

STCO01 - Algoritmos e Programação I - Aula05

Pedro Hokama

Apresentação Baseada:

- Stanford Algorithms
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V>
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBslsZ17A5HEK6>
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

1 / 16

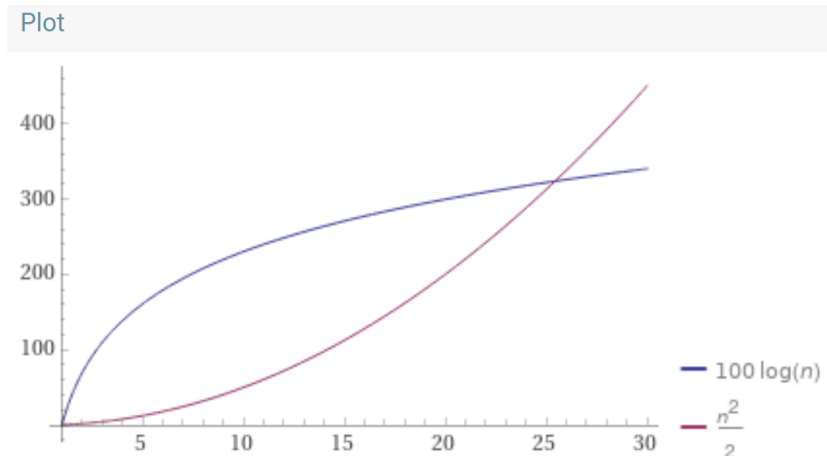
2 / 16

Princípios da Análise de Algoritmos

- Iremos fazer uma Análise **Assintótica** dos algoritmos, o que significa que estamos interessados no comportamento deles para instâncias **grandes**.
- Isso nos permite dizer que um algoritmo é mais rápido que outro assumindo que o tamanho n da instância é suficientemente grande.
- Por exemplo, podemos dizer com segurança que um algoritmo que executa em $20 \log n$ é mais rápido que um que executa em $\frac{1}{2}n^2$
- Note que isso pode não ser verdade para n pequeno, mas a partir de algum $n = n_0$ sempre será verdade.
- De fato, para n pequeno, tanto faz o algoritmo que você use. Estamos interessados em resolver problemas grandes!

