Sprawozdanie z ćwiczenia trzeciego

Algorytmy geometryczne

Jakub Pisarek
Informatyka, WIEiT, AGH

23 listopada 2022

1 Cel ćwiczenia

Celem ćwiczenia jest implementacja algorytmu sprawdzania, czy dany wielokąt jest monotoniczny, algorytmu klasyfikacji wierzchołków w dowolnym wielokącie oraz algorytmu triangulacji wielokąta monotonicznego. Algorytmy te należy następnie przetestować na różnych zestawach danych.

2 Wprowadzenie

Wielokąt prosty jest monotoniczny względem prostej l, gdy przecięcie dowolnej prostej l' prostopadłej do l z dowolnym łańcuchem jest spójne, tzn. jest odcinkiem, punktem lub jest puste.

Wierzchołki wielokąta prostego można sklasyfikować w następujący sposób:

- 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.

Triangulacji wielokąta monotonicznego można dokonać za pomocą prostego algorytmu zachłannego, który nie będzie tu przytaczany.

Należy przyjąć, że wielokąty zawsze zadawane są przeciwnie do ruchu wskazówek zegara.

Podczas działania algorytmów klasyfikacji wierzchołków i triangulacji wielokąta monotonicznego wymagane jest uszeregowanie punktów względem kąta, który wyznaczają z pewną prostą. Do celu tego można użyć wyznacznika odpowiedniej macierzy, analogicznie jak w poprzednich ćwiczeniach.

3 Przebieg ćwiczenia

1. Dostosować aplikację graficzną tak, aby można było zadawać proste wielokąty (w kierunku przeciwnym do ruchu wskazówek zegara) przy użyciu myszki, z dodatkowym zapisem i odczytem podanych wielokątów,

2. Zaimplementować (a następnie krótko opisać) funkcję sprawdzającą, czy podany wielokąt jest y-monotoniczny,
3. Zaimplementować algorytm, który dla zadanego wielokąta będzie klasyfikował jego wierzchołki jak w sekcji 2. Wierzchołki mają zostać odpowiednio pokolorowane zgodnie z klasyfikacją,
4. Zaimplementować procedurę triangulacji wielokąta monotonicznego zgodnie z algorytmem opisanym na wykładzie. Program powinien pokazywać kolejne kroki algorytmu, tzn. kolejne tworzone trójkąty,
5. Opisać, jak zaimplementowano struktury przechowujące wielokąt oraz utworzoną triangulację, uzasadnić wybór struktur,
6. Przetestować programy na różnych zestawach danych. Napisać, jakiego typu zestawy testowano, uzasadnić ich wybór.

4 Środowisko programistyczne

- System operacyjny Ubuntu 20.04.5 LTS x86_64,
- Procesor Intel Core i5-3320M 3.30 GHz,
- Python 3.8.10 z zewnętrzną biblioteką `matplotlib`,
- Środowisko Jupyter Notebook zintegrowane z edytorem Visual Studio Code.

5 Szczegóły implementacji algorytmów

5.1 Sprawdzanie y-monotoniczności wielokąta

Znaleziono wierzchołki o największej i najmniejszej współrzędnej y (jeżeli takich wierzchołków było więcej, przyjmowano pierwsze znalezione, co nie wpływa na poprawność działania funkcji). Następnie, dokonano dwukrotnie przejścia po kolejnych wierzchołkach wielokąta, od położonego najwyżej do najniżej, idąc po liście wierzchołków w jedną i drugą stronę. Jeżeli w którymkolwiek momencie wykryto, że następny w kolejności wierzchołek leży wyżej niż aktualnie rozważany, zwracano wartość `False`, jeżeli zaś nic takiego nie nastąpiło – `True`.

5.2 Klasyfikacja wierzchołków wielokąta

Dla każdego wierzchołka sprawdzano, czy leży wyżej/niżej od obydwu wierzchołków sąsiadujących, jak również kąt, jaki z nimi tworzy. Do tego celu użyto wyznacznika odpowiedniej macierzy 3x3 w implementacji własnej, przyjmując tolerancję dla zera $\varepsilon = 10^{-12}$.

Punkty klasyfikowano zgodnie z opisem z sekcji 2.

5.3 Triangulacja wielokąta y-monotonicznego

Wierzchołki posortowano względem współrzędnej y za pomocą dostępnej w bibliotece standardowej Pythona funkcji `sorted()`, korzystającej z algorytmu Timsort – hybrydy merge sort i insertion sort.

Jako stosu użyto kontenera `collections.deque` z biblioteki standardowej Pythona.

