Geometria obliczeniowa
Wykład 12

Planowanie ruchu

1. Najkrótsza ścieżka między dwoma punktami.
2. Znajdywanie ścieżki między dwoma punktami.
3. Ruch postępowy robota wielokątnego na płaszczyźnie.
4. Ruch z możliwością obrotów.

Algorytmy randomizowane

1. Programowanie liniowe w R^2.
2. Minimalne koło opisane na zbiorze punktów.
3. Lokalizacja punktu w siatce trapezów.

Planowanie ruchu – najkrótsza ścieżka między dwoma punktami.

Problem.

Na płaszczyźnie dany jest obszar D (ograniczony lub nie) z dziurami (będącymi zbiorami otwartymi) mający w sumie n krawędzi oraz dwa wybrane punkty s i t.

Znajdź najkrótszą ścieżkę łączącą s i t (o ile istnieje).

Lemat.

Najkrótsza ścieżka łącząca punkty s i t jest łamaną składającą się z krawędzi grafu widzialności dla zbioru krawędzi obszaru D oraz punktów s i t (tzn. najkrótsza ścieżka między punktami s i t może zmieniać kierunek jedynie w punktach będących wierzchołkami obszaru D).

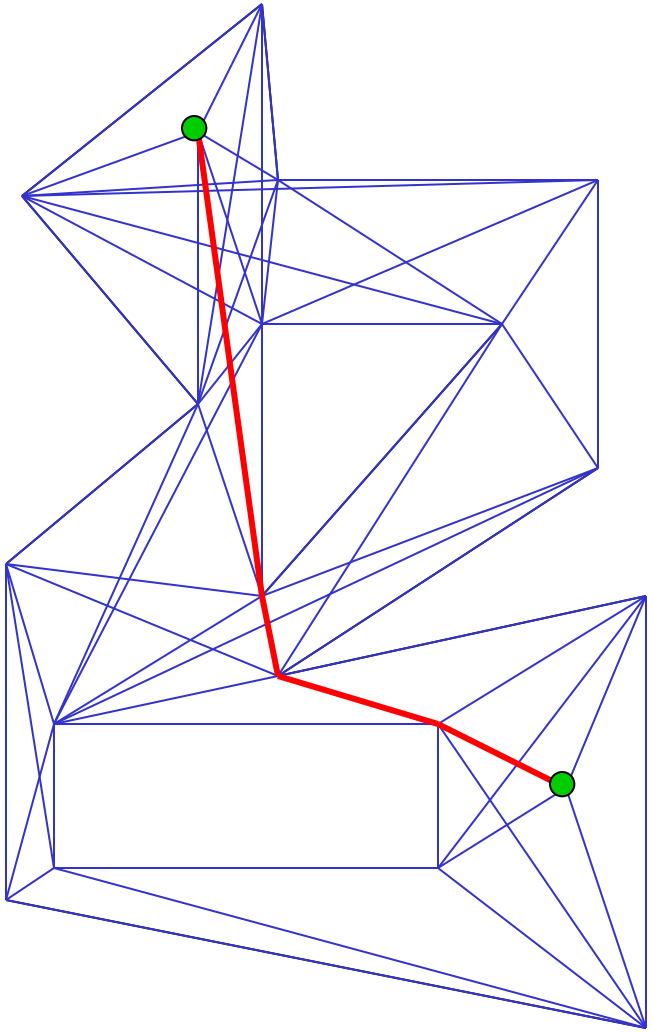
Algorytm.

stwórz graf widzialności W dla zbioru krawędzi obszaru D oraz punktów s i t;

if punkty s i t należą do tej samej spójnej składowej grafu W

then zastosuj algorytm Dijkstry w celu znalezienia najkrótszej ścieżki z s do t;

return najkrótsza ścieżka;



Twierdzenie.

Algorytm działa w czasie $O(n^2)$.

Dowód.

Graf W tworzymy w czasie $O(n^2)$. Spójne składowe sprawdzamy w tym samym czasie. Jeśli s i t są w tej samej składowej to najkrótszą ścieżkę znajdujemy w czasie $O(|V|\log|V|+|E|)$.

Twierdzenie (Hershberger, Suri 1997)

Ścieżkę o minimalnej długości między dwoma danymi punktami w obszarze z dziurami na płaszczyźnie, którego rozmiar wynosi n, można znaleźć w czasie O(n log n) wykorzystując pamięć rozmiaru O(n log n).

Znajdywanie ścieżki między dwoma punktami.

Problem.

Dany jest obszar z dziurami D na płaszczyźnie oraz dwa wyróżnione punkty s i t. Znajdź ścieżkę łączącą s i t (o ile istnieje).

Zakładamy, że dziury są zbiorami otwartymi.

Obszar D jest reprezentowany w postaci podwójnie łączonej listy krawędzi K.

Orientacja krawędzi obszaru D określa jego wnętrze i zewnętrze.

Mapa trapezowa.

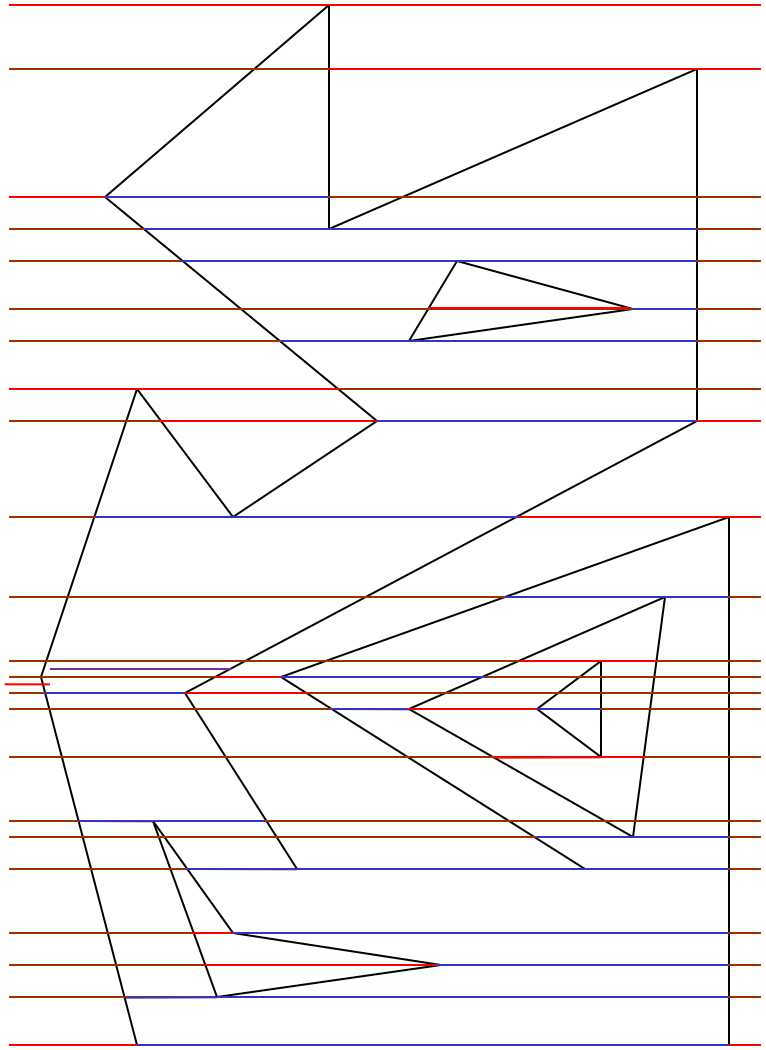
Wykorzystując metodę zamiatania dzielimy obszar D na trapezy o podstawach prostopadłych do kierunku zamiatania. Podział przechowujemy w strukturze K.

Strukturą zdarzeń będzie lista Q wierzchołków obszaru D uporządkowana względem kierunku zamiatania (np. osi y-ów).

Strukturą stanu będzie zrównoważone drzewo poszukiwań binarnych T zawierające aktywne krawędzie obszaru D uporządkowane względem kolejności ich przecięć z miotłą (wzdłuż osi x-ów).

Algorytm.

stwórz listę Q;
while Q ≠ ∅ ***do***
    q := POP(Q);
    usuń z T krawędzie, których końcem
    jest q i są za miotłą;
    znajdź w T sąsiadów q;
    for każdy sąsiad q w T ***do***
        if krawędź łącząca q z sąsiadem
            należy do obszaru
            then uaktualnij strukturę K;
    dodaj do T krawędzie, których koń-
    cem jest q i są przed miotłą;
return K;



Lemat.

Mapę trapezową wielokąta z dziurami o rozmiarze n można skonstruować w czasie O(n log n) i pamięci O(n).

Dowód.

W danym położeniu miotły operacja znalezienia sąsiadów badanego punktu w T wymaga czasu O(log n). Sprawdzenie należenia badanej podstawy trapezu do wnętrza obszaru D oraz aktualizacja struktury K wymagają czasu stałego.

