Trabalho Prático **03**

Documentação

DCC206 Algoritmos 1

Universidade Federal de Minas Gerais ***2022/02***
Bernardo Reis de Almeida *(2021032234)*

1. Introdução

O presente trabalho prático tem por objetivo a aplicação de conceitos relacionados à teoria dos **paradigmas de programação** e, em especial, à **programação dinâmica**, sobre uma situação problema real. Esta será computacionalmente modelada por algoritmos e estruturas que dados que a resolvam de maneira rápida e eficiente à luz do paradigma mencionado. A presente implementação foi realizada na linguagem **C++** - o compilador utilizado foi o **G++** - e em uma plataforma **Unix**. A seguir, o problema em questão e suas especificações serão detalhados.

Um dono de varejo comercializa rolos de tecido e, todo o dia, recebe uma nova remessa de material da qual deve selecionar uma quantidade de produtos para dispor na estante de sua loja. Tendo em vista a maximização dos lucros, é desejado que o **maior número possível de rolos** seja posto à mostra, mas, para fins de controle e de organização, estes devem ser organizados em **ordem decrescente de preço**. Cada rolo pode ser colocado na estante pelo lado esquerdo ou pelo lado direito, não sendo possível colocar alguma peça entre outras duas, caso já dispostas. Além disso, os rolos são recebidos um por um e, uma vez em mãos, a decisão quanto a sua inclusão ou não deve ser tomada, dado que eles são muito pesados e sua manipulação, custosa. Um último detalhe é que cada rolo apresenta um preço único, isto é, não há dois rolos com o mesmo preço. Com isso em mente, o dono da loja deseja saber, dados a **quantidade de rolos** presente na remessa e o **preço** de cada um, qual seria a **maior quantidade possível de produtos que podem ser colocados na estante de modo a satisfazer os critérios estabelecidos**.

Em termos técnicos, dados como **entrada** um inteiro que representa a **quantidade de rolos de tecido** e um **vetor contendo os preços individuais** de cada um na ordem em que foram recebidos, o programa deve **retornar** um inteiro que representa a **quantidade máxima** de rolos que podem ser dispostos na estante em ordem decrescente, satisfazendo as diretrizes de manipulação. O algoritmo ainda deve ser capaz de processar vários casos de teste, sendo a quantidade exata também passada como uma informação pela entrada de dados.

Em um primeiro momento, decisões de implementação e escolhas de estruturas de dados, assim como a equação de recorrência e a análise de complexidade do algoritmo, serão detalhados e justificados na seção **[Modelagem]**. Em seguida, um panorama geral de funcionamento, do fornecimento da entrada à produção da saída, será abordado na seção **[Execução & Funcionamento]**. Por fim, instruções para compilação e para execução e referências bibliográficas podem ser encontradas na seção **[Considerações Finais]**.

2. Modelagem

A **loja de tecidos** foi modelada por meio de um tipo **abstrato de dados**, o qual irá conter suas principais informações, sejam elas o **tamanho da remessa de rolos** e os **preços individuais de cada um**, organizados por ordem de recebimento. A primeira foi abstraída como uma variável simples, enquanto a segunda, por um vetor de inteiros, o qual é diretamente preenchido com as informações provenientes da entrada padrão, dado que estas já se encontram ordenadas conforme estabelecido. A escolha de um vetor é uma consequência direta da natureza linear e sequencial dos dados, tendo em mente a simplicidade da implementação.

Ao se analisar o problema, percebe-se que a resposta está associada a um **subconjunto máximo da remessa de rolos** que satisfaz os seguintes critérios: ao ser recebido, o rolo pode ser incluído pela esquerda, pela direita ou pode não ser incluído, além de que eles devem estar em ordem decrescente de preço na estante. Nesse sentido, observa-se que esse subconjunto é construído do interior para o exterior, de tal forma que, conforme recebidos em ordem, rolos menores que o atual menor são adicionados à direita, enquanto rolos maiores que o atual maior são adicionados à esquerda. Também é possível não adicionar, o que é válido quando não houver margem (os limites inferior e superior do subconjunto presente na estante não permitem) ou caso esta decisão local seja melhor para o resultado total (colocar limitaria as opções posteriormente). Uma consequência desses critérios é justamente que **os índices relativos à ordem de recebimento dos rolos diminuem do exterior para o interior no subconjunto final**, convergindo em um **elemento "central"** (o qual, inclusive, pode estar em uma das extremidades). A seguir, tem-se exemplos que ilustram essa propriedade em duas instâncias distintas.

