Algorytmy Geometryczne

Lokalizacja punktu w przestrzeni dwuwymiarowej
Metoda trapezowa

Oskar Blajsz
Maciej Wiśniewski
Grupa 3 Poniedziałek 16.45 A

Data wykonania 04.01.2025 Data oddania 05.01.2025

Sprawozdanie – część teoretyczna

1. Wprowadzenie

W ramach projektu przygotowano i zaimplementowano struktury oraz bazujący na nich algorytm, którego celem jest realizacja i wizualizacja **trapezowej** metody lokalizacji punktu na przestrzeni dwuwymiarowej. Początkowym etapem algorytmu jest obszar z **podziałem poligonowym**, wcześniej przygotowany tudzież zapodany przez użytkownika. Podany jest też dowolny **punkt P**, należący do wspomnianego poprzednio obszaru. Celem algorytmu jest znalezienie elementu, rozumianego tutaj jako **wielokąt**, do którego należy wybrany punkt. Dodatkowo zaimplementowano program realizujący **graficzną reprezentację** kroków omawianego algorytmu, tak aby algorytm mógł służyć jako **narzędzie dydaktyczne** objaśniające działanie algorytmu. Przeprowadzono również **analizę wydajnościową**, aby uwidocznić **efektywność** działania zaimplementowanego algorytmu.

2. Dane techniczne

Specyfikacja komputera na którym testowano algorytm: system Ubuntu 24.04.01 Linux 5.15 x64, procesor AMD Ryzen 7 5825U with Radeon 2GHz 8 rdzeni, 16GB pamięci RAM. Ćwiczenie zostało napisane w języku ***Python 3.9.20*** **w** ***Jupyter Notebook*** w środowisku programistycznym ***Visual Studio Code*****.** Aby wykonać ćwiczenie posłużono się bibliotekami: ***numpy, matplotlib, json, random i tkinter***. Do wykonania wizualizacji stworzono specjalne klasy oraz funkcje, które później zostaną szczegółowo opisane. Do procesu wizualizacja posłużono się również zmodyfikowanym narzędziem graficznym przygotowanym przez Koło Naukowe BIT.

3. Struktury przygotowane dla algorytmu i opis jego działania

3.1 Budowa mapy trapezowej

Dane wejściowe:
Zbiór odcinków S = {s1, s2, s3,…,sn} położonych na płaszczyźnie dwuwymiarowej w położeniu ogólnym

Wynik:
Mapa trapezowa T(S) oraz struktura przeszukiwań D dla T(S) w postaci grafu przeszukiwań

Oczekiwany czas konstrukcji mapy trapezowej - O(n log n)

Do budowy mapy trapezowej został wykorzystany randomizowany algorytm konstrukcji. Jego przebieg opisują dokładnie następujące kroki:

1. Wyznaczenie prostokąta zwierającego w sobie wszystkie odcinki z S.
2. Inicjalizacja mapy trapezowej T(S) oraz struktury przeszukiwań D

3. Dla każdego s_i:

1. Znalezienie zbioru trapezów $\Delta_0, \Delta_1, \dots, \Delta_k$ przeciętych przez s_i
2. Usunięcie znalezionych trapezów $\Delta_0, \Delta_1, \dots, \Delta_k$ z mapy trapezowej T(S)
3. Dodanie do T(S) nowo utworzonych trapezów, które stworzyły się po dodaniu odcinka s_i
4. Usunięcie liści zawierających trapezy $\Delta_0, \Delta_1, \dots, \Delta_k$ ze struktury przeszukiwań D
5. W strukturze przeszukiwań D stworzenie liści zawierających nowo utworzone trapezy i połączenie ich z istniejącymi węzłami, w razie potrzeby dodając nowe węzły

3.2 Wyznaczanie strefy dla odcinka s_i

Strefę dla odcinka s_i w mapach $T(S_i)$ i $T(S_{i-1})$ tworzą wszystkie trapezy przecinane przed odcinek s_i.

Pierwszym krokiem wyznaczanie strefy jest przeszukanie struktury D aż do znalezienia trapezu Δ_0, który zawiera lewy koniec p odcinka s_i.

Sposoby rozwiązywania wątpliwych sytuacji mogących wystąpić podczas przeszukiwania:

- Jeśli okaże się, że punkt p leży na prostej pionowej to przyjmujemy, że leży po jej prawej stronie
- Jeśli okaże się, że punkt p jest wspólnym punktem z innym odcinkiem z S, wtedy porównujemy nachylenie obu odcinków. Jeśli nachylenie si będzie mniejsze to przyjmujemy, że leży on poniżej drugiego odcinka.

Mając to na uwadze ze struktury D jest pozbierany trapez Δ_0.

Następnie rozpoczyna się wyznaczanie strefy dla odcinka s_i według poniższego algorytmu zapisanego w pseudokodzie:

j = 0
while q znajduje się na prawo od rightp(Δ_j) do:

jeśli rightp(Δ_j) znajduje się powyżej si, to:
$\Delta_{j+1} \leftarrow$ dolny prawy sąsiad Δ_j

w przeciwnym razie:
$\Delta_{j+1} \leftarrow$ górny prawy sąsiad Δ_j

$j \leftarrow j + 1$

zwróć $\Delta_0, \Delta_1, ..., \Delta_k$

3.3 Aktualizacja mapy trapezowej

Jeśli nowo dodawany odcinek s_i w całości zawiera się w jednym trapezie to aktualizacja mapy przebiega w następujących krokach:

1. Usuwamy trapez Δ zawierający odcinek si z mapy
2. Zastępujemy go przez odpowiednią liczbę nowych trapezów (maksymalnie 4)
3. Aktualizujemy informacje dla trapezów o sąsiadach, bottom(Δ),
4. top(Δ), leftp(Δ), rightp(Δ)

W przypadku, gdy nowo dodawany odcinek si zawiera się w więcej niż jednym trapezie to w celu aktualizacji mapy trapezowej musimy wykonać następujące kroki:

1. Wstawiamy rozszerzenia pionowe przechodzące przez oba końce odcinka si dzieląc trapezy Δ_0 i Δ_k.
2. Skracamy pionowe linie, które przecinają odcinek si, tak aby się z nim stykały.
3. Aktualizujemy informacje o sąsiadach dla zmienionych i utworzonych trapezów.

3.4 Algorytm wyszukiwania w grafie

Dane:
Punkt q zadany w postaci współrzędnych x i y

Wynik:
Trapez Δ zawierający punkt q

Algorytm:

- Jeśli bieżący element jest x-węzłem:
  - Jeśli szukany punkt znajduje się po lewej stronie pionowej linii przechodzącej przez punkt w x-węźle to przejdź do lewego potomka.
  - W przeciwnym przypadku przejdź do prawego potomka.
- Jeśli bieżący element jest y-węzłem:
  - Jeśli szukany punkt znajduje się poniżej poziomej linii w y-węźle to przejdź do lewego potomka.
  - W przeciwnym przypadku przejdź do prawego potomka.
- Jeśli bieżący element jest liściem to zakończ algorytm (znaleziono trapez, w którym znajduje się punkt).

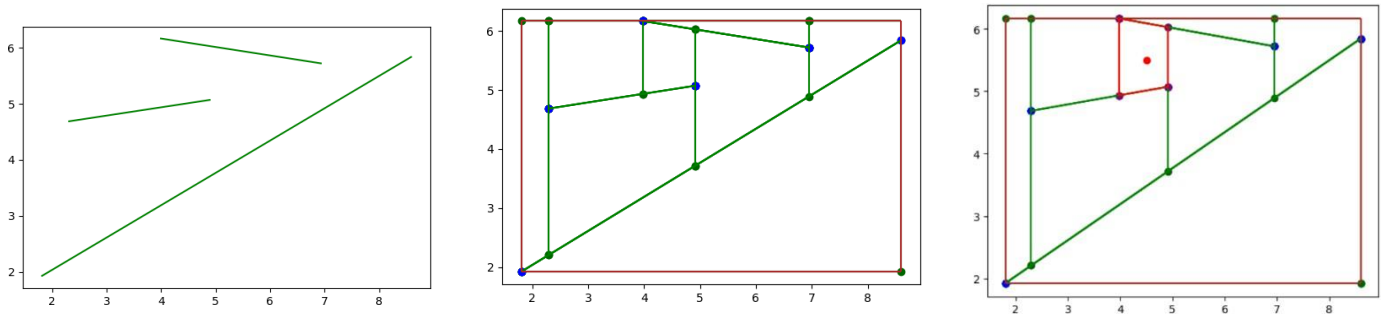
Implementacja grafu wyszukiwania opiera się na drzewie binarnym z różnymi typami węzłów. Dzięki temu oczekiwany czas wyszukiwania w grafie - O(log n), gdzie n to liczba liści grafu, zatem liczba trapezów w mapie trapezowej.

