Program zawiera algorytmy tworzenia mapy trapezowej do lokalizacji punktu na płaszczyźnie o podziale poligonowym oraz znajdowania wielokąta zawierającego zadany punkt używając stworzonej struktury.

1. Używane technologie

- Python wraz z biblioteką sortedcontainers (http://www.grantjenks.com/docs/sortedcontainers/)
- Jupyter Notebook
- React.js

2. Używanie programu

- **Tworzenie struktury poligonowej.**
  Najlepszym sposobem tworzenia struktury poligonowej jest używanie programu polygons_generator (React.js). Po jej włączeniu (npm start w folderze polygons_generator) należy kliknąć przycisk “enter polygonal space” a następnie klikając na kwadratowym obszarze tworzyć wielokąty. Każde kliknięcie dodaje wierzchołek do wielokąta. Ponowne kliknięcie na wierzchołek, od którego zaczęliśmy tworzyć aktualny wielokąt zakończy proces tworzenia tego wielokąta i go zapisze. Kolejne kliknięcie rozpocznie tworzenie kolejnej figury. Dla każdego wielokąta należy klikać na wszystkie jego wierzchołki, nawet gdy dany punkt był już wcześniej dodany do innego wielokąta. By jakiś punkt był wspólny dla kilku wielokątów trzeba klikać wystarczająco blisko istniejącego już punktu, ponieważ wtedy program automatycznie uzna ten punkt za jeden z należących do aktualnie tworzonego wielokąta. Nie należy tworzyć odcinków przecinających się. Zalecane jest by każdy kolejny wielokąt miał co najmniej jeden punkt wspólny z dowolnym poprzednim. By algorytm działał poprawnie to zadany musi być każdy wielokąt do którego może trafić punkt. W sytuacji niepoprawnego zadania punktu należy odświeżyć stronę i powtórzyć cały proces. Po dodaniu ostatniego wielokąta należy kliknąć przycisk “export polygons to json”. To pobierze nam plik typu json z wielokątami.
  Alternatywnym sposobem jest ręczne stworzenie pliku typu json z zadanymi wielokątami w formacie: [wielokąt1, wielokąt2, …, wielokątn], gdzie wielokąt jest w formacie: [punkt1, punkt2, …, punktn], a punkt w formacie: [współrzędna x, współrzędna y]. Algorytm działa na kwadracie o wierzchołkach: (0,0), (0,400), (400,400), (400,0). Przykład: [[[76.5,290],[254.5,295],[237.5,180],[176.5,181]],[[86.5,370],[76.5,290],[254.5, 295],[133.5,358]]]

- **Przeniesienie pliku**
  Należy umieścić plik typu json z podziałem poligonowym w folderze point_location
- **Stworzenie struktury**
  Należy wpisać do wiersza polecenia: “jupyter notebook” a następnie w przeglądarce otworzyć plik point_location.ipynb. Do stworzenia struktury do wyszukiwania punktów używamy przedostatniej komórki w notebooku. Tam w odpowiednie pole wprowadzamy nazwę pliku typu json z podziałem poligonowym. Przykład nazwy: “polygons.json”. Uruchamiamy wszystkie komórki poza ostatnią. Otrzymujemy wizualizacje tworzenia struktury do przeszukiwań.
- **Lokalizacja punktów**
  Klikamy “dodaj punkt” na naszej wizualizacji podziału poligonowego w notebooku. Następnie klikając na wizualizacji tworzymy punkty, które potem wyszukamy. Po dodaniu wszystkich punktów uruchamiamy ostatnią komórkę notebooka. Na czerwono są zaznaczone wielokąty w których znaleziono któryś z zadanych punktów.

3. Mapa trapezowa

- **Opis**
  Mapa trapezowa składa się z połączonych trapezów (i trójkątów) - klasa Area. Każdy z nich jest powiązany z lewym górnym, lewym dolnym (często jest to ten sam) oraz prawym górnym i prawym dolnym (również często są to te same) sąsiadem. Trapezy są rozdzielone odcinkami tworzącymi wielokąty oraz pionowymi odcinkami wychodzącymi z wierzchołków wielokątów. Zatem struktura mapy trapezowej próbuje odwzorowywać przestrzenny układ trapezów na płaszczyźnie.
- **Tworzenie mapy trapezowej**
  Początkowo jest tylko jeden obszar - prostokąt otaczający przestrzeń poligonową. Następnie przy każdym wywołaniu funkcji update_map aktualizujemy mapę trapezową jeden odcinek (losowy) z podziału poligonowego. Funkcja ta przyjmuje 2 argumenty: trapez zawierający początek nowo dodawanego odcinka oraz sam odcinek. W update_map jest wyszukiwana lista trapezów przecinanych przez nowy odcinek. Następnie przechodzimy po mapie i w miejsce tych obszarów wstawiamy nowe trapezy tak by uwzględniały nowy odcinek. Każdy nowy obszar wstawiamy do listy. Po zakończeniu aktualizacji mapy trapezowej zwracamy te trapezy, które zmieniliśmy oraz nowo wstawione trapezy. Następnie jest aktualizowana struktura przeszukiwań. Po takim dodaniu wszystkich odcinków otrzymujemy

