Geometria obliczeniowa
Wykład 3

Metoda zamiatania.

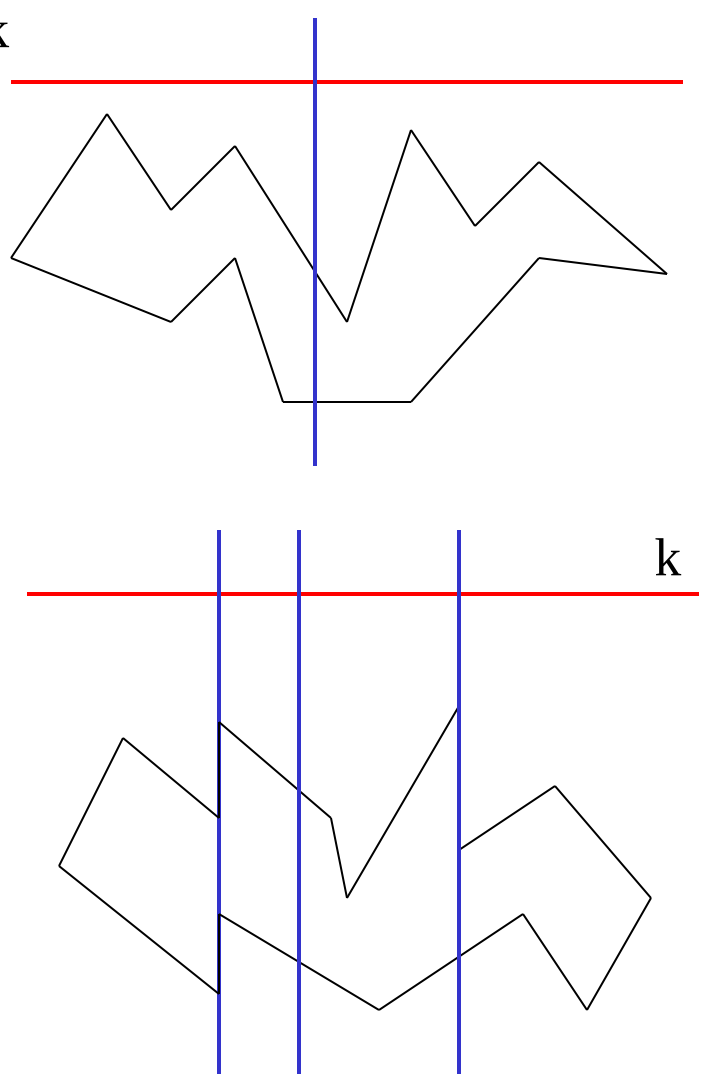
1. Triangulacja wielokąta monotonicznego.
2. Podział wielokąta prostego na wielokąty monotoniczne.
3. Znajdowanie par przecinających się odcinków.
4. Znajdowanie pary najbliższych punktów.

Triangulacja wielokąta monotonicznego.

Definicja.

Wielokąt prosty nazywamy ***ściśle monotonicznym*** względem prostej k (wyznaczającej ***kierunek monotoniczności***), gdy jego brzeg można podzielić na dwa spójne łańcuchy takie, że dowolna prosta prostopadła do k przecina każdy z łańcuchów w co najwyżej jednym punkcie.

Wielokąt jest ***monotoniczny***, gdy przecięcie dowolnej prostej prostopadłej do k z dowolnym łańcuchem jest spójne.



Algorytm.

znajdź górny i dolny łańcuch wielokąta względem kierunku monotoniczności;

posortuj wierzchołki i wstaw je do listy L;

usuń pierwsze dwa wierzchołki z L i wstaw je na stos Q;

while L ≠ ∅ ***do***

p:=FRONT(L); POP(L);

q:=FRONT(Q);

if p i q należą do różnych łańcuchów

then while Q ≠ ∅ ***do***

r:=FRONT(Q); POP(Q);

połącz p z r;

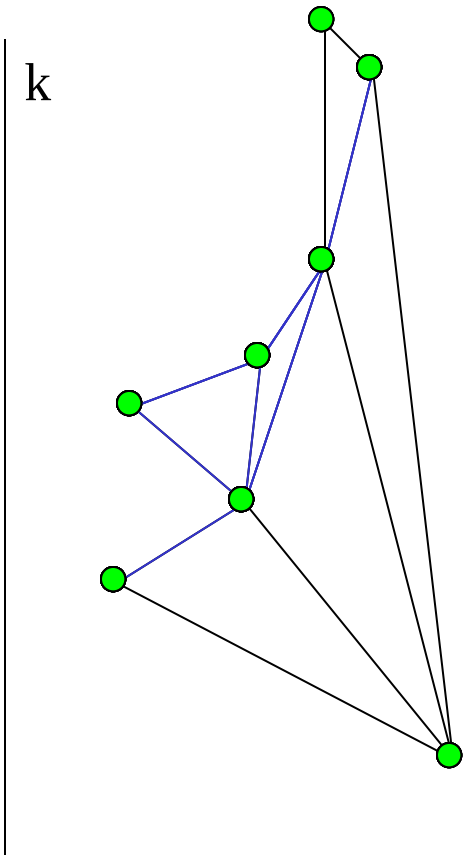
else POP(Q); r:=FRONT(Q);

while |∠pqr| < π ***do***

połącz p z r;

q:=r; POP(Q); r:=FRONT(Q);

wstaw q i p na stos Q;

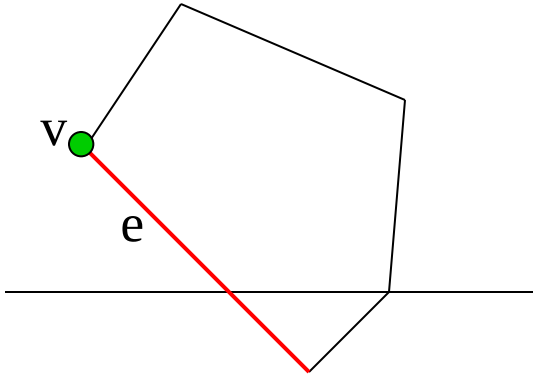


Poprawność i złożoność algorytmu.

