Report for Programming Problem 2 - 2048 Team: 2015236199_2017267408

Student ID: 2015236199 Name: Tiago José Rodrigues Menezes

Student ID: 2017267408 Name: Pedro Carreiro Carvalho

Algorithm Description

Inicialmente é lido a largura máxima, em peças, que é possível um arco ter e a altura máxima a que uma peça pode chegar. De seguida é colocada a primeira peça na *board [0] [0] [0]*, pois todos os arcos começam com uma peça a altura 0 na primeira posição em largura. De seguida, são percorridas todas as células das matrizes até à largura máxima e até ao máximo de altura possível e, com base nas peças da coluna anterior, é calculado o número de peças para cada altura da coluna atual, tanto de peças a subir como de peças a descer. De forma a tornar o algoritmo mais eficiente foi imposta uma limitação (um *speed-up trick*) de forma a que não seja calculado o número de peças que se iniciam a partir de uma altura na qual, com base no número de peças restantes máximo em termos de largura, descendo o máximo possível por peça, não é possível descer até à altura 0.

Para calcular o número de peças a subir existentes em cada altura que pertença ou esteja acima da diagonal da matriz utilizamos a lógica a seguir descrita. Por coluna é guardado numa variável o número de peças resultante da soma do número de peças a subir desde a altura atual menos 1 até à altura mínima a que essa peça possa chegar, tendo assim a variável o número de peças que existem a subir na coluna anterior e que conseguem dar origem a peças a subir na altura atual. Esta variável será referida como *tempUp*. Caso a altura seja igual ao número da peça em largura, ou seja, a diagonal da matriz, o número de peças existentes é o mesmo que existia na coluna anterior e com a altura atual menos 1. Simplificando, toda a diagonal da matriz vai ter o mesmo número de peças. Assim sendo, é adicionado a *tempUp* o valor anterior na diagonal e é definido que a célula atual da matriz é igual a *tempUp*. Para a altura seguinte é então adicionado a *tempUp* o número de peças existente imediatamente abaixo da altura atual e na coluna anterior, pois estas também dão origem a peças a subir na altura atual. Isto acontece até a altura atual ser maior ou igual ao número da coluna mais a altura de uma peça menos 1, pois esta é a altura máxima na qual podem ser colocadas peças a subir pela peça mais abaixo na coluna anterior. A partir dessa altura vai ser necessário também retirar de *tempUp* o número de peças na altura atual menos a altura da peça menos dois, pois as peças nesta posição já não conseguem dar origem a peças a subir na altura atual. De seguida é igualado o número de peças na altura atual a *tempUp.*

Para calcular o número de peças a descer existentes a uma certa altura é necessário ter em conta o número de peças a subir e a descer, pois qualquer uma destas pode dar origem a peças a descer. Para cada coluna é utilizada uma variável, *tempDown2*, inicializada a 1 devido à primeira iteração do

algoritmo, que armazena o número de peças existentes, a subir e a descer, até à altura de uma peça menos 1, ou seja, todas as peças existentes de forma a que ao descer na coluna a seguir, estas consigam chegar a uma peça colocada na altura 0. É utilizada esta abordagem de forma a utilizar a informação calculada na coluna anterior sem necessitar de mais um ciclo para calcular o valor para a altura 0 da coluna atual, aproveitando assim a iteração da coluna anterior. É também criada para cada altura uma variável *tempDown*, que vai ter o valor de peças a descer existentes na altura atual após realizadas as operações descritas abaixo.

Assim sendo, existem duas possibilidades: A altura ser 0 ou maior que 0. Caso a altura seja 0, *tempDown* é igualado a *tempDown2* e o seu valor é colocado a 0. Caso a altura seja maior que 0 então *tempDown* é igualado ao número de peças a descer na mesma coluna e na altura imediatamente abaixo. Após a definição de *tempDown* é retirado o número de peças a subir e a descer existentes na mesma altura na coluna anterior, pois estas já não podem dar origem a peças a descer na altura atual, e é adicionado a este o número de peças existentes a subir e a descer na altura igual à altura atual mais a o tamanho da peça menos um, pois estas ainda não faziam parte de *tempDown*. Por fim é igualado o número de peças dessa altura a *tempDown*.

Por fim basta percorrer a matriz correspondente às peças a descer, somando todos os valores existentes com uma altura 0, obtendo assim a resposta ao problema.

Os sub-problemas repetidos identificados são relativos ao cálculo das peças que podem dar origem às peças de cada altura, sendo a nossa aproximação inicial calcular este número através de ciclos. Após uma análise mais aprofundada reparámos que bastava apenas ir adicionando e retirando um valor do número calculado anteriormente, tal como é descrito acima, melhorando assim a complexidade temporal do algoritmo.

Data Structures

Para resolver este problema, criamos uma matriz tridimensional *board [2] [MAX_N] [MAX_H]*. São utilizadas duas matrizes pois é necessário uma board para armazenar as peças a subir e outra para as peças a descer. *MAX_N* representa o número máximo de peças em largura possíveis de existir e *MAX_H* representa o número máximo de altura ao qual o topo das peças podem chegar.

Correctness

Devido a termos obtido a classificação de 200 pontos iremos explicar o porquê da correção da nossa abordagem. A nossa abordagem é correta pois toda a informação que é útil é reaproveitada, existindo apenas dois ciclos e passando apenas uma vez por cada célula da matriz, aproveitando os ciclos para fazer todos os cálculos necessários para a coluna seguinte. A subestrutura é ótima pois ao calcular o número de soluções para uma determinada largura *x*, sendo

x maior que 3, são também calculadas todas as soluções para qualquer valor da largura entre 3 e *x*.

Algorithm Analysis

A complexidade temporal deste algoritmo é O((N-1)*(H-N)), no *worst case*, sendo N o número máximo de peças em largura e H a altura máxima a que uma peça pode chegar. Isto acontece porque as peças da primeira coluna são sempre iguais e não há necessidade de percorrer essa coluna e porque é impossível uma peça ser colocada a uma altura e o seu topo ultrapassar H, logo o máximo a que uma peça pode ser colocada é H-N, não ultrapassando assim a altura máxima e fazendo com que não haja a necessidade de analisar essas posições.

A complexidade espacial deste algoritmo é O(2*N*H) sendo N o número máximo de peças em largura e H a altura máxima a que uma peça pode chegar. São utilizadas duas matrizes pois uma armazena as peças quando estas estão a subir e a outra armazena as peças quando estas estão a descer pois qualquer peça pode dar origem a peças a descer mas as peças que estão a descer apenas podem dar origem a peças a descer.

References

Como referência foi apenas utilizado os slides fornecidos pelos professores e o apoio fornecido nas aulas práticas.