# Geometria obliczeniowa
## Wykład 1

1. Lokalizacja punktu.
2. Otoczka wypukła.

Założenia ogólne:

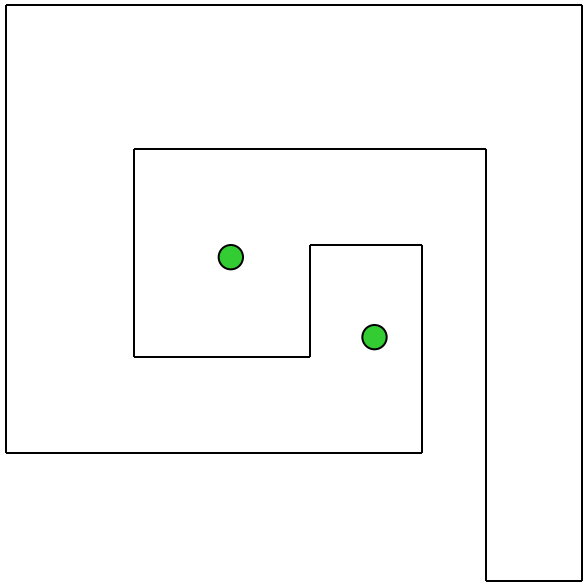
- rozpatrujemy problemy, których dane są w postaci ogólnej, tzn. nie generują szczególnych przypadków, np. nie ma trzech punktów współliniowych (czterech współokręgowych), punkty skrajne są określone jednoznacznie, wszystkie współrzędne punktów są różne itp.
- obliczenia są wykonywane dokładnie, tzn. w obliczeniach nie uwzględniamy błędów zaokrągleń,
- złożoność obliczeniową określamy liczbą działań dominujących (zazwyczaj będą to porównania) względem rozmiaru danych.

Lokalizacja punktu.

Dla danego punktu p sprawdź, czy p znajduje się wewnątrz, czy na zewnątrz n-kąta prostego reprezentowanego przez ciąg kolejnych krawędzi.

k =

```
poprowadź z p pionową półprostą l ;
k:=0;
wybierz bok wielokąta f ;
repeat
    if  l przecina f
        then k:=k+1;
    f:=kolejny bok wielokąta ;
until wszystkie boki wielokąta zostały zbadane ;
if  k parzyste then p jest na zewnątrz
                else p jest wewnątrz ;
```



Złożoność algorytmu.

Czas działania algorytmu jest proporcjonalny do liczby krawędzi wielokąta, czyli jest O(n).

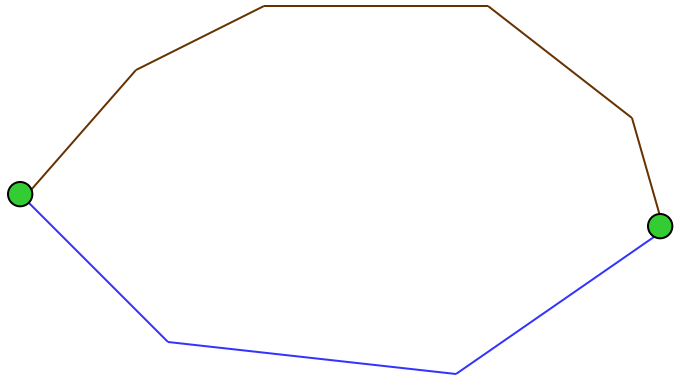
Szczególną uwagę należy zwrócić na sytuacje, gdy wybrana półprosta zawiera wierzchołek lub krawędź wielokąta. Problem ten można rozwiązać np. obracając wielokąt o mały kąt względem badanego punktu (lub wybierając inny kierunek prostej).

Jeśli jest to trudne z przyczyn numerycznych, musimy w algorytmie osobno rozpatrywać takie szczególne przypadki (analizujemy sąsiednie krawędzie – w zależności od ich położenia zliczamy przecięcie lub nie).

Lokalizacja punktu względem wielokąta wypukłego.

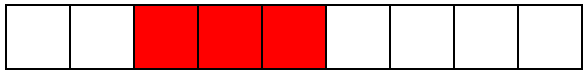
Definicja.

Górnym (dolnym) łańcuchem wielokąta wypukłego nazywamy ciąg jego krawędzi między skrajnie lewym a skrajnie prawym wierzchołkiem wielokąta odwiedzanych zgodnie z ruchem wskazówek zegara (przeciwnie do ruchu wskazówek zegara).

