Michał Szczurek
Informatyka II rok, grupa wtorek 12:50 B

Algorytmy geometryczne, ćwiczenie 4- Sprawozdanie

1. Wprowadzenie

Celem ćwiczenia była implementacja i przetestowanie algorytmu zamiatania wyznaczającego punkty przecięcia odcinków na płaszczyźnie. Algorytm zaimplementowano w dwóch wersjach - sprawdzającej czy istnieje dowolny punkt przecięcia i znajdującej wszystkie punkty przecięcia.

2. Specyfikacja środowiska, w którym wykonano ćwiczenie

Ćwiczenie zostało wykonane przy pomocy jupyter notebook i Python 3 na komputerze z systemem Windows 10 (64 bitowym) i procesorem Intel Core i-5- 9300H (64 bitowy). Wykresy wygenerowano przy pomocy dostarczonego na laboratoria narzędzia graficznego i bibliotek matplotlib, random, math i sortedcontainers.

3. Sposób wykonania ćwiczenia

Na wykonanie ćwiczenia składały się następujące czynności:

- Zaimplementowanie funkcji umożliwiających wprowadzanie jak i generowanie losowych odcinków.
- Zaimplementowanie algorytmu sprawdzającego, czy w zbiorze odcinków choć jedna para się przecina.
- Zaimplementowanie algorytmu wyznaczającego wszystkie punkty przecięcia w zbiorze.

4. Algorytm sprawdzający, czy dowolne dwie pary odcinków się przecinają

W celu zaimplementowania algorytmu konieczne było zaimplementowanie dwóch struktur:

- **Struktury zdarzeń** zrealizowanej przy pomocy posortowanej listy krotek postaci (informacja, czy punkt jest początkiem odcinka; współrzędna x, dla której zdarzenie występuje; odcinek dla którego zdarzenie występuje). Listę posortowano ze względu na drugą wartość krotek. Wybór tej struktury spowodowany był faktem, że algorytm będzie przeglądał zdarzenia po kolei, a do listy nigdy nie zostanie dodane nowe zdarzenie- wykrycie przecięcia się odcinków skutkuje zakończeniem

algorytmu, nie ma więc konieczności dodawania odpowiedniego zdarzenia do struktury zdarzeń.

- **Struktury stanu** zrealizowanej przy pomocy klasy SortedSet z biblioteki sortedcontainers. Klasa jest realizowana przez drzewo czerwono- czarne, co umożliwia wstawianie, usuwanie i znalezienie poprzednika/ następnika obiektu w czasie O(logn).

  Omawiając strukturę stanu warto pochylić się nad sposobem w jaki ustalana jest kolejność odcinków w SortedSet. Aby porządek w strukturze był zachowany podczas wkładania i usuwania elementu ze struktury odcinki są porównywane ze względu na wartość współrzędnej y odcinka dla x, w którym zdarzenie ma miejsce (zakładając, że linie nie są równe tj. mają różne indeksy). Jest to możliwe dzięki dodaniu do klasy Line referencji do współrzędnej, po której linie powinny być porównywane – pola X będącego obiektem klasy pomocniczej X_cord. Wartość tego pola zawsze przyjmuje wartość współrzędnej x obecnie obsługiwanego zdarzenia.

Algorytm przyjmuje na wejściu listę odcinków w postaci listy punktów, każdy odcinek przetwarza na obiekt klasy Line, będący rozszerzeniem linii wzbogaconym o metody porównywania i zwrócenia hasza (metody oparte są o unikalny indeks linii przypisany podczas tworzenia obiektu, a jeśli ten jest różny- o wartości współrzędnych). Wprowadzenie wspomnianych funkcji było wymuszone na skutek używania SortedSet. Następnie algorytm szuka punktu przecięcia w oparciu o algorytm zamiatania, czyli przegląda zdarzenia i sprawdza, czy usunięcie/ dodanie odcinka do struktury zdarzeń skutkuje znalezieniem przecięcia:

