CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Técnicas de análise de algoritmos

Algoritmos e Estruturas de Dados I

Natália Batista
nataliabatista@cefetmg.br

1º semestre/ 2022

1. Análise de programas (1/5)

- Determinar o tempo de execução de um programa pode ser um problema matemático complexo.
- Determinar a ordem do tempo de execução, sem preocupação com o valor da constante envolvida, pode ser uma tarefa mais simples.

1. Análise de programas (2/5)

- A análise utiliza técnicas de matemática discreta, envolvendo contagem ou enumeração dos elementos de um conjunto:
  - manipulação de somas;
  - produtos;
  - permutações;
  - fatoriais;
  - solução de **equações de recorrência;**
  - etc.

1. Análise de programas (3/5)

Princípios

- Comando de atribuição, de leitura ou de escrita: O(1).
- Seqüência de comandos: determinado pelo maior tempo de execução de qualquer comando da seqüência.
- Comando de decisão: tempo dos comandos dentro do comando condicional, mais tempo para avaliar a condição, que é O(1).
- Anel: soma do tempo de execução do corpo do anel mais o tempo de avaliar a condição para terminação (geralmente O(1)), multiplicado pelo número de iterações.

1. Análise de programas (4/5)

- **Procedimentos não recursivos**:
  - Cada um deve ser computado separadamente, iniciando com os que não chamam outros procedimentos.
  - Avalia-se então os que chamam os já avaliados (utilizando os tempos calculados).
  - O processo é repetido até chegar no programa principal.

1. Análise de programas (5/5)

- **Procedimentos recursivos**:
  - É associada uma função de complexidade f(n) desconhecida, onde n mede o tamanho dos argumentos.
  - A seguir, obtém-se uma **equação de recorrência**:
    - Define-se a função por uma expressão envolvendo a mesma função.
  - Solução da equação fornece fórmula fechada para f(n).

2. Exemplo de análise (1/7)

- Algoritmo para ordenar os n elementos de um conjunto A em ordem ascendente.
  - Seleciona o menor elemento do conjunto.
  - Troca este elemento com A[1].
  - Repete as duas operações acima com os n – 1 elementos restantes, depois com os n - 2, até que reste apenas um.

```
 void ordena (int v[],int n){
   int i, j;
   for (i=0; i<n-1; i++){
     int min = i;
     for (j = i+1; j < n; j++)
       if (v[j] < v[min])
         min = j;
     //troca v[min] e v[i]
     int x = v[min];
     v[min] = v[i];
     v[i] = x;
  }
}
```

2. Exemplo de análise (3/7)

- Entrada de dados: número n de elementos do conjunto.
- O programa contém dois anéis:
  - a análise deve iniciar pelo anel interno.

2. Exemplo de análise (4/7)

- Anel interno
  - Contém um comando de decisão, com um comando apenas de atribuição.
  - Ambos levam tempo constante para serem executados.
  - O tempo para incrementar o índice do anel e avaliar sua condição de terminação é O(1).

2. Exemplo de análise (5/7)

- O tempo combinado para executar uma vez o anel é

$$O(\max(1, 1, 1)) = O(1),$$

conforme regra da soma para a notação O.

- Como o número de iterações é n-i-1, o tempo gasto no anel é

$$O((n-i-1)x1) = O(n - i),$$

conforme regra do produto para a notação O.

2. Exemplo de análise (6/7)

- Anel externo:
  - Contém, além do anel interno, quatro comandos de atribuição:

    O(max(1, (n - i), 1, 1, 1)) = O(n - i).

  - A linha (1) é executada n-1 vezes, e o tempo total para executar o programa está limitado ao produto de uma constante pelo **somatório** de (n - i):

$$\sum_{1}^{n-1}(n-i) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

(Ziviani)

2. Exemplo de análise (7/7)

- Se considerarmos o número de comparações (linha 4) como a medida de custo relevante, o programa faz $n^2/2 - n/2$ comparações para ordenar n elementos.
- Considerarmos o número de trocas (linhas 7, 8 e 9), o programa realiza exatamente n – 1 trocas.

