Aleksander Jóźwik gr. 2, Pon. godz. 15:00 A

Data wykonania: 05.11.2024 Data oddania: 19.11.2024

Algorytmy Geometryczne - laboratorium 3

Triangulacja wielokątów monotonicznych

1. Dane techniczne

- System operacyjny: Fedora Linux (x86-64)
- Procesor: Intel Core i5-8350U (1.70 - 3.60 GHz)
- Pamięć RAM: 16GB (2133 MHz)
- Środowisko: Jupyter Notebook
- Język: Python 3.9.20

W realizacji ćwiczenia wykorzystano biblioteki *numpy*, *pandas* oraz narzędzie wizualizacji stworzone przez koło naukowe *BIT*. Użyta precyzji przechowywania zmiennych i obliczeń w ćwiczeniu to *float64*.

2. Cel ćwiczenia

Zapoznanie się z zagadnieniami dotyczącymi monotoniczności wielokątów oraz implementacja algorytmów sprawdzania y-monotoniczności wielokąta, klasyfikacji wierzchołków w dowolnym wielokącie oraz triangulacji wielokąta y-monotonicznego. Celem było także wykonanie poszczególnych wizualizacji oraz analiza danych.

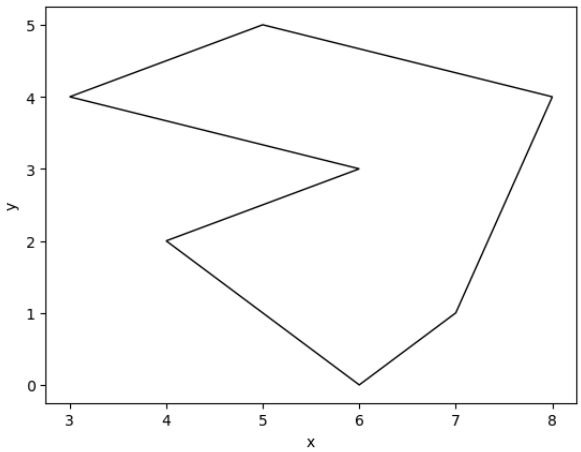
3. Wstęp teoretyczny

3.1. Monotoniczność wielokąta

Wielokąt prosty jest ściśle monotoniczny względem prostej l (wyznaczającej kierunek monotoniczności), kiedy jego brzeg można przedstawić w postaci dwóch spójnych łańcuchów, z których każdy przecina się z dowolną prostą l' prostopadłą do l w nie więcej niż jednym punkcie.

Przecięcie wielokąta z l' jest spójne, co oznacza że jest ono odcinkiem, punktem lub jest puste.

Wielokątem y-monotonicznym (Rysunek 1) nazywamy wielokąt, który jest monotoniczny względem osi y, czyli w przypadku przejścia z najwyższego wierzchołka do najniższego, wzdłuż prawego lub lewego łańcucha, zawsze poruszamy się w dół lub poziomo.

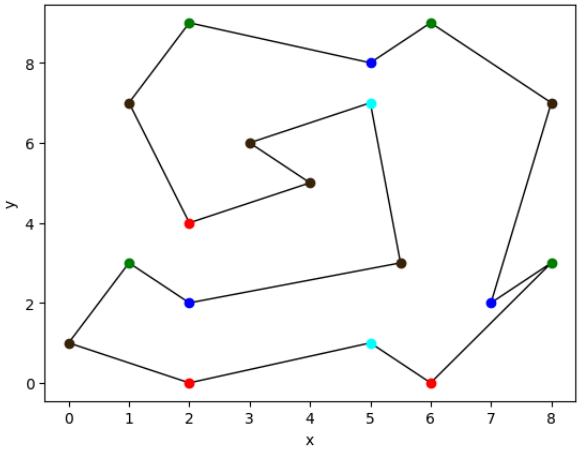


Rysunek 1: Przykładowy wielokąt y-monotoniczny

3.2. Klasyfikacja wierzchołków wielokąta

Wierzchołki wielokąta można podzielić na 5 kategorii:

1. **wierzchołek początkowy** - obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny $< \pi$,
2. **wierzchołek końcowy** - obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny $< \pi$,
3. **wierzchołek łączący** - obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny $> \pi$,
4. **wierzchołek dzielący** - obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny $> \pi$,
5. **wierzchołek prawidłowy** - dla pozostałych przypadków.

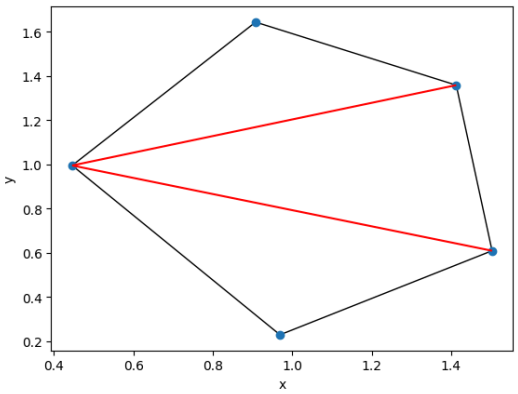


Rysunek 2: Przykładowy wielokąt niemonotoniczny z wierzchołkami pokolorowanymi zgodnie z ich klasyfikacją

Wielokąt y-monotoniczny nie posiada wierzchołków łączących i dzielących.

