Przedstawienie metod Konstrukcji:

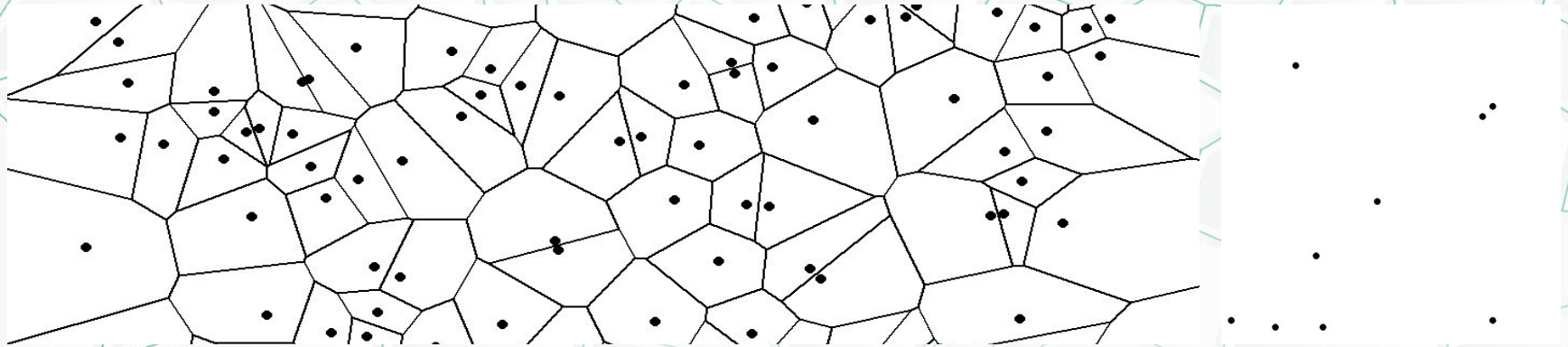
Wieloboki Voronoi

Przygotowali:
Piotr Rzadkowski
Olgierd Smyka

Wieloboki Voronoi - przedstawienie problemu

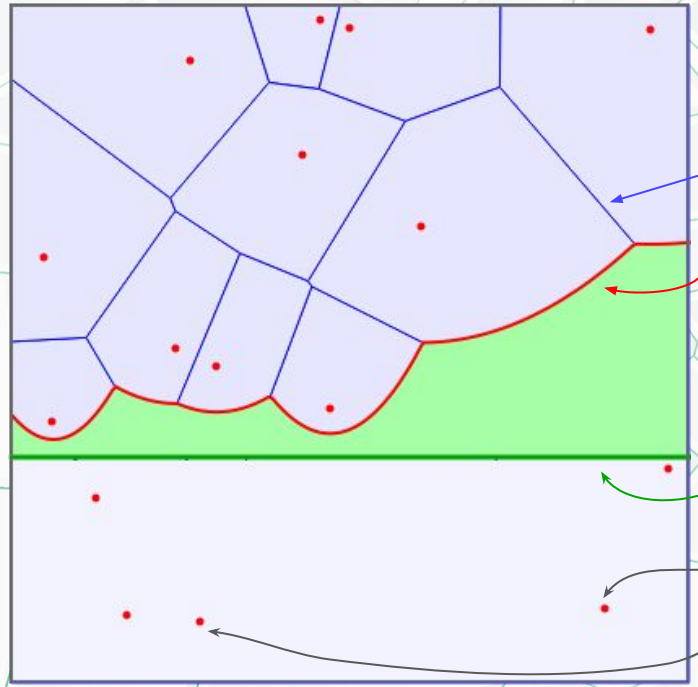
Dla chmury n punktów P = {P1, P2, …, Pn} ⊂ R² wieloboki Voronoi definiuje się jako:
$\mathbf{V_i = \{x \in R^2: d(x,P_i) < d(x,P_j)\ \forall\ j \neq i\ j = 1,\dots,n\}}$,
gdzie: d(.,.) – oznacza odległość (u nas z metryki euklidesowej)

Rozpatrując przestrzeń R² z definicji łatwo zauważyć, że chmura n punktów wejściowych wygeneruje n obszarów (komórek Voronoi), w taki sposób, że dla dowolnego z obszarów dowolny punkt w obszarze będzie znajdował się bliżej określonego punktu ze zbioru punktów, niż od pozostałych n-1 punktów chmury.



https://en.wikipedia.org/wiki/Voronoi_diagram#/media/File:Voronoi_growth_euclidean.gif

Algorytm Fortune’a - idea



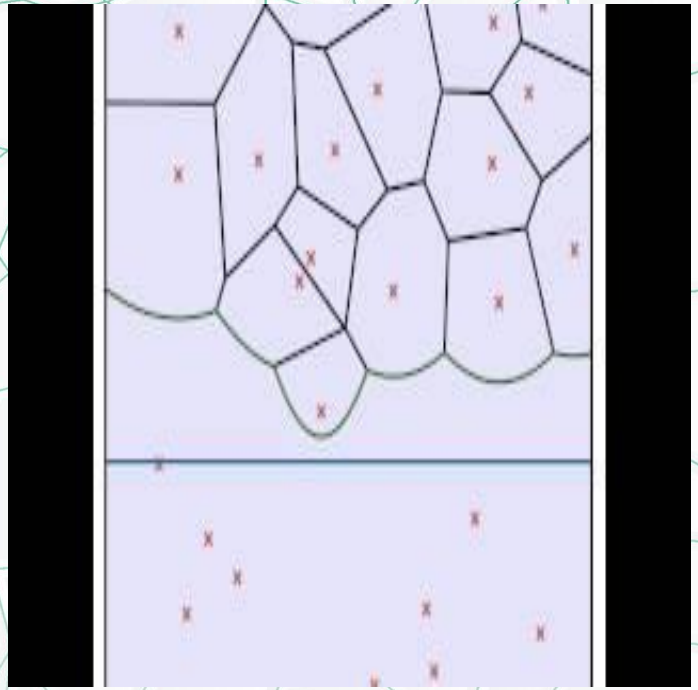
Znalezione krawędzie diagramu Voronoi

Linia brzegowa (struktura stanu T)

