Lokalizacja punktów w przestrzeni

Metoda trapezowa

Paweł Lamża

Piotr Czech

Spis treści

[Wprowadzenie 3](#_Toc123077156)

[Dokumentacja użytkownika 3](#_Toc123077157)

[Informacje o programie 3](#_Toc123077158)

[Korzystanie z programu 3](#_Toc123077159)

[Dokumentacja techniczna 3](#_Toc123077160)

[Klasy 3](#_Toc123077161)

[Point 3](#_Toc123077162)

[Line 3](#_Toc123077163)

[Trapeze 4](#_Toc123077164)

[XNode 4](#_Toc123077165)

[YNode 4](#_Toc123077166)

[LeafNode 5](#_Toc123077167)

[SearchGraph 5](#_Toc123077168)

# Wprowadzenie

Poniższy dokument zawiera podstawowe informacje umożliwiające poprawne korzystanie z programu w sekcji „Dokumentacja użytkownika”. Szczegółowy opis funkcji i struktur danych tworzących program znajduje się w sekcji „Dokumentacja techniczna”. Dokładny opis problemu jest w sekcji „Sprawozdanie”.

# Dokumentacja użytkownika

## Informacje o programie

Program zapisany jest w pliku o formacie .ipynb, który można uruchomić za pomocą np. Jupyter Notebook. Działanie algorytmu zawartego w programie ilustruje dołączona do programu prezentacja. Punktem określana jest para współrzędnych zmiennoprzecinkowych x i y, natomiast odcinkiem para punktów, tj. jego początek oraz koniec.

## Korzystanie z programu

Najpierw należy uruchomić moduły dotyczące narzędzia graficznego i funkcji pomocniczych w pliku project.ipynb, następnie zdefiniować zbiór odcinków na jeden z trzech sposobów: generując losowy zbiór odcinków za pomocą sekcji „Losowe odcinki”, zadając własny zbiór odcinków w sekcji „Własne rysowanie” lub wczytując zbiór odcinków z pliku .json za pomocą funkcji read\_lines\_from\_file(name). Po zdefiniowaniu zbioru odcinków należy uruchomić pozostałe moduły a następnie funkcję build\_map(lines), która stworzy graf przeszukiwań. Następnie należy stworzyć obiekt klasy Point o współrzędnych, które chcemy zlokalizować i uruchomić funkcję find\_point().

# Dokumentacja techniczna

## Klasy

### Point

Klasa reprezentująca punkty

#### A*trybuty:*

- *float* x - współrzędna x punktu  
 - *float* y - współrzędna y punktu

#### Metody:

- \_\_init\_\_(self, *float* x, *float* y) – funkcja inicjalizująca obiekt klasy Point o współrzędnych x,y

### Line

Klasa reprezentująca odcinki

#### A*trybuty:*

- *Point* left – punkt początkowy odcinka  
 - *Point* right – punkt końcowy odcinka (o większej współrzędnej x)  
 - *float* a – współczynnik kierunkowy prostej przechodzącej przez odcinek  
 - *float* b – wyraz wolny równania kierunkowego prostej przechodzącej przez odcinek

#### Metody:

- \_\_init\_\_(self, *Point* p, *Point* q) – funkcja inicjalizująca obiekt klasy Line o końcach w punktach p i q  
 - is\_line\_below\_point(self, *Point* point) – funkcja zwraca True jeśli punkt point leży nad odcinkiem, w przeciwnym razie False  
 - value\_at\_x(self, *float* x) - funkcja zwracająca wartość y dla argumentu x jeśli znajduje się on między początkiem a końcem odcinka, w przeciwnym razie funkcja zwraca None

### Trapeze

Klasa reprezentująca trapezy na płaszczyźnie

#### A*trybuty:*

- *Line* top – górny bok trapezu  
 - *Line* bottom – dolny bok trapezu  
 - *Point* left – lewy wierzchołek trapezu  
 - *Point* right – prawy wierzchołek trapezu  
 - *Trapeze* upper\_left – lewy górny sąsiad  
 - *Trapeze* lower\_left – lewy dolny sąsiad *- Trapeze* upper\_right – prawy górny sąsiad *- Trapeze* lower\_right – prawy dolny sąsiad  
 - *LeafNode* node – odwołanie do liścia w grafie przeszukiwania

