Report for Programming Problem 2 - ARChitecture

Team:

Student ID:2018288464 Name: João Alexandre Santos Cruz

Student ID:2018277921 Name: André Cristóvão Ferreira da Silva

1. Algorithm description

```
function archBuilder(width,blocksize,height)
for i=1 to width do
    temp1 = possibleMoves[height - blockSize][1]
    for i=height-blockSize-1 to 1 do // Descida
        somaDown[j + 1] = temp1
        sDown+=possibleMoves[j+blockSize][0]+somaDown[j]
        if j < (height-blockSize+1)-blockSize then // Retirar
            sDown-=possibleMoves[j+blockSize][0]+somaDown[j+blockSize]
        temp1 = possibleMoves[j][1]
        possibleMoves[j][1] = sDown
        if j < blockSize then // Somar ao Final
            nPosAvailable += temp1
    end for

    for i=1 to height-blockSize-1 do // Subida
        somaUp[j - 1] = temp
        if j - ((width - (i + 1)) * (blockSize - 1)) - blockSize
        ≤ 0 then// Ainda é possivel colocar mais blocos
            sUp += somaUp[j - 1]
            if j > blockSize - 1 then // Retirar
                sUp -= somaUp[j - blockSize]
            temp = possibleMoves[j][0]; possibleMoves[j][0] = sUp;
            if j < blockSize then // Somar ao final
                nPosAvailable += temp
        else
            break;
    end for

end for
```

O algoritmo é baseado numa matriz de altura máxima na qual os blocos podem ser colocados e a largura máxima. Para a calcular as subidas/descidas da coluna seguinte vai ser usado a altura/descida da coluna atual, deste modo ao resolver uma coluna estamos a resolver um *sub-problem* que irá ajudar a resolver o *sub-problem* seguinte.

Os *speed-up tricks* usados foram os seguintes:

- Fazer só numa coluna, ou seja, a coluna atual vai substituir os valores da seguinte na própria, fazendo com que não seja necessário cópias de vetores;
- Usar as propriedades do *mod* para diminuir o número de vezes que as funções são chamadas;
- Diminuir o *array* para ter só a altura até à qual é possível pôr o último bloco em altura, em vez de ter a altura total;
- A subida é parada se já for possível fazer a descida a tempo.

2. Data structures

Foram usados 3 vetores, um deles, vetor de vetor, em que as colunas representam a altura a que se vai colocar o bloco e o vetor interior com 2 colunas uma para a subida e outra para a descida. Os outros 2 foram usados como vetores temporários onde se vai guardar o valor da subida/descida da coluna anterior.

3. Correctness

Para este problema foi usada uma estrutura em matriz altura por largura onde vai ser guardado o valor da subida e descida para cada célula. Inicialmente a célula [0][0] é inicializada a 1,0 (sendo 1 a subida e 0 na descida), pois na primeira posição o bloco será sempre posto na altura 0 da coluna 0. Assim para cada coluna seguinte até à altura do bloco-1, uma vez que o bloco seguinte tem de ter pelo menos uma parte em comum com o bloco anterior, será somado o valor da subida de [0][0], isto é repetido até chegar a coluna final.

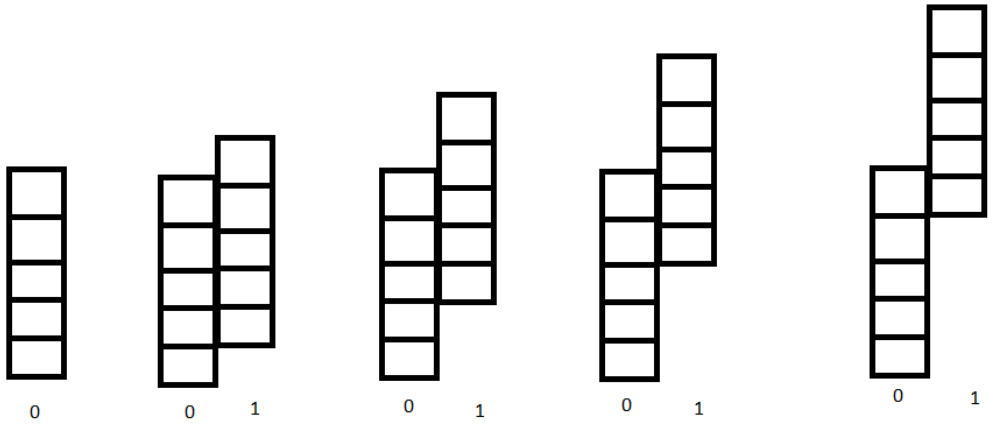
Para descida é realizado um processo semelhante, mas para este processo vai ser usado tanto o valor da subida como o valor da descida das células anteriores.

O resultado final é a soma de todas as peças que se encontram na posição 0 em qualquer coluna da sala.

Ex.: n=5, h=5, H=10;

5	**0,0**	**0,0**	**4,0**	**6,0**	**3,0**
4	0,0	1,0	3,0	3,4	1,6
3	0,0	1,0	2,1	1,7	0,10
2	0,0	1.0	1,2	0,10	0,17
1	0,0	1,0	0,3	0,13	0,27
0	1,0	0,0	0,4	0,12	0,38
	0	**1**	**2**	**3**	**4**

Exemplo da posição das alturas na coluna 1:



Dada a altura e coluna de um bloco sendo A(n,h) o número de arcos possíveis que se vai obter nessa posição, se for removida a coluna k, 0<k<=n, obtemos A(m,h)<A(n,h), pois se m<n então ao adicionar a coluna k, A(k,h)>A(n,h), o que é uma contradição.

4. Algorithm Analysis

Sendo n o número de colunas e m a altura máxima em que se pode colocar um objeto então no *worst case* a complexidade temporal é:
T(n,m) = n*2m ∈ O(n*m).

Já para o espaço, usando as mesmas variáveis a complexidade espacial é: T(n,m) = 2*m+m*2 = 4*m ∈ O(m).