Algorytmy geometryczne

Sprawozdanie z ćwiczeń 4.

Joanna Kulig, grupa 5

Dane urządzenia, na którym wykonano ćwiczenie:

Procesor: 1,4GHz Czterordzeniowy procesor Intel Core i5
Komputer z systemem macOS Monterey
Środowisko: jupyter notebook
Do rozwiązania ćwiczenia wykorzystano język Python wraz z bibliotekami numpy i matplotlib

1. Cel ćwiczenia.

Celem ćwiczenia była implementacja dwóch algorytmów. Pierwszy z nich miał weryfikować, czy wśród zadanych odcinków istnieją przynajmniej dwa, które się przecinają. Drugi miał wyznaczać wszystkie przecięcia zadanych odcinków. Do realizacji ćwiczenia wykorzystano algorytm zamiatania. Implementacje umożliwiają także wizualizację algorytmów krok po kroku.

2. Zestawy danych.

Każdy zestaw danych można zapisać w pliku .json w postaci listy odcinków. W tym celu stworzono dwie funkcje, które umożliwiają zapis i odczyt danych z pliku. Generowanie zestawów danych można zrealizować na dwa sposoby:

1. Wykorzystując zaimplementowaną funkcję umożliwiającą generowanie odcinków za pomocą funkcji random.uniform, która losowo wybiera współrzędne odcinków z zadanego przedziału.

2. Wykorzystując dostarczone narzędzie graficzne, które przy użyciu myszki umożliwia wprowadzanie kolejnych odcinków.

3. Struktury danych.

Punkty są reprezentowane jako klasa Point(). Klasa ta posiada parametry takie jak:

- Współrzędne: x oraz y,
- Oznaczenie odcinków, na których znajduje się dany punkt.

Odcinki są reprezentowane jako klasa Line(). Klasa ta posiada parametry takie jak:

- Punkt początkowy i końcowy odcinka,
- Współczynnik a i b, które pozwalają na odtworzenie równania prostej (y = ax + b).
- Obecna pozycja miotły

Taki sposób przechowywania informacji umożliwi łatwiejszy i przejrzysty dostęp do danych.
W implementacji algorytmu skorzystano także z następujących struktur danych:

- Struktura zdarzeń Q - kolejka priorytetowa PriorityQueue z biblioteki queue. Wykorzystana struktura umożliwi uporządkowanie względem współrzędnych x-owych.

- Struktura stanu T - SortedSet z biblioteki sortedcontainers.

Dzięki tej strukturze można w łatwy sposób odnaleźć sąsiadów danego elementu. Uporządkowane względem współrzędnych y-owych wyliczonych przy pomocy równania prostej.

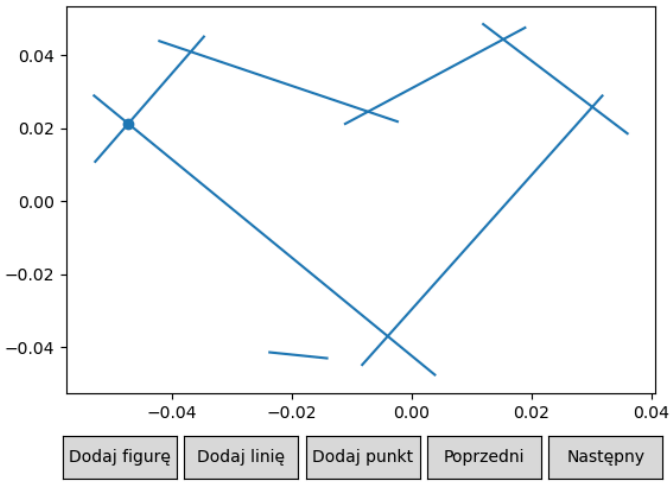
4. Opis algorytmów

Weryfikacja, czy istnieją przynajmniej dwa odcinki, które się przecinają.