Do wyznaczania, czy rozważany trójkąt leży wewnątrz wielokąta, użyto ponownie wyznacznika odpowiedniej macierzy 3x3 w implementacji własnej, przyjmując tolerancję dla zera $\varepsilon = 10^{-12}$.

Funkcja była zabezpieczona przed podaniem jej wielokąta, który nie byłby y-monotoniczny, wywoływała bowiem na początku funkcję z sekcji 5.1., i jeżeli ta zwróciła wartość `False`, kończyła swoje działanie.

6 Użyte struktury danych

Definicja wielokąta przechowywana była jako lista dwuelementowych krotek `(float, float)`, z których każda reprezentowała jeden wierzchołek (wierzchołki były zadawane w kierunku przeciwnym do ruchu wskazówek zegara). Struktura ta pozostawała niezmienna.

Gdy pojawiała się konieczność uszeregowania wierzchołków w innej kolejności, np. względem współrzędnej y, tworzono pomocniczą listę o tym samym rozmiarze, w której przechowywano indeksy wierzchołków z listy wyjściowej w ustalonej kolejności.

Klasyfikację wierzchołków przechowywano jako listę o tym samym rozmiarze, zawierającą odpowiednio zdefiniowane wartości typu wyliczeniowego `enum.Enum` z biblioteki standardowej Pythona.

Triangulację zwracano jako listę trójelementowych krotek `(int, int, int)`, z których każda reprezentowała jeden trójkąt z triangulacji, przechowując indeksy wierzchołków wielokąta wyjściowego składające się na dany trójkąt (wierzchołki trójkątów również były zadawane wyłącznie w kierunku przeciwnym do ruchu wskazówek zegara).

Wybór struktur pomocniczych typu lista, przechowujących indeksy wierzchołków, jest optymalny, gdyż zabiera minimum dodatkowej pamięci, pozwalając przy tym na operacje dostępu do danego wierzchołka w czasie $O(1)$. Operowanie na samej definicji wielokąta powodowałoby zbędne kopiowanie większej ilości danych, niż to konieczne, spowalniając tym samym wykonanie programu i zużywając więcej pamięci.

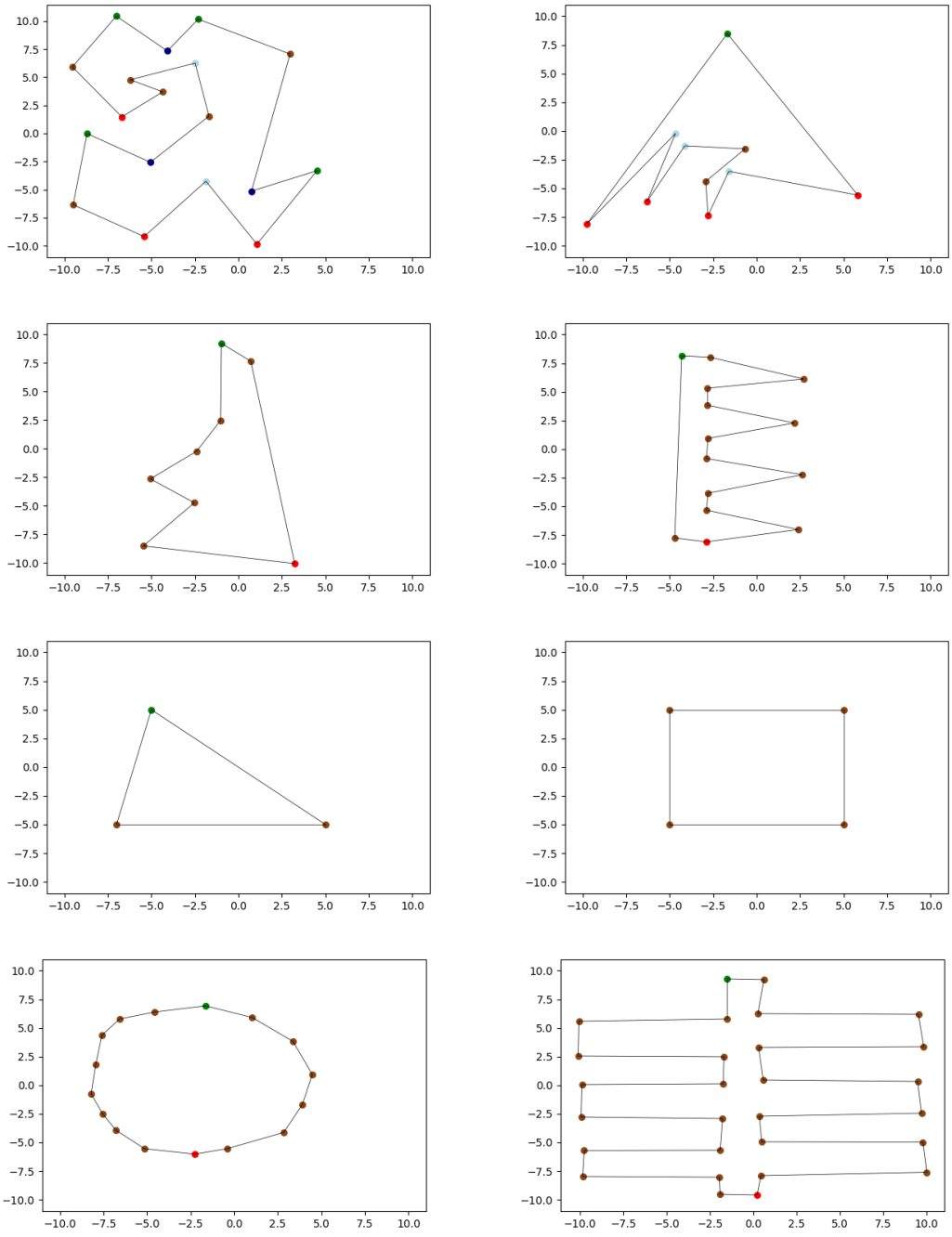
Co prawda, trzymanie klasyfikacji wierzchołków w jednej liście sprawia, że podczas rysowania wizualizacji klasyfikacji wymagane jest kilkukrotne przejście po tejże liście – nie spowodowało to jednak zauważalnego opóźnienia; gdyby nawet opóźnienie wystąpiło, można byłoby mu łatwo zaradzić przechodząc raz przez listę i tworząc kilka nowych, na etapie rysowania.

