Análise e Técnicas de Algoritmos

Jorge Figueiredo

Divisão e Conquista

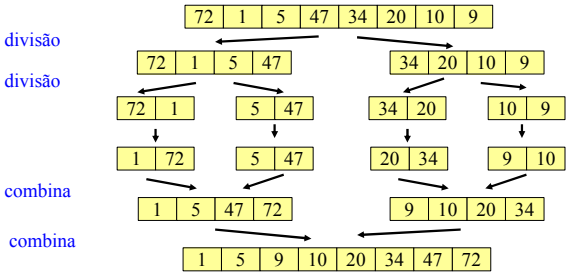
Agenda

- Conceitos Básicos
- *Template* Genérico
- Exemplos

Motivação

- Pegar um problema de entrada grande.
- Quebrar a entrada em pedaços menores (DIVISÃO).
- Resolver cada pedaço separadamente. (CONQUISTA)
  - Como resolver os pedaços?
- Combinar os resultados.

MergeSort



A Técnica

A técnica de divisão e conquista consistem de 3 passos básicos:

1. **Divisão:** Dividir o problema original, em subproblemas menores.
2. **Conquista:** Resolver cada subproblema recursivamente.
3. **Combinação:** Combinar as soluções encontradas, compondo uma solução para o problema original.

A Técnica

- Algoritmos baseados em divisão e conquista são, em geral, recursivos.
- A maioria dos algoritmos de divisão e conquista divide o problema em *a* subproblemas da mesma natureza, de tamanho *n/b*.
  - T(n) = a. T(n/b) + g(n)
  - Teorema Master para fazer análise.
- Vantagens:
  - Requer um número menor de acessos à memória.
  - São altamente *paralelizáveis*. Se existem vários processadores disponíveis, a estratégia propicia eficiência.

Quando Utilizar DeC?

Existem três condições que indicam que a estratégia de divisão e conquista pode ser utilizada com sucesso:

1. Deve ser possível decompor uma instância em sub-instâncias.
2. A combinação dos resultados deve ser eficiente.
3. As sub-instâncias devem ser mais ou menos do mesmo tamanho.

Quando Utilizar DeC?

É possível identificar pelo menos duas situações genéricas em que a abordagem por divisão e conquista é adequada:

1. Problemas onde um grupo de operações são correlacionadas ou repetidas. A multiplicação de matrizes, que veremos a seguir, é um exemplo clássico.
2. Problemas em que uma decisão deve ser tomada e, uma vez tomada, quebra o problema em peças disjuntas. Em especial, a abordagem por divisão-e-conquiasta é interessante quando algumas peças passam a ser irrelevantes.

Algoritmo Genérico

```
DivisãoeConquista(x)
if x é pequeno ou simples do
      return resolver(x)
else
      decompor x em conjuntos menores x0, x1, ... xn
      for i ← 0 to n do
            yi ← DivisãoeConquista(xi)
            i ← i +1
      combinar yi's
      return y
```


Exemplo 1

- O problema consiste em encontrar o maior elemento de um array A[1..n]

Solução Ingênua

```
Maxim(A[1..n])
max ← A[1]
for i ← 2 to n do
      if A[j] > max then
            max ← A[i]
return max
```


Exemplo 1

- O problema consiste em encontrar o maior elemento de um array A[1..n]

Solução Ingênua

```
Maxim(A[1..n])
maior ← A[1]
for i ← 2 to n do
      if A[j] > maior then
            maior ← A[i]
return maior
```

Solução DeC

```
Maxim(A[x..y])
if x – y ≤ 1 then
      return max(A[x], A[y])
else
      m ← x+y/2
      v1 ← Maxim(A[x..m])
      v2 ← Maxim(A[m+1..y])
return max(v1, v2)
```


Exemplo 2

- O problema consiste em computa a^n, em que $n \in N$.

Solução Ingênua

```
Pow(a, n)
p ← a
for i ← 2 to n do
      p ← p × a
return p
```

Solução DeC

```
Pow(a, n)
if n=0 then
      return 1
if n é par then
      return Pow(a, n/2) × Pow(a, n/2)
else
      return Pow(a,n-1/2) × Pow(a, n-1/2) × a
```

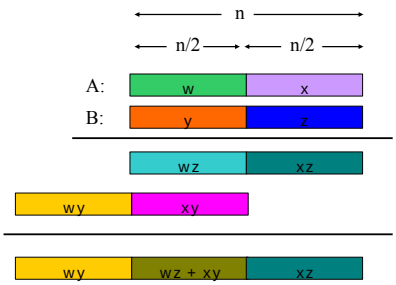
Multiplicação de Inteiros Grandes

- O problema consiste em multiplicar dois números inteiros grandes.
- A multiplicação clássica (a que aprendemos fazer na escola) requer tempo $\Theta(n^2)$. Isso porque fazemos multiplicação dígito a dígito.

