

# Sublista contígua de soma máxima

## Explicação do problema - Sublista contígua de soma máxima

Falando um pouco sobre a história desse problema, temos, tipicamente, uma lista de números a ser computada como entrada que pode ter números positivos e negativos.

O problema da soma máxima a partir de uma lista de números teve origem numa versão bidimensional mais complexa de um problema de emparelhamento de padrões inicialmente apresentado por Ulf Grenander, da Brown University.

Tipicamente, a lista de números a ser computada como entrada possui :

- **números positivos e negativos.** Dessa forma, ao encontrar um número negativo vizinho a uma sublista de maior soma computada até então, uma das dificuldades do problema no decorrer da sua resolução é avaliar se esse número negativo deve ser acrescentado à sublista ou não.
- Para uma lista que contenha apenas números positivos, o resultado de maior soma será simplesmente a lista inteira dada como entrada.

Para esse problema, temos 4 algoritmos, de diferentes complexidades:

- Algoritmo cúbico  $\rightarrow O(n^3)$
- Algoritmo Quadrático  $\rightarrow O(n^2)$
- Algoritmo por Divisão e conquista  $\rightarrow O(n \log n)$
- **Kadane's Algoritmo  $\rightarrow O(n)$**

Nesse trabalho iremos discutir e explicar o algoritmo de Kadane's, algoritmo criado por Jay Kadane, é refutado como a solução mais eficiente e a melhor solução possível para esse problema, pois qualquer algoritmo que pretenda resolver o problema em estudo deve necessariamente percorrer os  $N$  elementos da lista dada como entrada.

## Explicação da solução dinâmica - Passo a Passo

Primeiramente o algoritmo começa varrendo o vetor até encontrar o primeiro numero positivo, logo após encontrar o valor positivo, esse valor será guardado na variável **MAX\_ATUAL**.

Após isso, ele compara **MAX\_ATUAL** com **MAX\_TOTAL**, se **MAX\_ATUAL** for maior que **MAX\_TOTAL**, ele atribui o valor do Atual ao Total.

A ideia do algoritmo é percorrer o vetor fazendo essas atribuições e somando o atual com os próximos valores seguintes do vetor, até o **MAX\_ATUAL** ser menor que **MAX\_TOTAL**, assim, parando o algoritmo, e produzindo a sublista contígua de soma máxima.  $O(n)$ .

## Exemplo de Funcionamento

**Vetor :**

-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual** =  $0 + (-2) = -2$ ... Por o valor ser negativo, ainda não inicia o algoritmo.

**Max\_Total** = 0.

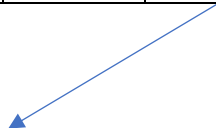
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual** =  $0 + (-3) = -3$ ... Por o valor ser negativo, ainda não inicia o algoritmo.

**Max\_Total** = 0

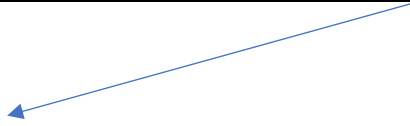
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual** =  $0 + (4) = 4$ ... Por o valor ser positivo, aqui começa o algoritmo, guardando o 4 no **Max\_Atual** e Atualizando o **Max\_Total**.

**Max\_Total = 4.**

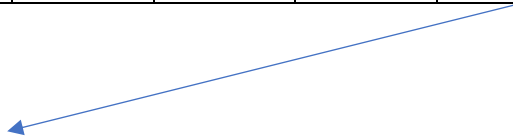
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual =  $4 + (-1) = 3$**  → Depois de fazer a soma, ele **verifica** se **Max\_Atual > Max\_Total**, caso seja, ele atribui o valor do **Atual** ao **Total**... Caso contrario, segue o algoritmo.

**Max\_Total = 4**

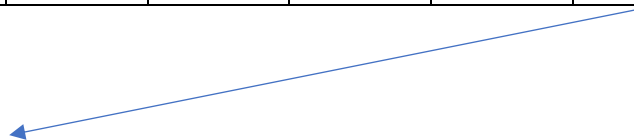
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual =  $3 + (-2) = 1$**

**Max\_Total = 4.**

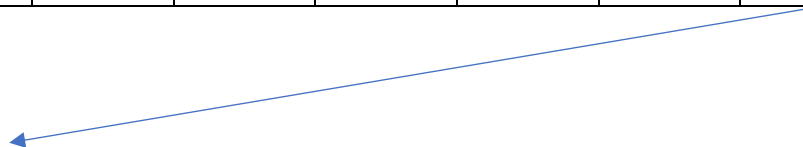
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual =  $1 + (1) = 2$**

**Max\_Total = 4.**

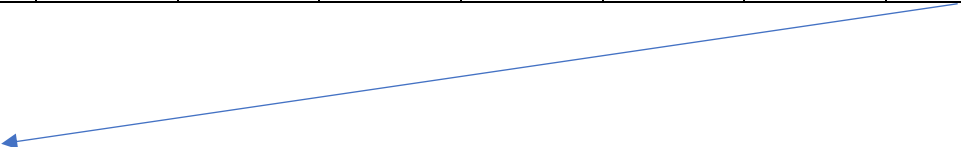
-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual =  $2 + (5) = 7$**  → Aqui **Max\_Atual > Max\_Total**, portanto, ocorre a copia do valor de **Max\_Atual = Max\_Total**.

**Max\_Total = 7.**

-2	-3	4	-1	-2	1	5	-3
----	----	---	----	----	---	---	----



**Max\_atual** =  $7 + (-3) = 4 \rightarrow \text{Max\_Atual} < \text{Max\_Total}$ , portanto, achamos a soma da sublista.

