Jan Kaczmarski

Informatyka II rok, grupa 5, 5.11.2024

Algorytmy geometryczne, laboratorium 3 - sprawozdanie

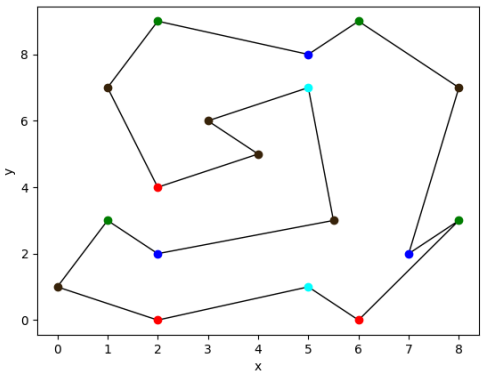
1. Cel ćwiczenia i wstęp

Celem ćwiczenia jest zapoznanie się z zagadnieniami monotoniczności wielokątów oraz implementacja algorytmów: sprawdzania, czy wielokąt jest y-monotoniczny; klasyfikacji wierzchołków w dowolnym wielokącie; triangulacji wielokąta monotonicznego, a także wizualizacja i opracowanie wyników.

Formalnie mówimy o wielokącie monotonicznym względem pewnej prostej l. W tym zadaniu będzie nas interesować y-monotoniczność (względem osi OY). Wielokąt nazywamy y-monotonicznym, gdy jego wierzchołki mogą być ułożone w taki sposób, że jego współrzędna y-owa zawsze rośnie lub maleje wzdłuż kolejnych wierzchołków. Innymi słowy, dla każdej pary sąsiednich wierzchołków wielokąta (oprócz wierzchołka startowego i końcowego), jeden z punktów ma większą (lub mniejszą) wartość danej współrzędnej niż drugi.

Kategorie wierzchołków:

1. **Początkowe** - gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma mniej niż 180^o
2. **Końcowe** - gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma mniej niż 180^o
3. **Dzielący** - gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma więcej niż 180^o
4. **Łączący** - gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma więcej niż 180^o
5. **Prawidłowy** - pozostałe przypadki (jeden sąsiad powyżej, drugi poniżej)



Rysunek 1.1 Przykładowe kolorowanie wierzchołków

Zarówno na rysunku 1.1, jak i w pliku z implementacją jest przyjęta następująca konwencja kolorowania w zależności od kategorii wierzchołka: 1 – zielony, 2 – czerwony, 3 – ciemnoniebieski, 4 – jasnoniebieski, 5 – czarny.

2. Metodologia, specyfikacja narzędzi i sprzętu

Do definiowania wielokątów przy użyciu myszy wykorzystałem funkcje dostępne w bibliotece matplotlib. Algorytm odpowiedzialny za klasyfikację wierzchołków na różne kategorie oraz za triangulację wykorzystuje wyznacznik macierzy 3x3, omawiany na laboratorium nr 1, do określania, czy trzy punkty tworzą układ o skręcie lewo- lub prawostronnym. W celu wizualizacji przykładowych oraz generowanych wielokątów, podziału wierzchołków na kategorie oraz kolejnych etapów triangulacji, stworzyłem wykresy z wykorzystaniem biblioteki matplotlib. Ponadto użyłem narzędzia przygotowanego przez koło naukowe Bit, które umożliwiło dokładne zobrazowanie całego procesu.

Dane wielokąta przechowywałem w tablicy monotonic_order, gdzie wierzchołki były posortowane według współrzędnej y. Tablica ta zawierała również informację o przynależności wierzchołków do lewego lub prawego łańcucha, co znacząco ułatwiło szybkie określenie ich właściwości, takich jak kolejność czy przynależność do danego łańcucha.

Triangulacja została zapisana w postaci tablicy, w której każda przekątna była reprezentowana przez parę wierzchołków wielokąta tworzących tę przekątną. Dzięki takiemu podejściu struktura triangulacji była przejrzysta i jednoznaczna.

