Kopiec Kacper
Zwonek Aleksandra **05.01.2025**

Algorytmy geometryczne
Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda trapezowa

1 Część techniczna

1.1 Moduły wykorzystane w programie

- Matplotlib - moduł odpowiedzialny za obsługę interfejsu graficznego,
- Tkinter - moduł odpowiedzialny za interfejs graficzny do rysowania odcinków,
- Numpy - moduł do operacji matematycznych, takich jak generowanie losowej permutacji zbioru.
- Pillow - moduł do tworzenia gifów

1.2 Struktury danych

1.2.1 Dokumentacja klasy Point

Klasa `Point` implementuje punkt w przestrzeni dwuwymiarowej.

- `__init__(self, x: float, y: float)`
  Inicjalizuje obiekt klasy `Point` z podanymi współrzędnymi x i y.
  Parametry:
  `x` (int): Współrzędna x punktu.
  `y` (int): Współrzędna y punktu.
- `__eq__(self, other)`
  Porównuje dwa punkty pod kątem równości współrzędnych.
  Zwraca:
  `bool`: `True`, jeśli punkty mają identyczne współrzędne, w przeciwnym razie `False`.
- `__gt__(self, other)`
  Sprawdza, czy bieżący punkt jest "większy"od drugiego punktu. Porównanie odbywa się najpierw po współrzędnej x, a w przypadku ich równości po współrzędnej y.
  Zwraca:
  `bool`: `True`, jeśli bieżący punkt jest większy, w przeciwnym razie `False`.
- `__hash__(self)`
  Oblicza wartość hash punktu, co umożliwia wykorzystanie go jako klucz w strukturze takiej jak `set`.
  Zwraca:
  `int`: Wartość hash punktu.
- `__str__(self)`
  Zwraca reprezentację tekstową punktu w formacie `(x, y)`.
  Zwraca:
  `str`: Tekstowa reprezentacja współrzędnych punktu.
- `to_tuple(self)`
  Konwertuje punkt do krotki (`tuple`) zawierającej współrzędne x i y.
  Zwraca:
  `tuple`: Krotka z współrzędnymi punktu w formacie `(x, y)`.

1.2.2 Dokumentacja klasy Segment

Klasa `Segment` implementuje odcinek w przestrzeni dwuwymiarowej.

- `__init__(self, p1: Point, p2: Point)`
  Inicjalizuje obiekt klasy `Segment` na podstawie dwóch punktów końcowych odcinka. Współczynniki równania ogólnego prostej $Ax + By + C = 0$ zostają znormalizowane.
  Parametry:
  `p1` (Point): Pierwszy punkt końcowy odcinka.
  `p2` (Point): Drugi punkt końcowy odcinka.
- `value_for_x(self, x: float)`
  Oblicza punkt na odcinku o zadanej współrzędnej x.
  Parametry:
  `x` (float): Współrzędna x punktu.
  Zwraca:
  `Point`: Punkt na odcinku o podanej współrzędnej x.
- `point_over_segment(self, p: Point, eps: float = 1e-8)`
  Określa względne położenie punktu względem odcinka (powyżej, poniżej lub na odcinku). Do określenia położenia obliczany jest wyznacznik macierzy 3x3.
  Parametry:
  `p` (Point): Punkt, którego pozycję względem odcinka chcemy określić.
  `eps` (float): Tolerancja dla współliniowości.
  Zwraca:
  `int`:
  – `1` jeśli punkt jest powyżej odcinka,
  – `-1` jeśli poniżej,
  – `0` jeśli leży na odcinku.
- `__eq__(self, other)`
  Porównuje dwa odcinki na podstawie ich końców.
  Zwraca:
  `bool`: `True`, jeśli końce obu odcinków są identyczne, w przeciwnym razie `False`.
- `__hash__(self)`
  Oblicza wartość hash odcinka, co umożliwia wykorzystanie go jako klucz w strukturze takiej jak `set`.
  Zwraca:
  `int`: Wartość hash odcinka.
- `__str__(self)`
  Zwraca reprezentację tekstową odcinka w formacie `(left, right)`.
  Zwraca:
  `str`: Tekstowa reprezentacja współrzędnych punktów końcowych odcinka.

1.2.3 Dokumentacja klasy Trapezoid

Klasa `Trapezoid` implementuje trapez w przestrzeni dwuwymiarowej.

- `__init__(self, left: Point, right: Point, top: Segment, bottom: Segment)`
  Inicjalizuje obiekt klasy `Trapezoid` na podstawie dwóch punktów skrajnych oraz dwóch odcinków definiujących jego górną i dolną krawędź.
  Parametry:
  `left` (Point): Lewy punkt trapezu.
  `right` (Point): Prawy punkt trapezu.
  `top` (Segment): Odcinek zawierający górną krawędź trapezu.

`bottom` (Segment): Odcinek zawierający dolną krawędź trapezu.

- `trapezoidBoundary(self)` Oblicza współrzędne wierzchołków trapezu. Punkty są wyznaczane jako przecięcia lewego i prawego ograniczenia trapezu z jego dolną i górną krawędzią.
  Zwraca:
  `tuple`: Krotka zawierająca współrzędne wierzchołków w formacie `(A, B, C, D)`, gdzie:
  - `A`: Punkt przecięcia lewej krawędzi z dolną krawędzią.
  - `B`: Punkt przecięcia lewej krawędzi z górną krawędzią.
  - `C`: Punkt przecięcia prawej krawędzi z dolną krawędzią.
  - `D`: Punkt przecięcia prawej krawędzi z górną krawędzią.
- `__hash__(self)`
  Oblicza wartość hash trapezu, co umożliwia wykorzystanie go jako klucz w strukturze takiej jak `set`.
  Zwraca:
  `int`: Wartość hash trapezu, obliczana na podstawie jego atrybutów `left`, `right`, `top`, `bottom`.
- `__str__(self)`
  Zwraca reprezentację tekstową trapezu w formacie `lp:[left], rp:[right], top:[top], bot:[bottom]`.
  Zwraca:
  `str`: Tekstowa reprezentacja punktów i odcinków definiujących trapez.