#### Metody:

- \_\_init\_\_(self, *Line* top, *Line* bottom, *Point* left, *Point* right) – funkcja inicjująca obiekt klasy Trapeze, argumentami są początkowe wartości atrybutów o tych samych nazwach

### XNode

Klasa reprezentująca węzły wierzchołków

#### A*trybuty:*

- *Point* point – punkt związany z węzłem  
 - *XNode/LeafNode/YNode* left – węzeł po lewej stronie  
 - *XNode/LeafNode/YNode* right – węzeł po prawej stronie

#### Metody:

- \_\_init\_\_(self, *Point* point, left=None, right=None) – funkcja inicjalizująca obiekt klasy XNode  
 - set\_left(self, node) – funkcja ustawia wartość atrybutu left dla obiektu XNode i aktualizuje listę rodziców wierzchołka node  
 - set\_right(self, node) – funkcja ustawia wartość atrybutu right dla obiektu XNode i aktualizuje listę rodziców wierzchołka node  
 - is\_point\_on\_left\_side(self, *Point* q) – funkcja zwraca True jeśli punkt q jest po lewej stronie punktu self.point, w przeciwnym razie zwraca False

### YNode

Klasa reprezentująca węzły odcinków

#### A*trybuty:*

- *Line* line – odcinek związany z węzłem  
 - *XNode/LeafNode/YNode* above – węzeł nad odcinkiem  
 - *XNode/LeafNode/YNode* below – węzeł pod odcinkiem

#### Metody:

- \_\_init\_\_(self, *Line* line, above=None, below=None) – funkcja inicjalizująca obiekt klasy YNode   
 - set\_above(self, node) - funkcja ustawia wartość atrybutu above dla obiektu YNode i aktualizuje listę rodziców wierzchołka node  
 - set\_below(self, node) - funkcja ustawia wartość atrybutu below dla obiektu YNode i aktualizuje listę rodziców wierzchołka node  
 - is\_point\_above(self, *Point* q) - funkcja zwraca True jeśli punkt q jest nad punktem self.point, w przeciwnym razie zwraca False

### LeafNode

Klasa reprezentująca węzły trapezów

#### A*trybuty:*

- *Trapeze* trapeze – trapez powiązany z danym węzłem  
 - *list* parents – lista węzłów będących rodzicami danego węzła

#### Metody:

- \_\_init\_\_(self, *Trapeze* trapeze) – funkcja inicjalizująca obiekt klasy LeafNode  
 - replace(self, search\_graph, node) – funkcja zmieniająca wskazania rodziców danego węzła na podany węzeł node

### SearchGraph

Klasa reprezentująca graf przeszukiwania.

#### A*trybuty:*

- *XNode/LeafNode/YNode* root - węzeł będący korzeniem grafu

#### Metody:

- \_\_init\_\_(self, root) – funkcja inicjalizująca obiekt klasy SearchGraph  
 - set\_root(self, root) – funkcja ustawiająca wartość parametru root na zadaną  
 - find(self, *Point* point, node) – funkcja zwracająca trapez, do którego należy dany punkt

## Funkcje

permute(*list[list[tuple]]* lines) – funkcja przyjmuje listę odcinków i zwraca jej losową permutację

get\_line(*list[tuple]* line) – funkcja przyjmuje współrzędne końców odcinka i zwraca utworzony z nich obiekt klasy Line

update\_left\_neighbours(*Trapeze* old\_trapeze, *Trapeze* new\_trapeze) – funkcja aktualizuje wskazania lewych sąsiadów usuwanego trapezu tak, aby wskazywały na nowy

update\_right\_neighbours(*Trapeze* old\_trapeze, *Trapeze* new\_trapeze) – funkcja aktualizuje wskazania prawych sąsiadów usuwanego trapezu tak, aby wskazywały na nowy

set\_neighbours(*Trapeze* trapeze, *Trapeze* upper\_right, *Trapeze* lower\_right, *Trapeze* upper\_left, *Trapeze* lower\_left) – funkcja ustawia odpowiednich sąsiadów trapezu

update\_neighbours(*Trapeze* trapeze, *Trapeze* upper\_right, *Trapeze* lower\_right, *Trapeze* upper\_left, *Trapeze* lower\_left) – funkcja aktualizuje odpowiednich sąsiadów trapezu jeśli wcześniej istnieli

find\_zone(*Trapeze* start\_trapeze, *Line* line) – funkcja zwraca listę wszystkich trapezów przecinanych przez odcinek line

