Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda trapezowa

Temat 2, grupa 4

Marta Stanisławska
Mateusz Wójcicki

1.Wstęp

Celem projektu było opracowanie programu umożliwiającego lokalizację punktu w przestrzeni dwuwymiarowej przy zastosowaniu metody trapezowej. W ramach realizacji zaimplementowano funkcję odpowiedzialną za konstrukcję mapy trapezowej oraz graf wyszukiwania dla zadanej siatki podziału przestrzeni. Dodatkowo proces budowy mapy oraz późniejszego wyszukiwania punktów został wzbogacony o wizualizację za pomocą interfejsu graficznego programu.

2.Dokumentacja - część użytkownika

Najważniejszym plikiem w projekcie jest projekt_kod.ipynb. Zawiera on implementację narzędzia umożliwiającego wprowadzanie własnych odcinków na płaszczyźnie, a także zbiór klas i funkcji,które łączą się, tworząc kompletny algorytm wraz z wizualizacją jego działania krok po kroku. Główną funkcją jest funkcja trapezoidal_map(S), której argumentem jest zbiór wcześniej wybranych krawędzi. Po jej wywołaniu możemy zaobserwować jak zachowuje się algorytm krok po kroku dzięki czytelnej wizualizacji

3.Dokumentacja - część techniczna

I. Spis modułów:

- matplotlib - moduł odpowiadający za obsługę interfejsów wykresów
- numpy - moduł odpowiedzialny za większość operacji matematycznych, takich jak generowanie liczb losowych

II. Opis struktur danych w programie:

a) Point

Klasa reprezentuje punkt w przestrzeni dwuwymiarowej, przechowując informacje o jego współrzędnych **x** i **y**.

Podstawową metodą klasy jest konstruktor **__init__(self,x,y)**, który inicjalizuje obiekt punktu na podstawie wartości współrzędnych **x** i **y** podanych jako argumenty. Konstruktor nie zwraca żadnej wartości.

Dodatkowo klasa zawiera metodę porównawczą **__gt__(self, o)ther**, służącą do porównywania punktów względem współrzędnej **x**. Przyjmuje ona jako argument inny obiekt klasy **Point** (oznaczony jako **other**) i zwraca wartość **True**, jeśli współrzędna **x** bieżącego punktu jest większa niż współrzędna **x** punktu **other**. W przeciwnym przypadku zwraca **False**.

b) Section

Klasa implementuje reprezentację odcinka. Zawiera jako atrybuty punkty **L** i **R** typu **Point**, które reprezentują odpowiednio skrajny lewy i prawy punkt odcinka. Ponadto, klasa posiada liczby rzeczywiste typu **float**, oznaczone jako **a** i **b**, przechowujące informacje o nachyleniu i stałej odcinka.

Podstawową funkcją klasy jest **__init__(self, left_end, right_end)**, będąca konstruktorem. Przyjmuje ona argumenty **left_end** i **right_end**, które odpowiadają dwóm skrajnym punktom odcinka. Konstruktor automatycznie ustawia te punkty w odpowiedniej kolejności jako atrybuty **L** i **R**, a następnie oblicza wartości **a** i **b**.

Metoda **above_point(self, point)** przyjmuje jako argument punkt typu Point i zwraca **True**, jeśli punkt znajduje się powyżej danego odcinka (czyli ma większą wartość współrzędnej **y** niż odcinek dla danego **x**). W przeciwnym przypadku metoda zwraca **False**.

