



AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE
AGH UNIVERSITY OF KRAKOW

Przeszukiwanie obszarów ortogonalnych

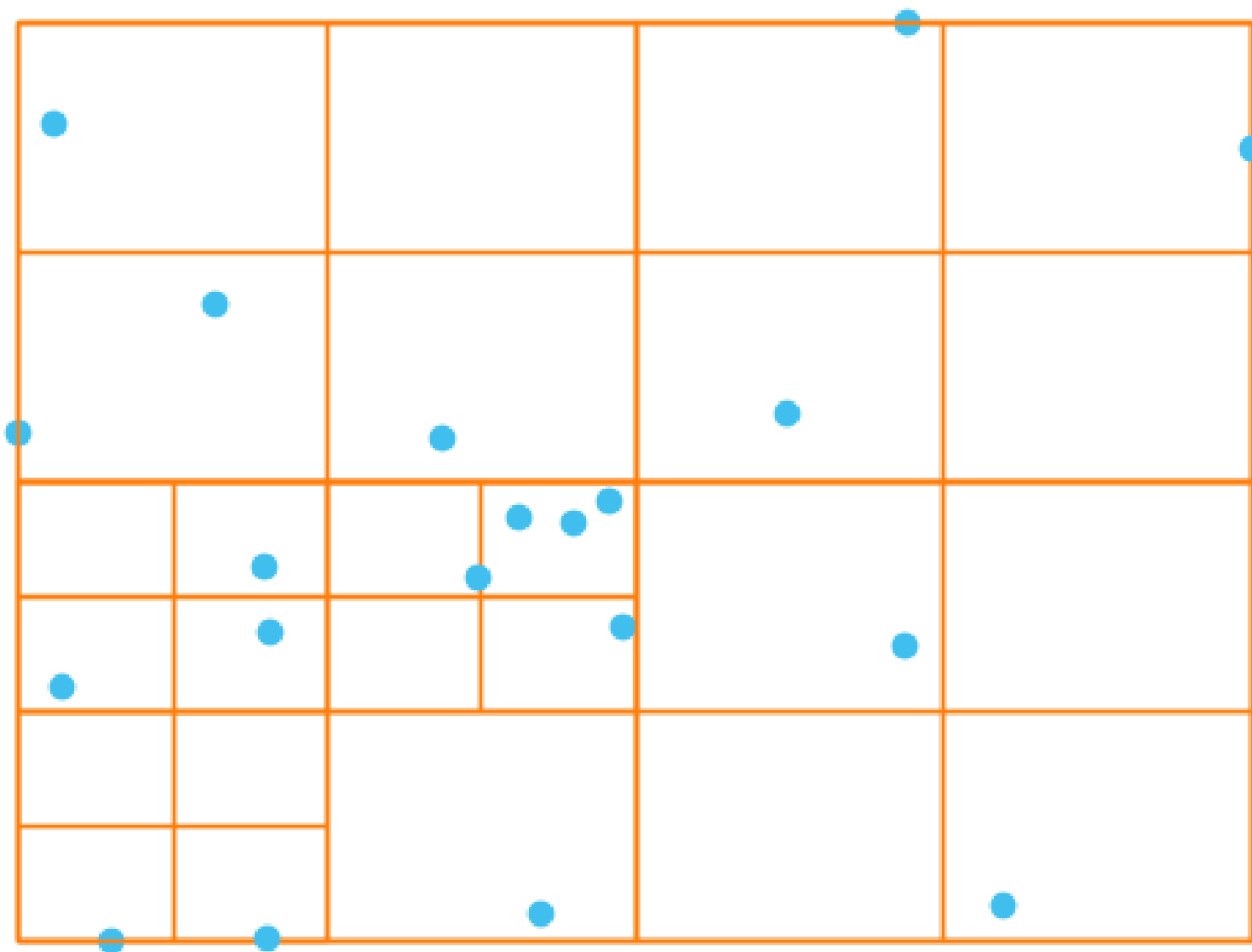
Autorzy: Dariusz Marecik, Piotr Sękowski

Temat i cel projektu

Dane: Zbiór punktów P na płaszczyźnie.

Zapytanie: Dla zadanych współrzędnych punktów lewego dolnego rogu (x_1, y_1) i oraz prawego górnego rogu (x_2, y_2) , należy znaleźć punkty q ze zbioru P , takie że spełniają one warunki: $x_1 \leq q_x \leq x_2$ oraz $y_1 \leq q_y \leq y_2$.

