Report for Programming Problem 2 - ARChitecture

Team: Student ID: 2018280762 Name: Mariana Loreto
Student ID: 2018288500 Name: Mariana Lança

1. Algorithm description

O algoritmo começa por verificar a altura máxima possível consoante o número de blocos e suas alturas, o que permite que, desde o início, se possam eliminar casos que se sabe serem impossíveis para o problema. Além disto, testa-se se o número de blocos é menor do que o mínimo permitido (3) ou se a altura destes é maior do que a altura máxima. Se tal for verdadeiro, imprime-se 0 sem que seja feito mais nenhum cálculo.

No cálculo do número de arcos, é chamada a função *arc*, cuja obrigação é percorrer o número de blocos e chamar a função de cálculo do resultado. Esta última, *calc*, calcula o número de possibilidades de arcos num dado **i** bloco.

```
calc (&prev, &curr, i)
     j ← 1
     while j<= H-h and (j-h<i or (j-h>=i and prev[j-h, 0] ≠ 0))
          do curr[j] = [0, 0]

          if j = 1 then
               k ← 1
               while j+k<=H-h and (prev[j+k,1]≠0 or
prev[j+k,2]≠0)
                    do curr[j,2] = mod_add(curr[j,2],
prev[j+k,1],                   module)
                    curr[j,2] = mod_add(curr[j,2], prev[j+k,2],
                    module)
                    k ← k + 1

          else
               curr[j, 2] = curr[j-1, 2]
               curr[j, 2] = mod_sub(curr[j, 2], prev[j, 2],
module)
               curr[j, 2] = mod_sub(curr[j, 2], prev[j, 0],
module)

               if j+h-1 <= H-h then
                    curr[j, 2] = mod_add(curr[j, 2],
                    previous[j+h-1, 1], module)
                    curr[j, 2] = mod_add(curr[j, 2],
                    previous[j+h-1, 2], module)

               if j >= i then
                    if curr[j-1, 0] = 0 then
                         k ← 1
                         while k<h and j-k>=0 do
                              curr[j, 1]= mod_add(curr[j, 1],
                              previous[j-k, 1], module)
```

```
27                              k ← k + 1
28                   else
29                        curr[j,1] = mod_abs(curr[j-1, 1],
30                        module)
31                        curr[j,1]=
mod_add(curr[j,1],prev[j-1,1], 32                         module)
33                        if j-h >= 0 then
34                             curr[j, 1]= mod_sub(curr[j, 1],
35                             previous[j-h, 1], module)
36         j ← j + 1
37    return curr[1, 2]
```

A esta função são passados dois vetores, um atual e o anterior, e o valor da coluna atual.

Estes vetores são utilizados para calcular as subidas e as descidas, sendo que se verifica o j (altura num dado instante) e, dependendo do seu valor, calculam-se os resultados de formas diferentes.

No final destes cálculos, soma-se à variável *total*, na função arc, o valor das descidas no bloco na altura j=0.

2. Data structures

Utilizamos apenas dois vetores, *vec1* e *vec2*, que representam a linha atual e a anterior. Cada um dos elementos destas estruturas é constituído por um par de valores, em que o primeiro índice refere-se à contagem de subidas e o segundo, às descidas.

3. Correctness

De forma a minimizar o número de cálculos realizados, começa-se por, se possível, limitar o H pelo valor da altura máxima que pode ser atingida, se este for menor do que o atual.

Já na função *arc*, é feita a alternação dos vetores vec1 e vec2, de modo a reduzir a quantidade de cópias feitas. Assim, numa chamada *calc*, o vetor “atual” passará, na chamada seguinte, a ser o “anterior”. Ambos os vetores estão preenchidos, pelo que é preciso remover os valores do vetor “atual”, *curr*. Para tal, à medida que se vai iterando no ciclo *for* na função *calc*, restaura-se esse valor a [0,0].

Para a descida, começa-se na altura j=0 (correspondente à altura h), que é o primeiro valor a ser calculado na coluna, obtém-se o resultado através da soma sucessiva do valor de altura e subida dos blocos nas alturas acima até h-1 (**linha 4 à 11 do pseudocódigo**). No entanto, se j>0, é possível saber o valor utilizando o de descida em j-1, subtraindo com o resultado em j no vetor *previous* e, se possível, somando com os números que se encontram j+h-1 blocos acima. Deste modo, melhoramos a performance ao reaproveitar valores de blocos próximos do atual.

Para a subida, o processo é bastante semelhante, mas, em vez de se começar em j=0, apenas se começa em j=i (diagonal, **linha 21**). Obtém-se o valor da subida do bloco abaixo, subtrai-se o bloco da mesma altura j no vetor anterior, e soma-se o número de subidas no bloco no patamar j-h, se possível.

3.1 Teorema da subestrutura ótima

3.1.1 Sub-problema

Encontrar o número de arcos T que é possível fazer com uma dada altura H e número de blocos n.

3.1.2 Subestrutura ótima

1. Se a solução ótima ao problema contém todas as alturas possíveis para um dado bloco i, então, ao removê-lo, temos uma solução ótima para o número de arcos sem esse bloco.
2. Se a solução ótima para o problema não contiver o bloco i, então temos uma solução ótima para os blocos n-1.

4. Algorithm Analysis

Na função *arc* temos um ciclo que percorre as n peças de lego - $O(n)$ - invocando a função *calc*.

Esta por sua vez contém um ciclo (**linha 2 do pseudocódigo**) que no pior dos casos itera H-h vezes - $O(H - h)$. Esta função ainda contém dois ciclos (**linhas 6 e 24 do pseudocódigo**), que podem acontecer no máximo em dois momentos e que, no pior caso, iteram h vezes - $O\,(h)$.

Assim, a complexidade temporal deste algoritmo é :

$$T(n) = O(n) * (O(H - h) + 2 * O\,(h)) = O(n(H + h))$$

A complexidade espacial é $2 * O(H - h) = O(H - h)$.

5. References

Cormen et al, Introduction to Algorithms (Section 2.1)

“Week 5 - T - Dynamic programming”, Março de 2021, Apresentação PowerPoint