Estruturas de Dados

Análise de Complexidade

2025/2026

Revisão

- Notação O-grande:
  - Utilizada para indicar o tipo de variação de uma função:
    - O(N)- Linear
    - O(N^2) – Quadrática
    - ...
  - Útil para analisar e comparar o comportamento de algoritmos sem a necessidade de calcular precisamente o tempo de execução.

O-Grande:Formalização

- **(O-Grande)** *T(N)* é *O(F(N))* se existem constantes positivas *c* e *N0* tal que *T(N)≤cF(N)* para *N≥N0*.

- Exemplo: $N^2 + N$ é $O(N^2)$?

Exemplo (1)

- $N^2 + N$ é $O(N^2)$?

Neste caso,

$T(N) = N^2+N$

$F(N) = N^2$

É preciso demonstrar que existem dois valores positivos N_0 e c tal que:

$N^2+N \leq cN^2$, *para* $N \geq N_0$

Se conseguirmos encontrar dois valores positivos N_0 e c que cumpram a expressão anterior, fica demonstrado que $N^2 + N$

Exemplo (2)

- $N^2 + N$ é $O(N^2)$?

A estratégia passa por fixar um dos dois valores (c ou N_0) e subsequentemente tentar encontrar um valor para o outro parâmetro que assegure:

$N^2+N \leq cN^2$, para $N \geq N_0$

Por exemplo, fazendo c=1 e substituindo, verificamos que a expresão reduz-se a:

$N^2+N \leq N^2$, para $N \geq N_0$

$N+1 \leq N$, para $N \geq N_0$

$1 \leq 0$, para $N \geq N_0$, o que é claramente falso (impossível).

Isto significa que a escolha c=1 não é adequada. Mas pode ser que funcione com outros valores de c.

Exemplo (4)

- $N^2 + N$ é $O(N^2)$?

Tentando agora com o valor c=3 e substituindo, verificamos que a expresão reduz-se a:

$N^2+N \leq 3N^2$, para $N \geq N_0$

$-2N^2+N \leq 0$, para $N \geq N_0$

$N(-2N+1) \leq 0$, para $N \geq N_0$

Como N >0, então

$-2N+1 \leq 0$, para $N \geq N_0$

$N \geq \frac{1}{2}$, para $N \geq N_0$. Esta expressão é sempre válida se N0 for igual ou superior a ½. Por isso, N_0 pode ter, por exemplo, o valor 0,9.

O-Grande:Formalização

- **(Omega-Grande)** *T(N)* é *Ω(F(N))* se existem constantes positivas *c* e *N0* tal que *T(N)* $\geq cF(N)$ para $N \geq N0$.

- Exemplo: $N \log N$ é *Ω(N)* ?

O-Grande:Formalização

- **(Teta-Grande)** *T(N)* é *Θ(F(N))* se *T(N)* é *Ω(F(N))* e *T(N)* é *O(F(N))*.

- Exemplo: $\dfrac{N^2}{N-10}$ é *Θ(N)* ?

O-Grande:Formalização

- **(O-Pequeno)** *T(N)* é *o(F(N))* se *T(N)* é *O(F(N))* e *T(N) não é Θ(F(N))*.

- Exemplo: $N^2 \log N$ é *o*(N^3) ?

Exercicio

- Considere que um algoritmo tem complexidade de ordem $O(N^2)$. Quanto é que o tempo de execução aumenta se o tamanho da entrada aumentar 10 vezes?
- Sabendo que $T(N)=cN^2$ então $T(10N)=100cN^2=100\ T(N)$
  - E para $O(N)$?
  - E para $O(N^3)$?
  - E para $O(N \log N)$?