QuadTree

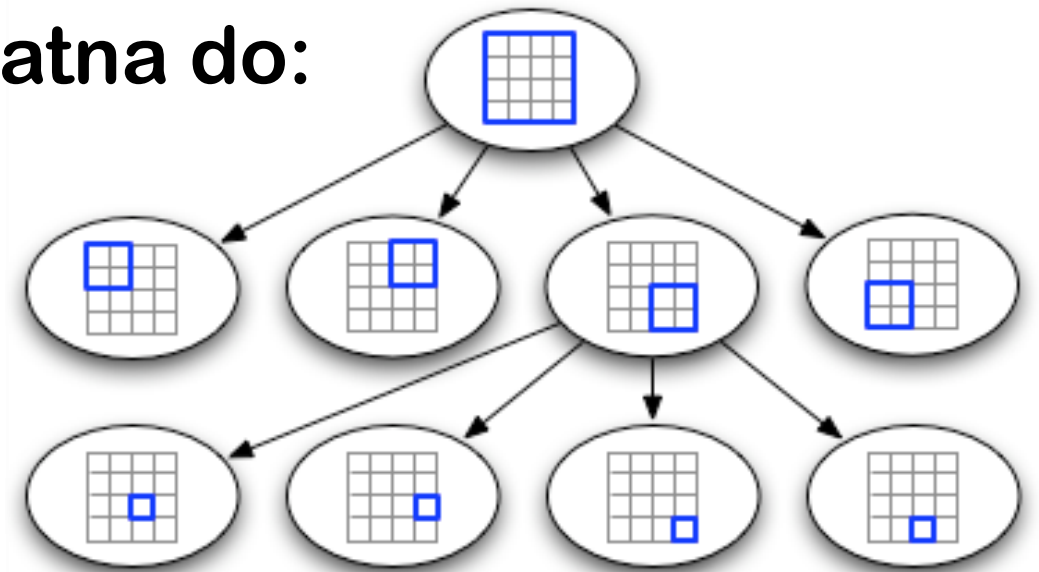


Opis

Quadtree jest to strukturą danych będąca drzewem, które służy do podziału przestrzeni dwuwymiarowej na mniejsze części, dzieląc ją na równe ćwiartki, a następnie dzieląc je na kolejne ćwiartki itd.

Jest to struktura szczególnie przydatna do:

- przetwarzania obrazu
- generowania siatki
- skutecznego wykrywania kolizji



Budowa Quadtree

Kod:

```
def __init__(self, rectangle, max_points = 1, depth = 0):  
    self.nw = None # lewy górny czworokąt  
    self.ne = None # prawy górny czworokąt  
    self.sw = None # lewy dolny czworokąt  
    self.se = None # prawy dolny czworokąt  
    self.rectangle = rectangle # czworokąt ograniczający dany węzeł  
    self.max_points = max_points # maksymalna liczba punktów w węźle  
    self.points = [] # punkty w węźle  
    self.depth = depth # głębokość węzła  
    self.divided = False # flaga mówiąca o tym czy węzeł ma dzieci
```

Wstawianie punktu

Kod:

```
def insert(self, point):
    if point.amount_of_dimensions != 2:
        raise ValueError("Niepoprawny wymiar punktów! \nQuadtree obsługuje tylko punkty dwuewymiarowe!")
    if not self.rectangle.is_point_in_rectangle(point):
        return False
    if len(self.points) < self.max_points:
        self.points.append(point)
        return True

    if not self.divided:
        self.divide()

    return self.se.insert(point) or self.ne.insert(point) or self.sw.insert(point) or self.nw.insert(point)

def divide(self):
    x_1, y_1 = self.rectangle.lower_left.cords
    x_2, y_2 = self.rectangle.upper_right.cords
    c_x = (x_1 + x_2) / 2
    c_y = (y_1 + y_2) / 2
    center = Point((c_x, c_y))
    bounds = (Point((x_1, y_1)), Point((x_2, y_2)), Point((x_2, y_1)), Point((x_1, y_2)))

    self.sw = QuadTree(Rectangle(bounds[0].lower_left(center), bounds[0].upper_right(center)), self.max_points, self.depth + 1)
    self.ne = QuadTree(Rectangle(bounds[1].lower_left(center), bounds[1].upper_right(center)), self.max_points, self.depth + 1)
    self.se = QuadTree(Rectangle(bounds[2].lower_left(center), bounds[2].upper_right(center)), self.max_points, self.depth + 1)
    self.nw = QuadTree(Rectangle(bounds[3].lower_left(center), bounds[3].upper_right(center)), self.max_points, self.depth + 1)
    self.divided = True
```

