Dokumentacja - Część techniczna

Mateusz Słuszniak, Paweł Lewkowicz

1.Ogólne Opis działania programów

Na program składają się dwie wersje algorytmu szukającego Diagramów Voronoi dla zadanego zestawu punktów. Pierwsza z nich to implementacja algorytmu Fortune'a , drugi zaś algorytm wykorzystuje najpierw triangulacje Delaunaya i na podstawie tej triangulacji wyznacza Diagram Voronoi. W tej i kolejnych sekcjach podzielę opisy na implementacje algorytmu Fortune'a oraz algorytmu wykorzystującego triangulacje Delaunaya.

Algorytm Fortune'a

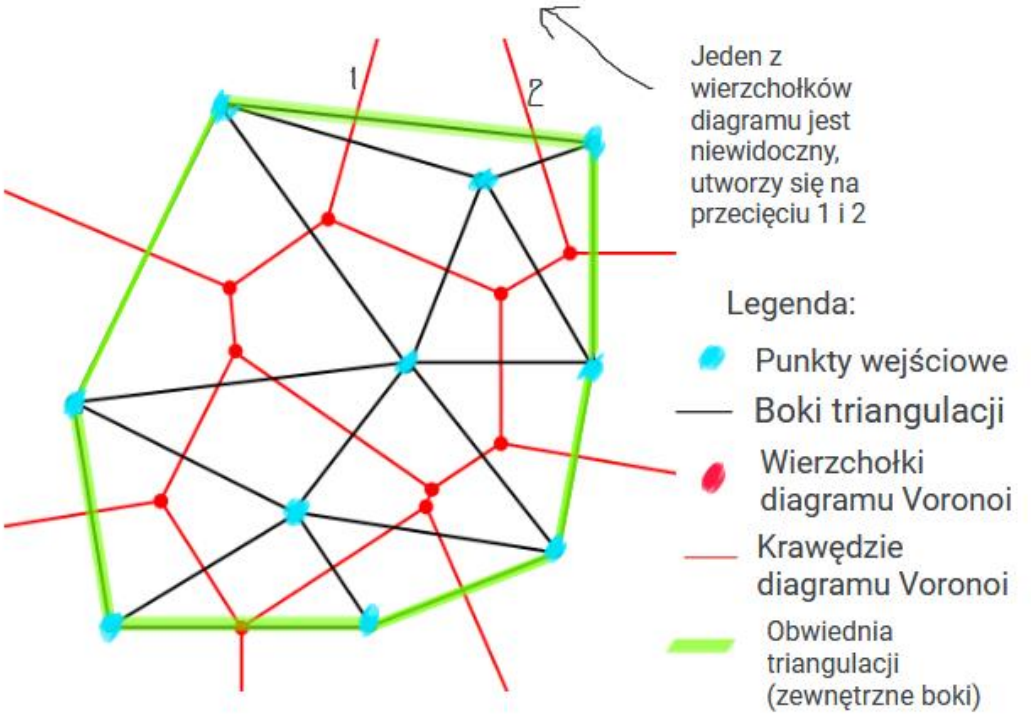
Nasz algorytm Fortune'a został podzielony na moduły, z których każdy odpowiada za inną część działania programu. Poprawia to czytelność kodu przez hermetyzację. Głównym plikiem, który stanowi jądro algorytmu jest plik *Fortune_algorithm.py*, w którym tworzony jest diagram Voronoi. Warto tutaj zwrócić uwagę na to, że pojawiają się pliki *Fortune_algorithm.py* oraz *Fortune_algorithm_time.py*. Pierwszy z nich oprócz diagramu tworzy też wizualizację, która następnie jest pokazywana w notatniku Jupyter. Druga wersja jest pozbawiona tych dodatków aby, jak sama nazwa wskazuje, podczas mierzenia czasu działania pokazywała faktyczny czas obliczania diagramu Voronoi, a nie uwzględniała dodatkowo czas poświęcony na obsługę animacji. W pliku *Fortune_algorithm.py* znajdują się oczywiście odpowiednie metody, które są szczegółowo opisane w sekcji drugiej tego dokumentu. Oprócz tego pliku pojawiają się między innymi *Beach_line.py*. Plik ten odpowiada za implementację struktury linii brzegowej, która jest realizowana jako drzewo czerwono czarne wzbogacone o pewne dodatkowe funkcje. W nodach drzewa trzymane są obiekty reprezentujące łuki. Plik *Event.py* zawiera implementacje zdarzeń (zarówno punktowych jak i okręgowych). Same eventy w algorytmie są trzymane w kolejce priorytetowej (from queue import PriorityQueue która stanowi strukturę zdarzeń. *Metric.py* zawiera implementacje, dwóch kluczowych operacji (znajdowanie środka okręgu opisanego dla trzech punktów oraz znajdowanie przecięć parabol) dla ustalonej metryki, w naszym przypadku euklidesowej. Visualization.py to narzędzie graficzne udostępnione na początku semestru. *Voronoi_diagram.py* stanowi implementacje obiektu, który przechowuje krawędzie diagramu, komórki oraz wierzchołki. Oferuje ona również metody związane m. in. z dodawaniem półprostych i punktów do diagramu. Realizuje ona listę dwukierunkową poprzez origin i destination. Ostatnim plikiem jest *Voronoi_diagrams.ipynb.* Zawiera on wizualizację kolejnych etapów działania algorytmu Fortune'a dla metryki euklidesowej. Oprócz tego oferuje on możliwość pomiaru czasu działania algorytmu.

