Algorytmy geometryczne
laboratorium 3 - sprawozdanie

Radosław Rolka
Informatyka WI, II rok
Gr 6, tydz. A czwartek 13:00

Spis treści

1 Wstęp

1.1 Cel ćwiczenia

Ćwiczenie wprowadzające w zagadnienia triangulacji – implementacja algorytmów sprawdzających, czy zadany wielokąt jest y-monotoniczny, klasyfikacja wierzchołków w zależności od ich położenia oraz algorytmu do triangulacji wielokątów. Ponadto przeprowadzenie testów, wizualizacja i opracowanie wyników.

1.2 Program ćwiczenia

- Zaimplementuj procedurę sprawdzającą, czy podany wielokąt jest y-monotoniczny.
- Zaimplementuj algorytm, który dla zadanego wielokąta bedzie wyszukiwał wierzchołki początkowe, końcowe, łączące, dzielące i prawidłowe. Wierzchołki mają zostać odpowiednio pokolorowane zgodnie z klasyfikacją.
- Zaimplementuj procedurę triangulacji wielokąta y-monotonicznego

2 Wykorzystane narzędzia

2.1 Środowisko

Ćwiczenie zostało wykonane w Jupyter Notebook wykorzystując język programowania Python oraz dodatkowe biblioteki, które zostały zawarte w projekcie dostarczonym na zajęciach.

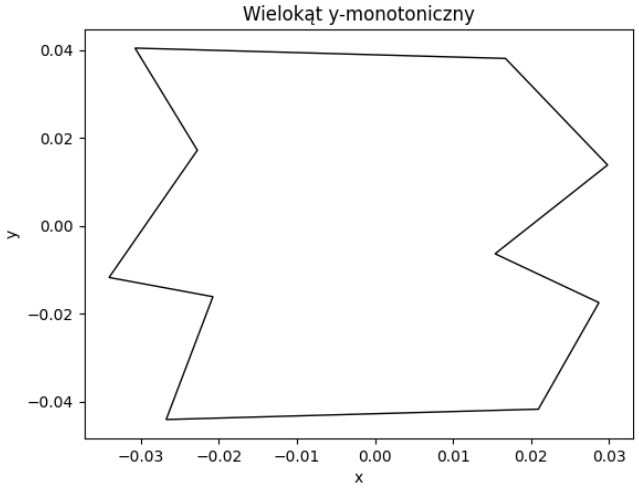
2.2 Sprzęt

Do wykonania został wykorzystany procesor Intel(R) Core(TM) I5-10300H 2.50GHz oraz system operacyjny Microsoft Windows 10 64bit ver 22H2.

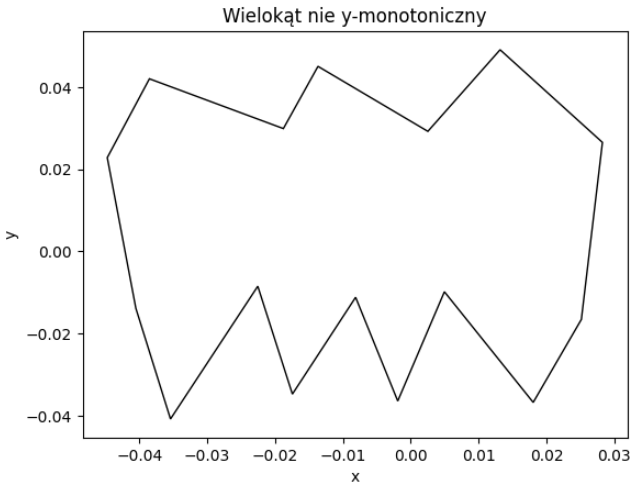
3 Przebieg ćwiczeń

3.1 Y-monotoniczność

Wielokąt jest y-monotoniczny, wtedy gdy żaden spośród jego wierzchołków nie jest wierzchołkiem łączącym, ani dzielącym. Zaimplementowany algorytm sprawdzający Y-monotoniczność zadanego wielokątu bazuje na powyższej zasadzie, mianowicie przechodzi po wszystkich wierzchołkach przeciwnie do ruchu wskazówek zegara i dla każdej trójki kolejnych punktów sprawdza, czy aktualnie przetwarznego wierzchołka nie można zakwalifikować jako łączącego, czy dzielącego. Jeśli algorytm wykryje takowy punkt, zwraca wartość False. Jednak, gdy pętla zostanie nieprzerwana, wtedy zostanie zwrócona wartość True, co oznacza zakwalifikowanie wielokąta jako y-monotonicznego.



