Algorytmy Geometryczne
Labolatorium nr 3

Szymon Szarek

27 listopada 2023

1 Opis ćwiczenia

Celem ćwiczenia było wyznaczenie triangulację wielokąta monotonicznego, sprawdzenie czy wielokąt jest y-monotoniczny oraz klasyfikacja wierzchołków wielokąta. Należało również umożliwić zadawanie wielokąta w sposób graficzny za pomocą myszki.

1.1 Definicja triangulacji

Niech K określa sympleks w danej przestrzeni.

T nazywamy triangulacją Ω, jeżeli spełnione są następujące warunki:

- $\Omega = \bigcup_{K \in T} K$
- Każdy element K ma niepuste wnętrze,
- Wnętrza róznych elementów są rozłączne,
- Przecięcie dwóch różnych elementów jest:
  a albo puste
  b albo zredukowane do jednego punktu,
  który jest wspólnym wierzchołkiem tych elementów
  c albo zredukowane do jednej krawędzi
  która jest wspólną krawędzią tych elementów
  d albo zredukowane do jednej ściany (3D)
  która jest wspólną ścianą tych elementów

1.2 Definicja wielokąta monotonicznego

1.Wielokąt prosty nazywamy **ściśle monotonicznym** względem prostej l (wyznaczającej **kierunek monotoniczności**), gdy jego brzeg można podzielić na dwa spójne łańcuchy takie, że dowolna prosta l' prostopadła do l przecina każdy z łańcuchów w co najwyżej jednym punkcie.

2.Wielokąt jest **monotoniczny** względem prostej l , gdy przecięcie dowolnej prostej l' prostopadłej do l z dowolnym łańcuchem jest spójne.
Przecięcie wielokąta z l' jest spójne – jest odcinkiem, punktem lub jest puste.

1.3 Klasyfikacja wierzchołków wielokąta

Rodzaje wierzchołków w wielokącie:

- początkowy, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny $< \pi$,
- końcowy, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny $< \pi$,

- łączący, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny $> \pi$,
- dzielący, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny $> \pi$,
- prawidłowy, w pozostałych przypadkach

2 Środowisko pracy

Ćwiczenie zostało napisane 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*.

Specyfikacja sprzętu:

- System Windows 11 x64
- Procesor AMD Ryzen PRO 5650U
- Pamięć RAM 16GB
- Python 3.9

3 Realizacja ćwiczenia

W ramach ćwiczenia zaimplementowane zostały wcześniej wspomniane algorytmy.

3.1 Algorytm sprawdzający y-monotoniczność

Idea algorytmu jest bardzo prosta i oparta na jednej z właściwości wielokąta y-monotonicznego tj. idąc z najwyższego wierzchołka do najniższego wzdłuż lewego (lub prawego) łańcucha zawsze poruszamy się w dół lub poziomo, nigdy w górę. Oznacza to, że wszystkie punkty z danego łańcucha są monotoniczne względem współrzędnej y. W algorytmie wykorzystałem tą zależność i skonstruowałem go następująco:

1. Niech polygon to będzie zbiór punktów wielokąta w kolejności przeciwnej do ruchu wskazówek zegara,
   $|polygon| = n$
2. Wyznacz y_{max} oraz index y_{max}=max_{idx}
3. Wyznacz y_{min} oraz index y_{min}=min_{idx}
4. $i = min_{idx}$
5. **while** $i \neq max_{idx}$:
6. jeżeli i+1-ty punkt leży niżej niż i-ty: zwróć fałsz
7. i=(i+1) mod n
8. **while** $i \neq min_{idx}$:
9. jeżeli i+1-ty punkt leży wyżej niż i-ty: zwróć fałsz
10. i=(i+1) mod n
11. zwróć prawdę

3.2 Algorytm klasyfikacji punktów

Algorytm został zaimplementowany zgodnie z definicją powyżej. Podczas iterowania po wierzchołkach sprawdzane były wierzchołki sąsiadujące oraz kąt wewnętrzny. Do obliczenia, czy kąt wewnętrzny jest mniejszy czy większy od π została użyta funkcja orientation sprawdzająca po której stronie prostej znajduje się dany punkt.
Algorytm kolorował również wierzchołki do dalszej wizualizacji:

- początkowy - zielony,
- końcowy - czerwony,
- łączący - ciemny niebieski,
- dzielący - jasny niebieski.
- prawidłowy - brązowy.