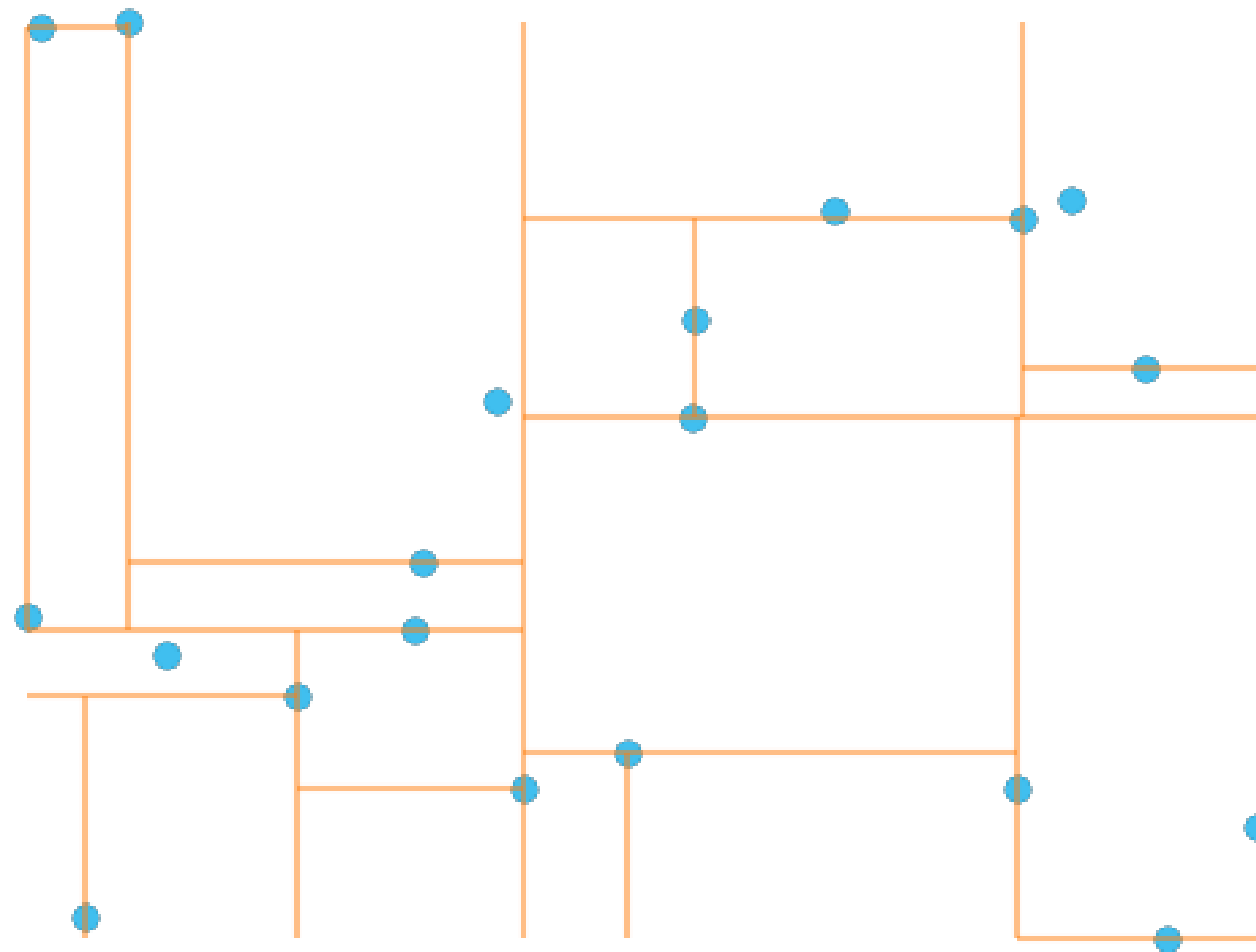
Przeszukiwanie struktury

Kod:

```
def search(self, boundary, found_points):  
    if not self.rectangle.is_intersect(boundary):  
        return []  
  
    for point in self.points:  
        if boundary.is_point_in_rectangle(point):  
            found_points.append(point.cords)  
  
    if self.divided:  
        self.se.search(boundary, found_points)  
        self.ne.search(boundary, found_points)  
        self.sw.search(boundary, found_points)  
        self.nw.search(boundary, found_points)  
    return found_points
```

1. Sprawdzenie czy prostokąt węzła ma jakąś część wspólną z przeszukiwanym obszarem jeśli nie to zwracamy pustą tablice
2. Sprawdzenie czy punkty zapisane w węźle zawierają się w szukanym prostokącie, jeśli tak to dodanie jego współrzędnych do listy found_points
3. Przeszukanie węzłów dzieci
4. Zwrócenie znalezionych do tej pory punktów

