# LABORATORIUM 3

## Algorytmy geometryczne

## Ćwiczenie 4 – sprawozdanie

Hieronim Koc

08.12.2022

grupa nr 4, czwartek_11_np_4

**Dane techniczne specyfikacji komputerowej**:

- Procesor : AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
- RAM: 8.00 GB
- Typ system: 64-bitowy system operacyjny
- Środowisko: Jupyter notebook
- System operacyjny: Ubuntu 22.04 LTS Jammy Jellyfish (2022-04-21)

Wszystkie testy i ich implementacje były wykonywane w języku programowania Python 3.9.12, z wykorzystaniem zewnętrznych bibliotek takich jak: *numpy*, *matplotlib, sortedcontainers, random.* Dodatkowe zmiany do narzędzia graficznego wprowadził Patryk Klatka.

# Cel ćwiczenia

Celem ćwiczenia było zapoznanie się przecinaniem się odcinków na płaszczyźnie oraz z algorytmami wyznaczania czy dane odcinki się przecinają, jeżeli tak to gdzie, jak i wyznaczanie wszystkich przecięć odcinków.

# Opis ćwiczenia

Ćwiczenie składało się z implementacji dwóch rozbudowanych algorytmów, gdzie jeden z nich sprawdzał tylko czy istnieje przecięcie, a drugi obliczał wszystkie przecięcia występujące na zadanej płaszczyźnie. Obowiązkowym rozszerzeniem do tego była implementacja generowania losowych odcinków przy zadanych parametrach oraz wczytywanie i zapisywanie odcinków.

# Struktury

Dla algorytmu wyszukiwania czy istnieje dane przecięcie została użyta własno-zaimplementowana struktura jaką jest drzewo czerwono-czarne z modyfikowalnym porównywaniem kluczy w oparciu o przekazaną funkcję, dla stanu miotły. Przy dodawaniu odcinków korzystała ona z porównywania danych po współrzędnej y.

Dla algorytmu wyznaczania wszystkich przecięć, dla stanu miotły została użyta struktura SortedSet zaimportowana z biblioteki *sortedcontainers*. Jest ona reprezentowana również jako drzewo czerwono-czarne.

Obydwie struktury opisane powyżej pozwalają na dodanie, usunięcie, przeszukiwanie, wyszukiwanie poprzednika i następnika w czasie $O(\log n)$.

Dla struktury zdarzeń w pierwszym algorytmie zostało uruchomione sortowanie odcinków po współrzędnej *x*-owej. Było to potrzebne by brać punkty zdarzeń w uporządkowany sposób.

Dla struktury zdarzeń w drugim algorytmie użyto *SortedSet*, gdyż zapewnia to uniknięcie dodawanie tego samego odcinka wiele razy oraz dzięki owej strukturze można w szybki sposób wstawiać i usuwać elementy.

# Obsługiwanie zdarzeń

Dla drugiego algorytmu zdarzenia początku, końca i przecięcia odcinków zostały obsłużone w następujący sposób:

- *początek odcinka :* aktualizacja struktury zdarzeń poprzez wstawianie nowego klucza, który staje się współrzędną *x*-ową aktualnie rozpatrywanego zdarzenia

Później następuje wykonanie procedury wstawiania nowego odcinka do struktury zdarzeń i sprawdzenie czy dany odcinek przecina się z odcinkiem, z którym sąsiaduje.

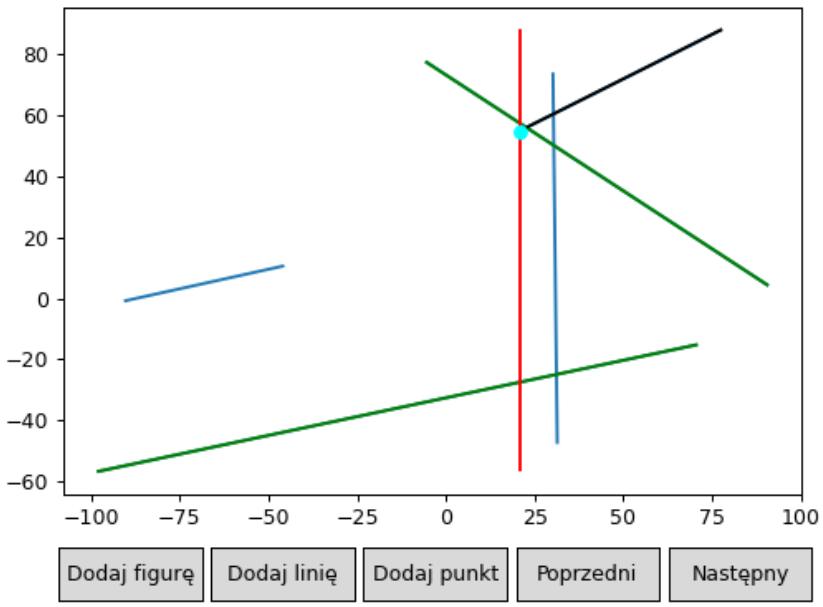
- *koniec odcinka :* jeżeli jest to koniec odcinka aktualizuje się struktura zdarzeń poprzez wstawienie nowego klucza, który staje się współrzędną *x*-ową aktualnie rozpatrywanego zdarzenia

Następuje wyszukiwanie czy dany odcinek przecina się z jego sąsiadami i po możliwym dodaniu przecięć następuje usunięcie owego odcinka z struktury stanu miotły.

