Szymon Nowak-Trzos 411246
Informatyka rok II, gr.2 środa nieparzysta 13:00-14:30

Algorytmy Geometryczne – Ćwiczenie 4 Sprawozdanie

1. Wstęp

1.1. Cel Ćwiczenia

Celem ćwiczenia była implementacja algorytmu Bentley'ego–Ottmanna do określania przecięć zadanych odcinków.

1.2. Użyte Algorytmy

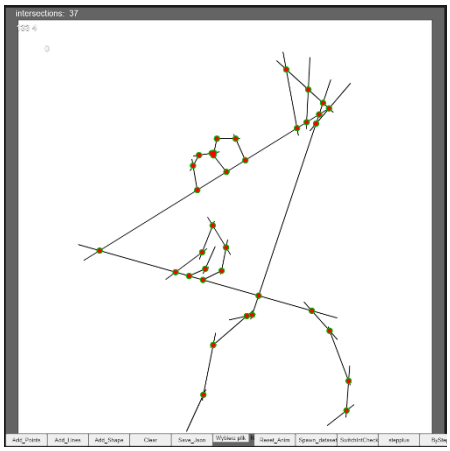
W ramach ćwiczenia zaimplementowane zostały algorytmy:

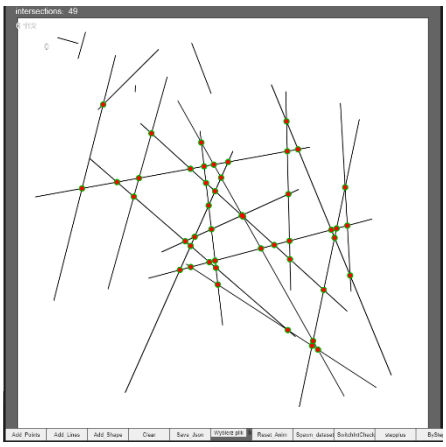
- Sprawdzanie czy jakiekolwiek odcinki się przecinają skróconym algorytmem Bentley'ego–Ottmanna,
- Sprawdzanie przecinania się odcinków algorytmem Bentley'ego–Ottmanna wraz z animacją,
- Sprawdzanie przecinania się odcinków algorytmem naiwnym O(n^2),
- Generowanie **n** odcinków na kanwie.

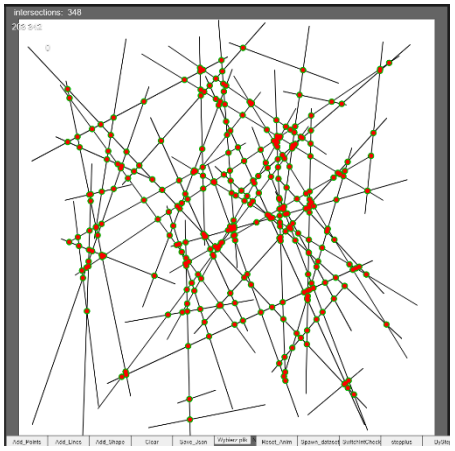
1.3. Użyte Narzędzia

Na potrzeby ćwiczenia zostało stworzone własne narzędzie graficzne obsługujące animacje, dodawanie odcinków, dodawanie wielokątów oraz zapisywanie i odczytywanie listy odcinków z pliku. Narzędzie to napisane zostało w języku JavaScript przy pomocy biblioteki p5.js. Narzędzie znajduje się w załączonym pliku oraz na stronie https://editor.p5js.org/Szyntos/sketches/Dx6ZNHKfl. Algorytm został uruchamiany na procesorze Intel Core i5-1135G7 2.40GHz.

2. Szczegóły wykonywania ćwiczenia







2.1. Algorytmy

2.1.1. Algorytm Bentley'ego–Ottmanna

W kodzie źródłowym algorytm znajduje się w pliku LinesIntersections.js oraz jest nazwany „CheckIntersectionsSweep".

