Geometria obliczeniowa

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Co to jest geometria obliczeniowa?

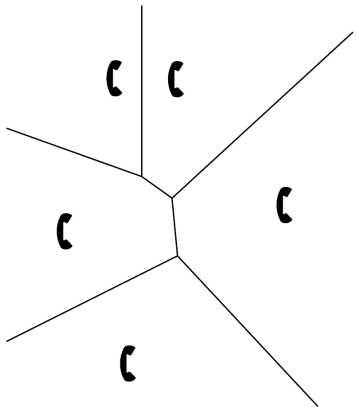
Geometria obliczeniowa jest gałęzią informatyki zajmującą się badaniem algorytmów, które mogą być wyrażone za pomocą geometrii. Jej początki sięgają lat 70-tych XX w.

Co to jest geometria obliczeniowa?

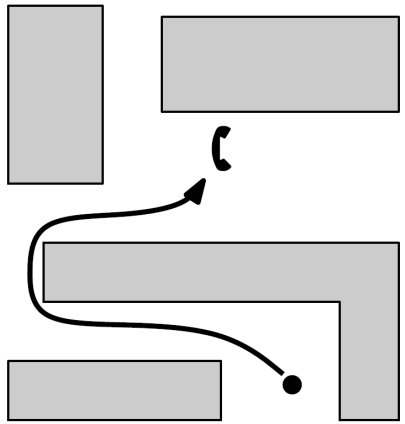
Geometria obliczeniowa jest gałęzią informatyki zajmującą się badaniem algorytmów, które mogą być wyrażone za pomocą geometrii. Jej początki sięgają lat 70-tych XX w.

Przykłady problemów, które możemy wyrazić za pomocą geometrii:

- Załóżmy, że znajdujemy się na terenie kampusu uczelni i pilnie potrzebujemy zatelefonować. Zatem musimy znaleźć najbliższy automat telefoniczny.



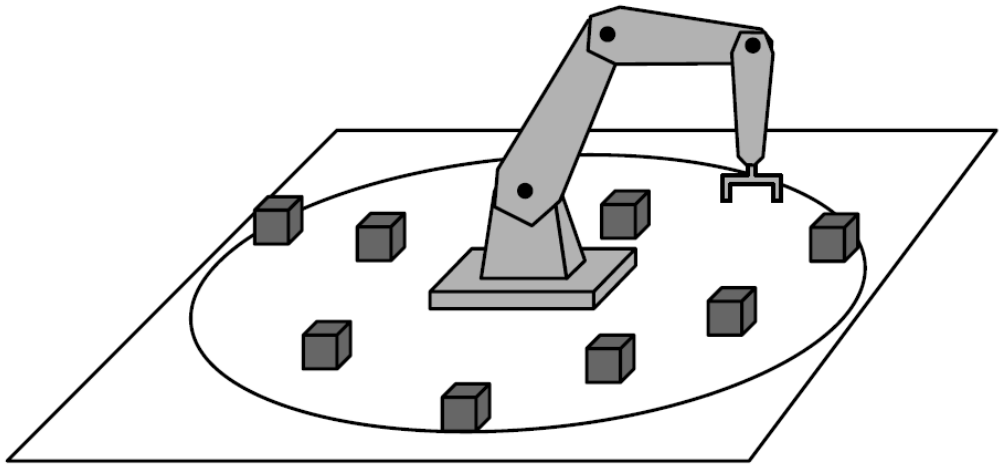
- Załóżmy, że zlokalizowaliśmy najbliższy automat telefoniczny. Musimy do niego dotrzeć najkrótszą ścieżką. Dla człowieka z mapą nie jest to problem, ale zaprogramowanie robota, aby dotarł do celu jest o wiele trudniejsze.



