# Trabalho Prático 3 - Exposição de tecidos

**Matheus Flávio Gonçalves Silva - 2020006850**

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

matheusfgs@ufmg.br

## 1 Introdução

O problema proposto consiste na análise de uma sequência de rolos possíveis de exibição na vitrine de uma loja de tecidos. Como os rolos são pesados, eles podem ser manuseados apenas uma única vez, colocados à direita ou esquerda dos rolos anteriormente colocados em exibição. Todos os rolos possuem preços diferentes, e Jorginho, o dono da loja, quer colocar em exibição a maior quantidade possível de rolos em ordem decrescente de preços, podendo ou não adicionar novos rolos à medida que são entregues de acordo com uma lista pré-estabelecida de valores dos rolos. Cada rolo deve ser necessariamente colocado à direita, esquerda ou ignorado na mesma ordem em que seu valor é apresentado na entrada.

O trabalho é entendido como uma implementação de algoritmos de programação dinâmica em que é encontrado a dimensão do(s) subvetor(es) que cria(m) o(s) subvetor(es) decrescente(s) de maior dimensão para aquela instância.

## 2 Instruções de Compilação e execução

A compilação e execução do programa é feita de acordo com o indicado na documentação de especificação do trabalho.

**Compilação:**

- Abra um terminal e acesse a pasta raiz do trabalho;
- Execute o comando **'make'**, sem aspas, no terminal;
- Com esse comando, são criados alguns **".o"** intermediários e o arquivo **"tp03"** que é o programa compilado e pronto para execução.

**Execução:**

- Uma vez com o projeto compilado, abra um terminal e acesse a pasta raiz do trabalho;
- Execute o programa com um caso teste com o comando **'./tp03 < arquivo_entrada'**, sem aspas, em que **'arquivo_entrada'** se refere à localização de um arquivo de testes

qualquer, que não necessariamente precisa ter esse nome;

- Com esse comando, é executado o programa para resolver todas as instâncias determinadas pelo arquivo de entrada e, para cada uma, é impresso no terminal um número inteiro referente à quantidade de rolos máxima em exibição para aquela execução.

# 3 Implementação

O programa, bibliotecas e cabeçalhos utilizados foram desenvolvidos na linguagem C++, sendo utilizado o compilador G++ (com padrão de execução conforme previamente ditado no arquivo Makefile disponível no arquivo TP3-Template-CPP).

O desenvolvimento se deu em um notebook Asus X555UB-BRA-XX299T com processador Core i5 6200u e 8GB de RAM 1333Mhz LDDR3 com SSD de 240GB SanDisk em ambiente Linux Mint 21. Foi utilizado o Visual Studio Code como IDE para desenvolvimento.

O trabalho foi entendido como um problema de encontrar o subvetor decrescente de maior comprimento a partir de um vetor original não ordenado. A ideia por trás da implementação é a utilização de subvetores a partir da configuração inicial, sendo um de valores maiores que o primeiro rolo e o outro de valores menores que o desse rolo. A implementação tem como base a ideia de dividir o problema em vários outros tomando como valor inicial sempre uma posição à frente em relação à execução passada. Assim sendo, no máximo são executadas **n - 1** vezes os algoritmos de comparação de dimensão.

Assim sendo, obtém-se a máxima quantidade de rolos em exibição ao pegar a maior soma das dimensões dos maiores subvetores crescentes e decrescentes adjacentes ao rolo de posição **"m"** como sendo o primeiro rolo adicionado à vitrine.

## 3.1 Estruturas de Dados

Para a implementação do trabalho foi usado um **Tipo Abstrato de Dados (TAD)** nomeado **"Store"**, que possui como atributos:

- 3 Inteiros: **size, sizeMore, sizeLess**;
- 3 Vetores de Inteiros: **rollArray, moreThan, lessThan**;

Cada inteiro, respectivamente, armazena a dimensão dos arrays: 'original' de cada rolo colocado como primeiro em exibição **(rollArray)**, array de valores maiores **(moreThan)** e array de valores menores **(lessThan)**. Assim sendo, cada um dos vetores, respectivamente, armazena os vetores 'originais' para cada rolo considerado como primeiro colocado em exibição **(rollArray)**, array de rolos com valores maiores que o primeiro rolo **(moreThan)** e array de rolos com valores menores que o primeiro rolo **(lessThan)**.

