Notas de aula: Análise de Algoritmos
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profa. Carla Negri Lintzmayer

Introdução à análise de algoritmos
(*Última atualização: 18 de fevereiro de 2019*)

O que você precisa para acompanhar o curso

- Não usaremos uma linguagem de programação específica, mas você deve pelo menos minimamente saber programar em alguma (C, Java, Python, ...).
  - Em particular, espero que você seja capaz de entender os pseudocódigos dados e, se necessário, implementá-los em alguma linguagem.
  - Recursão é um tópico básico.
- Idealmente, é bom que você tenha um conhecimento prévio em estruturas de dados básicas.
  - Vetores, listas, pilhas, filas e árvores.
- Seja capaz de reconhecer argumentos lógicos em uma prova matemática (por indução, contradição, construção).
- Tenha familiaridade com linguagem matemática (quantificadores lógicos, somatórios e manipulação de funções).
- Veja os materiais de apoio.

Referências e materiais complementares desse tópico

Livro Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.. Introduction to Algorithms. 2nd ed. MIT Press. 2002.
Capítulo 1.

Livro Dasgupta, S.; Papadimitriou, C.; Vazirani, U.. **Algorithms**. Boston: McGraw-Hill. 2008.
Capítulo 0.2.

Vídeos Vídeo aulas do prof. Tim Roughgarden, de Stanford, em inglês (com legendas)
Aulas **1.1**, **1.2** e **1.3**.

Sumário

1 O que é um algoritmo?

- É um conjunto de regras bem definidas que recebem uma entrada e produzem uma saída.
- É uma ferramenta para resolver um problema: ordenar um conjunto de itens, encontrar o menor caminho entre uma origem e um destino, alocar disciplinas a professores e a salas de aula, etc.
- Consideramos que todo problema possui uma entrada e uma saída bem definidas.
- Dizemos que um algoritmo está **correto** (ou que ele **resolve** o problema em questão) se **para qualquer** instância de entrada do problema, ele produz a saída correta.
  - Exemplo: se o problema é ordenar um conjunto de inteiros, um algoritmo estará correto se e somente se for capaz de ordenar qualquer conjunto de inteiros. Se houver algum número imaginário, por exemplo, dentre o conjunto de números, o algoritmo não necessariamente irá ordená-los.
- Iremos sempre supor que a entrada/instância recebida por um algoritmo satisfaz as restrições que estiverem definidas na descrição do problema.

2 Por que estudar algoritmos?

- Algoritmos (e estruturas de dados) são muito importantes em praticamente todas as áreas da Ciência da Computação.
  - Roteamento, criptografia, computação gráfica, banco de dados, ...
- Permite inovação tecnológica, superando até mesmo o avanço exponencial da lei de Moore.
  - Google PageRank, Projeto Genoma Humano, ...
- Auxilia com novos pontos de vista em outras áreas.
  - Economia, evolução, mecânica quântica, ...
- Desafiador e divertido.
  - Exige muita criatividade, raciocínio lógico e capacidade analítica

> *Suponha que os computadores fossem infinitamente rápidos e que a memória do computador fosse gratuita. Você teria alguma razão para estudar algoritmos? A resposta é sim, se não por outra razão, pelo menos porque você ainda gostaria de demonstrar que o método da sua solução termina, e o faz com a resposta correta.*
> *Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C..*

3 O que é análise de algoritmos?

- Tarefa que objetiva prever o comportamento/desempenho de um algoritmo sem ter que implementá-lo em um computador específico.
- Esse comportamento envolve o uso de recursos como memória, largura de banda e **tempo**.
- Descreveremos comportamento em função do tamanho da entrada, contando o número de **passos básicos** que são feitos pelo algoritmo.
  - somar/subtrair/multiplicar/dividir dois números pequenos, atribuir um número a uma variável, comparar dois números, ...

4 Exemplo

> PROBLEMA: MULTIPLICAÇÃO DE INTEIROS
>
> ENTRADA: dois números x e y contendo n dígitos cada.
> SAÍDA: o produto xy.

