Temat: Geometria obliczeniowa, część I.
Podstawowe algorytmy geometryczne.
Problem sprawdzania przynależności punktu do wielokąta.
Problem otoczki wypukłej – algorytmy Grahama, i Jarvisa.

1. **Oznaczenia**

Punkty - małe litery: p, q, r, s itd.
Współrzędne punktu p na płaszczyźnie - $(x(p), y(p))$
Odcinek o początku i końcu odpowiednio w punktach p i q: $p - q$
Wektor o początku i końcu odpowiednio w punktach p i q: $p \rightarrow q$
Prosta zawierająca punkty p *i* q: pq
Półprosta zaczynająca się w punkcie p i zawierająca punkt q: $\bullet pq$

2. **Operacje elementarne**

Operacje arytmetyczne: dodawanie, odejmowanie,
mnożenie, dzielenie,
pierwiastkowanie, itp

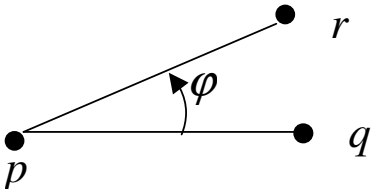
3. **Założenia i uwagi**

- rozważane są obiekty geometryczne na płaszczyźnie w kartezjańskim układzie współrzędnych
- algorytmy powinny realizować jak najmniej operacji powodujących przybliżenia

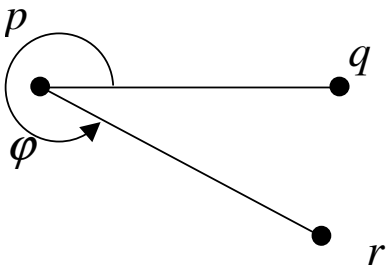
4. **Podstawowe algorytmy geometryczne**

a) Algorytm sprawdzania, po której stronie prostej leży punkt

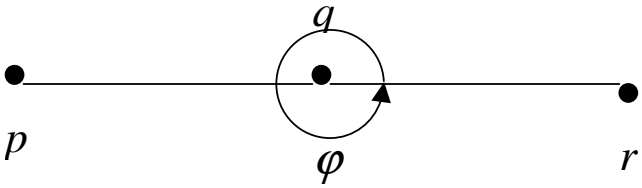
WP: Trzy punkty: $p = (x, y)$, $q = (z, t)$, $r = (u, v)$
WK: Odpowiedź na pytanie: Po której stronie prostej pq leży punkt r



Punkt r leży po lewej stronie prostej pq



Punkt r leży po prawej stronie prostej pq



Punkty p, q i r są współliniowe

Algorytm

Obliczamy wartość wyznacznika $\det(p,q,r)$, którego znak jest równy znakowi sinusa kąta nachylenia wektora $p \to r$ do wektora $p \to q$.

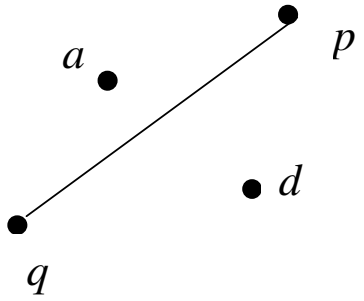
$$\det(p,q,r) = \begin{vmatrix} x & y & 1 \\ z & t & 1 \\ u & v & 1 \end{vmatrix}$$

- Jeżeli $\det(p,q,r) > 0$ to $\sin\varphi > 0$ i wówczas punkt *r* leży po lewej stronie prostej *pq*.
- Jeżeli $\det(p,q,r) < 0$ to $\sin\varphi < 0$ i wówczas punkt *r* leży po prawej stronie prostej *pq*.
- Jeżeli $\det(p,q,r) = 0$ to $\sin\varphi = 0$ i wówczas punkty *p*, *q* i *r* są współliniowe.

b) Algorytm sprawdzania, czy dwa dane punkty leżą po tej samej stronie prostej

WP: Cztery punkty: $p = (x, y)$, $q = (z, t)$, $a = (b, c)$, $d = (e, f)$

WK: Odpowiedź na pytanie , czy punkty a i b leżą po tej samej stronie prostej pq.



Algorytm

Przypomnijmy funkcję znaku liczby:

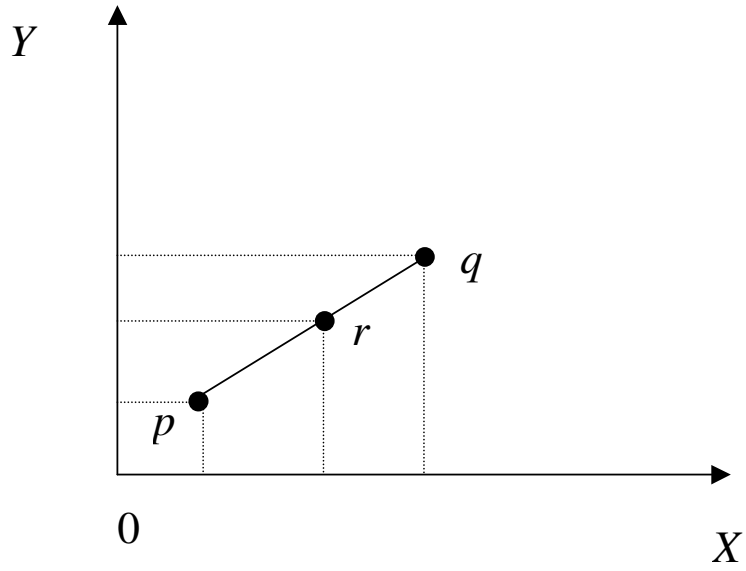
$$\operatorname{sgn}(x) = \begin{cases} 1 \text{ gdy } x > 0 \\ 0 \text{ gdy } x = 0 \\ -1 \text{ gdy } x < 0 \end{cases}$$

