Universidade de São Paulo
IME-USP

Relatório Final
MAC0331

Gabriel Fernandes de Oliveira - 9345370
Luis Gustavo Bitencourt Almeida - 9298207

1 Projeto

O projeto implementado foi o do par de pontos mais próximos. Programamos, como orientado, as quatro soluções: força bruta, o algoritmo aleatorizado de Golin et all descrito no livro de Kleinberg e Tardos[1], o algoritmo de divisão e conquista de Shamos e Hoey[2] visto em sala e um algoritmo de linha de varredura[3].

2 Implementação

Não serão tratados aqui os algoritmos força bruta e divisão e conquista, que foram abordados em classe.

2.1 Linha de varredura

A ideia do algoritmo é analisar os pontos da entrada em ordem crescente de coordenada x, mantendo um par de pontos mais próximo dentre os pontos analisados até o momento bem como a distância δ entre esses dois pontos. Para tanto, o algoritmo mantem em uma árvore de busca binária (uma Treap na nossa implementação) com os pontos já visitados cuja coordenada x difere de no máximo δ do ponto que está sendo analisado agora. A chave utilizada na ABBB é a coordenada y do ponto. Caso dois pontos tenham a mesma coordenada y, usa-se a coordenada x para desempatá-los.

A análise de cada ponto p se dá em três etapas:

1. Remove-se da ABBB todos os pontos com coordenada x menor que $p.x-\delta$, pois é impossível que esses pontos façam parte de um par de pontos cuja distância seja menor que δ.

2. Verifica-se se algum dos pontos armazenados na ABBB, cuja coordenada y difere de $p.y$ de no máximo δ, forma um par com o ponto p à distância menor que δ, atualizando δ se for o caso.

3. Insere-se p na ABBB.

Cada remoção no passo 1 custa tempo esperado $\mathcal{O}(\lg n)$, perfazendo um total de tempo esperado de $\mathcal{O}(n \lg n)$, pois cada ponto é removido uma única vez. Similarmente, o passo 3 consome tempo esperado $\mathcal{O}(\lg n)$ por ponto inserido, totalizando também tempo esperado $\mathcal{O}(n \lg n)$.

Para realizar o passo 2, primeiramente busca-se na ABBB o ponto q sucessor a $(p.x - \delta, p.y - \delta)$. Repetidamente calcula-se a distância entre p e q, movendo q para o seu sucessor na ABBB, até atingir um ponto maior, na ordem da ABBB, que o ponto $(p.x, p.y + \delta)$. O ponto p deverá ser comparado por todos os pontos da área cinza escura ilustada na figura 1. Durante esse processo, o valor de δ é atualizado bem como o par de pontos mais próximo corrente, se for o caso.

A busca inicial e as buscas de sucessor na ABBB custam tempo esperado $\mathcal{O}(\lg n)$ e são feitas no máximo 8 outras buscas por sucessor por uma análise

análoga à feita no algoritmo de divisão e conquista. Sendo assim, o tempo total esperado do passo 2 também é $\mathcal{O}(\lg n)$, totalizando assim um consumo de tempo total esperado $\mathcal{O}(n \lg n)$.

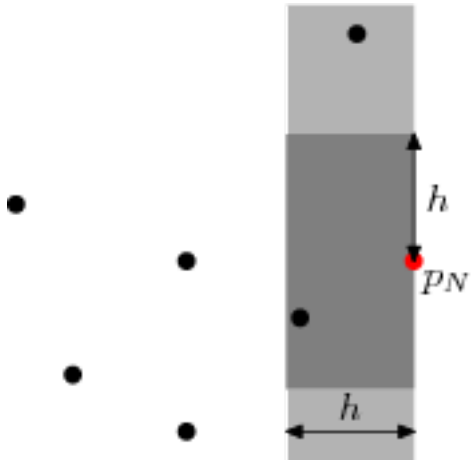


Figura 1: Exemplo do passo 2 de análise do ponto p_n

O passo 3, que consiste de uma simples inserção na ABBB, também ocorre em tempo $\mathcal{O}(\lg n)$

Portanto, como cada uma das três etapas da análise do line sweep é feita em complexidade $\mathcal{O}(\lg n)$ e essa análise vale para cada um dos pontos da entrada, concluímos que o algoritmo final roda em tempo $\mathcal{O}(n \lg n)$.