Liczba nowych wierzchołków i krawędzi w mapie trapezowej jest proporcjonalna do rozmiaru obszaru D. Rozmiary struktur zdarzeń i stanu są liniowe względem rozmiaru obszaru D.

Mapa drogowa.

Na podstawie mapy trapezowej tworzymy graf, który umożliwi nam stwierdzenie, czy dane dwa punkty p_1, p_2 można połączyć ścieżką.

Algorytm.

stwórz mapę trapezową;

for każdy trapez ***do***

wyznacz punkt wewnątrz trapezu i

połącz środki krawędzi tworzących

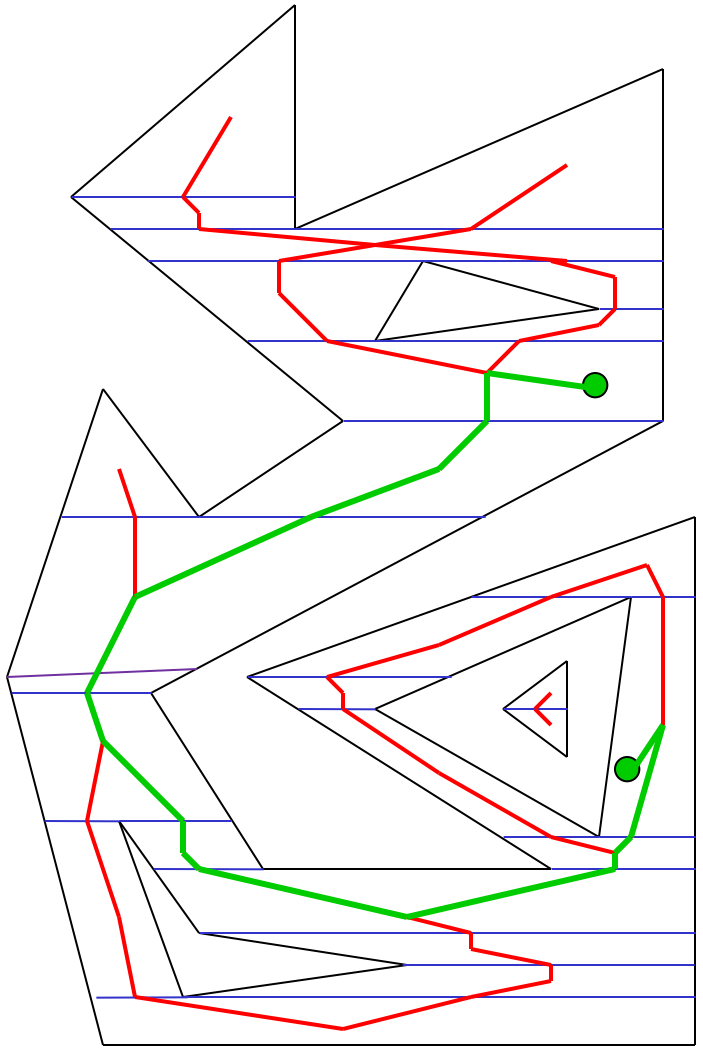
podstawy trapezu z tym punktem;

for i=1 ***to*** 2 ***do***

połącz p_i z wyznaczonym punktem

wewnątrz trapezu zawierającego p_i ;

sprawdź, czy w powstałym grafie p_1 i p_2 należą do tej samej spójnej składowej;



Lemat.

Mapę drogową w obszarze z dziurami o rozmiarze n możemy obliczyć w czasie O(n log n) i wymaga ona O(n) pamięci.

Dowód.

Mapę trapezową obliczamy w czasie O(n log n). Korzystając z podwójnie łączonych list krawędzi tworzymy mapę drogową w czasie proporcjonalnym do rozmiaru mapy trapezowej, czyli O(n). Zatem jej rozmiar jest również O(n).

Twierdzenie.

Ścieżkę między dwoma danymi punktami w obszarze z dziurami o rozmiarze n możemy znaleźć lub stwierdzić jej brak w czasie O(n log n).

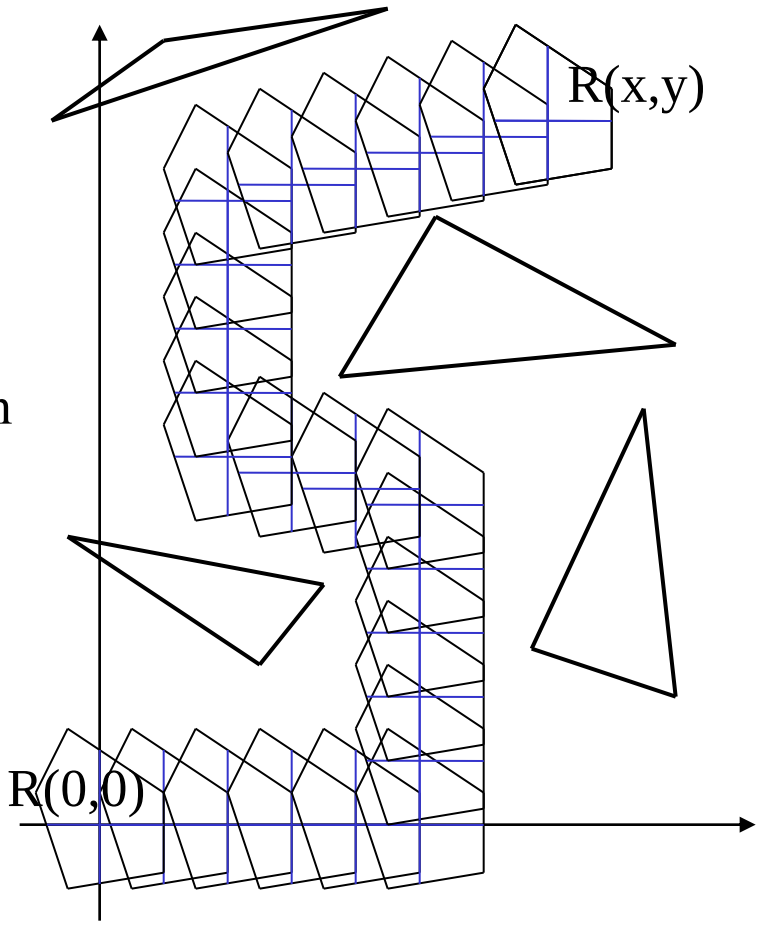
Dowód.

W czasie O(n log n) tworzymy mapę drogową. Zbadanie jej spójnych składowych i sprawdzenie, czy dane punkty należą do tej samej składowej, wymaga czasu O(n).

Ruch postępowy robota wielokątnego na płaszczyźnie.

Problem.

Dany jest robot, którego rzut na podłoże jest wielokątem (zazwyczaj wypukłym). Czy może on przemieścić się ruchem postępowym między dwoma danymi punktami w obszarze z przeszkodami, tzn. każdy z punktów robota porusza się po takim samym torze w tym samym czasie (robot nie obraca się) ?



Definicja.

Załóżmy, że środek układu współrzędnych należy do wnętrza robota.

Ruch tego punktu opisuje ruch całego robota. Nazywamy go ***punktem odniesienia.***

Niech R(0,0) oznacza listę położeń wierzchołków robota w chwili startu.

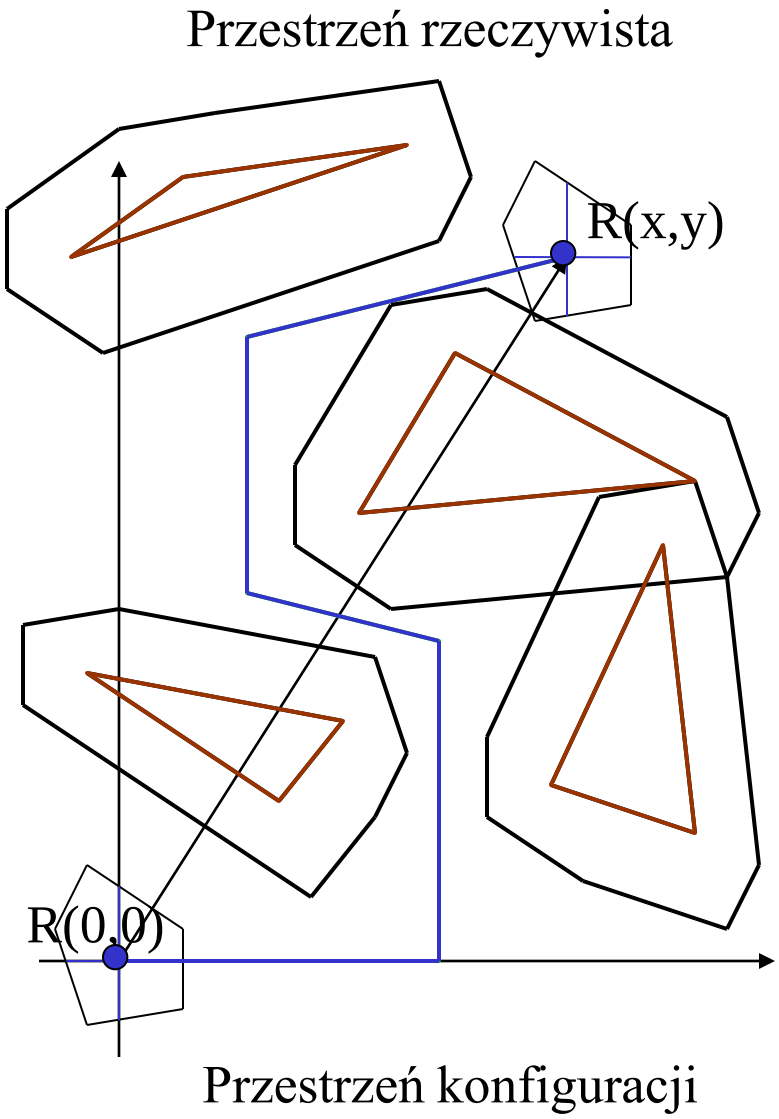
Wtedy R(x,y) := R(0,0) + (x,y)

Definicja.