Exemplo 1

Sequência de preços por ordem de recebimento
15 12 5 8 1 17 13 6 19 18 9 3 4 16 7 20 2 11 10 14

Subconjunto ordenado que maximiza a quantidade de rolos exibidos
-> 20**(15)** 18**(9)** 17**(5)** 15**(0)** 12**(1)** 8**(3)** 6**(7)** 4**(12)** 2**(16) <-**

Exemplo 2

Sequência de preços por ordem de recebimento
19 7 9 15 14 8 5 17 18 13 12 11 2 20 4 6 3 16 10 1

Subconjunto ordenado que maximiza a quantidade de rolos exibidos
-> 20**(13)** 18**(8)** 17**(7)** 15**(3)** 14**(4)** 13**(9)** 12**(10)** 11**(11)** 6**(15)** 3**(16)** 1**(19) <-**

A solução para o problema, pois, está dentro do **espaço de soluções** que compreende os subconjuntos máximos de elementos que podem ser construídos tomando-se um específico como o "central". É justamente essa propriedade que será o cerne para o algoritmo proposto: para cada rolo recebido, serão calculados o **maior subconjunto crescente de elementos com preço maior** e o **maior subconjunto decrescente de elementos com preço menor** que podem ser construídos com o restante dos rolos de índice maior, utilizando-se o paradigma da **programação dinâmica** como norte para os cálculos e para as operações.

Em mais detalhes, o vetor de preços será percorrido de trás para frente a partir do final, isto é, do último rolo recebido, e, para cada elemento, serão determinados os tamanhos dos subconjuntos máximos que podem ser formados com os demais posteriores incluindo-o e em ambas as ordens, crescente (para preços maiores) e decrescente (para preços menores).

Sequência de preços por ordem de recebimento
19 7 9 15 14 8 5 17 18 13 12 11 2 20 4 6 3 16 10 1

Subconjunto máximo crescente de preços maiores tomando o 15 como "central"
19 7 9 **| 15 |** 14 8 5 **17 18** 13 12 11 2 **20** 4 6 3 16 10 1

Subconjunto máximo decrescente de preços menores tomando o 15 como "central"
19 7 9 **| 15 | 14** 8 5 17 18 **13 12 11** 2 20 4 **6 3** 16 10 **1**

Esse cálculo, por sua vez, será uma função dos resultados anteriores - ou, em ordem de recebimento, do resultado para os rolos posteriores -, no sentido de que, dado um elemento **(i)**, determina-se o subconjunto máximo de elementos posteriores em que ele pode ser incluído de modo a maximizar a quantidade final de rolos. Dito de outra forma, o **problema** de se determinar os subconjuntos máximos para um item, tal como foi definido, será quebrado nos **subproblemas** de determinar os subconjuntos máximos para os elementos posteriores, sendo estes **armazenados** para a reutilização ao decorrer da execução do algoritmo. Dessa forma, é possível definir as **equações de recorrência** para ambas essas otimizações como uma função dos resultados anteriores. Ambas essas equações são apresentadas a seguir, em que **$OPT_{crescente}(i)$** equivale ao tamanho do máximo subconjunto crescente de elementos com preço maior começando e incluindo o elemento de índice **(i)**, **$OPT_{decrescente}(i)$** equivale ao tamanho do máximo subconjunto decrescente de elementos com preço menor começando e incluindo o elemento de índice **(i)** e **p(x)** equivale ao preço do elemento de índice **(x)**.

$$OPT_{crescente}(i) = \max_{i<j \,\&\, p(i)<p(j)} \{1 + OPT(j)\}$$
$$= 1 \text{, caso não haja tal (j).}$$

$$OPT_{decrescente}(i) = \max_{i<j \,\&\, p(i)>p(j)} \{1 + OPT(j)\}$$
$$= 1 \text{, caso não haja tal (j).}$$

