Análise da Complexidade de Algoritmos Recursivos I

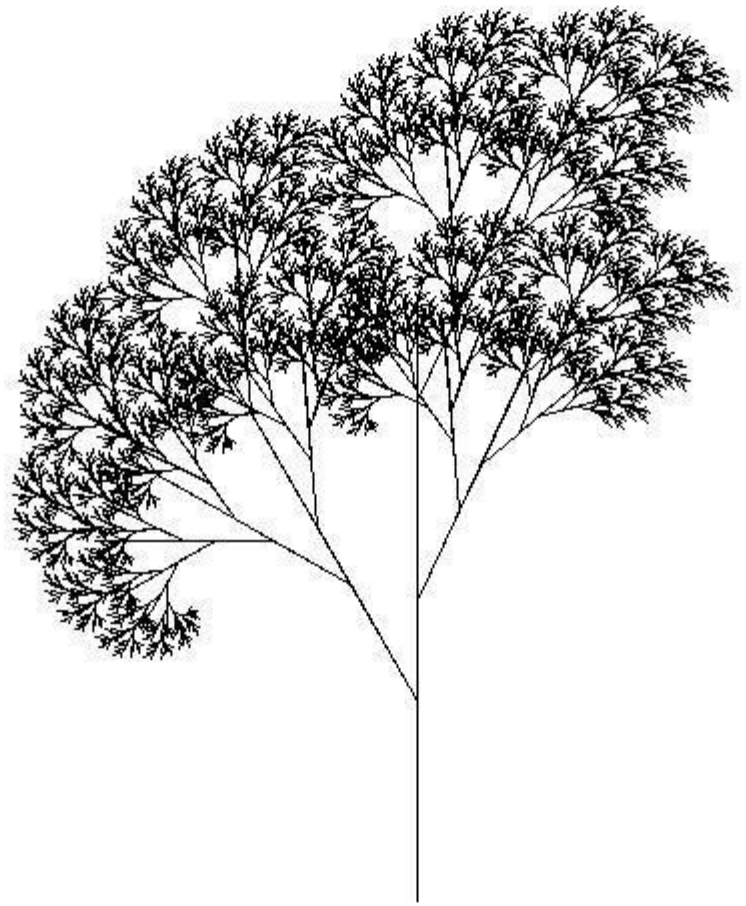
Joaquim Madeira

24/03/2020

Sumário

- Algoritmos recursivos
- Calcular $\boldsymbol{x^n}$
- Inverter a ordem dos elementos de um array
- Calcular o valor de um determinante
- As Torres de Hanói
- Exercícios adicionais
- Sugestões de leitura

Algoritmos recursivos



[Wikipedia]

Algoritmos recursivos

- Oferecem soluções concisas e elegantes
- MAS, nem sempre podem ser usados – EFICIÊNCIA
- Podem ser um primeiro passo para o desenvolvimento de um posterior algoritmo iterativo

- Decomposição do problema inicial em subproblemas mais simples e do mesmo tipo
  - Desenvolvimento Top-Down

Decomposição em subproblemas

- Diminuir-para-Reinar / Decrease-and-Conquer
  - Resolver 1 só subproblema em cada passo do processo recursivo
  - Lista / cadeia de chamadas recursivas

- Dividir-para-Reinar / Divide-and-Conquer
  - Resolver 2 ou mais subproblemas em cada passo do processo recursivo
  - Árvore de chamadas recursivas

Estratégia de decomposição

- Identificar o(s) caso(s) recursivo(s)
  - Problemas do mesmo tipo
  - Diminuição da "dificuldade"

- Identificar o(s) caso(s) de base / de paragem
  - São atingíveis?

$$n! = n \times (n - 1)!$$

$$0! = 1$$

Tipos de algoritmos – Recursividade Simples

- Diminuir-para-Reinar

- Executada 1 só chamada recursiva em cada passo

- Fatorial, mdc, travessia de um array / uma lista, procura binária, …

- Facilmente transformável num algoritmo iterativo, usando um ciclo

- Sugestão: implementar alguns destes algoritmos

Tipos de algoritmos – Recursividade Múltipla

- Dividir-para-Reinar

- Executadas 2 ou mais chamadas recursivas em cada passo

- Sucessão de Fibonacci, Combinações, …

- Usar STACK para transformar num algoritmo iterativo

- Sugestão: implementar alguns destes algoritmos

Eficiência

- Overhead associada a cada chamada recursiva
  - Salvaguarda do contexto
  - …

- MAS, nalguns casos, também ineficiência intrínseca
  - Recalcular inúmeras vezes os mesmos valores
  - Repetir as mesmas operações