Rysunek 1: Wielokąt y-monotoniczny
(Poprawna klasyfikacja)

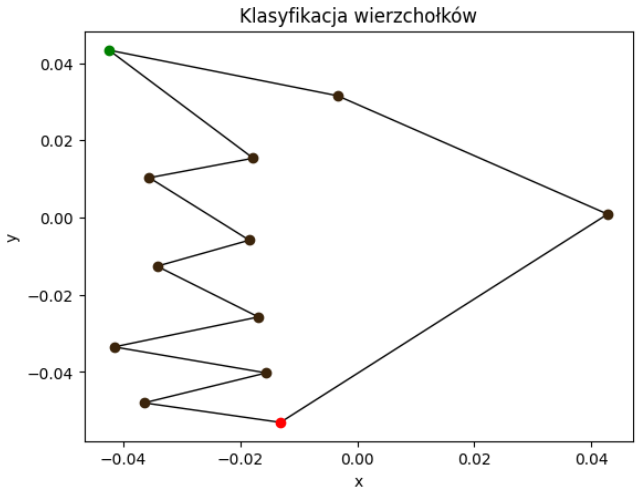


Rysunek 2: Wielokąt nie y-monotoniczny
(Poprawna klasyfikacja)

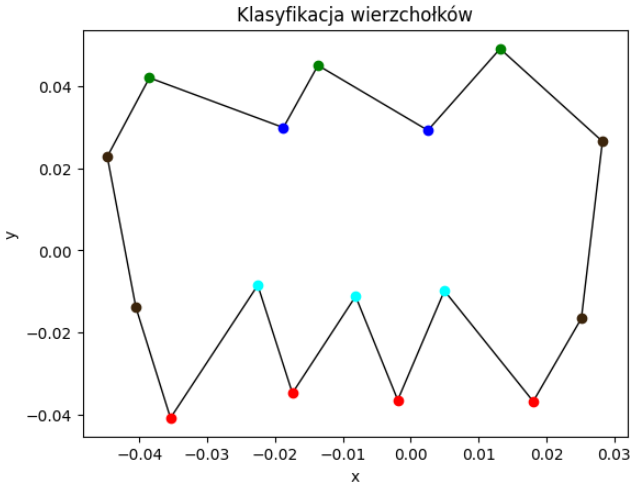
3.2 Klasyfikacja wierzchołków

Algorytm przetwarza każdy wierzchołek i dopasowywuje odpowiednią klasyfikację na podstawie jego położenia oraz sąsiadujących go wierzchołków za pomocą funkcji *orient*.

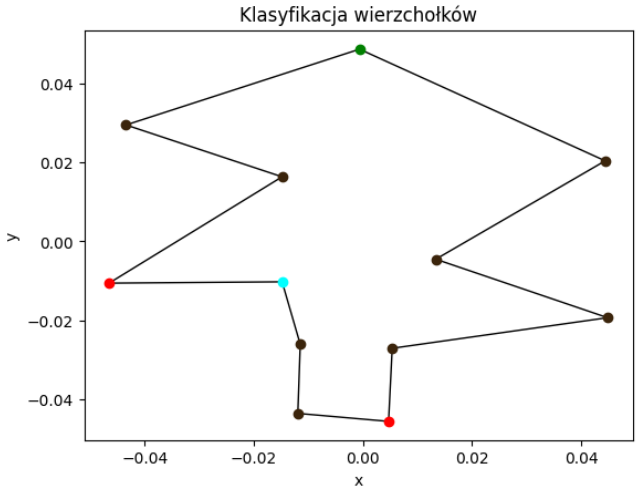
- Wierzchołek **początkowy**: obaj jego sąsiedzi leżą **poniżej**, a kąt wewnętrzny ma **mniej** niż 180 stopni.
- Wierzchołek **końcowy**: obaj jego sąsiedzi leżą **powyżej**, a kąt wewnętrzny ma **mniej** niż 180 stopni.
- Wierzchołek **dzielący**: obaj jego sąsiedzi leżą **poniżej**, a kąt wewnętrzny ma **więcej** niż 180 stopni.
- Wierzchołek **łączący**: obaj jego sąsiedzi leżą **powyżej**, a kąt wewnętrzny ma **więcej** niż 180 stopni.
- Wierzchołek **prawidłowy**: pozostałe przypadki.



