Mateusz Bartnicki, grupa nr 6 czwartek 13:00-14:30

Algorytmy geometryczne, sprawozdanie – ćwiczenie 3 - 16.11.2023 r.

1. Opis ćwiczenia

Głównym celem ćwiczenia było zaimplementowanie algorytmów: sprawdzania, czy dany wielokąt jest y-monotoniczny, klasyfikacji wierzchołków w dowolnym wielokącie oraz algorytmy triangulacji wielokąta monotonicznego.

2. Dane techniczne

Ćwiczenie zostało wykonane na systemie operacyjnym Windows na procesorze Intel Core i5-11400H 2.70GHz. Do jego wykonania posłużyłem się Jupyter Notebook. Program został napisany przy użyciu języka Python, wykorzystując bibliotekę matplotlib.

3. Przebieg ćwiczenia

1) Stworzenie funkcji, która umożliwia zadawanie wierzchołków wielokąta z myszki
2) Napisanie algorytmu sprawdzania, czy dany wielokąt jest y-monotoniczny.
3) Napisanie algorytmu klasyfikacji wierzchołków w dowolnym wielokącie.
4) Napisanie algorytmu triangulacji wielokąta monotonicznego.

4. Tworzenie własnych wielokątów

Do stworzenia tej funkcji posłużyłem się biblioteką matplotlib. Otwiera ona okienko, na którym należy klikać lewym przyciskiem myszy w miejsca, w których chcemy, aby były nasze kolejne wierzchołki. Istnieje możliwość zapisania figury, jeśli klikniemy w lewym dolnym rogu na ikonkę zapisu.

5. Procedura sprawdzania y-monotoniczności wielokąta

W celu sprawdzenia y-monotoniczności wielokąta, na początku wyznaczam dwa wierzchołki, o największej i najmniejszej współrzędnej y. Następnie w pętli while sprawdzam, czy współrzędne y kolejnych wierzchołków są większe niż poprzedni, zaczynając od wyznaczonego wcześniej wierzchołka o minimalnej współrzędnej y, a kończąc na drugim wyznaczonym wierzchołku. Analogicznie sprawdzam drugi łańcuch punktów, od najbardziej wysuniętego w górę punktu do najbardziej wysuniętego punktu w dół, tym razem jednak sprawdzam, czy współrzędne y są mniejsze od poprzedniego punktu.

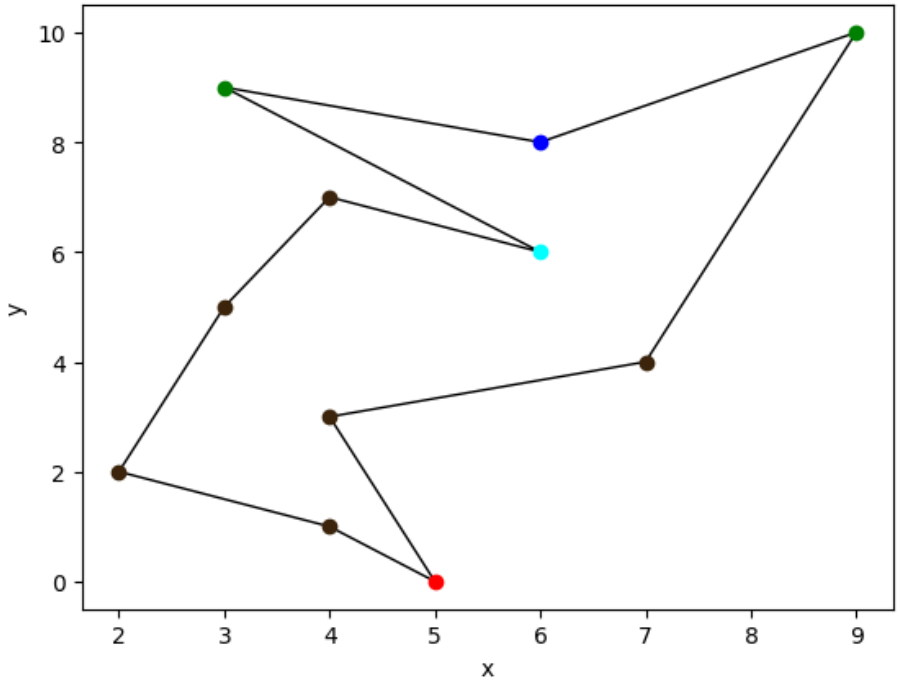
6. Klasyfikacja wierzchołków

Aby sklasyfikować punkty, wystarczy raz przejść po wszystkich wierzchołkach, porównując je ze swoimi sąsiadami (pierwszy wierzchołek jest również sąsiadem ostatniego). Punkty klasyfikowane były w sposób następujący:

- początkowe, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma mniej niż 180 stopni.
- końcowe, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma mniej niż 180 stopni.
- dzielący, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma więcej niż 180 stopni.
- łączący, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma więcej niż 180 stopni.
- prawidłowy, w pozostałych przypadkach

W celu wyznaczenia kąta wewnętrznego posłużyłem się wyznacznikiem.

Rysunek 1 - Przykładowy wielokąt z pokolorowanymi wierzchołkami



Legenda kolorów:
Zielony – wierzchołek początkowy
Czerwony - wierzchołek końcowy
Cyjanowy - wierzchołek dzielący
Niebieski- wierzchołek łączący
Czarny - wierzchołek prawidłowy

7. Algorytm triangulacji

Strukturą przechowującą wielokąt jest lista punktów, które stanowią wierzchołki wielokąta. Ich kolejność jest przeciwna do ruchu wskazówek zegara. Wynik triangulacji zwracany jest natomiast w liście krawędzi, których zawartość odpowiada indeksom wierzchołków w oryginalnym wielokącie. Wybrałem ten sposób, ponieważ dzięki temu można uniknąć duplikatów punktów, które by wystąpiły w liście trójek reprezentujących trójkąty. Oprócz tego, nie jest wymagana dodatkowa implementacja takich struktur, gdyż wystarczy użycie dostępnych w języku Python list. W ogólnym przypadku, warto zwrócić również listę początkowych wierzchołków w krotce, aby ułatwić użytkownikowi odczytanie wyników, jednak w przypadku mojej implementacji zwracana jest tylko lista krawędzi, ponieważ w ten sposób przystosowane były testy sprawdzające poprawność zwracanych wyników.

Kolejne kroki algorytmu:

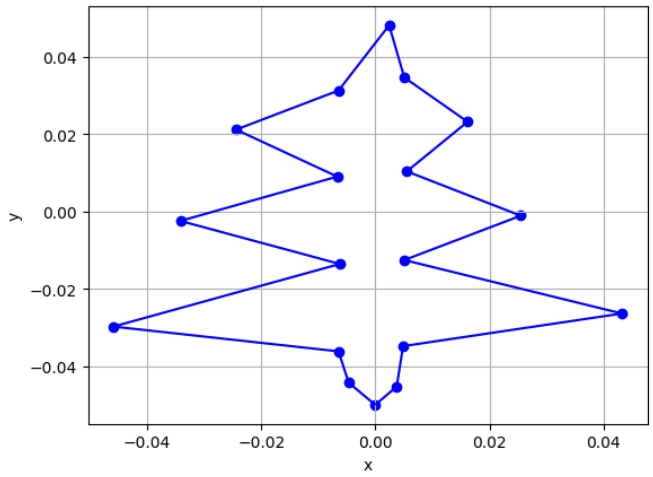
- wyznaczenie lewego i prawego łańcucha. Prawym łańcuchem są wierzchołki, które znajdują się na drodze z wierzchołka o najmniejszej współrzędnej y do wierzchołka o największej współrzędnej y. Lewy łańcuch zdefiniowany jest podobnie, lecz tym razem zaczynamy od najwyżej położonego wierzchołka, a kończymy na wierzchołku położonym najniżej.
- posortowanie listy punktów względem osi Y

- utworzenie stosu i dodanie na niego dwóch pierwszych wierzchołków (już posortowanych)
- dla każdego kolejnego wierzchołka sprawdzamy, czy należy do tego samego łańcucha, co wierzchołek z góry stosu:
    - jeśli nie, to łączę ten punkt z wszystkimi wierzchołkami na stosie. Po tej operacji, na stos wrzucam 2 ostatnie przetworzone wierzchołki
    - jeśli tak, to rozważam kolejne trójkąty, których wierzchołkami są aktualnie rozważany wierzchołek i 2 ostatnie wierzchołki stosu.
        - Jeśli trójkąt ten należy do wierzchołka, to dodaję przekątną między rozważanym wierzchołkiem a wierzchołkiem na górze stosu i rozważam kolejny trójkąt
        - Jeśli nie, to na stosie umieszczam rozważany wierzchołek i ostatni wierzchołek zdjęty ze stosu

