Geometria Computacional

Wladimir Araújo Tavares [1]

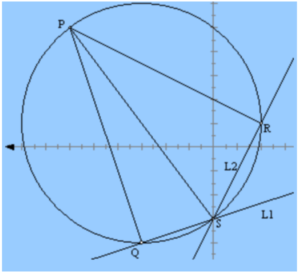
[1]Universidade Federal do Ceará - Campus de Quixadá

7 de junho de 2017

Problema: Círculo passando por três pontos

Algorithm 1 Algoritmo CirculoTresPontos

function CirculoTresPontos($P, Q, R, C, raio$)
    Encontre a reta PQ
    Encontre a reta PR
    Seja $L_1 \perp PQ$ e $L_2 \perp PR$
    Seja S a intersecção entre L_1 e L_2
    $C \leftarrow (P + S)/2.0$
    $raio \leftarrow dist(P, S)/2.0$



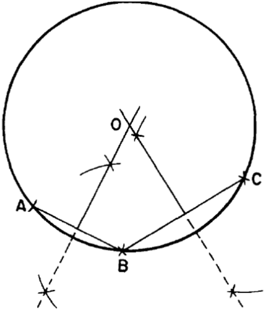
Círculo passando por três pontos

```
void circleByThreePoint(PointD P, PointD Q,
                        PointD R, PointD &C,
                        double &radius){
  Line PQ;
  Line PR;
  Line L1;
  Line L2;
  PointD S;
  pointsToLine(P,Q,PQ);
  pointsToLine(P,R,PR);
  pointAndSlopeLine(Q, 1.0/PQ.a, L1);
  pointAndSlopeLine(R, 1.0/PR.a, L2);
  areIntersect(L1,L2, S);
  C.x = midPoint(P,S);
  radius = dist(P,S)/2;
}
```

Problema: Círculo passando por três pontos

Algorithm 2 Algoritmo CirculoTresPontos

1: **function** CirculoTresPontos($A, B, C, O, raio$)
2: $midAB \leftarrow (A + B)/2.0$
3: $midBC \leftarrow (B + C)/2.0$
4: Encontre a reta $L_1 \perp AB$ passando por midPQ
5: Encontre a reta $L_2 \perp BC$ passando por midPR
6: $O \leftarrow L_1 \cap L_2$
7: $raio \leftarrow dist(O, B)$



Círculo passando por três pontos

```
void circleByThreePoint2(PointD P,
                         PointD Q,
                         PointD R,
                         PointD &C,
                         double &radius){

  Line PQ;
  Line PR;
  Line L1;
  Line L2;
  PointD S;

  pointsToLine(P,Q,PQ);
  pointsToLine(P,R,PR);
  PointD midPQ = midPoint(P,Q);
  PointD midPR = midPoint(P,R);
  pointAndSlopeLine(midPQ, 1.0/PQ.a, L1);
  pointAndSlopeLine(midPR, 1.0/PR.a, L2);
  areIntersect(L1,L2, C);
  radius = dist(C,P);
}
```

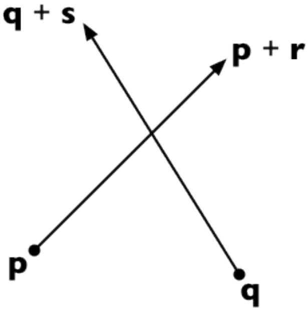
Smallest Circle Enclosing Problem

Algorithm 3 Algoritmo SmallestCircleEnclosingProblem

function SmallestCircleEnclosingProblem($P, centro, raio$)
    $centro \leftarrow P[0]$, $raio \leftarrow MaxValue$
    for $p, q \in P$ **do**
        $c \leftarrow (p+q)/2.0$, $r \leftarrow dist(p,q)/2.0$
        if $r < raio$ **then**
            if Se todos os pontos estão no círculo (c, r) **then**
                $raio \leftarrow r$, $centro \leftarrow c$
    for $p, q, r \in P$ **do**
        CirculoTresPontos(p,q,r,c,r)
        if $r < raio$ **then**
            if Se todos os pontos estão no círculo (c, r) **then**
                $raio \leftarrow r$, $centro \leftarrow c$

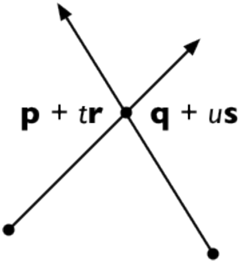
Intersecção de Segmentos

Suponha dois segmentos $p - p + r$ e $q - q + s$. Todos os pontos do primeiro segmento podem ser representados por $p + tr$, onde $t \in [0, 1]$ e qualquer ponto do segundo segmento pode ser representado por $q + us$, onde $u \in [0, 1]$



Interseção de Segmentos

As duas linhas se cruzam se podemos encontrar t e u tal que
$p + tr = q + us$



Interseção de Segmentos

- Realizando o produto vetorial por s em ambos os lados, temos
$$\begin{aligned} |(p+tr)\times s| &= |(q+us)\times s| \\ |p\times s| + |t(r\times s)| &= |q\times s| + |u(s\times s)| \end{aligned}$$
- Como $|s\times s| = 0$, temos
$$\begin{aligned} t|(r\times s)| &= |q\times s - p\times s| \\ t|(r\times s)| &= |(q-p)\times s| \\ t &= \frac{|(q-p)\times s|}{|r\times s|} \end{aligned}$$
- Realizando o produto vetorial por r em ambos os lados, temos
$$u = \frac{|(q-p)\times r|}{|r\times s|}$$
- Verifique se $t\in[0,1]$ e $u\in[0,1]$