3. Algoritmos recursivos (1/22)

Um objeto é recursivo quando é definido parcialmente em termos de si mesmo.

https://www.treinaweb.com.br/blog/desmistificando-os-algoritmos-recursivos/

3. Algoritmos recursivos (2/22)

- Exemplo 1: Números naturais
  - (a) 1 é um número natural
  - (b) o sucessor de um número natural é um número natural

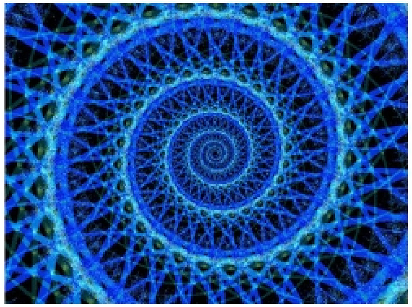
3. Algoritmos recursivos (3/22)

- Exemplo 2: Função fatorial
  - (a) $0! = 1$
  - (b) se $n > 0$ então $n! = n \times (n - 1)!$

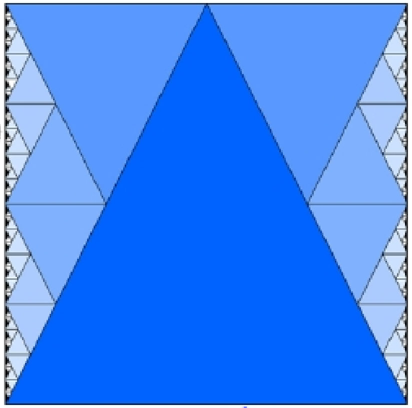
3. Algoritmos recursivos (4/22)

- Exemplos de objetos recursivos

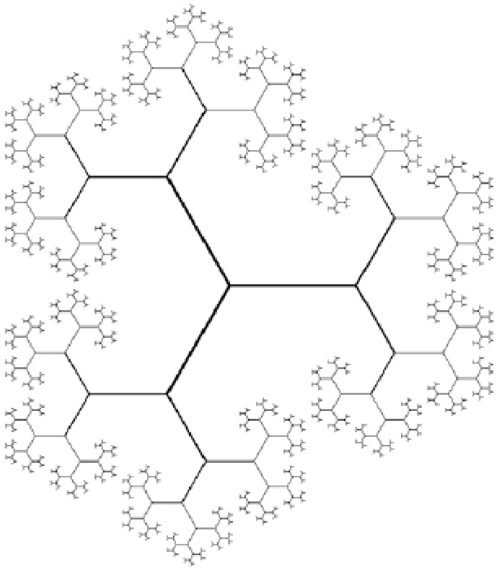
Fractal de Fern



Fractal espiral



Fractal de triângulos

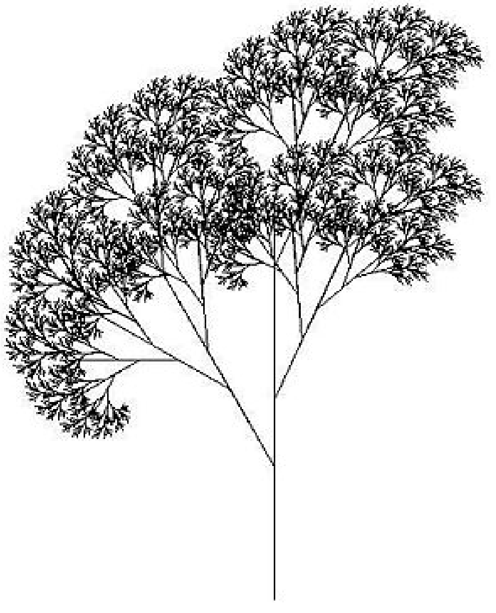


Derivação binária

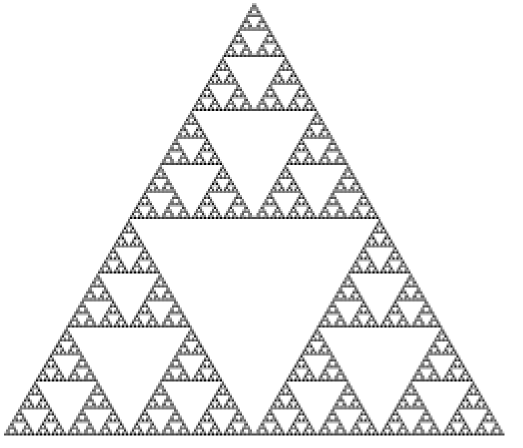
Fonte: internet.

