Aula 2 - Programação Dinâmica

Módulo 3 - PrepTech Google

Programação Dinâmica | Problemas Clássicos

Soma Máxima de um Subvetor Contíguo (MaxSum)

Maior subsequência crescente em um vetor - Longest Increasing Subsequence (LIS)

Soma Máxima de um Subvetor Contíguo

Soma Máxima

Conceito

Definição do problema:

Dado um vetor V com N inteiros, descobrir qual a maior soma que consegue ser obtida com um subvetor apenas com elementos contíguos desse vetor

Programação Dinâmica

Prática no LeetCode - https://leetcode.com/problems/maximum-subarray/

53. Maximum Subarray

Solved

Medium | Topics | Companies

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`

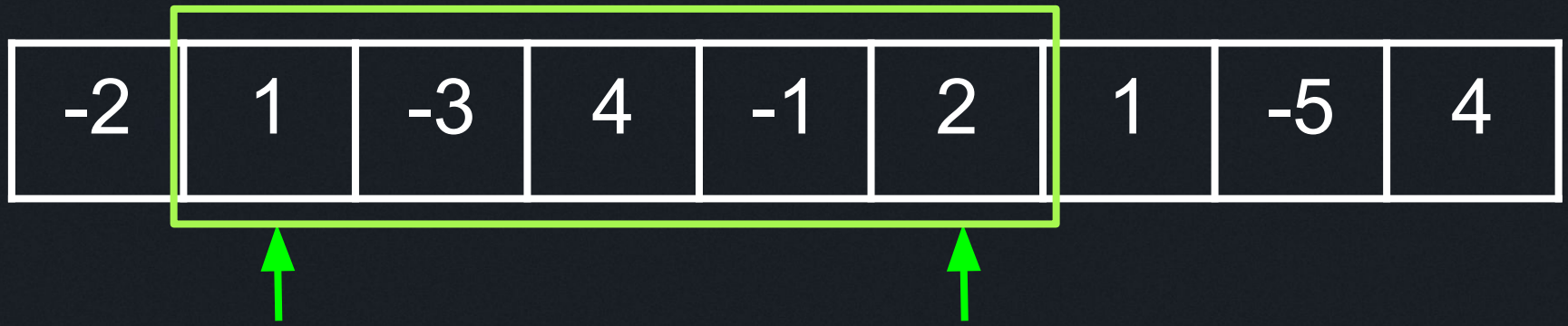
Soma Máxima

Entendendo o problema

Qual a maior soma que pode ser obtida apenas com elementos contíguos desse vetor?

Soma Máxima

Entendendo o problema



Exemplo: No intervalo destacado, dos índices 1 a 5, a soma dos inteiros é 3. Seguramente não é a maior soma, pois o 4 sozinho soma mais.

Soma Máxima

Entendendo o problema

Para esse vetor, a soma máxima contígua é **6**, obtida nesse intervalo.

Soma Máxima

Solução

Pense em uma primeira solução possível de força bruta

Soma Máxima

Solução

Pense em uma primeira solução possível de força bruta

- Se tentarmos todos os possíveis intervalos do vetor, vamos ter dois laços aninhados para todos os possíveis inícios e finais de intervalos
- Para cada início e término, podemos iterar desse índice de início até o de término e somar todos os elementos (um terceiro for aninhado)
- A maior soma que obtivemos é nossa solução

Solução O(n^3)

```python
def maxSubArray(self, nums: List[int]) -> int:
    n = len(nums)
    maxSum = nums[0]
    for start in range(n):
        for end in range(start, n):
            intervalSum = 0
            for i in range(start, end+1):
                intervalSum += nums[i]
            if intervalSum>maxSum:
                maxSum = intervalSum
    return maxSum
```

Soma Máxima

Solução

Seguramente um O(n^3) não vai passar pelo TLE do LeetCode. 1 ⩽ nums.length ⩽ 10^5.