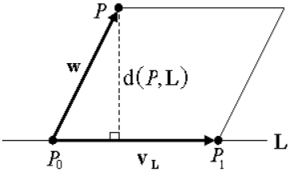
Intersecção de Segmentos

```
bool IntersectSegment(PointD p0, PointD p1,PointD p2, PointD p3, PointD &
  PointD r = p1-p0; //p2 = p1 + r
  PointD s = p3-p2; //p4 = p3 + s
  double denom = cross(r,s);
  if( fabs(denom) < EPSILON) return false;
  bool denomPositivo = denom > EPSILON;

  PointD v = p0-p2; //v  = p3-p1

  double s_number = cross(r,v);

  if( (s_number < 0) == denomPositivo)
    return false;

  double t_number = cross(s,v);

  if( (t_number < 0) == denomPositivo)
    return false;

  if( ((s_number > denom) == denomPositivo) ||
      ((t_number > denom) == denomPositivo) )
        return false;

  double t = t_number/denom;

  pt.x = p0.x + (t * r.x);
  pt.y = p0.y + (t * r.y);
}
```

Distância de um ponto para uma Reta

Algorithm 4 Algoritmo DistânciaPontoReta

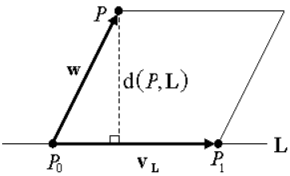
1: **function** DistânciaPontoReta($p, p0, p1$)
2: $\quad v \leftarrow p1 - p0$
3: $\quad w \leftarrow p - p0$
4: $\quad$ **return** $\frac{|v \times w|}{|v|}$



Distância de um ponto para uma Reta

Algorithm 5 Algoritmo PontoMaisProximoReta

```
function PontoMaisProximoReta(p, p0, p1)
    v ← p1 − p0
    w ← p − p0
    c1 ← v · w
    c2 ← v · v
    k ← c1 / c2
    return p1 + kv
```



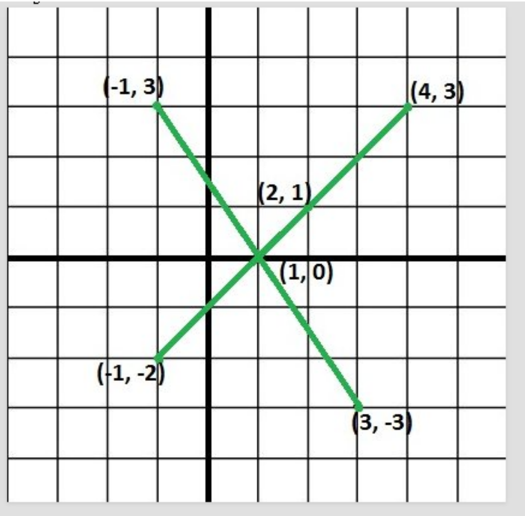
Distância Ponto Reta

```
double distPointToLine(PointD p, PointD p1, PointD p2){
   PointD v = p2-p1;
   PointD w = p-p1;
   return fabs(cross(v,w)/lenght(v));
}
PointD closestPointToLine(PointD p, PointD p1, PointD p2){
  PointD v = p2-p1;
  PointD w = p-p1;
  double c1 = dot(w,v);
  double c2 = dot(v,v);
  double k = c1 / c2;
  PointD Pc;
  Pc.x = p1.x + k*v.x;
  Pc.y = p1.y + k*v.y;
  return Pc;
}
```

Mínimo de Retas para cobrir todos os pontos

- Dado n pontos, queremos encontrar o número mínimo de retas que passa por um ponto específico (x_0, y_0) e cruza todos os pontos.
- Podemos resolver este problema, calculando o ângulo entre todos os pontos com (x_0, y_0).
- Se dois pontos distintos tem o mesmo ângulo então eles podem ser cobertos pela mesma linha.
- Problemas de precisão para o cálculo do ângulo podem ser evitados guardando dy e dx.

Mínimo de Retas para cobrir todos os pontos



Mínimo de Retas para cobrir todos os pontos

```
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
pair<int, int> getReducedForm(int dy, int dx)
{
    int g = gcd(abs(dy), abs(dx));
    bool sign = (dy > 0) == (dx > 0);
    if( sign )
      return make_pair( abs(dy)/g, abs(dx)/g);
    else
      return make_pair( -abs(dy)/g, abs(dx)/g);

}
```

Mínimo de Retas para cobrir todos os pontos

```
int minLinesToCoverPoints(vector <Point> P, int x0, int y0)
{
    //  set to store slope as a pair
    set< pair<int, int> > st;
    pair<int, int> temp;
    int minLines = 0;
    int N = P.size();

    //  loop over all points once
    for (int i = 0; i < N; i++)
    {
        //  get x and y co-ordinate of current point
        int curX = P[i].x;
        int curY = P[i].y;
        temp = getReducedForm(curY-y0, curX-x0);
        // if this slope is not there in set,
        // increase ans by 1 and insert in set
        if (st.find(temp) == st.end())
        {
            st.insert(temp);
            minLines++;
        }
    }
    return minLines;
}
```