4. Wizualizacja

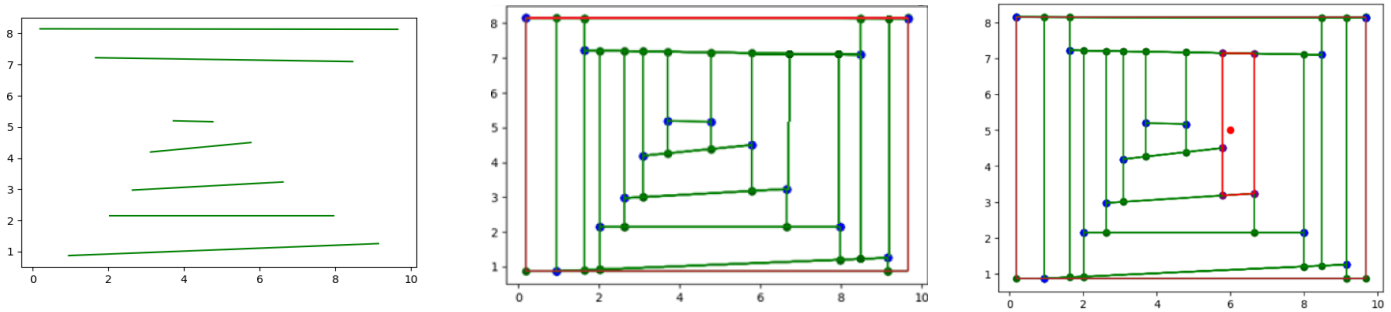
Aby zwizualizować działanie algorytmu zaimplementowano na początku zaimplementowano interaktywną funkcję do podawania odcinków poprzez rysowanie ich na płaszczyźnie. Implementację tej funkcji oparta na bibliotece ***tkinter***, ***matplotlib*** oraz na visualizerze stworzonym przez Koło Naukowe BIT. Dodatkowo dodano funkcje umożliwiające wczytywanie odcinków z plików zewnętrz jak również wpisywanie współrzędnych początku i końca odcinka ręcznie. Posłużono się również klasą ***Presenter***, która została zaimplementowana jako podstawa do krokowej wizualizacji etapów tworzenie mapy trapezowej. Zaimplementowano również funkcję, która umożliwia pokazania końcowego etapu znajdowania trapezu, w którym znajduje się szukany punkt.

5. Testy działania

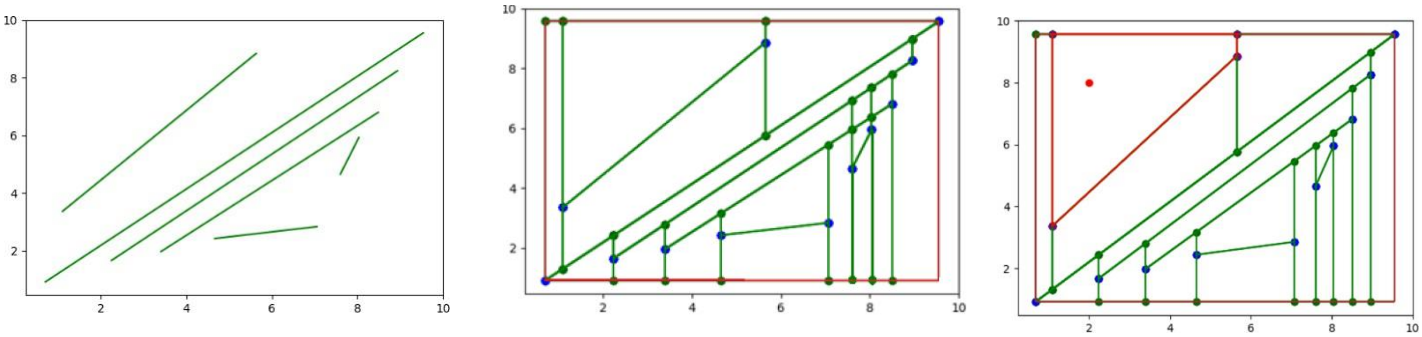
W celu przetestowania poprawnego działania programu przygotowano kilka zbiorów testowych, które mogły okazać się wymagające dla algorytmu. Aby przedstawić widoczna wizualizację zdecydowano się na zbiory zawierające małe liczby odcinków. Na rysunkach 1-7 przedstawiono działanie algorytmu dla różnych zbiorów danych, pierwsza część rysunku przedstawia wygenerowane odcinki, druga podział poligonowy(trapezowy), trzecia lokalizację punktu. Na wszystkich przykładach algorytm zadziałał prawidłowo.



