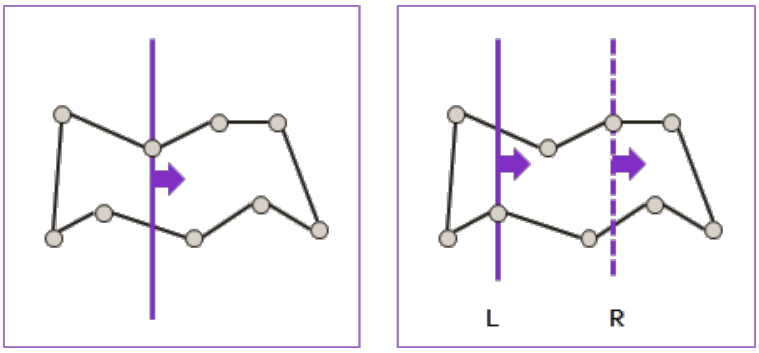
The Sweep the Line

Extent ★ Difficulty ★ ★

The Sweep the Line

The "scan lines" is the basis for the field of computational geometry techniques can be used to solve many computational geometry problems, Figure of BFS and DFS, like classic. It is not an object but a concept.

Translation of the scan line

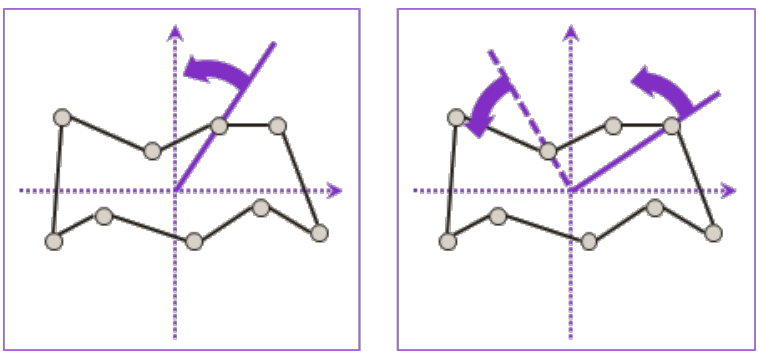


One (or two) infinitely long parallel lines along the vertical direction to move, move from the screen at one end to the other end, only stay in the vertices.

Implementation, usually press the coordinates of the order of all vertices, then the two index values, record the location of the parallel lines in which the vertex above. Two parallel lines, a main, across the entire screen; supplemented by one, followed by the main line of the status of pan. This is the truth of the yin and yang.

UVa 920 972

Rotation of the scanning lines



One (or two) rays emitted from the origin, do a 360 ° rotation, only in the vertices of stay.

Implementation, usually press the polar angle of the order of

all vertices, then the location of the two index values, record ray in which the vertex above. Two rays, one main, turned the whole picture; supplemented by one, followed by the main line of the status of pan. This is the truth of the yin and yang.

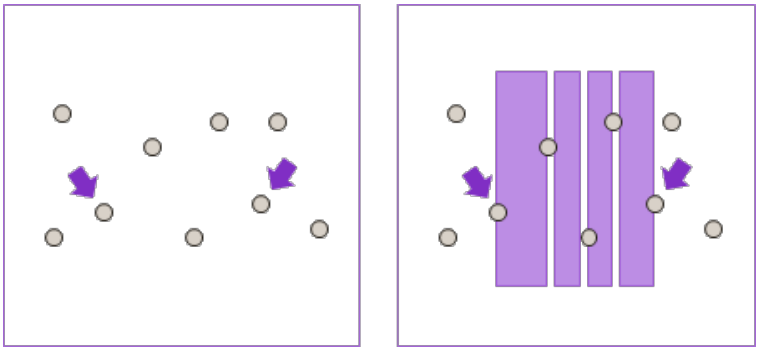
UVa 270 10927 11 529 11 696 11 704 10 691

ICPC 3259 4064

Steps when using the Sweep the Line design algorithm

To put it bluntly, the basic spirit of the scan line is this: first sort, and then search.

In two dimensions, an important feature is the "regional". For example, between two points will be closer the points are separated.



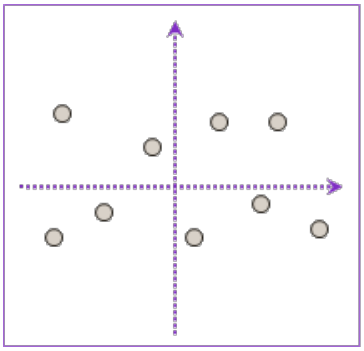
The purpose of sorting, which is to establish a "regional". With "regional", the search condition is even more precise, the speed of search faster.

