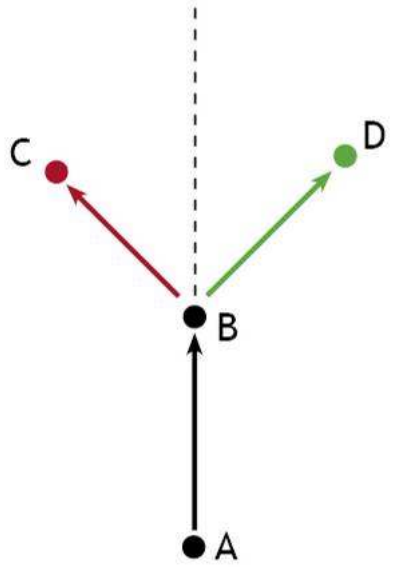
# Одбрани делови од пресметлива геометрија

Computational geometry

Се возите со автомобил по патот ***AB***. Но кога сте пристигнале до точката ***B***(крајот на патот ***AB***), треба да свртите или кон точката ***C*** или кон точката ***D***. Но која е пресметката која ќе ви каже дека од ***B*** сте свртеле кон ***C***, односно кон ***D***, во случај кога ги знаете координатите на контролните точки ***A, B, C*** и ***D***?




За да одговориме на ова прашање ќе се послужиме со векторскиот производ.

Векторскиот производ меѓу два вектори дава вектор со следниве координати:

$$\vec{a} = (a_1, a_2, a_3)$$

$$\vec{b} = (b_1, b_2, b_3)$$

$$\vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} i - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} j + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} k = (\begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix}, -\begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix}, \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix})$$

Во случај кога векторите ***a*** и ***b*** лежат во рамнината *Oxy*, тоа значи дека координатите $a_3$ и $b_3$ се еднакви на нула. Тоа значи дека

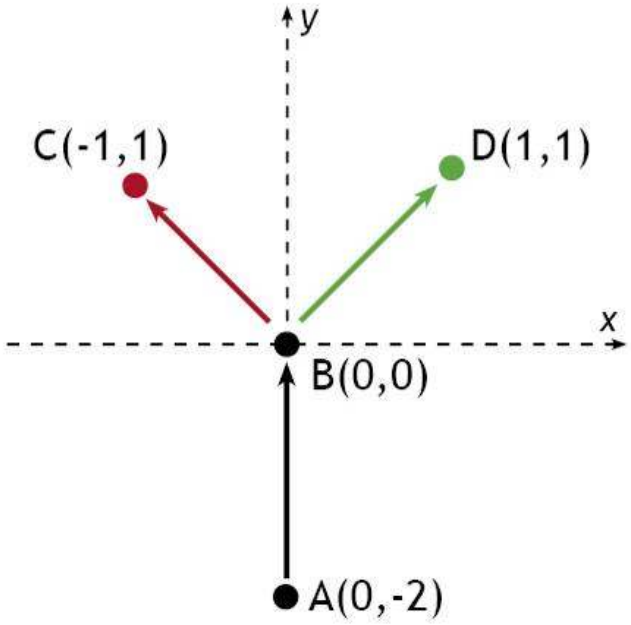
$$\vec{a} = (a_1, a_2, 0)$$

$$\vec{b} = (b_1, b_2, 0)$$

$$\vec{a} \times \vec{b} = (0,0, \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix}) = (0,0, a_1 b_2 - a_2 b_1)$$

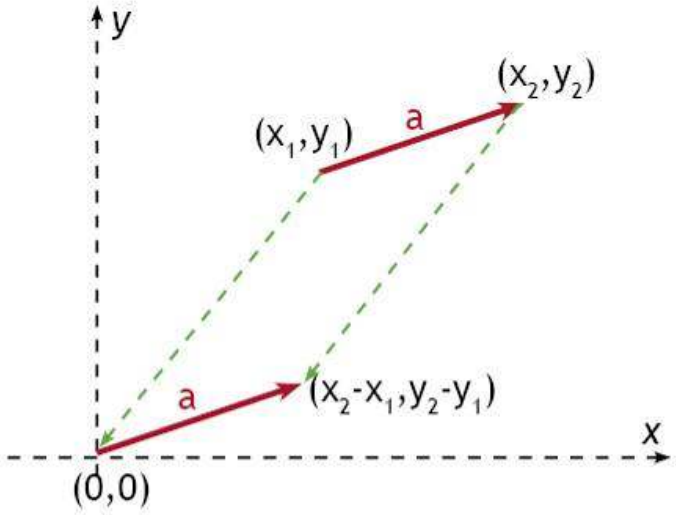
третата координата од векторскиот производ, односно $a_1*b_2-a_2*b_1$ ќе ни каже дали во однос на патот ***AB***, од ***B*** сме свртеле кон ***C***, односно кон ***D***.

Да го разгледаме случајот со патиштата, но така што овојпат ќе бидат зададени и конкретни координати.




Кога пресметуваме векторски производ, ни се потребни вектори. Но на кој начин ќе ги добиеме координатите на некој вектор, за кој ги знаеме координатите на почетната и крајната точка од векторот?

Значи, ако почетокот на векторот ***a*** е точката со координати ***(x1,y1)***, и крајот на векторот *a* е точката со координати ***(x2,y2)***, тогаш векторот ***a*** ќе има координати ***(x2-x1,y2-y1)***. Кога велиме дека векторот ***a*** има координати ***(x2-x1,y2-y1)***, значи дека векторот има почеток во ***(0,0)*** и крај во ***(x2-x1,y2-y1)***. Векторот е наполно одреден со неговата должина, правец и насока, а не со неговата почетна односно крајна точка.