Algorytm oparty o triangulację Delaunaya

Drugi z algorytmów znajdujący diagram Voronoi jest oparty na triangulacji Delaunaya. Implementacje triangulacji Delaunaya wzięliśmy z modułu scipy.spatial, który z kolei korzysta z innej biblioteki, a mianowicie Qhull (biblioteka do algorytmów geometrycznych). Mając daną triangulacje w postaci trójek punktów, które tworzą trójkąt triangulacji, można w relatywnie prosty sposób

uzyskać diagram Voronoia. Należy dla każdego z trójkątów znaleźć ich środki okręgów opisanych. Środki te stanowią wierzchołki diagramu Voronoia. Następnie należy łączyć ze sobą te wierzchołki diagramu, dla których bok pojawiał się w obydwu trójkątach (trójkątach, których środkami są wierzchołki diagramu, patrz rysunek poniżej). Część boków będzie pojawiała się tylko raz, będą to boki stanowiące obwiednie triangulacji. Wówczas od punktu, który jest środkiem okręgu opisanego na trójkącie zawierającym dany bok, należy poprowadzić półprostą. Półprosta ta będzie prostopadła do „osamotnionego” boku trójkąta.

Rys. 1 Diagram Voronoi (czerwnoe kropki i krawędzie) uzyskany na podstawie triangulacji Delaunaya



2.Opisy działania poszczególnych metod w modułach

Algorytm Fortune’a

Beach_line.py

```
def get_arc_above(self, point, l):
    '''
    jest to funkcja znajdująca jaki łuk z drzewa znajduje się nad zadanym punktem
    :param point: punkt nad którym szukamy
    :param l: kierownica dla łuków parabol
    :return: zwracany jest node który zawiera łuk, który znajduje się nad punktem
    '''
```

Reszta funkcji w tym pliku jest związana bezpośrednio z implementacją drzewa czerwono-czarnego, dlatego ich opisów nie dodaje.