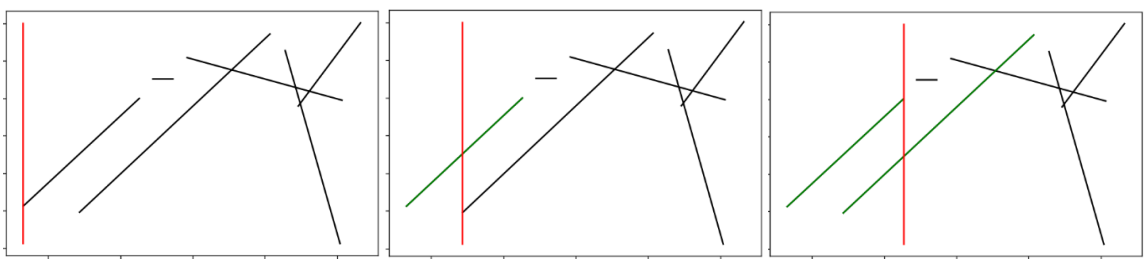
Jeśli zdarzeniem jest dodanie odcinka to sprawdzane jest, czy zachodzi przecięcie między dodawanym odcinkiem a odcinkami powyżej i poniżej w strukturze stanu.

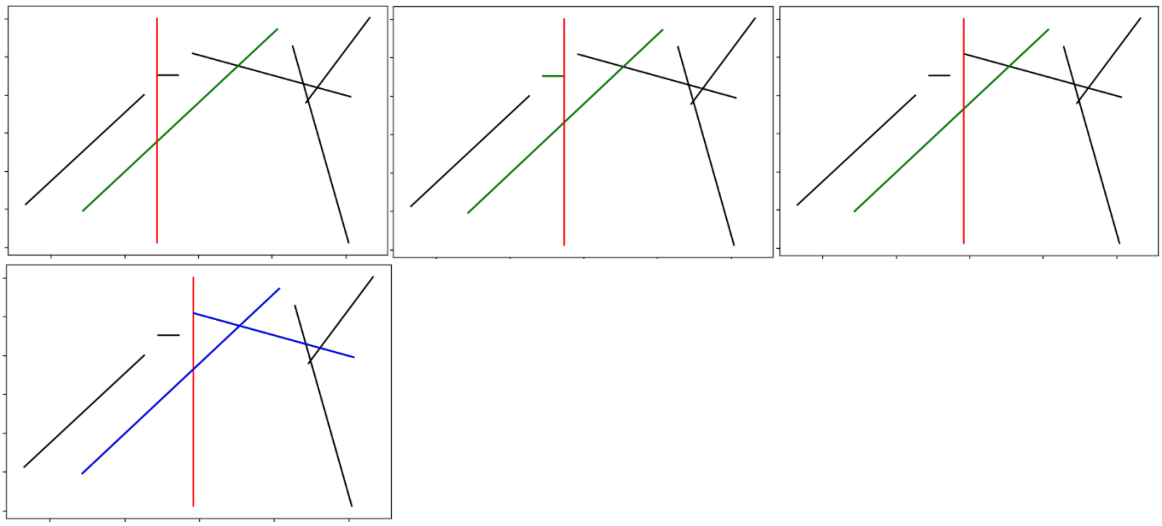
Jeśli zdarzeniem jest usunięcie odcinka to sprawdzane jest, czy zachodzi przecięcie pomiędzy odcinkami zlokalizowanym w strukturze stanu nad i pod usuwanym odcinkiem.

Na wyjściu algorytm zwraca krotkę postaci (True/False, scenes), gdzie pierwsza pozycja określa, czy występuje choć jedno przecięcie, a druga- zwraca sceny umożliwiające wizualizację działania algorytmu.

Działanie algorytmu można prześledzić na ilustracjach poniżej:

Zestaw rysunków 1. Przykładowe działanie algorytmu 1.





Na ilustracjach na czerwono pokolorowano miotłę, na zielono - odcinki znajdujące się obecnie w strukturze stanu miotły, a na niebiesko – odcinki dla których znaleziono punkt przecięcia.