3.3. Triangulacja wielokąta

Triangulacja wielokąta to podział wielokąta na rozłączne trójkąty poprzez dodanie nieprzecinających się przekątnych łączących jego wierzchołki. W procesie triangulacji wszystkie dodawane przekątne muszą leżeć wewnątrz wielokąta, a ich końce muszą być jego wierzchołkami. Dla wielokąta o n wierzchołkach triangulacja zawsze składa się z $n-2$ trójkątów połączonych $n-3$ przekątnymi. Przykład takiej triangulacji można zauważyć na Rysunku 3.



Rysunek 3: Przykładowa triangulacja wielokąta y-monotonicznego

4. Realizacja ćwiczenia

4.1. Szczegóły implementacyjne

4.1.1. Procedura sprawdzania y-monotoniczności wielokąta

Zaimplementowana metoda sprawdzania y-monotoniczności wykorzystuje własność figury, będącej łamaną zamkniętą (koniec ostatniego odcinka jest zarazem początkiem pierwszego odcinka) oraz to, że punkty zadane są w kierunku przeciwnym do ruchu wskazówek zegara. Dzięki cyklicznemu przechodzeniu po wierzchołkach możemy efektywnie sprawdzić y-monotoniczność bez obawy o wyjście poza zakres tablicy:

1. Wyznaczamy indeksy wierzchołków o minimalnej i maksymalnej wartości y (*min_idx* i *max_idx*).
2. Przechodzimy cyklicznie od *max_idx* do *min_idx*, sprawdzając, czy kolejne wierzchołki mają nierosnące wartości y.
3. Następnie kontynuujemy od *min_idx* do *max_idx*, weryfikując, czy wartości y są niemalejące.
4. Wykorzystanie operacji *modulo* ($\%n$) pozwala na cykliczne przechodzenie po indeksach wierzchołków, traktując tablicę punktów jako zamknięty okrąg.
5. Jeśli podczas iteracji warunek monotoniczności zostanie naruszony, funkcja zwraca *False*, w przeciwnym razie *True*.

4.1.2. Funkcja obliczająca wartość wyznacznika

W poniższych algorytmach znalazła zastosowanie funkcja obliczająca wartość wyznacznika 3x3, zdefiniowana w następujący sposób:

$$\det(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)\big(c_y - b_y\big) - \big(b_y - a_y\big)(c_x - b_x)$$

Ze względu na precyzję obliczeń, niepewność dla zera określono na $\boldsymbol{\varepsilon = 10^{-24}}$.

4.1.3. Algorytm klasyfikacji wierzchołków wielokąta

Algorytm klasyfikacji wierzchołków wielokąta działa zgodnie z warunkami opisanymi w Sekcji 3.2. Polega on więc na analizie każdego wierzchołka pod kątem położenia jego sąsiadów oraz wartości kąta wewnętrznego w tym wierzchołku. Na początku sprawdzane jest czy sąsiednie wierzchołki leżą powyżej czy poniżej bieżącego wierzchołka, poprzez porównanie ich współrzędnych y. Następnie aby określić, czy kąt jest wypukły czy wklęsły, obliczana jest wartość wyznacznika dla kolejnych punktów tj. (poprzedniego, aktualnego, następnego):

- kąt jest wklęsły $\Leftrightarrow \det(\mathit{prev}, \mathit{curr}, \mathit{next}) < 0$
- kąt jest wypukły $\Leftrightarrow \det(\mathit{prev}, \mathit{curr}, \mathit{next}) > 0$

Implementacja przeprowadza powyższe kroki dla każdego wierzchołka wielokąta, iterując przez listę punktów. Na podstawie wyników dokonywana jest klasyfikacja i przypisanie kategorii poszczególnym wierzchołkom o danych indeksach w tablicy.

4.1.4. Algorytm triangulacji wielokąta monotonicznego

Algorytm rozpoczyna się od podziału punktów (uporządkowanych w kolejności przeciwnej do ruchu wskazówek zegara) na dwa łańcuchy: lewy i prawy. Do krotek wierzchołków dodawana jest w tym czasie informacja o ich indeksie z oryginalnej tablicy. Sposób dokonania podziału jest analogiczny do metody sprawdzania y-monotoniczności opisanej w Sekcji 4.1.1. Następnie oba łańcuchy są scalane w jedną listę, którą cechuje malejący porządek względem y. Równolegle w krotkach punktów zapisywana jest informacja o przynależności do lewego lub prawego łańcucha. Kolejnym krokiem jest inicjalizacja stosu przechowującego wierzchołki, z którymi potencjalnie można utworzyć przekątne prowadzące do triangulacji. Wkładamy na niego dwa pierwsze punkty z posortowanej listy. W głównej pętli algorytmu iterujemy przez kolejne wierzchołki z uporządkowanej listy, zaczynając od indeksu 2. Rozpatrywana jest ich przynależność do łańcucha.

