Algorytmy geometryczne - Ćwiczenie 4. Paulina Jędrychowska

1. Dane techniczne urządzenia i wykorzystane narzędzia:

System Windows 10 x64

Procesor i5-9300H

Pamięć RAM 16GB

Środowisko Jupyter Notebook

Język programowania Python 3.0

2. Tworzenie figur:

Aplikacja umożliwia tworzenie odcinków: za pomocą myszki zaznaczamy wierzchołki początkowe i końcowe kolejnych odcinków. Następnie zbiory odcinków można zapisać do pliku typu json. Przed zapisaniem program usuwa odcinki nie zgodne z założeniami zadania.

Usuwane typy odcinków to: odcinki pionowe oraz dla każdej pary odcinków, w której któraś ze współrzędnych x jest taka sama, usuwany jest jeden z odcinków.

Zapisane zbiory odcinków można odczytywać.

Stworzone zbiory w celu przetestowania algorytmu:

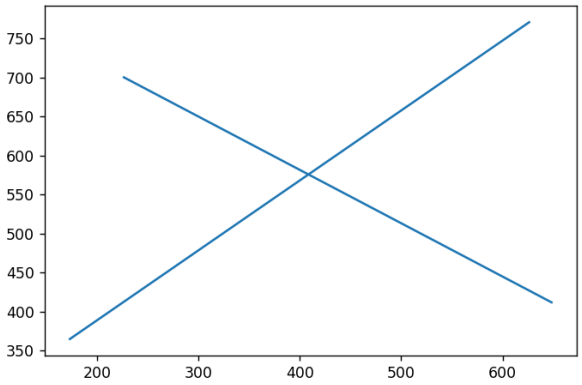
- Krzyż („cross”)
- Zbiór nieprzecinających się odcinków („nonintersecting”)
- Zbiór zaproponowany na ćwiczeniach („zbior_z_cwiczen”)
- Kot („cat”)
- Drabina („ladder”)
- Gwiazda („star”)
- Odcinki pokrywające się („overlap”)
- Potrójne przecięcie („triple_intersection”)

Dodatkowo program umożliwia generowanie zadaną liczbę losowych odcinków z podanego zakresu. Po wygenerowaniu odcinków usuwane są te, które nie spełniają warunków zadania, w taki sam sposób jak dla odcinków narysowanych.

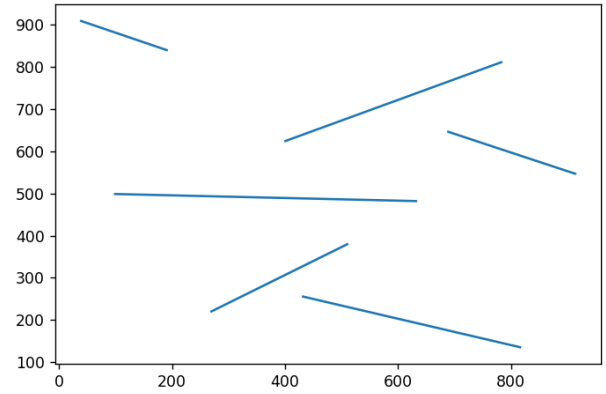
- Zbiór losowo wygenerowanych odcinków („random_lines”)

Wszystkie zbiory zostały zwizualizowane przez dostarczone narzędzie graficzne oparte o bibliotekę matplotlib.