- O clássico algoritmo que aprendemos no início de nossas vidas acadêmicas resolve esse problema.

Algoritmo 1 Algoritmo básico para o problema de multiplicação de inteiros.

função MULTIPLICA1(x, y, n)
    Seja $x = 5678$ e $y = 1234$ (ou seja, $n = 4$)

$$\begin{array}{r}
5678 \\
\times 1234 \\
\hline
22712 \\
170340 \\
+1135600 \\
5678000 \\
\hline
7006652
\end{array}$$

- EXERCÍCIO: claramente o Algoritmo 1 mostra apenas um exemplo. Descreva formalmente esse algoritmo.
- Uma vez que descrevemos um algoritmo para um problema, existem três principais perguntas que **sempre** devemos fazer:

  1. O algoritmo está **correto**?
     - Isso é, para **qualquer** entrada que ele recebe, ele devolve a saída esperada?
     - No problema da multiplicação de inteiros, seja $y = y_1 y_2 \dots y_n$ onde y_i é um dígito de 0 a 9. Note que o algoritmo faz

       $$(x \times y_n) + (x \times y_{n-1} \times 10) + \cdots + (x \times y_2 \times 10^{n-2}) + (x \times y_1 \times 10^{n-1}) \ ,$$

       que equivale exatamente a xy.

- Note como não fizemos nenhuma suposição sobre x, y ou n e, portanto, a análise acima indica que o algoritmo está correto.

2. Quanto **tempo** o algoritmo leva?
   - Como mencionado, nos preocupamos com o número de passos básicos realizados pelo algoritmo.
   - Note que no algoritmo de multiplicação, para obter o primeiro produto parcial ($x \times y_n$), precisamos de n multiplicações de um dígito e talvez mais $n-1$ somas (para os *carries*), isto é, no máximo $2n$ *operações básicas*[1]. Similarmente, para obter $x \times y_{n-1} \times 10$, outras no máximo $2n$ operações básicas foram necessárias. E isso é verdade para todos os produtos parciais. Assim, são no máximo $2n$ operações para cada um dos n dígitos de y, isto é, $2n^2$ operações no máximo. Contando com as adições finais, que são outras no máximo $2n^2$ operações, temos que o número total de operações básicas é no máximo cn^2, onde c é alguma constante (valor que independe de n).
   - PERGUNTA: o que acontece se o tamanho da entrada dobra? E se quadruplica?
3. Dá para fazer **melhor**?
   - Quer dizer, é possível encontrar xy com menos do que cn^2 operações básicas?
   - Até esse momento da sua vida, talvez você nunca tenha pensado nisso, mas a resposta é sim! Existem algoritmos muito melhores do que esse que aprendemos há tantos anos em nossas vidas.

> Perguntas importantes que sempre devem ser feitas:
>
> - O algoritmo está correto?
> - Quanto tempo ele leva?
> - Dá para fazer melhor?

- Vamos escrever $x = 10^{n/2}a + b$ e $y = 10^{n/2}c + d$, onde a, b, c e d são números com $n/2$ dígitos cada. Por exemplo, se $x = 5678$ e $y = 1234$, então $a = 56$, $b = 78$, $c = 12$ e $d = 34$.
- Podemos então reescrever xy como $(10^{n/2}a + b)(10^{n/2}c + d)$, o que é igual a

$$10^n ac + 10^{n/2}(ad + bc) + bd \ . \tag{1}$$

- Note que apenas **reduzimos** o tamanho do problema: antes, precisávamos multiplicar dois números de n dígitos cada e agora precisamos multiplicar vários números de $n/2$ dígitos cada.
- O Algoritmo 2 apresenta um algoritmo recursivo que também resolve o problema da multiplicação.
- O algoritmo MULTIPLICA2 está correto?

[1] O conjunto de instruções de um processador possui várias instruções básicas, como aritmética simples, armazenamento de valores na memória, comparação de números, etc.

