Geometria Computacional

Sweep line: algoritmos

Prof. Edson Alves

2019

Faculdade UnB Gama

Sumário

Par de pontos mais próximo

Par de pontos mais próximo

- Dado um conjunto S de N de pontos no plano bidimensional, o problema de encontrar o par de pontos mais próximo consiste em encontrar dois pontos $P, Q \in S$ tal que

$$\mathrm{dist}(P, Q) = \min\{\mathrm{dist}(P_i, P_j)\}, \quad \forall P_i \in S \quad \text{com} \quad i \neq j$$

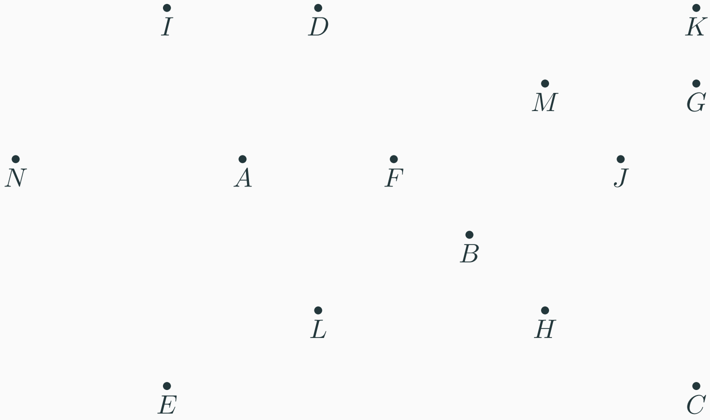
- Uma abordagem de busca completa consiste em computar as distância entre todos os pares de pontos possível, tendo complexidade $O(N^2)$
- Contudo, o problema pode ser resolvido em $O(N \log N)$ através do *sweep line*
- Os pontos deve ser ordenados em ordem lexicográfica

Par de pontos mais próximo

- Seja $d = \mathrm{dist}(P_1, P_2)$
- Agora, para todos pontos $P_3, P_4, \ldots, P_N$, deve-se computar todos os pontos vizinhos de $P_i = (x, y)$ tais que as coordenadas x estejam no intervalo $[x - d, x]$ e que as coordenadas y estejam no intervalo $[y - d, y + d]$
- Estes pontos podem ser identificados mantendo-se um conjunto de pontos cujas coordenadas estejam entre $[x - d, x]$, ordenado em ordem crescente de coordenada y
- Caso a distância de P_i para algum destes pontos seja inferior a d, o valor de d é atualizado e a varredura continua com este novo valor
- O ponto principal é que existem, no máximo, $O(1)$ pontos neste retângulo, o que faz com que a complexidade do algoritmo seja $O(N \log N)$, por conta da ordenação

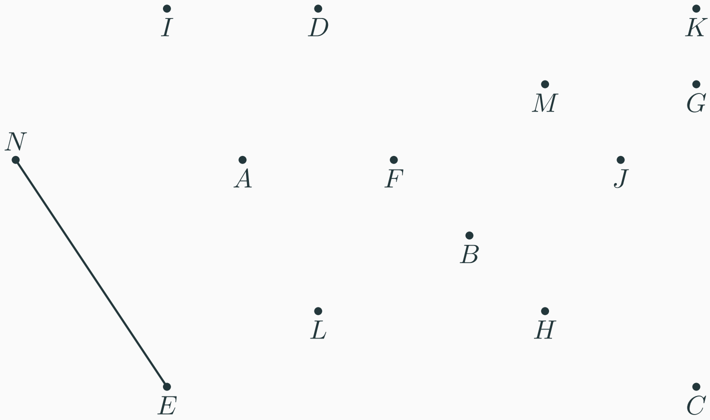
Visualização de identificação do par de pontos mais próximo

Par mais próximo: –



Visualização de identificação do par de pontos mais próximo

Par inicial, $\mathrm{dist}(N, E) = \mathbf{3.605551}$

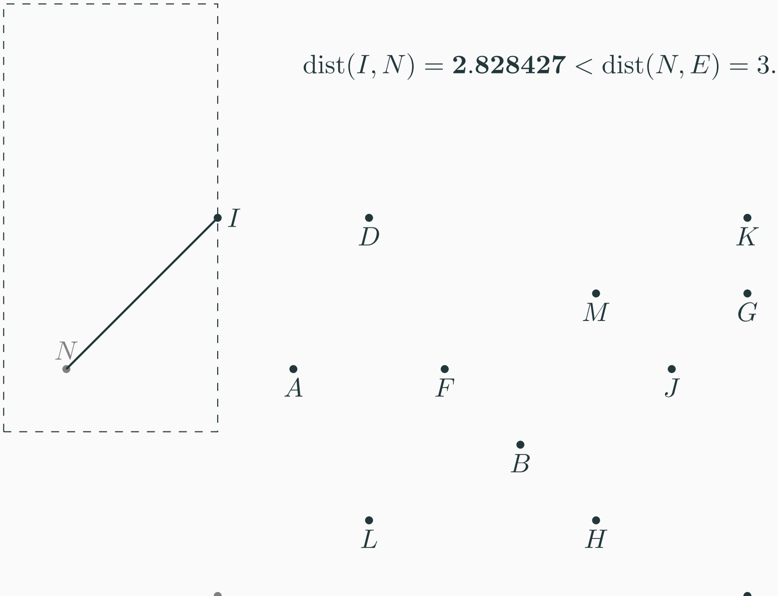


Visualização de identificação do par de pontos mais próximo

Avaliação do ponto I

Visualização de identificação do par de pontos mais próximo

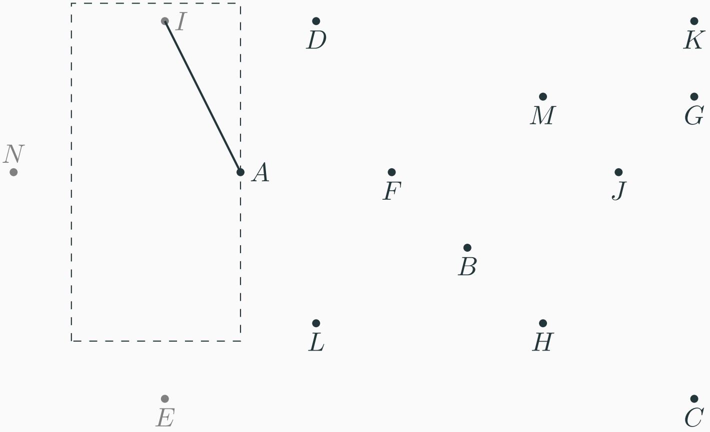
$$\mathrm{dist}(I, N) = \mathbf{2.828427} < \mathrm{dist}(N, E) = 3.605551$$