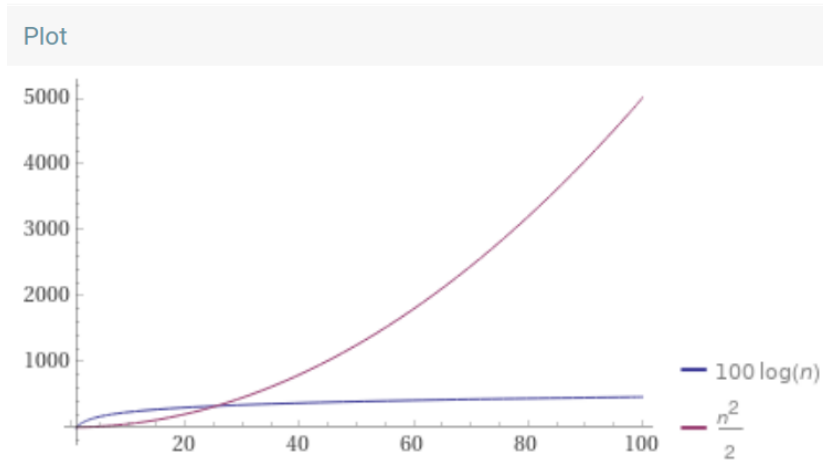
3 / 16

Princípios da Análise de Algoritmos



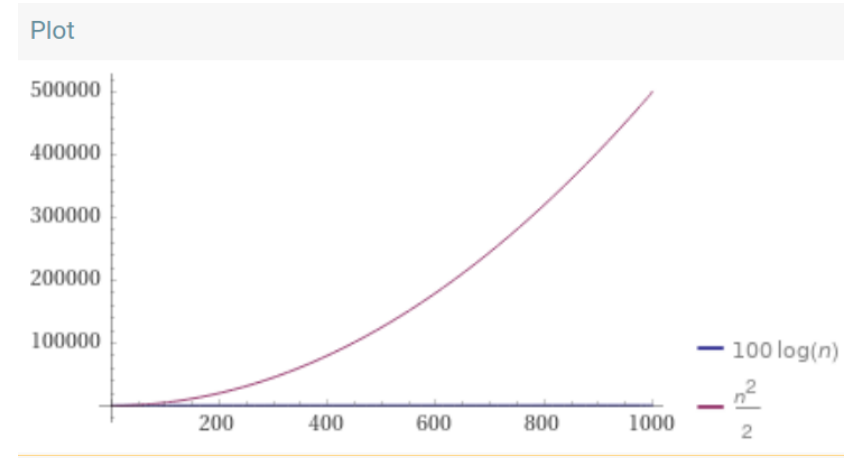
4 / 16

Princípios da Análise de Algoritmos



5 / 16

Princípios da Análise de Algoritmos



6 / 16

Princípios da Análise de Algoritmos

- Você poderia imaginar que com o avanço dos hardwares, bastaria eu usar um computador melhor.
- Na verdade quanto maior o poder computacional, MAIOR é a disparidade entre algoritmos eficientes.
- Podemos pensar no tamanho do problema que podemos resolver com computadores mais potentes usando diferentes algoritmos.
- Suponha que você tem dois algoritmos para um problema.

Algoritmo A	Algoritmo B
n	n^2

- Suponha que você fez um grande investimento e comprou um computador 4 vezes mais potente. Com o algoritmo A você pode resolver um problema 4 vezes maior, enquanto com o algoritmo B você só pode resolver um problema 2 vezes maior.

7 / 16

Princípios da Análise de Algoritmos

- Aplicaremos a Análise de Pior Caso
- Esse limite também é aplicado se um adversário tentasse atribuir números de forma a deixar o algoritmo lento.
- Essa análise é particularmente interessante por não precisar entender a aplicação do problema, padrões de entrada e ela é usualmente mais útil e mais fácil que as alternativas:
 - ▶ Análise de Caso Médio (Exige assumir alguma distribuição da entrada, ter conhecimento do domínio, mais difícil de ser feita)
 - ▶ Desempenho em *Benchmarks*
- A análise assintótica é uma ferramenta adequada pois:
 - ▶ É simples o bastante para suprimir detalhes de arquitetura/linguagem/compiler.
 - ▶ Mas é complexa o bastante para permitir a comparação entre diferentes algoritmos, especialmente em instâncias grandes.

8 / 16

Análise Assintótica

Ideia geral

Suprimir fatores constantes e termos de ordens inferiores.

- Por exemplo em:

$$3n^2 + 10n + 50$$

$10n + 50$ são termos de ordem inferior, 3 é constante então resultaria em:

$$n^2$$

- Então quando dizemos que o tempo de execução de um algoritmo é $O(n^2)$, ou de maneira geral quando dizemos que um algoritmo é $O(f(n))$. Estamos dizendo que depois de eliminar os termos de ordem inferior e constantes acabamos apenas com $f(n)$.

9 / 16

Análise assintótica de funções quadráticas - termos de menor ordem

Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

- Como se vê, $3n^2$ é o termo dominante quando n é grande.
- De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.

10 / 16

Exemplos

Exemplo de um laço

Algoritmo 1: Procura

Entrada: Um vetor A de tamanho n e um inteiro t

Saída: Verdadeiro se A contém t , Falso caso contrário

- 1 para k de 1 até n faça
- 2 se $A[k] == t$ então
- 3 devolva Verdadeiro;
- 4 devolva Falso;

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

- O tempo de execução do exemplo depende por exemplo se t está ou não em A , e se t estiver na primeira posição? Estamos interessados no pior caso!
- Quantas operações estamos fazendo nas linhas 1 e 2? Uma, duas, três? Isso é constante e é suprimido pela notação O .

