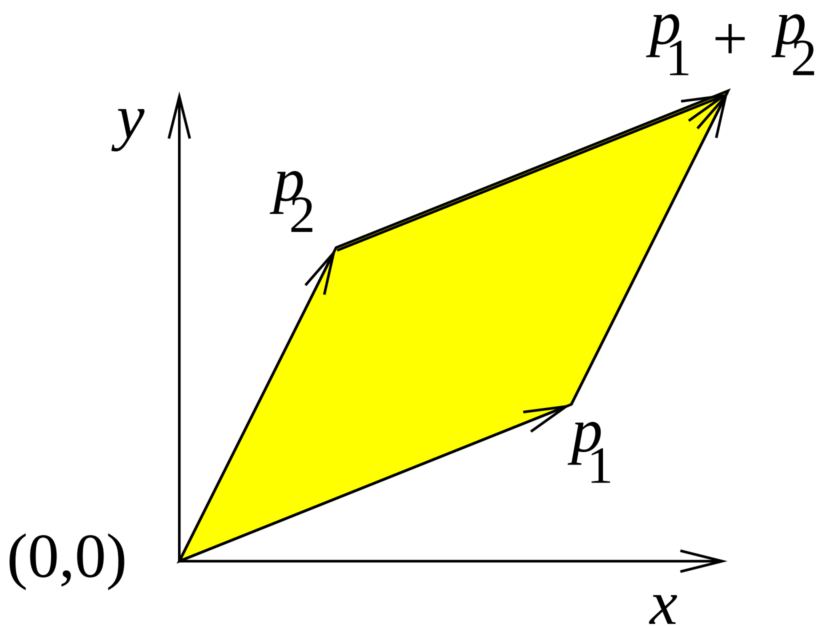
Geometria obliczeniowa

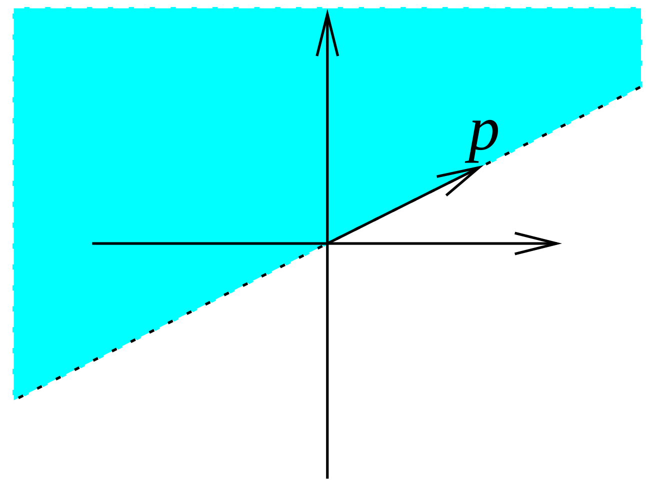
Definicje:

- *Wypukłą kombinacją* dwóch punktów $p_1 = (x_1, y_1)$ i $p_2 = (x_2, y_2)$ jest dowolny punkt $p_3 = (x_3, y_3)$, taki że $\exists_{0\leq\alpha\leq 1} \ (x_3 = \alpha x_1 + (1-\alpha)x_2 \wedge y_3 = \alpha y_1 + (1-\alpha)y_2)$.
- *Odcinek* $\overline{p_1 p_2}$ to zbiór wypukłych kombinacji p_1 i p_2.
- *Odcinek skierowany* $\overrightarrow{p_1 p_2}$ – gdy ma znaczenie kolejność końców.
- *Iloczyn wektorowy* $p_1 \times p_2$, gdzie $p_i = (x_i, y_i)$:

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

Iloczyn wektorowy





$p_1 \times p_2$ pole $((0,0), p_1, p_2, p_1 + p_2)$ ze znakiem '$+$' jeśli $0 < \angle p_1(0,0)p_2 < \pi$.

Aby sprawdzić czy $\overrightarrow{p_0 p_1}$ jest położony zgodnie z ruchem wskazówek zegara względem $\overrightarrow{p_0 p_2}$ sprawdzamy czy: $(p_1 - p_0) \times (p_2 - p_0) > 0$. (*idąc z p_0 przez p_2 do p_1 skręcamy w prawo*)

Jeśli $(p_1 - p_0) \times (p_2 - p_0) = 0$, to p_0, p_1, p_2 – *współliniowe*.

