Sprawozdanie Algorytmy Geometryczne

Ćwiczenie 4 przecinanie się odcinków

Norbert Wolniak
Grupa Czw_16.15_A

1. Cel ćwiczenia:
   Celem ćwiczenia jest zapoznanie się oraz zaimplementowanie algorytmów do wykrywania czy istnieje choć jedna para odcinków przecinających się oraz do wyznaczania przecięć odcinków na płaszczyźnie.

2. Uwagi techniczne:
   Ćwiczenie zostało przeprowadzone w oprogramowaniu Jupyter Notebook, na systemie Windows 10 Home, komputer wyposażony w procesor x64 przy użyciu proponowanego narzędzia graficznego, które napisane jest w języku Python3 i wykorzystuje bibliotekę MatPlotLib. Do liczenia wyznacznika użyto wyznacznika 3x3 zaimplementowanego samemu przy tolerancji dla zera $e=10^{-13}$, który okazał się najlepszy w ćwiczeniu 1. Dodatkowe biblioteki to queue oraz sortedcontainers dostarczające kolejkę priorytetową oraz posortowany zbiór.

3. Wprowadzanie odcinków:
   - Procedura pozwalająca wprowadzać odcinki interaktywnie przy pomocy narzędzia graficznego.
   - Procedura wyznaczająca losowo odcinki przy zadanej ilości odcinków oraz obszaru w którym mają być losowane.

   Odcinki pionowe oraz posiadające wspólne końce są eliminowane.

4. Algorytm nr 1 sprawdzający czy choć jedna para odcinków się przecina:
   Stan miotły - SortedSet() z biblioteki sortedcontainers, odcinki posortowane są w kolejności rosnących współrzędnych y-owych początkowych punktów odcinka.
   Struktura zdarzeń - PriorityQueue() z biblioteki queue, zdarzenia posortowane są w kolejności rosnących współrzędnych x-owych.
   Obie struktury zapewniają poprawne działanie algorytmu, ponieważ algorytm kończy się przy pierwszym znalezionym przecięciu.

5. Algorytm nr 2 wykrywający wszystkie przecięcia odcinków:
   Stan miotły - SortedSet() z biblioteki sortedcontainers, odcinki posortowane są w kolejności aktualnych współrzędnych y-owych odcinków przecinających miotłę.
   Struktura zdarzeń - SortedSet() z biblioteki sortedcontainers, zdarzenia posortowane są w kolejności rosnących współrzędnych x-owych.
   W porównaniu do algorytmu nr 1 struktura zdarzeń uległa zmianie, ponieważ SortedSet() zapewnia unikalność elementów wraz z dodatkową strukturą set(), która przetrzymuje wszystkie punkty przecięć na lewo od miotły mamy pewność, że żaden punkt przecięcia nie będzie wyznaczony drugi lub więcej razy.
   Stan miotły nie uległ zmianom, jednak do poprawnego działania algorytmu zaimplementowałem dodatkową klasę Line(), w której również zawarty jest klucz sortowania dla aktualnego stanu miotły - aktualna współrzędna x-owa miotły pozwala na poprawne wstawienie odcinka w posortowanej kolejności przecięć z miotłą od “dołu do góry”. Zatem elementy stanu miotły różnią się w dwóch algorytmach, jednak w strukturze zmianie uległ tylko klucz sortowania.

   W obu algorytmach obie struktury niekoniecznie muszą być takie same, lecz mogą być przy odpowiednim zaimplementowaniu. W moim przypadku w algorytmie nr 2 struktura zdarzeń może być PriorityQueue() jak w algorytmie nr 1, jednak wymagany byłby zbiór który przetrzymuje wszystkie punkty przecięć już wyznaczone po lewej jak i po prawej stronie miotły aby nie dochodziło do wielokrotnego wstawienia punktu do kolejki.

6. Wyznaczanie punktu przecięcia:
   Przy użyciu współrzędnej kierunkowej, wyrazu wolnego prostej oraz współrzędnej x-owej punktu przecięcia.

7. Obsługiwanie zdarzeń:
   Początek odcinka :
   1. Wstawiamy odcinek do stanu miotły.
   2. Jeśli istnieją odcinki bezpośrednio poniżej lub powyżej wstawianego odcinka to wykrywamy możliwe przecięcia z wstawionym odcinkiem.

