Szymon Szarek

Kacper Garus

Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda separatorów.

1. Wprowadzenie

W ramach projektu przygotowano implementację algorytmu lokalizacji punktu w przestrzeni dwuwymiarowej metodą separatorów.

2. Środowisko pracy

Program został napisany w języku Python za pomocą platformy Jupyter Notebook. Jako narzędzie graficzne do wizualizacji figur geometrycznych zostało użyte narzędzie przygotowane przez koło naukowe BIT, które wykorzystuje takie biblioteki jak np. matplotlib lub numpy.

Program został przygotowany na dwóch komputerach:

	PC 1	PC 2
System	Windows 11 x64	Windows 11 x64
Procesor	AMD Ryzen PRO 5650U	Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Pamięć RAM	16GB	16GB
Python	3.9	

3. Sposób obsługi

Aby używać programu, należy po kolei wywoływać komórki z kodem z pliku main o rozszerzeniu Jupyter Notebook (.ipynb). Aby wprowadzić własny podział poligonowy należy edytować plik raw.py, w którym znajduje się podział poligonowy w formie grafu skierowanego. Zmienna vertices zawiera listę dwuelementowych krotek, które oznaczają współrzędne danych wierzchołków, a zmienna edges zawiera indeksy wierzchołków z listy vertices pomiędzy którymi znajduje się krawędź. Podział poligonowy jak i postać grafowa muszą spełniać wymogi metody separatorów.

Zbiór wierzchołków podziału musi tworzyć ciąg uporządkowany względem współrzędnej y-owej (w przypadku równych wartości -względem współrzędnej x-owej). Wierzchołek vk jest regularny, gdy istnieją krawędzie (vi,vk) i (vk,vj) dla i < k < j. Podział jest regularny, gdy wszystkie wierzchołki ciągu poza pierwszym i ostatnim są regularne.

Każdy podział można zregularyzować stosując algorytm podobny do algorytmu podziału wielokąta na wielokąty monotoniczne (łączymy krawędzią wierzchołek nieregularny v z pomocnikiem sąsiadującej z v krawędzi) lub triangulując otoczkę wypukłą podziału.

4. Dokumentacja

4.1 Node

zmienne:

- x – pierwsza współrzędna
- y – druga współrzędna
- nodesOut – lista krotek zawierająca krawędzie wychodzące i ich wagi
- nodesIn – lista krotek zawierająca krawędzie wchodzące i ich wagi
- wIn – suma wag z nodesIn
- wOut – suma wag z nodesOut

4.2 sortEdges(vertices: List[Node])

Sortuje wierzchołki względem krawędzi, ustawiając je od najbardziej wysuniętych na prawo do najbardziej wysuniętych na lewo.

- Argumenty:

  - vertices (List[Node]): Lista wierzchołków do posortowania.

- Zwracana Wartość:

  - None

4.3 det(a, b, c)

funkcja zwraca orientacje 3 punktów

4.4 cmp1(n1, n2)

komparator potrzebny do funkcji sortEdges

4.5 cmp2(n1, n2)

komparator potrzebny do funkcji sortEdges

4.6 loadData(vertices: List[Tuple[float, float]], edges: List[Tuple[int, int]]) -> List[Node]

Wczytuje dane z wierzchołków i krawędzi, tworząc obiekty Node reprezentujące wierzchołki z połączeniami krawędziowymi.

- Argumenty:

  - vertices (List[Tuple[float, float]]): Lista krotek z współrzędnymi x i y wierzchołków.

  - edges (List[Tuple[int, int]]): Lista krotek reprezentująca połączenia krawędziowe między wierzchołkami.

- Zwracana Wartość:

  - List[Node]: Lista obiektów Node z wczytanymi danymi.

4.7 calculateWeights(vertices: List[Node])

Oblicza wagi wierzchołków na podstawie krawędzi, uwzględniając posortowany porządek krawędzi.

- Argumenty:

  - vertices (List[Node]): Lista posortowanych wierzchołków.

4.8 Separator

Klasa przechowująca punkty i krawędzie jako elementy separujące w grafie.

4.9 findSeparators(graph: List[Node]) -> List[Separator]

Generuje separatory na podstawie danego grafu, reprezentując je jako obiekty Separator.

- Argumenty:
  - graph (List[Node]): Graf reprezentowany jako lista wierzchołków.
- Zwracana Wartość:
  - List[Separator]: Lista obiektów Separator reprezentujących separatory w grafie.

