Documentação Trabalho Prático 3

Lucas Roberto Santos Avelar

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG) - Belo Horizonte – MG – Brasil

lucasrsavelar@ufmg.br

1. Introdução

Esta documentação lida com o problema da criação de um subvetor decrescente com o maior comprimento possível a partir de um vetor principal de inteiros. Para cada valor presente no vetor principal, existem três opções possíveis (considerando que o subvetor não está vazio; nesse caso qualquer opção de adição possui mesmo resultado): adicionar o valor ao início do subvetor, caso ele seja maior do que o atual valor da primeira posição; adicionar o valor ao final do subvetor, caso ele seja menor do que o atual valor da última posição; ou não adicionar o valor. Vale ressaltar que não há retorno para a última opção, visto que não é possível retornar a um valor que se decidiu por não adicionar. Desse modo, é importante considerar, para cada valor do vetor principal, as possíveis combinações levando em conta sua adição ou não ao subvetor.

2. Implementação

O código organiza-se principalmente em 2 blocos. Dentro dos arquivos, encontram-se alguns comentários sobre as operações de cada função. A seguir, há uma análise mais detalhada.

O primeiro bloco corresponde ao arquivo “main.cpp”, onde está localizada a função main do programa. Trata de um bloco mais trivial, que apenas declara algumas variáveis necessárias e lê da entrada padrão os valores da quantidade de casos de teste, a quantidade de rolos em cada caso de teste e o valor dos rolos. O programa, então, entra em loop enquanto existem casos de teste e, para cada um deles, cria um vetor contendo todos os rolos nas duas posições possíveis e chama a função LDS para calcular o maior número de produtos que

podem ser expostos na prateleira, imprimindo o resultado na saída padrão.

Nesse primeiro bloco, vale ressaltar o laço que percorre todos os elementos (menos o primeiro, que é adicionado anteriormente) do vetor de rolos e adiciona cada um deles tanto “à esquerda” (início) quanto “à direita” (fim) do vetor tam. Essa foi uma forma encontrada para encontrar todas as possibilidades incluindo ou não um rolo, ao invés de realizar chamadas recursivas considerando ou não a inclusão de um determinado rolo. A ideia é que caso um rolo seja menor do que o anterior ou maior que o próximo, ele se enquadra nas definições do enunciado e, portanto, será considerado no cálculo da Longest Decreasing Subsequence (LDS, a ser discutida em breve). Porém, caso ele seja maior que o anterior ou menor que o próximo, ele não segue as especificações do enunciado e a função não o levará em conta no cálculo. Dessa forma, é como se todos os casos (inclusão ou não de qualquer rolo) fossem considerados, mas apenas os que são efetivamente válidos são levados em conta nos cálculos.

Já o segundo bloco consiste nos arquivos “loja.hpp” e “loja.cpp”, que reúnem respectivamente a declaração e a implementação das funções fundamentais para o funcionamento do programa.

Nas funções, LDS consiste em uma adaptação do algoritmo de LIS, mas para calcular a maior subseqência decrescente (o que é exigido no enunciado), portanto é um algoritmo de Longest Decreasing Subsequence e recebe o vetor tam como parâmetro. Funciona de forma quase completamente análoga à LIS; utilizando um array para Programação Dinâmica que armazena o valor da maior subsequência incluindo cada rolo e dois laços aninhados que verificam se um elemento posterior é menor e se sua adição aumenta o tamanho da maior subsequência. Finalmente, a função retorna o maior elemento do array auxiliar, e para isso faz uso da função maiorElemento, que consiste apenas de um laço procurando o maior elemento do array.

A implementação, então, se deu principalmente a partir da relação de recorrência do algoritmo de LDS, sendo L o comprimento de LDS terminado com índice i:

$L(i) = 1$, se $i = 0$

$L(i) = \max(L(j)) + 1$, se $j < i$ e $rolo[j] > rolo[i]$

Para a complexidade, temos que o vetor tam consiste em 2n – 1 elementos, pois cada elemento é inserido no início e no final do vetor, com exceção do primeiro. Na função LDS, temos dois laços aninhados que iteram sobre cada elemento de tam, logo, temos (2n-1)^2 = 4n^2 + 4n + 1. Assintoticamente, podemos desconsiderar o + 1, restando 4n^2 + 4n = 4(n^2 + n). Novamente, podemos desconsiderar a constante multiplicativa e resta n^2 + n, o que pode ser considerado como O(n^2) para a resolução do problema. A função maiorElemento é O(n), já que apenas itera pelos n elementos de um array, então a complexidade O(n^2) é a que domina a função LDS e também o programa em um geral.

Previamente, havia chegado em um algoritmo recursivo com 2^n de complexidade (duas chamadas para cada rolo, incluindo ou não). Acredito que a entrega desse novo algoritmo mudando a ideia por trás de considerar ou não um rolo, mesmo com atraso, valha a pena, pois ele é muito mais suscetível a conseguir calcular entradas muito maiores.