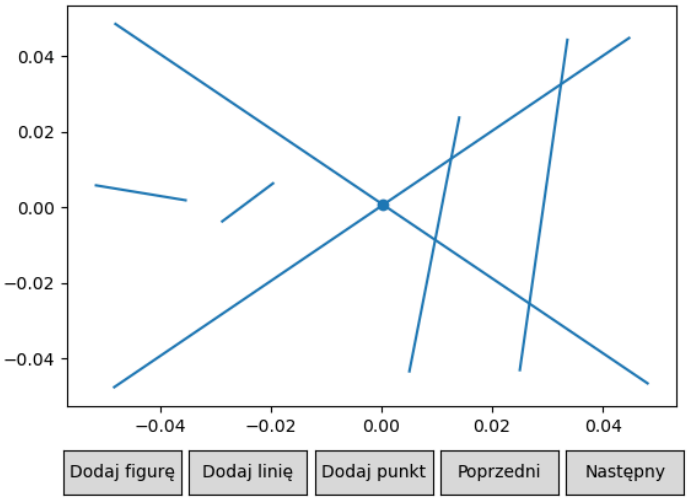
Przebieg algorytmu:

1. Wszystkie zdarzenia zostają dodane do kolejki priorytetowej z odpowiednim oznaczeniem, które później pozwoli zidentyfikować, czy jest to początek, czy koniec odcinka. W tym samym momencie wyszukiwany jest rozmiar miotły.
2. Miotła zostaje ustawiona w pozycji początkowej.
3. Obsługa zdarzeń:
   A. ***Początek odcinka*** - jeśli odcinek ma sąsiadów następuje weryfikacja, czy przecina się z nimi.
   B. ***Koniec odcinka*** - weryfikacja, czy sąsiedzi (jeśli istnieją) usuwanego ze struktury stanu odcinka się przecinają.
   C. Jeśli w którymś przypadku trafiono w przecinające się odcinki, przerywamy pętlę i zwracamy wartość True. W przeciwnym przypadku zwracamy wartość False.
4. Dostępna jest wizualizacja algorytmu krok po kroku.

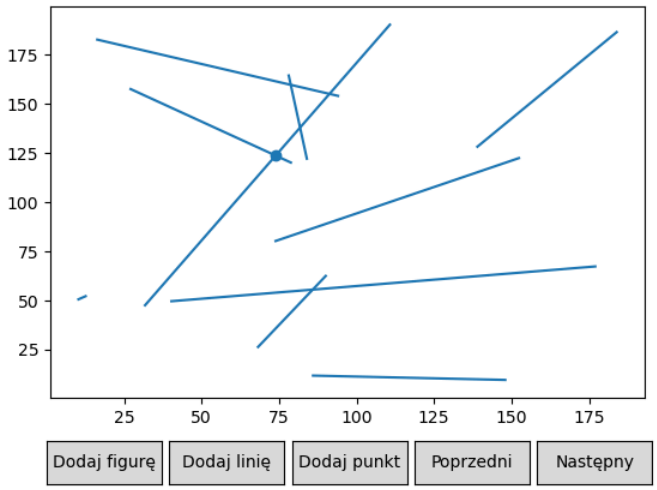
Wynik działania algorytmu weryfikującego, czy istnieją dwa odcinki które się przecinają:



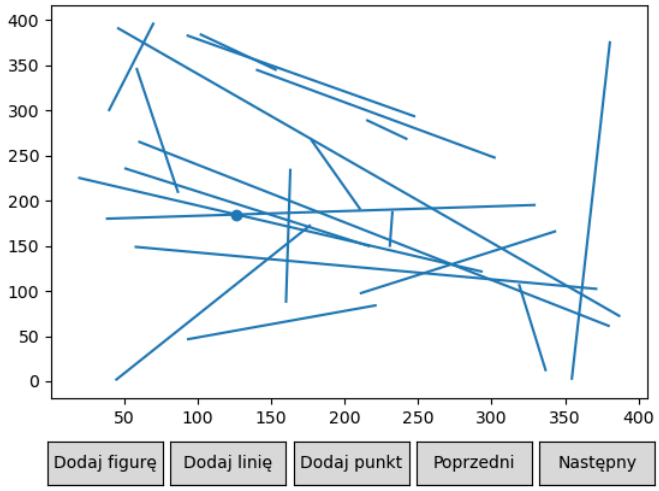
Wykres 1. Wynik dla pliku „serce.json”