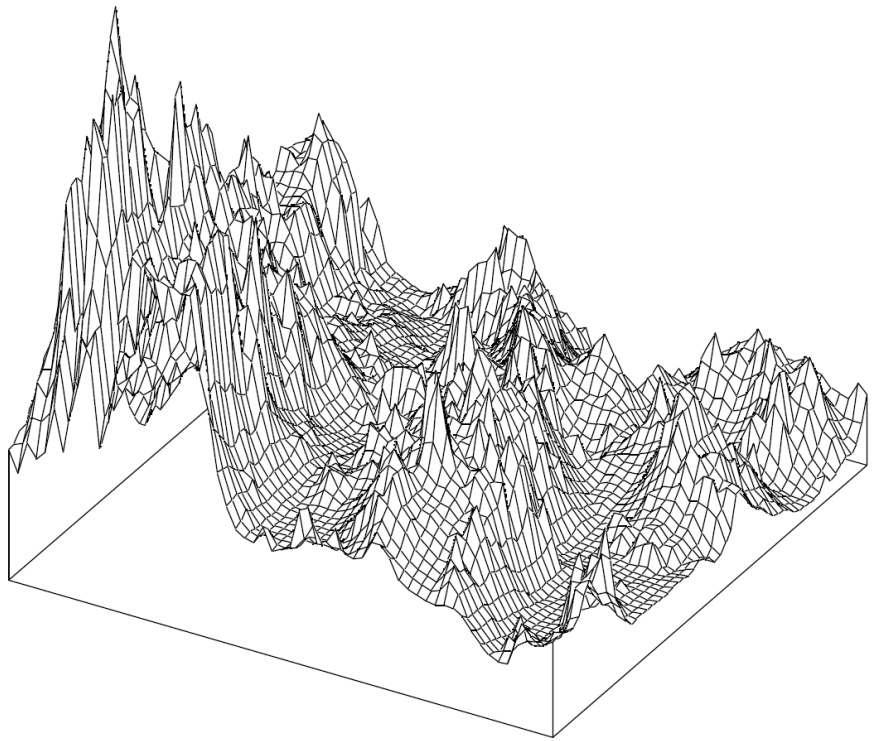
- Załóżmy, że mamy nie jedną a dwie mapy. Pierwszą z opisem budynków, a drugą z drogami. Aby zaplanować ścieżkę musimy nałożyć jedną mapę na drugą.

Dziedziny zastosowań geometrii obliczeniowej:

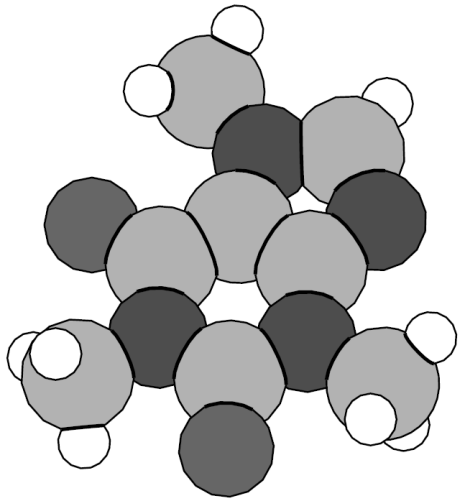
- *Grafika komputerowa*, np. w grafice 2D – znajdowanie przecięć pewnych prostych obrazów, określenie podzbioru obrazów leżących w wyszczególnionym obszarze, w grafice 3D – usuwanie niewidocznych powierzchni, obliczanie cieni, wykrywanie kolizji.
- *Robotyka*, np. planowanie ruchu robota, optymalne umiejscowienie ramienia robota.



- *Systemy informacji geograficznej* (GIS), np. przechowywanie danych geograficznych (mapy dróg, budynków itp.), tworzenie modelu terenu na podstawie danych z pewnych punktów pomiarowych, lokalizacja najbliższej budki telefonicznej, szpitala itp., wyznaczanie przecięcia modelów siatkowych.



- *CAD/CAM*, np. wyznaczanie przecięć i sum obiektów, rozkład obiektów i ich brzegów na prostsze kształty, sprawdzanie wykonalności projektu używając pewnego procesu wytwarzania.
- *Inne*, np. modelowanie molekularne (obliczanie sumy kulek atomów, określenie miejsca stykania się molekuł), rozpoznawanie obrazów (podobieństwo dwóch obiektów), bazy danych (zapytania).



caffeine

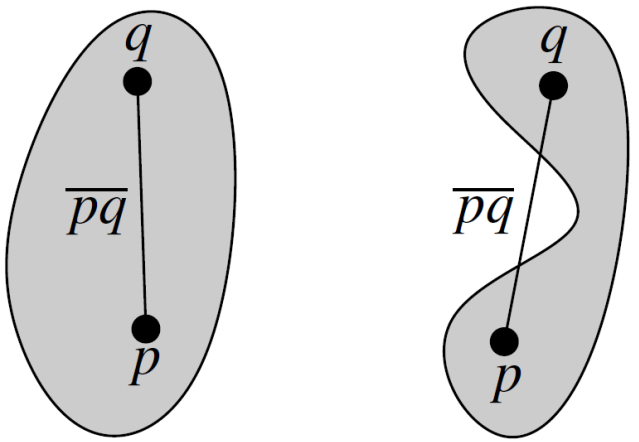
Otoczka wypukła

Problem wyznaczania otoczki wypukłej na płaszczyźnie był jednym z pierwszych problemów badanych w geometrii obliczeniowej.

