Геометрични задачи

за програмиране

Основни обекти в равнината:

Вектор

Точка

Две точки

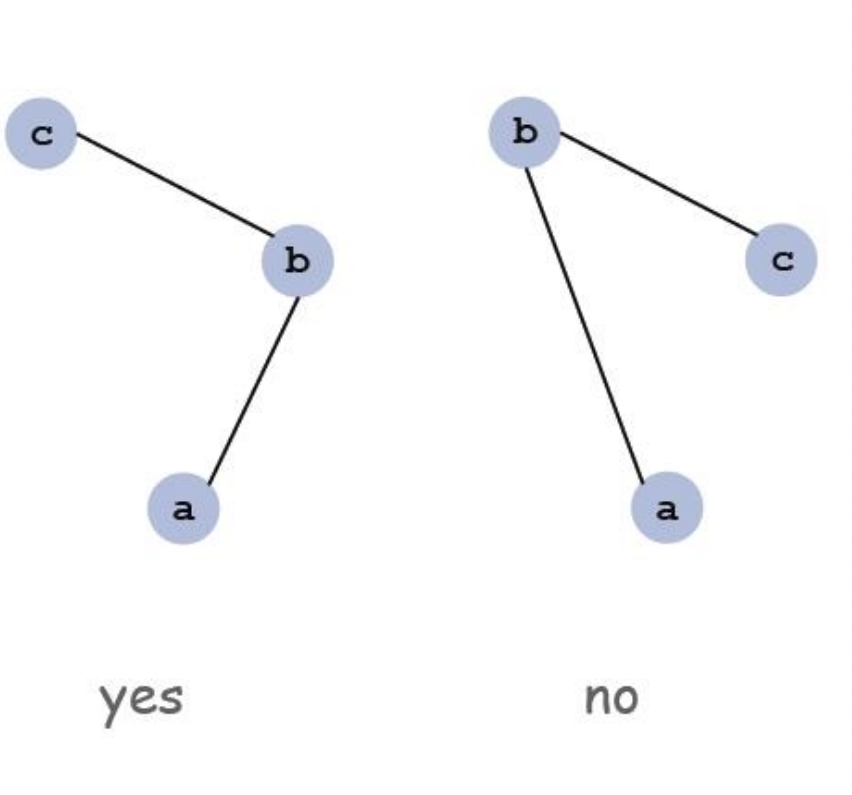
Три точки (два вектора)

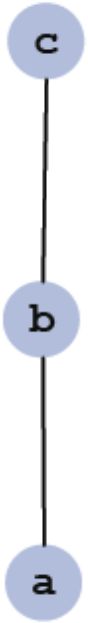
Целочислени координати

Права

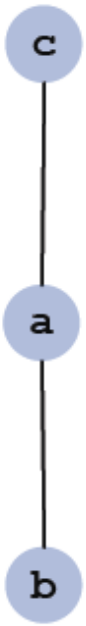
Ориентация на тройка точки

CCW: Кога трите точки *a*, *b* и *c* (в посочената последователност) са наредени в посока обратна на часовниковата стрелка?

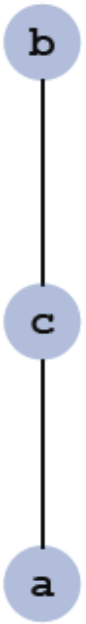




???
(collinear)

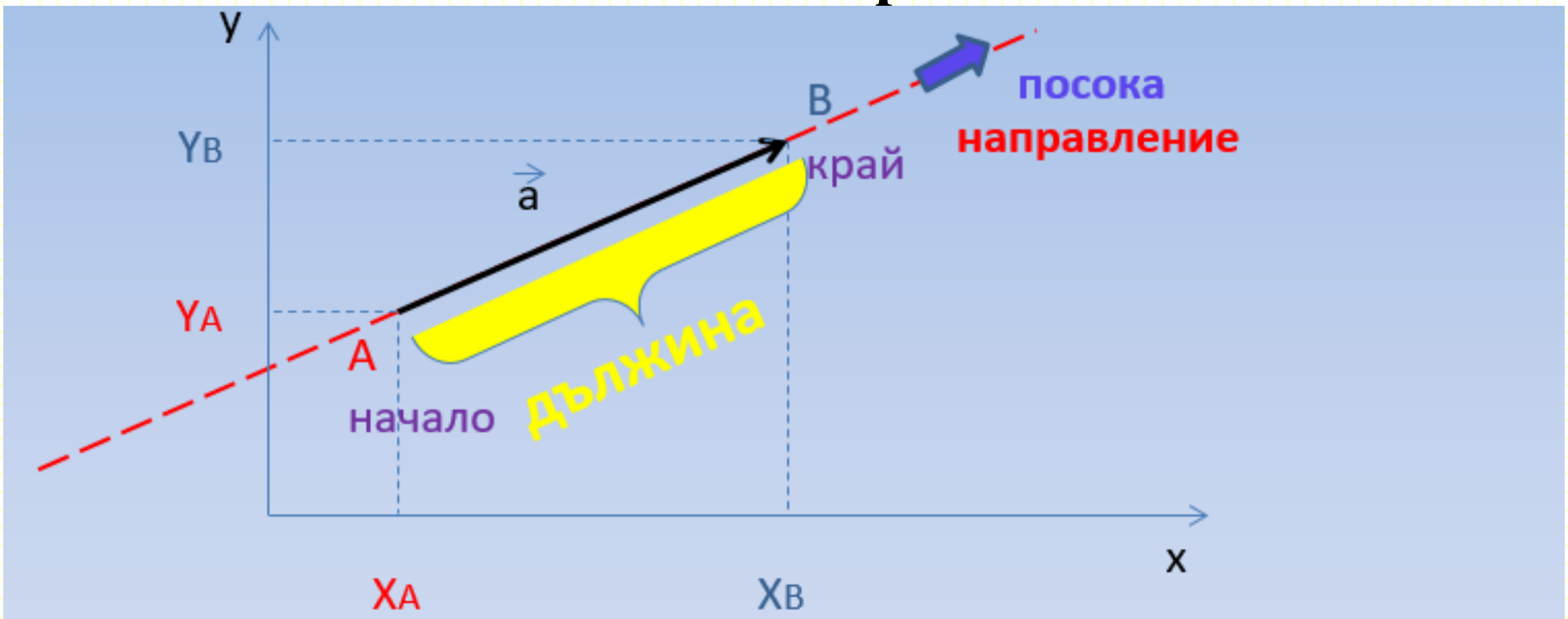


???
(collinear)



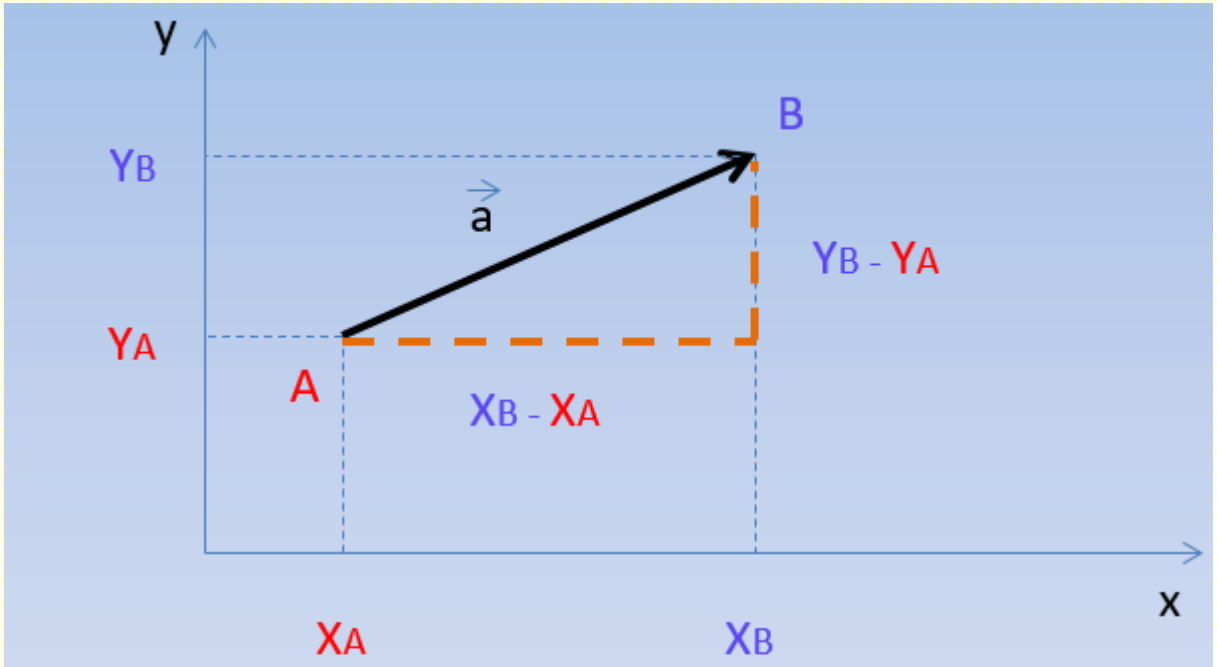
???
(collinear)

Вектор



Означения на вектор:

- с малка удебелена буква на латиница: **a, b, p, …**
- с буква на латиница и стрелка над нея $\vec{a}$, $\vec{b}$, ….
- с буквите на началото и края и стрелка над тях $\overrightarrow{AB}$, $\overrightarrow{MN}$, …



Координати на вектор: $\vec{a}(X_B - X_A , Y_B - Y_A) = (x_a , y_a)$

или $\begin{cases} x_a = X_B - X_A \\ y_a = Y_B - Y_A \end{cases}$

Формула за пресмятане лице на триъгълник

Нека е даден триъгълникът ABC с върхове A(x1 , y1), B(x2 , y2) и C(x3 , y3).
Пресмятаме следната детерминанта

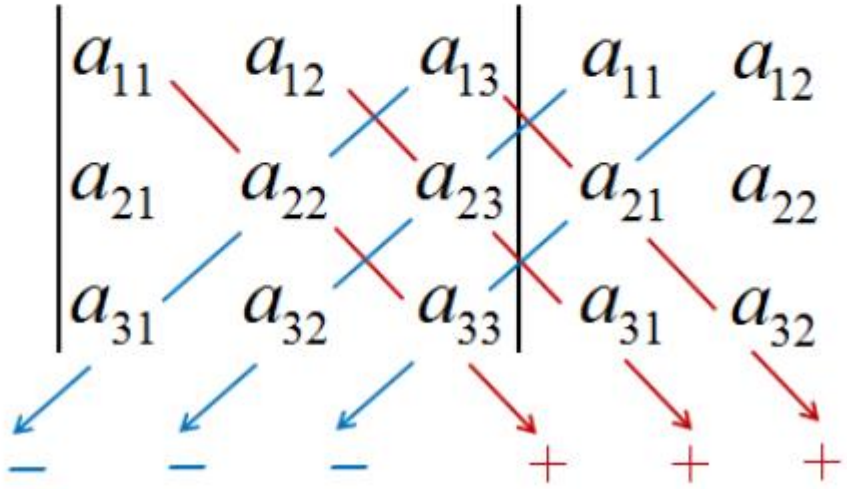
$$\Delta = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Тогава $S_{ABC} = \frac{1}{2}|\Delta|$

Пример. В равнината Oxy относно декартова координатна система е даден триъгълник ABC с върхове A(2, −2), B(7, 3) и C(5, −1). Намерете лицето на триъгълник ABC.

