



Wyszukiwanie geometryczne KD-tree Quadtree

Przeszukiwanie obszarów ortogonalnych

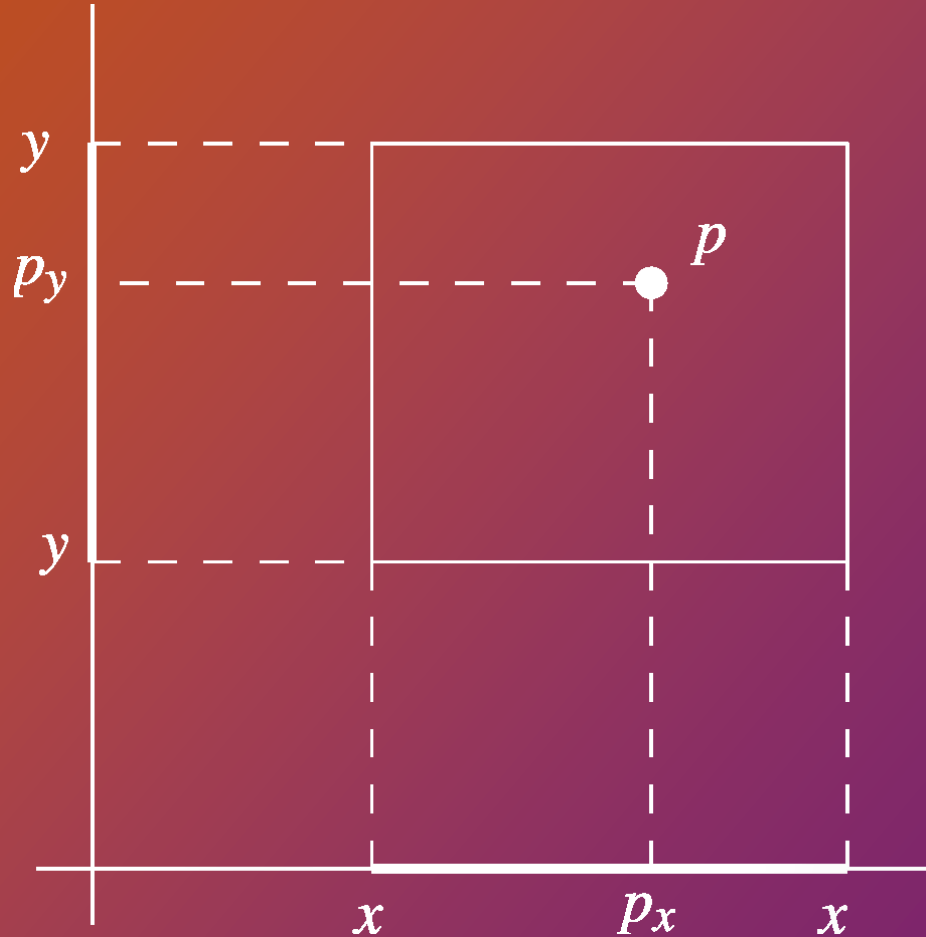
Aleksander Jóźwik, Szymon Hołysz



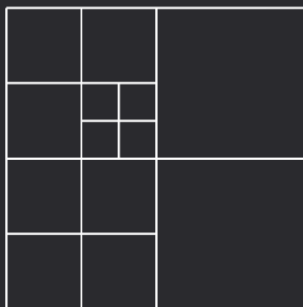
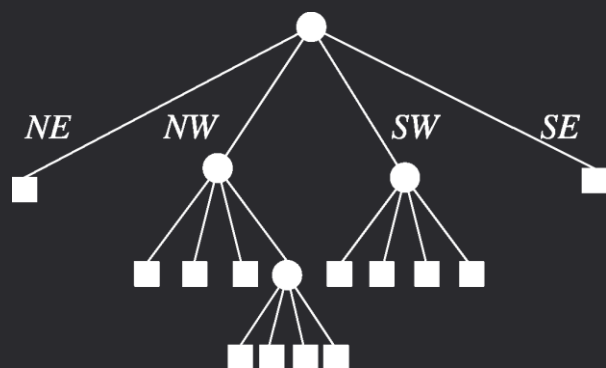
Opis problemu

Dany jest zbiór punktów P na płaszczyźnie. Dla zadanych x_1, x_2, y_1, y_2 znaleźć punkty p ze zbioru P takie, że $x_1 \leq p_x \leq x_2, y_1 \leq p_y \leq y_2$.

