Análise de programas imperativos

Paulo Mateus
Departamento de Matemática
IST

2018

Objectivos

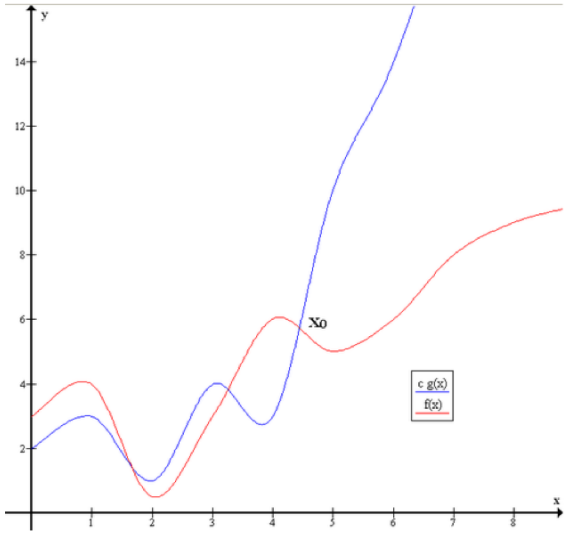
- Noção de invariante e variante de um ciclo
- Prova (informal) da correção de algoritmos imperativos
- Algoritmos de ordenação
- Análise da complexidade dos algoritmos de ordenação
- Análise do pior caso
- Análise de caso médio – suposições probabilísticas

Notação assimptótica

Def. $f(n) \in O(g(n))$ se $\exists k > 0, \exists n_0,\ \forall n > n_0,\ |f(n)| \leq |g(n) \cdot k|$

Análise do tempo de execução de um algoritmo

- constante: $O(1)$
- logarítmico: $O(\log n)$
- linear: $O(n)$
- quadrático: $O(n^2)$
- polinomial: $O(n^c),\ c > 1$
- exponencial: $O(c^n),\ c > 1$



Exercícios em Mathematica

Invariante

- Invariante na Lógica de Hoare:

$$\frac{\{C \wedge I\}\ \text{body}\ \{I\}}{\{I\}\ \textbf{while}\ (C)\ \text{body}\ \{\neg C \wedge I\}}$$

- **Def** (informal): Uma *asserção I* diz-se um *invariante* para um ciclo

while(C){ body}

se, quando for verificada no início do ciclo, também é verificada no fim de cada iterada do ciclo.

- Para tornar esta definição rigorosa era necessário definir:
  - Estado do programa (valores das variáveis + estado do controlo)
  - Sintaxe da lógica para os invariantes (primeira-ordem com predicados relevantes)
  - Semântica

- Nesta disciplina opta-se, como é comum na literatura, por utilizar linguagem matemática **informal** para analisar os algoritmos imperativos.

Invariante - factorial

```
// Exemplo:  factorial com contador a decrescer

public static int factorial(int n){
1     int i,r; r=1; i=n+1;
2     while(i>2) {// Invariante relaciona as variáveis à entrada do ciclo
3           i--;
4           r=r*i;
      }
      return r; // Retorno da função
}
```

$$r = \prod_{k=i}^{n} k \wedge i > 1$$

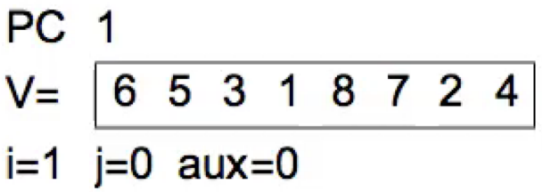
PC	r	i
2	1	4
2	3	3
2	6	2
Caso n=3		

Prop: O programa factorial, se terminar, calcula o factorial.

Algoritmo da Inserção

```
public static void insertionSort(int V[]){

1    int i=1, j, aux, n=V.length;
2    while(i < n){
3     aux=V[i];
4     j = i-1;
5     while(j>=0 && V[j] >aux) {
6        V[j+1] = V[j];
7        j--;}
8      V[j+1] = aux;
9      i++;}
}
```



Algoritmo da Inserção

```
public static void insertionSort(int V[]){

1 int i=1, j, aux, n=V.length;
2 while(i < n){    // Inv1: Os valores V[0]...V[i-1] estão ordenados e i<=n
3     aux=V[i];
4     j = i-1;
5     while(j>=0 && V[j] >aux) {  // Inv 2:V[j+1] > =aux ,  j>=-1 e
6        V[j+1] = V[j];           // {V[j+1],..,V[i]} , {V[0]...V[j]} ordenados
7        j--;}
8      V[j+1] = aux;
9      i++;}
}
```

Algoritmo da Inserção

```
public static void insertionSort(int V[]){

1 int i=1, j, aux, n=V.length;
2 while(i < n){    // Inv1: Os valores V[0]...V[i-1] estão ordenados e i<=n
3     aux=V[i];
4     j = i-1;
5     while(j>=0 && V[j] >aux) {
6         V[j+1] = V[j];
7         j--;}
8      V[j+1] = aux;
9      i++;}
}
```

PC	V	i
2	{**6**,4,3}	1
2	{**4,6**,3}	2
2	{**3,4,6**}	3
Caso V={6,3,4}		

Prop: A função insertionSort ordena por ordem crescente o vector V.

Variante

- **Def**. Uma relação (A,<) é *bem fundada* se qualquer subconjunto não vazio B de A tem elemento minimal.

- **Def**. Um *variante* de um ciclo **while(G){body} e invariante I** é um mapa dos "estados" que satisfazem I para o suporte A de uma relação bem fundada (A,**<**) cujo valor decresce estritamente em relação a **<** por cada iterada do ciclo.

- Informalmente entende-se por "estado" a configuração do programa, isto é, o valor das variáveis e o *program counter* (que comando está a ser executado).

Variante

- Regra da correção total do cálculo de Hoare

$$\frac{< \text{ is well-founded, } [I \wedge C \wedge V = z]\ S\ [I \wedge V < z]}{[I]\ \textbf{while}\ C\ \textbf{do}\ S\ [I \wedge \neg C]},$$

Teo: Dado um invariante I, um ciclo while termina **sse** existe um variante para esse ciclo e I.

Def: Um variante diz-se *natural* se o suporte da relação bem-fundada for um subconjunto dos números naturais.

Teo: Ciclos while com variantes naturais não são universais no que diz respeito a algoritmos computáveis à Turing (Ackermann 1929).

Nesta disciplina todos os procedimentos estudados **são algoritmos**, e o problema da terminação destes **é simples** e têm (quase sempre) **variante natural!**

Variante - factorial

```
public static int factorial(int n){
        int i,r;
        r=1;
        i=n+1;

        // Variante  ({2...n+1},<) e  f: {σ:σ[i]<=n+1}➔{2...n+1} tq f(σ)=σ[i]

        while(i>2) {
                i--;
                r=r*i;
        }
        return r; // Retorno da função
}
```

Prop: O programa factorial termina para todo o input.

Exer: Encontre variantes para os ciclos do Insertionsort.

Algoritmo da Inserção

```
public static void insertionSort(int V[]){

1    int i=1, j, aux, n=V.length;
2    while(i < n){
3     aux=V[i];
4     j = i-1;
5     while(j>=0 && V[j] >aux) {
6        V[j+1] = V[j];
7        j--;}
8      V[j+1] = aux;
9      i++;}
}
```