KDTree

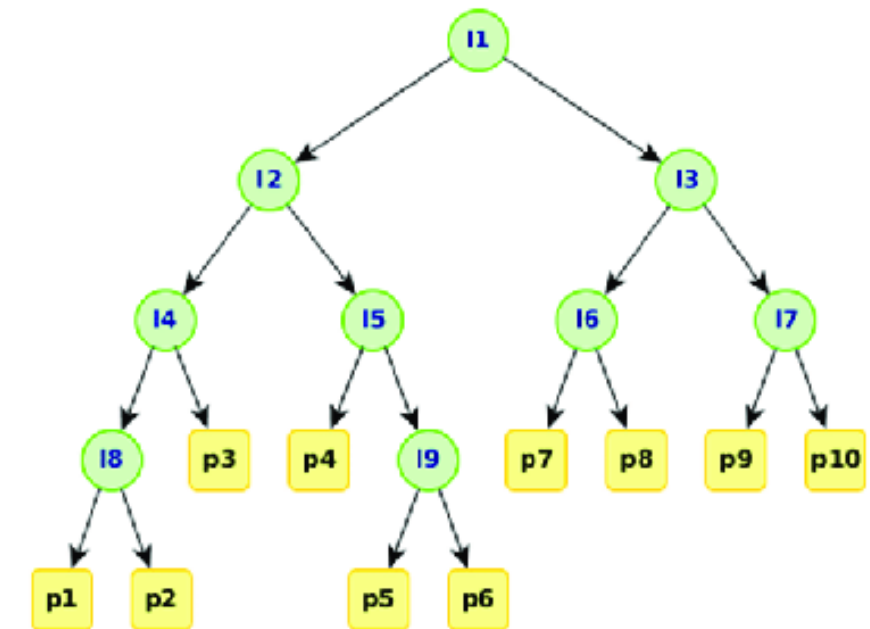


Opis

Struktura ta organizuje punkty w formie binarnego drzewa, gdzie każdy węzeł reprezentuje podział przestrzeni wzdłuż jednej z osi. Przy każdej iteracji podział odbywa się na przemian względem kolejnych wymiarów.

Najbardziej popularnym zastosowaniem drzewa jest algorytm kNN (k nearest neighbours) - podstawowy algorytm klasyfikacji uczenia maszynowego.

Także bardzo przydatnym jest wydajne filtrowanie danych. Jeśli mamy zbiór obiektów z pewnymi cechami i chcemy wybrać takie, wartości cech których mieszczą się w pewnych przedziałach, to możemy potraktować ich jak wielowymiarowe punkty.



Struktura drzewa

Kod:

```
class KdTree:
    def __init__(self, points, dimensions_amount, begining_axis=0, is_points_in_vertix = True):
        for point in points:
            if len(point)!= dimensions_amount:
                raise ValueError("zbiór punktów nie zgadza się z deklarowaną ilością wymiarów")
        points = [Point(point) for point in points]
        self.begining_axis = begining_axis
        self.root = KdTreeNode(points, dimensions_amount, begining_axis,
                                Rectangle(list_of_Point=points), is_points_in_vertix)
        self.dimensions_amount = dimensions_amount

class KdTreeNode:
    def __init__(self, points, dimensions_amount, depth, rectangle, is_points_in_vertix=True):
        if is_points_in_vertix or len(points)==1:
            self.points = points
        else:
            self.points = []
        self.dimensions_amount = dimensions_amount
        self.depth = depth
        self.dimension_number = self.depth%self.dimensions_amount
        self.left = None
        self.right = None
        self.is_points_in_vertix = is_points_in_vertix
        self.rectangle = rectangle
        if len(points)>1:
            self.build(points)
```

Budowa drzewa

Kod:

```
class KdTreeNode:
    def build(self, points):
        for _ in range(self.dimensions_amount):
            points.sort(key = lambda x: x.cords[self.dimension_number])
            median = math.ceil(len(points)/2)
            median-=1
            median = self.bsearch_right(points, self.dimension_number, points[median].cords[self.dimension_number])
            left_median = self.bsearch_left(points, self.dimension_number, points[median].cords[self.dimension_number])
            median+=1

            if median - left_median > 3*len(points)//4 or median == len(points):
                self.depth +=1
                self.dimension_number = (self.dimension_number+1)%self.dimensions_amount
            else:
                break

        self.axis = points[median-1].cords[self.dimension_number]
        left_rec, right_rec = self._split_region(self.rectangle, self.dimension_number, self.axis)
        self.left = KdTreeNode(points[0:median], self.dimensions_amount, self.depth+1, left_rec, self.is_points_in_vertix )
        self.right = KdTreeNode(points[median:], self.dimensions_amount, self.depth+1, right_rec, self.is_points_in_vertix )
```