- Jeżeli aktywnie przetwarzany punkt należy do innego łańcucha niż wierzchołek na szczycie stosu, to dodajemy przekątne między nim, a wszystkimi kolejnymi od wierzchu stosu wierzchołkami, o ile nie są one sąsiadami w oryginalnym wielokącie (sprawdzane przez różnicę zapisanych indeksów).
  Po dodaniu przekątnych, redukujemy stos do ostatniego wierzchołka.
- Jeżeli oba punkty należą do tego samego łańcucha, to analizujemy kolejne trójkąty jakie tworzy dany wierzchołek z punktami zdejmowanymi ze stosu. Możliwość utworzenia trójkątów sprawdzana jest poprzez weryfikację czy wierzchołki nie są sąsiadami w oryginalnym wielokącie (w analogiczny sposób co w poprzednim przypadku) oraz czy trójkąt należy do wielokąta. Ostatni warunek sprawdzany jest przy użyciu wyznacznika. Dla przetwarzanego wierzchołka z lewego łańcucha, punkty muszą tworzyć układ lewoskrętny ($\det > 0$), a z prawego - prawoskrętny ($\det < 0$). Jeżeli trójkąt spełnia te założenia, to dodajemy przekątną i usuwamy odpowiedni wierzchołek ze szczytu stosu. Kluczową optymalizacją w tej metodzie jest sposób wyboru trójkątów. Algorytm tworzy je wykorzystując punkt ze szczytu stosu oraz jeden z kolejnych punktów na stosie, co znacząco zmniejsza ryzyko powstania przecinających się przekątnych. Jest to możliwe dzięki specyficznej właściwości stosu - najniżej położone punkty znajdują się na jego szczycie. W rezultacie pierwszy poprawnie utworzony trójkąt, złożony z danego punktu i dwóch pierwszych odpowiednich wierzchołków ze stosu, będzie zawsze optymalnym wyborem dla triangulacji.

Następnie aktualnie przetwarzany punkt zostaje dodany na stos i rozpoczyna się przetwarzanie pozostałych wierzchołków.

Proces triangulacji jest usprawniony poprzez systematyczne usuwanie ze stosu tych punktów, które mogłyby tworzyć problematyczne trójkąty - czyli takie, których przekątne kolidowałyby z już istniejącą triangulacją. Zaczyna od wierzchołka stosu i usuwa kolejne punkty aż do znalezienia takiego, który pozwala utworzyć prawidłowy trójkąt. To podejście jest wydajne, ponieważ eliminuje konieczność ponownego sprawdzania tych samych trójkątów, umożliwiając liniowe przetwarzanie punktów na stosie. Algorytm korzysta z właściwości wielokąta y-monotonicznego, gdzie dowolna prosta l' prostopadła do l przecina go co najwyżej dwa razy. Dzięki temu możliwe jest podzielenie wielokąta na dwa łańcuchy i efektywne zastosowanie stosu do triangulacji. Jego złożoność obliczeniowa to $O(n)$, gdzie n to liczba wierzchołków.

Algorytm zwraca listę krotek zawierających indeksy wierzchołków, między którymi należy poprowadzić przekątne. Dane te są później przetwarzane na listę krawędzi wielokąta po triangulacji.

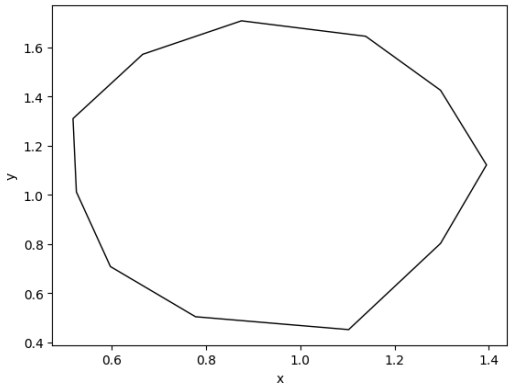
4.1.5. Struktury przechowujące wielokąty oraz utworzone triangulacje

Wielokąt przechowywany jest jako lista wierzchołków zadanych w kolejności przeciwnej do ruchu wskazówek zegara. Odcinki, którymi można je połączyć stanowią bok wielokąta. Zwracana przez algorytm triangulacja to tablica krotek, które reprezentują naniesione przekątne pomiędzy punktami o poszczególnych indeksach z tablicy wierzchołków wielokąta. Taka reprezentacja danych została wybrana ze względu na prostotę późniejszej wizualizacji przy pomocy narzędzia udostępnionego przez koło naukowe *BIT*. W przypadku reprezentacji przekątnych, operowanie na indeksach zamiast na jawnych współrzędnych punktów zwiększa precyzję obliczeń, ponieważ eliminuje ryzyko błędów związanych z arytmetyką zmiennoprzecinkową.