Metoda **get_point_at_x(self, x)** przyjmuje liczbę rzeczywistą **x** typu float i zwraca wartość współrzędnej **y** dla podanej wartości **x**, o ile leży ona w zakresie od współrzędnej **x** punktu **left** do współrzędnej **x** punktu **right**. Jeśli **x** znajduje się poza tym przedziałem, metoda zwraca **None**.

c) Trapezoid

Klasa reprezentująca trapez, który zawiera atrybuty opisujące jego właściwości geometryczne. Atrybuty t**op** i **bottom** typu **Section** przechowują informacje o górnej i dolnej krawędzi trapezu. Atrybuty **left_p** i **right_p** typu **Point** przechowują współrzędne punktów znajdujących się na lewej i prawej krawędzi trapezu. Atrybuty **right_upper_neighbour**, **right_lower_neighbour**, **left_upper_neighbour** i **left_lower_neighbour** typu **Trapezoid** odnoszą się do sąsiadujących trapezów: górnego prawgo, dolnego prawego, górnego lewego oraz dolnego lewego, w mapie trapezowej. Dodatkowo, atrybut **leaf** zawiera odwołanie do wierzchołka w grafie przeszukiwań.

Konstruktor klasy **__init__(self, top, bottom, left_point, right_point)** przyjmuje cztery argumenty: **top** i **bottom** typu **Section**, które reprezentują górny i dolny odcinek trapezu, oraz **left_point** i **right_point** typu **Point**, które są punktami należącymi odpowiednio do lewej i prawej krawędzi trapezu (odpowiadają wartościom **left_p** i **right_p**)

d) Node

Klasa zawierająca informacje o **węźle** w strukturze **przeszukiwań**. Zawiera atrybut **type** typu **String**, który określa **rodzaj węzła**, a także atrybut **label**, który w zależności od typu węzła przechowuje obiekt typu **Trapezoid**, **Section** lub **Point**. Dodatkowo, klasa zawiera atrybuty **left** i **right** typu **Node**, które reprezentują lewe i prawe dziecko w grafie przeszukiwań, lub **None**, jeśli węzeł jest **liściem**.

Konstruktor klasy **__init__(self, type, label)** przyjmuje parametr **type** typu **String**, który ustawia rodzaj węzła, oraz parametr **label**, który odpowiada za przechowywany przez węzeł obiekt.

e) SearchGraph

Klasa reprezentująca **strukturę wyszukiwań**. Zawiera atrybut **root** typu **Node**, który reprezentuje korzeń tej struktury.

Konstruktor klasy **__init__(self, root)** przyjmuje parametr **root** typu **Node** i ustawia go jako korzeń struktury **przeszukiwań**.

Funkcja **find(self, node, point, segment)** przyjmuje trzy parametry:

- **node** typu **Node**, który odpowiada za obecny węzeł w strukturze wyszukiwań,
- **point** typu **Point**, który odpowiada za szukany punkt,
- **segment** typu **Section**, który reprezentuje obecnie badany odcinek w mapie trapezowej (nie jest używany, gdy mapa trapezowa jest już gotowa, a trapez jest poszukiwany).

Wartością zwróconą jest najmniejszy trapez typu **Trapezoid**, który zawiera **szukany punkt**.

III. Opis funkcji potrzebnych do wizualizacji:

a) **`to_xy(section)`**
Funkcja przyjmuje obiekt **`section`** (typu **`Section`**) i zwraca krotkę z współrzędnymi dwóch punktów: lewego (**`section.L`**) i prawego (**`section.R`**) końca odcinka w formie (**`x, y`**).

b) **`draw_frame(frame, vis)`**
Funkcja rysuje ramkę na podstawie obiektu **`frame`** (zawierającego punkty **`left_p`** i **`right_p`** oraz górną i dolną krawędź). Rysuje cztery linie tworzące ramkę i dodaje je do obiektu **`vis`** (obiekt wizualizujący), używając czarnego koloru. Zwraca obiekt **`vis`** z dodanymi liniami.

c) **`draw_trapezoid(figure, vis, color="green")`**
Funkcja rysuje trapez (reprezentowany przez obiekt **`figure`**(`typu` **`trapezoid`**)) i jego krawędzie w określonym kolorze (domyślnie zielony). Rysuje punkty na lewych i prawych krawędziach, a także poziome i pionowe boki trapezu.

