Algorytmy geometryczne
Labolatorium nr 4

Patryk Klatka
Grupa nr 4

8 grudnia 2022

1 Opis ćwiczenia

Celem ćwiczenia było zaimplementowanie algorytmu do wykrywania przecięć odcinków na płaszczyźnie w dwóch wariantach: czy istnieje chociaż jedno przecięcie oraz znajdujący wszystkie przecięcia. Dodatkowo, w celu ułatwienia generowania zbiorów danych, należało zaimplementować funkcję do generowania losowych zbiorów odcinków o zadanych parametrach takich jak liczba odcinków oraz zakresy losowanych punktów.

1.1 Algorytm wyznaczający wszystkie przecięcia odcinków

Algorytm bazuje na badaniu punktów zdarzeń za pomocą pewnej hiperpłaszczyzny (np. prosta w $\mathbb{R}^2$), która nazywana jest miotłą, w naszym przypadku jest to prosta o równaniu $x = c$, gdzie c to wartość współrzędnej x punktu zdarzenia. Zdarzenia to punkty, w których zatrzymuje się miotła. Jest ona przesuwana w wyznaczonym kierunku, w naszym przypadku w stronę dodatnich wartości osi OX. Wszystkie zdarzenia przechowujemy w strukturze zdarzeń, a informacje potrzebne do obliczeń w strukturze stanu. W naszej implementacji będziemy korzystać ze struktury Q, która zawiera uporządkowane względem x-ów końce odcinków oraz punkty przecięć odcinków, które kiedykolwiek były sąsiadami w strukturze (inaczej: punkty zdarzeń posortowane po współrzędnej x) oraz ze struktury stanu T, która zawiera zbiór odcinków, które aktualnie przecinają miotłę, uporządkowane względem współrzędnych osi OY. Obie te struktury możemy prezentować za pomocą drzew BST, w których szukanie jak i wstawianie/usuwanie nowych elementów jest przeprowadzane w czasie $O(\log n)$. W Pythonie takie drzewa (dokładniej: drzewa czerwono-czarne) są zaimplementowane w bibliotece `sortedcontainers` pod postacią struktury danych o klasie `SortedSet`, która dodatkowo nie pozwala na duplikaty elementów. Każdy odcinek ma sąsiadów: sąsiadem nazywamy odcinek, który jest najbliżej badanego odcinka w porządku pionowym.

1. Zainicjuj struktury zdarzeń Q oraz T
2. Dopóki istnieje zdarzenie w Q
   - 2.1. Pobierz zdarzenie p (punkt w $\mathbb{R}^2$)
   - 2.2. Jeżeli p jest początkiem odcinka
     - 2.2.1. Dodajemy badany odcinek do T
     - 2.2.2. Sprawdzamy w T, czy lewy lub prawy sąsiad odcinka, gdzie p jest początkiem, nie przecina odcinka badanego. Jeżeli się przecinają, to dodajemy punkt przecięcia do Q
   - 2.3. Jeżeli p jest końcem odcinka
     - 2.3.1. Sprawdzamy w T, czy lewy i prawy sąsiad odcinka, mają punkt przecięcia. Gdy taki punkt istnieje, dodajemy punkt do Q
     - 2.3.2. Usuwamy badany odcinek z T
   - 2.4. Jeżeli p jest punktem przecięcia odcinków s i s'
     - 2.4.1. Zamieniamy kolejność odcinków s i s' w T

2.4.2. Dla sąsiadów s sprawdzamy w T czy nie przecinają się w punkcie na prawo od miotły. Gdy punkt przecięcia istnieje, dodajemy punkt do Q

2.4.3. Wykonujemy powyższą operację dla odcinka s'

1.2 Algorytm zamiatania sprawdzający, czy choć jedna para odcinków w zadanym zbiorze się przecina

Algorytm ma niewielką modyfikację względem powyższej procedury. Gdy tylko wykryjemy punkt przecięcia, procedura się zakończy. W celu przyspieszenia działania algorytmu, zamiast obliczania miejsc przecięć miotły z odcinkami w strukturze stanu T, przechowujemy odcinki posortowane po współrzędnych z osi OY pierwszego punktu (punkt o mniejszej współrzędnej na osi OX).

1. Zainicjuj struktury zdarzeń Q oraz T
2. Dopóki istnieje zdarzenie w Q
   2.1. Pobierz zdarzenie p (punkt w $\mathbb{R}^2$)
   2.2. Jeżeli p jest początkiem odcinka
      2.2.1. Dodajemy odcinek do T
      2.2.2. Sprawdzamy, czy lewy lub prawy sąsiad odcinka, gdzie p jest początkiem, nie przecina odcinka badanego. Jeżeli się przecinają, to kończymy działanie programu.
   2.3. Jeżeli p jest końcem odcinka
      2.3.1. Sprawdzamy, czy lewy i prawy sąsiad odcinka, przecinają się. Gdy punkt przecięcia istnieje, to kończymy działanie procedury.
      2.3.2. Usuwamy odcinek z T

1.3 Implementacja struktury stanu oraz zdarzeń

W algorytmach podanych powyżej, powinniśmy uważać na dodawanie wielokrotnie tych samych zdarzeń, ponieważ może to zwiększyć czas działania algorytmu. Dlatego dobrym wyborem, jako struktury danych reprezentującą struktury zdarzeń i stanu, jest struktura `SortedSet`. W Pythonie jest ona częścią zewnętrznej biblioteki `sortedcontainers`, autorstwa Granta Jenksa. `SortedSet` jest drzewem czerwono-czarnym, zatem każda operacja wstawiania/usuwania/wyszukiwania będzie miała złożoność $O(\log n)$.

1.4 Wykrywanie przecięć wielokrotnie w różnych zdarzeniach

Może się zdarzyć, że punkt przecięcia się odcinków może zostać wykryty wiele razy (tak jak w zbiorze odcinków nr 5). Dzięki użyciu `SortedSet` struktura zdarzeń Q oraz stanu T zawiera tylko elementy niepowtarzające się, co pozwala na szybsze działanie algorytmu.

2 Użyte narzędzia i środowisko pracy

Cały program został napisany w języku Python za pomocą popularnej, interaktywnej platformy Jupyter Notebook. W narzędziu graficznym zostały wykorzystywane biblioteki takie jak `matplotlib` (do graficznego przedstawiania danych) oraz `numpy` (do generowania zestawów danych). W otrzymanym narzędziu graficznym został zmieniony backend biblioteki matplotlib (`%matplotlib widget`) poprzez doinstalowanie biblioteki `ipympl`, aby narzędzie to mogło obsługiwać nowsze edytory, m.in. `VSCode`. Dodatkowo, dodana została możliwość tworzenia tytułów wykresów/podwykresów, zmiana zakresów osi OX oraz OY i zostały poprawione występujące błędy. W celu korzystania z dodatkowych struktur danych niedostępnych domyślnie w Pythonie, wykorzystywana jest biblioteka `sortedcontainers`.