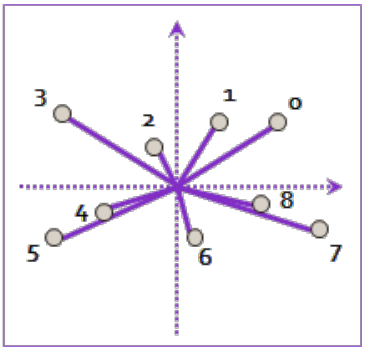
```
Observe whether the problem is the nature of the class do not ove
ap, "disjoint", and "interval" and "together".
And then selected the translation or rotation of the scan lines t
be scanned.
In other words, sort all the vertices to search.
```

The Sweep Line can solve the problem

```
Translation of the scan line
Closest the Pair Find the nearest point
Segment Intersections to identify all segments intersection
The Polygon Intersection to find the intersection of two polygons
union, difference
Voronoi Diagram
Delaunay Triangulation

Rotation of the scanning lines
The Convex Hull in to find out the convex hull
```

Rotation of the scan line: the polar angle sort (the Sorting Points b the Polar the Angle)






```
1.  typedef for  complex, < double >  the Point ;
2.  # Define the x real ()
3.  # Define y imag ()
4.
5.  a double  cross ( const  Point &  a ,  const  Point &
    )
6.  {
7.      return  a . x  *  y  *  b . x ;
8.  }
9.
10. / / Complex built-in function to calculate the size of
    the polar angle.
11. bool  cmp ( const  Point &  p1 ,  const  Point &  p2 )
12. {
13.     return  arg ( p1 ) <  arg ( p2 );
14. }
15.
16. / / Size of the polar angle arctan calculation.
17. / / Note that the size range (-180 °, +180 °].
18. bool  cmp ( const  Point &  p1 ,  const  Point &  p2 )
19. {
20.     return  atan2 ( p1 y ,  p1 . x ) <  atan2 ( p2 . y
    ,  p2, . x );
21. }
22.
23. / / First determine the quadrant, and then the outer
    product of the judgment order.
24. / / Polar angle of the same will be the length of the
    sort.
25. bool  cmp ( const  Point &  p1 ,  const  Point &  p2 )
26. {
27.     if  ( p1 y  ==  0  &&  p2, y  ==  0  &&  p1 . x  *
    p2, . x  <=  0 )  return  p1 . x  >  p2, . x ;
28.     if  ( p1 y  ==  0  &&  p1 is x  > =  0  &&  p2, .
     =  0 )  return  true ;
29.     if  ( p2 y  ==  0  &&  p2, x  > =  0  &&  p1 . y-
     =  0 )  return  false ;
```

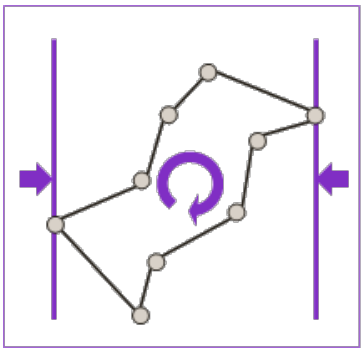
```
    if ( p1.y * p2.y < 0 ) return p1.y > p
y;
    a double c = cross(p1, p2);
    return c > 0 || c == 0 && fabs(p1.x) >
fabs(p2.x);
}

// First determine the quadrant, and then the outer
product of the judgment order.
// Polar angle for each point are different.
bool cmp(const Point &p1, const Point &p2)
{
    if (p1.y > 0 && p2.y > 0)
        return p2.x * p1.is.y < p2.y * p1.x
    else if (p1.y < 0 && p2.y < 0)
        return p2.x * p1.is.y < p2.y * p1.x
    else if (p1.y == 0)
        if (p1.x > 0)
            return true;
        else
            return p2.y < 0;
    else if (p2.y == 0)
        if (p2.x > 0)
            return false;
        else
            the return p1.y > 0;
    else
        the return p1.y > 0;
}

void sort_points_by_polar_angle()
{
    Point p[100];
    sort(p, p + N, cmp);
}
```

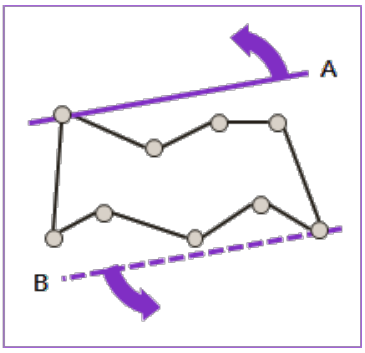
Rotating the Caliper

Extent ★ Difficulty ★ ★

Rotating the Caliper

"Rotating caliper" is also a calculation of geometric areas of basic practices, it is not an object but a concept. Reference information: http://cgm.cs.mcgill.ca/orm/rotcal.html The Chinese translation: http://blog.csdn.net/ACMaker/archive/2008/10.aspx .





The graphics plane 360 ° rotation, and two vertical lines at any time from about clamping the graphics, the measured width, find the extreme vertex.

Switch point of view, it becomes: two infinitely long parallel lines, doing a 360 ° rotation, try to grip the graphics plane.

Implementation, it is usually based on the two index values, record the location of the parallel lines in which the vertex above. Two parallel lines, the main one, turned the whole picture; a supplement, elastic follow the status of the main line. This is the truth of the yin and yang.

Implementation, just turn it 180 °. Half a turn, the two parallel lines reversed, the same effect as 360 °.

To put it bluntly, is the basic spirit of the rotating caliper: brute-force slope, the more inclined to judge the target object.