Visualização de identificação do par de pontos mais próximo

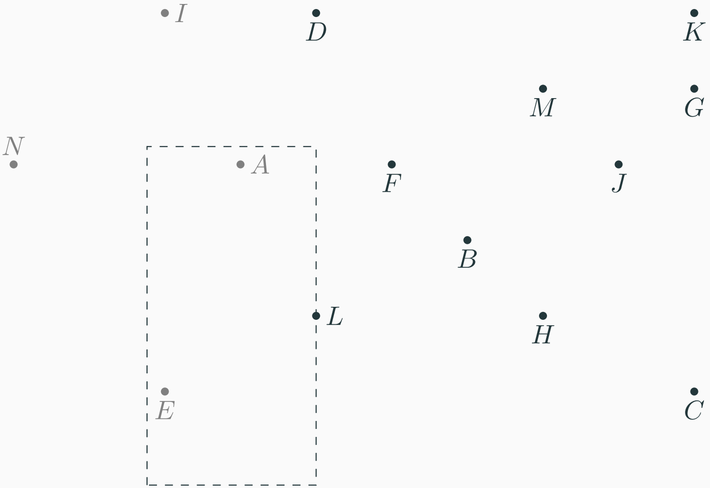
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(A, I) = \mathbf{2.236068} < \text{dist}(I, N) = 2.828427$$



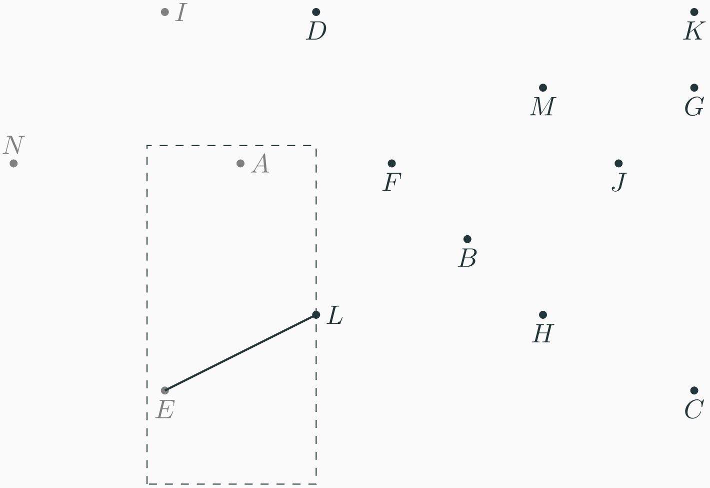
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto L



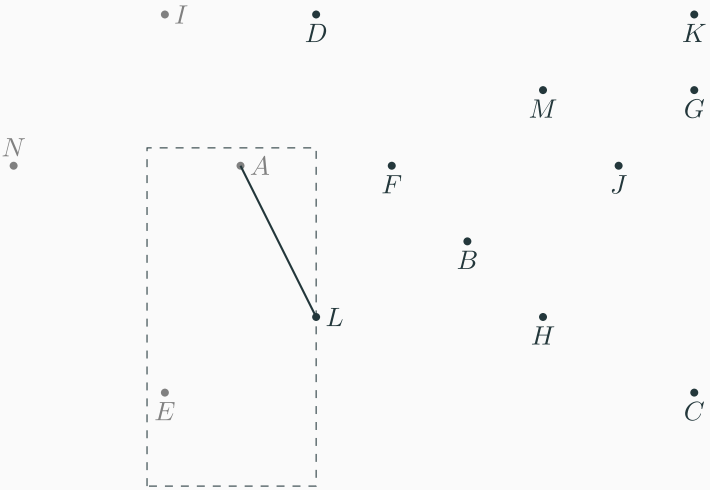
Visualização de identificação do par de pontos mais próximo

$$\mathrm{dist}(L, E) = \mathrm{dist}(A, I) = \mathbf{2.236068}$$



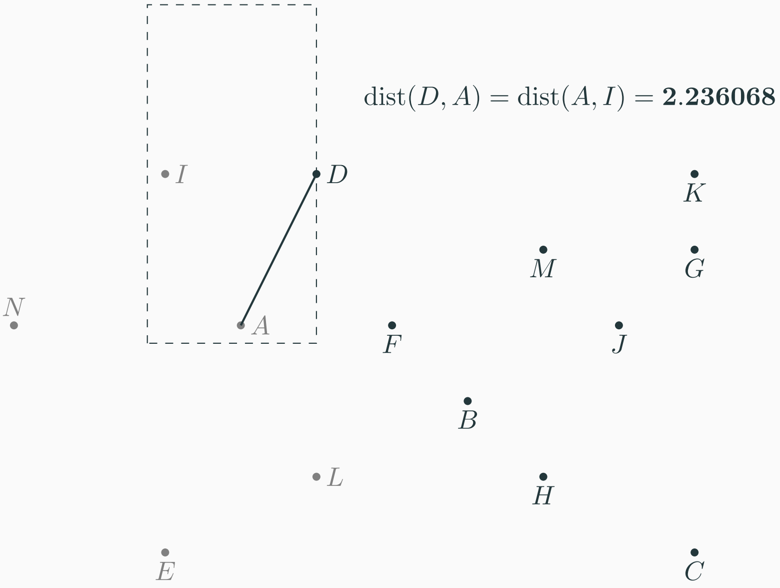
Visualização de identificação do par de pontos mais próximo