**Max\_Total** = 7.

## Implementação

Implementamos o algoritmo dinâmico em c++, utilizando a biblioteca vector.

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  void kadane(vector<int> &vet){
7
8      int max_atual = 0, max_total = -1;
9      int size_vet = vet.size();
10
11     for(int i = 0; i < size_vet; i++)
12     {
13         max_atual = max_atual + vet[i];
14
15         if(max_atual < 0)
16             max_atual = 0;
17         if(max_atual > max_total)
18             max_total = max_atual;
19     }
20
21     cout << "Soma maxima: " << max_total << endl;
22 }
23
24
25 }
```

Nesse exemplo a soma se inicia no 20, pois  $10 + 5 = 15 + (-17) = -2 > 0$ , então não começa o algoritmo.

```
int main(int argc, char** argv) {
    vector<int> vet(9);

    vet[0] = 10;
    vet[1] = 5;
    vet[2] = -17;
    vet[3] = 20;
    vet[4] = 50;
    vet[5] = -1;
    vet[6] = 3;
    vet[7] = -30;
    vet[8] = 10;

    kadane(vet);

    return 0;
}
```

Nesse exemplo, a soma máxima do vetor deverá ser 72

```
C:\Users\Victor\Documents\kadanesteste.exe
Soma maxima: 72
-----
Process exited after 0.236 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

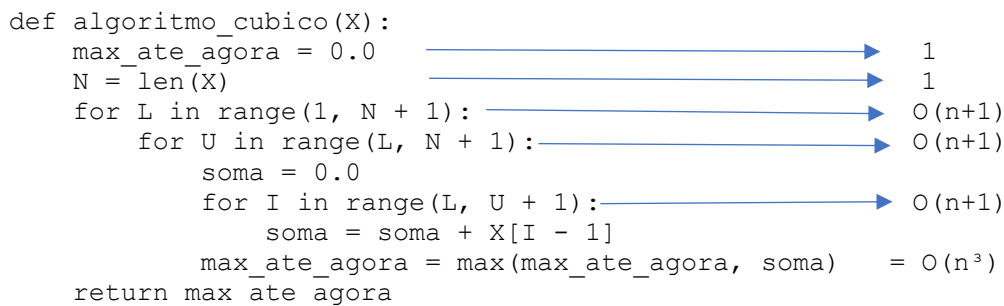
## Curiosidades

O algoritmo de Kadane's possui algumas variações, dentre elas temos a cúbica e a quadrática.

### Cúbica:

Utilizando a notação Big-Oh para a análise do tempo de execução do algoritmo, tem-se que cada loop (FOR) do mesmo tem custo  $O(N)$ , totalizando, assim, um tempo de execução de  $O(N^3)$  nesse primeiro algoritmo.

```
def algoritmo_cubico(X):  
    max_ate_agora = 0.0  
    N = len(X)  
    for L in range(1, N + 1):  
        for U in range(L, N + 1):  
            soma = 0.0  
            for I in range(L, U + 1):  
                soma = soma + X[I - 1]  
            max_ate_agora = max(max_ate_agora, soma)  
    return max_ate_agora
```

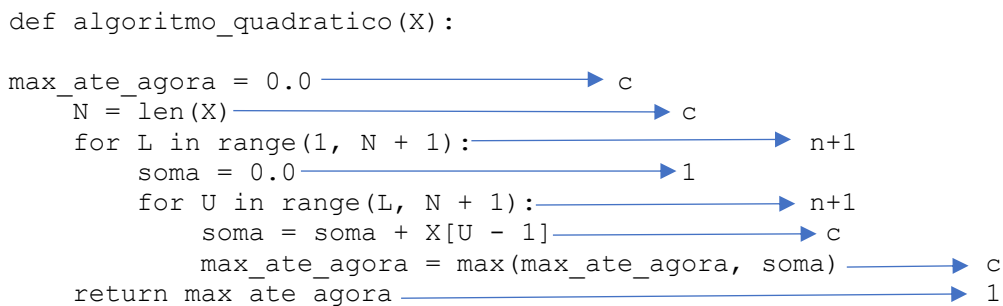


### Quadrático:


O algoritmo quadrático utiliza a técnica de guardar os resultados intermediários, para assim, evitar a recomputação. Basicamente o que ele faz é se beneficiar com o fato de que duas somas parciais vizinhas computadas ao longo da execução são diferentes por apenas **um elemento**, portanto, podemos reaproveitar os cálculos efetuados num determinado passo do algoritmo no passo seguinte.

Analisando o tempo de execução do algoritmo, temos que o primeiro loop é executado  $N$  vezes e o segundo também, totalizando um tempo de execução de  $O(N^2)$

```
def algoritmo_quadratico(X):  
    max_ate_agora = 0.0  
    N = len(X)  
    for L in range(1, N + 1):  
        soma = 0.0  
        for U in range(L, N + 1):  
            soma = soma + X[U - 1]  
            max_ate_agora = max(max_ate_agora, soma)  
    return max_ate_agora
```




## Analise Assintótica


Para  $i = 0$  até  $i < \text{Tam\_vet}$   N-1

{

$\text{Max\_Atual} = \text{Max\_Atual} + \text{vet}[i]$

Se  $\text{Max\_Atual} < 0$   c

$\text{Max\_Atual} = 0$

Se  $\text{Max\_Atual} > \text{Max\_total}$   c

$\text{Max\_total} = \text{Max\_Atual}$

}