3.3 Algorytm triangulacji wielokąta y-monotonicznego

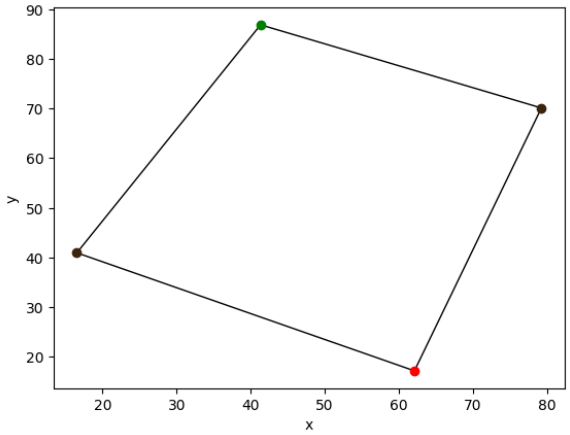
Opis algorytmu:

- 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 dodajemy przekątną i usuwamy odpowiedni wierzchołek ze szczytu stosu
  - w przeciwnym przypadku umieszczamy badane wierzchołki na stosie.

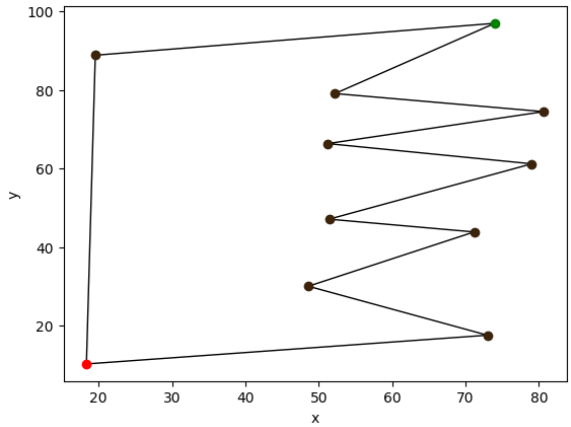
Do przechowywania wielokąta została użyta lista krotek. Jest to bardzo naturalna reprezentacja listy punktów, ponieważ stan krotki jest niezmienny i my nie będziemy zmieniać współrzędnych punktów. W celu sprawdzenia, czy dany punkt należy do prawego lub lewego łańcucha wykorzystana została struktura zbioru (jeden łańcuch to jeden zbiór). Jest ona wydajna do tego celu, ponieważ sprawdzenie czy element należy do zbioru odbywa się w czasie stałym ($O(1)$) dzięki metodzie hashującej w tej strukturze. Zbiór również został wykorzystany do przechowywania boków wielokąta. Podczas algorytmu zdarzy się, że zklasyfikuje on przekątną do dodania, która już istnieje w wielokącie jako jego bok. Z tego powodu musimy również sprawdzić, czy dana przekątna nie jest już bokiem wielokąta (nie należy do zbioru). Sama triangulacja może być przechowywana na dwa sposoby: lista krotek współrzędnych odcinków lub lista krotek indeksów współrzędnych w liście początkowej. W ramach ćwiczenia zostały zaimplementowane oba sposoby. Przechowywanie samych indeksów punków pozwala na zminimalizowanie redundancji danych, a zatem pozwala na zużycie dużo mniejszej ilości pamięci. Implementacja przechowująca współrzędne zamiast indeksów została użyta do wygodniejszej wizualizacji wielokątów.

