Algorytmy geometryczne

Sprawozdanie - ćwiczenie 3.

Dominik Adamczyk

Gr. 2

1. Specyfikacja techniczna urządzenia, na którym wykonano ćwiczenie

- System: Windows 11 Home x64
- Procesor: Intel Core i5-1135G7 2.4MHz – 4.2 MHz
- Pamięć ram: 16 Gb
- Język programowania: Python 3.11 z bibliotekami numpy, matplotlib, json, enum
- Środowisko: Jupyter Notebook z użyciem narzędzia załączonego na potrzeby laboratoriów

2. Treść i cel ćwiczenia

Ćwiczenie polegało na zapoznaniu się z tematem triangulacji wielokątów monotonicznych poprzez zaimplementowanie algorytmów sprawdzających monotoniczność wielokąta, kategoryzację jego wierzchołków i tworzenie jego triangulacji.

3. Przystosowanie aplikacji

Na potrzeby ćwiczenia do aplikacji graficznej zostały dodane odpowiednie funkcje umożliwiające zapisywanie i odczytywanie wykresów z plików. Została też dodana funkcja tworząca czyste płótno (z możliwością wskazania jego wymiarów), na którym można narysować wielokąt, a później przeprowadzić na nim odpowiednie operacje.

4. Reprezentacja wielokąta

Do reprezentacji wielokąta, a także składających się na niego punktów i krawędzi zostały utworzone nowe klasy obiektów, tak aby ułatwić przetwarzanie danych wejściowych i wykonywanych na nich operacji. Obiektowość umożliwia również dodanie atrybutów do każdego z obiektów – jest to szczególnie pomocne podczas wizualizacji, gdzie każdy punktów powinien mieć odpowiednie pokolorowanie wnioskowane na podstawie przypisanych atrybutów.

Podstawową klasą jest Point() reprezentująca pojedynczy punkt, w szczególności należący do zadanego wielokąta. Pola tego obiektu to współrzędne x i y, pola odpowiadające za typ punktu podczas wyświetlania animacji, oraz kategoryzacji, a także pole informujące o tym czy punkt jest w prawym łańcuchu wielokąta, czy w lewym. Pole ord przechowuje informację o indeksie punktu, jeżeli tablica z punktami reprezentowanymi wielokąt zostałaby posortowana po współrzędnych.

Klasa Edge() reprezentuje krawędź i tworzona jest w oparciu o dwa obiekty klasy point. W powyższych klasach metoda getIterable() zwraca listę/krotkę z danymi na temat wartości x i y danego obiektu. Jej użycie jest przydatne do przeprowadzenia wizualizacji przy pomocy załączonego narzędzia.

Najistotniejszą klasą z punktu widzenia ćwiczenia jest klasa Polygon(). Przechowuje ona wielokąt w sposób, który umożliwia na nim dalszą pracę, a także posiada metody będące niektórymi z koniecznych do zaimplementowania w ćwiczeniu algorytmów. Pola inicjalizowane w konstruktorze tej klasy to:
- vertices – tablica zawierające punkty wielokąta. Kolejność punktów w tej tablicy wyznacza kolejne boki wielokąta. Punkty wejściowe – przekazane przez konstruktor są odpowiednio parsowane, tak aby w tablicy vertices wielokąt był zadany przeciwnie do ruchu wskazówek zegara, a pod indeksem 0 znajdował się punkt o największej współrzędnej y i najmniejszej x.

- edges – tablica zawierające krawędzie tworzące zadany wielokąt
- category – tablica do której kategoryzowane są punkty, gdy chcemy je sklasyfikować
- maxPoint, minPoint – punkt największy i najmniejszy pod względem współrzędnej y
- scenes – tablica do której dodawane są sceny, wykorzystywane podczas wizualizacji
- sortedIndexes – tablica z indeksami wskazującymi na elementy tablicy vertices. Jeżeli potrzebne jest uzyskanie informacji o n-tym punkcie w tablicy vertices pod względem posortowania jej malejąco po wartościach y wystarczy użyć vertices[sortedIndexes[n]].
- triangulation – tablica przechowująca triangulację wielokąta (po wywołaniu odpowiedniej funkcji). Triangulacja jest przechowywana jako lista krotek, gdzie w każdej krotce są 3 indeksy wskazujące na 3 punkty z tablicy vertices tworzące trójkąt będący elementem triangulacji. Zadany w ten sposób trójkąt jest zapisany przeciwnie do ruchu wskazówek zegara.

Poza powyższymi polami klasa Polygon() posiada metody odpowiadające za wizualizację wyników, oraz niektóre algorytmy:
- addSceneToAnimation() – dodaje scenę do tablicy scenes, co umożliwia wizualizację algorytmu triangulacji.
- getAnimation() – zwraca tablicę scen, którą należy wykorzystać w obiekcie klasy Plot
- getLCPolygon() – zwraca obiekt LinesCollection zawierający boki wielokąta
- getLCTriangulation() – zwraca obiekt LinesCollection z triangulacją na podstawie listy triangulation.
- getPC – zwraca obiekt PointsCollection z wierzchołkami wielokąta
- getPCCategorized – zwraca listę obiektów PointsCollection, umożliwiającą wizualizację kategoryzacji.
- getPCAnimation – zwraca listę obiektów PointsCollection potrzebną do prezentowania stanu wierzchołków podczas animacji algorytmu triangulacji
- addTriangle() – dodaje trójkąt (we wcześniej omówionej postaci) do listy triangulation
- isYMonotone() – zwraca informację, czy wielokąt jest y-monotoniczny
- categorize() – przeprowadza kategoryzację punktów
- saveTriangulationToFile() – po wykonaniu algorytmu triangulacji wywołanie tej metody zapisze triangulację (w postaci listy trójek indeksów wskazujących na punkty) wraz z listą punktów do wskazanego w argumencie pliku.