Poprawność algorytmu wynika z faktu, że punkty znajdujące się na stosie Q muszą tworzyć wklęsłą łamaną. Z ostatniego punktu taka łamana jest widoczna niezależnie od strony, po której się znajduje.

Algorytm działa w czasie O(n), gdyż w każdym kroku wykonuje liczbę operacji proporcjonalną do liczby wierzchołków zdjętych ze stosu Q + pewna stała. Posortowanie wierzchołków wymaga czasu O(n), gdyż polega na scaleniu wierzchołków z obu łańcuchów w jedną listę.

Podział wielokąta prostego na wielokąty monotoniczne.

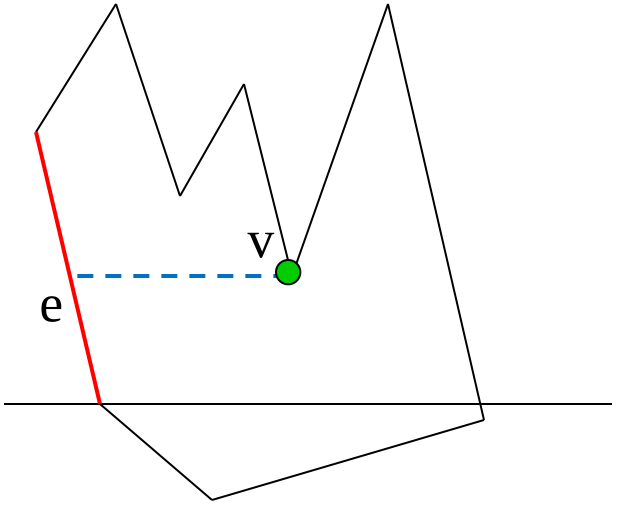


Do rozwiązania tego problemu wykorzystamy metodę zamiatania.

Miotła będzie równoległa do osi x-ów.

Definicje.

Pomocnikiem krawędzi e wielokąta P nazywamy wierzchołek v, który jest prawym końcem równoległego do miotły odcinka łączącego v z e, leżącego powyżej miotły i najbliższego niej, którego wnętrze jest zawarte we wnętrzu P.

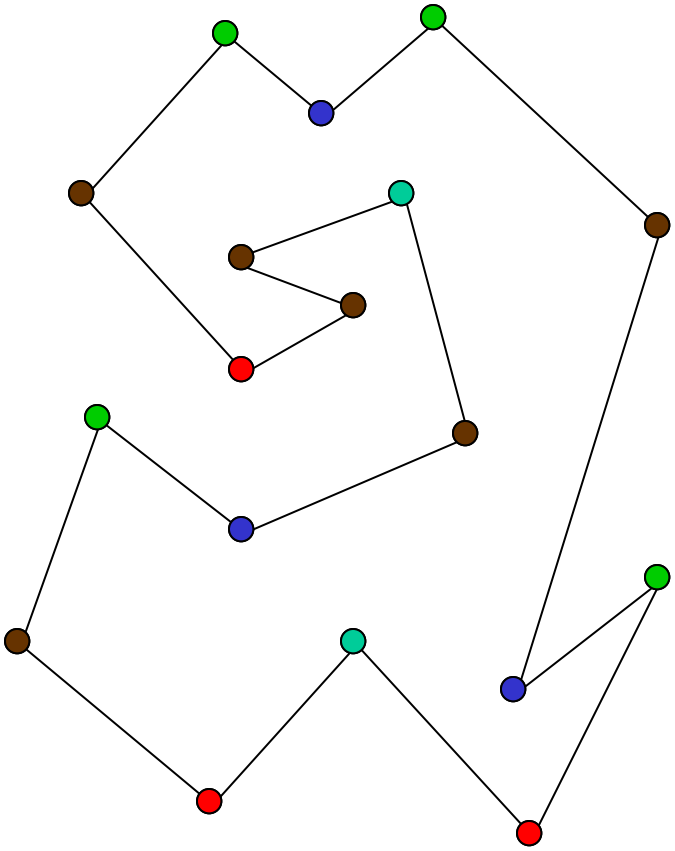


Wielokąt nazywamy ***y-monotonicznym***, gdy jest monotoniczny względem osi y-ów.

Wyróżniamy pięć rodzajów wierzchołków wielokąta w zależności od pozycji miotły przechodzącej przez dany wierzchołek

(badamy wewnętrzne kąty wielokąta) :

- **początkowy**, gdy obie wychodzące z niego krawędzie leżą poniżej miotły i tworzą kąt $< \pi$,
- **dzielący**, gdy obie wychodzące z niego krawędzie leżą poniżej miotły i tworzą kąt $> \pi$,
- **końcowy**, gdy obie wychodzące z niego krawędzie leżą powyżej miotły i tworzą kąt $< \pi$,
- **łączący**, gdy obie wychodzące z niego krawędzie leżą powyżej miotły i tworzą kąt $> \pi$,
- **prawidłowy**, gdy jedna z wychodzących z niego krawędzi leży poniżej a druga powyżej miotły.