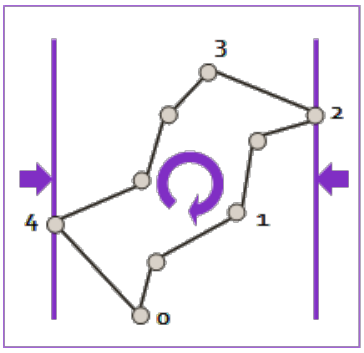
UVa 303 10173 10416 11 243

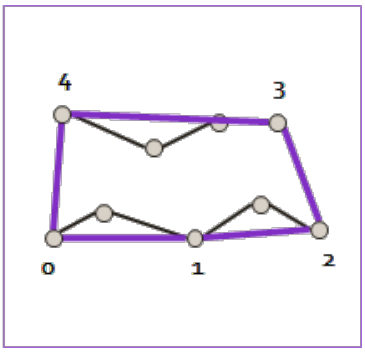
Problem can be solved by rotating the Caliper

```
Farthest the Pair find out all the farthest point on the
The Convex the Polygon convex polygon problems, such as seeking (
meter, merge, intersect, etc.
Bounding Box
```

In fact the caliper of rotation and translation of the scan line each other's points and lines on even the pan scan lines can be resolved, rotating caliper can be resolved, and vice versa.

Rotating caliper: the convex hull of The Convex Hull in





Using the rotating calipers grip graphics the caliper folder not into just the convex hull of the graph. Therefore, the rotating caliper suitable for the convex hull, convex polygon.

Rotation the caliper folder to the vertex order is the order of the convex hull of the vertices.

```
1.  struct  Point  { dobule  x ,  y ;}  P [ 10 ];
2.  Point  L [ 10 + 1 ],  U [ 10 ];    / / convex hull,
    convex hull
3.
4.  bool  cmp ( const  Point &  a ,  const  Point &  b )
5.  {
6.      return  a . x  <  b . x  | |  x  ==  b . x  &&  a
     <  b, y ;
7.  }
8.
9.  void  rotating_cliper ()
10. {
11.     / * Find the convex hull of Andrew's Monotone Chai
    * /
12.
13.     the sort ( P ,  P  +  10 ,  cmp );    / / first ro
    of the X coordinate, and then row Y coordinates
14.
15.     int  l  =  0 ,  u  =  0 ;        / / convex hull o
    points under the convex hull of points
16.     for  ( int  i = 0 ;  i < 10 ; + + i )
17.     {
18.         while  ( l  > =  2  &&  cross ( L [ l - 2 ],
    [ l - 1 ],  P [ i ]) <=  0 )  -;
19.         while  ( u  > =  2  &&  cross ( U [ u - 2 ],
    [ u - 1 ],  P [ i ])> =  0 )  u -;
20.         L [ l + +] =  P [ i ];
21.         U [ u + +] =  P [ i ];
22.     }
23.
24.     / * Rotate caliper * /
25.
26.     / / Convex hull of the extra added convex hull
```

```
    vertex in order to operate.
27.     if  ( u - 2  > =  0 )  L [ l ] =  U [ u - 2 ];
28.
29.     for  ( int  i = 0 ,  j = u - 1 ;  i < l  &&  j > 0
    ;)
30.     {
31.         cout  <<  "folder to the vertex L [i] and the
    vertex U [j] ;
32.
33.         / / The bottom edge and top edge of the openin
    angle:
34.         / / Less than 180 °, then the top of the
    advance. Thought of as the bottom is very flat, above
    the variable convex.
35.         / / Greater than 180 °, the bottom forward.
    Thought of as the top of a very flat, below changed
    convex.
36.         / / Equal to 180 °, while at the same time
    forward, can also choose a forward.
37.         if  ( cross ( L [ i + 1 ] -  L [ i ],  U [ j -
    ] -  U [ j ]) <  0 )
38.             i + +;     / / below the forward
39.         else
40.             j -;     / / top of the forward
41.     }
42. }
```

The loop section may do one of the main one in the form of: a vertex of each exhaustive immediately to find the point on the top.

```
1.     for  ( int  i = 0 ,  j = u - 1 ;  i < l  &&  j > 0
    ;  i + +)    / / i + + to move into the loop
2.     {
3.         cout  <<  "folder to the vertex L [i] and the
    vertex U [j] ;
4.         if  ( cross ( L [ i + 1 ] -  L [ i ],  U [ j -
    ] -  U [ j ])>  0 )  j -;
5.     }
```

Closest the Pair

Extent ★ difficulty ★

Closest the Pair

Group of points which, from the nearest two points called the "nearest point".

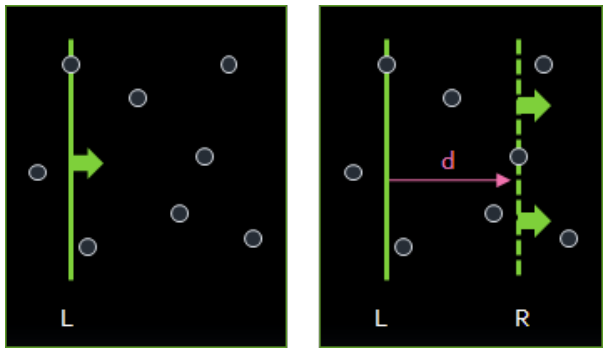
The brute-force method of time complexity is O (N ^ 2), you

