

3.6 Maior subsequência crescente

1. Explicação do problema

Este problema busca, a partir da entrada de um vetor de inteiros, retornar a maior subsequência crescente que está contida neste vetor.

2. Exemplos

Por exemplo, para a sequência $\{5,0,4,3,1,2,0\}$, a saída deverá ser 3. Pois, a maior subsequência possível de ser formada é $\{0,1,2\}$.

Também, se recebermos como entrada a sequência $\{8,5,6,1,7,3\}$, nosso retorno deverá ser de 3. Pois, nossa maior subsequência a ser formada é $\{5,6,7\}$.

3. Explicação da solução

Para solucionar este problema, foi utilizada a técnica de programação dinâmica. Foi elaborada a seguinte abordagem, definimos que nosso estado atual é a nossa atual posição no vetor. Ou seja, se o nosso vetor possui 6 posições, podemos afirmar que:

$$MSC(5) = MSC(4) + 1, \text{ se, e somente se, } V[5] > V[4]$$

Sendo assim, podemos dizer que para uma posição i no nosso vetor V , a maior subsequência crescente de i é composta por, a maior subsequência de j , tal que $j < i$, mais 1, se, e somente se, o valor na posição i for maior que o valor na posição j (estamos assumindo que o valor na posição i faz parte da maior subsequência, por isso somamos 1). Desta forma, podemos ter a seguinte equação de recursão:

$$MSC(i) = MSC(j) + 1, \text{ se, e somente se, } V[i] > V[j]$$

Seguindo este raciocínio, chegaremos à conclusão de que nosso caso base é representado quando $i = 0$, pois neste ponto, ele irá apenas formar a maior subsequência consigo próprio, já que não existem valores após esta posição.

No entanto nesta estratégia, podemos observar que são feitas chamadas repetidas de um mesmo subproblema. Desta forma, é possível a utilização de programação dinâmica para otimizar nossa solução.

Nosso caso base sempre estará na posição 0 do nosso vetor V_e . Sendo assim, o *bottom-up* será iniciado a partir da posição 0 de V_e . Iremos criar um vetor auxiliar V_a , populado inicialmente com zeros e do mesmo tamanho que nosso V_e . Iremos varrer V_e , a partir da sua posição 1 até sua

posição final. Para cada posição iremos varrer de 0 até ela, e faremos as seguintes validações

- Se, $V_a[j]$ for maior que o maior valor de 0 até j , definiremos o maior como $V_a[j]$.
- Se, $V_e[j] < V_e[i]$, definiremos $V_a[i]$ como o maior valor encontrado até agora mais um.

Ao final iremos ter, em nosso V_A a contagem da maior subsequência crescente, e é necessário retornamos apenas este maior valor ($\max(V_a)$).

4. Implementação

[https://github.com/duccl/CC5661-](https://github.com/duccl/CC5661-DynamicProgrammingList/tree/master/Maior%20subsequência%20crescente)

[DynamicProgrammingList/tree/master/Maior%20subsequência%20crescente](https://github.com/duccl/CC5661-DynamicProgrammingList/tree/master/Maior%20subsequência%20crescente)

5. Análise Assintótica

```
1. def msc_dp(vetor):  
2.     msc = [0 for k in range(len(vetor))]           n  
3.     msc[0] = 1                                     1  
4.     for i in range(1, len(vetor)):                 n  
5.         max_in_msc = 0                             n-1  
6.         for j in range(0, i):                     H  
7.             if msc[j] > max_in_msc:                H-1  
8.                 max_in_msc = msc[j]                H-1  
9.             if vetor[j] < vetor[i]:                H-1  
10.                 msc[i] = max_in_msc + 1           H-1  
11.     return max(msc)                               n
```

5.1 Determinando o valor de H

- **Melhor Caso**

Para isso ocorrer, nosso vetor deverá ter tamanho 1, pois não entraremos no primeiro laço (linha 4). Desta forma, não executaremos as linhas 5 até 10.

- **Pior Caso**

Para isso ocorrer, o nosso vetor deverá ter um tamanho maior que 1.

Sendo assim, para cada execução do primeiro laço, entraremos no segundo.

Para auxiliar, podemos pensar da seguinte maneira, quando $i = 1$, a linha 6 será executada 2 vezes. Ou seja,

i	j	Total Execuções
1	0, 1	2
2	0, 1, 2	3
3	0, 1, 2, 3	4
4	0, 1, 2, 3, 4	5

Logo,

$$\begin{aligned}
 H &= 2 + 3 + 4 + 5 + \dots + (n + 1) = \sum_{i=2}^n (n + 1) = \sum_{i=2}^n n + \sum_{i=2}^n 1 = \\
 &= \frac{n(n + 1)}{2} + (n + 1 - 1) = \frac{n^2 + 2n}{2}
 \end{aligned}$$

Conclui-se que,

$$\begin{aligned}
 T(n) &= n + 1 + n + n - 1 + \frac{n^2 + 2n}{2} + 4 \left[\left(\frac{n^2 + 2n}{2} \right) - 1 \right] + n \\
 T(n) &= \frac{18n + 5n^2 - 8}{2}
 \end{aligned}$$

Escolhendo o termo de maior grau, podemos afirmar o seguinte,

$$O(n) = n^2$$

Sendo assim, nossa solução é quadrática.