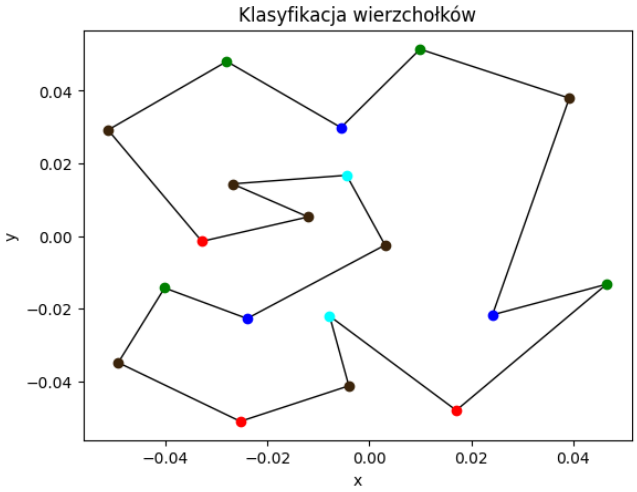
Rysunek 3: Klasyfikacja wierzchołków



Rysunek 4: Klasyfikacja wierzchołków



Rysunek 5: Klasyfikacja wierzchołków



Rysunek 6: Klasyfikacja wierzchołków

3.3 Triangulacja

1. Algorytm przeprowadzający triangulację wykona swoje zadanie poprawnie, gdy zadany wielokąt jest y-monotoniczny. Aby to sprawdzić wykorzystuję wcześniej zaimplementowaną funkcję.

2. Następnie dzielę wierzchołki na lewy i prawy łańcuch i sortuję malejąco wszystkie wierzchołki względem położenia na osi y

3. Pierwsze dwa wierzchołki trafiają na stos, a pozostałe po kolei przetwarzamy:

   (a) Jeśli ostatni wierzchołek stosu jest w innym łańcuchu, niż szczyt stosu to łączymy go z wszystkimi wierzchołkami będącymi na stosie, a stos aktualizujemy zostawiając tylko dwa ostatnio aktualizowane wierzchołki.

   (b) W przeciwnym wypadku, póki przetwarzany wierzchołek i dwa ostatnie punkty ze stosu tworzą trójkąt znajdujący się wewnątrz wielokąta to usuwamy wierzchołek ze stosu i dodajemy przekątną (jeśli nie jest krawędzią wielokąta). Gdy takowy trójkąt przestanie należeć do wielokąta, to umieszczamy badane wierzchołki na stosie i zaczynamy przetwarzać kolejny.

3.3.1 Przechowywanie triangulacji

Ze względu na istniejący moduł testowy, strukturą zwracana po zakończeniu wykonywania się algorytmu triangulacji jest lista przekątnych wielokąta, składająca się z par współrzędnych punktów, które te przekątne tworzą.

