UNIVERSIDADE DE ÉVORA

1º Trabalho - Conveyor Belts

Estrutura de Dados e Algoritmos II

Professor: Vasco Pedro
Grupo: G128
Realizado por: Miguel Horta (42731), Joaquim Barros (52832)

March 17, 2023

1 Introdução

O primeiro trabalho de Estruturas de Dados e Algoritmos II tem como objetivo resolver o problema Conveyor Belts, sendo este obter o maior valor possível de se obter, com cada produto num tapete diferente com o menor número de pares.

2 Desenvolvimento

2.1 Pensamento

Para a resolução deste problema, após a a leitura do input, guardando para cada produto, em dois arrayList distintos o tipo do produto(tipoProd) e o seu valor(prodVal), estes arrayLists vão conter por ordem o tipo do produto e o seu valor respetivamente, estes arraysLists tem tamanho proporcianal ao número de produtos em cada tapete. Após os dados serem tratados devidamente é chamada a função calMax que se trata da função iterativa que vai calcular o máximo valor possivel com o menos número de pares, tendo como argumentos os tipos de produtos de cada tapete (tipoProd1 e tipoProd2) e os seus valores (prodVal1 e prodVal2) tal como o número de produtos de cada respetivo tapete (nProdutos1 e nProdutos2).

O problema é resolvido através de duas matrizes uma para calcular o valor máximo e outra para calcular o valor minimo de pares. Tais matrizes são complementares, isto é, cada posição (l,c), sendo l o número de linhas e c o número de colunas da matriz de valores corresponde a posição (l,c) da matriz de número de pares.

O pretendido é chegar ao valor final de cada matriz, sendo este o máximo valor e menor número de pares da respetiva matriz, tal posição será a (n,m), sendo n o número de produtos do primeiro tapete e m o número de produtos do segundo tapete.

2.2 Descrição do Algoritmo

Para atingir a solução do problema começamos por determinar o caso base, sendo este a possibilidade de ocorrer que num tapete não exista qualquer produto, isto é, se o valor de n ou m for igual a zero, assim é impossivel fazer um par, sendo que retorna o valor máximo de 0 com 0 pares.

Em seguida resolvemos determinar a primeira posição de cada matriz, ou seja, matrizV[0][0] e matrizM[0][0], matrizV[n][m] representa a matriz dos valores e matrizM[n][m] representa os pares de pacotes, a posição calculada anteriormente determina se os primeiros produtos de cada tapete correspondem como um par, se tal acontecer soma-se os valores de cada arrayList na posição zero e coloca-se o resultado na posição calculada na matriz designada aos valores, como os tipos de cada posição são um par, na posição (0,0) da matrizM é atribuido o valor 1, determinando assim um par, se os tipos da primeira posição de cada tapete não corresponderem na posição (0,0) da matrizM e matrizV é atribuido o valor 0, pois não foi encontrado nenhum par e como tal é associado o valor zero para tal posição da matriz.

Em seguinda é tratada a primeira linha de cada matriz, tal representa o possivel "emparelhamento" do primeiro produto do primeiro tapete com o segundo até a posicão (m-1) do segundo tapete. Se o tipo entre cada tapete corresponder, soma-se os seus valores para a posição (0,j), sendo j o valor de colunas dentro do ciclo "for" que vai até (m-1) e na matrizM fica com o valor 1 respetivo ao par encontrado, se não for encontrado um par para a posição a ser determinada, a matrizV e matrizM ficará com o valor da coluna anterior pois este valor será determinante para o cálculo da restante matriz explicada mais a frente.

Posteriormente será calculada a primeira coluna de cada matriz, tal representa o possivel "emparelhamento" do primeiro produto do segundo tapete com os produtos da segunda posição até a posição (n-1) do primeiro tapete. Tal como calculado para a primeira linha se houver um par para a posição determinada soma-se os valores de cada produto e na matrizM é colocado o valor 1 para determinar esses par, não sendo possivel encontrar um par com o mesmo tipo a posição a ser determinada da matrizV ficará com o valor da matrizV[i-1][0], isto é o valor da matriz uma linha acima da matriz.

