Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda trapezowa

Maja Sankowska, Karolina Nitsch

Spis treści

1 Część techniczna **2**
1.1 Opis programu . . . 2
1.2 Podstawowe moduły . . . 2
1.2.1 Struktury danych . . . 2
1.2.2 Funkcje i procedury . . . 5
1.2.3 Biblioteki . . . 7
1.3 Wymagania techniczne . . . 8

2 Część użytkownika **8**
2.1 Sposób uruchamiania programu . . . 8
2.2 Korzystanie z funkcji programu . . . 8
2.2.1 Instrukcja obsługi rysowania odcinków: . . . 8
2.2.2 Instrukcja obsługi generowania losowych odcinków . . . 9
2.2.3 Obsługa funkcji do znajdywania trapezu z danym punktem . . . 9

3 Sprawozdanie **10**
3.1 Opis problemu . . . 10
3.2 Wykonane testy . . . 11
3.3 Uzyskane wyniki . . . 12
3.4 Analiza złożoności czasowej . . . 13

1 Część techniczna

1.1 Opis programu

Program rozwiązuje problem lokalizacji punktu na mapie trapezowej, odpowiadając na pytanie, w którym elemencie znajduje się dany punkt. Konstruuje mapę trapezową, korzystając z randomizowanego algorytmu przyrostowego oraz tworzy strukturę wyszukiwań na podstawie zadanych odcinków. Odcinki mogą być przetwarzane w losowej kolejności. Program umożliwia użytkownikowi zadawanie odcinków i punktów, a także prezentuje w sposób graficzny poszczególne etapy działania algorytmu.

1.2 Podstawowe moduły

1.2.1 Struktury danych

- **Point**
  Klasa implementuje punkt (w przestrzeni dwuwymiarowej) reprezentowany przez współrzędne x, y.
  Przechowywane dane:
  - x - współrzędna na osi x
  - y - współrzędna na osi y

  Funkcje:
  - `def __init__(self,x,y)`
    Konstruktor klasy, jako argumenty przyjmuje współrzędne x i y punktu.

- **Segment**
  Klasa implementuje odcinek, reprezentowany przez dwa punkty końcowe.
  Przechowywane dane:
  - `left` - punkt początkowy odcinka (punkt o mniejszej współrzędnej x).
  - `right` - punkt końcowy odcinka (punkt o większej współrzędnej x).
  - `a` - współczynnik kierunkowy prostej opisującej odcinek ($a = \frac{\Delta y}{\Delta x}$).
  - `b` - wyraz wolny w równaniu prostej ($y = ax + b$).

  Funkcje:
  - `def __init__(self, p, q)`
    Konstruktor klasy, jako argumenty przyjmuje dwa punkty końcowe odcinka (`p`, `q`). Orientuje odcinek tak, aby punkt z mniejszą współrzędną x był zawsze punktem początkowym (`left`), a drugi punktem końcowym (`right`). Oblicza współczynniki a i b równania prostej.
  - `def get_y(self, x)`
    Metoda oblicza współrzędną y na odcinku dla zadanego x, na podstawie równania $y = ax + b$.

  Założenia: Klasa zakłada, że odcinek nie jest pionowy ($\Delta x \neq 0$).

- **Trapezoid**
  Klasa implementuje trapez jako podstawowy element mapy trapezowej. Każdy trapez jest definiowany przez górną i dolną krawędź oraz lewy i prawy wierzchołek.
  Przechowywane dane:
  - `top` - górna krawędź trapezu (obiekt klasy `Segment`).
  - `bottom` - dolna krawędź trapezu (obiekt klasy `Segment`).
  - `leftp` - lewy wierzchołek trapezu (obiekt klasy `Point`).
  - `rightp` - prawy wierzchołek trapezu (obiekt klasy `Point`).
  - `dnode` - wskaźnik do węzła w strukturze wyszukiwań (obiekt klasy `DNode`).
  - `upper_left` - wskaźnik do trapezu sąsiadującego po lewej, górnej stronie.
  - `lower_left` - wskaźnik do trapezu sąsiadującego po lewej, dolnej stronie.
  - `upper_right` - wskaźnik do trapezu sąsiadującego po prawej, górnej stronie.
  - `lower_right` - wskaźnik do trapezu sąsiadującego po prawej, dolnej stronie.

  Funkcje:
  - `def __init__(self, top, bottom, leftp, rightp)`
    Konstruktor klasy, jako argumenty przyjmuje górną i dolną krawędź trapezu (`top`, `bottom`) oraz jego lewy i prawy wierzchołek (`leftp`, `rightp`). Inicjalizuje wskaźniki do węzła w strukturze wyszukiwań (`dnode`) oraz do sąsiednich trapezów jako `None`.

