Projeto e Análise de Algoritmos

Prof. Dr. Ednaldo B. Pizzolato

www.dc.ufscar.br/~ednaldo

ednaldo@dc.ufscar.br

Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Divisão e Conquista

- Closest-pair
- Convex-hull
- Exercícios
- Resumo

Introdução

- Divisão e conquista ou dividir para conquistar é, provavelmente, a técnica de projeto de algoritmos mais bem conhecida. Alguns dos mais eficientes algoritmos utilizam esta estratégia.

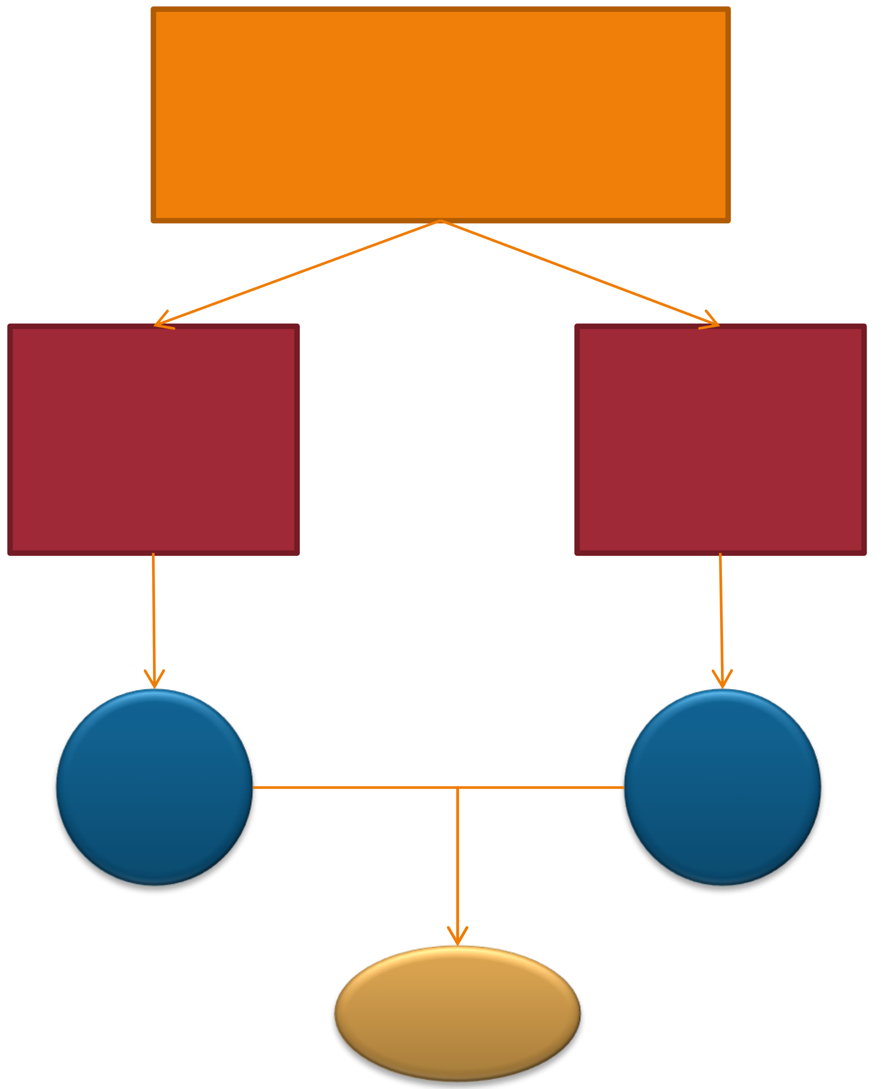
Introdução

- Os 10 mais importantes algoritmos do século 20
  - Metropolis Algorithm for Monte Carlo
  - Simplex Method for Linear Programming
  - Krylov Subspace Iteration Methods
  - The Decompositional Approach to Matrix Computations
  - The Fortran Optimizing Compiler
  - QR Algorithm for Computing Eigenvalues
  - Quicksort Algorithm for Sorting
  - Fast Fourier Transform
  - Integer Relation Detection
  - Fast Multipole Method

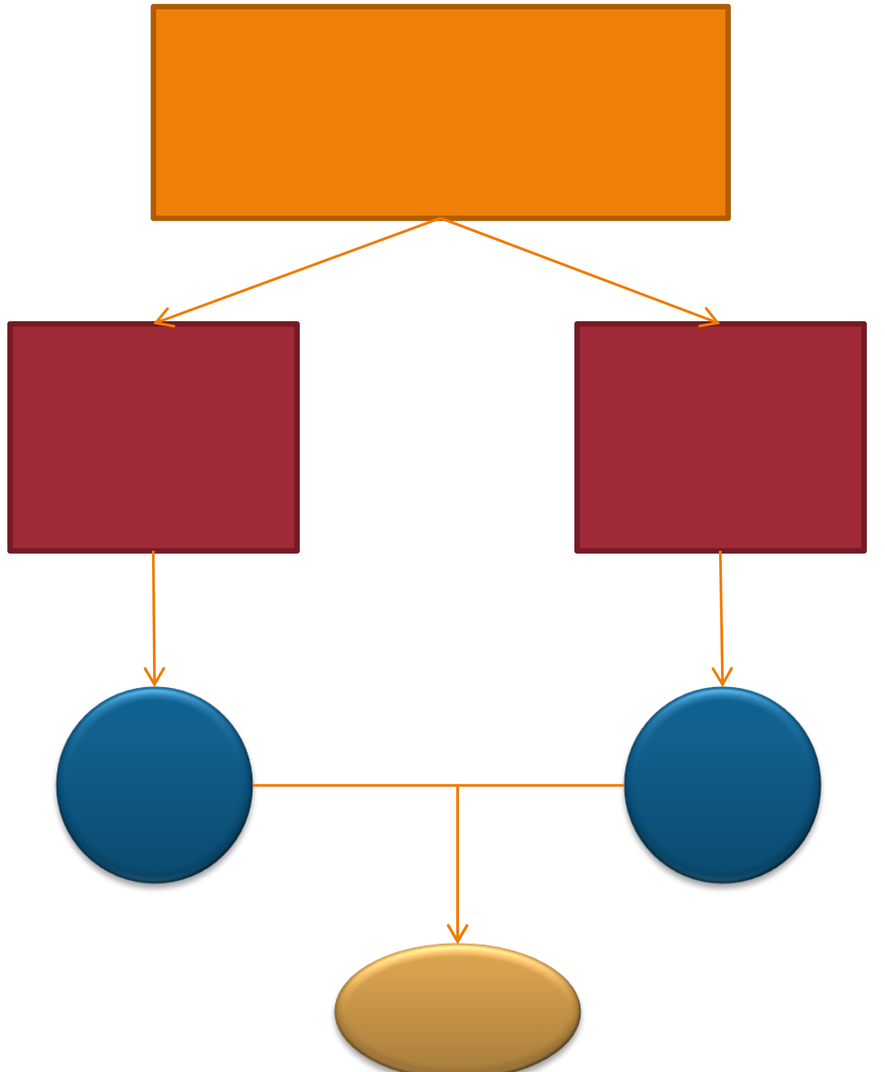
Introdução

- Divisão e conquista segue o seguinte plano geral:
  - Um problema é dividido em vários subproblemas (idealmente de mesmo tamanho);
  - Os subproblemas são resolvidos (tipicamente de forma recursiva);
  - Se necessário, as soluções dos subproblemas são combinadas para se obter a solução do problema original.