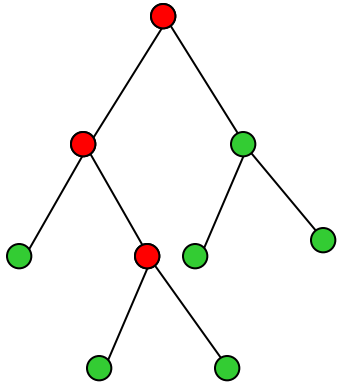


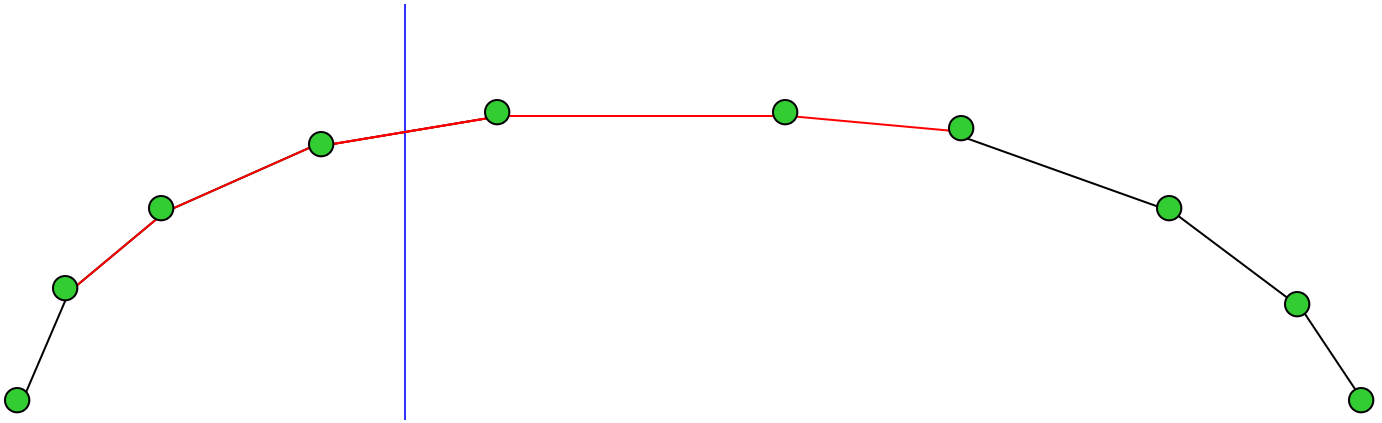
Wyszukiwanie przecięcia łańcucha i prostej.

Tablica



Zrównoważone drzewo poszukiwań





Półprosta poprowadzona z danego punktu przecina każdy z łańcuchów wielokąta wypukłego w co najwyżej jednym punkcie.

Przeszukując strukturę możemy to sprawdzić w czasie O(log n).

Zatem problem lokalizacji punktu względem wielokąta wypukłego możemy rozwiązać w czasie logarytmicznym.

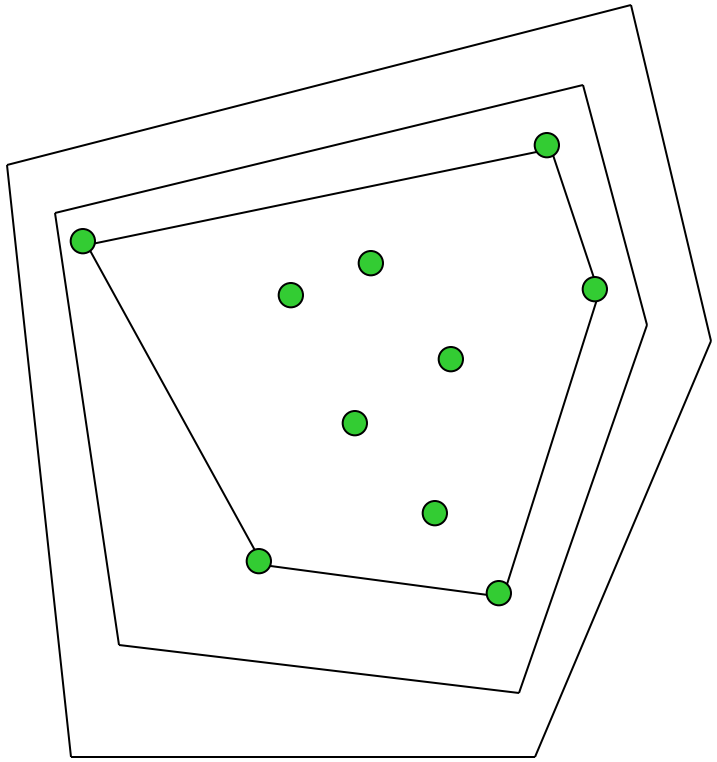
Ogólny problem lokalizacji punktu w przestrzeni trójwymiarowej możemy rozwiązać podobnie jak w dwóch wymiarach. Badamy przecięcia półprostej o początku w danym punkcie p ze ścianami danego wielościanu.