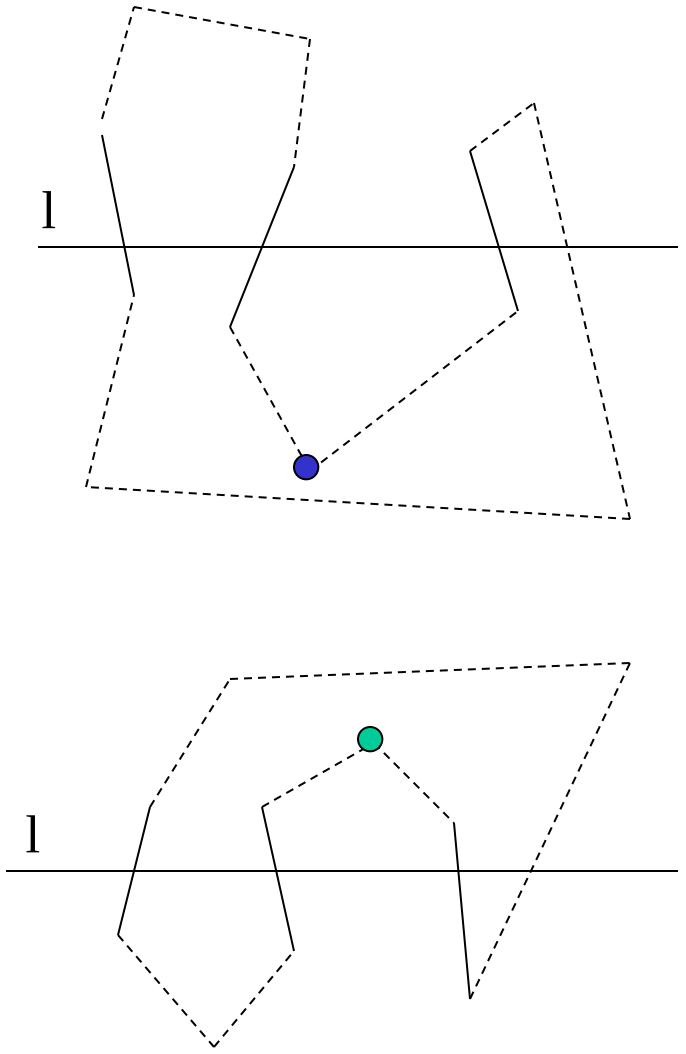


Lemat.

Wielokąt, który nie ma wierzchołków dzielących i łączących, jest y-monotoniczny.

Dowód.

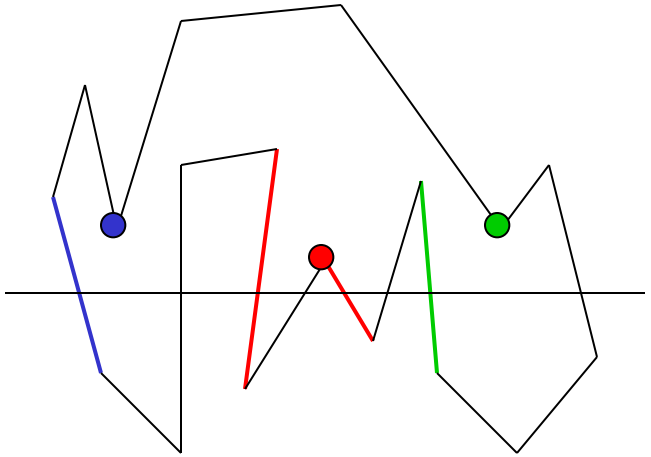
Załóżmy, że wielokąt nie jest y-monotoniczny. Wtedy przecięcie poziomej prostej l z wielokątem tworzy co najmniej dwa spójne przekroje. Istnieje odcinek na zewnątrz wielokąta łączący je. Weźmy ciąg krawędzi wielokąta łączących końce tego odcinka i nieprzecinający prostej. Najbardziej odległy od l wierzchołek takiej łamanej jest wierzchołkiem dzielącym lub łączącym.



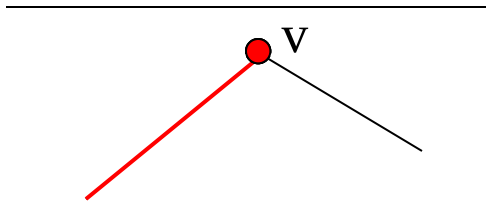
Naszym zadaniem będzie dodanie przekątnych likwidujących wierzchołki łączące i dzielące.

Strukturą zdarzeń będzie lista L uporządkowanych malejąco względem y-ów wierzchołków danego wielokąta P.

Strukturą stanu będzie wzbogacone, zrównoważone drzewo poszukiwań binarnych T przechowujące w liściach ciąg aktualnie przecinanych przez miotłę krawędzi ograniczających wielokąt P z lewej strony wraz z dowiązaniami do ich pomocników.



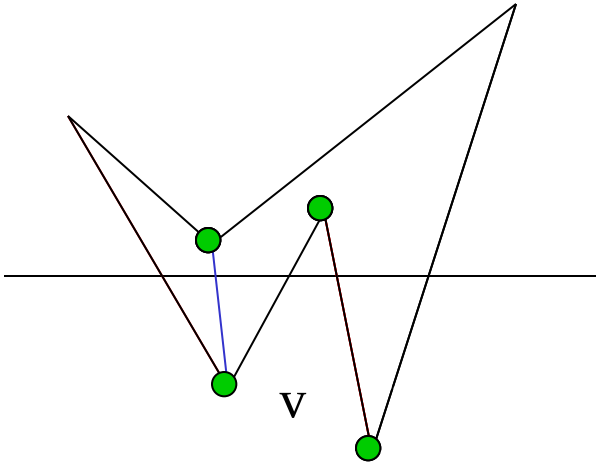
W zależności od rodzaju wierzchołka v, który odwiedza miotła wykonywane są następujące procedury. Niech e_l (e_p) oznacza lewą (prawą) krawędź o końcu w wierzchołku początkowym, końcowym, dzielącym lub łączącym. Niech e_v oznacza krawędź leżącą bezpośrednio na lewo od v.



