INF 112: Programação II

Aula 04

- **Introdução à análise de complexidade**

Fábio R. Cerqueira, UFV, DPI, frcerqueira@gmail.com

Análise de algoritmo

- O que analisar:
  - Eficiência de tempo;
  - Eficiência de espaço.

- Nosso foco será <u>tempo</u>.

Análise de algoritmo

- Em geral, os algoritmos gastam mais tempo para serem executados quando a entrada é aumentada.

- Portanto, é bastante lógico investigar a eficiência do algoritmo como uma função de algum parâmetro *n* que indique o tamanho da entrada do algoritmo.

Como medir tempo de execução?

- Podemos simplesmente usar alguma unidade padrão de medida de tempo (segundos, milissegundos, etc.) para medir o tempo de execução de um programa que implementa o algoritmo.

- Mas há alguns problemas com esta estratégia:
  - Dependência da velocidade de um computador particular;
  - Dependência da qualidade do programa que implementa o algoritmo e ainda do compilador utilizado;
  - E a dificuldade de cronometrar o tempo de execução real de um programa.

Como medir tempo de execução?

- Uma possível solução é contar o número de vezes que cada operação do algoritmo é executada.

- Veja que se quisermos, eventualmente, fazer uma estimativa precisa de tempo, isto seria interessante.

- No entanto, quando se analisa um algoritmo, o que se deseja, mais frequentemente, é perceber como o mesmo se comporta relativamente à entrada fornecida. Particularmente, como o número de operações realizadas cresce à medida que a entrada aumenta. Para o entendimento deste comportamento, contar todas as operações é normalmente desnecessário.

Como medir tempo de execução?

- O que se pode fazer é identificar a operação mais importante do algoritmo, chamada de ***operação básica***, que é aquela que mais contribui para o tempo de execução total.

- A partir daí, computar o número de vezes que a operação básica é executada.

- A operação básica é, via de regra, aquela que se encontra no laço mais interno do algoritmo e a que mais tempo consome.

Exemplo de tamanho de entrada e operações básicas

Problema	*Tamanho da entrada*	*Operação Básica*
Buscar um elemento em uma lista de *n* itens	Número de elementos na lista (ou seja, *n*)	Comparações
Multiplicação de duas matrizes	As dimensões da matriz ou o número total de elementos	Multiplicação de dois números
Checar se um dado inteiro *n* é primo	Número de dígitos (em representação binária)	Divisão
Problema típico de grafos	Número de vértices e arestas	Visitar um vértice ou atravessar uma aresta

Análise teórica de eficiência de tempo

- **Analisa-se a eficiência de tempo determinando-se o número de repetições da operação básica. Esta contagem será na forma de uma função do tamanho da entrada.**

- ***<u>Operação básica</u>*: aquela que mais contribui para o tempo total de execução do algoritmo**

Tamanho da entrada

$$T(n) \approx c_{op}C(n)$$

Tempo de execução

Tempo de execução da operação básica

Número de vezes que a operação básica é executada

Análise teórica de eficiência de tempo

- Assumindo-se que $C(n) = ½n(n\text{-}1)$, qual será a diferença no tempo para se executar um algoritmo se compararmos uma execução para um certo tamanho de entrada n e outra execução para $2n$ (ou seja, se dobrarmos o tamanho da entrada)? A resposta é: a segunda execução levaria, aproximadamente, quatro vezes mais tempo:

$$C(n) = ½n(n\text{-}1) = ½n^2 – ½n \approx ½n^2 \text{ (para valores de } n \text{ bem altos)}$$

e portanto:

$$T(2n)/T(n) \approx c_{op}C(2n) / c_{op}C(n) \approx ½(2n)^2 / ½n^2 = 4.$$

Ordem de complexidade ou taxa de crescimento

- **Vemos então que ignora-se constantes multiplicativas e concentra-se na análise da *taxa de crescimento* (também chamada de *ordem de complexidade*) do algoritmo para entradas de tamanho elevado.**

