Maciej Michoń, Informatyka II rok, gr. 2 Śr_13.00_A

Algorytmy geometryczne, sprawozdanie z ćwiczenia 4

1. Środowisko, biblioteki oraz dane techniczne urządzenia

Ćwiczenie zostało zrealizowane w Jupyter Notebooku i napisane w języku Python, z wykorzystaniem bibliotek numpy (umożliwiająca zaawansowane operacje numeryczne), pandas (do przedstawiania wyników poszczególnych obliczeń w DataFrame). Dodatkowo do rysowania wykresów (Visualizer) i sprawdzenia poprawności niektórych funkcji wykorzystałem projekt opracowany przez studentów Koła Naukowego BIT. Wszystko to było wykonane na laptopie z systemem operacyjnym Windows 11, procesorem AMD Ryzen 5 5600H 3.30 GHz oraz pamięcią RAM 32 GB.

2. Opis realizacji ćwiczenia:

Celem ćwiczenia była implementacja algorytmu wykrywającego przecięcia między odcinkami na płaszczyźnie 2D.

2.1. Generacja zbiorów danych (odcinków)

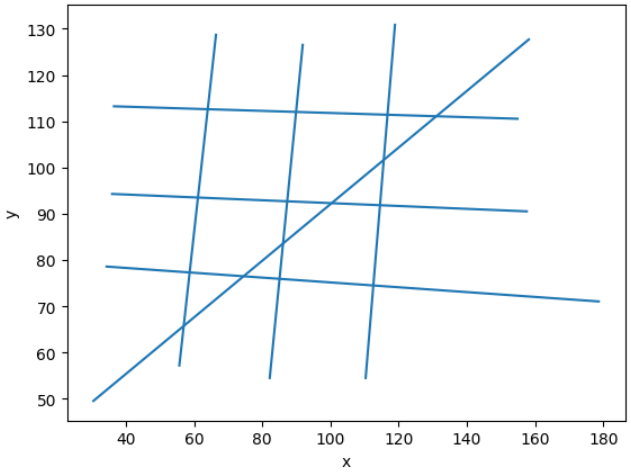
Program pozwala na tworzenie odcinków za pomocą własnoręcznie napisanej klasy LinesBuilder. Za pomocą myszki można wprowadzić kolejne odcinki. Po narysowaniu zostaje zwrócona lista odcinków. W ten sposób łatwo można ponownie narysować figurę. Dodatkowo zostały stworzone funkcje save_() i read_(), dla których można w łatwy sposób potrzebne nam dane zapisywać i odczytywać w pliku json.

Program także pozwala na generowanie losowych odcinków. Generowanie odcinków polega na losowaniu dla każdej linii (których ilość jest zadana parametrem) pary odcinków o współrzędnych mieszczących się w zadanym przedziale. Odcinki generowane mają założenie, że nie mogą powstawać odcinki pionowe oraz żadne dwa odcinki nie mają swoich końców o takiej samej współrzędnej x.

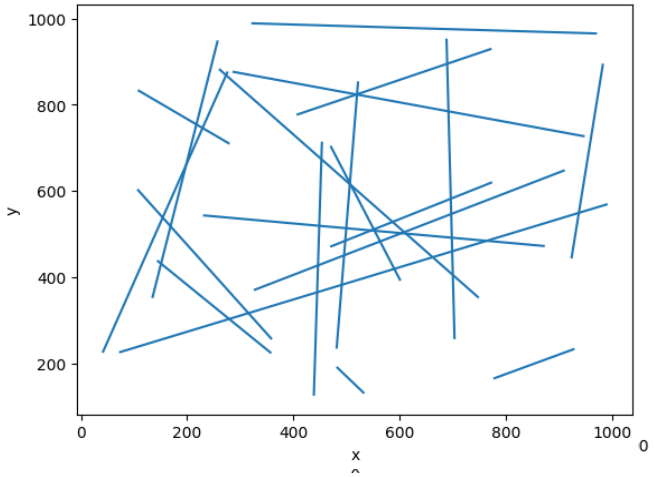
W celu przetestowania algorytmów wykorzystałem zestawy odcinków dostarczone przez testy z projektu oraz inne stworzone zestawy odcinków:

- Przecięta siatka
- Losowe 25 odcinków
- Test_BIT_1
- Test_BIT_2
- Brak przecięcia
- Przecięcia

