Sprawozdanie z ćwiczenia czwartego

Algorytmy geometryczne

Jakub Pisarek
Informatyka, WIEiT, AGH

6 grudnia 2022

1 Cel ćwiczenia

Celem ćwiczenia jest implementacja algorytmu wykrywania dowolnego przecięcia, a następnie wszystkich przecięć spośród odcinków na płaszczyźnie przy pomocy zamiatania (ang. sweeping). Algorytm ten należy następnie przetestować na różnych zestawach danych.

2 Przebieg ćwiczenia

1. Przygotować procedurę, pozwalającą wprowadzać w sposób interaktywny kolejne odcinki, a także generować losowo zadaną ich liczbę z podanego zakresu współrzędnych 2D, eliminując przy tym odcinki pionowe i odcinki, które miałyby końce o tej samej współrzędnej x co inne. Program powinien umożliwiać zapis i odczyt zbioru odcinków,
2. Zaimplementować algorytm zamiatania sprawdzający, czy choć jedna para odcinków w zadanym zbiorze się przecina – opisać, jak została zaimplementowana struktura stanu (stan miotły) oraz struktura zdarzeń,
3. Uzupełnić procedurę wykrywającą przecięcie o wizualizację kolejnych kroków (pozycja i stan miotły), przetestować program na różnych zestawach danych,
4. Zaimplementować algorytm wyznaczający wszystkie przecięcia odcinków. Na wyjściu program powinien podawać liczbę wykrytych przecięć, współrzędne przecięć oraz dla każdego przecięcia odcinki, które się przecinają. Zmodyfikować (jeśli to konieczne) procedurę wizualizacji dla tego zagadnienia,
5. Opisać, czy konieczne były zmiany w strukturze zdarzeń – jeśli tak, to jakie? Czy w przypadku obu algorytmów konieczne są takie same struktury zdarzeń? Uzasadnić odpowiedź,
6. Krótko opisać, jak obsługiwane są zdarzenia początku odcinka, końca odcinka i przecięcia odcinków z uwzględnianiem wybranych struktur danych,
7. Przetestować zmodyfikowany program na różnych zestawach danych,
8. Wprowadzić taki układ odcinków, przy którym pewne przecięcia będą wykrywane więcej niż jeden raz, opisać zachowanie programu w takiej sytuacji.

3 Środowisko programistyczne

- System operacyjny Ubuntu 20.04.5 LTS x86_64,
- Procesor Intel Core i5-3320M 3.30 GHz,
- Python 3.8.10 z zewnętrznymi bibliotekami `matplotlib`, `numpy` oraz `bintrees`,
- Środowisko Jupyter Notebook zintegrowane z edytorem Visual Studio Code.

4 Szczegóły implementacji algorytmów

4.1 Sprawdzanie, czy choć jedna para odcinków się przecina

Do implementacji struktury zdarzeń użyto standardowego kontenera typu lista, z racji na to, iż nigdy nie następuje modyfikacja tej struktury – w razie wykrycia przecięcia, funkcja kończy swoje działanie, zwracając wartość `True`. W strukturze tej trzymano unikalne indeksy punktów będących końcami odcinków, posortowane według odpowiednich współrzędnych x.

Struktura stanu została zaimplementowana przy użyciu kontenera `bintrees.RBTree`, tj. drzewa czerwono-czarnego dostępnego w zewnętrznej bibliotece `bintrees`. Kluczami były tu współrzędne y przecięcia miotły z danym odcinkiem, wartościami zaś indeksy wyjściowych odcinków.

Zdarzenia początku odcinka obsługiwane są poprzez dodanie odpowiedniego odcinka do struktury stanu i sprawdzenie przecięć z odcinkami sąsiadującymi. Zdarzenia końca odcinka obsługiwane są poprzez usunięcie odcinka ze struktury stanu i sprawdzenie, czy istnieje przecięcie między jego byłymi sąsiadami. Oczywistym jest fakt braku potrzeby obsługiwania zdarzenia przecięcia odcinków. Przed obsłużeniem każdego zdarzenia, aktualizowane są wartości kluczy w strukturze stanu, tj. obliczane są aktualne wartości współrzędnej y przecięcia odcinka z miotłą, dla każdego odcinka.