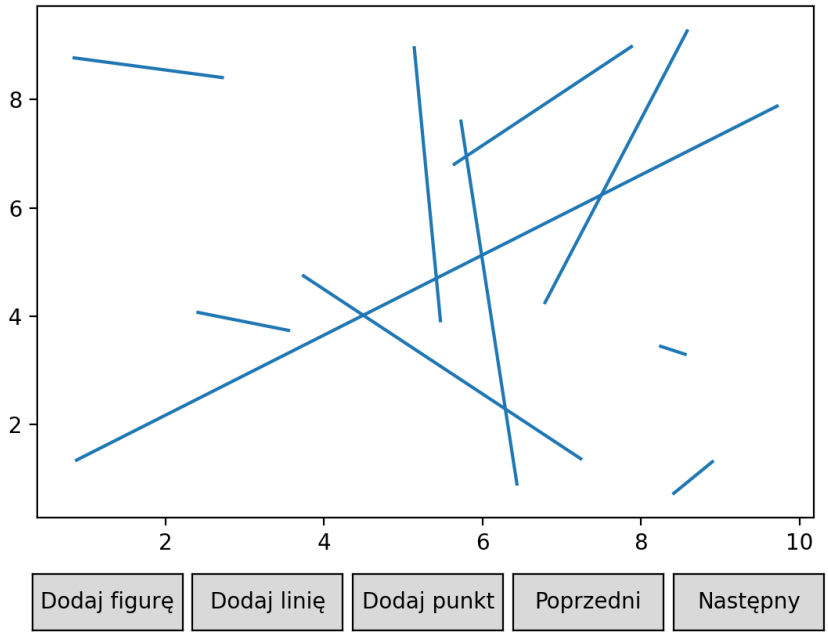
Dane sprzętowe:

- System Windows 10 x64

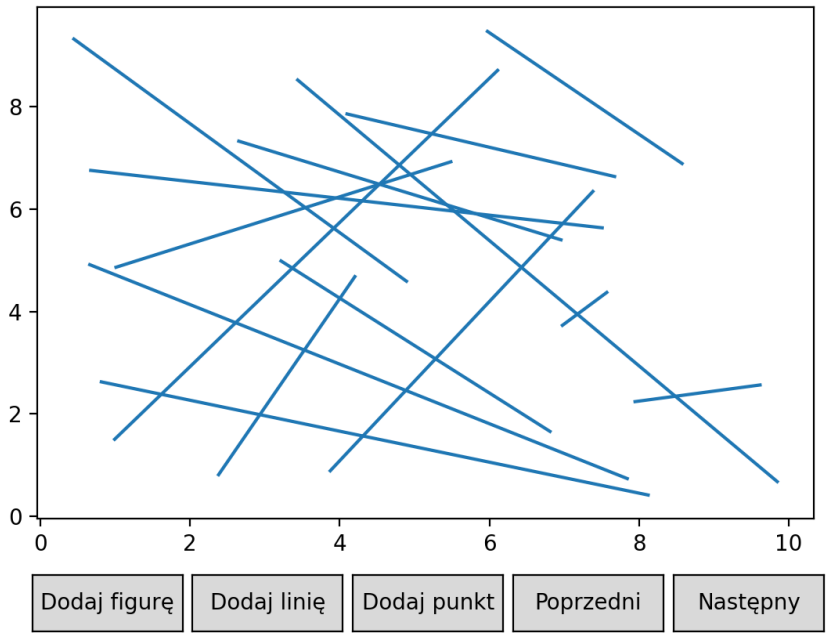
- Procesor Intel Core i5-8250U
- RAM 8GB
- Wersja języka Python: 3.10

3 Generowanie figur

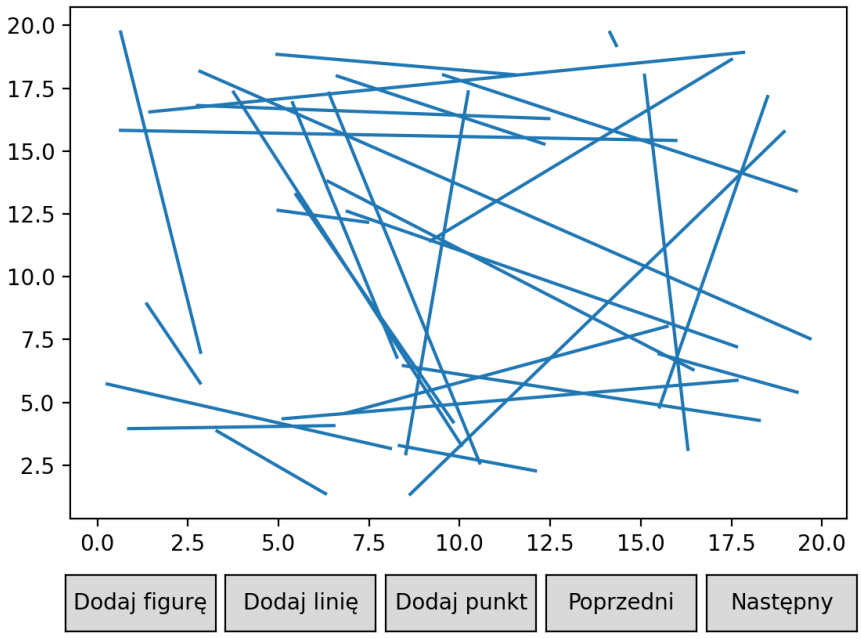
W celu przetestowania algorytmów zostało wygenerowane 7 zbiorów odcinków, z czego jeden nie ma przecięć.