can find out all the latest point.

```
1.  the struct  Point  { double  x ,  y ;}  p [ 10 ];
2.
3.  a double  closest_pair ()
4.  {
5.      double  d  =  1e9 ;  / / closest point on the
    distance
6.      Segment  cp. [ 10 ];  / / closest point on
7.      int  n  =  0 ;       / / closest point on the numb
    of
8.
9.      for  ( int  i = 0 ;  i < N ; + + i )          / /
    brute-force a
10.         for  ( int  j = i + 1 ;  j < N ; + + j )    /
    brute-force point b
11.         {
12.             double  dij  =  distance ( p [ i ],  p [ j
    ]);
13.             if  ( dij  <  d )
14.                 d  =  dij ,  n  =  0 ,  cp [ n + +] =
    Segment ( i ,  j );
15.             else  ( dij  ==  d )
16.                 cp [ n + +] =  Segment ( i ,  j );
17.         }
18.
19.     the return  d ;
20. }
```

Below I describe a brute-force method, the time complexity is still O (N ^ 2), but exhaustive than all points on the way much faster than the back to talk about the divide and conquer method is much faster.

```
First sort all the points, mainly the X axis, Y coordinates, it d
s not matter.
(2) from left to right sweep sequence exhaustive as the left endp
nt.
  A from the left point to right sweep, enumerating all the right
ndpoint.
  B, about two distance far worse, you can stop brute-force the r
ht end point.
```



Implementation, reduce the number of calls of the sqrt function, try to record the square of the distance, you can increase execution speed. Avoid direct sequencing of the original data, replication of an index or index values to sort, but also can improve the execution speed.

```
1.  the struct  Point  { double  x ,  y ;}  p [ 10 ];
2.  bool  cmp ( const  Point &  i ,  const  Point &  j ) {
    return  i . x  <  j . x ;}
3.
4.  a double  closest_pair ()
5.  {
6.      the sort ( p ,  p + 10 ,  cmp );  / / in accordanc
    with the X-coordinate sort
7.
8.      double  d  =  1e9 ;       / / closest point on the
    distance
9.      Segment  cp. [ 10 ];       / / closest point on
10.     int  n  =  0 ;            / / closest point on the
    number of
11.
12.     for  ( int  i = 0 ;  i < N ; + + i )
13.         for  ( int  j = i + 1 ;  j < N ; + + j )
14.         {
15.             if  ( p [ i ] x  +  d  <  p [ j ]. x )
    break ;
16.             double  dij  =  distance ( p [ i ],  p [ j
    ]);
17.             if  ( dij  <  d )
18.                 d  =  dij ,  n  =  0 ,  cp [ n + +] =
    Segment ( i ,  j );
19.             else  ( dij  ==  d )
20.                 cp [ n + +] =  Segment ( i ,  j );
21.         }
22.
23.     the return  d ;
24. }
```

UVa 10245 10750 11 378

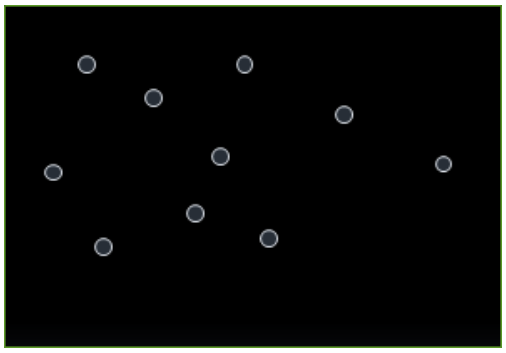
Closest the Pair

Degree of the ★ difficulty ★ ★ ★

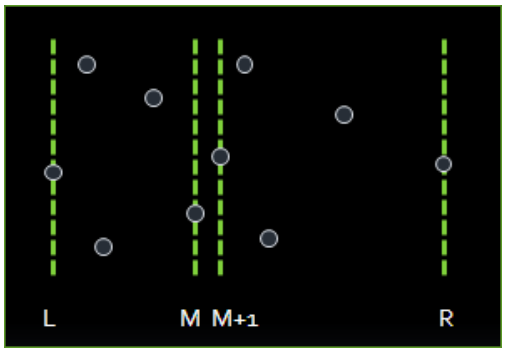
The second closest Pair

Divide and Conquer, the time complexity is O (N * (log N) ^ 2), you can find all the latest point. The principle is that the plane is cut into right and left sides, the answer is divided into "in the left point on the right point across both sides of the three situations. After the first left and right, respectively, recursive solving, and then use the left and right answer, quickly calculated across both sides of the answer.

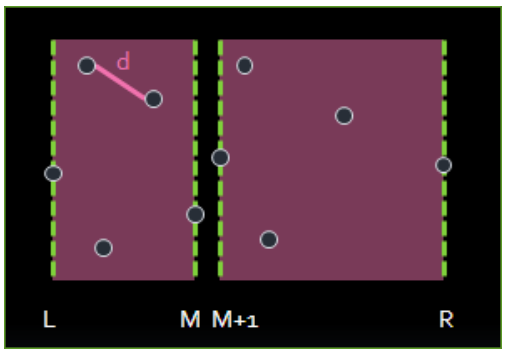
```
Preprocessing:
(1) Sort all, the X coordinate, Y coordinates does not matter. O
logN)
(2) Check whether there is total points. If yes, identify all the
ommon point end of the algorithm. O (N)
   (Just want to find out one group of points, you can omit this
ep.)

Divide:
All points is divided into left and right sides, left, right, as
ny points as possible.

Conquer:
Left, right, respectively, the recursive solution.

Merge:
(1) the optimal solution d of the left, right, calculated across
th sides of the solution:
  A first identify the points near the midline, horizontal distar
 less than d contains d. O (N)
     (Less than d does not contain d, only to find out one group
 the nearest point.)
  B, sort these points, Y coordinates of the main X-coordinates d
s not matter. O (NlogN)
  C, every point to find the neighboring points, take a look at i
not the solution.
     Nearby, not total points, and on the high side, the high-po
, only a maximum of four points. O (N * 4) = O (N)
     (If only looking for one group of points, at most only two
nts of O (N * 2) = O (N))
The answer to the left, right side, across both sides of the opti
l solution among the three.
```