Program został napisany w języku Python i zapisany w pliku „kaczmarski_3_kod.ipynb”. Uruchamiałem go w środowisku Jupyter Notebook na komputerze MacBook Pro z procesorem M2 Pro ARM oraz systemem macOS. Wszystkie wyniki zamieszczone w raporcie zostały wygenerowane w ramach działania tego programu.

3. Program Ćwiczenia

Na początku zaimplementowałem funkcje umożliwiające definiowanie wielokątów przy użyciu myszy. Zgodnie z dokumentacją zawartą w pliku implementacyjnym, wystarczy kliknąć w wybrane punkty, aby dodać kolejne wierzchołki, a następnie zakończyć proces definiowania. Wielokąt powinien być tworzony w kierunku przeciwnym do ruchu wskazówek zegara, ponieważ w tej formie działają zaimplementowane funkcje. Użytkownik może także wygenerować losowy wielokąt i przetestować algorytm na nim. Dodatkowo przygotowałem zestaw wielokątów testowych, których szczegółowy opis znajduje się w kolejnym rozdziale.

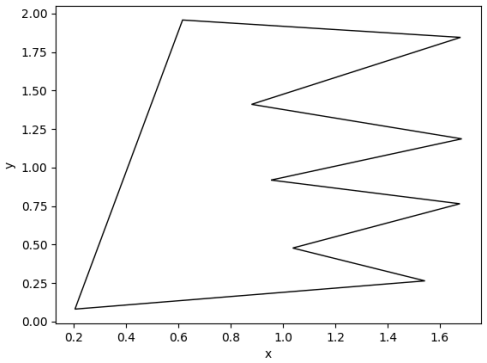
Kolejnym etapem było stworzenie procedury sprawdzającej, czy wielokąt jest y-monotoniczny. Szczegółowy kod tej metody został umieszczony w pliku .ipynb, a tutaj przedstawię jej główne założenia. W pierwszej kolejności wyznaczyłem lewy i prawy łańcuch wielokąta, zaczynając od identyfikacji wierzchołków skrajnych – górnego i dolnego (przy punktach o tych samych współrzędnych y wybierałem pierwszy napotkany). Ponieważ wierzchołki były definiowane przeciwnie do ruchu wskazówek zegara, określenie łańcuchów było dość intuicyjne: lewy łańcuch obejmuje wierzchołki ułożone przeciwnie do ruchu wskazówek zegara, natomiast prawy zgodnie z tym ruchem (przy założeniu, że współrzędne y sukcesywnie maleją).

Procedura analizuje, czy podczas przechodzenia przez każdy z łańcuchów wierzchołki układają się wyłącznie w kierunku malejących współrzędnych y. Jeśli podczas tej analizy wystąpi sytuacja, w której „wracamy” w górę, oznacza to, że wielokąt nie spełnia warunku y-monotoniczności. W przeciwnym wypadku wielokąt uznaje się za y-monotoniczny.

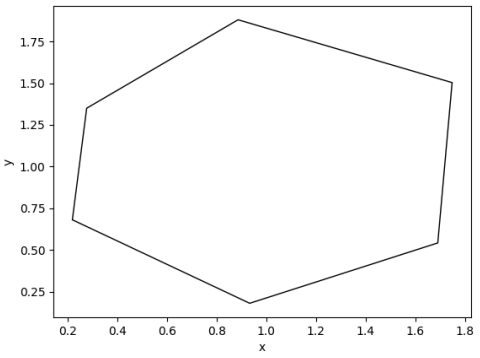
Na koniec zaimplementowałem funkcje odpowiedzialne za kategoryzowanie wierzchołków oraz przeprowadzanie triangulacji.