Miotła

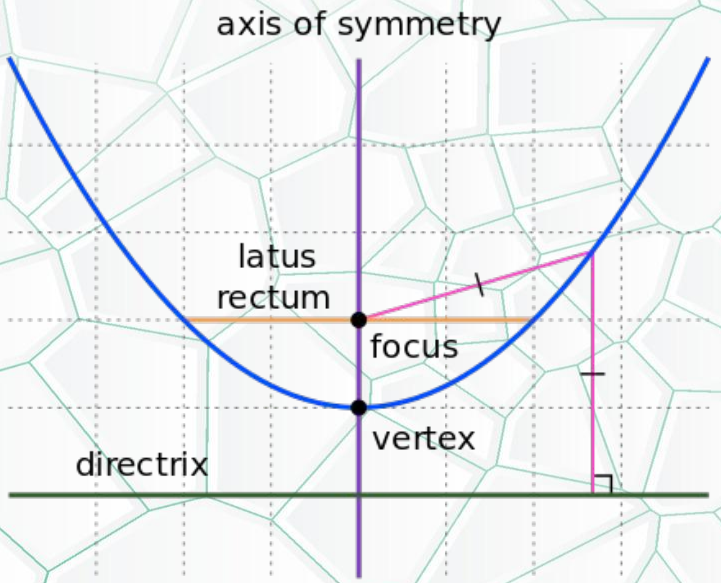
Zdarzenia punktowe

https://demonstrations.wolfram.com/FortunesAlgorithmForVoronoiDiagrams/



https://www.youtube.com/shorts/k2P9yWSMaXE

Postać łuku - parabola z kierownicą i ogniskiem

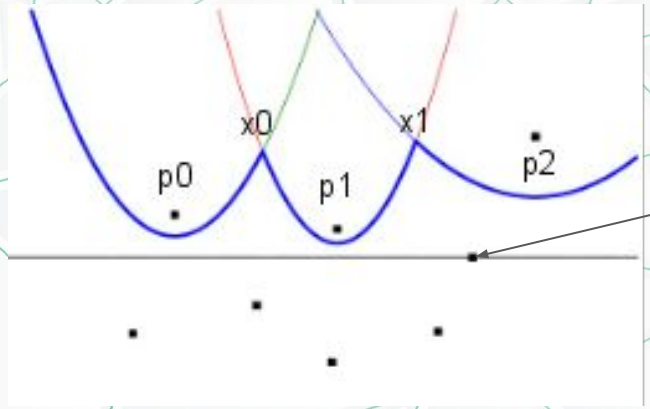


https://en.wikipedia.org/wiki/Parabola

```python
class Arc:
    directrix: float
    objects = []

    def __init__(self, focus):
        self.focus = focus
        self.a = None
        self.b = None
        self.c = None

    @classmethod
    def setDirectrix(cls, directrix, all=True):
        cls.directrix = directrix
        if all:
            for obj in cls.objects:
                obj.updateABC()

    def updateABC(self):
        if self.focus.y == self.directrix:
            return
        f = abs(self.focus.y - self.directrix)/2.0
        vertex = Point(self.focus.x, self.focus.y - f)
        self.a = 1.0 / (4*f)
        self.b = -vertex.x / (2*f)
        self.c = vertex.x**2 / (4*f) + vertex.y

    def intersect(self, arc) -> Point:
        a = self.a - arc.a
        b = self.b - arc.b
        c = self.c - arc.c
        x = (-b + sqrt(b**2 - 4*a*c))/(2*a)
        return Point(x, self.__unit_val(x))
```

Algorytm Fortune’a - zdarzenia punktowe

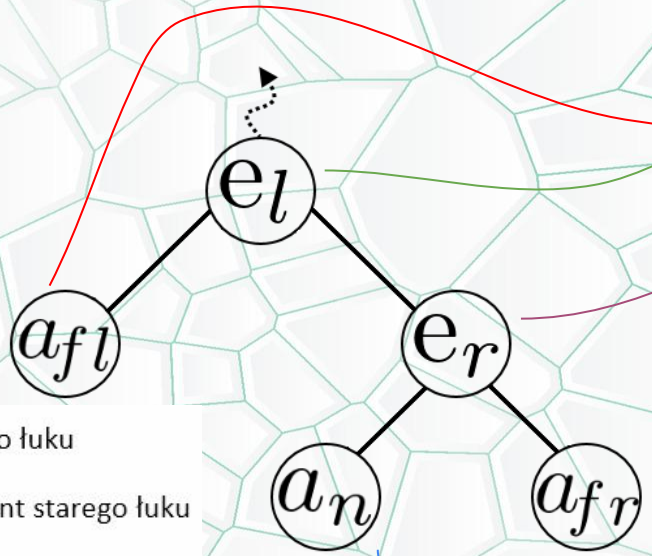
- Zdarzenia punktowe:
- Tworzone są tylko raz przy rozpoczęciu algorytmu, dla punktów z wejścia
- Zdarzenia punktowe dodają nowe łuki do struktury T

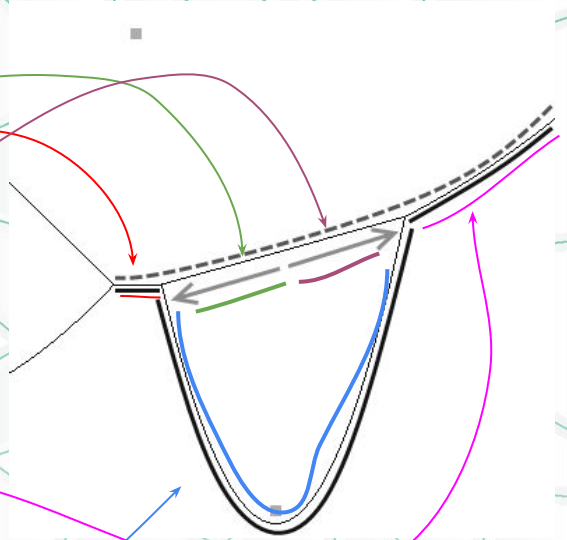


https://blog.ivank.net/fortunes-algorithm-and-implementation.html

- Zdarzenie punktowe dotyczące łuku o ognisku w punkcie p2

Algorytm Fortune’a - zdarzenia punktowe cz.2





a_{fl} – lewy fragment rozdzielonego łuku

e_l – krawędź łącząca lewy fragment starego łuku z rozdzielonym łukiem

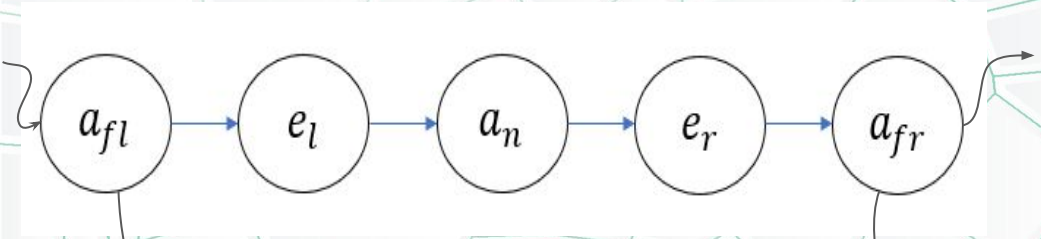
a_n – nowy łuk

e_r – krawędź łącząca nowy łuk z prawym fragmentem rozdzielonego łuku

a_{fr} – prawy fragment rozdzielonego łuku

Algorytm Fortune’a - zdarzenia punktowe cz.3

W naszym przypadku:



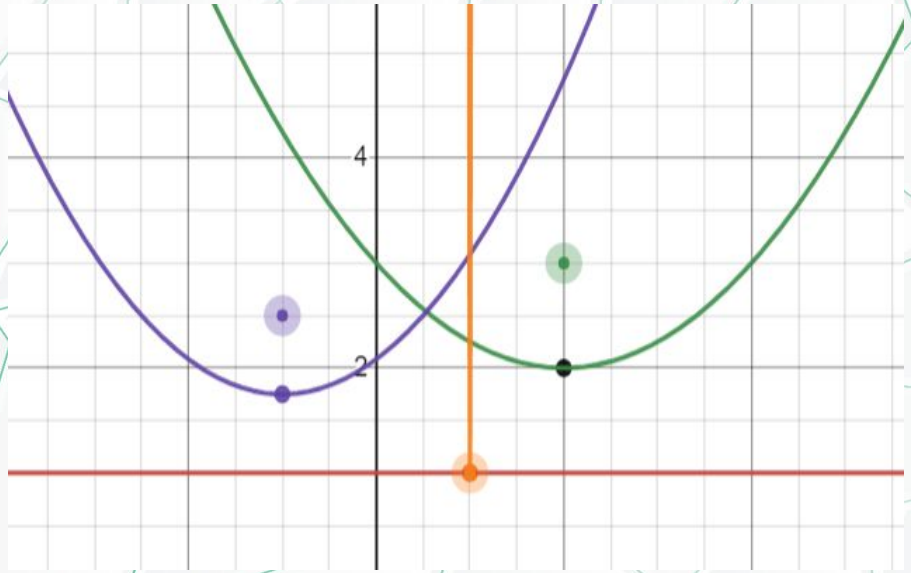
Należy sprawdzić zdarzenia okręgowe!