4.10 TreeNode

Klasa reprezentująca węzeł w drzewie separatorów z informacjami o segmentach i separatorze.

- Argumenty:
  - segments (List[Tuple[Tuple[float, float], Tuple[float, float]]]): Lista segmentów w węźle.
  - separator (Separator): Separator dla danego węzła.
  - parent (TreeNode): Rodzic węzła.

4.11 arrayToBST(array: List[TreeNode], parent: TreeNode) -> TreeNode

Konstruuje drzewo BST na podstawie posortowanej listy węzłów separatorów.

- Argumenty:
  - array (List[TreeNode]): Posortowana lista węzłów separatorów.

- parent (TreeNode): Rodzic aktualnie budowanego węzła.

- Zwracana Wartość:

- TreeNode: Korzeń zbudowanego drzewa BST.

4.12 location(point: Tuple[float, float], root: TreeNode, prev: TreeNode) -> Union[Tuple[Tuple[float, float], Tuple[float, float]], Separator]

Określa położenie punktu względem segmentów w danym węźle drzewa separatorów, zwracając odpowiedni segment lub separator.

- Argumenty:

- point (Tuple[float, float]): Współrzędne punktu, który ma zostać zlokalizowany.

- root (TreeNode): Aktualny węzeł drzewa separatorów.

- prev (TreeNode): Poprzedni węzeł w drzewie separatorów.

- Zwracana Wartość:

- Union[Tuple[Tuple[float, float], Tuple[float, float]], Separator]: Segment lub separator, w zależności od lokalizacji punktu.

4.13 second_separator(s: Separator, separators: List[Separator], point): Tuple[float, float]) -> Separator

Znajduje drugi separator, do którego należy punkt, na podstawie danego separatora s.

- Argumenty:

- s (Separator): Separator, dla którego znajdowany jest drugi separator.

- separators (List[Separator]): Lista separatorów.

- point (Tuple[float, float]): Współrzędne punktu, dla którego szukany jest drugi separator.

- Zwracana Wartość:

  - Separator: Drugi separator, do którego należy punkt.

4.14 exact_area_edges(sep1, sep2, point):

-Argumenty:

-sep1, sep2: wyznaczone przez nas separatory

-point: współrzędne lokalizowanego punktu

-Zwracana wartość

-krawędzie otaczające nasz punkt

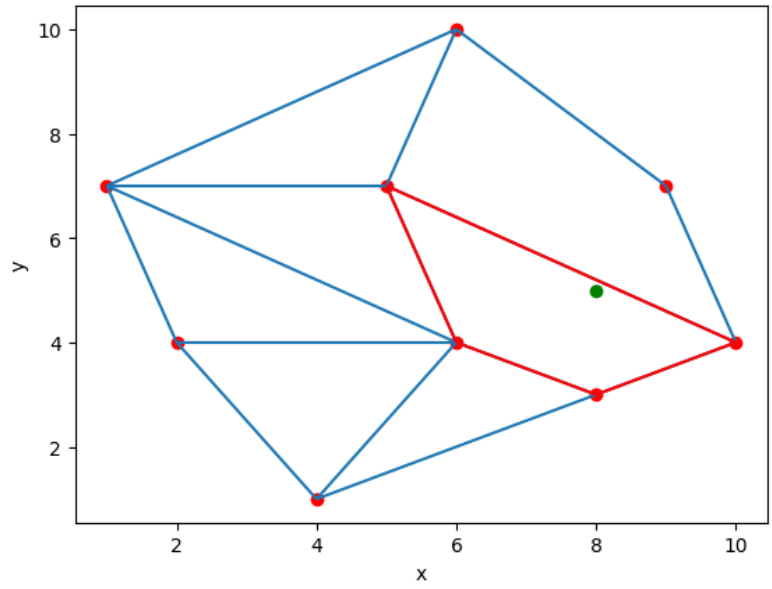
4.15
separators_method_point_location_algorithm(raw_vertices,raw_edges,point):

