Departamento de Informática

Estruturas de Dados e Algoritmos II

Ano Letivo de 2022/2023

Trabalho realizado por:

- Luís Gonçalo Carvalho Nº51817
- Pedro Emílio Nº52649
- Mooshak: g123

Docente: Vasco Pedro

índice

Introdução

Este relatório é referente ao primeiro trabalho da disciplina Estrutura de Dados e Algoritmos II – *"Conveyor Belts"*.

O problema consiste em dois *"Conveyor Belts"*, que carregam uma sequência de produtos (cada produto tem o respetivo nome, tipo e valor).

No decorrer dos *"Conveyor Belts"*, se for encontrado um produto em cada transportador com o mesmo tipo, então podemos fazer um par e podemos remover os produtos. O valor do par é a soma do valor dos dois produtos removidos.

O objetivo do problema é, dados os dois transportadores, é pretendido obter o máximo valor possível com o menor número de pares necessários para obter.

Análise do problema - Input

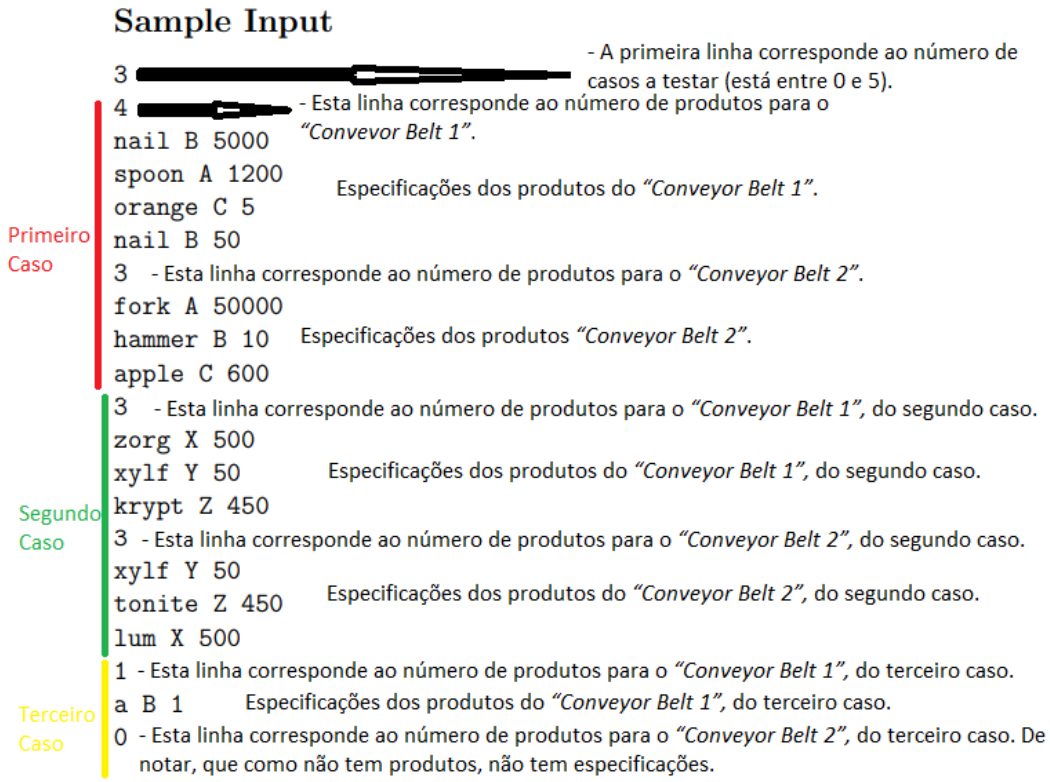

Figura 1: A nossa análise ao "Sample Input" dado pelo professor.

Estruturas Utilizadas

Foi implementada uma classe designada por **Product**, responsável pela criação dos produtos. A classe possuí três variáveis de classe:

- String name: Nome do produto.
- Char type: Tipo do produto.
- Int value: Valor do produto.