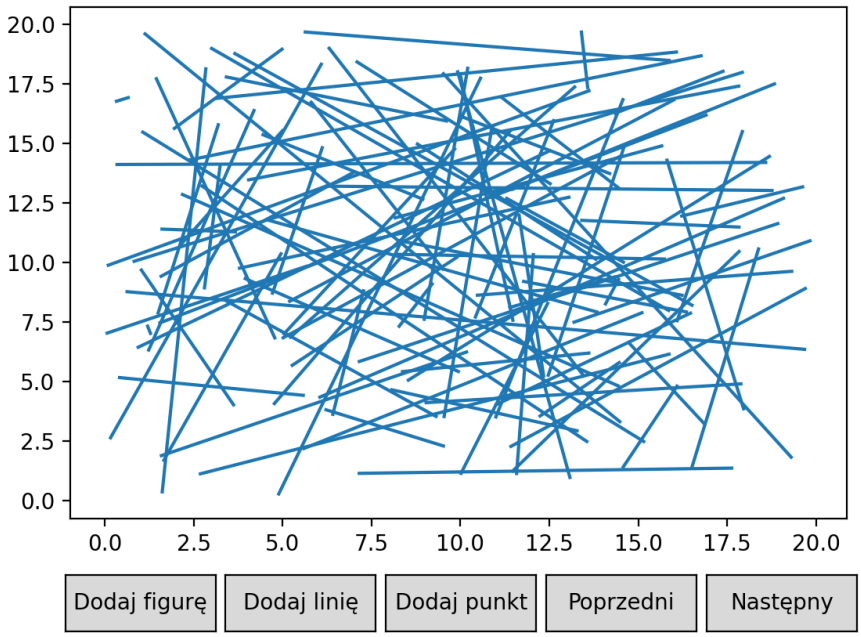
Rysunek 1: Zbiór nr 1



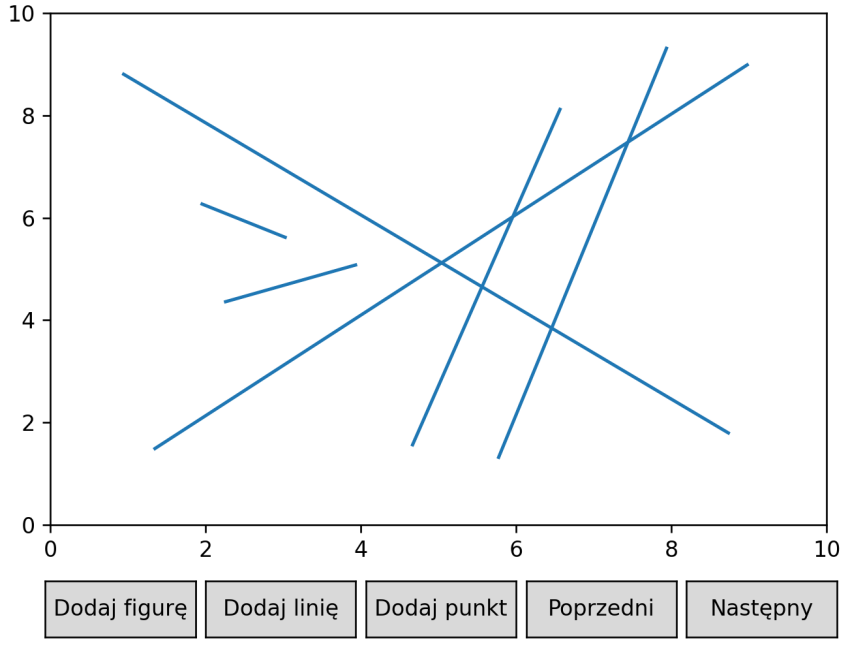
Rysunek 2: Zbiór nr 2



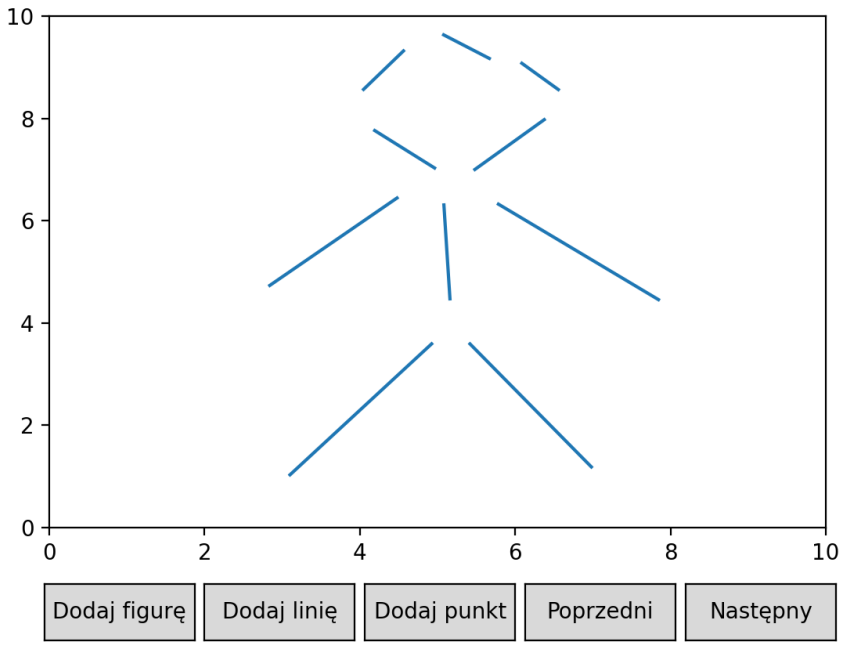
Rysunek 3: Zbiór nr 3



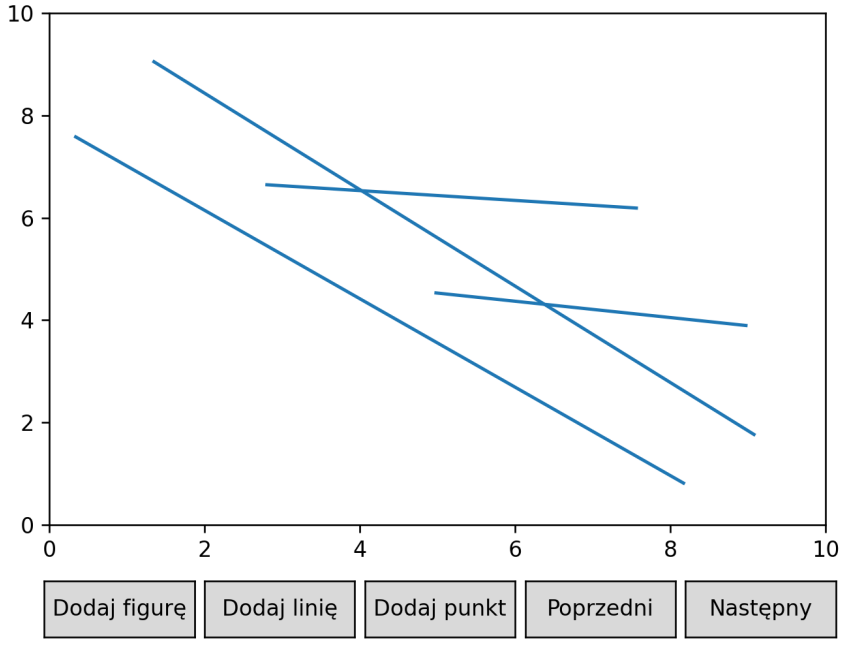
Rysunek 4: Zbiór nr 4



Rysunek 5: Zbiór nr 5



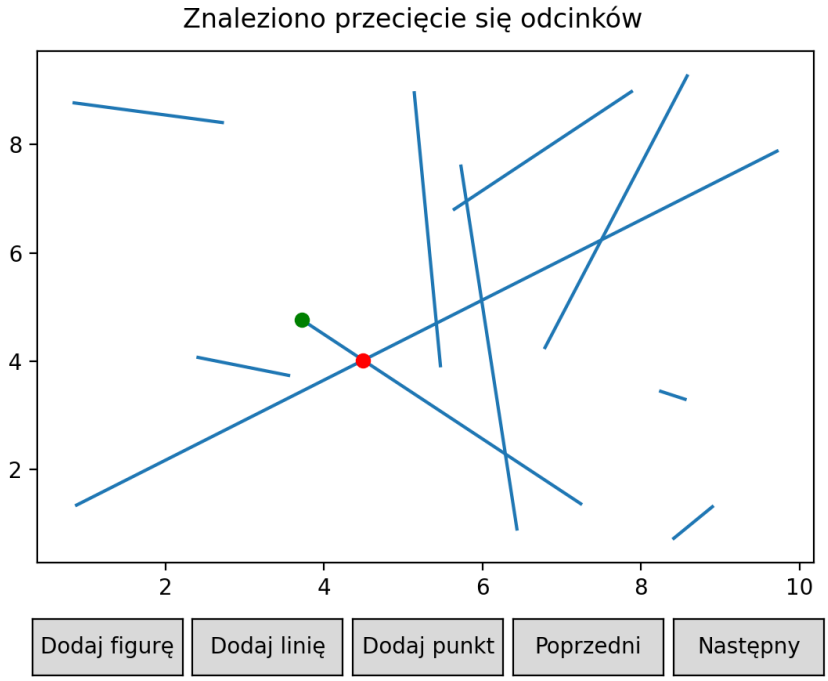
Rysunek 6: Zbiór nr 6



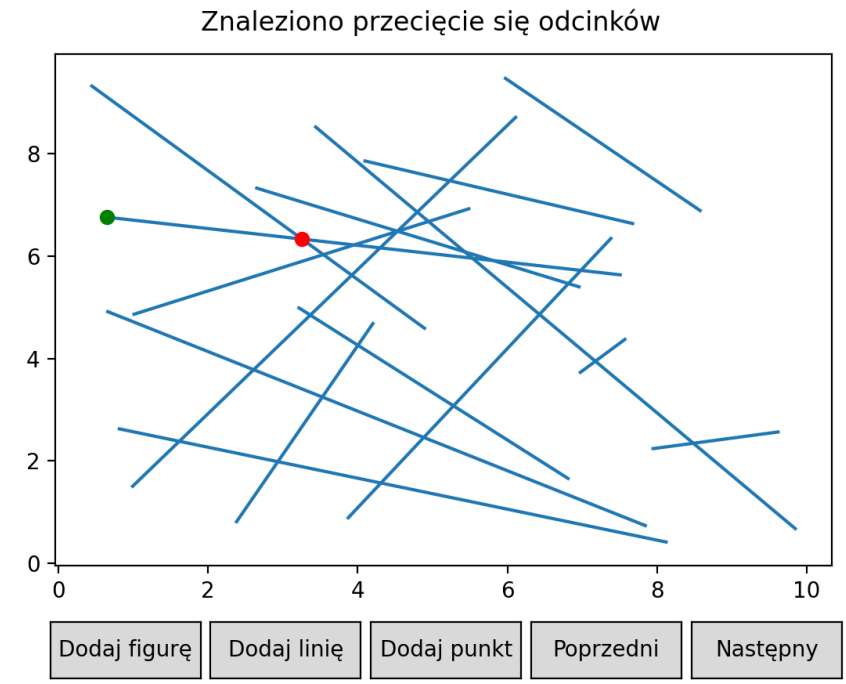
Rysunek 7: Zbiór nr 7

