Pontos mais próximos
Projeto e Análise de Algoritmos

Eric Azevedo de Oliveira[1],

[1]Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica Minas Gerais

1. Pontos mais próximos

O problema dos Pontos mais próximos consiste em um conjunto de **n** pontos em um plano com o intuito de encontrar o par de pontos mais proximos.

1.1. Representação

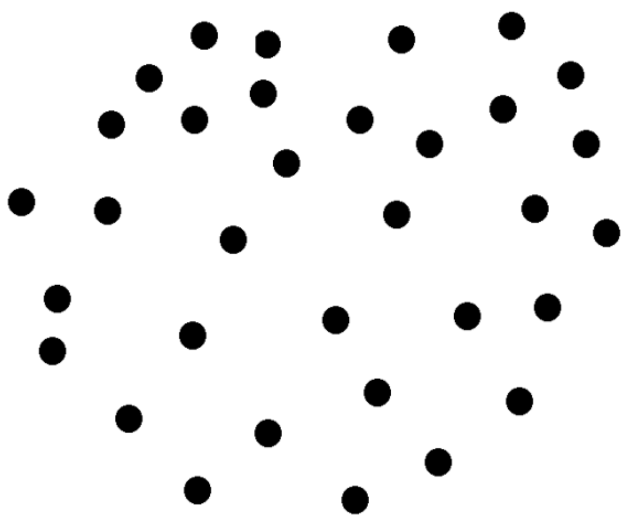


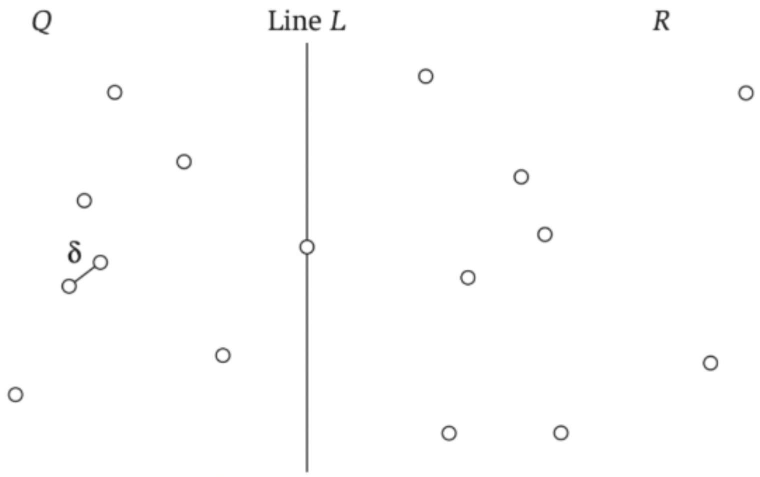
Figura 1. Pontos no Plano.

2. Sobre o Documento

Esse documento será dividido em oito partes abaixo discriminadas: [**2.1**] referenciando a máquina utilizada, [**2.2**] e [**2.3**] serão relacionados a dois diferentes custos computacionais na procura dos pontos mais próximos, sendo eles **O**$(nlogn)$ e **O**(n^2) com seus códigos, e a seção [**2.4**] sera as comparações desses dois métodos com os resultados de ambos, [**2.5**] como compilar e executar o código e a [**2.6**] como o trabalho foi separado.

2.1. Máquinas Utilizada

Processador: i5-3317U (4).
Memória : 8Gb.
GPU : Intel 3rd gen Core processador Grap.



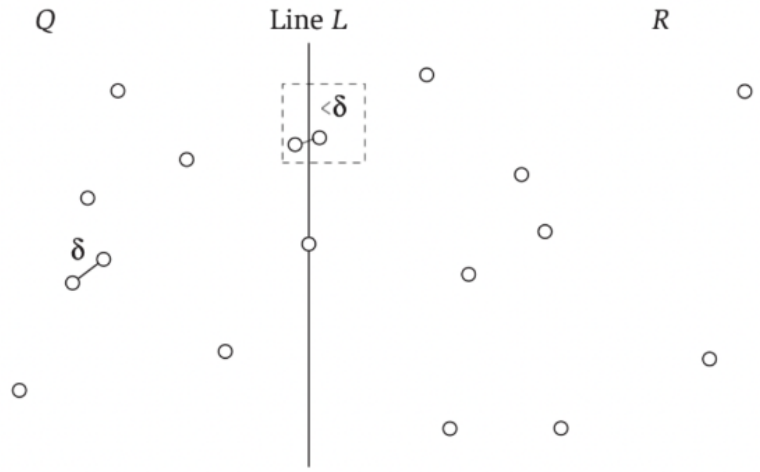
Algorithm design —Jon Kleinberg, Éva Tardos

2.2. O($nlogn$)

Com objetivo de alcançar a complexidade de **O**($nlogn$), foi utilizado a estratégia de divisão e conquista, na qual se consiste em pegar o problema completo e dividi-lo em vários problemas menores, os quais são independentes, para não processar **n** dados com outros **n**.