1.2.4 Dokumentacja klasy PointNode

Klasa `PointNode` reprezentuje wierzchołek struktury przeszukiwań zawierający punkt jako dane. Służy do przechowywania punktów i definiowania relacji pomiędzy wierzchołkami.

- `__init__(self, p: Point)`
  Inicjalizuje wierzchołek z danymi w postaci punktu. Lewy i prawy następnik są początkowo ustawione na `None`.
  Parametry:
  `p` (Point): Punkt przechowywany w wierzchołku.
- `getType(self)`
  Zwraca typ wierzchołka, gdzie `0` oznacza wierzchołek punktu (`PointNode`).
  Zwraca:
  `int`: `0` jako identyfikator typu wierzchołka.
- `setLeft(self, node)`
  Ustawia lewego następnika. Jeśli `node` jest wierzchołkiem trapezu (`TrapezoidNode`), dodaje bieżący wierzchołek jako rodzica.
  Parametry:
  `node`: Wierzchołek, który ma być ustawiony jako lewy następnik.
- `setRight(self, node)`
  Ustawia prawego następnika. Jeśli `node` jest wierzchołkiem trapezu (`TrapezoidNode`), dodaje bieżący wierzchołek jako rodzica.
  Parametry:
  `node`: Wierzchołek, który ma być ustawiony jako prawy następnik.

1.2.5 Dokumentacja klasy SegmentNode

Klasa `SegmentNode` reprezentuje wierzchołek struktury przeszukiwań zawierający odcinek jako dane. Służy do przechowywania odcinków i definiowania relacji pomiędzy wierzchołkami.

- `__init__(self, segment: Segment)`
  Inicjalizuje wierzchołek z danymi w postaci odcinka. Lewy i prawy następnik są początkowo ustawione na `None`.
  Parametry:
  `segment` (Segment): Odcinek przechowywany w wierzchołku.
- `getType(self)`
  Zwraca typ wierzchołek, gdzie `1` oznacza wierzchołek odcinka (`SegmentNode`).
  Zwraca:
  `int`: `1` jako identyfikator typu wierzchołka.
- `setLeft(self, node)`
  Ustawia lewego następnika. Jeśli `node` jest wierzchołkiem trapezu (`TrapezoidNode`), dodaje bieżący wierzchołek jako rodzica.
  Parametry:
  `node`: Wierzchołek, który ma być ustawiony jako lewy następnik.
- `setRight(self, node)`
  Ustawia prawego następnika. Jeśli `node` jest wierzchołkiem trapezu (`TrapezoidNode`), dodaje bieżący wierzchołek jako rodzica.
  Parametry:
  `node`: Wierzchołek, który ma być ustawiony jako prawy następnik.

1.2.6 Dokumentacja klasy TrapezoidNode

Klasa `TrapezoidNode` reprezentuje wierzchołek struktury przeszukiwań zawierający trapez jako dane. Dodatkowo umożliwia śledzenie rodziców wierzchołka.

- `__init__(self, trapezoid: Trapezoid)`
  Inicjalizuje wierzchołek z danymi w postaci trapezu. Lista rodziców węzła jest początkowo pusta.
  Parametry:
  `trapezoid` (Trapezoid): Trapez przechowywany w wierzchołku.
- `getType(self)`
  Zwraca typ wierzchołka, gdzie `2` oznacza wierzchołek trapezu (`TrapezoidNode`).
  Zwraca:
  `int`: `2` jako identyfikator typu wierzchołka.
- `replace(self, node)`
  Zastępuje bieżący wierzchołek nowym węzłem. Dla każdego rodzica bieżącego wierzchołka aktualizowane są odnośniki do nowego wierzchołka (`node`).
  Parametry:
  `node`: Wierzchołek, który zastępuje bieżący wierzchołek.

1.2.7 Dokumentacja klasy TrapezoidalMap

Klasa `TrapezoidalMap` oblicza trepzoidalną mapę oraz strukturę przeszukiwań dla danego ogólnego podziału płasczyzny. Umożliwia ona szybkie znajdywanie trapezu w którym znajduje się punkt w $\boldsymbol{O(logn)}$, przy liniowej złożoności pamięciowej.

- `__init__(self, segments: list[Segment], visualisation = False, random = True)`
  Tworzy trapezoidalną mapę, wstawiając segmenty w losowej kolejności. Ustawia początkowy trapez obejmujący całą przestrzeń, a następnie wstawia kolejne segmenty.
  Parametry:
  `segments` (`list[Segment]`): Lista odcinków wejściowych.
  `visualistazion` (`bool`): Ustawia tryb wizualizacji, zapusje każdy podział.
  `random` (`bool`): Ustawia czy dany zbiór odcinków ma być spermutowany.