QuadTree oraz KD-drzewo pozwalają szybko odpowiadać na takie zapytania.



Drzewa ćwiartek dla zbiorów punktów



Jedną ze struktur danych, która może przechowywać punkty na płaszczyźnie jest drzewo ćwiartek. Jest ono ukorzenionym drzewem, w którym każdy wewnętrzny węzeł v ma czworo dzieci. Odpowiadają one podziałowi v na cztery ćwiartki. W ten sposób liście drzewa tworzą podział korzenia. Przykładowy podział jest przedstawiony na rysunku. Dzieci korzenia są etykietowane *NE*, *NW*, *SW* i *SE*, aby wskazywać, której ćwiartce odpowiadają – *NE* to ćwiartka północno – wschodnia itd.

Konstrukcja drzewa ćwiartek

Drzewo ćwiartek konstruowane jest rekurencyjnie. Konstrukcja dla każdego węzła przebiega w następujący sposób:

- Jeżeli węzeł zawiera co najwyżej jeden punkt, to kończymy konstrukcję dla tego poddrzewa
- W przeciwnym wypadku dzielimy bieżący węzeł i punkty na ćwiartki, a następnie wywołujemy powyższą metodę dla wszystkich ćwiartek węzła.



Przeszukiwanie drzewa ćwiartek

Aby znaleźć punkty należące do zadanego obszaru ortogonalnego, używamy analogicznej metody rekurencyjnej. Znalezienie punktów w danym poddrzewie przebiega następująco:

- Sprawdzamy, czy obszar bieżącego węzła przecina się z zadanym obszarem, jeżeli nie, kończymy przeszukiwanie
- Jeżeli węzeł zawiera punkty, to sprawdzamy, czy należą do obszaru
- W przeciwnym wypadku wywołujemy powyższą metodę dla dzieci węzła

Implementacja drzewa ćwiartek – używane klasy

Klasa Rectangle

Reprezentuje przedział ortogonalny. Zawiera współrzędne (x, y) minimalnego i maksymalnego punktu

przedziału.

Atrybuty:

- self.min_x - współrzędna x minimalnego punktu
- self.min_y - współrzędna y minimalnego punktu
- self.max_x - współrzędna x maksymalnego punktu
- self.max_y - współrzędna y maksymalnego punktu

Klasa Node

Reprezentuje węzeł w strukturze drzewa ćwiartek. Zawiera odniesienia do rodzica i dzieci, drzewa,

przedziału, któremu odpowiada i zbioru punktów, które przechowuje.

2.4.3.1. Atrybuty

- self.tree - wskazanie na obiekt klasy Quad()
- self.quarter - wartość Quarter(Enum)
- self.parent - wskazanie na rodzica, obiekt klasy Node()
- self.square - wskazanie na przedział, któremu odpowiada; obiekt klasy Rectangle()
- self.ne, self.nw, self.se, self.sw - wskazania na dzieci obiekty klasy Node()
- self.points = wskazanie na zbiór set() punktów, które przechowuje; jeżeli węzeł nie jest liściem, to

self.points = None

Klasa Quad

Reprezentuje drzewo ćwiartek. Zawiera odniesienie do korzenia drzewa, zbiór punktów należących do drzewa i listę liści.

