Report for Programming Problem 2 - ARChitecture

Team:

Student ID: 2018285621 | Name: Nuno Marques da Silva

Student ID: 2018285632 | Name: Pedro Tiago dos Santos Marques

1. Algorithm description

A primeira versão do trabalho foi uma abordagem *top-down* com *memoization*. Esta estratégia tinha como premissa calcular todas as alturas máximas de todos os arcos possíveis e determinar todas as maneiras possíveis de descer para o lado esquerdo e para o lado direito. De seguida multiplicavam-se estes dois valores, obtendo o resultado esperado. Não alcançando a classificação máxima, optámos por transformar o método anteriormente referido numa abordagem *bottom-up*. Note-se que todo o código foi desenvolvido em C++.

De uma maneira simples, o algoritmo inicia com a leitura do *input* e faz a sua verificação. De seguida, cria dois *arrays* dinâmicos (cacheSubir e cacheDescer) com dimensões 'n' por 'H'. A estratégia utilizada reside no seguinte: calcular para todas as posições [i,j] da cache, quantas possibilidades existem de subir (ou descer caso se trate da cacheDescer) até esse ponto. De seguida, é só multiplicar a posição [i,j] da cacheSubir pela posição [i,j] da cacheDescer para saber quantas possibilidades diferentes de construir arcos existem para 'n' igual a i-1 e 'H' igual a j - 1.

- Por exemplo: se o valor de cacheSubir[6, 10] for igual a 20 então significa que existem 20 maneiras diferentes de posicionar blocos a partir da posição 1 e chegar à posição 7, terminando na altura 11. Por outro lado, se o valor de cacheDescer[6, 10] for igual a 15 então significa que existem 15 maneiras diferentes de posicionar blocos para partir da posição 'n' - 1 e chegar à posição 7, terminando na altura 11. Assim, se multiplicarmos estes dois valores obtemos que 20 x 15 = 300 possibilidades diferentes de construir arcos com comprimento 7 e altura 11.

Para concretizar esta estratégia e preencher os *arrays*, estes têm de ser inicializados de forma correta. A cacheSubir implica que todos os arcos começam na posição x = 0, logo apenas marcamos a posição [0][h - 1] a 1. No entanto, a cacheDescer tem de ser inicializada com a linha h - 1 toda igual a 1. Isto deve-se ao facto do arco, ao descer, poder tocar no solo em qualquer posição. O resto dos valores em ambas as caches é inicializado a zero.

Depois de ler o *input* e fazer as preparações, o programa executa o algoritmo dinâmico *bottom-up* onde calculamos cada posição com base em 'h' posições anteriores. Como havia muito *overlap* e estávamos a repetir muitas

contas que já tinham sido anteriormente calculadas, mudámos a estratégia de modo a que reaproveitássemos todas as contas anteriores. Para calcular o valor [i][j], escolhemos o valor imediatamente abaixo do atual ([i][j - 1]), subtraímos o valor [i - 1][j - h] e adicionamos o valor [i - 1][j – 1]. Ora isto permite otimizar muito o programa porque não precisamos de fazer um ciclo para calcular [i][j], ou seja, somar [i -1][j -1] + [i -1][j -2] + … + [i -1][j - h].

Depois de preencher as caches, resta multiplicar a posição [i,j] de cada uma e adicionar ao resultado final.

Esta estratégia não foi suficiente para chegar aos 200 pontos. Para tal foram utilizados 2 *speed-up tricks*. O primeiro é referente ao primeiro *if* presente no ciclo *for* (também utilizado quando se multiplicam as caches) que permite evitar iterações onde não é possível sequer construir arcos, dado que estes nunca poderiam chegar a essa altura naquela posição. O segundo truque é referente à função *int calculalimiares(int Hmax)* que determina para uma determinada *Hmax* qual é a posição máxima que podemos subir. Assim evitamos fazer mais contas quando multiplicamos as caches.

2. Data structures

Para a realização deste trabalho apenas utilizámos dois arrays dinâmicos de inteiros que são devidamente eliminados aquando o término do programa.

3. Correctness

Analisando o problema matematicamente, o nº de arcos possíveis para determinada altura é igual à multiplicação do nº de escadas a subir distintas pelo nº de escadas a descer distintas. Esta multiplicação poupou-nos problemas como não conseguirmos controlar se um arco atingiu altura máxima permitida ou se obedece às regras do enunciado para ser um arco válido.

Usámos duas estruturas para guardar dados (matrizes bidimensionais) que são preenchidas em simultâneo à medida que o programa percorre uma matriz de tamanho “n*H”. No fim deste ciclo, as duas estruturas estarão preenchidas com a quantidade de escadas válidas distintas para a peça do arco representada pelo índice [x][y] - na primeira estrutura: escadas a subir; e na segunda: escadas a descer. Após testes com apenas 1 estrutura para guardar dados, o programa mostrou ser mais lento e por isso descartámos tal opção.

Observando as estruturas de dados à procura de otimizações notámos uma semelhança na distribuição dos valores, há sempre um “triângulo” de zeros tanto no canto superior esquerdo para a estrutura das escadas a subir como no canto superior direito para a das escadas a descer, tendo este “triângulo” a mesma altura e largura para cada input. Isto deve-se ao facto de a largura “n” e a altura da peça “h” estarem relacionadas pois só conseguimos alcançar certas alturas dependendo do quão alto é o passo de peça para peça (h - 1) e da largura

pois precisamos de espaço para poder descer até ao nível inicial. Saltando essas iterações onde é impossível haver um arco válido otimizando o nosso programa ao ponto de alcançar a marca dos 200 pontos.

4. Algorithm Analysis

Dado que o programa segue uma abordagem *bottom-up* e após otimizações conseguimos alcançar uma complexidade temporal de O(2*n*H) = O(n*H), o programa baseia-se em dois ciclos de tamanho “n*H” seguidos. Já a complexidade espacial é de O(2*n*H) = O(n*H) pois são criadas duas estruturas de dados de tamanho “n*H” (cacheSubir e cacheDescer).

5. References

Todo o código apresentado foi desenvolvido pelo grupo sendo que apenas completámos a informação do relatório com informação proveniente dos slides disponibilizados na cadeira de Estratégias Algorítmicas. Finalmente, todas as estratégias de otimização foram introduzidas por nós ou pelo professor nas aulas teóricas.