Punkty a i d leżą po tej samej stronie prostej pq wówczas, gdy:

$$\operatorname{sgn}(\det(p, q, a)) = \operatorname{sgn}(\det(p, q, d))$$

c) Algorytm sprawdzania, czy punkt należy do odcinka

WP: Trzy punkty: $p = (x, y)$, $q = (z, t)$, $r = (u, v)$
WK: Odpowiedź na pytanie, czy punkt r należy do odcinka p - q.



Algorytm

Jeżeli punkt r należy do odcinka p-q, to rzuty prostokątne r na osie (OX i OY) "zawierają się" w rzutach prostokątnych odcinka p-q. Wynika stąd, że r należy do odcinka p-q wtedy i tylko wtedy gdy:

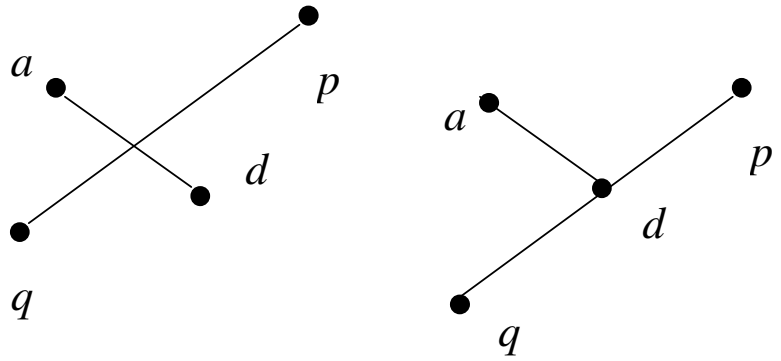
$x(p) \le x(r) \le x(q) \wedge \operatorname{sgn}(\det(p,q,r)) = 0$ przy założeniu, że $x(p) \le x(q)$

i

$y(p) \le y(r) \le y(q) \wedge \operatorname{sgn}(\det(p,q,r)) = 0$ przy założeniu, że $y(p) \le y(q)$

d) Algorytm sprawdzania, czy dwa odcinki się przecinają

WP: Cztery punkty: $p = (x, y)$, $q = (z, t)$, $a = (b, c)$, $d = (e, f)$ wyznaczające dwa odcinki: *p-q* i *a-d*.
WK: Odpowiedź na pytanie , czy odcinki *p-q* i *a-d* przecinają się.



Algorytm

Rozwiązanie opiera się na spostrzeżeniu, że dwa odcinki przecinają się wtedy i tylko wtedy, gdy punkty *p* i *q* leżą po przeciwnych stronach prostej *ad* i punkty *a* i *d* leżą po przeciwnych stronach prostej *pq* lub któryś z końców jednego z odcinków należy do drugiego.
Czyli:

$$(\operatorname{sgn}(\det(p,q,d)) \neq \operatorname{sgn}(\det(p,q,a)) \ \wedge \ \operatorname{sgn}(\det(a,d,p)) \neq \operatorname{sgn}(\det(a,d,q))) \ \vee$$
$$(\operatorname{sgn}(\det(p,q,a)) = 0 \ \wedge \ (x(p) \leq x(a) \leq x(q) \ \wedge \ y(p) \leq y(a) \leq y(q))) \ \vee$$
$$(\operatorname{sgn}(\det(p,q,d)) = 0 \ \wedge \ (x(p) \leq x(d) \leq x(q) \ \wedge \ y(p) \leq y(d) \leq y(q))) \ \vee$$
$$(\operatorname{sgn}(\det(a,d,p)) = 0 \ \wedge \ (x(a) \leq x(p) \leq x(d) \ \wedge \ y(a) \leq y(p) \leq y(d))) \vee$$
$$(\operatorname{sgn}(\det(a,d,q)) = 0 \ \wedge \ (x(a) \leq x(q) \leq x(d) \ \wedge \ y(a) \leq y(q) \leq y(d)))$$