Przestrzeń, w której porusza się robot nazywamy ***przestrzenią rzeczywistą.***

Przestrzeń parametrów robota nazywamy ***przestrzenią konfiguracji.*** W przestrzeni tej robot przyjmuje postać jednopunktową odpowiadającą jego punktowi odniesienia.

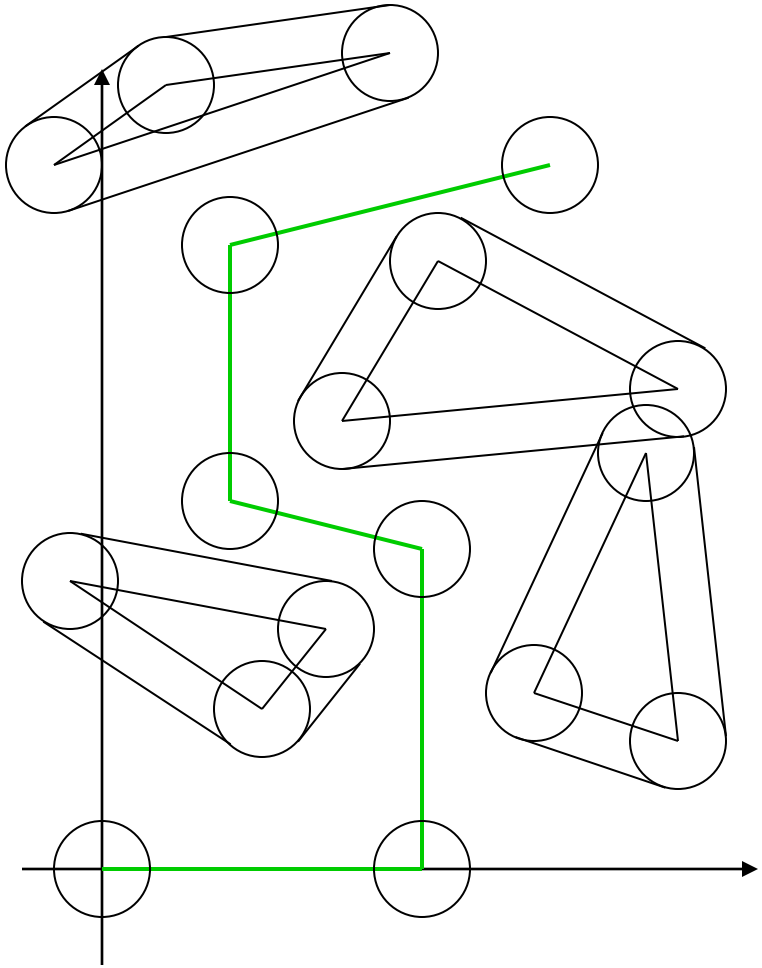
Jak wyglądają przeszkody (obszary zabronione) w przestrzeni konfiguracji ?



Przykład.

Gdy przesuwany obiekt jest kołem, to przeszkody należy rozszerzyć w każym kierunku o promień koła.

Gdy środek koła będzie leżeć poza powiększonymi przeszkodami, to jego brzeg nie będzie nachodzić na żadną z rzeczywistych przeszkód.

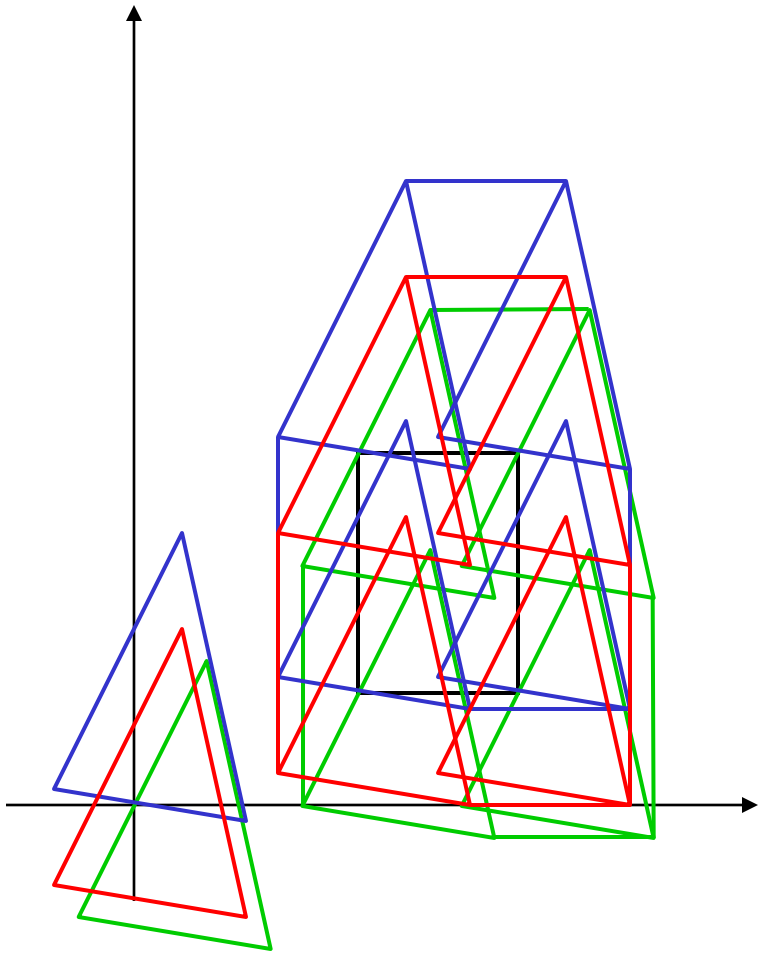


Suma Minkowskiego.

Definicja.

Rozpatrzmy wielokąty A i B jako zbiory wektorów o współrzędnych odpowiadających współrzędnym punktów należących do tych wielokątów. Sumą Minkowskiego wielokątów A i B jest

$A + B := \{x + y : x \in A \wedge y \in B\}$.



Fakt.

Suma Minkowskiego dwóch wielokątów zależy od ich położenia, ale jej kształt nie.

Załóżmy, że w położeniu początkowym środek układu współrzędnych znajduje się wewnątrz robota.

Fakt.

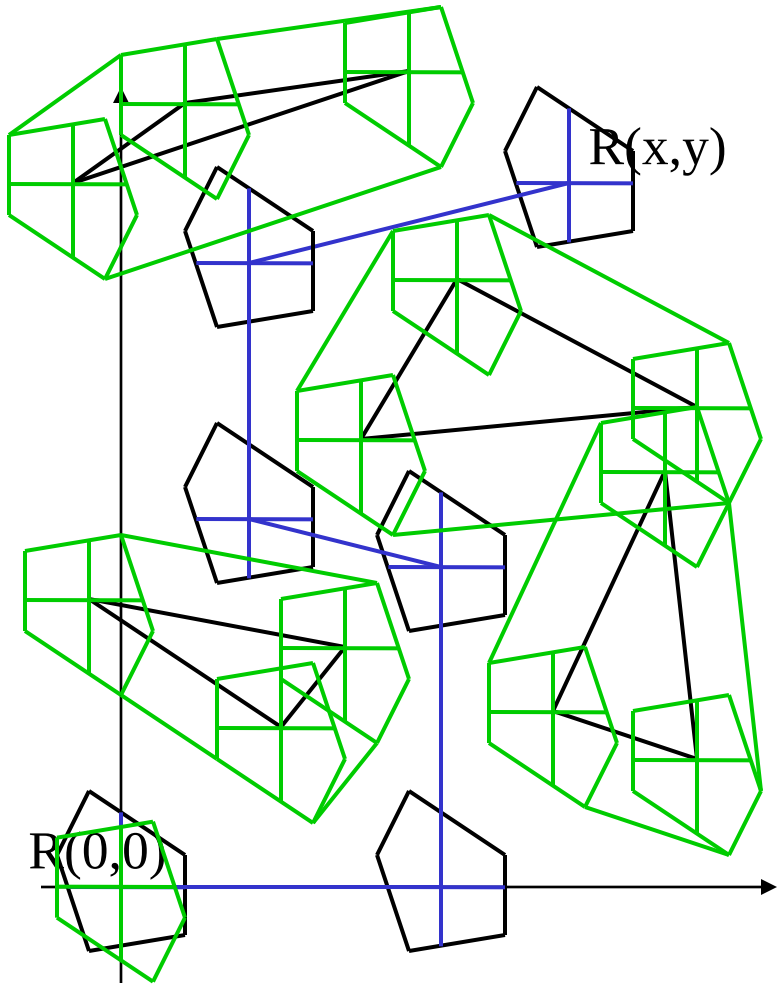
Zauważmy, że jeśli obiekt styka się z brzegiem obszaru lub dziury, to punkt odniesienia obiektu jest odległy od punktu styczności o wektor przeciwny do wektora łączącego punkt odniesienia z punktem styczności.

