



Dokumentacja

Kinga Bunkowska

Michalina Hytrek

Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda separatorów



Spis treści

1. Temat Projektu	2
2. Dane techniczne	2
3. Użyte struktury danych	2
Node (Punkt/Węzeł/Wierzchołek)	2
Edge (Krawędź).....	3
Graph (Graf)	3
TreeNode – węzeł drzewa binarnego przechowujący separatory	4
4. Wprowadzania grafu	4
5. Algorytm	5
6. Implementacja algorytmu	6
7. Testy efektywności.....	8
8. Wizualizacja algorytmu.....	13
9. Implementacja wizualizacji	13
10. Wnioski	14
11. Bibliografia.....	14



1. Temat Projektu

Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda separatorów.

Projekt zawiera implementacje algorytmu lokalizacji punktu metoda separatorów w danym obszarze poligonowym, program do zadawania obszarów poligonowych, program do wizualizacji algorytmu i jego wyników.

2. Dane techniczne

Komputer 1:

Procesor: 2,3 GHz Dwurdzeniowy procesor Intel Core i5

RAM: 8 GB

Użyte środowisko: Visual Studio Code

Oprogramowanie: macOS Ventura

Komputer 2:

Procesor: Intel Pentium G4560 3.50Hz

Karta graficzna: Nvidia GTX 1660Ti

RAM: 16GB

Użyte środowisko: Visual Studio Code

Oprogramowanie: windows 10 x64

Wykorzystano: interpreter Python 3.11.5

3. Użyte struktury danych

Node (Węzeł/Wierzchołek)

Zmienne:

- Wage_out – suma wag krawędzi wychodzących z danego węzła;
- Wage_in – suma wag krawędzi wchodzących do danego węzła;
- Coordinates – krotka zawierająca współrzędną x i y danego węzła;
- Edges_in – lista krawędzi wchodzących do danego węzła;
- Edges_out – lista krawędzi wychodzących z danego węzła;



- Id – identyfikator wężła (dla ułatwienia testowania).

Metody:

- `__init__`
- `__eq__`
- `__hash__`
- `__repr__`
- `__str__`

Edge (Krawędź)

Zmienne:

- Node1 – pierwszy wierzchołek krawędzi;
- Node2 – drugi wierzchołek krawędzi;
- Wage – waga danej krawędzi;
- A – współczynnik kierunkowy prostej przechodzącej przez dana krawędź;
- B – wyraz wolny prostej przechodzącej przez dana krawędź.

Metody:

- `Is_x_in_line()`
- `Are_intersecting()`
- `__eq__`
- `__hash__`
- `__repr__`
- `__str__`

Graph (Graf)

Zmienne:

- Nodes – lista wierzchołków danego grafu;
- Edges – lista krawędzi danego grafu.

Metody:

- `Add_edge();`
- `Draw();`
- `Draw_with_arrows();`
- `__repr__;`



TreeNode – węzeł drzewa binarnego przechowujący separator

Zmienne:

- Id – identyfikator separatora;
- Parent – separator znajdujący się w węźle powyżej aktualnego;
- Right – separator znajdujący się w węźle poniżej i w prawo aktualnego;
- Left – separator znajdujący się w węźle poniżej i w lewo aktualnego;
- Edges – krawędzie separatora, które nie znajdowały się w wcześniejszych węzłach.