$$\mathrm{dist}(L, A) = \mathrm{dist}(A, I) = \mathbf{2.236068}$$

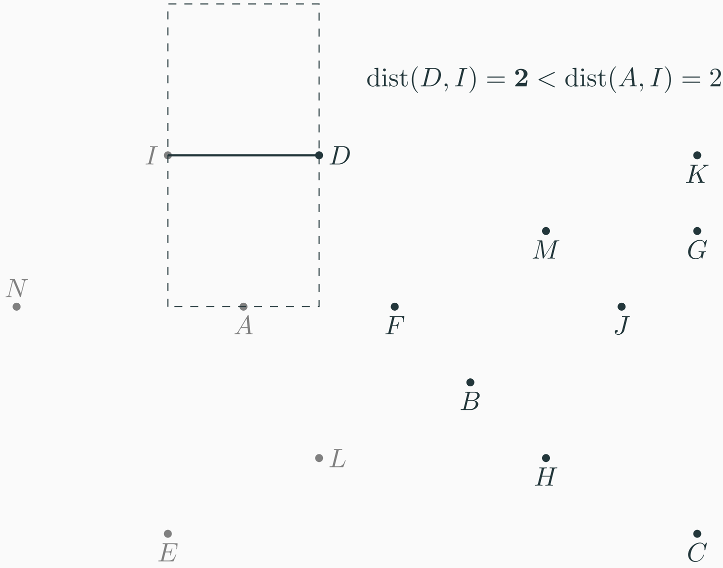


Visualização de identificação do par de pontos mais próximo

Visualização de identificação do par de pontos mais próximo



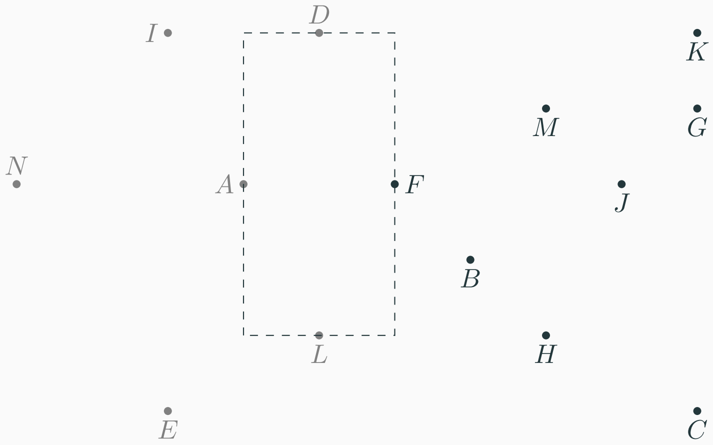
Visualização de identificação do par de pontos mais próximo



$$\mathrm{dist}(D, I) = \mathbf{2} < \mathrm{dist}(A, I) = 2.236068$$

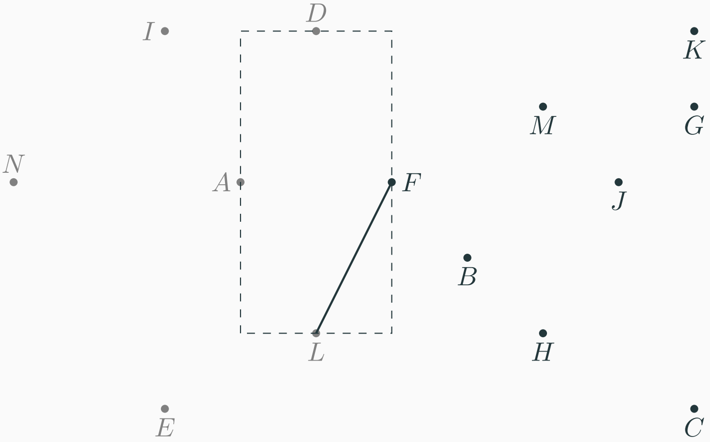
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto F



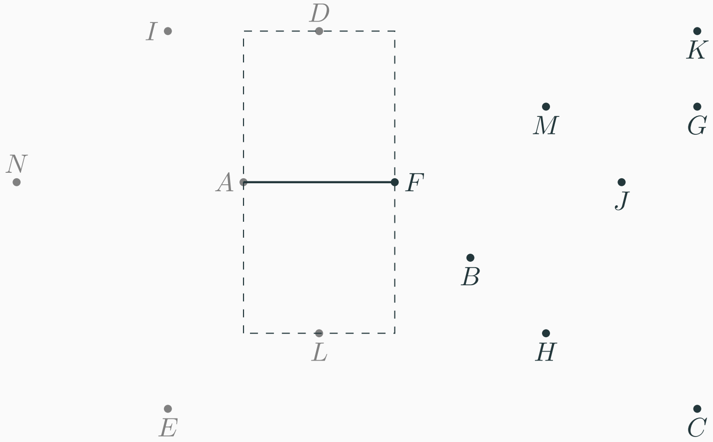
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(F, L) = 2.236068 > \text{dist}(D, I) = \mathbf{2}$$



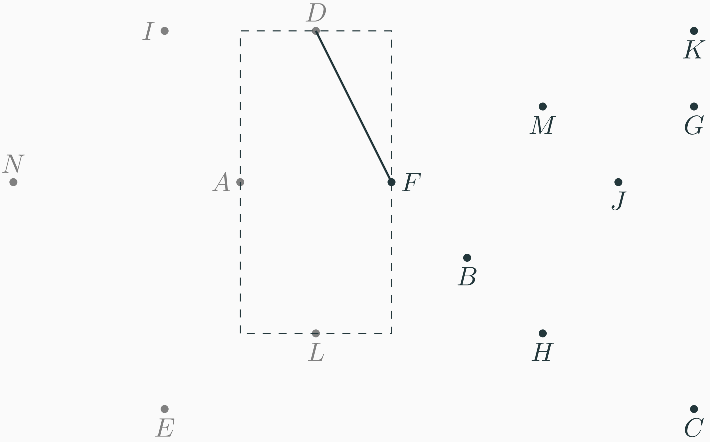
Visualização de identificação do par de pontos mais próximo