Problem ten możemy rozwiązać w czasie O(n).

# Otoczka wypukła.

Definicja.

Dany zbiór S zawierający n punktów na płaszczyźnie. ***Otoczką wypukłą*** zbioru S nazywamy najmniejszy wielokąt wypukły zawierający zbiór S.



Fakt.

Otoczka wypukła jest wyznaczona jednoznacznie.

Lemat.

Złożoność algorytmu znajdującego otoczkę wypukłą jest $\Omega(n \log n)$.

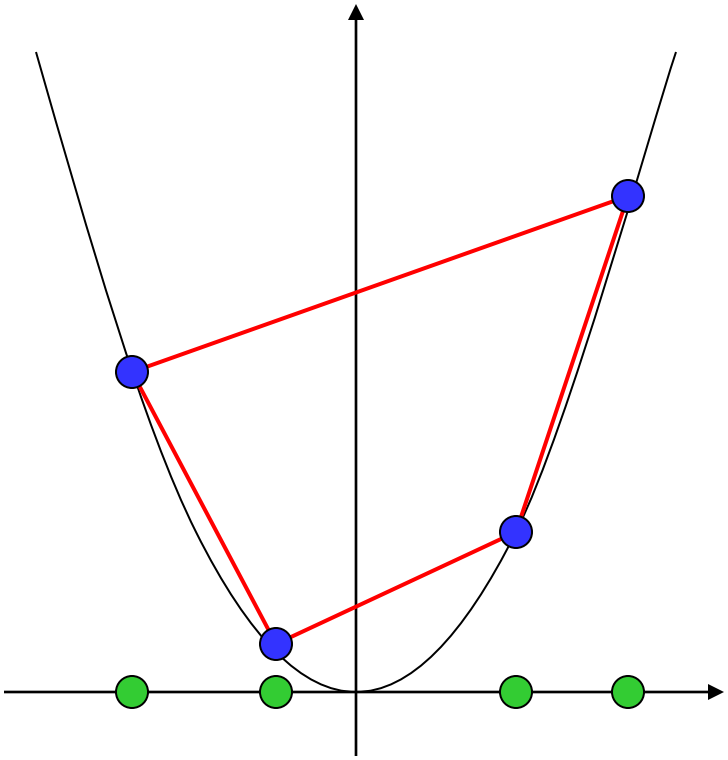
Dowód.

Problem sortowania jest redukowalny do problemu otoczki wypukłej.

Dowolnemu ciągowi arytmetycznemu $(a_i)_{i=1}^n$ przypisujemy zbiór punktów $\{(a_j, a_j^2) : a_j \in (a_i)_{i=1}^n\}$.

Punkty te leżą na paraboli.

Ich otoczka wypukła jednoznacznie określa porządek punktów.



Algorytm Jarvisa.

znajdź punkt $i_0$ z S o najmniejszej współrzędnej y-owej; $i:=i_0$ ;

***repeat***

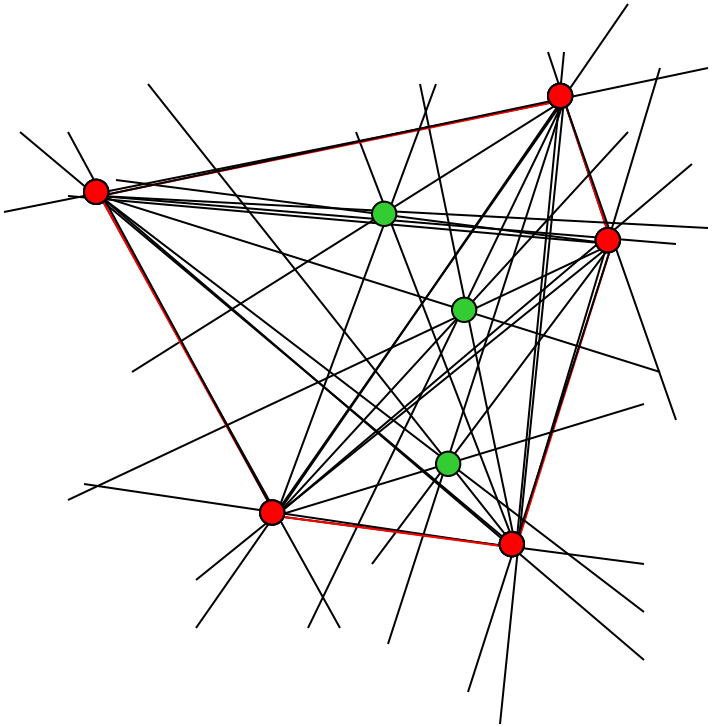
***for*** $j \neq i$ ***do***

znajdź punkt k taki, że żaden punkt z S nie leży na prawo od prostej zawierającej $\overline{ik}$ ;

dodaj $\overline{ik}$ do otoczki ;

i:=k ;

***until*** $i=i_0$ ;



Złożoność algorytmu.