d) **`draw_trapezoid_with_neighbours(trapezoid, vis, neighbours)`**
Funkcja rysuje trapez **`trapezoid`** oraz jego sąsiadów (jeśli istnieją). Usuwa poprzednie rysunki sąsiadów, rysuje trapez i jego sąsiednie trapezy w kolorze pomarańczowym. Zwraca zaktualizowaną listę sąsiadów.

e) **`find_all_trapezoids(node, trapezoids)`**
Rekurencyjna funkcja, która przechodzi przez strukturę drzewa i dodaje do listy **`trapezoids`** wszystkie trapezy (węzły liści) z drzewa, zaczynając od korzenia **node.**

f) **`draw_map(D, R)`**
Funkcja rysuje mapę trapezową. Rozpoczyna od rysowania ramki za pomocą funkcji **`draw_frame`**, następnie za pomocą funkcji **`find_all_trapezoids`** zbiera wszystkie trapezy z drzewa **`D.root`** i rysuje je za pomocą funkcji **draw_trapezoid**.

IV. Opis funkcji związanych z ręcznym wprowadzaniem odcinków:

a) on_key(event)
Funkcja reaguje na naciśnięcie klawisza. Jeżeli naciśnięty zostanie klawisz Enter, sprawdza, czy liczba punktów w **current_points** jest parzysta. Jeśli nie, wyświetla komunikat o błędzie i ponownie ustawia tytuł. Jeśli liczba punktów jest parzysta, wyświetla komunikat o zakończeniu tworzenia odcinków i zamyka okno po krótkim czasie.

b) onclick(event)
Funkcja obsługuje kliknięcia myszy na wykresie. Reaguje na kliknięcie w obszarze wykresu (sprawdzając **event.inaxes**). Jeśli kliknięty punkt spełnia określone warunki:

- Jeśli w **current_points** jest już jeden punkt i kliknięty punkt jest na tej samej współrzędnej x, wyświetla komunikat o niedozwolonych pionowych odcinkach.
- Jeśli punkt o tej samej współrzędnej x już istnieje w **used_x**, ale nie jest częścią połączonego odcinka, wyświetla komunikat o błędzie.

W przeciwnym razie dodaje punkt do listy **current_points**, a następnie rysuje go na wykresie. Po dodaniu dwóch punktów rysuje odcinek między nimi, zapisuje ten odcinek do listy **generate_sections** (zachowując kolejność punktów według współrzędnej x) i resetuje listę **current_points** na pustą.

V. Opis pozostałych funkcji:

a) find_boundaries(sections)

Funkcja ta służy do określenia granic obszaru na podstawie przekazanej listy odcinków. Analizując współrzędne każdego odcinka, funkcja wyznacza minimalne i maksymalne wartości w osi **X** i **Y**, które definiują prostokąt obejmujący wszystkie odcinki.

b) create_frame(sections)

Na podstawie granic wyznaczonych przez funkcję **find_boundaries**, funkcja **create_frame** tworzy ramkę w kształcie prostokąta wokół wszystkich odcinków.

c) find_intersected_trapezoids(D, section)

Funkcja przeszukuje graf wyszukiwania **D** i zwraca te trapezy, które zostały przecięte przez przekazany odcinek **section**.

d) UXYZ(OLD_NODE, section, NEW_LEFT, NEW_TOP, NEW_BOTTOM, NEW_RIGHT)

Funkcja aktualizuje strukturę grafu wyszukiwania w przypadku, w którym odcinek **section** w pełni znajduje się w rozpatrywanym trapezie reprezentowanym przez **OLD_NODE**

e) UYZ(OLD_NODE, section, NEW_LEFT, NEW_TOP, NEW_BOTTOM)

Funkcja aktualizuje strukturę grafu wyszukiwania w przypadku, w którym odcinek **section** zaczyna się w rozpatrywanym trapezie reprezentowanym przez **OLD_NODE**, ale kończy poza jego obszarem.

f) YZX(OLD_NODE, section, NEW_TOP, NEW_BOTTOM, NEW_RIGHT)