4. Testowane zbiory danych i wyniki działania algorytmów

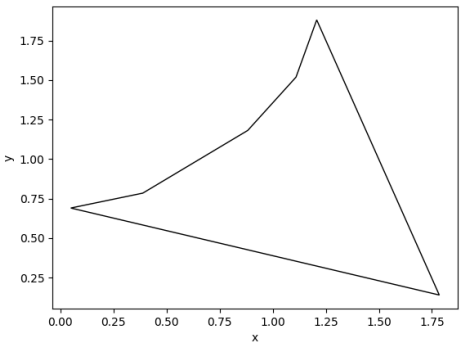
Aby testy były rzetelne, starałem się dobrać wielokąty różnej natury, które mogłyby potencjalnie sprawić problemy programowi / przetestować przypadki brzegowe. Na następnych stronach przedstawiam dobrane pięć wielokątów, dla których przetestuję działanie zaimplementowanych w ramach tego laboratorium algorytmów.



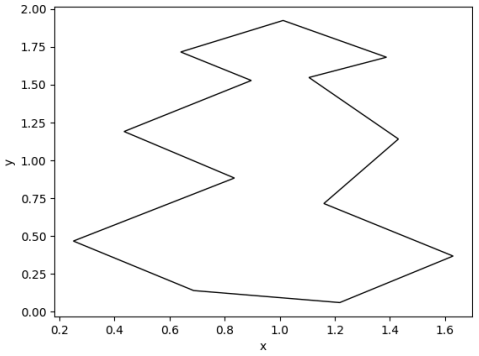
Rys. 4.1 Pierwszy testowany wielokąt – A



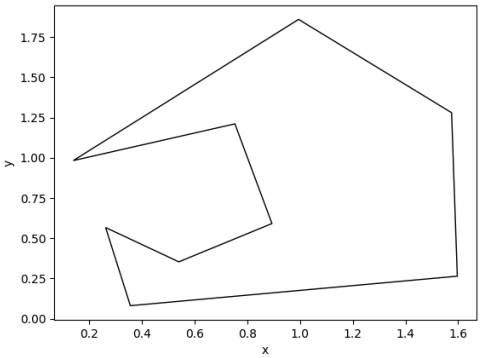
Rys 4.2 Drugi testowany wielokąt - B



Rys. 4.3 Trzeci testowany wielokąt - C



Rys. 4.4 Czwarty testowany wielokąt – D



Rys. 4.5 Piąty testowany wielokąt – E (uwaga, wielokąt nie jest monotoniczny)

Wielokąt B testuje zachowanie algorytmu w przypadku podstawowym. Wielokąty D i A zostały dobrane, aby sprawdzić działanie algorytmu w skrajnych sytuacjach, w których trudno jednoznacznie stwierdzić, czy dana krawędź triangulacji znajduje się wewnątrz wielokąta. Z kolei wielokąt C weryfikuje przypadek, w którym jeden z wierzchołków powinien zostać połączony z pozostałymi wierzchołkami.

W poniższej tabeli przedstawiam wyniki działania funkcji sprawdzającej monotoniczność wielokąta – TRUE, jeśli jest y-monotoniczny, FALSE w przeciwnym przypadku.

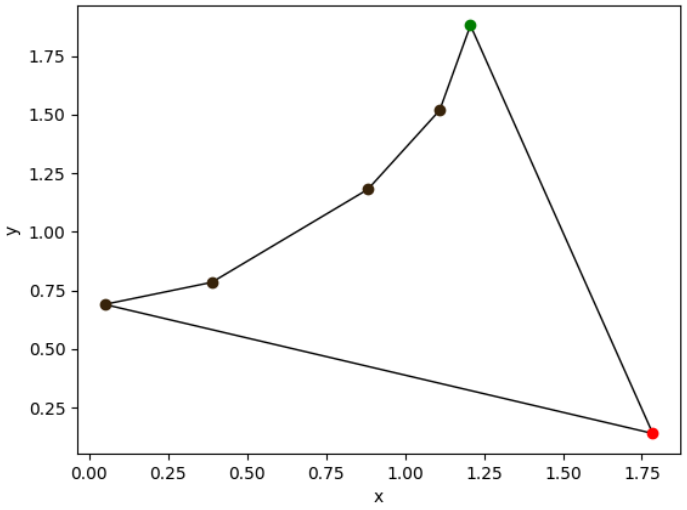
Wielokąt	A	B	C	D	E
Czy monotoniczny	TRUE	TRUE	TRUE	TRUE	FALSE

Tabela 4.6 Sprawdzenie monotoniczności wielokątów

Jak widać w tabeli 4.6, istotnie ostatni wielokąt nie jest y-monotoniczny, ale pozostałe są.