Пресмятане на детерминанта от трети ред:

- ***Правило на Сарус***



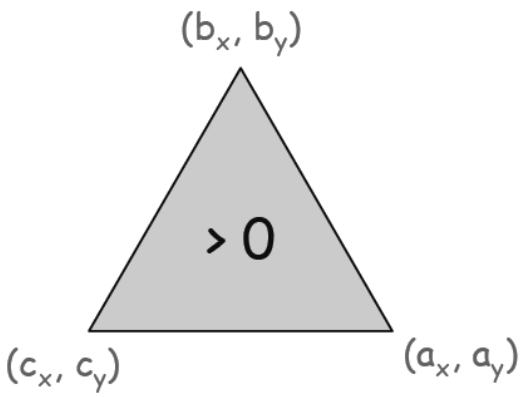
Може да представим пресмятането на детерминантата по следния начин:

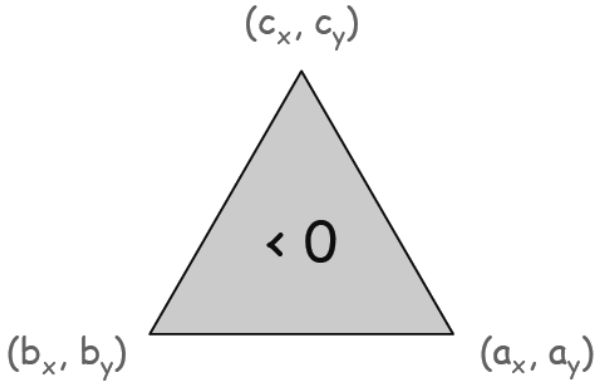
$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

Ако area > 0, тогава точките са наредени в посока обратна на часовниковата стрелка - counterclockwise.

Ако area < 0, тогава точките са наредени по посока на часовниковата стрелка - clockwise.

Ако area = 0, тогава точките са върху една права - collinear.

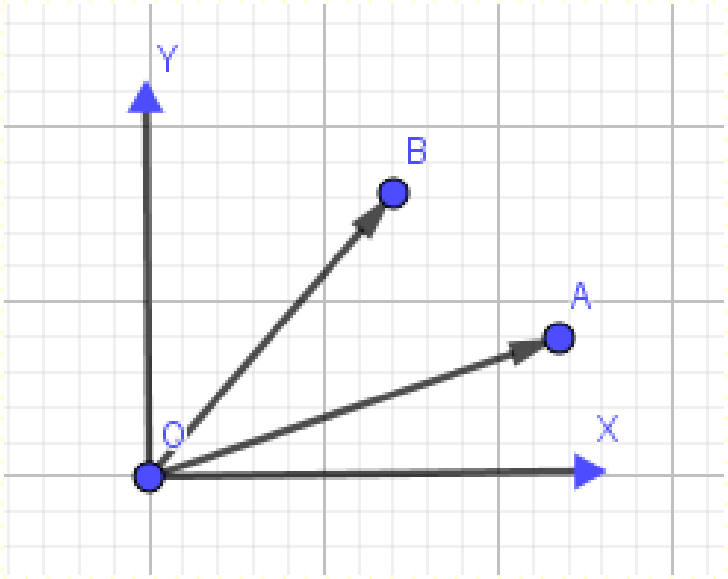




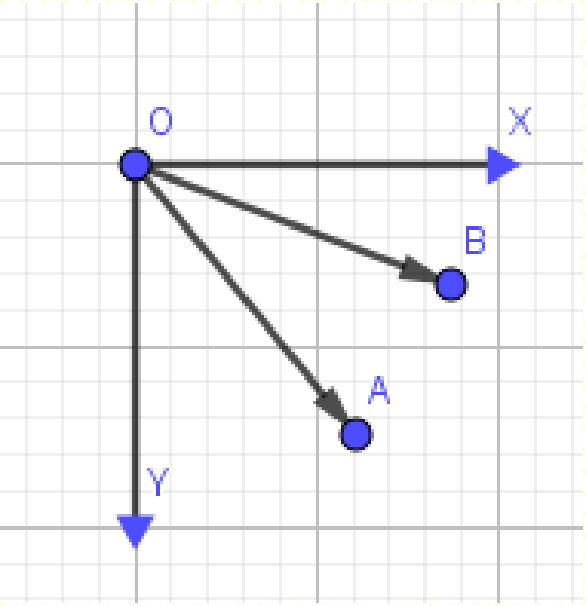

```
struct point{int x; int y;};
int cp(point p0, point p1, point p2)
{
   int dx1=p1.x-p0.x;
   int dy1=p1.y-p0.y;
   int dx2=p2.x-p0.x;
   int dy2=p2.y-p0.y;
   return dx1*dy2-dy1*dx2;
}
```

векторно произведение,
лицево произведение,
кръстосано произведение

cross product



При стандартна (математическа) координатна система:
движението обратно на часовниковата стрелка (CCW) от A към B
стойността на ср(OA,OB) е положителна



При друга координатна система (например екранна):
движението обратно на часовниковата стрелка (CCW) от A към B
стойността на ср(OA,OB) е отрицателна

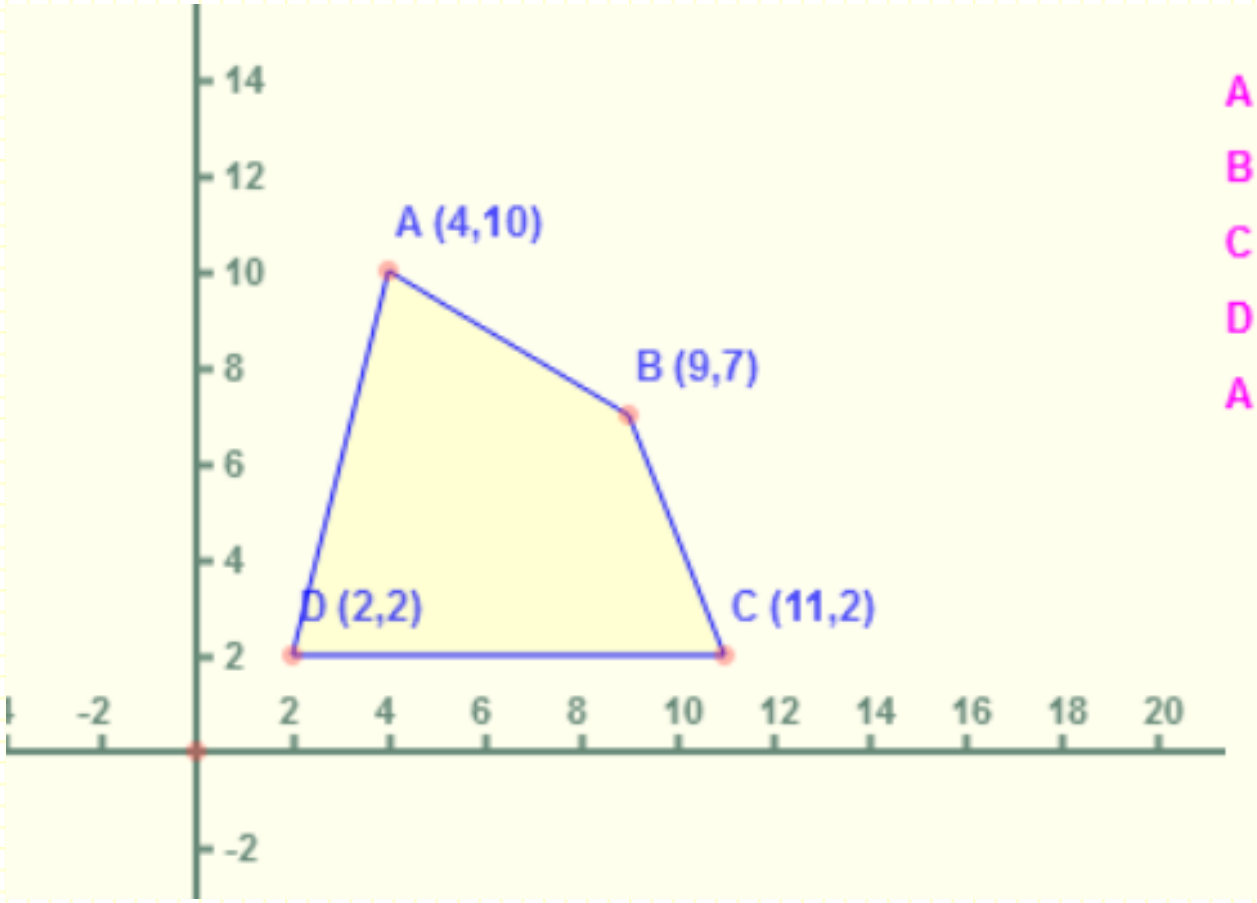
Задачи, които се решават с кръстосано произведение