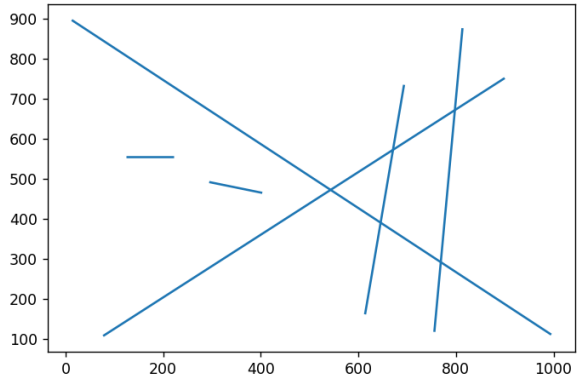
Wykres 1.1. Krzyż



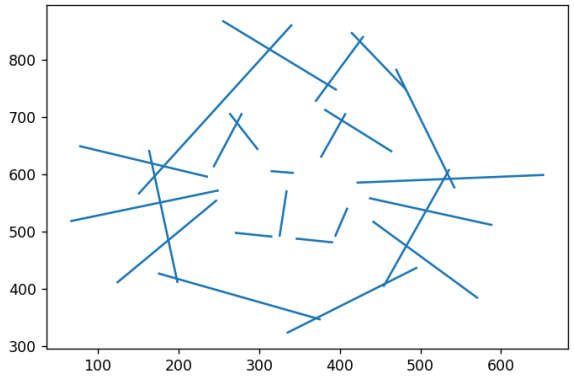
Wykres 1.2. Nieprzecinające się odcinki



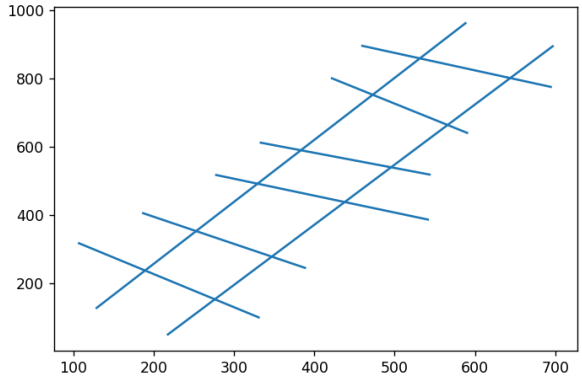
Wykres 1.3. Zbiór z ćwiczeń



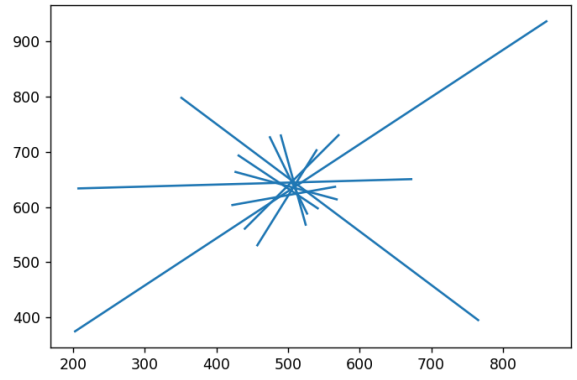
Wykres 1.4. Kot



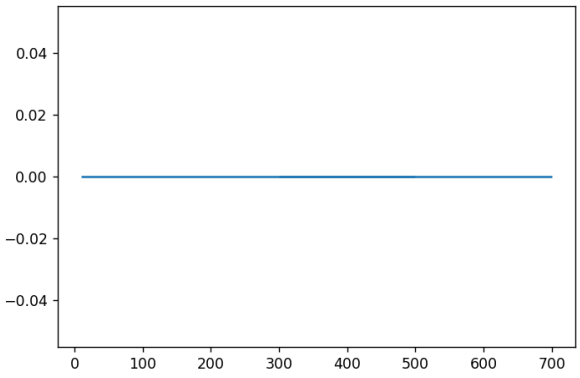
Wykres 1.5. Drabina



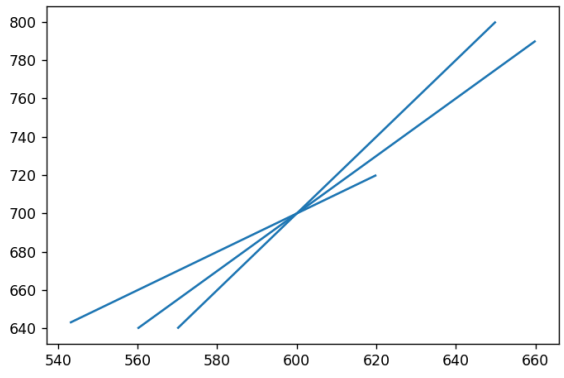
Wykres 1.6. Gwiazda



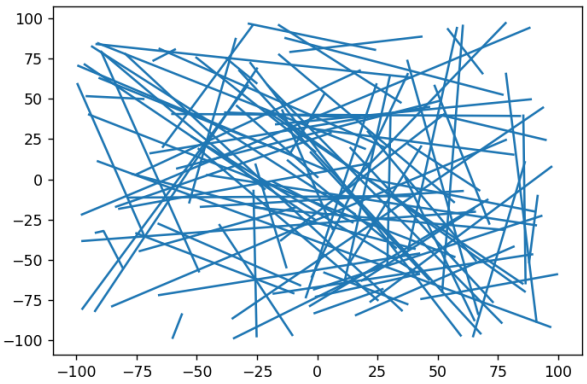
Wykres 1.7. Pokrywające się odcinki



Wykres 1.8. Potrójne przecięcie



Wykres 1.1. Zbiór 100 losowych linii z zakresu [-100, 100]



3. Struktury danych:

- Struktura zdarzeń Q – kolejka priorytetowa (PriorityQueue z biblioteki queue), jest to struktura oparta o zrównoważone drzewo binarne, w której operacje wstawiania i usuwania mają złożoność logarytmiczną. Struktura ta umożliwia szybkie i proste uzyskanie wartości o najmniejszym priorytecie. Do kolejki wsadzamy obiekty reprezentujące punkty. Zwracany jest punkt o najmniejszej współrzędnej x.
- Struktura stanu T – zbiór uporządkowany (SortedSet z biblioteki sortedcontainers), jest to struktura oparta o zrównoważone drzewo binarne, w której operacje wstawiania i usuwania mają złożoność logarytmiczna. Struktura ta trzyma wartości uporządkowane względem klucza. W zbiorze trzymamy obiekty reprezentujące aktualnie przetwarzane linie, są one uporządkowane według wartości y przy aktualnym położeniu miotły. Jeżeli współrzędne y są takie same (miotła znajduje się w punkcie przecięcia), to większy jest punkt, którego współczynik a jest większy. Wiemy, że współczynnik, a jest kątem nachylenia względem osi ox, a więc prosta z większym współczynnikiem a będzie się znajdować wyżej za punktem przecięcia.
- Stan miotły – obiekt Broom, przechowuje wartość x, która jest aktualizowana po przejściu na kolejny punkt w strukturze zdarzeń
- Punkty – obiekt Point, przechowuje wartość x, wartość y, dwa odcinki, do których należy (jeżeli należy do jednego odcinka to oba przechowywane odcinki są tym samym) oraz typ punktu, w formie stringa „start” – punkt początkowy, „end” – punkt końcowy, „intersection” – przecięcie odcinków.
- Odcinki – obiekt Line, przechowuje swój index w tablicy lines, punkt początkowy A oraz punkt końcowy B (jako obiekty typu Point), współczynnik a oraz współczynnik b dla równania prostej y = ax + b, oraz aktualny stan miotły.