- **DNode**
  Klasa implementuje węzeł w grafie wyszukiwań używanej w mapie trapezowej do lokalizacji punktu. Węzły mogą być typu x, y lub leaf i reprezentują odpowiednio punkty, odcinki lub trapezy.
  Przechowywane dane:
  - `node_type` - typ węzła (`'x'` dla punktu, `'y'` dla odcinka lub `'leaf'` dla trapezu).
  - `label` - wartość przechowywana w węźle (punkt, odcinek lub trapez, zależnie od typu węzła).
  - `left` - wskaźnik do lewego potomka (dotyczy węzłów typu x, gdzie określamy czy szukany punkt znajduje się po lewej czy prawej stronie węzła).
  - `right` - wskaźnik do prawego potomka (jak wyżej).
  - `above` - wskaźnik do potomka powyżej (dotyczy węzłów typu y, gdzie określamy czy szukany punkt znajduje się powyżej czy poniżej danego odcinka).
  - `below` - wskaźnik do potomka poniżej (jak wyżej).
  - `parents` - lista rodziców węzła, potrzebna do aktualizacji struktury wyszukiwań podczas zastępowania trapezów.

  Funkcje:

- `def __init__(self, node_type, label, left=None, right=None, above=None, below=None)`
  Konstruktor klasy, inicjalizuje węzeł na podstawie typu węzła (`'x'`, `'y'` lub `'leaf'`) i jego zawartości (`label`). Może także przyjąć wskaźniki do potomków (`left`, `right`, `above`, `below`).
- `def replace_node(self, new_node)`
  Metoda zastępuje bieżący węzeł nowym węzłem (`new_node`) we wszystkich jego rodzicach. Węzeł `new_node` staje się nowym potomkiem rodziców bieżącego węzła, a jego lista rodziców zostaje odpowiednio zaktualizowana.

Uwagi: Klasa jest używana w grafie wyszukiwań. Chociaż niektóre funkcjonalności są specyficzne dla konkretnych typów węzłów (np. lista rodziców dla węzłów typu leaf, reprezentujących trapezy), uproszczona konstrukcja pozwala na obsługę różnych typów węzłów w jednej klasie.

- **DTree**
  Klasa implementuje strukturę grafu wyszukiwań używaną w mapie trapezowej. Umożliwia szybkie znajdowanie trapezów zawierających dany punkt.
  Przechowywane dane:
  - `root` - korzeń drzewa wyszukiwań (obiekt klasy `DNode`).

  Funkcje:
  - `def __init__(self)`
    Konstruktor klasy, inicjalizuje pustą strukturę drzewa z `root` ustawionym na `None`.
  - `def query(self, point, a=None)`
    Metoda służy do znajdowania trapezu zawierającego dany punkt.
    Argumenty:
    * `point` - obiekt klasy `Point`, reprezentujący współrzędne (x, y).
    * `a` - współczynnik nachylenia prostej, opcjonalny parametr, używany przy wyszukiwaniu lewego punktu wstawianego odcinka podczas budowania mapy trapezowej.

    Działanie:
    * Algorytm zaczyna od korzenia drzewa (`root`) i iteracyjnie sprawdza typ bieżącego węzła:
      · Jeśli `node_type` to `'x'`, punkt jest testowany względem współrzędnej x węzła. W zależności od wyniku wybierany jest lewy (`left`) lub prawy (`right`) potomek.
      · Jeśli `node_type` to `'y'`, punkt jest testowany względem odcinka. Obliczana jest współrzędna y punktu na odcinku, a następnie wybierany jest odpowiedni potomek (`above` lub `below`). Jeśli punkt leży dokładnie na odcinku, parametr `a` jest używany do rozstrzygnięcia.
      · Jeśli `node_type` to `'leaf'`, algorytm zwraca etykietę liścia (trapez zawierający punkt).
    * Jeśli nie znaleziono trapezu, metoda zwraca `None`.

1.2.2 Funkcje i procedury

- `change_segment_representation(lines)`
  Funkcja zmienia reprezentację odcinków z listy krotek postaci $((x_1, y_1), (x_2, y_2))$ na listę obiektów klasy Segment.
  Argumenty:
  - lines - lista krotek postaci $((x_1, y_1), (x_2, y_2))$

  Zwracana wartość: lista obiektów klasy Segment

- `find_area(D,segment)`
  Funkcja znajduje strefę dla odcinka s_i, czyli wszystkie trapezy, które ten odcinek przecina.
  Argumenty:
  - D - graf wyszukiwań - obiekt klasy DTree
  - segment - wstawiany odcinek (s_i)

  Zwracana wartość: lista trapezów, które przecina odcinek w kolejności od lewej do prawej.