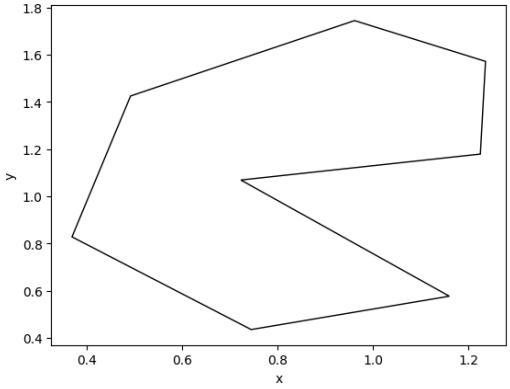
Ostatecznie informacje o wierzchołkach i przekątnych są wykorzystywane do stworzenia listy krawędzi, która przedstawia wielokąt po triangulacji. Tablica krotek reprezentuje każdą krawędź jako parę punktów. Postać taka jest łatwa do utworzenia oraz pozwala na łatwy zapis w celu późniejszego użycia.

4.2. Zestawy danych testowych

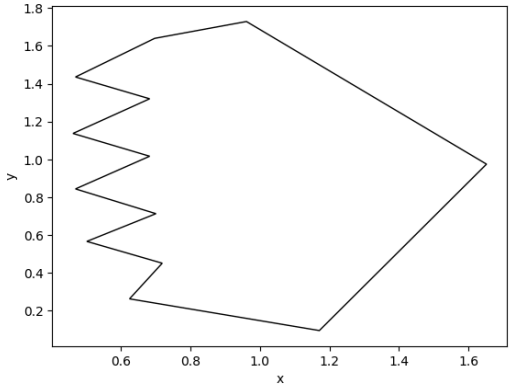
W celu przetestowania powyższych implementacji, wybrano zbiory danych testowych. Wielokąty były zadawane poprzez aplikację graficzną korzystającą z biblioteki *matplotlib*. Kolejność wprowadzania punktów była przeciwna do kierunku ruchu wskazówek zegara.



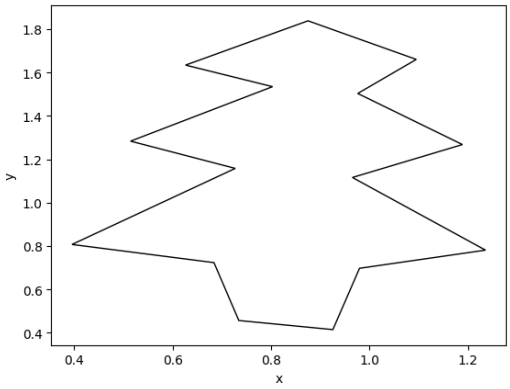
Rysunek 4: Figura A: wielokąt wypukły przypominający okrąg, zawiera 11 wierzchołków



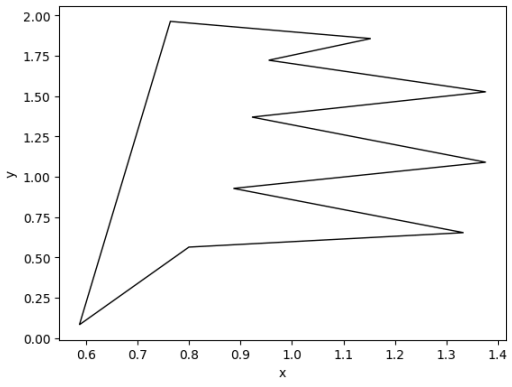
Rysunek 5: Figura B: posiada jeden kąt wklęsły, zawiera 8 wierzchołków



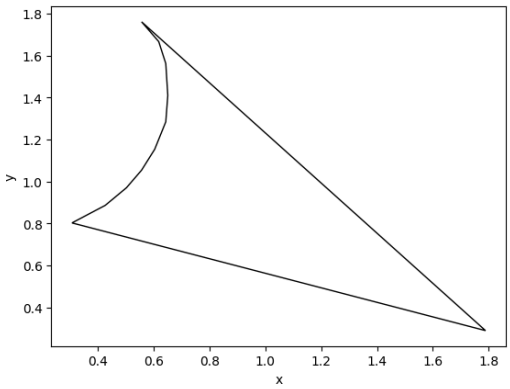
Rysunek 6: Figura C: posiada wiele kątów wklęsłych na lewym łańcuchu, przypominających „kolce jeża", zawiera 13 wierzchołków



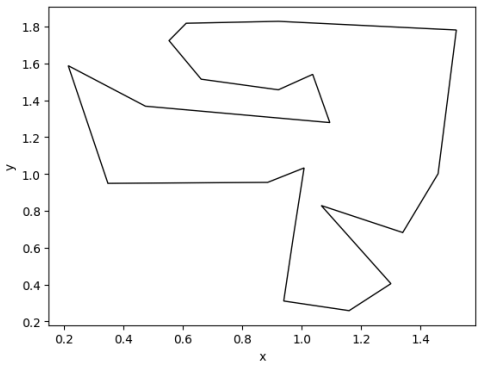
Rysunek 7: Figura D: wielokąt wklęsły przypominający choinkę, 15 wierzchołków



