DOCUMENTAÇÃO TP3

1. Identificação

Aluno: Henrique Daniel de Sousa

Matrícula: 2021031912

2. Introdução

Ao saber o preço de uma série de itens, um vendedor deseja ordená-los de forma decrescente em relação ao preço, em uma prateleira. Porém, o vendedor, a partir de um elemento inicial, só pode colocar itens na prateleira na esquerda, se o novo item tiver um preço maior, ou na direita, se tiver um preço menor.

Assim, uma possível questão seria: qual o primeiro elemento a ser inserido?

De fato, esse elemento deve maximizar a quantidade de itens a serem adicionados, tendo em vista toda lista de preços. Ele, então, seria o elemento o qual é possível criar a maior sequência crescente e a maior sequência decrescente, a partir dele.

3. Modelagem

Para isso podemos utilizar o algoritmo Longest Increasing Subsequence (LIS) para encontrar o ponto onde é possível uma maior sequência crescente e o algoritmo Longest Decreasing Subsequence (LDS) para encontrar o elemento para o qual é possível uma maior sequência decrescente.

- **LIS:**

Sabemos que a última posição nunca terá um elemento que a sucede. Assim, sua subsequência sempre terá tamanho 1. Podemos utilizar isso para aplicar uma estratégia Bottom-Up.

Seja OPT(i) = tamanho da maior subsequência crescente, a partir de i

E p(i) = preço de i

Algoritmo:

LIS(p, n):

OPT(n-1) = 1

for i = n-1 to 0:

  for j = i+1 to n-1:

    if p(i) < p(j):

      OPT(i) = max{1, OPT(j)}

return OPT

Assim, a equação de Bellman para essa função é:

$$OPT(i) = \max_{\substack{j = i+1,..n-1 \\ p(i)<p(j)}} \{1, 1 + OPT(j)\}$$

Ou seja, o valor de OPT(i) será o máximo entre os valores de {OPT(j), ..., OPT(n-1)}, para j > i, se p(i)<p(j).

O LIS retornará um array com o tamanho da maior subsequência crescente a partir de i, para todo i < n.

A complexidade de tempo para esse algoritmo é $O(n^2)$, o que pode ser visto pelos dois loops aninhados. A complexidade de espaço é $O(n)$, que é o espaço utilizado na parte de memoization.

- LDS:

O LDS funciona da mesma forma que o LIS, porém, utilizando p(i) > p(j) como condição para calcular OPT(i). Portanto, o LDS tem complexidade de tempo $O(n^2)$ e complexidade de espaço $O(n)$. Assim, o LDS retornará um array com o tamanho da maior subsequência decrescente a partir de i, para todo i < n.

- Solução do problema:

O problema pode ser resolvido utilizando os retornos do LIS e do LDS, de modo que:

LIS[i], então, representa o número de rolos que podemos alocar à esquerda de do elemento i, na estante.

LDS[i], por sua vez, representa o número de rolos que podemos alocar à direita de i.

O valor máximo em (LIS+LDS)-1 significa a quantidade máxima de rolos que poderão ser colocados na estante.

Exemplo:

Se recebermos a lista de preços como sendo 6,7,3,5:

LIS = [2, 1, 2, 1]

LDS = [2, 2, 1, 1]

Somando e subtraindo 1, teremos [3, 2, 2, 1]

Assim, o valor máximo desse vetor é 3, que representa a maior quantidade de rolos que o vendedor poderá organizar na estante.

Desse modo, esse problema pode ser resolvido em $O(n^2)$ de complexidade de tempo, e com complexidade de espaço $O(n)$.