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

Algorytmy geometryczne, sprawozdanie – ćwiczenie 4 - 30.11.2023 r.

1. Opis ćwiczenia

Głównym celem ćwiczenia było zaimplementowanie algorytmu zamiatania, za pomocą którego można wyznaczać przecięcia odcinków na płaszczyźnie.

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 biblioteki: numpy, matplotlib oraz sortedcontainers.

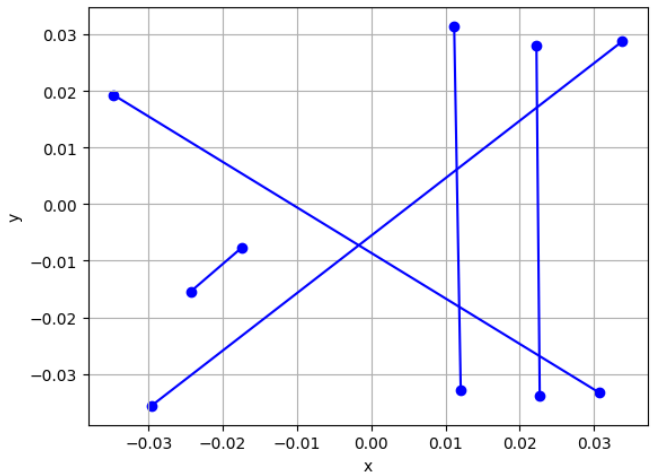
3. Przebieg ćwiczenia

1) Stworzenie funkcji generującej zadaną liczbę losowych odcinków z podanego zakresu
2) Stworzenie funkcji, która umożliwia zadawanie odcinków z myszki
3) Napisanie algorytmu, który wykrywa, czy występuje choć 1 przecięcie
4) Rozszerzenie poprzedniego algorytmu do wykrywania wszystkich przecięć

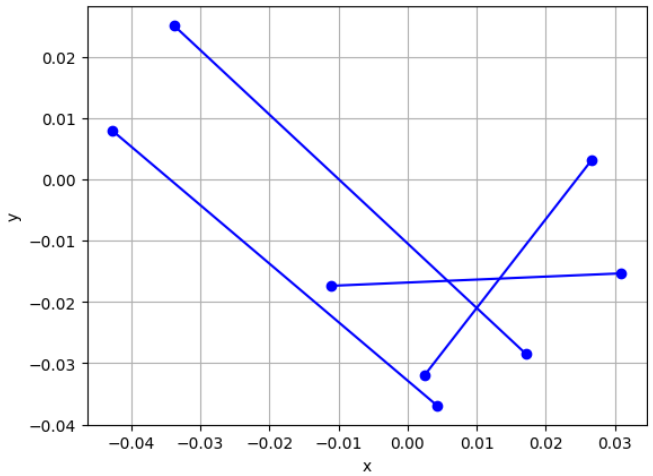
4. Tworzenie odcinków

Do generowania losowych odcinków wykorzystałem funkcję random.uniform z biblioteki numpy. Podczas generowania zostały wykluczone przypadki, gdzie generowane były odcinki pionowe. Uwzględnione również zostało, aby każda współrzędna x była unikalna.
Do stworzenia funkcji umożliwiającej zadawanie odcinków z myszki posłużyłem się biblioteką matplotlib. Otwiera ona okienko, na którym należy klikać lewym przyciskiem myszy w miejsca, w których chcemy, aby były nasze kolejne końce odcinków. Co 2 zadane punkty, automatycznie generowany jest odcinek łączący te 2 punkty. Istnieje możliwość zapisania całego wygenerowanego widoku, jeśli klikniemy w lewym dolnym rogu na ikonkę zapisu.
W dalszej części sprawozdania posłużę się następującymi zestawami danych:

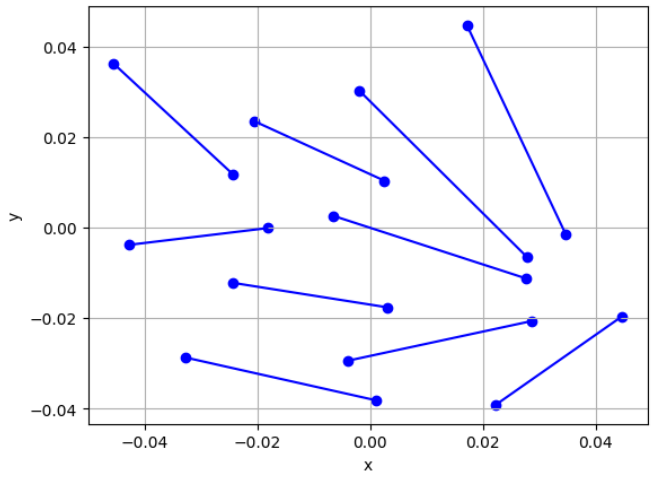
Rysunek 1 – zestaw danych 1



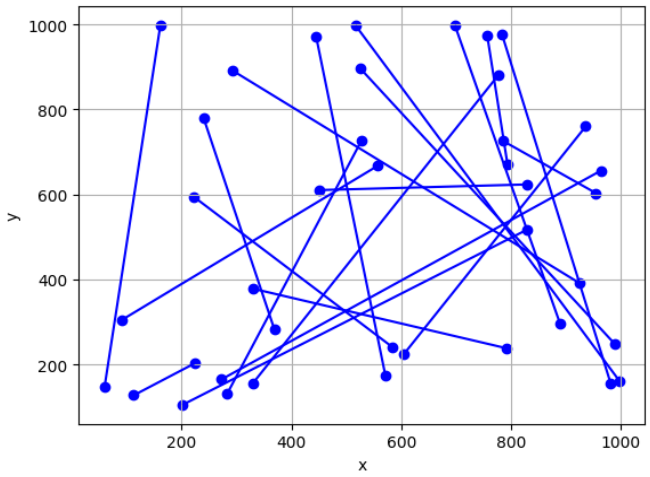
Rysunek 2 – zestaw danych 2



Rysunek 3 – zestaw danych 3



Rysunek 4 – zestaw danych 4



Zestaw 1 został wybrany ze względu na to, że podczas wykrywania wszystkich przecięć, przecięcie na samym środku rysunku zostanie wykryte więcej niż jeden raz. Jest to przykład, na którym można sprawdzić działanie pomijania duplikatów.
Zestaw 2 został natomiast wybrany, gdyż sprawdza on poprawność działania struktury stanu i jego sortowania.
Zestaw 3 nie zawiera żadnych przecięć, co jest skrajnym przypadkiem. W przypadku złej implementacji wykrywania przecięć, mogłyby tutaj pojawić się wykryte punkty, które nie istnieją.
Zestaw 4 składa się z 20 losowo wygenerowanych odcinków, więc jest to przypadek ogólny.

5. Struktury danych

W obydwóch algorytmach strukturą stanu jest SortedSet z biblioteki sortedcontainers. Używam w niej elementów zaimplementowanej przeze mnie klasy Line, która przeciąża operator porównywania, dzięki czemu można wyszukiwać odcinki sąsiadujące względem zadanego punktu w czasie logarytmicznym.
Strukturą zdarzeń jest natomiast SortedList z tej samej biblioteki. Przechowuję w niej krotkę składającej się z punktu zdarzenia oraz linii, na której leży dany punkt (lub krotki linii jeśli punkt jest przecięciem). Lista ta sortowana jest po współrzędnych x punktów, które są klasą Point i tak jak Line, posiadają przeciążony operator porównywania.
W mojej implementacji nie była wymagana zmiana żadnej struktury przy algorytmie wykrywania pojedynczych i wszystkich przecięć.

6. Wyznaczanie punktów przecięć

Aby wyznaczyć punkty przecięć, posługuję się parametryczną postacią linii. Obliczam parametry t_1 i t_2 ze wzorów:

$$t_1 = \frac{\Delta X_{AC}\Delta Y_{CD} - \Delta Y_{AC}\Delta X_{CD}}{\Delta X_{AB}\Delta Y_{CD} - \Delta Y_{AB}\Delta X_{CD}} \qquad t_2 = \frac{\Delta X_{AC}\Delta Y_{AB} - \Delta Y_{AC}\Delta X_{AB}}{\Delta X_{AB}\Delta Y_{CD} - \Delta Y_{AB}\Delta X_{CD}}$$

Gdzie A i B to współrzędne końców jednego odcinka, a C i D to współrzędne końców drugiego odcinka. Odcinki klasyfikowane jako przecinające się są w przypadku, gdy t_1 i $t_2 \in (0, 1)$. Przypadki, kiedy parametry wynoszą 0 lub 1 oznaczają, że jeden koniec odcinka leży na drugim odcinku. Uznałem, że nie jest to sytuacja oznaczająca przecinanie się odcinków.