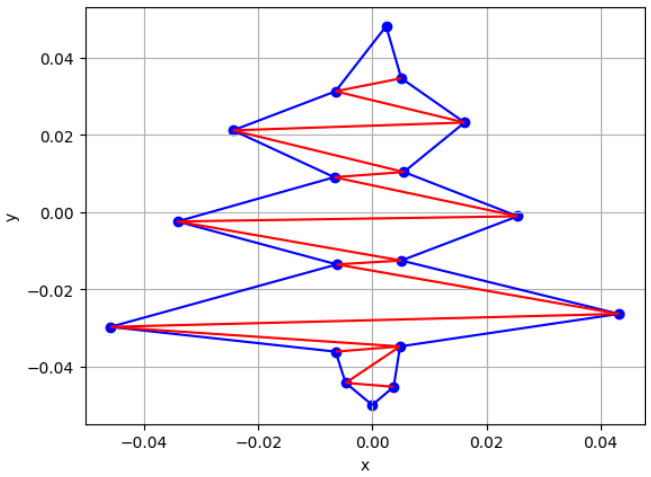
8. Rezultaty otrzymane po triangulacji

Jako rezultaty przedstawiam wizualizację wyników powyższego algorytmu na niektórych zbiorach zawartych w testach oraz jednym utworzonym własnoręcznie. Po lewej stronie znajdują się pierwotne zbiory, a po prawej zbiory wraz z zaznaczonymi uzyskanymi krawędziami

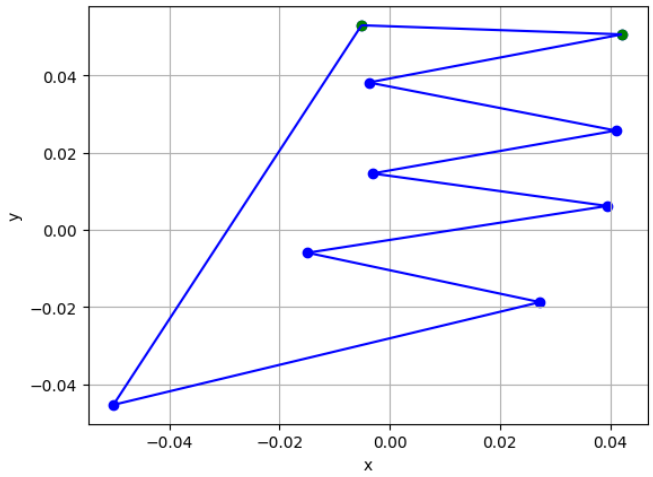
Rysunek 2 – zestaw testowy 3



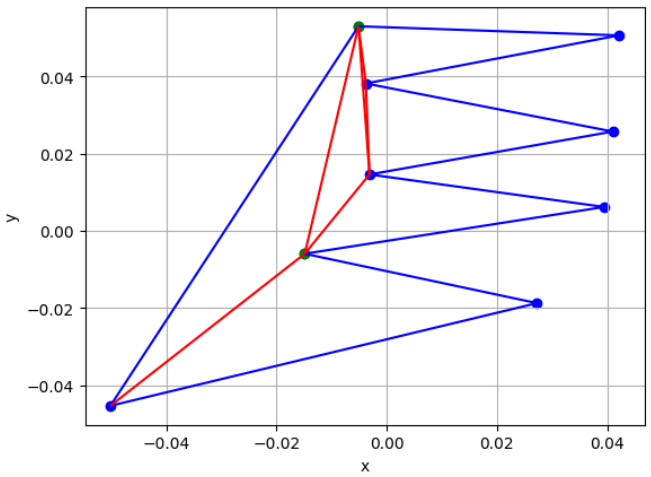
Rysunek 3 – zestaw testowy 3 po triangulacji