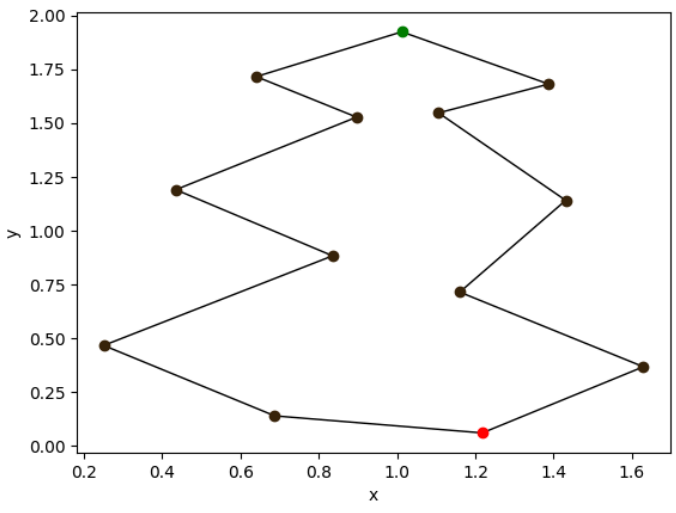
Teraz przedstawię podział wierzchołków na kategorie dla każdego z testowanych wielokątów. Konwencja kolorowania została wspomniana wcześniej, w rozdziale 2.