2.4.4.1. Atrybuty

- self.points - zbiór punktów, na podstawie których skonstruowane zostało drzewo
- self.leaves - lista liści
- self.root wskazanie na obiekt Node() będący korzeniem

Klasa Rectangle

– podział na ćwiartki

```
def rectangle_partition(self):  
    med_y = self.med_y()  
    med_x = self.med_x()  
    s_ne = Rectangle(med_x, med_y, self.max_x, self.max_y)  
    s_nw = Rectangle(self.min_x, med_y, med_x, self.max_y)  
    s_sw = Rectangle(self.min_x, self.min_y, med_x, med_y)  
    s_se = Rectangle(med_x, self.min_y, self.max_x, med_y)  
  
    return s_ne, s_nw, s_sw, s_se
```

Klasa Node – rekurencyjna budowa drzewa

```
def construct_subtree(self, points, forced = False):
    if len(points) <= BUCKET_SIZE and not forced:
        self.points = points
        self.tree.leaves.append(self)
    else:
        x = self.square.med_x()
        y = self.square.med_y()
        p_ne, p_nw, p_sw, p_se = set_partition(points, x, y)
        s_ne, s_nw, s_sw, s_se = self.square.rectangle_partition()

        self.ne = Node(self.tree, Quarter.NE, s_ne, self)
        self.nw = Node(self.tree, Quarter.NW, s_nw, self)
        self.sw = Node(self.tree, Quarter.SW, s_sw, self)
        self.se = Node(self.tree, Quarter.SE, s_se, self)

        self.ne.construct_subtree(p_ne)
        self.nw.construct_subtree(p_nw)
        self.sw.construct_subtree(p_sw)
        self.se.construct_subtree(p_se)
```




Klasa Node – rekurencyjne przeszukiwanie drzewa

```
def query_range_subtree(self, range_rect):  
    result = set()  
    if self.square.intersects(range_rect):  
        if self.points is not None:  
            for point in self.points:  
                if range_rect.contains(point):  
                    result.add(point)  
    if self.ne is not None:  
        r_ne = self.ne.query_range_subtree(range_rect)  
        if r_ne is not None: result.update(r_ne)  
    if self.nw is not None:  
        r_nw = self.nw.query_range_subtree(range_rect)  
        if r_nw is not None: result.update(r_nw)  
    if self.sw is not None:  
        r_sw = self.sw.query_range_subtree(range_rect)  
        if r_sw is not None: result.update(r_sw)  
    if self.se is not None:  
        r_se = self.se.query_range_subtree(range_rect)  
        if r_se is not None: result.update(r_se)  
    return result
```

KD-drzewa dla zbiorów punktów

Oryginalnie nazwa
ta oznaczała
drzewo
k-wymiarowe

Czym jest KD-drzewo?

Dla 1-wymiarowego zakresu

Zbiór (1-wymiarowy) punktów jest rozdzielany na dwa podzbiory o względnie równym rozmiarze. Jeden z podzbiorów („lewy”) zawiera punkty mniejsze lub równe z wartością podziału. Drugi podzbiór („prawy”) zawiera natomiast punkty, których wartość jest większa niż punktu podziału.

Wartość podziału jest przechowywana w korzeniu, a dwa podzbiory są przechowywane rekurencyjnie w dwóch poddrzewach.

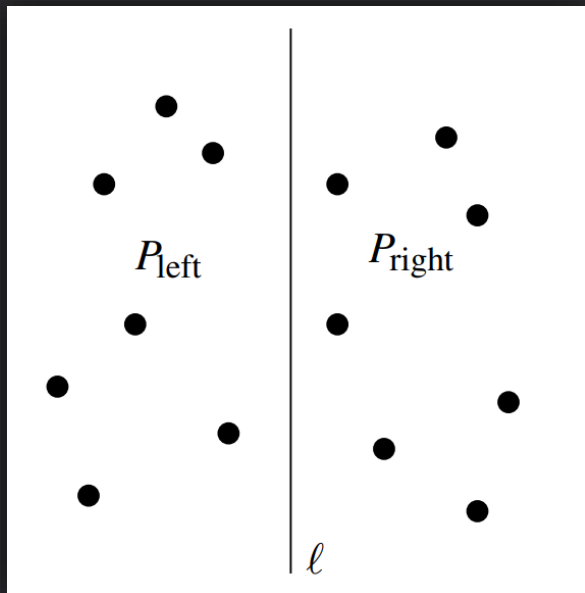
Dla jednego wymiaru sprowadza się do drzewa przedziałowego.