5. Działania algorytmów

Do wykonania implementowanych algorytmów konieczna jest funkcja wyznacznika, dzięki której można otrzymać informację o położeniu punktu c względem prostej ab. Na potrzeby tego ćwiczenia zaimplementowany został wyznacznik o wzorze $det(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$. Przyjęta tolerancja dla zera wynosi 0, chociaż możliwe jest jej zmienienie w programie, w celu sprawdzenia zachowania programu dla różnych jej wartości.

Pierwszym implementowanym algorytmem było sprawdzanie, czy wielokąt jest y-monotoniczny. Aby to sprawdzić wyznaczany jest punkt o najmniejszej i największej współrzędnej y wśród punktów wyznaczających wielokąt, a następnie lista tych punktów zostaje ułożona tak, żeby punkt o największej współrzędnej znajdował się na pierwszym miejscu listy (w przypadku mojej implementacji te kroki wykonywane są w konstruktorze obiektu klasy wielokąt). Po tym przygotowaniu następują dwie iteracje po każdych dwóch kolejnych punktach wielokąta. Pierwsza iteracja sprawdza, czy punkt poprzedzający ma

większą, bądź równą współrzędną od punktu następującego po nim, a kończy się w momencie, gdy badany zostaje punkt najmniejszy. Druga iteracja działa analogicznie, z tą różnicą, że punkt poprzedzający powinien mieć mniejszą, bądź równą współrzędną, co punkt następujący.

Kolejny implementowany algorytm kategoryzuje wierzchołki w zależności od tego, czy są początkowe, końcowe, łączące, dzielące i prawidłowe. Jego działanie jest oparte na sprawdzaniu każdych trzech kolejnych wierzchołków a, b, c wielokąta (w kolejności przeciwnej do ruchu wskazówek zegara). Kategoryzacja jest możliwa dzięki określeniu położenia wierzchołka b względem wierzchołków a i c, oraz określeniu położenia wierzchołka c względem prostej ab.

Algorytm triangulacji działa wyłącznie dla wielokątów y-monotonicznych. Wierzchołki wielokąta, który jest poddany triangulacji powinny być posortowane malejąco po współrzędnych y. W moim przypadku odpowiada za to lista sortedIndexes uzupełniana w konstruktorze obiektu klasy Polygon(). Dodatkowo wierzchołki powinny mieć przypisaną stronę na której się znajdują (prawa/lewa). W przypadku tej implementacji odpowiada za to odpowiednie pole w obiekcie klasy Point(), które jest uzupełniane w konstruktorze obietku klasy Polygon(). Algorytm przyjmuje, że wierzchołek największy i najmniejszy znajdują się na lewej krawędzi wielokąta. Po tych przygotowaniach możliwe jest przejście do właściwego algorytmu. Na stosie umieszczane są dwa pierwsze wierzchołki. Następnie przeprowadzona jest iteracja po pozostałych wierzchołkach z listy. Wierzchołek na którym aktualnie odbywa się iteracja będzie określany jako przetwarzany. Jeżeli wierzchołek przetwarzany znajduje się po innej stronie niż wierzchołek na szczycie stosu, to ze stosu zdejmowane są wszystkie wierzchołki, a każde dwa kolejne zdjęte wierzchołki razem z przetwarzanym tworzą trójkąt należący do triangulacji. Na stosie umieszczany jest wierzchołek poprzedzający przetwarzany i wierzchołek przetwarzany. Jeżeli natomiast wierzchołek przetwarzany i wierzchołek na szczycie stosu należą do tej samej strony, to sprawdzane są kolejne trójkąty tworzone przez dwa wierzchołki na szczycie stosu i wierzchołek przetwarzany. Jeżeli taki trójkąt zawiera się w wielokącie (sprawdza to funkcja isInside), to jest dodawany do triangulacji, a ze szczytu stosu zdejmowany jest wierzchołek i procedura sprawdzania (i ewentualnego zdjęcia ze stosu) powtarza się. W przeciwnym przypadku – gdy trójkąt nie należy do wielokąta, na stos dodawany jest wierzchołek przetwarzany.

Zaimplementowana w osobnej funkcji wizualizacja algorytmu triangulacji pokazuje każdy kolejny krok działania triangulacji – aktualną siatkę, wierzchołek przetwarzany, wierzchołki przetworzone, wierzchołki na stosie, a także trójkąty niezawierające się w wielokącie, gdy następuje ich sprawdzanie.