Przecięcia odcinków

Dane dwa odcinki: $\overline{p_1p_2}$ i $\overline{p_3p_4}$, gdzie $p_i = (x_i, y_i)$.
Chcemy sprawdzić czy $\overline{p_1p_2}$ i $\overline{p_3p_4}$ przecinają się.
Szybka eliminacja: Jeśli $\max\{x_1, x_2\} < \min\{x_3, x_4\}$ lub $\max\{x_3, x_4\} < \min\{x_1, x_2\}$ lub $\max\{y_1, y_2\} < \min\{y_3, y_4\}$ lub $\max\{y_3, y_4\} < \min\{y_1, y_2\}$, to $\overline{p_1p_2}$ i $\overline{p_3p_4}$ na pewno się nie przecinają.
W przeciwnym wypadku sprawdzamy czy $[(p_3 - p_1) \times (p_2 - p_1)] \cdot [(p_4 - p_1) \times (p_2 - p_1)] \leq 0$ (tzn. p_3 i p_4 nie są po tej samej stronie prostej p_1p_2) oraz $[(p_1 - p_3) \times (p_4 - p_3)] \cdot [(p_2 - p_3) \times (p_4 - p_3)] \leq 0$ (tzn. p_1 i p_2 nie są po tej samej stronie prostej p_3p_4).

Wykrywanie przecięć

Dane: n odcinków: $s_1, \ldots, s_n$. Chcemy sprawdzić czy istnieje para przecinających się odcinków. Najprostsze rozwiązanie – sprawdzanie wszystkich par – wymaga czasu $\Omega(n^2)$.

Założenia dodatkowe: Żaden z odcinków nie jest pionowy oraz żadne trzy odcinki nie przecinają się w jednym punkcie.

Metoda prostej zamiatającej: Przesuwamy pionową prostą (*miotłę*) po płaszczyźnie od lewej do prawej strony. W każdej chwili prosta jest przecinana przez pewien podzbiór T odcinków. *Zadrzeniem* jest zmiana T. Zdarzenie (dodanie lub usunięcie odcinka z T) może wystąpić tylko gdy miotła ma współrzędną x równą lewemu lub prawemu końcowi odcinka. Liczba zdarzeń: $2n$ (liczba końców odcinków).

Wykrywanie przecięć

s_1 i s_2 są *porównywalne w* x_0, jeśli oba przecinają miotłę o współrzędnej x_0. s_1 jest *powyżej* s_2 ($s_1 >_{x_0} s_2$) jeśli jego punkt przecięcia z miotłą s_1 jest powyżej przecięcia s_2 z miotłą.
Zakładamy, że mamy operacje:

- `Insert(`T`,`s`)` wstawia odcinek s do T
- `Delete(`T`,`s`)` usuwa s z T
- `Above(`T`,`s`)` zwraca odcinek bezpośrednio powyżej s w T
- `Below(`T`,`s`)` zwraca odcinek bezpośrednio poniżej s w T

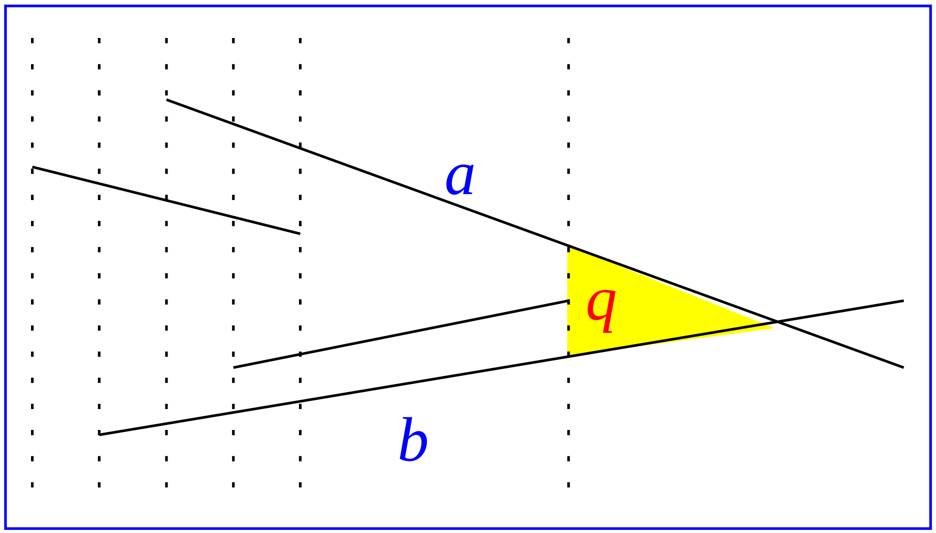
Jeśli T – drzewo czerwono-czarne, to każdą operację można wykonać w $O(\lg n)$. (Porównania kluczy można zastąpić sprawdzaniem iloczynów wektorowych (ćw.))