4. Wykrywanie punktów przecięć:

Wykrywanie punktów przecięć jest realizowane za pomocą wyznaczników:

Wzór 1. Metoda wyznacznika, dla prostych wyrażonych równaniem ogólnym

Dla prostych k, l danych równaniami

$$k: A_1x + B_1y + C_1 = 0,$$
$$l: A_2x + B_2y + C_2 = 0,$$

niech

$$W = \begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix} = A_1 \cdot B_2 - A_2 \cdot B_1$$
$$W_x = \begin{vmatrix} B_1 & C_1 \\ B_2 & C_2 \end{vmatrix} = B_1 \cdot C_2 - B_2 \cdot C_1$$
$$W_y = \begin{vmatrix} C_1 & A_1 \\ C_2 & A_2 \end{vmatrix} = C_1 \cdot A_2 - C_2 \cdot A_1$$

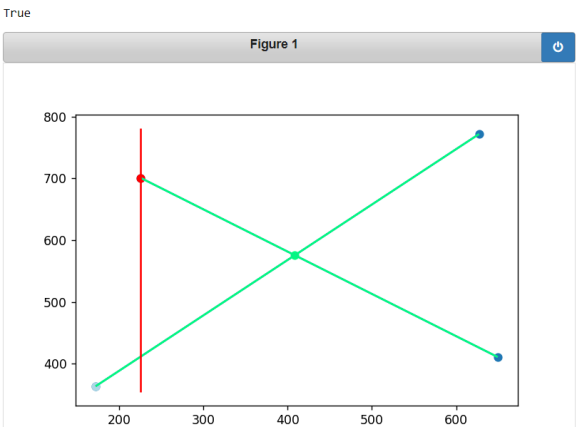
Warunkiem koniecznym i wystarczającym, aby proste się przecinały jest, by W ≠ 0. Punkt przecięcia ma wtedy wartość (x, y) = $\left(\frac{W_x}{W}, \frac{W_y}{W}\right)$.

W algorytmie dla sprawdzanych odcinków znajdujemy punkt przecięcia dwóch prostych na których one leżą i zwracamy ten punkt jeżeli: leży on na obu odcinkach oraz znajduje się on za miotłą.

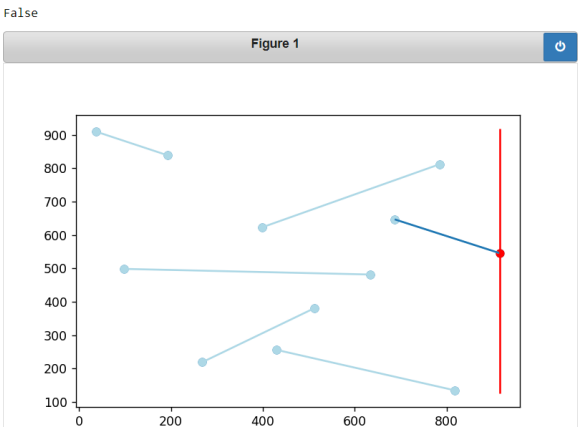
5. Algorytm, sprawdzający czy chodź jedna para odcinków się przecina:

Algorytm, jest zaimplementowany w taki sam sposób jak algorytm znajdujący wszystkie przecięcia, tylko po znalezieniu przecięcia jest on zakańczany i zwracana jest odpowiedni komunikat. Wykorzystane struktury w obu algorytmach były takie same. Algorytm poprawnie wykrywa istnienie lub brak przecięć.

Wykres 2.1. Wynajdowanie jednego punktu przecięć dla zbioru Krzyż

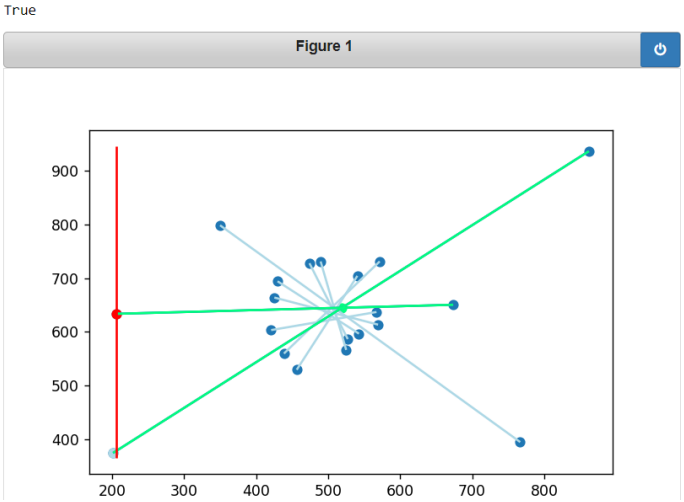


Wykres 2.2. Wynajdowanie jednego punktu przecięć dla braku przecięć



Wykres 2.3. Wynajdowanie jednego punktu przecięć

dla zbioru Gwiazda



6. Algorytm wynajdujący wszystkie punkty przecięcia:

Strukturę zdarzeń inicjujemy punktami początkowymi i końcowymi wszystkich prostych.