Será que precisamos MESMO do for mais interno? A variável *intervalSum* poderia acumular a soma que estivermos obtendo?

Deixe esse código O(n^2)!

Solução O(n^2)

```
def maxSubArray(self, nums: List[int]) -> int:
    n = len(nums)
    maxSum = nums[0]
    for start in range(n):
        intervalSum = 0
        for end in range(start, n):
            intervalSum += nums[end]
            if intervalSum>maxSum:
                maxSum = intervalSum
    return maxSum
```

Soma Máxima

Solução

Um O(n^2) também não vai passar pelo TLE do LeetCode. 1 ⩽ nums.length ⩽ 10^5.

Vamos aprender o Algoritmo de Kadane.

Soma Máxima

Kadane

- Vamos fazer memoization com duas variáveis apenas:
  - **max_atual**: Representa a maior soma de um subvetor que termina no elemento atual. Inicia com zero.
  - **max_global**: Representa a maior soma encontrada até o momento em qualquer subvetor. Inicia com nums[0], que é o intervalo contendo apenas o primeiro elemento do vetor.

Soma Máxima

Kadane


```
FUNÇÃO Kadane(array)
    inicializar max_atual como array[0]
    inicializar max_global como array[0]

    PARA i DE 1 ATÉ tamanho(array) - 1 FAÇA
        max_atual = MÁXIMO(array[i], max_atual + array[i])

        SE max_atual > max_global ENTÃO
            max_global = max_atual
        FIM SE
    FIM PARA

    RETORNAR max_global
FIM FUNÇÃO
```

ada

Algoritmo de Kadane

```java
class Solution {
    public int maxSubArray(int[] nums) {
        int maxAtual = nums[0];
        int maxGlobal = nums[0];

        for (int i = 1; i < nums.length; i++) {
            // Atualiza maxAtual
            maxAtual = Math.max(nums[i], maxAtual + nums[i]);

            // Atualiza maxGlobal se maxAtual é maior
            if (maxAtual > maxGlobal) {
                maxGlobal = maxAtual;
            }
        }
        return maxGlobal;
    }
}
```

Por que o algoritmo de Kadane é usado como exemplo para ilustrar a Programação Dinâmica?

1. Subproblemas Ótimos

- O algoritmo resolve o problema original (encontrar a soma máxima de uma submatriz) dividindo-o em subproblemas menores. A soma máxima de uma submatriz que termina em um índice específico pode ser obtida usando a solução da soma máxima de submatrizes que terminam nos índices anteriores.

2. Sobreposição de Subproblemas

- O algoritmo aproveita resultados de subproblemas já resolvidos. Por exemplo, ao calcular a soma máxima até o índice atual, ele usa a soma máxima até o índice anterior. Essa característica é fundamental em programação dinâmica, onde soluções para subproblemas são reutilizadas para resolver o problema maior.

3. Decisão Ótima

- Kadane faz uma escolha local em cada etapa (decidir se deve adicionar o elemento atual à soma anterior ou começar uma nova submatriz), que se revela uma decisão ótima para a solução global. Isso ilustra como decisões locais podem levar a uma solução global ótima.

4. Complexidade de Tempo Eficiente

- O algoritmo é muito eficiente, com complexidade de tempo O(n), o que demonstra como técnicas de programação dinâmica podem reduzir o tempo de execução em comparação com métodos de força bruta, que poderiam ter uma complexidade de $O(n^2)$ ou maior.

Longest Increasing Subsequence - LIS

Longest Increasing Subsequence

Conceito

Definição do problema:

Qual a subsequência mais longa estritamente crescente em um vetor com N inteiros?

IMPORTANTE: Uma subsequência não precisa ser contígua, ou seja, os elementos não precisam estar juntos no vetor original, mas devem manter a ordem relativa.