3. Algoritmos recursivos (5/22)

- Fractais



https://en.wikipedia.org/wiki/Recursion_(computer_science)



https://pt.wikipedia.org/wiki/Recursividade

3. Algoritmos recursivos (6/22)

- Formas visuais recursivas

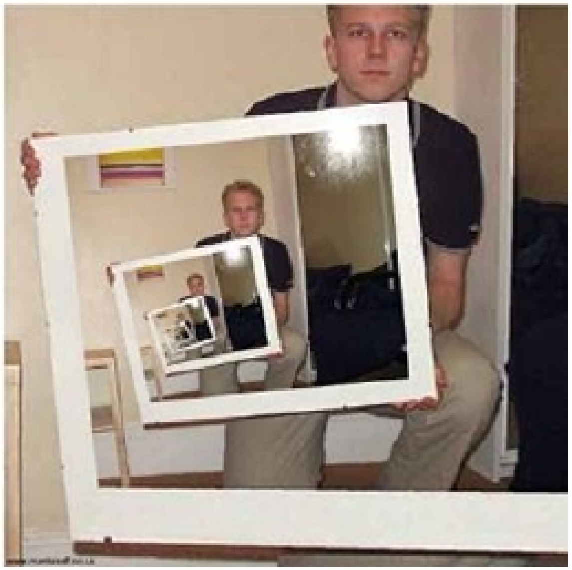
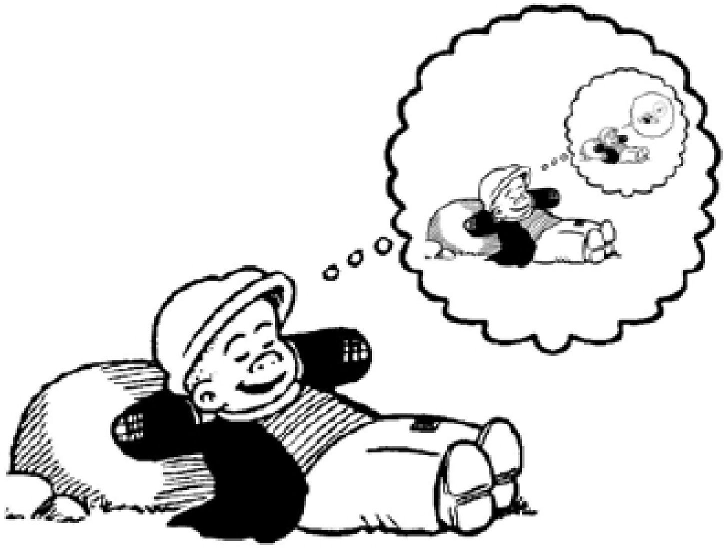


Foto recursiva

Imagem recursiva



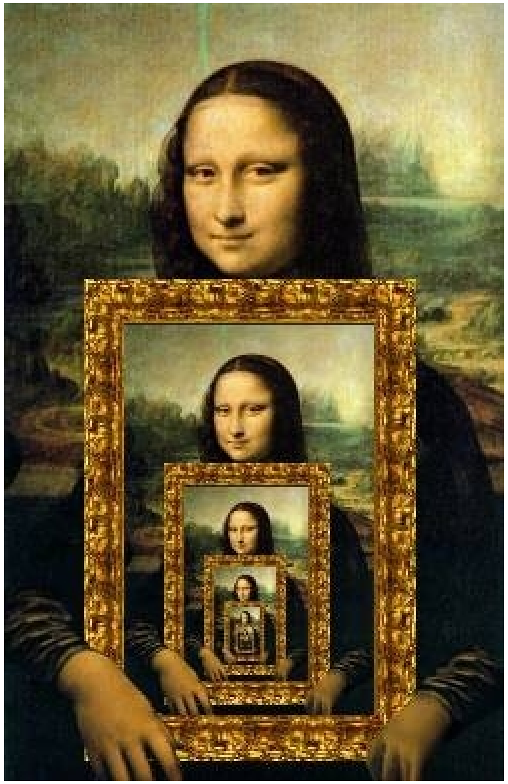
Pensamento recursivo

Fonte: internet.