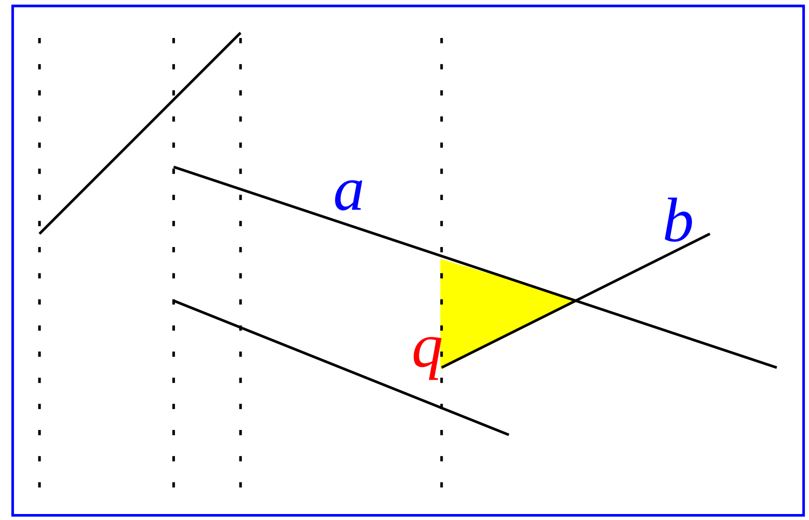
Any-Segments-Intersect

```
Any-Segments-Intersect(S)
T ← ∅
posortuj końce odcinków leksykograficznie
(współrzędna x – bardziej znacząca)
for each p na posortowanej liście końców do
    if p jest lewym końcem s then
        Insert(T,s)
        if Above(T,s) lub Below(T,s) przecina s
            then return TRUE
    if p jest prawym końcem s then
        if Above(T,s) przecina Below(T,s)
            then return TRUE
        Delete(T,s)
return FALSE
```

Czas: $O(n \lg n)$ (sortowanie $2n$ elementów i $O(\lg n)$ obsługa każdego zdarzenia)

analiza poprawności





Jeśli zwracana jest wartość $TRUE$, to istnienie przecięcia było sprawdzone w wierszu 6 lub 9.
Załóżmy nie wprost, że istnieje przecięcie ale zwrócone zostało $FALSE$. Niech p – pierwszy z lewej punkt przecięcia (remisy na korzyść mniejszej wsp. y). Niech a, b – odcinki przecinające się w p.

...

analiza poprawności

...

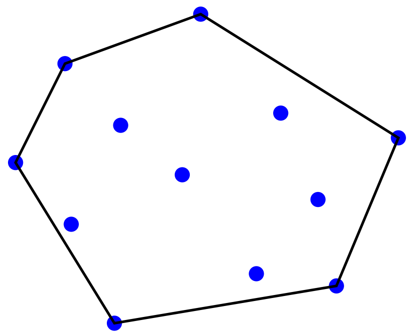
Na lewo od p nie ma żadnych przecięć, więc porządek elementów w T się nie zmienia. Żaden trzeci odcinek nie przecina się w p z a i b, więc istnieje prosta zamiatatająca z, w której a i b są sąsiadami w T. Niech q – koniec pewnego odcinka s – pierwsze zdarzenie, po którym a i b stają się sąsiadami w T. Są tylko dwie możliwości:

1. q jest lewym końcem jednego z odcinków a, b a drugi jest już w T (wtedy przecięcie wykryte w wierszach `4-7` i zwrócone $TRUE$).
2. a i b są już w T a q jest prawym końcem rozdzielającego je odcinka. (wtedy przecięcie wykryte w wierszach `8-11`i zwrócone $TRUE$).

Zatem nie ma możliwości zwrócenia $FALSE$ (wykonania wiersza `12`).

Wypukła otoczka

Wypukła otoczka zbioru punktów Q (oznaczana przez $CH(Q)$) to mnajmniejszy wielokąt wypukły P taki, że każdy punkt z Q leży we wnętrzu lub na brzegu P.



Algorytm Grahama

Wybieramy punkt p_0 – pierwszy z lewej z najniższych. Pozostałe sortujemy ze względu na kąt między odcinkiem łączącym z p_0 i poziomą półprostą o lewym końcu w p_0. (Spośród punktów o tej samej wartości kąta pozostawiamy jedynie najdalszy od p_0).

Niech $p_1, \ldots, p_m$ nieodrzucone punkty w posortowanej kolejności. Spacerujemy od p_0 przez $p_1, \ldots, p_m$ do p_0. Przy wchodzeniu do p_i tak długo wyrzucamy punkty ze szczytu S aż znajdzie się tam taki, w którym skręcamy w lewo aby dojść do p_i, po czym kładziemy p_i na stos.