- `createBoundary(self)`
  Tworzy początkowy trapez obejmujący całą przestrzeń, bazując na granicach odcinków.
  Zwraca:
  `Trapezoid`: Trapez obejmujący całą przestrzeń.
- `query(self, p: Point)`
  Wyszukuje trapez zawierający dany punkt `p`, przechodząc przez strukturę przeszukiwań.
  Parametry:
  `p` (`Point`): Punkt, dla którego szukany jest trapez.
  Zwraca:
  `TrapezoidNode`: Wierzchołek trapezu zawierający punkt `p`.
- `intersectingTrapezoids(self, segment: Segment)`
  Znajduje wszystkie trapezy przecięte przez dany odcinek `segment`.
  Parametry:
  `segment` (`Segment`): Odcinek, dla którego znajdowane są przecięte trapezy.
  Zwraca:
  `list[TrapezoidNode]`: Lista wierzchołków trapezów przeciętych przez odcinek.

- `changeOne(self, node: TrapezoidNode, segment: Segment)`
  Modyfikuje trapezoidalną mapę, gdy odcinek przecina dokładnie jeden trapez.
  Parametry:
  `node` (`TrapezoidNode`): Węzeł trapezu do zmodyfikowania.
  `segment` (`Segment`): Wstawiany odcinek.
- `changeMoreThanOne(self, nodes: list[TrapezoidNode], segment: Segment)`
  Modyfikuje trapezoidalną mapę, gdy odcinek przecina więcej niż jeden trapez.
  Parametry:
  `nodes` (`list[TrapezoidNode]`): Lista węzłów trapezów do zmodyfikowania.
  `segment` (`Segment`): Wstawiany odcinek.
- `getTrapezoids(self)`
  Pobiera listę trapezów reprezentowanych w aktualnej mapie.
  Zwraca:
  `list[Trapezoid]`: Lista trapezów.

1.3 Funkcje do wizualizacji

1.3.1 Dokumentacja funkcji `draw`

Funkcja `draw` umożliwia interaktywną definicję zestawu odcinków na płaszczyźnie oraz określenie punktu zapytania przez użytkownika. Użytkownik rysuje odcinki, klikając i przeciągając myszką, a na koniec dodaje pojedynczy punkt zapytania.

1.3.2 Dokumentacja funkcji `showMap`

Funkcja `showMap` generuje wizualizację podziału trapezowego, pokazując odcinki, trapezy, a także wskazuje trapez, w którym znajduje się punkt zapytania (jeśli podany). Odcinki są rysowane na niebiesko, trapezy na zielono, a punkt zapytania na czerwono.

`showMap(map, Trapezoids: list[Trapezoid], lines: list[Segment], q: Point = None)`
Parametry:
`map` (TrapezoidalMap): Mapa trapezowa.
`Trapezoids` (list[Trapezoid]): Lista trapezów do wizualizacji.

`lines` (list[Segment]): Lista odcinków tworzących podział trapezowy.
`q` (Point, opcjonalnie): Punkt zapytania do wyróżnienia na wykresie.

1.3.3 Dokumentacja funkcji `mapBuildingSteps`

Funkcja `mapBuildingSteps` wizualizuje kolejne kroki budowy mapy trapezowej, umożliwiając zrozumienie działania algorytmu przyrostowego.

`mapBuildingSteps(segments: list[Segment], random=True)`
Tworzy kolejne wizualizacje dodawania odcinków do mapy trapezowej. Na każdym etapie pokazuje dotychczasowy podział trapezowy.
Parametry:
`segments` (list[Segment]): Lista odcinków tworzących podział trapezowy.
`random` (bool, opcjonalnie): Określa, czy odcinki są dodawane w losowej kolejności.

1.3.4 Dokumentacja funkcji `makeGif`

Funkcja `makeGif` generuje animację w formacie GIF, pokazującą kolejne kroki budowy mapy trapezowej, wraz z wyróżnieniem trapezu zawierającego punkt zapytania.

`makeGif(segments: list[Segment], name: str, q: Point = None, random=True)`
Tworzy animowany GIF przedstawiający kolejne kroki budowy mapy trapezowej. Ostatecznie wyróżnia trapez zawierający punkt zapytania (jeżeli zadany).
Parametry:
`segments` (list[Segment]): Lista odcinków tworzących podział trapezowy.
`name` (str): Nazwa pliku wyjściowego GIF.
`q` (Point, opcjonalne): Punkt zapytania.
`random` (bool, opcjonalnie): Określa, czy odcinki są dodawane w losowej kolejności.

1.4 Funkcje do testowania

1.4.1 Dokumentacja funkcji `randomPoint`

Funkcja `randomPoint` generuje losowe punkty wewnątrz trapezu zadanego przez mapę trapezową.

`randomPoint(mapa: TrapezoidalMap, n: int = 1)`
Losuje `n` punktów w granicach zadanego trapezu określonego przez mapę trapezową.
Parametry:
`mapa` (TrapezoidalMap): Mapa trapezowa, która wyznacza granice do losowania.
`n` (int, opcjonalnie): Liczba losowych punktów do wygenerowania (domyślnie `1`).

1.4.2 Dokumentacja funkcji `generateSegment`

Funkcja `generateSegment` generuje zestaw odcinków poziomych w losowych lokalizacjach w określonym zakresie współrzędnych.

`generateSegment(n: int, a: int = 0, b: int = 1000)`
Tworzy `n` odcinków z losowymi współrzędnymi w zadanym przedziale. Zapewnia unikalność współrzędnych.
Parametry:
`n` (int): Liczba odcinków do wygenerowania.
`a` (int, opcjonalnie): Dolna granica przedziału współrzędnych (domyślnie `0`).
`b` (int, opcjonalnie): Górna granica przedziału współrzędnych (domyślnie `1000`).

1.4.3 Dokumentacja funkcji `testTime`

Funkcja `testTime` mierzy czas potrzebny na konstrukcję mapy trapezowej oraz wyszukiwanie w niej punktów dla różnych rozmiarów danych wejściowych. Generuje dwa wykresy: jeden dla czasu konstrukcji mapy trapezowej, a drugi dla czasu wyszukiwania.