Em virtude da divisão e conquista, com base na figura a cima, nosso plano foi dividido em 2 quadrantes, o **R** e o **Q** , nessa divisão é realizado somente a ordenação por meio do algorítimo heapSort do array contendo os pontos do eixo **X** e lançado a mão ao ponto que está caracterizado na metade desse array.

Porém analisando somente analisando o eixo **X**, temos um problema que aumenta a complexidade final de nosso algorítimo, conforme a imagem a seguir demonstra.



Algorithm design — Jon Kleinberg, Éva Tardos

Tentando evitar esse aumento de complexidade, além da realização da ordenação do eixo **X**, analisaremos a ordenação do eixo **Y**, tendo o foco de calcular menor distância de ambos quadrantes, por meio da recursão que é chamado delta.

Entretanto a utilização pos ordenacao do **Y** não dispensará que alguma distância na volta da recursão seja desconsiderada, impedindo que seja feita o cálculo desnecessário da distância dos pontos dos quadrantes com a distância delta.

Conforme relatado acima, utilizando a divisão e conquista, foi obtido as complexidades do heapShort() que é **O**($nlogn$), quando reconstruimos o heap, e na utilização do algoritímo na parte da recursão , utilizando os eixos **X** e **Y**, é obtido a complexidade de **O(n)**. Por esse motivo o algoritímo terá uma ordem de complexidade de **O**($nlogn$).

Algorithm 1 Divisão e conquista

$quantidade \leftarrow entrada$
$Pontos[] \leftarrow rand() * quantidade$
$y[] \leftarrow HeapSort(Pontos.Y)$
$X[] \leftarrow HeapSort(Pontos.X)$
$meio \leftarrow EncontrarPontoMID()$
while $quantidade \neq 0$ **do**
  if $dir - esq$= =1 **then**
return resposta
  else if $dir - esq$= =2 **then**
return resposta
  end if
  $resposta = divisao_Conquista() \leftarrow divisao_Conquista()$
end while
return resposta =0

2.3. O(n^2)

Para alcançar a complexidade de **O**(n^2), é utilizado o algorítimo de força bruta, no qual irá calcular todas as distâncias entre pares de pontos e selecionar o par que tiver menor distância. Tendo a fórmula matemática para **n** pontos, descrita como: $n2$

$$\frac{n!}{(n-2)! * 2!} = O(n^2)$$

Algorithm 2 força bruta

$quantidade \leftarrow entrada$
$Pontos[] \leftarrow rand() * quantidade$
$X \leftarrow 0$
while $quantidade \neq X$ **do**
  $Y \leftarrow X$
 for $quantidade \neq Y$ **do**
   if $Ponto[X]$ ==$Ponto[Y]$ **then**
    $menorDistancia \leftarrow distancia(Ponto[Y], Ponto[X])$
   end if
 end for
  end while
 return menorDistancia =0

2.4. Resultados

O(n^2)

Pontos	tempo
10	0,055193s
1000	0,960028s
10000	5,399122s
100000	529,0015s

O($nlogn$)

Pontos	tempo
10	0,000197s
1000	0,000237s
10000	0,000956s
100000	0,005584s

Com os resultados, utilizando a ideia de divisão e conquista, foi possível obter uma diferença gigantesca em relação ao tempo, contra os de força bruta, mas, em contrapartida o algorítimo de força bruta é muito mais simples de ser implementado, e quando utilizamos uma quantidade de pontos pequenas ele mesmo demorando um tempo maior seria funcional, pois ele da solução ótima.

2.5. Execução

Para executar o código basta compilar no terminal, e seguir as instruções que forem aparecendo.

2.6. Separação do trabalho

A separação do trabalho entre os integrantes do grupo foi: Eric Azevedo de Oliveira - Algoritmo e texto.

Referências

Jon Kleinberg and Eva Tardos. 2006. *Algorithm design*. Pearson Education India.