Funkcja aktualizuje strukturę grafu wyszukiwania w przypadku, w którym odcinek **section** kończy się w rozpatrywanym trapezie reprezentowanym przez **OLD_NODE**, ale zaczyna poza jego obszarem.

g) YZ(OLD_NODE, section, NEW_TOP, NEW_BOTTOM)

Funkcja aktualizuje strukturę grafu wyszukiwania w przypadku, w którym odcinek **section** zaczyna i kończy się poza obszarem rozpatrywanego trapezu reprezentowanego przez **OLD_NODE.**

h) handle_first(trapezoid, OLD_LOWER_LEFT, OLD_UPPER_LEFT, OLD_UPPER_RIGHT, OLD_LOWER_RIGHT, s, NEW_TOP, NEW_BOTTOM, NEW_LEFT, trapezoids=None)

Ta funkcja obsługuje przypadek wstawiania trapezu po lewej stronie przekazywanego odcinka s. Na podstawie granic starego trapezu, tworzy nowy trapez **NEW_LEFT**, aktualizując sąsiedztwa węzłów. Jeżeli nie dodajemy trapezu po lewej, funkcja sprawdza również, czy nowy odcinek jest wyżej czy niżej w stosunku do **left_p** starego trapezu, modyfikując odpowiednio strukturę grafu wyszukiwania.

i) **handle_last(trapezoid, OLD_LOWER_LEFT, OLD_UPPER_LEFT, OLD_UPPER_RIGHT, OLD_LOWER_RIGHT, section, NEW_TOP, NEW_BOTTOM, NEW_RIGHT, trapezoids=None)**
Funkcja ta obsługuje przypadek, gdy wstawiamy trapez po prawej stronie Ta funkcja obsługuje przypadek wstawiania trapezu po lewej stronie przekazywanego odcinka s. Na podstawie granic starego trapezu, tworzy nowy trapez **NEW_RIGHT**, aktualizując sąsiedztwa węzłów. Jeżeli nie dodajemy trapezu po lewej, funkcja sprawdza również, czy nowy odcinek jest wyżej czy niżej w stosunku do **right_p** starego trapezu, modyfikując odpowiednio strukturę grafu wyszukiwania.

j) **insert_into_one_trapezoid(trapezoid, section)**
Funkcja ta wykonuje wstawienie odcinka section do istniejącego trapezu trapezoid. Tworzy nowe trapezy **NEW_TOP** i **NEW_BOTTOM**, które dzielą oryginalny trapez na dwa mniejsze trapezy, a następnie odpowiednio wywołuje **funkcje handle_first** i **handle_last**, aby zaktualizować sąsiedztwa trapezów. Następnie, w zależności od utworzonych nowych trapezów, dokonuje aktualizacji drzewa trapezów za pomocą funkcji **UXYZ**, **UYZ**, **YZX** lub **YZ**.

k) **insert_into_many_trapezoids(S)**
Funkcja ta rozpatruje nowy odcinek **s** i aktualizuje strukturę grafu wyszukiwania. Dzieli trapezy na dwie grupy: te powyżej i te poniżej nowego odcinka. Dla każdego trapezu, sprawdza, czy odcinek przechodzi do jego górnego lub dolnego sąsiada. Na tej podstawie tworzy nowe trapezy, łączy je z odpowiednimi sąsiadami, a także aktualizuje strukturę drzewa wyszukiwania (dodając odpowiednie węzły i liście). Dla każdego trapezu, funkcja dostosowuje jego sąsiedztwo, tworząc nowe połączenia w strukturze, aby zachować spójność geometryczną.