5. Algorytm znajdujący wszystkie przecięcia w zbiorze odcinków

W celu zaimplementowania algorytmu konieczne było zaimplementowanie dwóch struktur:

Struktury zdarzeń zrealizowanej przy pomocy klasy SortedSet z biblioteki sortedcontainers sortującej elementy według ich współrzędnej x. Jak już wspomniano w punkcie 4. Klasa umożliwia szybkie wstawienie/ usunięcie elementu, jak i znalezienia najmniejszego z nich (korzystając z faktu, że elementy są posortowane). Zdarzenia będące elementami SortedSet zaimplementowane zostały jako klasa Line_event, która rozszerza zdarzenia z punktu 4. sprawozdania o porównywanie zdarzeń i funkcję haszującą (bazujące na unikalnych indeksach linii biorących udział w zdarzeniu). Dodatkowo klasa Line_event zawiera również pola przechowujące informację o współrzędnej y zdarzenia, jak i drugiej linii – pola te mają zastosowanie, gdy zdarzenie dotyczy przecięcia się linii, w przeciwnym przypadku ustawione są na None.

Struktury stanu zrealizowanej przy pomocy klasy SortedSet z biblioteki sortedcontainers w sposób podobny do tego jak miało to miejsce w algorytmu z punktu 4. Znaczącą różnicą jest rozbudowanie klasy o funkcje add, remove, intersection, które są wywoływane na skutek napotkania odpowiedniego zdarzenia. Dwie pierwsze spośród nich w poprzednim algorytmie były realizowane przez główną funkcję. Teraz odpowiedzialna za nie jest sama struktura, a znalezienie przecięcia jest obsługiwane poprzez dodanie do struktury zdarzeń odpowiedniego zdarzenia, nie zaś skończenia programu.

Algorytm na wejściu przyjmuje zestaw odcinków, przetwarza je na obiekty klasy Line, a następnie przegląda zdarzenia ze struktury zdarzeń, aż do momentu przeglądnięcia wszystkich. Warto zaznaczyć, że w przeciwieństwie do algorytmu z punktu 4. konieczne jest tworzenie i wstawianie do struktury zdarzeń informacji o przecięciach odcinków, wobec czego lista nie jest dobrą kandydatką na strukturę zdarzeń (zakładając, że wstawianie do niej nie jest oparte o wyszukiwanie połówkowe). Tworzenie takich zdarzeń i umieszczanie ich w strukturze ma miejsce, podczas modyfikacji struktury stanu, gdy wykryto przecięcie (pod warunkiem, że zdarzenie nie zostało już wcześniej wykryte tj. nie znajduje się w strukturze zdarzeń ani w wynikowym zbiorze przecięć. Takie sprawdzenie jest realizowane efektywnie dzięki zastosowaniu SortedSet dla struktury zdarzeń i zbioru wynikowego punktów przecięcia). Poszczególne zdarzenia obsługiwane są w następujący sposób:

- Zdarzenie początku odcinka - polega na wstawieniu odcinka do struktury stanu i sprawdzeniu, czy nie przecina on swojego poprzednika lub następnika w strukturze. Jeśli przecięcie ma miejsce należy dodać odpowiednie zdarzenie przecięcia do struktury zdarzeń.

- Zdarzenie końca odcinka - polega na usunięciu odcinka ze struktury stanu i sprawdzeniu, czy odcinki, „oddzielane” w strukturze stanu przez usunięty odcinek się przecinają. Jeśli przecięcie ma miejsce należy dodać odpowiednie zdarzenie przecięcia do struktury zdarzeń.

- Zdarzenie przecięcia odcinków – polega na:

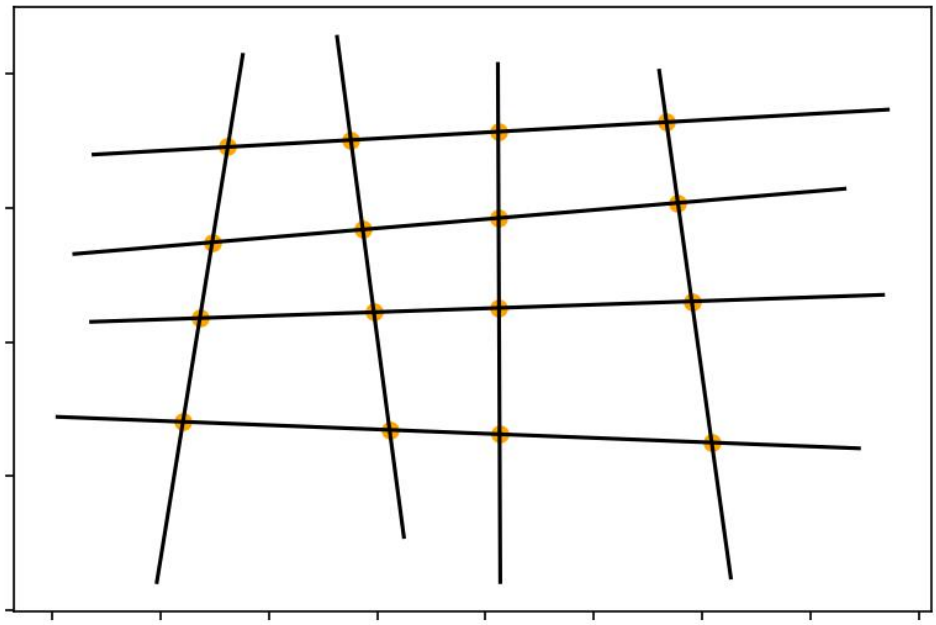
  - usunięciu odcinka - A znajdującego się wyżej (w miejscu bezpośrednio przed przecięciem) w strukturze stanu,
  - sprawdzeniu czy niższy odcinek - B przecina odcinek, od którego był „oddzielany” przez odcinek A,
  - usunięciu odcinka B ze struktury,
  - wstawieniu do struktury odcinka A i sprawdzeniu, czy przecina on odcinek, który przed rozpoczęciem procesu znajdował się bezpośrednio pod odcinkiem B,
  - wstawieniu do struktury odcinka B w taki sposób, by w strukturze stanu znajdował się nad odcinkiem A. Jest to realizowane poprzez zwiększenie wartości X_cord (opisanej w punkcie 4. Sprawozdania) o wartość epsilon taką, by kolejność odcinków została zmieniona, a na fragmencie (x, x+ epsilon] nie znajdowały się żadne zdarzenia.

Na wyjściu algorytm zwraca krotkę zawierającą zbiór zdarzeń przecięcia (jako obiektów klasy Line_event) i scen umożliwiających wizualizację przebiegu algorytmu.

6. Wyniki algorytmów dla przetestowanych danych

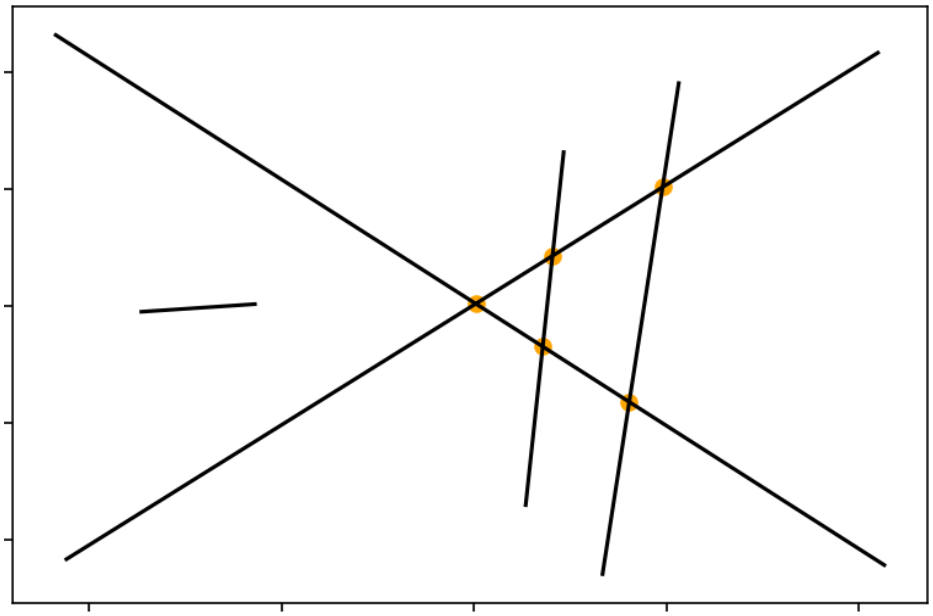
Poniżej zaprezentowano przykładowe ilustracje zawierające punkty przecięcia znalezione przez algorytm (zaznaczone na pomarańczowo):

Rys 1. Punkty przecięcia dla danych testowych 1.