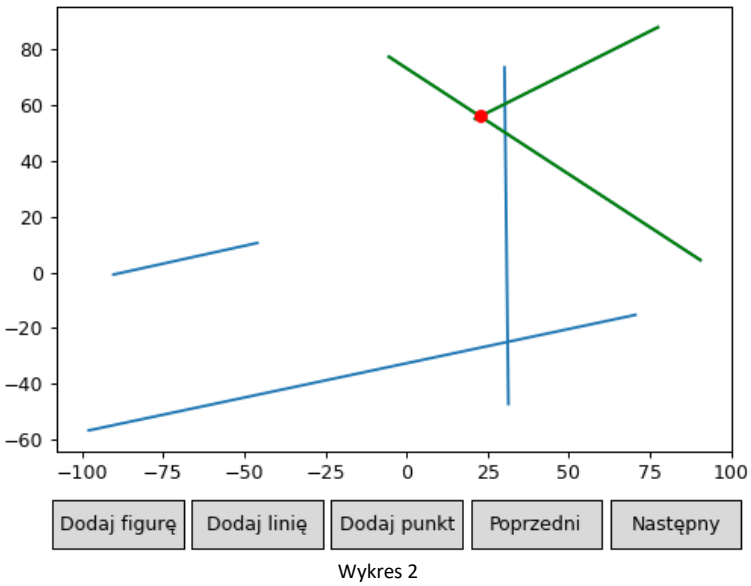
- *przecięcia odcinków :* dodanie wyszukanego przecięcia do zbioru punktów przecięć i następuje zmiana kolejności odcinków w strukturze zdarzeń poprzez usunięcie i dodanie na nowo do struktury zdarzeń.

# Przykładowe wyniki

## Algorytm sprawdzający czy istnieje przecięcie się odcinków

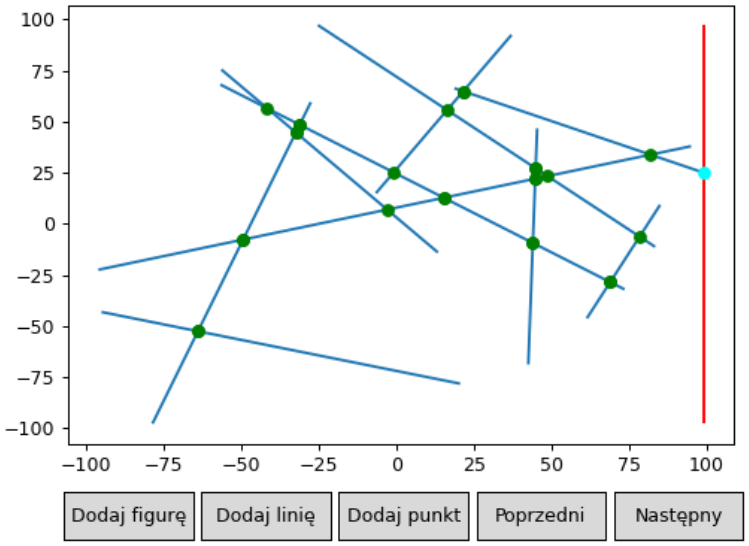



Wykres 1




Wykres 2

## Algorytm wyliczający przecięcia odcinków




Wykres 3

# Wnioski

Zgodnie z oczekiwaniami, algorytm do sprawdzania czy istnieje dane przecięcie i algorytm do wyznaczania wszystkich przecięć prawidłowo zadziałał. Struktury stanu miotły i zdarzeń, które zostały zaimplementowane, bądź zaimportowane pozwoliły na szybkie działanie algorytmu. Dzięki drzewom czerwono-czarnym dostęp do każdego elementu był szybki, a w drugim algorytmie pozwoliło na uniknięcie powtarzających się tych samych punktów. Zapisywanie do pliku odbyło się w formacie *json* z uwagi na przyjazne użytkownikowi odtworzenie odcinków. Dzięki zaimplementowanym dodatkowym klasom dla punktów i odcinków można było uniknąć błędu liczb zmienno-przecinkowych w przypadku porównywania ich.