AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Lokalizacja punktów w przestrzeni metodą trapezową.

Łukasz Dragon, Rafał Babski

Algorytmy Geometryczne 2024/2025, grupa 1

Spis treści

1 Wstęp

Celem projektu była implementacja algorytmu rozwiązującego problem lokalizacji punktu w przestrzeni dwuwymiarowej metodą trapezową. W sekcji 2 znajduje się opis teoretyczny działania algorytmu, wizualizacja wyników na zbiorach testowych oraz analiza efektywności działania algorytmu. Sekcja 3 zawiera dokumentację załączonego w projekcie programu.

2 Sprawozdanie

2.1 Wstęp teoretyczny

2.1.1 Opis problemu

Problem lokalizacji punktu w przestrzeni dwuwymiarowej można sformułować następująco:

Niech $\mathcal{S}$ oznacza podział przestrzeni dwuwymiarowej zawierający n krawędzi. Zapytanie o lokalizację punktu w $\mathcal{S}$ oznacza znalezienie takiej ściany f zawartej w $\mathcal{S}$, w której znajduje się zadany punkt q. Celem algorytmu jest stworzenie takiej reprezentacji $\mathcal{S}$, która pozwala odpowiadać na zapytanie lokalizacji punktu w jak najmniejszym czasie. Stworzona przez nas reprezentacja powinna także zajmować jak najmniej miejsca w pamięci.

2.1.2 Podział trapezowy

Metoda trapezowa, nazywana także podziałem trapezowym, polega na stworzeniu mapy trapezowej $\mathcal{T}(S)$ dla danego zbioru S zawierającego n odcinków, który reprezentuje podział przestrzeni. Oprócz mapy $\mathcal{T}(S)$, algorytm tworzy strukturę $\mathcal{D}$ pozwalającą odpowiadać na zapytania lokalizacji punktu. Mapa trapezowa $\mathcal{T}(S)$ budowana jest w sposób przyrostowy poprzez dodawanie kolejnych odcinków z S i tworzenie pionowych przedłużeń wychodzących z ich wierzchołków i kończących się po napotkaniu innego odcinka. Kolejność dodawania odcinków z S wpływa na wielkość i czas odpowiedzi na zapytania struktury $\mathcal{D}$, jednak przy użyciu metody randomizacji, jesteśmy w stanie założyć [1, s. 133-136], że złożoność pamięciowa struktury $\mathcal{D}$ jest rzędu $O(n)$, a czas odpowiedzi na zapytanie rzędu $O(logn)$. Ostatecznie, randomizowany algorytm przyrostowy budujący mapę $\mathcal{T}(S)$ i strukturę $\mathcal{D}$ ma oczekiwaną złożoność obliczeniową $O(nlogn)$.

Podział trapezowy przyjmuje następujące założenia dla zbioru S:

1. Odcinki nie przecinają się.
2. Wierzchołki odcinków mają parami różne współrzędne x.

Zbiór S spełniający powyższe założenia jest w *położeniu ogólnym*.

Jesteśmy w stanie pozbyć się założenia 2 poprzez dokonanie *przekształcenia ścinającego* na odcinkach z S o dostatecznie mały kąt φ i stworzenie mapy trapezowej dla nowo powstałego zbioru φS, który już jest w *położeniu ogólnym*. Okazuje się jednak [1, s. 137-139], że przekształcenie to może pozostać jedynie w domyśle i przy sprawdzaniu czy dany punkt leży na lewo od drugiego wystaczy porównywać je leksykograficznie.

2.1.3 Struktura mapy trapezowej

Mapa trapezowa $\mathcal{T}(S)$ reprezentowana jest jako graf trapezów. Każdy stworzony trapez ma dwa boki pionowe (uznajemy także boki o długości zerowej). Uznajemy, że dwa trapezy ze sobą sąsiadują gdy posiadają

wspólną pionową ścianę. Dla zbioru S w *położeniu ogólnym*, każdy trapez ma maksymalnie czterech sąsiadów, do których przechowujemy wskaźniki. Każdy trapez zawiera także wskaźnik do odpowiadającego mu węzła w strukturze przeszukiwań $\mathcal{D}$.

