**Dokumentacja**

Algorytm wyznaczania Wieloboków Voronoi

dla metryki nieuklidesowskiej wraz z wizualizacją

**Autorzy**:

**Adam Kania, Jan Trynda**

**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 początkowych 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

Plik **DataType** zawiera definicje klasy Event przechowującej dane o evencie oraz klasy Cell przechowywującej dane o komórce.

**Opis implementacji algorytmu**

Algorytm opiera się na 4 typach zdarzeń:

* środek komórki – nowy środek komórki zostaje dodany do zbioru aktywnych komórek oraz obliczane są symetralne punktu i środków komórek sąsiednich.
* załamanie symetralnej (punkt w którym symetralna zmienia kierunek lub zaczyna się jej fragment należący do diagramu) – obliczony zostaje punkt przecięcia z symetralnymi leżącymi po przeciwnych stronach punktów wyznaczających rozważaną symetralną.
* przecięcie symetralnych – jeżeli środka komórka wyznaczająca punkt przecięcia zostaje odcięta to komórka ta jest usunięta ze zbioru aktywnych komórek. W miejscu przecięcia ostają dodane dwa zdarzenia będące załamaniami symetralnej (lub 1 zdarzenie jeżeli jedna z komórek została usunięta ze zbioru aktywnych komórek)
* granica obszaru (potrzebny ze względu na wizualizację) – dodaje linie dochodzące do granicy rozważanego obszaru do diagramu Voronoi.

**Wykorzystywane struktury:**

* drzewo czerwono-czarne – przechowuje wszystkie aktywne komórki posortowane po współrzędnej x środka komórki. Umożliwia szybkie znajdowanie komórek sąsiednich.
* Kolejka priorytetowa – przechowuje eventy. Zawsze zwraca event z najniższym kluczem.

**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 funkcja **cross**(bisect1, bisect2) która dla dwóch symetralnych (listy odcinków) zwraca punkt przecięcia tych odcinków jeśli istnieje, lub False jeśli nie istnieje. Istotną funkcją jest funkcja line\_intersection inspirowana funkcją 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 **DataType** zawiera definicję klas:

* Event, która przechowuje następujące dane: współrzędne eventu, jego klucz, komórki diagramu powiązane z eventem (komórka po lewej i po prawej), typ eventu, wartość boolowską opisującą czy dany event jest nadal ważny oraz odcinki, które należy dodać do diagramu Voronoi.
* Cell, która przechowuje następujące dane dotyczące komórki: położenie, symetralne wyznaczone przez sąsiednie komórki, eventy związane z komórką (celem ewentualnego oznaczenia ich jako nieważne), ostatnie punkty leżące na symetralnych dodane do diagramu Voronoi.

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 – zawiera odcinki tworzące diagram Voronoi

scenes – zawiera sceny do wizualizacji

points – zawiera punkty będące środkami komórek w diagramie Voronoi

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 –** przetwarza eventy będące środkami komórek. Dodaje komórkę do struktury przechowywującej aktywne komórki i szuka symetralnych oraz ewentualnych punktów przecięcia.
* \_**process**\_**intersection -** przetwarza eventy będące punktami przecięcia. Jeżeli jakaś komórka przestaje być aktywna usuwa ją ze struktury aktywnych komórek. Szuka ewentualnych punktów przecięcia.
* \_**process**\_**bend -** przetwarza eventy będące zgięciami linii. Funkcja dodaje początkowy kawałek linii do diagramu Voronoi i szuka punktu przecięcia.
* **\_process\_bound** - przetwarza eventy będące przecięciami z granicą przetwarzanego obszaru.
* **\_finish\_edges** - przetwarza wszystkie eventy związane z granicą przetwarzanego obszaru.
* \_**invalidate**\_**events** - oznacza przestarzałe eventy jako nieważne, aby nie zostały przetworzone.

Mniejsze funkcje pomocnicze:

* \_get\_top\_segment, \_get\_mid\_segment, \_get\_bot\_segment – funkcje wyciągają z linii odpowiednio górny, środkowy i dolny segment (prosty odcinek)
* \_get\_top\_point, \_get\_bot\_point – funkcje zwracają punkt załamania linii będący najbliżej odpowiednio górnego i dolnego końca linii, ale różnego od punktu końcowego
* get\_lower\_point, \_get\_higher\_point – z dwóch przekazanych punktów zwracają niższy lub wyższy
* \_is\_in\_segment – sprawdza czy punkt leży w obszarze zadanego odcinka linii
* \_get\_bottom\_end – zwraca dolny koniec linii