`testTime(size: list[int])`
Parametry:
`size` (list[int]): Lista rozmiarów danych wejściowych (liczba odcinków).

2 Część użytkownika

2.1 Układ plików

Wszystkie programy oraz algorytmy znajdują się w folderze Code, poza tym folderem są dwa inne
Gify -> pliki z gifami
Trapezy -> pliki z stworzonymi mapami trapezowymi
Implementacja Algorytmu i struktury są stworzone w folderze **data_structures**
Oprawa graficzna:

- **drawApplication.py** -> otwieranie aplikacji do zadawania odcinków
- **visualisation.py** -> funkcje do wyświetlania mapy trapezowej oraz tworzenia gifów.

2.2 Aplikacja graficzna

W **drawApplication.py** znajduje się funkcja draw. Uruchamia ona okienko, w którym za pomocą myszki dodawane są odcinki. Należy pamiętać, aby odcinki spełniały położenie ogólne. Lewym przyciskiem myszy klikamy w miejsce, gdzie chcemy, by znajdywał się nasz odcinek, a następnie przeciągamy do miejsca, gdzie chcemy, by się kończył. Po tak zadanych odcinkach klikamy przycisk "koniec odcinków"i możemy, ale nie musimy, używając znów prawy przycisk myszy zadać odcinek szukany.

2.3 Odpalanie programu

Część do wywoływania algorytmów znajduje się w pliku **Kopiec_Zwonek_kod.ipynb**.
Wszystkie moduły mają komenatrze, które ułatwiają użytkownikowi posługiwać się tym plikiem. Możliwe czynności do wykonania:

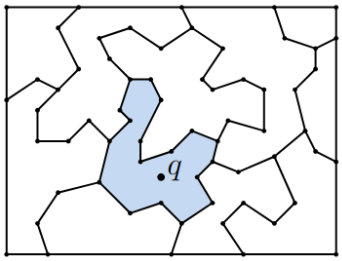
- **drawAndSave** oraz **fromFile** umożliwia tworzenie mapy trapezoidalnej i wyświetlanie jej wraz z szukanym punktem. Pierwsze przy użyciu aplikacji do rysowania odcinków, a drugie używa segmentów z pliku example. Pierwsza opcja podpisana jest "Tworzenie mapy trapezowej w aplikacji", a druga "Tworzenie mapy trapezowej z pliku". Utworzone trapezy i znaleziony trapez zapisują się w pliku tekstowym.
- **Znajdywanie trapezu** używając już wyznaczonej mapy trapezowej można po podaniu punktu trzeba wpisać : Point(a,b) -> a i b to współrzędne punktu, którego chcemy znaleźć.
- **Tworzenie gifów** używając funckji **makeGif** tworzymy gifa z uzyskanych już segmentów i jest on zapisywany do folderu Gify.
- **wizualizacja krokowa** używając funkcji **mapBuldingSteps** tworzymy wizualizację krokową.

3 Sprawozdanie

3.1 Opis problemu

W problemie lokalizacji punktu na płaszczyźnie mamy dany obszar z podziałem poligonowym, a celem jest przetworzenie tego podziału na strukturę danych, która umożliwia efektywne określenie, w której części (obszarze) podziału znajduje się zadany punkt q.

Na przykład, podział może reprezentować jednostki administracyjne, takie jak kraje, województwa czy powiaty, a naszym zadaniem jest zidentyfikowanie kraju, województwa lub powiatu, w którym znajduje się punkt na podstawie jego współrzędnych GPS.



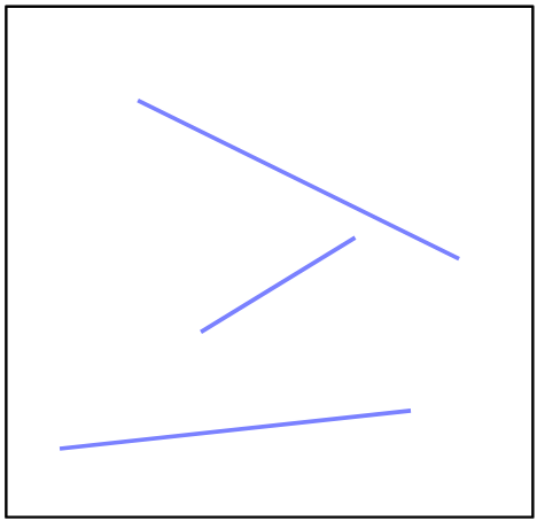
Rys 3.1.1. Lokalizacja punktu

3.2 Metoda trapezowa

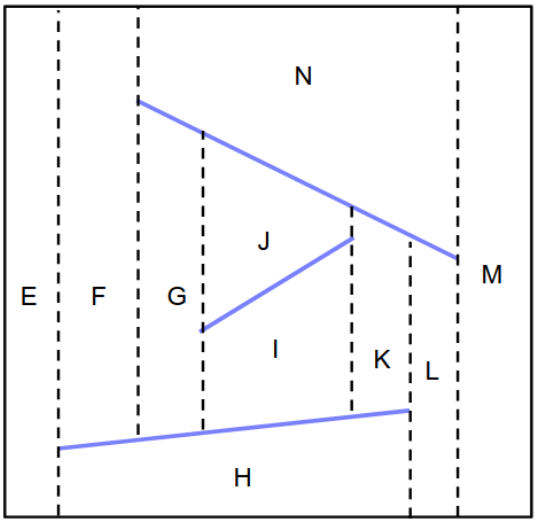
Metoda trapezowa jest jedną z technik lokalizacji punktu na płaszczyźnie. W tej metodzie podział przestrzeni określa się za pomocą zbioru odcinków. Załóżmy, że zbiór ten znajduje się w tzw. położeniu ogólnym, co oznacza, że:

- żaden z odcinków nie jest pionowy,
- współrzędne x-owe końców odcinków są unikalne (z wyjątkiem punktów będących wspólnymi końcami "połączonych"odcinków).