2.1.4 Struktura przeszukiwań

Struktura przeszukiwań $\mathcal{D}$ reprezentowana jest przez *acykliczny graf skierowany* przypominający strukturą drzewo binarne. Struktura ta zawiera trzy typy węzłów:

1. Węzeł trapezu - zawiera wskaźnik do trapezu w mapie $\mathcal{T}(S)$
2. Węzeł wierzchołka - zawiera wskaźnik do jednego z wierzchołków odcinków z S oraz wskaźniki do dwóch węzłów-dzieci (lewego i prawego).
3. Węzeł odcinka - zawiera jeden z odcinków z S oraz wskaźniki do dwóch węzłów-dzieci (lewego i prawego).

Zapytanie w strukturze przechodzi następująco:

Zaczynamy w korzeniu grafu, w każdym węźle wierzchołka testujemy czy zadany punkt leży na lewo czy na prawo od trzymanego wierzchołka; jeżeli leży na prawo, to idziemy do lewego dziecka, a w przeciwnym przypadku idziemy do prawego. Analogicznie postępujemy w węzłach odcinka, tym razem jednak testując czy zadany punkt leży nad odcinkiem (idziemy w lewo), czy pod (idziemy w prawo). Jeżeli zadany punkt jest równy jednemu z wierzchołków, bądź leży na jednym z odcinków, to zwracamy odpowiednią informację.

2.1.5 Opis algorytmu

Algorytm w i-tym kroku buduje mapę $\mathcal{T}(S_i)$ oraz strukturę $\mathcal{D}_i$ dla listy punktów $S_i = (s_1, s_2, ..., s_i)$. Wynikiem jest mapa $\mathcal{T}(S_n)$ oraz struktura $\mathcal{D}_n$, które zawierają wszystkie odcinki z S. Poszczególne kroki algorytmu przebiegają następująco:

1. Tworzymy prostokąt (trapez), w którym zawiera się każdy odcinek w S i za jego pomocą inicjalizujemy strukturę $\mathcal{T}(S_0) := \mathcal{T}(\emptyset)$.
2. Tworzymy listę $S_n = (s_1, s_2, ..., s_n)$ bedącą losową permutacją odcinków ze zbioru S.
3. Dla każdego $i = 1, 2, ..., n$ powtarzamy:
   (a) Znajdujemy w obecnej mapie $\mathcal{T}(S_{i-1})$ trapezy $\Delta_1, \Delta_2, ..., \Delta_k$ przecięte przez odcinek s_i (przecięcie tylko na wierzchołkach nie jest liczone).
   (b) Usuwamy z $\mathcal{T}(S_{i-1})$ trapezy $\Delta_1, \Delta_2, ..., \Delta_k$ zamieniając je na trapezy powstałe po dodaniu s_i. Otrzymujemy w ten sposób mapę $\mathcal{T}(S_i)$
   (c) Usuwamy z $\mathcal{D}_{i-1}$ węzły odpowiadające trapezom $\Delta_1, \Delta_2, ..., \Delta_k$ i dodajemy węzły dla nowych trapezów odpowiednio je łącząc za pomocą węzłów wierzchołków i odcinków.
4. Zwracamy $\mathcal{T}(S_n)$ jako wynik.

Krok 3.(a) jesteśmy w stanie wykonać wykorzystując połączenia sąsiedzkie trapezów w mapie. Najpierw znajdujemy trapez Δ_1, wykonując zapytanie w strukturze $\mathcal{D}_{i-1}$ dla punktu będącego lewym wierzchołkiem

odcinka s_i. Następnie podążamy wzdłuż odcinka s_i przechodząc po i dodając do listy przeciętych trapezów odpowiednich sąsiadów w mapie. Kończymy po napotkaniu prawego wierzchołka odcinka s_i. Dokładna implementacja wszystkich kroków algorytmu znajduje się w załączonym programie opisanym w sekcji 3. Opis każdego poszczególnego kroku algorytmu można znaleźć także w: [1, s. 129-133]