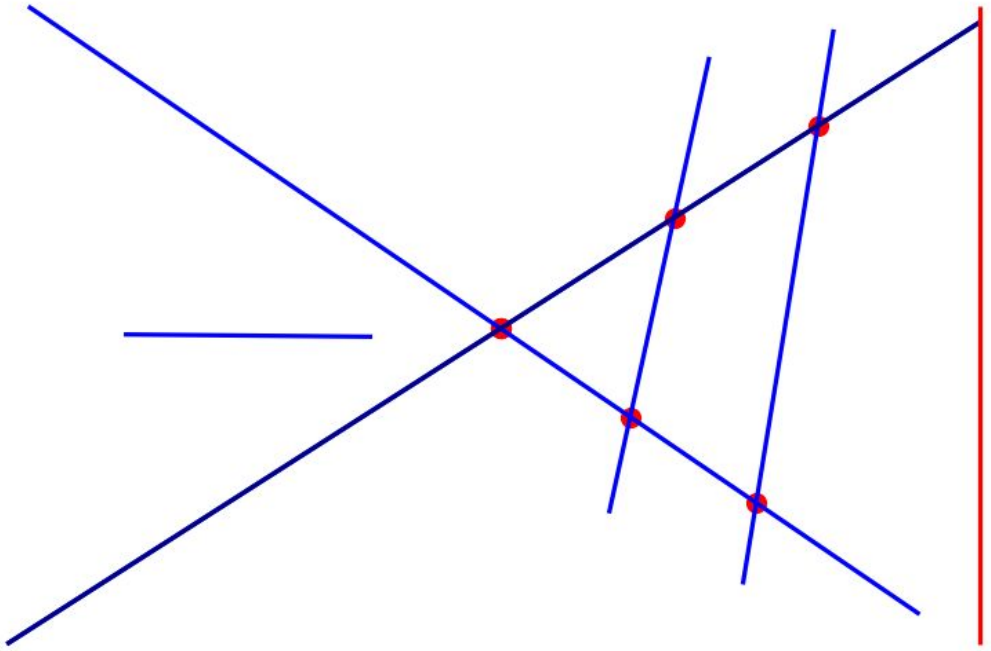
   Koniec odcinka :
   1. Jeśli istnieją odcinki bezpośrednio poniżej lub powyżej danego odcinka to wykrywamy możliwe ich przecięcie.
   2. Usuwamy odcinek, którego końcem jest dane zdarzenie (koniec odcinka).

   Punkt przecięcia :
   1. Zamieniamy kolejność dwóch odcinków na lewo od miotły, tak aby na prawo od miotły miały zmienioną kolejność.

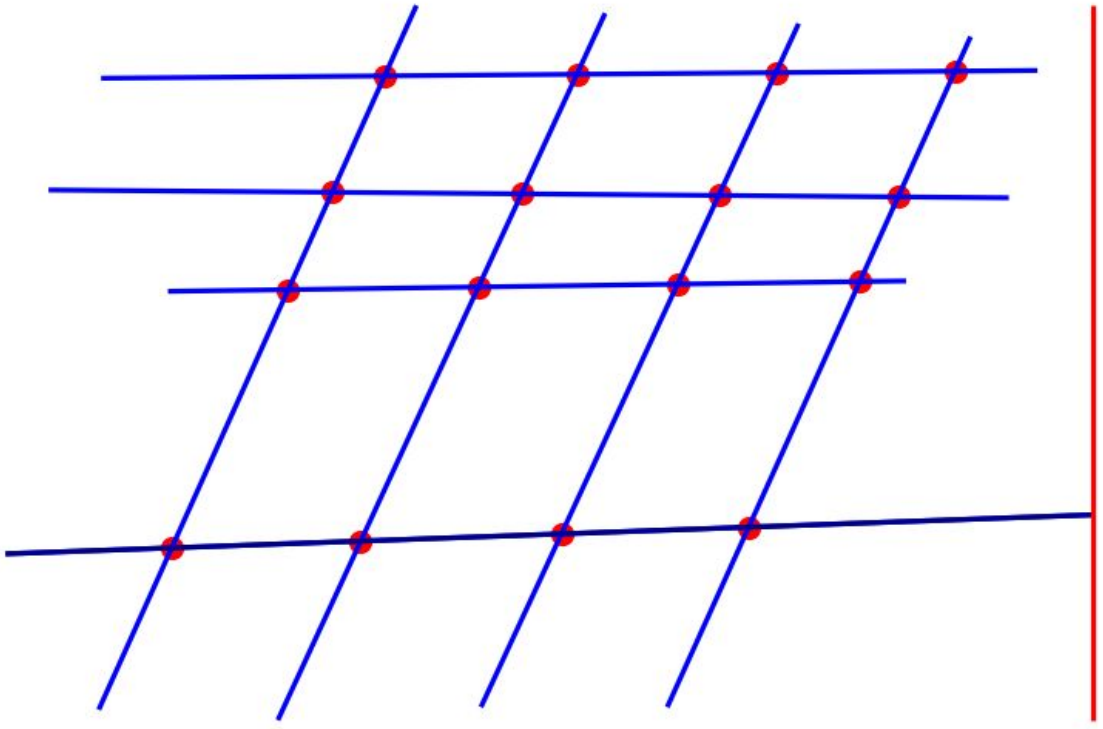
2. Wykrywamy możliwe przecięcia zmienionych odcinków z ich bezpośrednimi sąsiadami odpowiednio od ‘góry’ i od ‘dołu’.