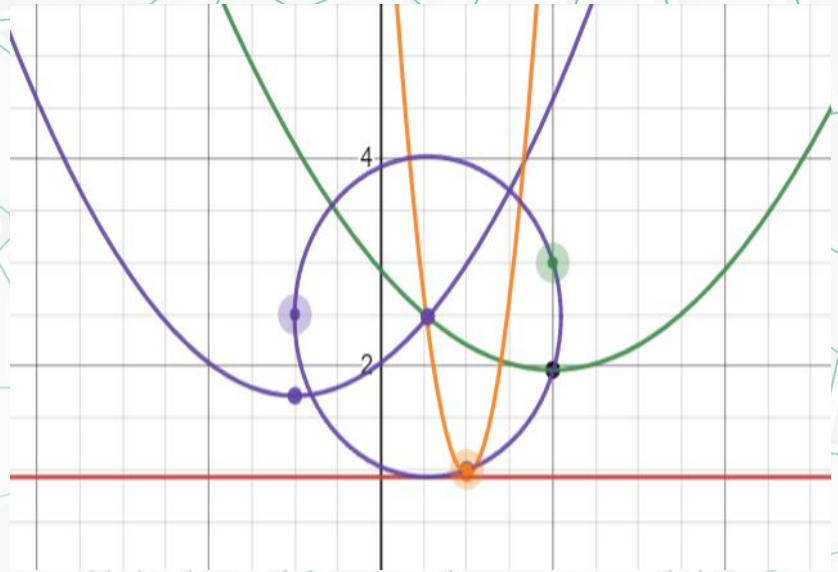
```
def replace(self, arc_node: Arc, new_arc: Arc):
    arc = arc_node.arc
    old_arc_l = Arc(arc.focus)
    old_arc_r = Arc(arc.focus)
    new_arc.setLeftEdge(old_arc_l)
    new_arc.setRightEdge(old_arc_r)
    old_arc_l.edgeGoingLeft = arc.edgeGoingLeft
    old_arc_r.edgeGoingRight = arc.edgeGoingRight

    #   old_arc_l → old_new → new_arc → new_old → old_arc_r
    old_new_node = Node()
    old_new_node.arc_pair = Pair(old_arc_l, new_arc)

    old_arc_leaf_l = Node()
    old_arc_leaf_l.arc = old_arc_l

    new_old_node = Node()
    new_old_node.arc_pair = Pair(new_arc, old_arc_r)

    new_arc_leaf = Node()
    new_arc_leaf.arc = new_arc

    old_arc_leaf_r = Node()
    old_arc_leaf_r.arc = old_arc_r
```

… i odpowiednio podłączyć do T.

Zdarzenia okręgowe - co to jest?

- Pojawiają się gdy łuk zostaje “zakryty” przez sąsiadów





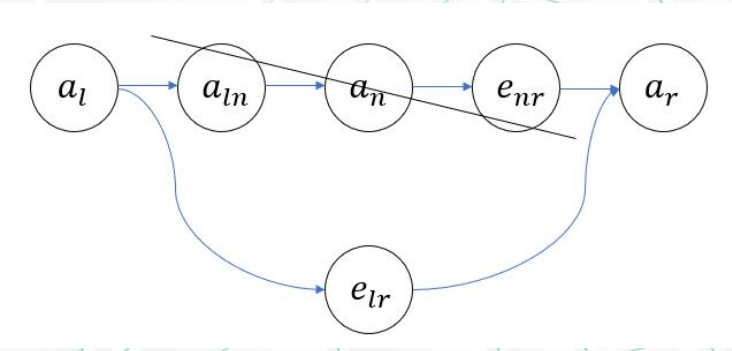
Znajdowanie zdarzeń okręgowych

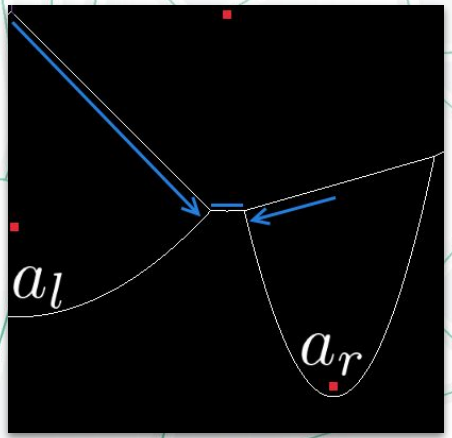
- Jeśli łuk nie ma prawego lub lewego sąsiedniego łuku to nie będzie zdarzenia okręgowego
- Jeśli łuki się nigdy nie przetną, również nie będzie zdarzeń okręgowych.

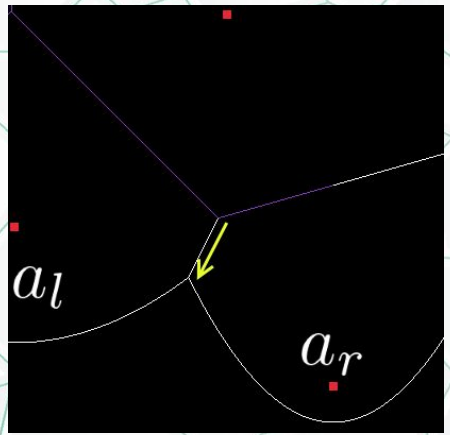
Zapisujemy zarówno środek okręgu, jak i jego najniższy punkt (klucz sortowania w kolejce zdarzeń)

```
def isCirleEvent(self, arc_node: Node):
    leftEdge = arc_node.arc.edgeGoingLeft
    rightEdge = arc_node.arc.edgeGoingRight
    if leftEdge == None or rightEdge == None:
        return None, None
    cirle_center = getIntersect(leftEdge.start, leftEdge.direction,
                                rightEdge.start, rightEdge.direction)
    if cirle_center == None:
        return None, None
    circle_event_point = Point(
        cirle_center[0], cirle_center[1]
        - distance(cirle_center, self.rightNbour(arc_node).arc.focus))
    return cirle_center, circle_event_point
```