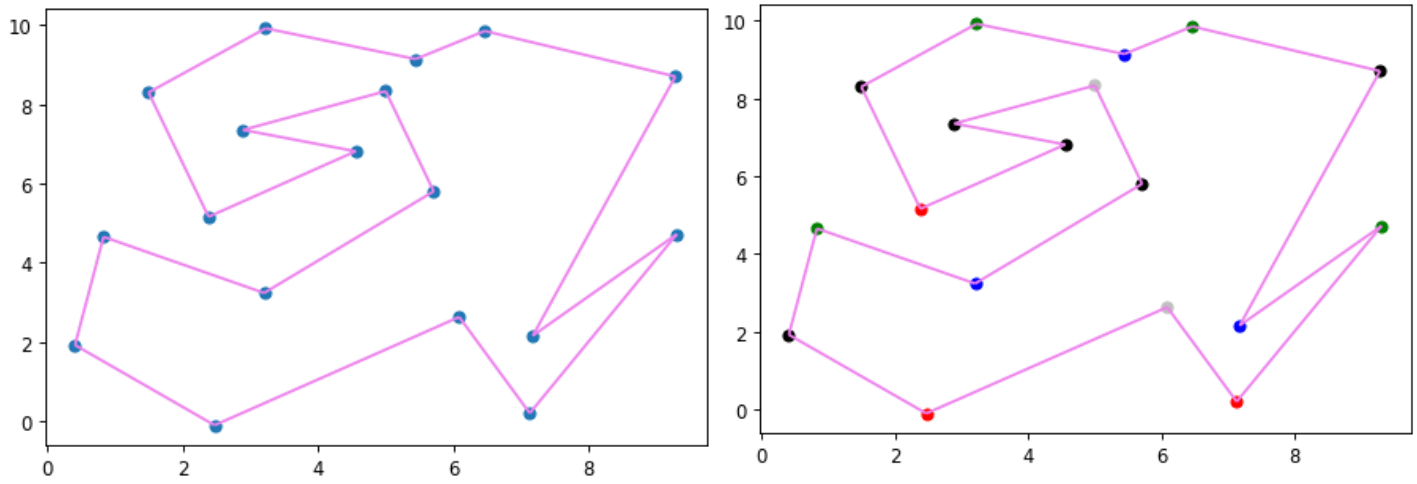
6. Przedstawienie klasyfikacji

Pierwszym krokiem bardziej rozbudowanego algorytmu triangulacji jest klasyfikacja wierzchołków według następującego schematu:

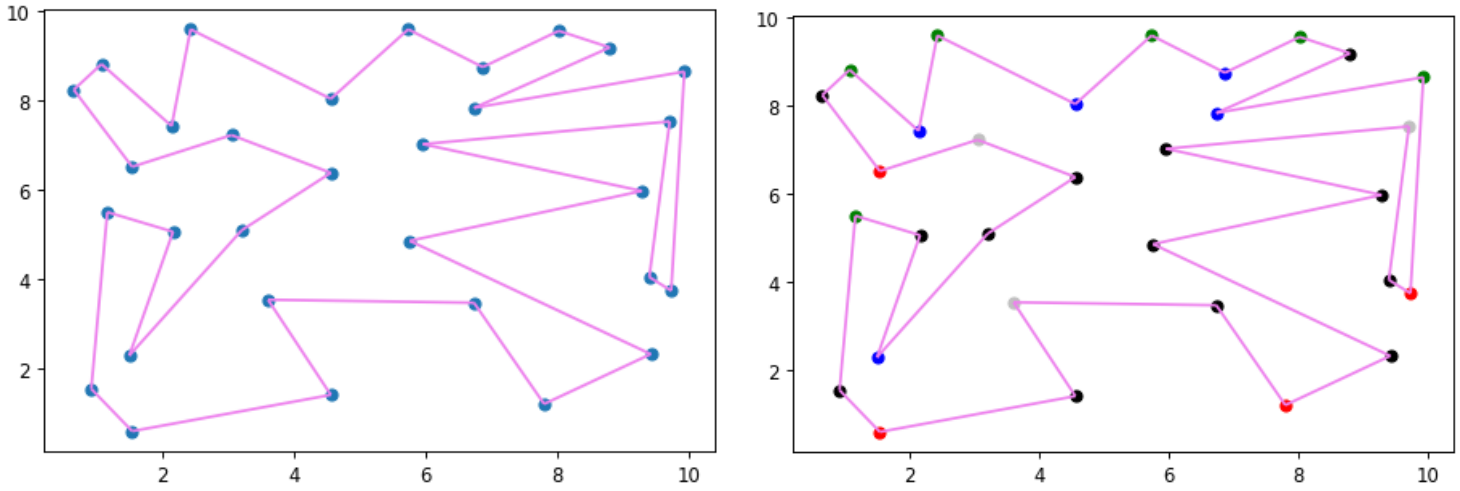
- Początkowy – obaj sąsiedzi wierzchołka leżą poniżej i kąt wewnętrzny $< \pi$ - kolor zielony
- Końcowy – obaj sąsiedzi leżą powyżej i kąt wewnętrzny $< \pi$ – kolor czerwony
- Łączący – obaj sąsiedzi leżą powyżej i kąt wewnętrzny $> \pi$ – kolor granatowy
- Dzielący – obaj sąsiedzi leżą poniżej i kąt wewnętrzny $> \pi$ – kolor szary
- Prawidłowy – w pozostałych przypadkach – kolor czarny.

Na grafikach prezentowane są wielokąty i kategoryzacja ich wierzchołków zgodnie z schematem kolorystycznym przedstawionym powyżej.

Rys. 1. Wielokąt przedstawiony na wykładzie i klasyfikacja jego wierzchołków



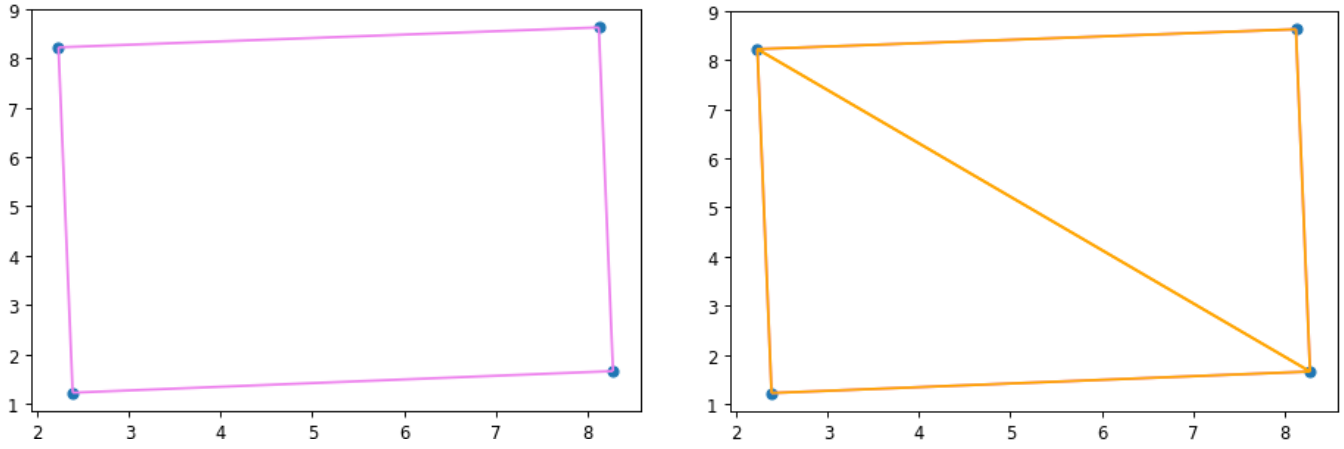
Rys. 2. Proponowany wielokąt niemonotoniczny i klasyfikacja jego wierzchołków