Wielokąt A



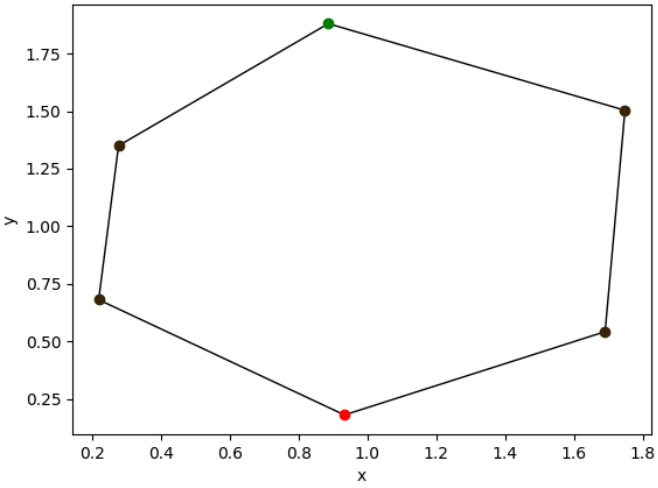
Pokolorowanie wielokąta A

Wielokąt B



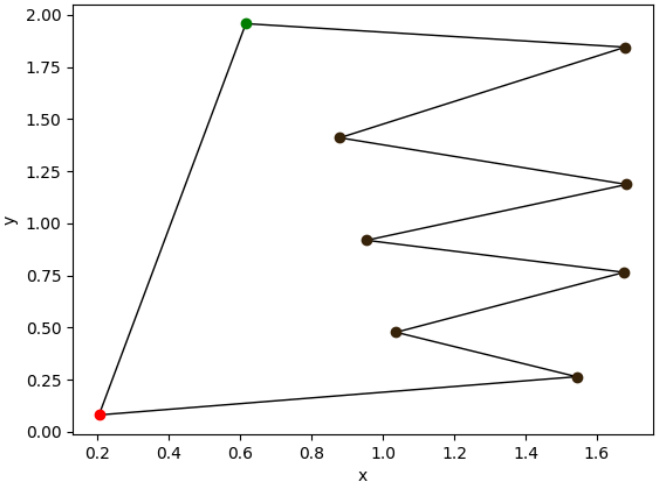
Pokolorowanie wielokąta B

Wielokąt C



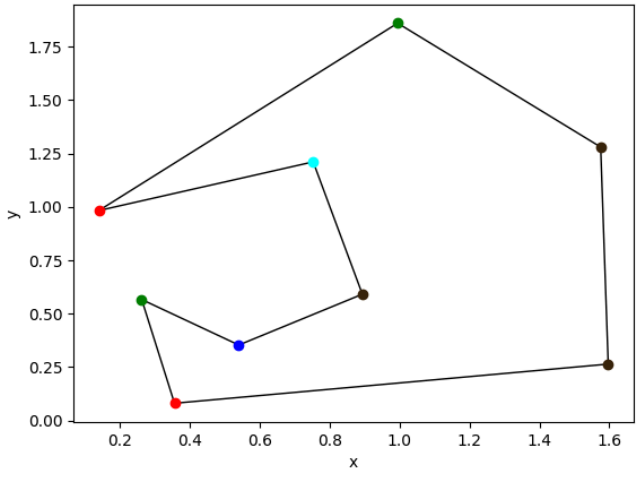
Pokolorowanie wielokąta C

Wielokąt D



Pokolorowanie wielokąta D

Wielokąt E



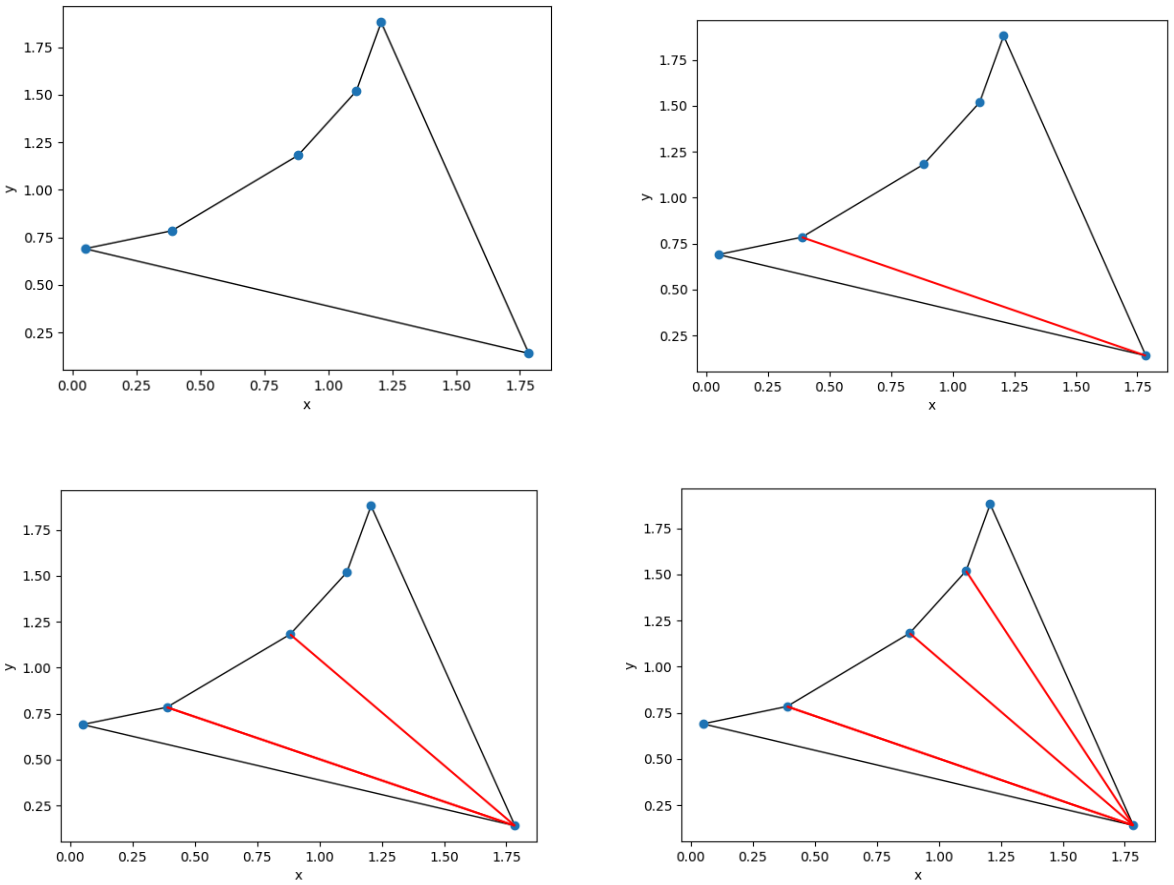
Rys 4.7 Pokolorowanie wielokąta E

Jak widać na rysunku 4.7, wielokąt E, jako niemonotoniczny, jest jedynym, który ma wierzchołki dzielące i łączące. Pozostałe