W głównej pętli algorytmu położenie miotły jest aktualizowane do pierwszego wierzchołka ze struktury stanu. Po dojściu miotły do kolejnego punktu zdarzenia mamy trzy możliwości:

- Zdarzenie jest początkiem odcinka – Dodajemy ten odcinek do struktury zdarzeń. Znajdujemy w strukturze zdarzeń odcinek leżący poniżej i powyżej (o ile istnieje) i sprawdzamy czy mają one punkty przecięcia. Znalezione punkty przecięcia dodajemy do struktury stanu.
- Zdarzenie jest końcem odcinka – Znajdujemy w strukturze zdarzeń odcinki leżący poniżej i powyżej rozpatrywanego odcinka (o ile istnieją) i sprawdzamy czy mają one punkt przecięcia, znaleziony punkt przecięcia dodajemy do struktury stanu o ile go jeszcze nie ma. Usuwamy rozpatrywany odcinek ze struktury zdarzeń.
- Zdarzenie jest punktem przecięcia – Cofamy miotłę, móc usunąć przecinające się odcinki. Po usunięciu aktualizujemy miotłę do punktu przecięcia i dodajemy odcinki z powrotem. Dla odcinków, które zmieniły swoje położenie sprawdzamy ich przecięcia z nowymi sąsiadami. Znalezione punkty przecięcia dodajemy do struktury stanu o ile go jeszcze nie ma.

Znalezione punkty przecięcia trzymamy w uporządkowanym zbiorze, dzięki czemu sprawdzenie czy jakiś wynik już wystąpił jest wykonywane w czasie O(1). Dodawanie przebiega w czasie O(logn), dodanie wszystkich wierzchołków będzie zatem wykonane w czasie O(Plogn), gdzie P to punkty przecięcia. Złożoność czasowa dla aktualizacji struktury stanu jest również O(Plogn). Zatem przechowywanie wyników w zbiorze uporządkowanym nie psuje złożoności, a wyniki są uporządkowane w ładny sposób i sprawdzanie czy dany wierzchołek już wystąpił jest wygodne.

W rezultacie algorytmu uzyskujemy: listę punktów przecięć, listę krotek składających się z punktów przecięć i obu przecinających się odcinków, wykres kolejnych kroków algorytmu dla danego zbioru.

6. Wizualizacja przebiegu algorytmu:

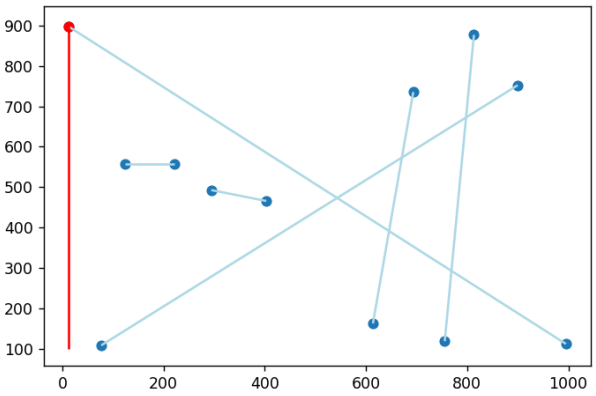
Legenda oznaczeń wykresów:

1. Wierzchołki:
   - Granatowy – wierzchołki wejściowe, przed przetworzeniem
   - Błękitny – wierzchołki wejściowe, po przetworzeniu
   - Zielony – znalezione punkty przecięcie
   - Czerwony – punkt który przecina miotła
2. Odcinki:
   - Błękitny – odcinki znajdujące się w zbiorze
   - Granatowy – odcinki aktualnie znajdujące się w strukturze zdarzeń
   - Zielony – odcinki, dla których w danym momencie znaleziono punkt przecięcia
   - Czerwony – miotła

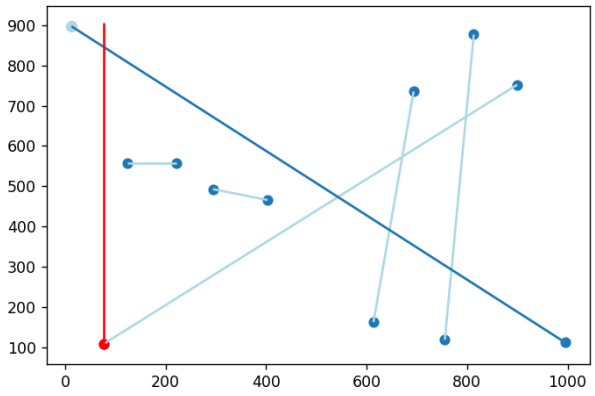
Wykres 3. Wizualizacja algorytmu dla zbioru

zaproponowanego na ćwiczeniach (z pominięciem nieciekawych scen)