Dla 2-wymiarowego zakresu

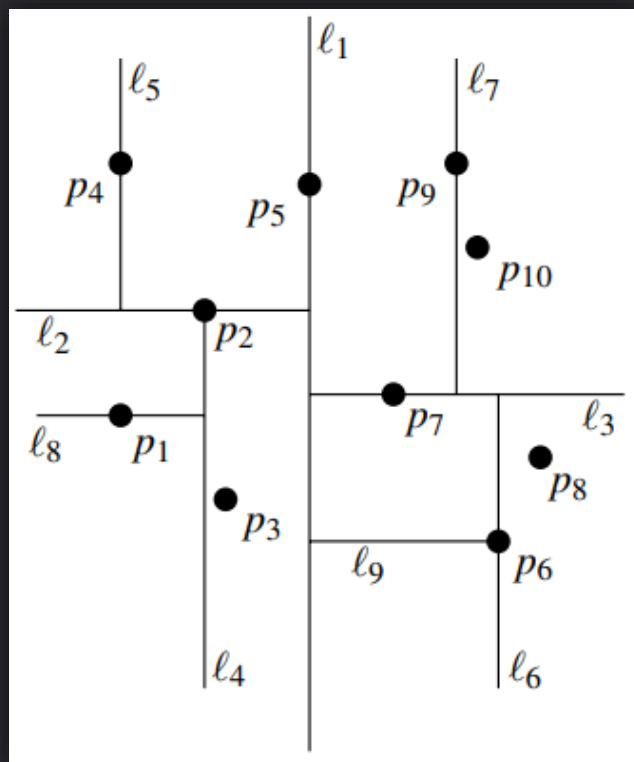
Każdy punkt zawiera współrzędne (x, y) . W efekcie podziału dokonujemy po x , gdy głębokość w drzewie jest parzysta, a po y , kiedy jest nieparzysta. Przestrzeń jest dzielona pionową (lub poziomą) linią na dwa podzbiory o względnie równym rozmiarze.

Linia podziału przechowywana jest w węźle.

Lewe poddrzewo zawiera punkty, których współrzędna (konkretnie ta, po której dokonaliśmy podziału) jest mniejsza lub równa od współrzędnej linii podziału. Prawe poddrzewo zawiera punkty, których współrzędne są większe.