2.2 Algoritmo randomizado

A implementação e análise desse algoritmo foram totalmente baseadas no livro de Kleinberg e Tardos [1], no capítulo 13, subseção 7.

Incialmente, se realiza um embaralhamento aleatório dos pontos recebidos na entrada. Assume-se, por simplicidade, que os pontos na ordem aleatória estabelecida são $p_1, p_2, \ldots, p_n$.

O algoritmo randomizado se organiza em etapas. Para cada etapa, define-se um valor δ, a distância mínima de um par de pontos analisada até o momento, e testa-se a hipótese de que δ é a menor distância entre todos pares de pontos.

Caso o teste confirme a hipótese, o programa é finalizado, retornando o valor atual de δ. Do contrário, atualiza-se o valor de δ, e repete-se o mesmo teste.

No início do algoritmo, δ é iniciado com a distância entre os pontos p_1 e p_2. Antes de se executar a rotina de teste, verifica-se se o valor de δ é zero. Se sim, o problema já está solucionado, pois não há como existir um par de pontos com distância negativa. Do contrário, executa-se normalmente a rotina de teste:

O primeiro passo consiste na subdivisão do plano em quadrados de lado $\delta/2$, formando uma grade, como representado na figura 2.

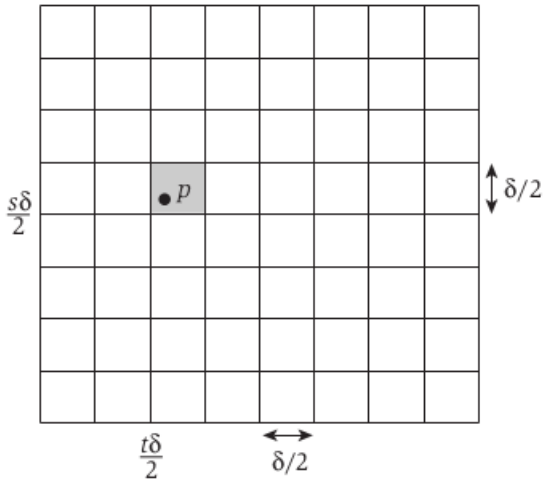


Figura 2: Subdivisão do plano

Em seguida, são analisados os pontos na ordem aleatória estabelecida. Para cada ponto, deve-se verificar se o mesmo dista menos que δ de algum ponto já adicionado.

Define-se como Q_{st} o quadrado da subdivisão de plano localizado na coluna s e linha t.

$$Q_{st} = \{(x, y) : s\delta/2 \leq x < (s+1)\delta/2;\ t\delta/2 \leq y < (t+1)\delta/2\}$$

Lema 2.1. *Para verificar se um ponto p_i atualiza δ basta avaliar a distância de p_i aos pontos já analisados que estiverem localizados em quadrados $Q_{s't'}$, onde $|s - s'| \leq 2$ e $|t - t'| \leq 2$.*

Demonstração. Considere um ponto p_j, que pertença a um quadrado $Q_{s't'}$ que não satisfaz a condição do lema em relação a um ponto p_i, pertencente ao quadrado Q_{st}, isto é, não vale que $|s - s'| \leq 2$ e $|t - t'| \leq 2$.

Sendo assim, conclui-se que, ou $|s - s'| > 2$ ou $|t - t'| > 2$. De qualquer modo, isso implica que entre o quadrado Q_{st} e $Q_{s't'}$, devem existir pelo menos dois outros quadrados.

O que, por sua vez, implica que os pontos p_i e p_j, distarão, mais que o dobro do lado de um quadrado, que vale, exatamente δ.