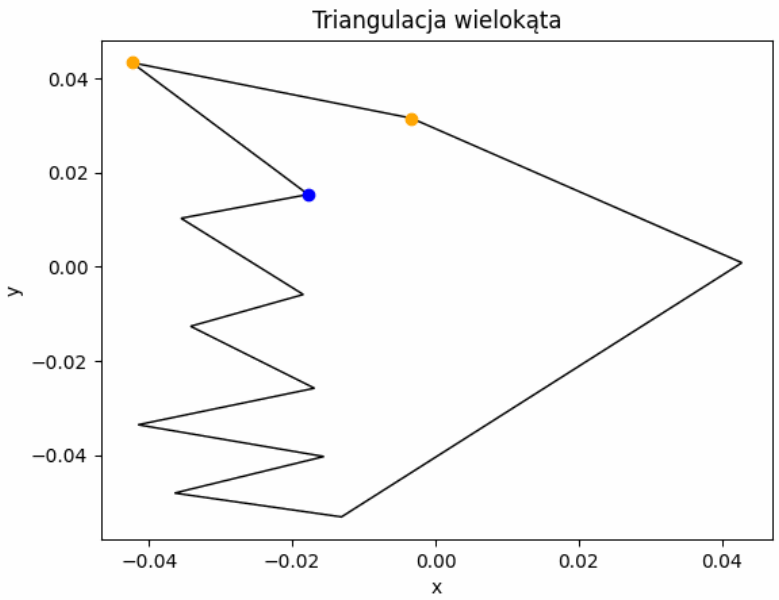
Jednak dla ułatwienia odtwarzania triangulacji została dodana możliwość zwracania przekątnych oraz odcinków, które są krawędziami wielokąta. Dzięki temu uzyskujemy więcej informacji, i mamy pełniejszy obraz triangulacji.

3.3.2 Wybór wielokątów testowych

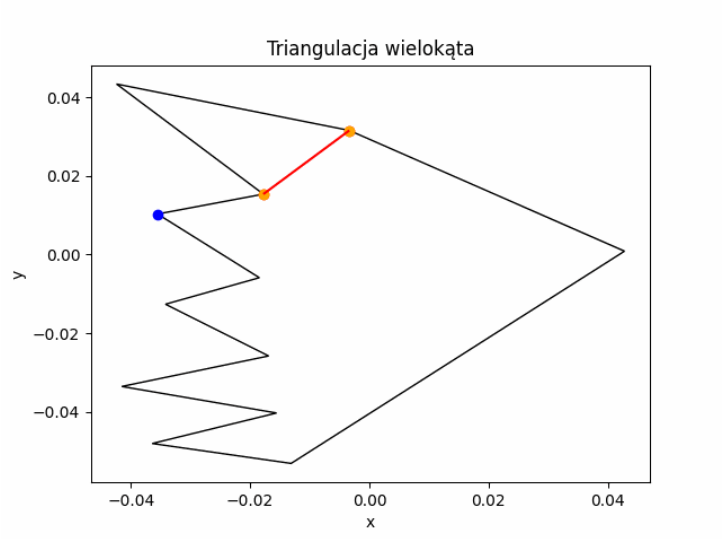
Wielokąty zostały wybrane, aby pokryły wszystkie możliwe operacje, jakie nasz algorytm wykonuje, a więc podpunkty oznaczone a i b w opisie algorytmu. Ponadto wybrany został wielokąt w kształcie "węża"oraz "z ząkbami", dla którego algorytm musi wytworzyć największą, względem ilości wierzchołków, ilość przekątnych.

3.3.3 Przykład

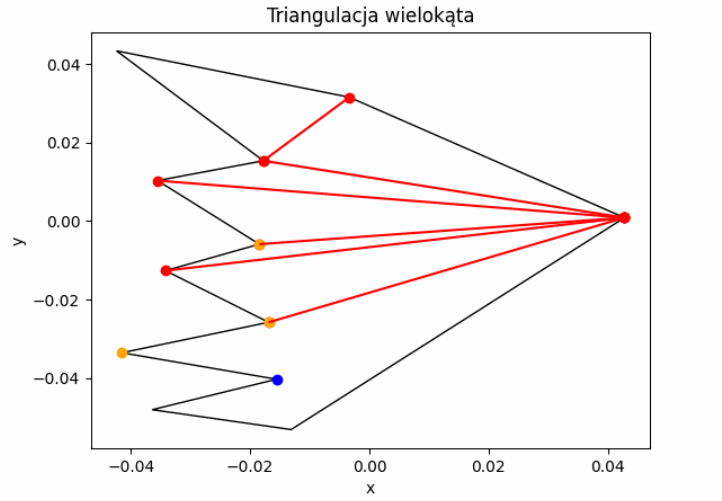
- Wierzchołek **niebieski**: aktualnie przetwarzany.
- Wierzchołek **pomarańczowy**: na stosie
- Wierzchołek/Odcinek **czerowny**: należący do triangulacji