3. Algoritmos recursivos (7/22)

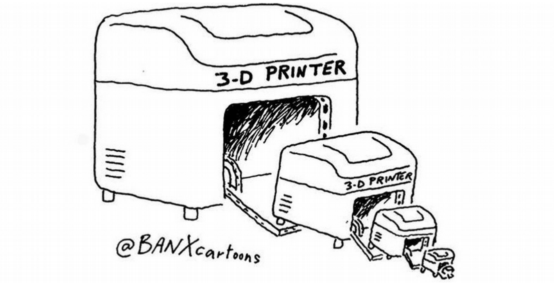
- Formas visuais recursivas

https://pt.wikipedia.org/wiki/Recursividade



http://jiangtanghu.com/blog/wp-content/uploads/2010/11/megamonalisa_recursion.jpg

https://www.todocoleccion.net/antiguedades/



http://wiki.secretgeek.net/unbounded-recursion

3. Algoritmos recursivos (8/22)

- Recursividade: consiste em definir um conjunto infinito de objetos através de um comando finito.
- Problema da terminação: os algoritmos recursivos têm que terminar.

3. Algoritmos recursivos (9/22)

- Um problema recursivo P pode ser expresso como

$$P \equiv \mathcal{P}[S_i, P],$$

onde *P* é a composição de comandos S_i e do próprio P

- Importante: constantes e variáveis locais a P são duplicadas a cada chamada recursiva.

3. Algoritmos recursivos (10/22)

Problema de terminação

- Definir uma condição de terminação.
- Ideia:
  - Associar um parâmetro, por exemplo n, com P e chamar P recursivamente com n - 1 como parâmetro.
  - A condição n > 0 garante a terminação.
  - Exemplo:

$$P(n) \equiv \textbf{if } n > 0 \textbf{ then } \mathcal{P}[S_i; P(n-1)].$$

3. Algoritmos recursivos (11/22)

Problema de terminação

- Importante: na prática é necessário
  - mostrar que o nível de recursão é finito, e
  - tem que ser mantido pequeno! Por quê?

3. Algoritmos recursivos (12/22)

- Razões para limitar a recursão:
  - Memória necessária para acomodar variáveis a cada chamada.
  - O estado corrente da computação tem que ser armazenado para permitir a volta da chamada recursiva.

3. Algoritmos recursivos (13/22)

- Exemplo: fatorial.

```
int F(int i){
  if (i>0)
    return i*F(i-1);
  return 1;
}
```

F(4) $\longrightarrow$

1	$4*F(3)$
2	$3*F(2)$
3	$2*F(1)$
4	$1*F(0)$

3. Algoritmos recursivos (14/22)

- Quando não usar recursividade
  - Algoritmos recursivos são apropriados quando o problema é definido em termos recursivos.
  - Entretanto, uma definição recursiva não implica necessariamente que a implementação recursiva é a melhor solução!

3. Algoritmos recursivos (15/22)

Logo,

$$P \equiv \texttt{if } B \texttt{ then } (S; P)$$

deve ser transformado em

$$P \equiv (x = x_0; \texttt{while } B \texttt{ do } S)$$

3. Algoritmos recursivos (16/22)

- Exemplo: fatorial iterativo.

```
int Fat(int n){
  int i, F;
  i = 0; F = 1;
  while(i < n){
    i = i + 1;
    F = F * i;
  }
  return F;
}
```

3. Algoritmos recursivos (17/22)

- Outro exemplo: Fibonacci.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

3. Algoritmos recursivos (18/22)

- Outro exemplo: Fibonacci.

