Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Algorytmy geometryczne

Ćwiczenie 4

Autor:
Andrzej Zaborniak
Informatyka, rok II gr. 4

08.12.2022 r.

1 Specyfikacja techniczna

Parametry techniczne komputera na którym zostało wykonane ćwiczenie:
Procesor: Intel® Core™ i7-5600U CPU @ 2.60GHz × 4
Karta graficzna: Mesa Intel® HD Graphics 5500 (BDW GT2)
Pamięć RAM: 8,0 GB
System operacyjny: Ubuntu 22.04.1 LTS
Wersja GNOME: 42.4
Użyty język programowania: Python 3.10.6
Wykorzystany program: Jupyter Notebook

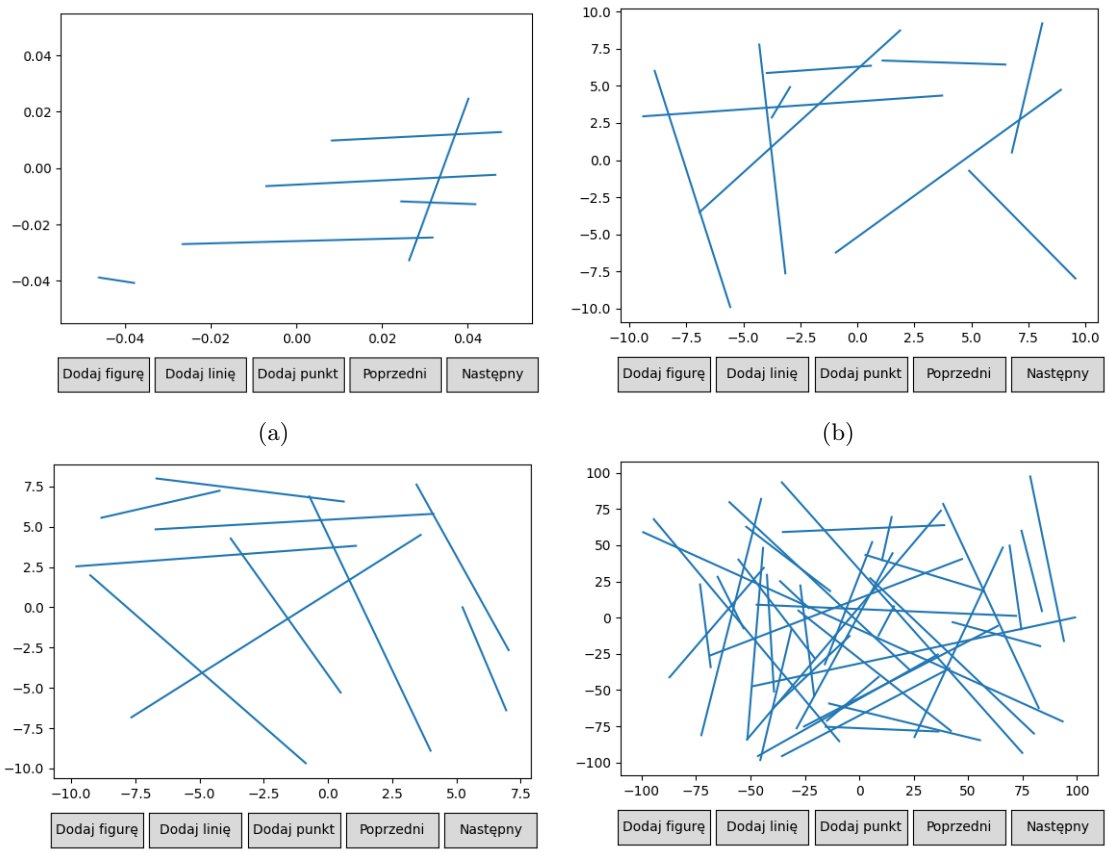
1.1 Narzędzie graficzne

W ćwiczeniu do wizualizacji użyte zostało narzędzie, które było rekomendowane na laboratoriach. Dane testowe można wprowadzić za pomocą myszki, ale została również zaimplementowana funkcja *generate_random_lines*, która przyjmuje jako argumenty: liczbę odcinków oraz zakres z którego odcinki będą losowane. Tak jak mówi nazwa, generują ona zadaną ilość odcinków na przedziale. Odcinki te spełniają następujące kryteria:
- żaden odcinek nie jest pionowy,
- zbiór nie posiada dwóch odcinków mających początki o tych samych współrzędnych x-owych.

