

Análise da Complexidade I

Introdução

Joaquim Madeira

11/03/2021

Sumário

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

Algoritmos

[Wikipedia]

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 usar !

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 oneroso determinar soluções exatas...
- Exemplos ?

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

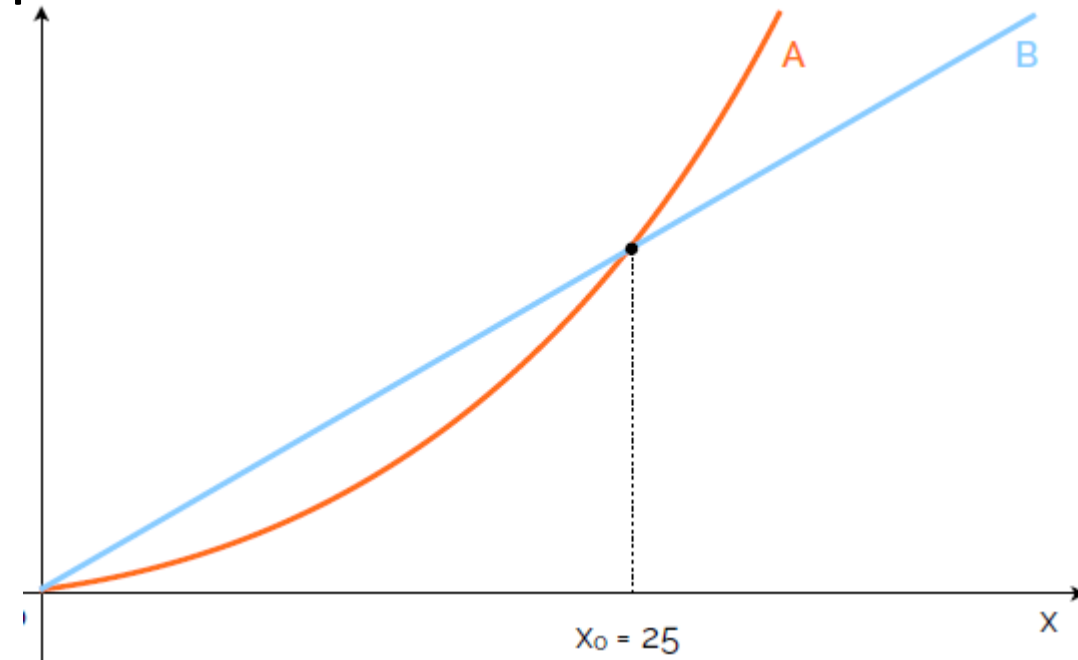
- **Análise experimental** do comportamento de um algoritmo
 - Testes computacionais com diferentes tipos de instâncias / dados de entrada
 - Contar o **nº de operações significativas** / Registrar o **tempo de execução**
 - Analisar os resultados
 - Classificar o algoritmo
- **MAS**, o tempo de execução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

Análise formal

- **Análise formal** --- “papel e lápis”
 - Escolher uma operação significativa
 - Obter uma **expressão matemática** para o nº de vezes que é executada, em função do tamanho dos dados
 - Analisar se a organização dos dados também influencia o nº de operações
 - Classificar o algoritmo
- Verificar se as **conclusões** são **compatíveis** com a análise experimental

Eficiência relativa

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



Exemplo 1

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

```
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
- Escrever os **invariantes**
- Quantas **comparações** ?
- Quantas **atribuições** ?
- A **ordem dos dados** é importante ?

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

Tarefa 1

- Determinar o maior de 4 valores a, b, c, d
- Usar as duas estratégias anteriores !!
- Faz sentido ?

Tarefa 2

- Determinar o **maior de n valores**
- Das anteriores, 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** ?

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 – 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++) {
```

```
        ...
```

```
    }
```

```
}
```

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

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;  
}
```

Sugestão 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**