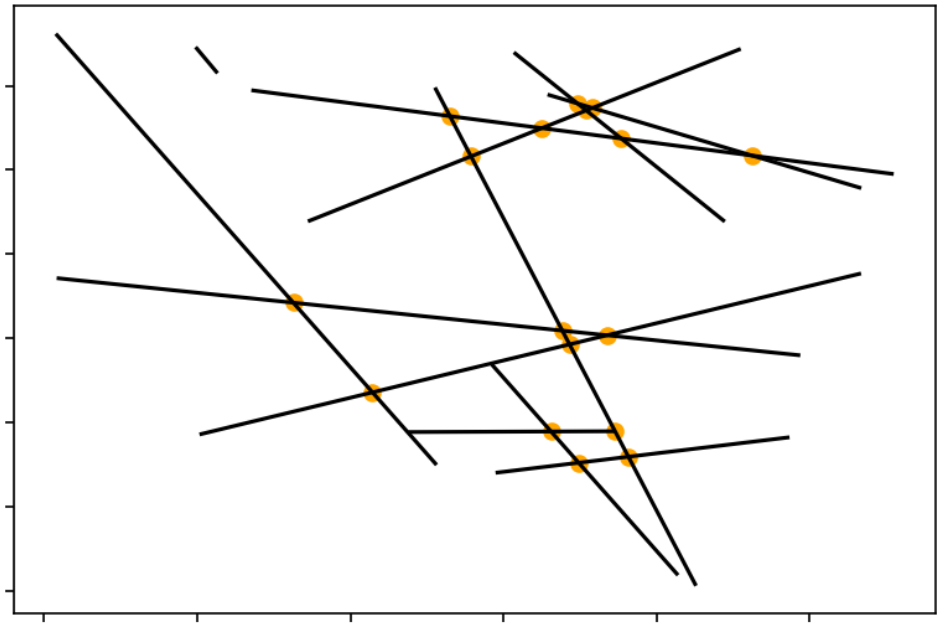
Liczba znalezionych punktów 16 jest zgodna z rzeczywistą. (Pierwsza wersja algorytmu zwróciła wartość True.)

Rys 2. Punkty przecięcia dla danych testowych 2.



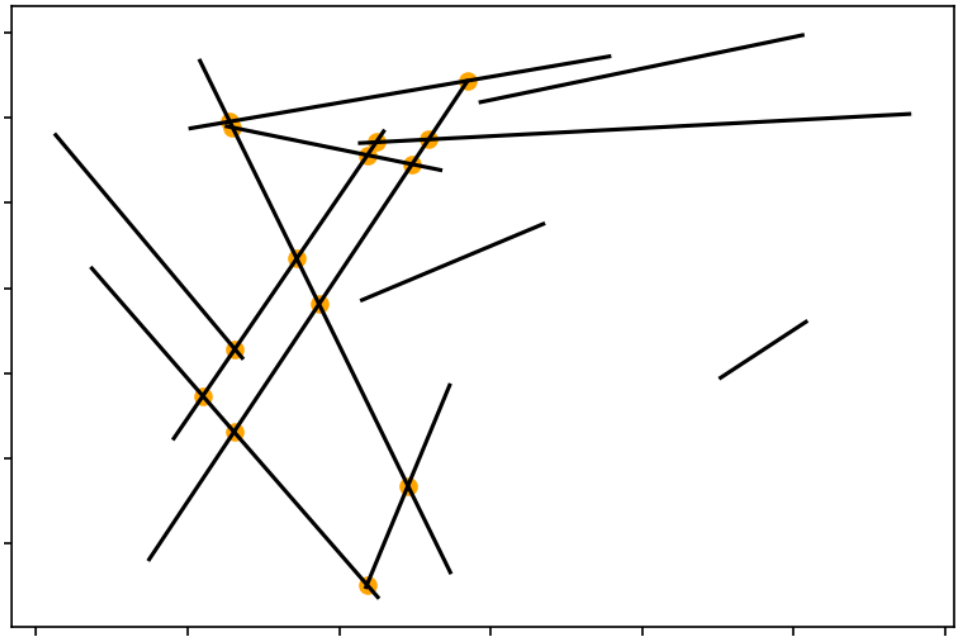
Liczba znalezionych punktów 5 jest zgodna z rzeczywistą. (Pierwsza wersja algorytmu zwróciła wartość True). Celem tego testu było sprawdzenie, czy algorytm zlicza kilkukrotnie te same przecięcia (potencjalnie takim przecięciem mogło być pierwsze od lewej – zostało wykryte zarówno podczas początku drugiej (względem x) linii jak i podczas zakończenia 3. linii). (Pierwsza wersja algorytmu zwróciła wartość True.)

