Algorytmy geometryczne

Sprawozdanie z ćwiczeń nr. 3

Wojciech Łoboda

Grupa nr. 1 (środa_13_P_1)

Środowisko oraz narzędzia:

- System operacyjny: Windows 10 x64
- Procesor: Intel Core i7-7700HQ 2.80GHz
- Pamięć RAM: 8 GB
- Środowisko: Jupyter Notebook
- Visual Studio Code

Ćwiczenie wykonano przy pomocy języka python3 z wykorzystaniem następujących bibliotek:

- Numpy
- Matplotlib

Opis ćwiczenia

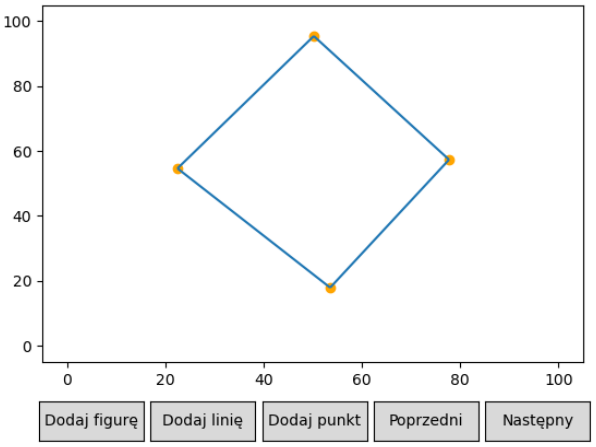
Ćwiczenie polegało na zaimplementowaniu procedury sprawdzającej, czy zadany wielokąt jest y-monotoniczny, zaimplementowaniu algorytmu, który dla zadanego wielokąta będzie wyszukiwał wierzchołki początkowe, końcowe, łączące, dzielące i prawidłowe oraz wizualizował podział punktów kolorując je zgodnie z klasyfikacją. Należało również zaimplementować algorytm triangulacji wielokąta monotonicznego z możliwością wizualizacji kolejnych kroków algorytmu. Algorytmy należało przetestować dla różnych zestawów danych.

Sposób zadawanie figury oraz wykorzystane zestawy danych

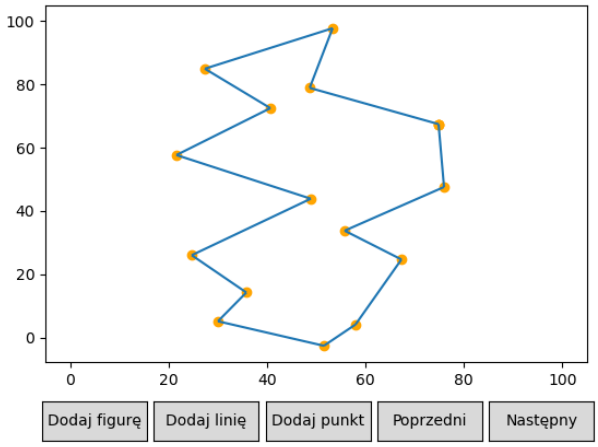
Zadawanie dowolnej figury możliwe jest dzięki zaproponowanemu narzędziu graficznemu wykorzystującemu bibliotekę *matplotlib*, wierzchołki i krawędzie wielokąta zadanego na wykresie w kierunku przeciwnym do ruchu wskazówek zegara (kolejność punktów w liście jest przeciwna do ruchu wskazówek zegara) zwracane są za pomocą metod *get_added_points()* oraz *get_added_figure()* obiektu plot.

Działanie zaimplantowanych algorytmów testowane było na następujących zestawach danych zawierających wielokąty y-monotoniczne oraz nie y-monotoniczne o różnych liczbie punktów i kształcie:

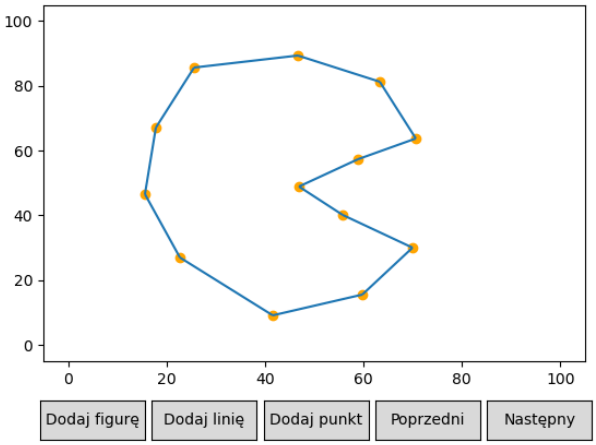
Wykres 1.1 Wielokąt y-monotoniczny (A)



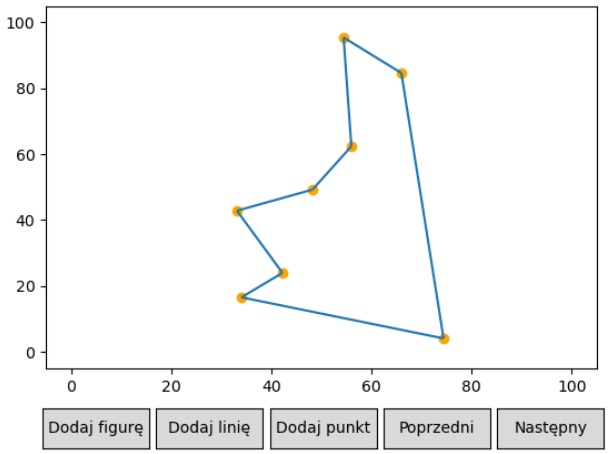
Wykres 1.2 Wielokąt y-monotoniczny (B)



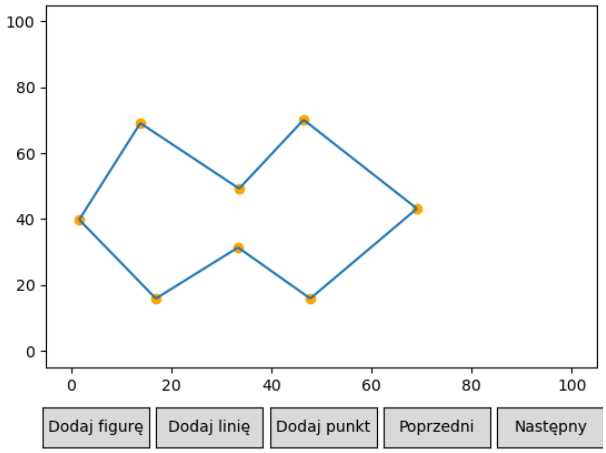
1.3 Wielokąt y-monotoniczny (C)



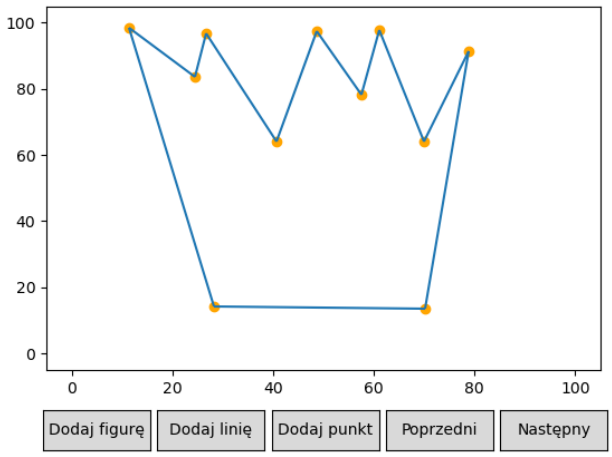
1.4 Wielokąt y-monotoniczny (D)



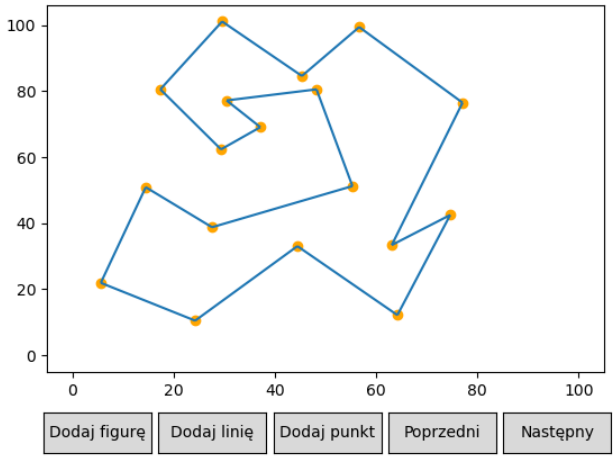
1.5 Wielokąt nie y-monotoniczny (E)