Wierzchołek początkowy .

wstaw e_l do T;

pomocnik(e_l) := v;



Wierzchołek końcowy.

if pomocnik(e_l) jest wierzchołkiem łączącym

then wstaw przekątną miedzy v

i pomocnik(e_l);

usuń e_l z T;

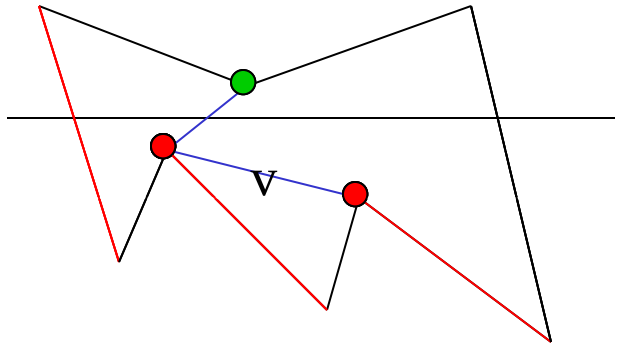
Wierzchołek dzielący.

znajdź w T krawędź e_v ;

wstaw przekątną między v i pomocnik(e_v);

pomocnik(e_v) := v;

wstaw e_p do T i pomocnik(e_p) := v;



Wierzchołek łączący.

if pomocnik(e_p) jest wierzchołkiem łączącym

then wstaw przekątną między v i pomocnik(e_p);

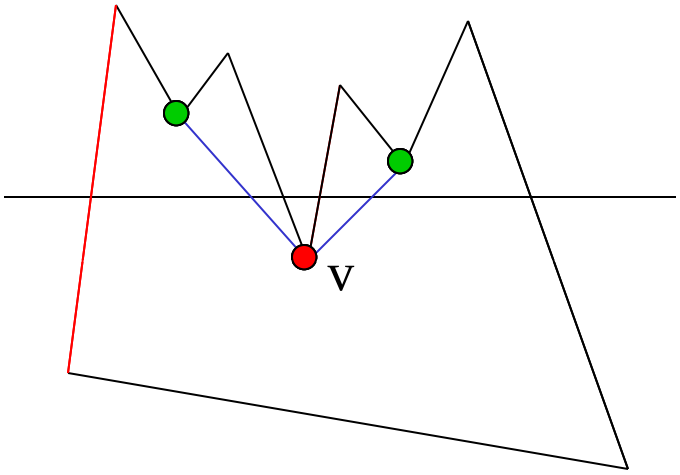
usuń e_p z T;

znajdź w T krawędź e_v ;

if pomocnik(e_v) jest wierzchołkiem łączącym

then wstaw przekątną między v i pomocnik(e_v);

pomocnik(e_v) := v;



Wierzchołek prawidłowy.

Niech e_g (e_d) będzie krawędzią powyżej (poniżej) v.

if wnętrze P leży na prawo od v

then if pomocnik(e_g) jest wierzchołkiem łączącym

then wstaw przekątną między v i pomocnik(e_g);

usuń e_g z T;

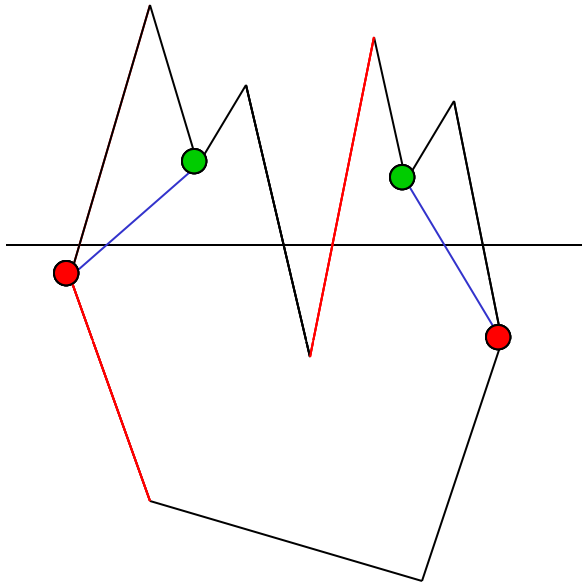
wstaw e_d do T i pomocnik(e_d) := v;

else znajdź w T krawędź e_v ;

if pomocnik(e_v) jest wierzchołkiem łączącym

then wstaw przekątną między v i pomocnik(e_v);

pomocnik(e_v) := v;



Algorytm podziału wielokąta P na wielokąty monotoniczne.

L := {p_1, p_2, ... , p_n}; T := puste ;

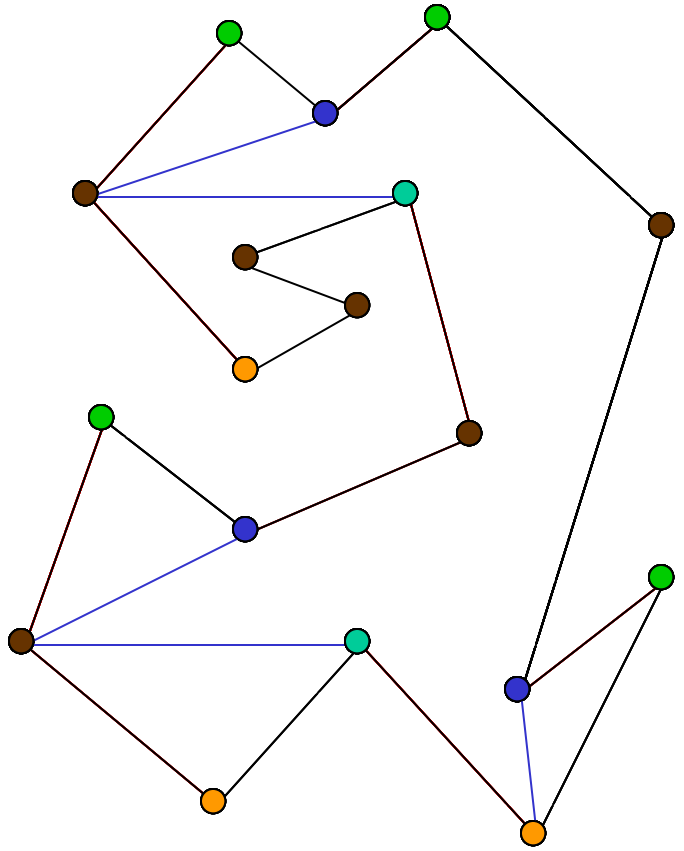
while L ≠ ∅ ***do***

v := FRONT(L);