Mapa trapezowa dla takiego zbioru odcinków to podział płaszczyzny na wypukłe wielokąty – trapezy lub trójkąty. Wszystkie te elementy, niezależnie od ich kształtu, określane są jako trapezy (trójkąt traktuje się jako szczególny przypadek trapezu, w którym jeden z boków redukuje się do punktu). Konstrukcja mapy trapezowej polega na prowadzeniu pionowych linii (rozszerzeń) z każdego końca odcinka. Linie te kończą się w momencie napotkania innego odcinka lub ramki otaczającej cały zbiór odcinków, która wyznacza granice obszaru podziału.



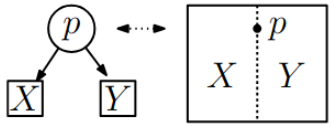
Rys 3.2.1. Przykładowy zbiór odcinków.



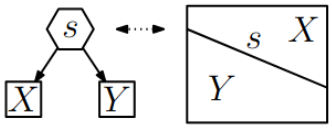
Rys 3.2.2. Mapa trapezowa dla **Rys 3.2.1**.

Struktura wyszukiwania dla mapy trapezowej jest reprezentowana przez skierowany graf acykliczny, w którym wierzchołki pełnią różne funkcje w zależności od swojego typu:

- **Wierzchołek typu x**: Przechowuje informację o punkcie. W tym wierzchołku wykonywane jest porównanie współrzędnej $\boldsymbol{x}$ poszukiwanego punktu z wartością w wierzchołku. Jeśli współrzędna $\boldsymbol{x}$ punktu jest mniejsza, należy przejść do lewego dziecka, w przeciwnym razie – do prawego (Rys 3.2.3).
- **Wierzchołek typu y**: Przechowuje informację o odcinku. W tym wierzchołku sprawdza się, czy poszukiwany punkt znajduje się powyżej odcinka. Jeśli tak, należy podążać do lewego dziecka, w przeciwnym razie – do prawego (Rys 3,2,4).
- **Liść**: Zawiera informację o trapezie, który obejmuje poszukiwany punkt.



Rys 3.2.3. Wierzchołek typu x



Rys 3.2.4. Wierzchołek typu y

Fig 3.2.1 pokazuje na prostym przykładzie zbioru dwóch odcinków, jak mogłaby wyglądać dla tego podziału struktura przeszukiwań. Żeby odpowiedzieć na zapytanie o punkt w trapezie D, przechodzimy od korzenia, punkt jest po prawej niego, czyli przechodzimy do prawego syna, następnie punkt jest po lewej stronie $\boldsymbol{q_1}$, więc wybieramy lewego syna, który reprezentuje segment $\boldsymbol{s_1}$, punkt jest poniżej niego, więc schodzimy do prawego syna. Wyszukiwanie kończy się gdy dojdziemy do liścia, który reprezentuje trapez, w którym znajduje się punkt.

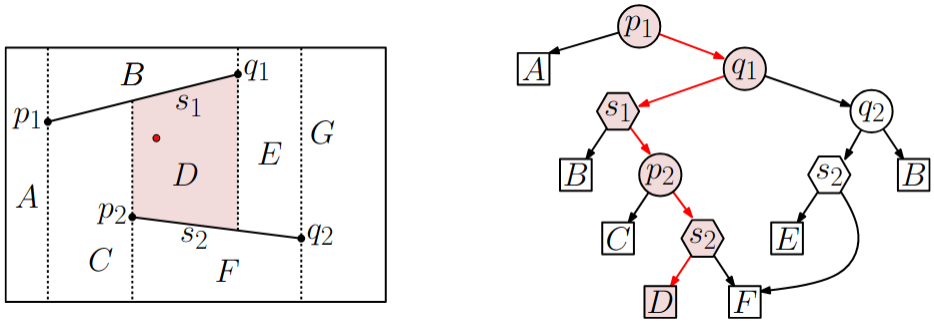


Fig 3.2.1. Możliwa struktura wyszukiwań dla danej mapy

Podstawą budowy mapy trapezowej jest randomizowany algorytm przyrostowy. Odcinki są losowo permutowane i dodawane do mapy w tej kolejności, co zapewnia efektywność algorytmu. Dzięki takiemu podejściu:

- Średnia złożoność konstrukcji mapy trapezowej jest korzystna,
- Oczekiwany rozmiar struktury wyszukiwania pozostaje liniowy w stosunku do liczby odcinków,
- Czas wyszukiwania trapezu zawierającego dany punkt jest proporcjonalny do logarytmu liczby odcinków w zbiorze.

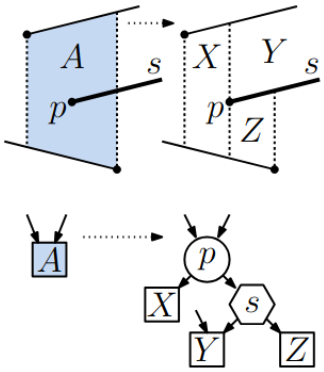
Kluczowym zagadnieniem jest sposób modyfikacji drzewa przy dodaniu nowego odcinka. Zauważmy, że dodanie nowego odcinka wymaga aktualizacji jedynie tej części drzewa, która dotyczy trapezów przeciętych przez dodawany odcienk. Każdy taki trapez zostaje zastąpiony lokalną strukturą wyszukiwania.