Deste modo, não há como p_i e p_j atualizarem o valor da distância mínima, δ, atual.

Devem considerar-se assim, apenas os pontos pertencentes a quadrados que estão em quadrados que satisfazem a condição do lema apresentado. □

Porém, isso implica que, para cada ponto p_i é necessário apenas analisar-se um subquadrado 5x5 centrado no quadrado que contem p_i, comparando a

distância de p_i com os pontos presentes na área deste subquadrado.

Além disso, como provaremos, no máximo existirá um ponto já analisado em cada quadrado do plano sem que a distância mínima seja menor que δ. Portanto, para analisar cada um dos 25 quadrados próximos à p_i, no pior dos casos, comparar-se-a o ponto p_i com 25 outros pontos.

Demonstração. Imagine que o algoritmo está analisando o ponto p_i, que a menor distância entre um par de pontos já analisados é δ e que existe um quadrado na divisão do plano que possui dois pontos p_j e p_k já analisados, ou seja, $j, k < i$.

Neste caso, como p_j e p_k pertencem ao mesmo quadrado de lado $\delta/2$, então sua distância máxima será igual à diagonal do quadrado. Porém tal diagonal tem valor $\delta/\sqrt{2}$, que é menor que δ. Portanto p_j e p_k necessariamente tem que distar menos que δ um do outro. Chegamos assim a um absurdo, já que faz parte da hipótese que δ é a distância mínima entre os pontos já analisados.

Provamos assim que deve haver no máximo um ponto para cada quadrado em que o plano foi subdividido. □

Esta análise realizada nos dá uma cota superior para o número de operações realizadas, um pouco maior que a cota superior real, que pode ser encontrada com cálculos mais detalhados, porém ela serve para nos indicar que o número de comparações que cada ponto analisado fará é constante. Totalizando assim um tempo $\mathcal{O}(n)$ para realizar a análise dos n pontos da entrada.

Ao analisar um ponto p_i, há duas possibilidades, ou ele forma, com um dos pontos previamente analisados uma distância menor que δ, ou não.

No primeiro caso, é necessário que se atualize o valor de δ para a menor distância formada em um par contendo p_i. Esse procedimento é custoso, pois implica que a subdivisão feita do plano, usando o valor anterior de δ, seja refeita para o novo valor δ.

Com a atualização da subdivisão, se faz necessária a repetição da rotina descrita até o momento. Não é necessário, no entanto, realizar novamente a análise de cada ponto até o i-ésimo ponto, já que eles, com certeza, não conseguirão diminuir o valor de δ atualizado.

A única operação que se faz necessária para esses pontos é inserí-los na nova divisão de plano feita com o valor de δ atualizado. Usando uma tabela de hash, é possível inserir pontos nos seus respectivos quadrados em tempo médio $\mathcal{O}(1)$. Como a atualização do i-ésimo ponto implica na reinserção de i pontos na tabela de hash, gasta-se um tempo médio total $O(i)$ caso o i-ésimo ponto altere o valor de δ.

A outra possibilidade é que o i-ésimo ponto não altere o valor de δ, neste caso, simplesmente o inserimos no seu quadrado correspondente da divisão de plano já existente, o que pode ser feito em tempo médio $\mathcal{O}(1)$ usando uma estrutura de Tabela de Hash.

Finalmente, temos que o consumo de tempo final do algoritmo é:

$$n + \sum_i iX_i$$

Definimos as variáveis aleatórias X_i como 1 se o i-ésimo ponto altera o valor de δ, gastando assim tempo $\mathcal{O}(i)$, e 0 do contrário, gastando então um tempo constante.

Analisaremos agora a probabilidade $Pr[X_i = 1]$, assumindo que o i-ésimo ponto muda o valor de δ.

Lema 2.2. $Pr[X_i = 1] \leq 2/i$

Demonstração. Imagine os primeiros i pontos da entrada na ordem aleatória definida. Definimos que a menor distância entre quaisquer dois desses pontos é atingida pelos pontos p e q.