- **Por que esta ênfase no cálculo da taxa de crescimento para entradas cada vez maiores?**

- **Simplesmente porque para entradas de tamanho pequeno, o tempo de execução normalmente não importa, não faz diferença.**

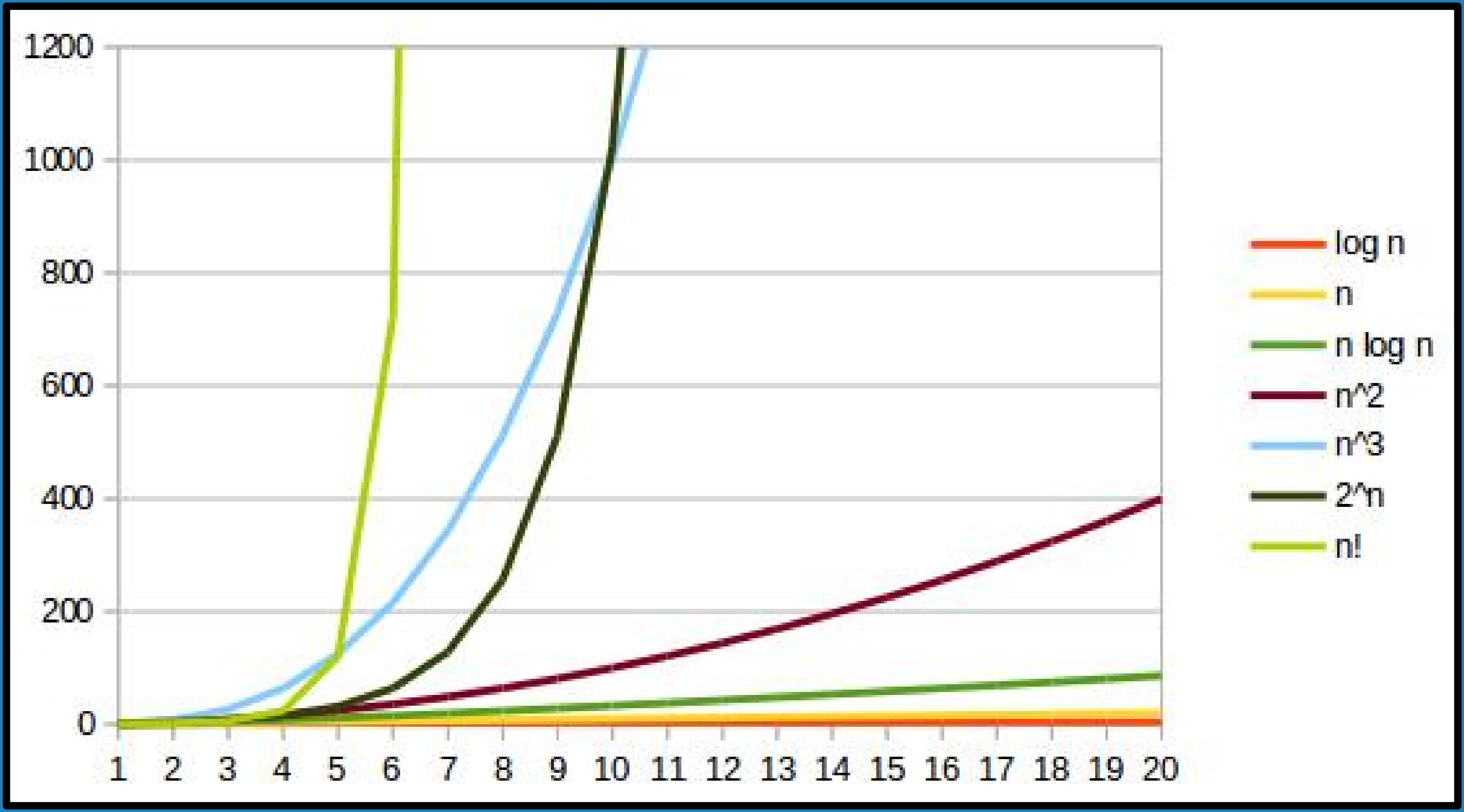
Valores de algumas funções importantes à medida que $n \to \infty$

- Vejamos a tabela abaixo para entender melhor:

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

Valores de algumas funções importantes à medida que $n \to \infty$



Exemplo para *n*!