4 Sprawdzanie, czy w zbiorze istnieje punkt będący przecięciem dwóch dowolnych odcinków

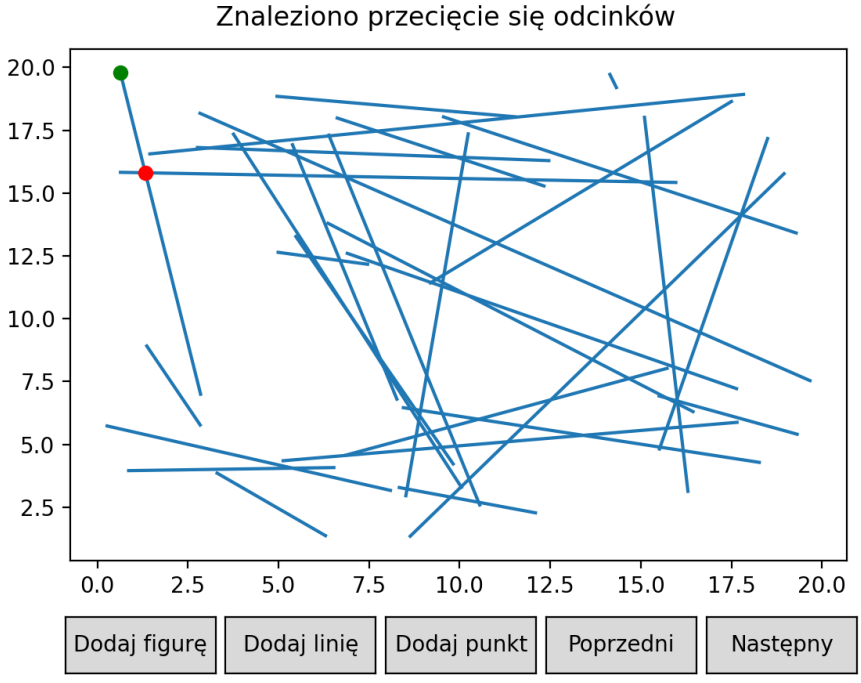
Wszystkie punkty przecięć, wnioskując z poniżej przedstawionych wyników, zostały wykryte prawidłowo. Dla zbioru numer 6 nie zostało znalezione przecięcie, co jest prawidłowym wynikiem. Na poniższych ilustracjach punkty zaznaczone na zielono to punkt, który był analizowany, gdy zostało znalezione przecięcie, zaznaczone kolorem czerwonym.