Przeszukiwanie struktury

Kod:

```
class KdTree:
    def check_contains(self, point):
        if not isinstance(point, Point):
            if len(point) != self.dimensions_amount:
                raise ValueError("Podano nieprawidłowy punkt do znalezienia")
            point = Point(point)
        return self.root.check_contains(point)

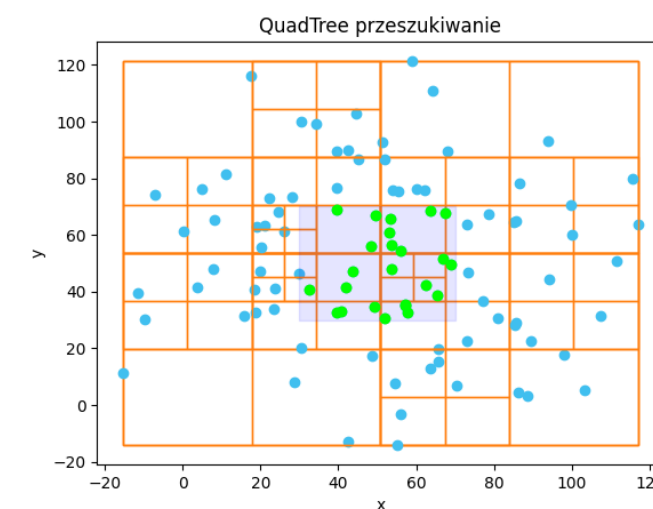
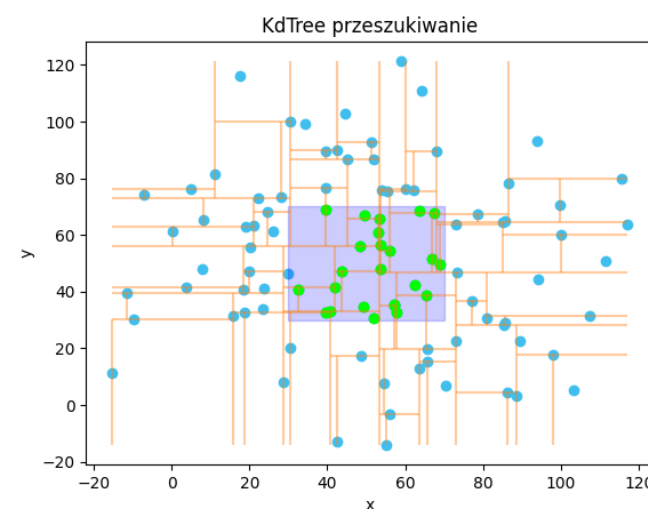
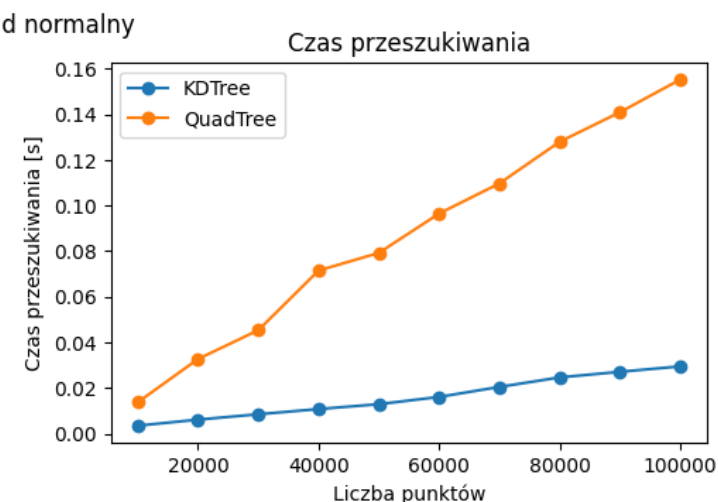
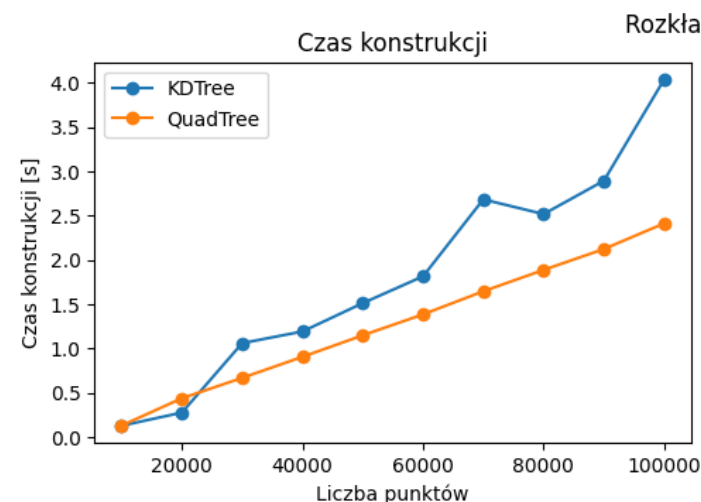
class KdTreeNode:
    def find_points_in_region(self, region):
        if self.is_leaf():
            is_in = region.is_point_in_rectangle(self.points[0])
            return self.points if is_in else []

        if region.is_contained(self.rectangle):
            return self.get_points()

        if region.is_intersect(self.rectangle):
            return self.left.find_points_in_region(region) + self.right.find_points_in_region(region)
        return []
```

