**UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO**
**ESCOLA DE INFORMÁTICA APLICADA**
**CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

# Segundo Trabalho
# Divisão e Conquista - Par de pontos mais próximos

**Bruno Lírio Alves**
**Pedro Paulo Gouveia**
**Thales Veras**

**Vânia Felix**

**Rio de Janeiro, 17 de outubro de 2013.**

## 2. Motivação

### 2.1. Definição do Problema

Resolver o problema da menor distância entre dois pontos possui uma série de aplicações em diversos algoritmos usados em diversos assuntos, desde a Física até a Computação Gráfica.

#### 2.1.2. Definição Formal do Problema (Entrada / Questão / Saída)

Dado um conjunto S de n pontos no plano, encontrar o par de pontos mais próximos.

Entrada: Temos um conjunto de n pontos S num plano.
Questão: Como podemos encontrar a menor distância entre dois pontos em S.
Saída: O par de pontos mais próximos.

Na primeira fase da questão, a divisão, o problema é decomposto em dois subproblemas.

- Quebrar S em S1 e S2.

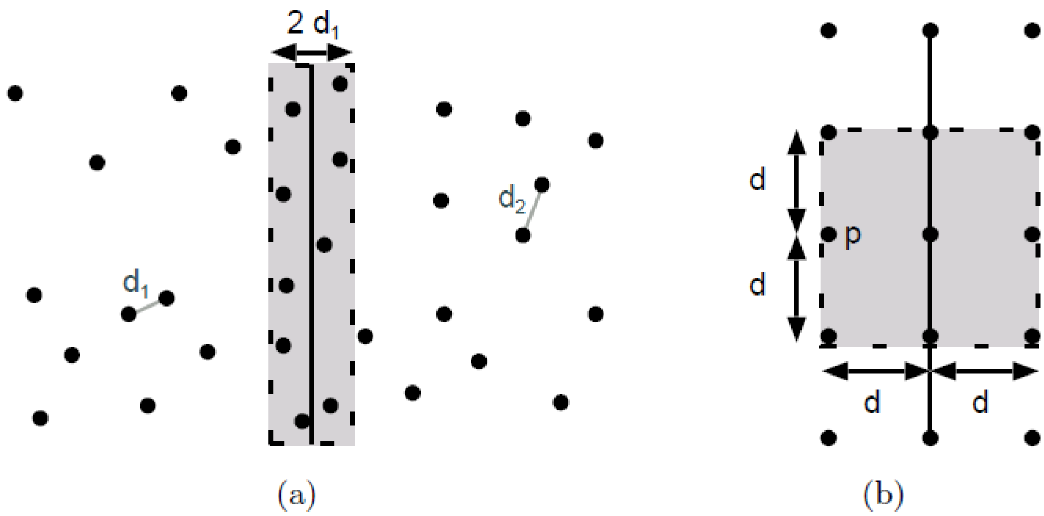
Na segunda fase da questão, a conquista, resolvemos os sub-problemas. A beleza da técnica reside no fato de que os problemas menores podem ser resolvidos recursivamente, usando o mesmo procedimento de divisão e conquista, até que o tamanho do problema seja tão pequeno que sua solução seja trivial ou possa ser feita mais rapidamente usando algoritmos mais simples.

- d1 = PontosMaisProximos (S1)
- d2 = PontosMaisProximos (S2)

Na terceira fase da questão, a combinação das soluções, temos que unir as soluções dos problemas menores para obtermos uma solução unificada. Este procedimento nem sempre é trivial, e muitas vezes pode ser simplificado se a divisão (primeira fase) for feita de modo inteligente.

- d = min (d1,d2)
- Determinar a faixa divisória e pontos
- Verificar se tem algum par com distância < d

### 2.2 Exemplos de Instâncias do Problema




(a) A execução do algoritmo para encontrar o par de pontos mais próximos.
(b) Número máximo de pontos que podem estar contidos no quadrado.

O problema do par de pontos mais próximo consiste em, dado um conjunto de n pontos P, encontrar o par de pontos cuja distância entre eles é a menor possível dentro daquele conjunto.

## 3. Algoritmo
### 3.1. Descrição da ideia do algoritmo

Um algoritmo ingênuo, ou força bruta, buscaria as distâncias entre todos os pontos o que e nos daria uma resposta seguramente correta, uma vez que todas as possibilidades são analisadas, porém teríamos um tempo de $O(n^2)$. Uma abordagem por divisão e conquista nos ajuda a diminuir a complexidade de tempo para $O(n * \log n)$.Na execução do algoritmo, quebramos o problema em duas metades, resolvendo recursivamente cada uma. Bastaria então "colarmos" os resultados. A base da recursão acontece quando o problema tem 3 ou menos pontos. A idéia chave é pré-ordenar os pontos antes da fase de divisão-e-conquista de alguma forma, no caso em relação aos eixos das abscissas e ordenadas. A partir daí, faz sentido podermos falar em duas metades, sendo que cada uma contenha n/2 pontos ou aproximadamente isto. Obtemos recursivamente as soluções para cada metade, guardando os pontos que estão a mínima distância em cada subsolução, teremos assim, d1 para uma metade e d2 para outra. Computamos d, mínimo entre essas duas distâncias (d1 e d2) e agora basta verificarmos se não existem dois pontos, um em cada metade, cuja distância é menor que d. E é aí que entra uma observação chave do problema, para cada ponto p em uma metade, precisamos calcular a distância apenas entre p e 6 pontos no máximo.