Załóżmy, że dodajemy odcinek $\boldsymbol{s}$. Powoduje to zastąpienie pewnego zbioru istniejących trapezów nowymi trapezami. Liście drzewa odpowiadające trapezom, przez które przechodzi odcinek, są wówczas zamieniane na lokalne struktury wyszukiwania, które lokalizują nowe trapezy. Wyróżniamy trzy przypadki w zależności od liczby końców odcinka znajdujących się w obrębie aktualnego trapezu:

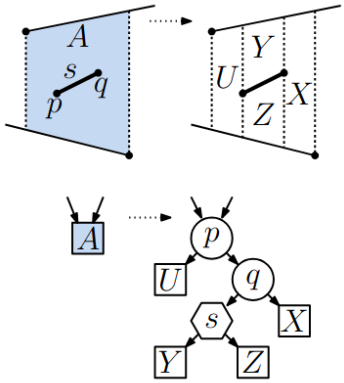
1. **Jeden koniec odcinka (lewy lub prawy) (Rys 3.2.5)**: Jeśli tylko jeden z końców odcinka leży w danym trapezie $\boldsymbol{A}$, to trapez ten zostaje zastąpiony trzema nowymi trapezami: $\boldsymbol{X}$, $\boldsymbol{Y}$ i $\boldsymbol{Z}$. Oznaczmy koniec odcinka jako $\boldsymbol{p}$. Tworzymy węzeł $\boldsymbol{x}$ dla punktu $\boldsymbol{p}$, gdzie jedno z dzieci

odpowiada trapezowi $\boldsymbol{X}$, znajdującemu się poza pionowym rzutem odcinka. Drugie dziecko to węzeł $\boldsymbol{y}$, którego potomkami są trapezy $\boldsymbol{Y}$ i $\boldsymbol{Z}$, leżące odpowiednio nad i pod odcinkiem $\boldsymbol{s}$.

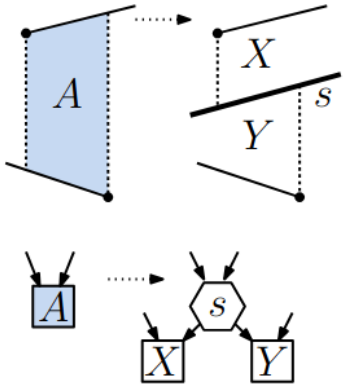
2. **Dwa końce odcinka (Rys 3.2.6)**: Gdy oba końce odcinka ($\boldsymbol{p}$ i $\boldsymbol{q}$) znajdują się w obrębie jednego trapezu, odcinek $\boldsymbol{s}$ w całości mieści się w trapezie $\boldsymbol{A}$. Wówczas trapez $\boldsymbol{A}$ zostaje podzielony na cztery trapezy: $\boldsymbol{U}$, $\boldsymbol{X}$, $\boldsymbol{Y}$ i $\boldsymbol{Z}$. Tworzymy dwa węzły $\boldsymbol{x}$ dla punktów $\boldsymbol{p}$ i $\boldsymbol{q}$ oraz węzeł $\boldsymbol{y}$ dla odcinka $\boldsymbol{s}$, które łączą się w odpowiednią strukturę, opisującą nowy podział.
3. **Brak końców odcinka (Rys 3.2.7)**: Jeśli odcinek $\boldsymbol{s}$ przecina trapez w całości, bez żadnych końców w obrębie trapezu, to trapez $\boldsymbol{A}$ zostaje zastąpiony dwoma nowymi trapezami: $\boldsymbol{Y}$ (leżącym nad odcinkiem) i $\boldsymbol{Z}$ (leżącym pod odcinkiem). Liść drzewa odpowiadający trapezowi $\boldsymbol{A}$ zostaje zamieniony na węzeł $\boldsymbol{y}$, którego dziećmi są liście dla trapezów $\boldsymbol{Y}$ i $\boldsymbol{Z}$.