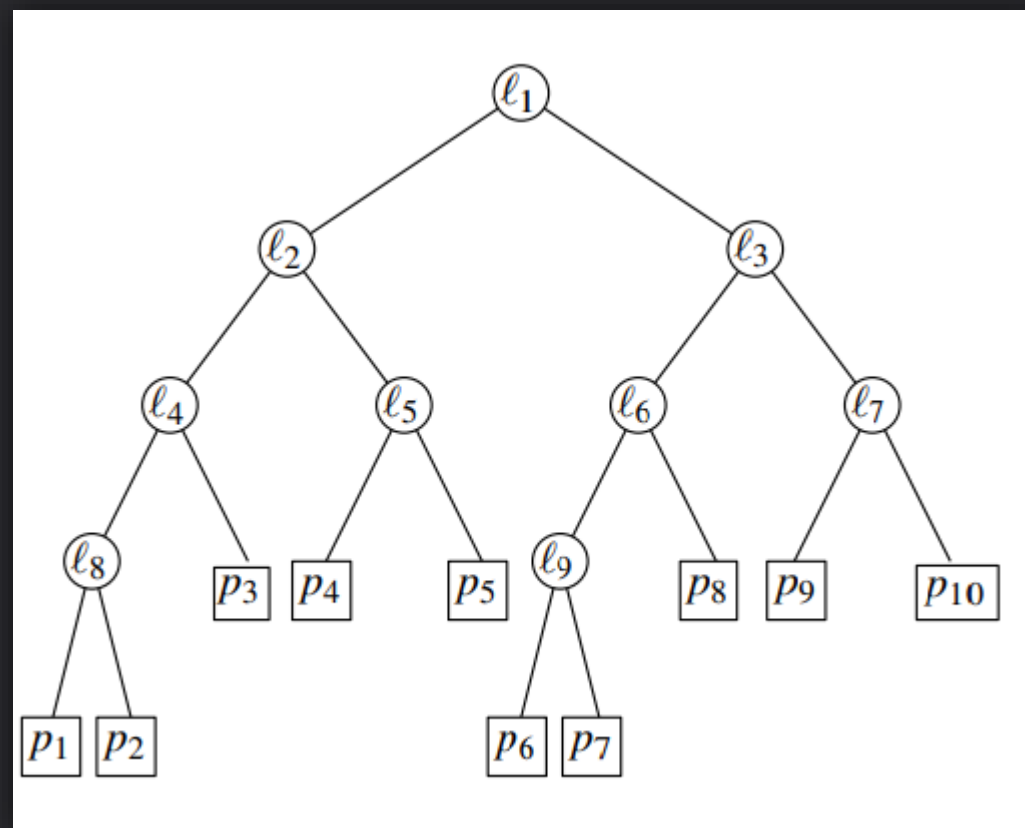


Przykładowy podział obszaru na lewy i prawy podzbiór punktów



Podział płaszczyzny oraz odpowiadające mu drzewo binarne

Złożoność pamięciowa:
 $O(n)$



Konstrukcja KD-drzewa

Algorytm BUILD_TREE(P, depth)

Dane wejściowe: Zbiór punktów P i aktualna głębokość depth.

Wynik: Korzeń KD-drzewa przechowującego P.

1. **if** P zawiera tylko jeden punkt
2. **then return** liść pamiętający ten punkt
3. **else if** depth jest parzyste
4. **then** Podziel P na dwa zbiory pionową prostą ℓ przechodzącą przez medianę współrzędnych x punktów z P. Niech P1 będzie zbiorem punktów na lewo od ℓ lub na ℓ , a P2 zbiorem punktów na prawo od ℓ .
5. **else** Podziel P na dwa zbiory poziomą prostą ℓ przechodzącą przez medianę współrzędnych y punktów z P. Niech P1 będzie zbiorem punktów poniżej ℓ lub na ℓ , a P2 zbiorem punktów powyżej ℓ .
6. $v_lewy \leftarrow \text{BUILD_TREE}(P1, \text{depth} + 1)$
7. $v_prawy \leftarrow \text{BUILD_TREE}(P2, \text{depth} + 1)$
8. Stwórz wierzchołek pamiętający ℓ , zrób v_lewy lewym dzieckiem, a v_prawy prawym dzieckiem v .
9. **return** v

Jak szybko uzyskiwać linię podziału (medianę)?

1. QuickSelect (znacząca stała, pomimo złożoności $O(n)$).
2. Zbiór P zastąpić listą k list posortowanych początkowo po k -tej współrzędnej.


Na każdym poziomie drzewa decyzję o tym, wg którego wymiaru wykonujemy podział można podejmować obliczając:

$oś = \text{głębokość} \% \text{ liczba wymiarów}$

Przy takim oznaczeniu odczyt mediany wygląda następująco:

$mediana = P[oś][(n - 1) // 2][oś]$

Gdzie n jest liczbą elementów podzbioru.



Konstrukcja KD-drzewa

Przy takim podejściu wystarczy później liniowo dzielić punkty w każdym wymiarze na dwie grupy: te, które są mniejsze lub równe medianie, oraz te, które są większe od mediany.

Złożoność zbudowania drzewa ze zbioru punktów wynosi:

$$O(nk \log n)$$

gdzie k jest liczbą wymiarów.