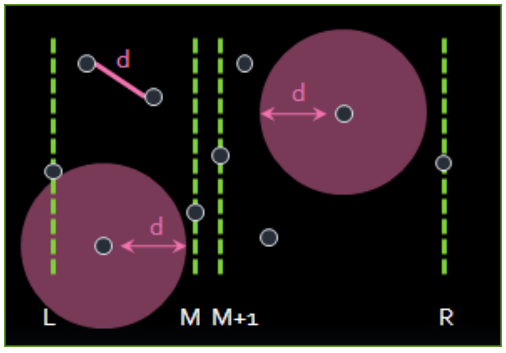
Illustrations are provided below. On the screen at some point.



The point is divided into left and right, as many points as possible. The junction of the left and right may converge or may not converge. Right and left sides usually is not as wide.

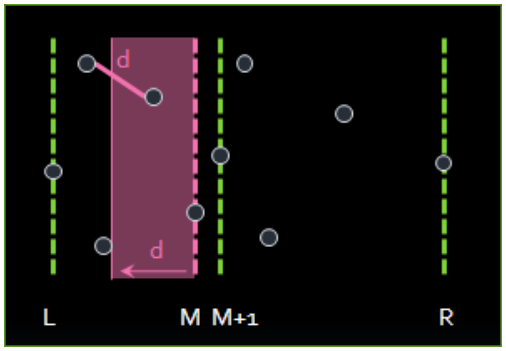


Left, right, respectively, recursive solving the last obtained in both cases, the nearest point. Recently on the point of the distance d.



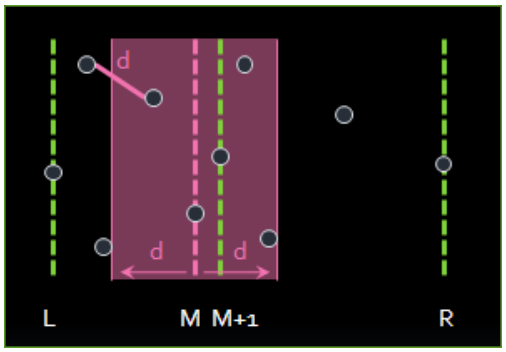
Can be found in the left side of each point, within the range of radius d (does not contain a boundary), there will be no other point on the left side of existence, can only be the other side of the

point. The right side of the same applies.

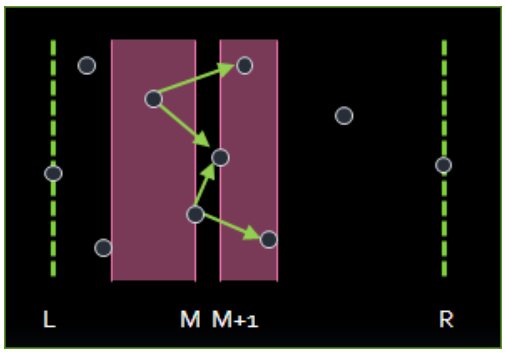


To find the right, across both sides of the point from the left side of the right line, to the left within the scope of the distance d (including the border), may be related to the right of the point formed near point (near point).

To the right side of the main.

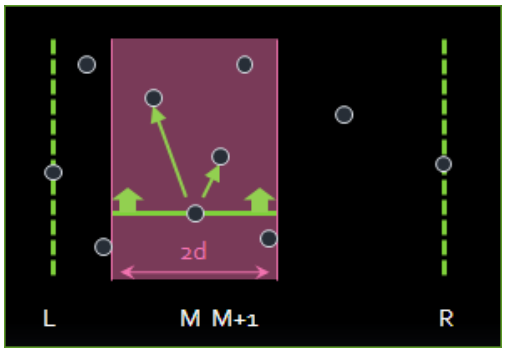


From the left side of the right line, the right distance d within the range of these points, it is possible the other end of the scope of the endpoint.

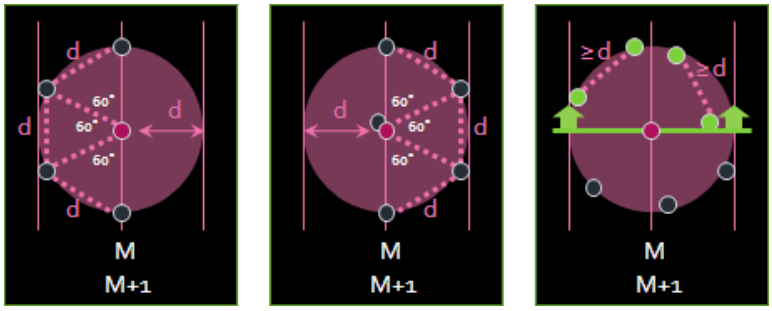


To find out across both sides of the nearest point of right, as long as the order of exhaustive left right border within the distance d at the left point, as the left endpoint; and then find the left side of the right boundary within the distance d, in the right point, do the right endpoint. Too far away from the point is not necessary to find.