Otoczka wypukła

Problem wyznaczania otoczki wypukłej na płaszczyźnie był jednym z pierwszych problemów badanych w geometrii obliczeniowej.

Podzbiór płaszczyzny S nazywamy wypukłym (ang. convex) $\iff$ dla dowolnej pary punktów $p, q \in S$ odcinek $\overline{pq}$ jest całkowicie zawarty w S.



Otoczka wypukła (ang. convex hull) $\mathcal{CH}(S)$ zbioru S jest to najmniejszy zbiór wypukły zawierający S, tzn.

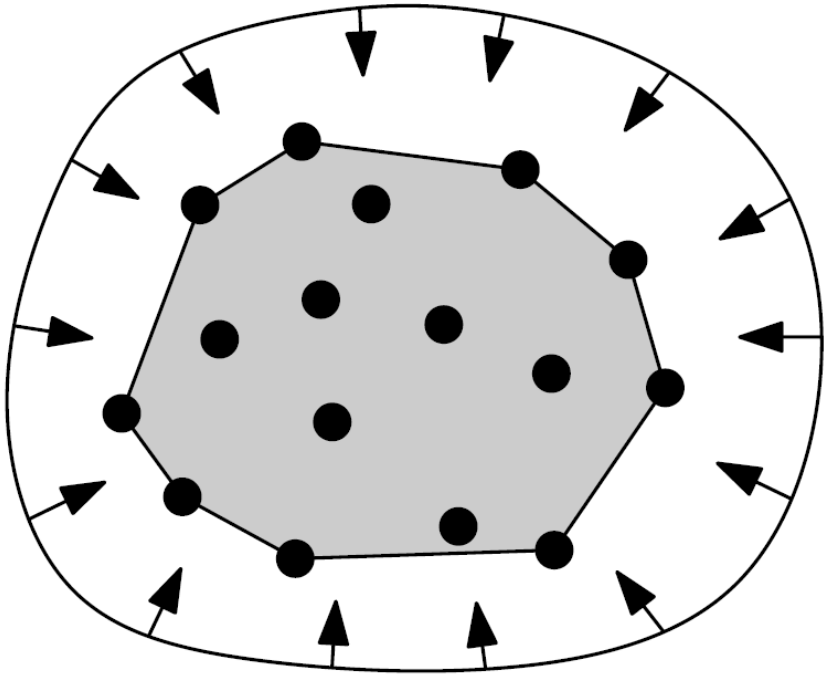
$$\mathcal{CH}(S) = \bigcap_{\substack{S \subset A \\ A - \text{wypukły}}} A.$$

Otoczka wypukła (ang. convex hull) $\mathcal{CH}(S)$ zbioru S jest to najmniejszy zbiór wypukły zawierający S, tzn.

$$\mathcal{CH}(S) = \bigcap_{\substack{S \subset A \\ A - \text{wypukły}}} A.$$

Będziemy szukać sposobu obliczania otoczki wypukłej skończonego zbioru P zawierającego n punktów na płaszczyźnie.

Wyobraźmy sobie, że punkty są gwoździami wystającymi z płaszczyzny. Bierzemy elastyczną gumową taśmę, trzymamy ją wokół gwoździ i puszczamy.



Obszar ograniczony przez gumową taśmę jest otoczką wypukłą zbioru P.

Zatem otoczkę wypukłą skończonego zbioru punktów na płaszczyźnie możemy zdefiniować w nieco inny sposób: jest to jedyny taki wielokąt wypukły, którego wierzchołki są punktami ze zbioru P i który zawiera wszystkie punkty zbioru P.

Zatem otoczkę wypukłą skończonego zbioru punktów na płaszczyźnie możemy zdefiniować w nieco inny sposób: jest to jedyny taki wielokąt wypukły, którego wierzchołki są punktami ze zbioru P i który zawiera wszystkie punkty zbioru P.