Algorytm Fortune’a - zdarzenia okręgowe







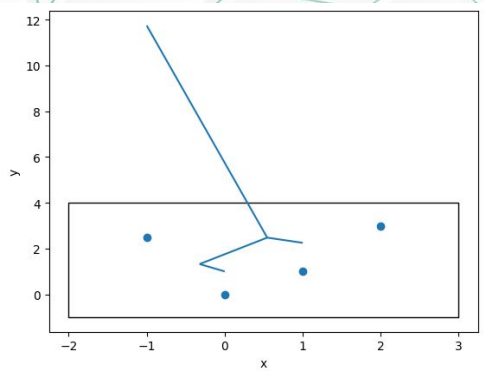

```python
def __handleCircleEvent(self, event):
    T = self.T
    Q = self.Q
    leaf: Node = event.node
    lnode: Node = T.leftNbour(leaf)
    rnode: Node = T.rightNbour(leaf)
    al: Arc = lnode.arc
    ar: Arc = rnode.arc

    point: Point = event.cirle_center_point

    closedLeft = leaf.arc.edgeGoingLeft
    closedRight = leaf.arc.edgeGoingRight

    closedLeft.end = point
    closedRight.end = point
    self.__addToDiagram(closedLeft.start, closedLeft.end)
    self.__addToDiagram(closedRight.start, closedRight.end)

    al.setRightEdge(ar, point)

    left_circle_event, right_circle_event = T.handleSquize(leaf)
    if left_circle_event != None:
        Q.add(left_circle_event)
    if right_circle_event != None:
        Q.add(right_circle_event)
```

https://jacquesheunis.com/post/fortunes-algorithm/

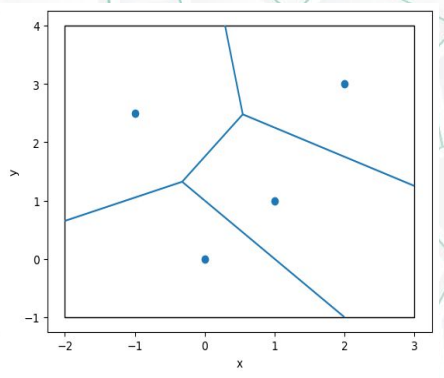
Zakończenie algorytmu

Na koniec algorytmu, musimy zdjąć z T krawędzie które zostały i odpowiednio skrócić je aby nie były zmieściły się w pudełku

Przed dokończeniem i skróceniem krawędzi



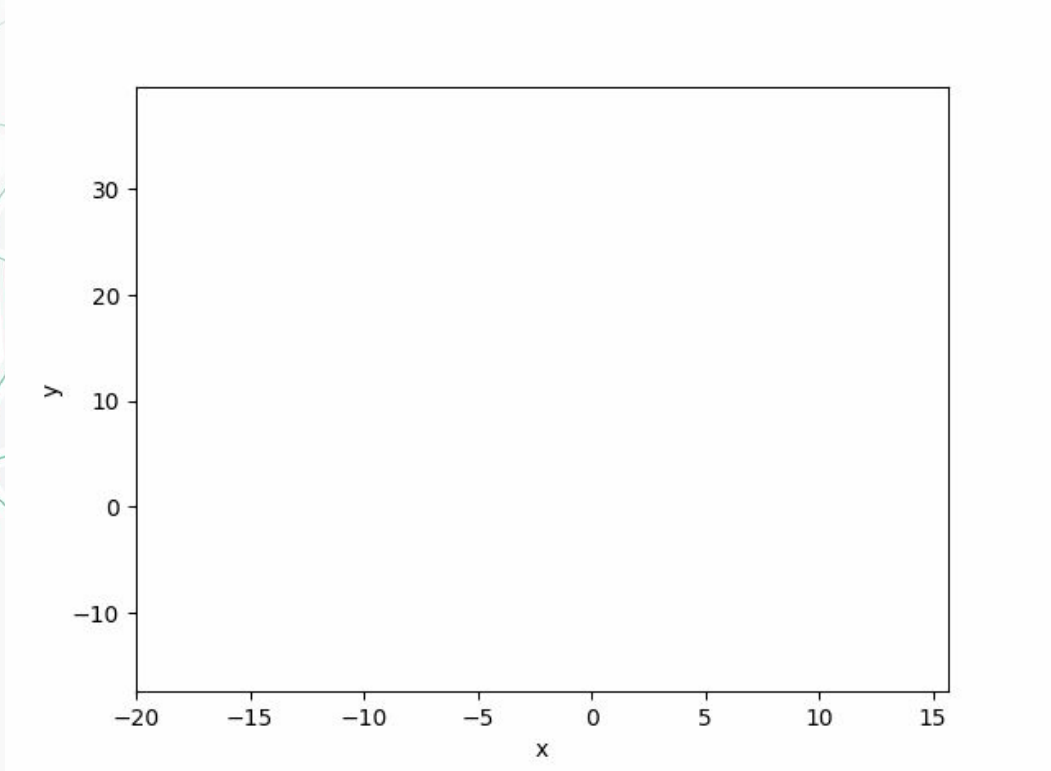
Po dokończeniu i skróceniu krawędzi

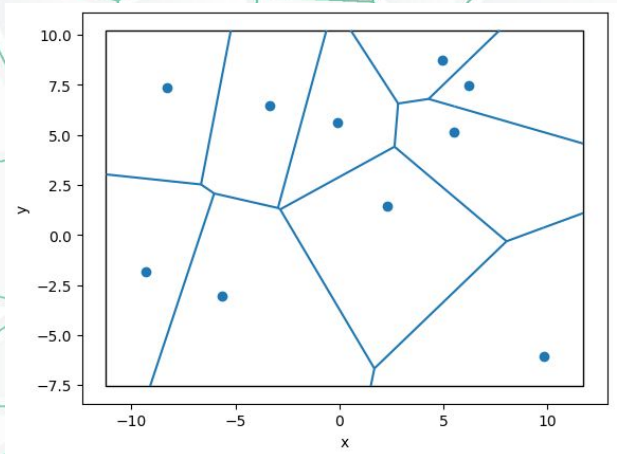


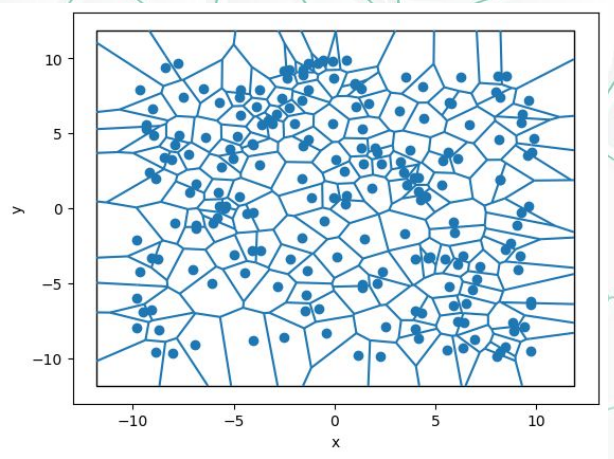

```python
def __finishEdges(self):
    r = self.T.head.next
    while r is not None:
        if r.arc.edgeGoingLeft is not None and r.arc.edgeGoingLeft.end is None:
            e = self.__finishEdgeWithBox(r.arc.edgeGoingLeft)
            if e:
                self.D.append(e)
        if r.arc.edgeGoingRight is not None and r.arc.edgeGoingRight.end is None:
            e = self.__finishEdgeWithBox(r.arc.edgeGoingRight)
            if e:
                self.D.append(e)
        if r.next is None:
            break
        r = self.T.rightNbour(r)

def __finishEdgeWithBox(self, edge: Edge):
    line_start = edge.start
    direction = edge.direction
    for start, end in self.edges_segments:
        intersect = lineSegmentIntersect(start, end, line_start, direction)
        if intersect:
            edge.end = intersect
            return (edge.start, edge.end)
```

Algorytm Fortune’a w akcji

Złożoność optymalna: O(nlogn)