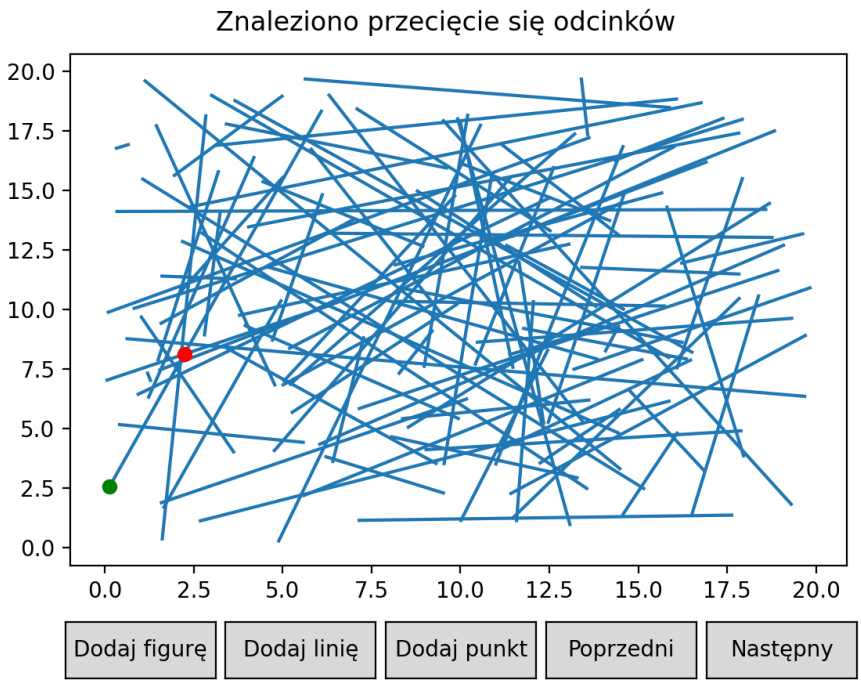
Rysunek 8: Testowanie zbioru nr 1



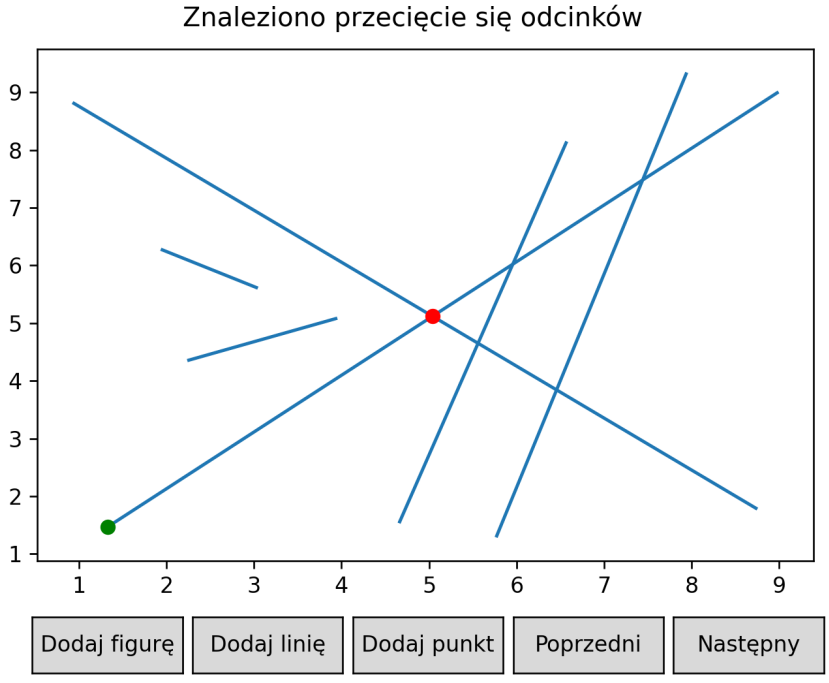
Rysunek 9: Testowanie zbioru nr 2



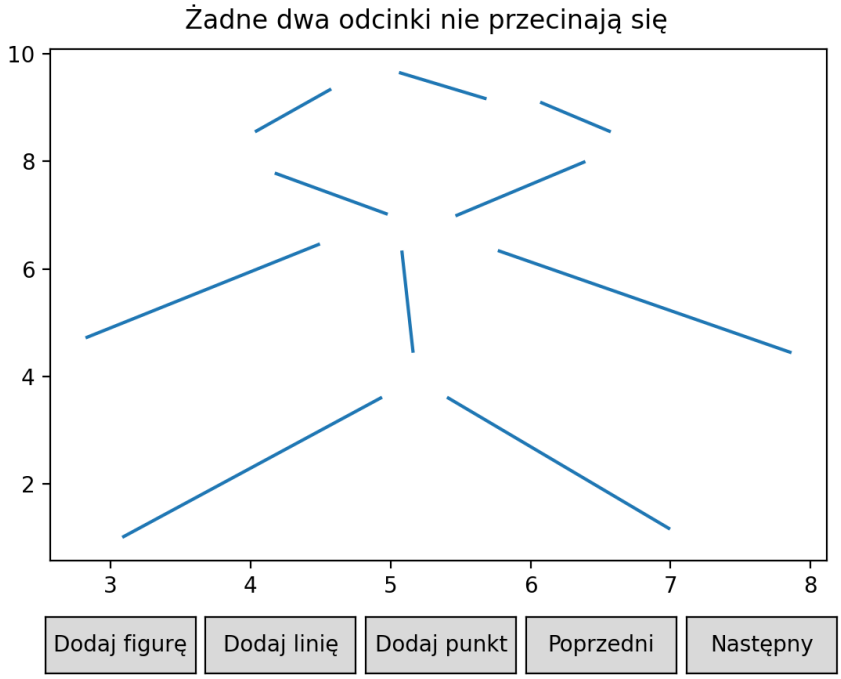
Rysunek 10: Testowanie zbioru nr 3



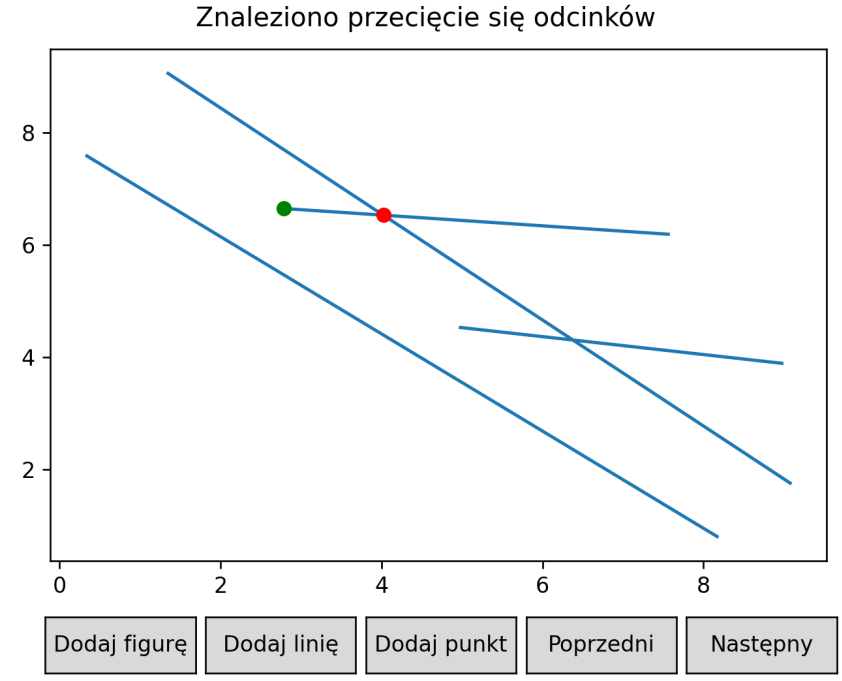
Rysunek 11: Testowanie zbioru nr 4



