Sprawozdanie z laboratorium 4

Temat: Przecinanie się odcinków

Data: 17.12.2023

Algorytmy geometryczne

Wojciech Kaźmierczak

Nr albumu: 416692

Gr. 4, czwartek 11:20, tydzień A

Informatyka, Wydział Informatyki

AGH UST

Procesor: AMD Ryzen 7 5700U 8 rdzeni

RAM: 16GB

System: Windows 11 Home

Użyty język: Python, Biblioteki: numpy, pandas, matplotlib, sortedcontainers, PriorityQueue

Środowisko: Jupyter Notebook

1. Opis ćwiczenia

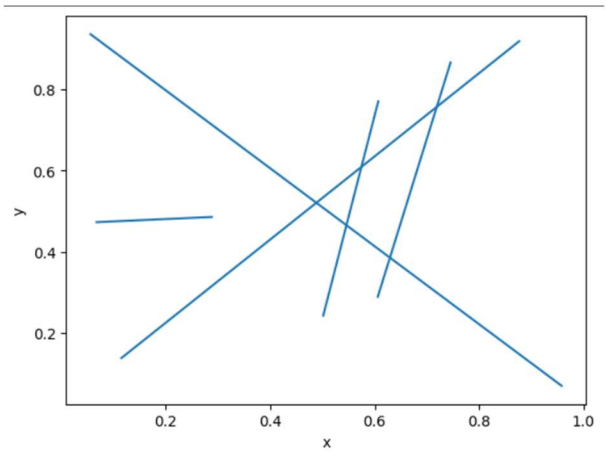
Ćwiczenie polegało na zaimplementowaniu algorytmu wykrywającego przecięcia odcinków leżących na płaszczyźnie.

2. Tworzenie zbiorów odcinków

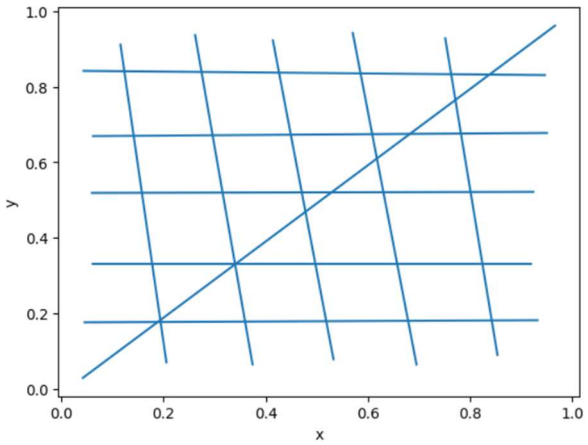
W napisanym programie zbiory odcinków można tworzyć na dwa sposoby:

- Generując zadaną liczbę odcinków, których punkty końcowe mają współrzędne z zadanego zakresu, funkcja generate_uniform_sections(max_x, max_y, n) (punkty końcowe odcinka z zakresu (0, max_x) dla współrzędnej x oraz (0, max_y) dla współrzędnej y.
- Wprowadzając odcinki za pomocą myszki funkcja add_sections() wykorzystuje ona funkcję ginput() z biblioteki matplotlib zwraca ona tablicę odcinków, pomijając odcinki pionowe. W celu zamknięcia okienka do wprowadzania odcinków należy nacisnąć Enter.

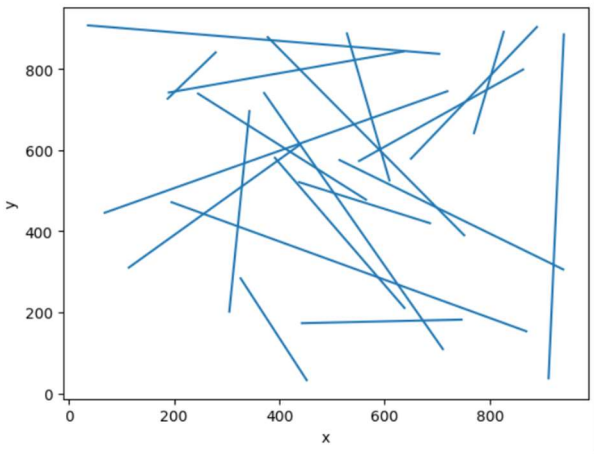
Użyte zbiory:



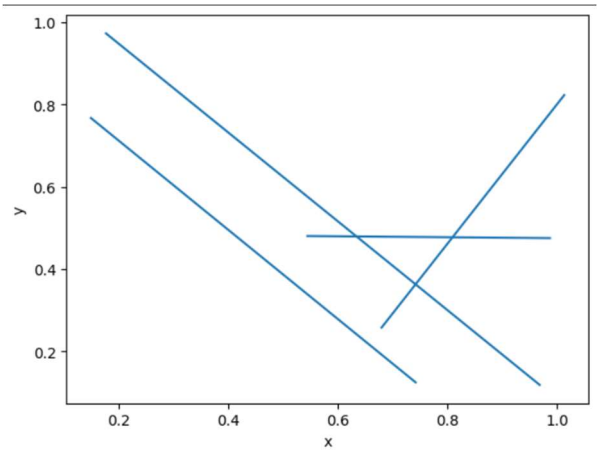
Wykres 1 (5 odcinków)



Wykres 2 (Siatka)



Wykres 3 (20 losowych odcinków)



Wykres 4 (4 odcinki)

3. Użyte struktury danych

Klasa Point reprezentująca punkt posiada współrzędne (x,y), informację do jakich odcinków należy dany punkt (seg1, seg2), oraz czy jest to punkt lewy, prawy czy też będący punktem przecięcia.

Klasa Segment reprezentująca odcinek przechowuje punkty będące końcami danego odcinka, współczynnik a oraz b wzoru funkcji liniowej do której należy dany odcinek, współrzędną x będącą miejscem, gdzie obecnie obliczane są wartości y dla odcinków w celu ich uporządkowania.

Jako strukturę przechowującą przyszłe zdarzenia wybrałem kolejkę PriorityQueue() która jako klucz posiada wartość współrzędnej x punktu.

Jako strukturę stanu wybrałem SortedSet(), który jako klucz posiada wartość y dla danego x (gdzie znajduje się miotła). Jest to struktura oparta na zrównoważonym drzewie binarnym. Wstawianie i usuwanie elementów ma złożoność logarytmiczną.

4. Zapobieganie duplikatów oraz sposób wykrywania przecięć.

Przecinanie się odcinków wykrywane jest przy pomocy postaci parametrycznej funkcji liniowej oraz wyznacznika macierzy. Sprawdzamy czy przecięcie się funkcji leży wewnątrz odcinków na nich leżących jeśli tak wyliczamy współrzędne punktu przecięcia.

Zapobiegamy duplikatom punktów przecięcia korzystając z wiedzy, że dwa odcinki mogą się przecinać w co najwyżej jednym punkcie. Zapamiętując indeksy owych odcinków dla kolejnych wyników sprawdzamy czy para indeksów odcinków przecinających nie zawierają się już w naszych uwzględnionych parach indeksów odcinków. W ten sposób unikamy ewentualnych błędów związanych dokładnością obliczeń współrzędnych punktów.

5. Opis działania algorytmu – obsługa zdarzeń

Uwzględniamy 3 rodzaje zdarzeń – początek odcinka, koniec odcinka oraz punkt przecięcia.

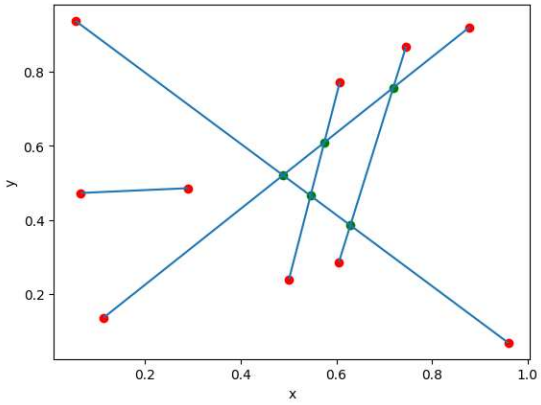
Gdy wykryjemy początek odcinka aktualizujemy położenie miotły przesuwając ją na miejsce współrzędnej x wykrytego początku punktu. Po czym dodajemy do struktury stanu odcinek związany z danym punktem. Jeśli dany odcinek ma sąsiadów to sprawdzamy czy się z którymś nie przecina jeśli tak to dodajmy w punkt przecięcia i umieszczamy go na strukturze zdarzeń.

W przypadku wykrycia końca odcinka sprawdzamy czy dany odcinek nie przecina się ze swoimi sąsiadami (o ile istnieją) i zdejmujemy go ze struktury stanu.