Como é utilizado somente uma função de encontrar o maior vetor crescente para todos os subvetores, que nem será explicado futuramente, o vetor **lessThan** tem a peculiaridade de ser criado de forma reversa, em que cada item ao invés de ser adicionado ao final do vetor é adicionado à frente dele. A criação desse vetor é similar à criação de uma lista com o método **push_front()**.

## 3.2 Funções e Procedimentos

### 3.2.1 Arquivo loja.cpp

- **Store::Store()**: Cria e inicializa o **TAD Store** com os vetores tendo tamanho **0**;
- **void Store::setSize(int size)**: Seta o valor **size** passado como argumento como sendo o atributo **size** do **TAD Store**;
- **void Store::setSizeMore(int size)**: Seta o valor **size** passado como argumento como sendo o atributo **sizeMore** do **TAD Store**;
- **void Store::setSizeLess(int size)**: Seta o valor **size** passado como argumento como sendo o atributo **sizeLess** do **TAD Store**;
- **void Store::resetVectors()**: Seta os vetores **moreThan, lessThan** como tendo tamanho **0** para a próxima execução;
- **int Store::getSize()**: Retorna o valor de **size** da **TAD Store**;
- **int Store::getSizeMore()**: Retorna o valor de **sizeMore** da **TAD Store**;
- **int Store::getSizeLess()**: Retorna o valor de **sizeLess** da **TAD Store**;
- **int Store::maxVectorSequenceSize()**: Retorna o valor correspondente ao comprimento da maior subsequência encontrada para a execução atual do programa. Faz uso da função **longestIncreaseSubArray** para encontrar o comprimento dos dois maiores subarrays compatíveis de elementos menores e maiores que o elemento atual. O valor de retorno é definido como **longestIncreaseSubArray(moreThan, sizeMore) + longestIncreaseSubArray(moreThan, sizeMore) + 1**. O retorno é armazenado em uma variável da **main** para comparação em função do comprimento de cada execução;
- **int Store::longestIncreaseSubArray(vector<int>array, int size)**: Recebe um subvetor e retorna o comprimento do maior subarray crescente encontrado dentro dele. É utilizada também para encontrar o maior subvetor decrescente ao se passar **lessThan** que foi criado de forma reversa. Faz uso de mais dois vetores de teste de casos limite **lowestIndices, lastIndices** que armazenam os índices dos menores valores e dos últimos índices que foram adicionados a **lowestIndices**. Faz uso de um laço **for** para comparar um valor atual do laço com o valor anteriormente armazenado, de modo a atualizar caso seja maior que o último armazenado em **lastIndices** ou caso seja menor que o valor armazenado em **lowestIndices**. Caso contrário, faz uso da função **ceilBinarySearch** para determinar um novo valor de teto para próxima possível subsequência analisada. Por fim, utiliza um laço **for** para calcular o tamanho do subvetor encontrado. Esse tamanho é, por fim, retornado;
- **int Store::ceilBinarySearch(vector<int>array, vector<int>& lowestIndices, int left, int right, int key)**: Função de busca binária que recebe um array, um valor de índice correspondente ao menor índice, valores de direita **(0)**, esquerda **(última posição do vetor)** e o valor atual em voga da execução de **longestIncreaseSubArray**. Essa função encontra o valor teto e retorna esse valor para continuar a execução da função que a invocou.

### 3.2.2 Arquivo main.cpp

O programa principal cria uma série de variáveis de auxílio para a execução do algoritmo. Ele é o responsável por fazer a leitura dos dados do arquivo de entrada, criar o loop de execução para todos os casos testes, cria a instância do TAD e atualiza essa instância a cada execução do algoritmo referente ao subvetor atual analisado.