Para que o i-ésimo ponto, ao ser analisado, atualize a menor distância de um par, deve valer que $p_i = p$ ou $p_i = q$. Como os pontos estão em ordem aleatória, existe uma probabilidade igual a $2/i$ de que $p_i = p \vee p_i = q$. □

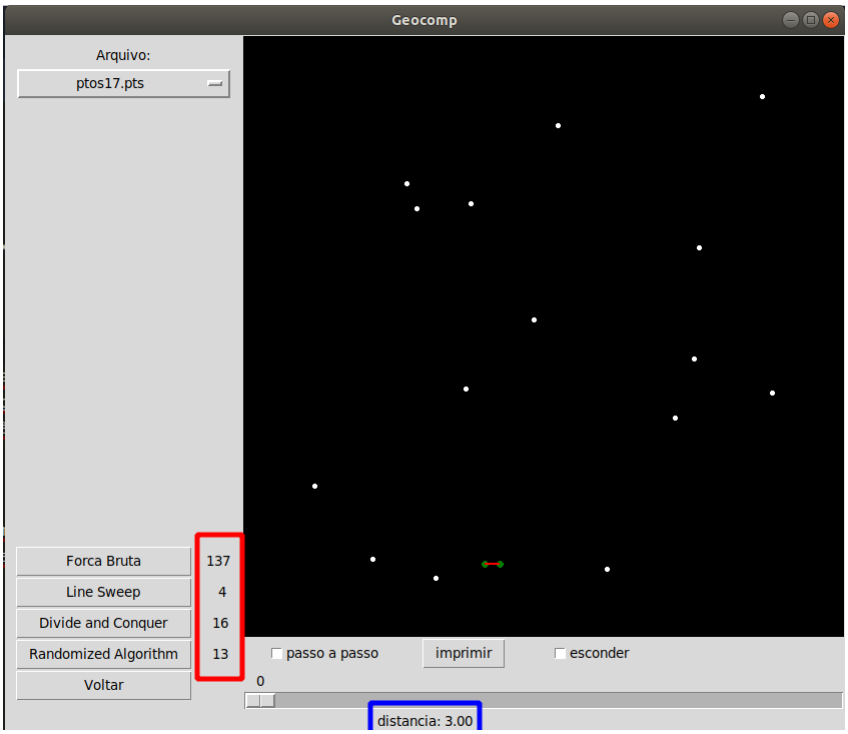
Podemos concluir assim que o tempo total esperado desse algoritmo será:

$$\begin{aligned} E[X] &= n + \sum_i iE[X_i] \\ E[X] &\leq n + 2n \\ E[X] &\leq 3n \end{aligned}$$

Temos então um algoritmo de tempo total esperado médio linear, caso se use uma implementação utilizando tabelas de hash de tempo médio constante por inserção e acesso.