- **Suponha que a operação básica de um certo algoritmo gaste 10^{-9} segundos para ser executada em um programa que o implemente e que esta operação seja executada *n*! vezes. Ou seja, suponha que:**

$$T(n) \approx 10^{-9} n!$$

- **Neste caso, teríamos:**

n	Tempo de execução
20	77 anos
21	1620 anos

Melhor caso, caso médio, pior caso

Para contar o número de operações realizadas pelo algoritmo, há três cenários possíveis para a análise de eficiência:

- **Pior caso: $C_{pior}(n)$ – máximo para entradas de tamanho n**

- **Melhor caso: $C_{melhor}(n)$ – mínimo para entradas de tamanho n.**

- **Caso médio: $C_{médio}(n)$ – média para entradas de tamanho n.**
  - **Número de vezes que a operação básica será executada para entradas <u>típicas</u>;**
  - **<u>NÃO</u> é a média do pior e melhor caso;**
  - **É o número esperado de operações básicas como uma variável aleatória advinda de uma suposição sobre a distribuição de probabilidade de todas as possíveis entradas.**

Exemplo: busca sequencial

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element of A that matches K
//          or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

- **Pior caso: $C_{pior}(n) = n$.**
- **Melhor caso: $C_{melhor}(n) = 1$.**
- **Caso médio ...**

Melhor caso, caso médio, pior caso

Caso médio da busca sequencial:

- **Suponha que:**
  - **A probabilidade de uma busca com sucesso seja *p***
  - **e a probabilidade da primeira ocorrência ocorrer na *i*-ésima posição da lista seja a mesma para todo *i*.**
- **No caso de uma busca com sucesso, a probabilidade da primeira ocorrência ser na *i*-ésima posição da lista é *p*/*n* para todo *i*. O número de comparações feitas pelo algoritmo será *i*.**
- **No caso de uma busca sem sucesso, o número de comparações será *n* com probabilidade 1-*p*.**

Melhor caso, caso médio, pior caso

- **Portanto, o número médio de comparações do algoritmo *SequentialSearch* será:**

$$C_{\text{médio}}(n) = [1.p/n + 2.p/n + \ldots + i.p/n + \ldots + n.p/n] + n.(1-p)$$
$$= p/n[1 + 2 + \ldots + i + \ldots + n] + n(1-p)$$
$$= p/n \times n(n+1)/2 + n(1-p) = p(n+1)/2 + n(1-p).$$

- **Note que:**
  - **Se $p = 1$ (i.e., a busca deverá ter sucesso), o número médio de comparações será de $(n+1)/2$**
  - **If $p = 0$ (i.e., a busca certamente não terá sucesso), o número médio de comparações será de n.**

Exercícios

1) Para cada caso a seguir, indique de quanto o valor da função será alterado se seu argumento for aumentado em quatro vezes:

a) $\log_2 n$ b) $\sqrt{n}$ c) n d) n^2 e) n^3 f) 2^n

2) De acordo com uma lenda bem conhecida, o jogo de xadrez foi inventado há vários séculos atrás, no noroeste da Índia, por um sábio chamado Shashi. Quando ele levou seu invento para o rei, o mesmo gostou tanto do jogo que ofereceu ao inventor qualquer recompensa que quisesse. Sashi pediu que lhe fossem dados grãos. A quantidade seria obtida da seguinte forma: Apenas um grão de trigo teria que ser colocado no primeiro quadrado do tabuleiro, dois grãos no segundo, quatro no terceiro, oito no quarto, e assim por diante, até que os 64 quadrados fossem todos preenchidos. Pergunta-se: Qual será o resultado final deste “algoritmo para obtenção de grãos”? (O que seria a entrada deste algoritmo?)