5. Opis Problemu

Projekt dotyczył implementacji algorytmu określania położenia punktu P w poligonowym podziale płaszczyzny. Celem było stworzenie efektywnej struktury danych, która pozwalałaby na szybkie odnajdywanie obszaru zawierającego dany punkt. Do realizacji tego zadania wybrano metodę trapezową, w której podział płaszczyzny reprezentowany jest jako zbiór odcinków. Podział ten przekształcany jest w mapę trapezową – strukturę dzielącą płaszczyznę na trapezy (lub trójkąty traktowane jako zdegenerowane trapezy), uzyskiwane poprzez prowadzenie pionowych rozszerzeń od końców odcinków aż do napotkania innego odcinka lub ramki ograniczającej.

Struktura danych składała się z dwóch powiązanych komponentów: mapy trapezowej T(S), reprezentującej relacje sąsiedztwa między trapezami, oraz struktury przeszukiwań D, będącej acyklicznym grafem skierowanym. Węzły grafu D dzieliły się na trzy typy:

- x-węzły, reprezentujące obiekty typu **Point**
- y-węzły, reprezentujące obiekty typu **Section**
- liście, które odpowiadały aktualnym trapezom z mapy T(S).

Wzajemne powiązania między strukturami zapewniały, że każdy trapez w T(S) miał wskaźnik do odpowiadającego mu liścia w D, a każdy liść w D – wskaźnik do trapezu w T(S).

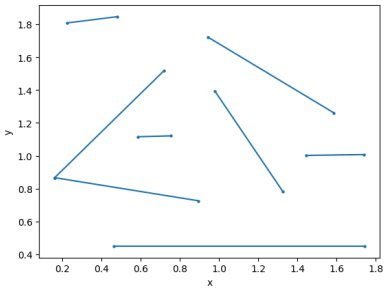
Kluczowym elementem implementacji było poprawne obsługiwanie zdarzeń związanych z dodawaniem odcinków do mapy trapezowej, szczególnie w przypadku odcinków rozpoczynających się w tym samym punkcie. W takich sytuacjach niezbędne było precyzyjne określenie pozycji punktu w strukturze D. Istotne było również prawidłowe aktualizowanie relacji sąsiedztwa między trapezami. Trapez mógł mieć maksymalnie czterech sąsiadów, ale przy wstawianiu nowych trapezów konieczne było skracanie pionowych rozszerzeń trapezów istniejących.

Konstrukcja mapy trapezowej odbywała się za pomocą randomizowanego algorytmu przyrostowego. Odcinki były wprowadzane w losowej kolejności, co zapewniało dobrą oczekiwaną złożoność czasową. Dla zbioru odcinków w położeniu ogólnym (bez odcinków pionowych i powtarzających się współrzędnych x) oczekiwany rozmiar struktury przeszukiwań był liniowy, a czas zapytania o trapez zawierający punkt – logarytmiczny względem liczby odcinków w zbiorze, dla którego przygotowywana jest mapa trapezowa.

6. Testy

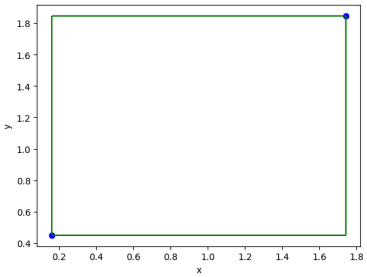
Algorytm został sprawdzony w testach. Zbiory danych wprowadzane były ręcznie, a także za pomocą funkcji generującej odcinki. Testy miały za zadanie sprawdzić poprawność konstrukcji mapy trapezowej oraz grafu przeszukiwań, lokalizacji punktu na utworzonej mapie oraz prawidłowość tworzenia wizualizacji.
Algorytm na testowanych przypadkach uzyskał wyniki zgodne z oczekiwaniami.

Poniżej przedstawiony jest jeden z testów, jakie zostały wykonane. Rysunek 6.1. przedstawia zadane odcinki. Szukanym punktem był punkt o współrzędnych (0.6, 1.0). Na rysunkach 6.2.-6.10. pokazana są kolejne etapy tworzenia mapy. Rysunek 6.11. przedstawia wynik lokalizacji podanego punktu. Zamieszczone wizualizacje pokazują, że algorytm uzyskał pozytywny wynik w tym teście.