- `insert_into_one_trapezoid(dtree, segment, trapezoid)`
  Funkcja wstawia odcinek do jednego trapezu, tworząc nowe trapezy po lewej i prawej stronie odcinka oraz trapezy powyżej i poniżej odcinka. Ustawia odpowiednie powiązania sąsiedztwa między trapezami oraz aktualizuje strukturę wyszukiwania (DTree).

  Argumenty:
  - `dtree`: Graf wyszukiwań (obiekt klasy `DTree`).
  - `segment`: Odcinek, który jest wstawiany.
  - `trapezoid`: Trapez, do którego wstawiany jest odcinek.

  Zwracana wartość: Zbiór nowo utworzonych trapezów.

- `insert_into_many_trapezoids(search_graph, intersecting_trapezoids, segment)`
  Funkcja wstawia odcinek do wielu trapezów. Dla każdego trapezu w liście `intersecting_trapezoi`
  tworzy nowe trapezy w wyniku przecięcia, aktualizuje powiązania między sąsiednimi trapezami, a także struktury w grafie wyszukiwania.

  Argumenty:
  - `search_graph`: Graf wyszukiwań (obiekt klasy `DTree`).
  - `intersecting_trapezoids`: Lista trapezów, które przecina odcinek.
  - `segment`: Odcinek, który jest wstawiany.

  Zwracana wartość: Zbiór nowo utworzonych trapezów.

- `update_left(old_trap, left, top, bottom)`
  Funkcja aktualizuje wskaźniki sąsiedztwa trapezów po lewej stronie strefy, tak aby wskazywały na lewy trapez (jeżeli istnieje) lub górny i dolny trapez (w przeciwnym przypadku).

  Argumenty:

  - `old_trap`: Trapez, który zostanie usunięty, wskazywany wcześniej przez trapezy z lewej strony.
  - `left`: Lewy trapez lub `None` jeśli nie ma takiego trapezu.
  - `top`: Górny trapez.
  - `bottom`: Dolny trapez.

- `update_right(old_trap, right, top, bottom)`
  Funkcja aktualizuje wskaźniki sąsiedztwa trapezów po prawej stronie strefy, tak aby wskazywały na prawy trapez (jeżeli istnieje) lub górny i dolny trapez (w przeciwnym przypadku).

  Argumenty:

  - `old_trap`: Trapez, który zostanie usunięty, wskazywany wcześniej przez trapezy z prawej strony.
  - `right`: Prawy trapez lub `None` jeśli nie ma takiego trapezu.
  - `top`: Górny trapez.
  - `bottom`: Dolny trapez.

- `shuffle_segments(segments)` Funkcja przetasowuje listę odcinków losowo.

  Argumenty:

  - `segments`: Lista odcinków, która ma zostać przetasowana.

  Zwracana wartość: Przetasowana lista odcinków.

- `outer_trapezoid(segments)`
  Funkcja oblicza minimalne i maksymalne wartości współrzędnych, a następnie generuje trapez, którego krawędzie są dostosowane do tych wartości (z dodatkowym marginesem), tak aby objąć wszystkie odcinki.

  Argumenty:

  - `segments`: Lista odcinków reprezentowanych przez pary punktów.

- `generate_segments(n, x_range, y_range)`
  Funkcja generuje losowo nieprzecinające się odcinki w położeniu ogólnym.
  Argumenty:

  - `n`: liczba odcinków do wygenerowania
  - `x_range`: przedział dopuszczalnych współrzędnych x w postaci krotki (x_{min}, x_{max})
  - `y_range`: przedział dopuszczalnych współrzędnych y w postaci krotki (y_{min}, y_{max})

- `build_trapezoidal_map(lines)`
  Funkcja buduję mapę trapezową na podstawie zadanych odcinków w postaci krotek $((x_1, y_1), (x_2, y_2))$.
  Argumenty:
  - lines: lista odcinków w postaci krotek współrzędnych.

  Zwracana wartość: graf wyszukiwań (DTree)

- `find_point(dtree, point)` Funkcja znajduje w grafie wyszukiwań trapez zawierający dany punkt.
  Argumenty:
  - dtree: graf wyszukiwań (obiekt klasy DTree)
  - point: punkt (obiekt klasy Point)

  Zwracana wartość: trapez zawierający dany punkt (obiekt klasy Trapezoid)

- `visualise_map_construction(lines)`
  Funkcja tworzy animację przedstawiającą kroki tworzenia mapy trapezowej dla zadanych odcinków w postaci krotek współrzędnych $((x_1, y_1), (x_2, y_2))$
  Argumenty:
  - lines: odcinki w postaci krotek współrzędnych $((x_1, y_1), (x_2, y_2))$

- `visualize_tree(node, graph=None)`
  Funkcja tworzy wizualizację grafu wyszukiwań przy użyciu biblioteki graphviz.