Algorytm został zaimplementowany następująco:

- Tworzymy strukturę zdarzeń Q
- Tworzymy strukturę stanów T
- Tworzymy listę przecięć
- Do struktury zdarzeń dołączamy początki i końce odcinków uporządkowane po współrzędnej x
- Dopóki Q nie jest puste:
    - Wyciągamy zdarzenie z Q o najmniejszej współrzędnej x
    - Jeśli zdarzenie jest początkiem odcinka:
        - Dołączamy odcinek do T
        - Sprawdzamy, czy odcinek przecina się z jego sąsiadami
            - Jeśli tak to dodajemy przecięcie do Q, jeśli nie ma go w liście przecięć
                - Dodajemy przecięcie do listy przecięć
    - Jeśli zdarzenie jest końcem odcinka:
        - Usuwamy odcinek z T
        - Sprawdzamy, czy byli sąsiedzi odcinka przecina się ze sobą

- Jeśli tak to dodajemy przecięcie do Q, jeśli nie ma go w liście przecięć
    - Dodajemy przecięcie do listy przecięć
- Jeśli zdarzenie jest przecięciem odcinków
    - Zamieniamy kolejność odcinków w T
    - Sprawdzamy, czy po zamianie odcinki się przecinają z ich sąsiadami
        - Jeśli tak to dodajemy przecięcie do Q, jeśli nie ma go w liście przecięć
            - Dodajemy przecięcie do listy przecięć
- Zwracamy listę przecięć

Wynikiem zaimplementowanego algorytmu Bentley'ego–Ottmanna jest krotka w postaci [PointsCollection, [[Line, Line]]], gdzie pierwszy element zawiera wszystkie punkty przecięć odcinków, a drugi odpowiadające tym punktom odcinki.

Za każdym razem, gdy jest wykrywane dane przecięcie sprawdzane jest, czy już nie zostało wcześniej zarejestrowane. Jest to osiągnięte poprzez dodawanie par odcinków do HashMapy, gdzie klucz danej pary odcinków jest taki sam niezależnie od ich kolejności. Jeśli tak jest, nie dodajemy go do listy przecięć.

2.1.2. Algorytm Sprawdzający Istnienie Jakichkolwiek Przecięć

Algorytm ten jest modyfikacją powyższego algorytmu Bentley'ego–Ottmanna, gdzie po znalezieniu przecięcia, zamiast dodawać przecięcie do listy, zwracamy od razu wartość 'true'. Jeśli zaś Q się opróżni to zwracamy 'false' oznaczające brak przecięć.

2.1.3. Algorytm Sprawdzania Przecięć Metodą Naiwną O(n^2)

Algorytm ten sprawdza przecięcia każdej linii z każdą.

2.2. Struktury Danych

W narzędziu zaimplementowane zostały następujące klasy:

- Point – posiada współrzędne punktu oraz jego typ
- Line – posiada 2 punkty, początkowy oraz końcowy
- PointsCollection – posiada listę punktów oraz typ
- LinesCollection – posiada listę linii

- Scene – posiada 2 listy obiektów PointsCollection oraz 2 listy obiektów LinesCollection

oraz na potrzeby zadania:

- PriorityQueue – implementacja została zaczerpnięta ze strony https://github.com/eyas-ranjous/datastructures-js
- AVLTree – implementacja została zaczerpnięta ze strony https://github.com/w8r/avl

Każdy odcinek jest przechowywany jako obiekt Line w strukturze LinesCollection jako para obiektów Point.

Struktura zdarzeń Q jest zaimplementowana za pomocą kolejki priorytetowej PriorityQueue. Trzyma ona posortowaną po współrzędnej x listę punktów wraz z informacją o typie punktu (początkowy, końcowy czy przecięcie) oraz do jakiego odcinka lub odcinków jest on przypisany. Pozwala to na szybki dostęp do następnego zdarzenia w algorytmie.

Struktura stanu miotły T jest zaimplementowane jako zbalansowane drzewo binarne AVLTree. Funkcje 'insert', 'pop', 'remove', 'next' oraz 'prev' zostały zmodyfikowane tak, aby porównywanie elementów można było wykonać porównując przecięcia odcinków w drzewie z miotłą. Jest to prawidłowy zabieg, gdyż kolejność odcinków na drzewie się nie zmienia się jedynie przy usuwaniu, dodawaniu oraz zamianie odcinków, gdzie te przypadki są rozpatrywane ręcznie w algorytmie Bentley'ego–Ottmanna.