Event.py

```
def __init__(self, y, event_type, site=None, arc=None, point=None):
    '''
    deklaracja eventu i jego parametrów
    :param y: współrzędna y eventu, po tym jest sortowany event w
strukturze zdarzeń
    :param event_type: rodzaj eventu, punktowy albo okręgowy
    :param site: związany z eventem obszar
    :param arc: łuk związany z eventem
    :param point: punkt związany z eventem
    '''
```

```
def __lt__(self, other):
    #jest też zdefiniowane porównywanie pomiędzy obiektami typu event, żeby
nie robić tego w priority queue
```

Metric.py

```
def compute_breakpoint(point1, point2, l):
    """
    :param point1: ognisko paraboli związanej z pierwszą parabolą
    :param point2: ognisko paraboli związanej z drugą parabolą
    :param l: współrzędna y kierownicy dla obu parabol
    :return: zwraca punkt przecięcia parabol (breakpoint)
    """
```

```
def compute_convergence_point(point1, point2, point3):
    """
    Kiedy zachodzi zdarzenie okręgowe to należy wyznaczyć środek okręgu
opisanego na tych trzech punktach oraz najmniejszą współrzędną y okręgu
    :param point1: pierwszy punkt
    :param point2: drugi punkt
    :param point3: trzeci punkt
    :return: zwraca punkt będący środkie okręgu opisanego na trzech
punktach a także położenie y najniższego punktu okręgu
    """
```

Voronoi_diagram.py

```
class VoronoiDiagram:
    '''
    struktura przechowuje diagram Voronoia, a w nim półproste, wierzchołki
diagramu, oraz site (które trzymają dodatkowo oprócz punktu jeszcze face
    czyli ściane,komórke diagramu)
    '''
```

