Análise da Complexidade de Algoritmos Recursivos III

Joaquim Madeira

31/03/2020

Sumário

- Recap
- Multiplicação por partição: o Algoritmo de Karatsuba
- Ordenação por fusão: o Algoritmo Mergesort
- Exercícios adicionais
- Sugestões de leitura

Recapitulação

Problema da Moeda Falsa – Tentaram fazer ?

- Dadas n moedas visualmente idênticas
  - n par vs n impar
- Encontrar a moeda que é falsa !
- Usar apenas uma balança !
- A moeda falsa é mais leve do que uma genuína !
- Eficiência ?

Procura num Array – Tentaram fazer ?

- Dado um array de n elementos inteiros
- Encontrar a 1ª ocorrência do maior elemento!

- Divide-And-Conquer
  - Encontrar o maior do sub-array esq : ((n+1) div 2) elmentos
  - Encontrar o maior do sub-array dir : (n div 2) elementos
  - Comparar os dois valores e devolver o maior

- Quantas comparações?

Procura num Array – Nº de comparações

- C(1) = 0
- C(2) = 1
- C(n) = 1 + C(n div 2) + c((n + 1) div 2)

- Seja $n = 2^k$
- C(n) = 1 + 2 x C(n / 2) = ?
- Fazer o desenvolvimento telescópico ou …

The Master Theorem

- Dada uma recorrência, para $n = b^k$, $k \geq 1$

$$T(1) = c \qquad \text{e} \qquad T(n) = a\,T(n / b) + f(n)$$

com $a \geq 1$, $b \geq 2$, $c \geq 0$

- Se $f(n) = \Theta(n^d)$, com $d \geq 0$, então

$$T(n) = \Theta(n^d), \text{ se } a < b^d$$

$$T(n) = \Theta(n^d \log n), \text{ se } a = b^d$$

$$T(n) = \Theta(n^{\log_b a}), \text{ se } a > b^d$$

The Master Theorem

- Obtemos a ordem de complexidade, dada uma recorrência
  - Mas não obtemos uma expressão para o número de operações !
- Semelhante para as notações $O(n)$ e $\Omega(n)$
- Exemplo

$$C(n) = 2 \times C(n / 2) + 1$$

$$f(n) = 1,\ f(n) = \Theta(n^0),\ d = 0$$
$$a = 2,\ b = 2,\ a > b^d$$

$$C(n) = \Theta(n)$$

The Smoothness Rule

- Seja T(n) eventualmente não decrescente.
- Seja g(n) uma função suave ("smooth function").

- Se T(n) = Θ(g(n)) para valores de n que são potências de b, b ≥ 2.

- Então T(n) = Θ(f(n)), para qualquer n.

- Resultados análogos para O(n) e Ω(n) !!

- Boas notícias !!

Smooth Functions

- Função eventualmente não decrescente

$$f(n_1) \leq f(n_2), \text{ para quaisquer } n_2 > n_1 \geq n_0$$

- Função suave ("smooth")
  1) f(n) é eventualmente não decrescente
  2) $f(2n) = \Theta(f(n))$

- Exemplos
  - log n , n, n log n e n^k são funções suaves
  - a^n não é!!

Decrease-and-Conquer – Outro padrão

$$T(1) = b$$

$$T(n) = a \times T(n - c) + d$$

- $a = 1$

$$T(n) = b + d \times (n - 1) / c \qquad \boldsymbol{T(n) \in \mathcal{O}(n)}$$

- Fizeram o desenvolvimento ?

Divide-and-Conquer – Outro padrão

$T(1) = b$

$T(n) = a \times T(n - c) + d$

- $a > 1$

$$T(n) = d/(1 - a) + (b - d/(1 - a)) \times a^{(n-1)/c}$$

$$\boldsymbol{T(n) \in \mathcal{O}(a^{\frac{n}{c}})}$$

- Fizeram o desenvolvimento ?

Decrease-and-Conquer – Outro padrão

$$T(1) = c$$

$$T(n) = T(n / b) + f(n)$$

- $T(n) = \Theta(\log n)$, se $f(n) = \text{constante}$
- $T(n) = \Theta(n)$, se $f(n) = \Theta(n)$

Multiplicação por partição

Multiplicar números inteiros muito longos

- Como fazer?
  - E.g., mais do que 100 algarismos
- Muito maiores do que o maior inteiro representável !
- Usar o algoritmo que aprendemos na escola ?
- Divide-and-Conquer ?
- Eficiência ?