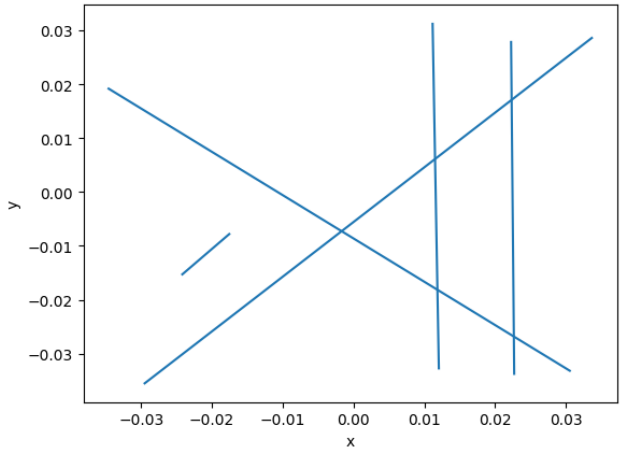
Rysunek 1.1 Zestaw odcinków ‘Przecięta siatka’



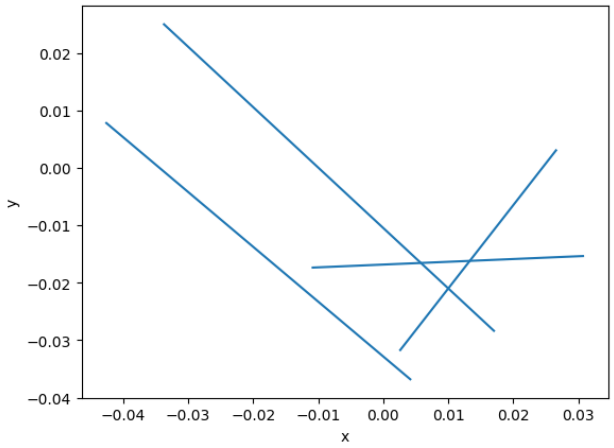
Rysunek 1.2 Zestaw odcinków ‘Losowe 25 odcinków’



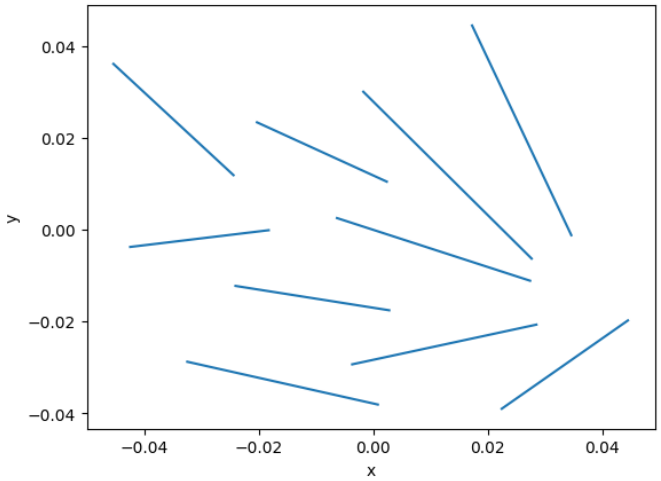
Rysunek 1.3 Zestaw odcinków ‘Test_BIT_1’



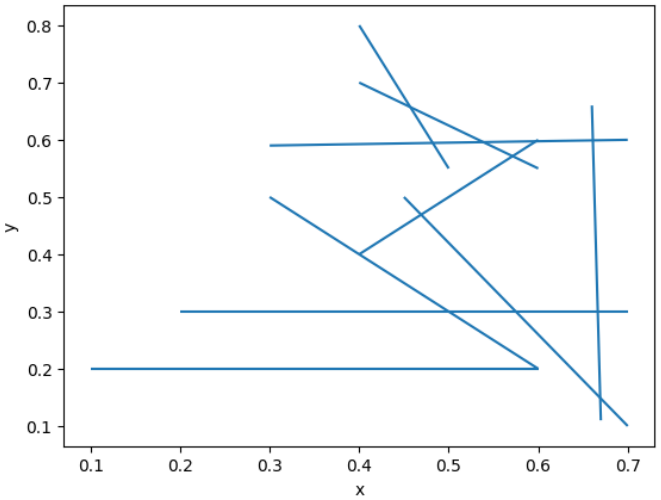
Rysunek 1.4 Zestaw odcinków ‘Test_BIT_2’



Rysunek 1.5 Zestaw odcinków ‘Brak przecięcia’



Rysunek 1.6 Zestaw odcinków ‘Przecięcia’



2.2. Struktury danych

Dla struktur zdarzeń i struktur stanów wykorzystano zbiór uporządkowany (SortedSet z biblioteki sortedcontainers) zarówno w przypadku wykrywającego jedno jak i wszystkie przecięcia. Jest to struktura oparta na zrównoważonym drzewie binarnym, a zatem operacje wstawiania i usuwania elementów mają złożoność logarytmiczną. Struktura ta zachowuje porządek wg. zadanego klucza – dla punktów jest to współrzędna x, a dla odcinków jest to jej wartość y w danym położeniu miotły.