Introdução



Introdução



Dividir ao meio é o mais comum.

Introdução

- Como exemplo, consideremos o problema da soma de n números $a_0,...,a_{n-1}$.Se n > 1, é possível dividir o problema em 2 instâncias: computar a soma dos primeiros $\left\lfloor \frac{n}{2} \right\rfloor$ números e computar a soma do restante dos números (claro que, se n = 1, o resultado será a_0).
- Será que é uma solução eficiente para o problema?

Introdução

- Será que é uma solução eficiente para o problema?
- R.: Não! Nem sempre produzirá uma solução eficiente! Mas, felizmente, na maioria das vezes sim!

Introdução

- Outra informação importante que não pode ser omitida é que a técnica de dividir para conquistar pode ser aplicada em computação paralela em que a computação dos subproblemas pode ser realizada simultaneamente utilizando processadores específicos.

Introdução

- O caso típico de dividir para conquistar é a divisão de um problema de tamanho n em 2 instâncias menores de tamanho n/2.
- Em um caso mais geral, um problema de tamanho n pode ser dividido em b instâncias de tamanho n/b sendo que a deles precisam ser resolvidos (com $a \geq 1$ e $b > 1$).
- Se assumirmos que n é potência de b (apenas para efeitos de simplificação), temos:

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

Introdução

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

- Sendo que f(n) é a função responsável por contabilizar o tempo com a divisão do problema e combinar as soluções.
- Claro que a ordem de crescimento da solução T(n) depende das constantes a e b e da ordem de crescimento da função f(n).
- Utilizando o teorema MESTRE é possível simplificar a análise.

Introdução

- Teorema MESTRE

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

Se f(n) ϵ $\Theta(n^d)$, onde $d \geq 0$, então

$$T(n) \in \begin{cases} \Theta(n^d) & se \quad a < b^d \\ \Theta(n^d \log n) & se \quad a = b^d \\ \Theta(n^{\log_b a}) & se \quad a > b^d \end{cases}$$

Introdução

- Exemplo

Considere o exemplo de adições de n números ($n = 2^k$) utilizando a estratégia de dividir para conquistar. Teríamos neste caso a = 2 e b = 2 e f(n) = 1.

$$T(n) = a\,T(n/b) + f(n)$$

Introdução

Se f(n) é 1, isso significa que f(n) ϵ Θ(n^d), onde d = 0. Então, pelo teorema MESTRE temos:

$$A(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$$

Através do teorema é possível identificar a eficiência do algoritmo sem ter que resolver a recorrência!

Introdução

Também é importante observar que, se $a = 1$, a recorrência $T(n) = a.T(n/b) + f(n)$ resolve problemas pela estratégia de decremento por um fator constante da estratégia mais geral decremento e conquista.

Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Mergesort

- Mergesort é um exemplo perfeito de aplicação bem sucedida da estratégia de divisão e conquista. O algoritmo ordena um conjunto de n elementos ($a_0,..,a_{n-1}$) através da divisão do problema em 2 instâncias com metade dos elementos que são resolvidas recursivamente.
- Os resultados são, então combinados para produção da solução final.

Mergesort

Algoritmo MS(A[0,...,n-1])

se n > 1

copia (A[0..$\lfloor n/2 \rfloor$-1] para B[0..$\lfloor n/2 \rfloor$-1]

copia (A[$\lfloor n/2 \rfloor$-1.. n-1) para C[0..$\lceil n/2 \rceil$-1]

MS(B[0..$\lfloor n/2 \rfloor$-1])

MS(C[0..$\lceil n/2 \rceil$-1])

combina(B,C,A)

Mergesort

- **Algoritmo de combinação das soluções**

```
combina(B[0..p-1],C[0..q-1],A[0..p+q-1])
    i ⟵ 0; j ⟵ 0; k ⟵ 0
    enquanto i < p E j < q faça
        se  B[i] ≤ C[j]
            A[k] ⟵ B[i]
            i ⟵ i + 1
        senão
            A[k] ⟵ C[j]
            j ⟵ j + 1
        k ⟵ k + 1
se i = p
    copia C[j..q-1] para A[k..p+q-1]
senão
    copia B[i..p-1] para A[k..p+q-1]
```