2.2 Specyfikacja środowiska użytego do obliczeń

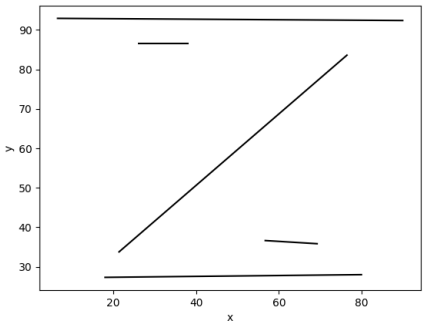
Wyniki pokazane w sprawozdaniu zostały wygenerowane przy użyciu interpretera języka Python wersji 3.9. Losowe zbiory odcinków przygotowane zostały przy pomocy biblioteki `numpy`. Wykresy i wizualizacja stworzonych przez algorytm podziałów trapezowych została przygotowana za pomocą narzędzia przygotowanego przez koło naukowe Bit, dostępnego pod adresem: *https://github.com/aghbit/Algorytmy-Geometryczne* Do przygotowania interaktywnego interfejsu pozwalającego na zadawanie własnych zbiorów odcinków użyta została biblioteka `matplotlib`. Wyniki przedstawione w tabeli 1 zostały wygenerowane na komputerze z systemem operacyjnym Debian 12 i procesorem AMD Ryzen 5 3600 3.6GHz.

2.3 Testy i wizualizacja

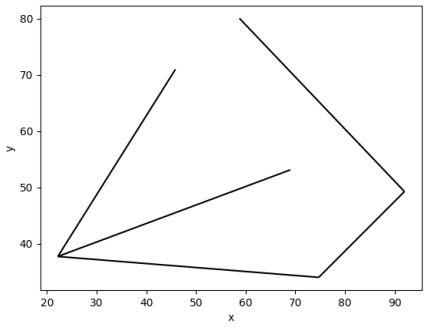
Przygotowaliśmy cztery zbiory odcinków w celu sprawdzenia poprawności i wizualizacji działania algorytmu. Zbiory te przedstawione są na rysunku 1. Rysunki 2, 3, 4, 5 zawierają wizualizację map trapezowych wygenerowanych przez algorytm kolejno dla zbiorów: A, B, C i D. Sekcje od 2.3.1 do 2.3.4 zawierają także krótki komentarz na temat poszczególnych zbiorów.

Animacje krokowe budowy mapy dla wspomnianych zbiorów dostępne są do wygenerowania w pliku *notatnika Python* `point_location.ipynb` (w tym celu należy zapoznać się z dokumentacją w sekcji 3). Ponadto, wspomniane animacje zawarte są z projektem w formie plików z rozszerzeniem `gif`. Pliki te mają nazwy: `test_a_map.gif`, `test_b_map.gif`, `test_c_map.gif`, `test_d_map.gif` i odpowiadają one kolejno zbiorom: A, B, C, D.

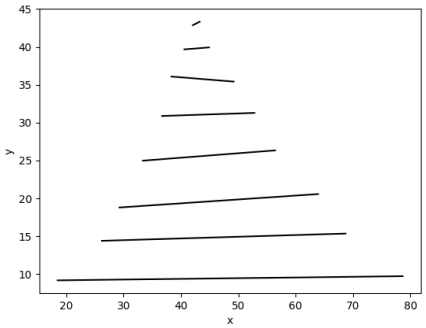
Animacje dla zbiorów zadanych przez użytkownika dostępne są do wygenerowania w pliku: `point_location.ipynb`.