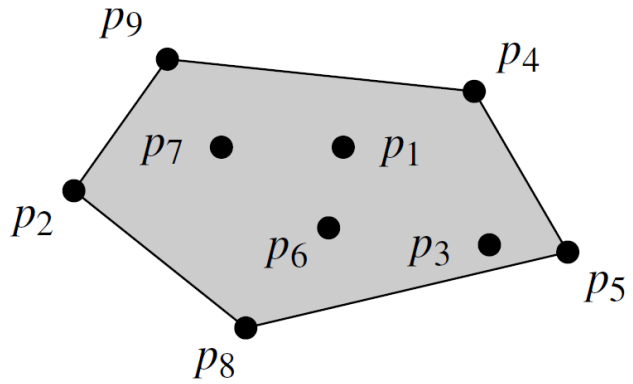
Przy takiej definicji nasz problem przyjmuje postać:
Dla danego zbioru $P = \{p_1, p_2, \ldots, p_n\}$ punktów na płaszczyźnie oblicz listę, która zawiera punkty z P będące wierzchołkami otoczki $\mathcal{CH}(P)$, wymienione w porządku zgodnym z ruchem wskazówek zegara.

input = set of points:

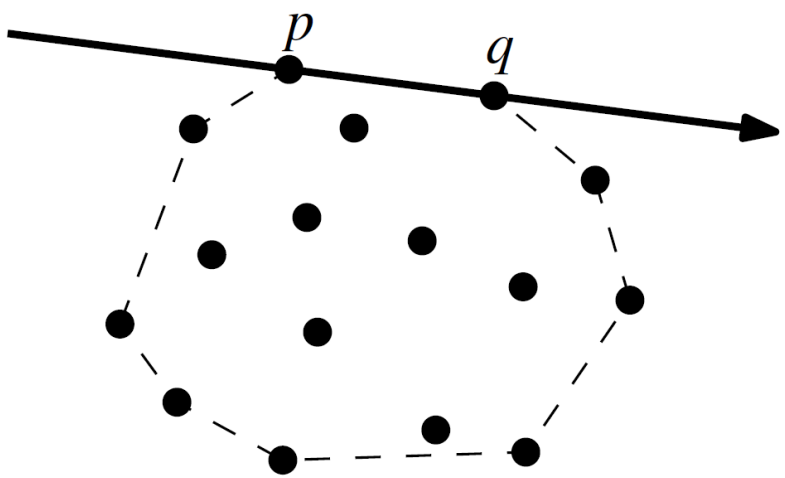
$p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

output = representation of the convex hull:

p_4, p_5, p_8, p_2, p_9



Zobaczmy czym są krawędzie otoczki wypukłej $\mathcal{CH}(P)$.
Oba końce p i q takiej krawędzi są punktami z P. Jeśli skierujemy prostą przez p i q tak, że $\mathcal{CH}(P)$ leży po jej prawej stronie, to wszystkie punkty P muszą leżeć po prawej stronie tej prostej.



Odwrotne stwierdzenie jest również prawdziwe.

Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

```
1.  E ← ∅.
2.  for all ordered pairs (p, q) ∈ P × P with p not equal to q
3.      do valid ← true
4.          for all points r ∈ P not equal to p or q
5.              do if r lies to the left of the directed line from p to q
6.                  then valid ← false.
7.          if valid then Add the directed edge pq→ to E.
8.  From the set E of edges construct a list L of vertices of CH(P), sorted in clockwise order.
```

Jak sprawdzić czy punkt leży po prawej/lewej stronie prostej?

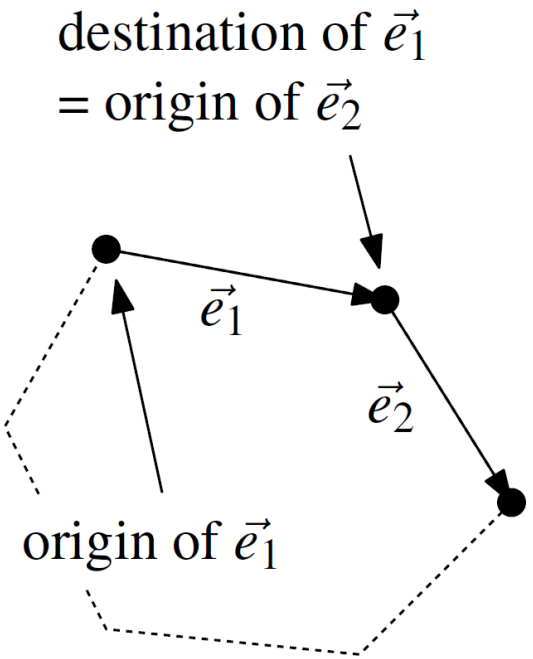
Załóżmy, że mamy punkty $p = (p_x, p_y)$, $q = (q_x, q_y)$ wyznaczające prostą o kierunku pq oraz $r = (r_x, r_y)$. Obliczamy:

$$D = \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}$$

Teraz:

- jeśli $D > 0$, to r leży po lewej stronie prostej pq,
- jeśli $D < 0$, to r leży po prawej stronie prostej pq,
- jeśli $D = 0$, to r leży na prostej pq.

W celu utworzenia listy $\mathcal{L}$ możemy postępować w następujący sposób.
Usuń dowolną krawędź $\overrightarrow{e}_1$ z E. Umieść początek $\overrightarrow{e}_1$ jako pierwszy punkt w $\mathcal{L}$, a koniec jako drugi punkt. Znajdź krawędź $\overrightarrow{e}_2$ w E, której początek jest końcem $\overrightarrow{e}_1$, usuń ją z E i dołącz jej koniec do $\mathcal{L}$.



Znajdź krawędź $\overrightarrow{e}_3$, której początek jest końcem $\overrightarrow{e}_2$, usuń ją z E i dołącz jej koniec do $\mathcal{L}$. Kontynuujemy to postępowanie tak długo, aż w E pozostanie tylko jedna krawędź.

Prosta implementacja tej procedury wymaga czasu $\mathcal{O}(n^2)$. Można ją poprawić do $\mathcal{O}(n \log n)$, ale czas potrzebny dla pozostałej części algorytmu przeważa.

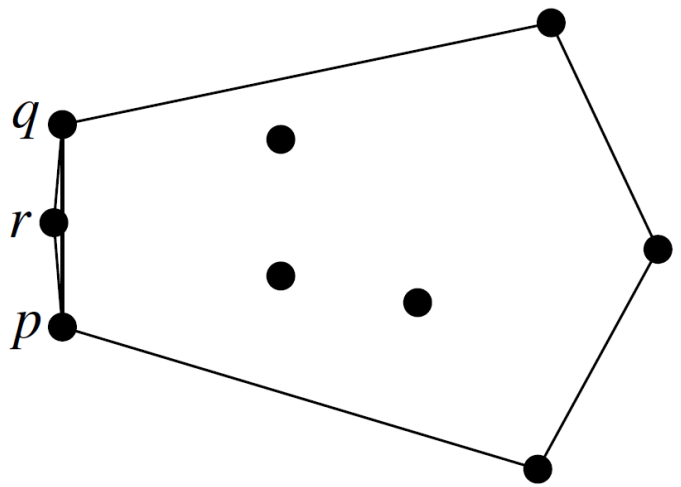
Całkowity czas algorytmu SlowConvexHull wynosi $\mathcal{O}(n^3)$.

W naszym algorytmie nie uwzględniliśmy przypadku gdy punkt r leży na prostej pq. Taką sytuację nazywamy przypadkiem zdegenerowanym lub w skrócie degeneracją.

Aby poprawić algorytm (linia 5) musimy użyć następującego kryterium:
Skierowana krawędź $\overrightarrow{pq}$ jest krawędzią $\mathcal{CH}(P) \iff$ wszystkie pozostałe punkty $r \in P$ leżą albo dokładnie po prawej stronie skierowanej prostej przechodzącej przez p i q, albo należą do otwartego odcinka $\overline{pq}$.

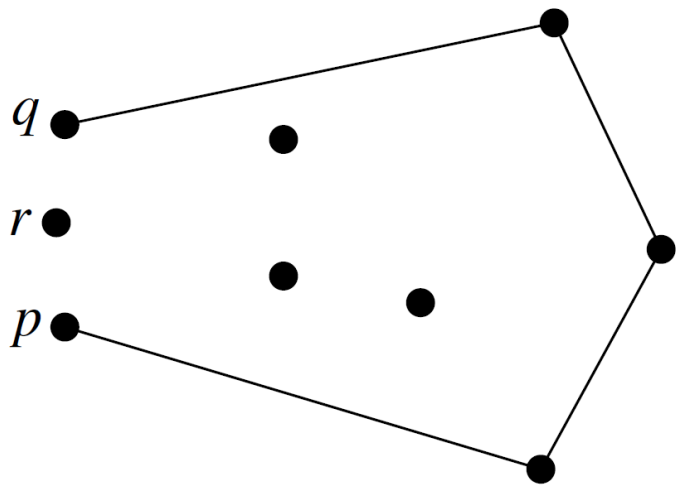
Algorytm ma jeszcze jedną wadę jeśli korzystamy ze współrzędnych zmiennopozycyjnych. W tym przypadku w obliczeniach pojawiają się błędy zaokrągleń, które mogą wypaczyć wynik testu.

