Algorytmy geometryczne

Sprawozdanie z laboratorium 4.

Dawid Zawiślak gr. Czw. 13:00 A

Dane techniczne urządzenia oraz narzędzia za pomocą których wykonano ćwiczenia:

- Laptop z systemem operacyjnym Windows 11
- Procesor Intel Pentium Gold 8505
- RAM 8 GB

Do realizacji ćwiczenia użyto środowiska Jupyter Notebook z Pythonem w wersji 3.9, wykorzystując bibliotekę visualizer przygotowaną przez KN BIT (do tworzenia wykresów), matplotlib (obsługa myszki i rysowanie wielokątów zadanych przez użytkownika), PriorityQueue z modułu queue, SortedSet (implementacja struktur zdarzeń i stanu) oraz bibliotekę random (generowanie losowych odcinków).

Opis ćwiczenia:

Naszym celem na laboratorium numer 4 było zaimplementowanie algorytmu zamiatania, który wyznacza punkty przecięcia danych odcinków na płaszczyźnie 2D. Należało zaimplementować:

- Funkcje tworzącą zbiory losowych odcinków ze współrzędnymi z zadanego zakresu oraz takich, że parami mają różne współrzędne x
- Procedurę pozwalającą wprowadzić odcinki myszką
- Funkcje pozwalające na zapis i odczyt z plików wygenerowanych odcinków
- Algorytm określający czy istnieją dowolne 2 odcinki, które się przecinają w danym zbiorze
- Algorytm wyznaczający przecięcia wszystkich odcinków wraz z wizualizacją

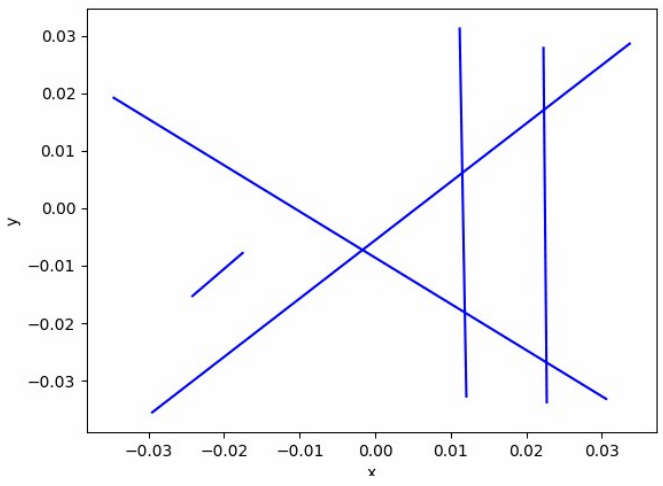
Wykonanie ćwiczenia

1) Generowanie zbiorów do testów

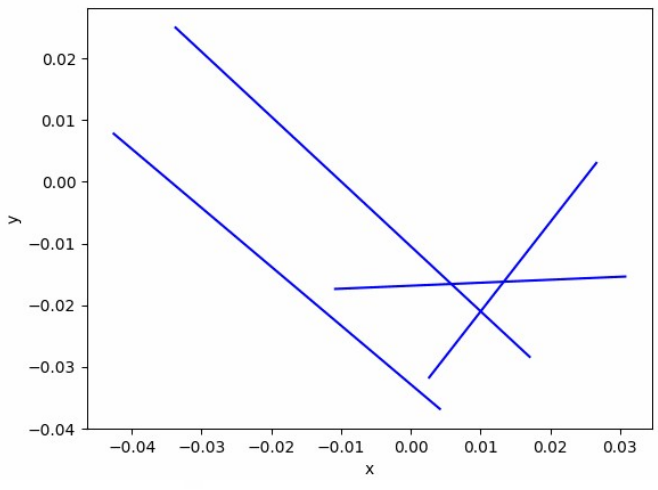
Zbiory można generować za pomocą funkcji wyznaczającej zbiory losowych odcinków o ograniczeniach na obu współrzędnych (funkcja **generate_uniform_sections**), lub wprowadzając je za pomocą myszki (sekcja **Add sections with mouse**). Za pomocą tej drugiej możemy tworzyć nowe zbiory, lub dopisywać odcinki do istniejących.

Zaimplementowano również funkcje pozwalające na zapis i odczyt zbiorów odcinków do plików (funkcje **save_sections_to_file** oraz **read_sections_from_file**).