Dodana została również opcja zapisu danych do pliku csv.

2 Zbiór odcinków

Poniższe obrazki przedstawiają zbiory odcinków na których został przetestowany zaimplementowany algorytm.
Pierwszy zbiór został zadany ręcznie za pomocą myszki, zaś 3 kolejne wygenerowane za pomocą wcześniej opisanej funkcji.



(a) (b)

(c) (d)

Rysunek 1

3 Ćwiczenie

3.1 Struktury stanu

Do przechowywania struktury zdarzeń jak i stanu zarówno w wyszukiwaniu jednego jak i wszystkich przecięć odcinków został użyty SortedSet zaimportowany z biblioteki sortedcontainers. Struktura ta działa na podstawie drzewa BST, dzięki czemu operacje wykonywane na niej są w czasie log(n), gdzie n jest liczbą aktualnie trzymanych w niej elementów.

W strukturze przechowującej zdarzenia Q, są w niej trzymane punkty z końców odcinków oraz punkty przecięć wszystkich par odcinków aktywnych, które kiedykolwiek były sąsiadami w strukturze, są one posortowane względem x-ów.

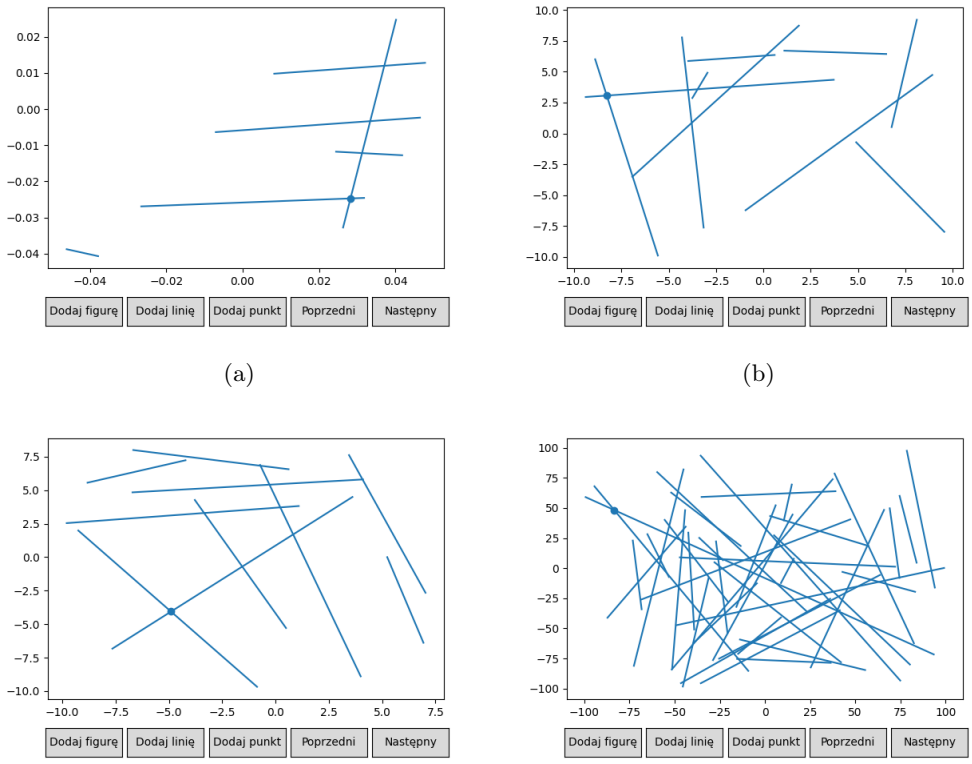
Struktura stanu T również jest zaimplementowana w SortedSecie, oraz jest posortowana względem współrzędnych y-owych, są w niej trzymane odcinki, które aktualnie przecina miotła.

Aby ułatwić przechowywanie jak i operowanie na punktach i odcinkach zostały napisane specjalne klasy: Point oraz Line.

Klasa Point oprócz współrzędnych przechowuje również zmienne: *one* i *two*, które odpowiadają indeksom linii z których dany punkt pochodzi. Jeżeli punkt jest początkiem lub końcem odcinka który został podany to obydwie wartości są takie same, zaś w przypadku gdy punkt powastł poprzez przecięcie się dwóch odcinków, zmienne przechowują odpowiednio indeksy prostych które się przecięły.