W poniższej procedurze: `Top(`S`)` – element na szczycie S, `Next-To-Top(`S`)` – drugi od góry element w S.

Graham-Scan

Graham-Scan(Q)

wybierz p_0 *z* Q *o minimalnej wsp.* y *(pierwszy z lewej)*

niech $p_1 \ldots p_m$ *– pozostałe punkty posortowane ze wzgl. na wsp. kątową w biegunowym ukł. współrz. o początku* p_0 *po pozostawieniu jedynie najdalszych od* p_0 *puntów o tym samym kącie*

top$[S] \leftarrow 0$

Push(p_0,S)

Push(p_1,S)

Push(p_2,S)

for $i \leftarrow 3$ to m do

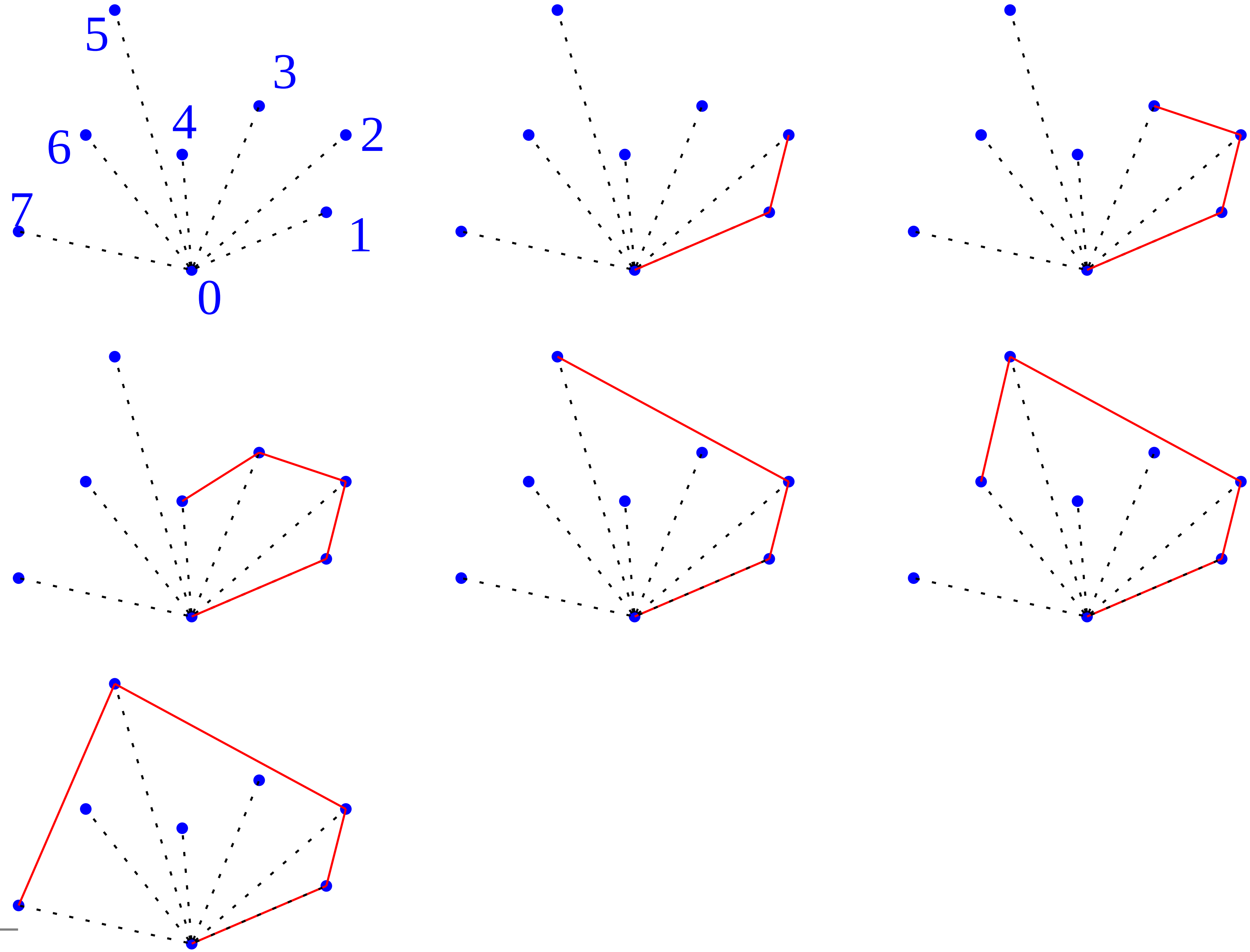
    while *nie skręcamy w lewo idąc z* Next-To-Top(S) *przez* Top(S) *do* p_i do