Wykres 2. Wynik dla pliku „cwiczenia.json”



Wykres 3. Wynik dla pliku „test1.json”



Wykres 4. Wynik dla pliku „test2.json”

Algorytm wyznaczający wszystkie przecięcia odcinków

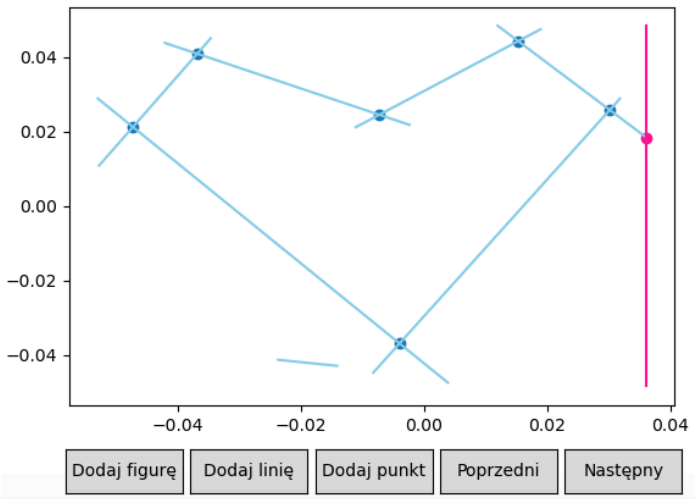
Wykorzystano implementację poprzedniego algorytmu modyfikując odpowiednie elementy tak, aby otrzymana implementacja umożliwiała wyznaczenie wszystkich przecięć.

Modyfikacja nastąpiła w obsłudze zdarzeń:

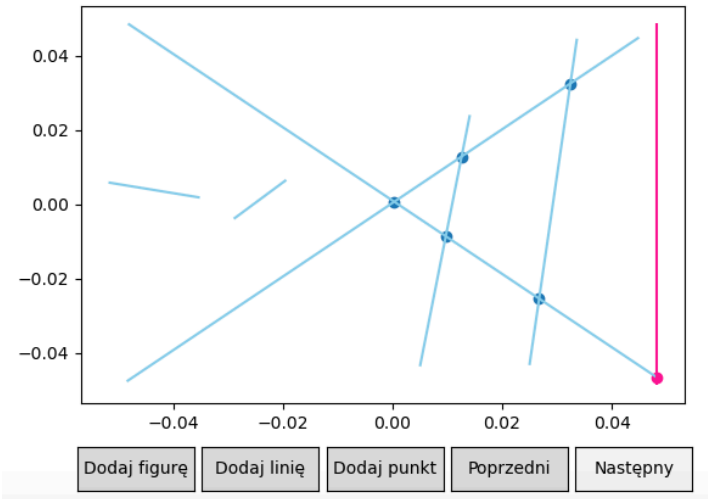
A. ***Początek odcinka*** - jeśli odcinek ma sąsiadów następuje weryfikacja, czy przecina się z nimi. Jeśli się przecina, to punkt przecięcia zostaje w odpowiednie miejsce i z odpowiednim oznaczeniem wstawiony do struktury zdarzeń.
B. ***Koniec odcinka*** - weryfikacja, czy sąsiedzi (jeśli istnieją) usuwanego ze struktury stanu odcinka się przecinają. Jeśli się przecinają, to punkt przecięcia zostaje w odpowiednie miejsce i z odpowiednim oznaczeniem wstawiony do struktury zdarzeń.
C. **Punkt przecięcia dwóch odcinków** - odcinki te zostają usunięte ze struktury stanu, współrzędna x miotły zostaje zmieniona na aktualną z niewielkim przesunięciem (10^-6). Następnie odcinki dodawane są ponownie, zamienione ze sobą miejscami w strukturze miotły. Sprawdzane jest, czy odcinki nie przecinają się ze swoimi nowymi sąsiadami.