Logaritmos

- O logaritmo (base 2) indica
  - O número de dígitos (bits) de um número
  - O número de vezes que 1 deve ser duplicado para atingir N
  - O número de vezes que N deve ser dividido para atingir 1.
    - Se um algoritmo demora O(1) a reduzir a dimensão de um problema numa determinada fracção (não necessariamente 50%), é O(log N)
      - Por exemplo, um algoritmo que em cada passo elimina 10% dos dados analisados até restar apenas um valor.

$$\boxed{O(\log_B N) = O(\log N)}$$

porquê?

Logaritmos

Como $\log_B X = \log_a X / \log_a b$

$$O(\log_B N) = O(1/\underbrace{\log B}_{\text{constante}} * \log N) = O(\log N)$$

Pesquisa

- Dado um número X e um array A, devolver a posição de X em A ou uma indicação que X não existe.
  – Pesquisa Sequencial
  – Pesquisa Binária
  – Pesquisa Interpolada

Pesquisa Sequencial

- Qual é a complexidade:
  - De uma pesquisa sem sucesso?
  - De uma pesquisa com sucesso, no pior caso?
  - De uma pesquisa com sucesso, no caso médio?

Pesquisa Binária

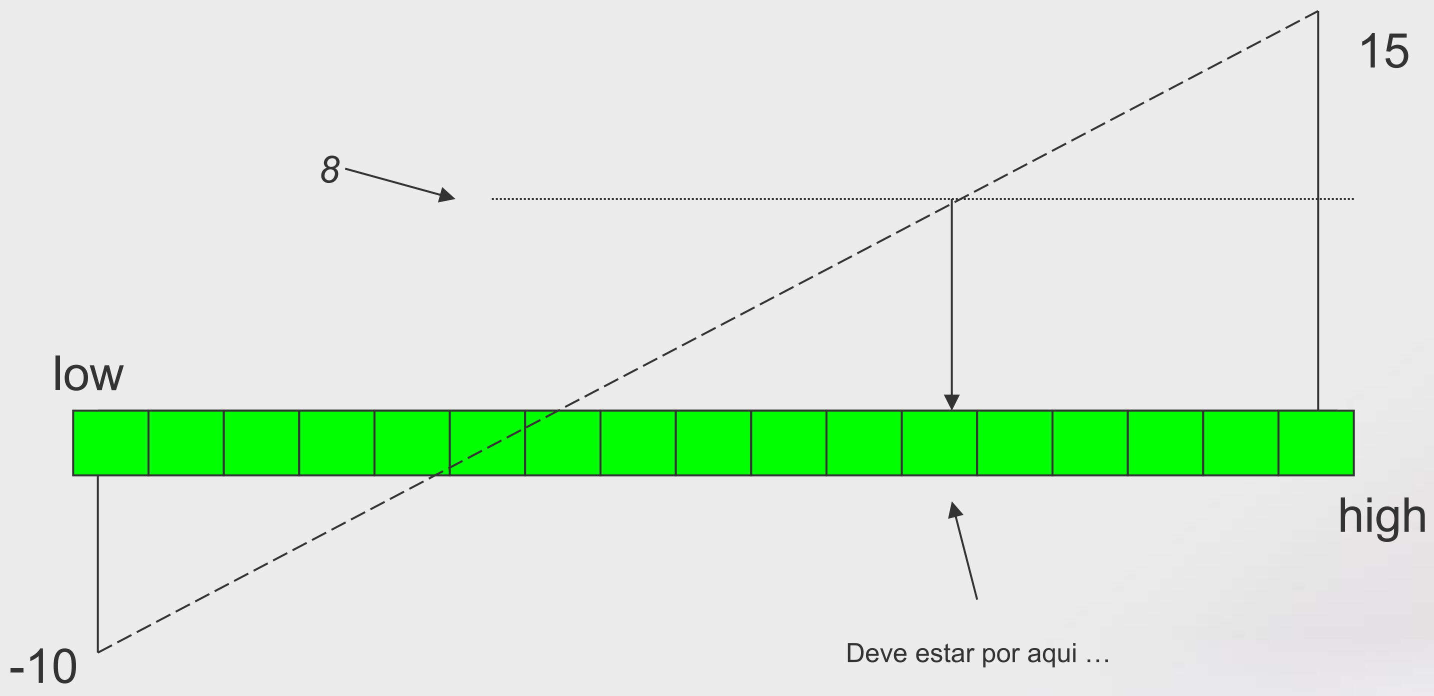
- Se o espaço de pesquisa está ordenado, procura-se o valor no ponto médio do array, e não no início.
  - Pode-se reduzir para metade o número de valores pesquisados em cada iteração.
    - E se procurassemos o ponto a ¾ do array, em vez de examinarmos o ponto médio?
- Qual a complexidade de uma pesquisa falhada?
- E de uma pesquisa pesquisa com sucesso, no pior caso?
- E de uma pesquisa com sucesso, no caso médio?