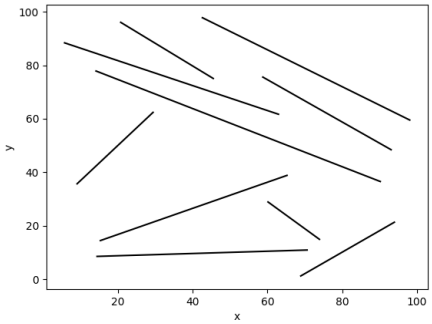
(i) Zbiór A



(ii) Zbiór B



(iii) Zbiór C

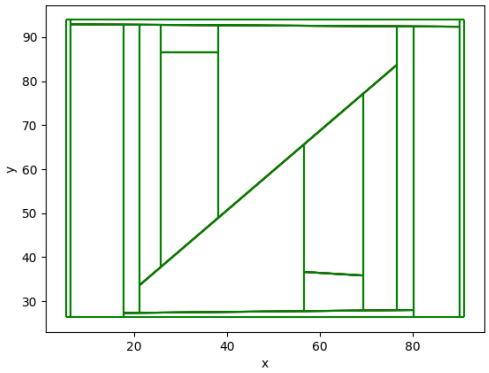


(iv) Zbiór D

Rysunek 1: Wybrane zbiory testowe

2.3.1 Zbiór A

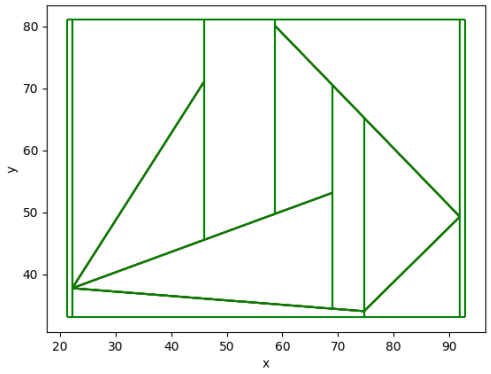
Zbiór ten został przygotwany w celu sprawdzenia czy algorytm poprawnie tworzy i scala trapezy. Rysunek 2 zawiera wizualizację wygenerowanej mapy.



Rysunek 2: Mapa trapezowa wygenerowana dla zbioru A

2.3.2 Zbiór B

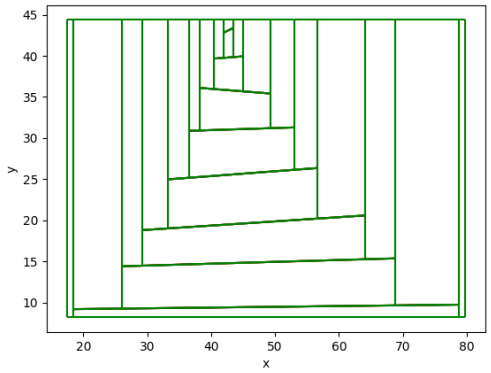
Zbiór ten został przygotowany w celu sprawdzenia czy algorytm poprawnie radzi sobie w przypadku, gdy odcinki łączą się w jednym punkcie. Rysunek 3 zawiera wizualizację wygenerowanej mapy.



Rysunek 3: Mapa trapezowa wygenerowana dla zbioru B

2.3.3 Zbiór C

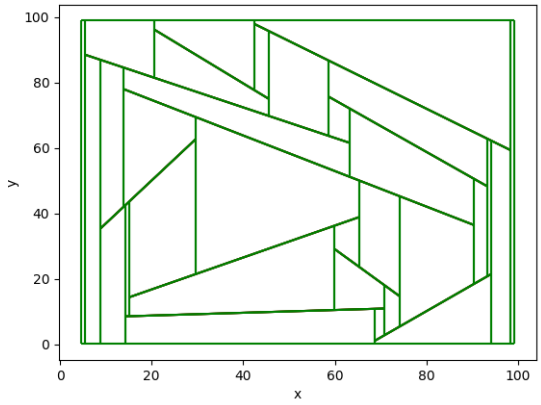
Zbiór ten został przygotowany w celu sprawdzenia czy algorytm poprawnie tworzy trapezy gdy każdy odcinek zawiera się całkowicie wewnątrz jednego trapezu. Rysunek 4 zawiera wizualizację wygenerowanej mapy.



Rysunek 4: Mapa trapezowa wygenerowana dla zbioru C

2.3.4 Zbiór D

Zbiór ten został wygenerowany losowo i sprawdza zarówno przypadki, w których odcinki zawierają się w jednym trapezie, jak i w wielu. Rysunek 5 zawiera wizualizację wygenerowanej mapy.