- `save_dtree_visualization(dtree, filename="dtree_visualization")`
  Funkcja renderuje graf za pomocą graphviz.Digraph i zapisuje go jako plik w domyślnym formacie.

- `visualize_map_point(point)`
  Funkcja wizualizuje mapę trapezową z zadanym punktem oraz zaznaczonym trapezem, w którym ten punkt się znajduje.
  Argumenty: point - punkt do zlokalizowania

1.2.3 Biblioteki

Program korzysta z następujących bibliotek:

- matplotlib - moduł odpowiadający za obsługę interfejsu graficznego - zadawanie odcinków i punktów myszką oraz wizualizacje
- random - moduł pozwalający generować liczby losowe, użyty do tworzenia odcinków i ustalenia kolejności ich dodawania do mapy.
- graphviz - moduł odpowiedzialny za wizualizację grafu przeszukiwań.
- numpy - moduł wykorzystany do analizy efektywności algorytmu, np. do wyznaczania średniego czasu działania.
- time - moduł wykorzystany w analizie efektywności algorytmu do mierzenia czasu działania.

1.3 Wymagania techniczne

Program wymaga następujących zasobów:

- **Wymagania sprzętowe:** Standardowy komputer z procesorem, pamięcią RAM o minimalnej wielkości 4GB oraz wolnym miejscem na dysku do 100MB.
- **Wymagania systemowe:** Program działa na systemach operacyjnych Windows, macOS i Linux.
- **Inne wymagania:** Zainstalowane środowisko Jupyter Notebook (wersja zgodna z Python 3.7 lub nowszą) oraz biblioteki matplotlib, numpy i graphviz.

2 Część użytkownika

2.1 Sposób uruchamiania programu

Poniżej przedstawiono instrukcję krok po kroku, jak uruchomić program:

1. **Otwórz plik z kodem.** Po pobraniu lub rozpakowaniu projektu, otwórz główny plik programu o nazwie `nitsch_sankowska_kod`. Plik zawiera implementację algorytmu oraz narzędzia do wizualizacji.
2. **Uruchomienie kodu.** Kod można uruchomić w dowolnym środowisku programistycznym, takim jak PyCharm lub VSCode. Zaleca się wykonywanie bloków kodu sekwencyjnie, aby móc śledzić jego działanie.
3. **Wybór odcinków.** Program umożliwia:
   - wygenerowanie losowych odcinków do analizy,
   - ręczne wprowadzenie odcinków w celu testowania własnych danych wejściowych.
4. **Wizualizacja budowania mapy (animacja).** Po przetworzeniu danych program wyświetli animację tworzenia mapy trapezowej oraz grafu przeszukiwań, ilustrując działanie algorytmu krok po kroku.
5. **Końcowa wizualizacja.** Program zakończy działanie, prezentując ostateczną mapę trapezową z zaznaczonym punktem oraz odnalezionym trapezem, w którym znajduje się punkt.

2.2 Korzystanie z funkcji programu

2.2.1 Instrukcja obsługi rysowania odcinków:

1. **Uruchomienie programu.** Po uruchomieniu aplikacji do zadawania odcinków otworzy się okno z układem współrzędnych, na którym można rysować odcinki.
2. **Rysowanie odcinka.**
   (a) Kliknij i przetrzymaj lewy przycisk myszy w dowolnym miejscu na wykresie, aby ustawić początek odcinka.

(b) Przesuń kursor w inne miejsce, aby określić koniec odcinka.

(c) Puść lewy przycisk myszy ponownie, aby zakończyć rysowanie odcinka.

3. **Dodawanie kolejnych odcinków.** Proces rysowania można powtarzać dowolną liczbę razy, dodając kolejne odcinki do wykresu.

4. **Zbieranie danych.** Program zapisuje współrzędne początku i końca każdego narysowanego odcinka w postaci listy, którą można uzyskać po zakończeniu rysowania, wywołując funkcję `get_lines()`.

5. **Zamykanie okna.** Po zakończeniu rysowania odcinków można zamknąć okno, klikając przycisk zamykania lub korzystając z kombinacji klawiszy `Alt+F4`.

Uwaga: Program działa w środowisku Python z zainstalowanym pakietem `matplotlib`. Aby uruchomić kod, upewnij się, że środowisko jest poprawnie skonfigurowane.