struktury pozwalające potem na efektywne wyszukiwanie obszaru, który zawiera zadany punkt.

4. Struktura przeszukiwań

Strukturą przeszukiwań jest acykliczny graf skierowany, w którym będziemy wyróżniać 3 typy węzłów : punkt, odcinek, obszar.

Inicjalizacja drzewa :

- def init_tree(self): początkowo tworzymy pusty duży obszar, który będzie ograniczeniem dla wszystkich analizowanych obszarów.
- def first_step(self, new_areas): tworzymy pierwszy podział, poprzez wybranie początkowego odcinka ze zbioru linii oraz sprawdzenie jakie nowe obszary <new_areas> on tworzy. Następnie nowo utworzone obszary/linie/punkty umieszczamy w naszym grafie.

Aktualizowanie węzła w drzewie:

- def add_leaf(self, parent, area, left_or_right): funkcja sprawdza czy nowo wstawiany obszar <area> jest już w drzewie, jeśli go nie ma to dodaje go do słownika. Następnie aktualizuje wskazany węzeł <parent> poprzez przypisanie węzłowi nowego syna (nowy obszar).
- def update_tree(self, root, new_ar, left_or_right): mając daną pełną listę nowo powstałych obszarów <new_ar> , wyszukujemy te które podzieliły aktualnie rozważany obszar <root>. Wyróżniamy trzy możliwe podziały : na 4 części , na 3 części oraz na 2 części. Następnie w zależności od tego jak ten podział nastąpił aktualizujemy węzeł w drzewie, poprzez zmianę aktualnego węzła na punkt/linie a następnie stworzenie nowych węzłów które będą odpowiadać nowym obszarom/punktom/linii.

Wyszukiwanie pierwszego obszaru, który przecina nowo dodana prosta:

- def find_area(self, line): funkcja znajduję obszar, w którym zaczyna się nowo dodana linia, iteracyjne wyszukiwanie w drzewie poprzez porównywanie wartości, jeśli węzeł jest punktem to porównujemy wartość na X, jeśli jest linią to porównujemy wartości na Y. Jeśli współrzędne X są takie same to przyjmujemy że nowa linia jest trochę na prawo, jeśli współrzędne Y są takie same, to wtedy położenie określamy na podstawie nachylenia prostych. Funkcja zwraca znaleziony obszar.

Dodatkowe funkcję :

- def build_tree(self, edges_list): w funkcji początkowo jest inicjalizowane drzewo a następnie jest ono aktualizowane wraz z dodaniem nowej krawędzi
- def print_tree(self, root): służy do wypisywania drzewa
- def in_order(self, old_area, new_areas, left_or_right): służy do przechodzenia drzewa w celu odnalezienia węzła o danej wartości

5. Realizacja projektu.

- **Założenia danych wejściowych:**
  Podział poligonowy zadany zgodnie z opisem w punkcie drugim, żaden z odcinków nie jest pionowy, brak punktów o tej samej współrzędnej x-owej.
- **Program tworzący podział poligonowy:**
  Przy prawidłowym używaniu (punkt 2) poprawnie zwraca listę wielokątów podziału poligonowego w formacie json.
- **Budowa struktury pozwalającej na efektywne wyszukiwanie punktu**
  Program poprawnie tworzy i aktualizuje strukturę mapy trapezowej. Istnieje problem przy aktualizacji struktury przeszukiwań, którego nie udało się rozwiązać, stąd program nie działa poprawnie dla wielu przypadków.
- **Lokalizacja punktu**
  Program poprawnie zwraca wielokąt, w którym jest zlokalizowany punkt pod warunkiem poprawnej struktury przeszukiwań.
- **Wizualizacja**
  Wizualizacja poprawnie pokazuje aktualny stan struktury na podstawie drzewa przeszukiwań oraz wielokąty, w których znaleziono punkt.

6. Źródła

“Computational Geometry” - Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars

7. Twórcy projektu

Łukasz Powęska
Samuel Hełdak