Após ter determinado a parte exterior da matriz, vamos calcular a parte interior da matriz desde a posição (1,1) até á posição (n-1, m-1), tal é calculada da seguinte forma, com dois ciclos, o ciclo exterior percorre as linhas de 1 até n-1, e o interior percorre as colunas de 1 até m-1, em seguinda inicializam-se duas variáveis do tipo inteiro, maxV e maxM, maxV corresponde ao valor máximo e maxM corresponde ao número de pares desse valor máximo, inicializadas ambas com o valor -1, usando os valores calculados pelos dois ciclos for anteriores, uma para as linhas e outro para as colunas vamos preencher a mztriz com base nesses valores, determinando inicialmente qual o maior valor entre as posições (i-1,j) e (i,j-1), se o valor da matrizV[i-1][j] for menor que o valor da matrizV[i][j-1] ou se tiverem o mesmo valor mas o número de pares da matrizM[i-1][j] for maior que o número de pares da matrizM[i][j-1] o maior valor será o da posição matrizV[i][j - 1], logo será atribuido á variavel maxV esse valor e por consequência a variavel maxM será atribuido o valor da matrizM[i][j-1], se tal não acontecer será atribuido a tais variáveis o valor de matrizV[i - 1][j] e matrizM[i - 1][j].

Estes valores são os valores antes de se verificar se nas posições i e j existe tipos compativeis para se poderem tornar mais um par, se tal acontecer na variável matchMaxV que é a variável que vai se atribuir o valor máximo a ser calculado na posição atual, tal cálculo é realizado de forma a obter a posição anterir a sua (i-1, j-1) da matrizV e somando a tal valor a soma dos valores de cada tapete nessa posição (i, j), o valor de pares matchMaxM é calculado de forma semelhante, acrescentado o valor 1 a posição (i-1, j-1) da matrizM, comparando em seguinda ambas variáveis com o valor máximo previamente calculado e seu par (maxV e maxM), se maxV for menor ao valor de matchMaxV ou se tem o mesmo valor mas maxM é maior do que matchMaxM, os valores de maxV e maxM são atualizados para os novos valores máximo com o menor número de pares. Colocando em seguida esses valores na posição (i,j) da respetivas matrizes matrizV[i][j] e matrizM[i][j], este ciclo irá se repetir até se obter o valor da matrizV[i][j] e matrizM[i][j], sendo estes o máximo valor com o menor número de pares. Por fim é retornado um array de inteiros com duas posições inicializado no inicio com os valores (0,0), onde a primeira posição corresponde ao máximo valor e a segunda ao menor número de pares.

3 Complexidade Temporal e Espacial

3.1 Temporal

Começámos por proceder à leitura de uma `string` com um número associado a uma variável, `nQuestoes`que determina o número de problemas a tratar, tendo assim complexidade de `O(1)`.

Em seguida, procedeu-se à leitura de cada problema, organizando os dados do primeiro tapete `nProdutos1`, com dois arrayList, uma para o tipo de cada produto `tipoProd1` e outro para o seu valor`prodVal1`, para cada arrayList estamos perante uma complexidade de `O(n)`, onde `n` é o **número de produtos do primeiro tapete**.

Depois procedeu-se exatamente ao mesmo processo, mas desta vez para o segundo tapete, tendo uma complexidade `O(m)` onde `m` é a **número de produtos do segundo tapete**.

Além disso, criámos duas matrizes que têm custo constante `O(1)`.

Finalmente, executámos o nosso **algoritmo**. Durante a execução estamos perante quatro **ciclos**, o primeiro e segundo existem **duas condições** de custo constante `O(1)`, e nos restantes, existe um interior e outro exterior. No ciclo interior existe **três condições** de custo constante `O(1)`, mas o ciclo tem um custo linear de `O(m)` onde `m` é a **número de produtos do segundo tapete**. No ciclo exterior é apenas realizado o ciclo interior, o que para cada iteração temos uma complexidade `O(n)`, então o ciclo vai ter um complexidade O(m x n)=O(n^2) onde `n` é o **número de produtos do primeiro tapete**.

Logo, a **complexidade do programa** será de

$$\mathbf{O(1) + O(n) + O(m) + O(m) + ((O(1) \text{ x } O(1) \text{ x } O(1) \text{ x } O(1)) \text{ x } O(n)) \text{ x } O(m) = O(}n^2\mathbf{)}$$

3.2 Espacial

Durante a inicialização dos `arrays` para guardar **os tipos de produto de cada tapete** e os **valores dos mesmos** é necessário saber quantos **produtos** existem inicialmente, por outras palavras ocupam em memória um número linear, que depende do **número de produtos de cada tapete**, originando assim uma complexidade `O(n)` para o primeiro tapete e `O(m)` para o segundo.

Na inicialização das `matrizes` é fundamental existir o número de **produtos do primeiro tapete e do segundo** para determinar o espaço que as `matrizes` devem ocupar em **memória**. Neste caso iremos ocupar em memória um número exponencial, pois tem que se multiplicar o **número de produtos do primeiro tapete** (n) pelo **número de produtos do segundo tapete** (m), originando assim O(m x n) = O(n^2).

Podemos dizer então que a **complexidade espacial** do programa será de

$$\mathbf{O(n) + O(m) + O(}n^2\mathbf{) = O(}n^2\mathbf{)}$$