2.2.2 Instrukcja obsługi generowania losowych odcinków

Funkcja `generate_segments(n, x_range, y_range)` umożliwia wygenerowanie losowych odcinków na płaszczyźnie. Poniżej przedstawiono szczegółową instrukcję obsługi:

1. **Składnia wywołania**

```
segments = generate_segments(n, x_range, y_range)
```

2. **Argumenty funkcji:**

   - `n` – Liczba odcinków do wygenerowania.
   - `x_range` – Zakres wartości współrzędnych na osi `x`, w postaci krotki `(x_min, x_max)`.
   - `y_range` – Zakres wartości współrzędnych na osi `y`, w postaci krotki `(y_min, y_max)`.

2.2.3 Obsługa funkcji do znajdywania trapezu z danym punktem

Aby skorzystać z funkcji służącej do znajdowania trapezu, w którym znajduje się dany punkt, należy wykonać następujące kroki:

1. **Zdefiniowanie punktu** Najpierw należy utworzyć obiekt reprezentujący punkt, korzystając ze struktury `Point`. Przykład definicji punktu:

```
point = Point(1, 2)
```

   Powyższy kod tworzy punkt o współrzędnych (1, 2).

2. **Wywołanie funkcji** Następnie, korzystając z wcześniej zdefiniowanego punktu, można wywołać funkcję:

```
result = find_point(dtree, point)
```

gdzie:

- `dtree` – Drzewo wyszukiwania trapezów.
- `point` – Punkt, dla którego szukamy trapezu.
- `result` – Zmienna przechowująca znaleziony trapez.

3. **Wizualizacja wyszukiwania** W podobny sposób można skorzystać z funkcji wizualizującej proces wyszukiwania trapezu. Przykład użycia wizualizatora:

```
visualize_map_point(dtree, point)
```

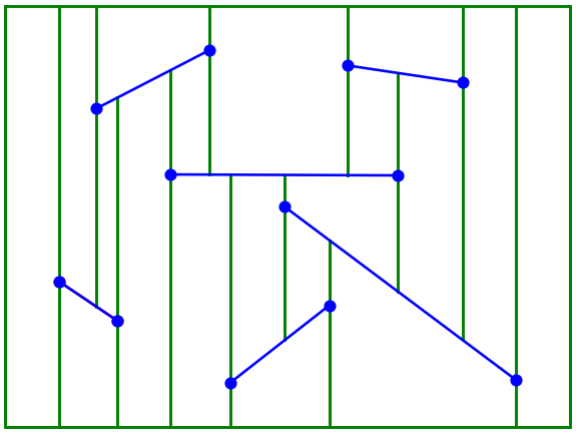
3 Sprawozdanie

3.1 Opis problemu

Problem lokalizacji punktu na płaszczyźnie polega na odszukaniu wielokąta (ściany) zawierającego podany punkt, mając zadany pewien poligonowy, planarny podział płaszczyzny. Jedną z metod rozwiązywania tego problemu jest metoda trapezowa.
Mapa trapezowa T(S) jest podziałem płaszczyzny na wielokąty wypukłe (trapezy lub trójkąty) otrzymanym przez poprowadzenie dwóch rozszerzeń pionowych z każdego końca odcinka. Rozszerzenia kończą się, gdy napotkają inny odcinek lub brzeg prostokąta. Mapa trapezowa reprezentowana jest w postaci zbioru odcinków $S = \{s_1, s_2, \ldots, s_n\}$

- Odcinki nie przecinają się, poza ewentualnie wierzchołkami.
- Dla uproszczenia zakładamy położenie ogólne:
  - żaden odcinek nie jest pionowy
  - wierzchołki żadnych dwóch odcinków nie mają takiej samej współrzędnej x (poza końcami połączonych odcinków).

Na rysunku 1 przedstawiono przykładowy podział płaszczyzny na trapezy. Odcinki z S są zaznaczone na niebiesko, a na zielono - prostokąt zewnętrzny i poprowadzone rozszerzenia pionowe.