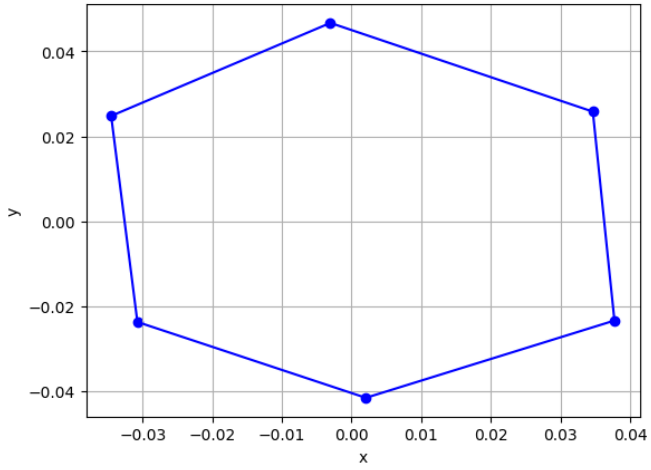
Rysunek 4 – zestaw testowy 6



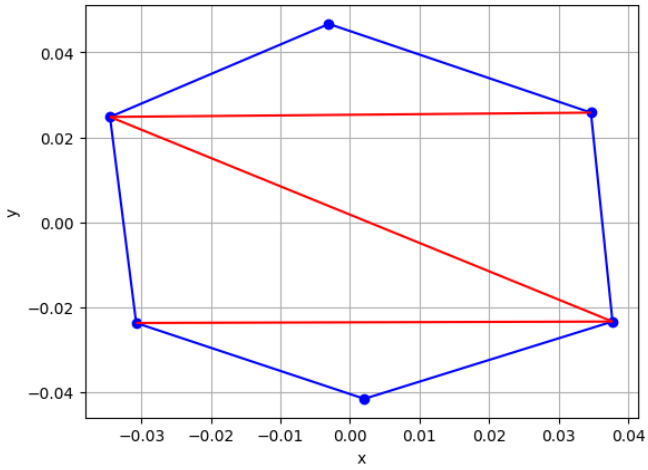
Rysunek 5 – zestaw testowy 6 po triangulacji



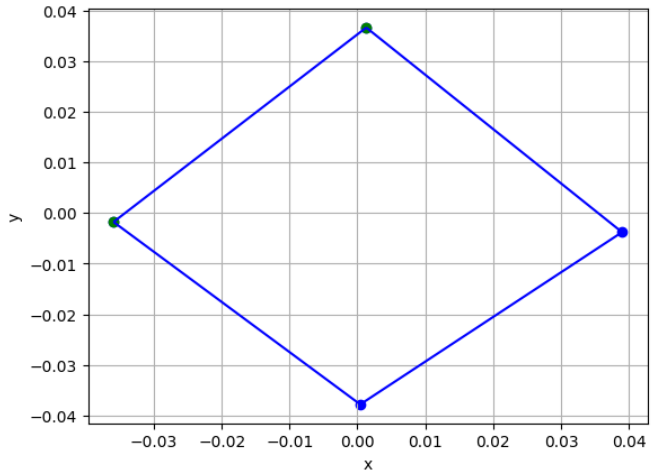
Rysunek 6 – zestaw testowy 5



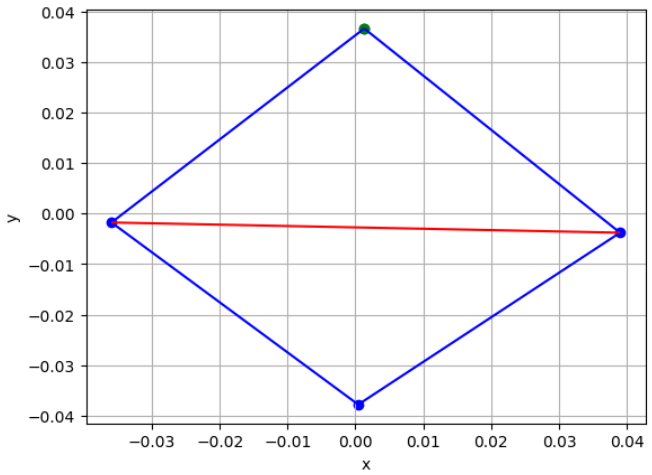
Rysunek 7 – zestaw testowy 5 po triangulacji