Varredura de Graham

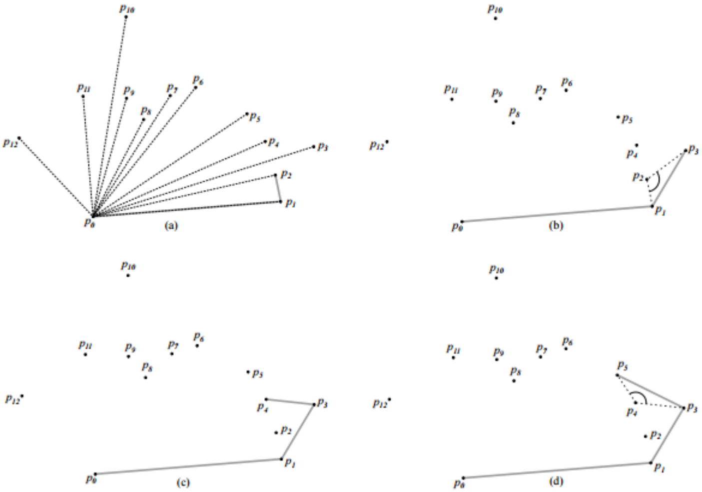
- O algoritmo de varredura de Grahan soluciona o problema da obtenção do fecho convexo mantendo uma pilha de candidatos.
- Cada ponto é testado para verificar se faz parte do fecho.
- Em caso negativo, o ponto é retirado da pilha.
- Ao término do algoritmo, a pilha contém exatamento os vértices que fazem parte do fecho convexo ordenado da direita para a esquerda.

Varredura de Graham

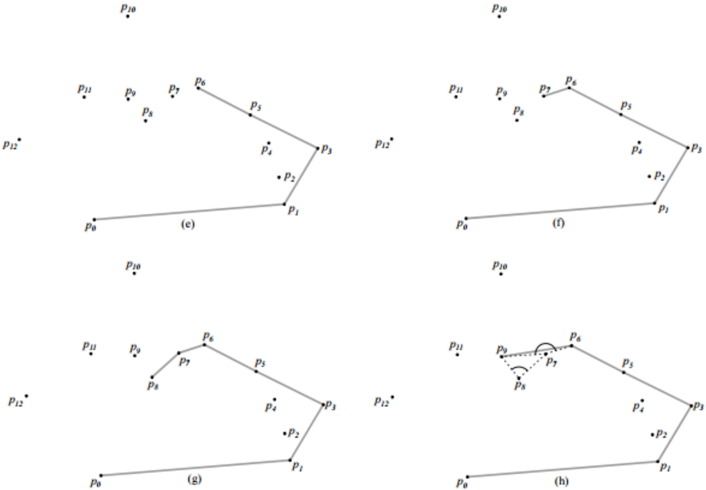
Algorithm 6 Algoritmo Varredura de Graham

```
function Varredura de Graham(P)
    P_0 ← ponto mais à esquerda de P.
    P ← ordenação do conjunto P com relação ao ângulo formado por P_0 P_i
    Empilhe P_0 e P_1 em hull
    topo ← 2
    while i < n do
        if DIRECAO(hull[topo − 1], hull[topo], P[i]) > 0 then
            hull[topo] ← P[i]
            topo ← topo + 1
            i ← i + 1
        else
            topo ← topo − 1
```

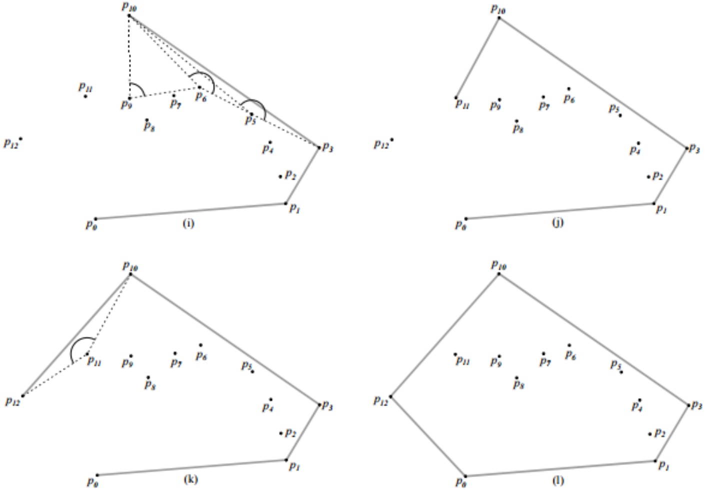
Execução da Varredura de Graham



Execução da Varredura de Graham



Execução da Varredura de Graham



Varredura de Graham

```
//p1 vem antes de p2
bool compare(const Point & p1, const Point & p2){
  int val = cross_product(p1.x,p1.y,p2.x,p2.y);
  if(val == 0){
    if( lenght(p1) <= lenght(p2) ) return true;
    else return false;
  }
  if( val > 0) return true;
  else return false;
}
void ordenacao_polar(vector <Point> & P){
  for(int i = 1; i < P.size(); i++){
    P[i] = P[i] - P[0];
  }
  sort(P.begin()+1, P.end(), compare);
  for(int i = 1; i < P.size(); i++){
    P[i] = P[i] + P[0];
  }
}
```