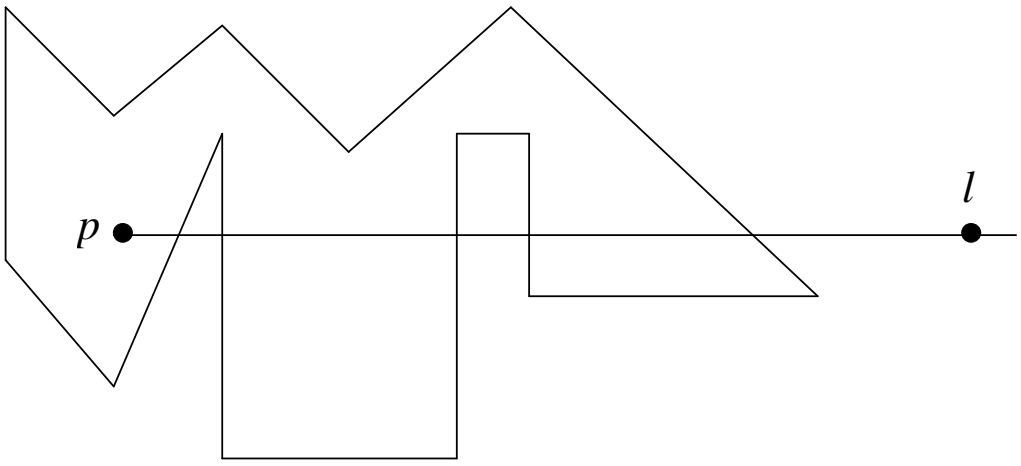
przy założeniu, że $x(p) \leq x(q)$ i $y(p) \leq y(q)$ w przypadku, gdy punkt *a* lub *d* należy do *p-q* i $x(a) \leq x(d)$ i $y(a) \leq x(d)$ w przypadku, gdy punkt *p* lub *q* należy do *a-d*

5. **Problem przynależności punktu do wielokąta**

WP: Dany jest ciąg punktów: $w_0, w_1, ..., w_{n-1}$ określający n - wierzchołkowy wielokąt W i taki, że dla każdego $i = 0,1,...,n-1$ odcinek $w_i - w_{i+1}$ jest bokiem wielokąta W (i+1 jest wyliczane modulo n).
Dany jest również punkt p.

WK: Odpowiedź na pytanie, czy punkt p należy do wielokąta W. Jeżeli punkt p leży na boku wielokąta, to stwierdzamy, że należy do wielokąta

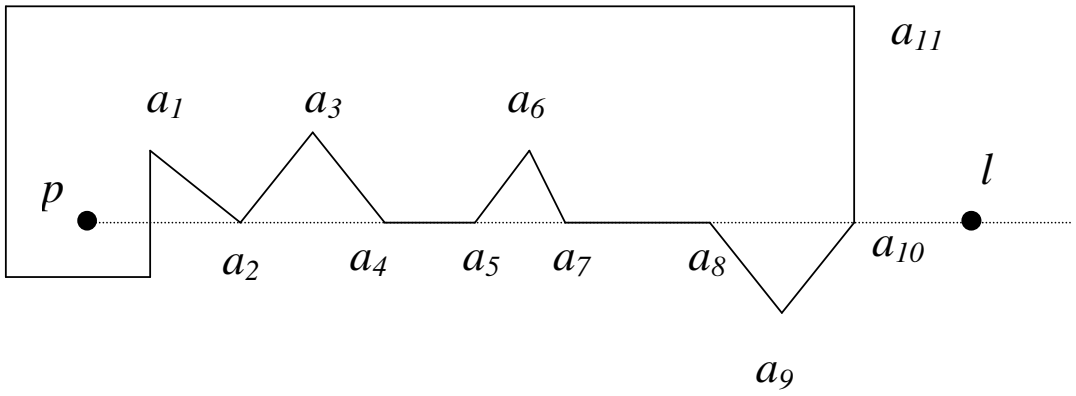
Algorytm rozwiązujący problem przynależności opiera się na zależności pomiędzy liczbą przecięć dowolnej półprostej o początku w punkcie p, a bokami wielokąta.



Punkt p należy do wielokąta W wtedy i tylko wtedy, gdy półprosta $.pl$ przecina boki wielokąta nieparzystą ilość razy. Zachodzą przy tym dwa przypadki szczególne:

- Półprosta $.pl$ przechodzi przez wierzchołek wielokąta.
- Półprosta $.pl$ zawiera bok wielokąta.

Uwaga !!!
Zanim zaczniemy wyznaczać liczbę punktów przecięcia półprostej z bokami wielokąta, sprawdzamy, czy punkt *p* nie zawiera się w którymś z boków wielokąta.



- Prosta *pl* zawiera bok a_4a_5 wielokąta. Niech a_3a_4 oraz a_5a_6 będą bokami sąsiadującymi z bokiem a_4a_5, a punkty a_3 i a_6 będą ich końcami tych boków. Jeżeli punkty a_3 i a_6 leżą po tej samej stronie prostej *pl*, to przyjmujemy, że liczba punktów przecięcia *pl* z bokami a_4a_5, a_3a_4 i a_5a_6 wynosi 0. W przeciwnym razie przyjmujemy, że liczba ta wynosi 1.

- Prosta *pl* zawiera wierzchołek a_2 wielokąta. Niech a_1a_2 oraz a_2a_3 będą bokami sąsiadującymi z wierzchołkiem a_2, a punkty a_1 i a_3 będą ich końcami tych boków. Jeżeli punkty a_1 i a_3 leżą po tej samej stronie prostej *pl*, to przyjmujemy, że liczba punktów przecięcia *pl* z bokami a_1a_2 i a_2a_3 wynosi 0. W przeciwnym razie przyjmujemy, że liczba ta wynosi 1.