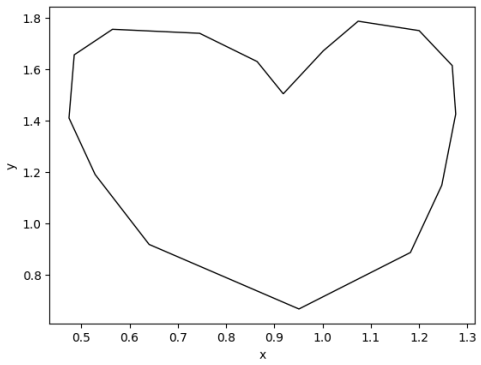
Rysunek 8: Figura E: przypomina figurę C, ale posiada kąty wklęsłe na prawym łańcuchu oraz skrajnie oddalony punkt o minimalnej współrzędnej y, 10 wierzchołków



Rysunek 9: Figura F: przypomina kształtem niepełny trójkąt z zaokrąglonym górnym lewym rogiem (łuk), zawiera 11 wierzchołków



Rysunek 10: Figura G: dowolny wielokąt wklęsły, który nie jest y-monotoniczny, 20 wierzchołków



Rysunek 11: Figura H: serce, nie jest y-monotoniczne, zawiera 16 wierzchołków

5. Analiza wyników

5.1. Sprawdzenie y-monotoniczności

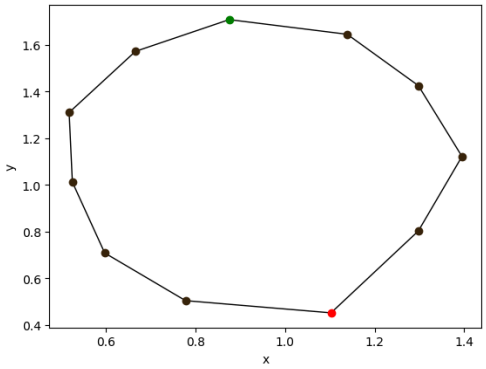
Figura	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Czy y-monotoniczna?	True	True	True	True	True	True	False	False

Tabela 1: Wyniki zwrócone przez funkcję sprawdzającą y-monotoniczność dla zadanych figur

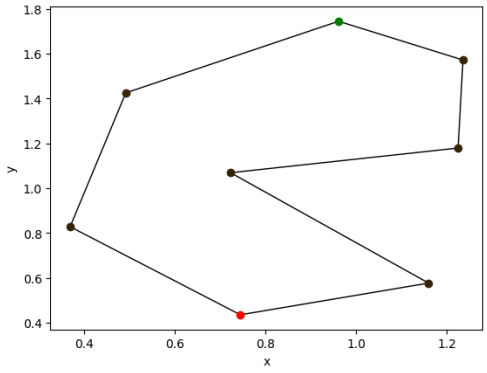
Jak można zauważyć w Tabeli 1, figury zawarte na Rysunkach 4, 5, 6, 7, 8, 9 są y-monotoniczne, a te na Rysunkach 10 i 11 nie są. Wyniki algorytmu są zgodne z wcześniejszymi oczekiwaniami.

5.2. Algorytm klasyfikacji wierzchołków wielokątów

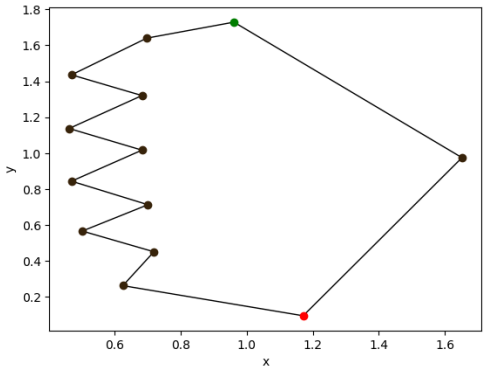
Na poniższych rysunkach zamieszczone zostały wizualizacje wielokątów testowych zwrócone przez algorytm, wraz z wierzchołkami pokolorowanymi zgodnie z przyjętą w Sekcji 3.2 konwencją.



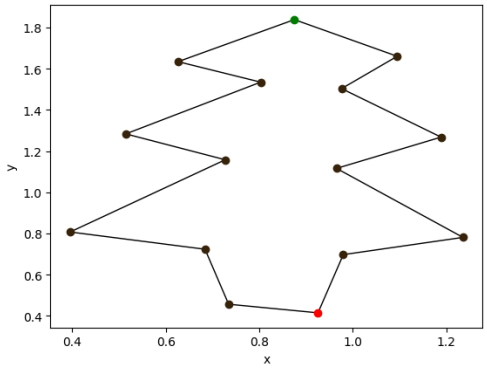
Rysunek 12: Wizualizacja klasyfikacji dla fig. A



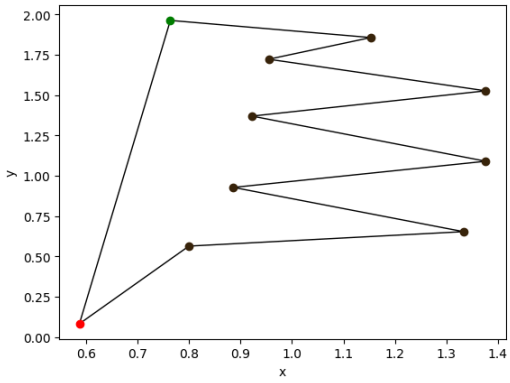
Rysunek 13: Wizualizacja klasyfikacji dla fig. B