        Pop(S)

    Push(S,p_i)

return S

Graham-Scan

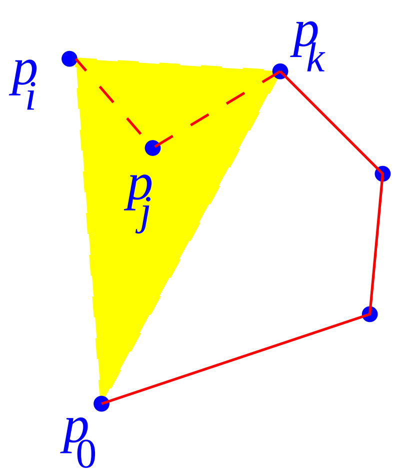


analiza poprawności

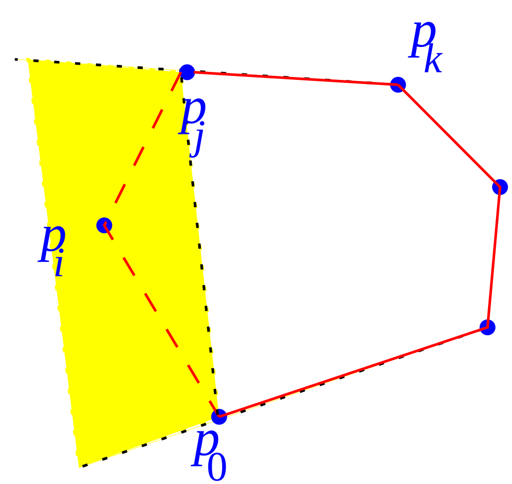
Tw. Jeśli uruchomimy `Graham-Scan` dla zbioru punktów Q, gdzie $|Q| \geq 3$, to punkt z Q pozostaje na stosie S po zakończeniu wtedy i tylko wtedy, gdy jest wierzchołkiem $CH(Q)$.

D-d. Każdy wierzchołek będący kombinacją wypukłą p_0 i pewnego innego wierzchołka nie należy do $CH(Q)$ i jest usuwany zaraz po posortowaniu (wiersz 2).

Możliwe sytuacje przy przetwarzaniu punktu p_i:



(a)



(b)

analiza poprawności

Niech p_j punkt zdjęty ze stosu przy przetwarzaniu p_i (przypadek (a)). Kąt $\angle p_k p_j p_i$ nie oznacza skrętu w lewo, gdzie p_k – element pod p_j na S. Ponieważ wierzchołki są rozpatrywane w kolejności rosnących współrzędnych kątowych, musi istnieć trójkąt $\triangle p_0 p_k p_i$, taki, że p_j leży w jego wnętrzu lub na boku $\overline{p_k p_i}$. Zatem $p_j \notin CH(Q)$.
Omówiliśmy wszystkie przypadki punktów, które mogą nie być na S po zakończeniu.

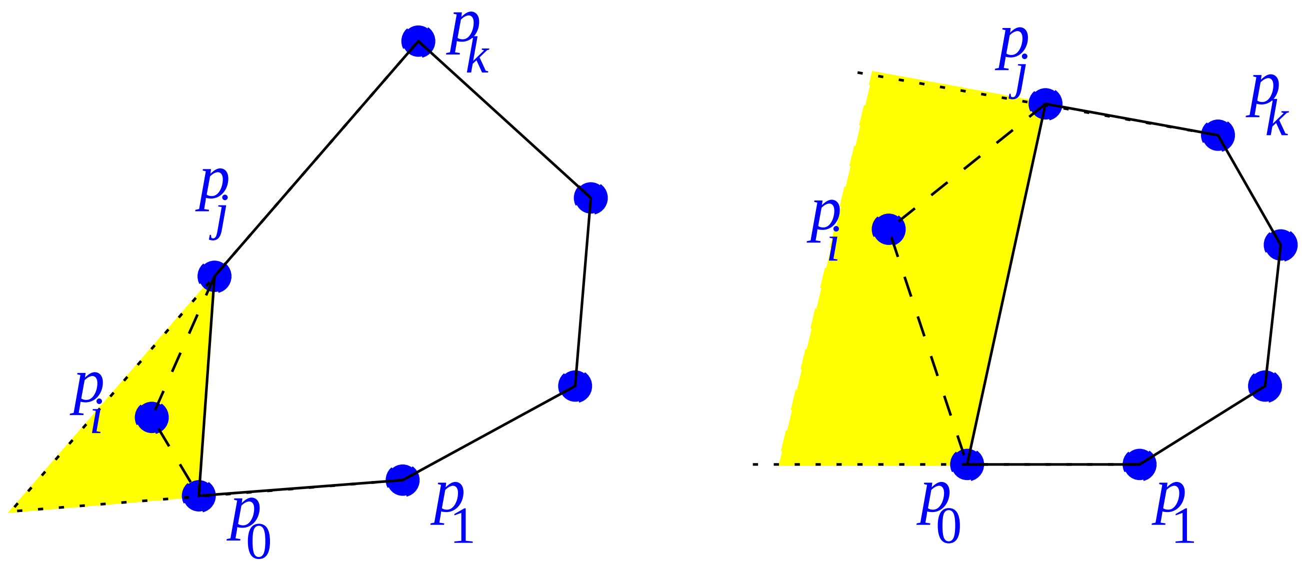
...