Załóżmy, że punkty p, q, r są prawie współliniowe, a pozostałe punkty leżą daleko na prawo od nich.



W takim przypadku może się zdarzyć, że błędy zaokrągleń spowodują, że uznamy, że r leży po prawej stronie pq, że p leży po prawej stronie rq i q leży po prawej stronie pr. Co jest niemożliwe.

Wszystkie trzy testy mogą też dać przeciwną odpowiedź. W takim przypadku algorytm odrzuci wszystkie trzy krawędzie, prowadząc do powstania wyrwy w brzegu otoczki wypukłej.



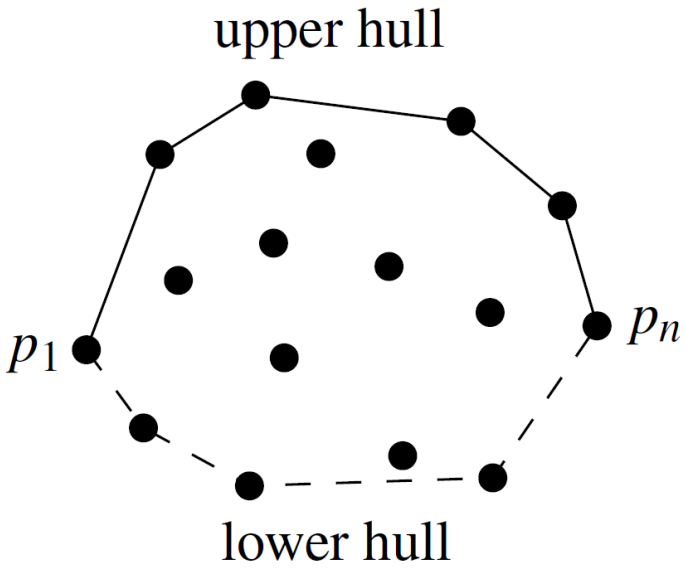
Nasz algorytm jest poprawny i bada wszystkie szczególne przypadki, ale nie jest odporny (ang. robust) na błędy.

Czas algorytmu $\mathcal{O}(n^3)$ jest wolny. W celu skonstruowania szybszego algorytmu użyjemy standardowej techniki projektowania algorytmów geometrycznych. Użyjemy algorytmu przyrostowego (ang. incremental algorithm), tzn. będziemy dodawać pojedynczo punkty z P, uaktualniając nasze rozwiązanie po każdym dodaniu.

Czas algorytmu $\mathcal{O}(n^3)$ jest wolny. W celu skonstruowania szybszego algorytmu użyjemy standardowej techniki projektowania algorytmów geometrycznych. Użyjemy algorytmu przyrostowego (ang. incremental algorithm), tzn. będziemy dodawać pojedynczo punkty z P, uaktualniając nasze rozwiązanie po każdym dodaniu.

Punkty w algorytmie będziemy dodawać od lewej do prawej dlatego najpierw sortujemy punkty względem współrzędnej x, otrzymując posortowany ciąg $p_1, p_2, \ldots, p_n$. Następnie dodajemy je w tym porządku.

Najpierw obliczamy wierzchołki otoczki wypukłej leżące na górnej otoczce, która jest jej częścią biegnącą od skrajnie lewego punktu p_1 do skrajnie prawego punktu p_n. W czasie powtórnego przeglądania punktów (od prawej do lewej) obliczamy pozostałą część otoczki wypukłej czyli otoczkę dolną.