Rysunek 1: Przykładowa mapa trapezowa

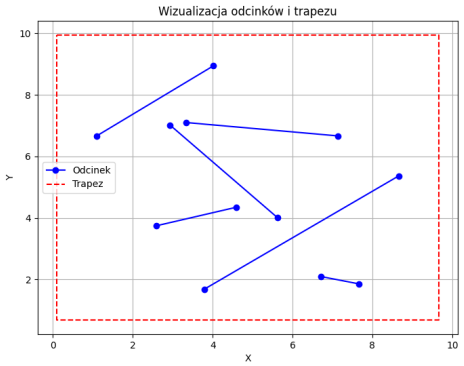
Strukturą przeszukiwań do mapy trapezowej jest acykliczny graf skierowany (DAG). Węzły mogą być różnych typów: x - związany z punktem, y - związany z odcinkiem, liść - związany z trapezem (lub trójkątem). W x-węźle porównywane są współrzędne x szukanego punktu i danego węzła. Jeżeli szukany punkt ma mniejszą współrzędną x, to należy kierować się do lewego dziecka, w przeciwnym razie do prawego dziecka. W węźle typu y porównywane są współrzędne y. Jeżeli szukany punkt leży nad odcinkiem, kierujemy się w lewo, w przeciwnym razie w prawo. Liść przechowuje informację o trapezie, który zawiera szukany punkt.

Do konstrukcji mapy trapezowej używany jest randomizowany algorytm przyrostowy. Odcinki są dodawane w losowej kolejności, co daje następującą oczekiwaną złożoność:

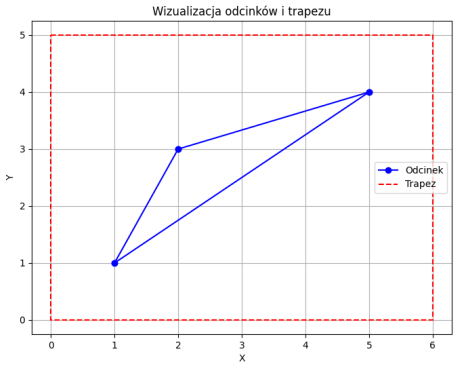
- Rozmiar grafu przeszukiwań - O(n)
- Czas konstrukcji grafu przeszukiwań - O(n log n)
- Czas zapytania o trapez zawierający podany punkt - O(log n)

3.2 Wykonane testy

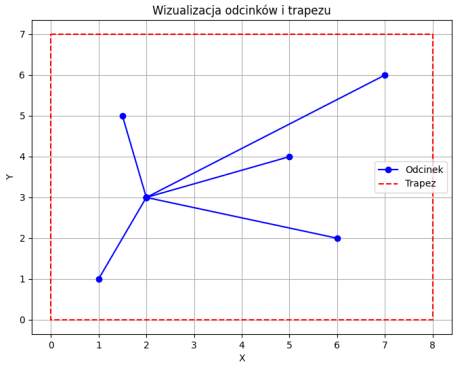
Wykonane zostało wiele testów zarówno, aby zbadać poprawność tworzenia mapy trapezowej jak i lokalizowania trapezu z zadanym punktem. Oprócz tego przeprowadzono analizę złożoności czasowej, generując losowo odcinki i mierząc czas działania algorytmów w zależności od liczby odcinków (szczegóły w sekcji 3.4). Poniżej przedstawiono 4 przykładowe zestawy odcinków, dla których badano poprawność algorytmu tworzenia mapy trapezowej. Na rysunkach zaznaczono dodatkowo trapez zewnętrzny (prostokąt) zawierający wszystkie odcinki.



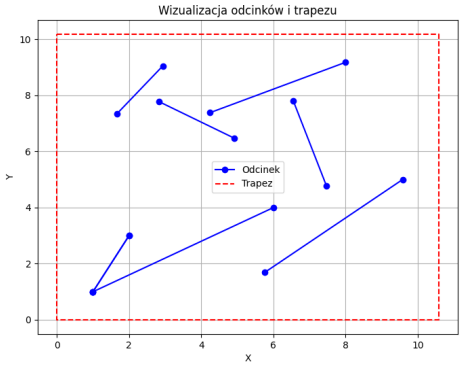
Rysunek 2: Zbiór A - Odcinki bez punktów wspólnych



Rysunek 3: Zbiór B - Odcinki tworzące trójkąt



Rysunek 4: Zbiór C - Wszystkie odcinki zaczynają się w jednym punkcie

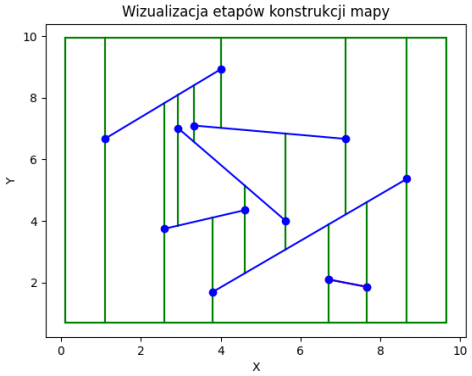


Rysunek 5: Zbiór D - Dwa odcinki mają wspólny koniec, pozostałe bez punktów wspólnych