Algorytm Jarvisa działa w czasie $O(n^2)$ (koszt znalezienia punktu skrajnego dla każdego z wierzchołków otoczki jest liniowy), lecz w przypadku, gdy liczba wierzchołków otoczki jest ograniczona przez stałą c, jego złożoność jest liniowa – O(cn).

Algorytm działa również w przestrzeni trójwymiarowej (w czasie $O(n^2)$). Rzutujemy punkty na płaszczyznę i znajdujemy krawędź otoczki wypukłej, znajdujemy przeciwobraz tej krawędzi i płaszczyznę zawierającą go. Następnie znajdujemy punkty wyznaczające płaszczyzny, które nie rozdzielają zbioru S (zawierają ściany otoczki), obracając daną płaszczyznę względem kolejno wyznaczanych krawędzi.

W literaturze algorytm ten często jest nazywany ***marszem Jarvisa***

lub algorytmem ***owijania prezentów*** (gift wrapping).

Algorytm dziel i rządź.

F:={S};

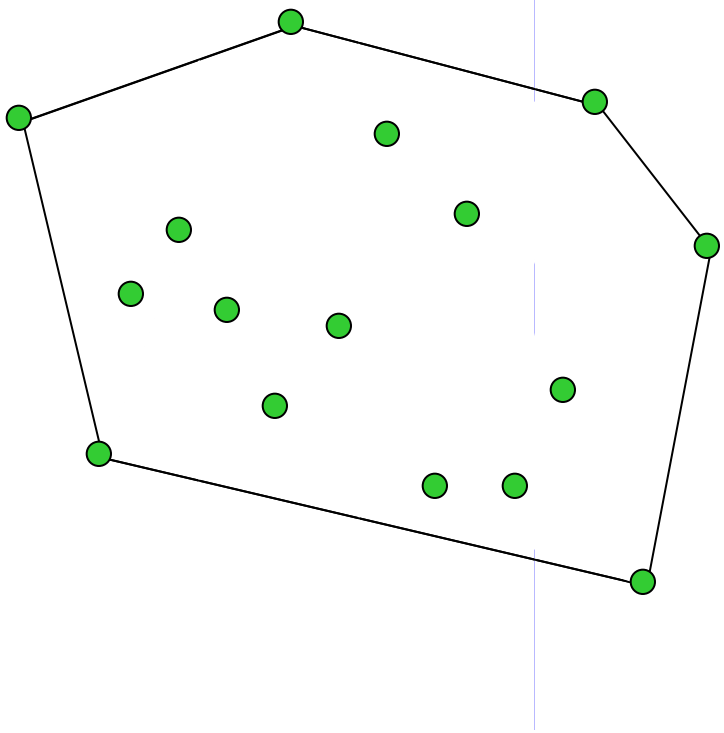
***while*** rozmiar dowolnego zbioru z F przekracza daną stałą k ***do***

dziel zbiory względem mediany x-owych współrzędnych punktów;

znajdź otoczki wypukłe zbiorów z F używając algorytmu naiwnego;

***while*** otoczka zbioru S nie została znaleziona ***do***

sklejaj otoczki sąsiadujących zbiorów w kolejności odwrotnej do podziału rodziny F;



Łączenie sąsiednich otoczek.

Niech l(a,b) oznacza prostą wyznaczoną przez punkty a i b. Styczna do zbiorów A i B to prosta przechodząca przez elementy obu zbiorów i wyznaczająca półpłaszczyznę zawierającą oba zbiory.

a := skrajnie prawy wierzchołek lewej otoczki A ;

b := skrajnie lewy wierzchołek prawej otoczki B ;