Współrzędne przecięcia wyznaczane są ze wzoru:
$X_p = X_A + t_1 * \Delta X_{AB}$
$Y_p = Y_A + t_1 * \Delta Y_{AB}$

7. Obsługa zdarzeń

Zdarzenia podzielone mogą zostać na 3 rodzaje: początek odcinka, koniec odcinka i punkt przecięcia. Każdorazowo, jeśli rozważanym aktualnie punktem jest dane wydarzenie, wykonywane są pewne kroki zależne od typu właśnie tego zdarzenia.

a) Początek odcinka

W tym przypadku porównywana jest odcinek, na którym leży dany punkt z jej sąsiadami (według współrzędnej y). Porównywanie polega na obliczeniu parametrów t_1 i t_2 wcześniej zdefiniowanych oraz wyznaczeniu współrzędnych przecięcia, jeśli takie istnieją. Po wykryciu przecięcia, wrzucane jest ono do struktury zdarzeń jako potencjalny punkt przecięcia.

b) Koniec odcinka

Jeśli zdarzeniem jest koniec odcinka, to porównywani ze sobą są tylko sąsiedzi. Podobnie jak w poprzednim przypadku, po wykryciu przecięcia, wrzucane jest ono do struktury zdarzeń jako potencjalny punkt przecięcia. Niezależnie od wyniku porównywania odcinków, rozważany odcinek jest usuwany ze struktury stanu.

c) Punkt przecięcia

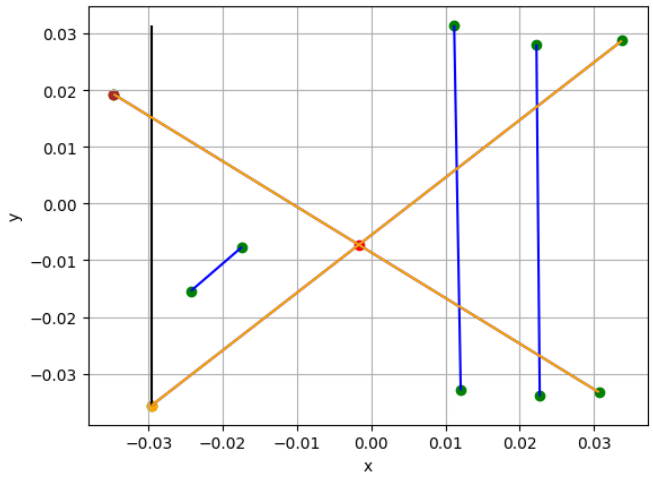
Dla zdarzenia będącego punktem przecięcia, procedura postępowania wygląda następująco: odcinki, do których należy wykryty punkt, są ze sobą zamieniane, a następnie porównywane z ich sąsiadami. W mojej implementacji zostało to rozwiązane w taki sposób, że do współrzędnej x dodaję niewielki epsilon, który spowoduje, że odwrócona zostanie kolejność odcinków (punkt na odcinku, dla którego wszystkie wcześniejsze punkty znajdowały się poniżej punktów drugiego odcinka, będzie teraz znajdował się wyżej). Górny odcinek (czyli taki, którego współrzędna y dla aktualnego x jest większa od drugiego odcinka) porównywany jest z sąsiadem znajdującym się powyżej, natomiast dolny odcinek porównywany jest z sąsiadem poniżej. Jeśli wykryte zostanie przecięcie, jest ono wrzucane do struktury zdarzeń.