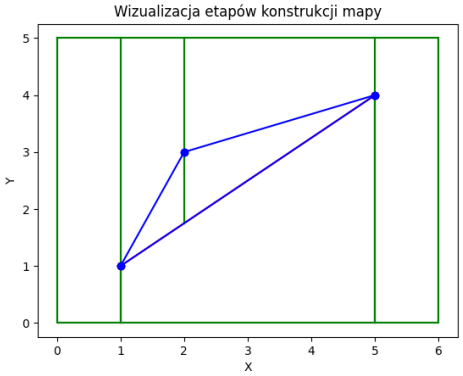
Odcinki były zadawane przy pomocy myszki, z wyjątkiem tych, które mają wspólne końce - zostały one dopisane ręcznie do listy. Testowane zbiory uwzględniają różne przypadki, takie jak odcinki zawierające się w całości w jednym trapezie, odcinki przecinające wiele trapezów i odcinki mające wspólne końce. Testy sprawdziły również poprawność lokalizacji punktu na mapie (podczas dodawania każdego odcinka wyszukiwany jest trapez zawierający jego lewy koniec) oraz tworzenia grafu wyszukiwań (dodatkowo sprawdzone przy pomocy wizualizacji struktury).

3.3 Uzyskane wyniki

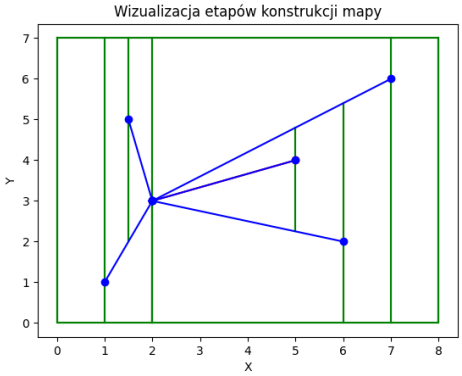
Program dobrze poradził sobie z zadanymi zbiorami odcinków, wyznaczył prawidłowo mapy trapezowe oraz struktury wyszukiwań i poprawnie lokalizował końce odcinków podczas tworzenia map. Poniżej znajdują się efekty końcowe algorytmu tworzenia mapy trapezowej dla zadanych zbiorów, zaprezentowane w sposób graficzny.