Algoritmo da escola primária

- Algoritmo para multiplicação de 2 números com n algarismos
- Quantos algarismos tem o resultado?
- 34 x 12 = ?

Algoritmo da escola primária

- 34 x 12 = ?

- Eficiência ?
  - 4 multiplicações algarismo a algarismo : $\Theta(n^2)$
  - 1 deslocamento (“shift”)
  - 3 adições
    - Podem ser mais ?

- Será possível reduzir o número de multiplicações ?

Divide-and-Conquer

- Ideia :
  - Partir cada número em duas metades
  - Multiplicar recursivamente essas metades
  - Compor o resultado final

- Questões
  - Quando parar o processo recursive ?
  - Como organizar / multiplicar ?
  - Como obter o resultado final ?

Divide-and-Conquer

- 1ª tentativa!
- 34 x 12 = ?
- Subdividir
  - a = 3 ; b = 4
  - c = 1 ; d = 2
- Multiplicar
  - a x c = 3 ; a x d = 6
  - b x c = 4 ; b x d = 8
- Obter o resultado final

$$34 \times 12 = (a \times c) \cdot 10^2 + (a \times d + b \times c) \cdot 10^1 + (b \times d) \cdot 10^0$$

Divide-and-Conquer

- 34 x 12 = ?
  - 4 multiplicações algarismo a algarismo
  - 2 deslocamentos
  - 3 adições

- Caso geral ?
  - 4 multiplicações de números com n/2 algarismos
  - $M(n) = 4\ M\ (n / 2)$, if n is par
  - Complexidade ? $\Theta(n^2)$!!

- Podemos fazer melhor ?

Divide-and-Conquer – Algoritmo de Karatsuba

- 2ª tentativa!
- 34 x 12 = ?
- Subdividir
  - a = 3 ; b = 4
  - c = 1 ; d = 2
- Multiplicar
  - **u = (a – b) x (d – c) = -1**
  - **v = a x c = 3**
  - **w = b x d = 8**

Divide-and-Conquer – Algoritmo de Karatsuba

- Resultado final
  - 34 x 12 = $v \cdot 10^2 + (u + v + w) \cdot 10^1 + w \cdot 10^0$
- 34 x 12 = ?
  - 3 multiplicações algarismo a algarismo
  - 2 deslocamentos
  - 6 adições / subtrações
- Caso geral ?
  - 3 multiplicações de números com n/2 algarismos
  - M(n) = 3 M (n / 2), if n é par
  - Complexidade ? $\Theta(n^{\log_2 3}) = \Theta(n^{1.585})$!!

Divide-and-Conquer – Algoritmo de Karatsuba

- X x Y = ?
- Subdividir
  - $X = a \cdot 10^{n/2} + b$
  - $Y = c \cdot 10^{n/2} + d$
- Multiplicar
  - u = (a – b) x (d – c)
  - v = a x c
  - w = b x d
- Obter o resultado
  - $X \times Y = v \cdot 10^{n} + (u + v + w) \cdot 10^{n/2} + w$

Divide-and-Conquer – Algoritmo de Karatsuba

- Quando parar o processo recursivo?
  - Números com 1 algarismo
  - OU quando o nº de algarismos for suficientemente pequeno !

- Eficiência ?
  - Para números moderadamente grandes, povelmente exige maior tempo de cálculo que o algoritmos da escola primária
  - Mas, verificou-se ser mais eficiente para multiplicar números com mais do que 600 algarismos !!

Divide-and-Conquer – Algoritmo de Karatsuba

- Vamos fazer !!