Pesquisa Interpolada

- Se o array estiver ordenado E a distribuição de números for uniforme…
  - Só é normalmente vantajoso se o acesso aos dados for muito custoso.
- Baseia-se em interpolar a posição do elemento procurado a partir dos valores dos maior e menor elemento.

Pesquisa Interpolada

- Procurar o número 8



Pesquisa Interpolada

- Qual é o pior caso?
  - Caso os valores não estejam uniformemente distribuidos, todos os elementos podem vir a ser pesquisados…
    - O(N)
- Qual é o caso médio?
  - Caso a distribuição seja relativamente uniforme:
    - O(log log N)

Análise Comparativa

- Vamos estudar um problema, para o qual muitos algoritmos diferentes existem. Vamos analisar:
  - Um algoritmo exaustivo, simples mas ineficiente.
  - Uma melhoria simples do algoritmo anterior.
  - Um algoritmo eficiente, mas menos óbvio.

Máxima Sequência Contígua

- Dado um conjunto de números inteiros (possivelmente negativos), determinar qual é a sequência contígua (inc. conjunto vazio) com a maior soma.
- Exemplo:
  - MSC({2,3,4,-2,1})=9
  - MSC({-2,1})=1
  - MSC({-1,-2,-3})=0
  - MSC({})=0

MSC – Versão 1 (Procura Exaustiva)

```
public static int maxSeqCont(int [] m)
{
int maxSoma=0;
Int N=m.length;
for(int i=0;i<N:i++)
        for(j=i;j<N;j++){
                int estaSoma=0;
                for(k=i;k<j;k++)
                        estaSoma+=m[k];
                if(estaSoma>maxSoma)
                        maxSoma=estaSoma;
        }
return maxSoma;
}
```

Qual é a complexidade deste algoritmo?

MSC – Versão 1

- O factor dominante neste algoritmo é o tempo gasto na linha 9.
- Um cálculo aproximado:
  - O ciclo exterior (i) tem *N* iterações.
  - O ciclo intermédio (j) tem entre *1* e *N-1* iterações
  - O ciclo interior (k) é executado entre *1* e *N-1* vezes.
- Multiplicando o número de iterações dos vários ciclos encadeados, podemos presumir que a complexidade do algoritmo deverá ser $O(N^3)$
  - *Esta hipótese pode ser confirmada através de uma análise mais precisa.*

MSC – Versão 1

- O número de execuções *exacto* da linha 9 é dado por…

$$\sum_{i=1}^{N}\sum_{j=i+1}^{N}\sum_{k=i}^{j}1=\frac{N(N+1)(N+2)}{6}$$

- O que confirma que o algoritmo é efectivamente de ordem cúbica.
  - Por mais pequeno que seja o tempo de execução da respectiva instrução, ele irá dominar o tempo de execução para valores elevados de *N*.

MSC – Versão 2

- Os algoritmos de ordem cúbica são normalmente inaplicáveis em termos práticos.
- Os 3 ciclos encadeados resultam num algoritmo de ordem cúbica…
- Será que é possível remover um dos ciclos?