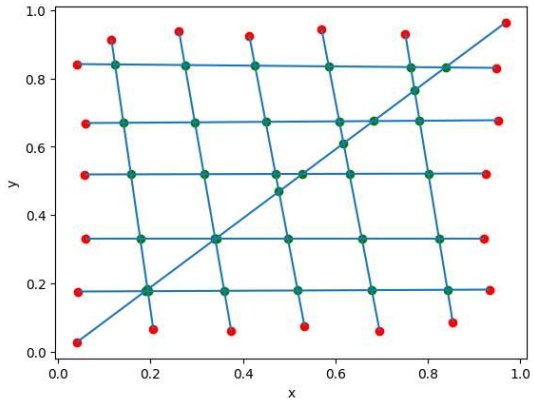
W przypadku wykrycia przecięcia odcinków, ze struktury stanu usuwamy odcinki, na których leżał przesuwamy miotłę o 10**(-9) w prawo dodajemy ponownie usunięte odcinki w ten sposób aktualizując poprawne sortowanie wewnątrz struktury stanu SortedSet(). Po czym sprawdzamy czy owe odcinki nie przecinają się ze swoimi sąsiadami (o ile istnieją), a jeśli tak to miejsca przecięć dodajemy do struktury zdarzeń.

6. Wyniki działania algorytmu dla wybranych zbiorów odcinków.

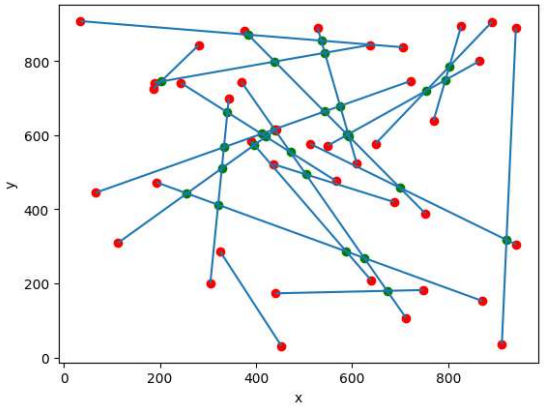
Na zielono zaznaczone punkty przecięcia odcinków, a na czerwono punkt będące początkami lub końcami odcinków.



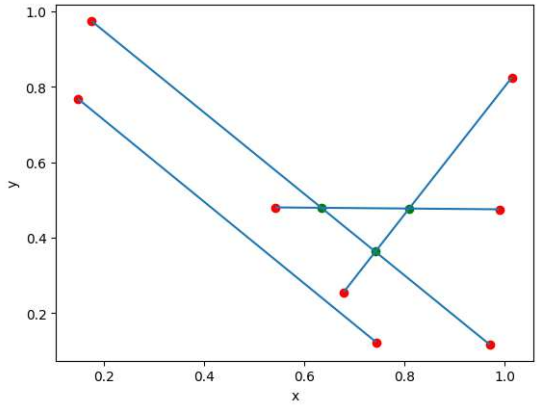
Wykres 5 (Wynik algorytmu dla zbioru odcinków z Wykresu 1)



Wykres 6 (Wynik algorytmu dla zbioru odcinków z Wykresu2)



Wykres 7 (Wynik algorytmu dla zbioru odcinków z Wykresu 3)



Wykres 8 (Wynik algorytmu dla zbioru odcinków z Wykresu 4)

Wizualizacje krokowe w postaci GIF-ów dostępne w pliku z kodem, możliwe jest ich pobranie za pomocą polecania vis.save_gif()

7. Wnioski

Wszystkie przypadki skrajne, które sprawdziłem algorytm rozwiązał poprawnie, takie jak pomijanie duplikatów, czy też ciągła aktualizacja struktury stanu ze względu na położenie miotły. Algorytm nie przeszedł dwóch dostarczonych testów, lecz wyniki które zwraca różnią się dopiero na dalekim miejscu po przecinku, jednak indeksy odcinków zwracanych przez algorytm i tych oczekiwanych są takie same, więc w przypadkach, gdzie nie przechodzi testów i tak działa poprawnie, a błąd związany jest ze sposobem wyznaczania współrzędnych punktu przecięcia. Jest to też dowód na to, że sprawdzanie duplikatów punktów przecięcia po współrzędnych byłoby nierozsądne i mogłoby generować błędy.