Rysunek 5: Mapa trapezowa wygenerowana dla zbioru D

2.4 Analiza efektywności algorytmu

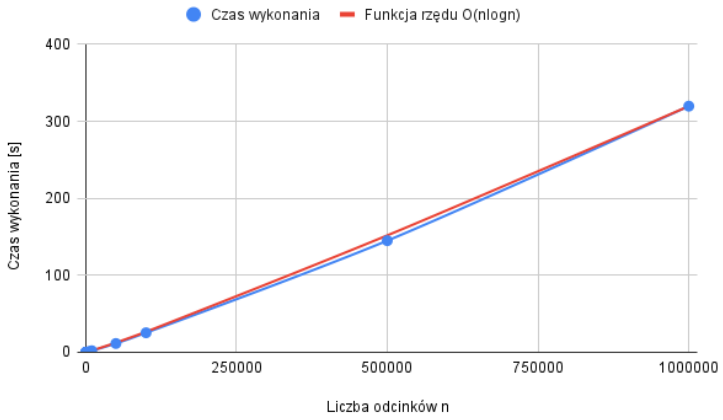
W celu analizy efektywności napisanego przez nas programu przygotowaliśmy funkcję generującą losowy zbiór poziomych odcinków. Na tak przygotowanych zbiorach, dla różnych liczb odcinków n, zmierzyliśmy średni czas budowy mapy trapezowej przy pięciu wywołaniach. Oprócz tego zliczyliśmy liczbę węzłów w wygenerowanych strukturach przeszukiwań w celu oszacowania złożoności pamięciowej programu (tutaj brane pod uwagę było jedno wywołanie). Ostatecznie, zmierzyliśmy średni czas wykonania zapytania lokalizacji punktu w wygenerowanych mapach dla 10000 losowo wygenerowanych punktów.

Tabela 1 zawiera wszystkie pomiary dla poszczególnych wartości n. Rysunki 6, 7, 8 zawierają wykresy poszczególnych zależności z tabeli 1 przyrównane do wykresów funkcji rzędu oczekiwanych złożoności, czyli:

- Rysunek 6 - Zależność czasu wykonania budowy mapy od n. Oczekiwana złożoność $O(nlogn)$
- Rysunek 7 - Zależność wielkości grafu przeszukiwań od n. Oczekiwana złożoność $O(n)$
- Rysunek 8 - Zależność czasu wykonania zapytania lokalizacji punktu od n. Oczekiwana złożoność $O(logn)$

n	Średni czas budowy mapy [s]	Liczba węzłów w mapie	Średni czas zapytania [s]
10	0,0023	81	0,000005872006416
50	0,0090	453	0,000007902321815
100	0,0111	928	0,000008317356110
500	0,0654	4930	0,000010902709960
1000	0,1456	9792	0,000011252136230
5000	0,8821	49847	0,000015353059770
10000	2,0014	99802	0,000016459221840
50000	11,4103	500394	0,000020709886550
100000	25,3333	1001918	0,000023330583570
500000	144,9936	4995205	0,000027396416660
1000000	319,7159	10003877	0,000031653146740

Tabela 1: Zależności czasu budowy mapy, jej wielkości oraz czasu zapytania dla różnych wartości n.



Rysunek 6: Zależność czasu wykonania budowy mapy od n.

Rysunek 7: Zależność wielkości grafu przeszukiwań od n.

Rysunek 8: Zależność czasu wykonania zapytania lokalizacji punktu od n.

Jak możemy zauważyć na rysunkach 6, 7, 8, oczekiwana złożoność czasowa i pamięciowa algorytmu budowy mapy oraz czasu zapytania pokrywa się ze zmierzonymi przez nas wartościami.

3 Dokumentacja

Program napisany w ramach projektu podzielony jest na dwa pliki: `point_location.ipynb` oraz `creator.py`. Pierwszy z nich jest plikiem *notatnika Python* przygotowanym w celach dydaktycznych dla użytkownika. Zawiera implementację, wizualizację na przykładach oraz możliwość testowania algorytmu na przygotowanych interaktywnie zbiorach odcinków. Drugi jest natomiast plikiem zawierającym kod realizujący interfejs graficzny dostępny w notatniku.