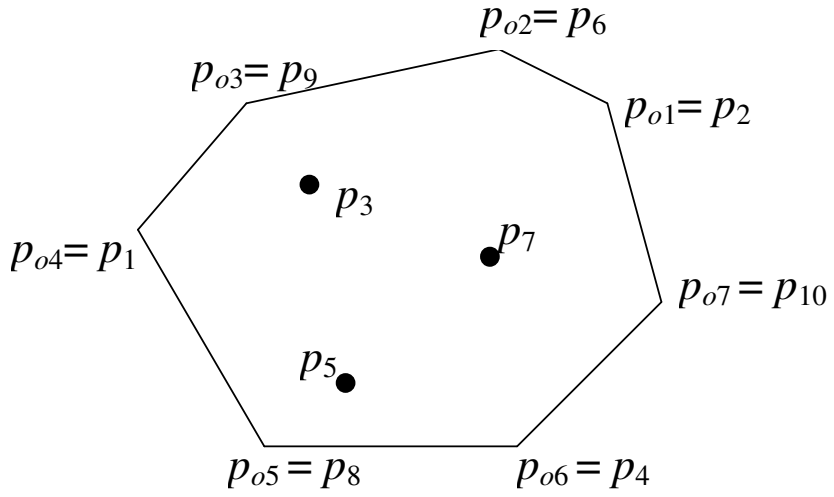
Ponieważ koszt obliczeń związanych z każdym bokiem i wierzchołkami *n* - kąta jest stały, złożoność sprawdzenia, czy punkt leży w jego wnętrzu, jest O(*n*).

6. **Otoczka wypukła**

Otoczką wypukłą dowolnego niepustego zbioru punktów *S* nazywamy najmniejszy zbiór wypukły zawierający *S*. Można udowodnić, że jeśli *S* jest zbiorem skończonym, to jego otoczka wypukła jest wielokątem wypukłym o wierzchołkach ze zbioru *S* (czasami zredukowanym do odcinka lub punktu).

WP: Skończony zbiór punktów $S = \{p_1, p_2, ..., p_n\}$.

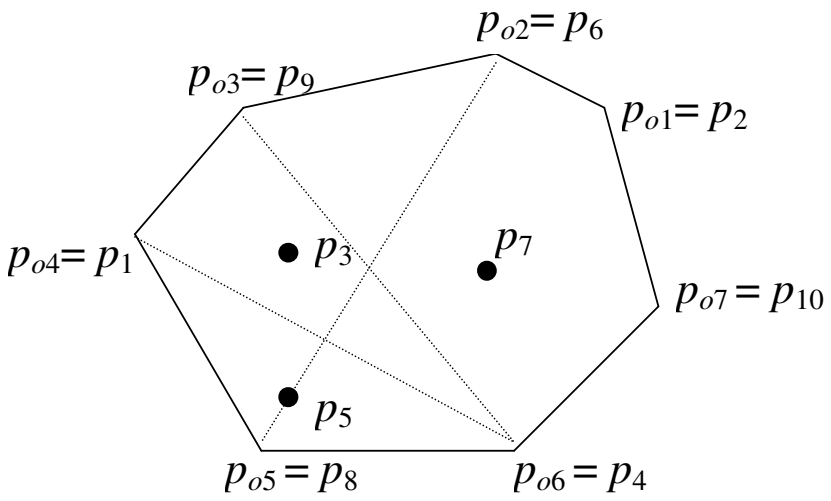
WK: Ciąg punktów $W : p_{o1}, p_{o2}, ..., p_{ow}$ taki, że $p_{oi} \in S$ (dla każdego $i = 1, 2, ..., w$), wyznaczający wierzchołki najmniejszego wielokąta wypukłego zawierającego wszystkie punkty z *S*. Kolejność punktów w ciągu *W* określa kolejność wierzchołków na obwodzie wielokąta w kierunku przeciwnym do ruchu wskazówek zegara.



a) Algorytm naiwny

Algorytm naiwny wyznaczania otoczki wypukłej opiera się na następującym spostrzeżeniu (*):

Punkt p nie jest punktem otoczki wypukłej zbioru S wtedy i tylko wtedy, gdy leży wewnątrz pewnego trójkąta o wierzchołkach z S, różnych od p, lub należy do odcinka łączącego dwa punkty z S, różne od p.



Punkt p_3 nie należy do otoczki wypukłej ponieważ leży wewnątrz trójkąta $p_9\, p_1\, p_4$.
Punkt p_5 nie należy do otoczki wypukłej ponieważ należy do odcinka p_6 - p_8 .

Algorytm naiwny

Krok 1: Sprawdzić, które punkty ze zbioru *S* należą do otoczki wypukłej stosując kryterium określonym w spostrzeżeniu (*).

Krok 2: Uporządkować znalezione punkty w kolejności ich występowania na obwodzie otoczki wypukłej.