analiza poprawności

Pokażemy, że każdy punkt pozostający na S jest w $CH(Q)$.
Niezmiennik: punkty na S tworzą wierzchołki wielokąta wypukłego w kolejności przeciwnej do ruchu wskazówek zegara. Prawdziwy na początku: trójkąt $\triangle p_0 p_1 p_2$ jest taki.
Punkty są wstawiane albo usuwane. Usunięcie punktu nie psuje niezmiennika (nigdy nie usuniemy p_1 – bo był najdalszym o najmniejszej współrzędnej kątowej – a zaraz po usuwaniu wstawiamy następny wierzchołek i mamy przynajmniej trójkąt).
Jeśli do wielokąta wypukłego S dodawany jest wierzchołek p_i wewnątrz obszaru ograniczonego przez jego ostatni bok $\overline{p_j p_0}$ i przedłużenia dwóch sąsiednich boków, to powstanie wielokąt wypukły.

...

analiza poprawności



Jeśli po wstawieniu p_i na S pod p_i są kolejno p_j i p_k, to p_i znajduje się po właściwej stronie prostej $p_k p_j$ (bo przechodząc przez p_j skręcamy w lewo).
Ponieważ współrzędna kątowa p_i jest większa od p_j, p_i jest po właściwej stronie $p_0 p_j$.
p_i musi też leżeć po właściwej stronie prostej $p_0 p_1$.
($0 < \angle p_1 p_0 p_i < \pi$).

analiza poprawności

Zatem po zakończeniu: Wierzchołki na $S \subseteq Q$ tworzą wielokąt wypukły, a pozostałe wierzchołki z Q nie należą do $CH(Q)$. Stąd $S = CH(Q)$. $\square$

Czas: $O(n \lg n)$.

- sortowanie: $O(n \lg n)$,
- eliminacja nienajdalszych od p_0 punktów o tej samej współrzędnej kątowej: $O(n)$,
- pętla `for`: $O(n)$ iteracji
- instrukcje poza pętlą `while` w jednej iteracji `for`: $O(1)$
- łączny czas `while` we wszystkich iteracjach `for`: $O(n)$ (każdy punkt może być ≤ 1 raz zdjęty z S)

Algorytm Jarvisa

Metoda *owijania* – zaczepiamy taśmę w najbardziej lewym z najniższych punktów i naciągamy w prawo, a następnie owijamy zbiór punktów obracając jej koniec w kierunku przeciwnym do ruchu wskazówek zegara.

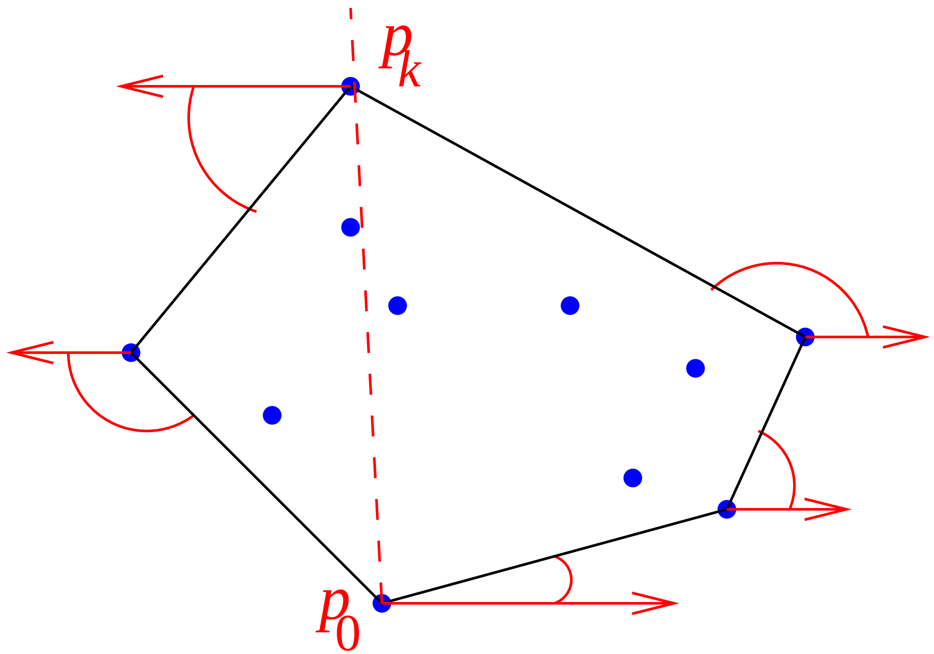
Stosujemy nieco inną wersję:

Niech p_0 najbardziej lewy z najniższych punktów, a p_k – najbardziej prawy z najwyższych punktów.

Idąc od p_0 do p_k konstruujemy *prawy łańcuch*: Najpierw dodajemy p_0. Po dodaniu p_i jako następny dodajemy wierzchołek o najmniejszej współrzędnej kątowej względem p_i w czasie $O(n)$.

Symetrycznie (t.j po odwróceniu osi X i Y) konstruujemy *lewy łańcuch* idąc od p_k do p_0.

Algorytm Jarvisa



Czas: $O(nh)$, gdzie h – liczność $CH(Q)$ (dla każdego dodanego punktu do $CH(Q)$ w czasie $O(n)$ szukamy następnego).
Jeśli $h = o(\lg n)$, to lepszy od algorytmu Grahama.

Znajdowanie pary najbliższych punktów

Odległość $p_1 = (x_1, y_1)$ od $p_2 = (x_2, y_2)$:
$|\overline{p_1 p_2}| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
Dane: zbiór punktów Q, $|Q| = n$.
Szukamy pary najmniej odległych punktów (n.p. przydatne w systemach kontroli ruchu samolotowego).
Algorytm "siłowy" – sprawdzanie wszystkich par – czas $\binom{n}{2} = O(n^2)$.
Podamy algorytm "dziel i zwyciężaj" o czasie:
$T(n) = 2T(n/2) + O(n)$ czyli $O(n \lg n)$.
Każde wywołanie rekurencyjne otrzymuje podzbiór $P \subseteq Q$ oraz tablice X i Y, zawierające P posortowane względem współrzędnej odpowiednio x i y.
Jeśli $|P| \leq 3$ to stosujemy metodę "siłową".
...

Znajdowanie pary najbliższych punktów

Dla $|P| > 3$:

Dziel: Znajdujemy pionową prostą l, dzielącą P na dwa podzbiory P_L i P_R tak, że $|P_L| = \lceil |P|/2 \rceil$, $|P_R| = \lfloor |P|/2 \rfloor$, P_R (odp. P_L) – punkty P na lewo (odp. prawo) od l. Rozdzielamy tablicę X na X_L, X_R – odpowiedniki X dla P_L i P_R. Podobnie rozdzielamy Y na Y_L i Y_R.

Zwyciężaj: Wywołania rekurencyjne dla (P_L, X_L, Y_L) oraz (P_R, X_R, Y_R). Niech δ_R i δ_L minimalne odległości znalezione w tych wywołaniach. Niech $\delta = \min\{\delta_L, \delta_R\}$.

Połącz: Szukamy pary punktów (p_L, p_R) o najmniejszej odległości takiej, że $p_L \in P_L$, $p_R \in P_R$ i $|\overline{p_L p_R}| < \delta$. Jeśli nie znajdziemy, to zwracamy lepszy z wyników wywołań rekurencyjnych.

Pokażemy jak wykonać **Dziel** i **Połącz** w czasie $O(n)$.