Zatem powiększone przeszkody określane są przez sumę Minkowskiego D-R.

Algorytm.

oblicz sumę Minkowskiego D-R;

sprawdź, czy istnieje ścieżka między położeniem początkowym i końcowym;



Złożoność algorytmu zależy od kształtu robota i przeszkód.

Załóżmy, że robot R ma stałą liczbę wierzchołków, a obszar D ma n wierzchołków.

R	D	Rozmiar sumy	Złożoność czasowa konstrukcji
wypukły	wypukły	O(n)	O(n)
wypukły	niewypukły	O(n)	O(n log n)
niewypukły	niewypukły	$O(n^2)$	$O(n^2 \log n)$

Twierdzenie.

W obszarze D z dziurami o łącznie n wierzchołkach czas znalezienia ścieżki, wzdłuż której można przesunąć wypukłego robota R o stałym rozmiarze między dwoma danymi położeniami bez kolizji z żadną przeszkodą (lub stwierdzić, że jest to niemożliwe), wynosi O(n log n).

Dowód.

Sumę Minkowskiego D-R znajdujemy w czasie O(n log n). Następnie znajdujemy ścieżkę między danymi punktami (jeśli istnieje) w czasie O(n log n).

Twierdzenie.

Niech D i R będą wielokątami o odpowiednio n i m wierzchołkach. Złożoność sumy Minkowskiego wielokątów D i R ma następujące ograniczenia:

- $O(n+m)$, jeśli oba wielokąty są wypukłe,
- $O(nm)$, jeśli jeden z wielokątów jest wypukły, a drugi nie,
- $O(n^2m^2)$, jeśli oba wielokąty nie są wypukłe.

Ograniczenia są ścisłe w pesymistycznym przypadku.

Ruch z możliwością obrotów.

W przypadku poruszania się robota, który może się obracać, mamy jeden stopień swobody więcej. Dlatego przestrzeń konfiguracji jest trójwymiarowa (trzeci wymiar odpowiada kątowi obrotu).

Sposób planowania ruchu robota jest podobny jak w poprzednim przypadku, choć nieco bardziej skomplikowany (ściany w przestrzeni konfiguracji nie są płaskie i wygodniej jest je przybliżyć, poza tym wraz ze wzrostem wymiaru rośnie też złożoność obliczeniowa zadania).

Twierdzenie (Canny, 1987)

Problem planowania ruchu, w którym robot ma d stopni swobody można rozwiązać w czasie $O(n^d \log n)$.

Podstawowe metody stosowane w algorytmach randomizowanych.

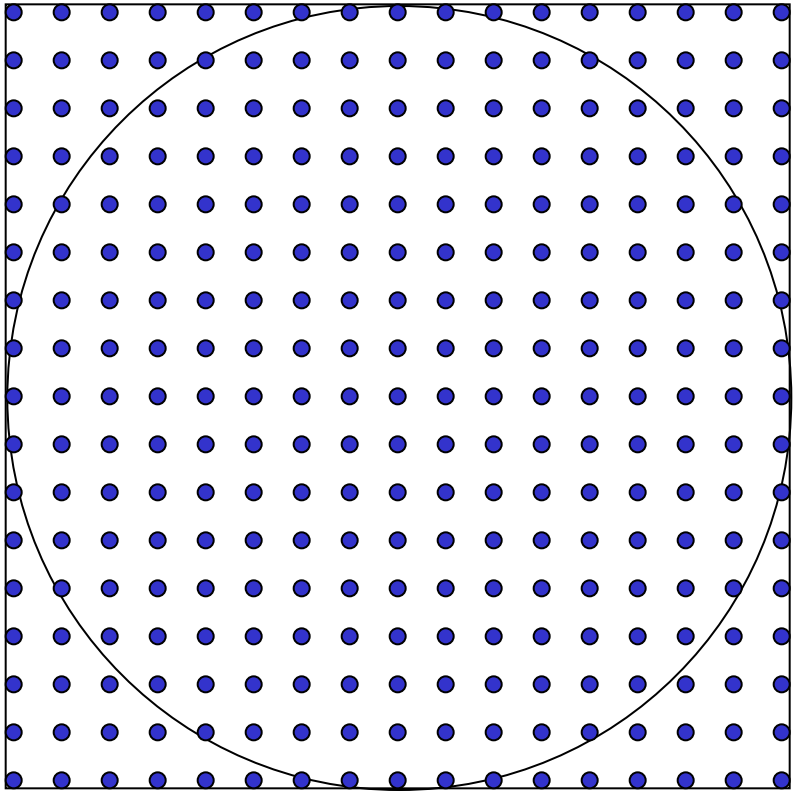
Metoda Las Vegas.
Metoda ta zawsze daje poprawne rozwiązanie. Natomiast czas trwania algorytmu może być zmienny w zależności od układu danych. Przykładem takiego algorytmu może być np. Quickhull lub dowolny algorytm przyrostowy z losowo uporządkowanymi danymi.

Metoda Monte Carlo.
Metoda ta zawsze kończy się w ustalonym czasie, ale może z pewnym prawdopodobieństwem zwrócić zły wynik bądź zwraca wynik tylko z pewną dokładnością. Przykładem takiego algorytmu może być np. szukanie ścieżki łączącej dwa punkty w labiryncie przy losowym wyborze kierunku poruszania się lub losowe próby lokalizacji punktu na płaszczyźnie podzielonej na obszary.

Formalnie efektywność takiego algorytmu jest zmienną losową określoną na zbiorze możliwych losowych ciągów danych. Wartość oczekiwana takiej zmiennej nazywana jest ***oczekiwanym czasem działania***.

Przykład.

Pole koła o promieniu 1 (wartość π).



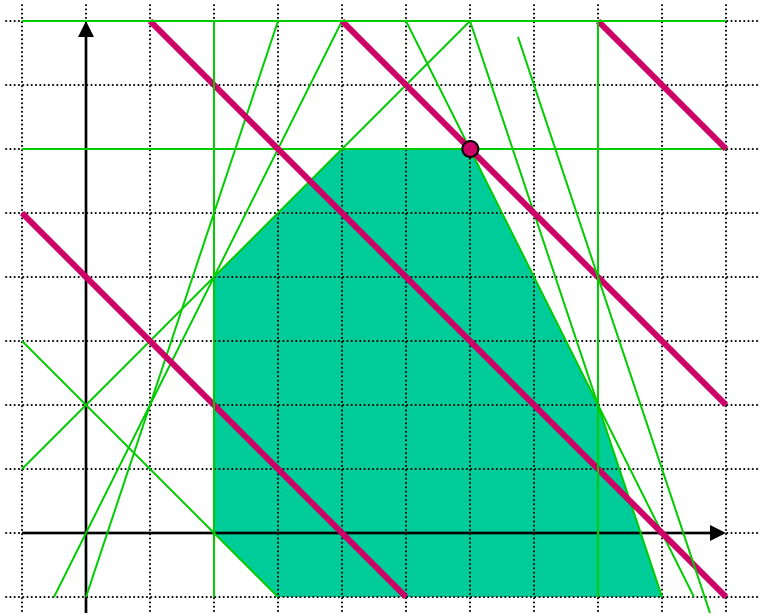
$4 \times 21/27 = 3{,}111$

$4 \times 221/289 = 3{,}059$

Programowanie liniowe w R^2.

Problem.

Dana jest liniowa funkcja celu c (postaci ax+by) oraz n warunków brzegowych (zbiór H nierówności liniowych $a_i x + b_i y + c_i \leq 0$, dla i = 1,2, ... , n) opisujących obszar dopuszczalny D, w którym poszukujemy punktu maksymalizującego (minimalizującego) wartość funkcji c.



