**Dokumentacja**

Algorytm wyznaczania Wieloboków Voronoi

dla metryki nieuklidesowskiej wraz z wizualizacją

**Wykorzystany Algorym**

Do rozwiązania zagadnienia użyliśmy **Algorytmu Fortuny**

Program napisaliśmy w języku **Python** w standardzie 3.7

**Złożoność obliczeniowa algorytmu**

**O(n\*logn)**, czyli złożoność algorytmu Fortuny.

Złożoność wynika z wykonania proporcjonalnej do ilości wierzchołków począkowych operacji dodania i usunięcia ze struktur gwarantujących czas logarytmiczny.

**Wykorzystana metryka**

**Metryka maximum**

**d(x,y)=max(|x1-x2|,|y1-y2|)**

**Obsługa**

Po uruchomieniu programu otwarte zostanie okno na którym dodajemy punkty które mają być punktami początkowymi wieloboków. Po dodaniu wszystkich punktów potwierdzamy wybór zamykając okno (przyciskiem x).

Następnie otwarte zostaje okno z wizualizacją, której kolejne kroki przewijamy przyciskiem "następny"

**Ogólny opis plików**

Plik **VoronoiCalculator** jest głównym plikiem przeznaczonym do wykonywania, zawierającym przebieg algorytmu

Plik **MaxMetric** jest plikiem pomocniczym zawierającym metody dotyczące zagadnień matematyczno-geometrycznych

Plik **Plot** zawiera implementację procesu wizualizacji algorytmu

Plik **RBTree** zawiera implementację drzewa czerwono-czarnego

**Szczegółowy opis plików**

Plik **Plot** - zawiera struktury przeznaczone do wizualizacji wzorowane na strukturach wizualizacji wykorzystywanych przez nas na laboratoriach, lecz przystosowane do działania i wizualizacji naszego algorytmu

Plik **RBTree** zawiera implementację drzewa czerwono-czarnego stworzoną przez github.com/**MSingh3012** a poprawioną przez github.com/**zhylkaaa**. Drzewo zawiera metody: insert(key), remove(key), minimum(), maximum(), successor(key), predecessor(key), nodes().

Plik **MaxMetric** zawiera implementacje funkcji geometrycznych dla metryki maximum. Najważniejszymi funkcjami są funkcje **bisector**(a,b) która zwraca symetralną dla dwóch podanych punktów a i b. Symetralna zwracana jest w formie listy odcinków. Kolejną funkcją jest funckcja **cross**(bisect1, bisect2) która dla dwóch symetralnych (listy odcinków) zwraca punkt przecięcia tych odcinków jeśli istnieje, lub False jeśli nie istnieje. Istotną funkcją jest funckja line\_intersection inspirowana funckją ze strony <https://stackoverflow.com/questions/20677795/how-do-i-compute-the-intersection-point-of-two-lines>

która sprawdza czy linie się przecinają.

Dodatkowe funkcje pomocnicze

dla bisector: leftEnd, rightEnd, same\_point, eq

dla cross: findCross, minThisMax, również line\_intersection

Plik **VoronoiCalculator** zawiera główny przebieg algorytmu wraz z obsługą wstawiania i wizualizacji z pliku Plot. Wykorzystuje drzewo czerwono-czarne z pliku RBTree i funkcje bisector oraz cross z pliku MaxMetric.

Wykorzystane struktury:

PriorityQueue importowane z queue

RBTree

Najistotniejsze listy:

output, scenes, points

Cały algorytm zaimplementowany jest w klasie Voronoi

**Funkcje zawarte w klasie Voronoi**: (w opisie pomijam "self")

* \_\_**init**\_\_(points) dodaje punkty jako eventy, inicjalizuje struktury, tworzy prostokąt okalający punkty
* \_make\_scene() dodaje krok do wizualizacji
* **process**() Zawiera przebieg głównej części algorytmu Fortuny. To jest - tak długo jak pozostają eventy do rozpatrzenia zabieramy kolejny event i przetwarzamy go korzystając z podfunkcji
* \_extract\_line\_part(line, a, b) zwraca część linii znajdującą się między a i b
* \_**intersection**\_**to**\_**event** Dodaje przewidywany punkt przecięcia do listy eventów
* \_**process**\_**cell**
* \_**process**\_**intersection**
* \_**process**\_**bend**
* \_process\_bound
* \_finish\_edges
* \_**process**\_**line**
* \_**invalidate**\_**events**

Mniejsze funkcje pomocnicze:

* \_get\_top\_segment, \_get\_mid\_segment, \_get\_bot\_segment
* \_get\_top\_point, \_get\_bot\_point, \_get\_lower\_point, \_get\_higher\_point
* \_is\_in\_segment
* \_segments\_from\_horizontal
* \_get\_bottom\_end
* \_is\_above
* print\_output, get\_output