W strukturze zdarzeń znajdują się krotki zawierające trzy wartości: współrzędne punktu, wartość czy punkt jest początkiem czy końcem(0 –początek,1 –koniec) oraz indeks odcinka na którym się ten punkt znajduje. Dodatkowo w tej strukturze mogą być punkty przecięcia, to w nich przechowuje się oba indeksy odcinków, na których znajduje się dany punkt przecięcia.

W strukturze stanu znajdują się obiekty reprezentujące linie. Każda linia zawiera dwa punkty (początkowy i końcowy) oraz dodatkowo współczynniki a i b opisujące tą linie równaniem y=ax+b. Wykorzystywane jest to do porządkowania linii wg. ich położenia pionowego.

W algorytmie wykrywania jednego przecięcia i algorytmie wykrywającym wszystkie przecięcia wykorzystano tą samą strukturę bez żadnych zmian. W przypadku tego pierwszego można jednak było uprościć, ponieważ wykrywamy tylko jedno przecięcie, więc nie ma konieczności zamieniania miejscami linii po wykryciu przecięcia. Dzięki temu łatwiej jest utrzymać porządek.

Sprawdzone też zostało działanie programu kiedy struktura zdarzeń i struktura stanów wykorzystuje tablice. Zmiana będzie taka że po dodaniu punktu, usunięciu zawsze będzie trzeba będzie sortować tablice, żeby zachować porządek.

2.3. Wykrywanie przecięć i zapobieganie duplikatów

Przecinanie się odcinków zostają znalezione z wykorzystaniem wyznaczników oraz parametrycznej reprezentacji linii. Sprawdzamy, czy przecięcie prostych, w których zawierają się nasze odcinki zawiera się w nich, a następnie wyliczamy dokładny punkt przecięcia. Znajdując nowy punkt przecięcia sprawdzamy czy nie jest on duplikatem. Wykorzystujemy założenie, które mówi, że dwie proste mogą przecinać się maksymalnie raz. Zatem dla nowego punktu sprawdzane jest dla wszystkich wcześniej odkrytych punktów przecięć czy nie są one punktami leżącymi na tych samych dwóch liniach. Jeśli nie są to nie jest to duplikat i dodajemy go do struktury zdarzeń i zbioru wynikowego.

Nie są sprawdzane duplikaty po współrzędnych ponieważ nie miałoby to sensu z tego powodu, że mogą wystąpić jakieś błędy związane ze skończoną precyzją obliczeń. Wtedy może program niepoprawnie wykrywać duplikaty tak jak to się najprawdopodobniej dzieje w testach od koła naukowego BIT, gdzie punkty różnią się na siedemnastym miejscu po przecinku, a wszystkie proste, które się przecinają są poprawne.

2.4. Obsługa zdarzeń

Wyróżniamy 3 rodzaje zdarzeń – punkt jest początkiem odcinka, punkt jest końcem odcinka, punkt jest punktem przecięcia dwóch odcinków.

W przypadku punktu początkowego aktualizujemy we wszystkich liniach współrzędną x miotły (aby przy wstawianiu nowej linii porównania były wykonane zgodnie z aktualnym stanem). Następnie wstawiamy linię powiązaną z tym punktem do struktury stanu. Po wstawieniu ma ona maksymalnie dwóch sąsiadów (dolny i górny) z którymi należy sprawdzić czy się nie przecina.