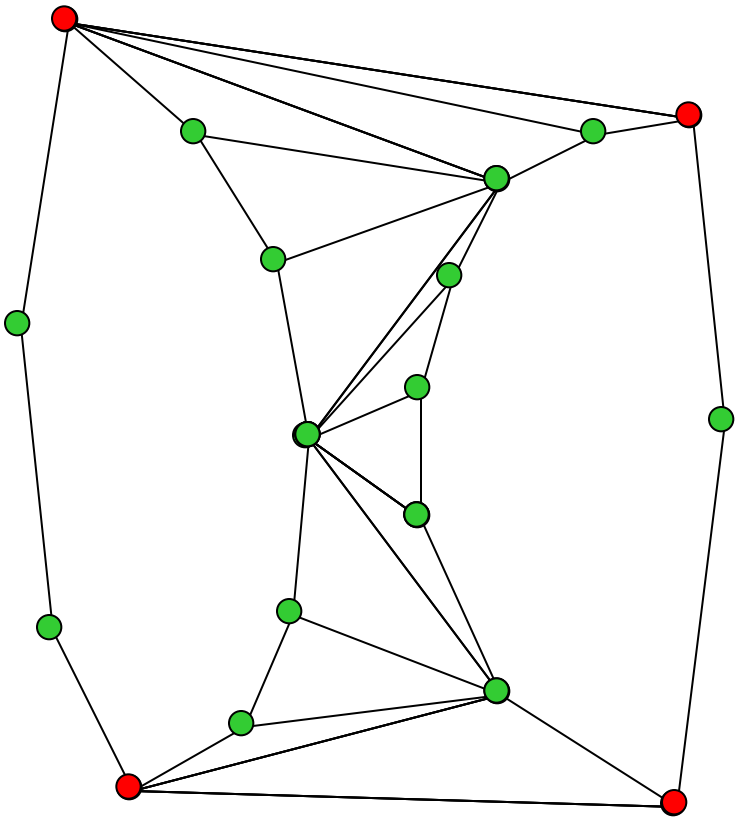
***while*** l(a,b) nie jest styczna do A i B ***do***

***while*** l(a,b) nie jest dolną (górną) styczną do B ***do***

b := dolny (górny) sąsiad b;

***while*** l(a,b) nie jest dolną (górną) styczną do A ***do***

a := dolny (górny) sąsiad a;



Przedstawiona metoda łączenia otoczek nazywa się „podnoszeniem mostów”. Jej autorami są Preparata i Hong.

Czas połączenia dwóch otoczek wynosi O(w(A)+w(B)), gdzie w(A) i w(B) są rozmiarami otoczek.

Zatem jedna faza algorytmu (łączenie otoczek pokrywających cały zbiór S) ma złożoność O(n), gdzie n jest rozmiarem S.

Złożoność algorytmu „dziel i rządź” można opisać równaniem $T(n)=2T(\lceil n/2 \rceil)+O(n)$, czyli wynosi O(n log n).

Zamiast rekurencyjnie dzielić zbiór S możemy go najpierw posortować, a następnie podzielić na dostatecznie małe podzbiory.

Czy dowolny podział zbioru S na małe podzbiory ma wpływ na złożoność czasową algorytmu „dziel i rządź” ?

Algorytm „dziel i rządź” znajduje również otoczkę wypukłą punktów w przestrzeni trójwymiarowej w czasie O(n log n).

Algorytm Kirkpatricka i Seidela.

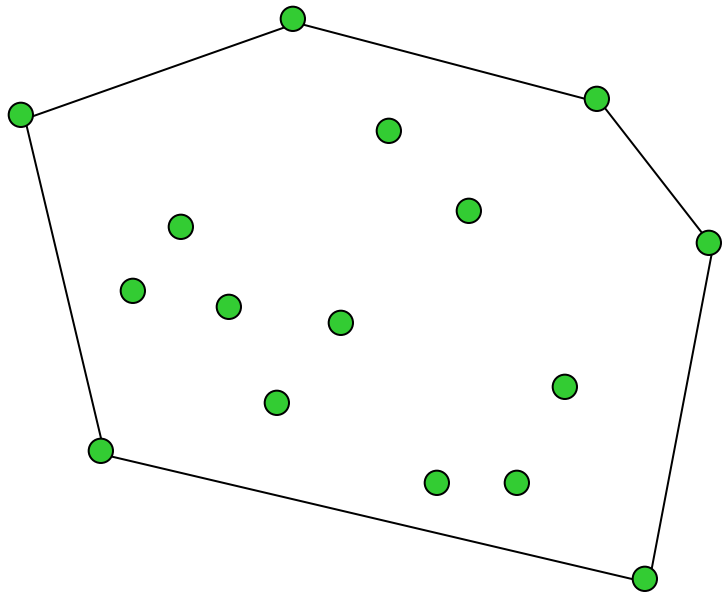
F:={S};

***while*** rozmiar dowolnego zbioru z F przekracza daną stałą k ***do***

dziel zbiory względem mediany x-owych współrzędnych punktów;

znajdź górny (dolny) most między podzbiorami F z sąsiadujących podziałów, jeśli taki most nie został znaleziony wcześniej ;

znajdź pozostałe krawędzie otoczki w każdym podziale;



Zauważmy, że w tym przypadku nie liczymy otoczek wypukłych dla elementów podziału, ale znajdujemy mosty w inny sposób (metodą „prune and search”, którą omówimy w jednym z kolejnych wykładów).