Programação Dinâmica

Prática no LeetCode - https://leetcode.com/problems/longest-increasing-subsequence/

300. Longest Increasing Subsequence

Solved

Medium | Topics | Companies

Given an integer array `nums`, return *the length of the longest* ***strictly increasing*** ***subsequence***.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

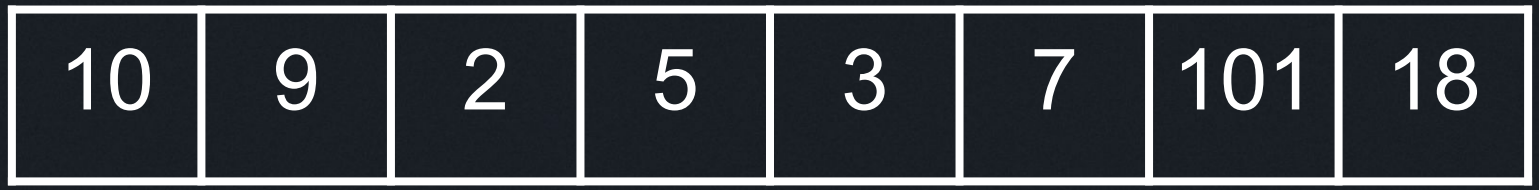
Constraints:

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

ada

Longest Increasing Subsequence

Entendendo o problema



Qual o tamanho da maior subsequência estritamente crescente desse vetor?

Longest Increasing Subsequence

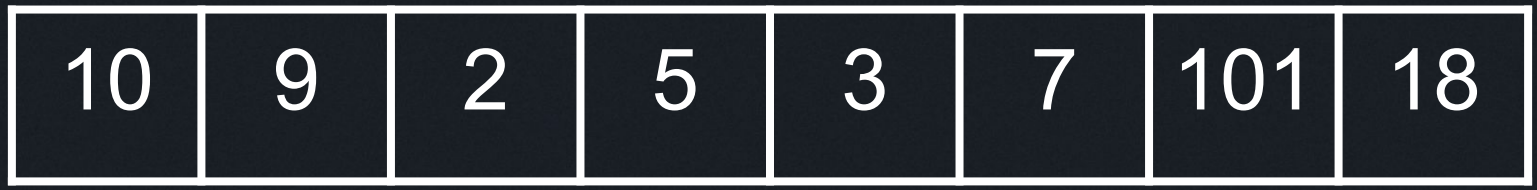
Exemplos de subsequencias estritamente crescentes em verde

10	9	2	5	3	7	101	18
10	9	2	5	3	7	101	18
10	9	2	5	3	7	101	18
10	9	2	5	3	7	101	18

Todas em verde são estritamente crescentes. A maior é a última, de tamanho 4. Há outras de tamanho 4.

