3 . O Problema do Par Mais Próximo

(*closest pair problem*)

Antonio L. Bajuelos
Departamento de Matemática
Universidade de Aveiro

Mestrado em Matemática e Aplicações

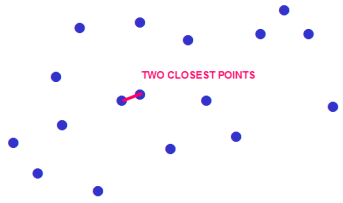
"Computers do not solve problems, People do!"

E. R. Davidson

O Problema…

- O Problema do Par mais Próximo (*the closest pair problem*)
  - **Enunciado:** Dados *n* pontos queremos encontrar dois cuja **distância*** entre eles é mínima

    *distância euclidiana: $d(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$



  - **Aplicação:** Sistema de Controlo de Tráfego Aéreo

3

O Problema do Par mais Próximo: um algoritmo ingénuo

- A primeira ideia de solução para este problema é:
  - Aplicar uma **busca exaustiva** em todos os pares de pontos da colecção dada e procurar a distância mínima.
    - Isso nos leva ao seguinte algoritmo ingénuo (ou por **força-bruta**)
    - **Algoritmo PmP1:**
      - Dada uma colecção **P= (p_1 , p_2, …, p_n)** de ***n*** pontos determina a distância mínima entre pontos de S.

1. D ← ∞
2. Para todo i em [1..n] faça
   2.1. Para todo j em [i + 1…n-1] faça
      2.1.1. D ← min{D, d(p_i,p_j)}
3. Devolve D

4

O Problema do Par mais Próximo: um algoritmo ingénuo

- **Algoritmo PmP1:**

  1. D ← ∞
  2. Para todo i em [1..n] faça
     2.1. Para todo j em [i + 1…n] faça
        2.1.1. D ← min{D, d(p_i,p_j)}
  3. Devolve D

- **Análise:** Como no passo 2.1.1. é calculado o mínimo entre a distância do par de pontos sendo considerado e a menor distância encontrada até o momento, D nunca cresce e assume o valor mínimo desejado.

  Como este passo é executado:

  $$(n-1)+(n-2)+\ldots+1=\frac{(n-1)n}{2}\in O(n^2)$$

  vezes, este algoritmo é um algoritmo de complexidade quadrática

Será possível encontrarmos um algoritmo mais eficiente?

5

Par mais Próximo: procurando melhores ideias

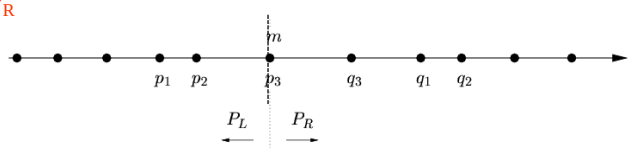
- **Ideia Nº 1:** Tentar fazer incursões em dimensões menores de modo a obter ideias que sugiram soluções eficientes para dimensões maiores.
  - **Par mais próximo em $\Re^1$** - algoritmo simples que resolve o problema de maneira muito eficiente:
    - **ordenar os pontos dados e procurar o par mais próximo através de um varrimento linear destes pontos**
    - Nesta estratégia o par de pontos mais próximos é necessariamente um par consecutivo no conjunto ordenado e é isto que permite desenvolver este algoritmo. Portanto, a noção de **consecutividade** e fundamental.

Não é possível generalizar esta ideia para outras dimensões!

6

Par mais Próximo: procurando melhores ideias

- **Ideia Nº 2:** Versão *divisão-e-conquista* em $\Re^1$
  - Dividimos um conjunto P de n pontos em $\Re^1$ pela sua mediana*:
    - temos dois subconjuntos disjuntos: P_L e P_R, (todos os pontos de P_L estão à esquerda da mediana e os de P_R à direita)
  - O par de pontos mais próximo em P será
    - um par contido em P_L ou
    - um par contido em P_R ou
    - um par muito particular de pontos de P_LxP_R
  - **Atenção:** Este último par necessariamente teria que ser composto do ponto mais à direita de P_L e do mais à esquerda de P_R, isto é, os dois pontos vizinhos da mediana - ponto de separação de P_L e P_R