Algorytm

if obszar wyznaczany przez półpłaszczyzny z H jest ograniczony w kierunku wzrostu wartości funkcji c i nie jest pusty

then określ półpłaszczyzny h_1, h_2 ograniczające wzrost wartości funkcji c, punkt przecięcia ich brzegów $x := \delta(h_1) \cap \delta(h_2)$ i $D := h_1 \cap h_2$

else return(„Brak rozwiązania”);

for $h \in H-\{h_1,h_2\}$ ***do***

if $x \notin h$ **then** $D := D \cap h$;

if $D \neq \varnothing$

then $x := \{q: c(q) = \max_{p \in \delta(h) \cap D} c(p)\}$

else return(„Brak rozwiązania”);

return(x, c(x));

Funkcja celu:

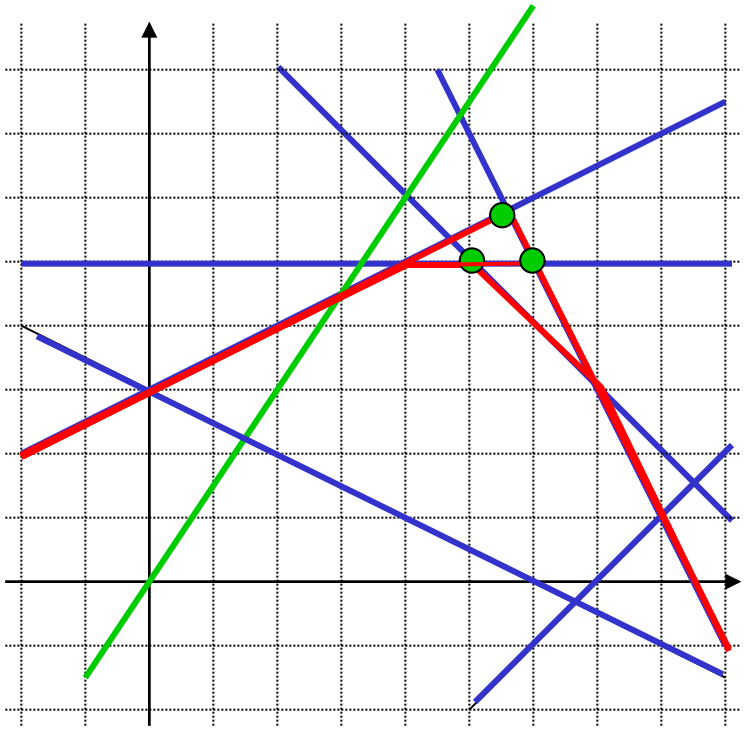
max 3x+2y

Warunki brzegowe:

$y - 0{,}5x - 3 \leq 0$ $\quad y - 5 \leq 0$

$y + 2x - 17 \leq 0$ $\quad -y + 0{,}5x - 3 \leq 0$

$-y + x - 7 \leq 0$ $\quad y + x - 10 \leq 0$



Twierdzenie

Problem programowania liniowego w R^2 dla n warunków brzegowych można rozwiązać w czasie oczekiwanym O(n) z wykorzystaniem O(n) pamięci.

Dowód.

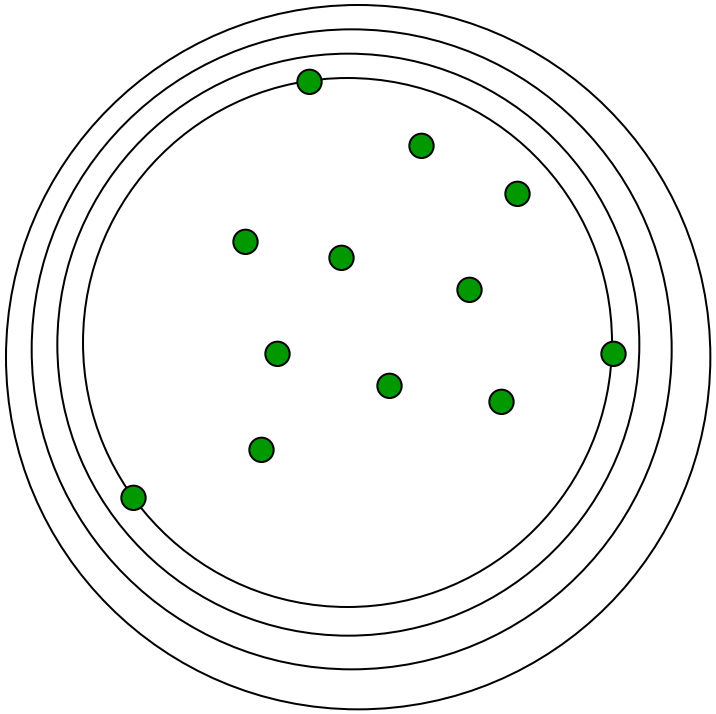
Niech v_k będzie punktem, w którym funkcja c przyjmuje maksimum (minimum) w obszarze wyznaczanym przez k półpłaszczyzn, a h_k będzie k-tą półpłaszczyzną.

Zdefiniujmy zmienną losową X_k przyjmującą wartość 1, gdy $v_{k-1} \notin h_k$ oraz 0 w przeciwnym przypadku.

Ponieważ znalezienie nowego maksimum (rozwiązanie problemu jednowymiarowego programowania liniowego), wymaga czasu O(n), więc oczekiwany czas działania algorytmu wynosi $E(\Sigma^n_{k=3} O(k)X_k) = \Sigma^n_{k=3} O(k)E(X_k)$.

Wartość $E(X_k)$ szacujemy stosując tzw. analizę wsteczną. W tym celu badamy, jak zmienia się problem, gdy mając k warunków brzegowych, odejmiemy jedną z półpłaszczyzn. Punkt, w którym funkcja c ma maksimum (minimum) może ulec zmianie, gdy odejmiemy co najwyżej dwie spośród półpłaszczyzn wyznaczających obszar (wyznaczają one wierzchołek, w którym przyjmowane jest maksimum (minimum) - jeśli przez ten punkt przechodzi więcej prostych będących brzegami danych półpłaszczyzn, to może się zdarzyć, że usunięcie dowolnej półpłaszczyzny nie wpływa na wartość funkcji celu). Zatem $E(X_k) \leq 2/(k-2)$. Stąd $\Sigma^n_{k=3} O(k)E(X_k) = \Sigma^n_{k=3} O(1) = O(n)$.

Minimalne koło opisane na zbiorze punktów.



Problem.

Dla danego zbioru punktów na płaszczyźnie P znajdź minimalne koło zawierające wszystkie punkty z P.

Weźmy losową permutację punktów z P.

Niech $P_i := \{p_1, \ldots, p_i\}$ a D_i będzie minimalnym kołem zawierającym P_i.

Lemat .

Dla $2 < i < n$ mamy:

- jeśli $p_i \in D_{i-1}$, to $D_i = D_{i-1}$,
- jeśli $p_i \notin D_{i-1}$, to p_i leży na brzegu D_i.

Załóżmy, że znane są dwa punkty z P
leżące na brzegu minimalnego koła.

MinDisc2P(P, q_1, q_2)

D_0 := najmniejsze koło z q_1, q_2 na brzegu;

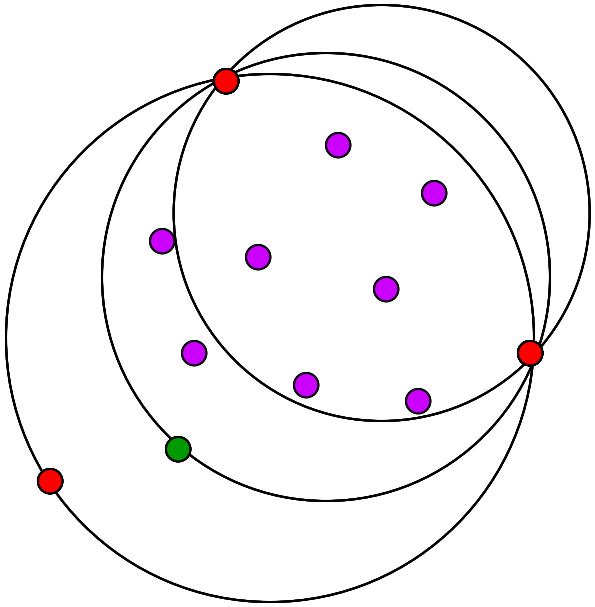
for k:=1 ***to*** n ***do***

if $p_k \in D_{k-1}$

  then $D_k := D_{k-1}$

  else D_k := koło z q_1, q_2, p_k na brzegu;

return D_n;



Załóżmy teraz, że znamy jeden punkt z P leżący na brzegu minimalnego koła.