Longest Increasing Subsequence

Entendendo o problema

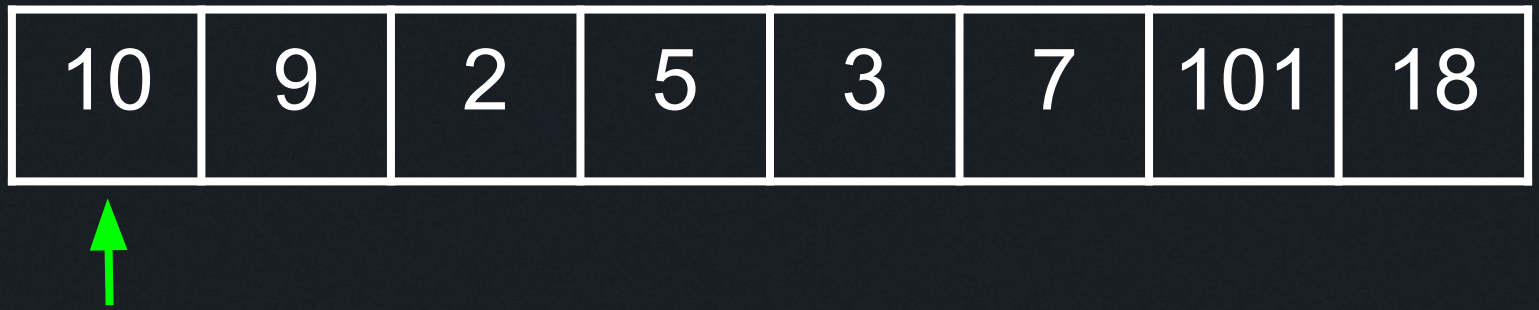


Podemos pensar que, para cada elemento do vetor, se tentarmos formar sequência com todos os elementos na direita que sejam maiores, encontraremos todas as subsequências estritamente crescentes.

Podemos retornar o tamanho da maior.

Longest Increasing Subsequence

Entendendo o problema



Quem poderia, nesse vetor, vir depois desse 10? Apenas 101 e 18, porque só eles são maiores que 10.

Se tivermos uma recursão, calculando o maior tamanho contendo cada índice, podemos fazer isso por **força bruta**.

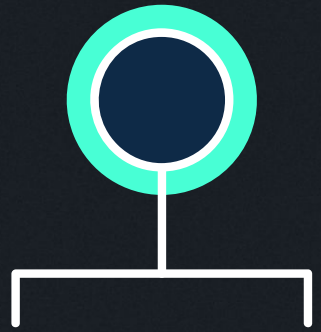
Longest Increasing Subsequence

Solução

Força Bruta

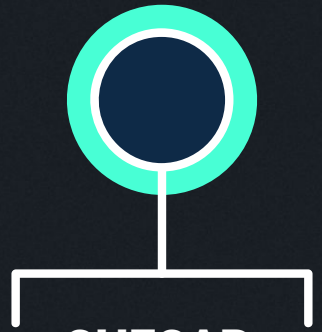
```
def LISWithStartingIndex(self, nums: List[int], index: int) -> int:
    maxSize = 1 #initialize with size 1, that it is just nums[index]
    for i in range(index+1, len(nums)):
        if nums[i]>nums[index]:
            size = 1 + self.LISWithStartingIndex(nums, i)
            #this subsequence starts with num[index] + lis of index i
            maxSize = max(maxSize, size) #keeping the longest
    return maxSize

def lengthOfLIS(self, nums: List[int]) -> int:
    maxSize = 0
    for i in range( len(nums) ): #trying to start in all indexes
        size = self.LISWithStartingIndex(nums, i)
        maxSize = max(maxSize, size) #keeping the longest
    return maxSize
```