Mergesort

- Exemplo

8	3	2	9	7	1	5	4

Mergesort

- Exemplo

8	3	2	9	7	1	5	4

8	3	2	9

7	1	5	4

Mergesort

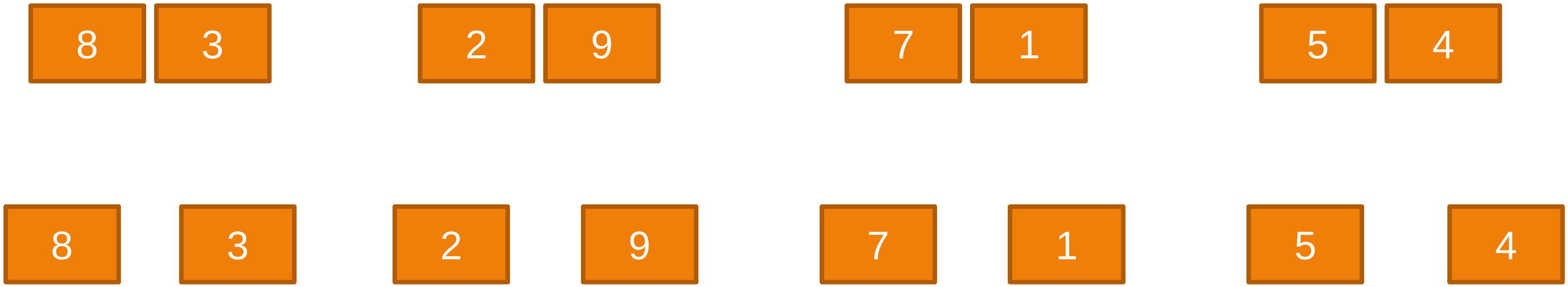
- Exemplo

8	3	2	9		7	1	5	4

8	3		2	9		7	1		5	4

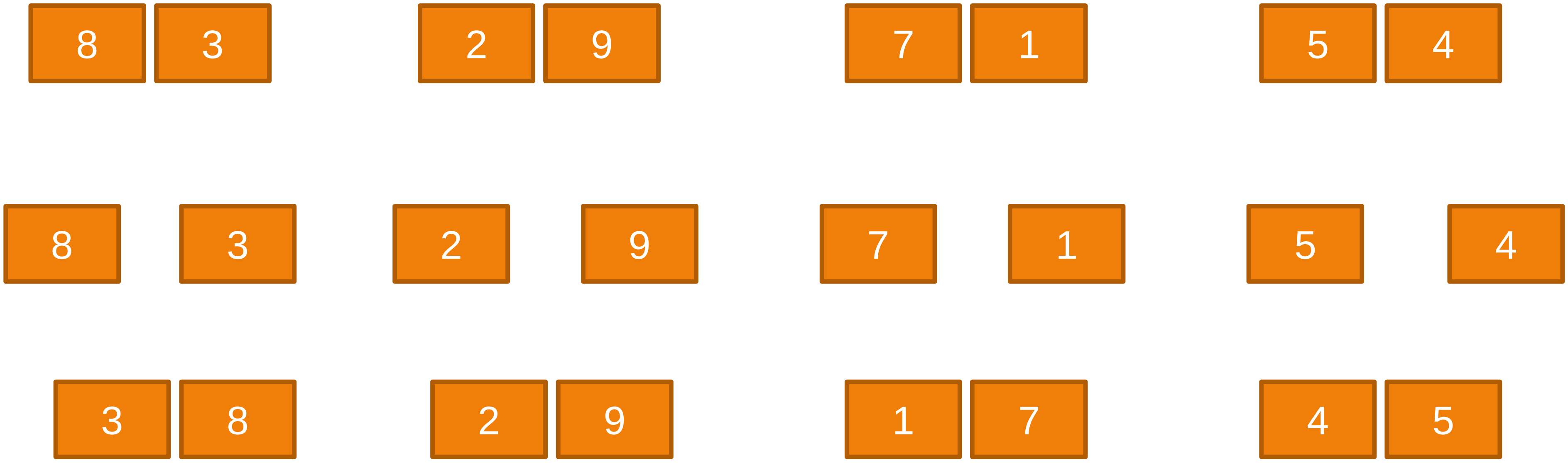
Mergesort

- Exemplo



Mergesort

- Exemplo



Mergesort

- Exemplo

3 8   2 9   1 7   4 5

2 3 8 9   1 4 5 7

Mergesort

- Exemplo

2	3	8	9	1	4	5	7

1	2	3	4	5	7	8	9

Mergesort

- Análise de eficiência

Por simplicidade, assuma que n é potência de 2. Assim, a recorrência levando-se em conta o número de comparações é dada pela fórmula:

$$C(n) = 2.\ C(n/2) + C_{merge}(n) \text{ para } n > 1$$

$$C(1) = 0$$

No pior cenário, existem n-1 comparações para a junção dos elementos, ou seja, $C_{merge}(n) = n - 1$.

Mergesort

- Análise de eficiência

Pelo teorema MESTRE temos que

$$C_{pior}(n) = n.\log_2 n - n + 1$$

Para n grande, $C(n) \in \Theta(n.\log n)$

Quicksort

- Quicksort é outro importante algoritmo baseado na técnica de dividir para conquistar. Na verdade foi eleito um dos 10 algoritmos mais importantes do século 20.
- Enquanto que o Mergesort “se preocupa” com a posição dos elementos no vetor, o Quicksort “se preocupa” com os valores de cada elemento.

Quicksort

- **Algoritmo**

```
Quicksort (A[l..r])
    if l < r
        s ← partição(A[l..r]) // s é o ponto de divisão
        Quicksort(A[l..s-1])
        Quicksort(A[s+1..r])
```

Quicksort

Algoritmo de partição

HoarePartition(A[l..r])

p ⟵ A[l]

i ⟵ l; j ⟵ r + 1

repita

repita

i ⟵ i + 1

até A[i] ≥ p

repita

j ⟵ j - 1

até A[j] ≤ p

troca(A[i],A[j])

até i ≥ j

Cont.

...

troca(A[i],A[j])

troca(A[l],A[j])

retorna j

Quicksort

- Para o melhor caso, o algoritmo pertence a $\Theta(n.\log_2 n)$
- Entretanto, o algoritmo não apresenta o mesmo desempenho para o pior caso. Qual seria ele e qual seria sua eficiência?

Quicksort

- Devido a sua importância, várias modificações foram sugeridas com o objetivo de melhorar seu desempenho. A combinação delas pode levar a melhoramentos da ordem de 20 a 30%. Eis algumas:
  - Melhor escolha de pivô;
  - Mudança para insertion sort quando se atinge vetores de tamanho pequeno (5 a 15 elementos);
  - Modificações no algoritmo de partição.

Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Exercícios

- Ache a ordem de crescimento dos algoritmos que respeitam as seguintes recorrências:

  a.) $T(n) = 4.\ T(n/2) + n$ com $T(1) = 1$

  b.) $T(n) = 4.\ T(n/2) + n^2$ com $T(1) = 1$

  c.) $T(n) = 4.\ T(n/2) + n^3$ com $T(1) = 1$

Exercícios

- Faça um algoritmo (utilizando divisão e conquista) para encontrar a posição do maior elemento em um vetor não ordenado de n elementos.
- Depois que fizer o algoritmo, compare-o com a versão baseada em força bruta.

Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Multiplicação de inteiros grandes

- Em algumas situações (como em criptografia) é necessário trabalhar com números com mais de 100 casas decimais. Como são números muito grandes para caber em uma palavra de computador, eles necessitam de tratamento especial.
- Claro que se fizermos o método convencional em que cada número tem n dígitos, vamos acabar chegando a um total de n^2 multiplicações.

Multiplicação de inteiros grandes

- Com a estratégia de dividir para conquistar é possível obter um algoritmo mais eficiente.
- Vejamos a ideia básica por detrás deste algoritmo com um exemplo: 23 x 14

$$23 = 1.10^1 + 3.\ 10^0$$

$$14 = 1.10^1 + 4.10^0$$

A multiplicação dos dois números produz:

$$(2.10^1 + 3.10^0) \times (1.10^1 + 4.\ 10^0)$$

Multiplicação de inteiros grandes

E

$$(2 \times 10^1 + 3 \times 10^0) \times (1 \times 10^1 + 4 \times 10^0)$$

pode ser expresso da seguinte forma:

$$(2 \times 1) \times 10^2 + (2 \times 4 + 3 \times 1) \times 10^1 + (3 \times 4) \times 10^0$$

$$= \quad (2 + 3) \times (1 + 4) - 2 \times 1 - 3 \times 4$$

Como 2 x 1 e 3 x 4 precisam ser computados de qualquer forma, podemos fazer uso deste resultado e computar o elemento do meio com apenas 1 multiplicação.