Determinados os subconjuntos máximos para cada rolo, a busca pela solução dentro do espaço de soluções se reduz a assumir que algum seja o elemento "central", somar o subconjunto máximo que pode ser colocado à sua esquerda (vulgo **$OPT_{crescente}(i)$**) ao subconjunto máximo que pode ser colocado à sua direita (vulgo **$OPT_{decrescente}(i)$**) e verificar se este é o maior resultado já encontrado. A solução desejada para o problema, pois, será justamente o maior valor obtido. Aqui, pontua-se que os critérios de organização dos rolos na estante são sempre mantidos, já que **$OPT_{crescente}(i)$** é um subconjunto de elementos com preço maior que o preço do elemento de índice **(i)**, **$OPT_{decrescente}(i)$**, de elementos com preço menor que o preço do elemento de índice **(i)** e os índices de todos os rolos nesses dois subconjuntos são maiores que **(i)**, de modo que possam ser colocados conforme as regras previamente estabelecidas.

A escolha por essa abordagem é justificada principalmente pelos ganhos em termos da **ordem de complexidade** do algoritmo. Em especial, observa-se que o cálculo das funções **OPT (i)** exige, para cada elemento **(i)**, percorrer-se no máximo as soluções para os outros **|vetor| - i** elementos posteriores a ele, em que **|vetor|** é o tamanho dessa estrutura de dados (ou o número de rolos na remessa). Esta operação, por sua vez, é **proporcional de maneira quadrática apenas à quantidade de elementos**, uma vez que as soluções são calculadas progressivamente a partir da extremidade final e armazenadas em uma estrutura auxiliar, de modo que, para cada novo elemento avaliado, não seja necessário recalcular a resposta para os anteriores, apenas as verificar. Além disso, a navegação no espaço de soluções em busca da resposta ótima é **linearmente proporcional à quantidade de elementos do vetor**, já que, para cada um, é verificado se o subconjunto máximo tendo ele como elemento "central" é maior do que aqueles já vistos, o que envolve apenas contabilizar os valores das funções **OPT (i)** em operações constantes. Em suma, **a complexidade temporal do algoritmo é da ordem O(|vetor|^2 + |vetor|) = O(|vetor|^2)**, isto é, **quadrática com relação à quantidade de rolos** recebidos pela loja.

3. Execução & Funcionamento

O programa começa sua execução declarando uma variável que armazenará a **quantidade de casos de teste** que serão avaliados, reiterando-se a capacidade do algoritmo de processar várias instâncias do problema em uma única execução. Essa informação é devidamente extraída da entrada de dados e armazenada internamente, como já definido.

Após isso, para cada instância do problema, um **objeto do tipo que abstrai a loja** é inicializado e suas informações iniciais, sejam elas a **quantidade de rolos** recebidos na remessa e o vetor com os **preços de cada um**, em ordem de recebimento, são lidas da entrada padrão e armazenadas nas estruturas de dados correspondentes.

Em seguida, o **algoritmo dedicado a encontrar o tamanho do subconjunto máximo de rolos que satisfazem os critérios estabelecidos** é executado sobre o objeto referente à loja por meio de um método que age conforme aquilo descrito na seção anterior. Este método retorna o **resultado**, seja ele um inteiro que indica tal quantidade, o qual **é impresso na saída padrão**.

Por fim, o próprio objeto já está implementado de tal forma a realizar a limpeza das variáveis e a desalocação da memória utilizada ao decorrer da execução.

4. Considerações Finais

Instruções para compilação e para execução

1. Extraia o conteúdo do arquivo .zip disponibilizado.

2. Acesse o diretório **</tp3>** por meio do comando **<$ cd tp3>**.

3. Execute no terminal o comando **<$ make all>**. Esse comando deverá gerar um executável denominado **tp03**.

4. A execução do programa é realizada por meio do comando **<$ ./tp03 < input.txt>**, sendo **input.txt** o arquivo de texto contendo os dados de entrada.

5. A saída será retornada pela saída padrão de dados.

Referências Bibliográficas

Stack Exchange, Inc.. *Stack Overflow* [Online]. Disponível em: https://stackoverflow.com/ (Acessado em: 09/12/2022).

GeeksforGeeks. *Longest Increasing Subsequence | DP-3* [Online]. Disponível em: https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/ (Acessado em: 09/12/2022).