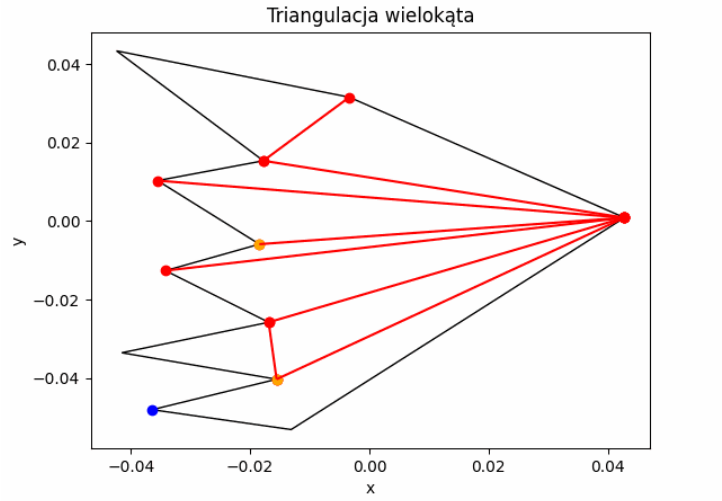
Rysunek 7: Wstawienie na stos dwóch najwyższych wierzchołków i przetwarzanie kolejnego z listy



Rysunek 8: Wykonanie operacji a, wstawienie pierwszej przekątnej do triangulacji i dodanie wierzchołków ponownie na stos