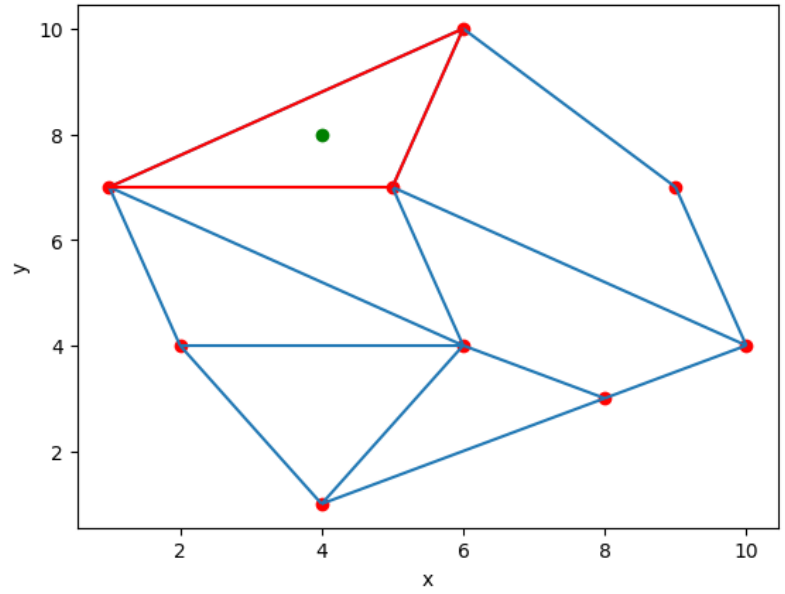
-funkcja przyjmująca dane wejściowe i wywołująca po kolei funkcje aby wyznaczyć dany obszar

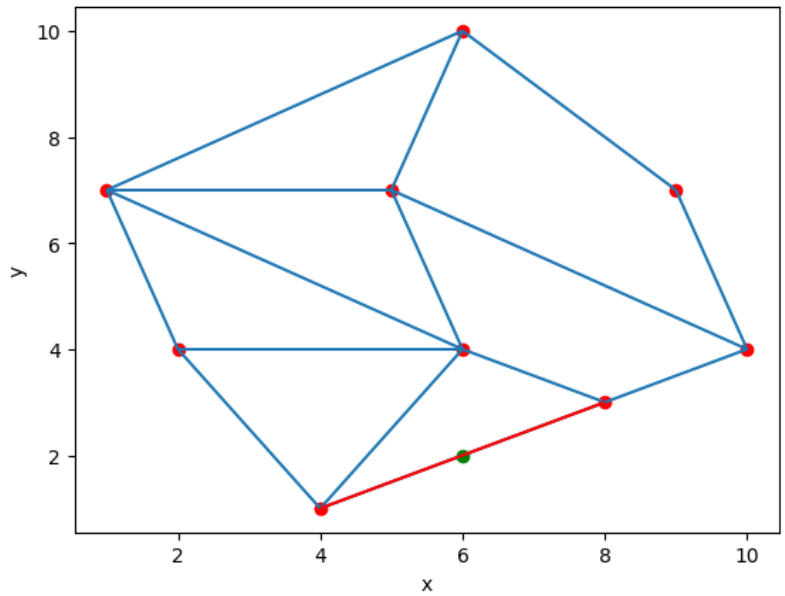
5. Sprawozdanie

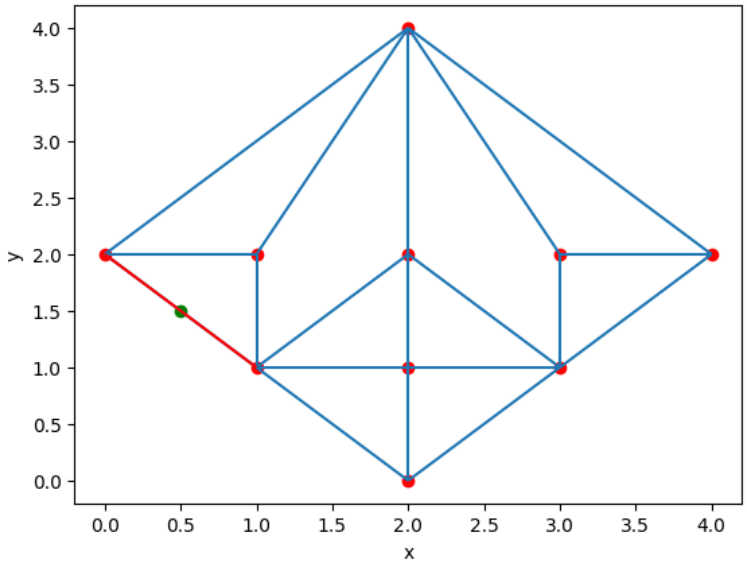
Algorytm przyjmuje dane wejściowe i zamienia je w strukturę grafu skierowanego ważonego. Na początku przechodzimy dwa razy po wierzchołkach grafu obliczając każdą wagę krawędzi. Wagi krawędzi oznaczają ilość separatorów, do których dana krawędź należy. Następnie wyznaczamy separatory przechodząc rekurencyjnie po krawędziach od źródła do ujścia. Po wyznaczeniu wszystkich separatorów otrzymujemy posortowaną listę separatorów. Z niej tworzymy drzewo BST, po którym przechodzimy wyznaczając dwa separatory wyznaczające obszar, do którego należy dany punkt. Jeżeli punkt leży na krawędzi – zwracamy od razu daną krawędź.