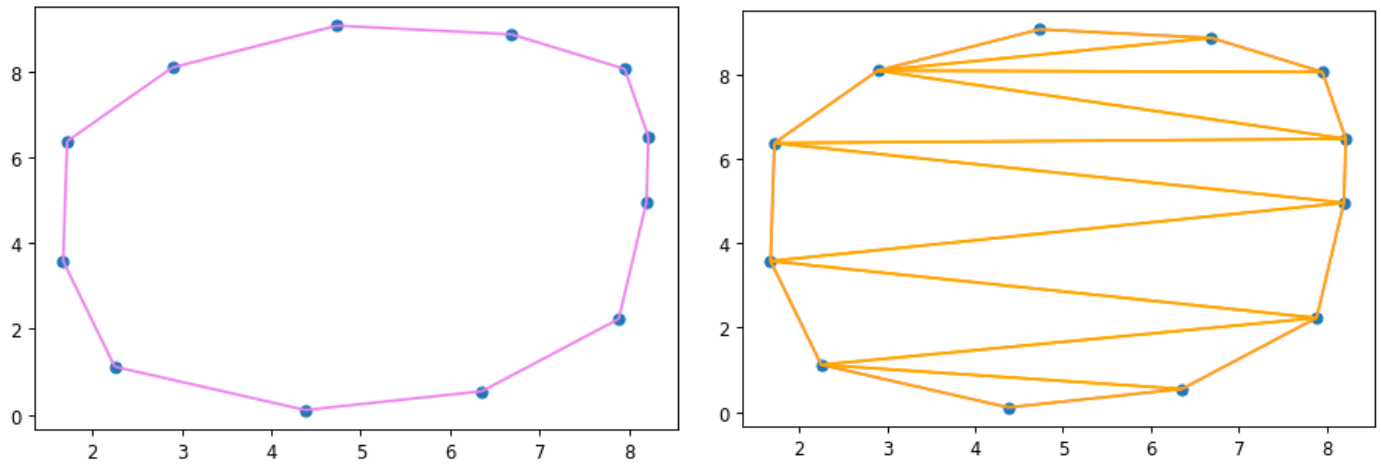
7. Przedstawienie triangulacji

Na prezentowanych poniżej grafikach przedstawione są zbiory danych będące wielokątami y-monotonicznymi wraz z ich triangulacjami. Dla każdego ze zbiorów danych możliwe jest również zobaczenie animacji za pośrednictwem plików jupyter notebooka.