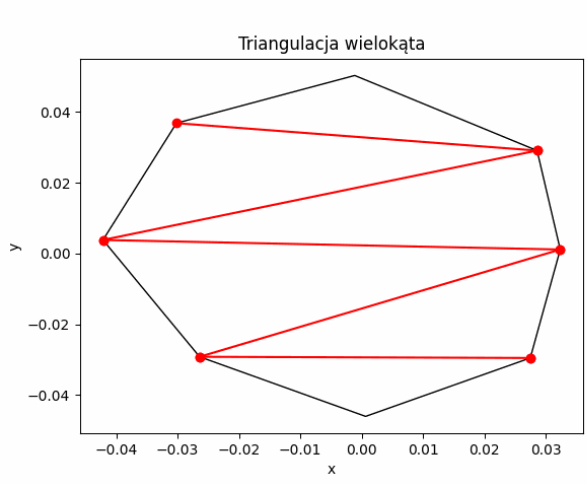


Rysunek 9: Przed wykonaniem operacji b

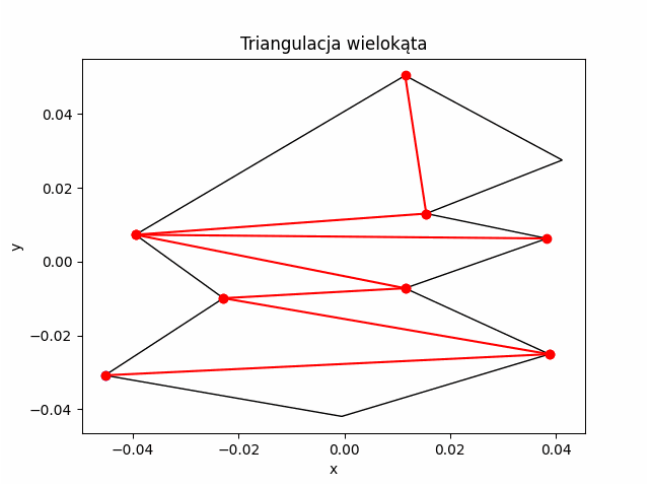


Rysunek 10: Dodanie kolejnego odcinka do triangulacji w tym samym łańcuchu

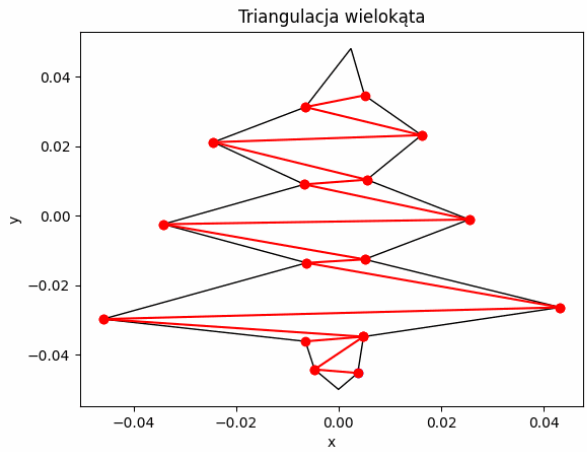
3.3.4 Rezultaty



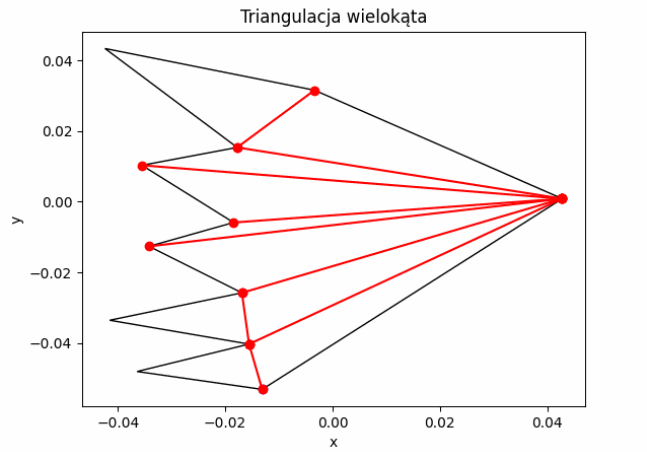
Rysunek 11: Ukończona triangulacja



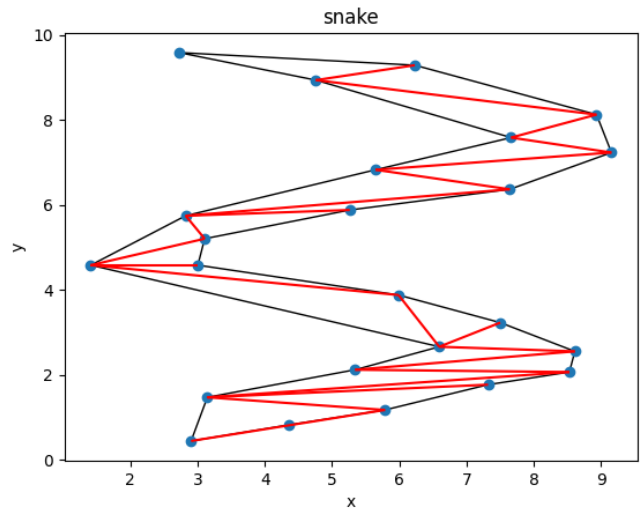
Rysunek 12: Ukończona triangulacja



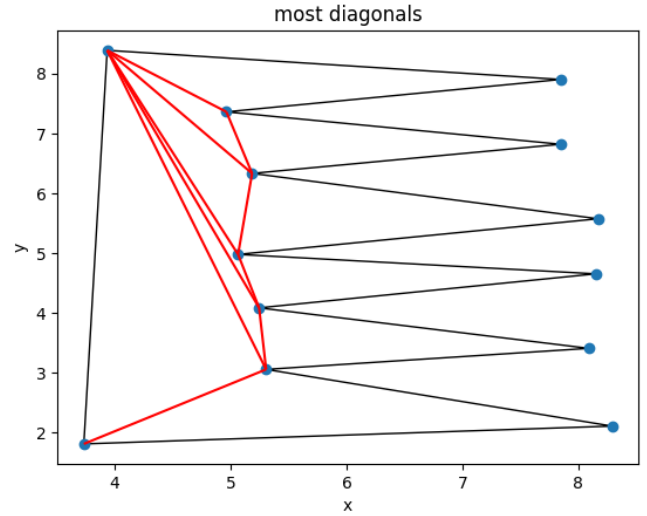
Rysunek 13: Ukończona triangulacja



Rysunek 14: Ukończona triangulacja



Rysunek 15: Wielokąt "wąż"



Rysunek 16: Wielokąt "z zębami"

4 Wnioski

Wszystkie algorytmy przeszły poprawnie wszystkie testy. Ponadto w trakcie opracowywania wyników nie zauważyłem żadnych błędów w działaniu. Można zatem stwierdzić, że ich implementacja nie zawiera żadnych błędów, a ich działanie spełnia postawione wymagania.