Koszt czasowy algorytmu naiwnego

Rozmiar zadania: $n = |S|$ - liczba punktów zbioru *S*.
Operacja elementarna:

- operacja sprawdzania, czy punkt należy do trójkąta (odcinka) w Kroku 1,
- operacja porównania wykonana w trakcie sortowania w Kroku 2.

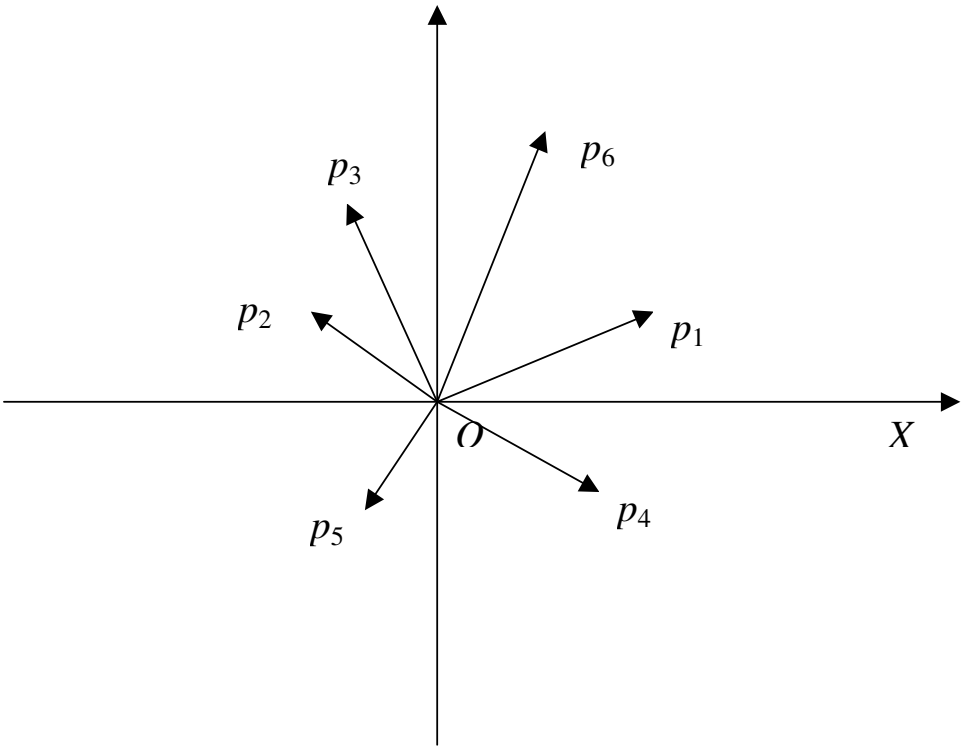
Koszt czasowy Kroku 1: Dla *n* punktów trzeba sprawdzić co najwyżej $\binom{n}{3}$ różnych trójkątów. Stąd koszt Kroku 1 jest $O(n^4)$.

Koszt czasowy Kroku 2: Jeżeli zastosujemy optymalną metodę sortowania to koszt czasowy Kroku 2 jest $O(n\log n)$. Całkowity koszt czasowy algorytmu naiwnego jest więc rzędu $O(n^4)$.
Efektywne algorytmy rozwiązujące problem otoczki wypukłej, algorytm Grahama i Jarvisa, mają zasadniczo niższy koszt.

Sortowanie zbioru punktów w celu wyznaczenia kolejności punktów otoczki po obwodzie wielokąta

WP: Zbiór $P = \{p_1, p_2, ..., p_n\}$ - punktów będących wierzchołkami wielokąta wypukłego *W*.

WK: Posortowany ciąg punktów zbioru *P* według niemalejącej wartości kąta nachylenia ich wektorów wodzących do osi *OX*.



Algorytm

1. Wyznaczamy centroid wielokąta *W*. Jest to punkt o współrzędnych:

$$\left(\frac{(x(p_1)+x(p_2)+...+x(p_n))}{n},\frac{(y(p_1)+y(p_2)+...+y(p_n))}{n}\right)$$

   Bez straty ogólności możemy przyjąć, że środek układu współrzędnych znajduje się w centroidzie wielokąta *W*.

2. W celu porównania kątów nachylenia wektorów wodzących sortowanych punktów do osi *OX* obliczamy wartości następującej funkcji *alfa*, określonej dla wszystkich punktów płaszczyzny różnych od punktu *O*.

$$alfa(p)=\begin{cases}\dfrac{y(p)}{d(p)}, & \text{gdy } x(p)>0\wedge y(p)\geq 0\\ 2-\dfrac{y(p)}{d(p)}, & \text{gdy } x(p)\leq 0\wedge y(p)>0\\ 2+\dfrac{|y(p)|}{d(p)}, & \text{gdy } x(p)<0\wedge y(p)\leq 0\\ 4-\dfrac{|y(p)|}{d(p)}, & \text{gdy } x(p)\geq 0\wedge y(p)<0\end{cases}$$

gdzie $d(p)=|x(p)|+|y(p)|$.