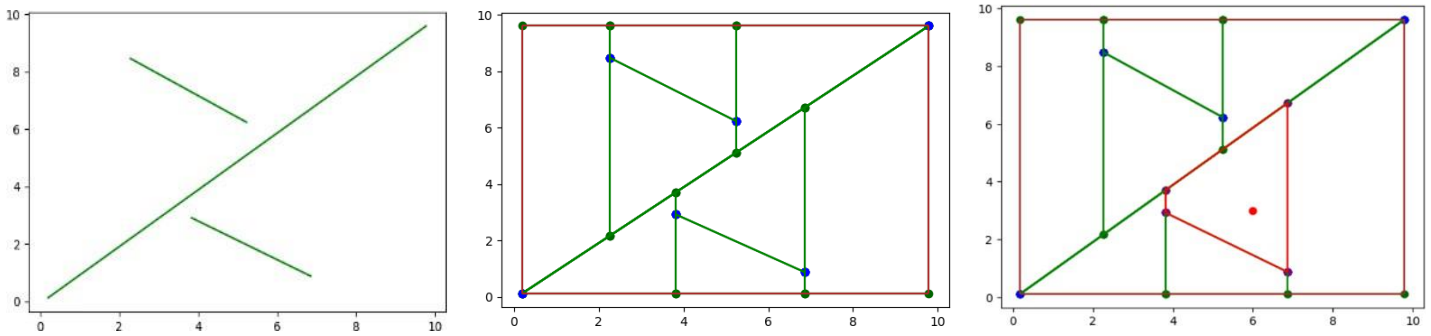
Rysunek 1 Etapy algorytmu dla Zbioru 1



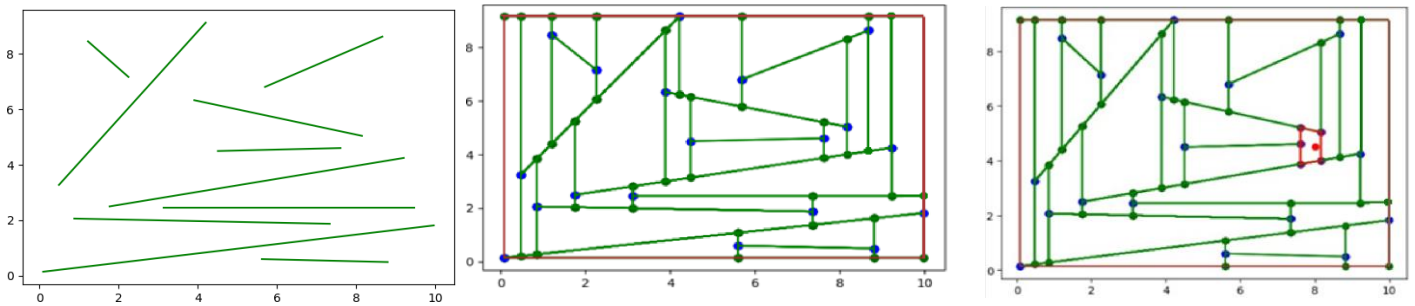
Rysunek 2 Etapy algorytmu dla Zbioru 2



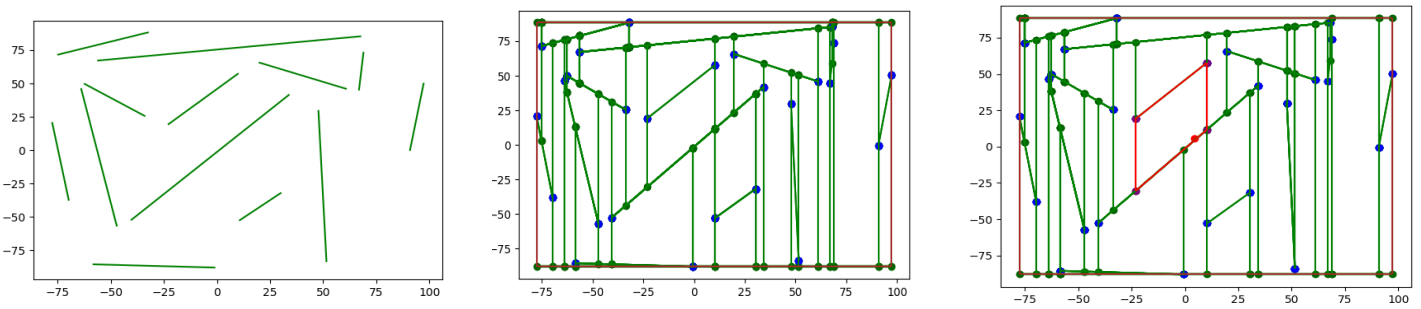
Rysunek 3 Etapy algorytmu dla Zbioru 3



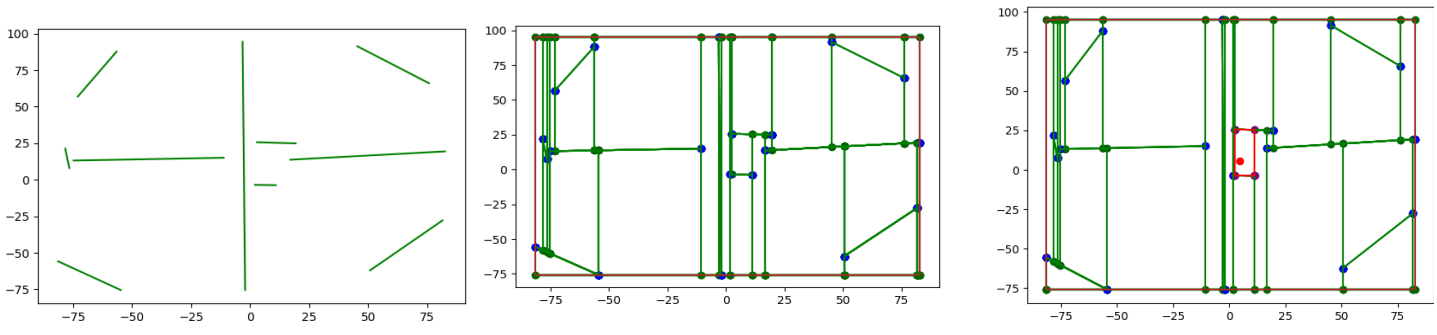
Rysunek 4 Etapy algorytmu dla Zbioru 4



Rysunek 5 Etapy algorytmu dla Zbioru 5



Rysunek 6 Etapy algorytmu dla Zbioru 6



Rysunek 7 Etapy algorytmu dla Zbioru 7

Motywacja wyboru przetestowanych zbiorów

Zbiory użyte w testach poprawności algorytmu zostały skonstruowane używając następujących kryteriów:

- **Zbiór 1 –** działanie algorytmu w najprostszym przypadku,
- **Zbiór 2 –** zbiór równoległych odcinków zmniejszających i zwiększających szerokość, sprawdzenie poprawności generowania trapezów, gdy odcinek prostopadły, wychodzący z wierzchołka, szybko natrafia na inne odcinki,
- **Zbiór 3 –** przypadek, kiedy większość odcinków znajduje się po jednej stronie prostokąta, czy algorytm poprawnie wychwyci pojedynczy odcinek znajdujący się po drugiej stronie przekątnej,
- **Zbiór 4 –** symetryczne umiejscowienie odcinków względem przekątnej, czy powstała mapa trapezowa będzie również symetryczna,
- **Zbiór 5 –** zachowanie algorytmu dla gęstego zbioru, czy zostaną znalezione wszystkie trapezy,

- **Zbiór 6 –** działanie algorytmu dla losowego zbioru odcinków spełniającego założenia,
- **Zbiór 7 –** zbiór, w którym szukany punkt znajduje się w małym trapezie, który to jest otoczony dużymi trapezami, czy algorytm poprawnie wytupuje małą przestrzeni pomiędzy większymi,

6. Testy wydajnościowe

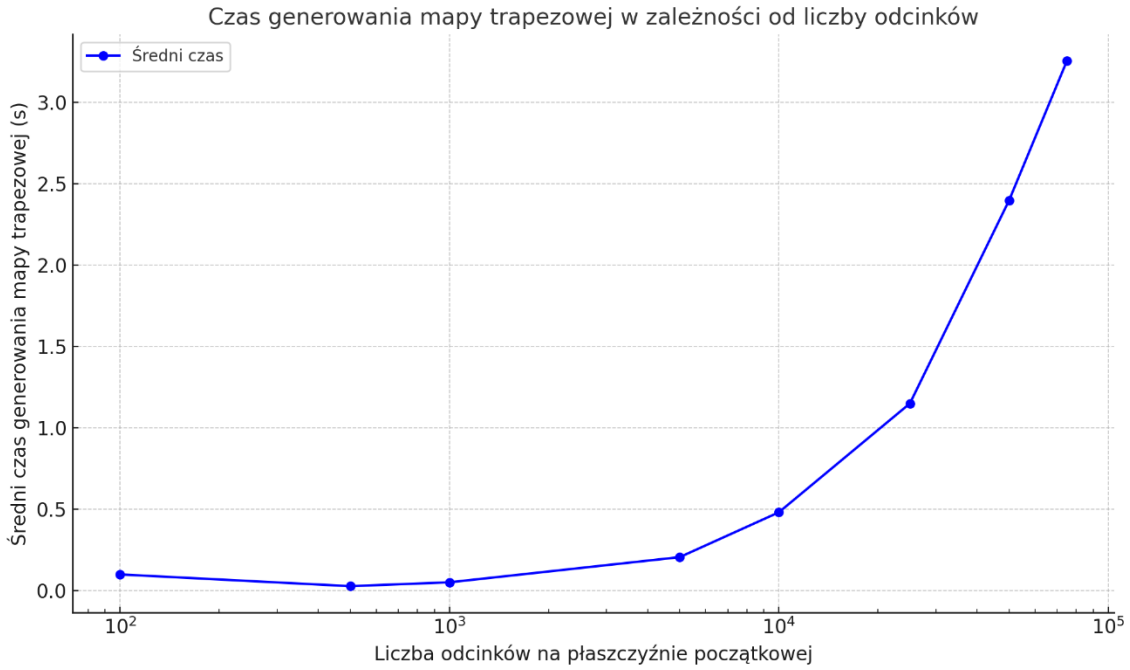
Aby określić wydajność algorytmu wykonano testy jego sprawności pod trzema względami : czasu budowy mapy trapezowej dla n punktów, rozmiaru struktury przeszukiwań dla n punktów oraz czas wyszukiwania trapezu zawierającego punkt.

Początkowo pod analizę efektywności poddano czas konstrukcji mapy trapezowej danej płaszczyzny. Do testów przygotowano różno-ilościowe zbiory odcinków. W celu uzyskania rzeczywistej efektywności algorytmu, pominięto w tej części wizualizacje, która i tak w tych przypadkach byłaby bardzo nieczytelna ze względu na duże stężenie odcinków. Aby zastosować się do wszystkich założeń programu generowany były tylko odcinki o takich samych wartościach współrzędnej y dla obu końców, a różnych wartościach współrzędnych x.

Wyniki przedstawiono w poniższej tabeli (Tabela 1) oraz na wykresie (Wykres 1):

Liczba odcinków na płaszczyźnie początkowej	**Czas generowania mapy trapezowej**			
	Próba 1	**Próba 2**	**Próba 3**	**Średnia**
100	0.0899 s	0.1010 s	0.1081 s	**0.0997 s**
500	0.0289 s	0.0259 s	0.0279 s	**0.0276 s**
1000	0.0462 s	0.0472 s	0.0604 s	**0.0513 s**
5000	0.2267 s	0.1966 s	0.1940 s	**0.2058 s**
10000	0.5387 s	0.5457 s	0.3088 s	**0.4811 s**
25000	1.4603 s	1.1363 s	0.8521 s	**1.1496 s**
50000	1.5314 s	3.4998 s	2.1576 s	**2.3963 s**
75000	2.214 s	2.9702s	4.5824 s	**3.2555 s**

