Algorytmy geometryczne

Sprawozdanie 4 | Przecinanie Odcinków

Paweł Fornagiel | Informatyka rok II | Grupa 1

Data Wykonania: 25.11.2024 | Data Oddania: 05.02.2025

1. Opis ćwiczenia i realizacja

1.1. Informacje wstępne

Celem ćwiczenia jest realizacja zagadnień związanych z wykrywaniem punktów przecięć odcinków na płaszczyźnie, w szczególności opierających się na implementacji algorytmu zamiatania (ang. *sweep line*) wraz z towarzyszącymi strukturami pomocniczymi: strukturą zdarzeń oraz strukturą stanu

1.2. Opis algorytmu

Algorytm zamiatania wykorzystuje miotłę, będącą hiperpłaszczyzną, co w omawianym przypadku sprowadza się do prostej w $\mathbb{R}^2$, która przesuwa się w kierunku osi $O(X)$ zwanym kierunkiem zamiatania. Miotła zatrzymywać będzie się w pozycjach zwanych *zdarzeniami*, w w celu przetworzenia danych geometrycznych. Informacje o zdarzeniach przechowywane są w strukturze zdarzeń, która porządkuje je według współrzędnej x ich wystąpienia. Dane potrzebne do obliczeń, takie jak aktualne odcinki przecinające miotłę, przechowywane są w strukturze stanu, która jest dynamicznie aktualizowana przy każdym zdarzeniu. Po przetworzeniu obszaru na lewo od miotły, rozwiązanie problemu geometrycznego dla tego obszaru jest już znane.

W każdym położeniu miotły można wyróżnić trzy kategorie odcinków:

- **Odcinki przetworzone**, odcinki, których oba końce znajdują się na lewo od miotły i które nie są już aktywne.
- **Odcinki aktywne**. odcinki przecinające miotłę w jej bieżącym położeniu.
- **Odcinki oczekujące**, odcinki, których oba końce znajdują się na prawo od miotły i nie zostały jeszcze przetworzone.

Algorytm zamiatania rozpoczyna się od inicjalizacji pustej struktury stanu T oraz struktury zdarzeń Q. Następnie algorytm iteracyjnie przetwarza kolejne zdarzenia z kolejki Q, aż do rozpatrzenia wszystkich zdarzeń na kolejce. W każdym kroku pobierane jest zdarzenie p, po czym następuje aktualizacja struktury stanu T, opisana w Sekcja 1.4 .Po każdej aktualizacji struktury stanu następuje również aktualizacja kolejki Q odpowiednio usuwając kolejne zdarzenia początku i końca odcinka oraz dodając zdarzenia po wykryciu przecięć odcinków.

1.3. Struktura zdarzeń Q

Struktura zdarzeń zawiera n uporządkowanych rosnąco względem współrzędnej x zdarzeń, będących jednym z trzech przypadków punktów na płaszczyźnie:

- punkt początku odcinka
- punkt końca odcinka
- punkt przecięcia dwóch odcinków

Aby uzyskać efektywne operacje odczytu odcinka o najmniejszej współrzędnej x oraz dodawania odcinka do uporządkowanego zbioru, do realizacji struktury użyto kolejki priorytetowej w oparciu o kopiec typu minimum, dostarczaną przez bibliotekę `heapq` języka `Python`. Umożliwia to wykonywanie wyżej wspomnianych operacji w złożoności $O(\log n)$.

Dodatkowo, w strukturze użyty został dodatkowy zbiór (`set`), umożliwiający identyfikację zdarzeń, które zostały wcześniej włożone na kolejkę i zabezpieczenie przed ich redundantnym rozpatrywaniem. W pliku realizującym laboratorium, implementacja struktury opisana jest w klasie o nazwie `EventQueue`.

1.4. Struktura stanu T

Struktura stanu miotły jest dynamicznie zarządzanym, uporządkowanym względem współrzędnych y odcinków na aktualnej linii miotły, zbiorem odcinków aktywnych, tj. tych, które aktualnie przecinają miotłę w jej bieżącym położeniu. Struktura umożliwia szybkie wyszukiwanie, wstawianie, usuwanie oraz sprawdzanie relacji sąsiedztwa między odcinkami.

W momencie zatrzymania miotły w punkcie zdarzenia, struktura stanu jest aktualizowana w następujący sposób:

- Jeśli zdarzenie to początek odcinka, odcinek jest wstawiany do struktury w odpowiednim miejscu porządku, po czym jest sprawdzany pod kątem przecięć z obydwoma jego sąsiadami.
- Jeśli zdarzenie to koniec odcinka, odcinek jest usuwany ze struktury, a sąsiedzi tego odcinka są sprawdzani pod kątem nowych przecięć.
- Jeśli zdarzenie to przecięcie odcinków, ich kolejność w strukturze jest zmieniana, po czym nowi sąsiedzi odcinków (obydwa zamienione odcinki będą posiadały dokładnie jednego nowego sąsiada) są sprawdzani pod kątem przecięć.