Лица:

1. Пресмятане лице на триъгълник и успоредник

2. Пресмятане лице на многоъгълник (с наредени върхове)

```
// (X[i], Y[i]) са координати на i-тата точка
double polygonArea(double X[], double Y[], int n)
{
    double area = 0.0;  int j = n - 1;
    for (int i = 0; i < n; i++)
    {
     area += (X[j] + X[i]) * (Y[j] - Y[i]);
     j = i;
    }
    return fabs(area / 2.0);
}
```



	x	y
A	4	10
B	9	7
C	11	2
D	2	2
A	4	10

$28 - 90 = -62$

$18 - 77 = -59$

$22 - 4 = 18$

$20 - 8 = 12$

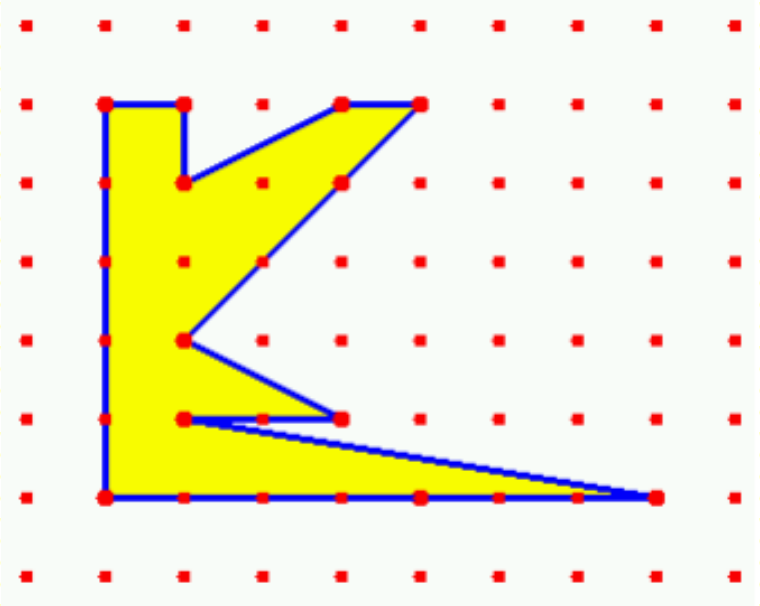
Total: −91

$$\text{Area} = \left| \frac{-91}{2} \right| = 45.5$$

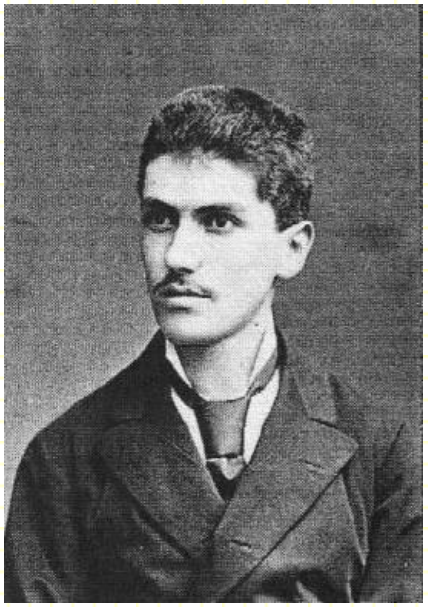
Формула на Пик (Pick)

намиране лице на многоъгълник с върхове във възли на квадратнамрежа

Задача: Намери лицето на фигурата, ако единичното квадратче има дължина на страната 1 единица.



Георг Александър Пик



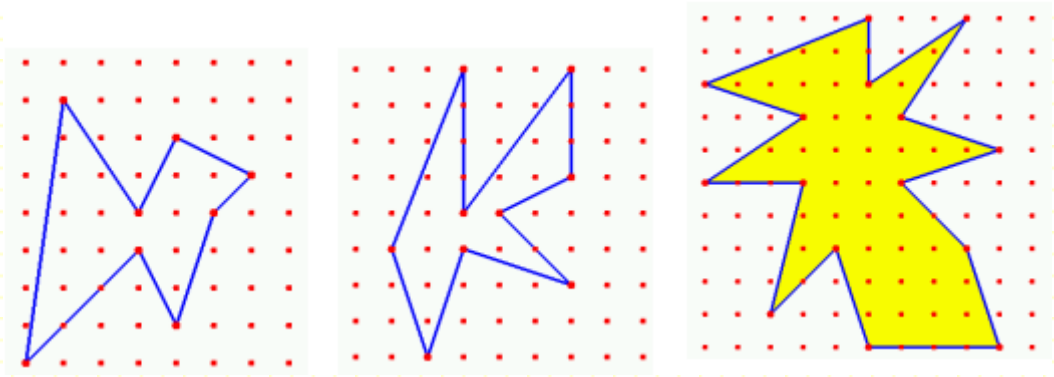
Формула за намиране на лицето ***S*** на многоъгълник с върхове във възлите на квадратна мрежа

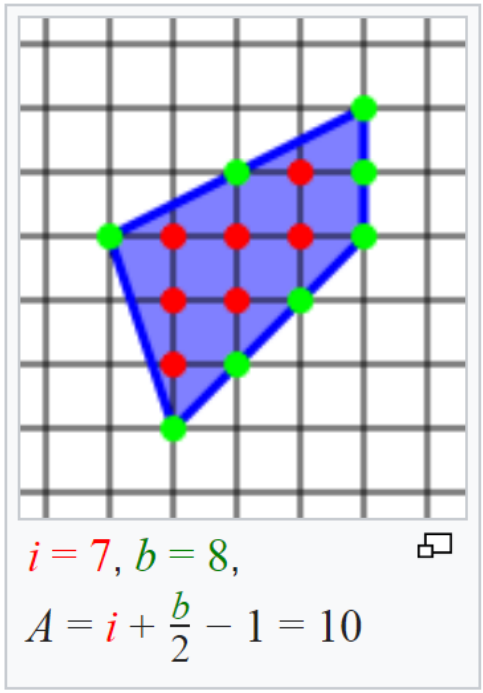
v - брой на възлите във вътрешността на многоъгълника

k - брой на възлите по неговия контур, включително и върховете на многоъгълника.

$$S = v + \frac{k}{2} - 1$$

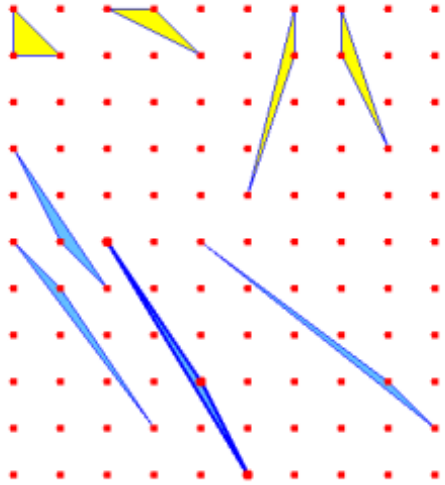
Задача. Намери лицето на фигурата.





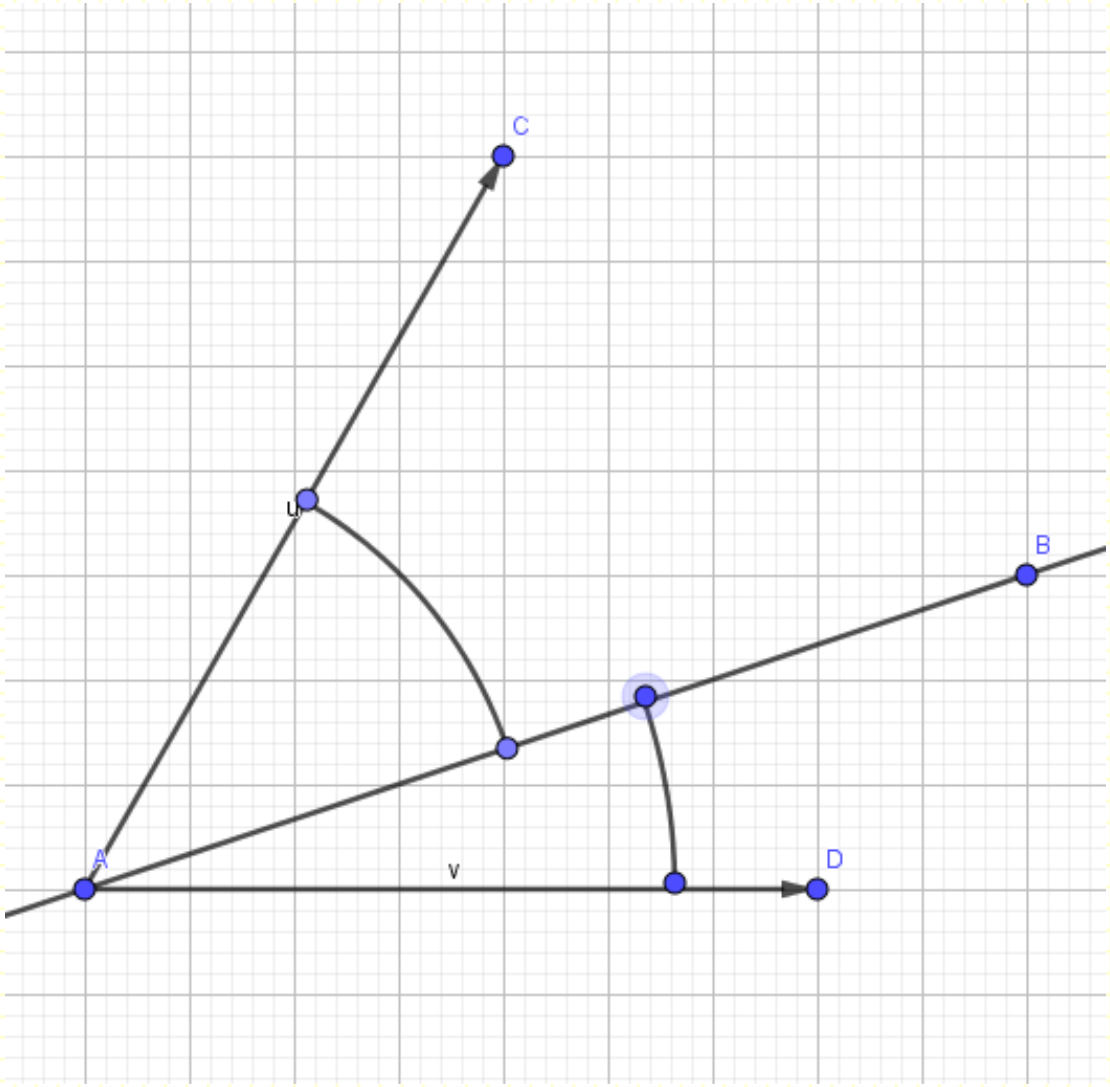
Задача. В квадратна мрежа с дължина на страната на единичното квадратче 1 см построй триъгълници с лице 0,5 кв. см.