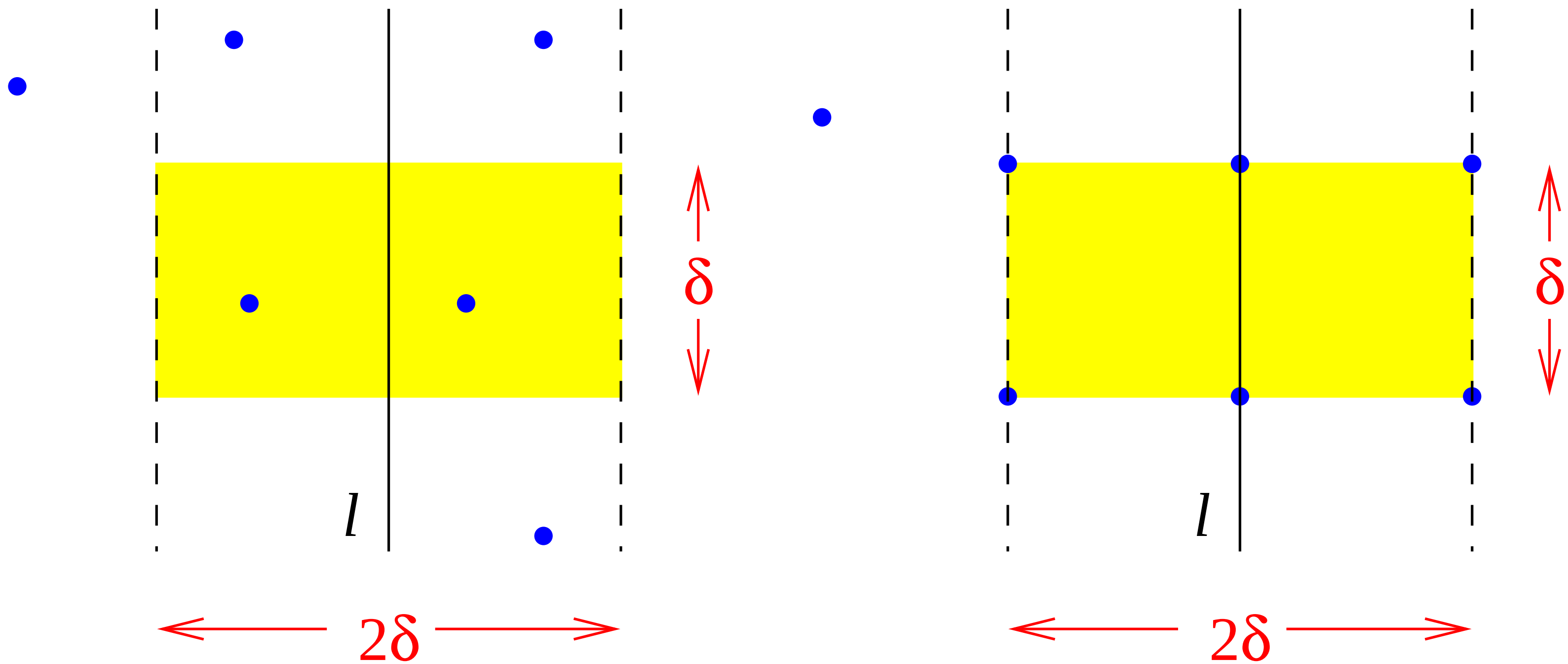
Dziel

Prostą l można wyznaczyć w $O(1)$ ze środkowych elementów tablicy X. (W zasadzie mamy wtedy też podział na X_L i X_R.) Tablie Y_L i Y_R wyznaczamy następująco:

length$[Y_L] \leftarrow$*length*$[Y_R] \leftarrow 0$
for $i \leftarrow 1$ **to** *length*$[Y]$ **do**
    if $Y[i] \in P_L$ **then**
        length$[Y_L] \leftarrow$*length*$[Y_L] + 1$
        $Y_L[$*length*$[Y_L]] \leftarrow Y[i]$
    else
        length$[Y_R] \leftarrow$*length*$[Y_R] + 1$
        $Y_R[$*length*$[Y_R]] \leftarrow Y[i]$

Czas: $O(n)$.

Połącz



Punkty p_L i p_R muszą być w odległości $< \delta$ od l i różnica ich wsp. y jest $< \delta$. Zatem oba znajdują się w jednym prostokącie wymiarach $2\delta \times \delta$, o środku na prostej l.
W takim prostokącie może znajdować się maksymalnie 8 punktów: 4 z P_L na rogach lewej połowy i 4 z P_R na rogach prawej połowy (bo odległość między dowolną parą wewnątrz P_L albo P_R jest $\geq \delta$).

Połącz

1. Z tablicy Y tworzymy Y' – wybieramy punkty odległe o $\leq \delta$ od l w tej samej kolejności co w Y (w czasie $O(|Y|)$).
2. Dla każdego p sprawdzamy odległość od 7 kolejnych punktów w Y', minimalną odległość pamiętamy w δ' i zapamiętujemy parę o najmniejszej odległości. (Czas: $O(|Y'|)$.)
3. Jeśli $\delta' < \delta$ to zwracana jest ta para. W przeciwnym wypadku zwracana jest mniej odległa z par znalezionych w P_L i P_R.

Czas całego algorytmu: $O(n \lg n)$. (Wstępne sortowanie tablic X i Y dla całego $P = Q$: $O(n \lg n)$. Dzielenie i łączenie w algorytmie rekurencyjnym: $O(n)$. Zatem czas części rekurencyjnej: $T(n) = O(n \lg n)$.)