Przechowywanie triangulacji jako listy trójelementowych krotek zawierających indeksy wierzchołków ma wiele zalet, z których największą jest jednoznaczny sposób wymieniania wszystkich trójkątów, na które wielokąt został podzielony. Jako wadę można wskazać natomiast fakt, że użycie listy dwuelementowych krotek zawierających indeksy wyłącznie tych wierzchołków, dla których triangulacja „dodała" odcinek je łączący, pozwoliłoby na zaoszczędzenie pamięci – nie duplikowałoby w opisie krawędzi wielokąta, co pozwoliłoby przekazać taką triangulację bezpośrednio do narzędzia rysującego. Znalezienie wszystkich trójkątów z takiej postaci triangulacji nie byłoby jednak trywialnym zadaniem; wyeliminowanie krawędzi wielokąta z pierwszej wymienionej postaci nie jest problemem, wystarczy, rozważając wszystkie utworzone trójkąty, odrzucać te ich boki, które zostały stworzone między sąsiednimi wierzchołkami wielokąta. Było to głównym powodem, dla którego zdecydowano się na opcję wcześniejszą.

7 Wyniki

Programy testowano na wielokątach, których wizualizacje są przedstawione na rycinach poniżej. Zaznaczono tam również klasyfikację wierzchołków, dla której użyty schemat kolorystyczny jest następujący:

- wierzchołki początkowe – kolor zielony,
- wierzchołki końcowe – kolor czerwony,
- wierzchołki łączące – kolor granatowy,
- wierzchołki dzielące – kolor jasnoniebieski,
- wierzchołki prawidłowe – kolor brązowy.



Ryc. 1-8: Testowane wielokąty wraz z klasyfikacją wierzchołków

Wybór takich wielokątów uzasadniono następująco:

1-2. Wielokąty inspirowane materiałami wykładowymi (test klasyfikacji wierzchołków),

3-4. Wielokąty y-monotoniczne inspirowane materiałami wykładowymi (test triangulacji, przypadek pesymistyczny dla liczby kamer w problemie monitorowania galerii),

5-6. Przypadki zdegenerowane – trójkąt, kwadrat (test triangulacji dla przypadku granicznego, dla wierzchołków o tej samej współrzędnej y),