MSC – Versão 2

- O algoritmo básico efectua muitos cálculos desnecessários:
  - O objectivo do ciclo interior é calcular $\sum_{k=i}^{j} A_k$
  - Sabendo que: $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$
  - Então bastará somar A_j ao resultado da iteração anterior para obter o mesmo resultado.
    - Desta forma será eliminado o ciclo interior.

MSC – Versão 2

```
public static int maxSeqCont(int [] m)
{
int maxSoma=0;
int N=m.length;
for(int i=0;i<N;i++){
    int estaSoma=0;
    for(j=i;j<N;j++){
        estaSoma+=m[j];
        if(estaSoma>maxSoma)
            maxSoma=estaSoma;
        }
  }
return maxSoma;
}
```

Basta somar o novo elemento ao resultado anterior.

MSC – Versão 2

- Qual a complexidade?
  - Foi removido um ciclo, pelo que a complexidade é reduzida para $O(N^2)$
- Cálculo preciso:
  - O número de execuções das linhas do ciclo interior (linhas 8 e 9) é dado por:

$$\sum_{i=1}^{N}\sum_{j=i+1}^{N} 1 = \frac{N(N-1)}{2}$$

MSC – Versão 3

- Se conseguirmos eliminar mais um ciclo, poderemos obter um algoritmo linear…
- Será que é possível?

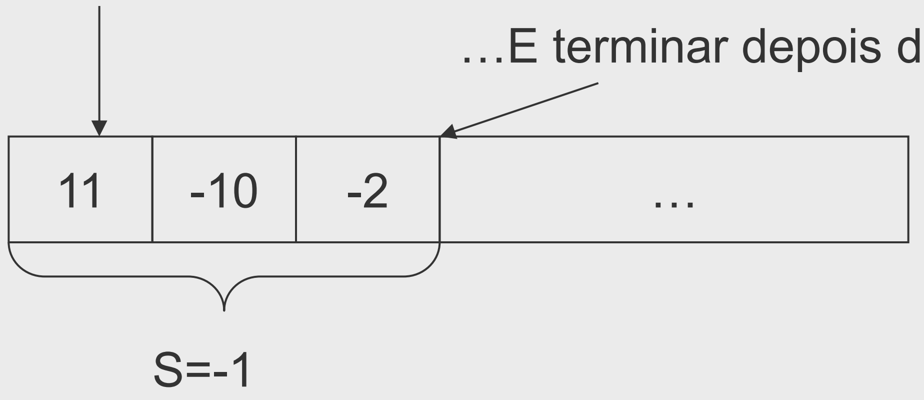
MSC – Versão 3

- Se $A_{i,j}$ for uma sequência com soma $S_{i,j}<0$, então $A_{i,q}$, com $q>j$, não é uma sequência de soma máxima.
  - Porque $S_{j+1,q}$ é garantidamente maior (ou igual) a $S_{i,j}$…
- Sabendo isto, o algoritmo pode ser modificado para ignorar subsequências que incluam subsomas negativas.

- # Exemplo

A sequência máxima nunca pode ter início aqui….

…E terminar depois deste ponto



…porque seria possivel remover esta porção
Para obter uma sequência ainda maior.

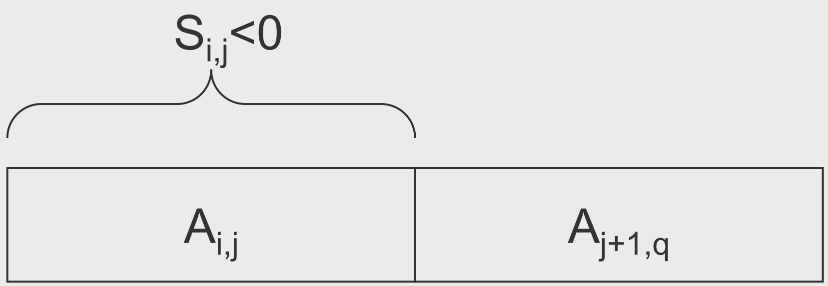
Versão 3.0a

```
public static int maxSeqCont(int [] m)
{
int maxSoma=0;
int N=m.length;
for(int i=0;i<N;i++){
    int estaSoma=0;
    for(j=i;j<N;j++){
            estaSoma+=m[j];
            if(estaSoma<0)
               break;
            if(estaSoma>maxSoma)
               maxSoma=estaSoma;
            }
return maxSoma;
}
```

