


Análise da Complexidade de Algoritmos I

25/09/2024

Sumário

- Análise do desempenho de algoritmos
- Complexidade temporal e espacial
- Análise experimental vs. análise formal da complexidade
- Eficiência relativa
- Exemplos simples
- Exercícios / Tarefas 
- Sugestões de leitura

Algoritmos

O que é um algoritmo?

- Sequência de **instruções não-ambíguas**
- Permitem **atingir um objetivo**
 - Efetuar um cálculo
 - Resolver um problema
 - Executar uma tarefa
- Num **espaço de tempo finito**
- Como estabelecer / definir ?
 - Texto; pseudo-código; **linguagem de programação**

Algoritmos Deterministas

- Um **algoritmo determinista**
 - Devolve sempre **o mesmo resultado**, qualquer que seja o número de vezes que é executado com **os mesmos dados de entrada**.
 - Executa sempre **a mesma sequência de instruções**, quando é executado com **os mesmos dados de entrada**.
- O tipo mais habitual de algoritmo !
- Há uma definição mais formal em termos de máquinas de estado...
- Os algoritmos que vamos analisar e desenvolver !

Algoritmos Não-Deterministas

- Um **algoritmo não-determinista**
 - Pode ter um **comportamento diferente**, para **os mesmos dados de entrada**, em **diferentes execuções**
 - Ao contrário de um algoritmo determinista !
- Habitualmente usados para obter **soluções aproximadas** em alguns tipos de situações
 - Quando é **demasiado dispendioso** determinar **soluções exatas...**
- Exemplos ?

Análise da Complexidade

Como escolher o algoritmo mais apropriado?

- Vários algoritmos para resolver uma instância de um problema
- Qual é o algoritmo mais eficiente / com melhor desempenho ?
- Tempo de execução / N^o de operações executadas
 - Complexidade temporal
- Espaço de memória necessário
 - Complexidade espacial

Como estimar o desempenho?

- Conhecemos o **desempenho** de um algoritmo para uma **instância** de um problema
 - **Tamanho e características da instância**
 - Tempo / N^o de operações
 - Espaço de memória
- Se o **tamanho** da instância se tornar **10 vezes maior**, o que acontece ?
- Se a **organização dos dados for diferente**, o que acontece ?

Análise experimental

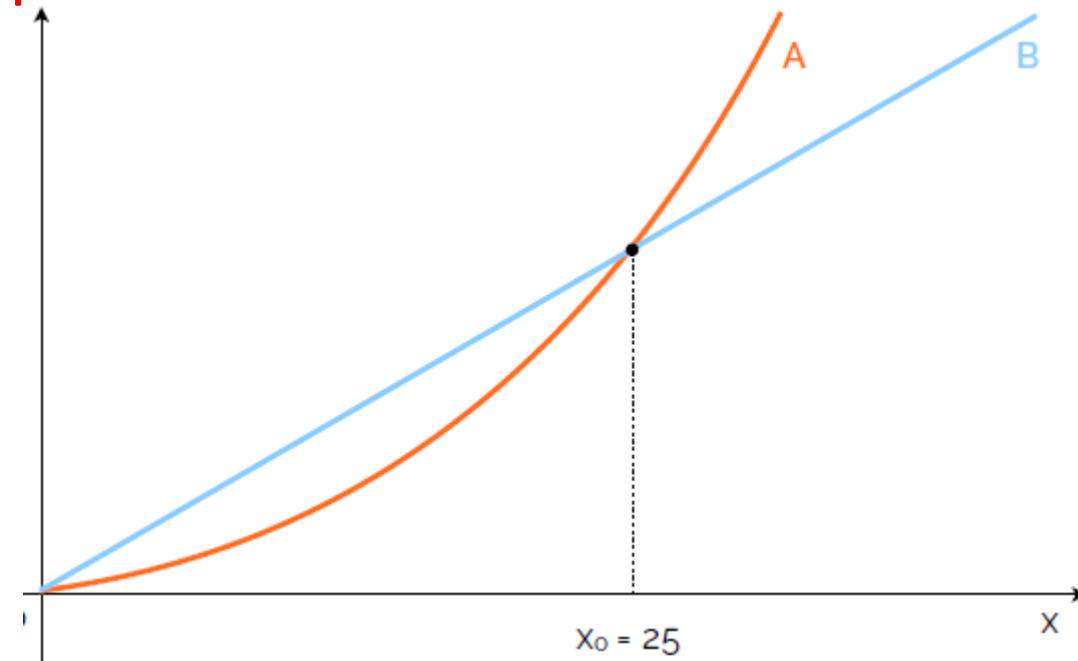
- **Análise experimental** do comportamento de um algoritmo
 - Testes computacionais com diferentes tipos de instâncias / dados de entrada
 - Contar o **nº de operações significativas** / Registrar o **tempo de execução**
 - Analisar os resultados
 - **Classificar o algoritmo** – qual é a sua **ordem de complexidade** ?
- **MAS**, o tempo de execução depende de **muitos fatores** !!