Решение: Търсените триъгълници нямат вътрешни възли и единствените възли върху страните им са във върховете на триъгълника

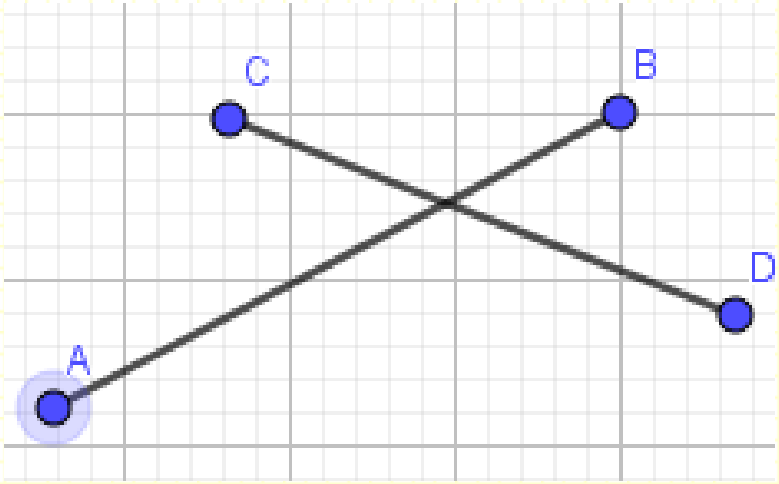


Местоположения:
1. Определяне дали две точки C и D лежат от различни полуравнини на права линия AB

cp(A,B,C) и cp(A,B,D) имат различни знаци.

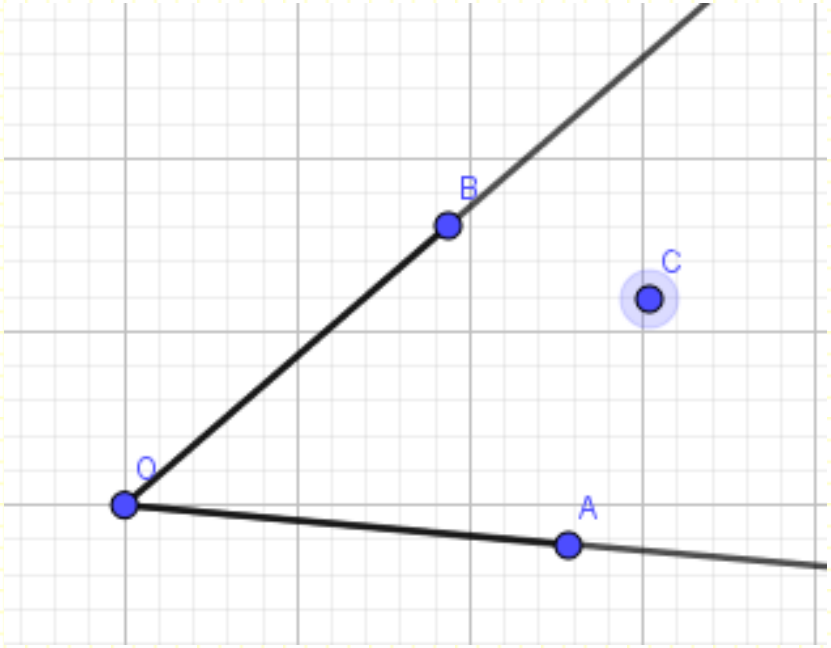


2. Определяне дали две отсечки AB и CD се пресичат



cp(A,B,C) и cp(A,B,D) имат различни знаци.

cp(C,D,A) и cp(C,D,B) имат различни знаци.

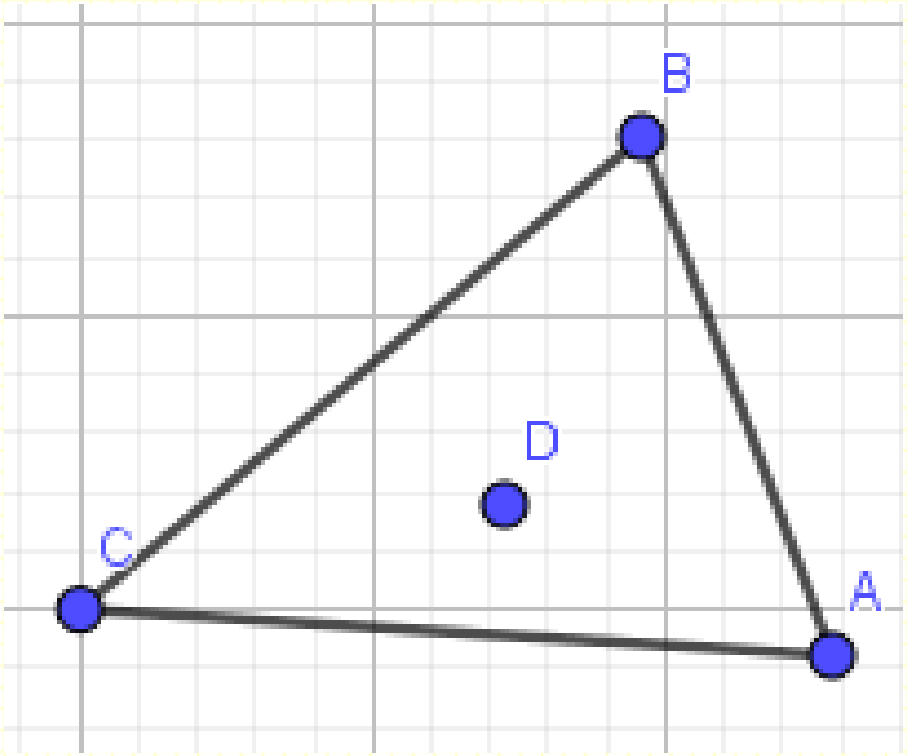


3. Определяне дали точка C е вътрешна за ъгъл OAB

Проверяваме дали C лежи в същата полуравнина спрямо OA, в която лежи B
и
Проверяваме дали C лежи в същата полуравнина спрямо OB, в която лежи A, т.е.

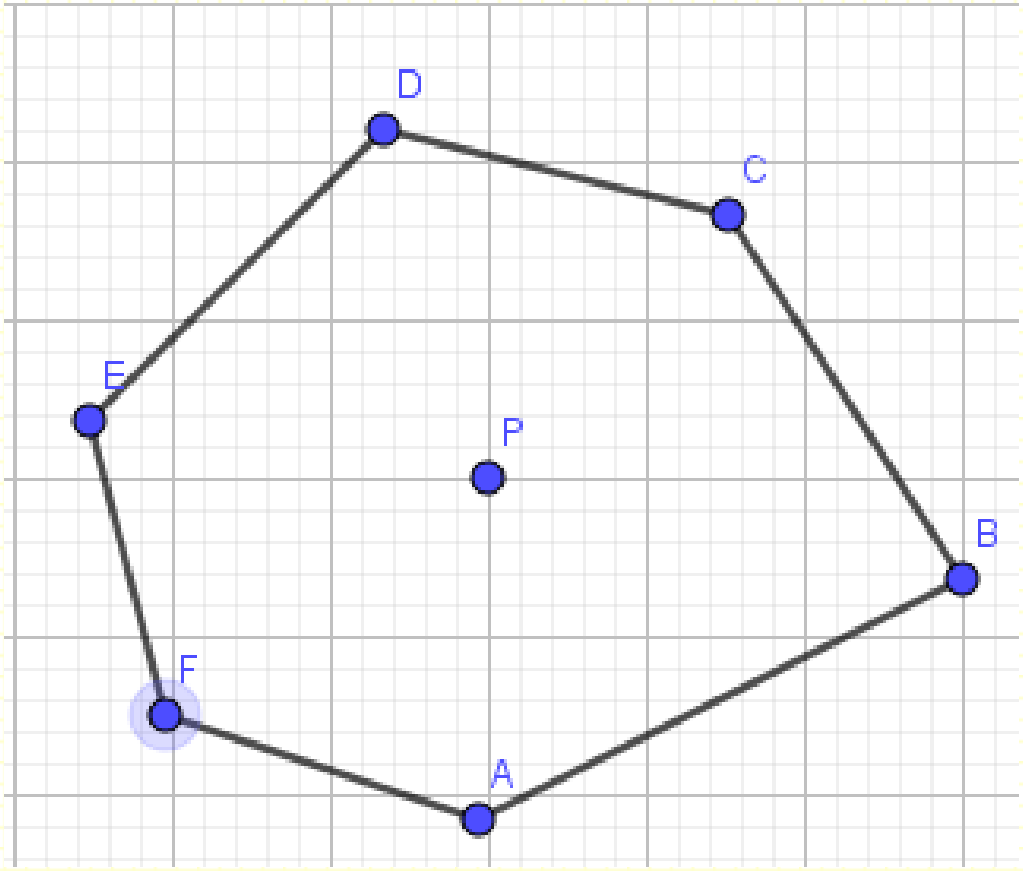
cp(OAC) и cp(OAB) имат еднакви знаци и
cp(OBC) и cp(OBA) имат еднакви знаци