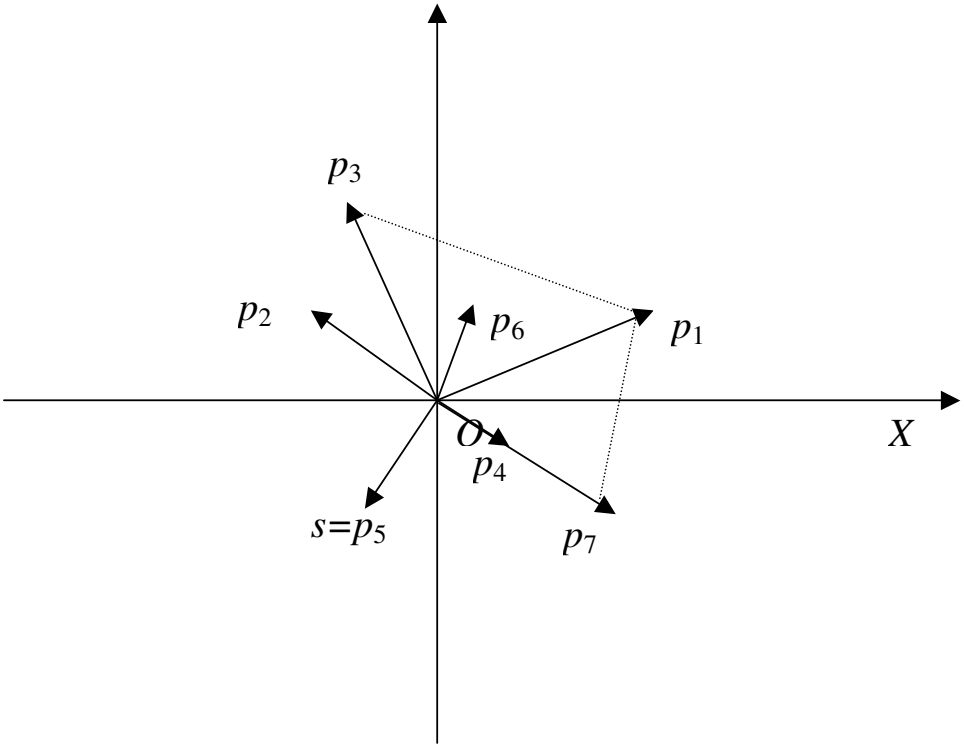
Można udowodnić, że kąt nachylenia wektora wodzącego punktu p_i jest mniejszy równy od kąta nachylenia wektora wodzącego punktu p_j wtedy i tylko wtedy, gdy:

$$alfa(p_i)\leq alfa(p_j).$$

Funkcja *alfa* umożliwia wyznaczenie kolejności wierzchołków na obwodzie wielokąta wypukłego w czasie $O(n\log n)$.

b) Algorytm Grahama

Algorytm Grahama opiera się na następującym spostrzeżeniu (**):
Każdy punkt nie będący wierzchołkiem otoczki wypukłej musi należeć do wnętrza trójkąta o wierzchołkach: *O* i pewne dwa kolejne wierzchołki otoczki (lub leży na jednym z boków takiego trójkąta).



Punkt p_6 nie należy do otoczki ponieważ leży we wnętrzu trójkąta $O\ p_1\ p_3$.
Punkt p_4 nie należy do otoczki ponieważ punkt ten leży na boku trójkąta $O\ p_7\ p_1$.

Algorytm

Krok 1:

Wybieramy dowolny punkt O leżący wewnątrz otoczki wypukłej, na przykład centroid. Umieszczamy w nim środek układu współrzędnych i obliczamy współrzędne punktów wejściowych w nowym układzie współrzędnych.

Krok 2:

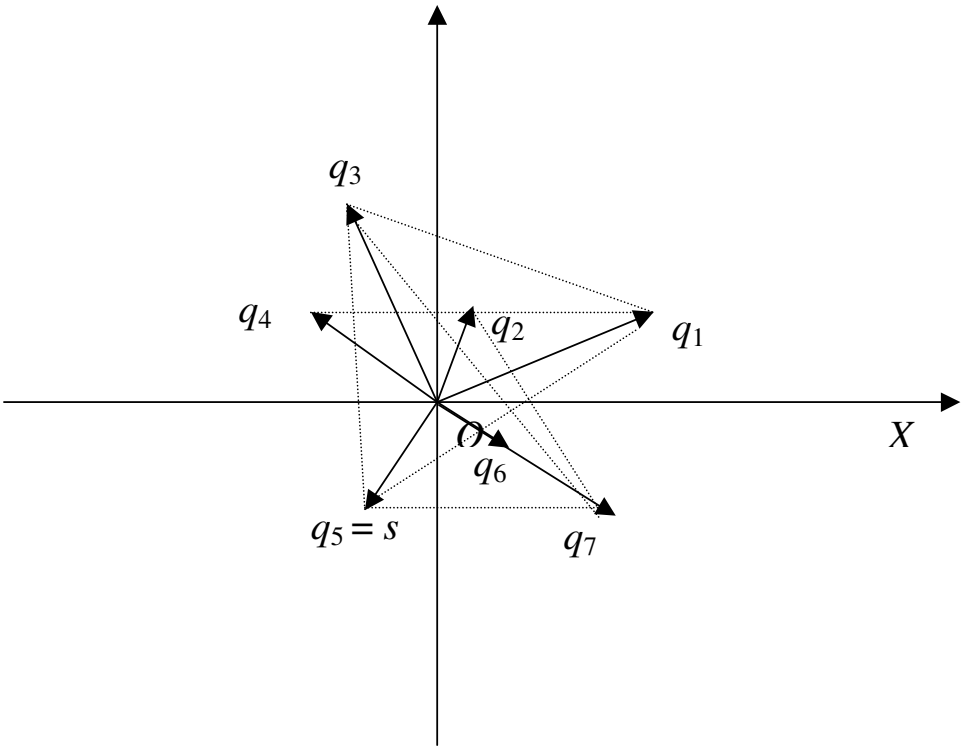
Porządkujemy punkty p_1, p_2, ... , p_n leksykograficznie względem współrzędnych biegunowych (α_i, r_i), gdzie α_i jest kątem nachylenia wektora wodzącego $O \to p_i$ do osi OX, a r_i jego długością. (Aby nie liczyć pierwiastków, w sortowaniu porównujemy $alfa(p_i)$ zamiast α_i oraz r_i^2 zamiast r_i).
Z uporządkowanych punktów tworzymy dwukierunkową listę cykliczną, w której dla każdego punktu p, p->$next$ jest następnym (cyklicznie) punktem w wyżej zdefiniowanym porządku, a p->$prev$ poprzednim.
Spośród punktów o najmniejszej współrzędnej y ustalamy punkt s z najmniejszą współrzędną x.

Krok 3:

Przeglądamy punkty na liście, usuwając te, które nie są wierzchołkami otoczki wypukłej. Po zakończeniu działania algorytmu lista będzie zawierała tylko wierzchołki otoczki wypukłej w kolejności ich występowania na obwodzie.
Listę przeglądamy, zaczynając od punktu *s* i kierując się w stronę przeciwną do ruchu wskazówek zegara (zgodnie ze wskaźnikiem *next*). W celu wyeliminowania zbędnych punktów zawsze sprawdzamy trzy kolejne punkty q_1, q_2 i q_3 z bieżącej listy. Jeśli punkt q_2 leży wewnątrz trójkąta Oq_1q_3 to usuwamy go z otoczki i przechodzimy do sprawdzania punktów q_0, q_1, q_3. W przeciwnym razie kolejną badaną trójką punktów są q_2, q_3, q_4. Przeglądanie kończymy z chwilą osiągnięcia wierzchołka startowego *s*.

Algorytm realizujący Krok 3 można zapisać następująco:

```
q=s;
while  (q->next != s)
   if  "q->next leży wewnątrz trójkąta O q q->next->next"
    {
         q->next=q->next->next;
         q->next->prev=q;
         if (q != s)  q=q->prev;
    }
   else q=q->next;
```



Koszt czasowy algorytmu Grahama

Na złożoność algorytmu Grahama decydujący wpływ ma Krok 2. Kroki 1 i 2 wykonywane są w czasie liniowym. Krok 2 można zrealizowany w czasie O($n$log$n$), stosując efektywną metodę sortowania (np. sortowanie przez scalanie, sortowanie przez połówkowe wstawianie czy sortowanie szybkie). Łatwo zauważyć, że zamiast sprawdzać, czy punkt *q->next* leży wewnątrz trójkąta *O q q->next->next* można testować, czy *q->next* leży po lewej stronie (lub należy do) wektora $q \rightarrow next -> next$.

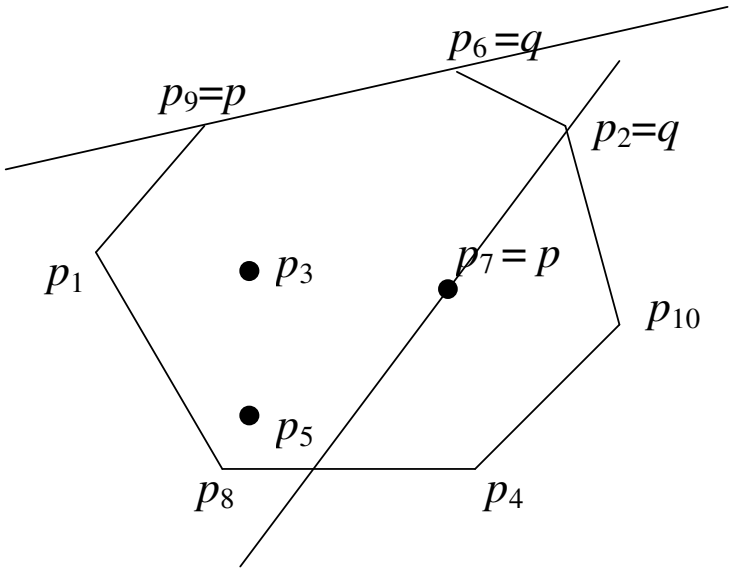
Złożoność algorytmu Grahama nie zależy od liczby punktów otoczki wypukłej. Następny algorytm rozwiązujący problem otoczki, algorytm Jarvisa, ma złożoność $O(kn)$, gdzie k jest liczbą punktów otoczki wypukłej w danym n elementowym zbiorze punktów.

c) Algorytm Jarvisa

Algorytm Jarvisa oparty jest na dwóch spostrzeżeniach:

1) Odcinek *p*-*q* o końcach ze zbioru *S* jest bokiem otoczki wypukłej wtedy i tylko wtedy, gdy wszystkie punkty z *S* należą do tej samej domkniętej półpłaszczyzny wyznaczonej przez prostą *pq*, a każdy punkt *S* leżący na tej prostej należy do odcinka *p*-*q*.