Rys 3.2.5. Jeden koniec odcinka w trapezie

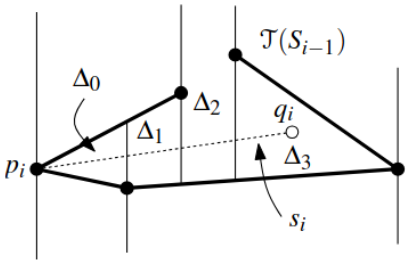


Rys 3.2.6. Oba końce odcinak w trapezie

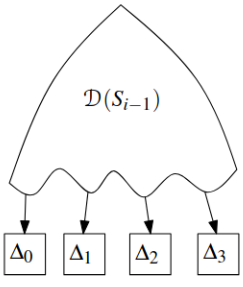


Rys 3.2.7. Brak końców odcinka w trapezie

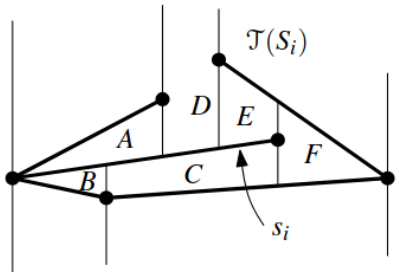
Przykładowo wstawiamy odcinek $\boldsymbol{s_i}$ jak na **Rys 3.2.8**, przechodzi on przez trapezoidy $\boldsymbol{\Delta_0}$, $\boldsymbol{\Delta_1}$, $\boldsymbol{\Delta_2}$, $\boldsymbol{\Delta_3}$, więc je będziemy zmieniać. Trapezoidy $\boldsymbol{\Delta_0}$ i $\boldsymbol{\Delta_3}$ to przypadki gdzie jeden koniec odcinka jest w nich zawarty (**Rys 3.2.5**), z tego powodu podzielą one trapezoid na 3 inne. Trapezoidy $\boldsymbol{\Delta_1}$ i $\boldsymbol{\Delta_2}$ nie zawierają w sobie żadnego końca odcinka $\boldsymbol{s_i}$, zatem oba zostaną podzielone na 2 inne (**Rys 3.2.7**). Końcowy wynik jest pokazany na **Rys 3.2.10**, oraz **Rys 3.2.11**. Warto zauważyć, że każdy trapezoid istnieje w strukturze wyszukiwać tylko raz, a wiele innych wierzchołków może mieć do liścia krawędź (z tego powodu struktura ta nie jest drzewem, a skierowanym acyklicznym grafem), dzięki dzieleniu liści możliwa jest złożoność pamięciowa $\boldsymbol{O(n)}$.



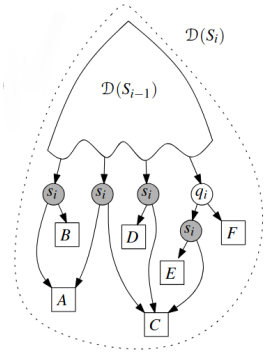
Rys 3.2.8. Wstawienie odcinka $\boldsymbol{s_i}$



Rys 3.2.9. Struktura przeszukiwań przed wstawieniem odcinka



Rys 3.2.10. Odcinek wstawiony s_i



Rys 3.2.11. Struktura przeszukiwań po wstawieniem odcinka

4 Testowanie

4.1 Środowisko oraz dane techniczne

Oprogramowanie było testowane na pojedynczym komputerze. Tabela poniżej przedstawia dane techniczne urządzenia, razem z wersjami użytego oprogramowania.

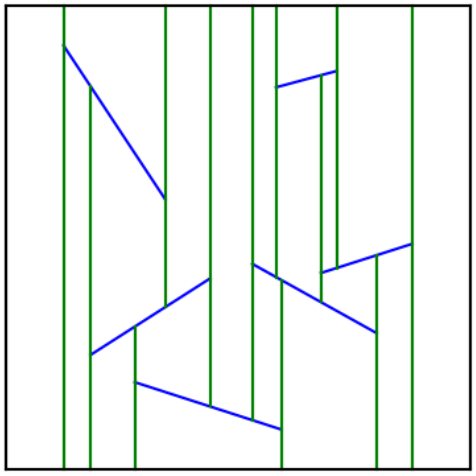
System operacyjny	Procesor	Pamięć RAM	Python	Jupyter	Numpy
Arch linux 6.6.10-arch1-1	AMD Ryzen 5 7535U	16GB	3.12.7	6.29.5	2.2.0

Tabela 4.1.1. Dane techniczne urządzenia i wersje oprogramowania.

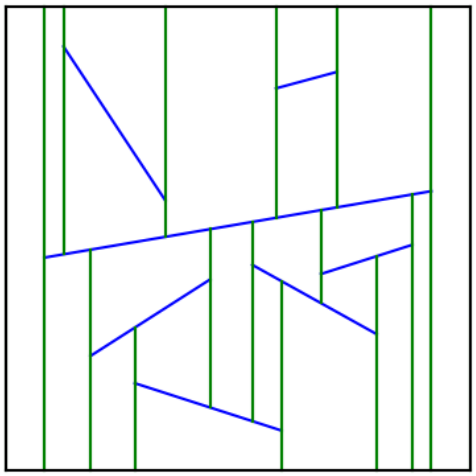
4.2 Testowanie poprawności

Podczas testowania generuję określoną liczbę, nieprzecinających się odcinków. Wygenerowany zbiór spełnia założenie, że żadne dwa krańce odcinków nie mają takich samych współrzędnych x-owych. Następnie przechodzę przez wizualizację krokową, aby zweryfikować, czy nowo dodany odcinek poprawnie dzieli trapez na nowe trapezy. Dla sprawdzenia poprawności działania algorytmu korzystam z aplikacji do rysowania odcinków(drawApplication), testując różne przypadki brzegowe, które mogą potencjalnie doprowadzić do błędów w konstrukcji.

- - odcinki
- - linie pionowe



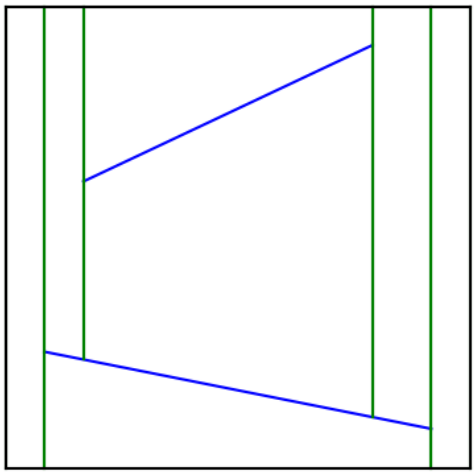
Rys 4.2.1 Mapa trapezoidalna



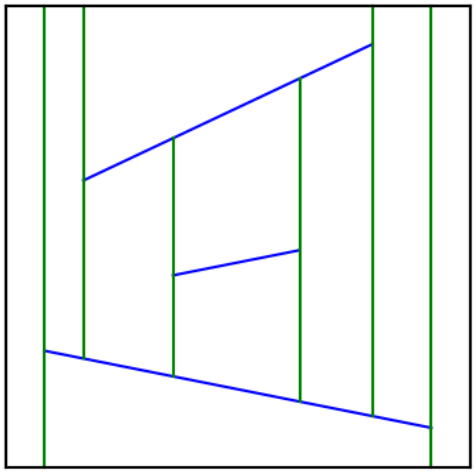
Rys 4.2.2 Mapa trapezoidalna po dodaniu odcinka przecinającego trapezy