4. Wprowadzania grafu

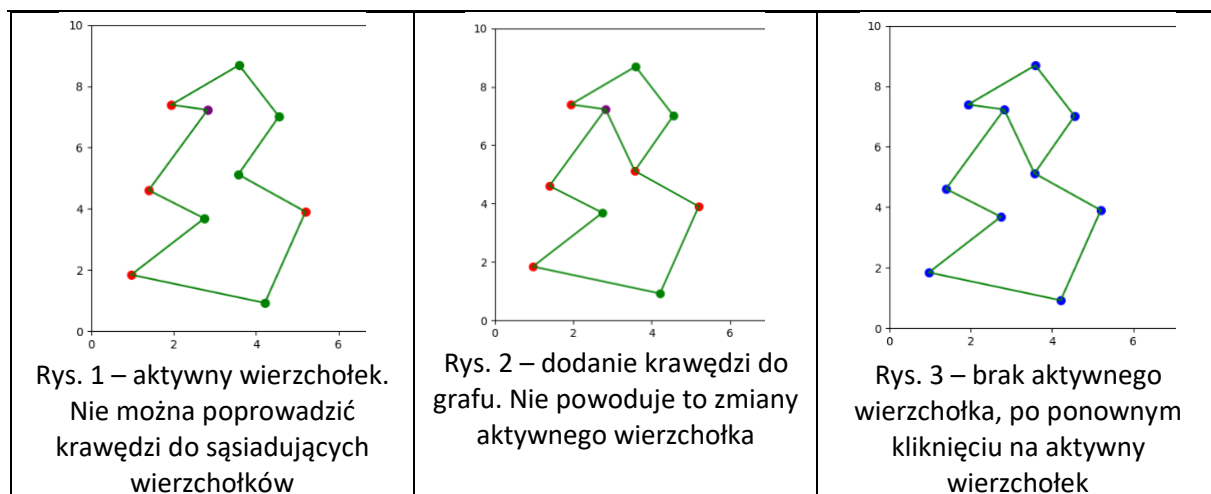
Graf należy wprowadzić przy pomocy paru funkcji.

Na początku wczytujemy wielokąt przy pomocy funkcji `create_polygon`. Uruchamia ona interaktywny wykres, na którym zaznacza się korzystając z myszki kolejne punkty wielokąta. Następnie należy wywołać funkcję `make_init_graph`, która zwróci graf powstały z zamiany wielokąta w graf i odwróci kolejność części krawędzi, tak by każda krawędź szła z dołu na górę (rosnąco względem y).

Później, w celu dodania następnych krawędzi stworzymy instancję klasy `Interactive_graph` podając graf jako argument. Po wywołaniu metody `run`, wyświetla się interaktywny wykres z grafem. Po kliknięciu jednego z wierzchołków grafu, zmienia on kolor na fioletowy, a reszta wierzchołków zmienia kolor w zależności czy można do nich poprowadzić krawędź (zielony) lub nie (czerwony) (rys. 1). Po kliknięciu prawidłowego wierzchołka zostaje dodana krawędź, co jest od razu zaznaczone na wykresie i można kontynuować dodawanie krawędzi z zaznaczonego wierzchołka (rys. 2). Aby zmienić wierzchołek, z którego dodajemy krawędzie, należy kliknąć w fioletowy wierzchołek, co spowoduje usunięcie go jako aktywnego wierzchołka i będzie można wybrać następny (rys. 3).

Końcowy graf jest w polu `graph` obiektu klasy `Interactive_graph`

Punkt wpisuje się przy pomocy funkcji `collect_points` która zwraca punkty kliknięte przy użyciu myszki na interaktywnym wykresie. W przypadku kliknięcia więcej niż jednego punktu, wybrany zostaje ostatnio kliknięty punkt.



5. Algorytm

Lokalizacja punktów metodą separatorów to podejście w geometrii obliczeniowej, które polega na użyciu specjalnych krzywych zwanych separatorami do podziału przestrzeni na obszary.

Separatorem nazywamy łamaną monotoniczną względem danego kierunku, rozdzielającą podział i tworzoną przez jego krawędzie.

Algorytm tworzy ciąg separatorów o następujących właściwościach:

- Każde dwa sąsiednie separatory wyznaczają jeden obszar należący do zadanego obszaru poligonowego;
- Każdy podobszar obszaru poligonowego jest wyznaczany przez separatory;
- Separatory nie przecinają się, mogą mieć co najwyżej wspólne krawędzie
- Separatory są posegregowane od separatora znajdującego się najbardziej na lewo, do separatora znajdującego się najbardziej na prawo;
- Separatory mają te same ujście i źródło.