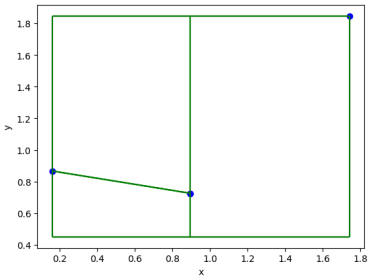


Rys. 6.1. Odcinki zadane w omawianym teście.

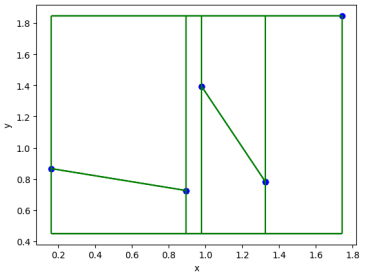
Rys. 6.2.-6.10. Kolejne etapy tworzenia mapy trapezowej w omawianym teście.



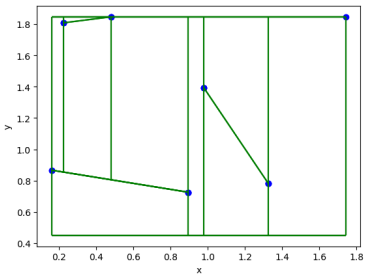
Rys. 6.2.



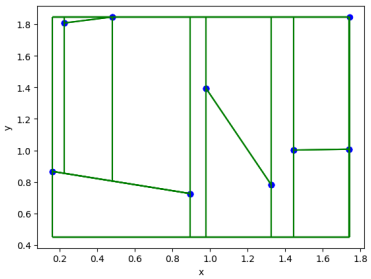
Rys. 6.3.



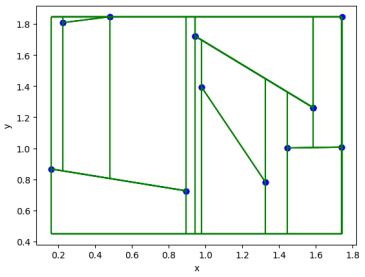
Rys. 6.4.



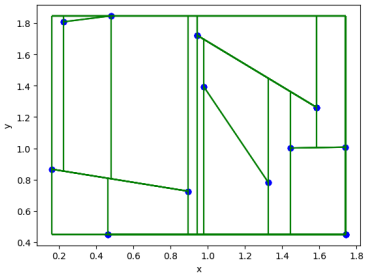
Rys. 6.5.



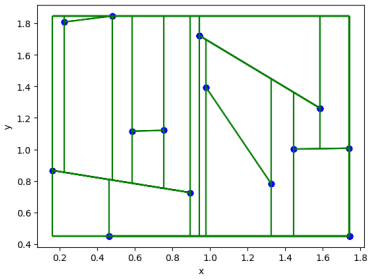
Rys. 6.6.



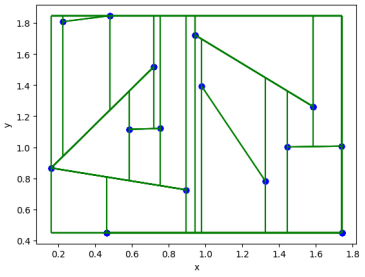
Rys. 6.7.



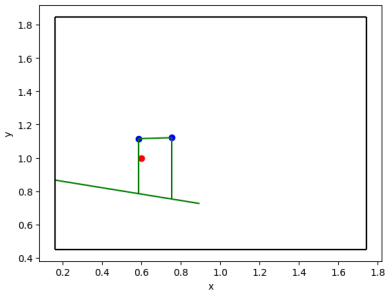
Rys. 6.8.



Rys. 6.9.



Rys. 6.10.



Rys. 6.11. Lokalizacja punktu o współrzędnych (0.6, 1.0) dla mapy trapezowej otrzymanej w omawianym teście.