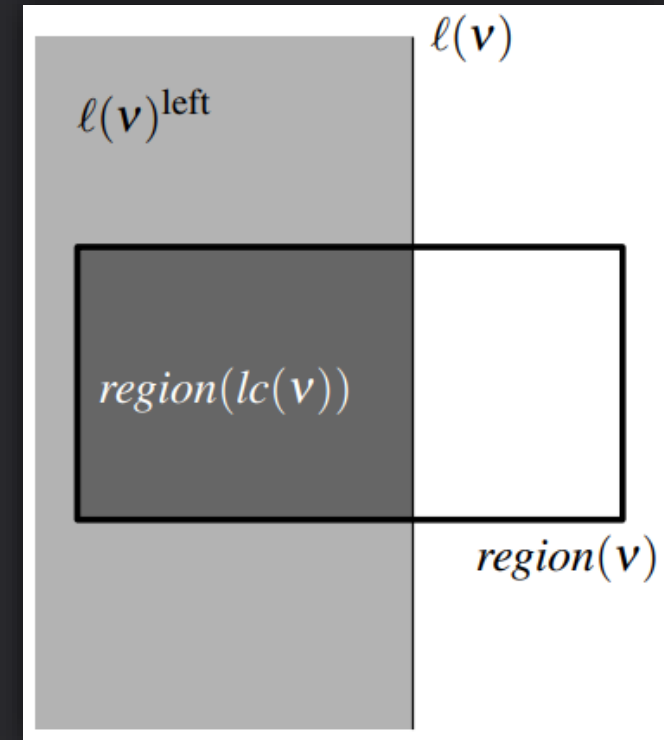
Przeszukiwanie przestrzenne KD-drzewa

Algorytm SEARCHKDTTree(v , R)

Dane wejściowe. Korzeń (poddzwewa) KD-drzewa i obszar R .

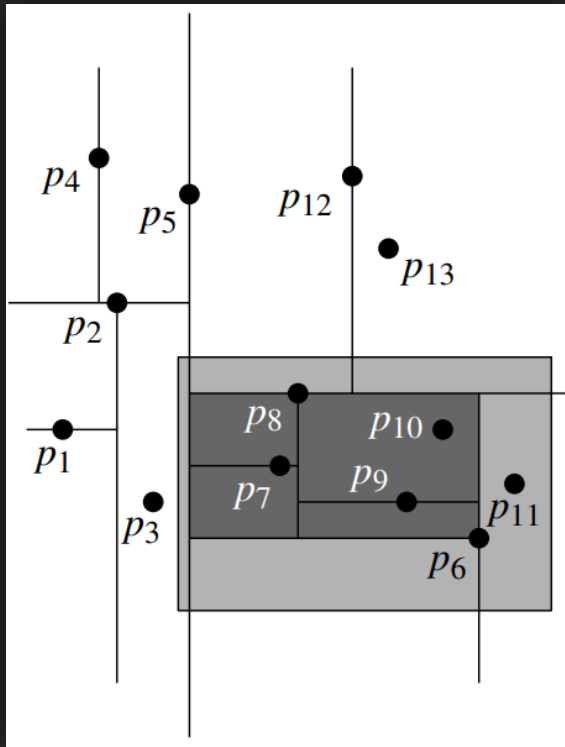
Wynik. Wszystkie punkty w liściach poniżej v , które leżą w danym obszarze.

1. **if** v jest liściem
2. **then** Podaj punkt pamiętany w v , jeśli leży w R
3. **else if** obszar($lc(v)$) jest całkowicie zawarty w R
4. **then** REPORTSUBTREE($lc(v)$)
5. **else if** obszar($lc(v)$) przecina R
6. **then** SEARCHKDTTree($lc(v)$, R)
7. **if** obszar($rc(v)$) jest całkowicie zawarty w R
8. **then** REPORTSUBTREE($rc(v)$)
9. **else if** obszar($rc(v)$) przecina R
10. **then** SEARCHKDTTree($rc(v)$, R)