Os Três Lembretes de Programação Dinâmica



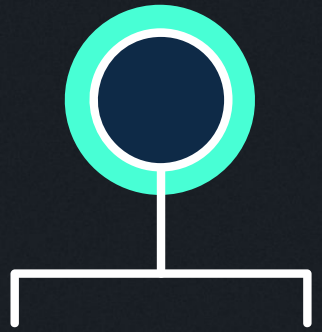
MEMOIZATION

Inicializar uma memória auxiliar para salvar resultados calculados



CHECAR MEMOIZATION

Antes de calcular, consultar no memo se já o fez **(como um cache)**



SALVAR NA MEMOIZATION

Ao final do cálculo, salvar o resultado no memo

Longest Increasing Subsequence

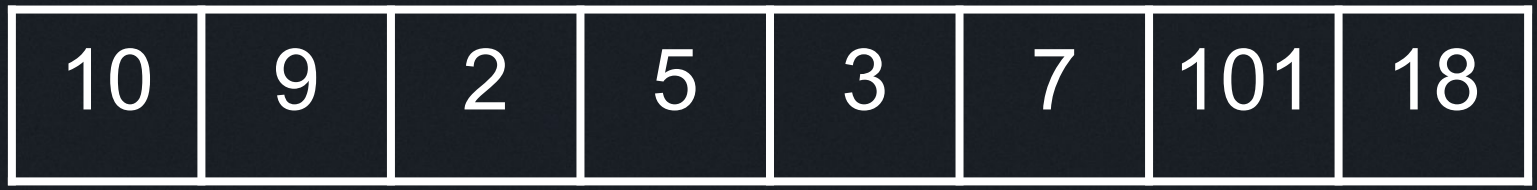
Solução

```
def LISWithStartingIndex(self, nums: List[int], index: int, memo) -> int:
    if index in memo: #Step 2 of PD Searching in memo
        return memo[index]
    maxSize = 1 #starts with nums[index], size 1
    for i in range(index+1, len(nums)):
        if nums[i]>nums[index]:
            size = self.LISWithStartingIndex(nums, i, memo) + 1
            maxSize = max(maxSize, size)
    memo[index] = maxSize #Step 3 of PD: Saving in memo
    return maxSize

def lengthOfLIS(self, nums: List[int]) -> int:
    memo = dict() #Step 1 of PD: Creating memo
    maxSize = 0
    for i in range( len(nums) ):
        size = self.LISWithStartingIndex(nums, i, memo)
        maxSize = max(maxSize, size)
    return maxSize
```

ada

Longest Increasing Subsequence

Entendendo o problema



Conseguimos pensar em uma solução iterativa e bottom-up?

DICA: e se viermos preenchendo de trás para frente, usando um memo do tamanho do vetor nums?

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

O memo[i] representa o maior LIS que começa obrigatoriamente com o i-ésimo elemento

O memo tem o tamanho do vetor:

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

Vamos preencher o memo de trás para frente, iterando sobre o vetor nums. Para a última posição, é direto, o maior lis do último é de tamanho 1

							1

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

O memo do penúltimo pode ser iniciado com 1.

Se o último fosse maior que o penúltimo, teríamos lis com os dois.

						1	1

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

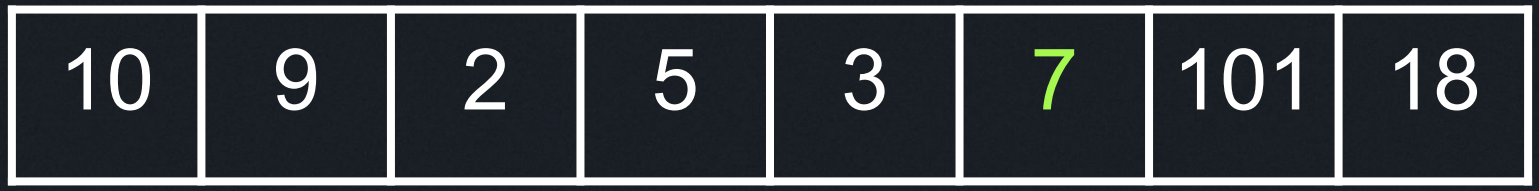
Podemos generalizar:

Para cada elemento, inicia o memo com 1.
Itera sobre todos os elementos maiores que ele na sua direita, e fica com o maior +1.

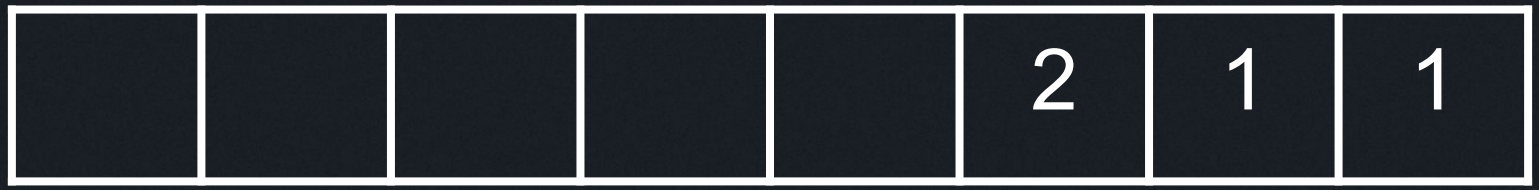
						1	1

Longest Increasing Subsequence

Entendendo o problema



101 e 18 são maiores que 7
O maior memo entre eles é 1, por isso o memo do índice vai ficar com 2 (1+1)