wielokąty mają jedynie wierzchołki początkowe, końcowe i prawidłowe.

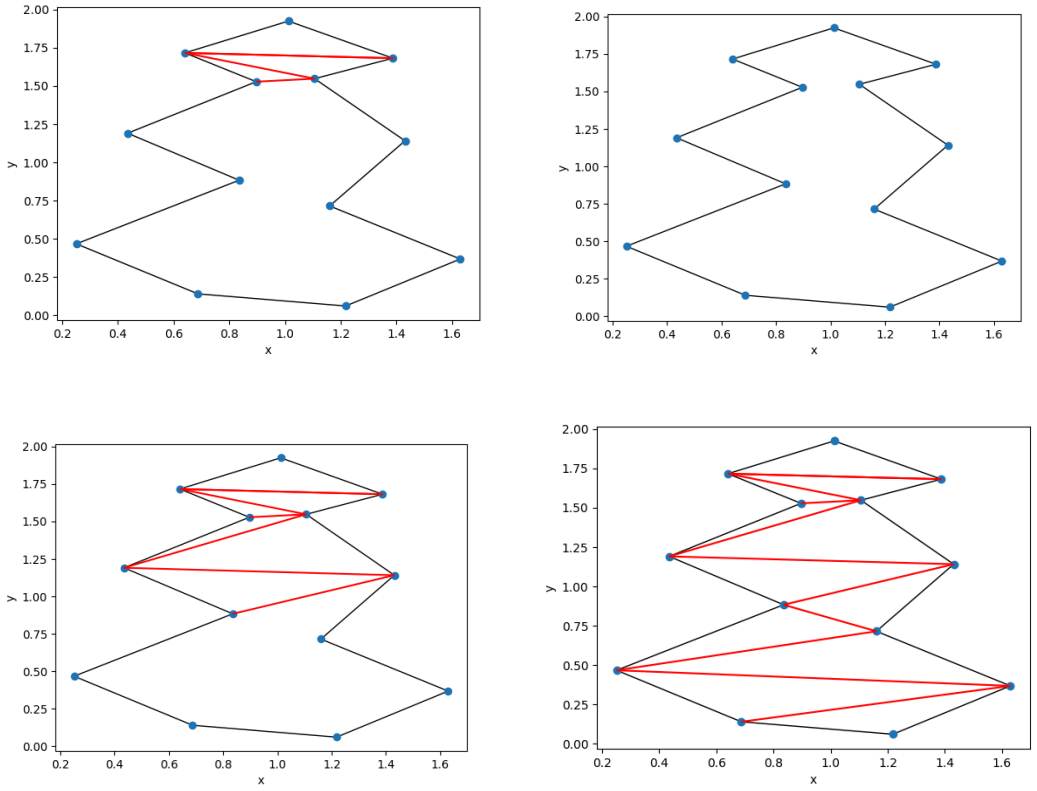
Triangulacja wielokątów (monotonicznych)

Dla każdego z wielokątów A, B, C i D zaprezentuję cztery kluczowe kroki w procesie ich triangulacji. W przypadku wielokąta E, triangulacja nie jest możliwa bez uprzedniego podzielenia go na wielokąty monotoniczne, dlatego ten przypadek nie będzie tu omawiany. Przekątne dodane w trakcie triangulacji zostały oznaczone kolorem czerwonym.