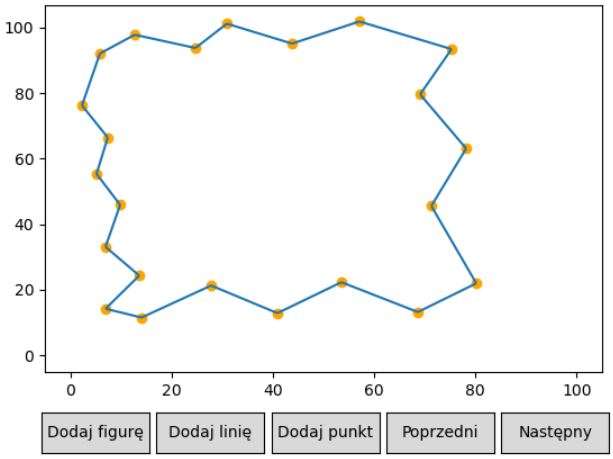
1.6 Wielokąt nie y-monotoniczny (F)



1.7 Wielokąt nie y-monotoniczny (G)



1.8 Wielokąt nie y-monotoniczny (H)



Algorytm klasyfikacji punktów wielokąta

Klasyfikacja punktów polega na przejściu po wszystkich wierzchołkach wielokąta, zaczynając od dowolnego wierzchołka i idąc w kolejności przeciwnej do ruchu wskazówek zegara. Dla każdego z wierzchołków P_t, jeżeli:

- P_t leży powyżej swoich sąsiadów (współrzędna y dla P_t jest większa od współrzędnych y jego sąsiadów) oraz kąt wewnętrzny wielokąta przy wierzchołku P_t jest mniejszy niż π ($det(P_{t-1}, P_t, P_{t+1}) > 0$):
  P_t to punkt początkowy (kolor: zielony)

- P_t leży powyżej swoich sąsiadów (współrzędna y dla P_t jest większa od współrzędnych y jego sąsiadów) oraz kąt wewnętrzny wielokąta przy wierzchołku P_t jest większy niż π ($det(P_{t-1}, P_t, P_{t+1}) < 0$):
  P_t to punkt dzielący (kolor: niebieski)

- P_t leży poniżej swoich sąsiadów (współrzędna y dla P_t jest mniejsza od współrzędnych y jego sąsiadów) oraz kąt wewnętrzny wielokąta przy wierzchołku P_t jest mniejszy niż π ($det(P_{t-1}, P_t, P_{t+1}) > 0$):
  P_t to punkt końcowy (kolor: czerwony)

- P_t leży poniżej swoich sąsiadów (współrzędna y dla P_t jest mniejsza od współrzędnych y jego sąsiadów) oraz kąt wewnętrzny wielokąta przy wierzchołku P_t jest większy niż π ($det(P_{t-1}, P_t, P_{t+1}) < 0$):
  P_t to punkt łączący (kolor: granatowy)

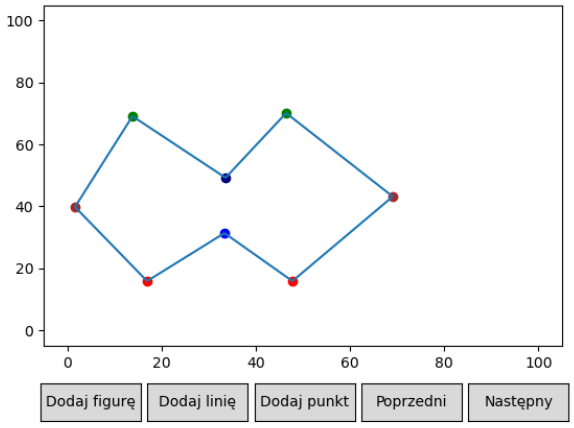
- Jeden z sąsiadów leży powyżej P_t a drugi leży poniżej. P_t to punkt prawidłowy (kolor: brązowy)