7. Algorytm wykrywania pojedynczego przecięcia

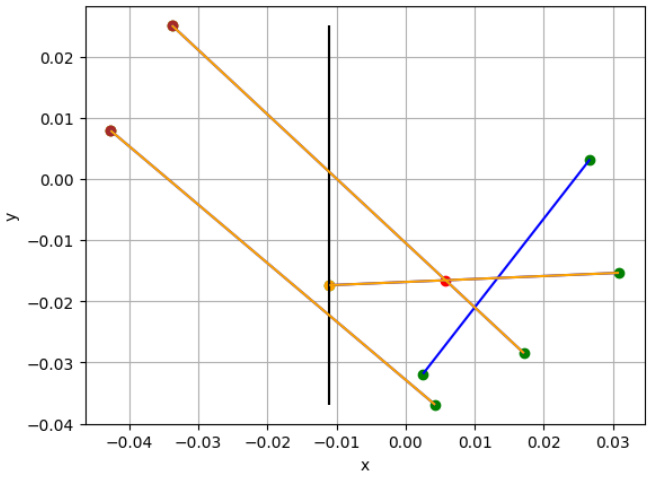
Na początku do struktury zdarzeń dodane zostają wszystkie punkty początków i końców odcinków. W całym algorytmie rozważane są tylko dwa przypadki w strukturze zdarzeń: początku i końca odcinka. Jeśli dla danego zdarzenia wykryte zostanie przecięcie, to program jest kończony i zwraca wartość True. Jeśli struktura zdarzeń będzie pusta, a przecięcie nie zostało wcześniej wykryte, zwracana jest wartość False.
Poniżej przedstawione zostają rysunki obrazujące działanie tego algorytmu.
Legenda:
Kolor niebieski – zadane odcinki
Kolor czarny - miotła
Kolor zielony – punkty znajdujące się w strukturze zdarzeń
Kolor pomarańczowy – linie aktualnie ze sobą porównywane
Kolor brązowy – punkty i linie już przetworzone

Kolor czerwony – wykryte przecięcie

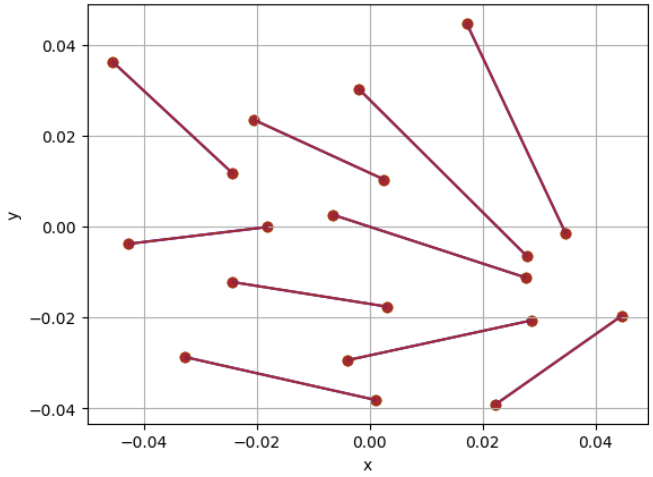
Rysunek 5 – wykryte przecięcie dla zestawu 1



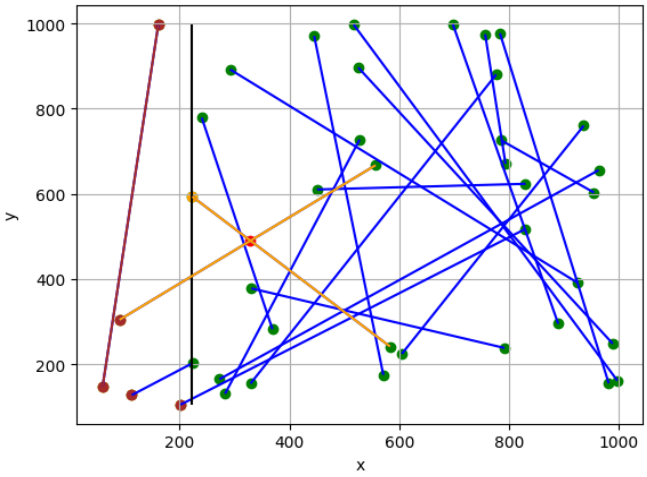
Rysunek 6 – wykryte przecięcie dla zestawu 2



Rysunek 7 – brak przecięcia dla zestawu 3



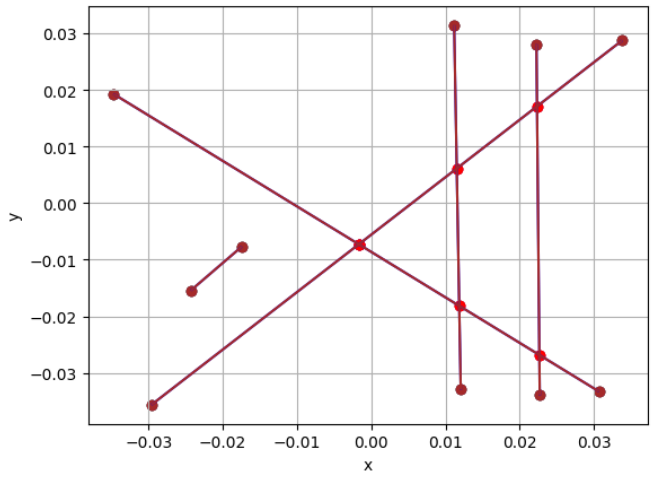
Rysunek 8 – wykryte przecięcie dla zestawu 4