Rysunek 12: Testowanie zbioru nr 5

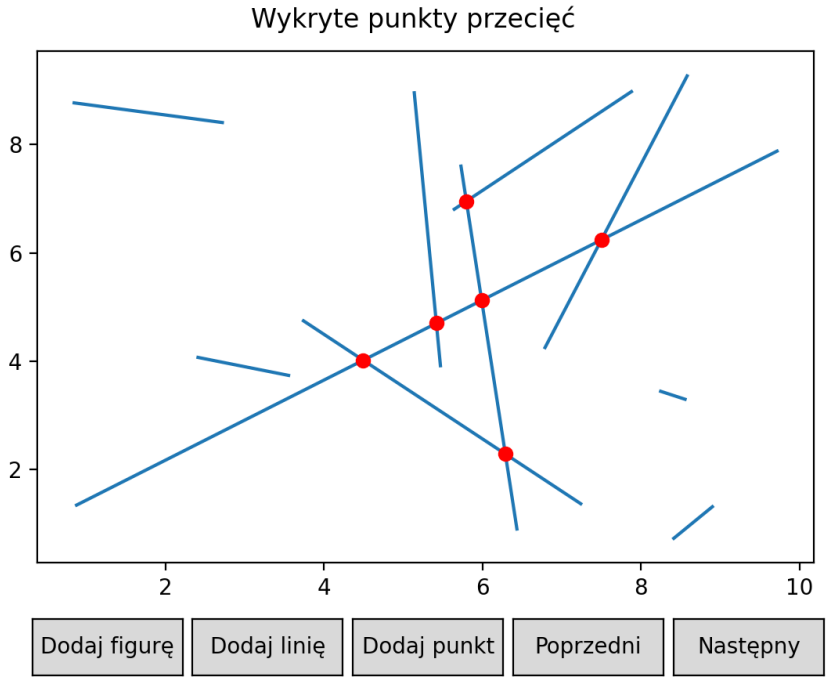


Rysunek 13: Testowanie zbioru nr 6

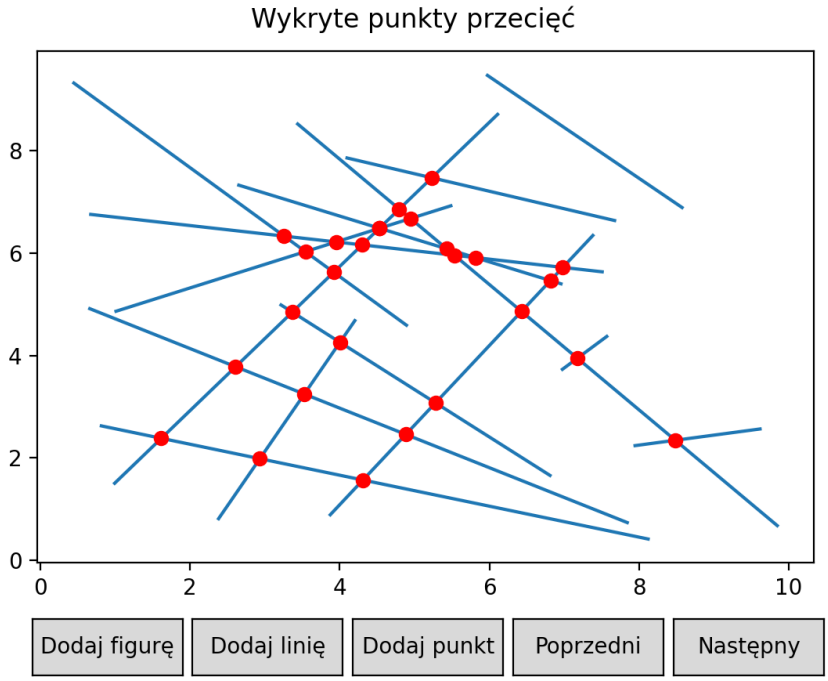


Rysunek 14: Testowanie zbioru nr 7

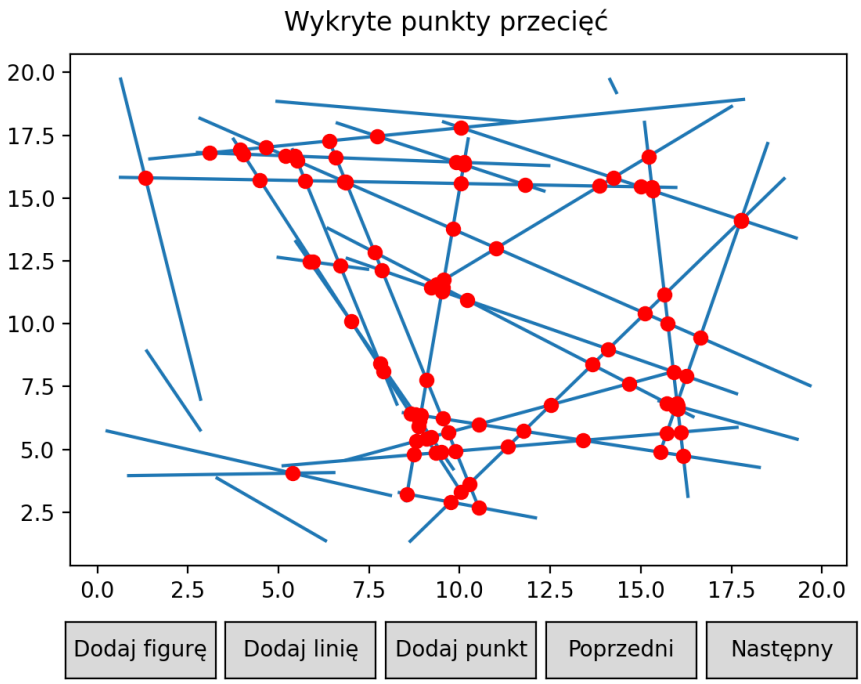
5 Wyszukiwanie wszystkich punktów przecięć