É também responsável por armazenar e comparar os tamanhos dos vetores indicados por cada execução, além de imprimir na tela esse valor uma vez findado o algoritmo.

# 4 Análise de Complexidade

## 4.1 Complexidade de tempo:

A execução do programa é dominado assintoticamente pela execução da função **longestIncreaseSubArray**. Pela definição do algoritmo em si, conforme explicado anteriormente, ela é executada de modo que:

$$\mathrm{F(n)} = \mathrm{F(n\text{-}1)} + \mathrm{n} + 1$$

Por sua vez, F(n) = F(n-1) + n + 1 é dita como:

$$\mathrm{F(n)} = \mathrm{F(n - 2)} + \mathrm{(n - 1)} + \mathrm{n} + 2$$

E assim sucessivamente, até que se obtém:

$$\mathrm{F(n)} = \mathrm{F(0)} + 1 + 2 + ... + \mathrm{(n - 2)} + \mathrm{(n - 1)} + \mathrm{n} + \mathrm{n}$$

Como, a definição do algoritmo é de executar até o caso em que n > totalSum, F(0) é $O(0)$, enquanto F(1) é $O(1)$ por se tratar de um vetor de tamanho 1, de modo que é trivial. Assim sendo, a relação de recorrência pode ser reescrita como:

$$\mathrm{F(n)} = 0 + \frac{n(n+1)}{2} + \mathrm{n}$$

$$\mathrm{F(n)} = \frac{n^2+n}{2} + \mathrm{n} = \frac{n^2+3n}{2}$$

$$\mathrm{F(n)} = O(n^2)$$

## 4.2 Complexidade de espaço:

A complexidade de espaço é determinada pela TAD **"Store"** e pelos vetores de caso limite da função **longestIncreaseSubArray**, em que o espaço utilizado é dominado pelos 3 vetores contidos na TAD e pelos 2 vetores criados pela função. Como o vetor **rollArray** armazena os **"n"** rolos, e os vetores **moreThan, lessThan** são na realidade subvetores de tamanho máximo **"n - 1"**, uma vez que são criados a partir de **"rollArray"** sem o primeiro elemento, além de que os vetores auxiliares tem o limite de tamanho determinado também por **"n - 1"**, uma vez que provém dos subvetores, temos que a complexidade de espaço é dada por:

$$\mathrm{F(n)} = O(n) + O(n-1) + O(n-1) + O(n-1) + O(n-1)$$

$$\mathrm{F(n)} = 5O(n)$$

$$\mathrm{F(n)} = O(n)$$

# 5 Conclusão

Por fim, a execução do trabalho foi concluída de modo a corresponder a 100% de acerto em relação aos casos testes apresentados. No entanto, a velocidade do algoritmo desenvolvido não parece muito promissora, sendo provavelmente mais lento do que era esperado.

Ademais, a compreensão e desenvolvimento do trabalho em si se deu com dificuldade, tanto por conseguir um programa aparentemente lento quanto por ter sido extremamente difícil chegar a um pensamento inicial que culminou na execução do trabalho entregue. O desenvolvimento do que é proposto só se deu a menos de um dia da entrega e com muita dificuldade.

Com isso, conclui-se que é necessário revisar os conceitos apresentados correspondentes ao que o trabalho propõe para melhor entendimento e execução de programas similares.

# References

**de Almeida, J. M.** Slides virtuais da disciplina de Algoritmos 1 (2022).

Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte. Acesso em 11 dez. 2022

**Mortensen, P.**"How do I erase an element from std::vector<>by index?"

Disponível em: <https://stackoverflow.com/questions/875103/how-do-i-erase-an-element-from-stdvector-by-index>. Acesso em 11 dez. 2022

**geeksforgeeks** "Longest Increasing Subsequence — DP-3"

Disponível em: <https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/. Acesso em 11 dez. 2022