Solução Alternativa por Divisão e Conquista

- Para evitar maiores complicações, vamos assumir que o número de dígitos em cada número é potência de 2.
- A multiplicação de um número A por um número B pode ser efetuada dividindo-se o número original em dois super-dígitos e procedendo a multiplicação .

Multiplicação de Inteiros Grandes



Multiplicação de Inteiros Grandes

$Mult(A,B) = Mult(w,y).10^{2m} + (Mult(w,z)+Mult(x,y)).10^{m} + Mult(x,z)$

- A multiplicação por 10^m pode ser vista como o deslocamento de m posições para a direita.
- As adições envolvidas tomam tempo $\Theta(n)$ cada.
- A multiplicação de dois inteiros longos é o resultado de 4 produtos de inteiros de tamanho metade do valor original, e um constante número de adições e deslocamentos, com tempo $\Theta(n)$.

Multiplicação de Inteiros Grandes

Uma solução por divisão e conquista:

1. **Divisão:** Dividir cada número em dois números com metade da quantidade de dígitos.
2. **Conquista:** Proceder a multiplicação das quatro partes.
3. **Combinação:** Combinar os resultados com deslocamento e adições.

A análise do algoritmo que utiliza divisão e conquista requer a solução da seguinte relação de recorrência:

- $T(n) = 4.T(n/2) + \Theta(n)$
- $\Theta(n^2)$

Multiplicação de Inteiros Grandes

Uma Solução por Divisão e Conquista Mais Eficiente

- Por que não temos a eficiência desejada? Ora, temos 4 multiplicaçõs de números de tamanho n/2.
- A solução seria reduzir o número de multiplicações? Isso é verdade pois sabemos que a adição e deslocamentos contribui com $\Theta(n)$.
- Se observarmos mais detalhadamente, podemos reduzir para três o número de multiplicações:
  - C= wy
  - D= xz
  - E= (wz + xy)

Multiplicação de Inteiros Grandes

- C= Mult(w,y)
- D= Mult(x,z)
- E= Mult((w+x, y+z)) – C – D = (wy+wz+xy+xz)-wy-xz=(wz + xy)
- Logo, $Mult(A,B) = C.10^{2m} + E.10^{m} + D$

- No total, fazemos 3 multiplicações, 4 adições e 2 subtrações de números com n/2 dígitos. É necessário ainda fazer deslocamentos mas, tudo isso representa $\Theta(n)$.
- $T(n) = 3.T(n/2) + \Theta(n)$
- T(n) é $\Theta(n^{1.585})$

Multiplicação de Matrizes

•Objetivo é multiplicar duas matrizes n×n.
•Por exemplo, no caso de n=2, é necessário efetuar 8 multiplicações. (2^x, em que $x = \log_2 8$)

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$C_{11} = a_{11}b_{11} + a_{12}b_{21}$
$C_{12} = a_{11}b_{12} + a_{12}b_{22}$
$C_{21} = a_{21}b_{11} + a_{22}b_{21}$
$C_{22} = a_{21}b_{12} + a_{22}b_{22}$

Algoritmo de Strassen

- Strassen mostrou que um multiplicação de matrizes 2x2 pode ser feita com 7 multiplicações e 18 operações de adição e subtração.**($2^{\log_2 7} = 2^{2.807}$)**
- Redução feita usando divisão e conquista.