4 Wyniki działania algorytmów

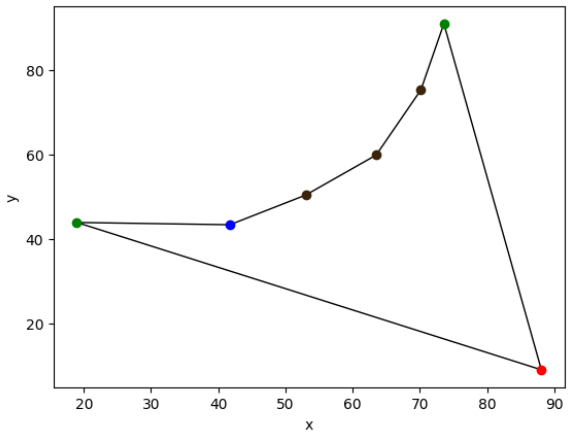
4.1 Klasyfikacja wierzchołków wielokąta



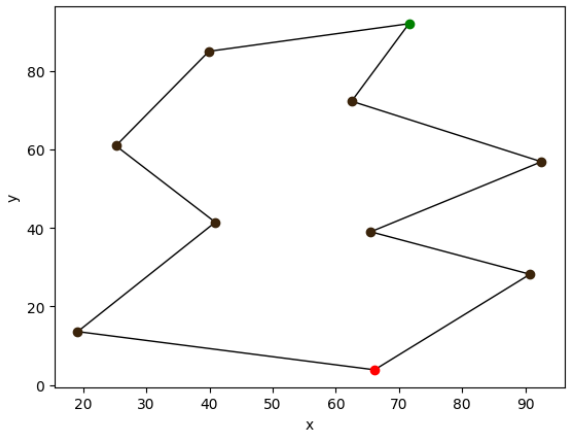
Wykres 1.



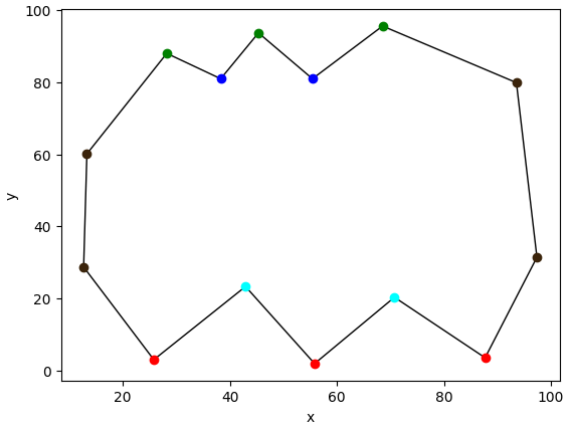
Wykres 2.



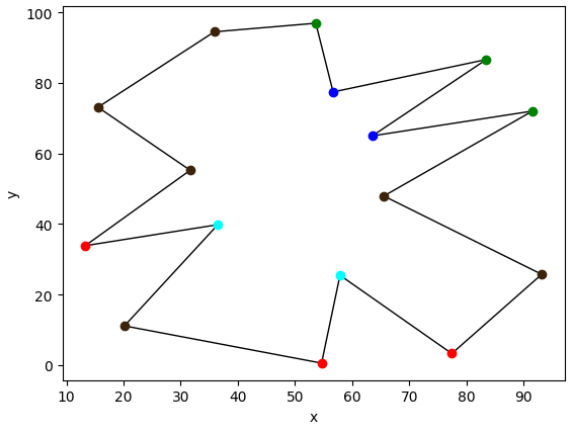
Wykres 3.



Wykres 4.

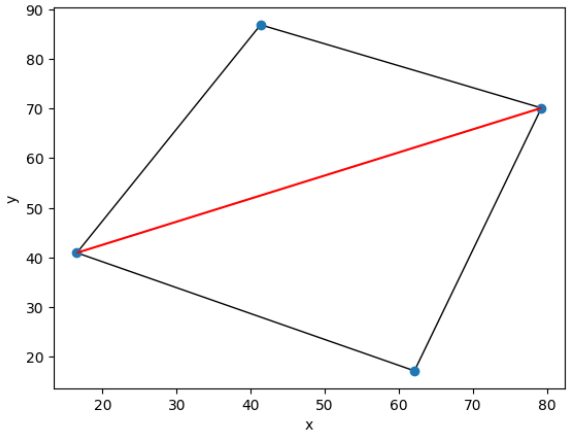


Wykres 5.

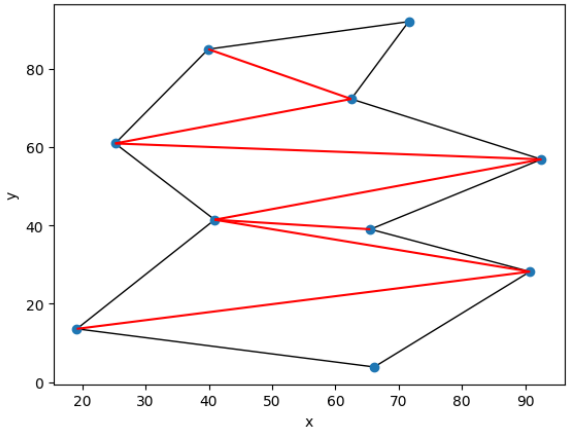


Wykres 6.