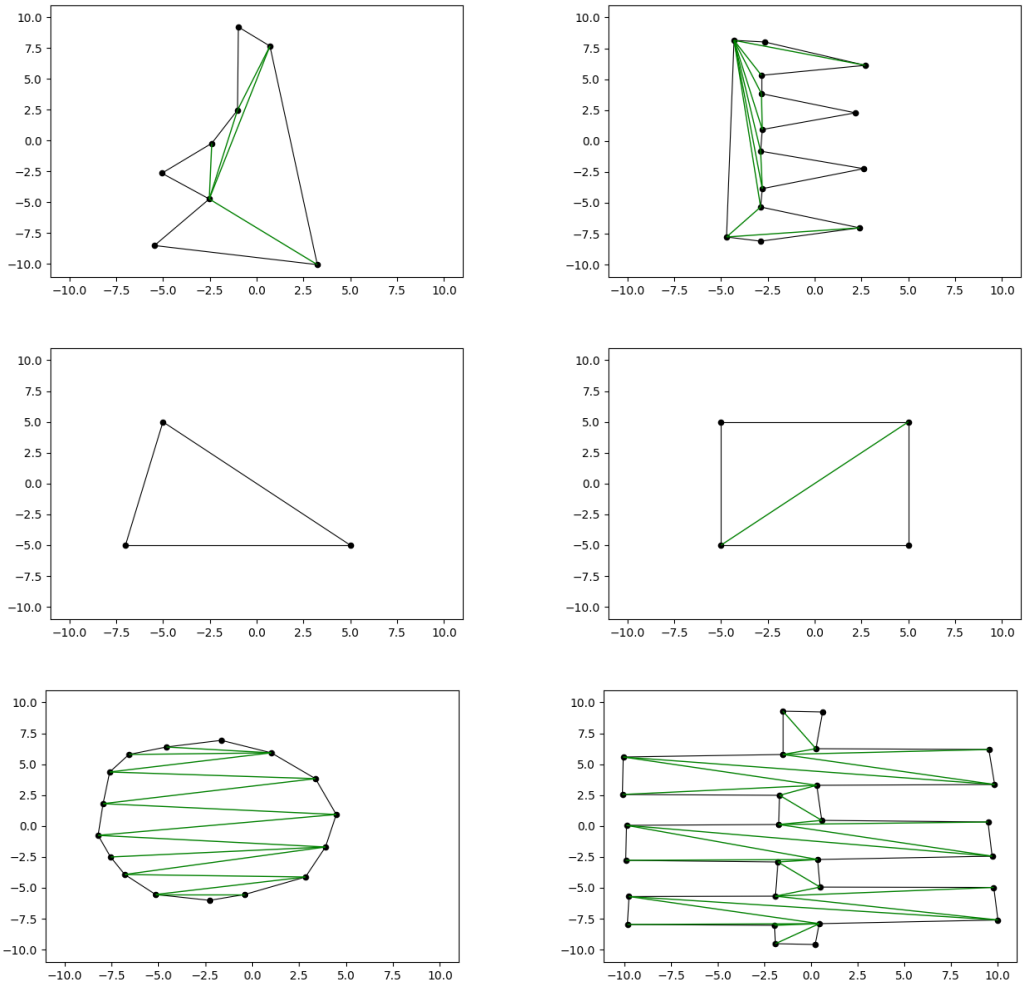
7. Wielokąt wypukły (test triangulacji przy większej liczbie wierzchołków),

8. Jak punkt 4. (test triangulacji przy wielu kątach wklęsłych w wielokącie).

Funkcja sprawdzania y-monotoniczności wielokąta zwróciła poprawne wyniki we wszystkich przypadkach, tzn. wartość `False` dla pierwszych dwóch wielokątów i `True` dla pozostałych.

Jak łatwo zauważyć z powyższych rycin, funkcja klasyfikacji wierzchołków również zwróciła poprawne wyniki we wszystkich przypadkach, co pozwala się upewnić o bezbłędności jej implementacji.

Otrzymane triangulacje (wielokątów y-monotonicznych, tj. wszystkich oprócz pierwszych dwóch) przedstawiono na poniższych wizualizacjach:



Ryc. 9-14: Otrzymane triangulacje testowanych wielokątów y-monotonicznych

Pełne wizualizacje kroków działania algorytmu triangulacji dostępne są w załączonym notebooku Jupyter. Krawędzie już obecne w triangulacji zaznaczono tam kolorem niebieskim, trójkąt aktualnie rozważany – kolorem czerwonym. Wierzchołki aktualnie obecne na stosie również wyróżniono kolorem czerwonym.

Nie stwierdzono błędów podczas wyznaczania żadnej z wyżej wymienionych triangulacji.

8 Wnioski

- Sprawdzenie monotoniczności wielokąta jest trywialne w implementacji,
- Klasyfikacja wierzchołków wielokąta także,
- Triangulacja wielokąta monotonicznego również jest stosunkowo prosta do wykonania przy użyciu nieskomplikowanego algorytmu zachłannego.