Multiplicação de inteiros grandes

Assim, para cada multiplicação c = a x b temos

$$c = c_2 \times 10^2 + c_1 \times 10^1 + c_0$$

Com

$$c_2 = a_1 \times b_1$$

$$c_0 = a_0 \times b_0$$

$$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$$

Como se vê é possível diminuir o número de multiplicações.

Multiplicação de inteiros grandes

Tal truque pode ser replicado para números maiores. Vamos considerar que temos 2 números de n dígitos (n par e positivo). Para aproveitar a técnica de dividir para conquistar, vamos dividir ambos os números ao meio:

$$c = a \times b = (a_1 \times 10^{n/2} + a_0) \times (b_1 \times 10^{n/2} + b_0)$$

$$= (a_1 \times b_1) \times 10^n + (a_1 \times b_0 + a_0 \times b_1) \times 10^{n/2} + (a_0 \times b_0)$$

$$= c_2 \times 10^n + c_1 \times 10^{n/2} + c_0$$

Multiplicação de inteiros grandes

Este algoritmo precisa de 3 multiplicações de números com n/2 dígitos. Assim, a recorrência em relação ao número de multiplicações é dada pela fórmula:

$$M(n) = 3.M(n/2) \quad \text{para } n > 1 \text{ com } M(1) = 1$$

Tendo em mente que $n = 2^k$ podemos resolver esta recorrência por substituição para trás obtendo:

$$M(2^k) = 3^k.M(2^{k-k}) = 3^k$$

Multiplicação de inteiros grandes

- Como k = $\log_2 n$ tem-se:

$$M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}$$

Lembrando que:

$$a^{\log_b c} = c^{\log_b a}$$

Multiplicação de inteiros grandes

- Descoberto em 1960 por um matemático russo de apenas 23 anos, o algoritmo provou que a ideia que se tinha até então que uma multiplicação de 2 números inteiros iria pertencer a $\Omega(n^2)$ era incorreta.

Algoritmo de Strassen

- Em 1969 Strassen apresentou seu algoritmo de multiplicação de matrizes que tinha desempenho melhor que o de força bruta.
- O insight sobre a solução veio da observação que era possível calcular o produto C de duas matrizes 2 x 2 com 7 multiplicações (e não 8 como requer o algoritmo baseado em força bruta).

Algoritmo de Strassen

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} + \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

Algoritmo de Strassen

$$m_1 = (a_{00} + a_{11}) \text{ x } (b_{00} + b_{11})$$
$$m_2 = (a_{10} + a_{11}) \text{ x } b_{00}$$
$$m_3 = a_{00} \text{ x } (b_{01} - b_{11})$$
$$m_4 = a_{11} \text{ x } (b_{10} - b_{00})$$
$$m_5 = (a_{00} + a_{01}) \text{ x } b_{11}$$
$$m_6 = (a_{10} - a_{00}) \text{ x } (b_{00} + b_{01})$$
$$m_7 = (a_{01} - a_{11}) \text{ x } (b_{10} + b_{11})$$

Algoritmo de Strassen

$m_1 = (a_{00} + a_{11}) \times (b_{00} + b_{11})$

$m_2 = (a_{10} + a_{11}) \times b_{00}$

$m_3 = a_{00} \times (b_{01} - b_{11})$

$m_4 = a_{11} \times (b_{10} - b_{00})$

$m_5 = (a_{00} + a_{01}) \times b_{11}$

$m_6 = (a_{10} - a_{00}) \times (b_{00} + b_{01})$

$m_7 = (a_{01} - a_{11}) \times (b_{10} + b_{11})$

7 multiplicações
10 adições/subtrações

Algoritmo de Strassen

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} + \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

7 multiplicações
18 adições/subtrações

Algoritmo de Strassen

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} + \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

7 multiplicações
18 adições/subtrações

Algoritmo de Strassen

1	0	2	1
4	1	1	0
0	1	3	0
5	0	2	1

$\times$

0	1	0	1
2	1	0	4
2	0	1	1
1	3	5	0

Algoritmo de Strassen

1	0	2	1
4	1	1	0
0	1	3	0
5	0	2	1

×

0	1	0	1
2	1	0	4
2	0	1	1
1	3	5	0

1	0
4	1

2	1
1	0

0	1
5	0

3	0
2	1

0	1
2	1

0	1
0	4

2	0
1	3

1	1
5	0

Algoritmo de Strassen

1	0	2	1
4	1	1	0
0	1	3	0
5	0	2	1

×

0	1	0	1
2	1	0	4
2	0	1	1
1	3	5	0

1	0
4	1

+

3	0
2	1

0	1
2	1

+

1	1
5	0

4	0
6	2

×

1	2
7	1

4x1 + 0x7	4x2 + 0x1
6x1 + 2x7	6x2 + 2x1

=

4	8
20	14

m_1

Algoritmo de Strassen

1	0	2	1
4	1	1	0
0	1	3	0
5	0	2	1

$\times$

0	1	0	1
2	1	0	4
2	0	1	1
1	3	5	0

0	1
5	0

$+$

3	0
2	1

3	1
7	1

$\times$

0	1
2	1

3x0 + 1x2	3x1 + 1x1
7x0 + 1x2	7x1 + 1x1

$=$

2	4
2	8

m_2

Algoritmo de Strassen

m_1

4	8
20	14

m_2

2	4
2	8

m_3

-1	0
-9	4

m_4

6	-3
3	0

m_5

8	3
10	5

m_6

2	3
-2	-3

m_7

3	2
-9	-4

$m_1+m_4-m_5+m_7$	m_3+m_5
m_2+m_4	$m_1+m_3-m_2+m_6$

Algoritmo de Strassen

4	8
20	14

6	-3
3	0

8	3
10	5

3	2
-9	-4

4+6-8+3	8-3-3+2
20+3-10-9	14+0-5-4

5	4
4	5

$m_1+m_4-m_5+m_7$	m_3+m_5
m_2+m_4	$m_1+m_3-m_2+m_6$

Algoritmo de Strassen

5	4	7	3
4	5	1	9
8	1	3	7
5	8	7	7

$m_1+m_4-m_5+m_7$	m_3+m_5
m_2+m_4	$m_1+m_3-m_2+m_6$

Algoritmo de Strassen

- Se n (número de linhas e colunas) não for potência de 2, basta completar com 0s em linhas e colunas até se obter potência de 2.
- Podemos analisar a eficiência assintótica do algoritmo considerando o número de multiplicações:

$$M(n) = 7.M(n/2) \qquad \text{para } n > 1$$

$$M(1) = 1$$

Algoritmo de Strassen

- Como $n = 2^k$, temos:

$$
\begin{aligned}
M(2^k) &= 7.M(2^{k-1}) \\
&= 7^2.M(2^{k-2}) \\
&= 7^3.M(2^{k-3}) \\
&= \ldots \\
&= 7^k.M(2^{k-k}) \qquad = 7^k
\end{aligned}
$$

Algoritmo de Strassen

- Como $k = \log_2 n$, temos:

$$M(2^k) = 7^k$$

$$= 7^{\log_2 n} = n^{\log_2 7} = n^{2.807}$$

Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Exercícios

- Faça a multiplicação de 2.101 por 1.130 através da técnica de divisão e conquista.