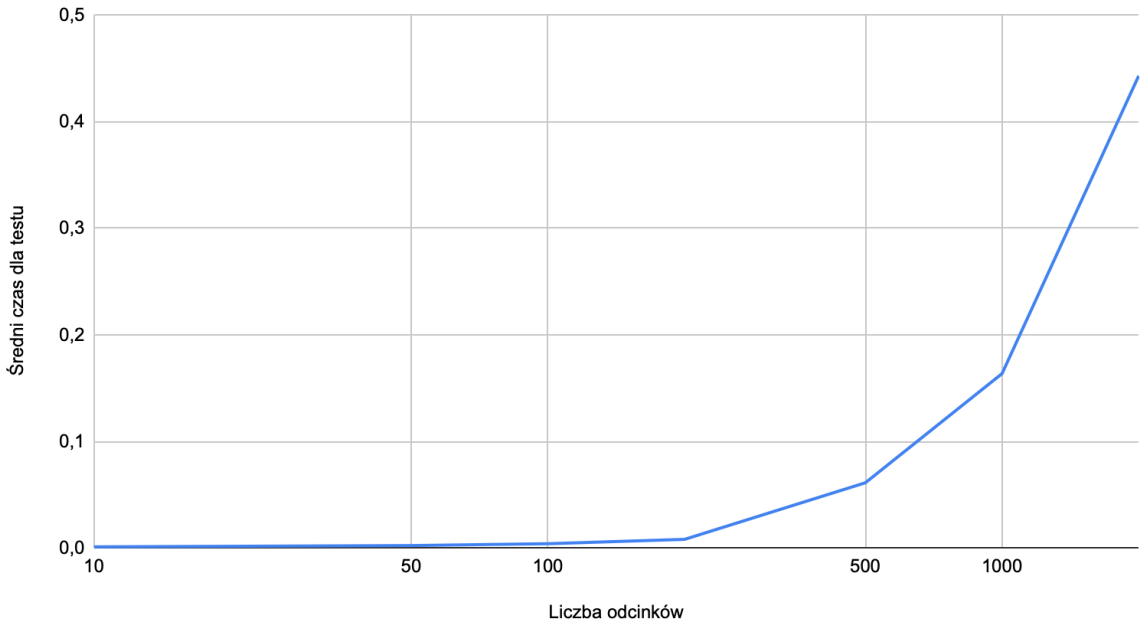
7. Czas trwania algorytmu

Wydajność algorytmu została sprawdzona na zbiorach danych różnej wielkości. Dla każdej liczby odcinków przeprowadzone zostały 3 testy, w których najpierw tworzona jest mapa trapezowa i graf przeszukiwań, a następnie wyszukiwany jest punkt. Funkcje służące do wizualizacji zostały wyłączone, by nie wpływały na uzyskane wyniki. Otrzymane czasy wykonania przedstawione są w poniższej tabeli 7.1. oraz wykresie 7.2.

Tabela 7.1. Czasy wykonania algorytmów dla przeprowadzonych testów

Liczba odcinków	Czas uzyskany w pierwszej próbie	Czas uzyskany w drugiej próbie	Czas uzyskany w trzeciej próbie	Średni czas dla testu
10	0,000499	0,002423	0,000312	0,001078
50	0,002151	0,003263	0,001397	0,00227
100	0,004877	0,003543	0,003569	0,003996
200	0,007992	0,007722	0,008609	0,008108
500	0,022492	0,136488	0,024527	0,061169
1000	0,149244	0,060561	0,280856	0,163554
2000	0,551031	0,353191	0,424566	0,442929

Wykres 7.2. Zależność czasu wykonania algorytmu od liczby odcinków



8. Bibliografia

- https://github.com/aghbit/Algorytmy-Geometryczne
- https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect08-trap-map.pdf
- https://faculty.sites.iastate.edu/jia/files/inline-files/13.%20trapezoidal%20maps.pdf
- https://youtube.com/playlist?list=PLubYOWSl9mIvTio-1bXWnhE9LdeXfox1z&si=Y4rtvJMhDcK7QziP