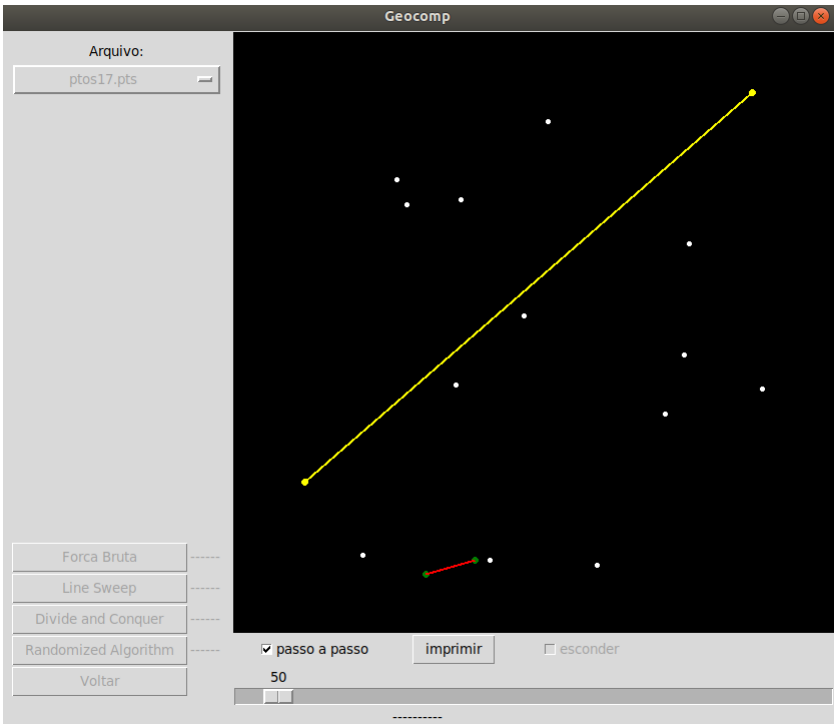
3 Animação

Nesta seção indicaremos como fizemos a escolha das cores da animação para cada um dos algoritmos implementados.

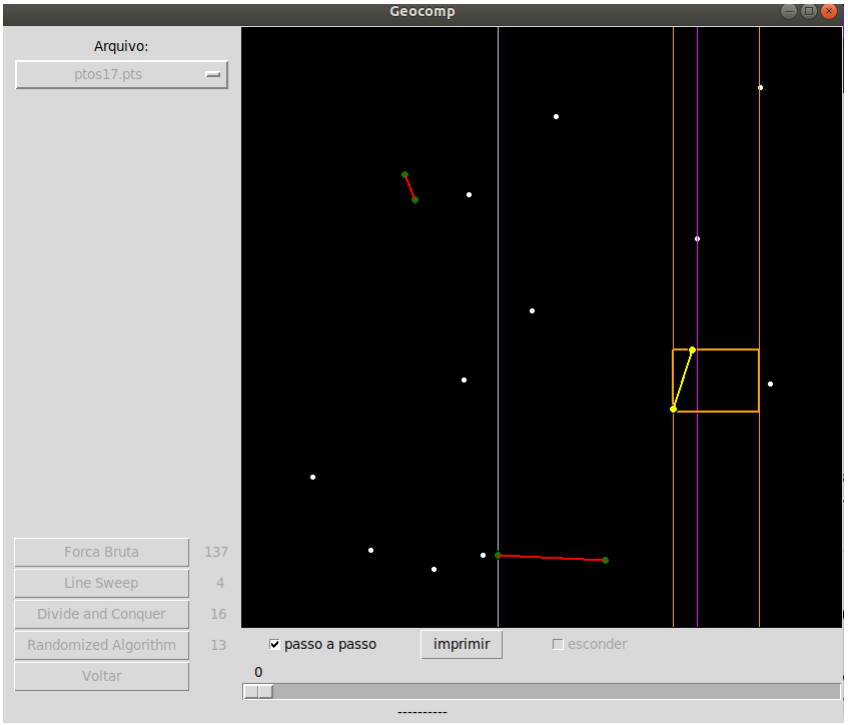


Em vermelho, segue indicado o número de chamadas à *dist*2 e em azul o valor da menor distância encontrado.