MinDisc1P(P, q)

Oblicz losową permutację $p_1, \ldots, p_n$ punktów z P;

D_1 := najmniejsze koło z p_1, q na brzegu;

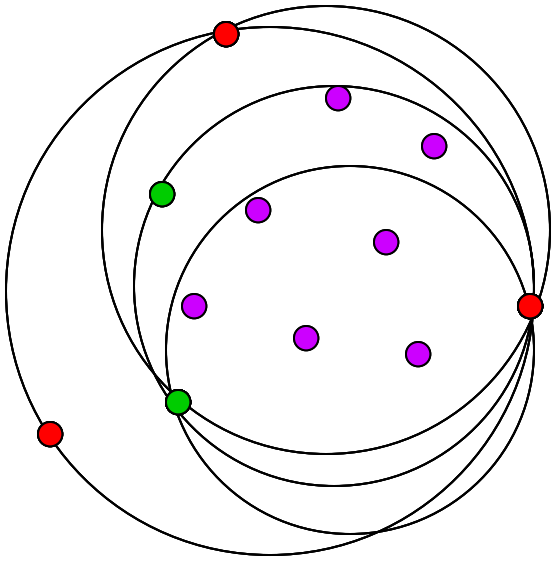
for j:=2 ***to*** n ***do***

if $p_j \in D_{j-1}$

then $D_j := D_{j-1}$

else D_j := ***MinDisc2P***($\{p_1,\ldots, p_{j-1}\}$, p_j, q);

return D_n;



Ostatecznie nasz program wygląda następująco.

MinDisc(P)

Oblicz losową permutację $p_1, \ldots, p_n$ punktów z P;

D_2 := najmniejsze koło zawierające $\{p_1, p_2\}$;

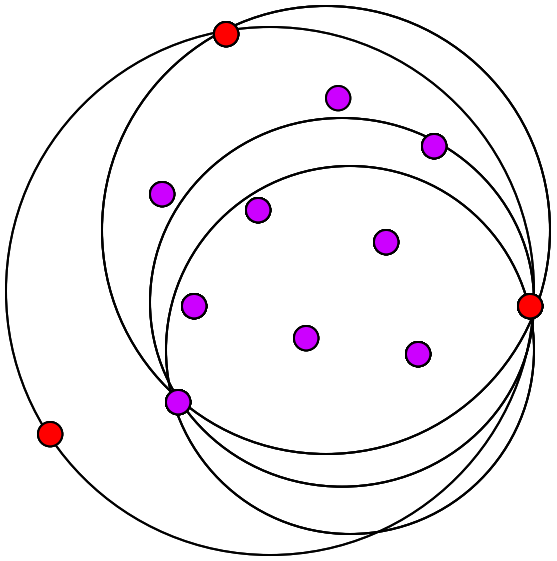
for i:=3 ***to*** n ***do***

if $p_i \in D_{i-1}$

then $D_i := D_{i-1}$

else D_i := ***MinDisc1P***($\{p_1,\ldots, p_{i-1}\}, p_i$);

return D_n;



Lemat.

Niech P będzie zbiorem punktów w R^2, a Q, być może pustym, zbiorem punktów w R^2, dla których $P \cap Q = \emptyset$ oraz $p \in P$. Wtedy zachodzą następujące warunki:

- Jeśli istnieje koło, które zawiera P i wszystkie punkty z Q są na jego brzegu, to najmniejsze takie koło jest wyznaczone jednoznacznie (oznaczmy je przez ***md(P,Q)***).
- Jeśli $p \in$ ***md(P-{p},Q)***, to ***md(P,Q)*** = ***md(P-{p},Q)***.
- Jeśli $p \notin$ ***md(P-{p},Q)***, to ***md(P,Q)*** = ***md(P-{p},Q$\cup${p})***.

Wniosek.

MinDisc poprawnie oblicza minimalne koło zawierające zbiór punktów.

Twierdzenie.

Minimalne koło zawierające zbiór n punktów na płaszczyźnie można obliczyć w oczekiwanym czasie O(n), używając w pesymistycznym przypadku pamięć rozmiaru O(n).

Dowód.

Oszacowanie rozmiaru pamięci wynika z faktu, że wszystkie trzy algorytmy ***MinDisc*** , ***MinDisc1P*** i ***MinDisc2P*** potrzebują O(n) pamięci.

Czas działania ***MinDisc2P*** wynosi O(n). ***MinDisc1P*** działa w czasie liniowym dopóki nie zachodzi konieczność wywołania ***MinDisc2P***. Prawdopodobieństwo takiego zdarzenia liczymy wykorzystując analizę wsteczną. Ustalmy zbiór $\{p_1,..., p_i\}$. Niech D_i będzie minimalnym kołem zawierającym $\{p_1,..., p_i\}$, które ma q na swoim brzegu. Załóżmy, że usuwamy jeden z punktów z $\{p_1,..., p_i\}$. Koło D_i ulega zmianie, gdy usuwamy jeden z trzech (lub dwóch) punktów na brzegu (gdy punktów jest więcej, usunięcie niektórych z nich nic nie zmienia). Jednym z tych punktów jest q, więc usunięcie co najwyżej dwóch punktów powoduje zmniejszenie koła. Prawdopodobieństwo, że p_i jest jednym z tych punktów wynosi 2/i. Zatem możemy oszacować oczekiwany czas działania ***MinDisc1P*** przez

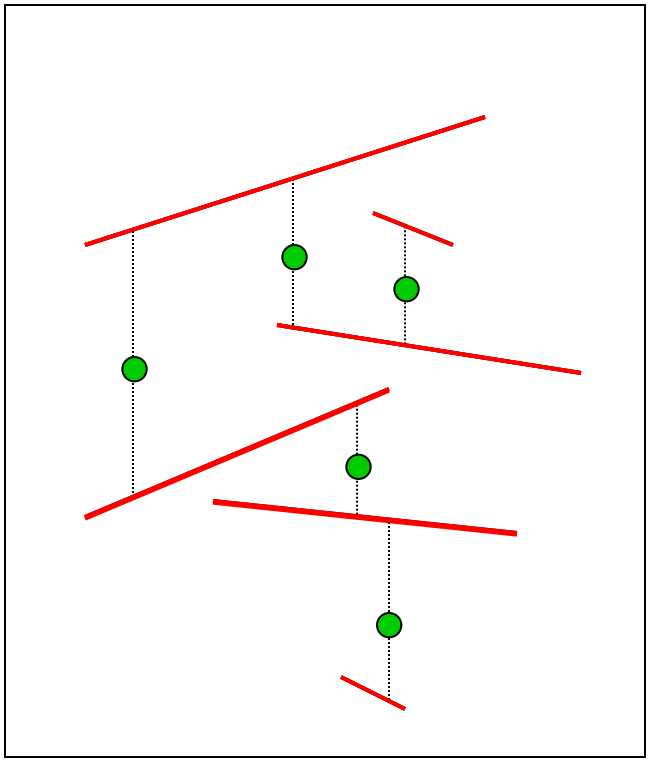
$O(n) + \Sigma^{n}_{i=2} O(i) \times 2/i = O(n)$.

Stosując identyczne rozumowanie dla ***MinDisc*** stwierdzamy, że oczekiwany czas działania tego algorytmu wynosi O(n).

Lokalizacja punktu w siatce trapezów.

Problem

Dany jest prostokątny obszar oraz n zawartych w nim rozłącznych odcinków. Odcinki są w położeniu ogólnym, tzn. żadne dwa końce nie mają tej samej współrzędnej x-owej ani y-owej (w szczególności, żaden nie jest pionowy ani poziomy).



Chcemy odpowiadać na pytanie: Miedzy którymi dwoma odcinkami (od góry i od dołu) znajduje się dany punkt ?

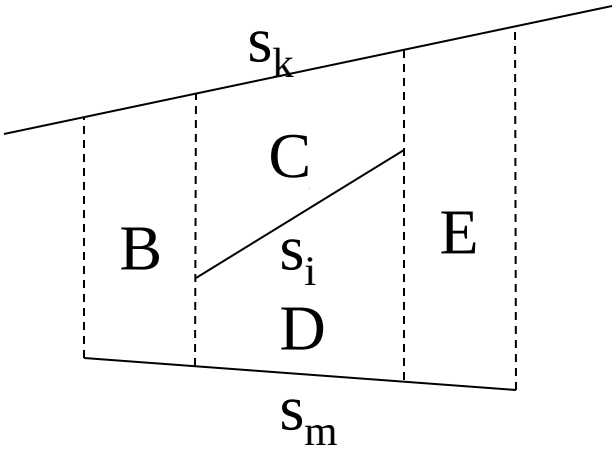
Przedstawimy algorytm przyrostowy znajdujący podział obszaru na trapezy (mapę trapezową).