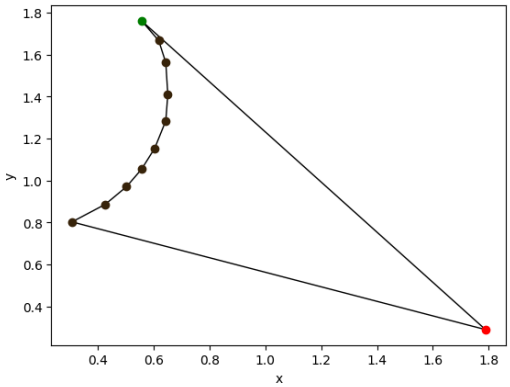
Rysunek 14: Wizualizacja klasyfikacji dla fig. C



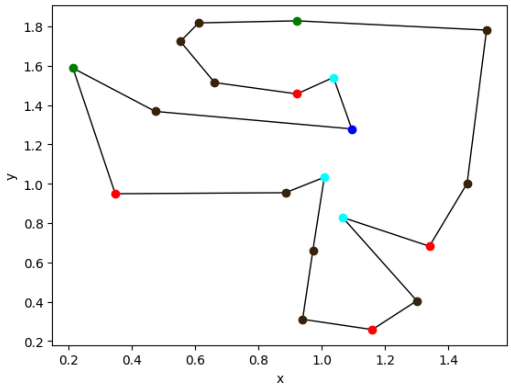
Rysunek 15: Wizualizacja klasyfikacji dla fig. D



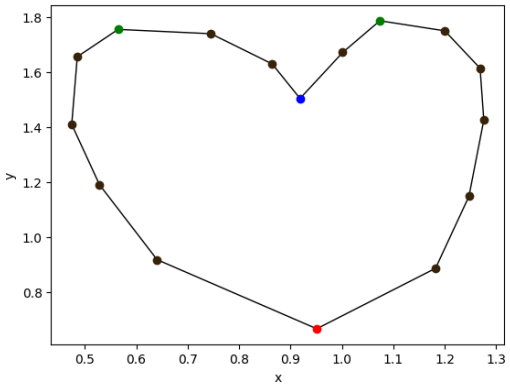
Rysunek 16: Wizualizacja klasyfikacji dla fig. E



Rysunek 17: Wizualizacja klasyfikacji dla fig. F



Rysunek 18: Wizualizacja klasyfikacji dla fig. G



Rysunek 19: Wizualizacja klasyfikacji dla fig. H

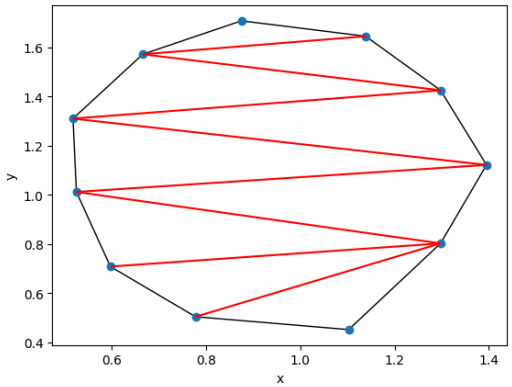
Jak widoczne jest na Rysunkach 12, 13, 14, 15, 16, 17, wielokąty y-monotoniczne posiadają jeden wierzchołek początkowy i jeden końcowy, a pozostałe są klasyfikowane jako wierzchołki prawidłowe. Jest to zgodne z twierdzeniem mówiącym o tym, że wielokąt y-monotoniczny nie posiada wierzchołków łączących i dzielących. Posiadają je natomiast figury, które nie są y-monotoniczne (Rysunek 18 oraz Rysunek 19). Wyniki przedstawione powyżej, zgodnie z oczekiwaniami, pokazują, że algorytm skutecznie identyfikuje wierzchołki kluczowe dla klasyfikacji.

5.3. Algorytm triangulacji wielokątów y-monotonicznych

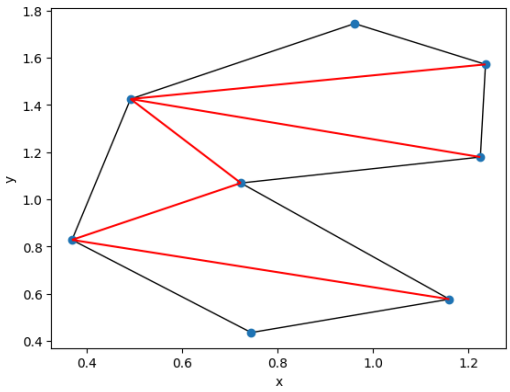
Dla zadanych wcześniej zbiorów testowych uruchomiono zaimplementowany algorytm triangulacji. Wykonano to także dla wielokątów, które nie są y-monotoniczne, w celu pokazania, że nie będzie on poprawnie funkcjonował bez spełnienia tego warunku. Jego wykorzystanie w tym przypadku wymagałoby podziału takich wielokątów na takie, które są y-monotoniczne.

Na zamieszczonych poniżej wizualizacjach, kolorem czarnym zaznaczono krawędzie będące bokami wielokąta, kolorem niebieskim jego wierzchołki, a na czerwono oznaczono dodane w celu triangulacji przekątne.