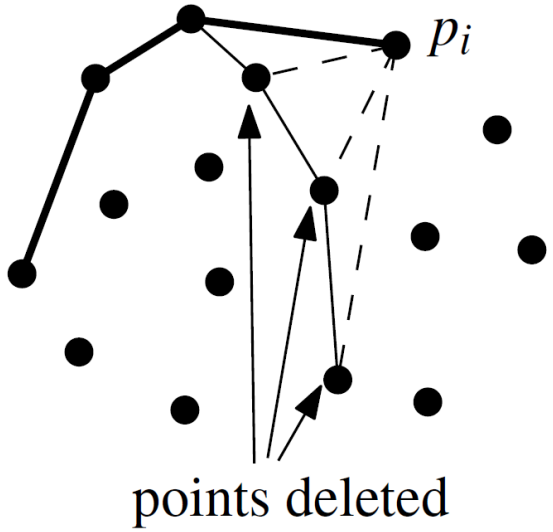


Podstawowym krokiem w algorytmie przyrostowym jest aktualizacja górnej otoczki po dodaniu punktu p_i, tzn. dla górnej otoczki punktów $p_1, \ldots, p_{i-1}$ musimy obliczyć górną otoczkę punktów $p_1, \ldots, p_i$.

Niech $\mathcal{L}_{upper}$ będzie listą, która przechowuje górne wierzchołki uporządkowane od lewego do prawego. Dołączamy p_i do $\mathcal{L}_{upper}$. Następnie sprawdzamy czy ostatnie trzy punkty w $\mathcal{L}_{upper}$ tworzą skręt w prawo:

- jeśli tak, to nic nie robimy,

- jeśli nie, to usuwamy środkowy punkt i ponownie sprawdzamy trzy ostatnie punkty w $\mathcal{L}_{upper}$. Robimy to tak długo aż trzy ostatnie punkty utworzą skręt w prawo lub pozostaną tylko dwa punkty po lewej stronie.



Algorithm CONVEXHULL(P)

Input. A set P of points in the plane.

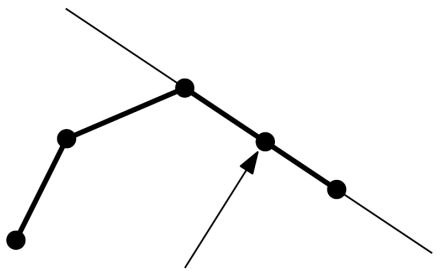
Output. A list containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

1. Sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$.
2. Put the points p_1 and p_2 in a list $\mathcal{L}_{\text{upper}}$, with p_1 as the first point.
3. **for** $i \leftarrow 3$ **to** n
4. **do** Append p_i to $\mathcal{L}_{\text{upper}}$.
5. **while** $\mathcal{L}_{\text{upper}}$ contains more than two points **and** the last three points in $\mathcal{L}_{\text{upper}}$ do not make a right turn
6. **do** Delete the middle of the last three points from $\mathcal{L}_{\text{upper}}$.
7. Put the points p_n and p_{n-1} in a list $\mathcal{L}_{\text{lower}}$, with p_n as the first point.
8. **for** $i \leftarrow n-2$ **downto** 1
9. **do** Append p_i to $\mathcal{L}_{\text{lower}}$.
10. **while** $\mathcal{L}_{\text{lower}}$ contains more than 2 points **and** the last three points in $\mathcal{L}_{\text{lower}}$ do not make a right turn
11. **do** Delete the middle of the last three points from $\mathcal{L}_{\text{lower}}$.
12. Remove the first and the last point from $\mathcal{L}_{\text{lower}}$ to avoid duplication of the points where the upper and lower hull meet.
13. Append $\mathcal{L}_{\text{lower}}$ to $\mathcal{L}_{\text{upper}}$, and call the resulting list $\mathcal{L}$.
14. **return** $\mathcal{L}$

Co musimy jeszcze poprawić w naszym algorytmie?

Nie wzięliśmy pod uwagę przypadku gdy dwa punkty mają tę samą współrzędną x. Wystarczy, że sortowanie punktów wykonamy używając porządku leksykograficznego, a nie sortowania jedynie po współrzędnej x.

A co jeśli trzy punkty, które badamy są współliniowe?



Współliniowe punkty traktujemy jak skręt w lewo.

Algorytm ConvexHull często nazywany jest skanowaniem Grahama i działa w czasie $\mathcal{O}(n \log n)$.

W literaturze możemy znaleźć wiele innych algorytmów wyznaczających otoczkę wypukłą na płaszczyźnie, np.

- marsz Jarvisa,
- algorytm Overmarsa i van Leeuvena,
- algorytm Kirkpatricka i Seidela,
- algorytm Chana.