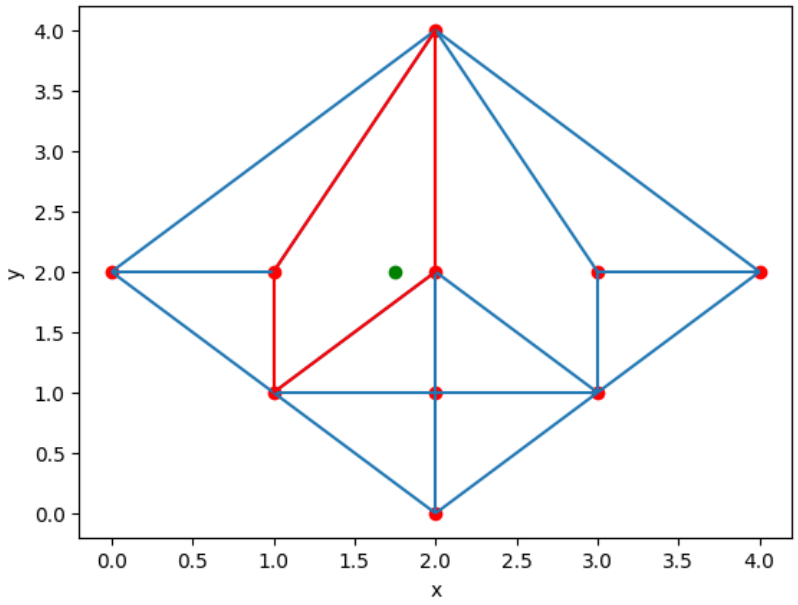
5.1 Wyniki algorytmu

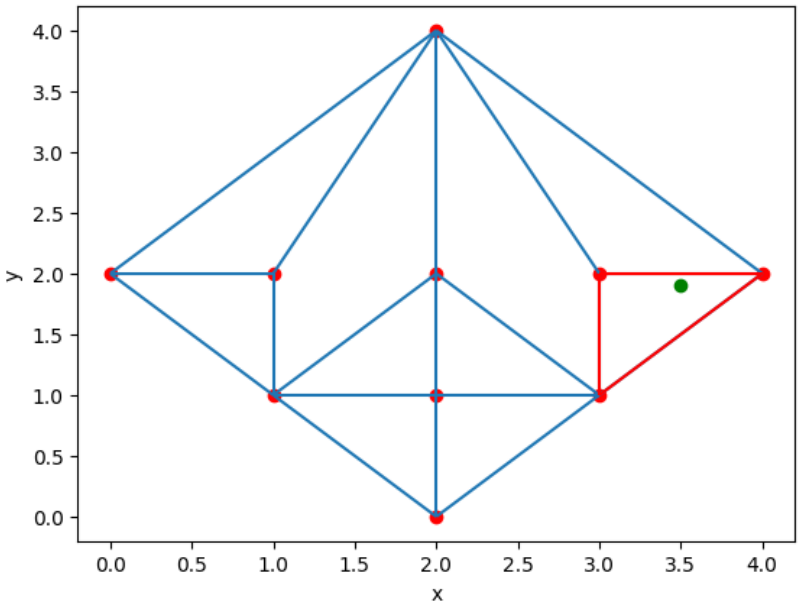

10
8
6
4
2
y
2
4
6
8
10
x

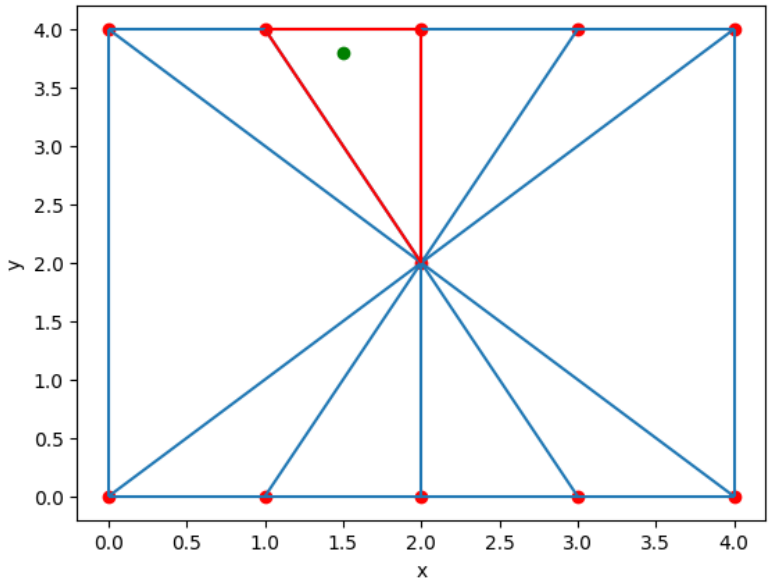

10
8
6
4
2
y
2
4
6
8
10
x

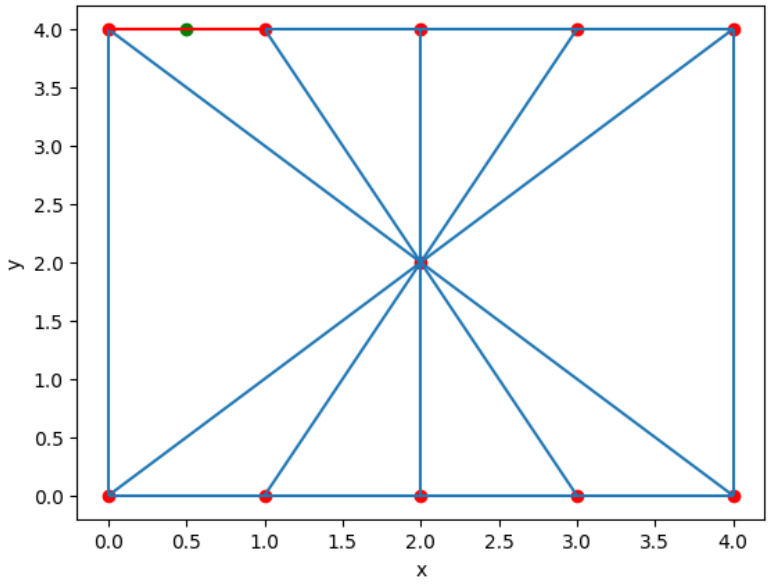

10
8
6
4
2
y
2
4
6
8
10
x

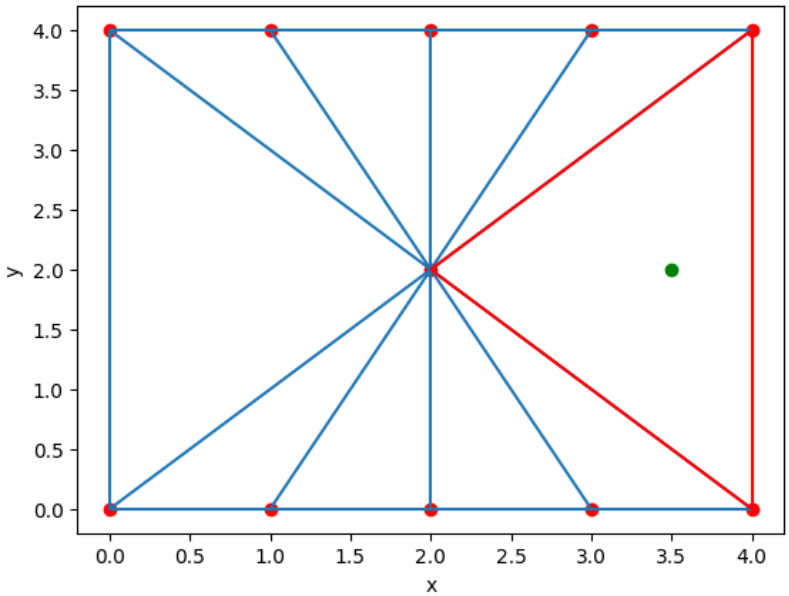

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x

5.2 Wnioski

Na testowanych danych algorytm działa poprawnie

6. Bibliografia

Wiedza byłą czerpana głównie z wykładu oraz poniższych książek:

H. Edelsbrunner, L. J. Guibas, J Stolfi: Optimal Point Location in a Monotone Subdivision; październik, 1984

E. Veach: Point location Methods; luty 1992