Fortune_algorithm.py

```
def diagram_edges(diagram):
    '''

    :param diagram: diagram czyli parametr FortuneAlgorith
    :return: edges; lista krawędzi w postaci tupli dwóch punktów
    '''
```

```
def __init__(self, points, named_metric='euclidean_2d', metric=None):
    '''
    inicjalizuje podstawowe struktury algorytmu zamiatającego, linie
brzegową oraz metryke w jakiej wyznaczamy diagram, a także sam pusty
diagram
    :param points: lista punktów dla których wyznaczamy diagram Voronoi
    :param named_metric: Wybrana metryka, co prawda mamy zrobioną tylko
metrykę euklidesową 2d, ale zostawiamy furtkę na inne metryki
    :param metric: wybrana metryka
    '''
```

```
def construct(self, points):
    """
    Wykonuje właściwe działanie algorytmu poprzez obsługę zdarzeń typu
zdarzenie punktowe oraz zdarzenie kołowe.
    Oprócz tego obługuje tworzenie się scen wizualizacji w miarę
obsługiwania kolejnych eventów.
    :return: Scenes; lista obiektów typu Scene z narzędzia graficznego
    """
```

```
def break_arc_by_site(self, arc, site):
    """
    :param arc: łuk, który ulegnie podziałowi na trzy mniejsze łuki
    :param site: punkt, który powoduje podział łuków na trzy
    :return: middle arc; jest to łuk związany z punktem wykrytym w ramach
site eventu
    """
```

```
def add_edge(self, left, right):
    '''
    funkcja dodaje półprostą w momencie obsługiwania zdarzenia punktowego
bądź też usuwania łuku
    :param left: jest to lewy łuk, wtedy półprosta z nim związana jest
prawą półprostą
    :param right: jest to prawy łuk względem lewego, wówczas dodawana
półprosta jest jego lewą półprostą
    '''
```

```
def add_event(self, left, middle, right, events, beachline_y):
    '''
    funckja odpowiada za dodanie zdarzenia (zdarzenie kołowe) do struktury
zdarzeń (kolejki priorytetowej).
    Zdarzenie nie może być fałszywym alarmem, więc sprawdzamy to w is_valid
    :param left: lewy łuk
    :param middle: środkowy łuk
    :param right: prawy łuk
    :param events: struktura zdarzeń (kolejka priorytetowa)
    :param beachline_y: linia brzegowa (drzewo czerowono czarne,
wzbogacone)
    '''
```

```
def handle_site_event(self, event, not_valid_events, events):
    '''
    funckcja odpowiada za obsługę zdarzenia punktowego, napierw dzieli łuk
na trzy łuki po napotakniu punktu
    Następnie zaś jeśli jest to możliwe, dodaje zdarzenie kołowe (muszą być
trzy punkty związane z kolejnymi łukami obok siebie)
    :param event: aktualnie obsługiwane zdarzenie
    :param not_valid_events: lista fałszywych alarmów
    :param events: struktura zdarzeń
    '''
```

```
def remove_arc(self, arc, vertex):
    '''
    funkcja odpowiadająca za usunięcie łuku, który zapadł się właśnie do
punktu diagramu voronoi
    Oprócz usunięcia łuku z linii brzegowej, to dodaje ona utworzoną dzięki
temu półprostą, a następnie domyka ją do krawędzi
    :param arc: Łuk który właśnie się zapadł do punktu i który należy
usunąć
    :param vertex: jest to punkt, do którego właśnie zapadł się łuk
paraboli
    '''
```

```
def handle_circle_event(self, event, not_valid_events, events):
    '''
    Metoda odpowiedzialna za obsługę zdarzenia okręgowego , najpierw do
diagramu Voronoia zostaje dodany nowo utworzony punkt
    Następnie z linii brzegowej zostaje usunięty łuk, który właśnie zapadł
się do punktu
    Na końcu sprawdza czy może w łukach nie są trzymane fałszywe alarmy,
jeśli tak to dodaje je do setu not_valid_events
    na samym końcu próbuje jak poprzednio dla zdarzeń punktowych dodać
zdarzenia okręgowe
    :param event: zdarzenie aktualnie obsługiwane (jest ono zdarzeniem
okręgowym)
    :param not_valid_events: zbiór fałszywych alarmów
    :param events: struktura zdarzeń
    '''
```

```
def adjust_box(self, x_left, y_left, x_right, y_right, points):
    '''
    metoda ta dopasowuje rysowanie półprostych tak, żeby półproste rysowane
na końcu, które odpowiadają za nieskończone obszary diagramu Voronoi
    przylegały do obramówki plota
    :param x_left: współrzędna x lewego dolnego rogu plota
    :param y_left: współrzędna y lewego dolnego rogu plota
    :param x_right: współrzędna x prawego górnego rogu plota
    :param y_right: współrzędna y prawego górnego rogu plota
    :param points: punkty przekazane do znajdowania diagramu Voronoia
    :return:
    '''
```

```
def get_intersection(self, x_left, y_left, x_right, y_right, origin,
direction):
    '''
    funkcja jest wywoływana na końcu, gdy należy obsłużyć zalegające
jeszcze łuki w linii brzegowej po przejściu wszystkich zdarzeń ze struktury
    zdarzeń. wtedy wyznacza intersection które jest punktem przecięcia się
półprostych odpowiedzialnych za nieskończone obszary diagramu Voronoi
    :param x_left: współrzędna x lewego dolnego rogu plota
    :param y_left: współrzędna y lewego dolnego rogu plota
    :param x_right: współrzędna x prawego górnego rogu plota
    :param y_right: współrzędna y prawego górnego rogu plota
    :param origin: punkt będący środkiem krawędzi łączącej początki
półprostych, dla których szukamy punktu przecięcia
    :param direction: jest to wektor powstały poprzez odjęcie współrzędej
jednego z początków i drugiego z początków. Następnie współrzędne
    x i y są zamienione , zaś współrzędna x ( już po zamianie_ jest mnożona
przez -1. Jest to robione dla uproszczenia potem obliczeń
    Same obliczenia do wyznaczania punktu przecięcia wynikają z geometrii
analitycznej , zaczerpnęliśmy je z
https://github.com/pvigier/FortuneAlgorithm/blob/master/src/Beachline.cpp
    :return:
    '''
```

```
def bound(self, x_left=float("inf"), y_left=float("inf"), x_right=-
float("inf"), y_right=-float("inf")):
    '''
    funkcja wywoływana na końcu constructa. Odpowiada ona za dodanie
odcinków symbolizujących nieskończone półproste diagramu voronoia
    :param x_left: współrzędna x lewego dolnego rogu plota
    :param y_left: współrzędna y lewego dolnego rogu plota
    :param x_right: współrzędna x prawego górnego rogu plota
    :param y_right: współrzędna y prawego górnego rogu plota
    '''
```

```
def left_and_right_bound(self, x_left=float("inf"), y_left=float("inf"),
x_right=-float("inf"),
                         y_right=-float("inf")):
    #funckja przydatna do określenia dokąd należy rysować parabole (żeby
nie wystawały znacznie poza wykres)
```