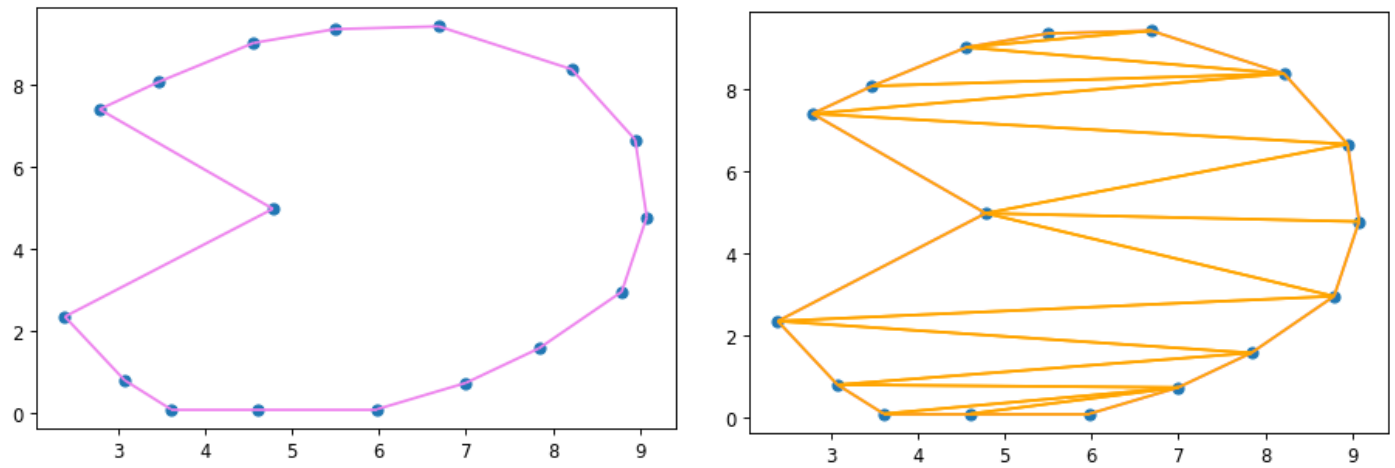
Rys. 3. Wielokąt 1 i jego triangulacja



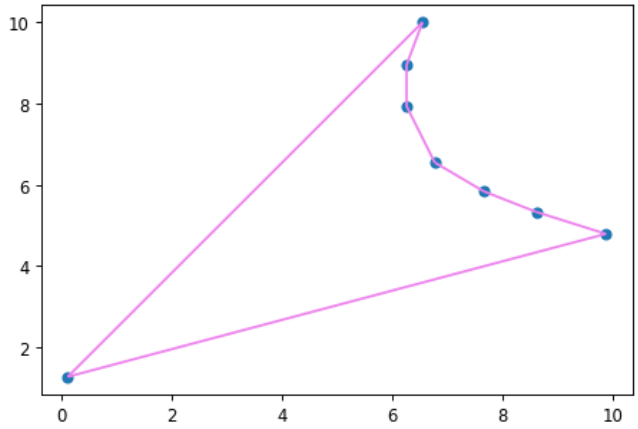
Rys. 4. Wielokąt 2 i jego triangulacja

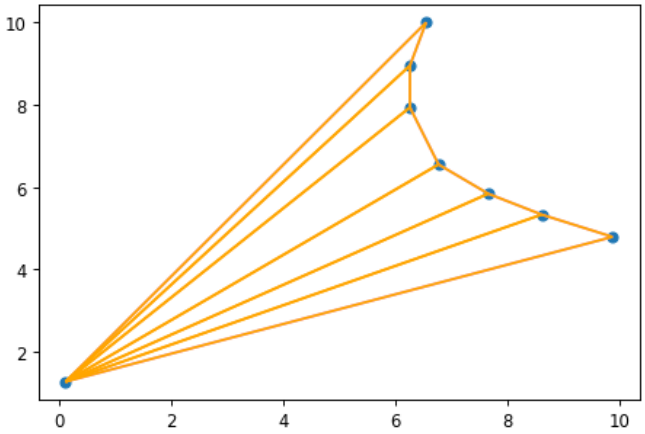


Rys. 5. Wielokąt 3 i jego triangulacja

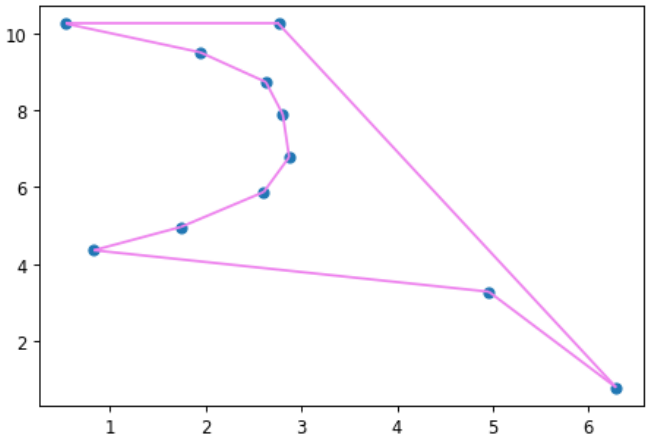


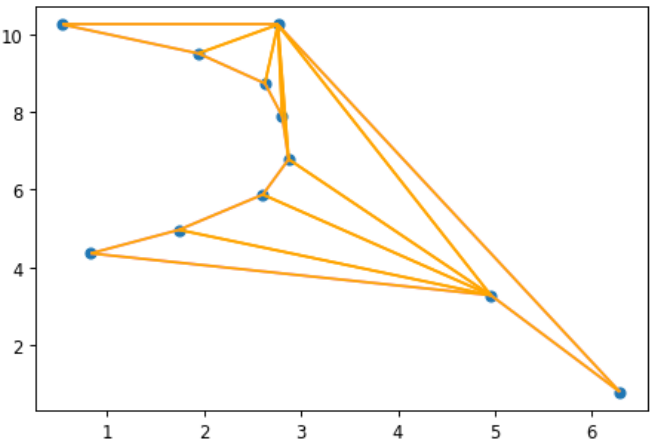
