Eficiência de Algoritmos

Liana Duenha

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Primeiro Semestre de 2015
Ciência da Computação e Engenharia da Computação

Conteúdo da aula:

- Algoritmos e Estruturas de Dados
- Introdução à Complexidade de Algoritmos
- Métodos Empíricos x Analíticos
- Notação Assintótica
- Notações O, Θ e Ω
- Classes de Comportamento Assintótico

Algoritmos, estruturas de dados e tipos abstratos

- Algoritmo: corresponde a uma descrição de um padrão de comportamento, expresso em termos de um conjunto **finito** de ações para obtenção de uma solução para um determinado problema.
- Estruturas de Dados: informações ou dados organizados, manipulados pelo algoritmo ou programa.
- Tipo abstrato de dados: é um modelo matemático acompanhado das operações definidas sobre o modelo. Ex: o conjunto dos inteiros acompanhado das operações de adição, subtração e multiplicação.

Introdução à Complexidade de Algoritmos

- Avaliar algoritmos. Para que?
- Métricas mais utilizadas: tempo e espaço.
  - Métodos empíricos: experimentação. Contras: análise é totalmente dependente da implementação; medições de tempo em um computador são imprecisas, dependentes do sistema operacional da máquina e da concorrência entre diferentes programas em execução.
  - Métodos analíticos: expressão matemática que traduz o comportamento de tempo de um algoritmo. Independe da implementação, da arquitetura, da linguagem ou do sistema operacional.

Introdução à Complexidade de Algoritmos

Seja A um algoritmo e $E = \{E_1, ..., E_m\}$ o conjunto de todas as entradas possíveis de A. Denote por t_i o número de passos efetuados por A, quando a entrada for E_i. Definem-se

- complexidade de pior caso: $max_{E_i \in E}\{t_i\}$
- complexidade de melhor caso: $min_{E_i \in E}\{t_i\}$
- complexidade de caso médio: $\sum_{1 \leq i \leq m} p_i t_i$, onde p_i é a probabilidade de ocorrência da entrada E_i.

Simplificações adotadas:

- definimos sobre qual variável de entrada a expressão matemática avaliará o tempo de execução;
- quantidade de dados manipulados é suficiente grande;
- não consideramos constantes aditivas ou multiplicativas na expressão matemática obtida;

Primeiro Exemplo: `max_min1`: calcula o maior e o menor números dentro de um array ou vetor de n inteiros denominado `A`.

```
// considre max e min duas variaveis globais inteiras
1. void max_min1 (int A[MAX],  int n)
2. {
3.     max = A[0];
4.     min = A[0];
5.     for (int i=1; i<n; i++)
6.     {
7.         if (A[i] > max)  max = A[i];
8.         if (A[i] < min)  min = A[i];
9.    }
10.}
```

Introdução à Complexidade de Algoritmos

Seja f uma função de complexidade tal que $f(n)$ é o número de comparações entre os elementos do array A com n elementos. Duas comparações são realizadas dentro do loop, logo as complexidades de pior caso, caso médio e melhor caso são dadas por:

- $f(n) = 2(n-1)$

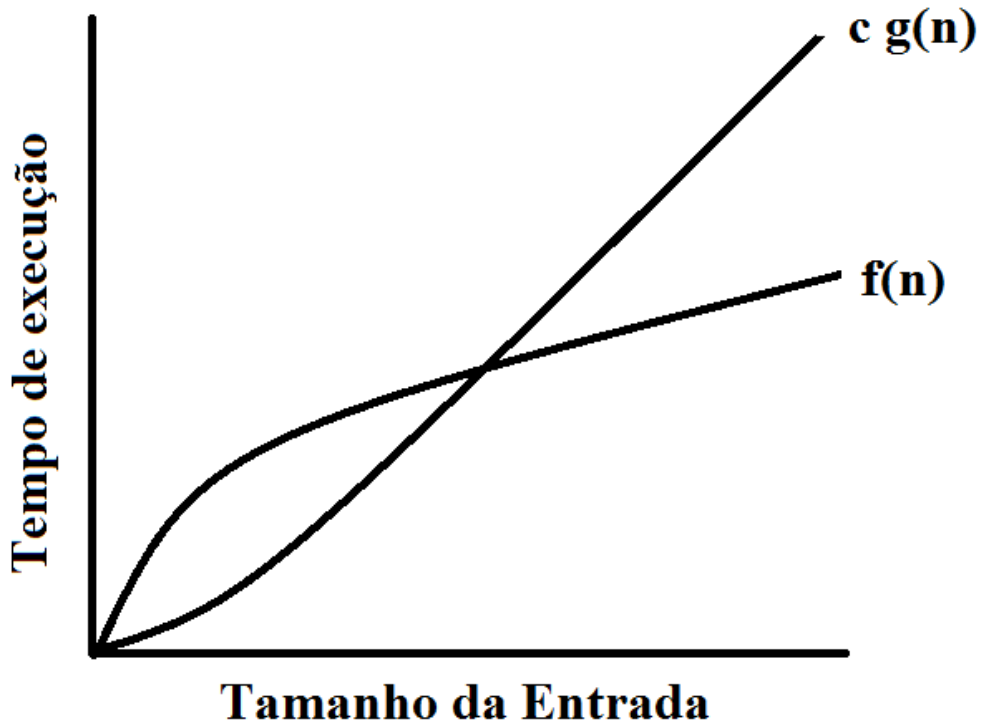
Segundo Exemplo: max_min2: melhoramento de max_min1

```
// considre max e min duas variaveis globais inteiras
1. void max_min2 (int A[MAX],  int n)
2. {
3.     max = A[0];
4.     min = A[0];
5.     for (int i=1; i<n; i++)
6.     {
7.         if (A[i] > max)  max = A[i];
8.         else if (A[i] < min)  min = A[i];
9.    }
10.}
```

- O melhor caso ocorre quando os elementos de A estão em ordem crescente.
  - melhor caso: $f(n) = n - 1$
- O pior caso ocorre quando os elementos de A estão em ordem descrescente.
  - pior caso: $f(n) = 2(n - 1)$
- Para avaliar o caso médio, considere que $A[i] > max$ é verdadeiro na metade das vezes.
  - caso médio: $f(n) = n - 1 + (n - 1)/2 = (3n/2) - (3/2)$, para $n > 0$

Notação Assintótica - Notação O

Uma função $g(n)$ domina assintoticamente outra função $f(n)$ se existem duas constantes positivas c e m tais que, para $n \geq m$, temos que $f(n) \leq cg(n)$. Nesse caso, dizemos que $f(n)$ é $O(g(n))$.



Notação Assintótica - Notação O

Alguns exemplos:

- A função $f(n) = 7n - 2$ é $O(n)$, visto que $7n - 2 \leq 7n$, para todo $n \geq 1$ (constantes $m = 1$ e $c = 7$).
- A função $f(n) = (n + 1)^2$ é $O(n^2)$, visto que $(n + 1)^2 \leq 4n^2$ para todo $n \geq 1$ (constantes $c = 4$ e $m = 1$).

Notação Assintótica - Notação O

- A função $f(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$, visto que $f(n) = 3n^3 + 2n^2 + n \leq 6n^3$, para todo $n \geq 0$ (constantes $c = 6$ e $m = 0$.
- Da mesma forma, é possível também mostrar que $f(n)$ é $O(n^4)$ ou $O(n^5)$, entretanto esses são resultados mais fracos do que dizer que $f(n)$ é $O(n^3)$.

Notação Assintótica - Notação O

Agora é com vocês:

- $f(n) = 3n^3 + 2n^2 + n$ é $O(n^3)$? E f(n) é $O(n^2)$?
- $f(n) = 2^{100}$ é $O(1)$?
- $f(n) = 3\log(n) + \log(\log(n))$ é $O(\log(n))$?
- $f(n) = 5/n$ é $O(1/n)$?