Значи за дадените координати од контролните точки ќе добиеме:

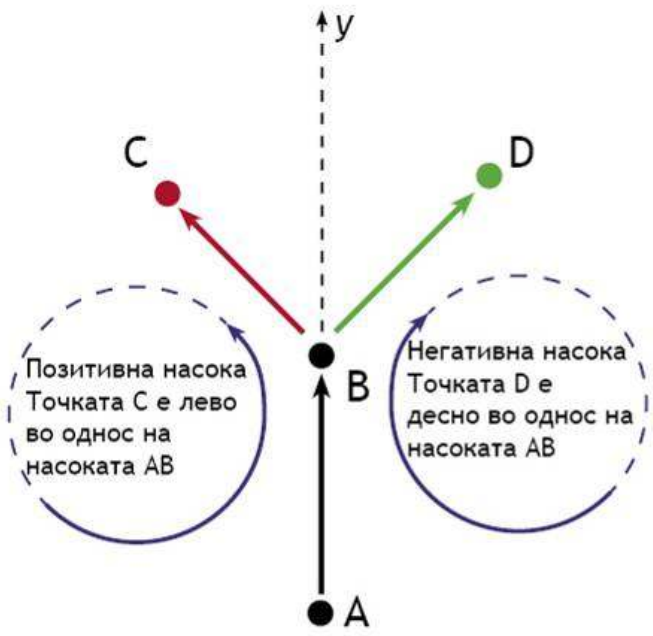
- Векторот ***AB*** ќе има координати (0-0,0-(-2)) = (0,2)
- Векторот ***BC*** ќе има координати (-1-0,1-0) = (-1,1)
- Векторот ***BD*** ќе има координати (1-0,1-0) = (1,1)

Но бидејќи работиме во рамнината ***Oxy*** и бидејќи сакаме да го примениме векторскиот производ, на овие вектори ќе им додадеме и трета координата која е еднаква на 0.

- Векторот ***AB*** ќе има координати (0-0,0-(-2)) = (0,2,**0**)
- Векторот ***BC*** ќе има координати (-1-0,1-0) = (-1,1,**0**)
- Векторот ***BD*** ќе има координати (1-0,1-0) = (1,1,**0**)

$$\overrightarrow{AB} \times \overrightarrow{BC} = (0,2,0) \times (-1,1,0) = (0,0,2)$$
$$\overrightarrow{AB} \times \overrightarrow{BD} = (0,2,0) \times (1,1,0) = (0,0,-2)$$

Забележуваме дека првиот векторскиот производ меѓу ***AB*** и ***BC***, како резултат дава вектор, чија трета координата е позитивна. Ова значи дека точката ***C*** се наоѓа лево во однос на векторот (насоката) ***AB***. Вториот векторскиот производ меѓу ***AB*** и ***BD***, како резултат дава вектор, чија трета координата е негативна. Ова значи дека точката ***D*** се наоѓа десно во однос на векторот (насоката) ***AB***.




### Задача 1.

Дадени се две отсечки преку координати на нивни крајни точки ( ***(x1,y1)*** и ***(x2,y2)*** за првата отсечка, и ***(x3,y3)*** и ***(x4,y4)*** за втората отсечка). Да се напише програма која ќе каже дали отсечките се сечат(имаат пресечна точка) или не.

### Решение:

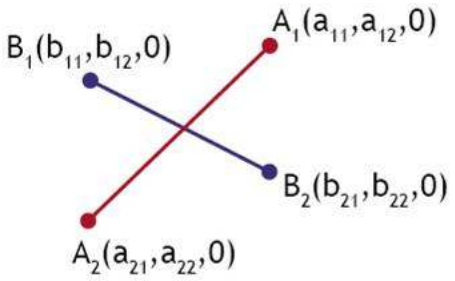
Идејата за решавање на задачата со векторски производ е следнава:

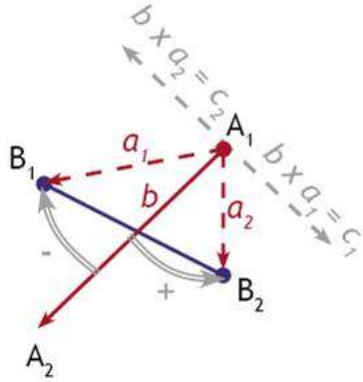
Значи имаме две отсечки; едната со краеви ***A1*** и ***A2***, а другата со краеви ***B1*** и ***B2***. Од едната отсечка со краеви ***A1=($a_{11}$, $a_{12}$, 0)*** и ***A2=($a_{21}$, $a_{22}$, 0)*** создаваме вектор ***b*** со координати ***($a_{21}$-$a_{11}$, $a_{22}$-$a_{12}$, 0)***. Потоа од темиња ***A1*** и ***B1*** на истиот начин правиме

вектор $a_1$ кој ќе има координати $(b_{11}-a_{11}, b_{12}-a_{12}, 0)$. Од темињата ***A1*** и ***B2*** се добива вектор $a_2 =( b_{21} - a_{11}, b_{22} - a_{12}, 0)$.

Потоа проверуваме дали ***b*** x $a_1$ како вектор има иста насока со векторот ***b*** x $a_2$. Тоа се проверува на тој начин што, откако ќе се пресмета векторскиот производ на два вектора изразени во координати, се земама само третата координата. Ако третата координата од векторот ***b*** x $a_1$ има ист знак[1] како и третата координата на ***b*** x $a_2$, тогаш следува дека точките ***B1*** и ***B2*** лежат на иста страна од отсечката(правата) определена со точките ***A1*** и ***A2***. Ако овие две вредности имаат спротивен знак, тогаш точките ***B1*** и ***B2*** лежат на различна страна од отсечката(правата) определена со точките ***A1*** и ***A2***.

Истата проверка со двете отсечки се прави повторно само што во овој случај отсечките си ги менуваат улогите. Со други зборови, проверуваме дали крајните точки на едната отсечка лежат на различна страна од правата која е опредлена со крајните точки од другата отсечка и обратно.



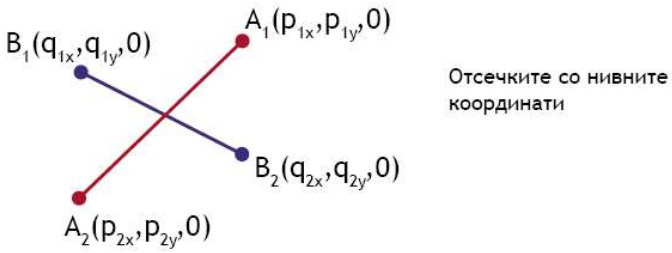



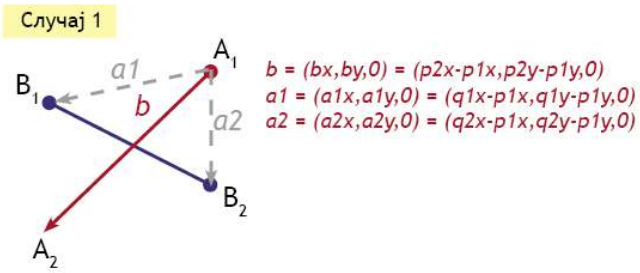

Едно решение е понудено со кодот што следува, но веднаш после ова решение следува код, кој е среден и далеку поразбирлив:

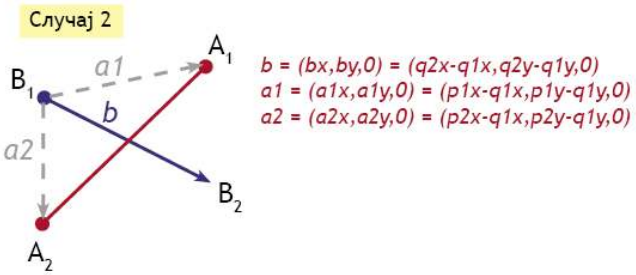
```
program otsecki;
var
      p1x,p1y,p2x,p2y:integer;
      q1x,q1y,q2x,q2y:integer;
      bx,by,a1x,a1y,a2x,a2y:integer;
      se_secat:boolean;
begin
      writeln('Vnesete gi koordinatite na tockite od otseckata A');
      readln(p1x,p1y,p2x,p2y);
      writeln('Vnesete gi koordinatite na tockite od otseckata B');
```

[1] Ист знак...= се мисли на тоа дека по вредност и двете се или поголеми од нула, или помали од нула.

```
        readln(q1x,q1y,q2x,q2y);
        se_secat:=true;
        {slucaj 1}
        bx:=p2x-p1x;
        by:=p2y-p1y;
        a1x:=q1x-p1x;
        a1y:=q1y-p1y;
        a2x:=q2x-p1x;
        a2y:=q2y-p1y;
        if not((bx*a1y-by*a1x)*(bx*a2y-by*a2x)<=0) then se_secat:=false;

        {slucaj 2}
        bx:=q2x-q1x;
        by:=q2y-q1y;
        a1x:=p1x-q1x;
        a1y:=p1y-q1y;
        a2x:=p2x-q1x;
        a2y:=p2y-q1y;
        if not((bx*a1y-by*a1x)*(bx*a2y-by*a2x)<=0) then se_secat:=false;

        if se_secat = true
              then
                    writeln('Otseckite se secat !')
              else
                    writeln('Otseckite ne se secat !');
        readln;
end.
```



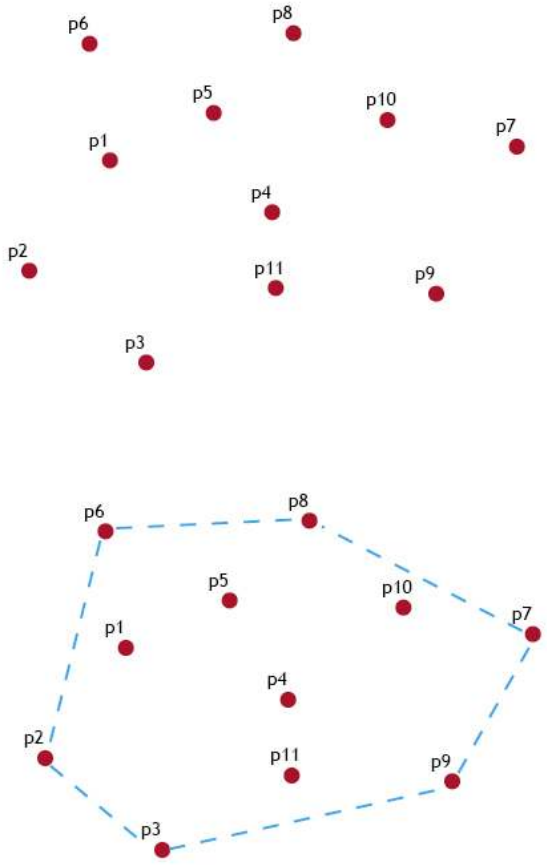



```
program otsecki_sredena;
type
      tocka=record
                  x:integer;
                  y:integer;
            end;
      otsecka=record
                    toc1:tocka;
                    toc2:tocka;
              end;
var
      A,B:otsecka;
      se_secat:boolean;

{funkcija za proverka dali tockite K i M se od razlicna strana na otseckata
P}
function razl_str(P:otsecka;K:tocka;M:tocka):boolean;{funkcija za proverka
na toa dali tockite K i M se na razlicni strani od otseckata P}
var
      vek_proiz1,vek_proiz2:integer;
      px,py,pkx,pky,pmx,pmy:integer;
begin
      {vektor_0 od tockite na otseckata P}
      px:=P.toc2.x-P.toc1.x;
      py:=P.toc2.y-P.toc1.y;

      {vektor_1 od poc.tocka na P i tockata K}
      pkx:=K.x-P.toc1.x;
      pky:=K.y-P.toc1.y;

      {vektor_2 od poc.tocka na P i tockata M}
      pmx:=M.x-P.toc1.x;
      pmy:=M.y-P.toc1.y;

      {vektorski proizvod megu vektor_0 i vektor_1}
      vek_proiz1:=px*pky-py*pkx;

      {vektorski proizvod megu vektor_0 i vektor_2}
      vek_proiz2:=px*pmy-py*pmx;

      razl_str:=(vek_proiz1*vek_proiz2 <= 0);
end;

begin{glavna programa}
      writeln('Vnesete gi koordinatite na tockite od otseckata A');
      readln(A.toc1.x,A.toc1.y,A.toc2.x,A.toc2.y);
      writeln('Vnesete gi koordinatite na tockite od otseckata B');
      readln(B.toc1.x,B.toc1.y,B.toc2.x,B.toc2.y);

      se_secat:=razl_str(A,B.toc1,B.toc2) and razl_str(B,A.toc1,A.toc2);
      if se_secat = true
            then
                  writeln('Otseckite se secat !')
            else
                  writeln('Otseckite ne se secat !');
      readln;
end.
```

Друг проблем кој може да се реши со примена на векторскиот производ е барањето на конвексен многуаголник.

**Задача 2.**

Нека се дадени n точки во една ***Oxy*** рамнина, зададени како подредени парови $(x_i, y_i), i = 1...n$, Сакаме да најдеме конвексен многуаголник чии темиња ќе бидат некои од веќе спомнатите точки, а сите останати точки ќе лежат во внатрешноста на овој многуаголник.