$$\mathrm{dist}(F, A) = \mathrm{dist}(D, I) = \mathbf{2}$$



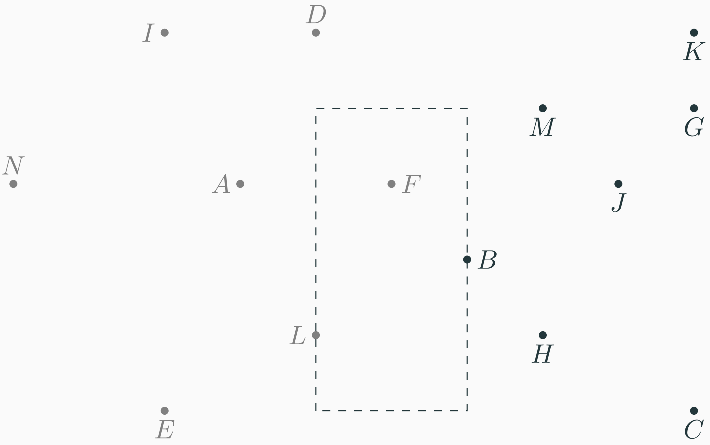
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(F, D) = 2.236068 > \text{dist}(D, I) = \mathbf{2}$$



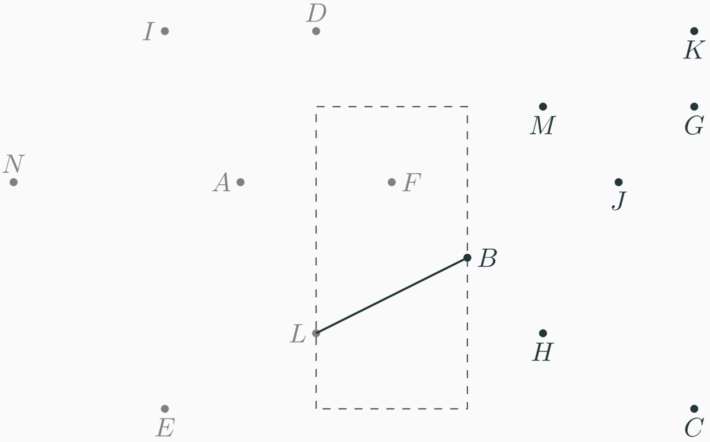
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto B



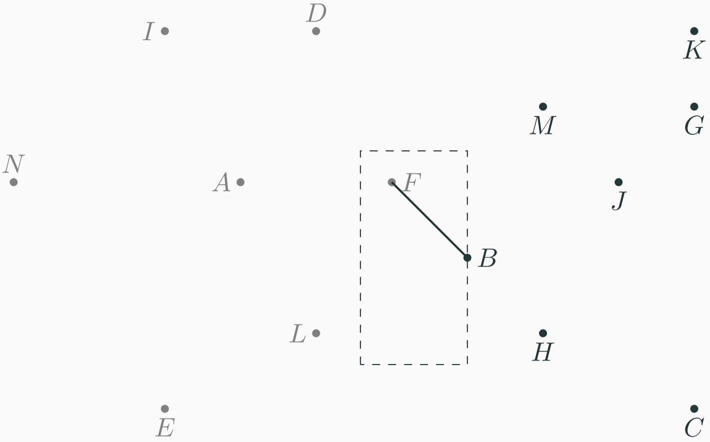
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(B, L) = 2.236068 > \text{dist}(D, I) = \mathbf{2}$$



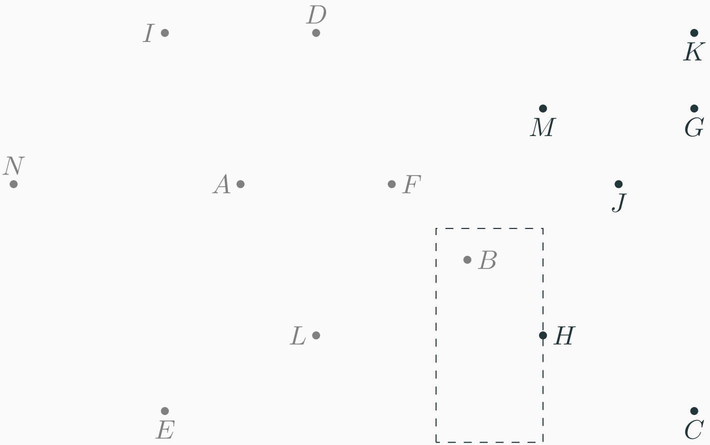
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(B, F) = \mathbf{1.414213} > \text{dist}(D, I) = 2$$



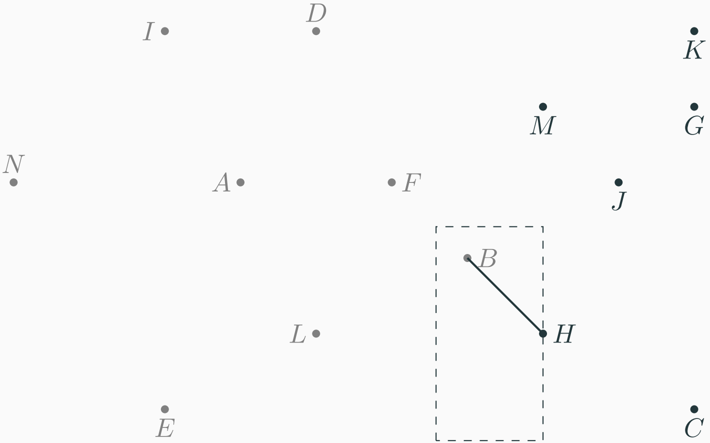
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto H