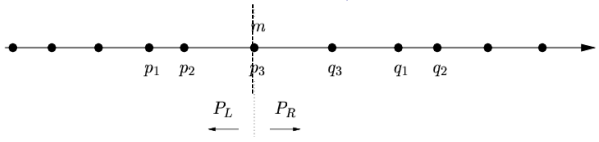


* a mediana será o elemento central (n+1)/2

7

Par mais Próximo: *divisão-e-conquista* em $\Re^1$

- **Dividir:**
  - Compute a mediana m de P.
  - Particione P em P_L e P_R tal que $p \leq m < q \;\; \forall\, p \in P_L$ e $q \in P_R$
- **Conquistar:**
  - Resolva, separadamente, o Problema do Par-Mais-Proximo (PmP) para P_L e P_R .
    - Sejam $\{p_1,p_2\}$ o PmP em P_L e $\{q_1,q_2\}$ o PmP em P_R
- **Combinar:**
  - Seja $\delta = \min\{|p_1-p_2|,|q_1-q_2|\}$. O PmP de P é
    - $\{p_1,p_2\}$ ou $\{q_1,q_2\}$ ou $\{p_3,q_3\}$, onde $p_3 \in P_L$ e $q_3 \in P_R$
  - Se PmP de P é $\{p_3,q_3\}$, então $|p_3-q_3| < \delta$
  - **Quantos pares de pontos são candidatos a $\{p_3,q_3\}$?**
    - **Resposta: No máximo 1!** ⇨ **p_3=m e q_3=min{q| q ∈ P_R}**



8

Par mais Próximo: *divisão-e-conquista* em $\Re^1$

- **Algoritmo PmP2:**

  Entrada: uma colecção $P = \{p_1, p_2, \ldots, p_n\}$ de n pontos em $\Re^1$.

  Saída: δ, a distância mínima entre um par $\{p_i, p_j\}$, $i \neq j$.

```
1.  if  n = 1 then return δ ← +∞
2.  else if n = 2 then return distância (p1,p2)
3.    else
4.      m ← mediana(P)
5.      Sejam PL := {p ∈ P | p ≤ m} e PR := {q ∈ P | q > m}
6.      δL ← PmP2(PL)
7.      δR ← PmP2(PR)
8.      p3 ← max {p ∈ PL}
9.      q3 ← min {q ∈ PR}
10.     δ  ← min {δL, δR, q3 - p3}
11.     return δ
```

9

Par mais Próximo: *divisão-e-conquista* em $\Re^1$

- **Algoritmo PmP2:**

```
1.  if  n = 1 then return δ ← +∞
2.  else if n = 2 then return distância (p1,p2)
3.    else
4.      m ← mediana(P)
5.      PL := {p ∈ P | p ≤ m} e PR := {q ∈ P | q > m}
6.      δL← PmP2(PL)
7.      δR ←  PmP2(PR)
8.      p3 ← max {p ∈ PL}
9.      q3 ← min {q ∈ PR}
10.     δ  ← min {δL, δR, q3 - p3}
11.   return δ
```

- **Análise do Algoritmo:**
  - Se, num pré-processamento, ordenarmos os pontos, então as linhas 4, 5, 8, 9 e 10 do algoritmo podem ser executadas em tempo constante. A complexidade de tempo $T(n)$ do algoritmo (sem o pré-processamento):

$$T(n) \leq \begin{cases} T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + cn & se\ n > 2 \\ b & se\ n = 2 \\ a & se\ n = 1 \end{cases}$$

⇒ **O algoritmo PmP2 é $O(n \log n)$**

10

Par mais Próximo em $\Re^2$

Será possível aplicar um método de divisão e conquista para o problema PmP em $\Re^2$?

Sim!!!