Jak widać na **Rys 4.2.2** nowy odcinek, który przecinał wiele trapezów, na różne sposoby został dodany poprawnie. Dodanie tego odcinak przedstawia dwa różne przypadki przepinania węzłów i dodawania nowcyh trapezów:

- - przechodzi przez cały trapez, czyli **Rys 3.2.7**
- - zawiera jeden punkt odcinka w trapezie **Rys 3.2.5**



Rys 4.2.3 Mapa trapezoidalna



Rys 4.2.4 Mapa trapezoidalna po dodaniu odcinka przecinającego trapezy

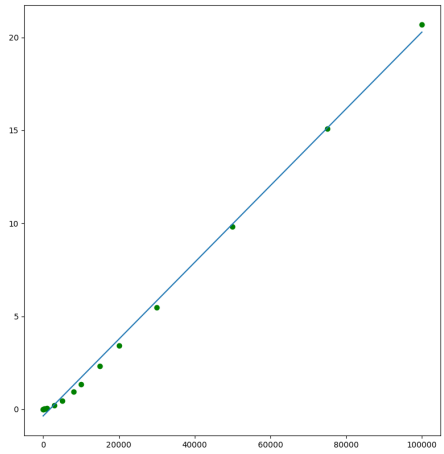
Na **Rys 4.2.3** oraz **Rys 4.2.4** przedstawiony jest przypadek z **Rys 3.2.6**, czyli gdy nowy odcinek zawiera się w pełni w trapezie i za jego miejsce powstają nowe cztery trapezy.

4.3 Testowanie złożoności

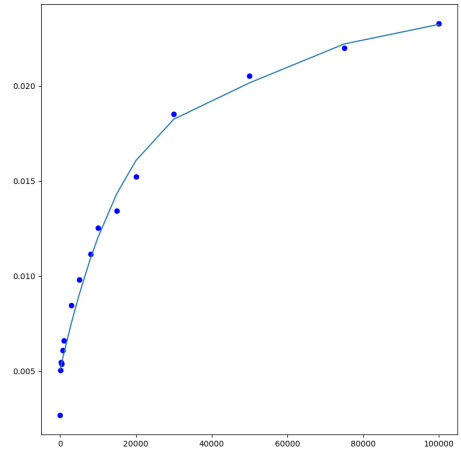
Aby zweryfikować, czy złożoność algorytmu jest zgodna z oczekiwaną, przeprowadzamy serię testów dla różnych liczności losowo wygenerowanych zbiorów odcinków. Ponieważ algorytm ma charakter randomizowany, dla każdego testu obliczamy średni czas z 10 prób.
Podobnie testujemy algorytm do znajdowania położenia punktów. Dla każdej wygenerowanej mapy trapezowej mierzymy czas wykonania 1000 zapytań. Nie ograniczamy się do jednego zapytania, ponieważ czasy te byłyby zbyt zbliżone, z uwagi na oczekiwaną złożoność logarytmiczną. Wykonanie 1000 zapytań dla każdego zbioru odcinków nie zmienia asymptotycznej złożoności, a jedynie wpływa na stałą proporcjonalności. Punkty wykorzystywane w zapytaniach są losowo generowane dla każdej próby i każdego rozmiaru n.

liczba odcinków	średni czas konstrukcji[s]	średni czas 1000 zapytań [s]
10	0.0027	0.0004
50	0.0050	0.0028
200	0.0055	0.0100
400	0.0054	0.0235
700	0.0061	0.0384
1000	0.0066	0.0586
3000	0.0085	0.2137
5000	0.0098	0.4654
8000	0.0112	0.9572
10000	0.0125	1.3535
15000	0.0134	2.3173
20000	0.0152	3.4195
30000	0.0185	5.4659
50000	0.0205	9.8322
75000	0.0220	15.0900
100000	0.0233	20.6885

Tabela 4.3.1 Czas trwania algorytmu konstrukcji mapy jak i wyszukania punktu w zależności od liczby odcinków.



Wykres 4.3.1 Zależność czasu trwania algorytmu konstrukcji od liczby odcinków.



Wykres 4.3.2 Zależność czasu trwania algorytmu wyszukania 1000 punktów od liczby odcinków.

Wykres 4.3.1 wyraźnie pokazuje, że algorytm działa w złożoności liniowej, co wskazuje na złożoność konstrukcji rzędu **O(n)**. Natomiast **wykres 4.3.2** przedstawia zależność o charakterze logarytmicznym, co potwierdza, że złożoność pojedynczego zapytania wynosi **O(logn)**.

5 Bibliografia i inspiracje programistyczne

1. Computational Geometry - Algorithms and Applications
2. CMSC 754: Lecture 9 [Trapezoidal Maps and Planar Point Location]
3. Point Location in Trapezoidal Maps, Claudio Mirolo, Dip. di Scienze Matematiche, Informatiche e Fisiche Università di Udine, via delle Scienze 206 – Udine
4. Trapezoidal Map library — a data structure for fast point location queries, micycl1, github
5. Geometria obliczeniowa Wykład 13 Algorytmy randomizowane
6. Point Location and Trapezoidal Map | Computational Geometry - Lecture 06