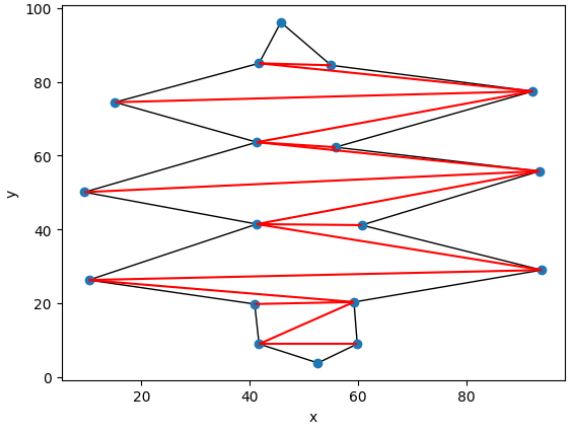
4.2 Triangulacja wielokąta y-monotonicznego



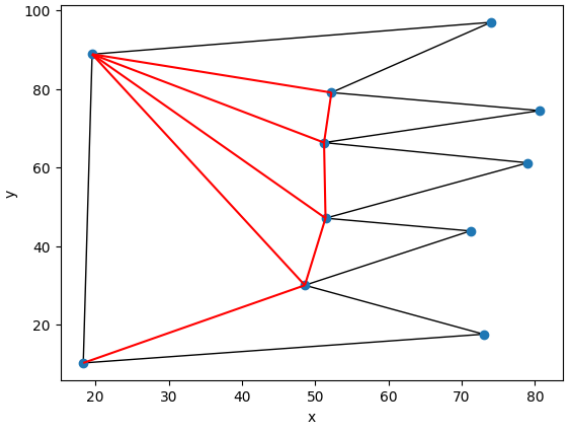
Wykres 7.



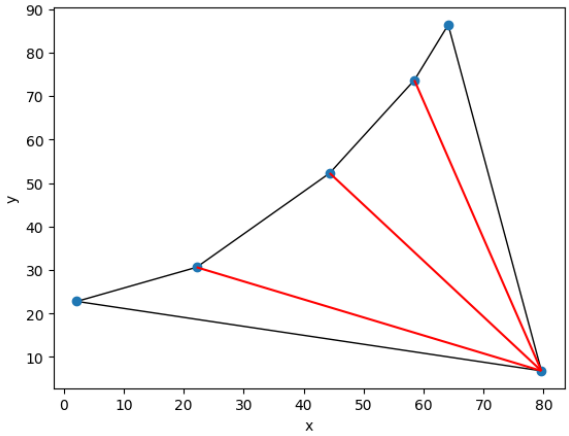
Wykres 8.



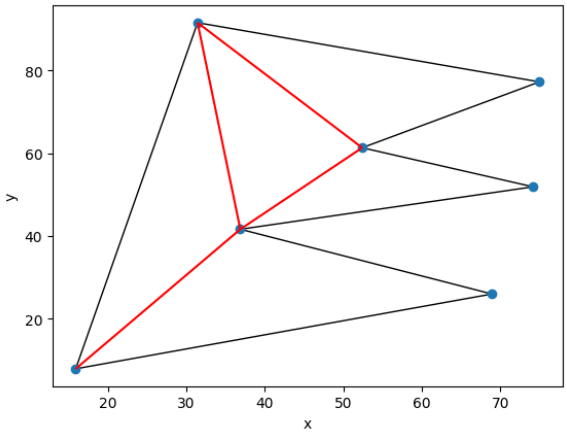
Wykres 9.



Wykres 10.



Wykres 11.

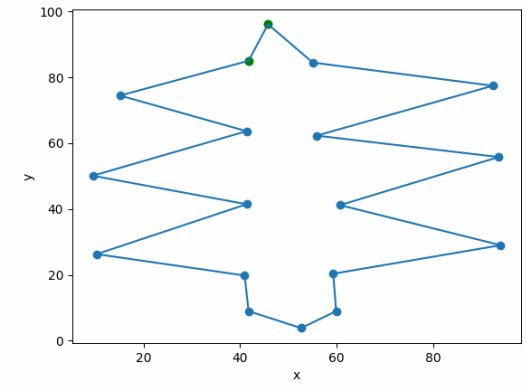


Wykres 12.