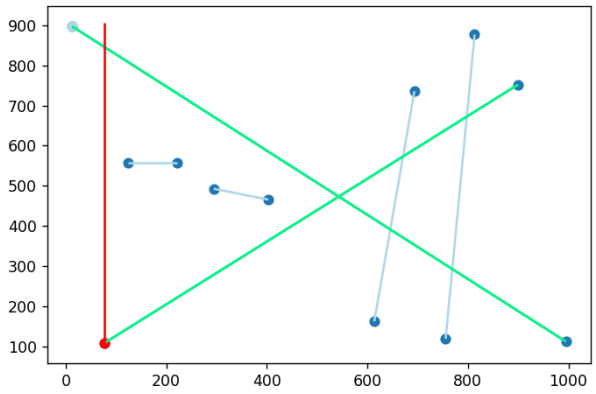
Wykres 3.1. Wszystkie punkty znajdują się w strukturze stanu



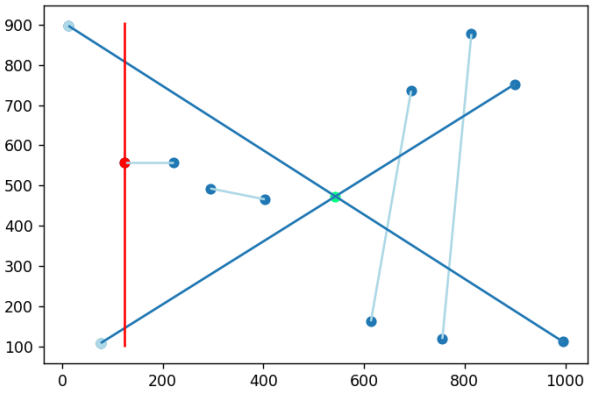
Wykres 3.2. Dodanie pierwszego odcinka do struktury zdarzeń, miotła przesuwa się



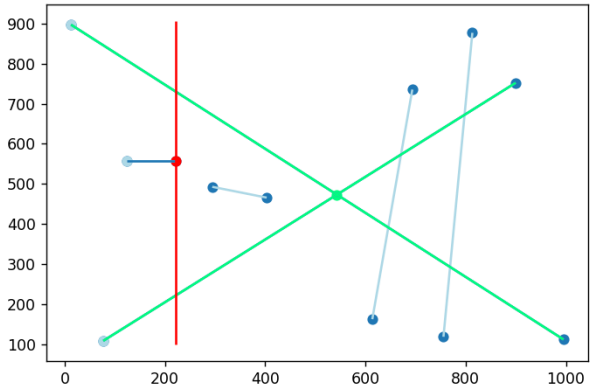
Wykres 3.3. Dla następnego odcinka znajdujemy punkt przecięcia



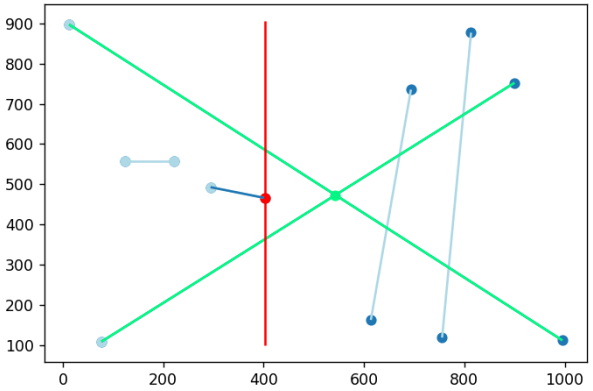
Wykres 3.4. Miotła przesuwa się, rozpatrujemy kolejny odcinek



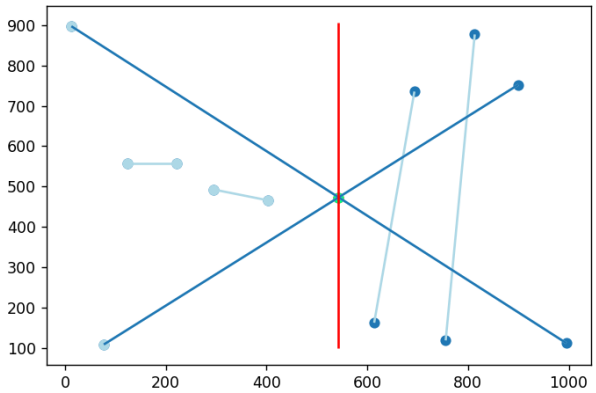
Wykres 3.5. Przy usuwaniu odcinka ze struktury zdarzeń punkt przecięcia jest wykrywany ponownie