Fecho Convexo

```
vector<Point> convex_hull(vector <Point> & P){
  int n = (int)P.size();
  if( n <= 2 ){
    if( P[0] == P[1]  ) P.pop_back();
    return P;
  }
  //Encontre o ponto com o menor y em caso de empate o menor x
  int pivot = 0;
  for(int i = 1; i < n; i++){
    if( P[i].y < P[pivot].y || (P[i].y == P[pivot].y && P[i].x < P[pivot].
      pivot = i;
  }
  swap(P[0],P[pivot]);
  ordenacao_polar(P);
  vector <Point> S;
  S.push_back(P[0]);
  S.push_back(P[1]);
  int i = 2;
  while(i < n){
    int j = (int) S.size() - 1;
    int dir = DIRECTION(S[j-1],S[j],P[i]);
    if( dir >0 ) S.push_back(P[i++]);
    else S.pop_back();
  }
  return S;
}
```

Marcha de Jarvis

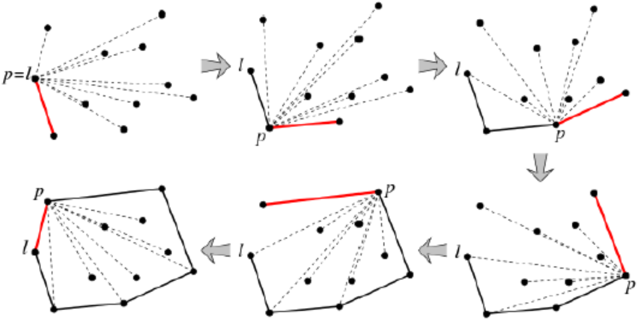
- A marcha de Jarvis é um algoritmo simples usado para resolver o problema da obtenção do fecho convexo.
- O algoritmo é análogo ao algoritmo de ordenação por seleção.
- A cada passo, escolhe-se um vértice para ser acrescentado ao fecho convexo.
- A escolha do próximo vértice é realizada $O(n)$.
- Se o fecho convexo tem h vértices, então o algoritmo tem complexidade $O(nh)$

Marcha de Jarvis

Algorithm 7 Algoritmo Marcha de Jarvis

```
function MarchaJarvis(P)
    PontoNoFecho ← o ponto mais à esquerda de P.
    i ← 0
    repeat
        S[i] ← PontoNoFecho
        next ← P[0]
        for j ← 1 até |P| do
            if (next == pontoNoFecho) ou (P[j] está à esquerda da linha de S[i] até próximo) then
                next ← P[j]
        i ← i + 1
        pontoNoFecho ← next
    until next == S[0]
    return S
```

Execução da Marcha de Jarvis



The execution of Jarvis's March.

Marcha de Jarvis

```
vector <Point> convexHull(vector <Point> P){
    vector<Point> hull;
    int n = P.size();
    if (n < 3) return hull;
    // Encontre o ponto mais à esquerda
    int l = 0;
    for (int i = 1; i < n; i++) if (P[i].x < P[l].x) l = i;
    //Este loop roda em tempo O(h), onde h é o numero de vértices
    // do fecho
    int p = l, q;
    do
    {
        hull.push_back(P[p]);
        //Procure o ponto q que está à esquerda para todos os pontos x
        //A ideia é manter o último visitado mais à esquerda em q
        q = (p+1)%n;
        for (int i = 0; i < n; i++)
           // Se i está mais à esquerda que o ponto atual
           if (orientation(P[p], P[i], P[q]) == 2) q = i;
        //Agora q está mais no sentido anti-horário com respeito a p.
        // q será adicionado na próxima iteração
         p = q;
    } while (p != l);  // Enquanto não voltar para o primeiro ponto
  return hull;
}
```

Par de Ponto mais próximo

- Dado um vetor de pontos em um plano, o problema é encontrar o par de pontos mais próximo.
- O problema surge em várias aplicações. Por exemplo, no controle de tráfego aéreo.
- O algoritmo ingênuo é $O(n^2)$.
- O problema pode ser resolvido por divisão e conquista em $O(nlogn)$.

Algoritmo de Divisão e Conquista

1. Ordene os pontos com relação as coordenadas X
2. Divida os pontos em duas metades
3. Recursivamente encontre a menor distância nas duas metades.
4. Seja d o mínimo entre as menores distâncias das duas metades.
5. Crie um vetor strip[] que armazena todos os pontos que estão na distância no máximo d das linhas que divide os dois conjuntos.
6. Encontre a menor distância entre os pontos de strip[]
7. Retorne o mínimo entre d e a menor distância calculada no passo anterior.

Algoritmo de Divisão e Conquista

- Mantendo os vetores ordenados com as coordenadas y, a menor distância em strip[] pode ser calculada em $O(n)$.
- O vetor strip será implicitamente ordenado para manter a complexidade O(n).

Par de ponto mais próximo

```
// The main functin that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    Point Px[n];
    Point Py[n];
    for (int i = 0; i < n; i++)
    {
        Px[i] = P[i];
        Py[i] = P[i];
    }

    qsort(Px, n, sizeof(Point), compareX);
    qsort(Py, n, sizeof(Point), compareY);

    // Use recursive function closestUtil() to find the smallest distance
    return closestUtil(Px, Py, n);
}
```