8. Przykłady :

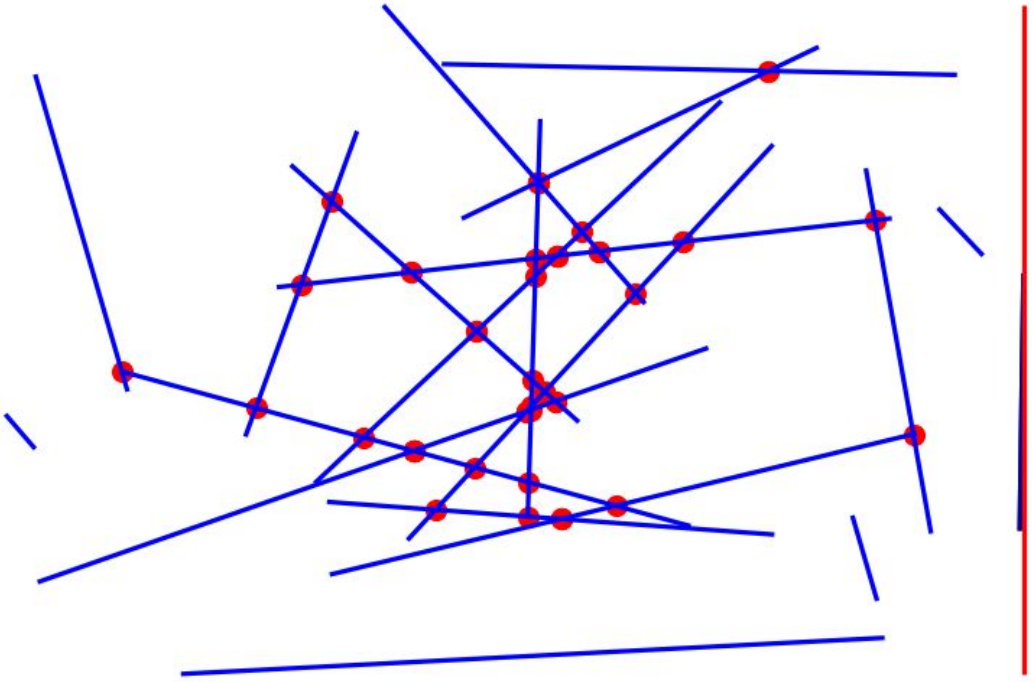
Rysunek nr 1 : 5 punktów przecięcia.



Rysunek nr 2 : 16 punktów przecięć.



Rysunek nr 3: Losowe odcinki - 33 punkty przecięć.



9. Zwracana struktura:
   Algorytm zwraca listę zawierającą dla każdego punktu przecięcia:
   1. Współrzędne punktu przecięcia.
   2. Dwa odcinki przecinające się w danym punkcie.

10. Podsumowanie :
    Zaimplementowane algorytmy do wyznaczania przecięć odcinków działają poprawnie oraz umożliwiają graficzną prezentację poszczególnych kroków algorytmów.