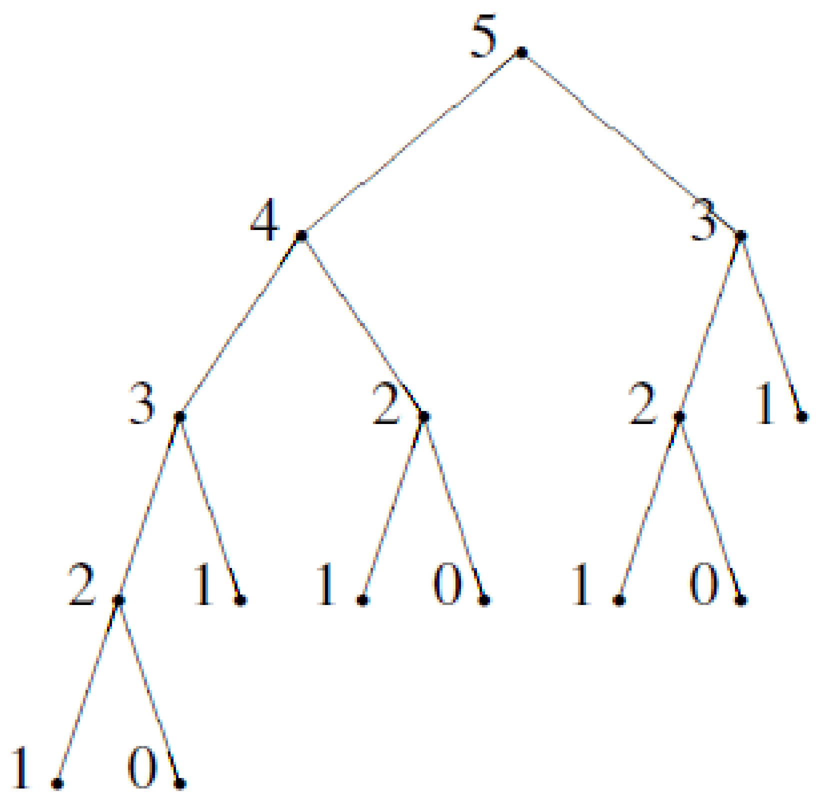
```
int Fib_rec(int n){
  if (n == 0)
    return 0;
  else  if (n == 1)
    return 1;

  return Fib_rec(n-1) + Fib_rec(n-2);
}
```

n é o número do termo, tal que n ≥ 0

3. Algoritmos recursivos (19/22)

- Observação: para cada chamada a Fib(n), Fib é ativada 2 vezes.



3. Algoritmos recursivos (20/22)

- Solução sem recursividade

```
int Fib(int n){
  int i, Temp, F, Fant;
  i = 1; F = 1; Fant = 0;
  if (n == 0)
    return 0;
  while(i < n){
    Temp = F;
    F = F + Fant;
    Fant = Temp;
    i = i + 1;
  }
  return F;
}
```

3. Algoritmos recursivos (21/22)

- Análise: número de adições.
- Solução sem recursividade:
  - Complexidade de tempo: $T(n) = O(n)$.
- Solução com recursividade:
  - Complexidade de tempo: $T(n) = O(\phi^n)$, onde ϕ é a razão de ouro (ϕ ~1,618).

3. Algoritmos recursivos (22/22)

- Comparação das versões recursiva e iterativa:

n	20	30	50	100
Recursiva	1 seg	2 min	21 dias	10^9 anos
Iterativa	1/3 mseg	1/2 mseg	3/4 mseg	1,5 mseg

(Ziviani)

4. Equações de recorrência (1/10)

- **Equação de recorrência**:
  - é uma equação (ou inequação) que descreve uma função em termos dela mesma em entradas de tamanho menor.

4. Equações de recorrência (1/10)

- Exemplo: algoritmo recursivo Pesquisa

```
    Pesquisa(n);
(1) if n ≤ 1
(2) then "inspecione elemento" e termine
    else begin
(3)         para cada um dos n elementos "inspecione elemento";
(4)         Pesquisa(n/3);
         end;
```

4. Equações de recorrência (2/10)

- Para cada procedimento recursivo é associada uma função de complexidade f(n) desconhecida, onde n mede o tamanho dos argumentos para o procedimento.
- Obtemos uma equação de recorrência para f(n).