Par de ponto mais próximo

```
float closestUtil(Point Px[], Point Py[], int n){
    if (n <= 3)
        return bruteForce(Px, n);
    int mid = n/2; // Find the middle point
    Point midPoint = Px[mid];
    // Divide points in y sorted array around the vertical line.
    // Assumption: All x coordinates are distinct.
    Point Pyl[mid+1];   // y sorted points on left of vertical line
    Point Pyr[n-mid-1];  // y sorted points on right of vertical line
    int li = 0, ri = 0;  // indexes of left and right subarrays
    for (int i = 0; i < n; i++){
      if (Py[i].x <= midPoint.x) Pyl[li++] = Py[i];
      else Pyr[ri++] = Py[i];
    }
    float dl = closestUtil(Px, Pyl, mid);
    float dr = closestUtil(Px + mid, Pyr, n-mid);
    float d = min(dl, dr); // Find the smaller of two distances
    // Build an array strip[] that contains points close (closer than d)
    // to the line passing through the middle point
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(Py[i].x - midPoint.x) < d) strip[j] = Py[i], j++;
    // Find the closest points in strip.  Return the minimum of d and clos
    // distance is strip[]
    return min(d, stripClosest(strip, j, d) );
}
```

Par de ponto mais próximo

```
// A utility function to find the distance beween the closest points of
// strip of given size. All points in strip[] are sorted accordint to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d;  // Initialize the minimum distance as d

    // Pick all points one by one and try the next points till the differe
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}
```

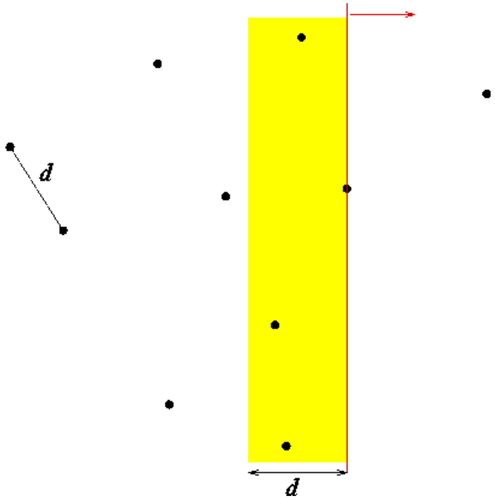
Algoritmo de Linha de Varredura

- Linha de varredura é uma estratégia para solução de problemas geométrico.
- A ideia geral é manter uma linha (com algumas informações auxiliares) que varre através do plano e resolve o problema localmente
- Definimos eventos que causam a mudança da nossa estrutura de dados auxiliar.

Par de pontos mais próximos

- Durante o algoritmo de Linha de Varredura manteremos as seguintes informações:
- O par mais próximo encontrado até o momento.
- A distância d entre pontos do par anterior.
- Todos os pontos em uma distância no máximo d à esquerda da linha de varredura. Esses pontos serão armazenados em um conjunto ordenado D pela coordenada y.

Algoritmo de Linha de Varredura



Algoritmo de Linha de Varredura

Cada vez que uma linha de varredura encontra um ponto p, as seguintes ações serão executadas:

1. Remova todos os pontos com a distância superior d à esquerda de p do conjunto ordenado D.
2. Determine o ponto à esquerda de p mais próximo dele.
3. Se esta distância for menor que d então atualize a distância mínima.

Algoritmo de Linha de Varredura

O resumo da análise do algoritmo de linha de varredura:

- Ordenar os pontos com relação a coordenada x leva $O(n \ log \ n)$.
- Inserir e remover um ponto do conjunto ordenado D leva $O(log \ n)$.
- Encontrar a distância mínima dentro de strip é $O(n)$.

Par de ponto mais próximo

```
double sweepLine(Point P[], int n){
  qsort(P, n, sizeof(Point), compareX);
  int back=0;
  double d=dist(P[0],P[1]);
  multiset<Point,classcomp> S;
  S.insert(P[0]);
  S.insert(P[1]);
  for (int i=2; i<n; i++){
    S.insert(P[i]);
    multiset<Point,classcomp>::iterator pos= S.find(P[i]), tmp;
    tmp=pos;
    if( tmp != S.end() ){
      tmp++;
      while (tmp!=S.end() && (tmp->y-pos->y)<d){
        d=min(dist(*tmp,*pos),d); tmp++;
      }
    }
    tmp=pos;
    if( tmp != S.begin() ){
      tmp--;
      while ( tmp != S.begin() && (pos->y-tmp->y)<d){
        d=min(d,dist(*tmp,*pos)); tmp--;
      }
    }
    for (;P[i].x-P[back].x>d && back<i; back++)
      S.erase(S.find(P[back]));
  }
  return d;
}
```

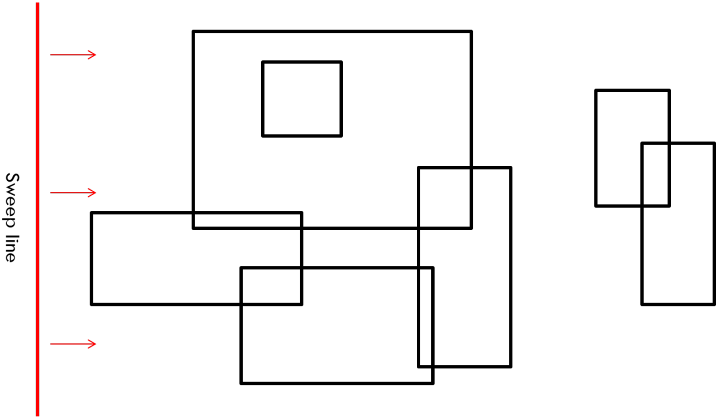
Par de ponto mais próximo

```
double sweepLine2(Point P[],int n){
  qsort(P, n, sizeof(Point), compareX);
  set< pair<int,int> > box;
  box.insert( make_pair(P[0].y, P[0].x) );
  box.insert( make_pair(P[1].y, P[1].x) );
  double best=dist(P[0],P[1]);
  int left = 0;
  for (int i=2;i<n;++i)
  {
    //Atualiza pontos ativos
    while (left<i && P[i].x - P[left].x > best){
      box.erase( box.find( make_pair( P[left].y , P[left].x ) ) );
      left++;
    }
    //Atualiza distancia
    set< pair<int,int> >::iterator it;
    it=box.lower_bound(make_pair(P[i].y-best,P[i].x-best));
    for(;it!=box.end() && P[i].y+best>=it->first;it++)
      best = min(best, dist(P[i], Point(it->second, it->first) ) );
    //Insere um ponto ativo
    box.insert( make_pair( P[i].y, P[i].x) );
  }
  return best;
}
```