Gdzie: $$\det(a,b,c) = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}$$

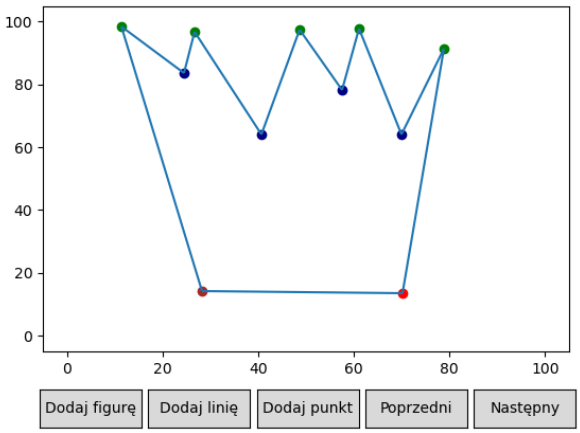
Algorytm jest implementowany przez funkcje *classifyVertices()* która przyjmująca listę wierzchołków wielokąta w kolejności przeciwnej do ruchy wskazówek zegara jako argument i zwracająca listę zawierającą listy zawierające kolejno punkty początkowe, dzielące, końcowe, łączące i prawidłowe.

Wizualizacja klasyfikacji jest implementowana przez funkcje *visualize_points_classification()*.

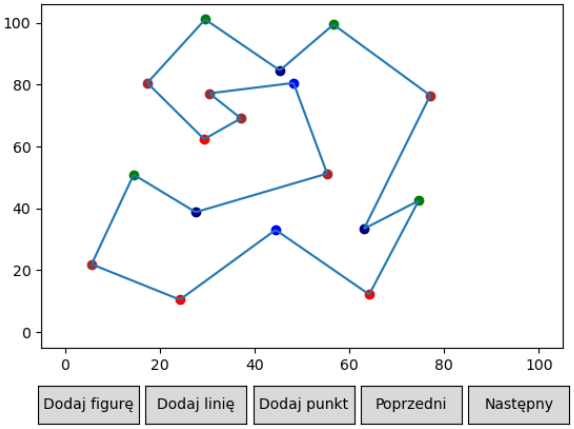
Wykres 2.1 Klasyfikacji wierzchołków wielokąta E



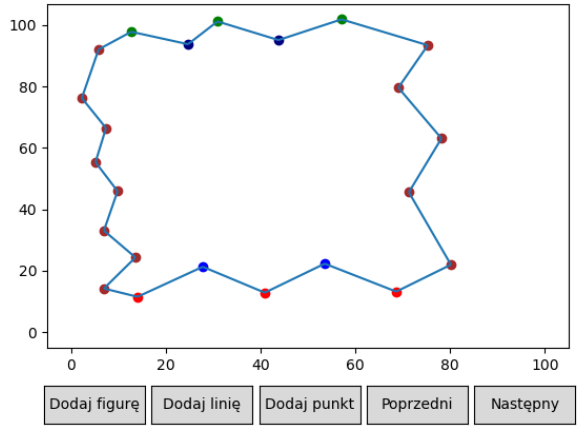
Wykres 2.2 Klasyfikacji wierzchołków wielokąta F



Wykres 2.3 Klasyfikacji wierzchołków wielokąta G



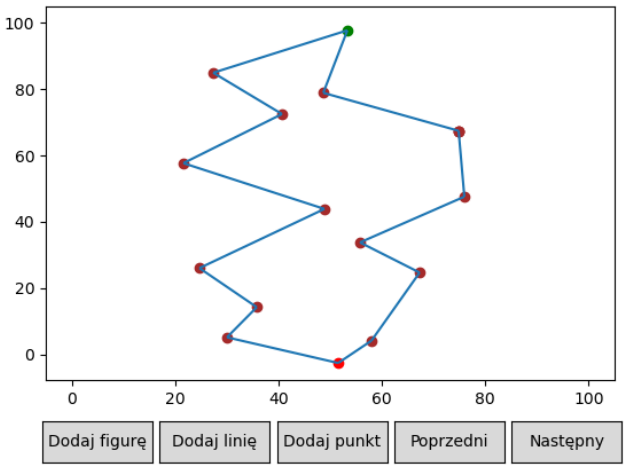
Wykres 2.4 Klasyfikacji wierzchołków wielokąta H