4.2 Znajdowanie wszystkich przecięć

Do implementacji struktury zdarzeń użyto kontenera `bintrees.RBTree`, z racji na konieczność dodawania do niej zdarzeń przy okazji wykrycia nowych przecięć, co dzieje się w czasie logarytmicznym. Kluczami były wartości współrzędnych x odpowiednich punktów, zaś wartościami – krotki zawierające typ zdarzenia (typ enumeracyjny o trzech możliwych wartościach) i jeden lub dwa indeksy odcinków, które w danym zdarzeniu uczestniczyły.

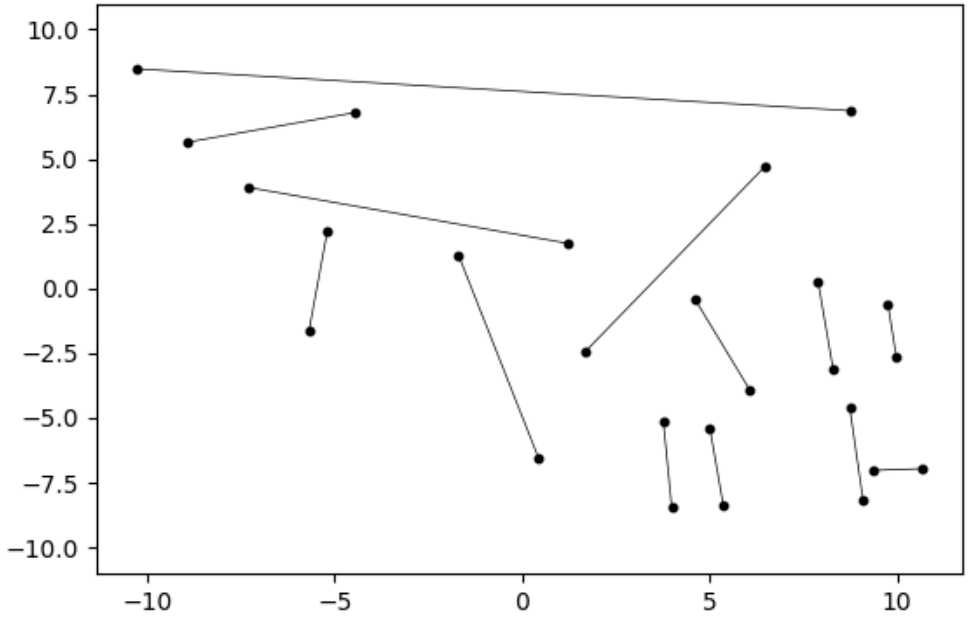
Struktura stanu została zaimplementowana w identyczny sposób jak w poprzednim algorytmie.

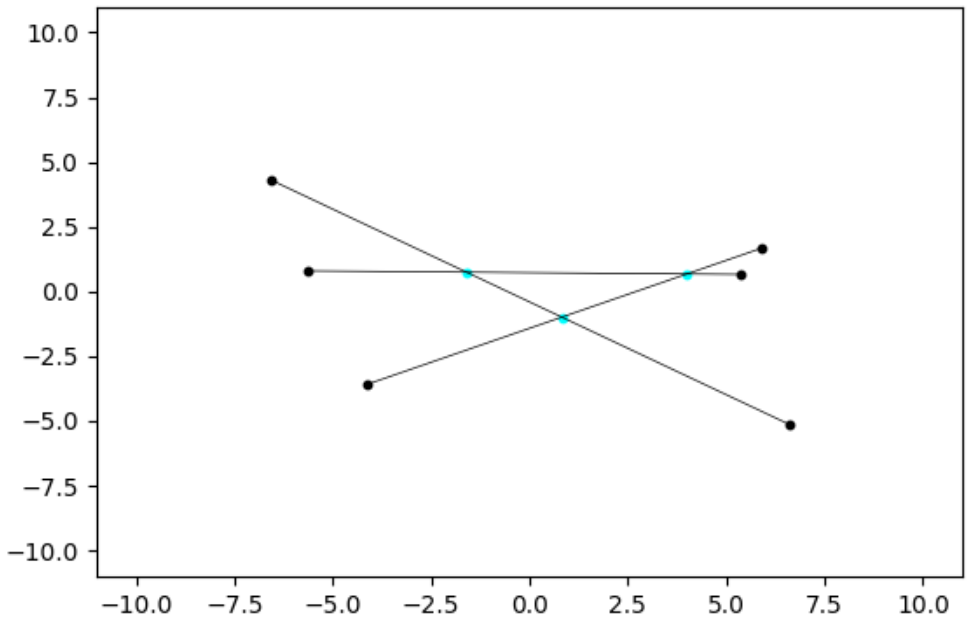
Ponownie, zdarzenia początku odcinka obsługiwane są poprzez dodanie odpowiedniego odcinka do struktury stanu i sprawdzenie przecięć z odcinkami sąsiadującymi; zdarzenia końca odcinka zaś, poprzez usunięcie odcinka ze struktury stanu i sprawdzenie, czy istnieje przecięcie między jego byłymi sąsiadami. Przed obsłużeniem zdarzeń niebędących przecięciami, aktualizowane są wartości kluczy w strukturze stanu, tj. obliczane są aktualne wartości współrzędnej y przecięcia odcinka z miotłą, dla każdego odcinka. Zdarzenia przecięcia natomiast powodują zamianę kluczy odcinków sąsiadujących ze sobą w strukturze stanu, a następnie sprawdzenie przecięć z odpowiednimi odcinkami leżącymi bezpośrednio pod/nad nimi.