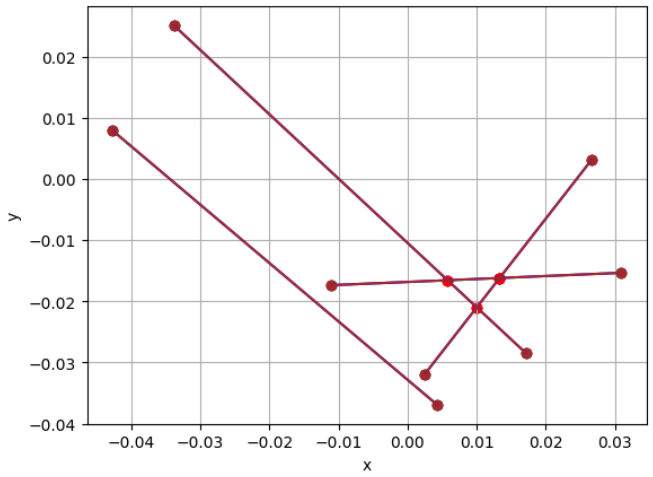
8. Algorytm wykrywania wszystkich przecięć

Podobnie jak w poprzednim przypadku, na początku do struktury zdarzeń dodane zostają wszystkie punkty początków i końców odcinków. Tym razem jednak rozpatrywane będą wszystkie zdarzenia. W tym algorytmie musimy zbadać każdy punkt początku i końca odcinka a także punkty przecięć, dlatego może wystąpić problem powtórzeń – dany punkt będzie wykrywany kilkukrotnie. W tym celu zastosowana zostaje pomocnicza struktura – zbiór. Zawiera ona indeksy odcinków, pomiędzy którymi wykryte zostało przecięcie. Z racji tego, że zbiór umożliwia szybki dostęp do sprecyzowanego elementu, nie zmienia to złożoności programu.

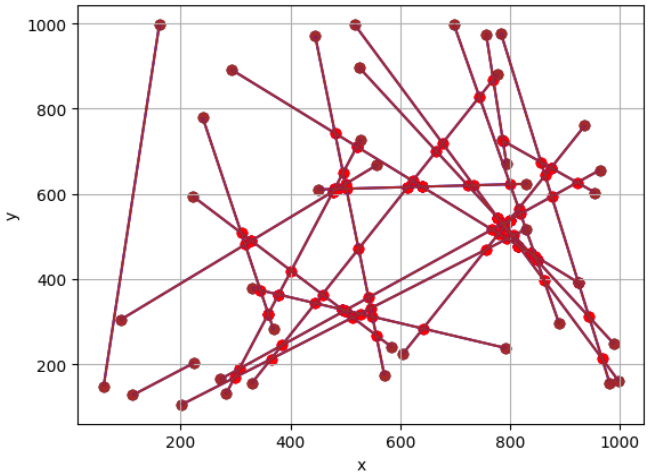
Rysunek 9 – wykryte 5 przecięć dla zestawu 1



Rysunek 10 – wykryte 3 przecięcia dla zestawu 2



Rysunek 11 – wykryte 70 przecięć dla zestawu 4



Wizualizacja dla zestawu 3 jest taka sama jak dla wykrywania pojedynczego przecięcia, dlatego została tu pominięta.

9. Wnioski

Zaimplementowane algorytmy wykrywają przecięcia, jednak w przypadku mojej implementacji nie są one w pełni dokładne. Wykrywane punkty różnią się od wyników przygotowanych testów o liczby rzędu mniejszego niż 10^{-18}. Może to wynikać z niedokładności komputera i jego ograniczonego zakresu zapisu liczb, ponieważ nawet zamiana kolejnością linii podczas obliczania postaci parametrycznej powoduje zmianę w wynikach. Oprócz tego marginesu błędu, który mógłby zostać wyeliminowany poprzez wprowadzenie odpowiedniego epsilonu przy porównywaniu odpowiedzi, wszystko działa poprawnie. Dzięki zastosowanym strukturom duplikaty zostają pominięte, natomiast złożoność algorytmu pozostaje O((n+k)*log n), gdzie n oznacza ilość odcinków, a k oznacza liczbę przecięć.