POP(L);

wywołaj procedurę

odpowiadającą rodzajowi v;



Lemat.

Powyższy algorytm znajduje zbiór nieprzecinających się przekątnych, które dzielą wielokąt prosty P na wielokąty monotoniczne.

Dowód.

Algorytm likwiduje wierzchołki łączące i dzielące, więc otrzymujemy podział na wielokąty monotoniczne.

Wierzchołek przestaje być dzielący w momencie zamiecenia.

Wierzchołek łączący jest pomocnikiem najbliższej krawędzi z lewej strony. Przestaje taki być najpóźniej po osiągnięciu dolnego końca tej krawędzi (ale może wcześniej, gdy na prawo od krawędzi pojawi się inny wierzchołek).

Z definicji pomocnika wynika, że w chwili dodawania przekątnej między miotłą a odcinkiem łączącym go z odpowiadającą mu krawędzią nie ma innych elementów. Zatem można go połączyć z aktualnie badanym wierzchołkiem przekątną nie przecinającą się z żadną inną.

Lemat.

Prosty wielokąt o n wierzchołkach można powyższym algorytmem podzielić na y-monotoniczne wielokąty w czasie O(n log n) używając O(n) pamięci.

Dowód.

Sortowanie wierzchołków wymaga czasu O(n log n).

Przetworzenie jednego wierzchołka wymaga czasu logarytmicznego, więc zamiatanie wykonujemy w czasie O(n log n).

Wszystkie wykorzystywane struktury danych mają liniowy rozmiar.

Triangulacja otrzymanych wielokątów monotonicznych wymaga czasu O(n), więc dowolny wielokąt prosty można striangulować w czasie O(n log n).

Twierdzenie (Chazelle).

Dowolny wielokąt prosty można striangulować w czasie O(n).

Znajdowanie par przecinających się odcinków.

Problem: Dla zbioru n odcinków S na płaszczyźnie znajdź wszystkie pary przecinających się odcinków z S.

Problem przecinających się odcinków można łatwo rozwiązać w czasie $O(n^2)$. W tym celu wystarczy sprawdzić oddzielnie każdą parę odcinków. Takie rozwiązanie jest optymalne, gdy liczba par przecinających się odcinków jest kwadratowa.

Chcemy znaleźć algorytm wrażliwy na wynik, tzn. algorytm, którego złożoność będzie zależeć od rozmiaru rozwiązania.

Metoda zamiatania.

Miotła będzie zamiatać wzdłuż osi x-ów.

W każdym położeniu miotły odcinkami ***przetworzonymi*** nazywamy wszystkie odcinki, których końce znajdują się na lewo od niej. Odcinkami ***aktywnymi*** są odcinki aktualnie przecinające miotłę. Do zbioru odcinków ***oczekujących*** należą odcinki o obu końcach na prawo od miotły.

Będziemy korzystać z dwóch struktur danych.

Struktura zdarzeń Q jest zrównoważonym drzewem poszukiwań binarnych zawierającym uporządkowane rosnąco względem x-ów końce odcinków oraz punkty przecięć wszystkich par odcinków aktywnych, które kiedykolwiek były sąsiadami w strukturze stanu.

Struktura stanu T jest wzbogaconym, zrównoważonym drzewem poszukiwań binarnych przechowującym w liściach zbiór odcinków aktywnych uporządkowanych względem współrzędnych y-owych.

Po dojściu miotły do kolejnego punktu zdarzenia mamy trzy możliwości.

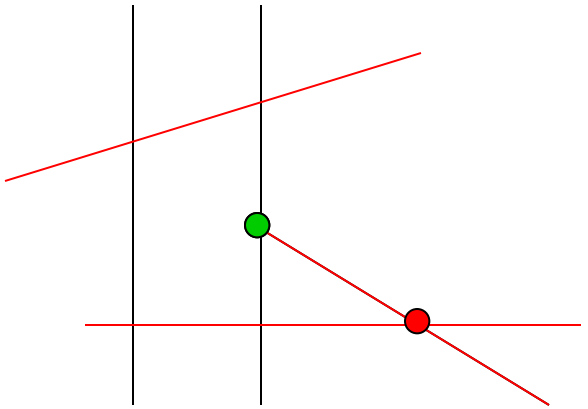
Zdarzenie jest początkiem odcinka s z S.

wstaw s do T; uaktualnij dowiązania;

if w jest sąsiadem s w T

  then if w przecina s w punkcie p

    then wstaw p do Q;



Zdarzenie jest końcem odcinka s z S.