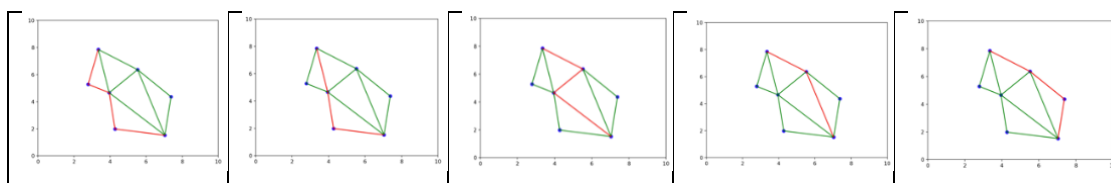
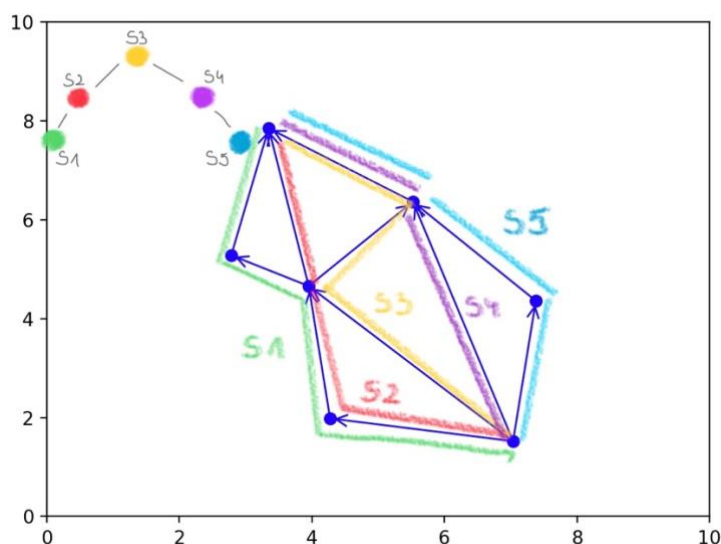


Tabela 1. Przykład ciągu separatorów jakie powinien znaleźć nasz algorytm.



Aby zoptymalizować wyszukiwanie, do przetrzymywania separatorów używamy zoptymalizowanego drzewa binarnego, którego węzłami są odpowiednie separatory. Dodatkowo w węzłach trzymamy krawędzie separatora, które nie były zawarte w wyższych węzłach. Tym sposobem możemy zaoszczędzić pamięć i przyspieszyć algorytm.



Rys 4. Pokazanie struktury drzewa dla separatorów z tabeli 1.

Główną częścią algorytmu jest dwukrotne wyszukiwanie binarne. Na poziomie pierwszym przeszukujemy separatory, a głębiej przeszukujemy zawarte w węźle krawędzie separatora określając czy punkt znajduje się po lewej czy prawej stronie separatora.

Jako iż, separatory jednoznacznie określają nam podobszar podziału poligonowego. Po znalezieniu separatora, który spełnia warunki:

- Znajduje się poniżej punktu;
- Separator go poprzedzający znajduje się powyżej szukanego punktu;

jednoznacznie możemy określić na którym znajduje się punkt. Algorytm lokalizuje punkt w czasie: $O(\log^2 n)$, gdzie n – liczba krawędzi.

6. Implementacja algorytmu

Warunki: Podział obszaru poligonowego musi być obszarem regularnym.

Obszar regularny – zbiór wierzchołków podziału tworzy zbiór uporządkowany względem współrzędnej y-owej, a dla każdego wierzchołka (z wyjątkiem najniższego i najwyższego) – v_p , istnieją krawędzie (v_i, v_p) oraz (v_p, v_j) gdzie, indeksy spełniają warunki $i < p < j$.



Wagi krawędzi obszaru należy dobrać tak, aby:

- waga każdej krawędzi są większe lub równe 1;
- suma wag krawędzi wchodzących do danego wierzchołka były równa sumie wag krawędzi wychodzących;