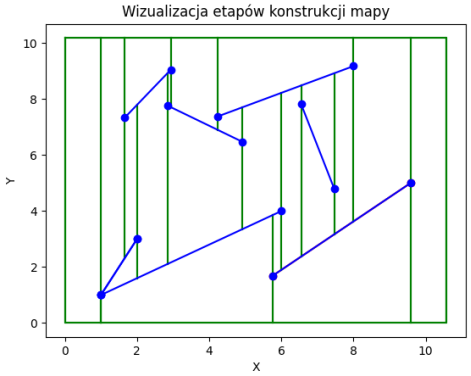
Rysunek 6: Mapa trapezowa - zbiór A



Rysunek 7: Mapa trapezowa - zbiór B



Rysunek 8: Mapa trapezowa - zbiór C



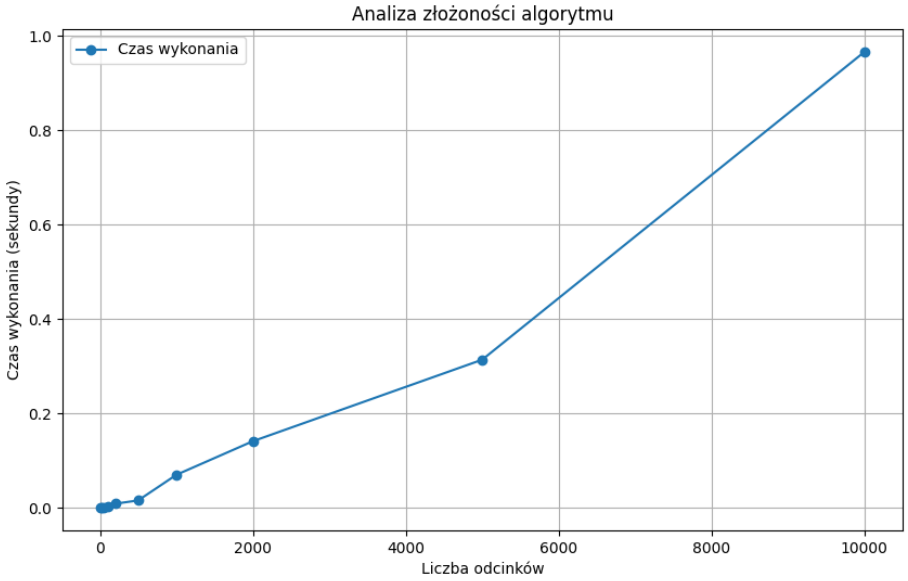
Rysunek 9: Mapa trapezowa - zbiór D

3.4 Analiza złożoności czasowej

Aby zbadać czas działania algorytmów, przeprowadzono serię pomiarów dla różnej ilości odcinków (generowanych losowo). Pierwszym testowanym algorytmem była konstrukcja mapy trapezowej - funkcja `build_trapezoidal_map`. Wygenerowano zbiory odcinków i dla każdego zbioru zmierzono czas działania funkcji 5 razy, przetwarzając odcinki w różnej kolejności. Następnie wyciągnięto średnią, co pozwoliło na bardziej wiarygodne wyniki. Tabela 1 zawiera średnie czasy działania algorytmu w zależności od liczby odcinków (do testów użyto funkcji bez wizualizacji). Dodatkowo utworzono wykres (rysunek 10) na podstawie uzyskanych danych.

Liczba odcinków	Czas wykonania (s)
10	0.000152
20	0.000511
50	0.000883
100	0.002841
200	0.008374
500	0.015358
1000	0.069622
2000	0.140771
5000	0.313462
10000	0.965642

Tabela 1: Czas konstrukcji mapy trapezowej w zależności od liczby odcinków



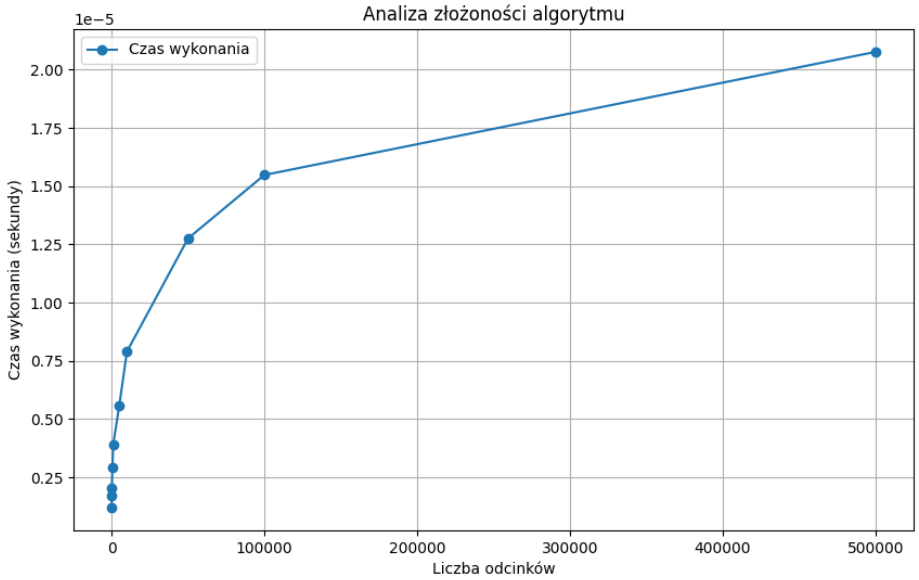
Rysunek 10: Czas konstrukcji mapy trapezowej w zależności od liczby odcinków

Oczekiwana złożoność algorytmu konstrukcji mapy trapezowej wynosi $O(n \log(n))$ Uzyskane wyniki wskazują, że udało się zachować te założenia. Ewentualne odchylenia mogą być spowodowane losową kolejnością przetwarzania odcinków lub procesami działającymi w tle.

Drugim testowanym algorytmem była lokalizacja punktu na mapie trapezowej. Wygenerowano zbiory odcinków o różnej liczności i dla każdego z nich zmierzono czas lokalizacji 10000 losowych punktów. Uzyskane wyniki znajdują się w tabeli 2. Obliczono również średni czas lokalizacji jednego punktu. Na podstawie średnich wyników utworzono wykres (rysunek 11).

Liczba odcinków	Czas wykonania (s)
10	0.010443
50	0.015717
100	0.021677
500	0.031262
1000	0.055152
5000	0.077533
10000	0.124205
50000	0.140771
100000	0.143965
500000	0.20644

Tabela 2: Czas lokalizowania 10000 punktów na mapie trapezowej w zależności od liczby odcinków



Rysunek 11: Czas lokalizacji punktu w zależności od liczby odcinków

Na podstawie otrzymanego wykresu można stwierdzić, że złożoność czasowa funkcji lokalizującej punkt jest zgodna z założeniami teoretycznymi i wynosi $O(\log n)$

Bibliografia

- https://algo.uni-trier.de/lectures/algeo/slides/
- Wykład w ramach przedmiotu Algorytmy Geometryczne
- Mark de Berg - "Computational Geometry - Algorithms and Applications"