Klasa Line przechowuje punkty (początkowe oraz koniec) oraz stałe a i b z równania: $y = ax + b$, które pomagają w późniejszym sortowaniu linii według współrzędnej y-owej.

3.2 Wykrywanie jednego przecięcia



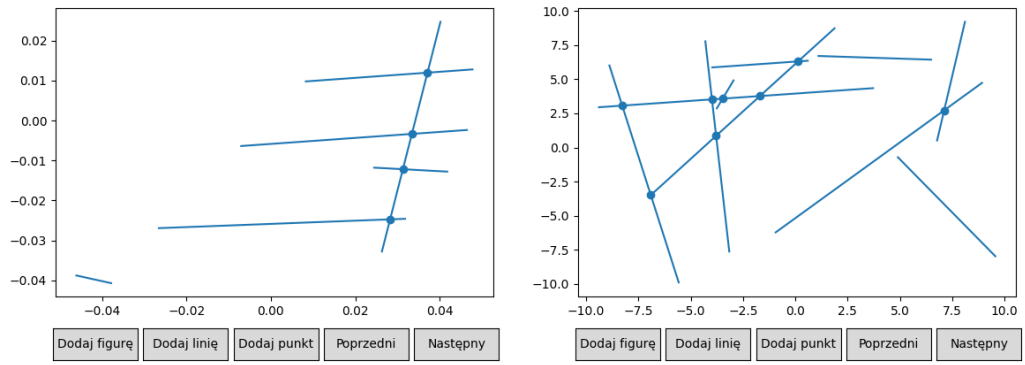
(a) (b)

(c) (d)

Rysunek 2

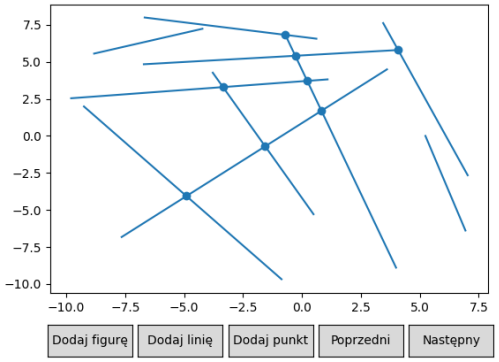
Powyższe rysunki przedstawiają poprawne działanie zaimplementowanego algorytmu.

3.3 Wykrywanie wszystkich przecięć odcinków

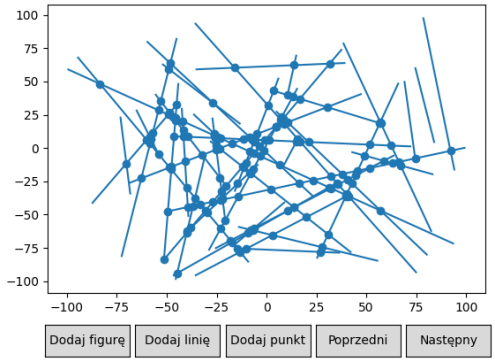


(a)

(b)



(c)



(d)

Rysunek 3

3.3.1 Zapobieganie duplikatów

Znajdując nowy punkt przecięcia algorytm sprawdza czy nie jest on duplikatem, wykorzystując fakt że para linii może przeciąć się tylko raz w jednym miejscu, dlatego dla nowego punktu sprawdzane są wszystkie poprzednie, czy któryś punkt nie posiada takich samych parametrów *one* oraz *two*, ponieważ to by oznaczało, że punkt który tworzą dwie proste jest już dodany.

3.3.2 Wykrywanie przecięć

Przecięcia odcinków są wykrywane z wykorzystaniem wyznaczników i rozwiązania dwóch równań wyznaczonych przez punkty tworzące odcinki.

3.3.3 Zdarzenia

Wyróżniamy trzy rodzaje zdarzeń:
- początek odcinka,
- przecięcie się odcinków,
- koniec odcinka.