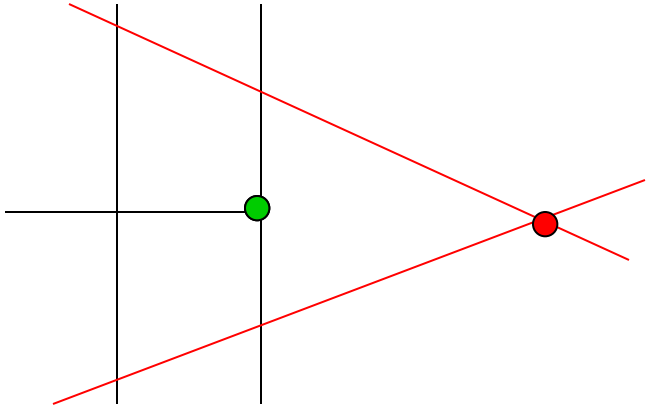
usuń s z T; uaktualnij dowiązania;

if s miał w T dwóch sąsiadów w, z

  then if w przecina z w punkcie p

    then wstaw p do Q, jeśli go tam

      jeszcze nie ma;



Zdarzenie jest punktem przecięcia odcinków s,z $\in$ S.

Zamień kolejność s i z w T;

if w jest sąsiadem s

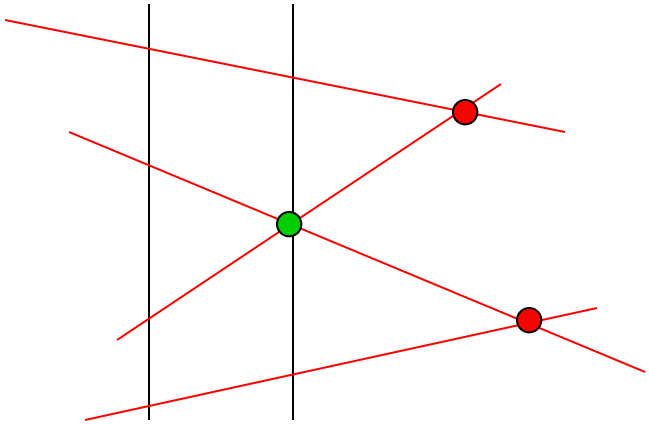
then if w przecina s w punkcie p

then wstaw p do Q , jeśli go tam jeszcze nie ma;

if v jest sąsiadem z

then if v przecina z w punkcie q

then wstaw q do Q , jeśli go tam jeszcze nie ma;

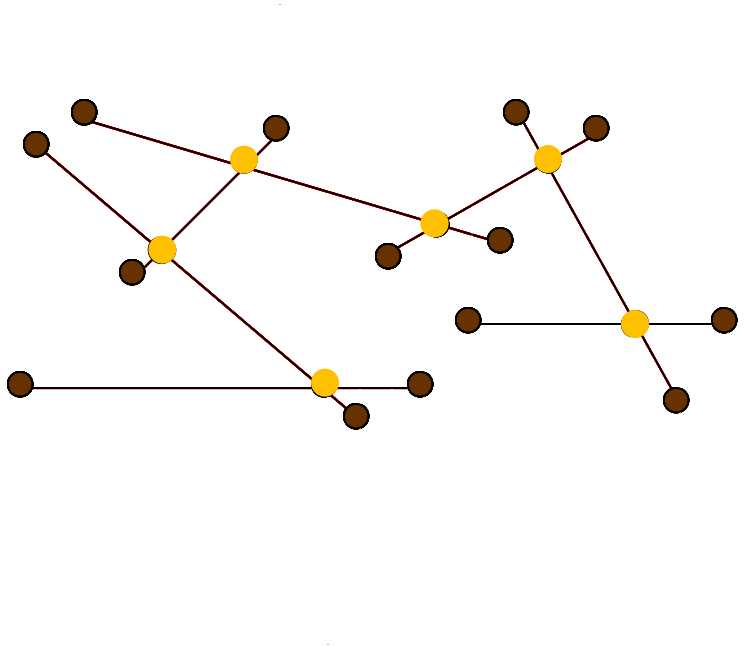


Lemat.

Punkt przecięcia dwóch aktywnych odcinków leżący po prawej stronie, najbliżej miotły znajduje się w strukturze Q.

Algorytm (Bentley-Ottmann).

wstaw do Q końce odcinków z S;
T := puste;
while Q pusty ***do***
    q := minimalny element w Q;
    usuń q z Q;
    wywołaj odpowiednią procedurę
    badającą q zależnie od jego
    rodzaju;



Twierdzenie.

Algorytm znajduje wszystkie przecięcia odcinków ze zbioru S w czasie $O((n+k) \log n)$, gdzie k jest liczbą przecięć.

Dowód.

Jeśli dwa odcinki przecinają się, to istnieje położenie miotły poprzedzające to zdarzenie, w którym przecinające się odcinki sąsiadują w strukturze T. Zatem każdy punkt przecięcia jest znajdywany.

Operacje wykonywane w dowolnym kroku algorytmu wymagają czasu logarytmicznego (stała liczba wstawień, usunięć, wyszukiwań elementów w zrównoważonym drzewie poszukiwań binarnych). Algorytm wykonuje $2n+k$ kroków. Zatem jego złożoność wynosi $O((n+k) \log n)$.

Wniosek (problem decyzyjny).

Znalezienie odpowiedzi na pytanie, czy w zbiorze S istnieje para przecinających się odcinków wymaga czasu $O(n \log n)$.

Szkic algorytmu Balabana.

Obszar, w którym występują odcinki podzielmy na pionowe pasy.

Porządkiem odcinków względem pionowej prostej nazywamy kolejność ich punktów przecięć z prostą.