Sekcja 3.1 zawiera instrukcje potrzebne do wykonania programu, sekcja 3.2 zawiera dokumentację skierowaną do użytkownika, a sekcja 3.3 opisuje dokładnie najważniejsze funkcje, metody i obiekty stworzone w celu implementacji algorytmu.

3.1 Instrukcja wykonania programu

Notatnik `point_location.ipynb` należy wykonać wewnątrz środowiska dostępnego w narzędziu przygotowanym przez studentów z koła naukowego Bit, które jest dostępne pod adresem:
https://github.com/aghbit/Algorytmy-Geometryczne
Program należy wykonywać w ten sam sposób co zawarte w narzędziu programy przygotowane na laboratoria.

3.2 Część użytkownika

W celu uruchomienia programu należy po kolei wykonywać komórki wewnątrz notatnika `point_location.ipynb`. Plik ten zawiera krótkie opisy problemu, algorytmu oraz zaimplementowanych przez nas struktur. Oprócz tego nad każdą metodą i obiektem znajdują się komentarze tłumaczące ich cel i działanie.

Pod nagłówkiem *"Interaktywne zadawanie zbioru odcinków"* znajduje się odpowiednio komórka, której wykonanie otworzy okno pozwalające na interaktywne zadawanie zbioru odcinków. Kliknięcie lewym przyciskiem myszy w dwóch miejscach stworzy odcinek. Program nie pozwoli na stworzenie odcinka przecinającego

się z innym. Przycisk *"Wyczyść"* usuwa wszystkie stworzone odcinki. Odcinki można zapisać do pliku o formacie `json` po kliknięciu przycisku *"Zapisz"*. Alternatywnie do zadawania odcinków myszką, możliwe jest wygenerowanie zbioru odcinków losowo po wciśnięciu przycisku *"Wygeneruj losowo odcinki"*, wpisaniu interesujących nas parametrów i zatwierdzeniu przyciskiem *"OK"*. Zamknięcie okna możliwe jest przy pomocy przycisku *"OK"*.

Po wygenerowaniu zbioru odcinków, wywołanie kolejnej komórki wypisze na ekran ich współrzędne oraz wyświetli krokową wizualizację budowy mapy trapezowej na zadanym przez użytkownika zbiorze.

Następne komórki zawierają kod generujący animacje krokowe dla wybranych zbiorów testowych oraz dane potrzebne do analizy efektywności algorytmu.

3.3 Część techniczna

Algorytm budujący podział trapezowy dla zadanego zbioru odcinków skupiony jest wewnątrz funkcji `build_trapezoidal_map`. Funkcja ta zwraca obiekt `TrapezoidalMap`, który zawiera w sobie zarówno reprezentację mapy trapezowej, jak i graf przeszukiwań. Dokładne kroki budowy znajdują się wewnątrz metod obiektu `TrapezoidalMap`. Oprócz obiektu mapy istnieją także następujące obiekty pomocnicze:

- `Trapezoid` - reprezentuje pojedynczy trapez, zawiera indeksy odcinków definiujących go z góry i z dołu (`top` i `bottom`) oraz indeksy wierzchołków definiujących go z lewej i z prawej (`leftp`, `rightp`). Metoda `as_verticies` pozwala obliczyć dokładne współrzędne wierzchołków trapezu.

- `TrapezoidNode` - reprezentuje węzeł trapezu w grafie przeszukiwań. Zawiera wskaźniki do odpowiadającego mu obiektu `Trapezoid` oraz do wszystkich z jego maksymalnie czterech sąsiadów (`left_lower`, `left_upper`, `right_lower`, `right_upper`). Posiada także wskaźniki do własnych przodków w grafie.

- `XNode` - reprezentuje węzeł wierzchołka. Zawiera indeks odpowiadającego mu punktu (`x`) oraz wskaźniki do lewego i prawego dziecka (`left` i `right`).

- `YNode` - reprezentuje węzeł odcinka. Zawiera indeks odpowiadającego mu odcinka (`edge`) oraz wskaźniki do lewego i prawego dziecka (`left` i `right`).