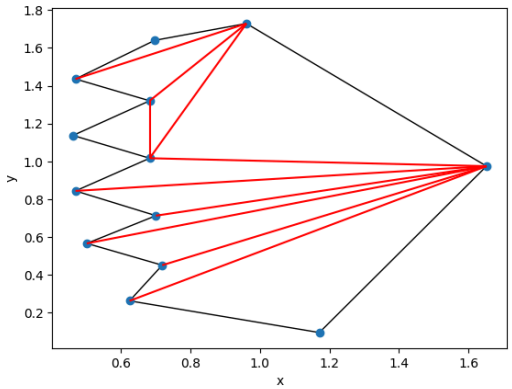
Wynikowe wielokąty po triangulacji oraz wizualizacje funkcjonowania algorytmu krok po kroku, zostały wypisane i zamieszczone w pliku *Jupyter* wraz z kodem źródłowym.



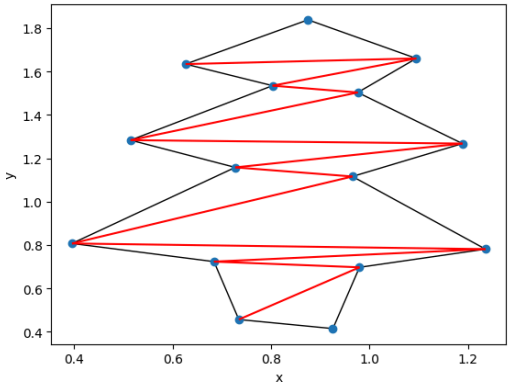
Rysunek 20: Triangulacja figury A



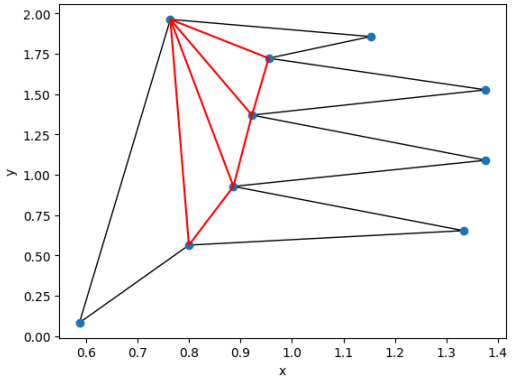
Rysunek 21: Triangulacja figury B



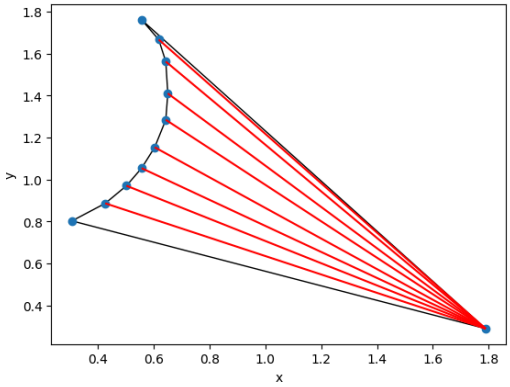
Rysunek 22: Triangulacja figury C



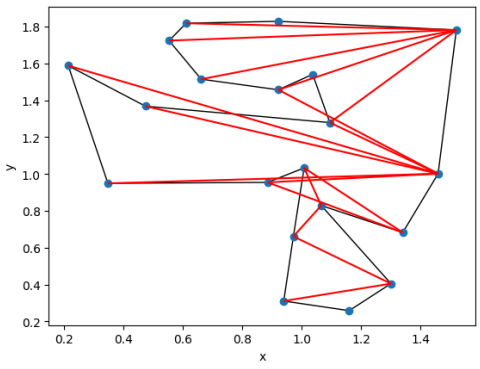
Rysunek 23: Triangulacja figury D



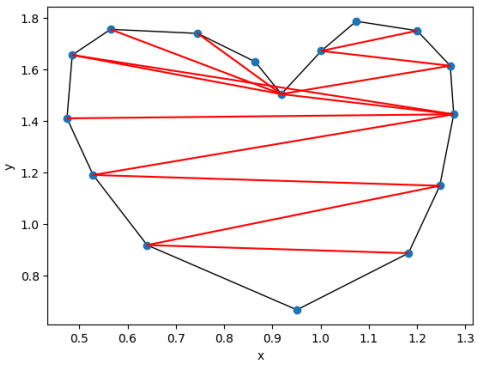
Rysunek 24: Triangulacja figury E



Rysunek 25: Triangulacja figury F



Rysunek 26: Triangulacja figury G
(nie jest y-monotoniczna)



Rysunek 27: Triangulacja figury H
(nie jest y-monotoniczna)

Jak wynika z wcześniejszych przewidywań, wykorzystanie wyżej opisanych algorytmów nie jest efektywną metodą triangulacji wielokątów, które nie są y-monotoniczne. Na Rysunku 26 oraz 27 widać, że przekątne nie tylko tworzą trójkąty zewnętrzne, ale również przecinają boki figur. Wynika to z faktu, że dla niespełnionego przypadku monotoniczności, nie działa poprawnie funkcja rozdzielająca wierzchołki na dwa łańcuchy, a w efekcie także funkcja je scalająca. Algorytm triangulacji nie jest również odpowiednio przystosowany do takich figur. Wymagałyby one m.in. wcześniejszego podziału na wielokąty y-monotoniczne.