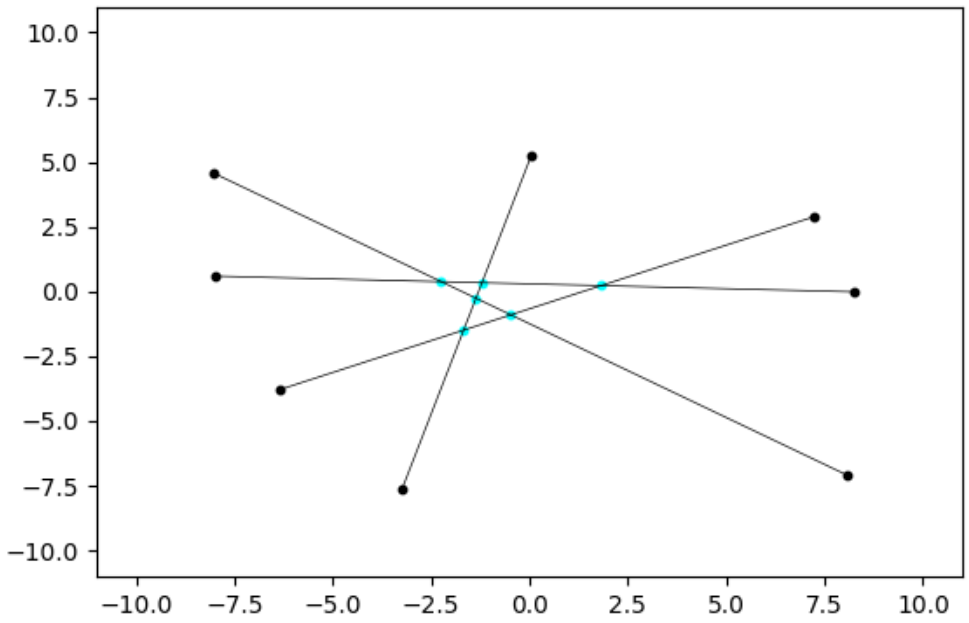
Przy wykryciu przecięcia, sprawdzano, czy nie zostało ono już wcześniej wykryte, do czego użyto kontenera typu set dostępnego w Pythonie, w którym trzymano unikalne indeksy wykrytych zawczasu przecięć.