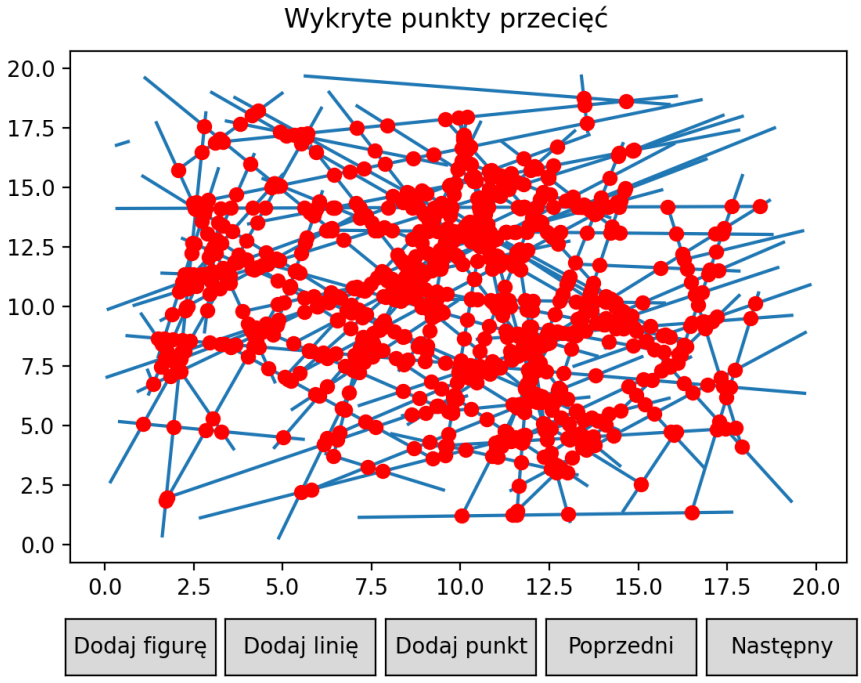
Rysunek 15: Punkty przecięć dla zbioru nr 1



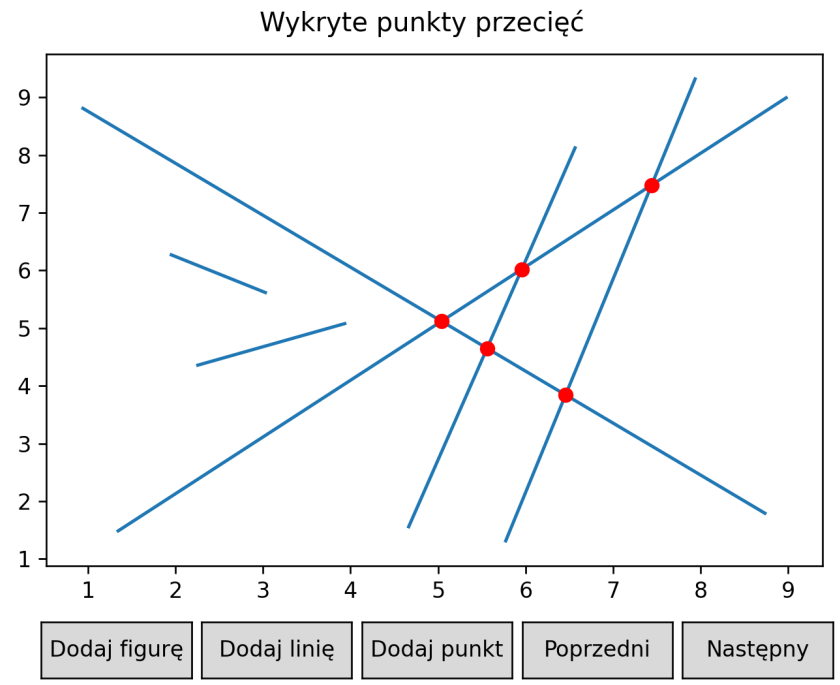
Rysunek 16: Punkty przecięć dla zbioru nr 2



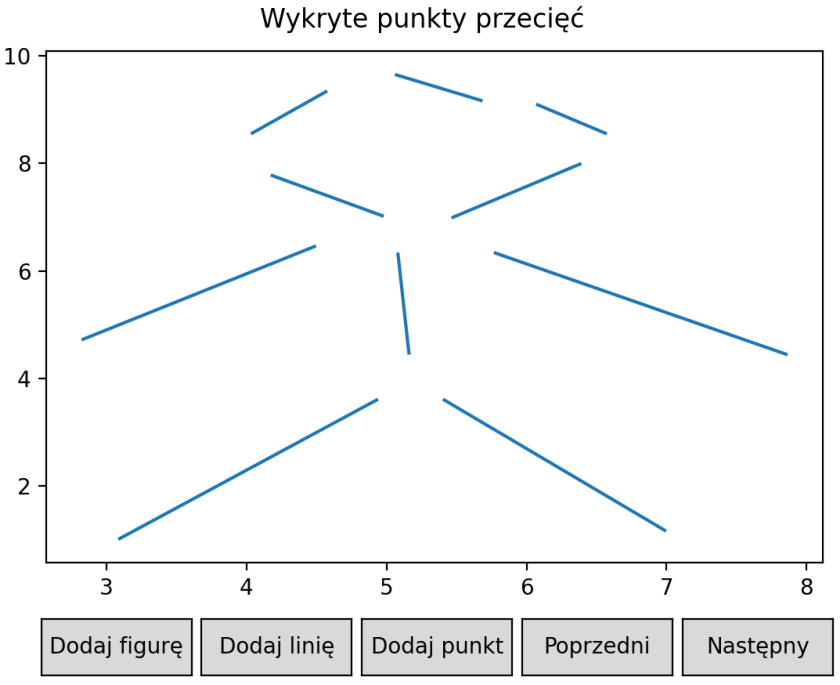
Rysunek 17: Punkty przecięć dla zbioru nr 3