Although this method is intuitive, but it is not easy to implement.

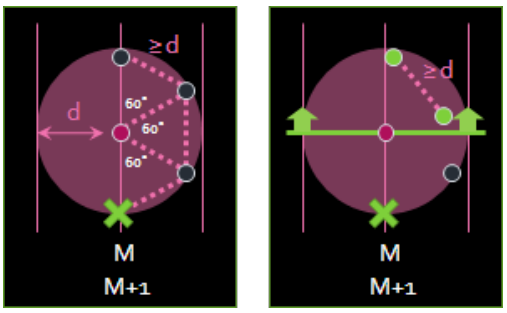


A relatively easy to implement, within the scope of these points all mixed together, regardless of left and right, and then scanned from bottom to top.



For a point, just need to find the distance less than or equal to d endpoint on the line, which is within the scope of the radius d (including border).

Bad luck, the mutual convergence of the left side, right side of the border, but also happens to be a point on the boundary, then the distance is less than d is equal to the endpoint, there will be a maximum of nine points: four points is the same side , the semicircle weeks in the ipsilateral angle from 60 ° position; five points of the opposite side, four points in the opposite side of the semicircle weeks on the angle from 60 ° position, and another point overlap with the origin. This is forced upon the results from the rougher edges of madness. When we scanned from bottom to top, just check the subsequent five-point; when we began to have disposed of a total point of the problem, just check the next four points.



If you want to find out one group of the nearest point on, you just need to find a distance less than d endpoint on the line, do

not have to find the distance equal to the endpoints of d. Distance less than d endpoint, only a maximum of three points: three points are the opposite side, free to move within the half circle of the opposite side. From the rougher edges of madness thrust upon not fit into the four-point, there must be a little touched the circumference. When we scanned from bottom to top, just check the subsequent two.

You're done.

Implementation, when the problem is split into two points, three points left, you can directly solve the stop recursion down, so you can increase execution speed.

```
Only to find out the nearest point on the distance
1. the struct Point { double x, y;} p[10], t[10];
2. bool cmpx(const Point &i, const Point &j) {return i.x<j.x;}
3. the bool cmpy(const Point &i, const Point &j) {return i.y<j.y;}
4.
5. a double DnC(int L, an int R)
6. {
7.     if(L>=R) return 1e9;  / / no point, on one point.
8.
9.     / * Divide: all divided into right and left sides, as many points as possible. * /
10.
11.    int M=(L+R)/2;
12.
13.    / * Conquer: left, right, respectively, the recursive solution. * /
14.
15.    double d=min(DnC(L,M), DNC(M+1,));
16. / / If (d == 0.0) return d; / / end early
17.
18.    / * Merge: Find the point near the midline, and so according to the Y coordinates. Of O (NlogN). * /
19.
20.    an int N=0;   / / number of points near the center line of
21.    for(int i=M;    i>=L, &&  p[M].x-[i].x<d;-i)  t[N++]=p[i];
22.    for(int i=M+1; i<=R && p[i].x-[M].x<d;++i)  t[N++]=p[i];
23.    sort(t, t+N, cmpy);  / / Quicksort O (NlogN)
```

```
24.
25.     / * Merge: finding across both sides of the point.
    (N). * /
26.
27.     for  ( int  i = 0 ;  i < N - 1 ; + + i )
28.         for  ( int  j = 1 ;  j <= 2  &&  i + j < N ; +
    j )
29.             d  =  min ( d ,  distance ( t [ i ],  t [
    + j ]));
30.
31.     the return  d ;
32. }
33.
34. a double  closest_pair ()
35. {
36.     sort ( p ,  p + 10 cmpx );
37.     the return  DNC ( 0 ,  N - 1 );
38. }
```

If the Merge stage, all points in the Merge the Sort to re-sort of in accordance with the Y coordinates, the time complexity can be reduced to O (NlogN). However, the additional burden of the implementation of the Merge the Sort is too great, and usually much slower than the original.