Rys. 6. Wielokąt 4 i jego triangulacja



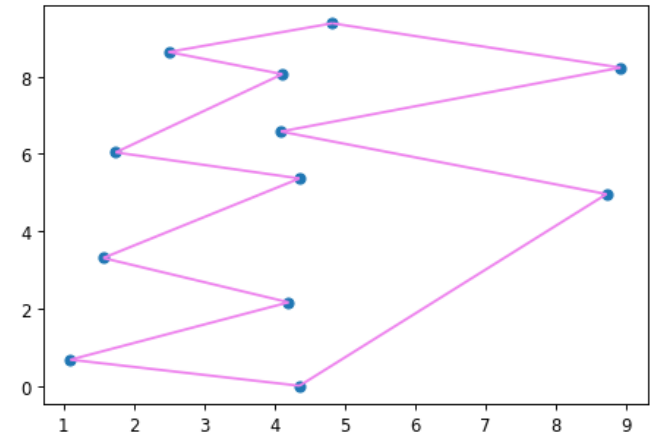


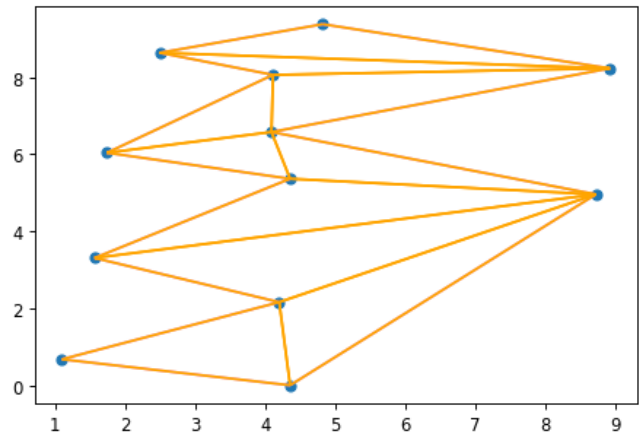
Rys. 6. Wielokąt 5 i jego triangulacja



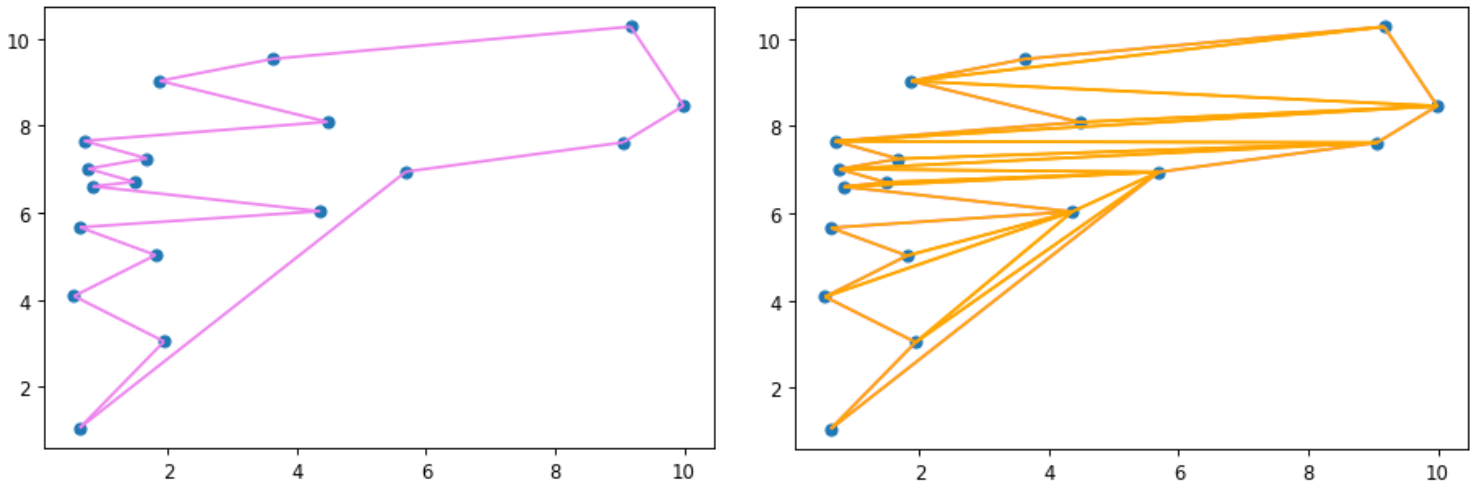


Rys. 7. Wielokąt 6 i jego triangulacja



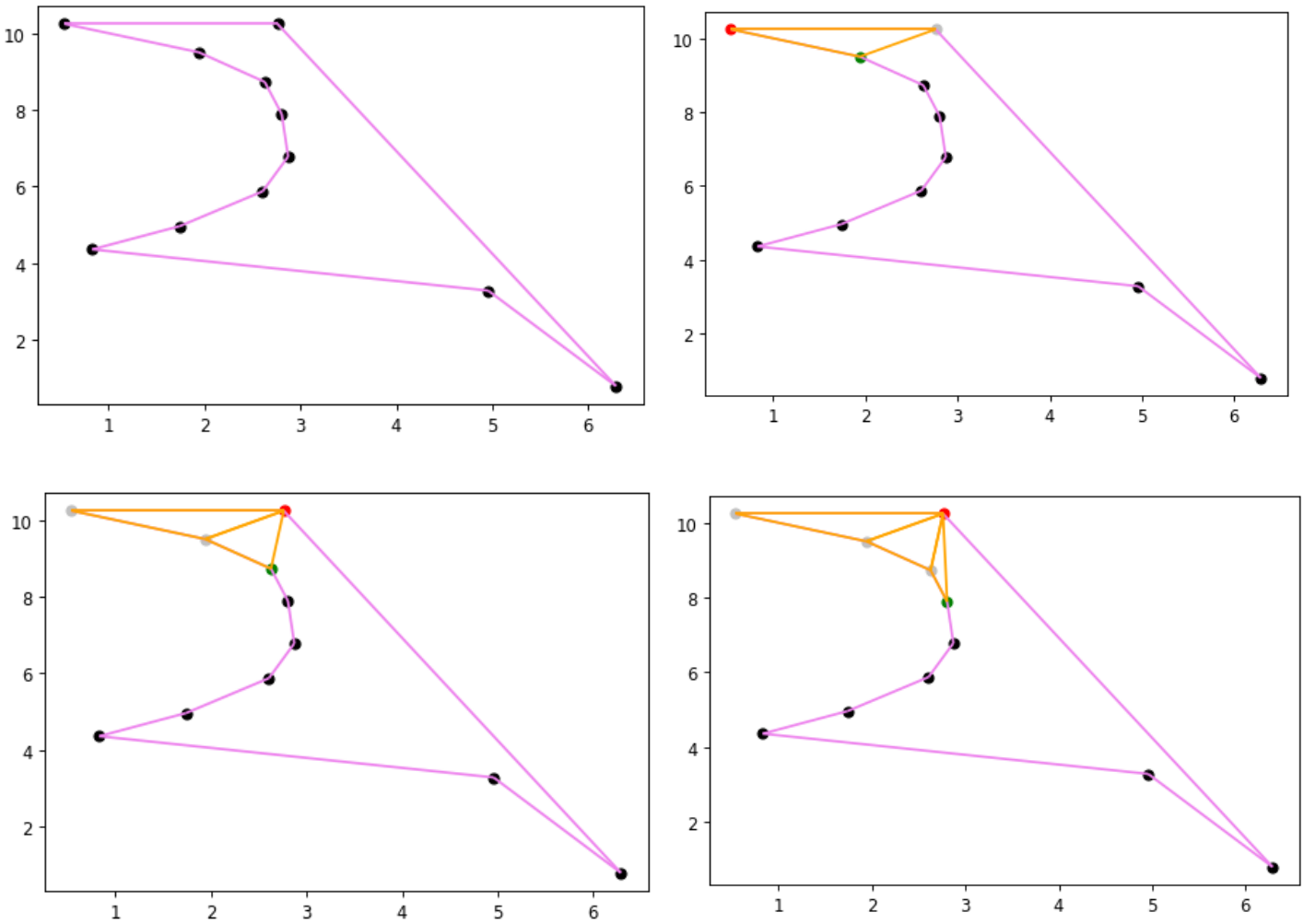


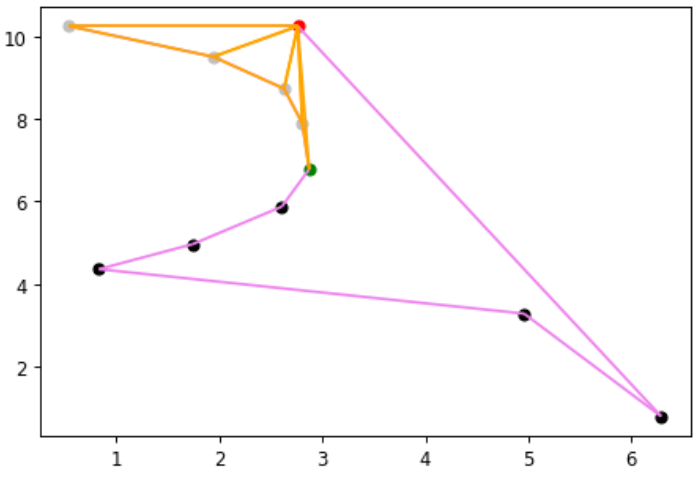