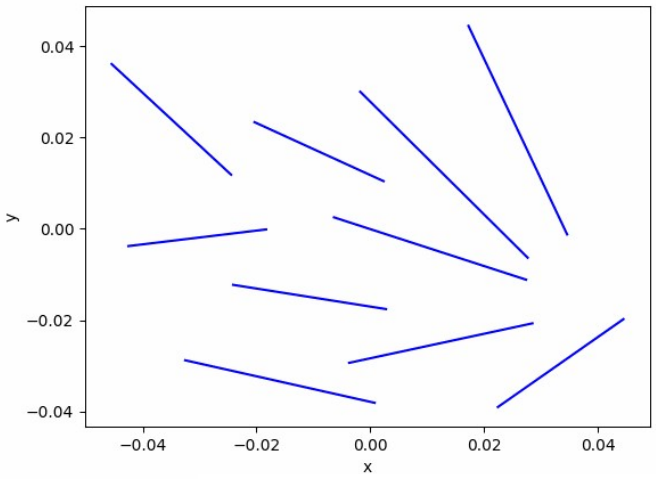
Przygotowane zbiory do testów:



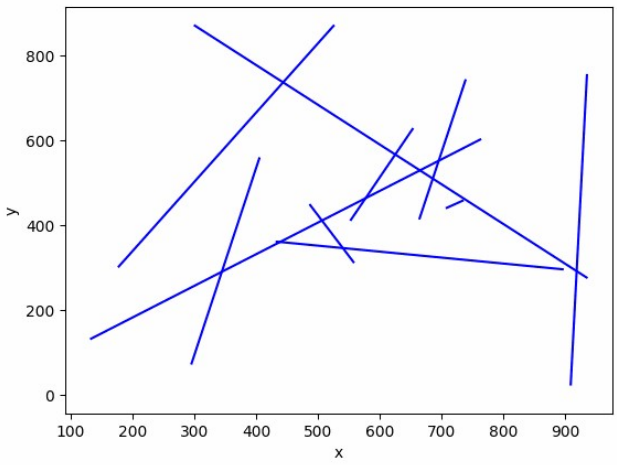
Rysunek 1 Zbiór testowy A



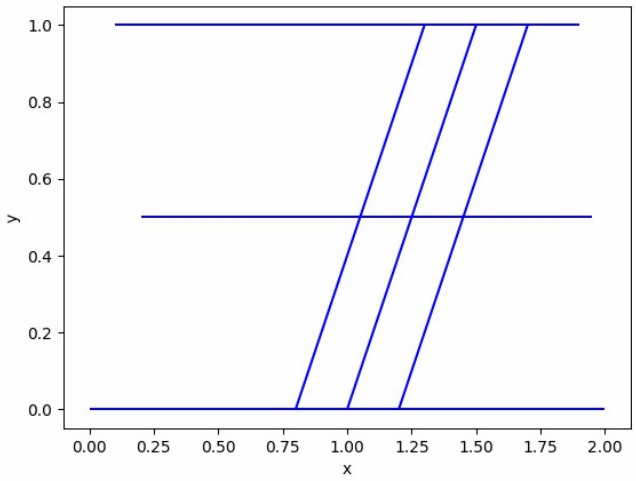
Rysunek 2 Zbiór testowy B



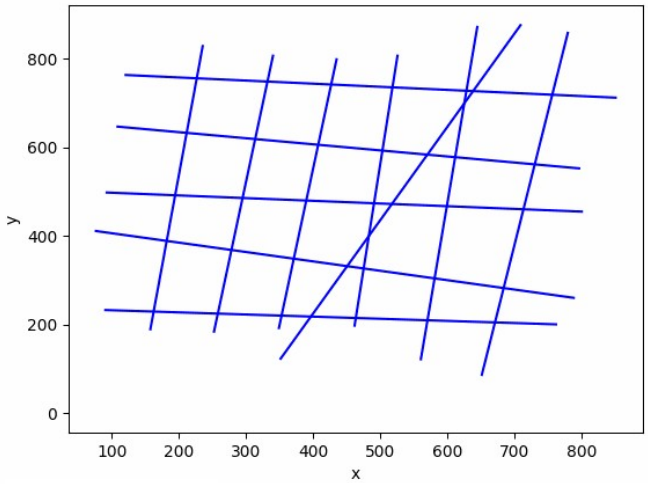
Rysunek 3 Zbiór testowy C



Rysunek 4 Zbiór testowy D



Rysunek 5 Zbiór testowy E



Rysunek 6 Zbiór testowy F

2) Struktura zdarzeń oraz struktura stanu użyta w implementacjach

Odcinki reprezentowane są za pomocą klasy **Section**, która posiada pola określające współrzędne obu końców odcinka oraz metody do pobierania odcinka w postaci krotki i określoną funkcje komparatora sparametryzowaną przez x, który dla jakiej współrzędnej x porównujemy położenie odcinków. Ten komparator używany jest później w drzewie do utrzymywania w strukturze stanu miotły posortowanych odcinków. Dostepna jest również metoda aktualizująca komparator nową współrzędną x (po wstawieniu nowego odcinka lub napotkaniu przecięcia).

Do przechowywania struktury zdarzeń została użyta **PriorityQueue** z modułu **queue,** która jest zaimplementowana na kopcu binarnym, ponieważ pozwala ona kolejkować kolejne zdarzenia względem współrzędnej x w czasie O(nlogn).

Struktura stanu przechowywana jest za pomocą klasy **SortedSet** z modułu **sortedcontainers,** która jest zbalansowanym drzewem przeszukiwań binarnych (czerwono-czarnym) pozwalającym na znajdywanie, usuwanie, oraz dostęp do poprzednika/następnika elementów w czasie O(logn). Przetrzymuje ona aktualnie analizowane odcinki (stan miotły).

3) Opis działania algorytmu zamiatania

W mojej implementacji założyłem, że w danym punkcie przecina się maksymalnie 2 odcinki ze zbioru do analizy. Przypadki gdzie 2 odcinki stykają się tylko końcem również traktowane są jako przecięcia. Również zakładamy, że żadne końce odcinków nie mają tej samej współrzędnej x.
Za wartość tolerancji dla 0 przyjąłem ε = 1e-12

Algorytm działa w następujący sposób:

Na początku tworzymy pustą kolejkę priorytetową **Q** jako kolejkę zdarzeń z kluczem jako współrzędna x i dodajemy do niej wszystkie zdarzenia początku i końca odcinków. Tworzymy również drzewo przeszukiwań binarnych **T**, które posłuży jako struktura do przechowywania stanu miotły oraz listę która będzie przechowywać przecięcia w postaci krotek (punkt przecięcia, indeks jednego odcinka, indeks drugiego odcinka) i zbiór przechowujący pary indeksów odcinków które już zostały wykryte jako przecięcia aby uniknąć dodawania duplikatów do zbioru wynikowego.

Następnie wyciągamy z kolejki zdarzenia dopóki kolejka się nie opróżni. Możemy napotkać na 3 typu zdarzeń:

- Start odcinka:

  Wstawiamy nowy odcinek do struktury **T**, oraz aktualizujemy komparator w drzewie tak aby porównywał on współrzędne y odcinków dla współrzędnej x startu dodawanego odcinka (jeśli początek nowego odcinka leży na innym odcinku przesuwam komparator o ε, aby oba odcinki miały inny klucz przy porównywaniu w drzewie). Dla nowego odcinka sprawdzamy, czy nie przecina się on ze swoim poprzednikiem i następnikiem w drzewie (jeśli istnieją). Jeśli się przecinają i przecięcie nie zostało już wcześniej wykryte dodajemy do **Q** nowe zdarzenie przecięcia w punkcie ze współrzędną x punktu przecięcia, dodajemy przecięcie do zbioru wykrytych przecięć oraz do zbiory wynikowego.

- Koniec odcinka:

  Usuwamy odcinek ze struktury **T**, oraz sprawdzamy czy jego poprzednik oraz następnik (sąsiedzi) się przecinają, gdy to przecięcie nie zostało wcześniej wykryte, wstawiamy to zdarzenie do **Q** i odpowiednich zbiorów.

- Zdarzenie przecięcia:

  Usuwamy z **T** odcinki które się przecinają, przestawiamy komparator aby porównywał współrzędne y odcinków dla $x=x_{przecięcia} + \varepsilon$. Wstawiamy z powrotem odcinki do kolejki (będą one teraz zamienione miejscami w strukturze **T**). Sprawdzamy również czy po zamianie przecinają się z nowymi sąsiadami, jeśli tak postępujemy tak samo jak opisano wyżej.