2) Jeśli odcinek *p*-*q* jest bokiem otoczki wypukłej, która nie jest zdegenerowana do odcinka lub punktu, to musi istnieć bok różny od *p*-*q*, zaczynający się w *q* (analogiczny warunek jest też spełniony dla punktu *p*).



Algorytm

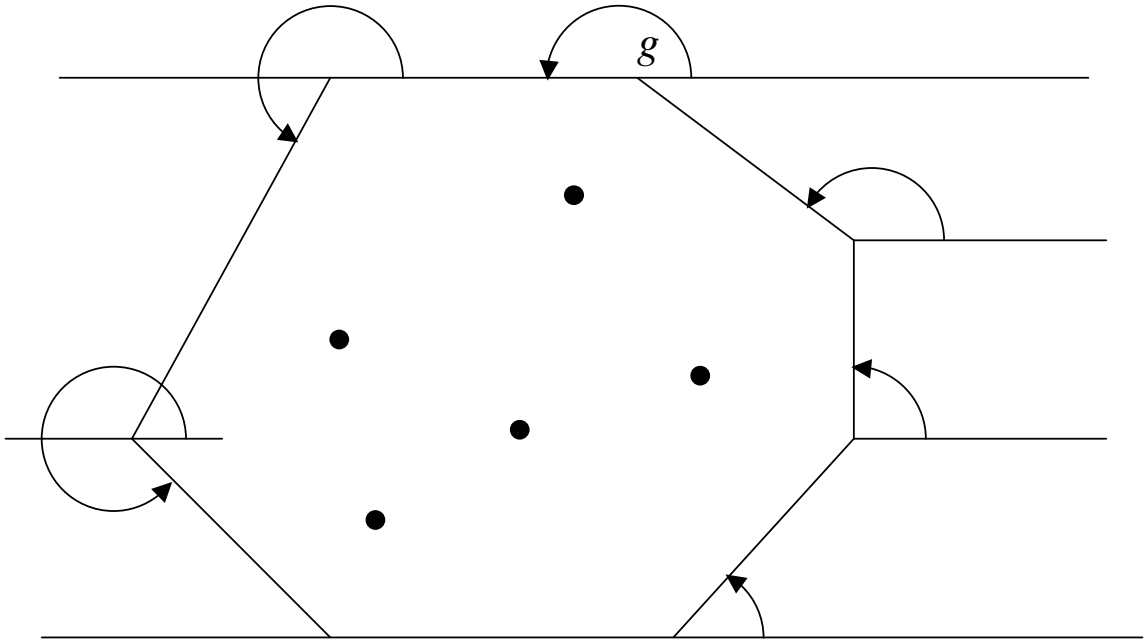
Krok 1:

Ustalamy punkt *d*, który ma najmniejszą współrzędną *x* spośród wszystkich punktów z najmniejszą współrzędną *y*. Ustalamy punkt *g*, który ma największą współrzędną *x* spośród wszystkich punktów z największą współrzędną *y*. Obydwa punkty *d* i *g* są wierzchołkami otoczki wypukłej.

Krok 2:

```
p=d;
while (p != g)
   {
     „umieszczamy środek układu współrzędnych
       w punkcie p”;
     r  = „punkt o najwiekszej odległości od p, wśród
          wszystkich punktów o najmniejszym kącie
          nachylenia wektora wodzącego do osi pX”;
      //   wszystkie punkty z S leżą w jednej półpłaszczyźnie
      //  wyznaczonej przez prostą pr. Odcinek p-r jest
      //  kolejnym bokiem otoczki  wypukłej
     p->next=r; r->prev=p; p=r;
   }
```

Krok 3:
Powtarzamy Krok 2, przyjmując, że za punkt startowy *g*, a punkt końcowy *d*. Rozważmy tylko punkty o kątach nachylenia promieni wodzących większych równych 180^0.



Złożoność czasowa algorytmu Jarvisa

Każda iteracja w Krokach 2 i 3 jest wykonywana w czasie $O(n)$. Ponieważ liczba iteracji jest równa liczbie wierzchołków otoczki, to koszt całkowity algorytmu Jarvisa wynosi $O(kn)$.

Algorytm ten jest szczególnie przydatny wtedy, gdy wiemy, że liczba punktów otoczki wypukłej jest niewielka w porównaniu z rozmiarem zbioru S (tj. ograniczona przez stałą niezależną od n).