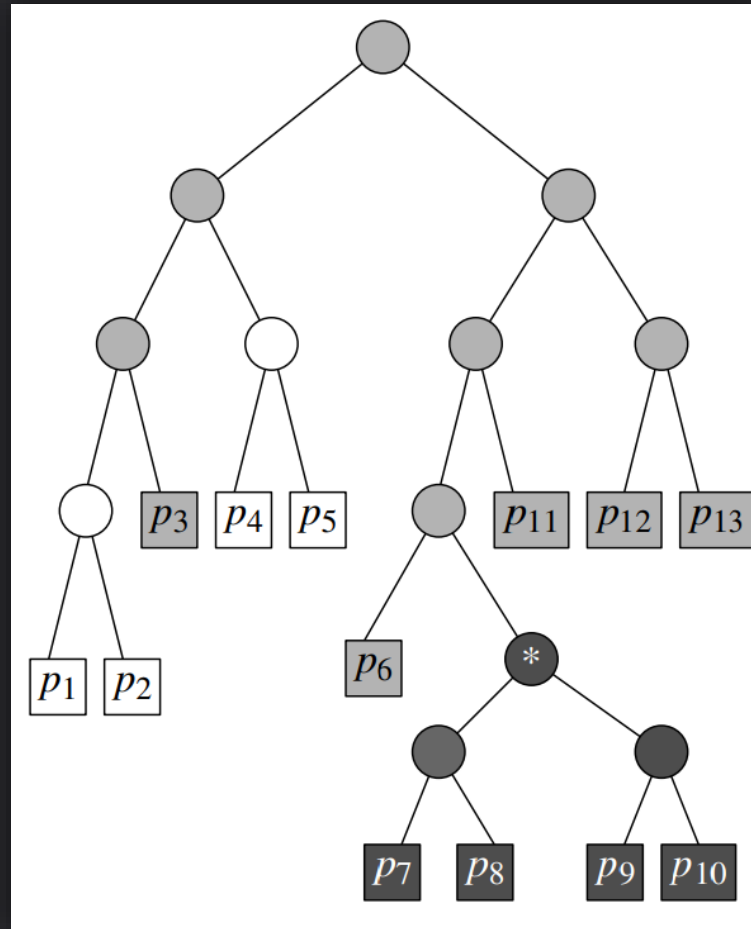


Przykład obszaru($lc(v)$)

Przeszukiwanie przestrzenne KD-drzewa



Przykładowe zapytanie do KD-drzewa



Złożoność zapytania dla 2 wymiarów:

$$O(\sqrt{n} + d)$$

Gdzie d jest liczbą znalezionych punktów.

Złożoność zapytania dla k wymiarów:

$$O(n^{1-\frac{1}{k}} + d)$$

Zastosowania KD-drzewa

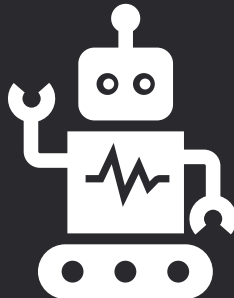
Systemy informacji geograficznej (GIS)

- **Wyszukiwanie obiektów w regionie:** Znajdowanie budynków, dróg, czy punktów zainteresowania (POI) w określonym obszarze mapy.



Robotyka

- **Planowanie ścieżek:** Wyszukiwanie przeszkód lub punktów docelowych w określonym regionie podczas nawigacji robota.
- **Wykrywanie obiektów w otoczeniu:** Określenie, które obiekty znajdują się w obszarze zainteresowania robota (np. w polu chwytaka).



Grafika komputerowa i symulacje 3D

- **Wykrywanie obiektów w obszarze widzenia kamery:** Wyświetlanie tylko tych obiektów, które znajdują się w aktualnym polu widzenia.
- **Symulacje fizyczne:** Znajdowanie cząstek lub obiektów w zadanym regionie przestrzeni, aby określić, które z nich mogą oddziaływać ze sobą.



Bazy danych przestrzennych

- **Zapytania przestrzenne w SQL:** Znajdowanie rekordów związanych z obszarem (np. `SELECT * FROM points WHERE x BETWEEN a AND b AND y BETWEEN c AND d`).
- **Optymalizacja wyszukiwań:** KD-drzewa przyspieszają operacje na danych przestrzennych w porównaniu do liniowego przeszukiwania.



Dziękujemy za uwagę

Bibliografia:

[1] dr inż. Barbara Głut, Algorytmy geometryczne — wykład.

[2] Computational Geometry, 3. wyd. Springer Berlin, Heidelberg.