Wagi krawędzi w zaimplementowanym przez nas algorytmie są wyliczone na podstawie poniższego pseudokodu:

for każda krawędź w : $W(e) := 1$

for każdy wierzchołek v w kolejności indeksów rosnących: $W_{IN}(v) := \sum_{e \in IN(v)} W(e)$

d : = pierwsza z lewej krawędź odchodząca z v

if $W_{IN}(v) > W_{OUT}(v)$

then $W(d) := W_{IN}(v) - W_{OUT}(v) + W(d)$

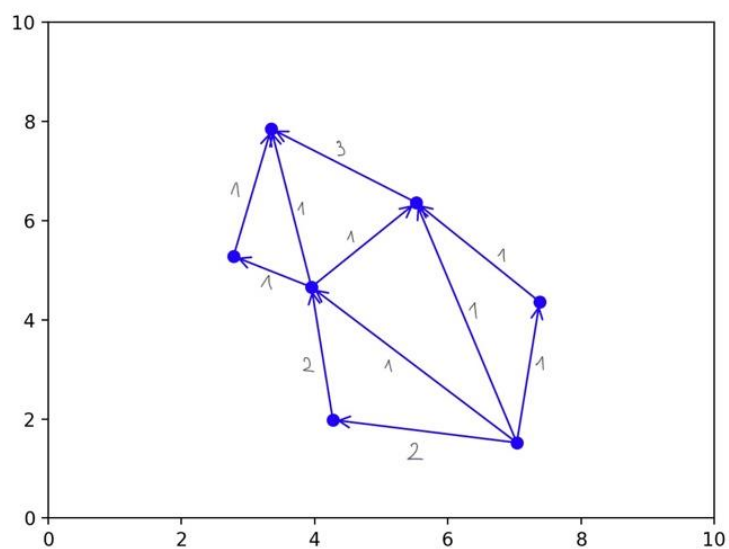
for każdy wierzchołek v w kolejności indeksów malejących:

$W_{OUT}(v) := \sum_{e \in OUT(v)} W(e)$

d : = pierwsza z lewej krawędź dochodząca do v

if $W_{OUT}(v) > W_{IN}(v)$

then $W(d) := W_{OUT}(v) - W_{IN}(v) + W(d)$



Rys 4. Przykład działania pseudokodu nadającego wagi krawędziom podziału.



Następnie wyznaczamy separatory w odpowiedniej kolejności, rekurencyjnie przemieszczając się po grafie. Przy poruszaniu zawsze wybieramy położoną najbardziej po lewej krawędź wychodząca z aktualnego punktu i mającą niezerową wagę. Po wybraniu krawędzi zmniejszamy o jeden jej wagę.

7. Testy efektywności

Test 1.

Liczba krawędzi: $n = 5$

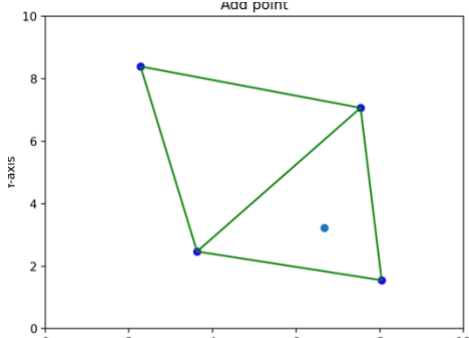
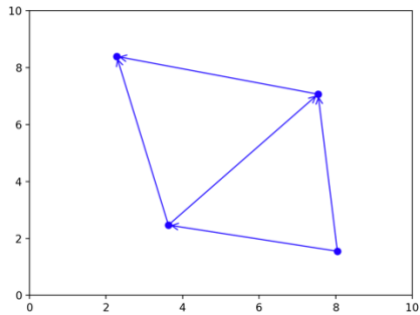
Obszar	Graf skierowany
 <p>Rys. 5. Obszar testowy z 4 wierzchołkami.</p>	 <p>Rys. 6. Obszar testowy z (rys.5) przekształcony na graf skierowany.</p>

Tabela 2.

Separatory:

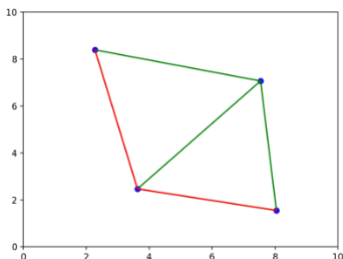
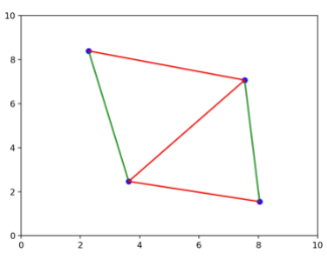
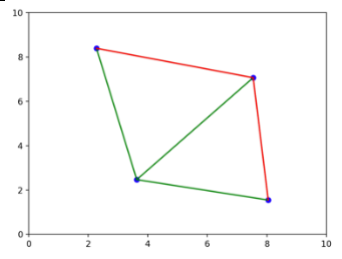
 <p>Rys. 6. Separator o indeksie 1, danych testowych nr.1</p>	 <p>Rys. 7. Separator o indeksie 2, danych testowych nr.1</p>	 <p>Rys. 8. Separator o indeksie 3, danych testowych nr.1</p>
--	--	--

Tabela 3.