Tabela 1. Wyniki analizy czasu generowania mapy trapezowej

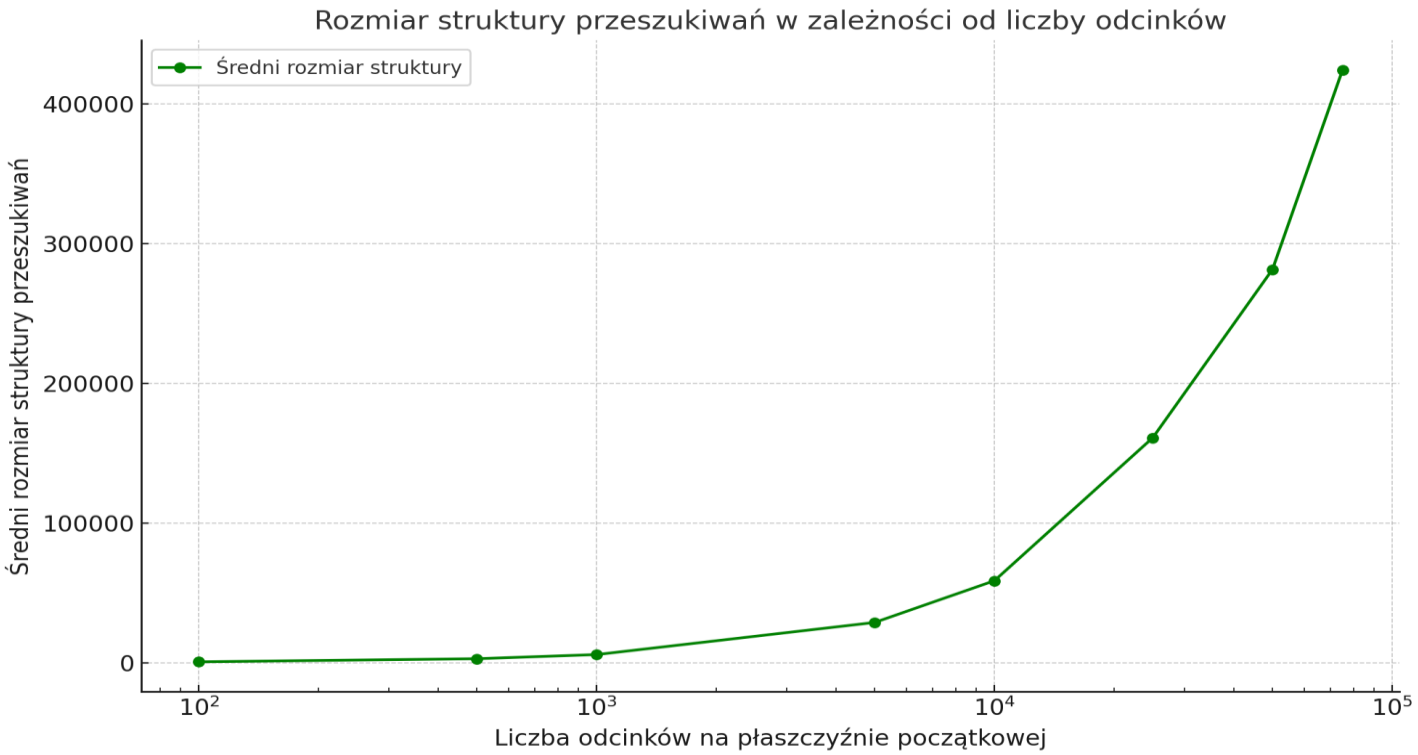


Wykres 1. Wyniki analizy czasu generowania mapy trapezowej w zależności od liczby odcinków.

Do testów wydajnościowych także przeanalizowano rozmiar struktury przeszukiwań, jako której rozmiar definiujemy jako jej liczbę węzłów (wszystkich, nie tylko liści). Wyniki drugiej analizy przedstawiony w poniższej tabeli (Tabela 2.) oraz na wykresie (Wykres 2):

Liczba odcinków	**Rozmiar struktury (liczba węzłów)**			
	Próba 1	**Próba 2**	**Próba 3**	**Średnia**
100	636	592	610	**613**
500	2747	2732	3017	**2832**
1000	5850	5989	5531	**5790**
5000	27711	27761	30853	**28775**
10000	54818	59227	61891	**58645**
25000	162712	167807	152211	**160910**
50000	259582	268175	316357	**281371**
75000	394055	459937	418360	**424117**

Tabela 2. Wyniki analizy rozmiaru struktury



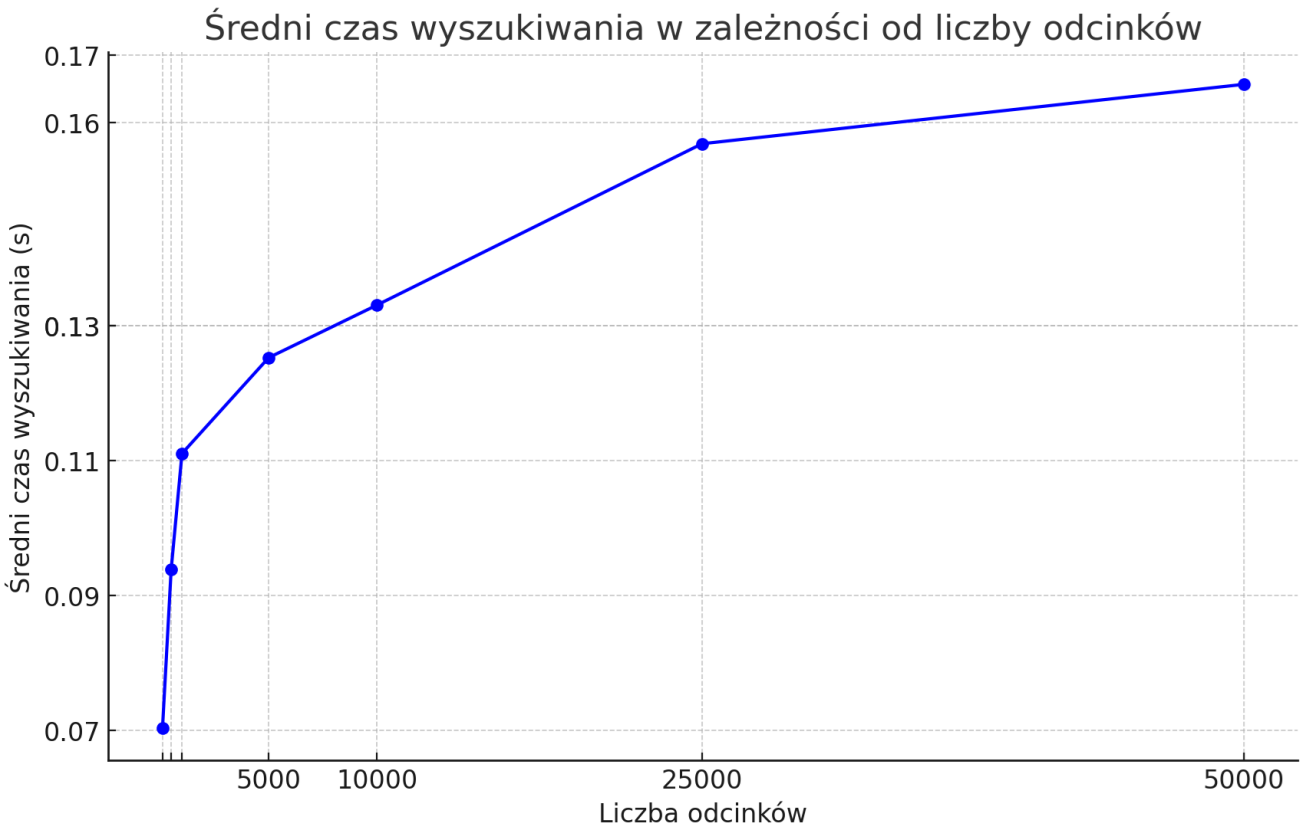
Wykres 2. Wyniki analizy rozmiaru struktury przeszukiwań w zależności od liczby odcinków

Jako ostatni element testów wydajnościowych, rozważono szybkość znajdowania 10000 w strukturach o różnych rozmiarach. Innymi słowy, czas w jakim algorytm znajduje trapez, w którym znajduje się punkt przypadkach dla różnych rozmiarów struktury przeszukiwania.

Wyniki tej analizy przedstawiono w poniższej tabeli (Tabela 3) oraz na wykresie (Wykres 3).