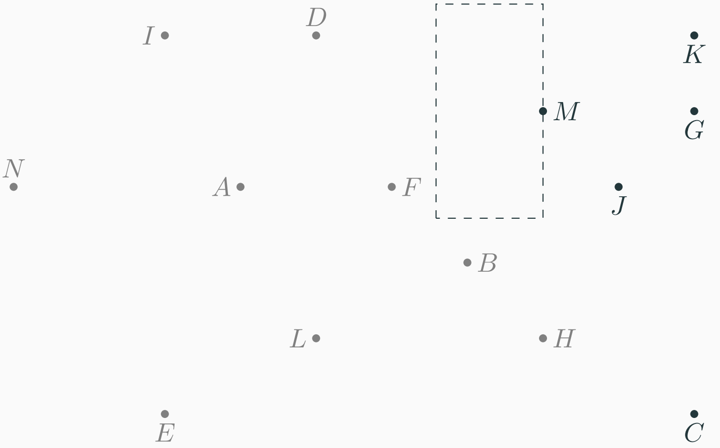
Visualização de identificação do par de pontos mais próximo

$$\mathrm{dist}(H, B) = \mathrm{dist}(B, F) = \mathbf{1.414213}$$



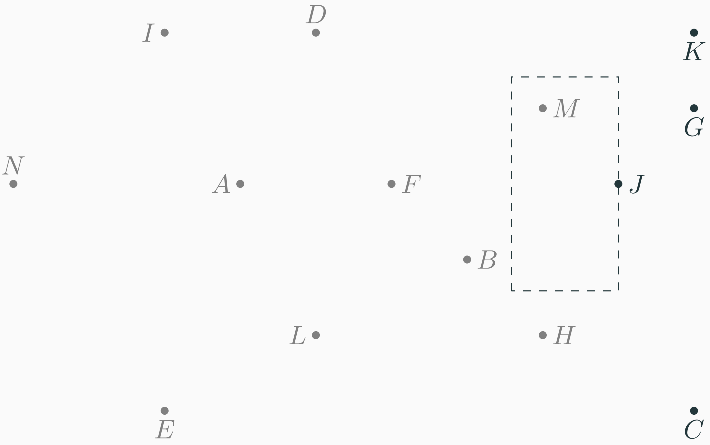
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto M



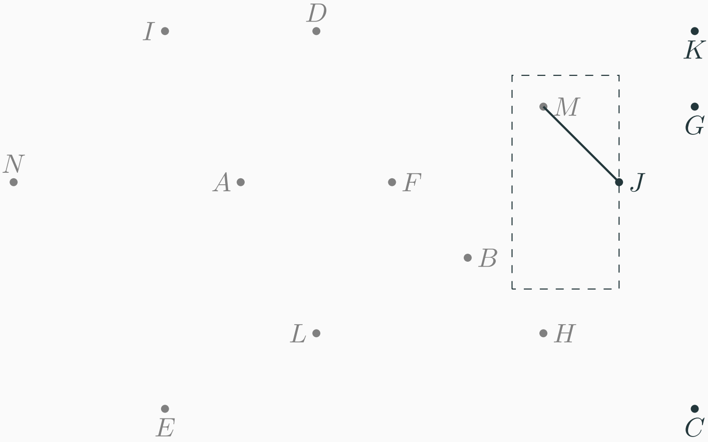
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto J



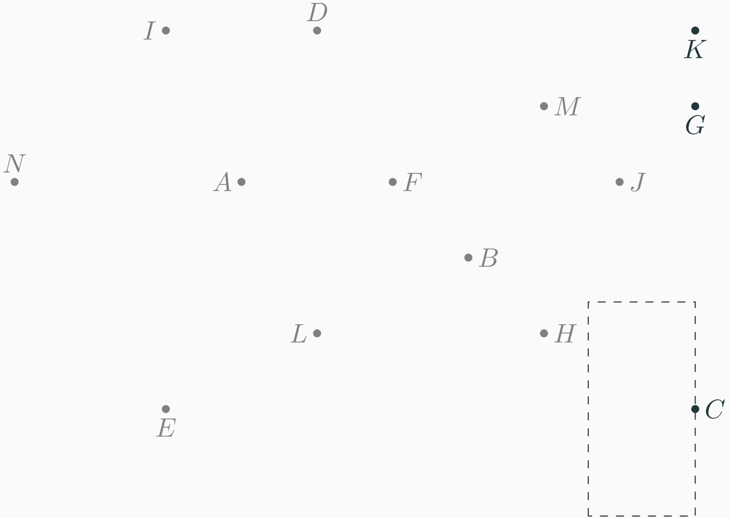
Visualização de identificação do par de pontos mais próximo

$$\text{dist}(J, M) = \text{dist}(B, F) = \mathbf{1.414213}$$



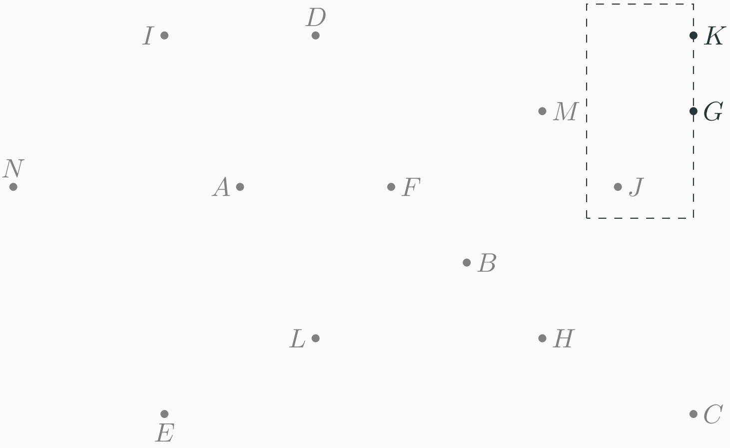
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto C



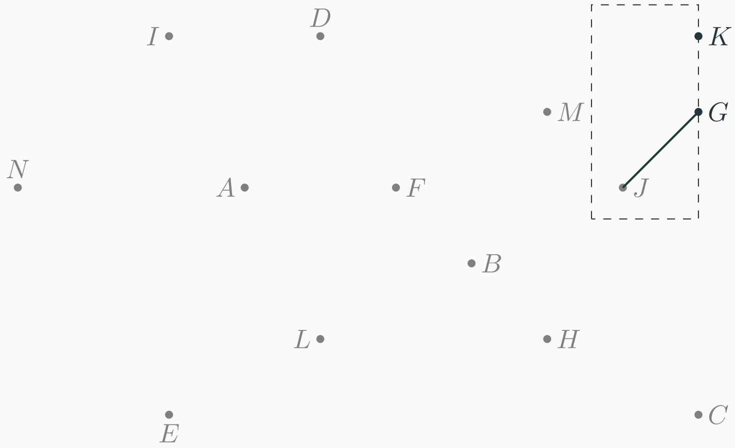
Visualização de identificação do par de pontos mais próximo