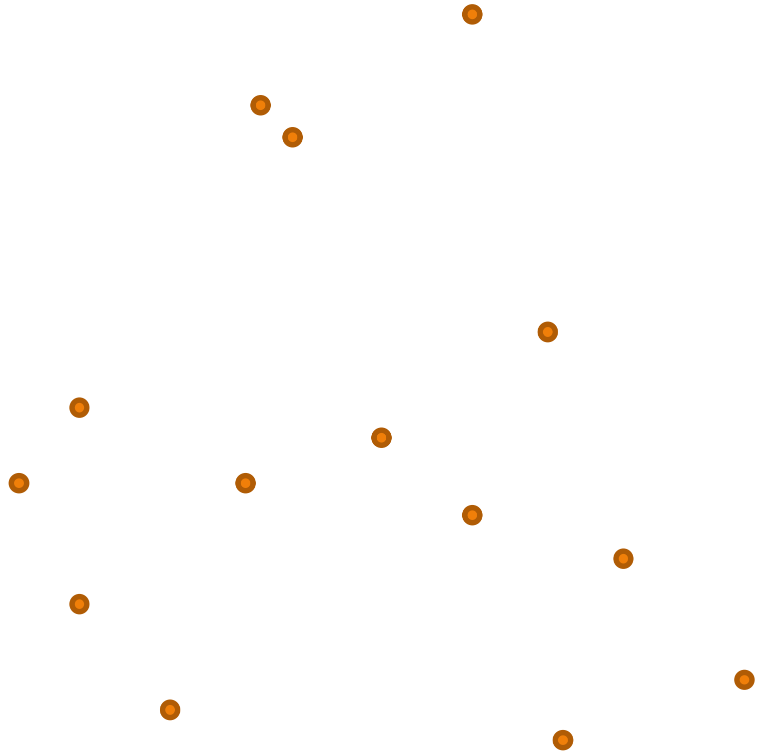
Divisão e Conquista

- Introdução
- Merge Sort
- Quicksort
- Exercícios
- Multiplicação de números inteiros grandes
- Multiplicação de matrizes
- Exercícios

Divisão e Conquista

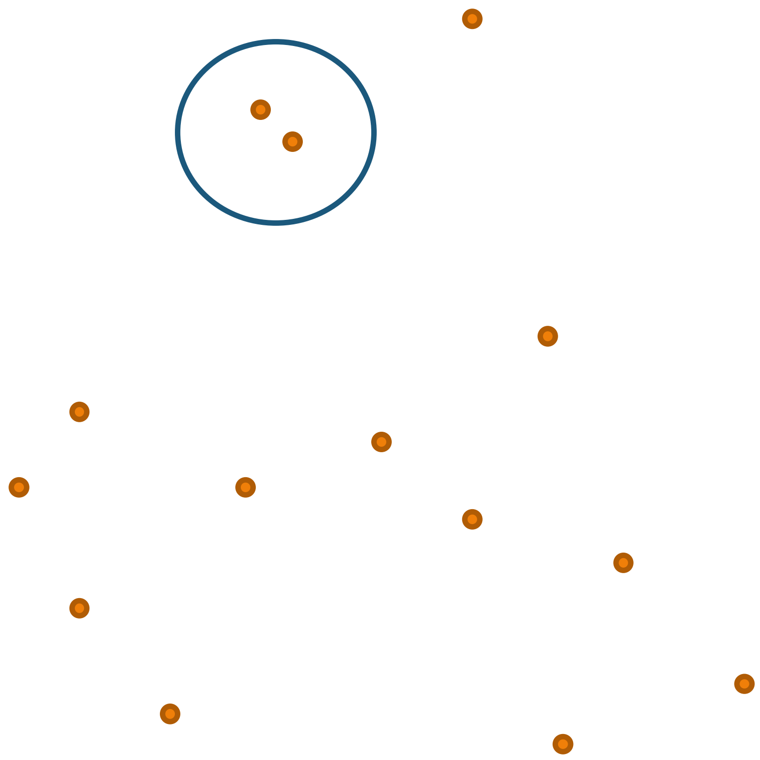
- Closest-pair
- Convex-hull
- Exercícios
- Resumo

Closest-pair



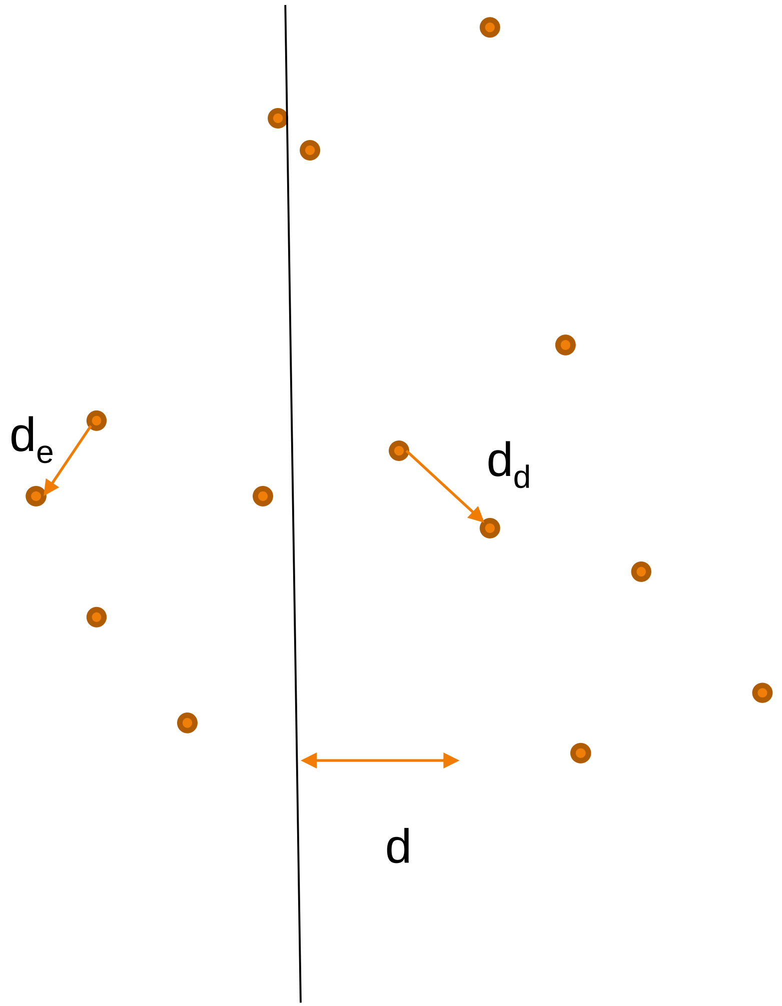
O problema se resume a encontrar 2 pontos mais próximos um do outro.