Divide-and-Conquer – Algoritmo de Karatsuba

```
// KARATSUBA - PSEUDO-CÓDIGO

BigNum multRec(BigNum X, BigNum Y) {
  // CASO DE BASE

  if (numDigitos(X) < LIMIAR && numDigitos(Y) < LIMIAR) {
    return X * Y;
  }

  int n = numDigitos(maior(X, Y));

  // TAMANHO DA METADE

  n /= 2;
```

Divide-and-Conquer – Algoritmo de Karatsuba

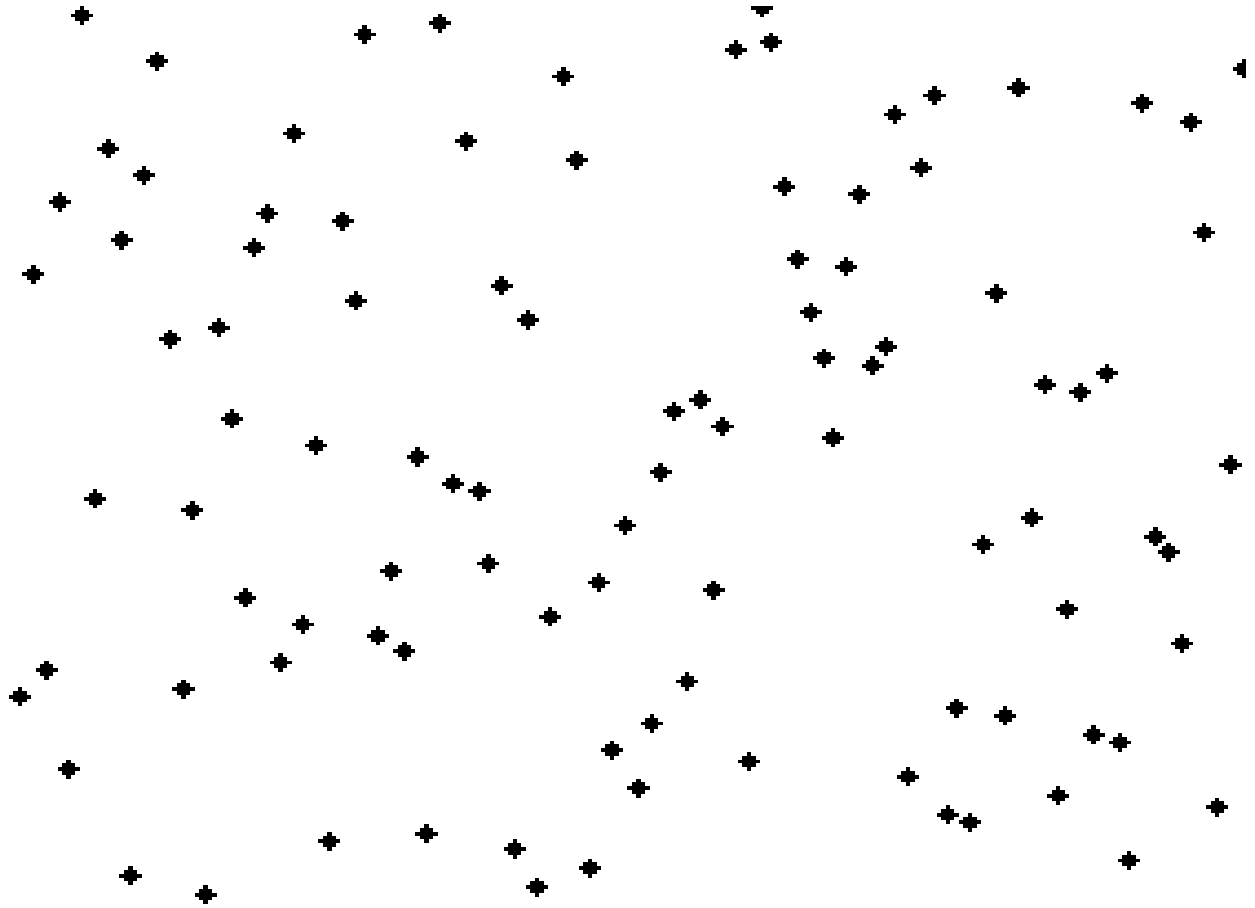
```
// SUBDIVIDIR

BigNum xHIGH = deslocarDireita(X, n);

BigNum xLOW = X - deslocarEsquerda(xHIGH, n);

BigNum yHIGH = deslocarDireita(Y, n);

BigNum yLOW = Y - deslocarEsquerda(yHIGH, n);

// RESULTADOS PARCIAIS
```