**Решение:**

За да го решиме овој проблем, ќе искористиме една варијација на алгоритамот на Грахам, за наоѓање на темиња на конвексниот многуаголник.

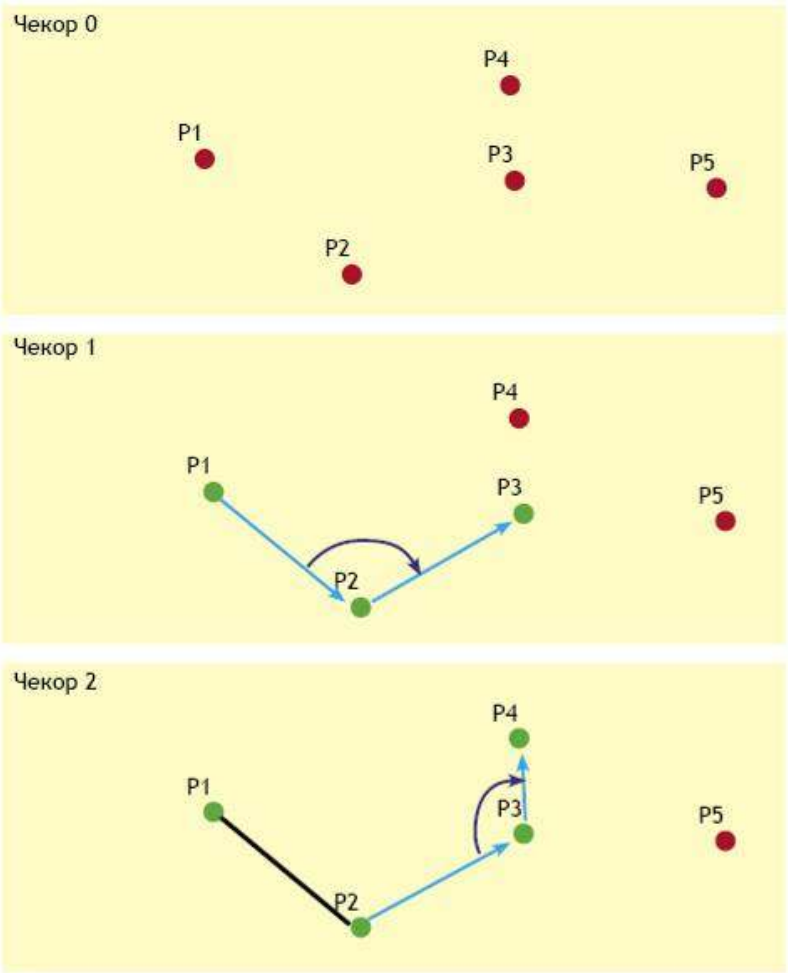
Која е суштината на овој алгоритам?

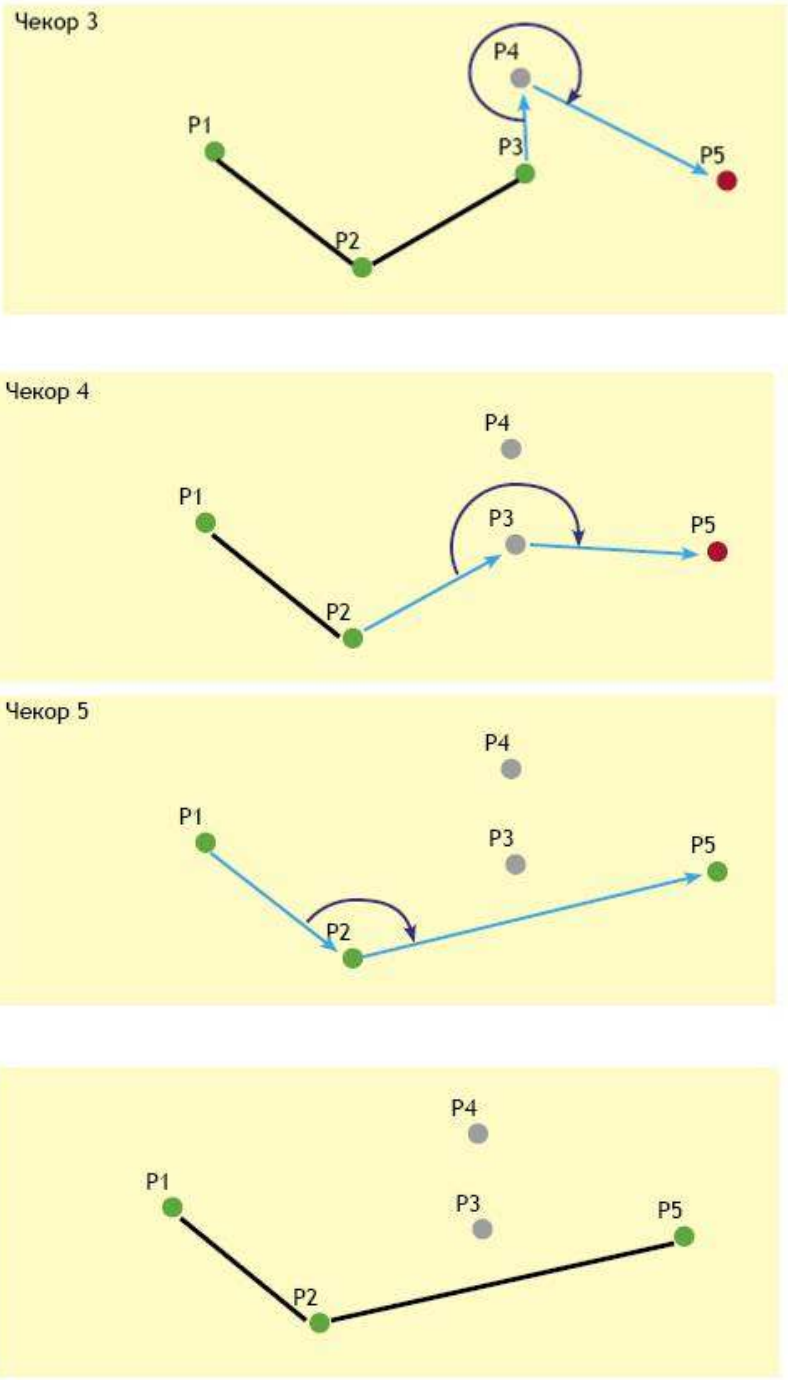
Од претходната слика забележуваме дека точката ***p9*** лежи на десната страна од насоката определена со векторот ***p2p3***. Точката ***p7*** лежи на десната страна од насоката определена со векторот ***p3p9*** итн.