Avaliação do ponto G



Visualização de identificação do par de pontos mais próximo

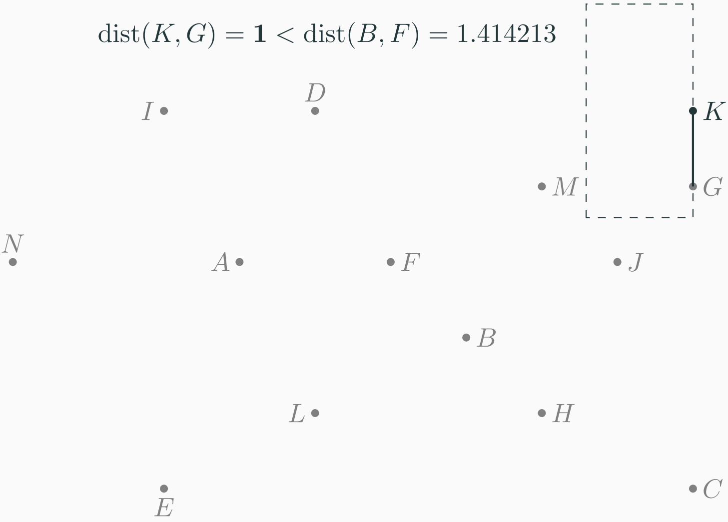
$$\text{dist}(G, J) = \text{dist}(B, F) = \mathbf{1.414213}$$



Visualização de identificação do par de pontos mais próximo

Visualização de identificação do par de pontos mais próximo

$$\text{dist}(K, G) = \mathbf{1} < \text{dist}(B, F) = 1.414213$$



Implementação da identifacação do par mais próximo

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<double, double>;
using point = pair<double, double>;

#define x first
#define y second

double dist(const point& P, const point& Q)
{
    return hypot(P.x - Q.x, P.y - Q.y);
}

pair<point, point> closest_pair(int N, vector<point>& ps)
{
    sort(ps.begin(), ps.end());

    // Assume que N > 1
    auto d = dist(ps[0], ps[1]);
    auto closest = make_pair(ps[0], ps[1]);
```

Implementação da identifacação do par mais próximo

```
    set<ii> S;
    S.insert(ii(ps[0].y, ps[0].x));
    S.insert(ii(ps[1].y, ps[1].x));

    for (int i = 2; i < N; ++i)
    {
        auto P = ps[i];
        auto it = S.lower_bound(point(P.y - d, 0));

        while (it != S.end())
        {
            auto Q = point(it->second, it->first);

            if (Q.x < P.x - d)
            {
                it = S.erase(it);
                continue;
            }

```

Implementação da identifacação do par mais próximo

```
            if (Q.y > P.y + d)
                break;

            auto t = dist(P, Q);

            if (t < d)
            {
                d = t;
                closest = make_pair(P, Q);
            }