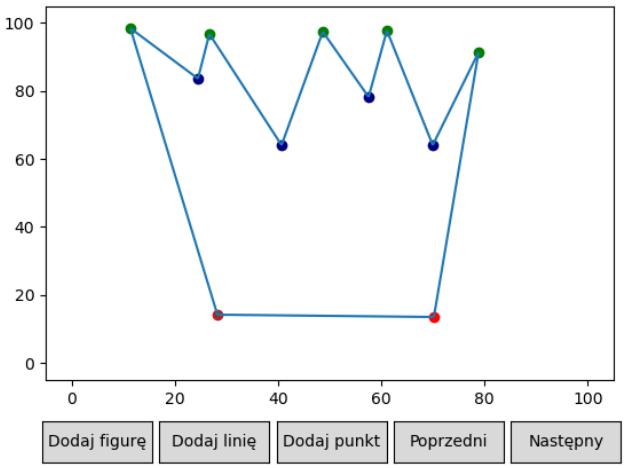
Algorytm sprawdzający czy zadany wielokąt jest y-monotoniczny

Algorytm sprawdzający czy zadany wielokąt jest y-monotoniczny polega na użyciu algorytmu do klasyfikacji wszystkich wierzchołków opisanego powyżej. Jeżeli istnieje wierzchołek wielokąta zakwalifikowany jako wierzchołek łączący lub dzielący to wielokąt nie jest y-monotoniczny w przeciwnym wypadku wielokąt jest y-monotoniczny. Poprawność algorytmu wynika z twierdzenia: Wielokąt jest y-monotoniczny gdy nie ma wierzchołków dzielących i łączących. (Wykład). Algorytm implementuje funkcja *is_y_monotone()*. Przyjmująca listę wierzchołków wielokąta w kolejności przeciwnej do ruchy wskazówek zegara jako argument i zwracająca True jeżeli wielokąt jest y-monotoniczny i False w przeciwnym przypadku

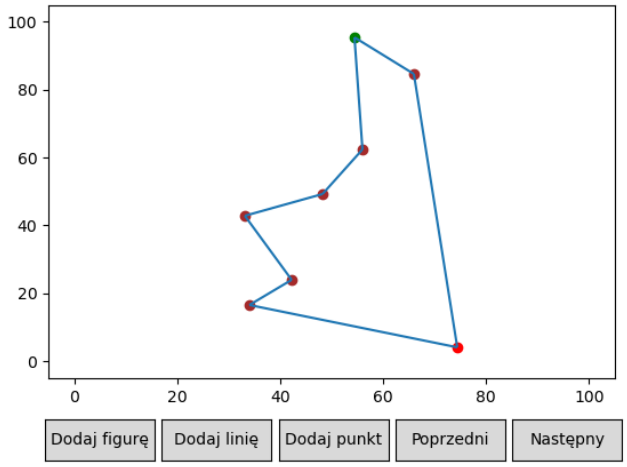
Wykres 3.1 Wielokąt B
Wynik algorytmu: True



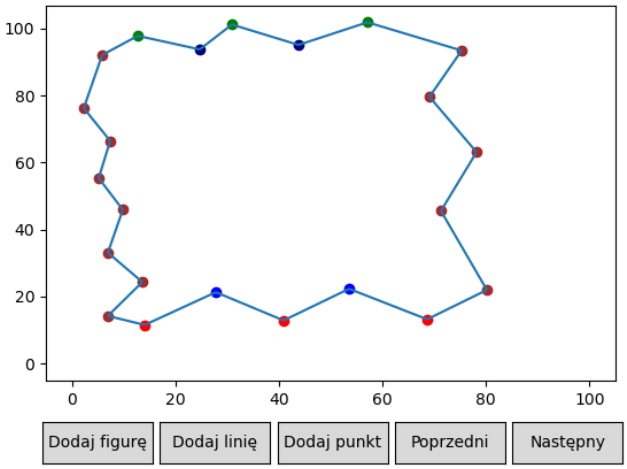
Wykres 3.2 Wielokąt F
Wynik algorytmu: False