### 3.2. A partir da ideia, mostrar exemplos de solução para as instâncias apresentadas

Para o par de pontos mais próximos de S1 temos a distância d1 e o par de pontos mais próximos de S2 temos a distância d2. Podemos obter o par de pontos mais próximos de S1 U S2. Então o par de pontos mais próximos tem distância menor ou igual a min (d1, d2). É possível que o par que estamos buscando tenha um ponto em S1 e outro em S2. Assim podemos descartar seguramente os pontos que distam mais que delta da reta vertical que usamos para dividir os pontos. O algoritmo calcula o par de pontos mais próximos em S' (subconjunto) e compara a sua distância com min (d1, d2), analisando a menor.
Para todo ponto p em S', podemos calcular a distância entre p e os pontos de S' cuja a diferença de coordenada y seja menor que delta. Por exemplo, como o número de pontos é no máximo 6 como mostra a figura (a), fica claro que, garatimos que só precisamos fazer um número linear de comparações , para justificarmos que o número desses pontos é no máximo 6 para cada ponto p, os pontos de S' que distam até delta de p, estão dentro de um quadrado de lado 2min(d1,d2), e não há mais outros dois pontos na mesma metade desse quadrado que distem menos de min(d1, d2).

### 3.3. Descrição Formal do Algoritmo

**closestPair** of (xP, yP)
            where xP is P(1) .. P(N) sorted by x coordinate, and
                 yP is P(1) .. P(N) sorted by y coordinate (ascending order)
**if** N ≤ 3 **then**
  **return** closest points of xP using brute-force algorithm
**else**
  xL ← points of xP from 1 to ⌈ N/2⌉
  xR ← points of xP from ⌈ N/2⌉ +1 to N
  xm ← $xP(\lceil N/2 \rceil)_x$
  yL ← { p ∈ yP : $p_x$ ≤ xm }
  yR ← { p ∈ yP : $p_x$ > xm }
  (dL, pairL) ← *closestPair* of (xL, yL)
  (dR, pairR) ← *closestPair* of (xR, yR)
  (dmin, pairMin) ← (dR, pairR)
  **if** dL < dR **then**
    (dmin, pairMin) ← (dL, pairL)
  **endif**
  yS ← { p ∈ yP : |xm - $p_x$| < dmin }
  nS ← number of points in yS
  (closest, closestPair) ← (dmin, pairMin)
  **for** i **from** 1 **to** nS - 1
    k ← i + 1
    **while** k ≤ nS **and** $yS(k)_y$ - $yS(i)_y$ < dmin
      **if** |yS(k) - yS(i)| < closest **then**
        (closest, closestPair) ← (|yS(k) - yS(i)|, {yS(k), yS(i)})
      **endif**
      k ← k + 1

```
    endwhile
  endfor
  return closest, closestPair
endif
```

## 4. Análise do Algoritmo

### 4.1 Prova de corretude (mostrar que o algoritmo funciona de fato)

Para qualquer dado conjunto S de pontos, após as ordenações, é traçada uma reta r dividindo os pontos em dois conjuntos S1 e S2, após calculadas recursivamente a menor distância d em cada um dos conjuntos, para cada ponto p em S1 com distância menor ou igual a d da reta r, são aferidas distâncias entre p e os pontos de S2 que atendem a condição de estarem no retângulo com lados 2d e d e vice-versa, o que permite obter pontos que possuam distância igual ou menor a d em S1 U S2, finalmente, comparadas as distâncias obtidas em S1, S2 e S1 U S2 sempre chegaremos a menor distância entre dois pontos no conjunto S.

### 4.2. Demonstração da complexidade/tempo do algoritmo

Na maioria das situações, já reduziria bastante o tempo de processamento, no pior caso, pode ser que não descartemos nenhum ponto, teríamos de calcular $O(n^2)$ distâncias.
Para analisarmos a complexidade do tempo deste algortimo, construímos a recorrência,
$T(n) = 2T(n/2) + f(n)$
ao analisarmos S1 U S2 chegamos a f(n) que inicialmente seria $O(n^2)$ $(O(n/2) * O(n/2))$ mas, como vimos anteriormente é necessário apenas o cálculo para 6 pontos o que nos leva a $6 * O(n/2) = O(3n)$, ou seja, tempo linear O(n).
Assim, pelo Teorema Mestre temos
$T(n) = 2T(n/2) + O(n) = n * \log n$

## 5. Conclusão e Discussões

### 5.1 Vantagens e desvantagens

O mesmo algoritmo não se aplicaria em caso de um espaço com dimensão d diferente de dois. No entanto é um tempo bastante aceitável para sua tarefa comparados ao inicial $O(n^2)$;

### 5.2 É possível ter um algoritmo melhor?

Não dado o fato que foram consideradas as devidas ordenações iniciais mas, diferentes definições de distância por exemplo, poderiam influenciar sua performance.

## 6. Referências Bibliográficas

Distância. (2003). WIKIPEDIA. ***Site.*** Disponível em: <http://pt.wikipedia.org/wiki/Dist%C3%A2ncia>. Acesso em: 17 out. 2013.

Closest-pair Problem (2013). WIKI. ***Site.*** Disponível em: <http://rosettacode.org/wiki/Closest-pair_problem>. Acesso em: 17 out. 2013.

Closest pair of points problem (2013). WIKIPEDIA. ***Site.*** Disponível em: <http://en.wikipedia.org/wiki/Closest_pair_of_points_problem>. Acesso em: 17 out. 2013.