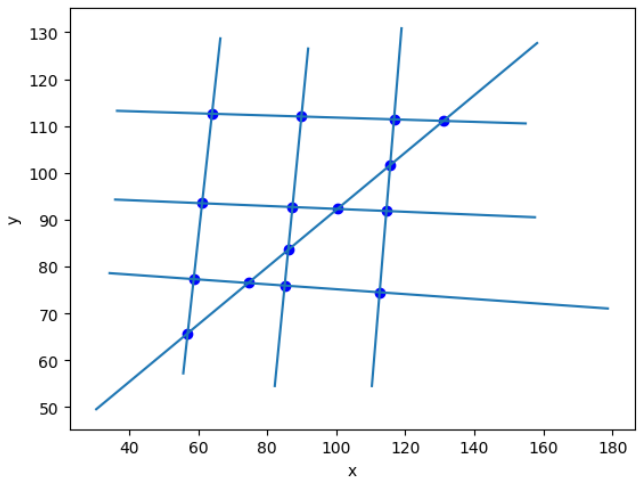
W przypadku punktu końcowego aktualizujemy we wszystkich liniach współrzędną x miotły, sprawdzamy czy górny i dolny sąsiad (o ile istnieją) się przecinają, a następnie usuwamy ten odcinek ze struktury stanu.

W przypadku punktu przecięcia usuwamy ze struktury stanu oba odcinki, na których leży ten punkt, następnie aktualizujemy we wszystkich liniach współrzędną x miotły z lekkim przesunięciem w prawo (aby miotła była za przecięciem) po czym dodajemy je ponownie. Dzięki zaktualizowaniu odcinki zostaną zamienione miejscami. Następnie sprawdzamy czy odcinki te nie przecinają się z nowymi sąsiadami (nad i pod nimi).

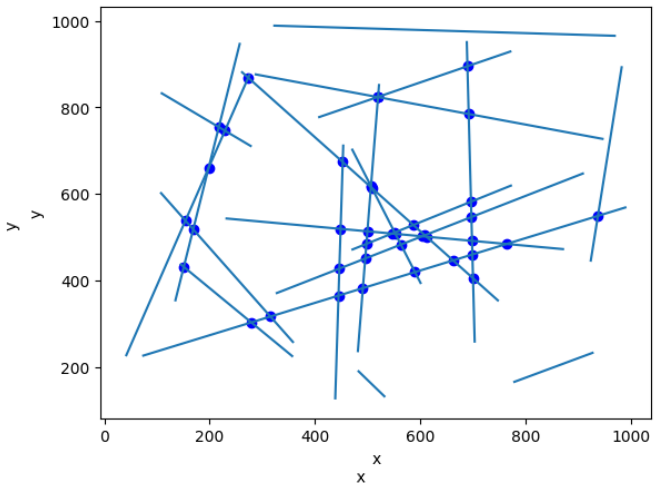
2.5. Wynik algorytmu

Algorytm na wejściu przyjmuje tablicę linii i w zależności jakiej użyjemy funkcji zwróci tablice punktów przecięć w postaci trzyelementowych krotek w których pierwszym elementem są współrzędne danego punktu, a drugim i trzecim indeksy prostych z listy wejściowej, które się przecinają w tym punkcie współrzędnych lub wizualizacje procesu zamiatania dla przykładowego zestawu odcinków. Zarówno dla struktur opartych na sortedcontainers jak i zwykłej tablicy, wynik był poprawny i taki sam.

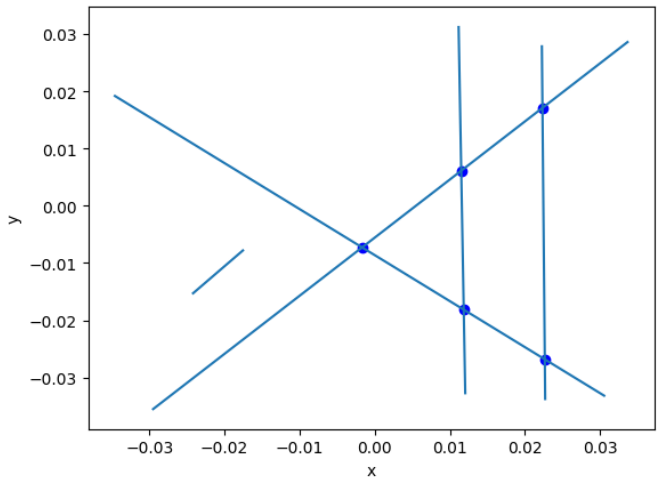
Rysunek 2.1 Wynik algorytmu dla zestawu odcinków ‘Przecięta siatka’