            ++it;
        }

        S.insert(ii(P.y, P.x));
    }

    return closest;
}
```

Interseção de segmentos de reta

Problema

- O problema da interseção de segmentos de reta consiste em determinar se, em um conjunto S composto por N segmentos de reta, existe um par de segmentos $r, s \in S$ tal que $r \cap s \neq \emptyset$
- Uma variante comum é determinar todos os pontos de interseção entre estes segmentos
- A solução de busca completa testa cada elemento de S contra todos os demais
- Como a interseção entre dois segmentos pode ser obtida em $O(1)$ e existem $N(N-1)/2$ pares de segmentos distintos possíveis, esta abordagem tem complexidade $O(N^2)$
- Existe um algoritmo com menor complexidade para o problema apresentado, e algoritmos sensíveis à entrada para a variante

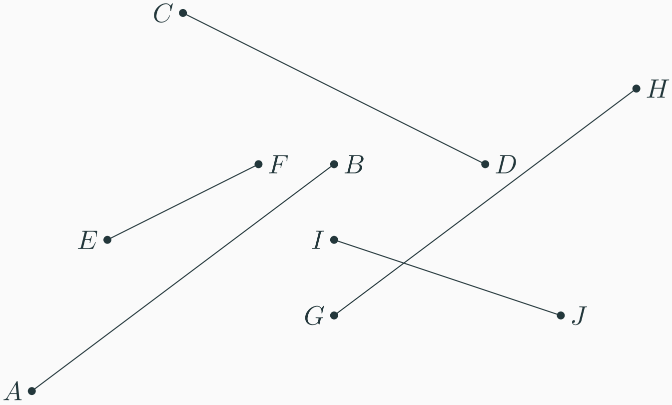
Algoritmo de Shamos-Hoey

- Shamos e Hoey propuseram, em 1976, um algoritmo capaz de determinar se existe ao menos uma interseção entre N segmentos de reta com complexidade $O(N \log N)$ e memória $O(N)$
- A ideia é ordenar os N segmentos do conjunto S em ordem lexicográfica e manter uma árvore binária balanceada A de segmentos ativos
- Cada segmento gera dois eventos: o ponto inicial do segmento gera um evento de inclusão de intervalo (1) e o ponto final do segmento um evento de exclusão do intervalo (2)
- A fila dos eventos deve ser ordenadas pelo ponto $P = (x_e, y_e)$ que deu origem ao evento
- Para cada evento, a árvore de segmentos ativos A deve estar ordenada pela coordenada y dos pontos dos segmentos com coordenada $x = x_e$

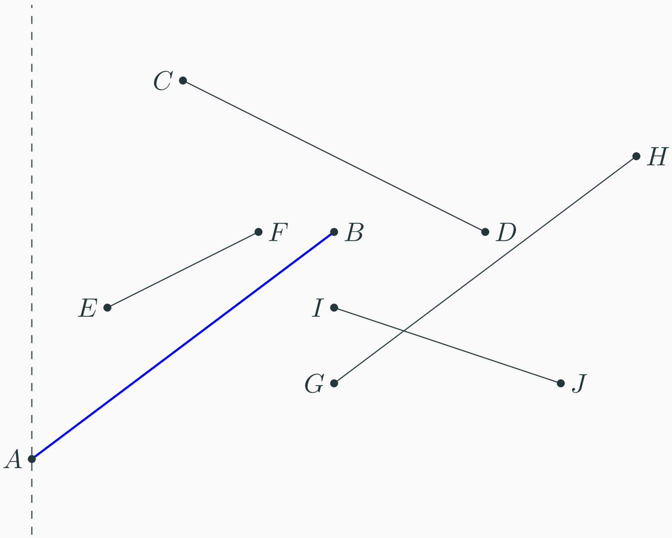
Algoritmo de Shamos-Hoey

- Para manter esta ordenação é necessário utilizar uma árvore binária de busca balanceada
- Uma alternativa é implementar tal árvore (por exemplo, uma árvore *red-black*)
- Outra alternativa é utilizar um `set` da linguagem C++, em conjunto com uma variável global que armazene o valor da coordenada x do evento atual e que seja utilizada na rotina de comparação
- Observe que, uma vez que um segmento r esteja abaixo de um outro segmento s em um ponto x, esta relação só mudará para valores maiores do que x caso exista uma interseção ambos
- No caso do algoritmo de Shamos-Hoey a existência de interseção é um critério de parada, logo não há necessidade de tratar tais casos

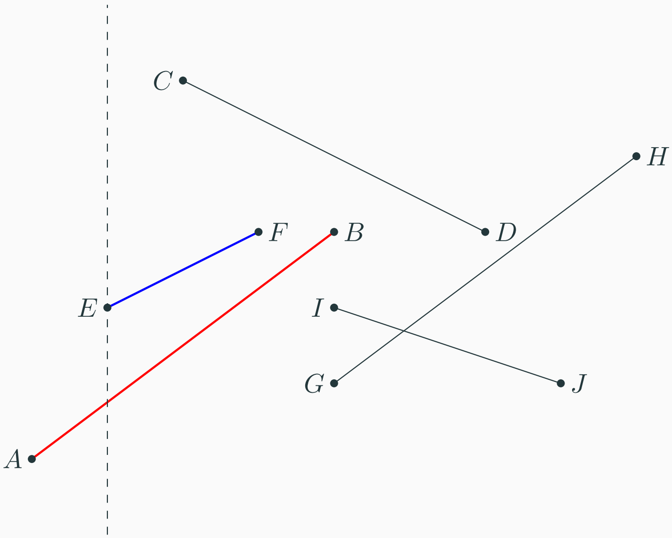
Visualização de identificação de interseção de segmentos



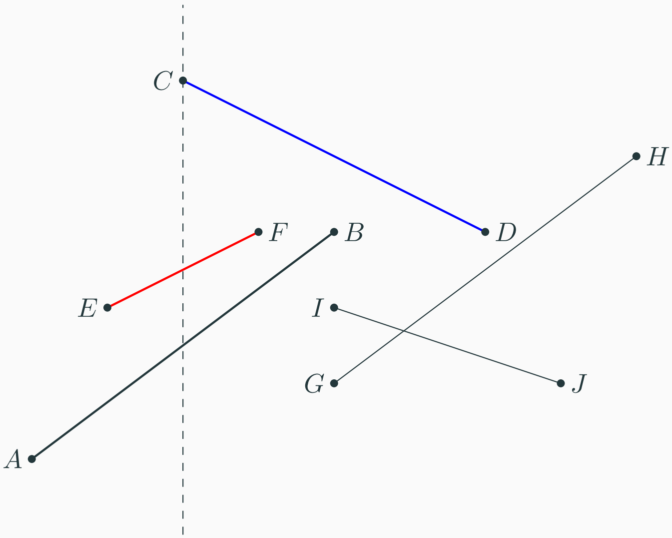
Visualização de identificação de interseção de segmentos



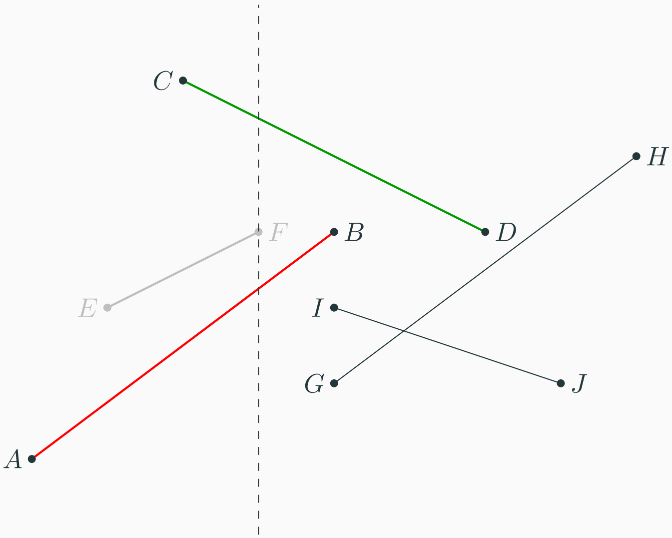
Visualização de identificação de interseção de segmentos



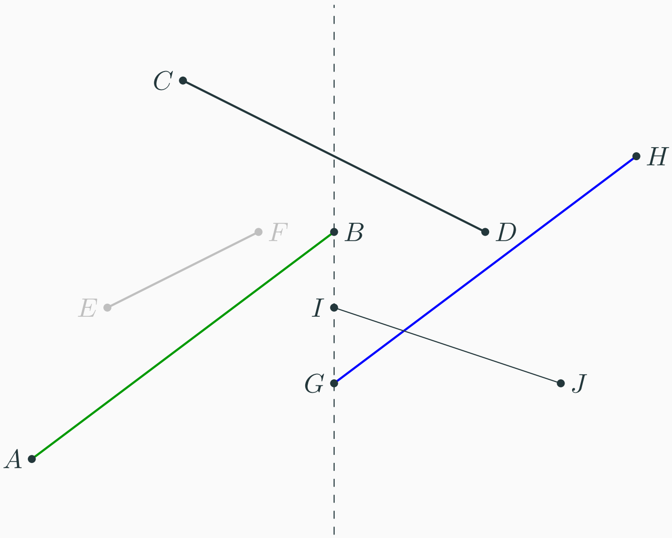
Visualização de identificação de interseção de segmentos



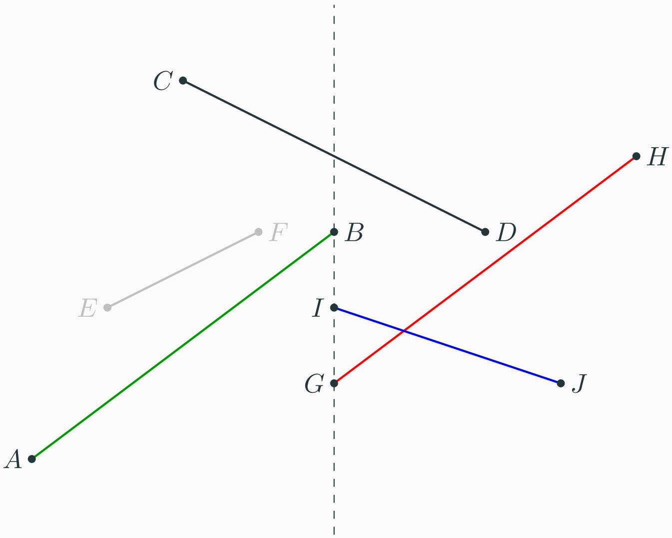
Visualização de identificação de interseção de segmentos



Visualização de identificação de interseção de segmentos



Visualização de identificação de interseção de segmentos



Implementação do algoritmo de Shamos-Hoey