Análise formal

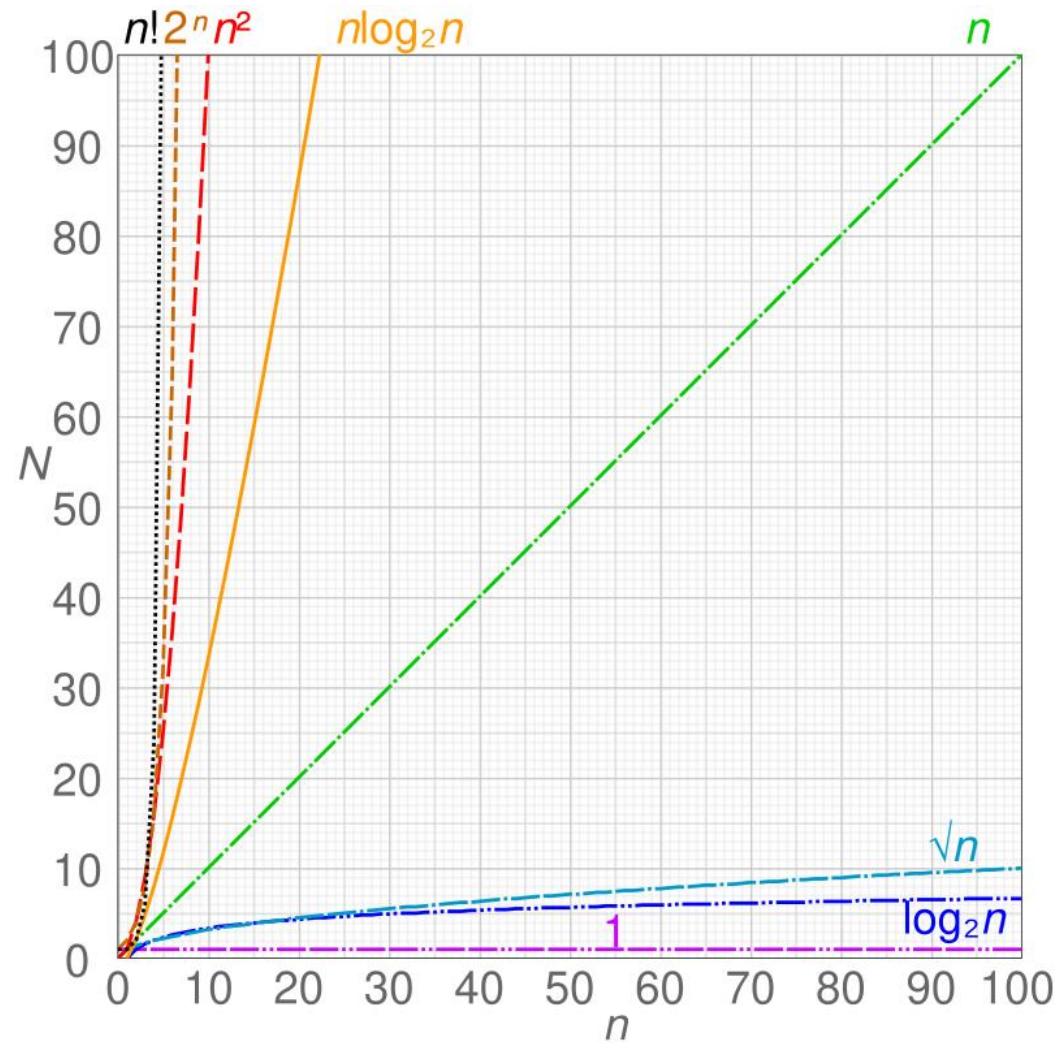
- **Análise formal** --- “papel e lápis”
 - Escolher uma **operação significativa**
 - Obter uma **expressão matemática** para o n^o de vezes que é executada, em função do “**tamanho**” dos dados
 - Analisar se a “**organização**” dos dados também influencia o n^o de operações
 - Classificar o algoritmo – qual é a sua **ordem de complexidade** ?
- Verificar se as **conclusões** são **compatíveis** com a análise experimental

Eficiência relativa

- Dois algoritmos para um mesmo problema, p.ex., array com **n** elementos
- **Algoritmo A** : comportamento **quadrático** : $n^2/5$
- **Algoritmo B** : comportamento **linear** : $5n$
- Qual é “**melhor**” ?
- Para **grandes volumes** dos dados ?



Classes de complexidade mais habituais



[Wikipedia]

Exemplos Simples

Exemplo 1

- Determinar o **maior de 3 valores** a, b, c
- Quantas **comparações** são efetuadas ?
- Quantas **atribuições** são efetuadas ?
- A **ordem dos dados** é importante ?
- Escrever os **invariantes**

```
if (a > b) {  
    if (a > c)  
        maior = a;  
    else  
        maior = c;  
} else {  
    if (b > c)  
        maior = b;  
    else  
        maior = c;  
}
```

Exemplo 2

- Determinar o **maior de 3 valores** a, b, c
- Quantas **comparações** são efetuadas ?
- Quantas **atribuições** são efetuadas ?
- A **ordem dos dados** é importante ?
- Escrever os **invariantes**

```
maior = a;  
if (b > maior)  
    maior = b;  
if (c > maior)  
    maior = c;  
...
```


Exemplo 3

```
inicializar array contador[256];  
de i = 0 até 256:  
    contador[i] = 0;  
enquanto não fim de ficheiro:  
    ler próximo carater;  
    incrementar contador[próximo carater];
```

- Contar o nº de **ocorrências de caracteres** num ficheiro
- **Inicialização** : quantas **atribuições** ao array ?
- **Leitura do ficheiro** : quantas vezes **incrementamos o array** ?
- Qual é o factor que determina o **desempenho** ?
- O esforço da fase de **inicialização** é importante ?