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- Seja **P = (p_1 , p_2, …, p_n)** um conjunto de **n** pontos em $\Re^2$
  - Se o número de pontos for menor que três, então o problema é resolvido através dum algoritmo de força bruta em tempo constante.
  - Caso contrário em cada nível da **recursividade**, executar as fases **Dividir**, **Conquistar** e **Combinar**, como passamos a descrever a seguir:

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

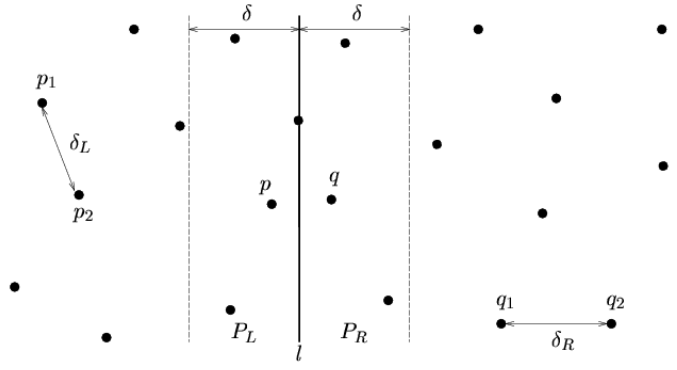
- **Dividir:**
  - Particione P em P_L e P_R tal que cada ponto de P_L este à esquerda de cada ponto de P_R
  - Em **pré-processamento ordene** os pontos de P segundo as suas *x*-coordenadas

  ⇒ em cada nível da *recursividade*, a fase **Dividir** é ***O(n)***

13

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- **Conquistar:**
  - O problema deve ser resolvido *recursivamente* para P_L e P_R, obtendo assim δ_L e δ_R, as distâncias mínimas entre pares de pontos $\{p_1,p_2\}$ em P_L e $\{q_1,q_2\}$ em P_R respectivamente.

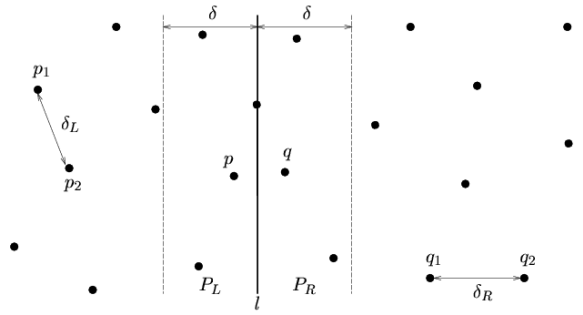


14

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- **Combinar:**
  - Seja $\delta = \min\{\delta_L , \delta_R\}$ e seja l uma recta tal que todo ponto de P_L está à esquerda ou sobre l e todo ponto de P_R está à direita ou sobre l
  - Pode ocorrer que:
    - o par mais próximo {p, q} de P seja tal que $p \in P_L$ e $q \in P_R$

      ⇨ este par deve estar na faixa de largura 2δ que tem como centro a recta l



15

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- **Combinar (cont…):**
  - Seja P_L' - sub-colecção dos pontos de P_L que estão a uma distância menor do que δ da linha l e seja P_R' a correspondente sub-colecção de pontos de P_R.
  - Para cada $p \in P_L'$ determinar os pontos $q_j \in P_R'$ tal que $d(p, q_j) < \delta$

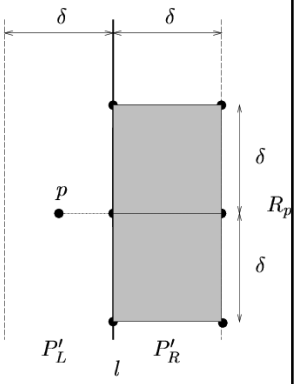
  ⇨ os $q_j \in R_p$ (pontos de P_R' que estão no rectângulo R_p de altura 2δ e largura δ)

  No máximo quantos pontos de P_R' podem estar contidos no tal rectângulo?

  ⇨ **No máximo 6!** Em caso contrario existe um par de pontos em P_R cuja distância seria menor ou igual que δ o que não pode ocorrer.