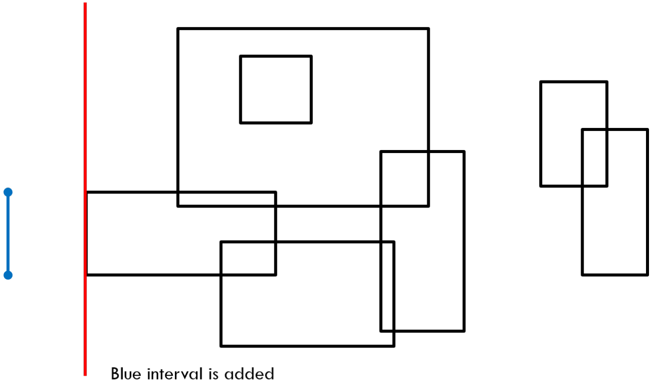
União de retângulos

- Dado n retângulos alinhados com o eixo x encontre a área da união deles.
- Nós vamos varrer o plano da esquerda para direita
- Eventos: Aresta esquerda e aresta direita dos retângulos.
- A ideia principal é manter um conjunto de retângulos ativos.

Exemplo

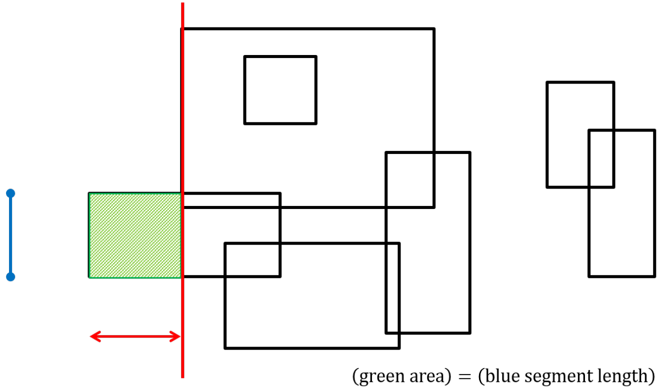


Example



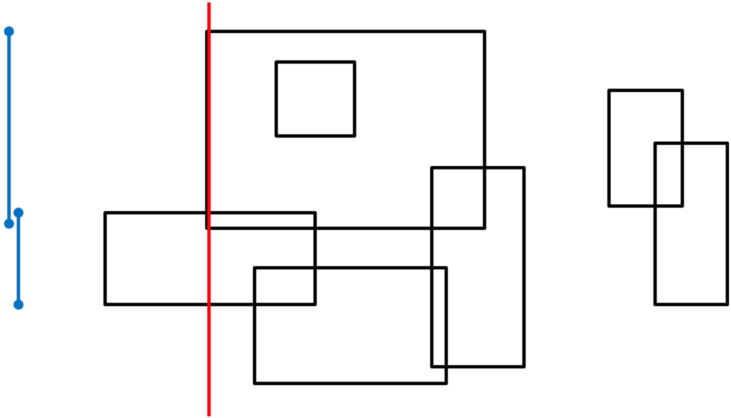
Blue interval is added
to the data structure

Example

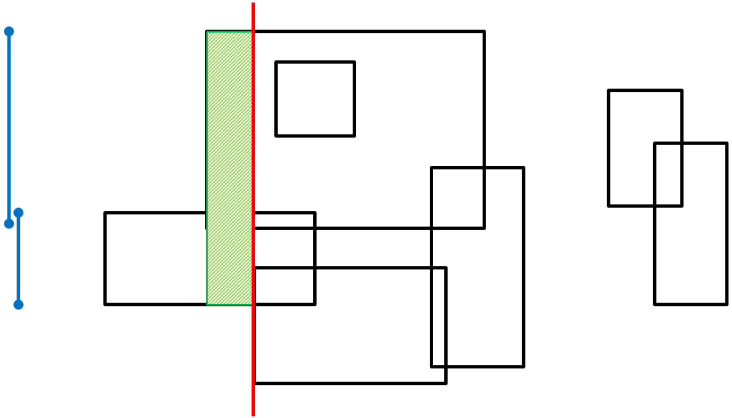


(green area) = (blue segment length)
× (distance that the sweep line traveled)