Да разгледаме еден пример:

Дадени се 5 темиња кои се подредени во природен редослед ***(Чекор 0)***. Нека првите две темиња ***P1*** и ***P2*** се темиња на многуаголникот. Затоа како следна точка ја

испитуваме ***P3***. Ги формираме векторите ***P1P2*** и ***P2P3***. Забележуваме дека нивниот векторски производ дава вектор со позитивна трета координата односно точката ***P3*** се наоѓа на десната страна од насоката определена со векторот ***P1P2***. Затоа точката ***P3*** ја означуваме како теме на многуаголникот.***(Чекор 1)*** Следна точка што ја испитуваме е точката ***P4***. Затоа ги формираме векторите ***P2P3*** и ***P3P4***, така што се доаѓа до истиот заклучок како во *Чекор 1*, а тоа е дека точката ***P4*** лежи на десната страна од насоката определена со векторот ***P2P3***. Затоа точката ***P4*** ја означуваме како теме на многуаголникот.***(Чекор 2)*** Следна точка на испитување е точката ***P5***. Ги формираме векторите ***P3P4*** и ***P4P5***. Но, во овој случај векторскиот производ на овие два вектора дава вектор со негативна трета координата, што значи дека точката ***P5*** лежи на левата страна од насоката определена со векторот ***P3P4***. Тоа е знак дека точката ***P4*** не претставува теме на многуаголникот. ***(Чекор 3)***. Во следниот чекор, се враќаме точка наназад, па затоа ги формираме векторите ***P2P3*** и ***P3P5*** и забележуваме дека повторно точката ***P5*** е на левата страна од насоката определена со векторот ***P2P3***. Тоа е знак дека точката ***P3*** не претставува теме на многуаголникот. ***(Чекор 4)*** Се враќаме точка наназад. Ги формираме векторите ***P1P2*** и ***P2P5***. Забележуваме дека во овој случај ***P5*** лежи на десната страна од насоката определена со векторот ***P1P2***. Затоа точката ***P5*** ја означуваме како теме на многуаголникот. ***(Чекор 5)***. Значи дека само точките ***P1***, ***P2*** и ***P5*** се темиња на многуаголникот после извршувањето на *Чекор 5*.

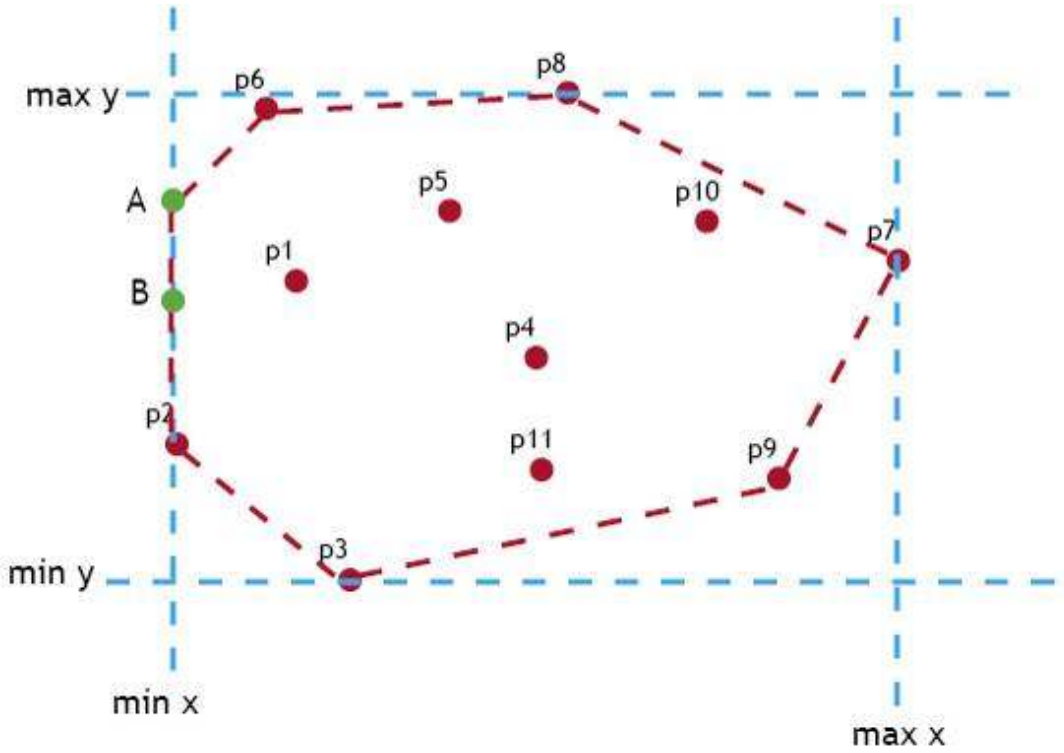





Најпрво во алгоритамот ги читаме ***n***-те точки. Овие точки имаат произволно подредување. Но ние треба да најдеме друго подредување на овие точки со цел да го олесниме пребарувањето на темињата на многуаголникот. Како почетна точка ја бираме точката со најмала ***x***-координата. Ако има повеќе точки со најмала ***x***-координата, од нив ја бираме онаа која има најмала ***y***-координата. Оваа точка сигурно е теме на многуаголникот.