11 / 16

Exemplos

Exemplo de dois laços consecutivos

Algoritmo 2: Procura2

Entrada: Dois vetores A e B de tamanho n e um inteiro t

Saída: Verdadeiro se A ou B contém t , Falso caso contrário

- 1 para k de 1 até n faça
- 2 se $A[k] == t$ então devolva Verdadeiro;
- 3 para k de 1 até n faça
- 4 se $B[k] == t$ então devolva Verdadeiro;
- 5 devolva Falso;

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

- Evidentemente, nesse problema a entrada é na verdade de tamanho $2n$, então o algoritmo não seria $O(2n)$? Essa constante é suprimida na notação O .

12 / 16

Exemplos

Exemplo de dois laços aninhados

Algoritmo 3: ProcuraComum

Entrada: Dois vetores A e B de tamanho n

Saída: Verdadeiro se A ou B têm um número em comum, Falso caso contrário

```
1 para  $j$  de 1 até  $n$  faça
2   para  $k$  de 1 até  $n$  faça
3     se  $A[j] == B[k]$  então devolva Verdadeiro;
4 devolva Falso;
```

Qual o tempo de execução desse algoritmo?

- a $O(1)$
 - b $O(\log n)$
 - c $O(n)$
 - d $O(n^2)$
- Pior caso ✓. Constantes ✓.
 - Nesse caso se o tamanho dos vetores dobrar, o tempo de execução quadruplica!

13 / 16

Notação O

- Big Oh, Ózão, O grande.
- Notação O é utilizada em funções definidas nos inteiros positivos.
- Seja $T(n)$ uma função sobre $n = 1, 2, 3, \dots$
- $T(n) : \mathbb{Z}_+^* \rightarrow \mathbb{R}$
- Pergunta: Quando podemos dizer que $T(n) = O(f(n))$?
- Resposta: Se eventualmente, para um n suficientemente grande, $T(n)$ é limitado superiormente por uma alguma constante vezes $f(n)$.

15 / 16

Exemplos

Exemplo de dois laços aninhados

Algoritmo 4: ProcuraComum

Entrada: Um vetor A de tamanho n

Saída: Verdadeiro se A tem números duplicados, Falso caso contrário

```
1 para  $j$  de 1 até  $n$  faça
2   para  $k$  de  $j + 1$  até  $n$  faça
3     se  $A[j] == A[k]$  então devolva Verdadeiro;
4 devolva Falso;
```

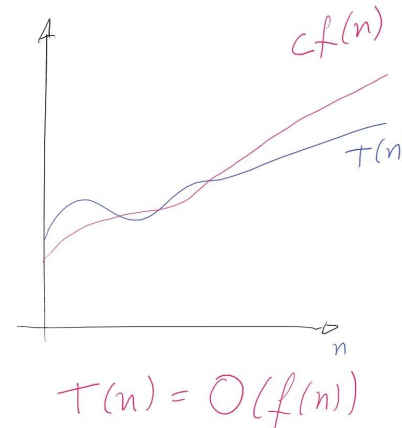
- Dessa vez, obviamente, procuramos no vetor A ao invés de B .
- Ao invés de olhar todas combinações de j e k , olhamos apenas aquelas em que $k \geq j + 1$.
- Isso é válido pois comparar, por exemplo, o primeiro com o terceiro elemento é a mesma coisa que comparar o terceiro com o primeiro. Então diminuímos nossas comparações pela metade!

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

Exercício: É possível fazer esse algoritmo mais eficiente? Como?

14 / 16



- $T(n) = O(f(n))$ se quanto multiplicado por alguma constante c tal que $c \cdot f(n)$ está sempre acima de $T(n)$

16 / 16