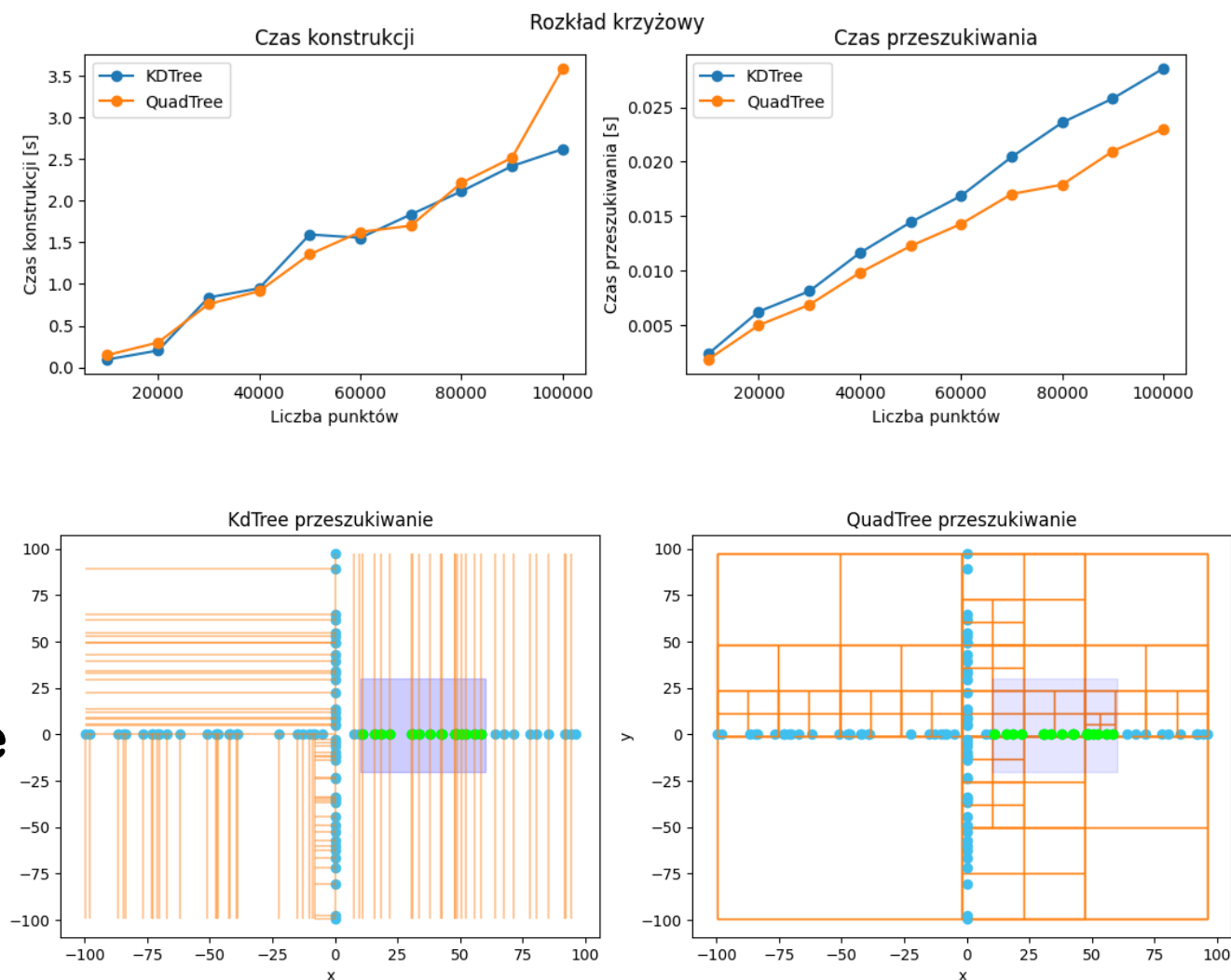
Rozkład normalny

Jest to rozkład statystycznie najczęściej występujący w naturze, jeżeli chcemy, więc przyjrzeć się naszemu problemowi dla najbardziej powszechnych przypadków, powinniśmy wziąć pod lupę właśnie tak wygenerowany zbiór punktów. Jak wychodzi z wykresu KdTree poradziło sobie lepiej niż Quadtree.