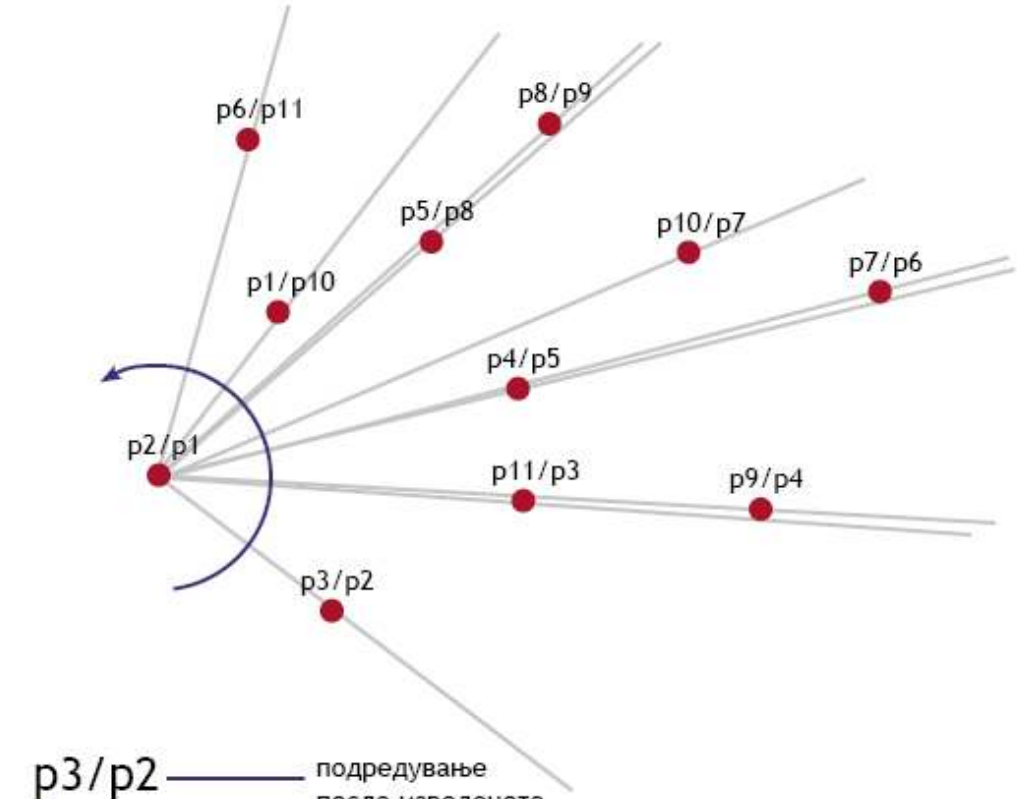
Навистина, целиот многуаголник можеме да го сместиме во правоаголник кој е определен со точките ***(minX, minY), (minX, maxY), (maxX, minY)*** и ***(maxX, maxY)***, каде ***minX(maxX)*** е минимум(максимум) од ***x***-координатите на точките, а ***minY(maxY)*** е минимум(максимум) од ***y***-координатите на точките. Ова значи дека точкта со најмала ***x***-координата ќе биде теме на многуаголникот. Во случај да има

повеќе такви точки, односно точки со најмала ***x***-координата, тогаш како почетна точка ќе ја земеме онаа со најмала ***y***-координата. Од сликата што следува се забележува дека иако точките ***A***,***B*** и ***p2*** имаат најмала ***x***-координата, сепак, како почетна ја бираме ***p2***. Точките ***p2*** и ***A*** се темиња на многуаголникот, но не и точката ***B***.



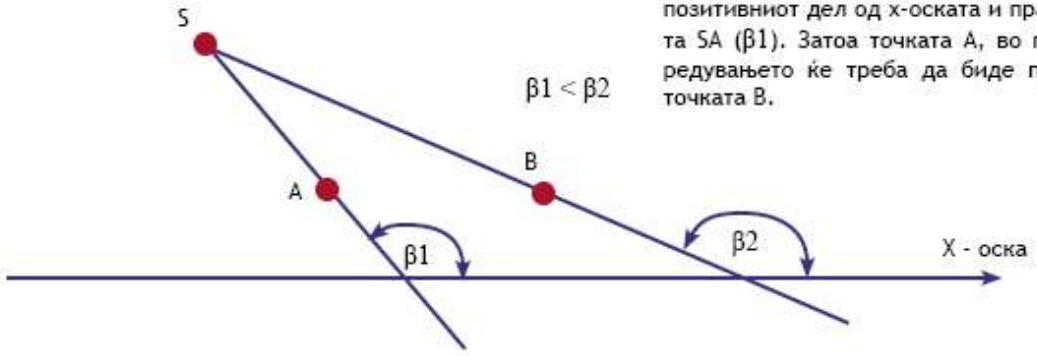

Откако ќе го определиме почетното теме, останатите темиња ќе ги сортираме во насока обратна од насоката на движење на стрелките на часовникот, во однос на центар определен со почетното теме што го определивме во претходната фаза. Од сликата што следува ***p2*** е почетната точка. Од оваа точка повлекуваме (полу)прави кон секоја останата точка. Потоа овие прави(односно точки) ги сортираме во насока обратна од насоката на движење на стрелките на часовникот.

Но за да го изведеме ваквото сортирање, ќе мора кај секоја од овие прави да измериме некаков параметар. Тоа ќе биде аголот(тангенсот од аголот) што се зафаќа меѓу позитивниот дел од ***x***-оската и правата определена со почетното теме и точката на разгледување.