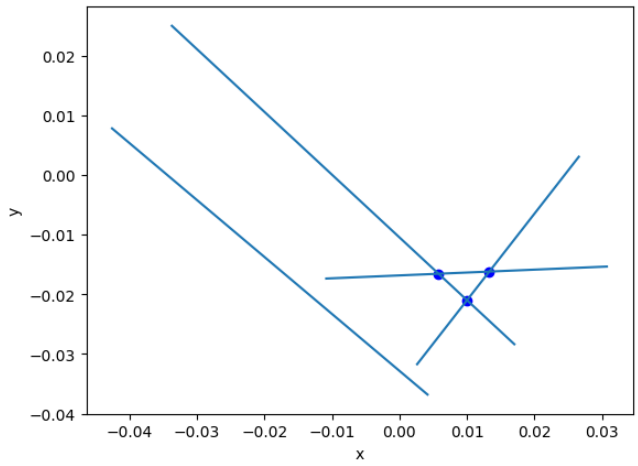
Rysunek 2.2 Wynik algorytmu dla zestawu odcinków ‘Losowe 25 odcinków’



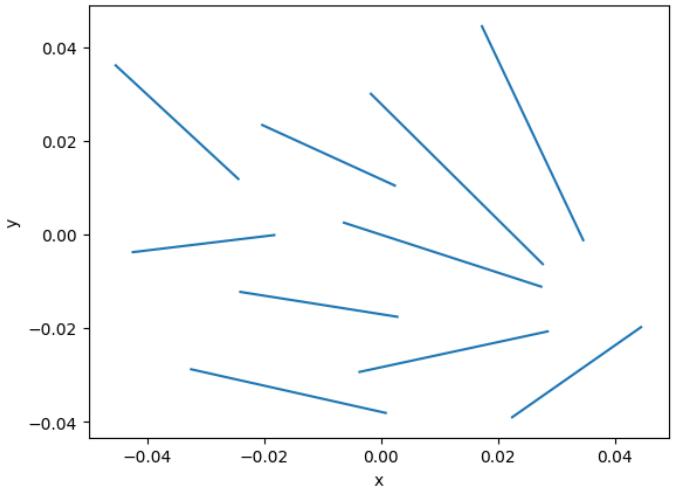
Rysunek 2.3 Wynik algorytmu dla zestawu odcinków ‘Test_BIT_1’



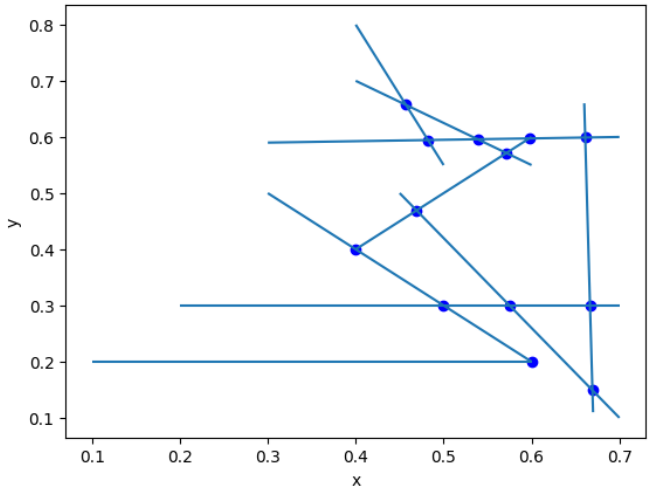
Rysunek 2.4 Wynik algorytmu dla zestawu odcinków ‘Test_BIT_2’



Rysunek 2.5 Wynik algorytmu dla zestawu odcinków ‘Brak przecięcia’



Rysunek 2.6 Wynik algorytmu dla zestawu odcinków ‘Przecięcia’



3. Wnioski

Algorytm zadziałał poprawnie dla testowanych zbiorów odcinków oraz dla dużych zbiorów odcinków generowanych losowo. Dzięki zastosowaniu strukturom danych złożoność jest optymalna dla tego typu algorytmu. Wykrycie tylko jednego odcinka jest dosyć proste, natomiast to właśnie obsługa zdarzeń będących punktami przecięcia jest trudnością w algorytmie głównym. Kolejną trudnością było usuwanie duplikatów wynikająca z tego, że nie wystarczyło sprawdzić po współrzędnych, ponieważ różniły się one na kilku miejscach po przecinku. Więc tutaj trzeba było skorzystać z faktu, że para linii może przecinać się w maksymalnie jednym punkcie. Dzięki takiemu warunkowi dla dowolnej konfiguracji odcinków duplikaty nie mają prawa wystąpić. Zarówno dla struktury opartej na sortedcontainers jak i zwykłej tablicy czasy były bardzo podobne działania co w teorii tak powinno nie być. Może to wynikać z testów, które są za małe żeby zaobserwować różnice w czasie działania.