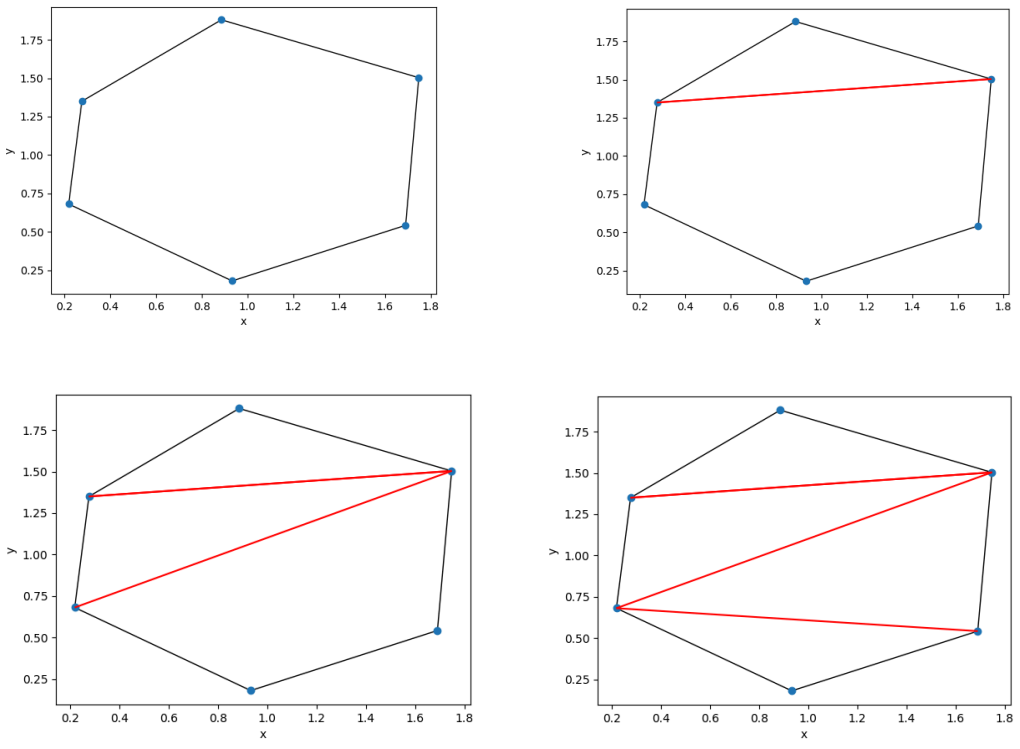
Warunek `if abs(current[1] - stack[j][1]) not in (1, n-1)`: zapewnia, że algorytm dodaje jedynie przekątne łączące niesąsiadujące wierzchołki wielokąta. Indeksy różniące się o 1 lub o $n-1$ wskazują na wierzchołki sąsiadujące, które tworzą krawędzie oryginalnego wielokąta. Dzięki temu warunkowi algorytm unika dodawania istniejących krawędzi jako przekątnych i zachowuje poprawność triangulacji.