Jeżeli w którymkolwiek z wyżej wymienionych przypadków zostało wykryte przecięcie dwóch odcinków, do kolejki zdarzeń Q zostaje dodane zdarzenie o współrzędnej x odpowiadającej rzędnej punktu przecięcia. Sprawdzenie, czy dwa odcinki przecinają się ze sobą, realizowane jest na podstawie wyznaczenia orientacji wzajemnego położenia punktów $(p1, p2)$, $(q1, q2)$ reprezentujących odcinki p oraz q, która uzyskiwana jest przy użyciu znaku następującego wyznacznika:

$$\det(p_1, p_2, p_3) = \begin{vmatrix} p_{1_x} - p_{3_x} & p_{1_y} - a_y \\ p_{2_x} - a_x & p_{2_y} - p_{3_y} \end{vmatrix} = \left(p_{1_x} - p_{3_x}\right) \cdot \left(p_{2_y} - p_{3_y}\right) - \left(p_{1_y} - p_{3_y}\right) \cdot \left(p_{2_x} - p_{3_x}\right)$$

Jeżeli $p1$ i $p2$ lezą po przeciwnych stronach prostej wyznaczonej przez odcinek $(p1, p2)$ oraz $p1$, $p2$ leżą po przeciwnych stronach prostej wyznaczonej przez odcinek $(q1, q2)$, to odcinki uznaje się za przecinające się w punkcie obliczanym na podstawie równania kierunkowego prostych.

Realizacja wyżej opisanych operacji w implementacji struktury opiera się na klasie `SortedSet`, będącej uporządkowanym zbiorem dostarczanym przez bibliotekę `SortedContainers` języka `Python`. Pozwala ona na dokonywanie dodawania, usuwania oraz utrzymywania porządku przechowywanych odcinków w złożoności $O(\log n)$, gdzie n jest liczbą odcinków w strukturze. Dodatkowo, podobnie jak w przypadku zastosowania innych uporządkowanych struktur drzewiastych, dzięki jej użyciu istnieje możliwość aktualizacji relacji porządku pomiędzy elementami w czasie $O(1)$. Relacja porządku wyznaczana jest na podstawie porównanie współrzędnych y odcinków obliczanych dynamicznie z użyciem równania $y = ax + b$, gdzie a, b opisują kolejno współczynnik kierunkowy oraz wyraz wolny prostej, a x jest współrzędną, na której nastąpiło zatrzymanie miotły. Dzięki temu, że zmienia porządku podlegają zawsze dokładnie dwa odcinki, aktualizacja relacji nie przynosi niespodziewanych efektów.

2. Dane techniczne

Zadanie zostało przeprowadzone z użyciem narzędzi o następujących parametrach:

- Komputer HP EliteBook 840 G6:
  - System operacyjny: Windows 11 x64
  - Procesor Intel(R) Core(TM) i5-8365U CPU 1.60GHz 1.90 GHz
  - Pamięć RAM: 8GB
- Środowisko: Jupyter Notebook
- Język: Python 3.9.20
- Biblioteki języka: Numpy, Matplotlib, SortedContainers

2.1. Analiza wyników

W ramach laboratorium algorytm został użyty w celu sprawdzenia istnienia dowolnego przecięcia w zadanym zbiorze odcinków S oraz znalezienia wszystkich przecięć odcinków S, za które to zadania odpowiedzialne są odpowiednio procedury `is_intersection` oraz `find_intersections`.

Pierwsza z wyżej wspomnianych procedur nie wymaga dodatkowego zabezpieczenia przed wielokrotnym włożeniem na kolejkę zdarzeń Q tego samego zdarzenia, gdyż po znalezieniu pierwszego z nich kończy się jej wykonywanie. Mimo tego, w obydwu procedurach użyte zostały identyczne struktury danych z uwzględnieniem opisanego zabezpieczenia, ze względu na wysokie podobieństwo w realizacji rozwiązania obydwu problemów.

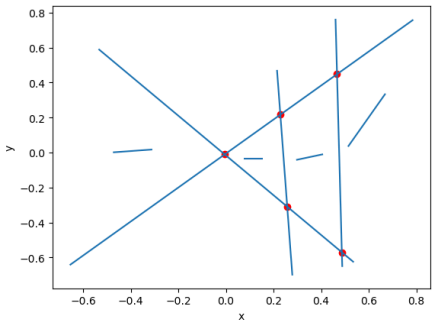
Zbiory odcinków testowych w ramach laboratorium był generowane losowo na określonym przedziale jednostajnym z wykorzystaniem funkcji `np.random.uniform` biblioteki `numpy` języka `Python` oraz tworzone poprzez ręczne zadanie odcinków za pomocą narzędzia graficznego.