Algoritmo 2 Algoritmo básico recursivo para o problema de multiplicação de inteiros.

função MULTIPLICA2(x, y, n) $\triangleright$ Suponha que n é potência de 2
  se $n = 1$ **então**
    devolve xy
  Seja $x = 10^{n/2}a + b$ e $y = 10^{n/2}c + d$, onde a, b, c e d são números com $n/2$ dígitos cada
  $p_1 \leftarrow$ MULTIPLICA2(a, c, $n/2$)
  $p_2 \leftarrow$ MULTIPLICA2(a, d, $n/2$)
  $p_3 \leftarrow$ MULTIPLICA2(b, c, $n/2$)
  $p_4 \leftarrow$ MULTIPLICA2(b, d, $n/2$)
  devolve $10^n p_1 + 10^{n/2}(p_2 + p_3) + p_4$

  - Por hora, vamos fazer uma análise bem "solta"[2]: claramente, o algoritmo calcula xy pois o que ele faz é calcular a Equação 1.

- Quanto tempo ele leva?
  - Em particular, será que ele é melhor que o MULTIPLICA1?
  - Veja que não é fácil contar o número de passos básicos que MULTIPLICA2 faz, mas nós veremos isso durante o curso.
  - Por hora, a resposta é não: MULTIPLICA2 também faz umas dn^2 operações básicas, onde d é uma constante.

- Dá para fazer melhor?
  - Sim!! O Algoritmo 3, conhecido como algoritmo de Karatsuba, é recursivo também e resolve o problema da multiplicação.
  - **Atenção:** nós só podemos afirmar que o algoritmo de Karatsuba é melhor do que MULTIPLICA2 quando mostrarmos quanto tempo ele leva para multiplicar dois números de tamanho n (para somente então comparar isso com o dn^2 feito pelo MULTIPLICA2).

Algoritmo 3 Algoritmo de Karatsuba para o problema de multiplicação de inteiros.

função KARATSUBA(x, y, n) $\triangleright$ Suponha que n é potência de 2
  se $n = 1$ **então**
    devolve xy
  Seja $x = 10^{n/2}a + b$ e $y = 10^{n/2}c + d$, onde a, b, c e d são números com $n/2$ dígitos cada
  $p_1 \leftarrow$ KARATSUBA(a, c, $n/2$)
  $p_2 \leftarrow$ KARATSUBA(b, d, $n/2$)
  $p_3 \leftarrow$ KARATSUBA($a + b$, $c + d$, $n/2$)
  $\triangleright$ Note que $(a + b)(c + d) = ac + ad + bc + bd$, logo $p_3 - p_1 - p_2 = ad + bc$
  devolve $10^n p_1 + 10^{n/2}(p_3 - p_1 - p_2) + p_2$

- O algoritmo de Karatsuba é correto?
  - Novamente, de forma simples: sim, pois ele claramente calcula a Equação 1[3].

[2] Veremos maneiras de formalizar esse tipo de análise durante o curso. Aqui, o correto seria fazer uma prova por indução no valor de n.

[3] EXERCÍCIO: prove por indução em n

- Quanto tempo ele leva?
  - Também não é óbvio contar o número de passos básicos que Karatsuba faz, mas veremos formas simples de fazer isso nas próximas aulas. Por hora, saiba que ele faz umas $en^{1.59}$ operações básicas, onde e é uma constante.
  - Note como $n^{1.59} < n^2$ para valores de n maiores do que 1. É claro que, dependendo de d e e, podemos ter $en^{1.59} > dn^2$: por exemplo, se $e = 15$ e $d = 2$, então $15n^{1.59} < 2n^2$ apenas quando $n > 136$. Mas note que como e e d são constantes (não dependem de n), sempre será possível dizer que $en^{1.59} < dn^2$ a partir de algum certo valor de n. Assim, realmente podemos dizer que Karatsuba é melhor do que Multiplica2 e Multiplica1.

> Note como existem várias formas de resolver o problema de multiplicação.
>
> E vimos apenas 3 delas (existem várias outras)!
>
> E estamos falando de um problema super simples que é multiplicar dois inteiros!