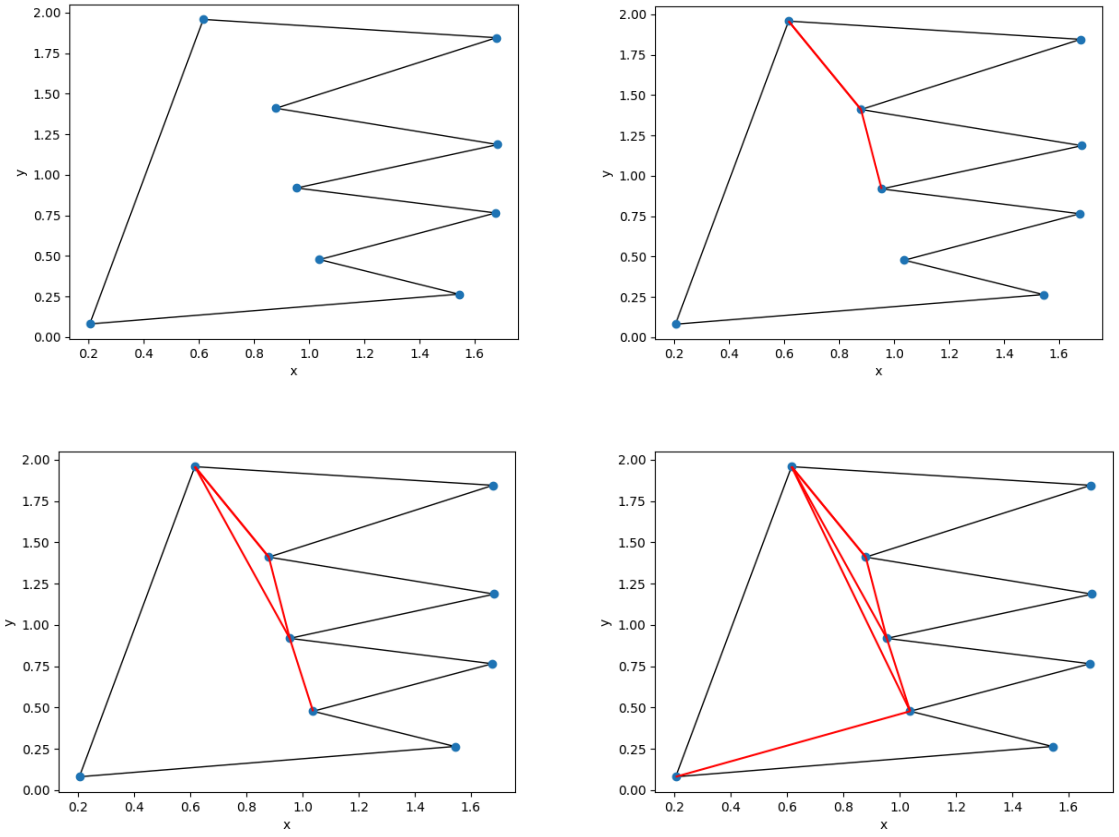
Rys. 4.8 Wybrane etapy triangulacji wielokąta A



Rys. 4.9 Wybrane etapy triangulacji wielokąta B



Rys. 4.10 Wybrane etapy triangulacji wielokąta C



Rys. 4.11 Wybrane etapy triangulacji wielokąta D

5. Podsumowanie i wnioski

Do testowania programu wybrałem wielokąty o różnej specyfice, aby sprawdzić, czy w procesie triangulacji nie zostaną dodane nieprawidłowe przekątne. Takie błędy mogłyby obejmować przekątne wychodzące poza wielokąt, pokrywające się z jego bokiem lub przecinające już istniejące przekątne, co skutkowałoby niepoprawną triangulacją. Zarówno dla przygotowanych wielokątów testowych, jak i dla tych użytych w testach Koła Naukowego AGH Bit, program działał prawidłowo. Na wygenerowanych rysunkach triangulacji można zobaczyć poprawnie utworzone trójkąty. Program, oparty na algorytmie z wykładu, nie obsługuje jednak wielokątów niemonotonicznych – dla nich wyniki byłyby nieprawidłowe. Kluczowe dla poprawnego działania algorytmu okazała się przyjęta konwencja reprezentacji wielokąta oraz pomocnicza struktura wspierająca proces triangulacji, która współgrała z użytym algorytmem.