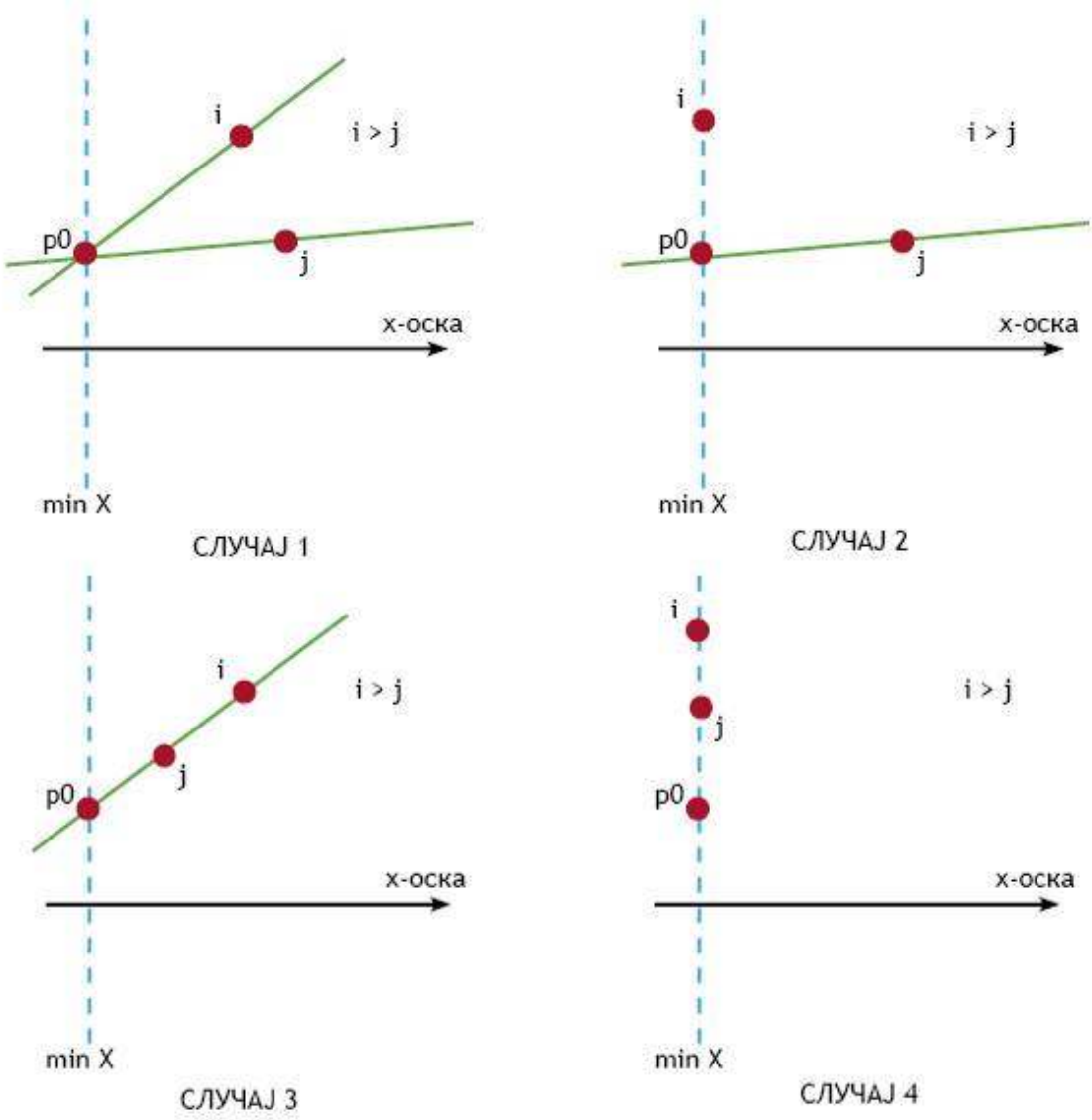

p3/p2 — подредување после изведеното сортирање на точките

подредување при читање на точките




Аголот што го формираат позитивниот дел од x-оската и правата SB (β2) е поголем од аголот што го формираат позитивниот дел од x-оската и правата SA (β1). Затоа точката A, во подредувањето ќе треба да биде пред точката B.

При сортирањето изведено во кодот кој што ќе следи подоцна во обзир се земени следниве четири случаи, прикажани на сликата подолу:




На претходната слика се прикажани точку четирите случаи при кои се извршува процедурата swap за замена на редоследот на i –тата и j-тата точка,. Ваквите четири случаи се јавуваат заради пресметката на тангенсот кој за агол од 180 степени дава вредност бесконечност, а во компјутерот тоа не е возможно да се претстави.

Пред да дадеме конкретна реализација на алгоритмот за одредување на темињата на конвексниот многуаголник, наброени во насока обратна од насоката на движење на стрелките на часовникот, ќе кажеме дека истата задача може да се искористи и при одредување на пар точки кои во рамки на множество точки се најодалечени една од друга[1]. Имено, овој пар точки ќе претставуваат темиња на конвексниот многуаголник кој ќе го најдеме со реализираниот алгоритам што следува подоле.

Исто така, вреди да се спомене дека овој алгоритам може да се реализира со примена на структурата стек[1], која е објаснета во документото ***stek_red.doc***, кој можете да го најдете на ***algoritam.blog.com.mk***.