Notebook dla tej implementacji nie wymaga tłumaczenia, wywołane są dwie metody z Fortune_algorithm i generowanie określonych punków.

Algorytm oparty o triangulację Delaunaya

Szczegółowe opisy każdej z funkcji wchodzącej w skład znajdują się nad każdą z funkcji w komórce notebooka zawierającego implementacje algorytmu *Voronoi_diagrams.ipynb.*

3. Specyfikacja techniczna

W przeprowadzonym projekcie dla zaimplementowanego algorytmu zastosowano precyzję obliczeń na poziomie 10^{-15}. Dokładność porównywania współrzędnych wierzchołków i ich klasyfikacja została zrealizowana na poziomie 10^{-9}. Obliczenia były przeprowadzane na architekturze systemu 64-bitowego oraz procesorze IC i7-8750H. Algorytmy były uruchamiane w Pythonie wersji 3.9.1.

4. Dane bibliograficzne

Tutaj zamieszczam wszelkie dane bibliograficzne z których korzystaliśmy podczas pisania algorytmów, sprawozdań, prezentacji itd.

Zagadnienia teoretyczne projektu:

1. Mark de Berg - "Computational Geometry - Algorithms and Applications"

2. Barbara Głut – Prezentacje z wykładu poświęcone Diagramom Voronoi

3. Współrzędne środka okręgu opisanego na trójkącie
https://en.wikipedia.org/wiki/Circumscribed_circle

4. Definicja paraboli jako zbioru punktów równoodległych od kierownicy i ogniska paraboli
https://pl.wikipedia.org/wiki/Parabola_(matematyka)

Zagadnienia implementacyjne:

1. Triangulacja Delaunaya z biblioteki Scipy
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html

2. Diagramy Voronoi z biblioteki Scipy
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html

3. Pomysł na realizacje parabol w narzędziu graficznym jako „gęstego” zbioru punktów
https://katarzynabuzniak.pl/python-wykres-funkcji-kwadratowej/

4. Implementacja drzewa czerwono-czarnego z której skorzystaliśmy
https://github.com/keon/algorithms/blob/master/algorithms/tree/red_black_tree/red_black_tree.py

5. Ogólne inspiracje, w szczególności obsługa łuków po obsłużenia struktury zdarzeń, znajdowanie łuku znajdującego się nad punktem
https://github.com/pvigier/FortuneAlgorithm/blob/master/src/Beachline.cpp

Ciekawe zdjęcia i ilustracje:

1. https://vividmaps.com/nearest-national-capital-europe/

2. https://researchgate.net/figure/fig5_320027110

3. https://towardsdatascience.com/how-to-find-the-nearest-hospital-with-voronoi-diagram-63bd6d0b7b75

4. https://www.codingame.com/playgrounds/243/voronoi-diagrams/what-are-voronoi-diagrams

5.
http://jwilson.coe.uga.edu/EMAT6680Fa08/Kuzle/Math%20in%20Context/Voronoi%20diagrams.html