Liczba odcinków na mapie	**Liczba szukanych punktów**	**Czas znalezienia punktów**			
		Próba 1	**Próba 2**	**Próba 3**	**Średnia**
100	**10000**	0.0778 s	0.0662 s	0.0670 s	**0.0703 s**
500	**10000**	0.0926 s	0.0975 s	0.0914 s	**0.0938 s**
1000	**10000**	0.1137 s	0.1113 s	0.1081 s	**0.1110 s**
5000	**10000**	0.1220 s	0.1175 s	0.1360 s	**0.1252 s**
10000	**10000**	0.1298 s	0.1408 s	0.1284 s	**0.1330 s**
25000	**10000**	0.1546 s	0.2236 s	0.1524 s	**0.1569 s**
50000	**10000**	0.1636 s	0.1712 s	0.1623 s	**0.1657 s**
75000	**10000**	0.1765 s	0.1853 s	0.1685 s	**0.1768 s**

Tabela 3. Wyniki analizy czasu znajdowania punktów w różnych strukturach



Wykres 3. Wyniki analizy czasu znajdowania punktów w różnych strukturach w zależności od liczby odcinków

7. Podsumowanie i wnioski

Zaimplementowany algorytm trapezowej metody lokalizacji punktu działa poprawnie i wydajnie, co potwierdziły przeprowadzone testy funkcjonalne i wydajnościowe. Wizualizacje skutecznie ilustrują działanie algorytmu, czyniąc go przydatnym narzędziem dydaktycznym. Podsumowanie danych z tabel zaprezentowane na wykresach (Wykresy 1-3) jednoznacznie wskazuje na poprawność zaimplementowanych struktur i algorytmów, generowanie struktury odbywało się w czasie O(nlogn), rozmiar struktury osiągnął złożoność pamięciową rzędu O(n), a średni czas wyszukiwania odcinków odbywał się w czasie O(logn).

8. Bibliografia

https://algo.uni-trier.de/lectures/algeo/slides/ag-ws20-vl06-point-location.pdf
https://upel.agh.edu.pl/pluginfile.php/433098/mod_resource/content/1/wyklad_lokpkt_m.pdf
https://users.dimi.uniud.it/~claudio.mirolo/teaching/geom_comput/presentations/trapezoidal_map.pdf
http://cglab.ca/~cdillaba/comp5008/trapezoid.html

Dokumentacja – część techniczna

1. Uruchamianie

Przed uruchamianiem należy zapoznać się z plikami w folderze uruchamianie. Aby poprawnie uruchomić program należy uruchomić plik .ipynb w środowisku zawierającym również przygotowany visualizer, na przykład przy użyciu aplikacji Jupyter Notebook. Zalecanym środowiskiem jest środowisko bitalg, opis jego znajduje się w pliku srodowisko.txt. Do poprawnego działania program niezbędne są stosunkowo aktualne wersje programów i bibliotek, zostało to opisane w pliku requirements.txt. Aby program działał prawidłowo należy uruchomić wszystkie komórki kodu dostępne w pliku, zalecane jest kompilowanie pliku w ustalonej kolejności.

2. Przykłady uruchomienia

Przed uruchomieniem program należy się zastanowić, którą z dostępnych opcji chcielibyśmy użyć. Przed rozpoczęciem jakiekolwiek działania zaleca się skompilować wszystkie komórki powyżej akapitu „Tworzenie mapy trapezowej raz z wizualizacją” oraz uruchomienie pierwszej, następnej komórki odpowiedzialnej za modyfikację i stworzenie wcześniej przygotowanego wizualizera.
Jeśli naszym celem jest zapoznanie się z **przygotowanymi** już przykładami testowymi, wystarczy po kolei skompilować 5 komórek zatytułowanych „Przygotowane zbiory testowe”. Opisana operacja pozwoli nam zapoznać się z działaniem algorytmu tworzenia mapy trapezowej w formie krokowej. Następnie żeby zobaczyć wizualizację głównego algorytmu, należy znaleźć akapit ‘Wizualizacja głównego algorytmu’ i wykonywać po kolei komórki zatytułowane ‘Wizualizacje dla przykładowych zbiorów’.
Jeśli naszym celem jest **ręczne dodanie** płaszczyzny i późniejsza wizualizacja działania algorytmu na zapodanej płaszczyźnie, należy uruchomić komórkę zatytułowaną „Funkcja do interaktywnego wczytywania”. Po uruchomienie tej funkcji, powinno ukazać się nam program, który pozwala na wpisywanie ręcznie współrzędnych odcinków lub ich rysowanie. Po wczytywaniu wymyślonej konfiguracji, możemy kliknąć ‘q’ , wtedy współrzędne zadanych przez nas odcinków zostaną zapisane w programie i odpowiednio zmodyfikowane na potrzeby używanych struktur, możemy też kliknąć ‘s’, które umożliwi nam zapisanie wybranej konfiguracji do pliku formatu JSON. Aby zobaczyć wizualizację należy uruchomić komórkę zatytułowaną „Wizualizacja podziału ręcznie wprowadzonego zbioru”. Następnie żeby zobaczyć wizualizację głównego algorytmu, należy znaleźć akapit ‘Wizualizacja głównego algorytmu’ i wykonywać po kolei komórki zatytułowane ‘Wizualizacja algorytmu dla płaszczyzny stworzonej ręcznie’.

Jeśli chcemy wczytać dane z **plików formatu JSON**, należy uruchomić komórkę zatytułowaną „Wczytywanie z plików JSON”, aby funkcja działała poprawnie należy wprowadzić nazwę pliku JSON, z którego chcemy wczytywać i dodać ścieżkę dostępu do tego pliku (lub przenieść plik JSON do folderu z algorytmem). Aby zobaczyć wizualizację należy uruchomić komórkę zatytułowaną „Wizualizacja podziału płaszczyzny z pliku JSON”. Następnie żeby zobaczyć wizualizację głównego algorytmu, należy znaleźć akapit ‘Wizualizacja głównego algorytmu’ i wykonywać po kolei komórki zatytułowane ‘Wizualizacja algorytmu dla płaszczyzny z pliku JSON’.

3. Spis używanych bibliotek

Podczas realizacja algorytmu i jego wizualizacja posłużono się następującymi bibliotekami:
numpy – użyto do niektórych operacji matematycznych, ale także do testowania wydajności,
json – obsługa plików formatu JSON,
random – randomizacja, implementacja ‘losowości’
matplotlib – wizualizacja, funkcja do wczytywanie punktów
tkinter – algorytmiczna część funkcji do wczytywania punktów

4. Używane klasy i ich metody

Point

Przetrzymywanie danych o punkcie i operacje na punktach.

__init__(self, x, y)
Argumenty:

- x - współrzędna reprezentująca punkt na osi x.
- y - współrzędna reprezentująca punkt na osi y.

Wartość zwracana: brak
Opis:
Funkcja jest konstruktorem klasy i przypisuje współrzędne x i y obiektowi.
Przechowywane dane:

- x - współrzędna reprezentująca punkt na osi x.
- y - współrzędna reprezentująca punkt na osi y.

---

get_x(self)
Argumenty: brak
Wartość zwracana:
Współrzędna x punktu.
Opis:
Funkcja zwraca współrzędną x punktu.

---

get_y(self)
Argumenty: brak
Wartość zwracana:
Współrzędna y punktu.
Opis:
Funkcja zwraca współrzędną y punktu.

__gt__(self, other)
Argumenty:

- other - inny obiekt klasy Point.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy współrzędna x bieżącego punktu jest większa niż współrzędna x punktu other.

Opis:
Przeciążenie operatora > umożliwia porównywanie dwóch obiektów klasy Point na podstawie ich współrzędnych x.

---

to_tuple(self)
Argumenty: brak
Wartość zwracana:
Krotka (x, y) zawierająca współrzędne punktu.
Opis:
Funkcja zwraca reprezentację punktu w postaci krotki (x, y).

__str__(self)
Argumenty: brak
Wartość zwracana:
Łańcuch znaków w formacie "x,y", gdzie x i y to współrzędne punktu.
Opis:
Funkcja zwraca tekstową reprezentację punktu w formacie "x,y".

__hash__(self)
Argumenty: brak
Wartość zwracana:
Wartość skrótu (hash) obiektu, obliczona na podstawie współrzędnych x i y.
Opis:
Funkcja umożliwia użycie obiektu klasy Point jako klucza w strukturach danych opartych na funkcjach skrótu, takich jak słowniki czy zbiory.

Segment

Przetrzymywanie odcinków i operacje na nich.

__init__(self, point1, point2)
Argumenty:

- point1 - pierwszy punkt klasy Point definiujący segment.
- point2 - drugi punkt klasy Point definiujący segment.

Wartość zwracana: brak
Opis:
Funkcja jest konstruktorem klasy. Na podstawie dwóch punktów definiuje segment, określając, który z nich jest lewym, a który prawym końcem segmentu. Oblicza również parametry prostej (a i b), którą reprezentuje segment.
Przechowywane dane:

- left - punkt leżący bardziej na lewo (mniejsza wartość x).
- right - punkt leżący bardziej na prawo (większa wartość x).
- a - współczynnik kierunkowy prostej.
- b - wyraz wolny prostej.

_calculate_line_parameters(self)
Argumenty: brak
Wartość zwracana: brak
Opis:
Funkcja pomocnicza obliczająca parametry prostej (a i b) na podstawie współrzędnych punktów left i right.

---

set_reference_x(cls, x) *(metoda klasowa)*
Argumenty:

- x - współrzędna x używana jako punkt odniesienia do porównywania segmentów.