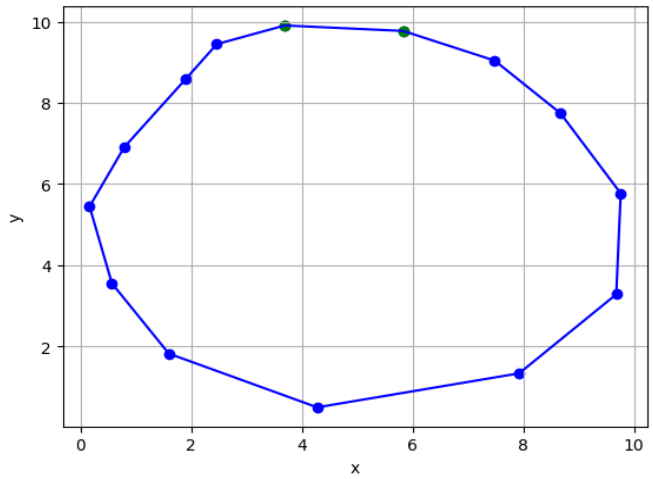
Rysunek 8 – zestaw testowy 7



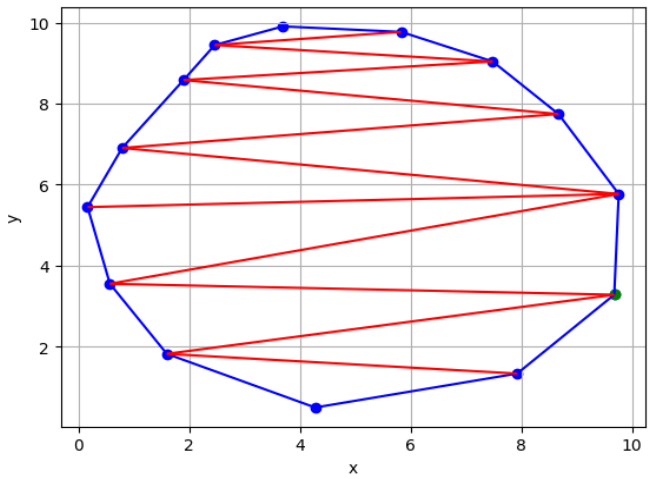
Rysunek 9 – zestaw testowy 7 po triangulacji



Rysunek 10 – zestaw własny



Rysunek 11 – zestaw własny po triangulacji



Zestaw przedstawiony na rysunku nr 2 i 3 wybrałem, ponieważ jego wierzchołki posortowane po współrzędnej y układają się łańcuchami naprzemiennie – raz prawy, raz lewy. Podobnie zdecydowałem się na wybór zestawu testowego 6, tutaj jednak wszystkie wierzchołki, poza górnym, należą do prawego łańcucha. O wyborze zestawu nr 5 zadecydowały jego wierzchołki położone na równych współrzędnych y. Na wybór zestawu z rysunków 8 i 9 przyczynił się fakt, że jest to najbardziej trywialny zestaw, którego wynikiem powinna być jedna linia. Można by

ten zestaw podzielić na trójkąty na inny sposób: łącząc wierzchołek górny z dolnym. Dlatego właśnie rysunek 9 pokazuje, w jaki sposób działa zaimplementowany algorytm. Na dobór zestawu własnego zdecydowałem się z tego samego powodu, co dla zestawu testowego 3 - posortowane wierzchołki są ułożone naprzemiennie łańcuchami.

9. Wnioski

Wszystkie zaimplementowane algorytmy spełniały swoje zadania i przechodziły wszystkie przygotowane testy. Poprawność ich działania można również łatwo zweryfikować, wykorzystując twierdzenie przedstawione na wykładzie: każdy prosty wielokąt można striangulować, a każda triangulacja prostego wielokąta o n wierzchołkach składa się z dokładnie n-2 trójkątów. Dzięki przygotowanej funkcji do zadawania wielokątów, która umożliwia testowanie działania algorytmów na własnoręcznie zadanych zbiorach punktów, możemy zobaczyć, że na zadanych wielokątach monotonicznych, zawsze dostaniemy n-2 trójkątów. Przykładem wykorzystania tej właśnie funkcji jest rysunek 11. Dodatkowo, lista krawędzi, która jest wynikiem, pozwala nam również łatwo zweryfikować te odpowiedzi, gdyż lista ta zawiera indeksy wierzchołków, między którymi dodane zostały przekątne, co pozwala w prosty sposób odwołać się do indeksów początkowego wielokąta i porównać wyniki z wykresem.