Closest-pair



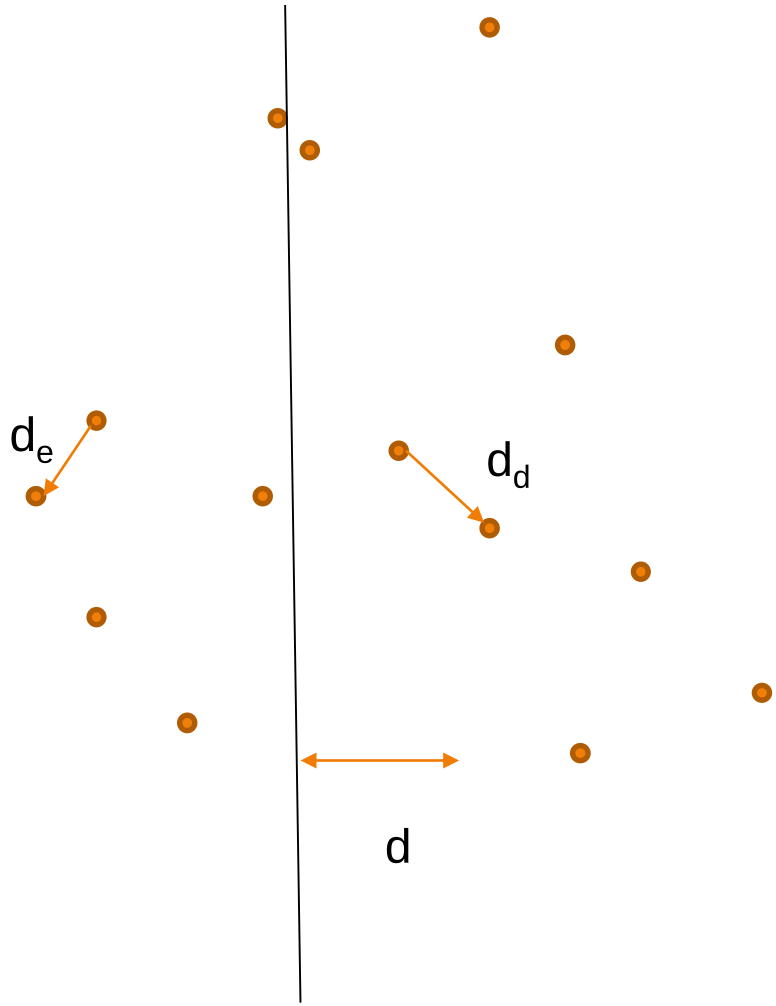
O problema se resume a encontrar 2 pontos mais próximos um do outro.

Closest-pair



Se o número de pontos for 2 ou 3, então pode-se resolver o problema pela força bruta...

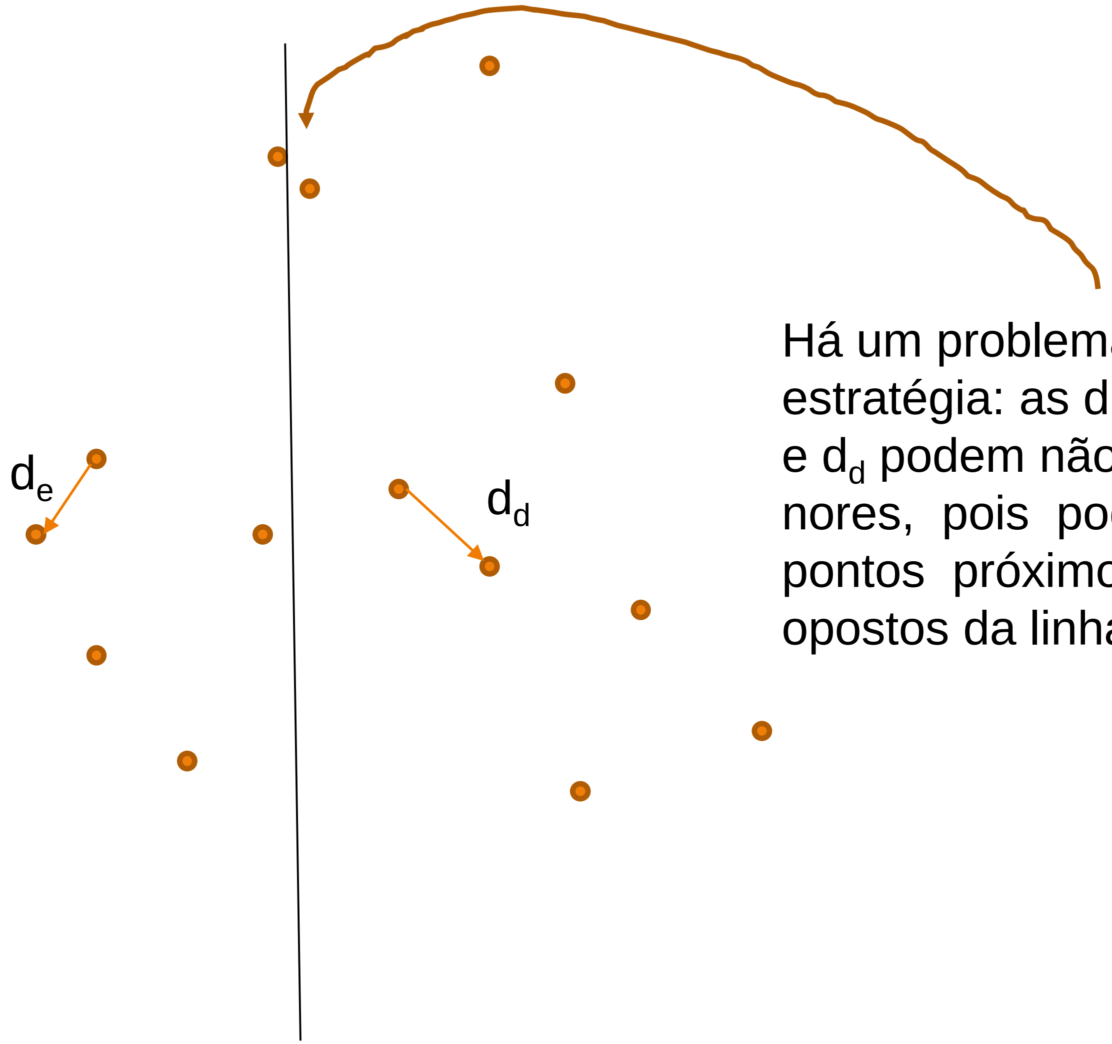
Closest-pair



Se n for maior que 3, então pode-se dividir o conjunto em 2 partes: a direita e a esquerda da linha divisória.