Propriedades da Notação O

Algumas operações que podem ser realizadas com a notação O são apresentadas abaixo.

- $f(n) = O(f(n))$
- $cO(f(n)) = O(f(n))$, c constante
- $O(f(n)) + O(f(n)) = O(f(n))$
- $O(O(f(n))) = O(f(n))$
- $O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$
- $O(f(n))O(g(n)) = O(f(n)g(n))$
- $f(n)O(g(n)) = O(f(n)g(n))$

Notação Assintótica - Notação Ω

- Uma função $f(n)$ é $\Omega(g(n))$ se existirem duas constantes c e m tais que $f(n) \geq cg(n)$ para todo $n \geq m$.
- Por exemplo, para mostrar que $f(n) = 3c^3 + 2n^2$ é $\Omega(n^3)$, basta fazer $c = 1$ e então $3n^3 + 2n^2 \geq n^3$ para todo $n \geq 0$ (constantes $c = 1$ e $m = 0$).

Notação Assintótica - Notação Θ

- Uma função $f(n)$ é $\Theta(g(n))$ se existirem constantes positivas c_1, c_2 e m tais que $0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$ para todo $n \geq m$.
- Em outras palavras, para todo $n \geq m$, a função $f(n)$ é igual a $g(n)$ a menos de uma constante. Nesse caso, $g(n)$ é um limite assintótico **justo** ou **firme** para $f(n)$.

Classes de Comportamento Assintótico

Se $f(n)$ é a função de complexidade para um algoritmo F, em termos de entradas de tamanho n, então $O(f(n))$ é considerada a complexidade assintótica ou o comportamento assintótico de F.

- $f(n) = O(1)$. Algoritmos de complexidade constante. O tempo de execução do algoritmo independe do tamanho do problema (valor n).
- $f(n) = O(\log n)$. Algoritmos de complexidade logarítmica. Comumente utilizamos base 2, mas pode-se também variar a base de acordo com o problema em questão.
- $f(n) = O(n)$. Algoritmos de complexidade linear.
- $f(n) = O(n \log n)$. Comportamento de algoritmos que resolvem problemas dividindo-os em subproblemas. É bastante comum nos melhores algoritmos de ordenação por comparação.

Classes de Comportamento Assintótico

- $f(n) = O(n^2)$. Algoritmos de complexidade quadrática.
- $f(n) = O(n^3)$. Algoritmos de complexidade cúbica.
- $f(n) = O(n^k)$, $k = 1, 2, ...$ Algoritmos de complexidade polinomial. Englobam os lineares, quadráticos, cúbicos, etc.
- $f(n) = O(c^n)$, c constante. Algoritmos de complexidade exponencial. Geralmente não são úteis do ponto de vista prático (não terminam em tempo hábil). Por exemplo, um algoritmo cuja função de complexidade seja 3^n, para $n = 60$ o algoritmos terminaria sua execução em aproximadamente 10^{13} séculos.
- $f(n) = O(n!)$. Algoritmos de complexidade fatorial. Para ter uma ideia de quão ineficiente é um algoritmo de complexidade fatorial, para um problema de "tamanho" $n = 40$, o tempo de execução do algoritmo é proporcional a 40!, um número de 48 dígitos.

Classes de Comportamento Assintótico

Exercício: Forneça uma análise para cada um dos laços abaixo. A análise deve ser realizada em termos de alguma operação primitiva relacionada ao loop e você pode considerar que as variáveis de controle do laço (i,j,n) já tenham sido declaradas.

1. ```
   1. int s =0;
   2. for (i=0; i<n*n;i++)
   3.     for (j=0; j<i; j++)
   4.         s = s+i;
   ```
2. ```
   1. int p =0;
   2. for (i=0; i<n;i++)
   3.     for (j=0; j<i*i; j++)
   4.         p = p*2;
   ```