- A estratégia de Programação Dinâmica é uma possível alternativa, para determinados problemas

Análise formal da complexidade

- Identificar a operação mais relevante
- Obter uma expressão recorrente para o número de operações efetuadas
- Se possível, desenvolver a expressão para obter uma "fórmula fechada"

- Vamos ilustrar / aprender analisando exemplos

Calcular $\boldsymbol{x^n}$

Calcular $\boldsymbol{x^n}$

$$x^n = x \times x^{n-1}, n > 0$$
$$x^0 = 1$$

```
double p(double x, unsigned int n) {
        if(n > 0) return x * p(x, n – 1);
        return 1;
}
```

Contar o número de multiplicações

$$M(n) = 1 + M(n-1), n > 0$$
$$M(0) = 0$$

- Desenvolvimento telescópico

$$M(n) = 1 + M(n-1) = 2 + M(n-2) = \cdots = k + M(n-k)$$
$$M(n) = n + M(0) = n$$

$$M(n) \in \mathcal{O}(n)$$

Calcular $\boldsymbol{x^n}$

- Há outros algoritmos recursivos para o cálculo de potências de expoente natural
- Por exemplo:

$$x^n = x^{\left\lfloor \frac{n}{2} \right\rfloor} \times x^{\left\lceil \frac{n}{2} \right\rceil}$$

- Quais são os casos de base?
- Qual é o número de multiplicações efetuadas?
- Sugestão: implementar e comparar

Inverter a ordem dos elementos de um array com n elementos

Inverter a ordem dos elementos

```
void inverter(int*  v, int esq, int dir) {
        if(esq < dir) {
                trocar(&v[esq], &v[dir]);
                inverter(v, esq + 1, dir – 1);
        }
}
```

Nº de trocas de elmentos?

$$T(n) = 1 + T(n-2), n > 2$$
$$T(2) = 1$$
$$T(1) = 0$$

$$T(n) = 1 + T(n-2) = 2 + T(n-4) = \cdots = k + T(n\ - 2k)$$

- Nº par de elementos vs Nº impar de elementos

Nº de trocas de elmentos?

$$T(n) = k + T(n - 2k)$$

- Seja o nº de elementos par e maior do que 2

$$n - 2k = 2 \Longrightarrow T(n) = \frac{n-2}{2} + T(2) = \frac{n}{2}$$

- Tarefa: fazer para n impar
- Verificar que para ambos os casos:

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor$$

Calcular o valor de um determinante usando o Teorema de Laplace

Exemplo – Desenvolver pela 1ª coluna

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 \times \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 4 \times \begin{vmatrix} 2 & 3 \\ 8 & 9 \end{vmatrix} + 7 \times \begin{vmatrix} 2 & 3 \\ 5 & 6 \end{vmatrix}$$

$$= 1 \times [5 \times 9 - 8 \times 6] - 4 \times [2 \times 9 - 8 \times 3] + 7 \times [2 \times 6 - 5 \times 3]$$

$$= 0$$

- Estratégia recursiva: decomposição em determinantes de menor dimensão

Um possível algoritmo recursivo

```
double Laplace( matriz A, unsigned int n) {
        …
        i(n == 1) return A[0][0];
        sinal = -1; soma = 0;
        for(i = 0; i < n; i++) {
                aux = subMatriz(A, i, 0);          // retira a 1ª coluna e a linha i
                sinal *= -1;
                soma += sinal * A[i][0] * Laplace(aux, n – 1);
        }
        return soma;
}
```

Nº de multiplicações efetuadas?

- $M(n) = \begin{cases} 0, \ n = 1 \\ 2 \times n + n \times M(n-1), \ n \geq 2 \end{cases}$