Only to find out the nearest point on the distance

```
1.  the struct  Point  { double  x ,  y ;}  p [ 10 ],  t [
    10 ];
2.  bool  cmpx ( const  Point &  i ,  const  Point &  j )
    return  x  <  j x ;}
3.  the bool  cmpy ( const  Point &  i ,  const  Point &
    ) { return  i . y  <  j y ;}
4.
5.  a double  DnC ( int  L ,  an int  R )
6.  {
7.      if  ( L  > =  R )  return  1e9 ;  / / no point, on
    one point.
8.
9.      / * Divide: all divided into right and left sides,
    as many points as possible. * /
10.
11.     int  M  = ( L  +  R ) /  2 ;
12.
13.     / / First X-Block of the center line mark up, will
    run away because the wait will reorder.
14.     double  x  =  p [ M ] x ;
15.
16.     / * Conquer: left, right, respectively, the
    recursive solution. * /
17.
```

```
    / / Recursive solution, and reordered according to the Y coordinate.
    double  d  =  min ( DnC ( L , M ),  DNC ( M + 1 , ));
/ / If (d == 0.0) return d; / / end early

    / * Merge: Find the point near the midline, and so according to the Y coordinates. O (N). * /

    / / Find point near the midline, go first to the left side. Points according to Y coordinates sort.
    an int  N  =  0 ;   / / number of points near the center line of
    for  ( int  i = 0 ;  i <= M ; + + i )
        if  ( x  -  p [ i ] x  <  d )
            t [ N + +] =  p [ i ];

    / / Find the point near the center line, find the right side. Points according to Y coordinates sort.
    an int  P  =  N ;   / / P position of the separate
    for  ( int  i = M + 1 ;  i <= R ; + + i )
        if  ( p [ i ]. x  -  x  <  d )
            t [ N + +] =  p [ i ];

    / / Y coordinate sort. Using the Merge Sort, merge the sorted array.
    inplace_merge ( t ,  t + P t + N ,  cmpy );

    / * Merge: finding across both sides of the point. (N). * /

    for  ( int  i = 0 ;  i < N ; + + i )
        for  ( int  j = 1 ;  j <= 2  &&  i + j < N ; + j )
            d  =  min ( d ,  distance ( t [ i ],  t [ + j ]));

    / * The Merge: Y coordinates re-sort all points. O (N). * /

    / / As a result, a larger sub-problems can directl use the Merge the Sort.
    inplace_merge ( p + L, ,  p + M + 1 ,  p + R + 1  cmpy );

    the return  d ;
}

a double  closest_pair ()
{
```

```
    sort ( p ,  p + 10 cmpx );
    the return  DNC ( 0 ,  N - 1 );
}
```

If you start all the copy of the first sort Y coordinates, then handed back to the way do not have every sort. So make the time complexity of the explosion is O (N ^ 2), even slower than the brute-force method.

Only to find out the nearest point on the distance

```
struct  Point  { double  x ,  y ;}  px [ 10 ],  py [ 1
], t [ 10 ];
bool  cmpx ( const  Point &  i ,  const  Point &  j )
return  x  <  j x ;}
the bool  cmpy ( const  Point &  i ,  const  Point &
) { return  i . y  <  j y ;}

a double  DnC ( int  L ,  an int  R )
{
    if  ( L  > =  R )  return  1e9 ;

    int  M  = ( L  +  R ) /  2 ;
    double  d  =  min ( DnC ( L , M ),  DNC ( M + 1 ,
));
/ / If (d == 0.0) return d; / / end early

    / * Merge: in accordance with the Y coordinate ord
to find a point near the midline. O (N). * /

    an int  N  =  0 ;   / / number of points near the
center line of
    for  ( int  i = 0 ;  i < 10 ; + + i )     / / all
points are looking again
        if  ( py [ i ] x  -  px [ M ] x  <  d )
            t [ N + +] =  py [ i ];

    for  ( int  i = 0 ;  i < N - 1 ; + + i )
        for  ( int  j = 1 ;  j <= 2  &&  i + j < N ; +
j )
            d  =  min ( d ,  distance ( t [ i ],  t [
+ j ]));

    the return  d ;
}

a double  closest_pair ()
{
    the sort ( px ,  px + 10 cmpx );
    the copy ( px ,  px + 10 ,  py );     / / first co
of
```

```
31.     the sort ( py ,  py + 10 ,  cmpy );   / / Y
    coordinates and then follow the sort
32.     the return  DNC ( 0 ,  N - 1 );
33. }
```

Farthest the Pair

Degree of the ★ difficulty ★ ★ ★

Farthest the Pair

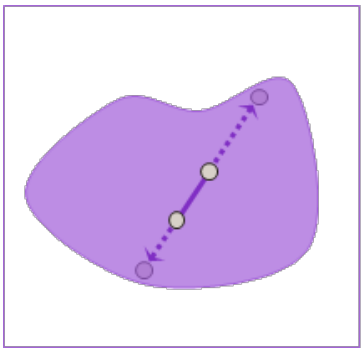
The group of points which, from the farthest two points called the "farthest point".

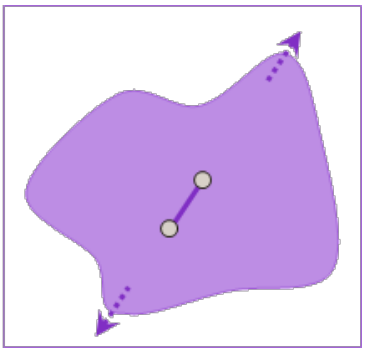
The brute-force method of time complexity is O (N ^ 2), you can find out all the farthest point.

Using the rotating calipers, the time complexity of O (NlogN).