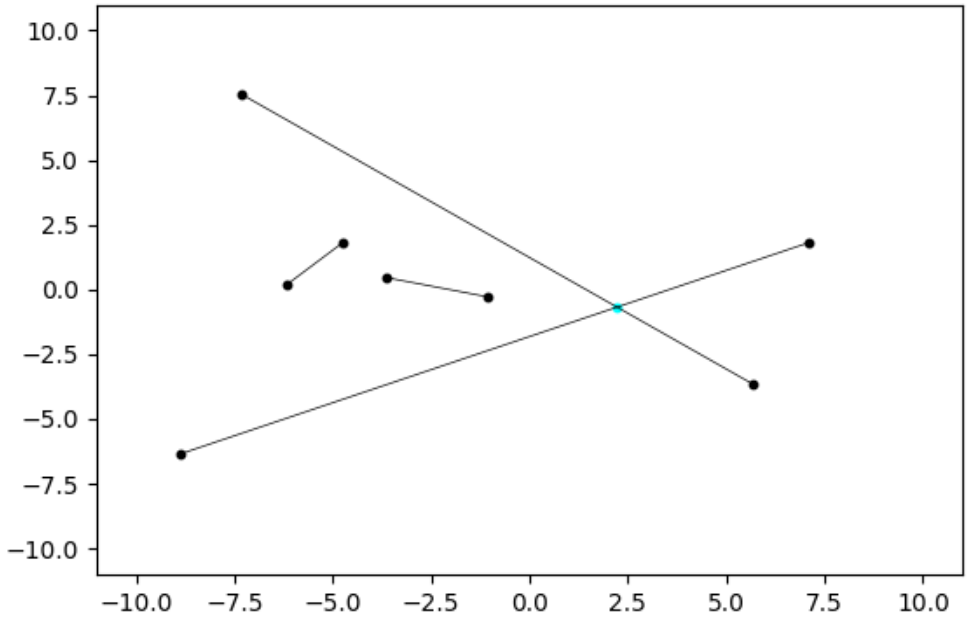
5 Wyniki

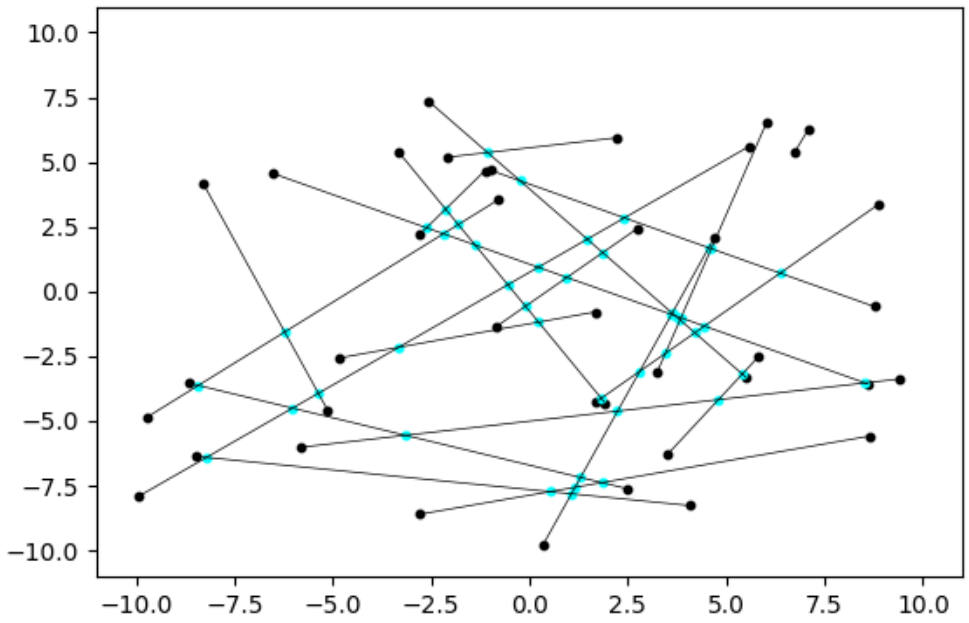
Programy testowano na odcinkach, których wizualizacje są przedstawione na rycinach poniżej. Wykryte przecięcia oznaczono kolorem cyjanowym.






10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



Ryc. 1-5: Testowane odcinki

Pierwszy algorytm w każdym przypadku zwracał poprawną wartość, tj. `False` dla pierwszego zbioru i `True` dla pozostałych. Podobnie, nie stwierdzono błędów podczas wyznaczania żadnego z wyżej wymienionych zbiorów przecięć. Z racji na użycie pomocniczej struktury na znalezione przecięcia odcinków, ponowne wykrycie danego przecięcia (co miało miejsce m.in. dla zbioru czwartego) nie powodowało dodania go po raz kolejny do wyniku działania funkcji.

Pełne wizualizacje kroków działania obu algorytmów dostępne są w załączonym notebooku Jupyter. Przecięcia wykryte zaznaczono tam kolorem cyjanowym, punkt aktualnie rozważany i pozycję miotły – czerwonym. Odcinki aktywne (obecne na w strukturze stanu) wyróżniono kolorem niebieskim.

6 Wnioski

- Sprawdzenie, czy jakiekolwiek dwa odcinki na płaszczyźnie się przecinają, jest możliwe za pomocą algorytmu zamiatania,
- Wykrycie wszystkich takowych przecięć również,
- Struktura zdarzeń użyta w pierwszym przypadku może być "uboższa" niż ta w drugim i nie pozwalać (lub pozwalać w czasie liniowym) np. na dodawanie nowych zdarzeń, co nigdy nie zajdzie.