Wynik programu:

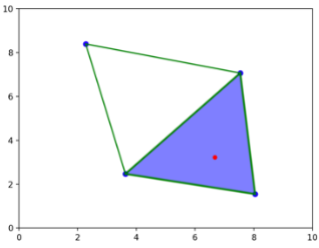
 <p>Rys. 9. Obszar znaleziony przez algorytm dla danych testowych 1.</p>	Czas działania poszczególnych elementów algorytmu	
	Znalezienie separatorów	0.0 s
	Stworzenie drzewa binarnego	0.0 s
	Lokalizacja punktu	0.0 s

Tabela 4.

Test 2.

Liczba krawędzi: $n = 16$

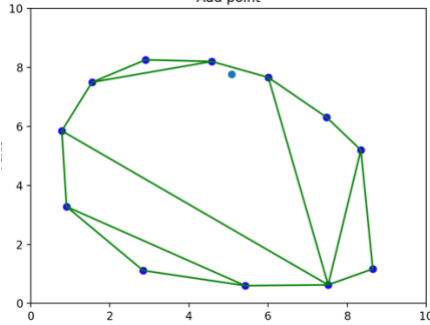
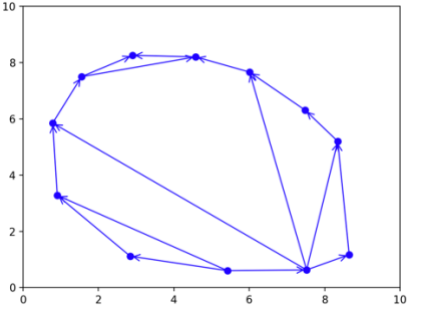
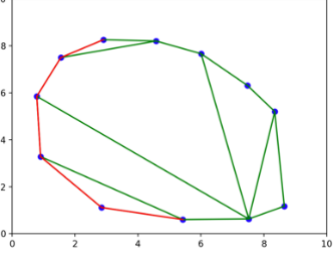
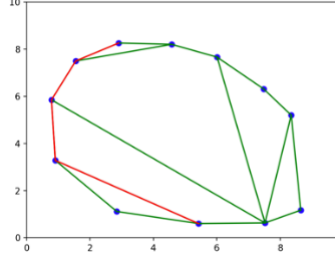
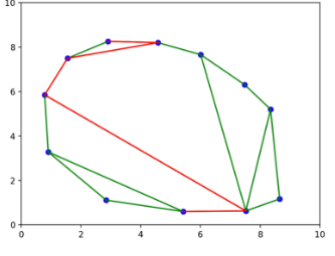
Obszar	Graf skierowany
 <p>Rys. 10. Obszar testowy nr 2.</p>	 <p>Rys. 11. Obszar testowy z (rys.10) przekształcony na graf skierowany.</p>

Tabela 5.

Separator:

 <p>Rys. 12. Separator o indeksie 1, danych testowych nr.2.</p>	 <p>Rys. 13. Separator o indeksie 2, danych testowych nr.2.</p>	 <p>Rys. 14. Separator o indeksie 3, danych testowych nr.2.</p>
--	--	--