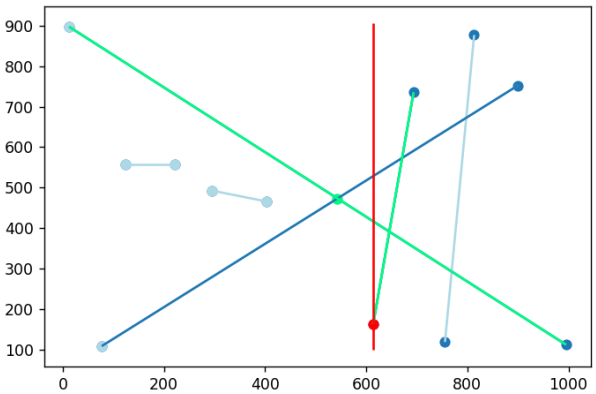
Wykres 3.6. Dla kolejnego odcinka dzieje się to samo



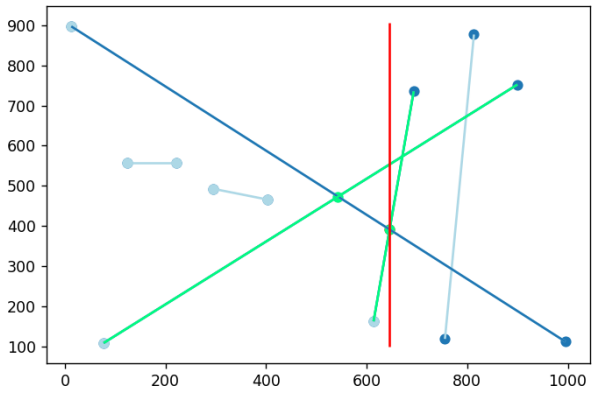
Wykres 3.7. Miotła zatrzymuje się na punkcie przecięcia



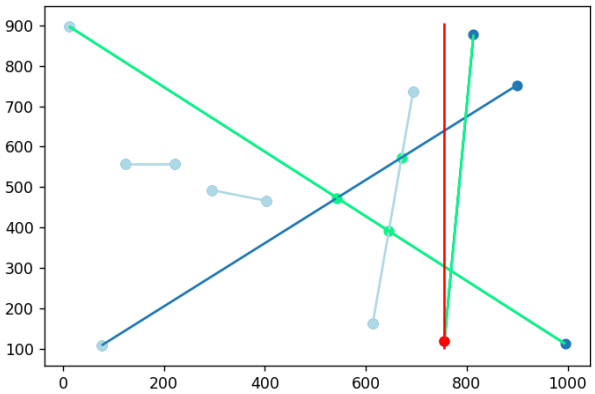
Wykres 3.8. Dla kolejnego dodanego odcinka znajdowany jest punkt przecięcia



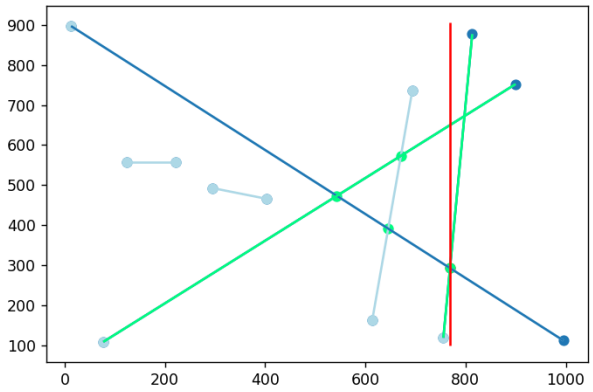
Wykres 3.9. Miotła zatrzymuje się w punkcie przecięcia, znajdowany jest kolejny punkt przecięcia



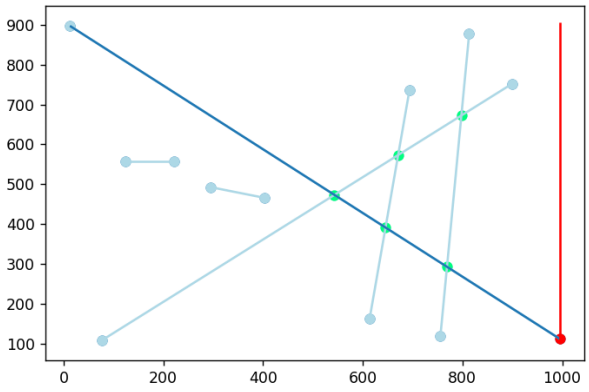
Wykres 3.10. Dla kolejnego odcinka znajdowany jest punkt przecięcia



Wykres 3.11. Miotła zatrzymuje się w punkcie przecięcia, znajdowany jest kolejny punkt przecięcia

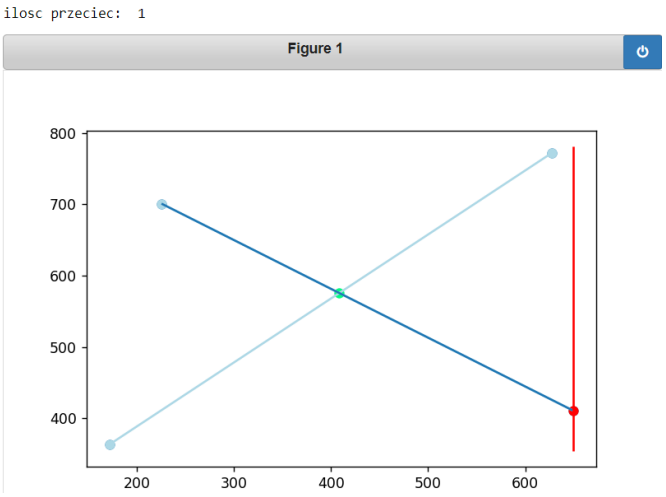


Wykres 3.12. Miotła przesuwa się do końca, nie znajduje już więcej punktów przecięć