Divide-and-Conquer – Algoritmo de Karatsuba

```
BigNum p1 = multRec(xHIGH, yHIGH);

BigNum p2 = multRec(xLOW, yLOW);

BigNum p3 = multRec(xHIGH - xLOW, yHIGH - yLOW);

// RESULTADO FINAL

return deslocarEsquerda(p1, 2 * n) + deslocarEsquerda(p1 + p2 + p3, n) + p2;
}
```

Mergesort
– Ordenação por Fusão

Mergesort



[Wikipedia]

Mergesort

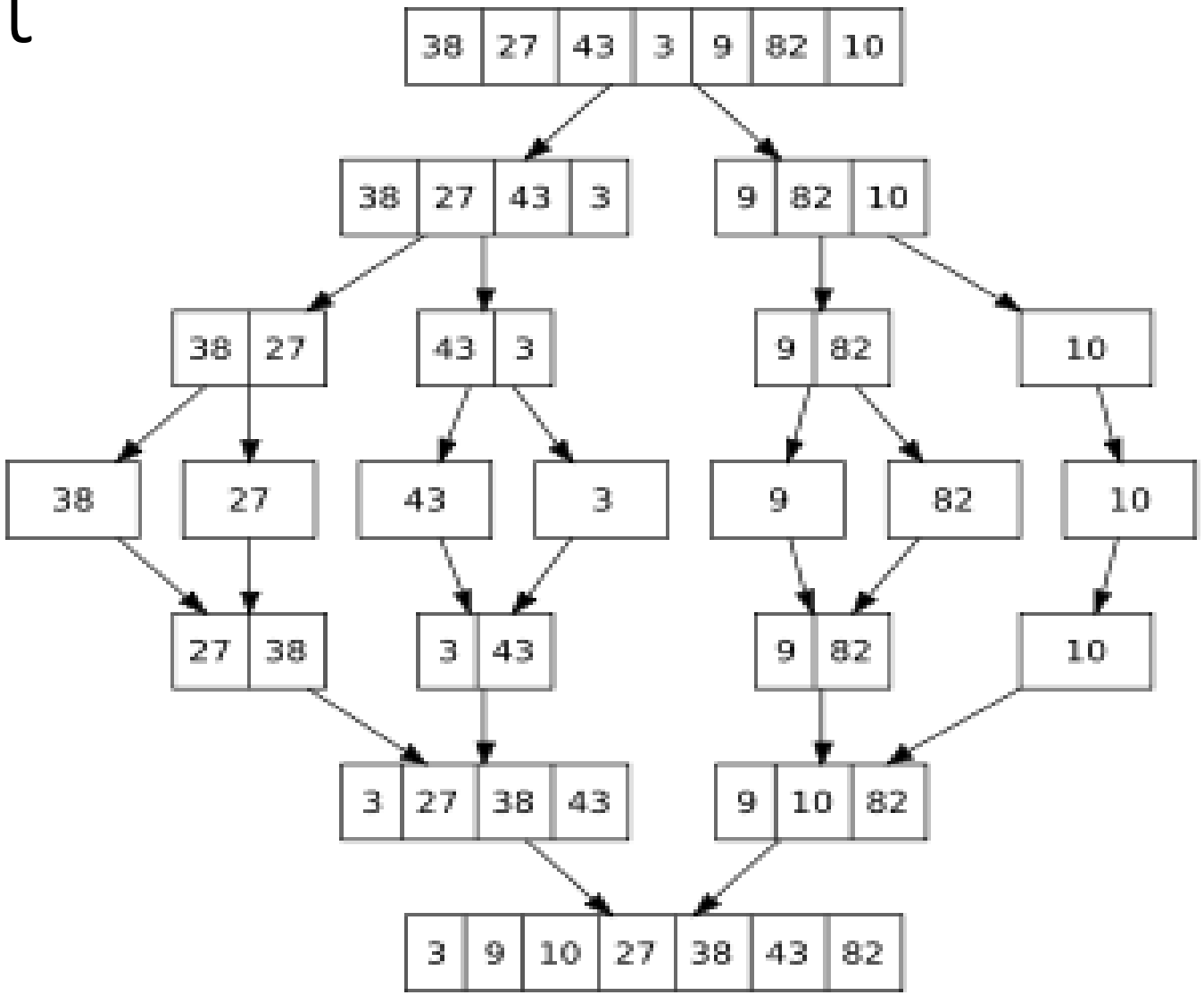
[Wikipedia]

Mergesort

- Ordenar um array / lista
  - Se o tamanho é 0 ou 1, já está ordenada
  - Caso contrário, subdividir em duas “metades”
    - Aprox. do mesmo tamanho !!
  - Ordenar recusivamente cada “metade”
  - Fundir as duas “metades” ordenadas num só array / lista

- Questão : usar ou não um array adicional ?