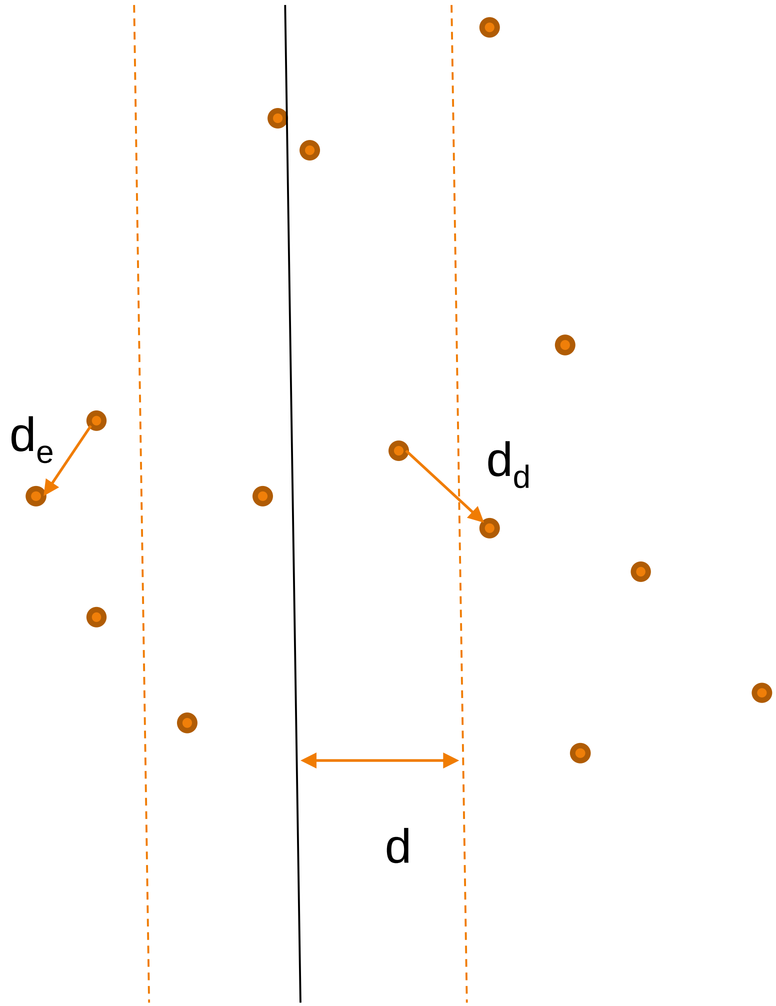
A linha divisória pode ser obtida através da média dos valores das coordenadas x.

Closest-pair



Há um problema com esta estratégia: as distâncias d_e e d_d podem não ser as menores, pois podem existir pontos próximos em lados opostos da linha divisória.

Closest-pair



Mas para verificar se isso realmente acontece, basta procurar dentro da faixa que compreende a linha divisória e as linhas tracejadas.

Closest pair (algoritmo)

- Tarefa para casa
  - Analisar o algoritmo a seguir e indicar sua eficiência

Closest pair (algoritmo)

closestPair (xP, yP)

// xP é uma lista de pontos P_1 .. P_n ordenada pela coordenada x

// yP é uma lista de pontos P_1 .. P_n ordenada pela coordenada y

se $n \leq 3$ **então retorna** closest-pair por força bruta

senão

xL $\longleftarrow$ pontos de xP de 1 a $\lceil n/2 \rceil$

xR $\longleftarrow$ pontos de xP de $\lceil n/2 \rceil$+1 até n

$x_m \longleftarrow xP(\lceil N/2 \rceil)_x$

$yL \longleftarrow \{ p \in yP : p_x \leq x_m \}$

$yR \longleftarrow \{ p \in yP : p_x > x_m \}$

(dL, pairL) $\longleftarrow$ *closestPair* of (xL, yL)

(dR, pairR) $\longleftarrow$ *closestPair* of (xR, yR)

(dmin, pairMin) $\longleftarrow$ (dR, pairR)

se dL < dR **então**

(dmin, pairMin) $\longleftarrow$ (dL, pairL)

$yS \longleftarrow \{ p \in yP : |x_m - p_x| < dmin \}$

nS $\longleftarrow$ número de pontos em yS

$(c_1, c_2) \longleftarrow$ (dmin, pairMin)

para i de 1 até nS - 1

k $\longleftarrow$ i + 1

enquanto $k \leq nS$ **E** $yS(k)_y - yS(i)_y < dmin$

se |yS(k) - yS(i)| < closest **então**

$(c_1, c_2) \longleftarrow$ (|yS(k) - yS(i)|, {yS(k), yS(i)})

k $\longleftarrow$ k + 1

retorna (c_1, c_2)

Divisão e Conquista

- Closest-pair
- Convex-hull
- Exercícios
- Resumo

Convex hull

- Procurando fazer uma abordagem reversa, vamos analisar o algoritmo a seguir referente ao problema Convex Hull (aquele que deve achar elementos que delimitam uma região convexa dentre um conjunto de elementos) e tentar desenhar o que acontece…

Convex-hull (quickhull)

Entrada um conjunto S de n pontos

Assume-se que existem pelo menos 2 pontos no conjunto S

QuickHull (S)

{

// Ache a "convex hull" do conjunto S

Convex Hull = {}

Ache os pontos mais a esquerda e mais a direita (A e B) e inclua-os na convex hull

O segmento de reta AB divide os (n-2) pontos que sobraram em 2 grupos S1 e S2

➔ S1 são os pontos à direita da linha AB; (ou acima)

➔ S2 são os pontos à direita da linha BA (ou abaixo)

FindHull (S1, A, B)

FindHull (S2, B, A)

}

Convex-hull (quickhull)

FindHull (S_k, P, Q)

{

// Ache os pontos da convex hull dentro do conjunto S_k que estão à direita da linha P $\rightarrow$ Q

se S_k não tem pontos,

então retorna.