```
#include <bits/stdc++.h>

using namespace std;
using ll = long long;

template<typename T>
bool equals(T a, T b)
{
    if (std::is_floating_point<T>::value)
    {
        const double EPS { 1e-6 };

        return fabs(a - b) < EPS;
    } else
        return a == b;
}

```

Implementação do algoritmo de Shamos-Hoey

```cpp
template<typename T>
struct Point
{
    T x, y;

    bool operator<(const Point& P) const
    {
        return x != P.x ? x < P.x : y < P.y;
    }

    bool operator==(const Point& P) const
    {
        return x == P.x and y == P.y;
    }
};

template<typename T>
T D(const Point<T>& P, const Point<T>& Q, const Point<T>& R)
{
    return (P.x*Q.y + P.y*R.x + Q.x*R.y) - (R.x*Q.y + R.y*P.x + Q.x*P.y);
}
```

Implementação do algoritmo de Shamos-Hoey

```cpp
template<typename T>
struct Segment
{
    T a, b, c;
    Point<T> A, B;

    Segment(const Point<T>& P, const Point<T>& Q)
        : a(P.y - Q.y), b (Q.x - P.x), c(P.x*Q.y - Q.x*P.y), A(P), B(Q)
    {
        sweep_x = -1;
    }

    bool operator<(const Segment& line) const
    {
        return (-a*sweep_x - c)*line.b < (-line.a*sweep_x -line.c)*b;
    }

    bool intersect(const Segment& s) const
    {
        auto d1 = D(A, B, s.A);
        auto d2 = D(A, B, s.B);
```

Implementação do algoritmo de Shamos-Hoey

```

        if ((equals(d1, 0LL) && contains(s.A)) ||
            (equals(d2, 0LL) && contains(s.B)))
             return true;

        auto d3 = D(s.A, s.B, A);
        auto d4 = D(s.A, s.B, B);

        if ((equals(d3, 0LL) && s.contains(A)) ||
            (equals(d4, 0LL) && s.contains(B)))
             return true;

        return (d1 * d2 < 0) && (d3 * d4 < 0);
    }

    bool contains(const Point<T>& P) const
    {
        if (P == A || P == B)
             return true;

```

Implementação do algoritmo de Shamos-Hoey

```cpp
        auto xmin = min(A.x, B.x);
        auto xmax = max(A.x, B.x);
        auto ymin = min(A.y, B.y);
        auto ymax = max(A.y, B.y);

        if (P.x < xmin || P.x > xmax || P.y < ymin || P.y > ymax)
            return false;

        return equals((P.y - A.y)*(B.x - A.x), (P.x - A.x)*(B.y - A.y));
    }

    static T sweep_x;
};

template<typename T>
T Segment<T>::sweep_x;

```

Implementação do algoritmo de Shamos-Hoey

```cpp
template<typename T>
bool shamos_hoey(const vector&lt;Segment&lt;T&gt;&gt;& segments)
{
    struct Event
    {
        Point<T> P;
        size_t i;

        bool operator<(const Event& e) const { return P < e.P; }
    };

    vector<Event> events;

    for (size_t i = 0; i < segments.size(); ++i)
    {
        events.push_back({ segments[i].A, i });
        events.push_back({ segments[i].B, i });
    }

    sort(events.begin(), events.end());
    set&lt;Segment&lt;T&gt;&gt; sl;
```

Implementação do algoritmo de Shamos-Hoey

```
    for (const auto& e : events)
    {
        auto s = segments[e.i];
        Segment<T>::sweep_x = e.P.x;

        if (e.P == s.A)
        {
            sl.insert(s);

            auto it = sl.find(s);

            if (it != sl.begin())
            {
                auto L = *prev(it);

                if (s.intersect(L)) return true;
            }

            if (next(it) != sl.end())
            {
                auto U = *next(it);
```

Implementação do algoritmo de Shamos-Hoey

```

                if (s.intersect(U)) return true;
            }
        } else
        {
            auto it = sl.find(s);

            if (it != sl.begin() and it != sl.end())
            {
                auto L = *prev(it);
                auto U = *next(it);

                if (L.intersect(U)) return true;
            }

            sl.erase(it);
        }
    }

    return false;
}
```

Referências

1. **De BERG**, Mark; **CHEONG**, Otfried. *Computational Geometry: Algorithms and Applications*, 2008.
2. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
3. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.
4. **SUNDAY**, Dan. Intersections of a Set of Segments, acesso em 25/05/2019.
5. Wikipedia. Sweep line algorithm, acesso em 22/05/2019.