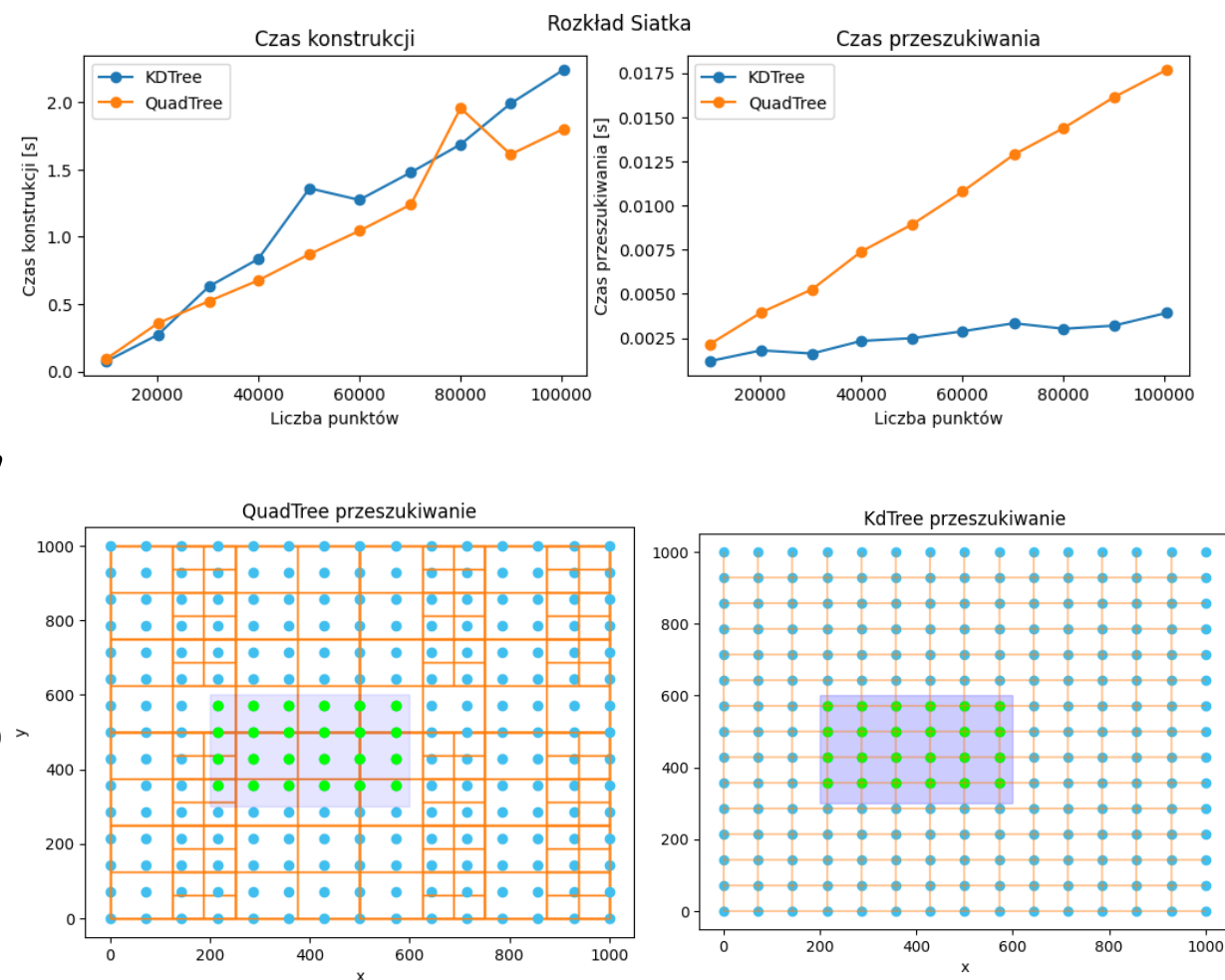
Rozkład krzyżowy

Punkty w tym zbiorze są rozmieszczone w sposób zbliżony do równomiernego na osiach układu współrzędnych, co wpływa na charakter podziału przestrzeni przez struktury danych. W przypadku takiego rozmieszczenia, duża liczba punktów współliniowych może prowadzić do nieoptymalnego dzielenia obszarów, w wyniku czego struktura danych może nieefektywnie zarządzać przestrzenią. Jak widać na wykresie QuadTree poradziło sobie trochę lepiej.



Zbiór siatka

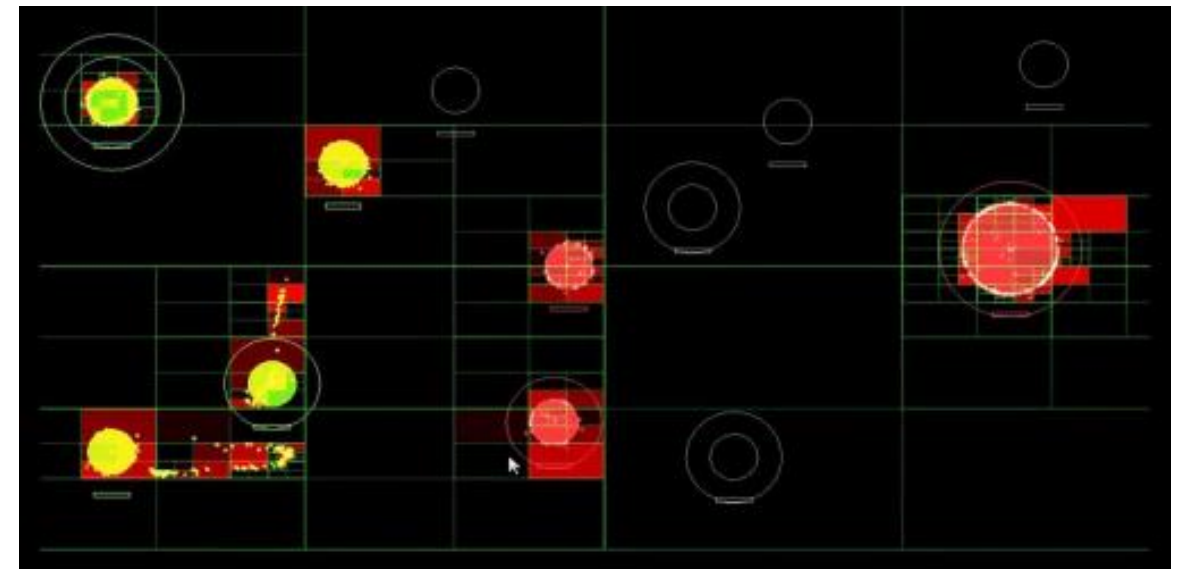
Zbiór składa się z punktów ułożonych w równych odległościach od siebie. Takie rozmieszczenie przypomina rozkład jednostajny, lecz dokłada problem współliniowych punktów względem osi układu współrzędnych. Jak widać po wykresie lepiej sprawdza do tego zbioru drzewo kd