Przed sprawdzeniem czy oba odcinki się przecinają i ewentualnym dodaniem do zbioru przecięć i zdarzenia przecięcia za każdym razem sprawdzam za pomocą wyznacznika czy któryś z końców odcinków nie leży na innym odcinku, jeśli tak uznaję taką sytuacje jako przecięcie zgodnie z założeniami i nie wstawiam do kolejki zdarzenia przecięcia.

Przecięcia odcinków sprawdzam za pomocą funkcji **get_intersection_point**, która najpierw za pomocą wyliczenia wyznacznika 2x2 sprawdza czy dwa odcinki w ogóle mają szanse się przecinać, tzn. czy dwa końce jednego odcinka leżą po przeciwnych stronach drugiego i vice versa. Następnie za pomocą równań kierunkowych prostych leżących na odcinkach wyznaczam punkt przecięcia i go zwracam. Jeśli odcinki się nie przecinają funkcja zwraca None.

Złożoność obliczeniowa algorytmu: O((n+k)log(n+k))
, gdzie n = liczba odcinków, k = liczba przecięć

4) Wizualizacja działania algorytmu

W wizualizacji przyjęte zostały następujące oznaczenia:

- zielone punkty – wykryte punkty przecięcia

- niebieskie odcinki – odcinki nie przetworzone

- żółte odcinki – odcinki aktualnie znajdujące się w strukturze T

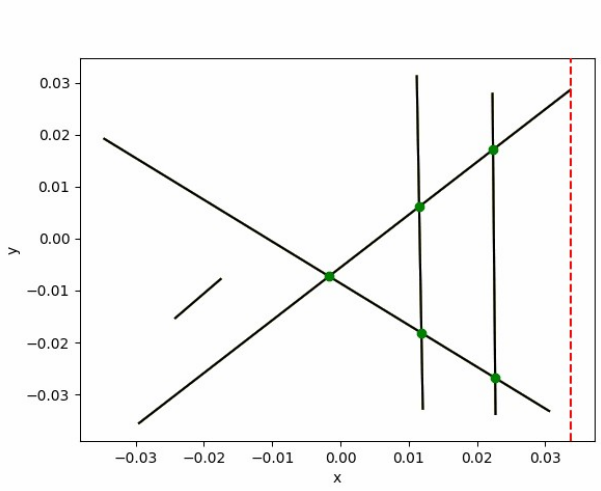
- odcinki oznaczone przerywaną zieloną linią – odcinki aktualnie porównywane

- odcinki czarne – odcinki przetworzone, które usunięto z T

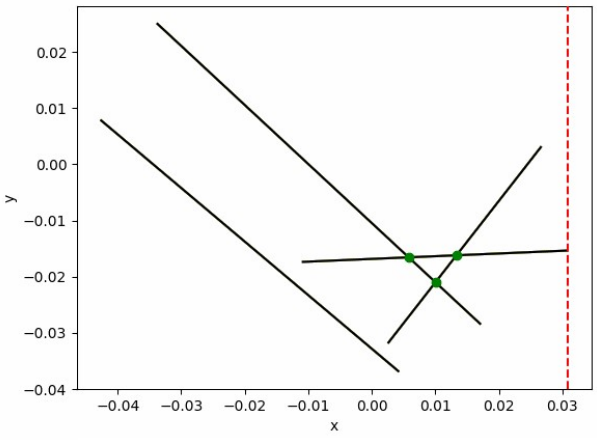
- przerywana czerwona linia – położenie „miotły”

Gify przedstawiające pełny proces krokowo dostępne są w paczce wizualizacje.zip załączonej ze sprawozdaniem.

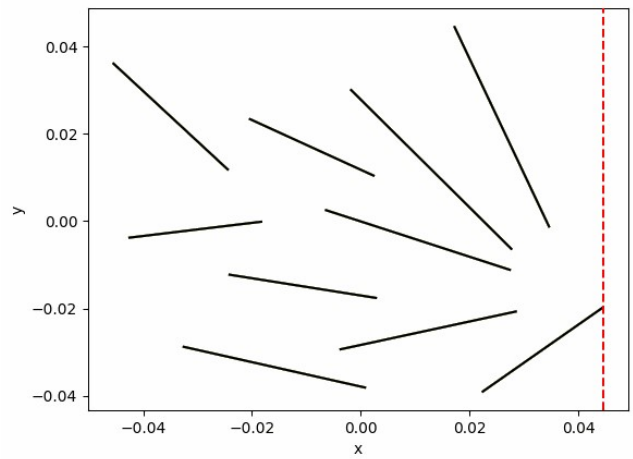
Poniżej przedstawiam gotowe wyniki wizualizacji dla każdego zbioru testowego:



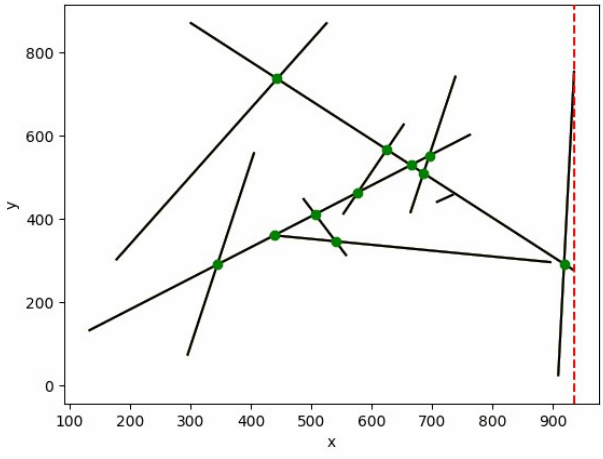
Rysunek 7 Wykryte przecięcia w zbiorze A



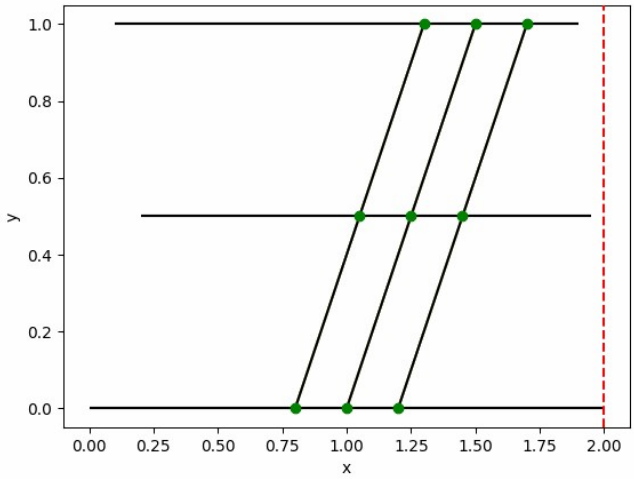
Rysunek 8 Wykryte przecięcia w zbiorze B



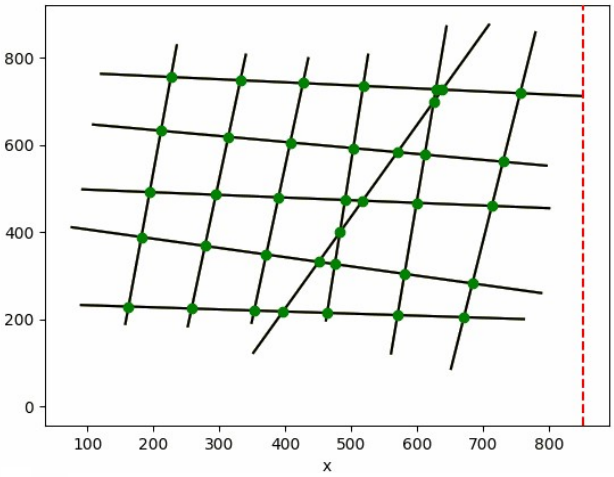
Rysunek 9 Wykryte przecięcia w zbiorze C



Rysunek 10 Wykryte przecięcia w zbiorze D



Rysunek 11 Wykryte przecięcia w zbiorze E



Rysunek 12 Wykryte przecięcia w zbiorze F

5) Wnioski

Algorytm zadziałał poprawnie dla wszystkich testów, w których zostało ujęte kilka przypadków brzegowych. Na zbiorze A widzimy, że algorytm radzi sobie z wykrywaniem duplikatów i nie wykrywa kilka razy środkowego przecięcia. Za pomocą zbioru B przetestowane zostało, czy algorytm poprawnie ustawia kolejność odcinków w strukturze T (bierze pod uwagę wartość współrzędnej y dla x startu dodawanego odcinka). Zbiór C nie zawiera żadnych przecięć i algorytm również daje oczekiwane wyniki. Na zbiorze E zaprezentowany jest przypadek gdzie końce odcinków leżą na innym odcinku, algorytm zgodnie z założeniami wykrywa taką sytuacji jako przecięcie.