Wyniki algorytmu dla przykładowych zbiorów:

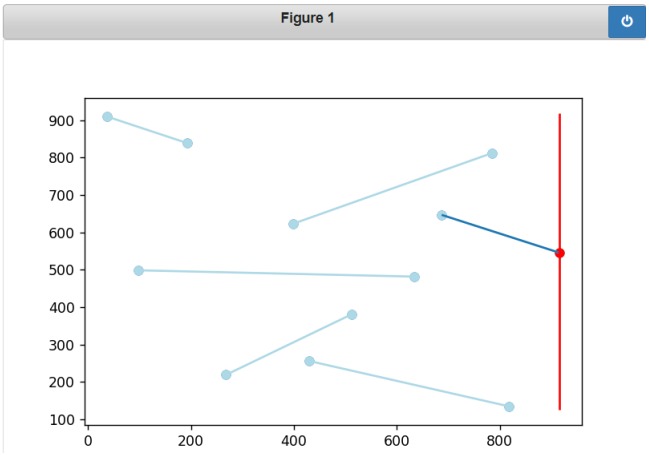
Wykres 4.1. Wynik algorytmu dla zbioru Krzyż



Wykres 4.2. Wynik algorytmu dla

zbioru nieprzecinających się odcinków

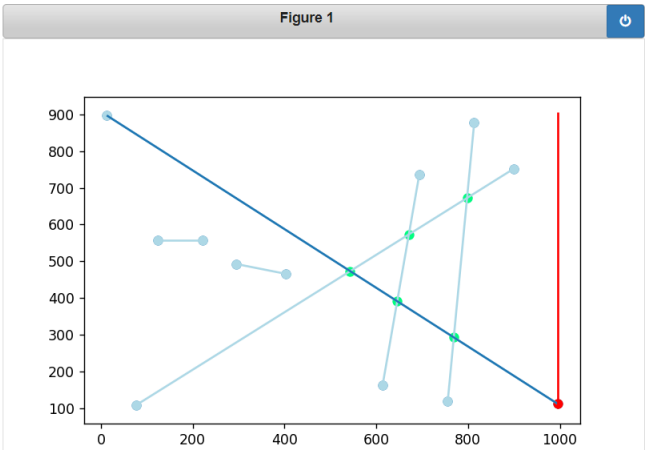
```
ilosc przeciec:  0
```



Wykres 4.3. Wynik algorytmu dla

zbioru zaproponowanego na ćwiczeniach

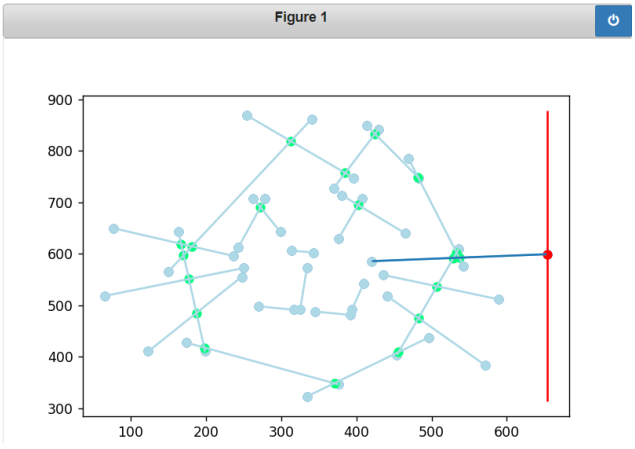
```
ilosc przeciec:  5
```



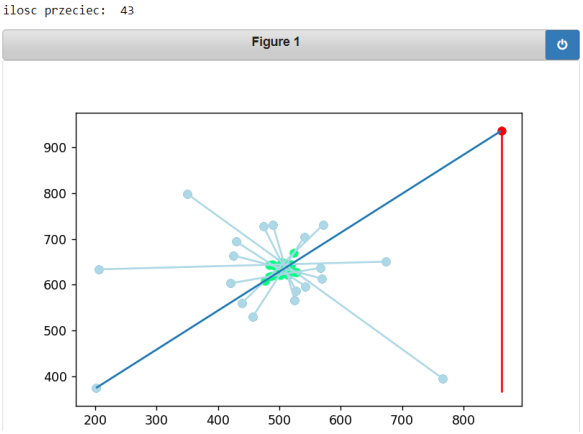
Wykres 4.4. Wynik algorytmu dla

zbioru Kot

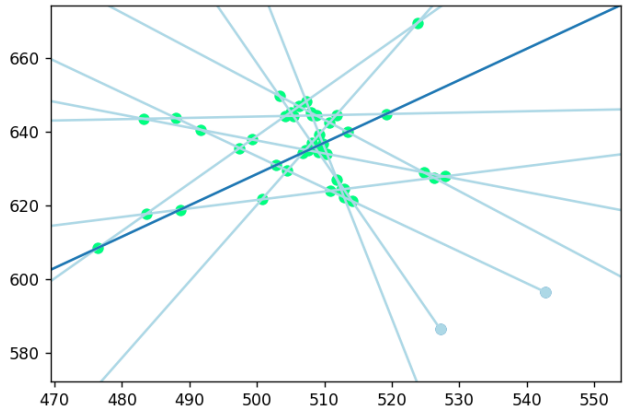
```
ilosc przeciec:  19
```



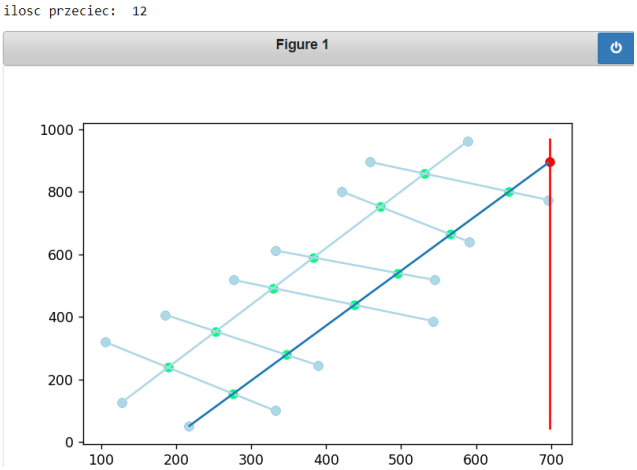
Wykres 4.5. Wynik algorytmu dla zbioru Gwiazda



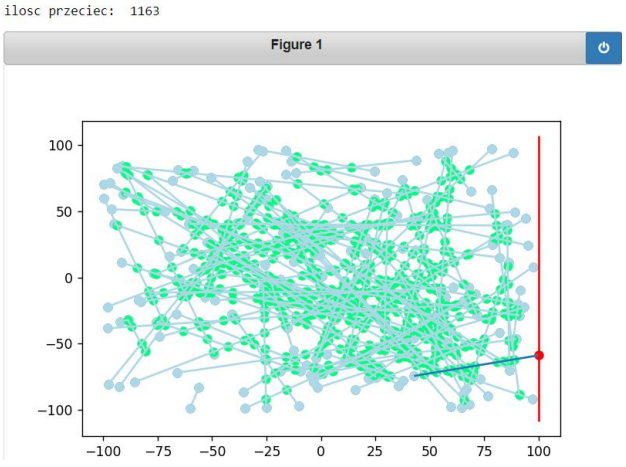
Wykres 4.6. Przybliżenie wyniku dla zbioru Gwiazda



Wykres 4.7. Wynik algorytmu dla zbioru Drabina



Wykres 4.8. Wynik algorytmu dla zbioru 100 losowych odcinów z zakresu [-100, 100]



Zbiór odcinków pokrywających się oraz zbiór z trzema odcinkami przecinającymi się w jednym miejscu nie spełniają założeń zadania. Algorytm dla obu tych zbiorów nie wykona się i wyświetli błąd. Dla odcinków nachodzących na siebie wyeliminowanie tych odcinków przy tworzeniu zbioru jest możliwe, ale na tyle skomplikowane i ten przypadek jest na tyle rzadki, że zakładam iż w zbiorze nie może być takich odcinków. Dla trzech odcinków przecinających się