Omawiany algorytm (podobnie jak np. algorytm Jarvisa) jest „output sensitive” tzn. jego złożoność zależy od rozmiaru wyniku.

Niech k będzie rozmiarem otoczki wypukłej.

Złożoność algorytmu Kirkpatricka i Seidela opisuje równanie:

$T(n,k) = (\max_{q+r=k} T(\lceil n/2 \rceil, q) + T(\lfloor n/2 \rfloor, r)) + O(n)$,

którego rozwiązaniem jest $T(n,k) = O(n \log k)$.

Złożoność pesymistyczna tego algorytmu wynosi $O(n \log n)$.

Metoda zamiatania.

Ustalamy pewną hiperpłaszczyznę (np. prostą w $R^2$, płaszczyznę w $R^3$).
Nazywamy ją ***miotłą***.
Przesuwamy miotłę w wyznaczonym kierunku (kierunku zamiatania).
Pozycje, w których miotła zatrzymuje się nazywamy ***zdarzeniami***.
Informacje o nich przechowujemy w ***strukturze zdarzeń***.
Informacje potrzebne do obliczeń przechowujemy w ***strukturze(-ach) stanu***. Struktura stanu jest aktualizowana w każdym zdarzeniu.
Niezmiennik metody zamiatania polega na tym, że na zamiecionym obszarze znane jest rozwiązanie badanego problemu dla zdarzeń należących do tego obszaru.

Algorytm zamiatania.

Struktura zdarzeń – lista L uporządkowanych po x-ach punktów z S.

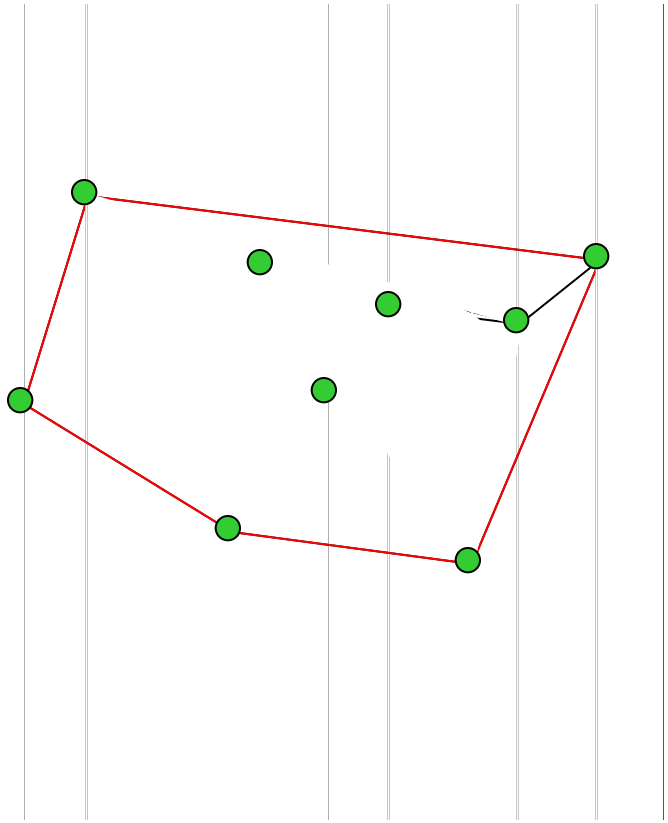
Struktura stanu – dwukierunkowa lista CH opisująca otoczkę dla „zamiecionych” punktów.

i:=pierwszy element L; CH:=puste;

***while*** i ≠ nil ***do***

znajdź styczne do otoczki z CH zawierające i;

aktualizuj CH i L;



Złożoność algorytmu.

Algorytm działa w czasie O(n log n) :

- uporządkowanie punktów wymaga czasu O(n log n),

- znalezienie stycznych do aktualnej otoczki zawierających badany punkt wymaga czasu O(n) (zależy od liczby usuwanych wierzchołków

i rozmiaru zbioru S),

- usunięcie wierzchołków „pochłanianych” przez tworzoną otoczkę

wymaga czasu O(n).

Algorytm zamiatania działa również w przestrzeni trójwymiarowej w czasie $O(n^2)$.