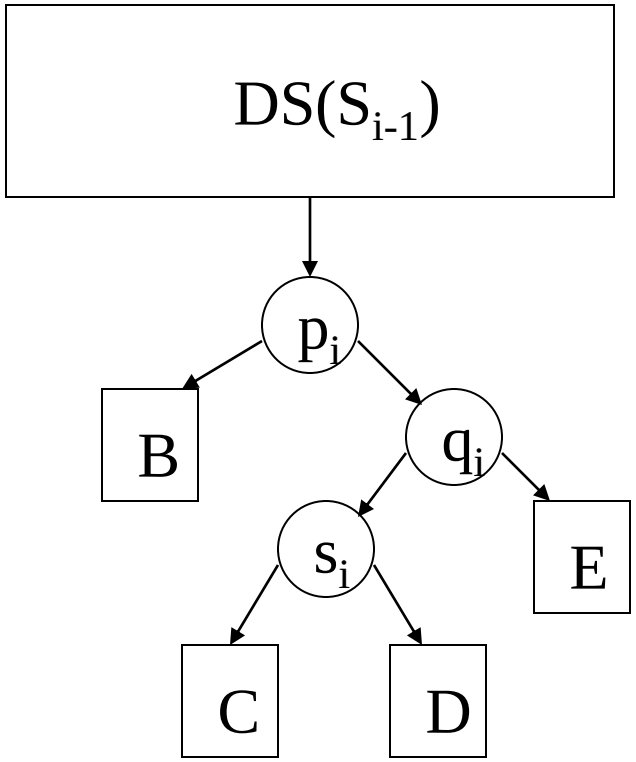
Równocześnie skonstruujemy strukturę danych umożliwiającą odpowiedź na zapytania o położenie punktów, wykorzystującą podział obszaru na trapezy .

Struktura jest grafem skierowanym, którego wierzchołki odpowiadają trapezom podziału, końcom odcinków i samym odcinkom. Wierzchołki odpowiadające odcinkom mogą występować wielokrotnie. Wierzchołki odpowiadające trapezom są liśćmi (wierzchołkami o zerowym stopniu wyjściowym).

Niech p_i i q_i oznaczają odpowiednio początek i koniec i-tego odcinka s_i.

Gdy odcinek s_i zawiera się w jednym z już istniejących trapezów, to w miejsce odpowiadającego mu wierzchołka wstawiamy wierzchołek p_i, którego lewym synem jest wierzchołek odpowiadający trapezowi powstającemu po lewej stronie p_i a prawym synem jest q_i. Prawym synem q_i jest wierzchołek odpowiadający trapezowi powstającemu po prawej stronie q_i a lewym synem jest s_i. Lewy i prawy syn s_i odpowiadają odpowiednio trapezowi powyżej i poniżej odcinka s_i.

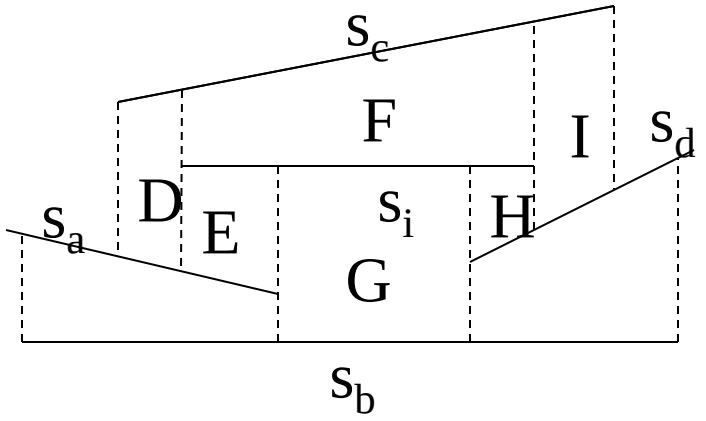


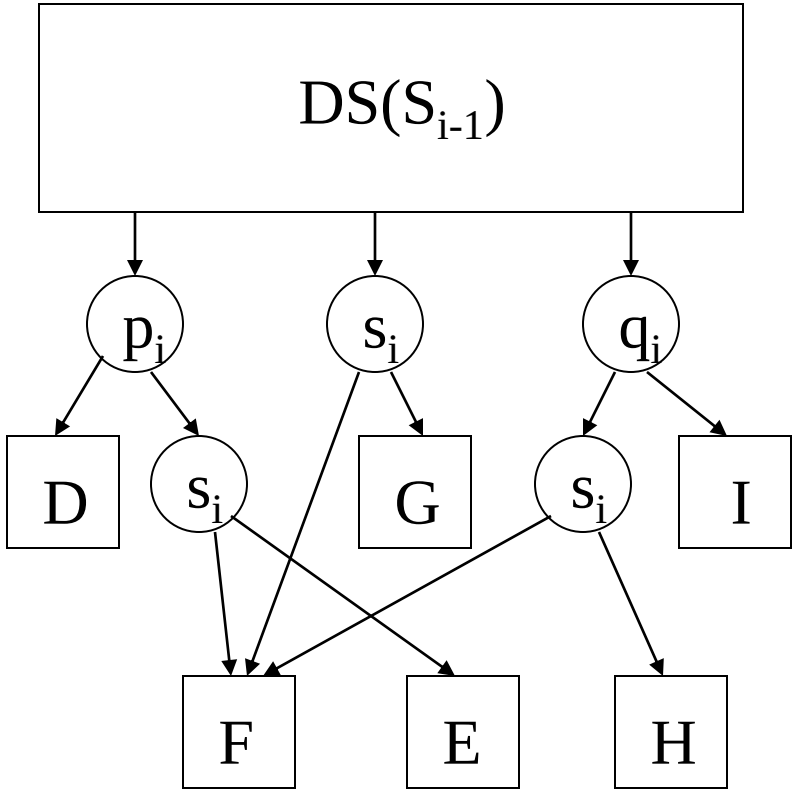


Gdy odcinek s_i przecina wiele istniejących już trapezów, to w miejsce wierzchołka odpowiadającego skrajnie lewemu trapezowi wstawiamy p_i, którego lewym synem jest wierzchołek odpowiadający trapezowi powstającemu po lewej stronie p_i a prawym synem jest s_i.

W miejsce wierzchołka odpowiadającego skrajnie prawemu trapezowi wstawiamy q_i, którego prawym synem jest wierzchołek odpowiadający trapezowi powstającemu po prawej stronie q_i a lewym synem jest s_i.

Pozostałym trapezom odpowiadają wierzchołki s_i. Lewy i prawy syn dowolnego wierzchołka s_i odpowiada odpowiednio trapezowi powstałemu powyżej i poniżej odcinka s_i w miejscu poprzedniego trapezu.





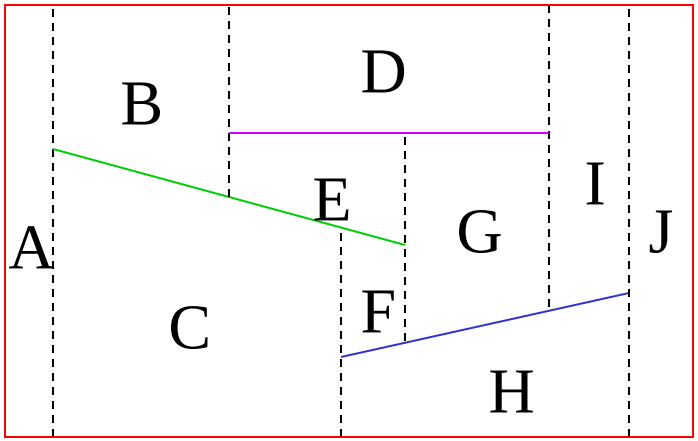
Algorytm

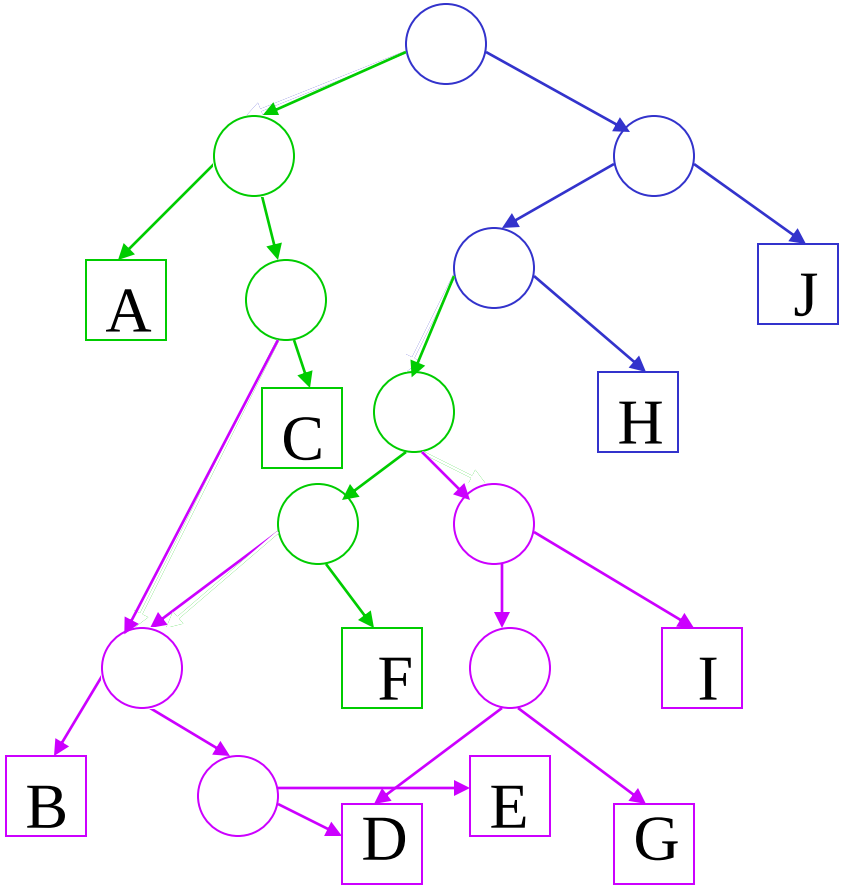
inicjalizuj strukturę DS ;

for i:=1 ***to*** n ***do***

z pomocą struktury DS znajdź trapezy przecinane przez odcinek s_i ;

zastąp w strukturze DS wierzchołki odpowiadające tym trapezom nowym układem wierzchołków;