w jednym miejscu mój algorytm dodaje punkt przecięcia wielokrotnie, ponieważ przy zdarzeniu rozpatrywania wierzchołka startowego nie sprawdzam czy dany wierzchołek już był znaleziony. Natomiast po dodatniu tego warunku algorytm i tak będzie nie poprawny, gdyż po rozpatrzeniu punktu przecięcia zamieniamy ze sobą pary odcinków, a przy trzech odcinkach powinniśmy rozpatrzeć wzajemne położenie trzech odcinków na raz.

7. Wnioski:

Dla testowanych zbiorów odcinków algorytm zadziałał poprawnie, a dzięki stosowanym strukturom złożoność wyniosła O((P+n) log n), gdzie P to liczba przecięć. Największą trudnością tego algorytmu było poprawne rozpatrywanie zdarzeń, szczególnie punktów przecięcia. Dla bardzo dużych losowych zbiorów danych algorytm często nie wykonuje się, jest to spowodowane tym, że dla takich przypadków zagęszczenie jest zbyt duże i jest częściej generują się odcinki nie spełniające warunków zadania (czyli pokrywające się lub więcej niż kilka linii przecinających się w jednym punkcie). Dla gęstych zbiorów w których wiele odcinków przecina się złożoność czasowa staje się niekorzystna $O(n^2 \log n)$, ponieważ liczba przecięć jest rzędu n^2, a więc korzystniejsze jest porównanie ze sobą wszystkich odcinków po kolei. Algorytm będzie działał sprawnie dla rzadkich zbiorów odcinków, o których wiemy że spełniają założenia zadania.