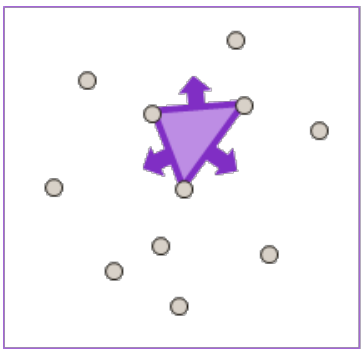
Idea

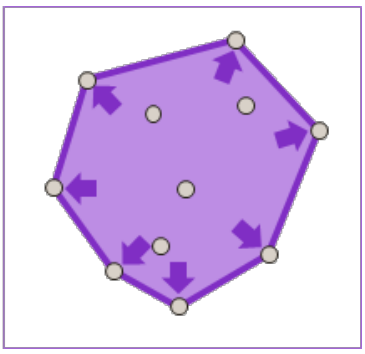
Distance away, is the expansion. Expansion of two connections, or expansion of the boundary, the distance is farther away.





Find the farthest point, you can use the expansion of the concept of the boundary. Expansion of the border until the limit, the boundary would touch to the most remote point, and finally the formation of the convex hull.





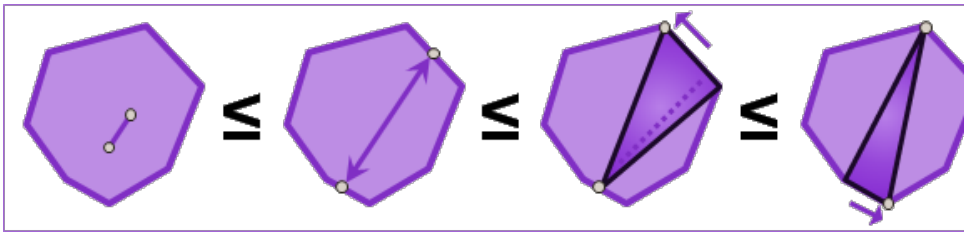
Therefore, the farthest point on the diagonal must be the

convex hull. Just like everyday life, the width of the rougher edges of the widest, the easiest card to.

Perhaps this argument is too abstract, not strict enough. To scrutiny some

Convex polygon within the farthest distance is the diagonal distance.

In other words, the convex polygon within any linear segment must be shorter than the length of a one diagonal. Here to switch to the expansion of the concept of two connection instructions.



The range of a convex polygon, take any two points.

Second, to extend the two-point connection, until the boundary.

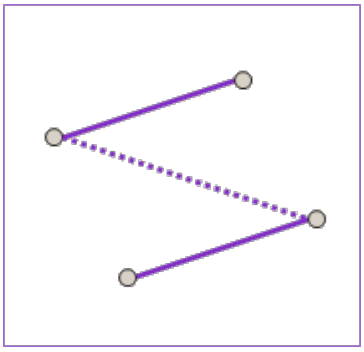
Third, move point to the vertex, as far as possible from the pedal.

Four, move to another point at the peak, ibid.

With this nature, to rotate the caliper, brute-force the slope of the longest diagonal length of the longest diagonal of the measurement, you can easily find the furthest point.

The longest diagonal of the convex polygon have slopes are different.

With a slope of more than two longest diagonal, it will produce a contradiction——delineate the two longest diagonal parallelogram can be found that the longer the diagonal of a parallelogram. Will be able to successfully delineate the convex polygon within the parallelogram, please refer to the definition of convex.



Convex polygon range with a slope of up to only one of the longest diagonal. Therefore, with a slope of the rotating caliper only folder to one of the longest diagonal.

The number of the longest diagonal of the convex polygon, no more than the number of vertices.

N points in the plane, the convex hull of a maximum of N vertices. Each vertex, with the caliper rotation may be as under one of the longest diagonal endpoints, you can infer a maximum of N convex hull of the longest diagonal, form a positive polygonal star.

The farthest point of the N points on the plane right, only a maximum of N group.

Algorithm

```
One, find the convex hull filter out may be the farthest point o
he point. O (NlogN)
Rotating calipers to identify the longest diagonal, that is the
thest point. O (N)
```

PKU 2187 **UVa** 11012

Segment Intersection (Under Construction!)

Degree of the ★ difficulty ★ ★ ★

Segment Intersection

Using the method of exhaustion exhaustive two segments, the time complexity is O (N ^ 2) can be obtained all the intersection.

The algorithm described here is sort of the first segment, and then from left to right order of exhaustive of each segment, to determine the intersect, the time complexity is O ((N + K) * logN). Similar to the nearest point on the method of exhaustion.

```
First, sort all endpoints:
  A, X-coordinates, from small to big.
  B, Y coordinates, from small to big.
  C, the left point at the right end point.
  D, the next endpoint in on the endpoint. The following can be u
d as the left can be used as on the right.
Two sequentially scan each endpoint:
  A, if the left endpoint, the segment into the binary tree. And
termine whether the intersection, seeking the point of intersect
.
  B, if the right end point, come up with segments from a binary
ee.
  C. If the intersection of the order of reversal of the corresp
ing segment in which the binary tree.
```

If you just want to determine whether there is an intersection, not to find out all the intersection, then changed the algorithm easier to implement in the form of the time complexity of O (NlogN). Refer to CLRS.

```
Shamos-Hoey O (NlogN)
Bentley-Ottmann O (NlogN + KlogK) = O ((N + K) * logN) K <C (N, 2
<N ^ 2
Chazelle & Edelsbrunner O (NlogN + K)
Balaban O (NloglogN + K)
```

ICPC [4125]