W przypadku, gdy punkt P zabrany z Q jest początkiem odcinka, wykonujemy następujące kroki:

- Obliczamy wysokość przecięcia odcinka z miotłą
- Wkładamy odcinek do T „w miejscu P.x"
- Szukamy górnego i dolnego sąsiada odcinka w T „w miejscu P.x"
- Sprawdzamy przecięcia sąsiadów z odcinkiem
- Jeśli wykryliśmy niezarejestrowane przecięcie, dodajemy je do Q oraz do listy przecięć

W przypadku, gdy punkt P zabrany z Q jest końcem odcinka, wykonujemy następujące kroki:

- Obliczamy wysokość przecięcia odcinka z miotłą
- Szukamy górnego i dolnego sąsiada odcinka w T „w miejscu P.x"
- Usuwamy odcinek z T „w miejscu P.x"
- Sprawdzamy przecięcia sąsiadów ze sobą
- Jeśli wykryliśmy niezarejestrowane przecięcie, dodajemy je do Q oraz do listy przecięć

W przypadku, gdy punkt P zabrany z Q jest przecięciem odcinków, wykonujemy następujące kroki:

- Usuwamy oba odcinki z T „w miejscu P.x-delta"
- Wkładamy oba odcinki do T „w miejscu P.x+delta"
- Szukamy sąsiadów zamienionych odcinków w T „w miejscu P.x+delta"
- Sprawdzamy przecięcia odcinków z ich sąsiadami w T

- Jeśli wykryliśmy niezarejestrowane przecięcie, dodajemy je do Q oraz do listy przecięć

Wyrażenie „w miejscu P.x” oznacza wykonanie danej operacji na drzewie binarnym tak, że klucze wszystkich elementów są obliczane dynamicznie jako przecięcie danego odcinka z miotłą. Wzajemna kolejność elementów na drzewie się nie zmienia, ponieważ każda zmiana kolejności jest rozpatrywana ręcznie w algorytmie.

Algorytm, który sprawdza istnienie jakichkolwiek przecięć korzysta z tych samych struktur danych.

2.3. Animacje

Animacje zostały zaimplementowane poprzez zatrzymywanie działania algorytmu w kluczowych momentach. Kolorem brązowym zaznaczone zostały odcinki aktualnie znajdujące się na miotle, kolorem niebieskim odcinki należące do sprawdzanego punktu. Punkty przecięć znalezione algorytmem Bentley’ego–Ottmanna zostały zaznaczone na czerwono, punkty znalezione algorytmem naiwnym pokolorowane są zielonym kolorem.

2.4. Uruchamianie Algorytmu

Aby uruchomić algorytm zalecane jest wejście na stronę edytora online kodu p5js https://editor.p5js.org/Szyntos/sketches/Dx6ZNHKfl. W lewym górnym rogu znajduje się przycisk uruchamiający program. Uruchomienie programu z poziomu pliku jest możliwe jeśli program zostanie uruchomiony z poziomu serwera lokalnego, więc uruchamianie ze strony internetowej jest preferowane.

Na kanwie programu znajdują się przyciski:

- Add_Points – dodaje nowy zbiór punktów
- Add_Lines – dodaje nowy zbiór linii
- Add_Shape – dodaje kształt
- Clear – czyści kanwę
- Save_Json – zapisuje aktualną scenę do pliku Json
- Wybierz plik – ładuje plik Json z dysku
- Reset_Anim – resetuje animacje
- Spawn_Dataset – generuje n nie pionowych linii z różnymi współrzędnymi x

  (ze względu na naturę p5js algorytm pracuje na wartościach int z przedziału 30, 970 (rozmiar kanwy w pikselach), więc implementacja dodatkowego wprowadzania zakresu generowanych linii jest bezcelowe, ponieważ i tak musiałby te linie być konwertowane na wartości int)

- SwitchIntCheck – włącza i wyłącza działanie naiwnego algorytmu
- Stepplus – przechodzi do następnego kroku algotytmu
- ByStep – przełącza widok z animacji na wynik algorytmu

3. Podsumowanie

Poprawne wykonanie ćwiczenia wymagało zaimplementowania odpowiednich struktur danych, bez których algorytm działałby o wiele wolniej. Należało również dokładnie przyjrzeć się algorytmowi Bentley'ego–Ottmanna. Algorytm działa poprawnie nawet dla dużych zbiorów odcinków.