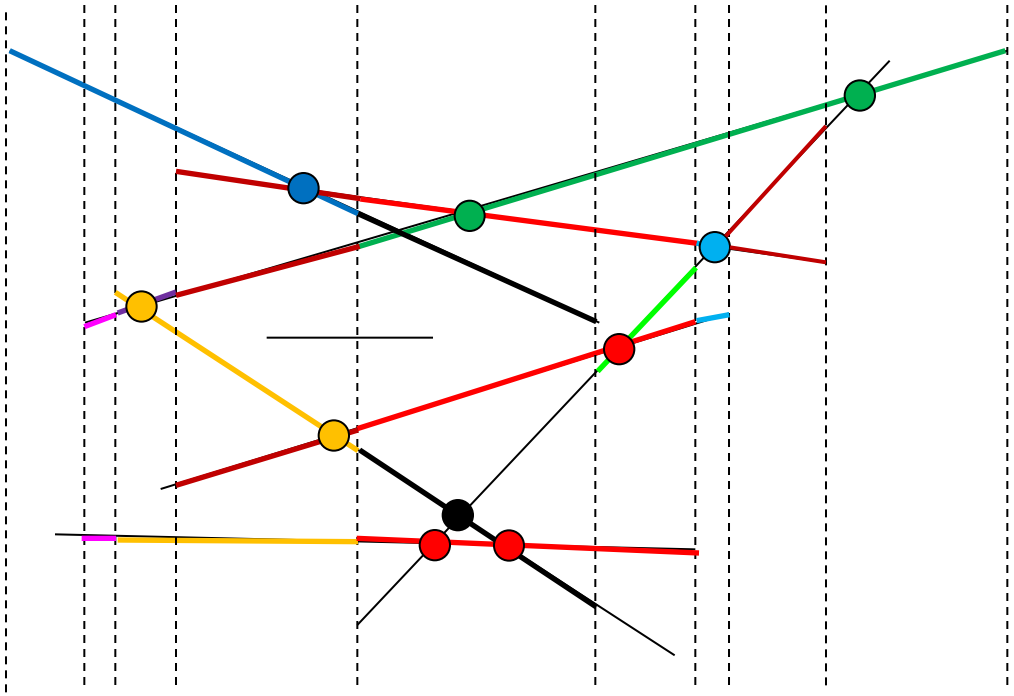
Wykorzystujemy strukturę drzewa binarnego do zapisu podziału na pasy.

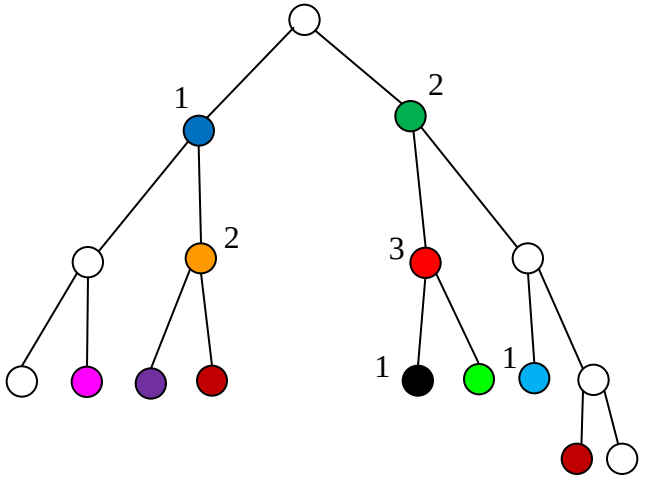
Schodami w danym pasie nazywamy maksymalny zbiór (niekoniecznie największy) nieprzecinających się części odcinków, z których każda łączy oba brzegi pasa.

Po określeniu schodów w danym pasie, dzielimy pozostałe odcinki względem mediany x-owych współrzędnych ich końców w pasie i znajdujemy schody w nowych pasach.

Po dojściu do liści (osobne końce odcinków) zawracamy i zliczamy w danym pasie punkty przecięć schodów z odcinkami analizowanymi na niższych poziomach (badamy przecięcia tych odcinków z coraz szerszymi pasami, więc każdy odcinek sprawdzamy co najwyżej logarytmicznie wiele razy) oraz dodajemy informację o przecięciach tych odcinków .

Przykład.





Łącznie 10 przecięć.

Twierdzenie.

Algorytm Balabana jest optymalnym algorytmem czułym na wynik.

Jego złożoność czasowa wynosi $O(n \log n + k)$ i wymaga $O(n)$ pamięci gdzie k oznacza liczbę przecięć odcinków.

Znajdowanie pary najbliższych punktów na płaszczyźnie.

Problem: W zbiorze S zawierającym n punktów na płaszczyźnie znajdź parę wyznaczającą odcinek o najmniejszej długości.

Problem ten można łatwo rozwiązać w czasie $O(n^2)$ badając wszystkie pary punktów.

Wykorzystamy algorytm zamiatania w celu znalezienia rozwiązania problemu w czasie O(n log n).

Miotła poruszać się będzie wzdłuż osi x-ów.

Punktami aktywnymi będziemy nazywać punkty z S znajdujące się w lewostronnie otwartym pasie X o szerokości równej najmniejszej odległości między dotychczas zamiecionymi punktami. Początkowo szerokość pasa jest nieskończona. Niech ix oznacza minimalny indeks punktu z X.

Strukturą zdarzeń jest uporządkowana lista L punktów z S.

Strukturą stanu będzie zrównoważone drzewo poszukiwań binarnych T przechowujące zbiór punktów aktywnych uporządkowanych względem współrzędnej y.

Lemat.