A × B = R

A_0	A_1
A_2	A_3

×

B_0	B_1
B_2	B_3

=

$A_0 \times B_0 + A_1 \times B_2$	$A_0 \times B_1 + A_1 \times B_3$
$A_2 \times B_0 + A_3 \times B_2$	$A_2 \times B_1 + A_3 \times B_3$

Algoritmo de Strassen

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11} + A_{22})(B_{11}+B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

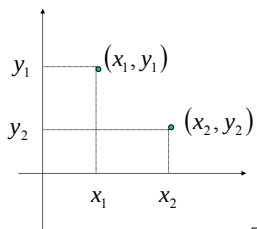
$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

Algoritmo de Strassen

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 \\ &= (A_{11} + A_{22})(B_{11}+B_{22}) + A_{22} * (B_{21} - B_{11}) - (A_{11} + A_{12}) * B_{22} + \\ &\quad (A_{12} - A_{22}) * (B_{21} + B_{22}) \\ &= A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22} + A_{22}B_{21} - A_{22}B_{11} - \\ &\quad A_{11}B_{22} - A_{12}B_{22} + A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22} \\ &= A_{11}B_{11} + A_{12}B_{21} \end{aligned}$$

Menor Distância Entre Pontos

- Distância Euclideana

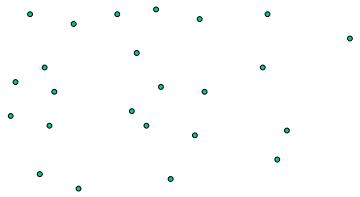


$$\|(x_1, y_1) - (x_2, y_2)\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

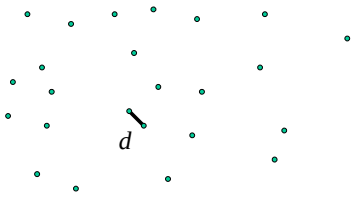
Menor Distância Entre Pontos

Closest Pair
Entrada: Um conjunto de pontos n $\mathcal{P} = \langle p_1, p_2, ..., p_n \rangle$, em duas dimensões.
Saída: O par de pontos p_1 e p_2 que apresenta a menor distância euclideana.

Menor Distância Entre Pontos



Menor Distância Entre Pontos

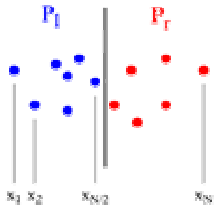


Menor Distância Entre Pontos

- Solução Força Bruta é $O(n^2)$.
- Vamos assumir:
  - Não existem pontos com a mesma coordenada x.
  - Não existem pontos com a mesma coordenada y.
- Como resolver este problema considerando 1D?
- É possível aplicar Divisão e Conquista?

Menor Distância Entre Pontos – 2D

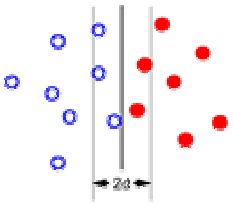
- Como dividir em sub-problemas?
  - Ordenar de acordo com a coordenada x e dividir em duas partes: esquerda e direita.



Menor Distância Entre Pontos – 2D

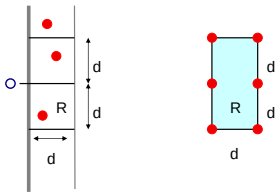
- Resolver recursivamente cada sub-problema, obtendo d_l e d_r.
- O que podemos observar?
  - Já temos a menor distância em cada uma das partes.
  - Fazer d= min{d_l , d_r}.
  - Falta analisar distância entre pontos de sub-problemas distintos.
  - Devemos analisar todos os casos?
    - Somente pontos que se encontram em uma faixa de tamanho 2d em torno da linha divisória.

Menor Distância Entre Pontos – 2D



Menor Distância Entre Pontos – 2D

- Qual a quantidade de pontos que se encontram dentro da faixa de tamanho 2d?
  - Se considerarmos um **p** ∈ P_l, todos os pontos de P_r que devem ser considerados devem estar em um retângulo **R** de dimensões **d** × **2d**.



Menor Distância Entre Pontos – 2D

- Como determinar os seis pontos?
  - Projeção de pontos nos eixos ***x*** e ***y***.
  - Pode-se fazer isso para todo **p** ∈P_l **e** P_r, em O(n) (pontos ordenados).
- Relação de recorrência é **T(n)= 2.T(n/2) + O(n)**
  - Sabemos que isso é **O(n.log n)**

ClosestPair(P)

Pré-processamento
Construir P_x e P_y como listas ordenadas pelas coordenadas **x** e **y**

Divisão
Quebrar P em P_l e P_r

Conquista
d_l = ClosestPair(P_l)
d_r = ClossetPair(P_r)

Combinação
d= min{d_l, d_r}
Determinar faixa divisória e pontos
Verificar se tem algum par com distância < d