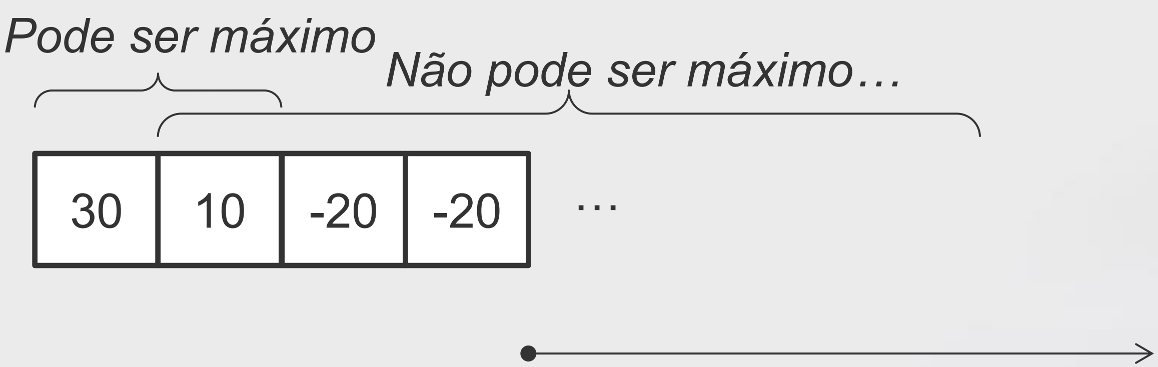
Pode-se ignorar as restantes sequências a partir deste ponto…

Será suficiente?

Versão 3.0b



Se Si,j for a primeira sequência com Si,j<0, então o máximo já foi encontrado ou tem inicio depois de *j*.
Por isso, pode-se procurar um novo máximo a partir de *j*.
:



Uma seq- máxima alternativa só pode ter início depois deste ponto

Versão 3.0b

```
public static int maxSeqCont(int [] m)
{
int maxSoma=0;
int N=m.length;
for(int i=0;i<N;i++){
    int estaSoma=0;
    for(j=i;j<N;j++){
            estaSoma+=m[j];
            if(estaSoma<0){
              i=j;
             break;
              }
            if(estaSoma>maxSoma)
               maxSoma=estaSoma;
            }
  }
return maxSoma;
}
```

Para além de abandonar o ciclo caso encontremos um sequência negativa, podemos também começar a procurar um novo máximo a partir desta posição.

Podemos agora reescrever o código para eliminar um dos ciclos.

Versão 3.0c

Versão Equivalente

```
public static int maxSeqCont(int [] m)
{
int maxSoma=0;
int N=m.length;
int estaSoma=0;
for(int i=0;i<N;i++){
    estaSoma+=m[i];
    if(estaSoma>maxSoma)
        maxSoma=estaSoma;
    if(estaSoma<0)
        estaSoma=0;
    }
return maxSoma;
}
```

Qual é a complexidade?

Máxima Sequência Contígua

- Análisámos 3 algoritmos para o mesmo problema:
  - *Procura Exaustiva – $O(N^3)$ – Versão óbvia e fácil de compreender.*
  - *Melhoria simples – $O(N^2)$ -*
  - *Versão Optimizada – $O(N)$ – Versão linear, mas de compreensão menos imediata.*
- As optimizações têm custos a nível de clareza.
- Por esse motivo, é normalmente necessário demonstrar a correcção dos algoritmos optimizados