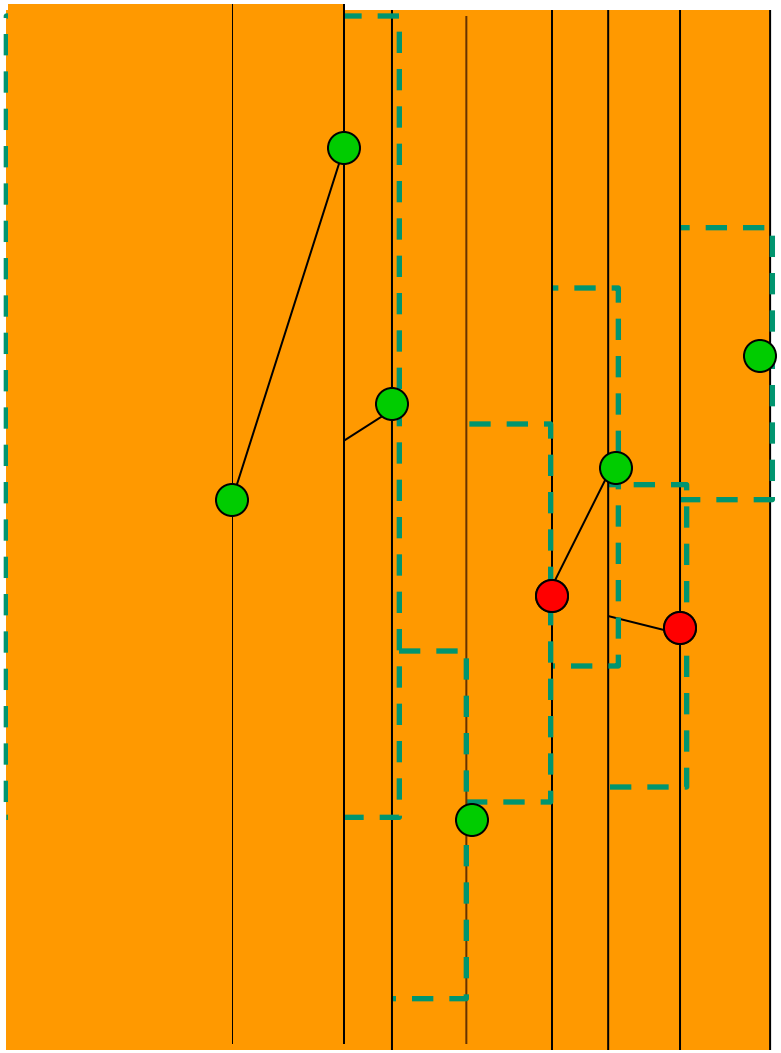
Punkty w pasie X można uporządkować względem współrzędnej y.

Dowód.

Ponieważ pas jest lewostronnie otwarty i ma szerokość równą najmniejszej odległości między dotychczas zamiecionymi punktami, więc żadne dwa punkty w pasie nie mogą mieć tej samej współrzędnej y-owej.

Algorytm.

L := {p_2, p_3, ... , p_n}; T := puste ;
dmin:= +∞; ix:= 1;
while L ≠ ∅ ***do***
    v := FRONT(L);
    POP(L);
    while d(ix,v) ≥ dmin ***do***
        T := T – {p_{ix}};
        ix:= ix+1;
    znajdź w T punkty odległe od v względem współrzędnej y o mniej niż dmin;
    w:= punkt w X najbliższy v;
    d:=d(v,w) (d := +∞ jeśli w nie istnieje);
    T := T ∪ {v};
    if d < dmin ***then***
        dmin := d;
        C := {v, w};



Lemat.

W każdym kroku algorytmu badamy co najwyżej 5 par punktów.

Twierdzenie.

Dla danego zbioru n punktów na płaszczyźnie najbliższą parę punktów możemy znaleźć w czasie O(n log n).

Dowód.

Strukturę T uaktualniamy co najwyżej O(n) razy. Kosztuje to O(n log n).

Dla każdego zdarzenia wykonujemy stałą liczbę operacji, z których każda kosztuje co najwyżej O(log n). Ponieważ zdarzeń jest n, więc złożoność algorytmu wynosi O(n log n).

Dziękuję za uwagę.

Ćwiczenia 3.

1. Udowodnij, że każdy wielokąt umożliwia triangulację, nawet jeśli ma dziury. Co można powiedzieć o liczbie trójkątów w triangulacji ?

2. Udowodnij lub zaprzecz: dla każdego wielokąta monotonicznego istnieje triangulacja, której graf dualny jest łańcuchem, tzn. każdy węzeł tego grafu ma stopień 2.

3. Podaj algorytm, który w czasie O(n log n) oblicza przekątną dzielącą prosty wielokąt o n wierzchołkach na dwa wielokąty z co najwyżej $\lfloor 2n/3 \rfloor$ +2 wierzchołkami w każdym z nich.

4. Niech P będzie prostym wielokątem o n wierzchołkach, który podzielono na monotoniczne części. Udowodnij, że suma rozmiarów tych części jest O(n).

5. Niech S będzie zbiorem n trójkątów na płaszczyźnie. Brzegi trójkątów są rozłączne, ale jest możliwe, że trójkąt leży całkowicie wewnątrz drugiego trójkąta. Niech P będzie zbiorem n punktów na płaszczyźnie. Podaj algorytm działający w czasie O(n log n), który znajduje punkty leżące poza trójkątami.

6. Niech S będzie zbiorem n rozłącznych trójkątów na płaszczyźnie. Chcemy znaleźć zbiór n-1 odcinków o następujących własnościach:

- każdy odcinek łączy punkty brzegowe dwóch trójkątów,
- odcinki nie przecinają się i ich wnętrza są rozłączne z trójkątami,
- ciągi odcinków łączą razem wszystkie trójkąty, tzn. z każdego trójkąta można dojść do dowolnego innego.

Stwórz algorytm działający w czasie O(n log n).

7. Mając dany ciąg kolejnych krawędzi wielokąta prostego oblicz w czasie liniowym jego pole.

8. Niech S będzie zbiorem n okręgów na płaszczyźnie. Opisz algorytm zamiatania obliczający wszystkie punkty przecięć okręgów. (okręgi mogą zawierać się jeden w drugim)

9. W każdym kroku algorytmu znajdowania najbliższej pary punktów badamy co najwyżej 5 par punktów.