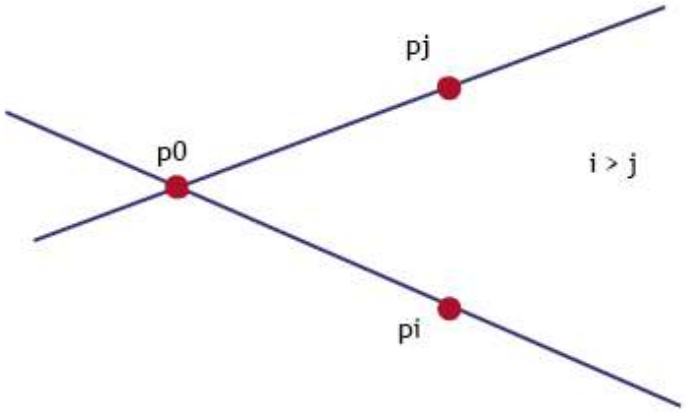
Кодот, кој ја реализира задачата поставена како проблем и кој вклучува користење на датотеки за читање на влезни податоци и запишување на излезните податоци е следниов:

```
program mnoguagolnik;{programski jazik Pascal}
const
     max_tocki=1000;

var
   x,y:array[1..max_tocki] of integer;
   vm:array[1..max_tocki] of boolean;
   pt,tt,sl:integer;
   n,i,j,min_index:integer;
   fi,fo:text;

procedure swap(i:integer;j:integer);{procedura za menuvanje na redsledot na
i-tata i j-tata tocka}
var
   pom:integer;
begin
     pom:=x[i];
     x[i]:=x[j];
     x[j]:=pom;
     pom:=y[i];
     y[i]:=y[j];
     y[j]:=pom;
end;

function rast(i:integer;j:integer):real;{funkcija za presmetuvanje na
rastojanieto megu i-tata i j-tata tocka}
begin
     rast:=sqrt((x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]));
end;

function k_koor(pt,tt,sl:integer):integer;{funkcija za presmetuvanje na
tretata koordinata od vektorskiot proizvod megu vektorot sostaven od pt-
tata i tt-tata tocka, i vektorot sostaven od tt-tata i sl-tata tocka}
begin
      k_koor:=((x[tt]-x[pt])*(y[sl]-y[tt])-(x[sl]-x[tt])*(y[tt]-y[pt]));
end;

function tanges(i,j:integer):real;{funkcija za presmetuvanje na tangensot
na agolot sto go zafaka pozitivniot del od x-oskata i pravata opredelena so
i-tata i j-tata tocka}
begin
     tanges:=(y[j]-y[i])/(x[j]-x[i]);
end;

procedure citaj;{delot od algoritamot za citanje i opredeluvanje pocetna
tocka-teme}
begin
     assign(fi,'vlez.in');
     assign(fo,'izlez.out');
     reset(fi);
     rewrite(fo);
     readln(fi,n);
     {gi citame tockite edna po edna}
```

```
     min_index:=1;{pretpostavuvame deka prvata tocka e tocka so najmala x-
koordinata}
     for i:=1 to n do
         begin
              readln(fi,x[i],y[i]);
              if x[min_index]>x[i]{ako i-tata tocka ima pomala x-koordinata
od pocetnata(min_index) tocka}
                  then
                      min_index:=i;{togas taa tocka stanuva pocetna tocka so
najmala x-koordinata}
              if (x[min_index]=x[i]) and (y[i]<y[min_index]){ako postoi
druga tocka, so ista x-koordinata
             kako pocetnata tocka, no pomala y-koordinata togas novata
tocka se zema za pocetna}
                  then
                      min_index:=i;
         end;
     swap(1,min_index);{ja zamenuvame pocetnata tocka so prvata tocka sto
sme ja procitale za taa da stane prva}
end;

procedure sortiraj;{procedura za sortiranje na ostanatite n-1 tocka;
algoritmot koj se primernuva e Buble Sort}
begin
     {sortiranje na ostanatite tocki spored agolot megu pozitivniot del na
x-oskata
     i pravata formirana od pocetnata tocka i tockata na razgleduvanje}
     for i:=2 to n-1 do
         for j:=i+1 to n do
             begin
                  if ((x[1]<>x[i]) and  (x[1]<>x[j]))
                     then
                       begin
                              if (tanges(1,i)>tanges(1,j)){1 slucaj}
                                 then
                                     swap(i,j)
                                 else
                                     if ((tanges(1,i)=tanges(1,j))and
(rast(1,i)>rast(1,j))){3 slucaj}
                                        then
                                            swap(i,j);
                           end;
                  {2 slucaj}
                  if((x[1]<>x[j]) and (x[i]=x[1]))
                                  then
                                swap(i,j);

                  {4 slucaj}
                  if((x[1]=x[j]) and (x[i]=x[1]) and (rast(1,i)>rast(1,j)))
                                  then
                                      swap(i,j);
                  {end 4 slucaj}

             end;
end;

procedure opredeli_mnoguagolnik;{Backtrek algoritam za opredeluvanje na
teminjata na mnoguagolnikot}
begin
     tt:=2;{tt - tekovna tocka na obrabotka}
     sl:=3;{sl - tocka-sledbenik}
```

```
     pt:=1;{pt - tocka - prethodnik}
     fillchar(vm,sizeof(vm),false);{vm nizata ja polnime so false}
     vm[tt]:=true;
     while vm[1]=false do
           begin
                if k_koor(pt,tt,sl)<0{slucaj na cekor 3 i cekor 4 od
slikite}
                   then
                       begin
                            vm[tt]:=false;{tt-tata tocka ne e teme od
mnoguagolnikot}
                            tt:=pt;
                            pt:=pt-1;
                            while (vm[pt]=false)and(pt>1) do pt:=pt-1;
                            {so ovoj ciklus se bara tocka (so vrakanje
nanazad)koja pripaga na mnoguagolnikot}
                       end
                   else{slucaj od cekor 1,2 i 5}
                       begin
                            vm[tt]:=true;{tt-tata tocka e teme od
mnoguagolnikot}
                            pt:=tt;
                            tt:=sl;
                            sl:=sl+1;
                            if sl>n then sl:=1;
                       end;
           end;
end;

procedure pecati_mnoguagolnik;{pecatenje na teminjata od mnoguagolnikot}
begin
      for i:=1 to n do
            if vm[i]=true
               then
                   writeln(fo,x[i],' ',y[i]);

      close(fi);
      close(fo);

end;

begin{glavna programa}
     citaj;
     sortiraj;
     opredeli_mnoguagolnik;
     pecati_mnoguagolnik;
end.
```

Наместо да се примени сортирањето што го објаснивме претходно, можно е да се примени друга споредба користејќи го векторскиот производ, а не тангенсот од аголот меѓу позитивниот дел од x-оската и соодветната права.




Во однос на насоката определена со векторот ***p0*** и ***i***-тата точка ***pi***, точката ***pj*** се наоѓа од левата страна, и знаеме дека во тековното(моменталното) подредување ***i***-тата точка се наоѓа после ***j***-тата точка( ***i>j*** што е прикажано на претходната слика), а всушност треба обратно: при поминување на точките во процедурата ***opredeli-_mnoguagolnik*** мора прво да ја поминеме ***i***-тата точка (***pi***), а потоа ***j***-тата точка (***pj***).

Значи ако ***j***-тата точка се наоѓа лево во однос на векторот определен со почетната точка и ***i***-тата точка, односно ако третата координата на векторскиот производ меѓу векторите ***p0pi*** и ***pipj*** е негативна, тогаш треба ***i***-тата и ***j***-тата точка треба да си ги заменат местата.

Ако третата координата на векторскиот производ даде вредност 0, тоа ќе значи дека точките ***p0***, ***pi***, ***pj*** се колинеарни. Во тој случај се проверува ***случајот 3***, што важи за претходниот алгоритам.

Се што кажавме претходно можеме да го искажеме преку следнава реализација во програмскиот јазик Паскал:

```
procedure sortiraj;{procedura za sortiranje na ostanatite n-1 tocka;
algoritmot koj se primernuva e Buble Sort}
begin
     {sortiranje na ostanatite tocki so koristenje na vektorskiot proizvod}
     for i:=2 to n-1 do
         for j:=i+1 to n do
             begin
                  if k_koor(0,i,j)<0
                        then
                              swap(i,j);
                  if (k_koor(0,i,j)=0)and(rast(1,i)>rast(1,j))
                        then
                              swap(i,j);
             end;
end;
```

Се забележува дека овој код е далеку пооптимален и поразбирлив, и затоа се препорачува да се примени.

Постои и друга можност за подобрување на алгоритмот, а тоа е во замената на алгоритмот Buble Sort со некој пооптимален(побрз), како што е Quick Sort.

# Користена литература

[1] Introduction to algorithms – Thomas H. Corman, Charles E. Leiseron, Ronald L. Rivest, Clifford Stein; MIT Press, 2001.

[2] Takmicene iz informatike 1995 – 1998 – Dragan Urosevic; Krug, 1998.