4. Equações de recorrência (3/10)

- Seja T(n) uma função de complexidade que represente o número de inspeções nos n elementos do conjunto.
- O custo de execução das linhas 1 e 2 é O(1) e da linha 3 é O(n).
- Qual o custo de execução da linha 4?

```
    Pesquisa(n);
(1) if n ≤ 1
(2) then "inspecione elemento" e termine
    else begin
(3)         para cada um dos n elementos "inspecione elemento";
(4)         Pesquisa(n/3);
        end;
```

4. Equações de recorrência (4/10)

- Usa-se uma **equação de recorrência** para determinar o número de chamadas recursivas:
  - O termo T(n) é especificado em função dos termos anteriores T(1), T(2), ... , T(n - 1).
- No algoritmo Pesquisa:

$$T(n) = n + T(n/3), \;\; T(1) = 1$$

(para $n = 1$ fazemos uma inspeção)

4. Equações de recorrência (5/10)

- Por exemplo,

$$T(1) = 1$$

$$T(3) = T(3/3) + 3 = 4$$

$$T(9) = T(9/3) + 9 = 13$$

e assim por diante.

Pode-se assumir que n é uma potência de 3.

4. Equações de recorrência (6/10)

- **Fórmula fechada**: Substitui-se os termos T(k), k < n, até que todos os termos T(k), k > 1, tenham sido substituídos por fórmulas contendo apenas T(1).

$$
\begin{aligned}
T(n) &= n + T(n/3) \\
T(n/3) &= n/3 + T(n/3/3) \\
T(n/3/3) &= n/3/3 + T(n/3/3/3) \\
&\vdots \\
T(n/3/3 \cdots /3) &= n/3/3 \cdots /3 + T(n/3 \cdots /3)
\end{aligned}
$$

4. Equações de recorrência (7/10)

- Adicionando lado a lado:

$$T(n) = n + n \cdot (1/3) + n \cdot (1/3^2) + n \cdot (1/3^3) + \cdots + T(n/3/3 \cdots /3)$$

que representa a soma de uma série geométrica de razão 1/3, multiplicada por n, e adicionada de T(n/3/3 ... /3), que é menor ou igual a 1.

4. Equações de recorrência (8/10)

- Se considerarmos o termo T(n/3/3/3 ... /3) e denominarmos x o número de subdivisões por 3 do tamanho do problema, então

$$n/3^x = 1 \text{ e } n = 3^x.$$

- Logo $x = \log_3 n$.
- Lembrando que T(1) = 1, então

$$T(n/3^x) = 1$$

4. Equações de recorrência (9/10)

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{x-1} \frac{n}{3^i} + T(\frac{n}{3^x}) \\
&= n \sum_{i=0}^{x-1} (1/3)^i + 1 \\
&= \frac{n(1-(\frac{1}{3})^x)}{(1-\frac{1}{3})} + 1 \\
&= \frac{3n}{2} - \frac{1}{2}.
\end{aligned}
$$

x é o número de subdivisões do problema, logo x-1 é o expoente do último termo da P.G.

4. Equações de recorrência (10/10)

- Logo, o algoritmo Pesquisa é O(n).

4. Equações de recorrência (10/10)

- Métodos para resolver recorrências:
  - Expansão
  - Mudança de variáveis
  - Substituição (indução)
  - Árvores de recursão
  - Teorema mestre

Livro Ziviani - Capítulo 2

4. Equações de recorrência (10/10)

- Alguns somatórios úteis:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1}-1}{a-1}(a \neq 1)$$

$$\sum_{i=0}^{k} 2^{\mathrm{i}} = 2^{k+1} - 1$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

Soma dos termos de uma P.G. finita:

$$S_n = a_1 \cdot \frac{(1-q^n)}{(1-q)}$$

Referências

- Nivio Ziviani. Projeto de algoritmos: com implementações em Java e C++. 3 ed. Editora Cengage Learning, 2007.
- Antonio Alfredo Ferreira Loureiro. **Projeto e Análise de Algoritmos: Análise de Complexidade**. Notas de aula, 2010.