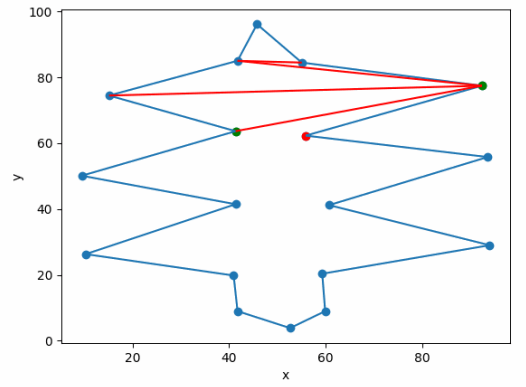
5 Wizualizacja działania algorytmu triangulacji wielokąta y-monotonicznego

Legenda kolorów:

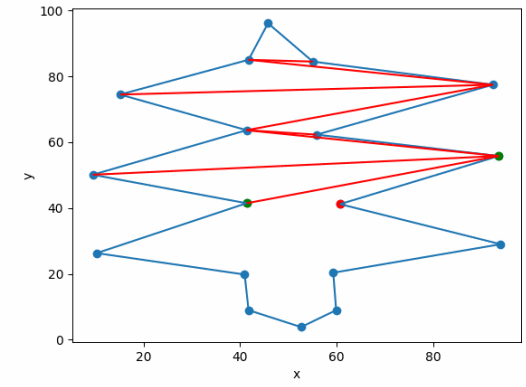
- Zielony - punkt na stosie
- Czerwony - punkt aktualnie rozpatrywany / przekątna triangulacji