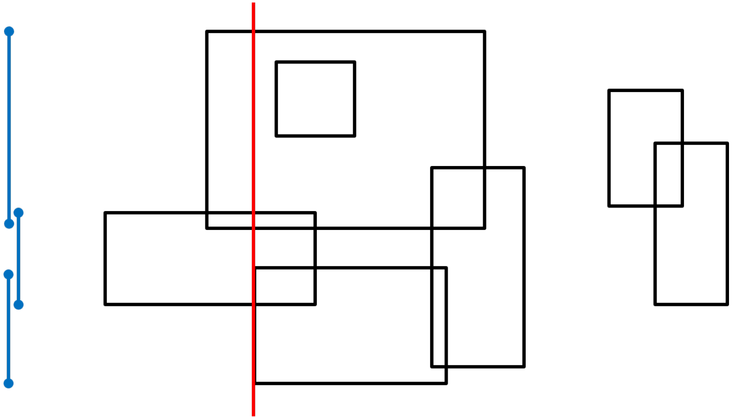
Example



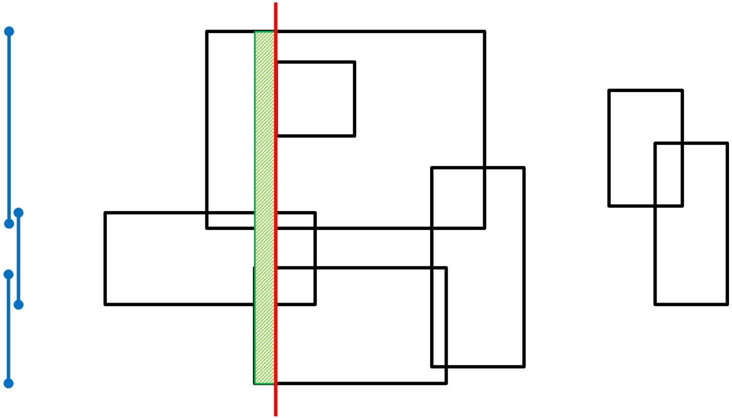
Example



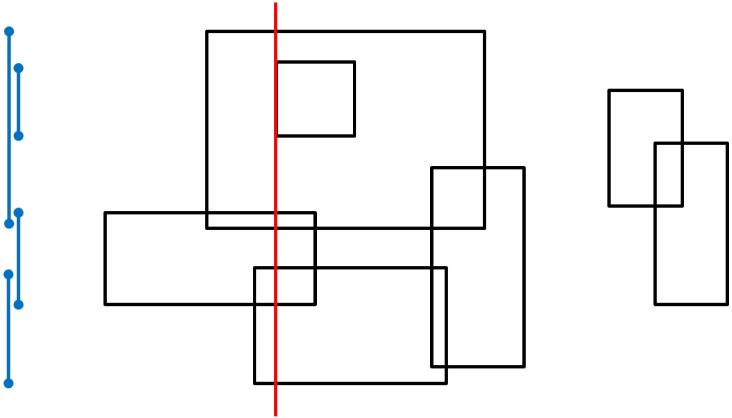
Example



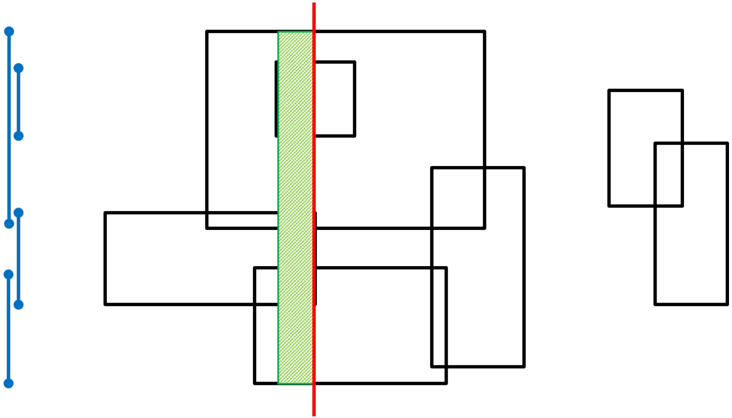
Example



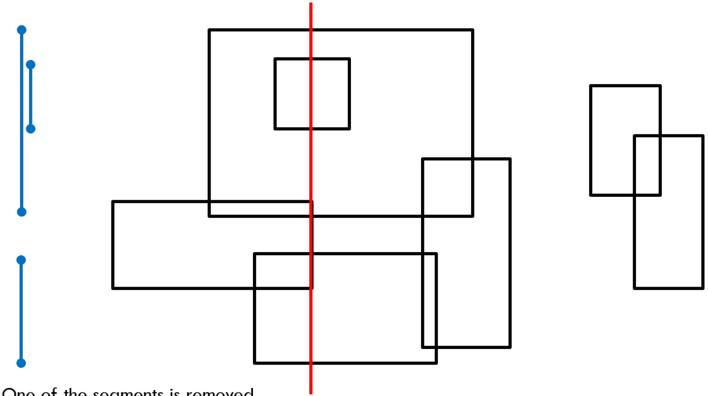
Example



Example

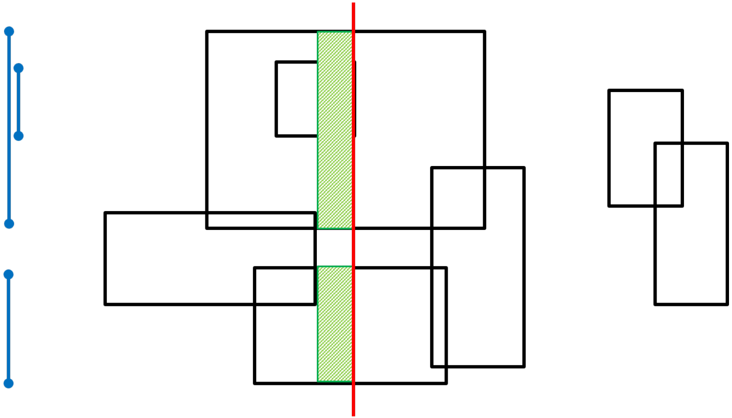


Example



One of the segments is removed

Example



Linha de Varredura

Outras aplicações:

- Fecho convexo
- Encontrar o perímetro da união de retângulos
- Encontrar todas as k intersecções entre n segmentos em $O((n+k)\ log\ n)$

Exercício

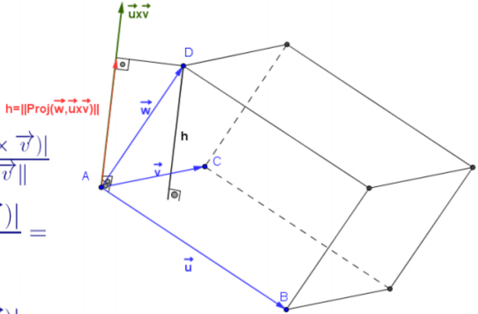
Calcule o volume do paralelepípedo a partir de três vetores $\vec{u}, \vec{u}$ e $\vec{w}$.

Volume do paralelepípedo é área da base vezes altura.

V = S. h , onde

S é a área da base e

h a altura.



$$h = \|Proj(w, \vec{u} \times \vec{v})\| = \frac{|\vec{w} \cdot (\vec{u} \times \vec{v})|}{\|\vec{u} \times \vec{v}\|}$$

$$V = \|\vec{u} \times \vec{v}\| \cdot \frac{|\vec{w} \cdot (\vec{u} \times \vec{v})|}{\|\vec{u} \times \vec{v}\|} =$$

O Volume é V= $|\vec{w} \cdot (\vec{u} \times \vec{v})|$

Operações

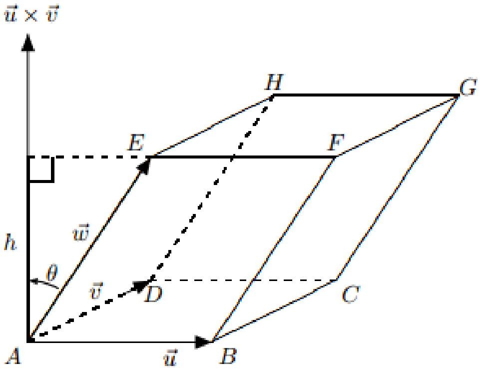
Seja $\overrightarrow{u} = (a_1, a_2, a_3)$ e $\overrightarrow{v} = (b_1, b_2, b_3)$,

$$\overrightarrow{u} \times \overrightarrow{v} = \begin{bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{bmatrix} \overrightarrow{i} + \begin{bmatrix} a_1 & a_3 \\ b_1 & b_3 \end{bmatrix} \overrightarrow{j} + \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix} \overrightarrow{k} \tag{1}$$

$$\overrightarrow{u} \times \overrightarrow{v} = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 \tag{2}$$

Exemplo

O volume do paralelepípedo ABCDEFGH, onde
$A = (1,2,0), B = (0,1,2), D = (1,1,3)$ e $E = (2,3,5)$ é 7.



Exercício

Calcule o volume do tetraedro a partir de três vetores $\vec{u}, \vec{u}$ e $\vec{w}$.

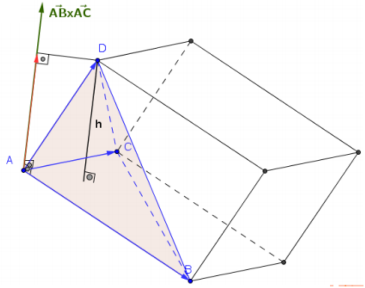
Volume de um tetraedro é $V = \frac{S \cdot h}{3}$

S área de base e h altura.

$$S = \frac{\left\|\overrightarrow{AB} \times \overrightarrow{AC}\right\|}{2} \quad \text{e h} = \frac{\left|(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot \overrightarrow{AD}\right|}{\left\|\overrightarrow{AB} \times \overrightarrow{AC}\right\|}$$

$$V = \frac{1}{3} \cdot \frac{\left\|\overrightarrow{AB} \times \overrightarrow{AC}\right\|}{2} \cdot \frac{\left|(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot \overrightarrow{AD}\right|}{\left\|\overrightarrow{AB} \times \overrightarrow{AC}\right\|}$$

$$V_{\text{tetraedro}} = \frac{\left|(\overrightarrow{AB} \times \overrightarrow{AC}) \cdot \overrightarrow{AD}\right|}{6}$$



Exercício

1. http://www.spoj.com/problems/QCJ4/
2. http://www.spoj.com/problems/BLMIRANA/
3. http://www.spoj.com/problems/CLOPPAIR/
4. http://www.spoj.com/problems/MPOLY/
5. http://www.spoj.com/problems/CROSSPDCT/
6. http://www.spoj.com/problems/INOROUT/
7. https://www.urionlinejudge.com.br/judge/pt/problems/view/1295
8. https://www.urionlinejudge.com.br/judge/pt/problems/view/1039
9. https://www.urionlinejudge.com.br/judge/pt/problems/view/2045