De seguida, encontra-se as variáveis que são utilizadas no método **main**:

- Int numCases: Número de casos a testar.
- Int numProducts1: Número de produtos do *"Conveyor Belt 1".*
- List<Product> products1: É uma ArrayList que contém todos os produtos do *"Conveyor Belt 1".*
- Int numProducts2: Número de produtos do *"Conveyor Belt 2".*
- List<Product> products2: É uma ArrayList que contém todos os produtos do *"Conveyor Belt 2".*
- Int[][] maxValue: É uma matriz que vai guardar os valores máximos.
- Int[][] minPairs: É uma matriz que vai guardar os respetivos pares (o mínimo possível) para cada valor máximo.
- Int value: Esta variável vai guardar o valor de um par, ou seja, se for encontrado os dois produtos avaliados com o mesmo tipo, nas diferentes transportadoras, então é somado o valor de cada um e guardado nesta variável.

Algoritmo

1. Criar a classe Product com os atributos nome, tipo e valor.
2. Na classe Main, criar um objeto BufferedReader para ler a entrada do usuário (BufferedReader input). De seguida, ler o número de casos de teste a serem executados e guardá-los na variável numCases.
3. Iniciar um loop for, que começa no primeiro caso e vai até numCases (inclusivo), para testar cada caso separadamente.
4. Para cada caso, ler o número de produtos no conveyor belt 1 e armazená-lo na variável numProducts1. Criar uma ArrayList para guardar os produtos do conveyor belt 1 (ArrayList<Product> products1). Igualmente para o conveyor belt 2.
5. Iniciar um loop for de 0 até numProducts1 e para cada iteração, ler os atributos do produto utilizando o método readLine() e split() da classe BufferedReader. Em seguida, criar um objeto Product com esses valores e adicionar na ArrayList correspondente.

6. Após a leitura de dados, criámos duas matrizes 2D de inteiros, maxValue e minPairs, com dimensões suficientemente grandes para guardar todos os valores máximos e o respetivo número minimo de pares em cada iteração.
7. Iniciar dois loops for, um para percorrer cada produto do conveyor belt 1 e outro para percorrer cada produto do conveyor belt 2.
8. Para cada par de produtos, se os tipos forem iguais, calculamos o valor da soma dos valores dos respetivos produtos.
9. Em cada iteração, é atualizado a matriz maxValue na posição (i, j) com o maior valor entre a posição anterior na mesma coluna (i-1, j), ou o maior valor entre a posição anterior na mesma linha (i, j-1) e o valor da posição diagonal anterior (i-1, j-1) somado com o valor "value" que foi atualizado se houve produtos do mesmo tipo.
10. Em cada iteração, é atualizada também a matriz minPairs de acordo com o valor atualizado na matriz maxValue. Se o valor máximo for igual à posição acima na mesma coluna, então o número de pares é igual. Se o valor máximo for igual na coluna anterior na mesma linha, então o número de pares é igual. Caso o valor máximo seja diferente e se foi encontrado um novo produto, ou seja, a variável value seja maior que zero, então é incrementado o valor minímo de pares nessa posição (i,j) com base ao valor da posição anterior na

diagonal direita, caso contrário, o número mínimo de pares contínua igual ao da posição anterior na diagonal direita.

11. Como queremos o número máximo com o minímo de pares possíveis, visto que esses valores estarão na última posição das matrizes maxValue e minPairs, para dar output, basta fazer System.out.println de cada matriz com as coordenadas (i,j) da última posição, que será numProducts1 e numProducts2, respetivamente.

Análise do problema – Output

Com base no input dado pelo professor e de acordo com o nosso programa, obtivemos o output esperado para todos os casos. Nas seguintes tabelas, encontra-se mais detalhadamente os