Do conjunto S_k ache o ponto mais distante do segmento de reta PQ (seja tal ponto C)

Inclua o ponto C na convex hull

Tres pontos {P, Q e C} repartem o conjunto S_k em 3 subconjuntos: S_0, S_1, and S_2

onde S_0 são pontos dentro do triângulo PCQ; S_1 são pontos à direita de PC; e S_2 são pontos à direita de CQ

FindHull(S_1, P, C)

FindHull(S_2, C, Q)

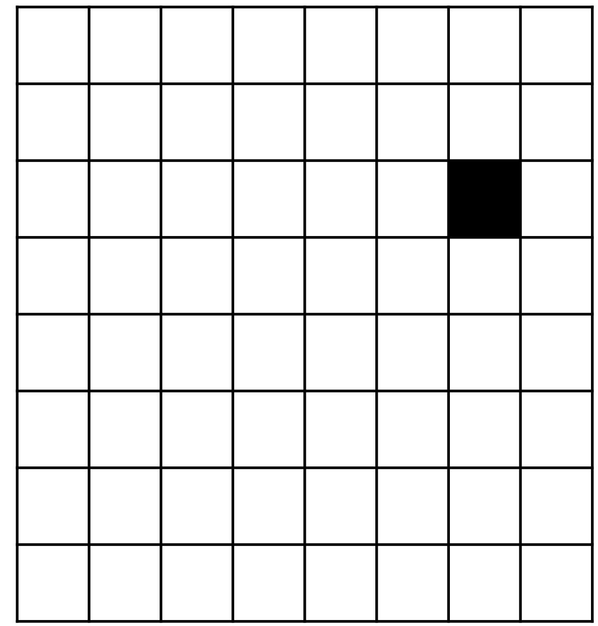
}

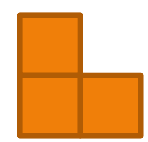
Divisão e Conquista

- Closest-pair
- Convex-hull
- Exercícios
- Resumo

Exercícios

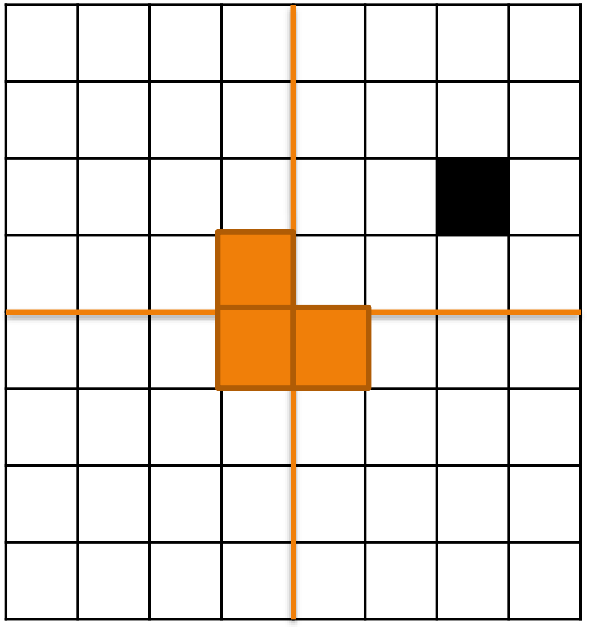
Tromino é uma peça em L formada por 3 quadrados de igual tamanho (1). Há um desafio que é colocar trominos em um tabuleiro de tamanho 2^n x 2^n de tal forma que o tabuleiro fique todo preenchido exceto uma posição que ficará sem preencher. Projete um algoritmo para resolver tal problema.

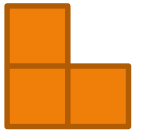




Exercícios

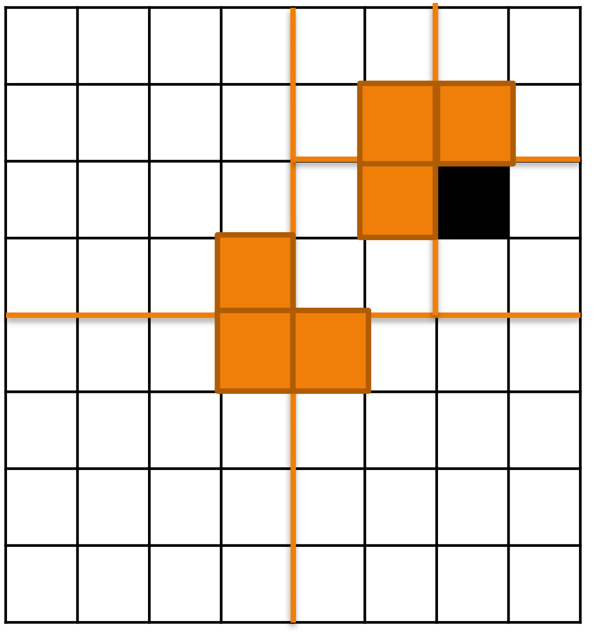
Tromino é uma peça em L formada por 3 quadrados de igual tamanho (1). Há um desafio que é colocar trominos em um tabuleiro de tamanho 2^n x 2^n de tal forma que o tabuleiro fique todo preenchido exceto uma posição que ficará sem preencher. Projete um algoritmo para resolver tal problema.

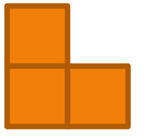




Exercícios

Tromino é uma peça em L formada por 3 quadrados de igual tamanho (1). Há um desafio que é colocar trominos em um tabuleiro de tamanho 2^n x 2^n de tal forma que o tabuleiro fique todo preenchido exceto uma posição que ficará sem preencher. Projete um algoritmo para resolver tal problema.





Exercícios - tarefa

- Implemente o algoritmo closest-pair em C;
- Crie um conjunto de n pontos bidimensionais randomicos e aplique o algoritmo desenvolvido para achar o par com menor distância entre seus elementos.

Divisão e Conquista

- Closest-pair
- Convex-hull
- Exercícios
- Resumo

Resumo

- Divisão e conquista é uma técnica geral de projeto de algoritmos que resolve problemas através da estratégia de solução de partes menores do mesmo problema (normalmente de tamanhos iguais) e combinando os resultados de cada subproblema para a obtenção da solução do problema maior.

Resumo

- O tempo de execução $T(n)$ de muitos algoritmos que se encaixam nessa técnica satisfaz a recorrência $T(n) = a\,T(n/b) + f(n)$. O teorema Mestre estabelece a ordem de crescimento de suas soluções.
- Mergesort é um algoritmo de ordenação que se encaixa perfeitamente como exemplo típico de solução por divisão e conquista. Sua eficiência é $\Theta(n.\log n)$ em todos os casos sendo o número de comparações muito próximo do mínimo teórico.

Resumo

- Quicksort é outro exemplo de aplicação da estratégia para ordenação. Funciona através de ordenações recursivas de partições segundo um elemento pivô. Sua eficiência é da ordem de n. log n, mas no pior caso tem comportamento quadrático.
- É possível, utilizando a estratégia de dividir para conquistar, multiplicar 2 inteiros de n dígitos utilizando $n^{1.585}$ multiplicações de 1 dígito.

Resumo

- Como indicado em aulas anteriores, é possível realizar a multiplicação de matrizes n x n através de um algoritmo com eficiência $n^{2.807}$ obedecendo o algoritmo de Strassen.
- A estratégia pode ser aplicada (e produzir bons resultados) em 2 problemas clássicos: closest-pair e convex-hull.

THE END