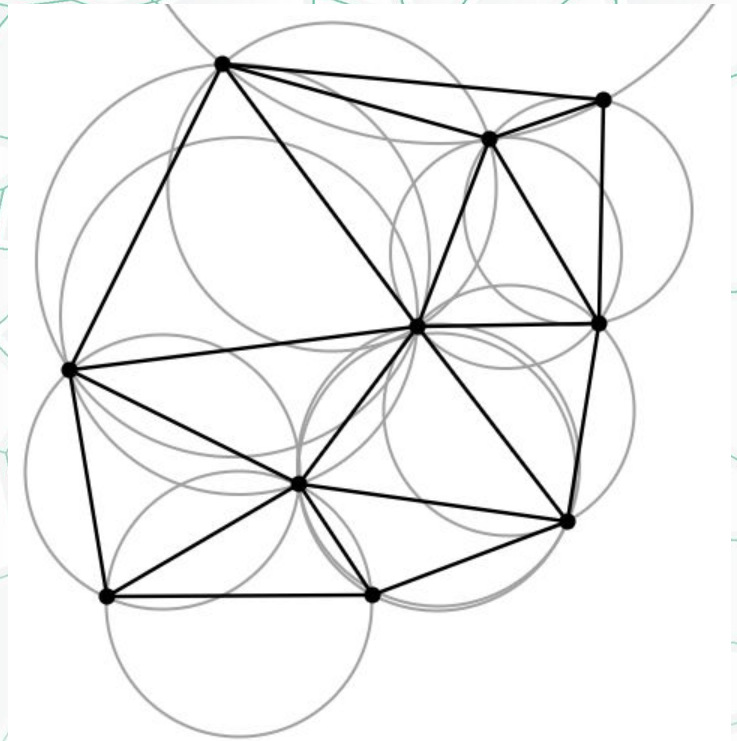


Triangulacja Delaunay’a

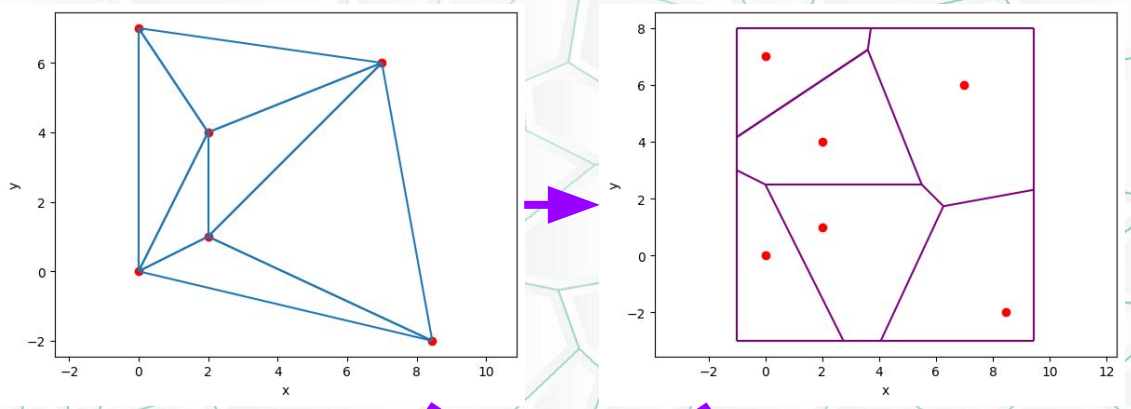
Kryterium kuli opisanej dla przestrzeni R^2

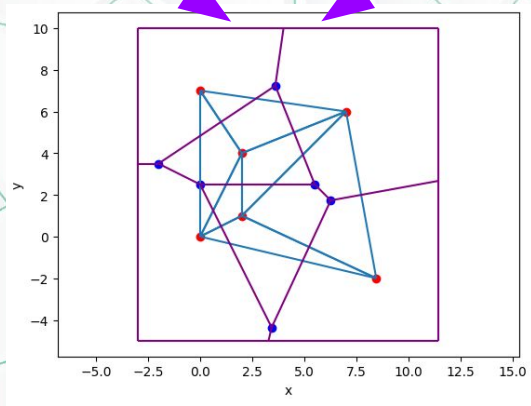
W przestrzeni R^2 triangulacja Delaunaya jest to jedyna taka triangulacja powłoki wypukłej chmury punktów P na płaszczyźnie, że żaden punkt ze zbioru P nie znajduje się wewnątrz okręgu opisanego na trójkącie należącym do triangulacji.

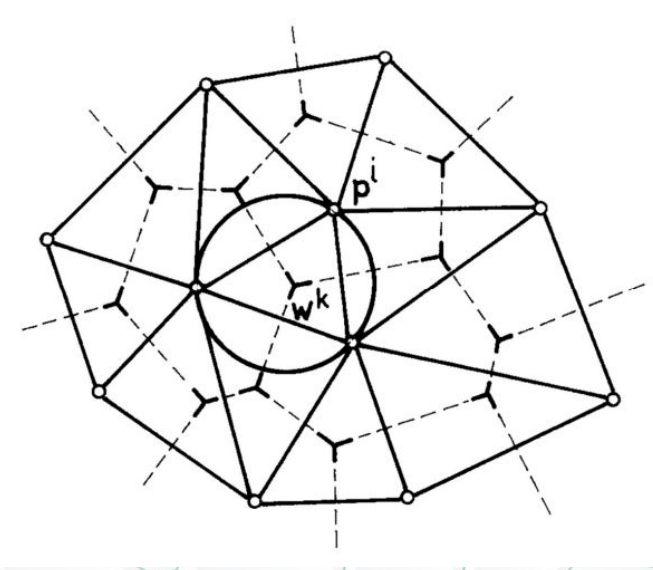
Warunkiem jednoznaczności triangulacji jest brak współokręgowości dowolnych czterech punktów zbioru wejściowego P.



Konstrukcja dualna triangulacji Delaunay'a





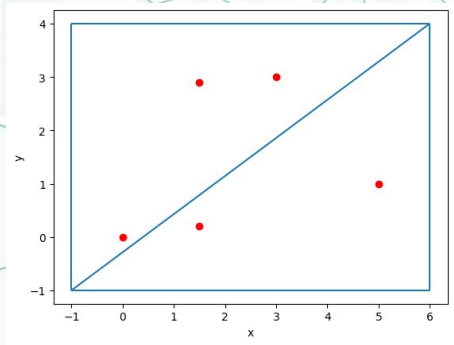


https://upel.agh.edu.pl/pluginfile.php/298460/mod_resource/content/1/wyklad_vor_del.pdf

Algorytm inkrementacyjny Bowyera-Watsona

Pierwszym krokiem algorytmu jest utworzenie początkowej (sztucznej) triangulacji T_0, której otoczka wypukła zawiera wszystkie punkty zbioru wejściowego.