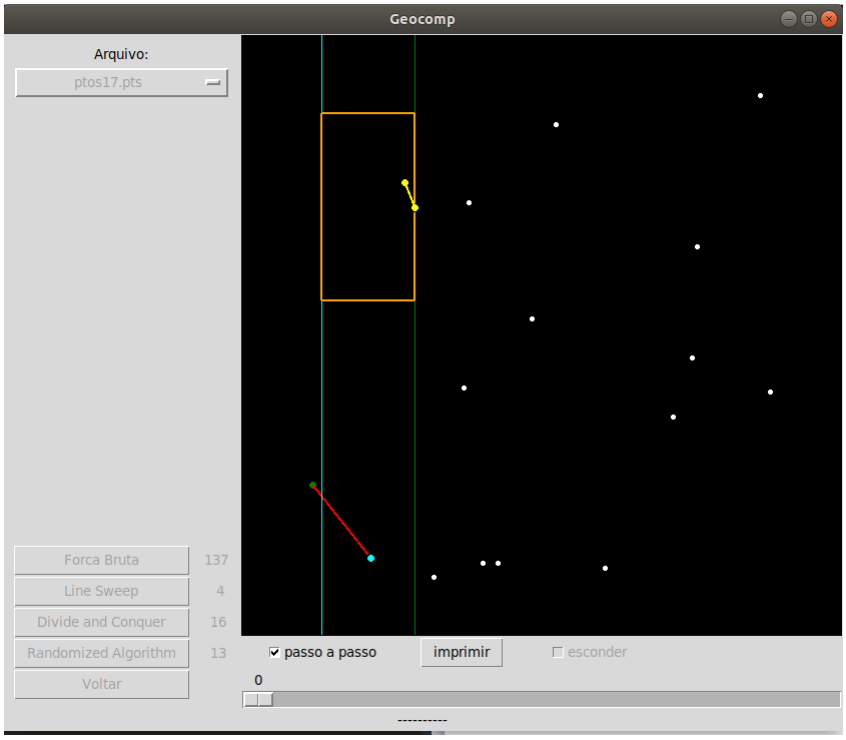
3.1 Força bruta



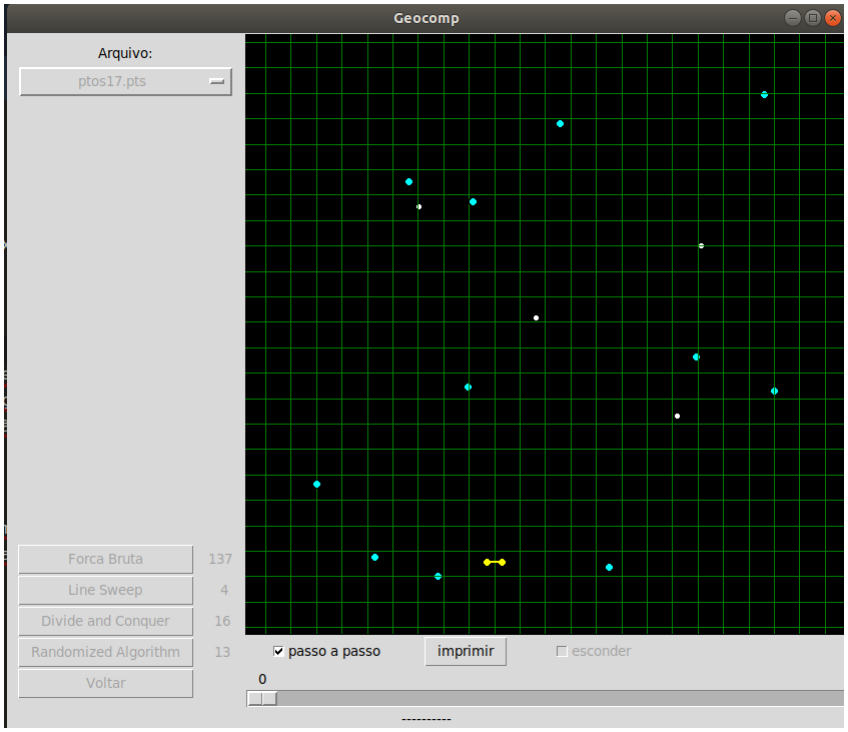
3.2 Divisão e conquista



3.3 Linha de varredura



3.4 Algoritmo randomizado



Referências

[1] Jon Kleinberg e Éva Tardos, *Algorithm Design*. Addison-Wesley Professional; 1ª Edição

[2] M. I. Shamos e D. Hoey. *Closest-point problems*. In Proc. 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 151—162, 1975 (DOI 10.1109/SFCS.1975.8)

[3] Post de bmerry sobre algoritmos de Line Sweep
`https://www.topcoder.com/community/competitive-programming/tutorials/line-sweep-algorithms/`