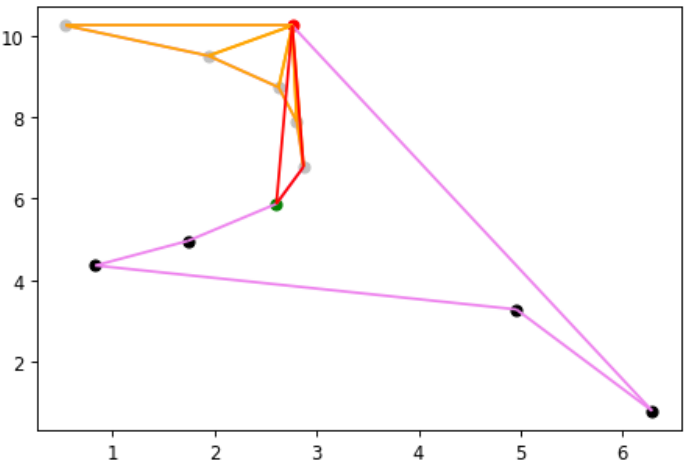
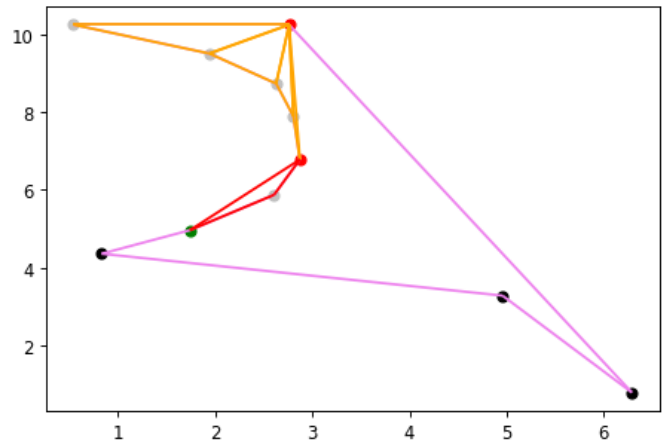
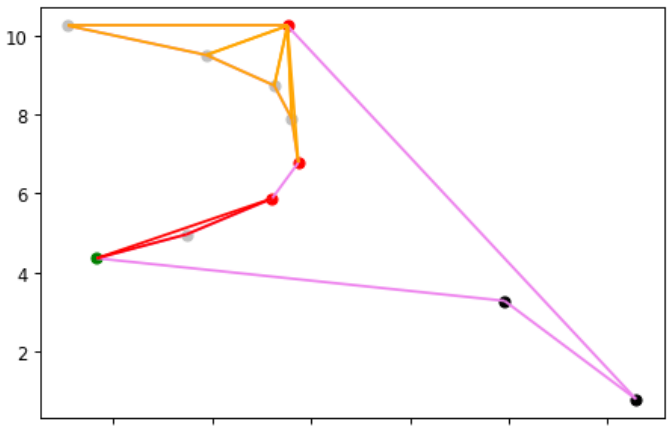
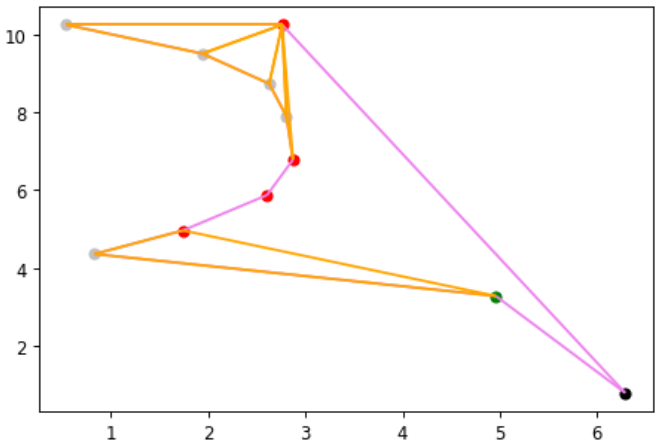
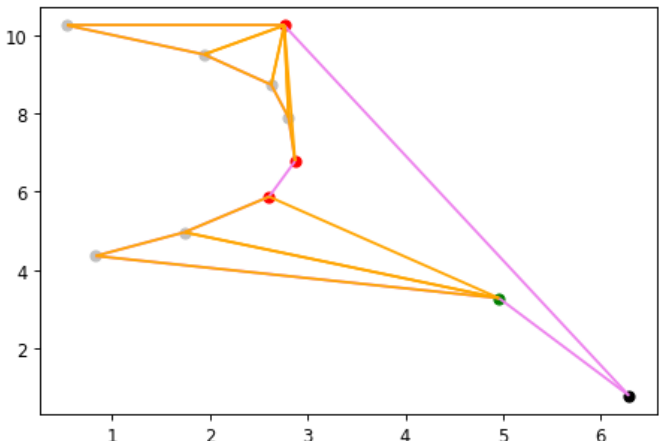
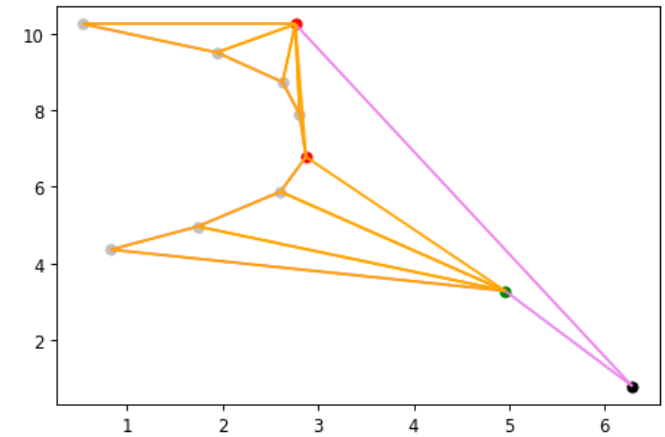
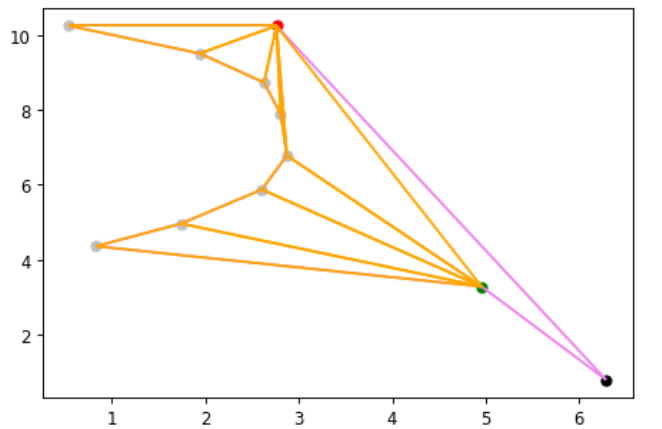
Rys. 8. Wielokąt 7 i jego triangulacja