Mergesort



[Wikipedia]

Tarefa: associar a cada seta um rótulo que identifica a sequência pela qual as chamadas são executadas

Mergesort

- Vamos fazer !!

Mergesort

```
// mergeSort(A, tmpA, 0, n - 1);

void mergeSort(int* A, int* tmpA, int left, int right) {
  // Mais do que 1 elemento ?

  if (left < right) {
    int center = (left + right) / 2;

    mergeSort(A, tmpA, left, center);
    mergeSort(A, tmpA, center + 1, right);
    merge(A, tmpA, left, center + 1, right);
  }
}
```

Mergesort

```
void merge(int* A, int* tmpA, int lPos, int rPos, int rEnd) {
  int lEnd = rPos - 1;
  int tmpPos = lPos;
  int nElements = rEnd - lPos + 1;

  // COMPARAR O 1o ELEMENTO DE CADA METADE
  // E COPIAR ORDENADAMENTE PARA O ARRAY TEMPORÁRIO

  while (lPos <= lEnd && rPos <= rEnd) {
    if (A[lPos] <= A[rPos])
      tmpA[tmpPos++] = A[lPos++];
    else
      tmpA[tmpPos++] = A[rPos++];
  }
```

Mergesort

```
  // SOBRA, PELO MENOS, 1 ELEMENTO NUMA DAS METADES

  while (lPos <= lEnd) {···
  }

  while (rPos < rEnd) {···
  }

  // COPIAR DE VOLTA PARA O ARRAY ORIGINAL

  for (int i = 0; i < nElements; i++, rEnd--) {
    A[rEnd] = tmpA[rEnd];
  }
}
```

Mergesort

- Eficiência ?
- Todas as comparações são feitas pela função de fusão
- Escrever a recorrência !
- Melhor caso / Pior caso / Caso médio ?
- O(n log n) !!

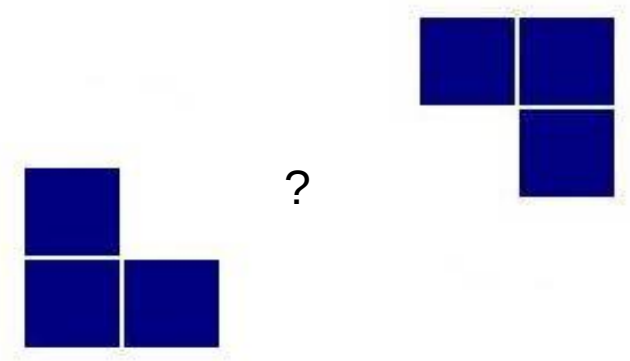
Mergesort

- Vamos fazer !!

Exercício adicional

Preencher um tabuleiro com trominós

- Preencher um tabuleiro com trominós em forma de L
  - Tabuleiro de tamanho n x n, com $n = 2^k$
- MAS, uma qualquer das casas está ocupada !
- Como preencher as outras casas ?
- Experimentem!!
  - http://www.cut-the-knot.org



[Wikipedia]

Preencher um tabuleiro com trominós

- Ideia
  - Subdividir em instâncias mais pequenas e semelhantes
    - (n / 2) x (n / 2)
  - Ocupar uma casa em 3 dos sub-tabuleiros
    - Agora temos 4 sub-problemas do mesmo género !!
  - Proceder de modo recursivo
- Casos de base ?

Preencher um tabuleiro com trominós

- Como representar o tabuleiro ?
  - Casas vazias ?
  - Casas ocupadas ?
- Como representar / colocar um trominó ?
  - Localização ?
  - Orientação ?
- Eficiência
  - Tempo proporcional ao nº de trominós colocados
  - Complexidade ?

Sugestões de leitura

Sugestões de leitura

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2012
  - Capítulo 5: secção 5.1
  - Capítulo 5: secção 5.4
  - Apêndice B