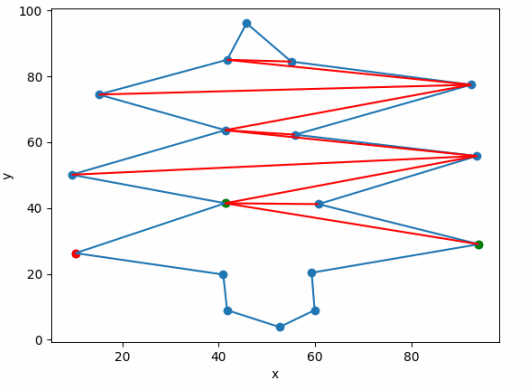
Wykres 13. Wielokąt nr 1



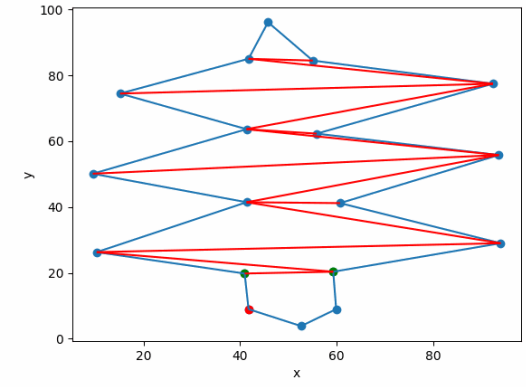
Wykres 14. Wielokąt nr 1



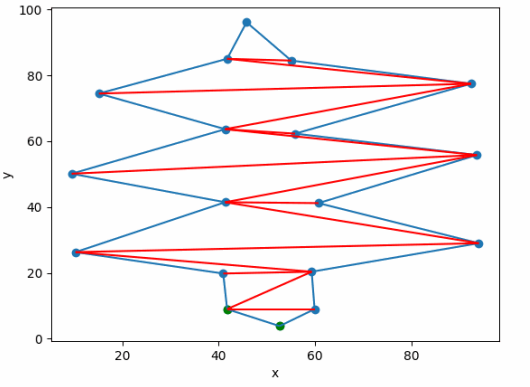
Wykres 15. Wielokąt nr 1



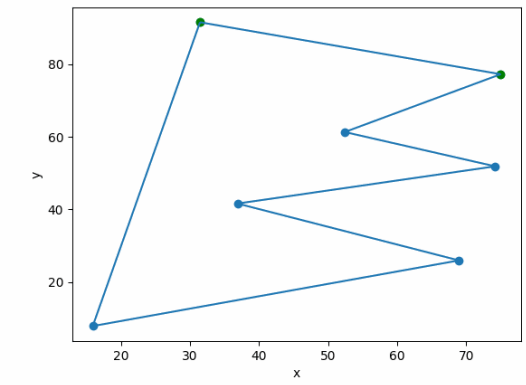
Wykres 16. Wielokąt nr 1



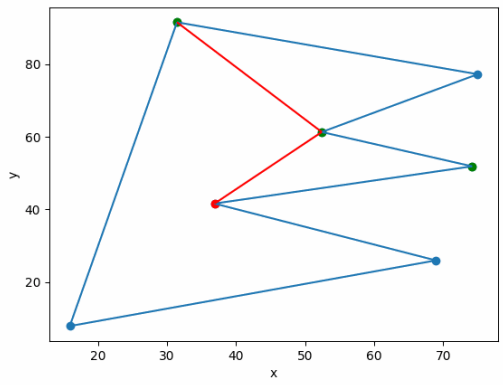
Wykres 17. Wielokąt nr 1



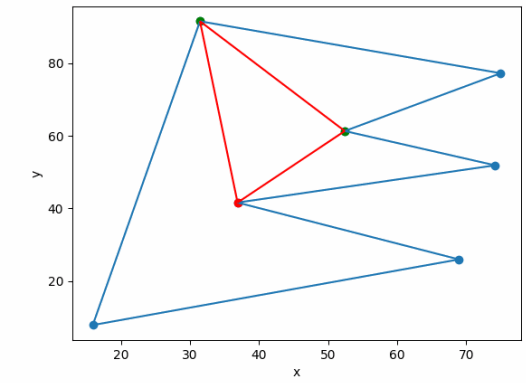
Wykres 18. Wielokąt nr 1



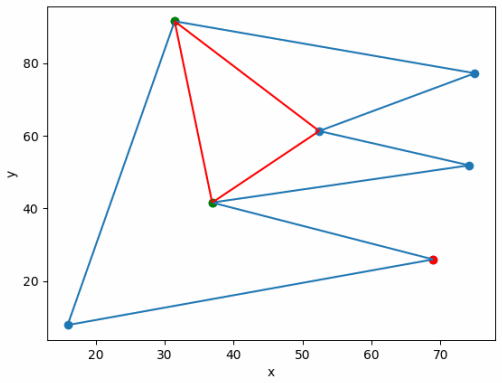
Wykres 19. Wielokąt nr 2



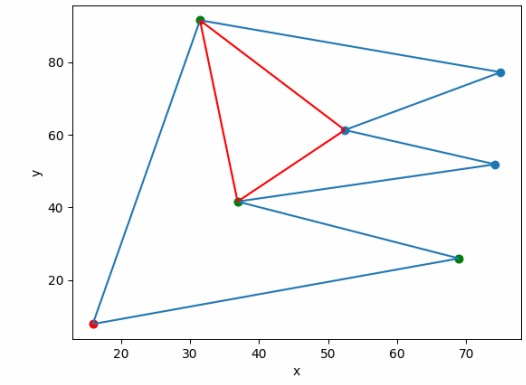
Wykres 20. Wielokąt nr 2



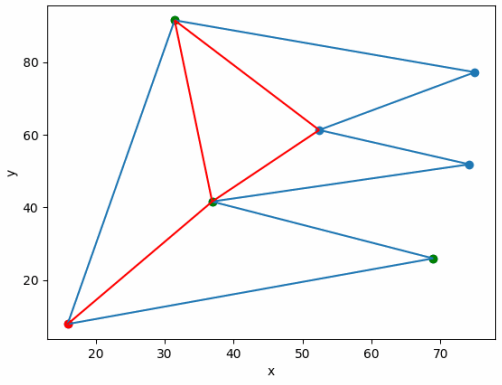
Wykres 21. Wielokąt nr 2



Wykres 22. Wielokąt nr 2



Wykres 23. Wielokąt nr 2



Wykres 24. Wielokąt nr 2

6 Wnioski

Algorytmy po przetestowaniu na powyższych wielokątach działają poprawnie. Biorąc pod uwagę, że są to w większości przypadki skrajne, można stwierdzić, że same algorytmy zostały odpowiednio zaimplementowane. Wszystkie algorytmy działają w czasie liniowym $O(n)$, więc są algorytmami szybkimi i deterministycznymi.