	1	2	3
1	maxValue[1][1] = 0 (Não tem nenhuma correspondência de produtos). minPairs[1][1] = 0	maxValue[1][2] = 5010 (Tem correspondência de 2 produtos do tipo B e é o valor máximo e tem um par). minPairs[1][2] = 1	maxValue[1][3] = 5010 (Não tem correspondência, mas o valor máximo é igual à posição anterior na mesma linha). minPairs[1][3] = 1
2	maxValue[2][1] = 51200 (Tem uma nova correspondência o qual tem valor máximo – os produtos spoon A e fork A – Tem um par para este valor máximo). minPairs[2][1] = 1	maxValue[2][2] = 51200 (Não tem correspondência, mas o valor máximo é igual à posição anterior na mesma linha). minPairs[2][2] = 1	maxValue[2][3] = 51200 (Não tem correspondência, mas o valor máximo é igual à posição anterior na mesma linha). minPairs[2][3] = 1
3	maxValue[3][1] = 51200 (Não tem correspondência, mas o valor máximo é igual à posição acima na mesma coluna). minPairs[3][1] = 1	maxValue[3][2] = 51200 (Não tem correspondência, mas o valor máximo é igual à posição anterior na mesma linha). minPairs[3][2] = 1	maxValue[3][3] = 51805 (Tem correspondência entre os produtos C, os valores A+C passam a ser o máximo). minPairs[3][3] = 2
4	maxValue[4][1] = 51200 (Não tem correspondência, mas o valor máximo é igual à posição acima na mesma coluna). minPairs[4][1] = 1	maxValue[4][2] = 51260 (Tem correspondência entre os produtos B, os valores A+B passam a ser o máximo). minPairs[4][2] = 2	maxValue[4][3] = 51805 (Não tem correspondência, mas o valor máximo é igual à posição acima na mesma coluna). minPairs[4][3] = 2

Figura 2: Valores das duas matrizes obtidas no primeiro caso a testar.

	1	2	3
1	maxValue[1][1] = 0 (Não tem correspondência de produtos). minPairs[1][1] = 0	maxValue[1][2] = 0 (Não tem correspondência de produtos). minPairs[1][2] = 0	maxValue[1][3] = 1000 (Tem correspondência de produtos do tipo X). minPairs[1][3] = 1
2	maxValue[2][1] = 100 (Tem correspondência de produtos do tipo Y). minPairs[2][1] = 1	maxValue[2][2] = 1000 (Não tem correspondência, é igual à posição anterior na mesma linha). minPairs[2][2] = 1	maxValue[2][3] = 1000 (Não tem correspondência, é igual à posição anterior na mesma coluna). minPairs[2][3] = 1
3	maxValue[3][1] = 100 (Não tem correspondência, é igual à posição anterior na mesma coluna). minPairs[3][1] = 1	maxValue[3][2] = 1000 (Tem correspondência dos produtos Z e Y). minPairs[3][2] = 2	maxValue[3][3] = 1000 (Não tem correspondência, é igual à posição anterior na mesma coluna, pois tem menor número de pares que a anterior na mesma linha). minPairs[3][3] = 1

Figura 3: Valores das duas matrizes obtidas no segundo caso a testar.

No último caso, o *"Conveyor Belt 2"* não tem produtos, logo o valor máximo é zero e o número de pares é zero.

Complexidade Temporal

A afetação da linha 23, 25, 29, 30, 42, 43, 55, 56 têm complexidade temporal *O*(1).

Os ciclos das linhas 32-39 têm complexidade temporal *O*(numProducts1) e das linhas 45-52 têm complexidade temporal *O*(numProducts2). Dentro desses ciclos as linhas de código têm complexidade temporal *O*(1).

Os ciclos das linhas 59-97 têm complexidade temporal *O*(numProducts1 + numProducts2). As linhas 64, 65, 75, 81, 82, 88, 89, 90, 91, 92, 93, 94 têm todas complexidade temporal *O*(1).

O ciclo for da linha 27, que representa os casos a testar, tem complexidade temporal *O*(numCases * (numProducts1 + numProducts2)).

Concluido:

O(1) + *O*(numProducts1) + *O*(numProducts2) + *O*(numProducts1 + numProducts2) + *O*(numCases * (numProducts1 + numProducts2)) = *O*(numCases * (numProducts1 + numProducts2))

O programa tem complexidade temporal: *O*(numCases * (numProducts1 + numProducts2)).

Complexidade Espacial

As matrizes “maxValue” e “minPair” são criadas com base nos numProducts1 e numProducts2. Logo, a complexidade espacial destas matrizes são *O*(numProducts1 * numProducts2).

No programa, usámos também duas listas, cada uma com o numProducts1 e numProducts2, como número máximo de elementos *O*(numProducts1 + numProducts2).

Logo, a complexidade espacial do programa é de *O*(numProducts1 * numProducts2 + numProducts1 + numProducts2).