4. Определяне дали една точка е вътрешна за триъгълник



cp(ABD), cp(BCD), cp(CAD)

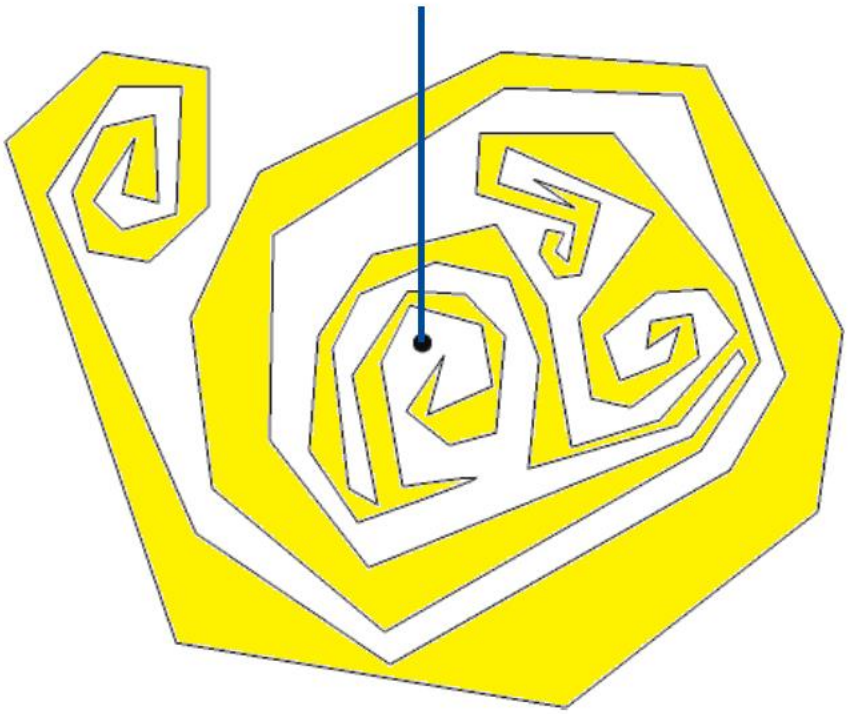
5. Определяне дали една точка е вътрешна за (прост и **изпъкнал**) многоъгълник



ср(ABC), ср(BCD), ср(CDE),... имат еднакви знаци за изпъкнал многоъгълник

P е вътрешна, ако ср(ABP), ср(BCP), ср(CDP), ... имат еднакви знаци

Произволен прост (т.е. без самопресичания) многоъгълник



Метод с лъч. Броим колко пъти лъчът пресича страни на многоъгълника (четен или нечетен). Усложнение: когато лъчът пресече връх.

Скаларно произведение на два вектора

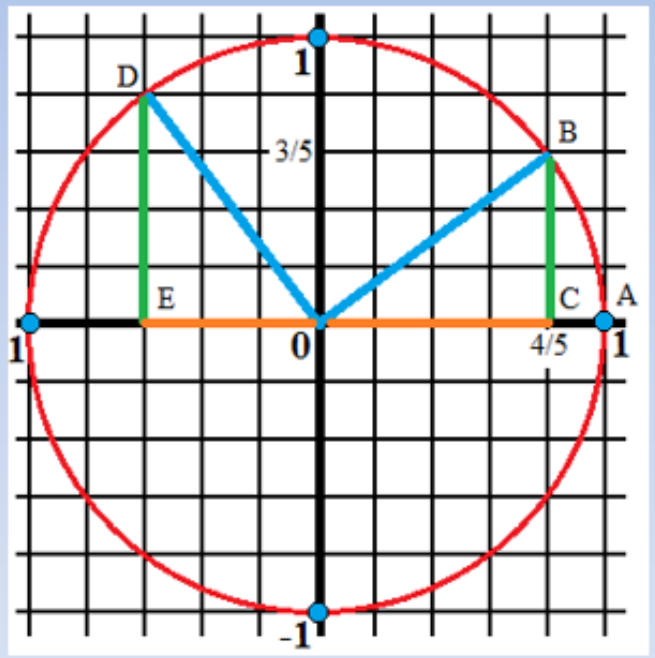
Скаларното произведение на векторите **a** и **b** ще го означим с (a,b)

$$(a,b) = |a|.|b|.\cos\varphi$$

където φ е ъгълът между **a** и **b**.

Изразено с координати, скаларното произведение е

$$(a,b) = x_a.x_b + y_a.y_b$$



Намерете скаларното произведение на векторите $\overrightarrow{OA}$ и $\overrightarrow{OB}$ и косинуса на ъгъла, заключен между тях.

Скаларно произведение на два вектора (или тройка точки)
(dot product, scalar product)

```
struct point{int x; int y;};
int sp(point p0, point p1, point p2)
{
  int dx1=p1.x-p0.x;
  int dy1=p1.y-p0.y;
  int dx2=p2.x-p0.x;
  int dy2=p2.y-p0.y;
  return dx1*dx2+dy1*dy2;
}
```

Модификации на `cp` и `sp` за вектори

```
int cp(v1, v2);

int sp(v1, v2);
```

Свойства:

`sp(v,v)` е **квадрата** на дължината на вектор `v`
дължината на v означаваме също като d(v) или като |v|.

sp(v1,v2) = |v1| |v2| cos(v1,v2);
cp(v1,v2) = |v1| |v2| sin(v1,v2);

sp(v1,v2) = 0, тогава и само тогава, когато v1 и v2 са перпендикулярни
cp(v1,v2) = 0, тогава и само тогава, когато v1 и v2 са колинеарни

Ъгълът между векторите може да се пресметне чрез acos или asin:
sp(v1,v2) = |v1| |v2| cos(v1,v2
cos(v1,v2) = sp(v1,v2)/(|v1| |v2|)

ъгъл (v1,v2) = acos (sp(v1,v2)/(|v1| |v2|))
Тук sp(v1,v2) се пресмята чрез координатите на векторите и дължините се смятат по познатия начин.
Аналогично   при cp(v1,v2) = |v1| |v2| sin(v1,v2)

Модификация на cp и sp за двойки от STL

cp(pair<int, int>, pair<int, int>)
sp(pair<int, int>, pair<int, int>)

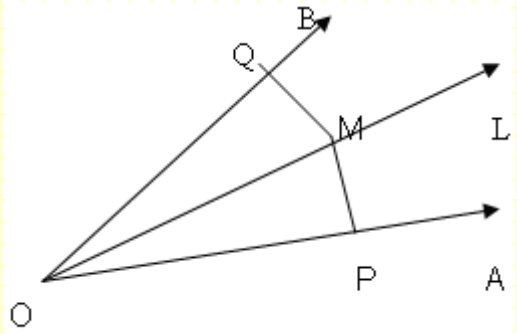
Задачи:

1. Среда C на отсечка AB

$Cx=(Ax+Bx)/2$
$Cy=(Ay+By)/2$

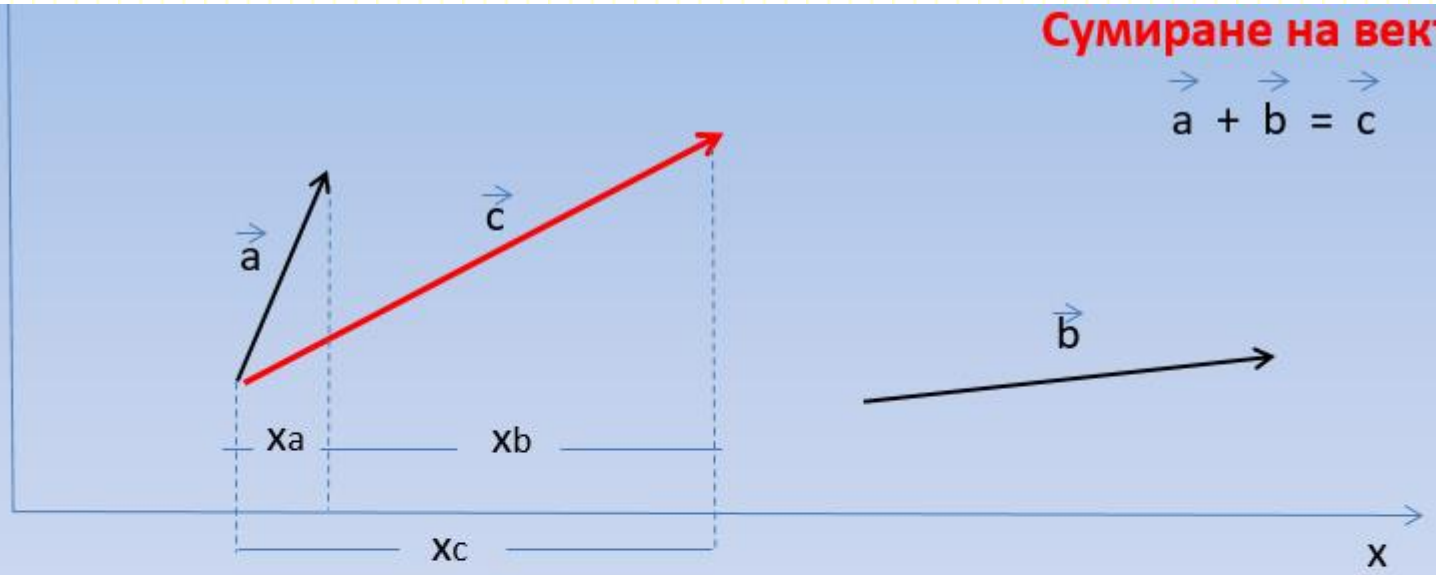
2. Ъглополовяща на ъгъл OAB

вектор OA+OB



Сумиране на вектори:

$\vec{a} + \vec{b} = \vec{c}$

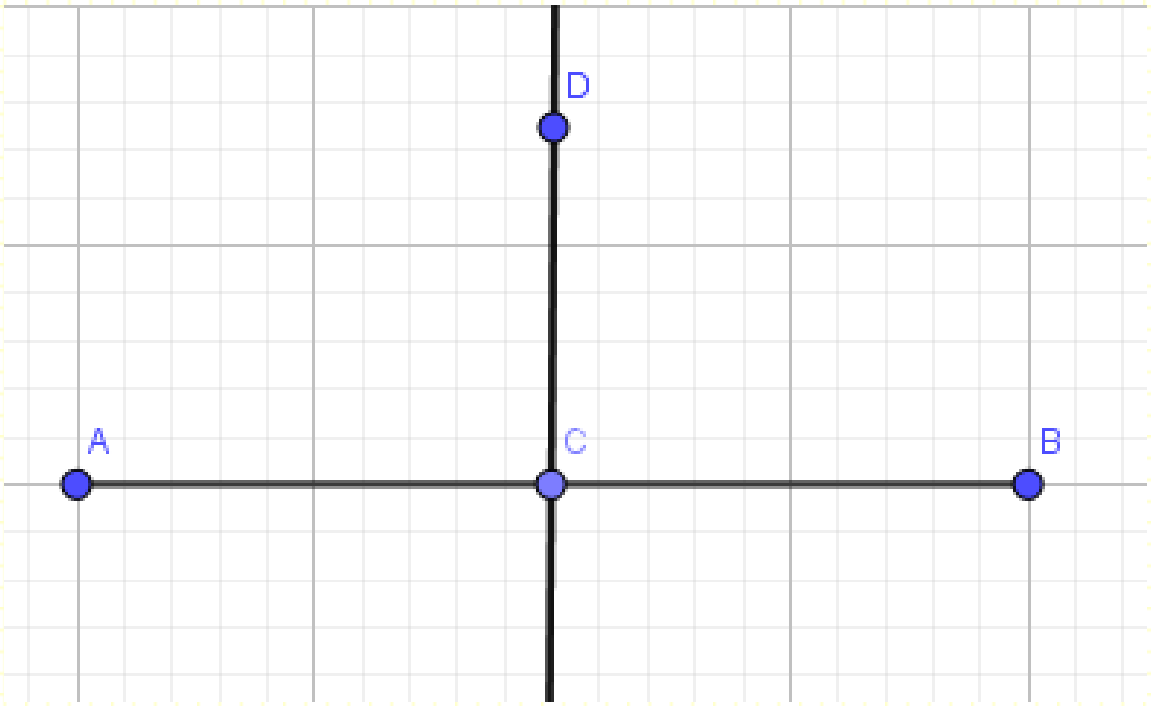


Правило: Поставяме началото на вектор $\vec{b}$ в края на вектор $\vec{a}$.
Сумата е вектор $\vec{c}$, чието начало съвпада с началото на $\vec{a}$ и краят му - с края на $\vec{b}$.

Координати на с: $\vec{a}\,(x_a, y_a) + \vec{b}\,(x_b, y_b) = \vec{c}\,(x_a + x_b, y_a + y_b) = \vec{c}\,(x_c, y_c)$

или $\begin{cases} x_c = x_a + x_b \\ y_c = y_a + y_b \end{cases}$

3. Симетрала на отсечка AB



Пресмятане C да е среда на AB

Пресмятаме D така че векторът CD да е перпендикулярен на AB

Ако x,y са координатите на AB, тогава -y,x са координати на CD, защото скаларното произведение е 0.

Още за кръстосаното произведение на два вектора в равнината

За двата вектора $a = (a_1, a_2)$ и $b = (b_1, b_2)$ тяхно кръстосано произведение наричаме числото:
$\text{ср}(a,b) = a_1 b_2 - a_2 b_1$.

В сила са свойствата (линейна антисиметрична функция):
$\text{ср}(a,a) = 0$;
$\text{ср}(a,b) = -\text{ср}(b,a)$.
Ако d е вектор, такъв че $d = t\,e + u\,f$, където e и f са други вектори, а t и u са числа, тогава:
$\text{ср}(a,d) = \text{ср}(a, t\,e + u\,f) = t\,\text{ср}(a,e) + u\,\text{ср}(a,f)$ (дистрибутивно свойство)

```
struct dpoint {double x; double y;};
typedef dpoint dvec;

double cp(dvec a, dvec b)
{
  return a.x*b.y-a.y*b.x;
}
```

Решаване на система от две линейни уравнения

$a_1 x + b_1 y = c_1$

$a_2 x + b_2 y = c_2$,

където неизвестните са x и y.

Умножаваме първото уравнение с b_2, а второто с b_1 и изваждаме първото уравнение от второто:

$(a_2 b_1 - a_1 b_2) x = b_1 c_2 - b_2 c_1$

Когато коефициентът $(a_2 b_1 - a_1 b_2)$ пред x е различен от нула, получаваме:

$x = (b_1 c_2 - b_2 c_1) / (a_2 b_1 - a_1 b_2)$

Ако образуваме от a_1 и a_2 вектор $a = (a_1, a_2)$ и аналогично запишем $b = (b_1, b_2)$ и $c = (c_1, c_2)$, тогава x се изразява чрез частното на две кръстосани произведения: $x = \text{ср}(b,c) / \text{ср}(b,a)$, където е използвано, че кръстосаното произведени $\text{ср}(a,b)$ на два вектора a и b е равно на $a_1 b_2 - a_2 b_1$.

Аналогично може да изразим стойността на y. Така решението на системата уравнения е:

$x = \text{ср}(b,c) / \text{ср}(b,a)$
$y = \text{ср}(a,c) / \text{ср}(a,b)$

```
bool solve_S2LE(dvec a, dvec b, dvec c, dvec& r)
{
   if(cp(a,b)==0) return false;

   r.x=cp(b,c)/cp(b,a);
   r.y=cp(a,c)/cp(a,b);

   return true;
}
```

Намиране на координатите на пресечна точка на две прави

Правите са зададени чрез координатите на две свои различни точки, съответно чрез точките z_{11} и z_{12} за първата права, и z_{21}и z_{22} за втората права.

Означаваме с d_1 векторът определен от точка z_{11} до z_{12} и в посока към z_{12}, и аналогично означаваме с d_2 векторът определен от точка z_{21} до z_{22} и в посока към z_{22}.

Всичките точки от първата права имат радиус-вектор от вида $z_{11} + t\, d_1$, където t пробягва всичките реални числа,
а всичките точки от втората права имат радиус-вектор от вида $z_{21} + u\, d_2$, където u пробягва всичките реални числа.

Търсим стойности на t и u, за които

$z_{11} + t\, d_1 = z_{21} + u\, d_2.$

Равенството при означението $d = z_{21} - z_{11}$ може да запишем като векторно равенство:

$t\, d_1 - u\, d_2 = d$

Умножаваме (като скаларно произведение) двете страни на горното равенство с d_1, и след това същото равенство умножаваме с d_2. Така получаваме система от две числови уравнения за неизвестните t и u:

$t\,(d_1\, d_1) - u\,(d_1 d_2) = (d_1 d)$

$t\,(d_1\, d_2) - u\,(d_2 d_2) = (d_2 d)$

След като решим тази система за t, търсената пресечна точка на двете прави се определя с радиус-вектора:
$z_{11} + t\, d_1$.

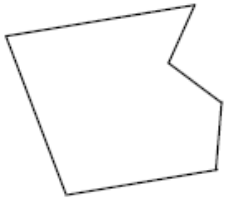
Когато знаменателят ср(d_1,d_2) е равен на 0, тогава двете прави са или успоредни или съвпадат. В този случай, за да съвпаднат правите, трябва ср(d,d_2)=0.

```
bool cross_(dpoint z11,dpoint z12,
            dpoint z21,dpoint z22, dpoint& r)
{
    dvec d1;
      d1.x=z12.x-z11.x;
      d1.y=z12.y-z11.y;
    dvec d2;
      d2.x=z22.x-z21.x;
      d2.y=z22.y-z21.y;
    dvec d;
      d.x=z21.x-z11.x;
      d.y=z21.y-z11.y;
   if(cp(d1,d2)!=0)
    { double t=cp(d,d2)/cp(d1,d2);
      r.x=z11.x + t*d1.x;
      r.y=z11.y + t*d1.y;
      return true;
    }
    else return false;
}
```

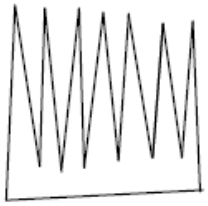
Различни многоъгълници

Is a given polygon simple?

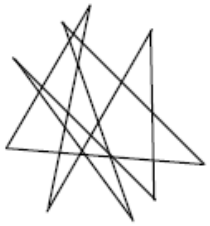
no crossings



1	6	5	8	7	2
7	8	6	4	2	1



1	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	2	18	4	18	4	19	4	19	4	20	3	20	3	20



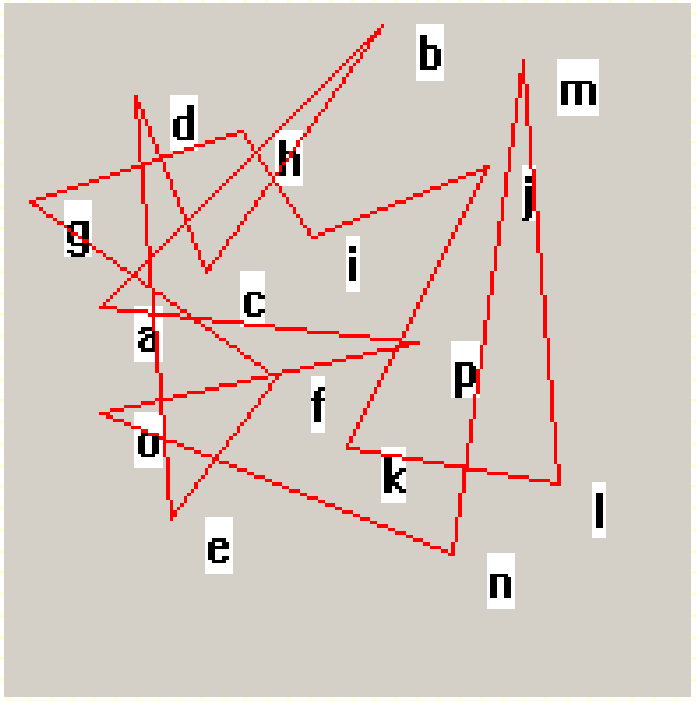
1	10	3	7	2	8	8	3	4
6	5	15	1	11	3	14	2	16