Algorytm wymagał także rozwiązania problemu występowania duplikatów. Aby zapewnić niepowtarzalność punktów przecięcia utworzono zbiór (set), w którym zapisywane są przecinające się odcinki w postaci krotki. Po wykryciu punktu przecięcia następuje weryfikacja, czy dana para odcinków już nie wystąpiła w zbiorze. Jeśli wystąpiła, to punkt przecięcia nie zostaje dodany po raz kolejny.

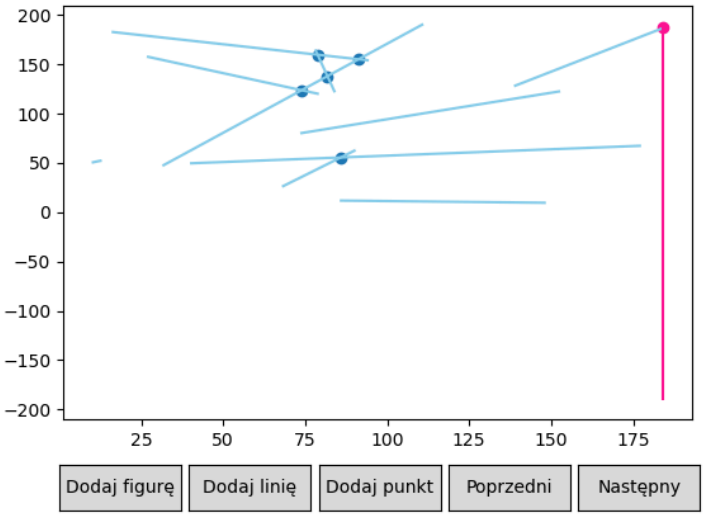
Wynik działania algorytmu wyznaczającego wszystkie przecięcia:



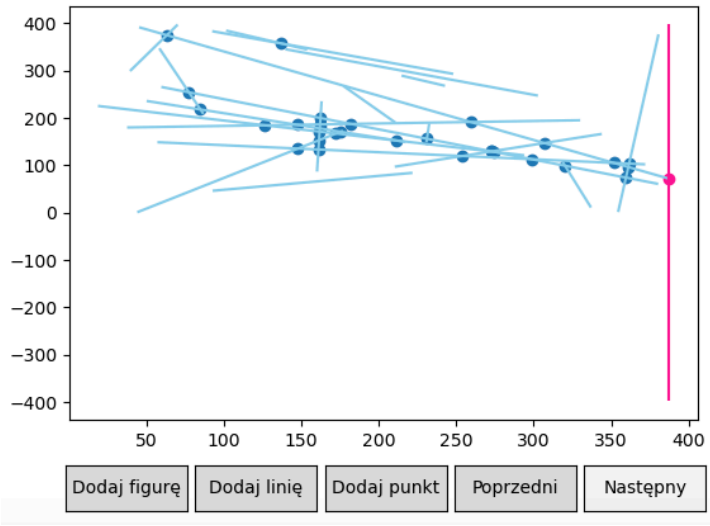
Wykres 5. Wynik dla pliku „serce.json”



Wykres 6. Wynik dla pliku „ćwiczenia.json”



Wykres 7. Wynik dla pliku „test1.json”



Wykres 8. Wynik dla pliku „test2.json”

5. Wnioski

Zaimplementowane algorytmy poprawnie wyznaczały punkty przecięcia dla wszystkich testowanych przypadków.

Użyte struktury danych (PriorityQueue oraz SortedSet) ułatwiły implementację algorytmów. Umożliwiły także wydajniejsze działanie i poprawne uporządkowanie.

Aby uniknąć duplikatów, wykryte punkty przecięcia zapisywane są w zbiorze. Są one reprezentowane jako pary linii, które się przecinają. Gdyby zamiast dwóch odcinków algorytm porównywał współrzędne punktów mogłoby to doprowadzić do błędów.

Zapobieganie duplikatom przy użyciu zbioru zapobiegło niepoprawnemu działaniu algorytmu.