Algorytm Grahama.

znajdź punkt c z S o najmniejszej współrzędnej y-owej;

posortuj biegunowo względem c pozostałe punkty z S i stwórz listę L;

wstaw c i pierwsze dwa elementy z L do stosu CH;

z:=kolejny element z L;

***while*** z ≠ nil ***do***

zdejmij dwa elementy x,y ze stosu CH;

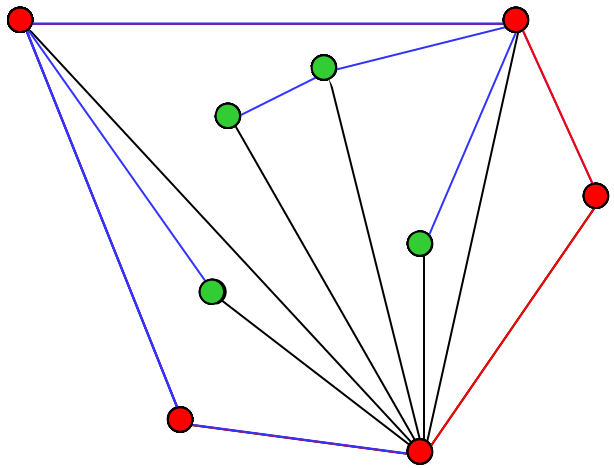
***if*** $|\angle xyz| \geq \pi$

***then*** wstaw x,z do CH

***else*** wstaw x,y,z do CH;

z:=kolejny element z L;

***return*** CH;



Złożoność algorytmu.

Algorytm działa w czasie O(n log n) :

- uporządkowanie punktów wymaga czasu O(n log n),

- czas znalezienia otoczki wynosi O(n) (jest proporcjonalny do sumy liczby usuwanych wierzchołków i rozmiaru zbioru S).

Algorytm Grahama nie ma bezpośredniego odpowiednika w wymiarach wyższych niż 2 z uwagi na niemożność liniowego uporządkowania danych punktów.

Algorytm przyrostowy.

weź trzy punkty ze zbioru S i stwórz z nich otoczkę CH;

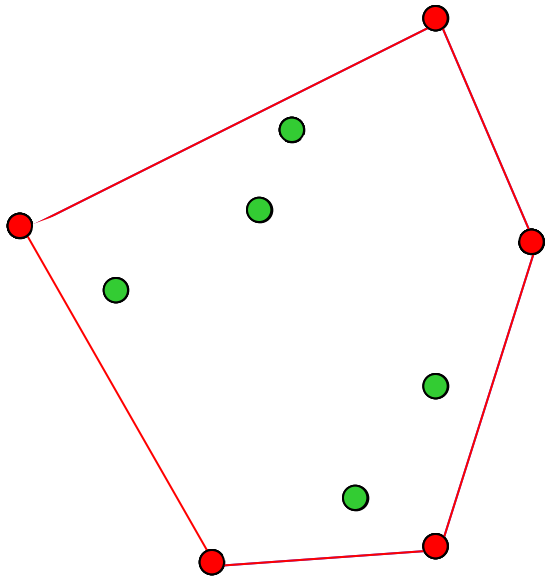
***for*** i:=4 ***to*** n ***do***

    ***if*** i-ty punkt z S nie należy do wnętrza CH

        ***then***

            znajdź styczne do CH przechodzące przez ten punkt;

            aktualizuj CH;



Złożoność algorytmu.

Struktura CH jest wzbogaconym, zrównoważonym drzewem poszukiwań binarnych (porządek wyznacza kolejność punktów na otoczce, ich współrzędne są pamiętane w węzłach drzewa). Możemy też użyć zrównoważonych drzew BST dla dolnego i górnego łańcucha CH.

Koszt sprawdzenia, czy badany punkt leży na zewnątrz aktualnej otoczki wypukłej jest logarytmiczny.

Liczba usuwanych wierzchołków jest O(n) a koszt każdego usunięcia jest logarytmiczny. Podobnie jest dla wstawień wierzchołków do CH.

Zatem algorytm ma koszt O(n log n).

W trzech wymiarach algorytm ma złożoność $O(n^2)$.

Algorytm Quickhull

CH – lista opisująca wierzchołki otoczki z dowiązaniami do list punktów na zewnątrz otoczki.

znajdź w S punkty skrajne względem współrzędnych i wstaw je do CH;

wybierz punkt wewnątrz otoczki;