- Exercício: (CLRS) Para cada função $f(n) \in \{\log n, \sqrt{n}, n, n \log n, n^2, n^3, 2^n, n!\}$ e cada tempo $t \in \{$ 1 segundo, 1 minuto, 1 hora, 1 dia, 1 mês, 1 ano, 1 século$\}$, determine o maior valor de n tal que $f(n)$ microssegundos é no máximo t.

5 Por que analisar algoritmos?

- Basicamente, para poder comparar diferentes algoritmos que resolvem um mesmo problema e escolher qual utilizar sem antes ter que implementar todos.

6 Outro exemplo

- A sequência de Fibonacci é muito famosa.
- É a sequência infinita de números: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- Por definição, o n-ésimo número da sequência, escrito como F_n, é dado por

$$F_n = \begin{cases} 1 & \text{se } n = 1 \\ 1 & \text{se } n = 2 \\ F_{n-1} + F_{n-2} & \text{se } n > 2 \end{cases} \tag{2}$$

- Curiosidades:
  - $F_{30} > 1$ milhão
  - F_{100} é um número com 21 dígitos
  - Em geral, $F_n \approx 2^{0.694n}$

PROBLEMA: NÚMERO DE FIBONACCI

ENTRADA: inteiro $n \geq 0$.
SAÍDA: F_n.

- O Algoritmo 4 calcula, de maneira recursiva, F_n para um n dado como entrada (ele resolve o problema do número de Fibonacci descrito acima).

Algoritmo 4 Algoritmo recursivo para o problema do número de Fibonacci.

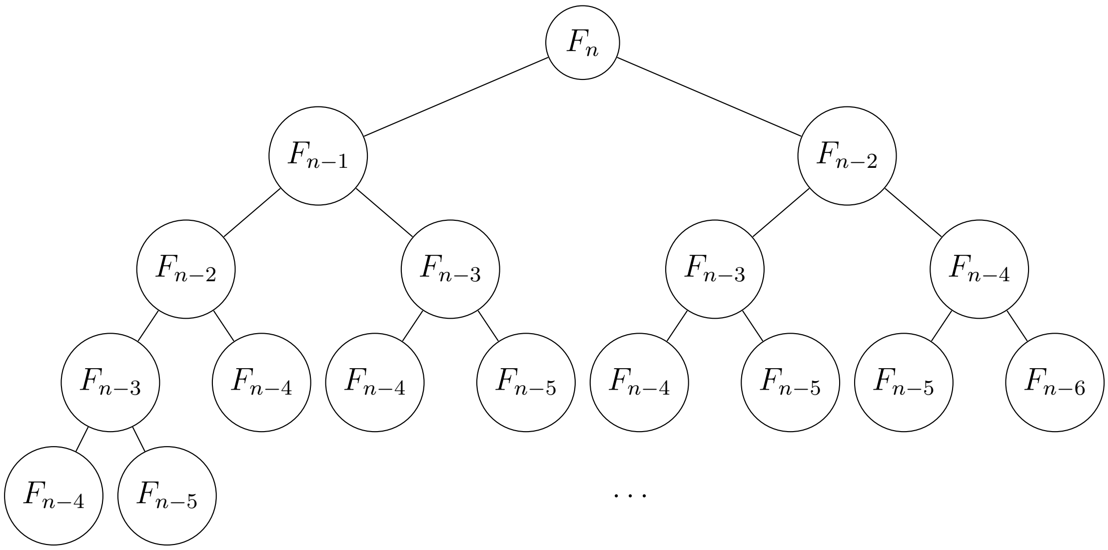
```
função FIBONACCI1(n)
    se n ≤ 2 então
        devolve 1
    a ← FIBONACCI1(n − 1)
    b ← FIBONACCI1(n − 2)
    devolve a + b
```

- Esse algoritmo é correto?
  - Claramente ele calcula F_n usando a Equação 2[4].
- Quanto tempo ele leva?
  - Seja $T(\star)$ o tempo (número de passos básicos) necessário para computar $F_\star$.
  - Se $n \leq 2$, então $T(n) \leq 2$ (foi feita uma comparação e um comando de retorno).
  - Se $n > 2$, então $T(n) = T(n-1) + T(n-2) + 3$ (foi calculado F_{n-1}, F_{n-2}, foi feita uma comparação, uma soma e um comando de retorno).
  - Mas então $T(n) \geq F_n$, ou seja, $T(n) \approx 2^{0.694n}$.
  - Na prática, o que isso significa?