Wartość zwracana: brak
Opis:
Ustawia globalny punkt odniesienia x dla wszystkich obiektów klasy Segment.

---

get_y(self, x)
Argumenty:

- x - współrzędna x, dla której ma zostać obliczona wartość y na prostej segmentu.

Wartość zwracana:
Wartość y odpowiadająca współrzędnej x, jeśli x leży w zakresie segmentu. W przeciwnym razie zwraca None.
Opis:
Funkcja zwraca wartość y na prostej, którą reprezentuje segment, dla podanej współrzędnej x.

---

get_a(self)
Argumenty: brak
Wartość zwracana:
Współczynnik kierunkowy a prostej segmentu.
Opis:
Funkcja zwraca współczynnik kierunkowy a prostej, którą reprezentuje segment.

---

get_b(self)
Argumenty: brak
Wartość zwracana:
Wyraz wolny b prostej segmentu.
Opis:
Funkcja zwraca wyraz wolny b prostej, którą reprezentuje segment.

---

is_above(self, point)
Argumenty:

- point - obiekt klasy Point, względem którego sprawdzana jest pozycja.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy punkt point leży powyżej segmentu.
Opis:
Funkcja sprawdza, czy podany punkt leży powyżej prostej reprezentowanej przez segment.

__lt__(self, other)
Argumenty:

- other - inny obiekt klasy Segment.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy bieżący segment znajduje się poniżej innego segmentu w punkcie odniesienia Segment.x.
Opis:
Przeciążenie operatora < umożliwia porównywanie segmentów względem ich pozycji w punkcie Segment.x.

__gt__(self, other)
Argumenty:

- other - inny obiekt klasy Segment.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy bieżący segment znajduje się powyżej innego segmentu w punkcie odniesienia Segment.x.
Opis:
Przeciążenie operatora > umożliwia porównywanie segmentów względem ich pozycji w punkcie Segment.x.

__hash__(self)
Argumenty: brak
Wartość zwracana:
Wartość skrótu (hash) obiektu, obliczona na podstawie końcowych punktów segmentu.
Opis:
Funkcja umożliwia użycie obiektu klasy Segment jako klucza w strukturach danych opartych na funkcjach skrótu, takich jak słowniki czy zbiory.

---

to_tuple(self)
Argumenty: brak
Wartość zwracana:
Krotka zawierająca współrzędne punktów left i right w formacie ((left.x, left.y), (right.x, right.y)).
Opis:
Funkcja zwraca reprezentację segmentu w postaci krotki.

__eq__(self, other)
Argumenty:

- other - inny obiekt klasy Segment.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy dwa segmenty są równe.
Opis:
Funkcja sprawdza, czy dwa segmenty mają takie same końcowe punkty.

---

update_x(x) *(metoda statyczna)*
Argumenty:

- x - nowa współrzędna x używana jako punkt odniesienia do porównywania segmentów.

Wartość zwracana: brak
Opis:
Ustawia globalny punkt odniesienia x dla wszystkich obiektów klasy Segment.

Trapez

Reprezentacja i obsługa trapezów
__init__(self, top, bottom, p, q)
Argumenty:

- top - obiekt klasy Segment, reprezentujący górną krawędź trapezu.
- bottom - obiekt klasy Segment, reprezentujący dolną krawędź trapezu.
- p - obiekt klasy Point, reprezentujący lewy wierzchołek trapezu.
- q - obiekt klasy Point, reprezentujący prawy wierzchołek trapezu.

Wartość zwracana: brak
Opis:
Funkcja jest konstruktorem klasy, inicjalizującym trapez na podstawie jego górnej i dolnej krawędzi oraz lewego i prawego wierzchołka. Dodatkowo przygotowuje pola dla sąsiadów trapezu i węzła w strukturze danych.
Przechowywane dane:

- top_segment - górna krawędź trapezu (obiekt klasy Segment).
- bottom_segment - dolna krawędź trapezu (obiekt klasy Segment).
- left_point - lewy wierzchołek trapezu (obiekt klasy Point).
- right_point - prawy wierzchołek trapezu (obiekt klasy Point).
- top_left, bottom_left - sąsiednie trapezy z lewej strony (górny i dolny).
- top_right, bottom_right - sąsiednie trapezy z prawej strony (górny i dolny).
- node - węzeł w strukturze danych reprezentującej trapezy.

__str__(self)
Argumenty: brak
Wartość zwracana:
Łańcuch znaków w formacie "<bottom_segment> <top_segment>", gdzie bottom_segment i top_segment są tekstowymi reprezentacjami segmentów.
Opis:
Funkcja zwraca tekstową reprezentację trapezu, składającą się z jego dolnej i górnej krawędzi.

__eq__(self, other)
Argumenty:
- other - inny obiekt klasy Trapez.

Wartość zwracana:
Wartość logiczna (True lub False), określająca, czy dwa trapezy są identyczne.
Opis:
Funkcja sprawdza równość dwóch trapezów na podstawie ich górnej i dolnej krawędzi oraz lewego i prawego wierzchołka.

---

copy(self)
Argumenty: brak
Wartość zwracana:
Nowy obiekt klasy Trapez, będący kopią bieżącego trapezu.
Opis:
Funkcja tworzy kopię trapezu, zachowując wszystkie jego właściwości, w tym sąsiadów i węzeł w strukturze danych.

---

split_at_point(self, point)
Argumenty:
- point - obiekt klasy Point, na podstawie którego trapez ma zostać podzielony.

Wartość zwracana:
Krotka (left_trapez, right_trapez), gdzie:
- left_trapez - nowy trapez po lewej stronie punktu podziału.
- right_trapez - nowy trapez po prawej stronie punktu podziału.
  Jeśli punkt point nie leży w zakresie trapezu, zwracana jest krotka (None, None).

Opis:
Funkcja dzieli trapez na dwa nowe trapezy na podstawie współrzędnej x punktu point.

Rectangle

Reprezentacja prostokąta zawierającego całą płaszczyznę
create_rectangle(lines)
Argumenty:
- lines - lista linii, gdzie każda linia to para punktów w postaci krotek (x, y).

Wartość zwracana:
Obiekt klasy Trapez reprezentujący prostokąt zbudowany na podstawie dostarczonych linii.

Opis:
Funkcja statyczna generuje prostokąt na podstawie podanych linii, wyznaczając ekstremalne współrzędne x i y. Tworzy cztery wierzchołki prostokąta:

- low_left (dolny lewy),
- low_right (dolny prawy),
- up_left (górny lewy),
- up_right (górny prawy).

Na podstawie tych punktów tworzy dwa segmenty:

- top_segment - segment górny łączący wierzchołki up_left i up_right.
- bottom_segment - segment dolny łączący wierzchołki low_left i low_right.

Funkcja zwraca trapez, który reprezentuje prostokąt o podanych współrzędnych.

Node

Klasyczna definicja drzewa
__init__(self, node_type, data)
Argumenty:

- node_type - typ węzła, który może być używany do rozróżnienia funkcji węzła w strukturze danych.
- data - dane przechowywane w węźle, mogą być dowolnego typu w zależności od zastosowania.

Wartość zwracana: brak
Opis:
Funkcja jest konstruktorem klasy, inicjalizującym obiekt węzła z określonym typem i danymi. Ustawia również dwa atrybuty left i right jako None, które mogą być używane do reprezentowania dzieci węzła w strukturze drzewiastej.
Przechowywane dane:

- node_type - typ węzła, np. "liść", "wewnętrzny węzeł".
- data - dane przechowywane w węźle.
- left - lewy podwęzeł (domyślnie None).
- right - prawy podwęzeł (domyślnie None).

Tree

Struktura grafu przeszukań
__init__(self, root)
Argumenty:

- root - obiekt klasy Node, który jest korzeniem drzewa.

Wartość zwracana: brak
Opis:
Funkcja jest konstruktorem klasy, inicjalizującym drzewo za pomocą zadanego korzenia.
Przechowywane dane:

- root - korzeń drzewa (obiekt klasy Node).

---

update_root(self, root)
Argumenty:

- root - nowy korzeń drzewa (obiekt klasy Node).

Wartość zwracana: brak
Opis:
Funkcja aktualizuje korzeń drzewa, przypisując do niego nowy obiekt Node.

---

search(self, node, point, segment=None)

Argumenty:

- node - obiekt klasy Node, od którego zaczyna się wyszukiwanie.
- point - obiekt klasy Point, reprezentujący punkt do wyszukania.
- segment - opcjonalny argument (domyślnie None), segment do porównania, jeśli jest potrzebny.