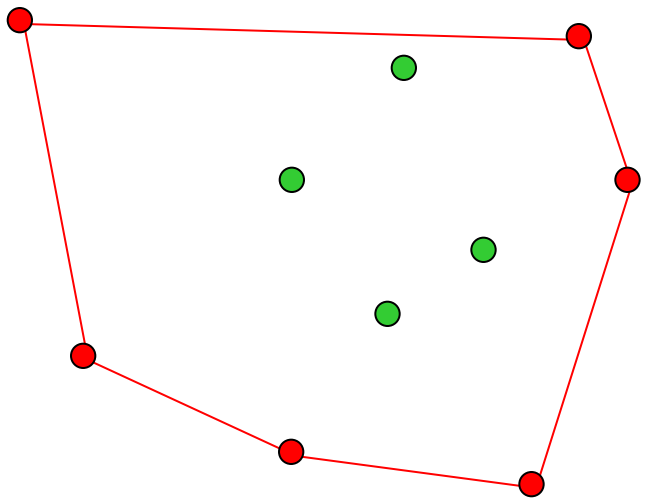
***while*** są punkty z S na zewnątrz otoczki ***do***

    przyporządkuj punkty z S kątom tworzonym przez wybrany punkt i wierzchołki aktualnej otoczki CH;

    w każdym kącie znajdź punkt z S najbardziej odległy od otoczki;

    dodaj wybrane punkty do otoczki;

***return*** CH;



Złożoność algorytmu.

Pesymistyczna złożoność algorytmu wynosi $O(n^2)$.

Jeśli rozmiar otoczki wypukłej jest ograniczony przez stałą c, to złożoność algorytmu wynosi O(cn).

Algorytm również działa w przestrzeni trójwymiarowej.

Po znalezieniu zbioru punktów powiększających otoczkę, znajdujemy jej ściany przed dokonaniem kolejnego podziału zbioru punktów leżących na zewnątrz otoczki. Pesymistyczna złożoność tego algorytmu wynosi $O(n^2)$.

Oczekiwana złożoność algorytmów znajdywania otoczki wypukłej.

Staramy się oszacować oczekiwaną liczbę wierzchołków otoczki wypukłej dla losowego układu punktów. Wynik zależy od wyboru przestrzeni probabilistycznej. Mamy:

-dla równomiernego rozkładu punktów w wielokącie wypukłym – $O(\log n)$ [Rênyi-Sulanke],

- dla równomiernego rozkładu punktów w kole – $O(n^{1/3})$ [Raynaud],

- rozkładu normalnego na płaszczyźnie – $O(\log^{1/2} n)$ [Raynaud].

W zależności od wyboru modelu mamy też odpowiednie oczekiwane złożoności dla algorytmów Jarvisa i Quickhull.

Dziękuję za uwagę.

Ćwiczenia 1.

1. Jak sprawdzić, po której stronie prostej przechodzącej przez punkty p i q znajduje się punkt r ?
2. Pokaż, że problem lokalizacji punktu w przestrzeni trójwymiarowej możemy rozwiązać w czasie O(n). (wielościan jest zadany w postaci uporządkowanych list sąsiedztwa z dowiązaniami)
3. Udowodnij, że otoczka wypukła jest wyznaczona jednoznacznie.
4. Czy dowolny podział zbioru S na małe podzbiory ma wpływ na złożoność czasową algorytmu „dziel i rządź” ?
5. Udowodnij, że algorytm „dziel i rządź” znajduje otoczkę wypukłą punktów w przestrzeni trójwymiarowej w czasie O(n log n).
6. Jak znaleźć styczne w algorytmie Kirkpatricka-Seidela ?
7. Pokaż, że pesymistyczna złożoność algorytmu zamiatania w przestrzeni trójwymiarowej wynosi $\Omega(n^2)$.
8. Pokaż, że pesymistyczna złożoność algorytmu QUICKHULL w przestrzeni dwuwymiarowej wynosi $\Omega(n^2)$.

9. Niech S będzie zbiorem n (być może przecinających się) jednostkowych okręgów na płaszczyźnie. Chcielibyśmy obliczyć otoczkę wypukłą S.

   - Pokaż, że brzeg otoczki wypukłej składa się z odcinków i kawałków okręgów z S.

   - Pokaż, że każdy okrąg może wystąpić co najwyżej raz na brzegu otoczki wypukłej.

   - Niech S' będzie zbiorem punktów, które są środkami okręgów z S. Pokaż, że okrąg z S pojawia się na brzegu otoczki wypukłej wtedy i tylko wtedy, gdy środek tego okręgu leży na otoczce wypukłej S'.

   - Podaj algorytm obliczania otoczki wypukłej S w czasie O(n log n).

   - (*) Podaj algorytm obliczania otoczki wypukłej w czasie O(n log n) w przypadku, w którym okręgi z S mają różne promienie.

10. Dla n danych punktów na płaszczyźnie zbuduj w czasie O(n log n) wielokąt prosty (tzn. łamaną zamkniętą, której krawędzie nie przecinają się) o wierzchołkach w tych punktach.

11. Niech S będzie zbiorem n punktów na płaszczyźnie a współrzędne tych punktów będą liczbami całkowitymi od 0 do $n^d$-1, gdzie d jest stałą. Podaj algorytm znajdujący otoczkę wypukłą S w czasie liniowym.