Rys 3. Punkty przecięcia dla danych testowych 3.



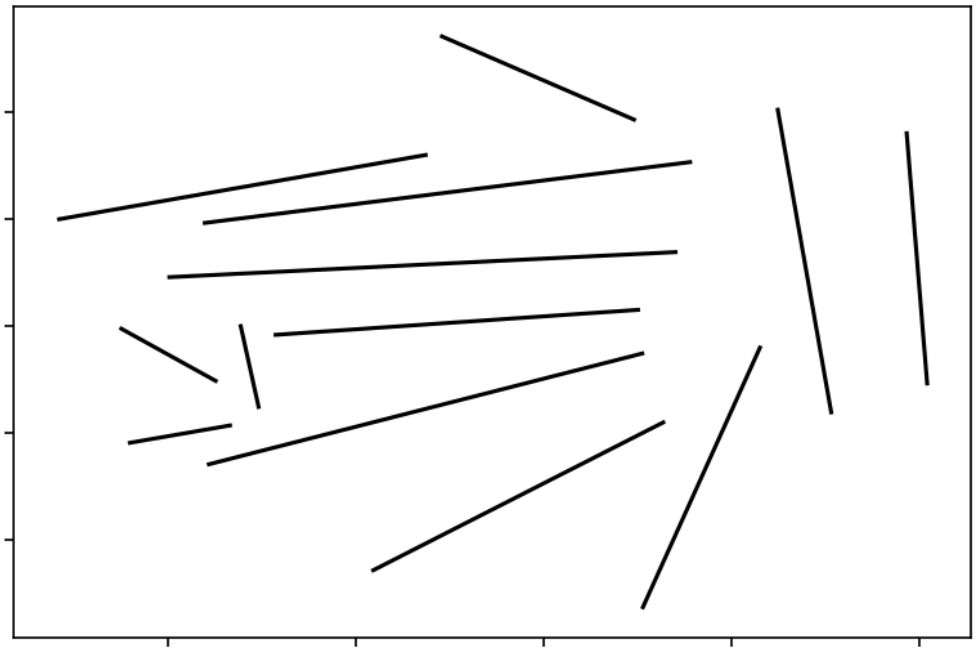
Liczba znalezionych punktów 17 jest zgodna z rzeczywistą. (Pierwsza wersja algorytmu zwróciła wartość True.)

Rys 4. Punkty przecięcia dla danych testowych 4.



Liczba znalezionych punktów 14 jest zgodna z rzeczywistą. (Pierwsza wersja algorytmu zwróciła wartość True.)

Rys 5. Punkty przecięcia dla danych testowych 5.



Liczba znalezionych punktów 0 jest zgodna z rzeczywistą. (Pierwsza wersja algorytmu zwróciła wartość False.)

Wszystkie powyższe wyniki są zgodne z oczekiwanymi. Dane testowe, dla których sporządzono powyższe ilustracje znajdują się na końcu notebooka dostarczonego wraz ze sprawozdaniem.