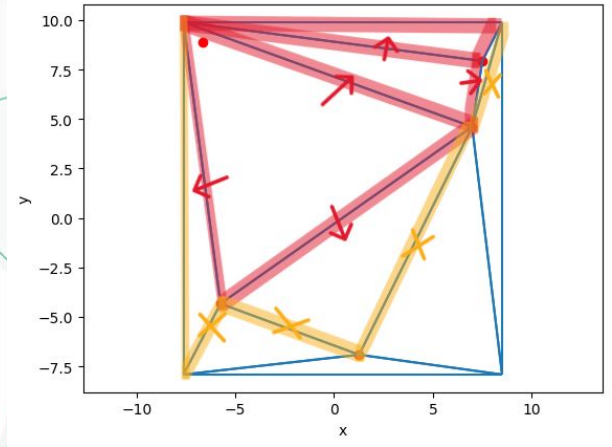
W naszym algorytmie ze względu na prostotę i bezproblemowość w implementacji na początkową triangulację składają się dwa trójkąty prostokątne tworzące prostokąt ograniczający.




```python
class Triangulation:
    def __init__(self, P):
        self.edges = {}
        self.triangles = set()
        self.data_frame, self.map_vertexes, self.central_point, self.central_triangle = self.get_triangulation_start(P)

    def get_triangulation_start(self, P):
        low_left = Point(float('inf'), float('inf'))
        up_right = Point(float('-inf'), float('-inf'))
        for p in P:
            if p.x < low_left.x: low_left.x = p.x
            if p.x > up_right.x: up_right.x = p.x
            if p.y < low_left.y: low_left.y = p.y
            if p.y > up_right.y: up_right.y = p.y
        data_frame = [Point(low_left.x, low_left.y), Point(up_right.x, up_right.y)]
        central_point = Point((low_left.x + up_right.x)/2, (low_left.y + up_right.y)/2 + 1e-2)
        low_left.x += -10**4
        low_left.y += -10**4
        up_right.x += 10**4
        up_right.y += 10**4
        map_vertexes = [low_left, Point(up_right.x, low_left.y), up_right, Point(low_left.x, up_right.y)]
        self.triangles.add(Triangle(low_left, Point(up_right.x, low_left.y), up_right))
        self.triangles.add(Triangle(low_left, up_right, Point(low_left.x, up_right.y)))
        central_triangle = None
        for tri in self.triangles:
            if central_point in tri: central_triangle = tri
            self.edges[(tri.p1, tri.p2)] = tri.p3
            self.edges[(tri.p2, tri.p3)] = tri.p1
            self.edges[(tri.p3, tri.p1)] = tri.p2
        return data_frame, map_vertexes, central_point, central_triangle
```

Algorytm inkrementacyjny Bowyera-Watsona cz.2

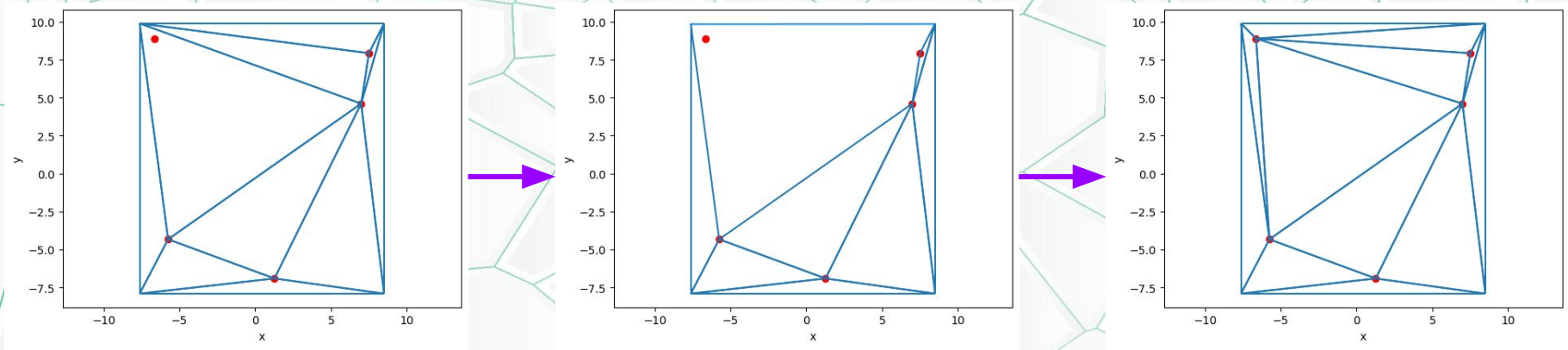
W celu stworzenia triangulacji T_i dodajemy i-ty punkt ze zbioru wejściowego P. Naszym celem jest uzyskanie prawidłowej triangulacji z punktów stanowiących wierzchołki T_0 oraz z dotychczas dodanych punktów w ilości i. Poszukujemy podobszar do retriangulacji poprzez sąsiedztwo topologiczne. Obszar ten to wszystkie trójkąty triangulacji, których koła opisane zawierają dodany punkt.




```python
def delauney(Points):
    P = get_points(Points)
    T = Triangulation(P)

    for p in P:
        containing_tri = find_containing(T, p)
        tri_to_remove = [] # trójkąty do usunięcia
        tri_visited = [] # odwiedzone trójkąty
        stack = [containing_tri]

        while len(stack) > 0:
            curr_tri = stack.pop()
            tri_visited.append(curr_tri)

            if curr_tri.in_circle(p):
                tri_to_remove.append(curr_tri)
                tri_adjacent = T.find_all_adjacent_tri(curr_tri) # trójkąty sąsiadujące z curr_tri

                for triangle in tri_adjacent:
                    if triangle not in tri_visited and triangle not in stack:
                        stack.append(triangle)

        T.adjust_triangulation(tri_to_remove, p)

    T.remove_map_vertexes()
    return T
```

Algorytm inkrementacyjny Bowyera-Watsona cz.3

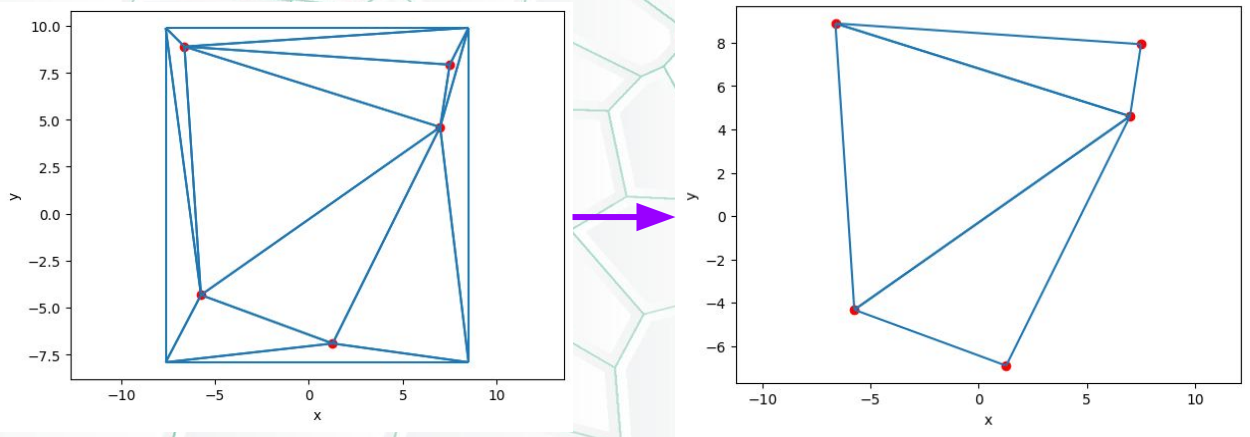
Kolejnym krokiem jest retriangulacja wyznaczonego obszaru (wnęki). Osiągniemy to poprzez usunięcie wszystkich przekątnych wnęki, a następnie dodanie krawędzi łączących wierzchołki wnęki z nowo-dodanym punktem. Warto zaznaczyć, że dla złożoności algorytmu istotnym jest złożoność operacji znajdowania trójkąta triangulacji zawierającego dodawany w iteracji punkt.