Twierdzenie

Algorytm oblicza mapę trapezową T(S) dla zbioru n odcinków S i tworzy strukturę danych DS(S) dla mapy T(S) w oczekiwanym czasie O(n log n). Oczekiwany rozmiar struktury wynosi O(n), a lokalizacja punktu wymaga oczekiwanego czasu O(log n).

Dowód.

Zmiana wierzchołka odpowiadającego trapezowi zwiększa długość ścieżki wyszukującej punkt o co najwyżej 3 wierzchołki. Jednak szacowanie długości ścieżki wyszukiwań w ten sposób jest zbyt grube.

Rozważmy ścieżkę wyszukiwań punktu q w strukturze danych DS.

Niech X_i oznacza dla $1 \leq i \leq n$ liczbę wierzchołków na ścieżce wyszukiwań dodanych w i-tej iteracji. Zatem oczekiwana długość ścieżki wyszukiwań wynosi $E(\Sigma^n_{i=1} X_i) = \Sigma^n_{i=1} E(X_i)$.

Niech P_i oznacza prawdopodobieństwo przejścia w trakcie lokalizacji punktu q przez wierzchołki stworzone w i-tej iteracji. Mamy $E(X_i) \leq 3P_i$ oraz $P_i = P[t_q(S_i) \neq t_q(S_{i-1})]$, gdzie $t_q(S_i)$ oznacza trapez zawierający punkt q w mapie powstałej po i-tej iteracji.

Aby oszacować prawdopodobieństwo P_i zastosujemy analizę wsteczną.

W mapie powstałej po i-tej iteracji, zmianę trapezu zawierającego punkt q spowodować może usunięcie co najwyżej czterech odcinków:

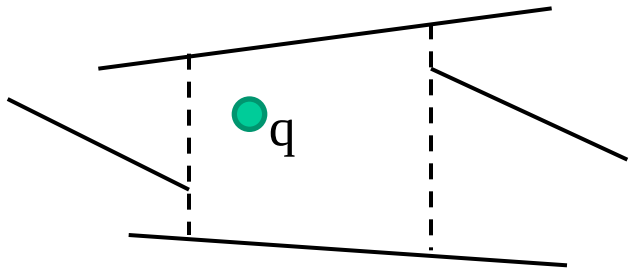
- będącego górną krawędzią trapezu,
- będącego dolną krawędzią trapezu,
- wyznaczającego poprzez swój koniec lewą ścianę trapezu,
- wyznaczającego poprzez swój koniec prawą ścianę trapezu.

Jeśli koniec odcinka będącego np. dolną krawędzią trapezu wyznacza równocześnie np. lewą ścianę trapezu, to liczba odcinków, których usunięcie może zmienić trapez zawierający punkt q, jest mniejsza niż 4.

Zatem $P_i = P[t_q(S_i) \neq t_q(S_{i-1})] = P[t_q(S_i) \notin T(S_{i-1})] \leq 4/i$.

Stąd $\Sigma^n_{i=1} E(X_i) \leq \Sigma^n_{i=1} 3P_i \leq \Sigma^n_{i=1} 12/i = 12\, \Sigma^n_{i=1} 1/i = 12H_n = O(\log n)$.

czyli oczekiwany czas lokalizacji punktu jest $O(\log n)$.



Zbadajmy oczekiwany rozmiar struktury.

Wynosi on: (liczba trapezów) + $\Sigma^{n}_{i=1}$(liczba węzłów stworzonych w i-tej iteracji).

Liczba trapezów szacuje się przez O(n).

Natomiast liczba wierzchołków dodanych w jednej iteracji może być liniowa względem liczby zbadanych odcinków. Prowadzi to do kwadratowego (pesymistycznego) oszacowania rozmiaru omawianej struktury danych .

Niech k_i oznacza liczbę trapezów tworzonych w i-tej iteracji.

Zatem liczba nowych wierzchołków wewnętrznych wynosi k_i-1.

Niech $\delta(t,s)$ będzie równe 1, gdy trapez $t \in T(S_i)$ nie będzie należeć do $T(S_{i-1})$ po usunięciu odcinka s oraz 0 w przeciwnym przypadku.

Mamy $\Sigma_{s\in S} \Sigma_{t\in T(Si)} \delta(t,s) \leq 4|T(S_i)| = O(i)$.

Stąd $E(k_i) = 1/i \Sigma_{s\in S} \Sigma_{t\in T(Si)} \delta(t,s) \leq O(i)/i = O(1)$,

czyli oczekiwana liczba nowych wierzchołków wewnętrznych powstałych w i-tej iteracji jest stała.

Zatem oczekiwany rozmiar struktury danych wynosi

$O(n) + \Sigma^{n}_{i=1} E(k_i-1) = O(n) + \Sigma^{n}_{i=1} E(k_i) = O(n) + \Sigma^{n}_{i=1} O(1) = O(n)$,

czyli jest liniowy względem liczby odcinków.

Teraz możemy obliczyć oczekiwany czas pracy algorytmu, który wynosi:

(Koszt inicjalizacji) + $\Sigma^{n}_{i=1}$ (średni czas wyszukiwania położenia końców odcinka dodawanego w i-tej iteracji + liczba nowych wierzchołków dodawanych w i-tej iteracji) =

$O(1) + \Sigma^{n}_{i=1} (O(\log i) + O(E(k_i))) = O(1) + O(n \log n) + O(n) = O(n \log n)$, co kończy dowód.

Dziękuję za uwagę.

Ćwiczenia.

1. Udowodnij, że ścieżka o minimalnej długości łącząca dane punkty s i t wewnątrz danego wielokąta prostego jest łamaną składającą się z krawędzi grafu widzialności dla zbioru krawędzi danego obszaru A oraz punktów s i t (tzn. najkrótsza ścieżka między punktami s i t może zmieniać kierunek jedynie w punktach będących wierzchołkami obszaru A).
2. Dlaczego wybieramy punkty wewnątrz trapezów w mapie drogowej ?
3. Udowodnij, że kształt sumy Minkowskiego robota i przeszkody nie zależy od wyboru punktu odniesienia.
4. Pokaż, że suma Minkowskiego dwóch wielokątów niewypukłych, z których jeden ma skończoną liczbę wierzchołków a drugi ma n wierzchołków, może mieć rozmiar $O(n^2)$.

5. Który z poniższych algorytmów generuje losowe permutacje dla danej tablicy A długości n (tzn. każda możliwa permutacja A jest równie prawdopodobna jako wynik):

a) (bez identyczności) for i:=1 to n do zamień(A[i],A[RANDOM(i+1,n)]);
b) for i:=1 to n do zamień(A[i],A[RANDOM(1,n)]);
c) for i:=n downto 2 do zamień(A[i],A[RANDOM(1,i)]);

6. Udowodnij, że liczba wewnętrznych węzłów w strukturze przeszukiwań DS algorytmu tworzenia mapy trapezowej wzrasta o k_i-1 w iteracji i, gdzie k_i jest liczbą nowych trapezów w $T(S_i)$ (tzn. nowych liści w DS).

7. Podaj przykład układu odcinków generującego w pesymistycznym przypadku strukturę podziału na trapezy rozmiaru $O(n^2)$.

8. Udowodnij, że mapa trapezowa n odcinków w położeniu ogólnym ma co najwyżej 3n+1 trapezów.

9. Prosty wielokąt nazywamy gwiaździstym, gdy zawiera punkt q widoczny z każdego punktu wielokąta. Podaj algorytm, którego oczekiwany czas działania jest liniowy i sprawdza, czy dany prosty wielokąt jest gwiaździsty.