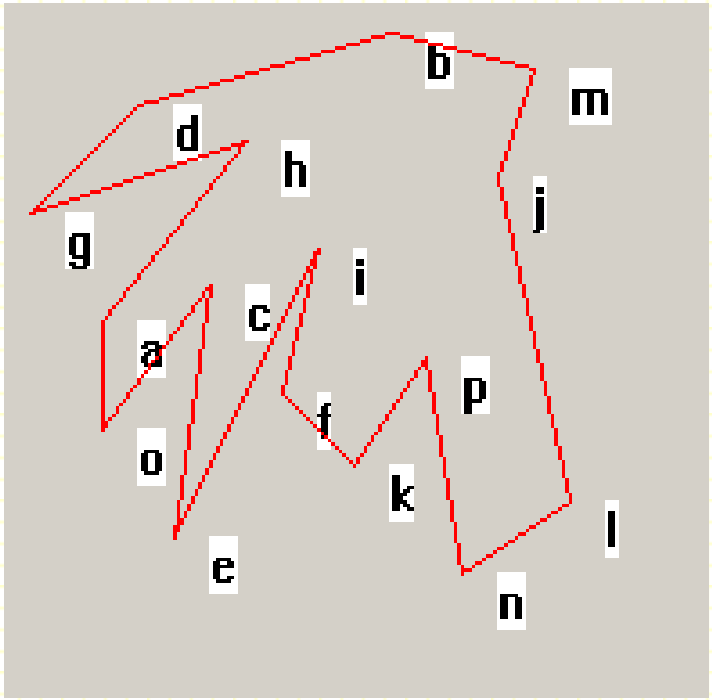
we think of this

algorithm sees this

Пренареждане на върхове на многоъгълник, за да се получи прост (т.е. без самопресичания) контур:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
x	30	110	60	40	50	80	10	70	90	140	100	160	150	130	30	120
y	90	10	80	30	150	110	60	40	70	50	130	140	20	160	120	100





```
void arrange()
{int m=0;
 for(int i=1; i<n; i++)
  if(p[i].y < p[m].y) m=i;
 for(int i=0; i<n; i++)
  if(p[i].y==p[m].y)
   if(p[i].x > p[m].x) m=i;
 swap(p[0],p[m]);
 sort(p+1,p+n,cmp);
}
```

p[0] има най-малкото y и ако има няколко точки с най-малко y, то е с най-голямото x измежду всичките точки с най-малкото y, т.е. p[0] е най-долу и най-вдясно.

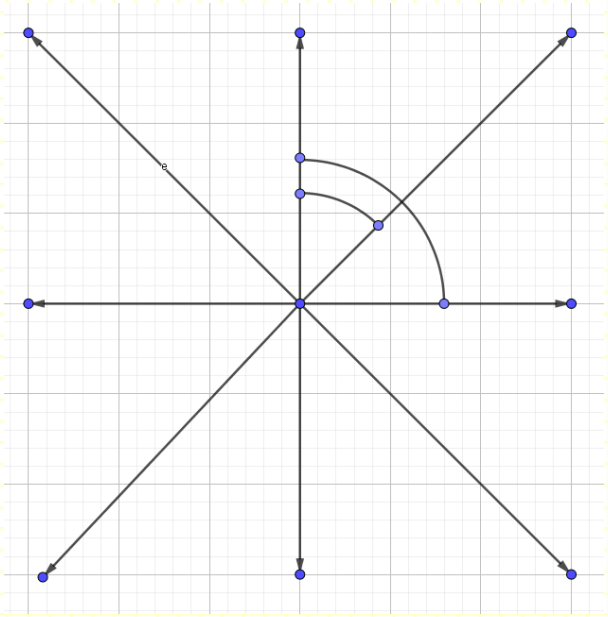
за `cmp` (за сортиране по ъгъл) може да използваме (лош подход):

Свързваме мислено p[0] с всяка от останалите точки p[i] и сортираме по ъгъла между оста Ox и лъча (p[0],p[i])

```
struct dpoint{double x; double y;};
bool cmp(dpoint p1, dpoint p2)
{return
  atan2(p1.y-p[0].y, p1.x-p[0].x) <
  atan2(p2.y-p[0].y, p2.x-p[0].x);
}
```

Функции atan2 и atan

```
struct dpoint{double x; double y;};
dpoint p[]={{0,  1},{1, 1},{1, 0},{1,-1},{0,-1},
            {-1,-1},{-1,0},{-1,1}};
for(int i=0;i<=7;i++) cout <<
(180.0/M_PI)*atan2(p[i].x,p[i].y) << endl; cout << endl;
for(int i=0;i<=7;i++) cout <<
(180.0/M_PI)*atan2(p[i].y,p[i].x) << endl; cout << endl;
for(int i=0;i<=7;i++) cout <<
(180.0/M_PI)*atan(p[i].x/p[i].y) << endl;
```



```
 0, 45, 90, 135, 180, -135, -90, -45

90, 45,  0, -45, -90, -135, 180, 135

 0, 45, 90, -45,  -0,   45, -90, -45
```

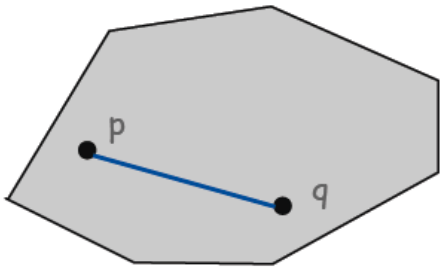
По-добро решение: без math (#include<cmath>):
p[0] е екстремната точка

```
bool cmp(dpoint p1, dpoint p2)
{
  double v=cp(p1-p[0],p2-p[0]);
  if(v!=0) return v<0;
  return dist(p1,p[0]) < dist(p2,p[0]);
}
```

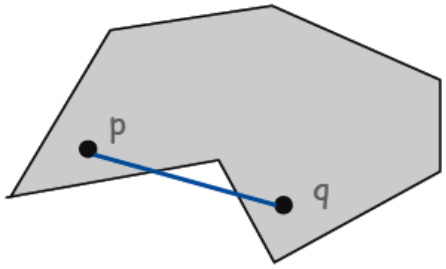
Предефиниране на операциите +, - и *

```
dpoint operator+(dpoint p1, dpoint p2)
{ dpoint p; p.x=p2.x+p1.x; p.y=p2.y+p2.y;
  return p;}

dpoint operator-(dpoint p1, dpoint p2)
{ dpoint p; p.x=p2.x-p1.x; p.y=p2.y-p2.y;
  return p;}

dpoint operator*(double d, dpoint p)
{ dpoint p1; p1.x=d*p.x; p1.y=d*p.y;
  return p1;}

double dist(dpoint p1, dpoint p2)
{return sqrt((p1.x-p2.x)*(p1.x-p2.x)+
             (p1.y-p2.y)*(p1.y-p2.y));}
```

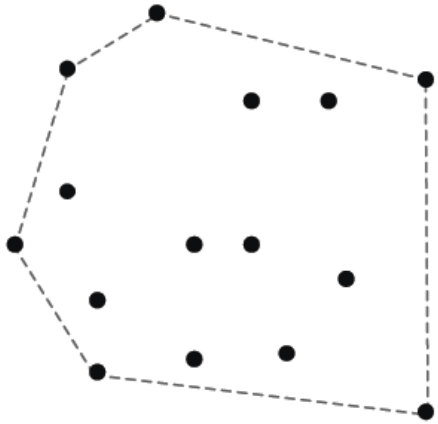
Изпъкнало множество от точки (convex set)
Изпъкнала обвивка (convex hull)