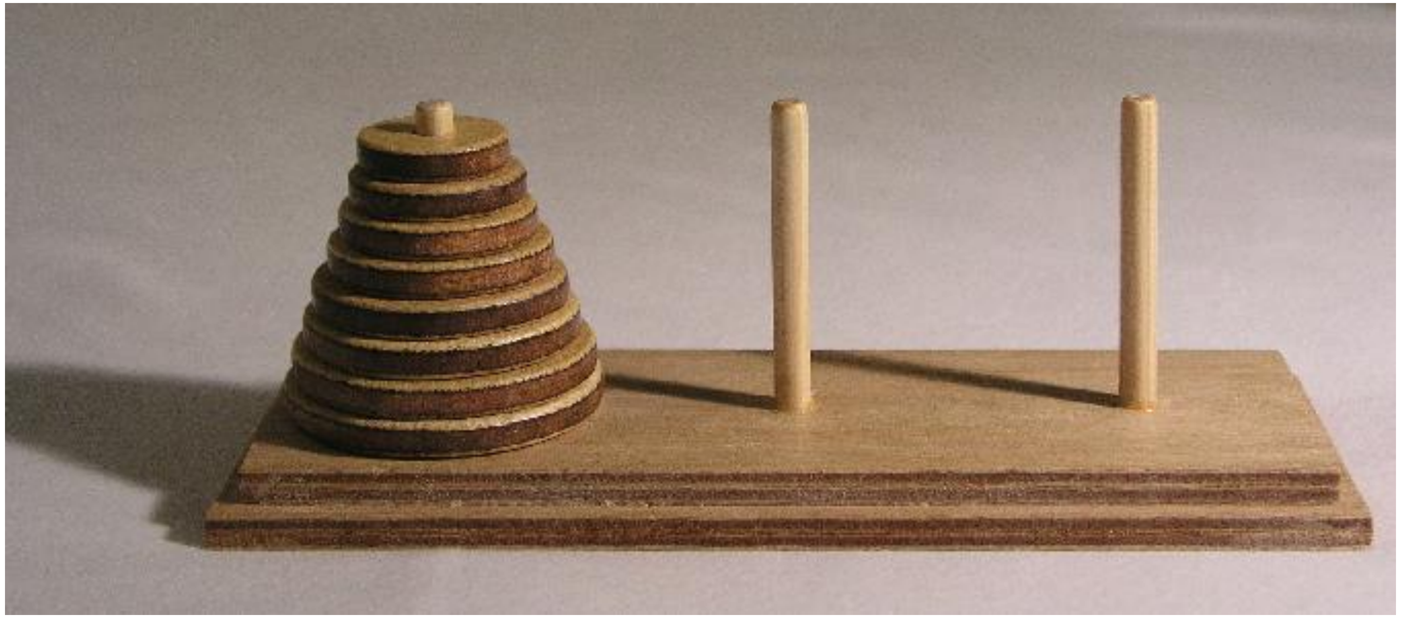
- n iterações do ciclo
- $2 \times n$ multiplicações explícitas
- n chamadas recursivas

Nº de multiplicações efetuadas?

- $M(n) = \begin{cases} 0, \; n = 1 \\ 2 \times n + n \times M(n-1), \; n \geq 2 \end{cases}$

- Não há uma “fórmula fechada” !!
- Verificar a rapidez com que cresce usando o Wolfram Alpha

- $M(n) \approx 2(e-1)n!$



[Wikipedia]

As Torres de Hanói

Função recursiva

- Fazer durante a sessão

Função recursiva

```c
// torresDeHanoi('A', 'B', 'C', 8);

void torresDeHanoi(char origem, char auxiliar, char destino, int n) {
  if (n == 1) {
    contadorGlobalMovs++;
    moverDisco(origem, destino);  // Imprime o movimento
    return;
  }
  // Divide-and-Conquer
  torresDeHanoi(origem, destino, auxiliar, n - 1);

  contadorGlobalMovs++;
  moverDisco(origem, destino);

  torresDeHanoi(auxiliar, origem, destino, n - 1);
}
```

Nº de movimentos realizados

- $M(1) = 1$
- $M(n) = M(n-1) + 1 + M(n-1) = 1 + 2\ M(n-1)$

- Fazer o desenvolvimento telescópico e obter “formula fechada”

- Verificar que obtêm um algoritmo EXPONENCIAL

Exercícios adicionais

Análise formal – Funções do próximo slide

- Obter uma expressão para o resultado de cada função
- Obter uma expressão para o nº de chamadas recursivas efetuadas

- Confirmar os resultados obtidos com o Wolfram Alpha

https://www.wolframalpha.com/

Resultado? – Nº de chamadas recursivas?

```
unsigned int
r1(unsigned int n) {
   if(n == 0) return 0;
   return 1 + r1(n – 1);
}
```

```
unsigned int
r2(unsigned int n) {
   if(n == 0) return 0;
   if(n == 1) return 1;
   return n + r2(n – 2);
}
```

```
unsigned int
r3(unsigned int n) {
   if(n == 0) return 0;
   return 1 + 2 * r3(n – 1);
}
```

```
unsigned int
r4(unsigned int n) {
   if(n == 0) return 0;
   return 1 + r4(n – 1) + r4(n – 1);
}
```

Sugestões de leitura

Sugestões de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
  - Capítulo 1: secções 1.5 e 1.6

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
  - Capítulo 2: secções 2.4 e 2.5
  - Apêndice B