a leitura dos dados necessários para a resolução do problema, vamos analisar o mosaico, que se encontra guardado numa matriz, começamos por inicializar 4 variáveis que nos vão auxiliar na resolução e análise do mosaico, cada um tem um custo de `O(1)` logo `4 x O(1)`, posteriormente começamos a analisar o mosaico.

Para a análise do mosaico foram utilizados 2 ciclos, um interno e outro externo. No ciclo interno existem 5 condições de custo constante `O(1)`, mas o ciclo tem um custo `O(n)` onde `n` é o numero de colunas do mosaico. O ciclo externo tem 1 condição que tem um custo constante `O(1)`, um custo de `O(m)` onde `m` é o numero de linhas do mosaico e dentro deste ciclo ocorre também o ciclo interno.

Logo a análise do mosaico tem um custo de `4 x O(4) + O(n x m) = O(n²)` onde n^2 é o numero de posições do mosaico.

A função "possibleCombinations"começa por inicializar uma matriz o que tem um custo constante `O(1)`, depois de ser definido o caso base vamos é executado um ciclo que , tem um custo linear de `O(n)` onde `n` é o tamanho da sequência, neste ciclo existe um outro ciclo no qual existe uma condição que tem um custo de `O(1)` , mas o ciclo tem um custo de `O(m)` onde `m` é o numero de peças diferentes.

Logo esta função tem um custo de :

$$O(1) + O(n) \times O(m) \times O(1) = O(n^2)$$

Como na analise de dados a função "possibleCombinations"é chamada dentro dos dois ciclos a complexidade temporal do programa é:

$$O(n^2) \times O(n^2) = O(n^4)$$

4.2 Espacial

Para calcular a complexidade espacial, foi necessário ter em conta a estrutura de dados utilizada para guardar os dados necessários para a realização do problema. Para guardar o mosaico foi utilizada uma matriz bidimensional, para inicializar a matriz foi necessário saber o número de linha e colunas, esses dois valores ocupam em memoria `O(1)` cada.

Para inicializar a matriz foi necessário saber o número de linha e de colunas do mosaico para determinar o espaço que a matriz iria ocupar em memoria. Seja n o numero de linhas do mosaico e m o numero de colunas do mosaico, a complexidade espacial desta matriz é $O(n \times m) = O(n^2)$

Podemos concluir que a complexidade espacial do programa é:

$$O(1) + O(1) + O(n^2) = O(n^2)$$

5 Comentários

Durante a análise de dados, no caso de guardarmos todas as sequências do mosaico e só no fim calcularmos o numero de combinações possíveis melhoraríamos a complexidade temporal ($O(n^3)$) mas estaríamos a piorar a complexidade espacial. ($O(n^2 + n)$)

6 Conclusão

Considerando a complexidade temporal e a complexidade espacial obtida podemos concluir que o desafio proposto foi concluído com sucesso. Tendo sido possível aplicar e consolidar os conhecimentos adquiridos ao longo da elaboração do trabalho.