Wykres 3.3 Wielokąt D
Wynik algorytmu: True



Wykres 3.4 Wielokąt H
Wynik algorytmu: False



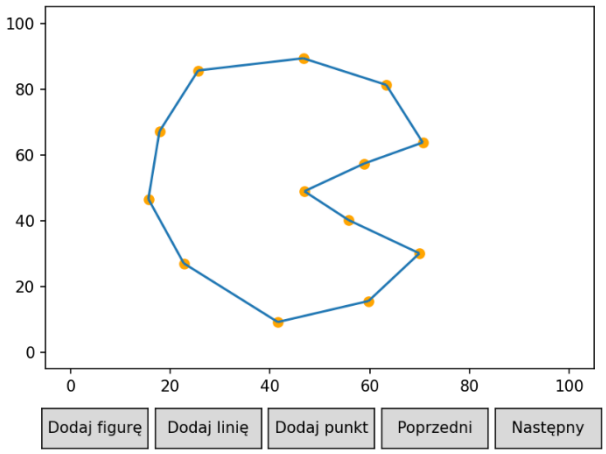
Algorytm triangulacji wielokąta y-monotonicznego

Algorytm triangulacji wielokąta monotonicznego:

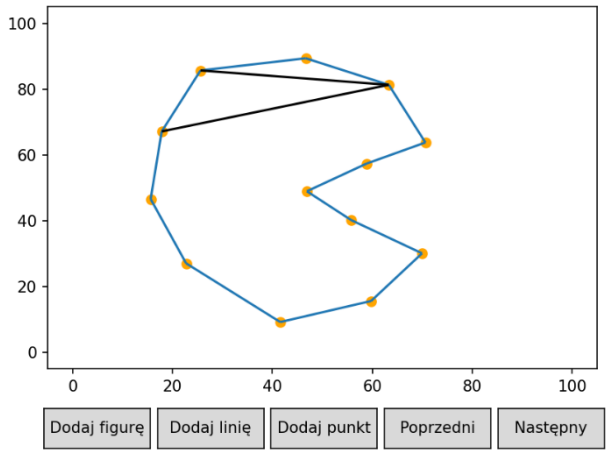
- Określamy lewy i prawy łańcuch wielokąta względem kierunku monotoniczności.
- Porządkujemy wierzchołki wzdłuż kierunku monotoniczności.
- Wkładamy dwa pierwsze wierzchołki na stos.
- Jeśli kolejny wierzchołek należy do innego łańcucha niż wierzchołek stanowiący szczyt stosu, to możemy go połączyć ze wszystkimi wierzchołkami na stosie. Na stosie zostają dwa wierzchołki, które były „zamiatane" ostatnie.
- Jeśli kolejny wierzchołek należy do tego samego łańcucha co wierzchołek ze szczytu stosu, to analizujemy kolejne trójkąty, jakie tworzy dany wierzchołek z wierzchołkami zdejmowanymi ze stosu.
    - Jeśli trójkąt należy do wielokąta, to usuwamy wierzchołek ze szczytu stosu
    - w przeciwnym przypadku umieszczamy badane wierzchołki na stosie.

Wizualizacja wybranych kroków dla wielokąta C:

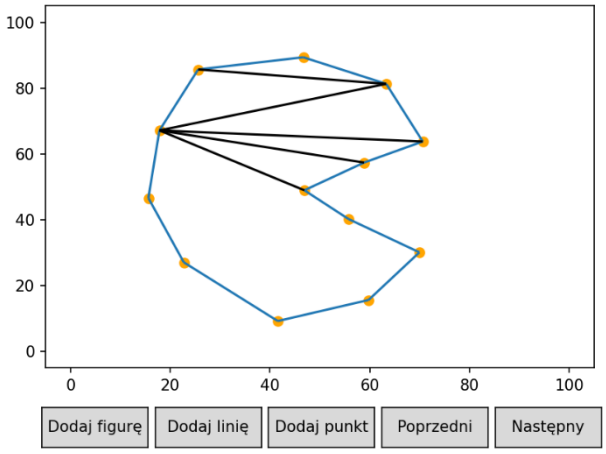
Wykres 4.1 Krok 0



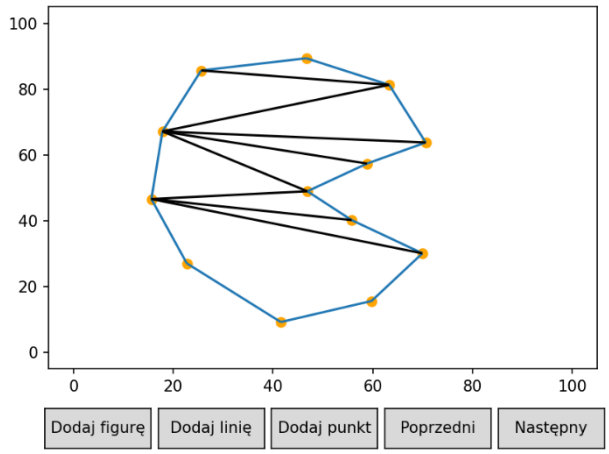
Wykres 4.2 Krok 2



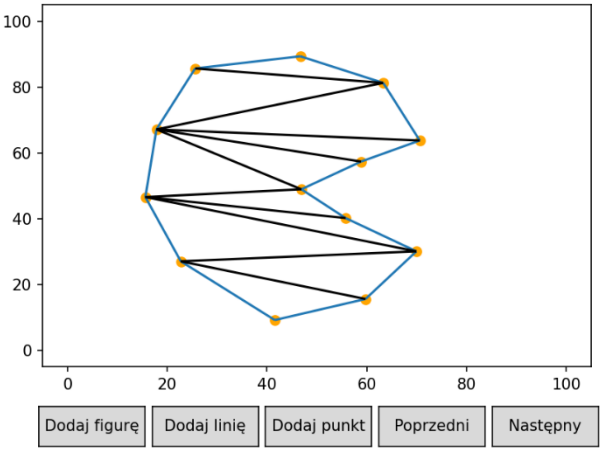
Wykres 4.3 Krok 5



Wykres 4.4 Krok 9

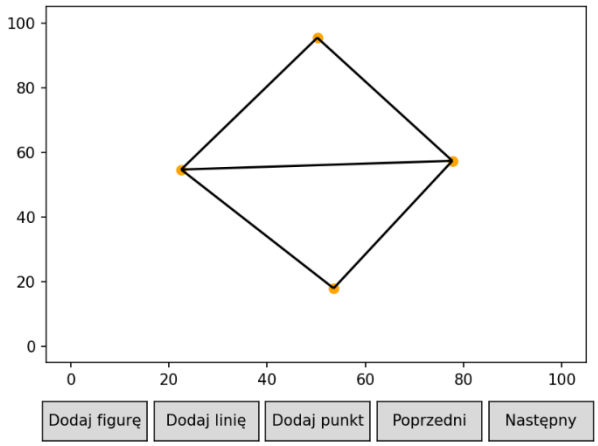


Wykres 4.5 Krok 12

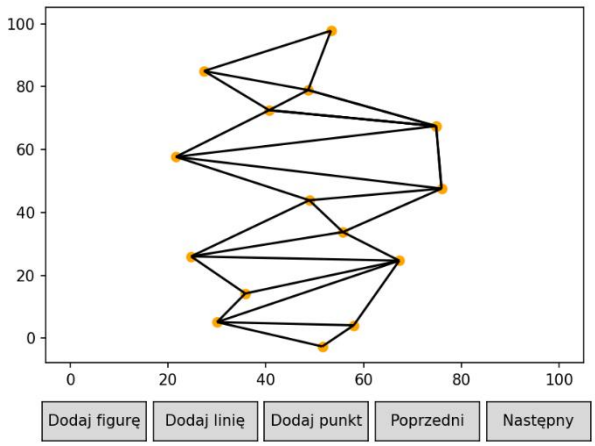


Wizualizacja wynikowej triangulacji dla wszystkich zbiorów y-monotonicznych:

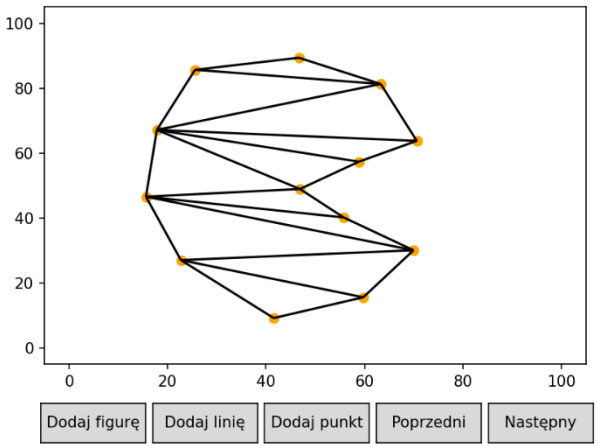
Wykres 5.1 Triangulacja wielokąta A



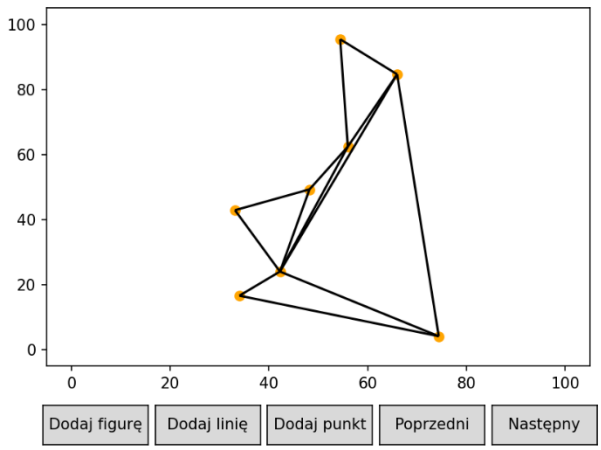
Wykres 5.2 Triangulacja wielokąta B



Wykres 5.3 Triangulacja wielokąta C



Wykres 5.4 Triangulacja wielokąta D



Algorytm implementowany jest przez funkcje *triangulate_monotone_polygon*(), która jako argument przyjmuje listę wierzchołków wielokąta, reprezentowanych jako para współrzędnych. Wierzchołki powinny być uporządkowanych przeciwne do ruchu wskazówek zegara. Funkcja zwraca listę *diagonals* która zawiera w sobie pary indeksów z zadanej listy wierzchołków które są łączone przez dodawane przekątne.

Wykorzystane struktury danych

Do reprezentowania wielokąta wykorzystuje link-listę dwukierunkową. Każdy element listy zawiera współrzędne wierzchołka wielokąta, indeks identyfikujący wierzchołek oraz wskaźniki:
next – wskaźnik na następny wierzchołek w porządku przeciwnym do ruchu wskazówek zegara.
prev – wskaźnik na poprzedni wierzchołek w porządku przeciwnym do ruchu wskazówek zegara.
Wykorzystanie takiej struktury zapewnia szybki dostęp do sąsiadów każdego z wierzchołków, wygodny sposób na weryfikacje czy przekątna którą próbujemy dodać w trakcie triangulacji nie należy do zbioru krawędzi oraz łatwą iteracje po wierzchołkach wielokąta w odpowiedniej kolejności. Podwójnie łączona lista upraszcza w ten sposób implementacje algorytmu do klasyfikacji wierzchołków oraz do triangulacji wielokąta.

Triangulacja wielokąta jest reprezentowana jako zbiór par indeksów (z listy w której podane są wierzchołki) punktów które są łączone przez dodaną przekątną, oddzielając w ten sposób informacje geometryczną od topologicznej (w implementacji lista *diagonals*) oraz ułatwiając prace nad danymi.

Wnioski

Zaimplementowane algorytmy działały poprawne dla wszystkich wykorzystanych zestawów danych na podstawie czego można wnioskować że algorytmy zostały odpowiednio zaimplementowane.