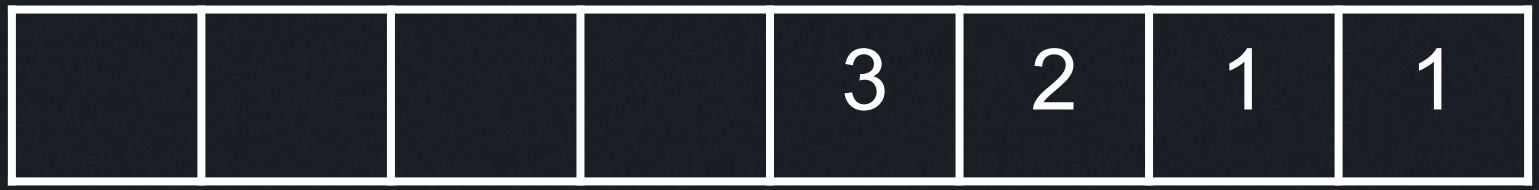


ada

Longest Increasing Subsequence

Entendendo o problema

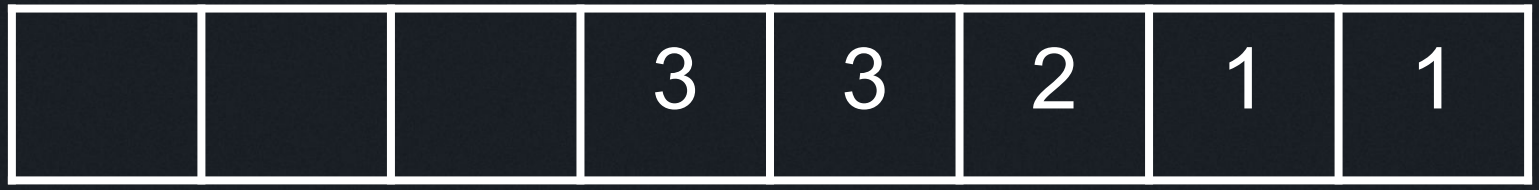
7, 101 e 18 são maiores que 3
O maior memo entre eles é 2, por isso o memo do índice vai ficar com 3 (2+1)



Longest Increasing Subsequence

Entendendo o problema

7, 101 e 18 são maiores que 5
O maior memo entre eles é 2, por isso o memo do índice vai ficar com 3 (2+1)



ada

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

Todos na direita são maiores que 2
O maior memo entre eles é 3, por isso o memo do índice vai ficar com 4 (3+1)

		4	3	3	2	1	1

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

101 e 18 são maiores que 9
O maior memo entre eles é 1, por isso o memo do índice vai ficar com 2 (1+1)

	2	4	3	3	2	1	1

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

101 e 18 são maiores que 10
O maior memo entre eles é 1, por isso o memo do índice vai ficar com 2 (1+1)

2	2	4	3	3	2	1	1

Longest Increasing Subsequence

Entendendo o problema

10	9	2	5	3	7	101	18

Terminada a iteração, o maior valor no memo é o tamanho da maior subsequência estritamente crescente

2	2	4	3	3	2	1	1

Longest Increasing Subsequence

Solução

```
def lengthOfLIS(self, arr: List[int]) -> int:
    memo = [-1]*len(arr)#memo of size len(arr) with -1's
    memo[len(arr)-1] = 1 #lis in last position = 1
    for i in range(len(arr)-2, -1, -1): #from end to 0
        maxSequence = 0
        for j in range(i+1, len(arr)):
            if arr[j]>arr[i]:
                maxSequence = max(maxSequence, memo[j])
        memo[i] = 1+maxSequence
    return max(memo) #return max value from memo
```

Obrigada