Ciclos — Contagem do número de iterações

Ciclos – Quantas iterações? – Expressão?

```
for(k=1; k<6; k++) {
```

```
    ...
```

?

```
}
```

```
for(i=0; i<m; i++) {
```

```
    for(j=0; j<n; j++) {
```

```
        ...
```

?

```
    }
```

```
}
```

Ciclos – Quantas iterações? – Expressão?

```
for(k=1; k<6; k++) {
```

```
    ...
```

```
}
```

$$\sum_{k=1}^5 1 = 5$$

```
for(i=0; i<m; i++) {
```

```
    for(j=0; j<n; j++) {
```

```
        ...
```

```
    }
```

```
}
```

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} 1 = \sum_{i=0}^{m-1} n = m \times n$$

Alguns resultados úteis

- $\sum_{k=1}^n 1 = n$
- $\sum_{k=1}^n k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$

- $\sum_{k=1}^n k^3 = \left(\frac{n(n+1)}{2}\right)^2$
- $\sum_{k=1}^n \frac{1}{k} = \log n + \underbrace{\gamma}_{\substack{\text{Euler's constant} \\ \approx 0.577216}} + \frac{1}{2n} + O(n^{-2})$
- $\sum_{k=n}^m f(k) = \sum_{k=1}^m f(k) - \sum_{k=1}^{n-1} f(k)$

Multiplicação de matrizes quadradas

Multiplicação de matrizes quadradas

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

- Quantos elementos tem a matriz resultado ?
- Quantas multiplicações se efetuam para calcular cada elemento ?
- Qual é o número total de multiplicações ?

Multiplicação de matrizes quadradas

```
for(int i=0; i<n; i++) {  
    for(int j=0; j<n; j++) {  
        c[i][j] = 0;  
        for(int k=0; k<n; k++) {  
            c[i][j] += a[i][k] * b[k][j];  
        }  
    }  
}
```



$$\sum_{i=0}^{n-1} \left[\sum_{j=0}^{n-1} \left(\sum_{k=0}^{n-1} 1 \right) \right]$$

Algoritmo **cúbico**

Generalização

- E para **matrizes de qualquer dimensão** :

$$A(m \times n) \times B(n \times p) = C(?)$$



- Quantas **multiplicações** são efetuadas ?
- O que é necessário alterar no código anterior ?



Exercícios / Tarefas

Tarefa 1

- Determinar o maior de 4 valores a, b, c, d
- Usar as duas estratégias apresentadas !!
- Faz sentido ?
- Como generalizar para n valores ?

Tarefa 2 – Obter as expressões finais

$$\sum_{k=2}^9 1 = ?$$

$$\sum_{k=a}^b 5 = ?$$

$$\sum_{i=a}^b i = ?$$

$$\sum_{i=1}^n 5i = ?$$

$$\sum_{i=1}^n 5i^2 = ?$$

$$\sum_{i=1}^n \sum_{j=1}^n 15 = ?$$

$$\sum_{i=1}^n \sum_{j=i}^n j = ?$$

$$\sum_{i=1}^n \sum_{j=1}^i 6j = ?$$

Tarefa 3

- Expressão para o nº de vezes que a **instrução mais interna** é executada
- Expressão para o **resultado** de cada uma das funções

```
int f3(int n) {  
    int i,j,r=0;  
    for(i = 1; i <= n; i++)  
        for(j = i; j <= n; j++)  
            r += 1;  
    return r;  
}
```

```
int f4(int n) {  
    int i,j,r=0;  
    for(i = 1; i <= n; i++)  
        for(j = 1; j <= i; j++)  
            r += j;  
    return r;  
}
```

```
int f1(int n) {  
    int i,r=0;  
    for(i = 1; i <= n; i++)  
        r += i;  
    return r;  
}
```

```
int f2(int n) {  
    int i,j,r=0;  
    for(i = 1; i <= n; i++)  
        for(j = 1; j <= n; j++)  
            r += 1;  
    return r;  
}
```

Tarefa 4

- Determinar o **maior de n valores** armazenados num **array**
- Qual é a “melhor” **estratégia** ?
- Escrever e testar o algoritmo
- Quantas **comparações** são efetuadas ?
- Quantas **atribuições** são efetuadas ?
- A **ordem dos dados** é importante ?
- Faz sentido **contar todas as atribuições** ?

Sugestões de leitura

Sugestões de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
 - Capítulo 1: **secção 1.1**
- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
 - Capítulo 1: **secção 1.1**