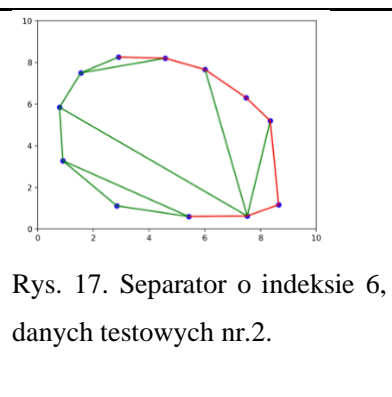
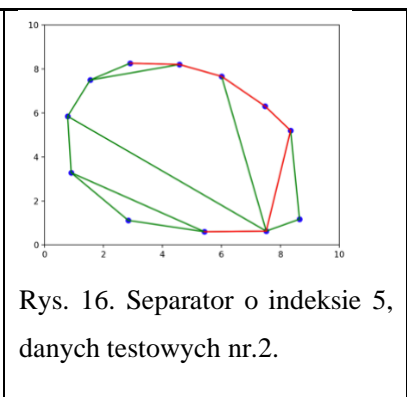
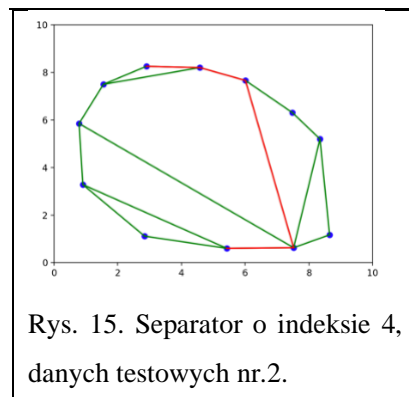


Tabela 6.

Wynik programu:

	Czas działania poszczególnych elementów algorytmu	
	Znalezienie separatorów	0.0 s
	Stworzenie drzewa binarnego	0.0 s
	Lokalizacja punktu	0.0 s

Tabela 7.

Test 3.

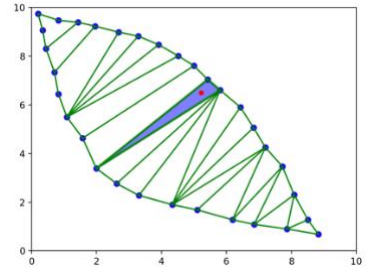
Liczba krawędzi: $n = 54$

Obszar	Graf skierowany

Tabela 8.



Wynik programu:

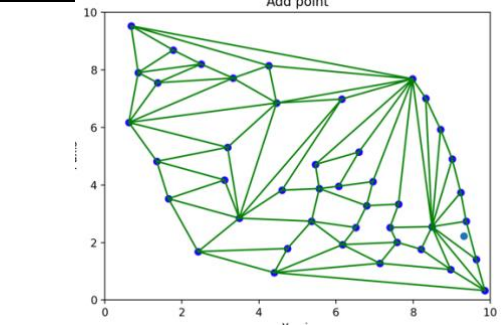
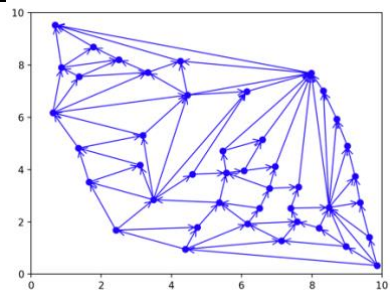
	Czas działania poszczególnych elementów algorytmu	
	Znalezienie separatorów	0.0 s
	Stworzenie drzewa binarnego	0.0 s
	Lokalizacja punktu	0.0 s

Rys. 21. Obszar znaleziony przez algorytm dla danych testowych 3.

Tabela 9.

Test 4.

Liczba krawędzi: $n = 91$

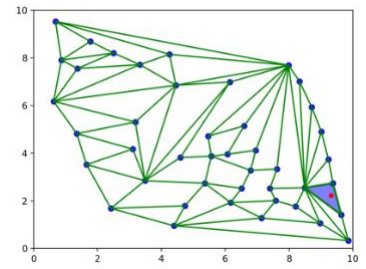
Obszar	Graf skierowany
	

Rys. 22. Obszar testowy nr 4.

Rys. 23. Obszar testowy z (rys.22) przekształcony na graf skierowany.

Tabela 10.

Wynik programu:

	Czas działania poszczególnych elementów algorytmu	
	Znalezienie separatorów	0.0 s
	Stworzenie drzewa binarnego	0.0 s
	Lokalizacja punktu	0.0 s

Rys. 24. Obszar znaleziony przez algorytm dla danych testowych 4.

Tabela 11.