convex



not convex

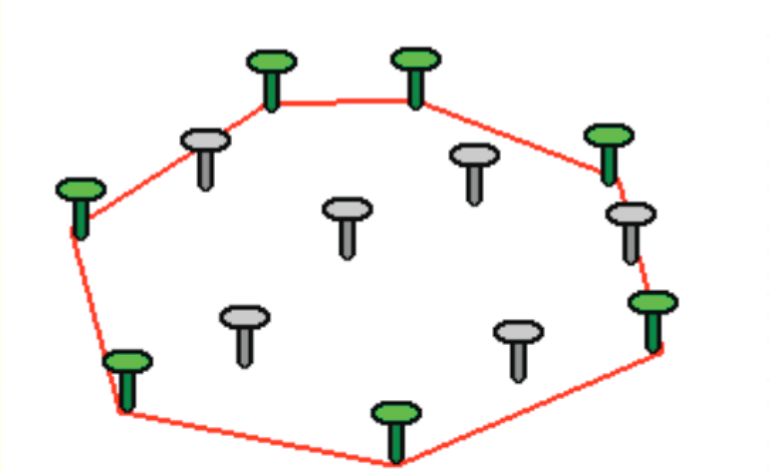


convex hull

Свойства на изпъкналата обвивка на краен брой точки

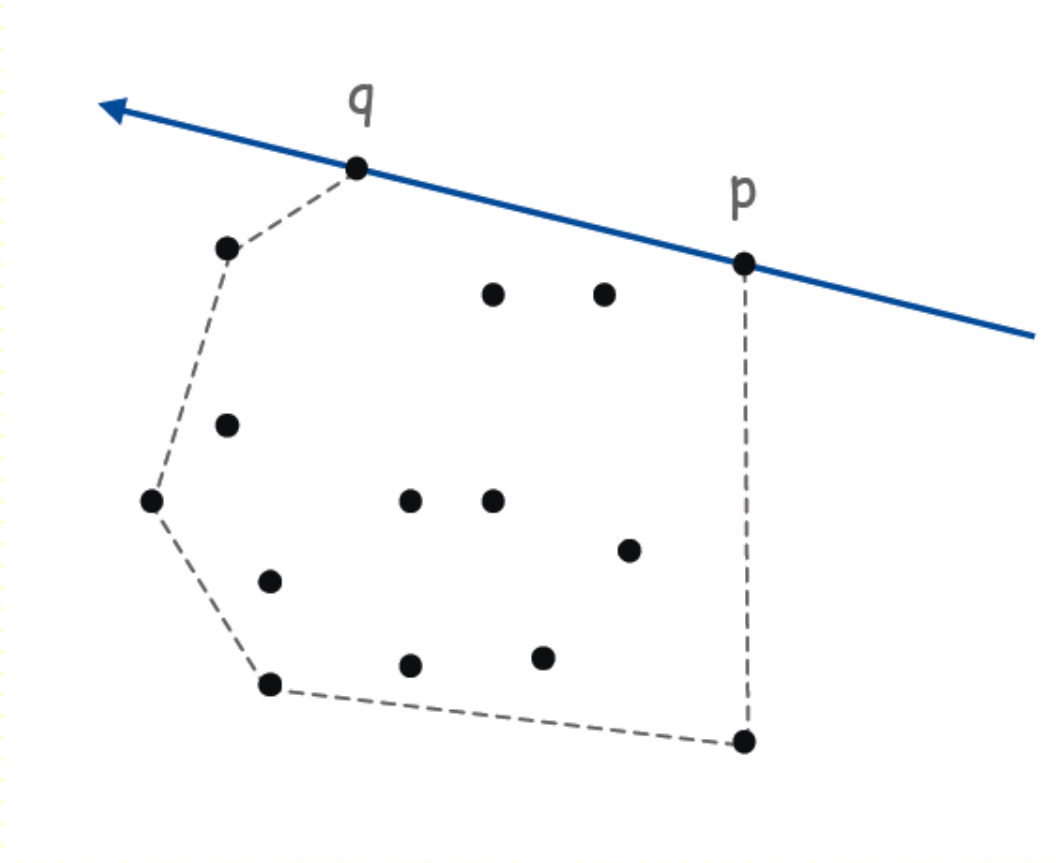
- Най-малък (по включване) изпъкнал многоъгълник
- Многоъгълник с най-малък периметър
- Многоъгълник с най-малко лице

Механична интерпретация:

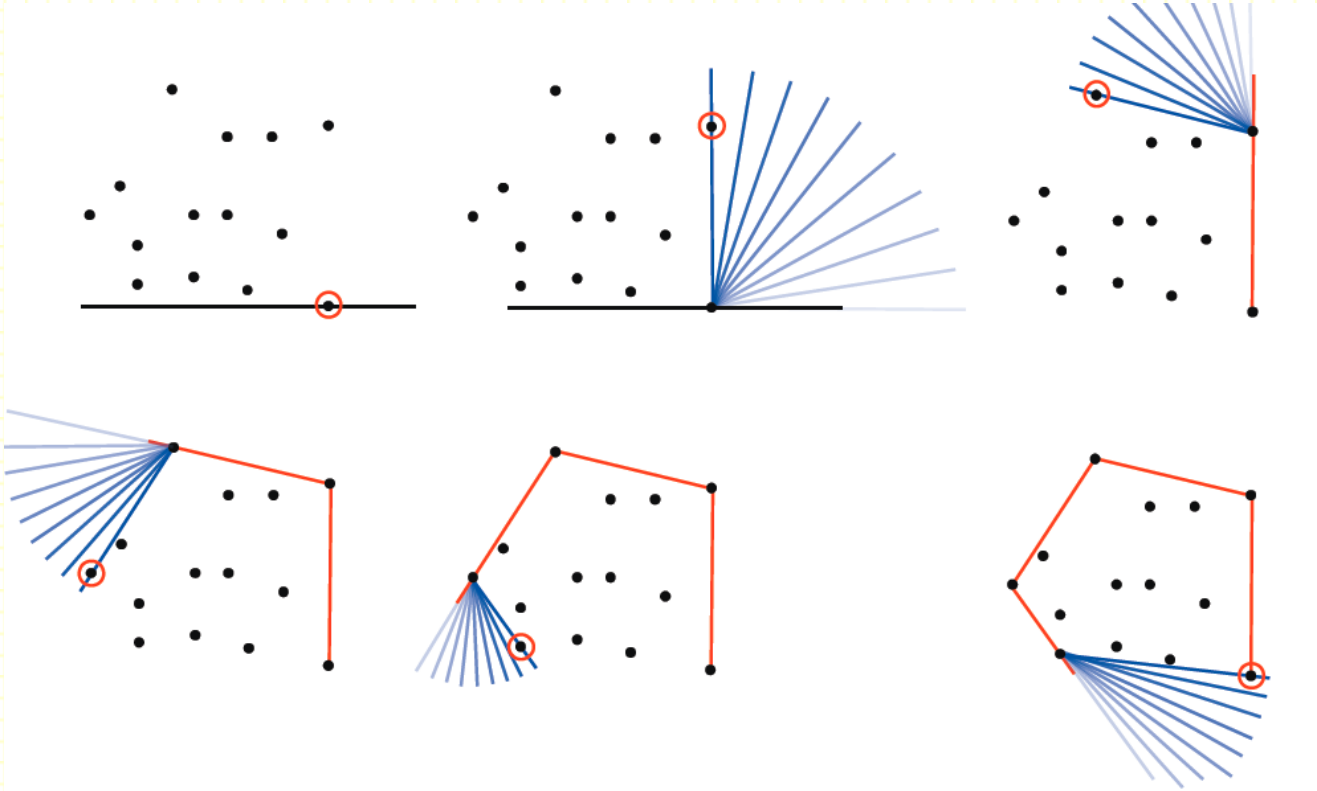


Намиране чрез brute force $O(N^3)$

Ребра през всеки две точки

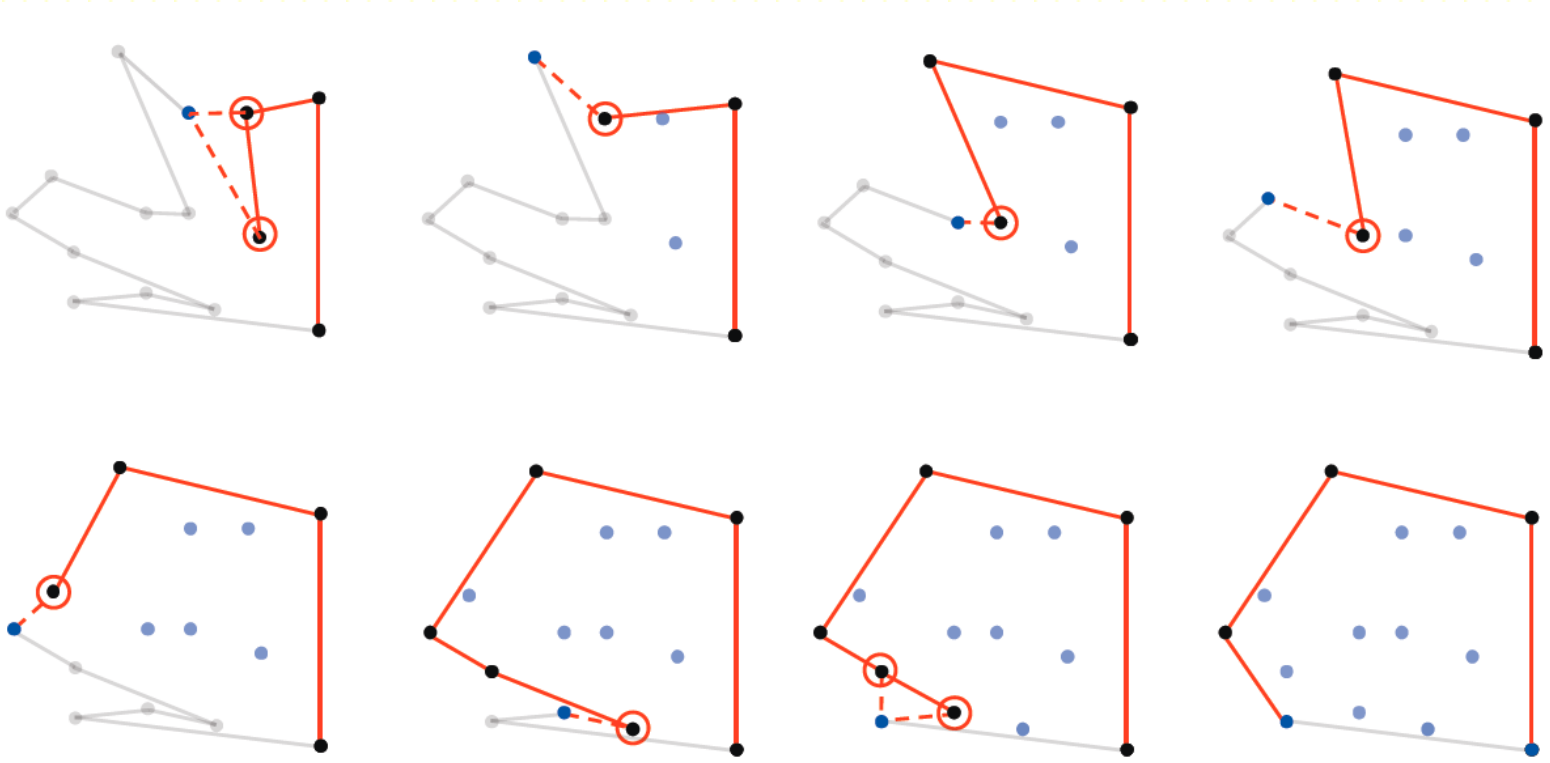


Метод на опаковането $O(N^2)$



Метод на Греъм (Graham) O(N log N)

- Избираме точка p най-долу и най-отдясно.
- Сортираме точките, за да образуват прост полигон.
- Обработваме поред тройки точки и изхвърляме средната точка, когато тройката е в посока на часовниковата стрелка (т.е. искаме само леви завои).



// **даден е прост полигон с върхове** p[0], p[1], .., p[n-1]

```
p[n]=p[0];
int m=2;
for(int i=3;i<=n;i++)
{
  while(cp(p[m],p[m-1],p[i])>=0) m--;
  m++; swap(i,m);
}
n=m;
```