Przypadki dla wielokątów y-monotonicznych opisano poniżej.

Figura A przedstawia wielokąt wypukły z punktami równomiernie rozmieszczonymi na obu łańcuchach (lewym i prawym). W procesie triangulacji kolejno przetwarzane wierzchołki zawsze należą do przeciwnego łańcucha niż dwa punkty znajdujące się na szczycie stosu, co uniemożliwia powstanie trójkątów zewnętrznych. Triangulacja widoczna jest na Rysunku 20.

W figurze B występuje jeden kąt wklęsły ulokowany na prawym łańcuchu. Podczas triangulacji tego wielokąta (Rysunek 21) sprawdzane jest, czy odcinek stanowiący jego bok nie zostanie niepotrzebnie dodany do listy przekątnych.

Triangulacja figury C (Rysunek 22) miała na celu sprawdzenie, jak algorytm poradzi sobie z wieloma kątami wklęsłymi znajdującymi się na jednym z łańcuchów, podczas gdy pozostałe jego kąty są wypukłe. Algorytm w sposób właściwy zarządzał obsługą stosu oraz prawidłowo reagował na trójkąty wykraczające poza obszar wielokąta lub na odcinki współliniowe z jego bokami.

W przypadku figury D (Rysunek 23), algorytm musiał poradzić sobie z częstym usuwaniem punktów ze stosu. Powodem tego było przetwarzanie wierzchołków należących do różnych łańcuchów.

Triangulacja figury E (Rysunek 24) jest podobna do tej przeprowadzonej dla wielokąta C. Różnica polega na tym, że tym razem sprawdzane jest funkcjonowanie stosu dla wielu punktów z prawego łańcucha. Ponadto wierzchołek o minimalnej wartości y jest skrajnie od nich oddalony.

Figura F (Rysunek 25), przypominająca wachlarz, bada przypadek, kiedy wszystkie wierzchołki poza jednym należą do tego samego łańcucha. W trakcie przeprowadzania triangulacji, są one kolejno dodawane na stos. Algorytm musi w tym czasie poradzić sobie z tym, aby poprawnie odrzucać trójkąty zewnętrzne. Wszystkie wierzchołki z łuku ostatecznie tworzą przekątne z jedynym punktem umiejscowionym na prawym łańcuchu.

Figura	**A**	**B**	**C**	**D**	**E**	**F**
Liczba wierzchołków	11	8	13	15	10	11
Liczba dodanych przekątnych	8	5	10	12	7	8

Tabela 2: Porównanie liczby dodanych przekątnych w trakcie triangulacji wielokątów y-monotonicznych

Jak można zauważyć w Tabeli 2, liczba dodanych przekątnych spełnia zależność opisaną w Sekcji 3.3 i wynosi $n-3$, gdzie n jest liczbą wierzchołków wielokąta. Algorytm wyznaczył poprawne triangulacje dla tych figur.

6. Wnioski

1. Implementacja algorytmów sprawdzania y-monotoniczności, klasyfikacji wierzchołków oraz triangulacji wykazała zgodność z założeniami teoretycznymi. Wyniki dla zbiorów testowych potwierdzają, że algorytmy poprawnie identyfikują y-monotoniczność wielokątów oraz klasyfikują ich wierzchołki zgodnie z przyjętą konwencją.
2. Triangulacja wielokątów y-monotonicznych została przeprowadzona poprawnie we wszystkich przypadkach, co dowodzi, że założenie monotoniczności pozwala na zastosowanie wydajnego algorytmu o złożoności $O(n)$. Wyniki te pokazują, że y-monotoniczność znacząco ułatwia proces triangulacji.
3. Algorytmy nie są bezpośrednio przystosowane do wielokątów, które nie spełniają warunku y-monotoniczności. Przykłady takie jak figury G i H wskazują, że w takich przypadkach konieczny jest m.in. wcześniejszy podział wielokątów na części monotoniczne. Próba zastosowania algorytmu bez tego kroku prowadzi do powstawania błędnych przekątnych.
4. Wykorzystanie stosu w procesie triangulacji okazało się efektywne w eliminowaniu potencjalnych błędów związanych z przecinaniem się przekątnych. Algorytm poprawnie identyfikował odpowiednie trójkąty, co zmniejszało ryzyko powstawania problematycznych konfiguracji.
5. Liczba dodanych przekątnych dla wielokątów y-monotonicznych w pełni zgadzała się z teoretyczną zależnością $n - 3$, co potwierdza poprawność zaimplementowanego algorytmu.

Podsumowując, realizacja ćwiczenia pozwoliła na skuteczne zapoznanie się z problematyką triangulacji wielokątów y-monotonicznych oraz udowodniła efektywność zaimplementowanych algorytmów w zadanych warunkach. Uzyskane rezultaty potwierdzają założenia teoretyczne i są zgodne z oczekiwaniami.