Algorytm inkrementacyjny Bowyera-Watsona cz.3

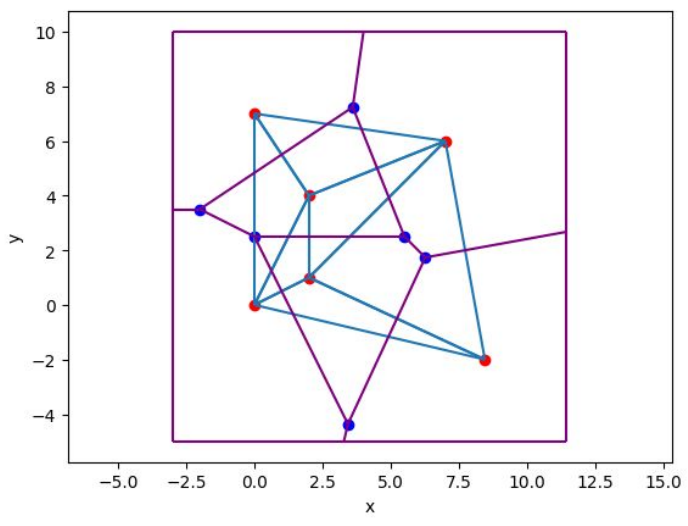
Ostatnim etapem algorytmu Bowyera-Watsona jest usunięcie z triangulacji T_n wszystkich trójkątów, które posiadają przynajmniej jeden wierzchołek należący do zbioru wierzchołków triangulacji T_0.

```python
class Triangulation:
    def __init__(self, P):
        self.edges = {}
        self.triangles = set()
        (self.data_frame, self.map_vertexes, self.central_point,
        self.central_triangle) = self.get_triangulation_start(P)

# pozostałe metody klasy...

    def remove_map_vertexes(self):
        triangles = list(self.triangles)
        for tri in triangles:
            if tri.p1 in self.map_vertexes: self.remove(tri)
            elif tri.p2 in self.map_vertexes: self.remove(tri)
            elif tri.p3 in self.map_vertexes: self.remove(tri)
```



Jak z triangulacji Delaunay’a otrzymać diagram Voronoi?

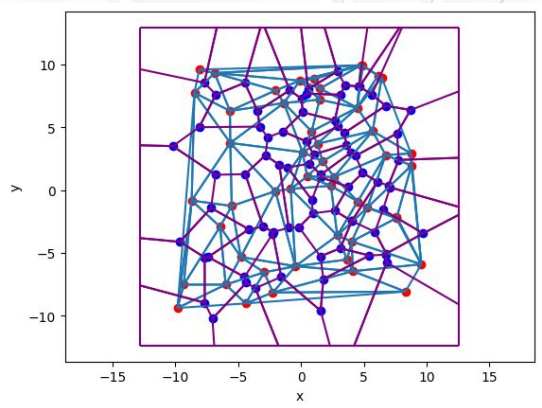
Aby skorzystać z dualności konstrukcji triangulacji Delaunay’a należy dla każdego trójkąta zbadać położenie środka jego okręgu opisanego i środka okręgu opisanego na trójkącie dzielącym z nim krawędź (o ile istnieje).

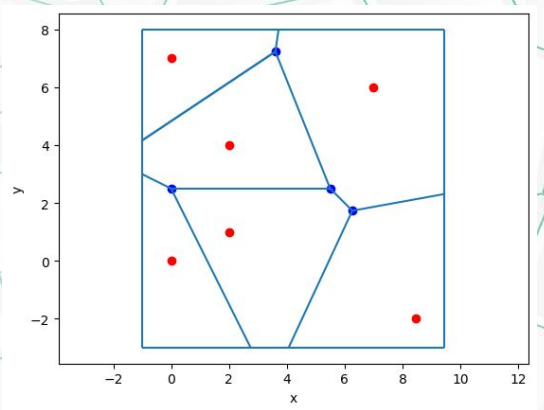


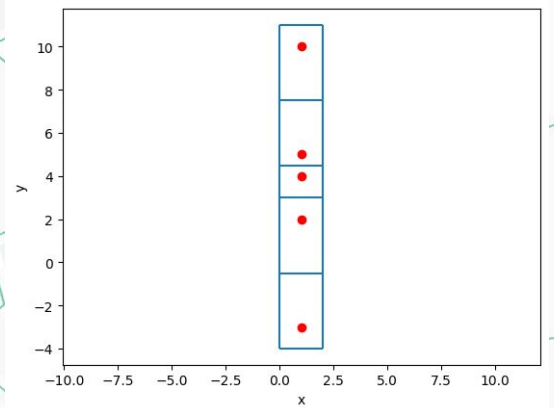

```
def add_edgeV(T, V, triangle_edge, triangle):
    triangles_o = get_triangles_o(triangle, triangle_edge, T)
    if len(triangles_o) == 2:
        found_adjacent(T, V, triangles_o)
    elif len(triangles_o) == 1:
        not_found_adjacent(T, V, triangle_edge, triangle)


def voronoi(T, Points):
    V = Voronoi()
    for triangle in T.triangles:
        add_edgeV(T, V, (triangle.p1, triangle.p2), triangle)
        add_edgeV(T, V, (triangle.p2, triangle.p3), triangle)
        add_edgeV(T, V, (triangle.p3, triangle.p1), triangle)

    if len(T.triangles) == 0:
        no_triangles(T, V, Points)

    map_edges = [(T.map_vertexes[0], T.map_vertexes[1]), (T.map_vertexes[1], T.map_vertexes[2]),
                 (T.map_vertexes[2], T.map_vertexes[3]), (T.map_vertexes[3], T.map_vertexes[0])]
    for edge in map_edges:
        V.edges.add(edge)

    return V.edges
```

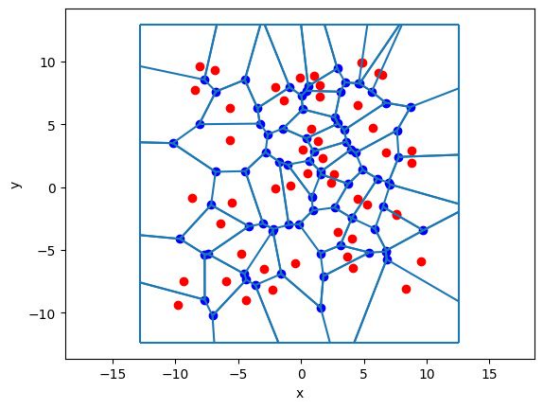
Wyniki algorytmu

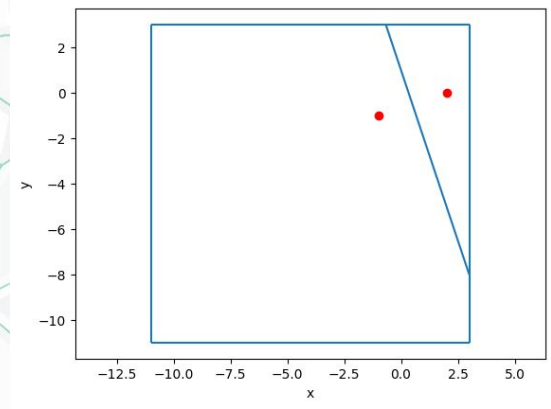
- Punkty zbioru wejściowego
- Środki okręgów opisanych na trójkątach triangulacji

