Trabalho Prático 2
Exposição de Tecidos

Vitor Emanuel Ferreira Vital - 2021032072

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

vitorvital@dcc.ufmg.br

1. Introdução

Esta documentação tem como objetivo apresentar a implementação e modelagem da resolução do Trabalho Prático 3 da disciplina de Algoritmos 1. Em tal prática, os principais temas explorados, solucionados na linguagem C++, são:

- Resolução de problemas reais via modelos algorítmicos
- Utilização de estruturas de dados estudados na disciplina
- Utilização de algoritmos de divisão e conquista

O problema a ser tratado lida com a exposição de rolos de tecidos em uma loja, no qual, dado uma remessa de rolos de tecido e seus respectivos preços, deve-se encontrar uma forma de expor a maior quantidade de rolos de forma que os seus valores estejam em ordem decrescente, sendo possível inserir à esquerda, à direita ou não inserir nas prateleiras, na sequência em que eles chegaram. Com o objetivo de uma exposição máxima dos produtos, visto que a disposição dos produtos em ordem decrescente de valor aumenta o número de vendas, encontraremos as melhores ações à disposição.

A documentação explicita os métodos e modelagens utilizados para resolução do problema em questão.

2. Modelagem

Conforme explicitado, os tecidos devem ser inseridos (ou não) na prateleira conforme sua ordem de chegada, devido à dificuldade de movimentação dos rolos. Dentre todas as escolhas realizadas, o programa deve, ao final, possuir à prateleira com valores decrescentes com maior quantidade de rolos possíveis, seguindo a seguinte regra:

1. colocá-lo na prateleira pelo lado direito e empurrá-lo até encostar nos rolos já existentes,
2. colocá-lo na prateleira pelo lado esquerdo e empurrá-lo até encostar nos rolos já existentes,
3. simplesmente não colocá-lo na prateleira.

Com isso em mãos podemos fazer a modelagem seguindo o problema de encontrar a maior sequência decrescente dado um vetor inicial. Nesse caso, dado um array, deseja-se verificar a sequência decrescente com a maior quantia de elementos possível.

Para o caso em questão utiliza-se o algoritmo de programação dinâmica, esse algoritmo possui comportamento semelhante ao da divisão e conquista, entretanto leva em conta uma subestrutura ótima e uma superposição de subproblemas. Nesse caso, utilizamos resultados de operações anteriores para solucionarmos um subproblema em questão.

Algoritmo de Programação Dinâmica:

Ao invés de calcular a subsequência iniciando em cada rolo de tecido e terminando em outro, o que geraria um custo exagerado de tempo, o algoritmo em questão utiliza os dados já calculados anteriormente para utilizado no futuro, ou seja, caso dado obtido é então armazenado e, caso venha a ser necessário posteriormente no programa, ele é obtido em O(1).

Como o problema em questão permite a inserção de rolos de tapetes tanto ao lado esquerdo quanto ao direito, ocorre um problema que deve ser avaliado. Se considerarmos uma sequência de tapetes sendo inseridos em sequência, sendo seus respectivos valores {6,7,3,5}, a maior sequência decrescente obtida podendo inserir apenas em um lado é {6,3} ou {7,5}, porém o que ocorre nesse problema é que podemos obter a sequência {7,6,5}, pois podemos inseri-los em lados opostos para posteriormente analisar a ordem decrescente.

Portanto a solução obtida segue o seguinte algoritmo:

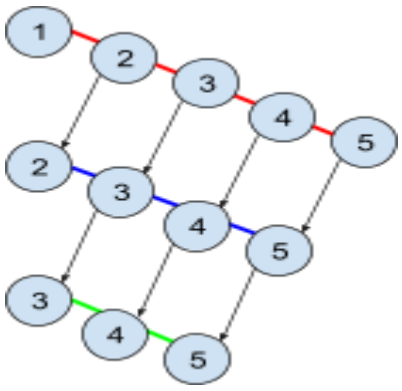
1. a partir de um rolo de tecido i, analisamos a maior sequência decrescente dos rolos posteriores e guardamos a quantidade obtida em um vetor;
2. após avaliar a sequência decrescente, faremos a análise procurando a maior sequência crescente dos valores dos rolos, armazenando os dados em um novo vetor;
3. por fim, analisamos a soma obtida no passo 1 mais a obtida no passo 2, pois podemos criar uma sequência crescente do lado direito da prateleira e uma decrescente do lado esquerdo;
4. realizamos tais passos iniciando em cada rolo, na sequência em que eles chegam e retornamos a maior soma obtida.