Wszystkie obiekty typu `Node` posiadają metodę `follow`, która przyjmuje punkt i zwraca wskaźnik do dziecka, do którego powinniśmy dalej przejść. `XNode` prowadzi do lewego dziecka gdy dany punkt leży po lewej od trzymanego wierzchołka i do prawego gdy leży po prawej. `YNode` prowadzi do lewego dziecka gdy punkt leży ponad trzymanym odcinkiem i do prawego gdy leży pod. Jeżeli punkt znajduje się wewnątrz węzła (jest wewnątrz trapezu, jest już wierzchołkiem, bądź leży na odcinku) zwracana jest wartość `None`.

Obiekt `TrapezoidalMap` zawiera w sobie wskaźnik do węzła korzenia (`root`), indeks ostatnio dodanego wierzchołka oraz wskaźnik do listy odcinków (`sections`). Inicjalizując obiekt tworzymy mapę zawierającą tylko jeden trapez - prostokąt, otaczający wszystkie odcinki.

Opiszemy teraz najważniejsze metody obiektu `TrapezoidalMap`:

- `advance_i` - dodaje kolejny odcinek z podanej listy do mapy, aktualizując wewnętrzną strukturę tak, aby reprezentowała graf przeszukiwań i mapę trapezową uwzględniającą nowo dodany odcinek.

- `find_section_zone` - zwraca listę trapezów przeciętych przez podany odcinek.

- `partition_zone` - dzieli wszystkie trapezy przecinane przez dany odcinek na mniejsze, aktualizując sąsiedztwa i strukturę grafu przeszukiwań. Dzieli się na cztery przypadki rozważane w następujących metodach:

  1. `partition_one` - odcinek przecina dokładnie jeden trapez i znajduje się całkowicie wewnątrz niego. Zawsze tworzy cztery trapezy.
  2. `partition_one_tri_left` i `partition_one_tri_right` - przypadki symetryczne: odcinek przecina dokładnie jeden trapez, lecz jeden z jego wierzchołków (lewy lub prawy) znajduje się już w mapie. Zawsze tworzy trzy trapezy.
  3. `partition_multi` - odcinek przecina więcej niż jeden trapez. Algorytm idzie wzdłuż odcinka, usuwając stare trapezy i tworząc nowe, łącząc niektóre z nich w jeden.

- `find` - zwraca węzeł w grafie przeszukiwań zawierający dany punkt.

- `find_trapezoid` - działa analogicznie do `find`, lecz zawsze zwraca trapez. Gdy punkt jest już wierzchołkiem, przyjmujemy, że leży on po prawo, a gdy leży na odcinku, to uznajemy, że leży nad nim.

- `get_trapezoids` - zwraca listę trapezów wewnątrz mapy.

  W celu wizualizacji kroków algorytmu stworzona została także metoda:
  `build_trapezoidal_map_visualization` korzystająca z narzędzia do wizualizacji przygotowanego przez koło naukowe Bit.

4 Podsumowanie

W ramach projektu zaimplementowaliśmy algorytm lokalizacji punktu w przestrzeni metodą trapezową. Zaimplementowana przez nas funkcja budująca graf przeszukiwań i mapę trapezową przeszła wszystkie testy, pokrywając się po wizualizacji z naszymi oczekiwaniami. Przygotowaliśmy także animacje krokowe pokazujące działanie algorytmu, które poza wsparciem nas w wykrywaniu błędów na etapie pisania kodu, pokazują jasno kolejne kroki algorytmu. Animacje dla zbiorów testowych dołączone są razem z projektem w postaci plików: `test_a_map.gif`, `test_b_map.gif`, `test_c_map.gif` oraz `test_d_map.gif`. Zaimplementowany został także interfejs pozwalający użytkownikowi na zadawanie własnych zbiorów odcinków i pokazujący zbudowaną dla nich mapę trapezową. Ostatecznie zmierzyliśmy czas wykonania budowy mapy, jej wielkość oraz czas wykonania zapytania lokalizacji punktu dla różnych liczb odcinków w zbiorze wejściowym. Wykresy zmierzonych wartości pokrywają się z wykresami oczekiwanych złożoności pamięciowych i obliczeniowych algorytmu.

Literatura

[1] M. van de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, (1997) *Computational Geometry*, Springer.