2.2. Wizualizacja przykładowych zbiorów

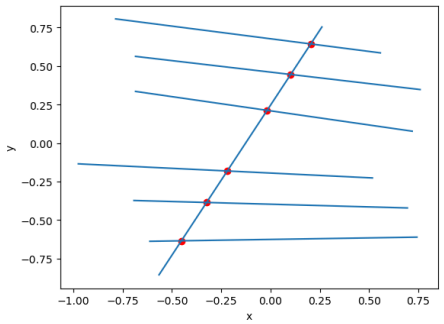
Poniższe wykresy przedstawiają przykładowe zbiory odcinków z zaznaczonymi punktami przecięć po wykonaniu algorytmu zamiatania.

Wykres 1 obrazuje poprawne rozpatrywanie zdarzenia początku i końca odcinka w sytuacji, gdy mają one miejsce bezpośrednio przed oraz po zdarzeniach przecięć. Wykres 2 i Wykres 5 przedstawiają rozpatrywanie sytuacji, w których występują przecięcia kilku prostych oraz wiele przecięć na tej samej prostej. Wykres 3 przedstawia sytuację, gdzie przecięcia prostych nie występują. Wykres 4 przedstawia przypadek zbioru z gęsto rozmieszczonymi przecięciami. Wizualizacja każdego z wyżej wymienionych przypadków potwierdza poprawne działanie algorytmu na różnorodnych zbiorach danych.

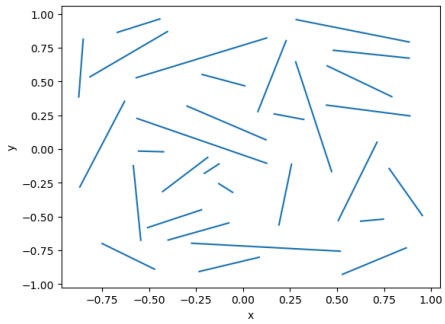
Dodatkowo, procedura znajdywania przecięć została dodatkowo przedstawiona „krok po kroku” w plikach o rozszerzeniu `gif`, dołączonych do realizacji laboratorium w katalogu o nazwie`fornagiel_gif`.



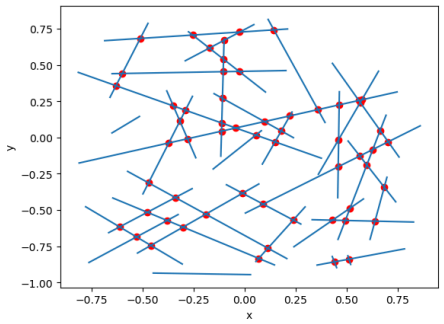
Wykres 1 : Wizualizacja *Zbioru 1*



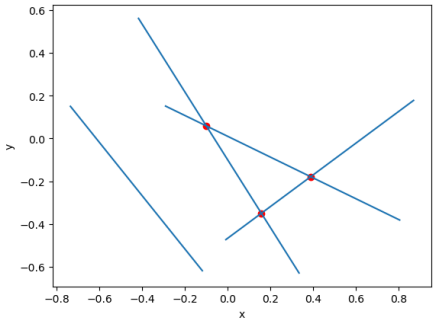
Wykres 2 : Wizualizacja *Zbioru 2*



Wykres 3 : Wizualizacja *Zbioru 3*



Wykres 4 : Wizualizacja *Zbioru 4*



Wykres 5 : Wizualizacja *Zbioru 5*

3. Wnioski

- **Efektywność algorytmu**
  Algorytm zamiatania okazał się skuteczny w wykrywaniu punktów przecięć odcinków na płaszczyźnie. Zastosowanie odpowiednich struktur danych, takich jak kolejka priorytetowa i uporządkowany zbiór, w ramach realizacji struktury zdarzeń oraz stanu miotły umożliwiło uzyskanie czasu działania algorytmu rzędu $O((P + n) \log n)$, gdzie n odpowiada liczbie odcinków, a P liczbie przecięć. Implementacja algorytmu potwierdziła teoretyczne przewidywania dotyczące jego złożoności czasowej.
- **Walidacja poprawności działania**
  Poprawność implementacji została potwierdzona na podstawie testowych zbiorów odcinków, zarówno generowanych losowo, jak i definiowanych ręcznie. Wizualizacje wyników wskazały na prawidłowe wykrywanie punktów przecięć oraz poprawne zarządzanie zdarzeniami początku i końca odcinka dla każdego z rozpatrywanych przypadków. Algorytm poprawnie obsługiwał przypadki specjalne, takie jak wiele przecięć w jednym punkcie, odcinki równoległe oraz brak przecięć. Wykresy obrazujące te sytuacje stanowią o uniwersalności zaimplementowanego rozwiązania.