Ciekawostki

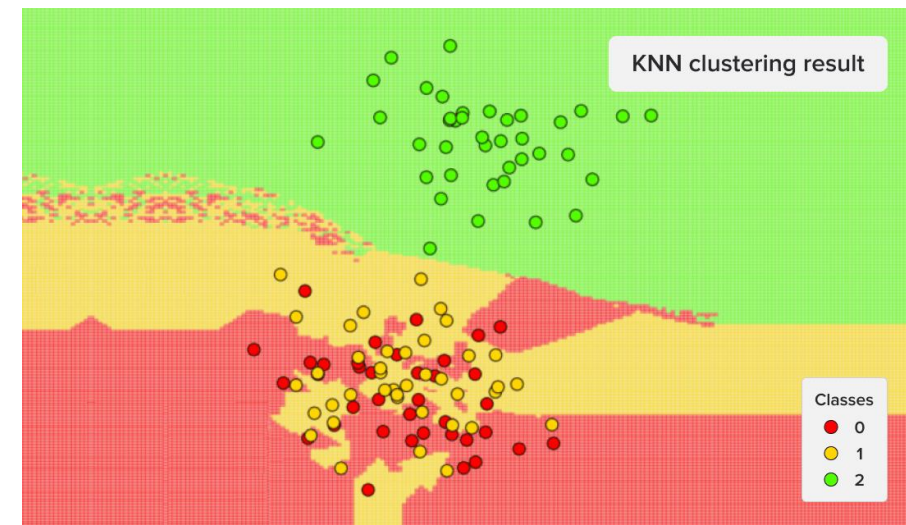
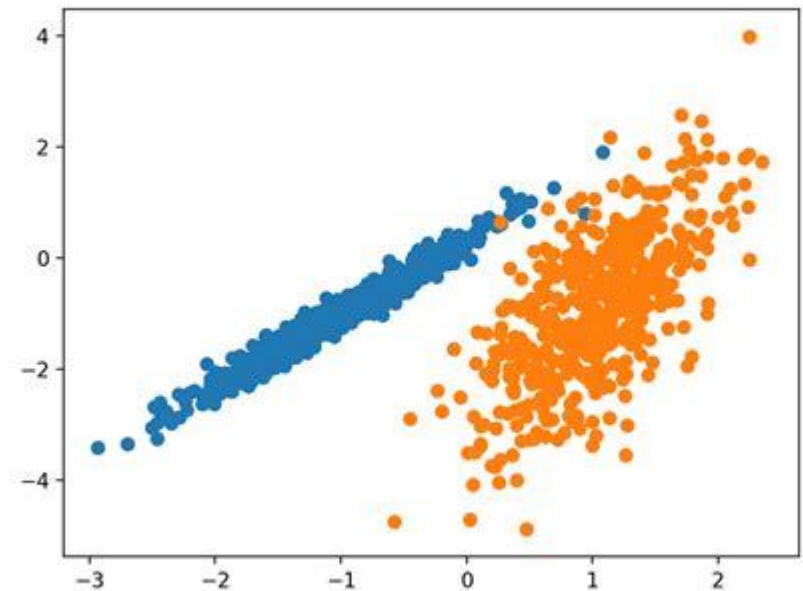
Quadtree znajduje szerokie zastosowanie przede wszystkim w kompresji obrazu, im głębsze drzewo, tym jakość będzie lepsza, lecz w tym tkwi zaleta quadtree, nie w każdym miejscu musimy kopać tak samo głęboko.

Strukturę tą wykorzystuje się także w grach komputerowych do wykrywania kolizji. Dzięki budowie quadtree możemy uwzględnić daną kolizję nie zmieniając całości drzewa, a tylko jego fragment.



Jednym z najczęstszych zastosowań drzew kd jest wyszukiwanie najbliższego sąsiada, gdzie są używane do szybkiego znalezienia najbliższego punktu do zadanego punktu. Dzięki temu są one użytecznym elementem wielu podejść w uczeniu maszynowym, takich jak metoda k-najbliższych sąsiadów (KNN) zerocka gama algorytmów klastrowania opartych na odległości.

Innym zastosowaniem drzew kd jest wyszukiwanie w zakresie (ang. range search), gdzie służą do znajdowania wszystkich punktów w określonej odległości od punktu zapytania. Jest to przydatne w analizie danych przestrzennych, systemach informacji geograficznej (GIS) i innych zastosowaniach wymagających identyfikacji punktów w określonym zakresie.



Podsumowanie

Jak widzimy, każde drzewo ma swoje zastosowania, które są spowodowane pewnymi własnościami:

QuadTree jest zazwyczaj bardzo szybki dla zbalansowanych zbiorów i nie ma pewnych wyjątkowych sytuacji przy których jest bardzo niewydajny.

KDTree ma największą zaletę w wielowymiarowości i dodatkowych operacjach (np. szukanie najbliższego sąsiada), które zapewniają popularność tej struktury. Przedziałowe przeszukiwanie w 2D przestrzeni jest bardzo wydajne na obu strukturach, ale wymaga dodatkowego zastanowienia się nad postacią zbioru punktów jak pokazaliśmy ciut wcześniej, żeby uwzględnić własności drzew, i odpowiednio wybrać najbardziej odpowiednie.