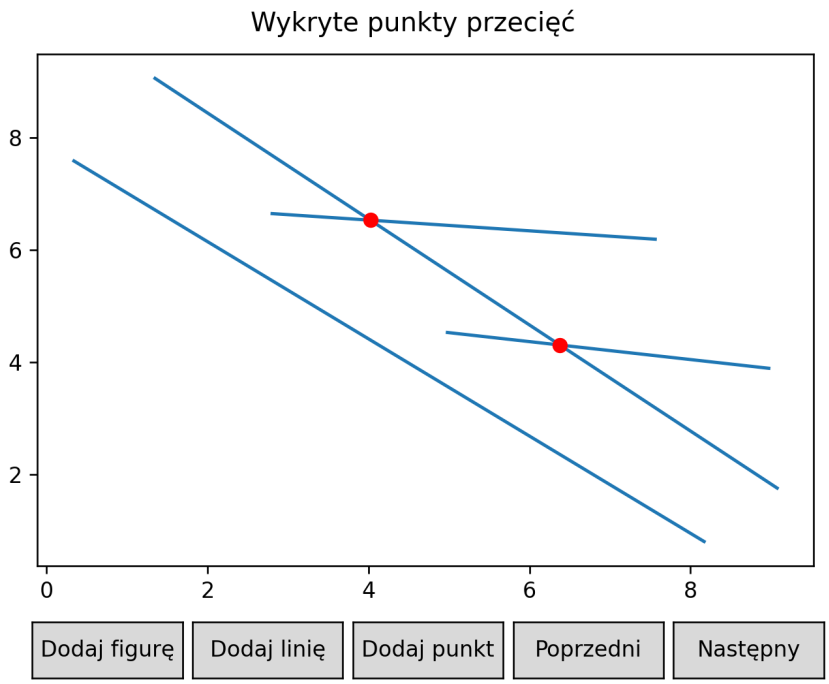
Rysunek 18: Punkty przecięć dla zbioru nr 4



Rysunek 19: Punkty przecięć dla zbioru nr 5



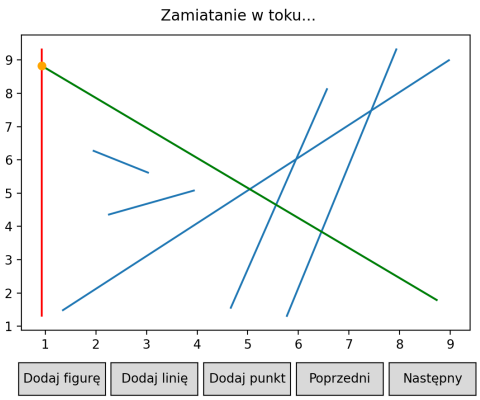
Rysunek 20: Punkty przecięć dla zbioru nr 6



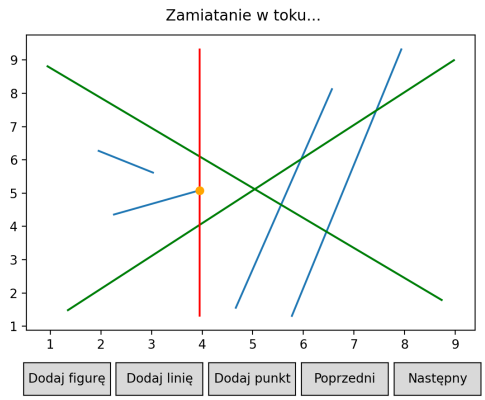
Rysunek 21: Punkty przecięć dla zbioru nr 7

5.1 Przykładowe działanie algorytmu dla zbioru nr 5

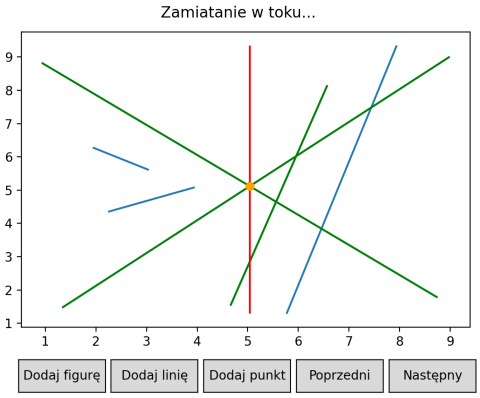
W celu ilustracji działania algorytmu każdy wykres zapisuje przebieg dodawanych punktów przecięć jak i aktualną pozycję miotły. Miotła jest reprezentowana przez czerwoną, pionową prostą. Punkt o kolorze pomarańczowym, to wierzchołek aktualnie badany przez miotłę, kolorem czerwonym zaznaczone są wykryte punkty przecięć. Odcinki w kolorze zielonym to segmenty aktualnie przecinane przez miotłę.



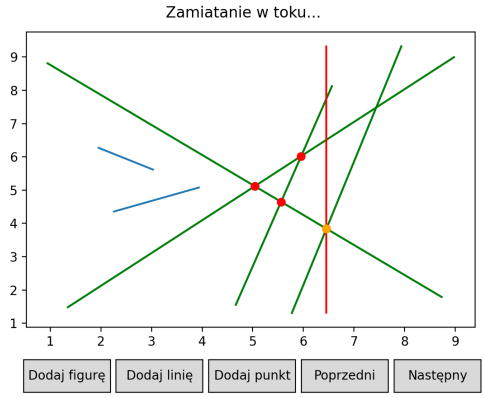
(a) Krok 1



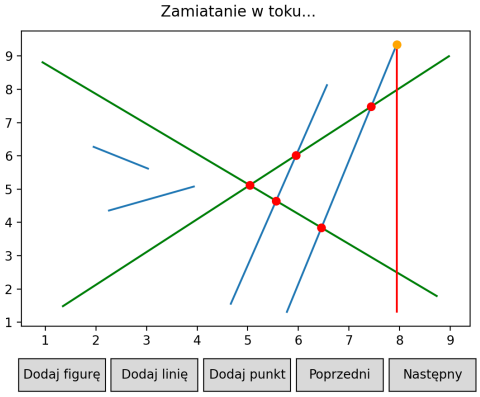
(b) Krok 2



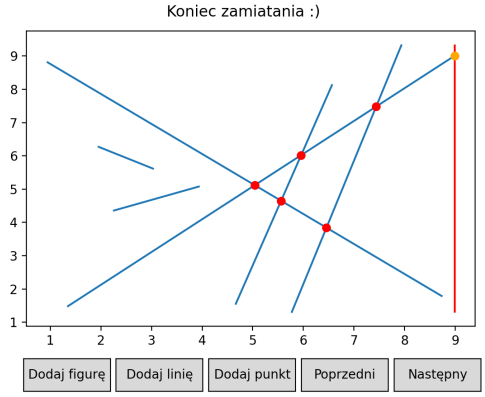
(c) Krok 3



(d) Krok 4



(e) Krok 5



(f) Krok 6

Rysunek 22: Poszczególne kroki algorytmu wykrywania przecięć dla zbioru nr 5

6 Wnioski

Analizując powyższe wyniki algorytmów można wywnioskować, że zostały one zaimplementowane prawidłowo. Użycie struktury `SortedSet` z biblioteki `sortedcontainers` pozwoliło na wygodne oraz szybkie otrzymywanie dostępu do zdarzeń.