  ⇨ ∀ p precisamos calcular a distância em no máximo 6 pontos de P_R'

  ⇨ **O número de distâncias que teremos que calcular na fase de COMBINAÇÃO é não superior a 6n.**



16

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- **Algoritmo PmP2(S)**

  Dado um conjunto $S = (p_1, p_2, ..., p_n)$ de **n** pontos em $\Re^2$ devolver a distância mínima entre pontos de **S**

  - **Ordenar** os pontos de **S** por abcissa e armazene num vector V_x
  - **Ordenar** os pontos de **S** por ordenada e armazene num vector V_y
  - **Retornar** o par obtido pelo algoritmo **PmP2-Aux(S)**

17

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

Algoritmo PmP2-Aux(S)

1. Se $|S| = 2$ então retornar (p_1, p_2)

2. Senão calcular a mediana m_x das abcissas de S. Seja r a recta vertical com abcissa m_x

3. Dividir S em duas colecções S_1 e S_2 com $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$ pontos, respectivamente. Todos os pontos de S_1 estão à esquerda de (ou sobre) a recta r e os pontos de S_2 estão à direita de (ou sobre) a recta r

4. Obter recursivamente o Par mais Próximo

de S_1: (p_{i1}, p_{j1}) = PmP2-Aux(S_1)

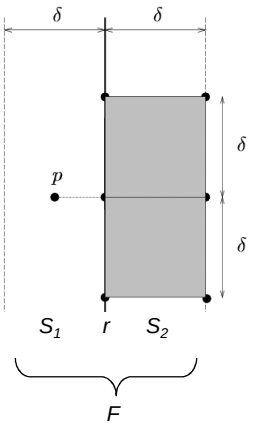
de S_2: (p_{i2}, p_{j2}) = PmP2-Aux(S_2)

5. Seja $\delta = \min \{d(p_{i1}, p_{j1}), d(p_{i2}, p_{j2})\}$

6. Seja F a faixa de largura 2δ centrada na recta r. Procure por varrimento vertical o par de pontos $p_k \in S_1$ e $p_l \in S_2$ mais próximos (i.e. $\forall\ p_k$ procuramos o par mais próximo em S_2 tal que $| p_{ky} - p_{ly} | < \delta$)

7. Retornar:

$\mathbf{\min\{d(p_{i1}, p_{j1}), d(p_{i2}, p_{j2}), d(p_1, p_k))\}}$



18

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

Algoritmo PmP2-Aux(S)

$O(1)$ ⇨ **1.** Se $|S| = 2$ então retornar (p_1, p_2)

$O(1)$ ⇨ **2.** Senão calcular a mediana m_x das abcissas de S. Seja r a recta vertical com abcissa m_x

Dividir: $O(n)$ ⇨ **3.** Dividir S em duas colecções S_1 e S_2 com $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$ pontos, respectivamente. Todos os pontos de S_1 estão à esquerda de (ou sobre) a recta r e os pontos de S_2 estão à direita de (ou sobre) a recta r

4. Obter recursivamente o **Par mais Próximo**

de S_1: (p_{i1},p_{j1}) = PmP2-Aux(S_1)

de S_2: (p_{i2},p_{j2}) = PmP2-Aux(S_2)

$O(1)$ ⇨ **5.** Seja $\delta = \min \{d(p_{i1},p_{j1}), d(p_{i2},p_{j2})\}$

Combinar: $O(n)$ ⇨ **6.** Seja F a faixa de largura 2δ centrada na recta r. Procure por varrimento vertical o par de pontos $p_k \in S_1$ e $p_l \in S_2$ mais próximos (i.e. $\forall\ p_k$ procuramos o par mais próximo em S_2 tal que $|p_{ky} - p_{ly}| < \delta$)

$O(1)$ ⇨ **7.** Retornar:

$\min\{d(p_{i1},p_{j1}), d(p_{i2},p_{j2}), d(p_1,p_k))\}$

19

Par mais Próximo: *divisão-e-conquista* em $\Re^2$

- **Teorema:** Se um ***algoritmo recursivo*** do tipo ***dividir-para-conquistar*** para resolver um problema decompõe uma instância de tamanho ***n*** em duas instâncias de tamanho ***n/2*** do mesmo problema, e se o processamento necessário à execução das etapas de **DIVISÃO** e **COMBINAÇÃO** tem, no total, complexidade ***O(n)***, então o algoritmo resultante tem complexidade ***O(n log n)***

- **Teorema:** *O problema do par mais próximo entre n pontos no plano pode ser resolvido com um algoritmo de complexidade $O(n \log n)$ e este é o algoritmo óptimo.*
- **Teorema:** *A determinação do par mais próximo entre um conjunto de n pontos no espaço $\Re^d$ pode ser resolvido em tempo $O(n \log n)$ e este é o algoritmo óptimo.*

20