Com base nessa implementação, precisa-se manter o registro das maiores sequências obtidas, ou seja, ao calcularmos a forma decrescente ou crescente, armazenamos respectivamente em lds[i] e lis[i], o que indica que a maior sequência crescente ou decrescente iniciando em i é lds[i] ou lis[i].

Por fim, basta que analisemos a soma (lds[i] + lis[i] - 1) - o menos um ocorre pois o elemento i é incluído 2 vezes - para todas as n possibilidades retornando a maior soma obtida.

Relação de recorrência:

É importante salientar que ao iniciarmos a análise das sequências pelo primeiro elemento, devemos analisar os posteriores. Isso ocorre pois, para sabermos quais elementos são mais interessantes de se inserir na prateleira, precisamos saber se a sequência posterior a sua inserção é maior que as outras possíveis. Ou seja, se quisermos inserir um elemento i+1. ao iniciarmos pelo elemento i, precisamos saber se ele permite a inserção de outros que virão posteriormente e sendo possível, se é a melhor solução.



Portanto, é claro que, ao calcular para i, calculamos para todos os seus posteriores, assim é interessante armazenar esses dados para que futuramente, ao tentar calcular esse valor, já termos em mãos e gastarmos menos tempo para obtê-lo.

Ao lado podemos verificar tal comportamento, ao calcularmos os valores 2,3,4 e 5 na sequência iniciada em 1 (reta vermelha) podemos utilizá-los nas sequências seguintes (azul e verde), já que seus dados já foram calculados.

3. Análise de complexidade

O programa possui duas funções relevantes para resolução do problema, sendo o custo de tempo ditado por elas. Portanto, analisaremos o custo dela e de suas chamadas sucessivas.

Função LIS - complexidade de tempo - essa função encontra a maior sequência crescente do array iniciando em uma posição i. Dessa forma, tratando-se de uma análise sequencial dos elementos posteriores do vetor, existem 2 cenários: O(1) se é o último elemento do vetor ou se possui apenas um elemento e O(n), caso seja um elemento inicial/intermediário de um vetor com n posições. Portanto, temos uma função com ordem de complexidade que possui melhor e pior caso, sendo eles, respectivamente O(1) e O(n).

Função LDS - complexidade de tempo - essa função encontra a maior sequência decrescente do array iniciando em uma posição i. Dessa forma, tratando-se de uma análise sequencial dos elementos posteriores do vetor, existem 2 cenários: O(1) se é o último elemento do vetor ou se possui apenas um elemento e O(n), caso seja um elemento inicial/intermediário de um vetor com n posições. Portanto, temos uma função com ordem de complexidade que possui melhor e pior, sendo eles, respectivamente O(1) e O(n).

Função main - complexidade de tempo - a função main é incluída nessa análise pois ele realiza a chamada de ambas funções anteriores repetidas vezes. De forma mais exata, como precisamos analisar as maiores sequências para todos os elementos do vetor, precisamos realizar tal operação n vezes, logo, o custo total do programa é dado por n chamadas das funções LIS E LDS, como ambas possuem custo O(n), no pior caso e O(1), temos diferentes situações. Portanto, temos uma função com ordem de complexidade O(1) se o vetor possuir somente um elemento e $O(n^2)$ caso possua n elementos.

4. Conclusão

Ao fim do problema conclui-se que é possível modelar esse problema como encontrar o elemento com maior sequência crescente e decrescente posterior, pois precisa-se definir a melhor organização de rolos de tecidos na prateleira, assim obtém-se uma modelagem de resolução fácil no qual é plausível utilizar o algoritmo de programação dinâmica, que realiza o cálculo dividindo o vetor inicial em subproblemas relacionados e sobrepostos menores para obtenção de uma solução ótima posteriormente. O algoritmo foi utilizado pois, além de utilizar a programação dinâmica, possui custo $O(n^2)$ de tempo com a entrada.