Trabalho Prático 3 - Programação Dinâmica
Universidade Federal de Minas Gerais

Igor Rahzel Colares Galdino
Matricula 2020096255

Dezembro 2022

1 Introdução

O Trabalho Prático consiste na modelagem, através do paradigma da Programação Dinâmica, do problema de se descobrir o maior subconjunto de rolos de tecido que podem ser posicionados em uma prateleira de modo que seus preços estejam em ordem decrescente, respeitando as seguintes restrições: os rolos são manuseados um por vez e podem ser inseridos pela esquerda, pela direita, há também a possibilidade de não inserir o rolo na prateleira.

2 Modelagem

Como mencionado anteriormente, as instruções do trabalho prático restringiam a modelagem do problema à utilização do paradigma da Programação Dinâmica, ou seja, devemos resolver o problema por meio dos seus resultados previamente computados.

Para resolver um problema por meio de Programação Dinâmica devemos obter a equação de Bellman. Vamos então inicialmente definir OPT_E(i) como número máximo de elementos do conjunto $A = \{j \in R \mid i < j \leq n\}$ que podem ser adicionados à esquerda do *i-ésimo* rolo simultaneamente, onde R é o conjunto contendo todos os rolos, e **n** é o número de rolos recebidos. Analogamente definimos OPT_D(i), porém nesse caso consideramos a quantidade máxima de rolos que podem ser adicionados à direita. O objetivo através da Programação Dinâmica é calcular $OPT_E(1)$ e $OPT_D(1)$, pois assim será possível determinar a melhor configuração possível para cada um dos rolos, respeitando as restrições impostas.

Após computarmos $OPT_E(1)$ e $OPT_D(1)$, basta calcular $max\{$ OPT_E*(i)* + OPT_D*(i)*$\}$, onde $1 \leq i \leq n$, e então adicionar 1 no resultado obtido, pois o i-ésimo rolo que maximiza o número de elementos a sua esquerda e direita também deve ser incluído.

2.1 Equação de Bellman

A partir das definições acima podemos escrever para OPT_E(i) e OPT_D(i) a seguinte equação de Bellman:

$$OPT(i) = \begin{cases} 0, \text{ se tal j não existe} \\ 1 + Max(OPT(j)), \text{ se } i < j \leq n \end{cases} . \tag{1}$$

Essa solução explora a propriedade transitiva dos operadores $<$*(menor que)* e $>$*(maior que)*, pois como começamos percorrendo o vetor de preços de trás para frente na construção do *OPT(i)* até chegarmos ao *OPT(1)*, é possível verificar quantos números vão poder ser inseridos a sua esquerda e sua direita apenas olhando para a melhor das soluções anteriores que for compatível.

3 Exemplo

Seja **A** o vetor contendo os preços dos rolos de tecido na ordem em que foram recebidos.

A =

9	1	3	8	4	5	10	7	6

E sejam os vetores **E**, **D**, a quantidade máxima de rolos que pode ser colocada a esquerda e a direita respectivamente do i-ésimo rolo utilizando os rolos nas posições i+1 até n do vetor **A**:

E =

0	0	0	0	0	0	0	0	0

D =

0	0	0	0	0	0	0	0	0

Dessa forma devemos computar $E[9]$ até o $E[1]$ e faremos o mesmo para $D[i]$, conforme descrito anteriormente no algoritmo. Nesse exemplo trabalharemos somente com o vetor **E**, porém o raciocínio utilizado é o mesmo para preencher o vetor **D**.Temos que $E[9] = 0$, uma vez que não existe j, tal que $9 < j \leq 9$, o $E[8] = 0$, uma vez que $A[8] > A[9]$, pela mesma razão $E[7] = 0$, como A[6] < A[7] temos que $E[6] = 1 + E[7] = 1$, é importante observar que A[6] também é comparado com A[8] e A[9], no entanto independentemente da configuração escolhida o número máximo de rolos que poderão estar a esquerda do sexto rolo é 1. Esse processo é repetido até chegarmos a E[1], no final de todo esse processo os vetores **E** e **D** apresentaram a seguinte configuração:

E =

1	4	3	2	2	1	0	0	0

D =

3	0	0	2	0	0	2	1	0

Temos que o MAX{E[i] + D[i]} = 4, adicionado 1 no máximo obtido, ou seja, incluindo o i-ésimo rolo, obtemos que é possível colocar um máximo de 5 rolos na prateleira de modo que os preços estejam em ordem decrescente.

4 Algoritmos e Estruturas de Dados

Para resolver esse problema foi criada a classe *loja* que possui como atributos *int numMaterial* que é responsável por armazenar o número de rolos do caso de teste, *vector<int>left* e *vector<int>right* que armazenam o número máximo de elemento que podem ser adicionados a esquerda e a direita do i-ésimo rolo respectivamente, e por fim *vec_prices* armazena os preços dos rolos. Os métodos dessa classe estão descritos abaixo:

- *void setVecPrices(vector<int>vec)*: atribui o vetor de inteiros com os preços dos rolos a *vector<int>vec_prices*.
- *void setNumMaterial(int n)*: atribui n a *numMaterial*.
- *void initializeLeftAndRight(int numMaterial)*: incializa os vetores *left* e *right* com tamanho igual ao número de rolos do caso teste e todas as entradas com zero.
- *int findOptimal()*: Encontra o número máximo de elementos que podem ser inseridos a esquerda e direita de cada um dos elementos da entrada e retorna o máximo possível.
- *int MAX*: retorna max{*left[i] + right[i]*}

5 Análise de Complexidade

A função *setNumMaterial* tem complexidade $O(1)$, pois apenas realiza uma atribuição, já a função *setVecPrices* tem complexidade $O(n)$,pois copia-se cada elemento do vetor recebido como parâmetro para *vec_prices* .O método *initilizeLeftAndRight* tem complexidade $O(n)$, pois inicializa todas as entradas dos vetores *left* e *right* com zero. A função *MAX* tem complexidade assintótica $O(n)$, pois percorre cada um dos elementos dos vetores *left* e *right* para identificar a maior soma *left[i] + right[i]*. Por fim o método *findOptimal* tem complexidade $O(n^2)$, pois o loop externo vai de n-2 até zero, já o loop interno itera desde n-1 até a posição do loop externo. Essa configuração implica na seguinte soma a cada iteração do loop externo:

$$1 + 2 + 3 + \cdots + n - 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = O(n^2) \tag{2}$$

Como cada um dos métodos descritos acima é chamado apenas uma vez a cada caso teste, temos que a complexidade assintótica é igual $O(n^2)$.

6 Conclusão

O trabalho prático visou a modelagem do problema de encontrar o maior subconjunto de elementos que podem ser colocados em ordem decrescente, de modo que cada elemento pudesse ser inserido na esquerda, na direita ou descartado daquela configuração, porém havia a restrição de modelá-lo através do paradigma da Programação Dinâmica, para isso visou-se obter o elemento do conjunto que permitia a maior quantidade de elementos adicionados simultaneamente a sua direita e a sua esquerda.

O fato do trabalho prático exigir o uso da modelagem do problema por meio do paradigma da Programação Dinâmica foi desafiador, porém muito interessante e proveitoso, pois isso implicou que o problema não fosse abordado apenas com o intuito de ser resolvido, mas sim ser resolvido de uma forma específica, o que propõe uma maneira, talvez, não convencional de enxergá-lo.