Wartość zwracana:
Dane z węzła, w którym zakończyło się wyszukiwanie.
Opis:
Funkcja rekurencyjnie przeszukuje drzewo, porównując punkt z danymi w węzłach i przechodząc odpowiednią gałęzią. Jeśli węzeł jest typu l_type_node, zwraca dane. Jeśli jest typu x_type_node lub y_type_node, przechodzi do odpowiedniego podwęzła.

---

query(self, node, point, segment=None)
Argumenty:

- node - obiekt klasy Node, od którego zaczyna się zapytanie.
- point - obiekt klasy Point, reprezentujący punkt do zapytania.
- segment - opcjonalny argument (domyślnie None), segment do porównania, jeśli jest potrzebny.

Wartość zwracana:
Dane z węzła, który spełnia warunki zapytania.
Opis:
Funkcja rekurencyjnie przeszukuje drzewo, wykonując zapytanie w zależności od typu węzła (x_type_node lub y_type_node). Podejmuje decyzje na podstawie relacji punktu z danymi w węźle.

---

update_node_both_sides(trapezoid, segment, left, top, bottom, right)
Argumenty:

- trapezoid - obiekt klasy Trapez, w którym aktualizowane są węzły.
- segment - obiekt klasy Segment, który będzie zaktualizowany w węźle.
- left, top, bottom, right - obiekty klasy Point, reprezentujące punkty trapezu.

Wartość zwracana: brak

Opis:
Funkcja aktualizuje węzeł dla trapezu, tworząc i łącząc odpowiednie węzły w drzewie.

---

update_node_right_side(trapezoid, segment, top, bottom, right)
Argumenty:

- trapezoid - obiekt klasy Trapez, w którym aktualizowane są węzły.
- segment - obiekt klasy Segment, który będzie zaktualizowany w węźle.
- top, bottom, right - obiekty klasy Point, reprezentujące punkty trapezu.

Wartość zwracana: brak
Opis:
Funkcja aktualizuje węzeł po prawej stronie trapezu, tworząc i łącząc odpowiednie węzły w drzewie.

---

update_node_left_side(trapezoid, segment, left, top, bottom)
Argumenty:

- trapezoid - obiekt klasy Trapez, w którym aktualizowane są węzły.
- segment - obiekt klasy Segment, który będzie zaktualizowany w węźle.
- left, top, bottom - obiekty klasy Point, reprezentujące punkty trapezu.

Wartość zwracana: brak
Opis:
Funkcja aktualizuje węzeł po lewej stronie trapezu, tworząc i łącząc odpowiednie węzły w drzewie.

---

update_node_no_sides(trapezoid, segment, top, bottom)
Argumenty:

- trapezoid - obiekt klasy Trapez, w którym aktualizowane są węzły.
- segment - obiekt klasy Segment, który będzie zaktualizowany w węźle.
- top, bottom - obiekty klasy Point, reprezentujące punkty trapezu.

Wartość zwracana: brak
Opis:
Funkcja aktualizuje węzeł, nie tworząc nowych gałęzi po obu stronach trapezu, tworząc odpowiednie węzły w drzewie.

---

update_node_multiple(trapezoids, segment, new_above, new_below, left, right)
Argumenty:

- trapezoids - lista obiektów klasy Trapez.
- segment - obiekt klasy Segment, który będzie zaktualizowany w węzłach.
- new_above - lista nowych trapezów znajdujących się powyżej.
- new_below - lista nowych trapezów znajdujących się poniżej.
- left, right - obiekty klasy Point, reprezentujące punkty trapezu.

Wartość zwracana: brak
Opis:
Funkcja aktualizuje węzły dla wielu trapezów, tworząc odpowiednie gałęzie i łącząc węzły w drzewie.

TrapezoidalMap

Przechowywanie i operacje na mapie trapezowej
__init__(self)
Argumenty:
Brak
Wartość zwracana:
Brak
Opis:
Funkcja jest konstruktorem klasy. Inicjuje obiekt klasy TrapezoidMap i ustawia atrybut tree na None. Atrybut ten będzie później wykorzystywany do przechowywania drzewa trapezoidów, w którym segmenty będą wstawiane.

---

insertIntoOne(T, trapezoid, segment)
Argumenty:

- trapezoid - obiekt klasy Trapez, do którego wstawiany jest nowy segment.
- segment - obiekt klasy Segment, który ma zostać wstawiony do trapezoidu. Wartość

zwracana:
Brak
Opis:
Funkcja wstawia segment do pojedynczego trapezoidu, dzieląc go na dwa nowe trapezoidy. Segment jest wstawiany w taki sposób, aby nie naruszyć struktury trapezoidu. Funkcja decyduje, czy segment przecina trapezoid po lewej, czy po prawej stronie, i odpowiednio aktualizuje sąsiednie trapezoidy.

- Nowo utworzone trapezoidy są przechowywane w zmiennych top i bottom (górny i dolny).
- Po wstawieniu segmentu, sąsiednie trapezoidy po lewej i prawej stronie mogą zostać zaktualizowane.
- Funkcja aktualizuje drzewo trapezoidów, aby odzwierciedlić zmiany.

---

insertIntoMany(T, trapezoids, segment)
Argumenty:

- trapezoids - lista obiektów klasy Trapez, które będą dzielone przez nowy segment.
- segment - obiekt klasy Segment, który ma zostać wstawiony do wielu trapezoidów.

Wartość zwracana:
Brak
Opis:
Funkcja wstawia segment do wielu trapezoidów w mapie trapezoidów. Segment dzieli istniejące trapezoidy na mniejsze trapezoidy, które znajdują się powyżej i poniżej segmentu. Proces wstawiania jest iteracyjny, gdzie dla każdego trapezoidu sprawdzane jest, czy segment znajduje się nad, czy poniżej trapezoidu, a odpowiednio tworzone są nowe trapezoidy.

- Funkcja tworzy dwa typy nowych trapezoidów: jeden powyżej segmentu, a drugi poniżej segmentu.
- Zmienna merge śledzi, który z nowych trapezoidów jest "wyższy" lub "niższy" względem segmentu.
- Wszystkie nowo utworzone trapezoidy są przechowywane w listach newTrapezoidsAbove i newTrapezoidsBelow.
- Funkcja aktualizuje sąsiednie trapezoidy i wstawia nowe trapezoidy do drzewa.

Presenter

Wizualizacja w wersji krokowej
__init__(self, scenes)
Argumenty:

- scenes - lista obiektów, które reprezentują różne sceny w prezentacji. Każda scena zawiera dane do wyświetlenia i dane do rysowania na wykresach.

Wartość zwracana:
Brak
Opis:
Konstruktor klasy inicjuje obiekt Presenter. Inicjuje on dwie listy:

- self.scenes – lista przechowująca dane do wizualizacji.
- self.scene_data – lista przechowująca dane do rysowania (np. wykresy).

Ponadto, przypisuje self.i jako indeks ostatniej sceny. Ustawia również marginesy wykresów przy pomocy plt.subplots_adjust(bottom=0.2).

---

configure_buttons(self) Argumenty:
Brak
Wartość zwracana:
Lista z przyciskami (Button), które umożliwiają nawigację po scenach.
Opis:
Funkcja tworzy i konfiguruje przyciski do nawigacji między scenami. Tworzy dwa przyciski:

- Następny – umożliwia przejście do następnej sceny.
- Poprzedni – umożliwia powrót do poprzedniej sceny.

Przyciski są przypisane do metod next i prev, które są wywoływane po ich kliknięciu. Po skonfigurowaniu przycisków, funkcja zwraca je w postaci listy.

---

draw(self)
Argumenty:
Brak
Wartość zwracana:
Brak
Opis:
Funkcja rysuje bieżącą scenę na wykresie. Najpierw czyści obecny wykres (za pomocą self.ax.clear()), a następnie dla każdego obiektu w aktualnej scenie (dzięki self.scenes[self.i]) wywołuje metodę draw dla rysowania danego obiektu na wykresie. Na końcu funkcja ustawia autoskalowanie wykresu i wykonuje rysowanie przy pomocy plt.draw().

---

next(self, event)
Argumenty:

- event - obiekt reprezentujący zdarzenie wywołane przez kliknięcie przycisku.

Wartość zwracana:
Brak
Opis:
Funkcja służy do przejścia do następnej sceny. Indeks sceny (self.i) jest aktualizowany w sposób cykliczny, czyli po osiągnięciu ostatniej sceny, wracamy do pierwszej. Następnie funkcja wywołuje metodę draw, aby narysować nową scenę.

---

prev(self, event)
Argumenty:

- event - obiekt reprezentujący zdarzenie wywołane przez kliknięcie przycisku.

Wartość zwracana:
Brak
Opis:
Funkcja służy do przejścia do poprzedniej sceny. Indeks sceny (self.i) jest aktualizowany w sposób cykliczny, tzn. po osiągnięciu pierwszej sceny, przechodzi do ostatniej. Następnie funkcja wywołuje metodę draw, aby narysować poprzednią scenę.