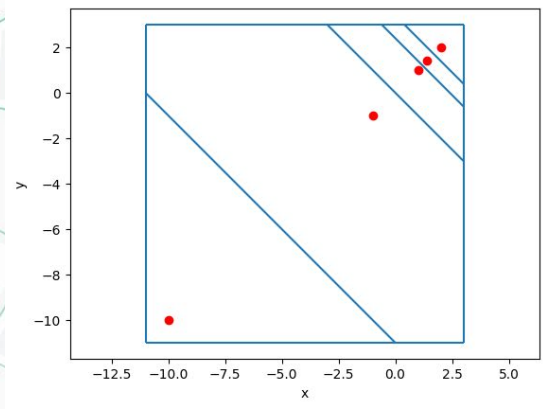






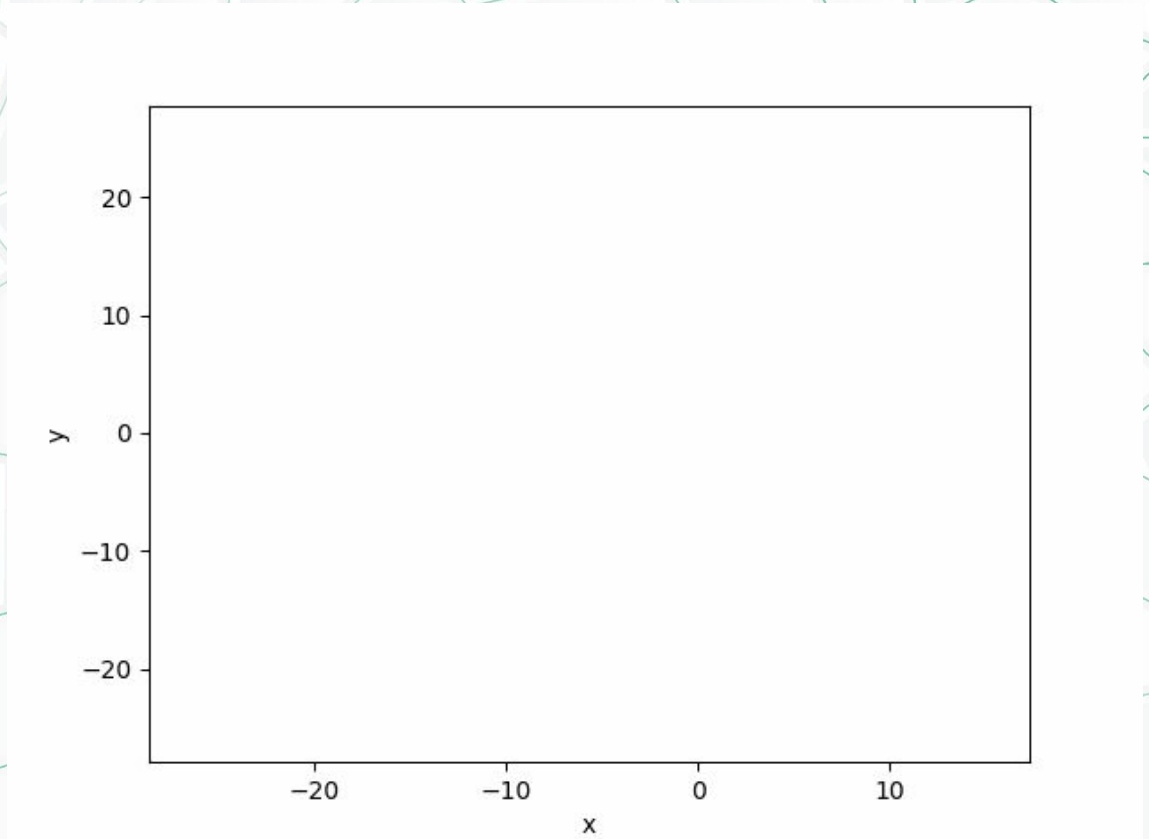






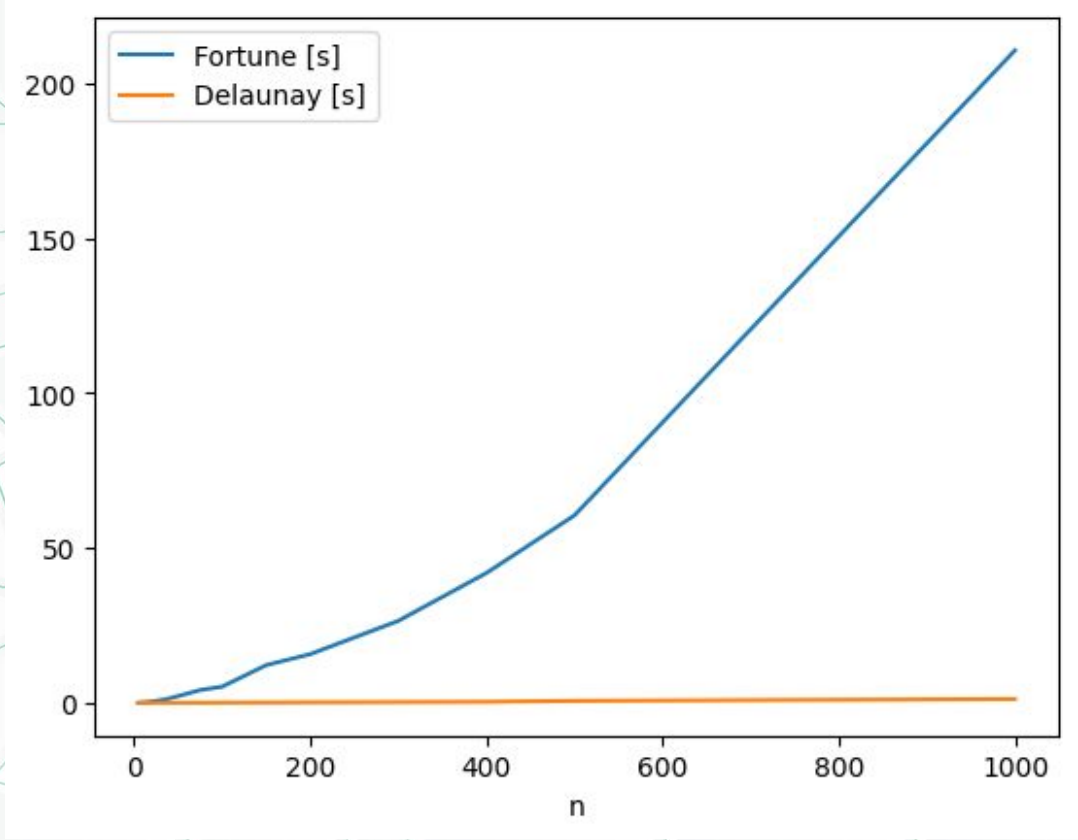
Wizualizacja algorytmu

Złożoność optymalna: O(nlogn)



Porównanie czasów wykonania obu algorytmów

n	Fortune [s]	Delaunay [s]
5	0.0369065	0.002403021
10	0.1323271	0.005480766
20	0.4149733	0.011526108
35	1.0874844	0.026529312
50	2.1889706	0.035391092
75	4.1368194	0.052121162
100	5.2197244	0.086582422
150	12.1797957	0.118786812
200	15.7140830	0.17547369
300	26.4622085	0.270816565
400	41.9027805	0.365414143
500	60.4842865	0.645820618
1000	210.5087557	1.244166613



Bibliografia

[1] Monika Wiech, *“Triangulacja Delaunaya w geometrii obliczeniowej”*, Uniwersytet Jagielloński, Kraków, obrona 2020-09-22 (*“Delaunay triangulation in computational geometry”*)

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf, *“Geometria Obliczeniowa - Algorytmy i zastosowanie”* (polskie tłumaczenie *“Computational Geometry - Algorithms and Applications”* - z angielskiego przełożył Mirosław Kowaluk)

[3] https://jacquesheunis.com/post/fortunes-algorithm/

[4] https://pvigier.github.io/2018/11/18/fortune-algorithm-details.html

[5] Dr inż. Barbara Głut, Wykład przedmiotu *Algorytmy Geometryczne* na Akademii Górniczo-Hutniczej w Krakowie im. Stanisława Staszica w Krakowie (2023/2024)