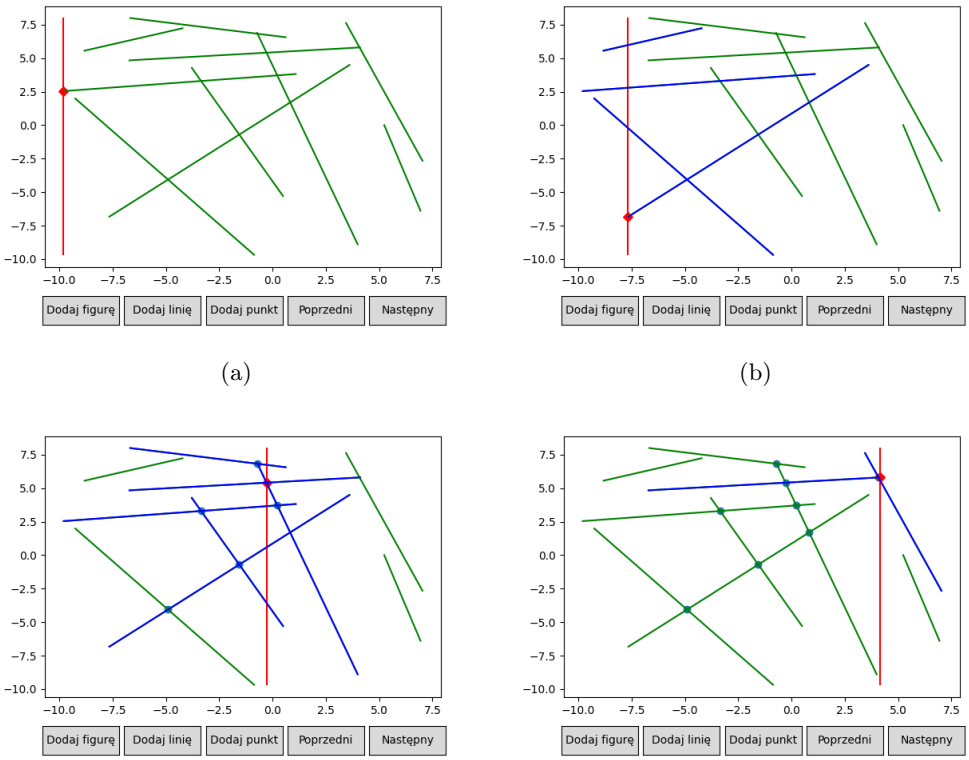
W przypadku punktu początkowego aktualizowany jest punkt X który zawiera się w strukturze *Line*, aby w przypadku dodania nowej linii do struktury stanu T została ona przyporządkowana w odpowiednim miejscu względem aktualnego położenia miotły.
Nowa dodana linia może mieć maksymalnie dwóch sąsiadów (górnego oraz dolnego), to właśnie z nimi sprawdzam czy linia się nie przecina.

W przypadku w którym miotła znajduje się w punkcie przecięcia, obydwa odcinki tworzące przecięcie są usuwane ze struktury po czym miotła jest przesunięta o znikomą wartość w prawo (dodatnim kierunku x), dzięki czemu przy ponownym wstawieniu linii do struktury są one zamienione miejscami.
Po zamianie odcinki mają nowych sąsiadów (górnego oraz dolnego) i to właśnie z nimi algorytm ponownie sprawdza czy nie powstaną nowe przecięcia.

Po dojściu do punktu końcowego odcinka ponownie sprawdzani są sąsiedzi oraz możliwe z nimi przecięcia, po czym jest on usuwany ze struktury.

3.4 Wizualizacja działania algorytmu

W wizualizacji odcinki które aktualnie przecina miotła są koloru niebieskiego, zaś te które zostały lub dopiero będą przetworzone przez miotłe są koloru zielonego.



(a) (b)

(c) (d)

Rysunek 4

4 Wnioski

Algorytm dla wszystkich danych zadziałał poprawnie wykrywając wszystkie punkty przecięć. Jednym z problemów które można było napotkać podczas implementacji algorytmu jest możliwość wystąpienia duplikatu punktu. Mając na uwadze, że nie operujemy na liczbach całkowitych, podczas przechowywania jedynie współrzędnych wyznaczonych punktów moglibyśmy niepoprawnie porównywać czy dany punkt został już wyznaczony z uwagi na możliwy błąd przy liczeniu dokładności. Dlatego najlepszym zastosowanym rozwiązaniem jest przechowywanie dla każdego punktu przecięcia z jakich dwóch odcinków on powstał.