---

set_axes(self, ax)
Argumenty:

- ax - obiekt typu Axes, który będzie używany do rysowania wykresu.

Wartość zwracana:
Brak
Opis:
Funkcja ustawia oś (ax) na której będą rysowane wszystkie sceny. Funkcja ta jest wywoływana, gdy chcemy przypisać oś do prezentera.

---

display(self)
Argumenty:
Brak
Wartość zwracana:
Brak
Opis:
Funkcja wyświetla okno z prezentacją. Zamyka bieżące okno wykresu, tworzy nowe okno za pomocą plt.figure(), konfiguruje przyciski nawigacyjne za pomocą funkcji __configure_buttons, a następnie wywołuje plt.show() do wyświetlenia okna. Na końcu funkcja rysuje pierwszą scenę, wywołując metodę draw.

5. Funkcje i algorytmy

W tym akapicie znajduje się techniczny opis zaimplementowanych funkcji i algorytmów, bardziej szczegółowy sposób ich działania został przedstawiony wcześniej w części teoretycznej i na dołączonej prezentacji.

Funkcja create_trapezoidal_map
Argumenty:

- segments: Lista segmentów, które będą użyte do tworzenia mapy trapezoidalnej.

Wartość zwracana:

- Zwraca drzewo trapezoidalne T, które reprezentuje wynikową mapę trapezoidalną.

Opis:
Funkcja create_trapezoidal_map buduje mapę trapezoidalną na podstawie podanych segmentów.

1. **Przygotowanie segmentów:**
   - Dla każdego segmentu w liście segments tworzy obiekty klasy Segment, przypisując im odpowiednie punkty (początkowy i końcowy).
2. **Tworzenie prostokąta:**
   - Tworzy prostokąt, który obejmuje wszystkie segmenty, używając metody Rectangle.create_rectangle.
3. **Tworzenie drzewa trapezoidalnego:**
   - Tworzy korzeń drzewa trapezoidalnego (Node), oparty na prostokącie, a następnie tworzy drzewo (Tree) z tym korzeniem.
4. **Dodawanie segmentów do drzewa:**
   - Dla każdego segmentu:
     - Przeszukuje drzewa trapezoidalne, aby znaleźć przecięte trapezy.
     - Segment jest dodawany do odpowiednich trapezów w drzewie. Jeśli segment przecina jeden trapez, zostaje dodany do niego. Jeśli przecina więcej niż jeden trapez, segment jest dodawany do kilku trapezów.
5. **Zwracanie wyniku:**
   - Zwraca drzewo trapezoidalne T, które zawiera wszystkie trapezy po dodaniu wszystkich segmentów.

Funkcja trapezoidal_map_vis
Argumenty:

- segments: Lista segmentów, które będą wykorzystywane do stworzenia mapy trapezoidalnej.

Wartość zwracana:

- Zwraca drzewo trapezoidalne T oraz listę scen scenes, które wizualizują kolejne etapy algorytmu.

Opis:

Funkcja trapezoidal_map_vis generuje wizualizację mapy trapezoidalnej. Przetwarza segmenty i dodaje je do struktury drzewa trapezoidalnego, a następnie wizualizuje proces tworzenia tej struktury.

1. **Wizualizacja początkowa**:
   - Tworzy obiekt Visualizer, który służy do rysowania.
   - Rysuje wszystkie segmenty w kolorze zielonym i zapisuje wizualizację jako pierwszą scenę.
2. **Tworzenie segmentów i drzewa**:
   - Przekształca każdą parę punktów segmentu na obiekty klasy Point, a następnie tworzy obiekty klasy Segment.
   - Tworzy prostokąt obejmujący wszystkie segmenty za pomocą Rectangle.create_rectangle.
   - Tworzy korzeń drzewa trapezoidalnego oraz inicjalizuje drzewo T.
3. **Dodawanie segmentów do mapy trapezoidalnej**:
   - Dla każdego segmentu:
     - Rysuje prostokąt i segmenty w bieżącej wizualizacji.
     - Przeszukuje drzewo w celu znalezienia trapezów, które są przecięte przez segment.

        - Wizualizuje te trapezy w kolorze fioletowym, a sam segment w kolorze czerwonym.
        - Na podstawie wyniku dodaje segment do odpowiednich trapezów w drzewie.
    4. **Zwracanie wyniku**:
        - Zwraca drzewo trapezoidalne T oraz listę wizualizacji scenes, które przedstawiają kolejne etapy procesu.

Funkcja findPointVisualised
Argumenty:

- lines: Lista segmentów, które tworzą mapę trapezoidalną.
- point: Punkt klasy Point, który ma być zlokalizowany w obrębie mapy trapezoidalnej.

Wartość zwracana:

- Brak (wizualizacja jest wyświetlana na ekranie).

Opis:
Funkcja findPointVisualised lokalizuje punkt na mapie trapezoidalnej. Pozycja punktu jest wyświetlana w kontekście utworzonych trapezów.

1. **Tworzenie mapy trapezoidalnej**:
    - Wywołuje funkcję trapezoidal_map_vis, aby stworzyć mapę trapezoidalną z segmentów.
2. **Lokalizacja punktu**:
    - Wyszukuje trapez, który zawiera podany punkt, za pomocą funkcji T.query.
    - Wizualizuje mapę trapezoidalną, dodaje punkt do wizualizacji w kolorze czerwonym, a także wizualizuje trapez, do którego punkt należy.

Funkcja draw_map
Argumenty:

- T: Drzewo trapezoidalne, które zawiera wszystkie trapezy.
- B: Prostokąt (obiekt klasy Trapez), który stanowi granice mapy.

Wartość zwracana:

- Zwraca obiekt wizualizacji, który przedstawia całą mapę trapezoidalną.

Opis:
Funkcja draw_map rysuje mapę trapezoidalną na podstawie danych zawartych w drzewie trapezoidalnym. Wizualizuje trapezy oraz prostokąt graniczny.

1. **Wyszukiwanie trapezów**:
    - Wyszukuje wszystkie trapezy w drzewie T poprzez rekurencyjne wywołanie funkcji find_all_trapezoids.
2. **Rysowanie trapezów i siatki**:
    - Każdy trapez jest rysowany za pomocą funkcji draw_trapezoid.
    - Dodaje również siatkę graniczną prostokąta za pomocą funkcji draw_grid.

Funkcja find_all_trapezoids
Argumenty:

- node: Węzeł drzewa trapezoidalnego, który może być typu 'l_type_node' lub inny.
- trapezoids: Lista, do której będą dodawane trapezy.

Wartość zwracana:

- Brak (lista trapezów jest aktualizowana).

Opis:
Funkcja find_all_trapezoids przechodzi przez całe drzewo trapezoidalne i dodaje trapezy (węzły typu 'l_type_node') do listy trapezoids.

1. **Rekurencyjne przetwarzanie węzłów**:
    - Jeśli węzeł jest typu 'l_type_node', dodaje jego dane (trapez) do listy trapezoids.
    - W przeciwnym razie, wywołuje funkcję rekurencyjnie dla lewego i prawego dziecka węzła.

Funkcja draw_grid
Argumenty:

- R: Prostokąt (obiekt klasy Trapez), który stanowi granice mapy.
- vis: Obiekt wizualizacji, który będzie modyfikowany.

Wartość zwracana:

- Zwraca zaktualizowaną wizualizację.

Opis:
Funkcja draw_grid rysuje siatkę graniczną prostokąta na mapie trapezoidalnej. Dodaje linie tworzące granice prostokąta.

Funkcja draw_trapezoid
Argumenty:

- trapezoid: Trapez (obiekt klasy Trapez), który ma zostać narysowany.
- vis: Obiekt wizualizacji, który będzie modyfikowany.
- color: Kolor, w jakim ma być narysowany trapez (domyślnie zielony).

Wartość zwracana:

- Zwraca zaktualizowaną wizualizację.

Opis:
Funkcja draw_trapezoid rysuje trapez, biorąc pod uwagę jego górny i dolny segment oraz punkty graniczne (lewy i prawy).

Funkcja draw_trapezoid_query
Argumenty:

- trapezoid: Trapez (obiekt klasy Trapez), który ma zostać narysowany.
- vis: Obiekt wizualizacji, który będzie modyfikowany.
- color: Kolor, w jakim ma być narysowany trapez (domyślnie czerwony).

Wartość zwracana:

- Zwraca zaktualizowaną wizualizację.

Opis:
Funkcja draw_trapezoid_query rysuje trapez w odpowiedzi na zapytanie, rysując jego granice oraz zaznaczając punkty graniczne.