line\_in\_single\_trapeze(*SearchGraph* search\_graph , *Trapeze* trapeze , *Line* line) – funkcja obsługująca przypadek, gdy odcinek znajduje się w całości w jednym trapezie, zwraca nowo powstałe 4 trapezy

line\_in\_multiple\_trapezes(*SearchGraph* search\_graph, *list[Trapeze]* trapezes, *Line* line) – funkcja obsługująca przypadek, gdy odcinek przecina wiele trapezów, zwraca wszystkie nowo powstałe trapezy

make\_outer\_trapeze(lines) – funkcja tworząca trapez zewnętrzny (prostokąt)

get\_corners(lines) – funkcja zwracająca skrajne punkty mapy odcinków

visualise(line, added\_trapeze\_lines, all\_lines\_added, zone, added\_trapezes, scenes) – funkcja   
 dodająca sceny do wizualizacji przebiegu algorytmu

trapeze\_lines(trapeze) – funkcja zwracająca linie danego trapezu

cut\_vertical\_lines(lines, trapeze) – funkcja przycinająca pionowe odcinki

build\_map(lines) – funkcja tworząca mapę trapezową i zwracająca graf przeszukiwania

find\_point(*SearchGraph* search\_graph, *Point* point, *Scene* background) – funkcja wizualizująca położenie szukanego punktu na mapie trapezowej

# Sprawozdanie

## Opis problemu

Projekt polegał na przygotowaniu implementacji algorytmu określania położenia punktu P na płaszczyźnie dwuwymiarowej. Aby program wykonywał się efektywnie zgodnie z teoretyczną złożonością obliczeniową należało wybrać odpowiednie struktury danych. W naszej implementacji użyto do tego mapy trapezowej T reprezentowanej jako struktury powiązań sąsiedzkich między trapezami oraz struktury przeszukiwań D jako grafu wyszukiwania. Graf wyszukiwania zawierał 3 rodzaje węzłów: x-węzeł - reprezentujący punkt; y-węzeł - reprezentujący odcinek; liść-węzeł reprezentujący aktualnie istniejące trapezy na mapie trapezowej. Zaimplementowane struktury były ze sobą wzajemnie powiązane, ponieważ trapez z T(S) ma wskaźnik do odpowiadającego mu liścia w D, a liść w D ma wskaźnik do odpowiadającemu mu trapezu w T(S). Struktury musiały także poprawnie obsługiwać wydarzenia dla dowolnych odcinków zaczynających się w tych samych punktach. W takim przypadku niezbędne było prawidłowe określenie położenia punktu w strukturze przeszukiwań D. W algorytmie należało także poprawnie określić sąsiedztwo trapezów. Dzięki położeniu ogólnym wierzchołków trapez mógł posiadać tylko 4 sąsiadów. Jednak mimo tak małej ilości sąsiadów, należało rozważyć kilka możliwości ich występowania. Przy usuwania trapezu i wstawianiu nowych trapez z wyznaczonej dla odcinka sfery należało poprawnie zaktualizować sąsiedzkie dowiązania trapezów oraz pamiętać aby skracać rozszerzenia pionowe trapezów, które wystawały ponad bok wstawianych trapezów.

## Testy

Aby zweryfikować poprawność działania algorytmu przeprowadzono szereg testów weryfikujących działanie programu. Testy te miały szereg zadań, m.in. weryfikację poprawności tworzenia nowych trapezów, weryfikację poprawności tworzenia i edytowania mapy trapezowej oraz grafu przeszukiwania. Testy były wprowadzane przez interfejs graficzny użytkownika, zawierały różne konfiguracje odcinków o różnym stopniu złożoności.

## Wnioski

Algorytm dokonuje poprawnego określenia pozycji zadanego punktu w podziale planarnym, co zweryfikowano z pomocą dołączonej do algorytmu wizualizacji kolejnych etapów działania algorytmu (zarówno tworzenia samej mapy trapezowej jak i wyszukiwania na niej zadanych wierzchołków). Sama implementacja również wpisuje się w założenia co do użytej pamięci oraz złożoności obliczeniowej algorytmu (odpowiednio O(n) oraz O(nlogn) dla tworzenia mapy trapezowej oraz O(logn) dla lokalizacji punktu). Nie udało się natomiast obsłużyć przypadku, gdy punkt początkowy odcinka, dla którego wyznaczamy strefę, znajduję się uprzednio w grafie przeszukiwania.