	$n = 10$	$n = 100$	$n = 200$
Máquina 1: 4 bilhões de instruções por segundo	< 1s	≈ 6174 anos	$\approx 5 \times 10^{21}$ milênios
Máquina 2: 40 trilhões de instruções por segundo	< 1s	≈ 226 dias	$\approx 5 \times 10^{17}$ milênios

- Dá para fazer melhor?
  - Primeiro outra pergunta: por que será que o FIBONACCI1 é tão ruim?

[4] Use indução para formalizar essa prova.



- Porque existe muita repetição!

- O Algoritmo 5 mostra outra forma de resolver o problema do número de Fibonacci. Ele não é recursivo e faz uso de uma **estrutura de dados** para melhorar o tempo de execução.

Algoritmo 5 Algoritmo não recursivo para o problema do número de Fibonacci.

1: **função** FIBONACCI2(n)
2:     Seja F um vetor de tamanho n
3:     $F[1] \leftarrow 1$
4:     $F[2] \leftarrow 1$
5:     **para** $i \leftarrow 3$ até n **faça**
6:         $F[i] \leftarrow F[i-1] + F[i-2]$
7:     **devolve** $F[n]$

- Esse algoritmo é correto?
  - Ele também calcula F_n usando a Equação 2.
- Quanto tempo ele leva?
  - Note que as linhas 3, 4 e 7 fazem um número constante (≤ 2) de operações básicas. A linha 6 também, mas ela é executada quase n vezes. Vamos considerar que a linha 2 faz cerca de n operações também. Somando tudo, vemos que o número total de operações básicas é no máximo cn para alguma constante c.
  - Novamente, na prática, o que isso significa?

	$n = 10$	$n = 100$	$n = 200$	$n = 1000000$	$n = 10$ bilhões
Máquina 1	< 1s	< 1s	< 1s	< 1s	2.5s
Máquina 2	< 1s	< 1s	< 1s	< 1s	< 1s

- Dá para fazer melhor?
  - Sim!
- **Atenção**: A análise acima não está 100% correta de acordo com o que veremos durante o curso com relação ao número total de operações básicas. Note que acima dizemos

que a linha 6 faz um número constante de operações (isso é verdade: uma soma e uma atribuição), mas a soma feita ali pode não levar um número constante de *operações básicas* para ser realizada. É razoável imaginar que um número de 32 bits ou de 64 bits possa ser somado com outro rapidamente (os processadores atuais fazem isso), mas o n-ésimo número da sequência de Fibonacci precisa de uns $0.694n$ bits para ser armazenado e isso é bem maior do que 64 conforme n cresce.

- Essa análise não cuidadosa foi proposital (primeira aula, certo?). Note que mesmo com ela podemos ver a diferença entre os dois algoritmos para o problema do número de Fibonacci.
- Estritamente falando, o Algoritmo 4 recursivo faz cerca de F_n somas mas usa um número de passos básicos proporcional a nF_n. Já o Algoritmo 5 não recursivo faz cerca de n somas mas usa um número de passos básicos proporcional a n^2 (de onde vemos que ele continua sendo melhor do que o anterior).

7 O que veremos no curso?

- *V*ocabulário básico para projeto e análise de algoritmos (notação assintótica).
- Técnicas de projeto de algoritmos (divisão e conquista, algoritmos gulosos, programação dinâmica).
  - Não existem receitas prontas para solução de problemas em geral.
- Uso e análise de estruturas de dados.
  - Uma única estrutura não funciona bem para todos os propósitos.
- Problemas em NP-Completo e o que fazer com eles.
  - Em geral consideramos eficiente um algoritmo que seja rápido e veremos vários problemas para os quais bons algoritmos existem. Mas existem vários problemas para os quais não se conhece solução eficiente.