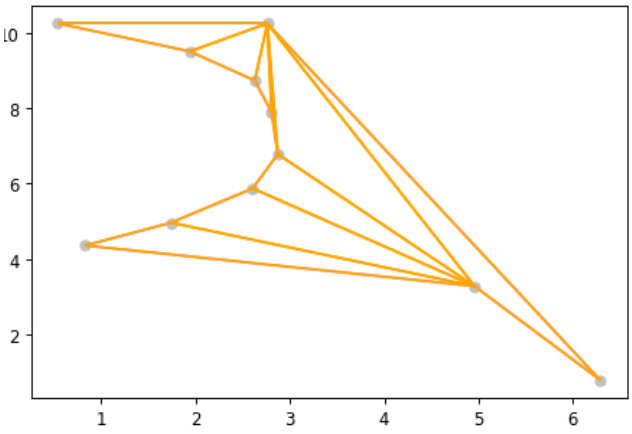


Poniżej przedstawiane są kolejne kroki (z pominięcie zdejmowania/dodawania wierzchołków na stos) algorytmu triangulacji trójkątów dla wielokątu 5.

Rys. 8. Kolejne etapy algorytmu triangulacji dla wielokątu 5.





Schemat kolorystyczny animacji triangulacji:
- czerwony wierzchołek – wierzchołek znajdujący się na stosie
- zielony wierzchołek – wierzchołek aktualnie przetwarzany
- szary wierzchołek – wierzchołek zdjęty ze stosu
- czarny wierzchołek – wierzchołek nieprzetworzony
- żółty trójkąt – element triangulacji
- czerwony trójkąt – kandydat na element triangulacji, który leży poza wielokątem.

8. Wnioski

Zaimplementowane w ćwiczeniu algorytmy triangualyzacji i klasyfikacji wierzchołków zwracają oczekiwane wyniki. Dla wszystkich z proponowanych zbiorów danych (a także dla tych, które nie pojawiły się w sprawozdaniu, a mogą być sprawdzane przy użyciu załączonego kodu jupyter notebook) algorytm triangulacji stworzył właściwą siatkę. Zbiory zostały dobrane tak, aby sprawdzać różne przypadki występujące w algorytmie. Algorytm prawidłowo interpretuje sytuacje gdy badane wierzchołki znajdują się na różnych stronach wielokąta, jak i te, gdzie punkty są po tej samej stronie. Poprawnie działa też określanie czy trójkąt należy do wnętrza wielokąta, co można zaobserwować przy pomocy animacji. Zaimplementowane na cel ćwiczenia struktury danych spełniły swoje zadanie i ułatwiły pracę, dzięki możliwości zarządzania punktami podczas tworzenia wielokąta, a sposób przechowywania siatki trójkątów (w postaci wskaźników na odpowiednie wierzchołki wielokąta) pozwala zaoszczędzić zasoby pamięciowe.