AKADEMIA
GÓRNICZO-HUTNICZA



8. Wizualizacja algorytmu

Przy pomocy funkcji `draw_separator` przyjmującej jako argument graf oraz separator, wyświetlane są separatory znajdujące się przez funkcję `find_separators`.

Funkcja `draw_two_separators` przyjmująca jako argumenty graf oraz 2 separatory, wyświetla na jednym grafie oba separatory. Wykorzystywana jest do wizualizacji separatorów znalezionych przez funkcję `localize_point`.

Współrzędne obszaru, w którym jest punkt, są znajdowane korzystając z funkcji `find_field_coordinates`, a następnie są wyświetlane przy pomocy funkcji `draw_field` wraz z szukanym punktem.

9. Implementacja wizualizacji

Do stworzenia wizualizacji została użyta biblioteka `matplotlib`. Wykorzystano pomocniczą metodę klasy `Graph` – `draw`, która poza narysowaniem grafu zwraca wykres na którym się znajduje, dzięki czemu można na grafie dorysowywać dodatkowe elementy. Metoda `draw_with_arrows` działa analogicznie, z tą różnicą, że na końcowym wykresie pokazany jest kierunek krawędzi, dzięki wykorzystaniu klasy `FancyArrowPatch` z modułu `matplotlib.patches`. Klasa `Interactive_graph` została napisana na potrzeby dodawania krawędzi przez użytkownika. Zawiera ona pole przechowujące aktualnie aktywny wierzchołek grafu. Podczas rysowania, które odbywa się w momencie dokonania zmian, ten punkt rysowany jest na fioletowo. Natomiast kolor wszystkich innych punktów jest zależny od wyniku metody `possible_connections`, która sprawdza możliwość poprowadzenia krawędzi między tym wierzchołkiem a aktywnym wierzchołkiem, biorąc pod uwagę czy krawędź między tymi wierzchołkami już istnieje oraz czy dodanie takiej krawędzi nie sprawiłoby, że krawędzie by się przecinały. Wykorzystano do tego analityczną metodę rozwiązywania równań prostych, na których leżą poszczególne krawędzie.

Zarówno `draw_separator` i `draw_two_separators` iterują po separatorach i dorysowują je na grafie.



Funkcja `find_field_coordinates` wyszukuje punktów tworzących wielokąt poprzez znalezienie krawędzi separatorów, które nie są takie same, a następnie dodaje na dwie listy różnice w punktach. Gdy separatory ponownie się spotkają, jedna z list jest obracana (pierwszy element zostaje ostatnim) i doczepiona na koniec tej drugiej. Dzięki temu punkty są ułożone w kolejności zgodnej lub niezgodnej z ruchem wskazówek zegara, co pozwala na wyświetlenie wielokąta przy pomocy klasy `Polygon` modułu `matplotlib.patches`.

10. Wnioski

Algorytm działa poprawnie i szybko. Metoda okazuje się efektywna czasowo zwłaszcza dla dużych zbiorów danych, ponieważ separator pozwala na szybkie wykluczenie obszarów, które nie zawierają poszukiwanego punktu. Dodatkowo metoda działa zarówno dla podziału przestrzeni na trójkąty jak i inne wielokąty. Istnieją jednak problemy takie jak np. zmiana obszaru na obszar regularny, które stwarzają dość duże problemy implementacyjne i mogą być trudne do zakodowania. Nie zawsze jest więc możliwość użycia naszej implementacji.

11. Bibliografia

1. Dr. Inż. Barbara Głut, prezentacja *Lokalizacja punktu*, stworzona na potrzeby programu Algorytmy Geometryczne [odczyt 27.12.2023].
2. Herbert Edelsbrunner, Leo J. Guibas, Jorge Stolfi, *Optimal Point Location in Monotone Subdivision* [dok. elektroniczny], <https://graphics.stanford.edu/courses/cs268-07-winter/manuals/monotone-point-loc.pdf> [odczyt 27.12.2023], 1984.
3. Artykuł. *Point Location* [dok. elektroniczny], https://en.wikipedia.org/wiki/Point_location [odczyt 27.12.2023].
4. Jack Snoeink, *Point location* [dok. elektroniczny] <https://www.csun.edu/~ctoth/Handbook/chap38.pdf> [odczyt 27.12.2023].