Wyszukiwanie geometryczne - przeszukiwanie obszarów ortogonalnych

KDTree i QuadTree

Andrzej Karciński i Aleksandra Smela

Cel projektu

Opis problemu

Dane: zbiór punktów P na płaszczyźnie; x_1, x_2, y_1, y_2

Zapytanie: dla zadanych x_1, x_2, y_1, y_2 znaleźć punkty q ze zbioru P takie, że $x_1 \leq q_x \leq x_2,\ y_1 \leq q_y \leq y_2$.

Celem projektu jest zaimplementowanie odpowiednich struktur danych – QuadTree oraz KDTree, które pozwalają szybko odpowiadać na takie zapytania.

Istotne są analiza i porównanie algorytmów!

Algorytm podstawowy

Liniowe przejście po zbiorze P i sprawdzenie warunku $x_1 \leq q_x \leq x_2$, $y_1 \leq q_y \leq y_2$ dla każdego punktu z osobna.

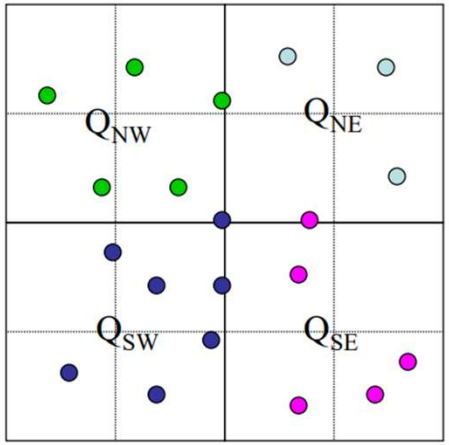
Wady: nie jest to wydajne i może zostać zrobione lepiej.

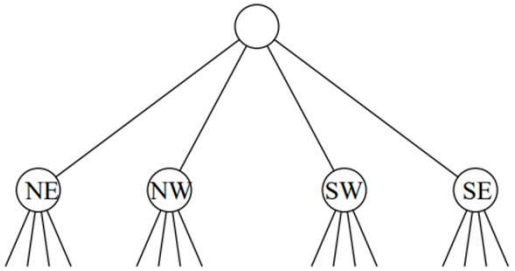
np. poprzez wykorzystanie struktur drzewiastych!

QuadTree

QuadTree – opis struktury

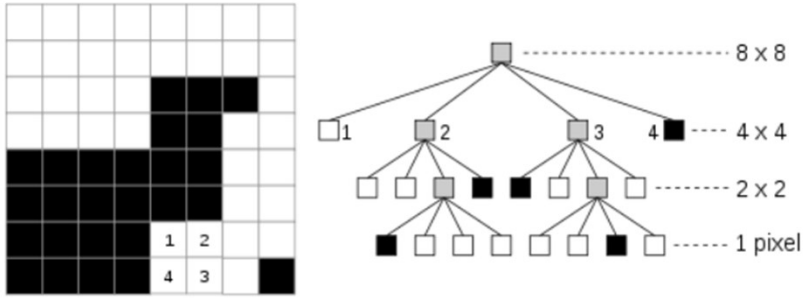
- Jest to struktura drzewiasta służąca przeszukiwaniu obszarów przestrzeni dwuwymiarowej dzieląc ją rekurencyjnie na cztery ćwiartki.





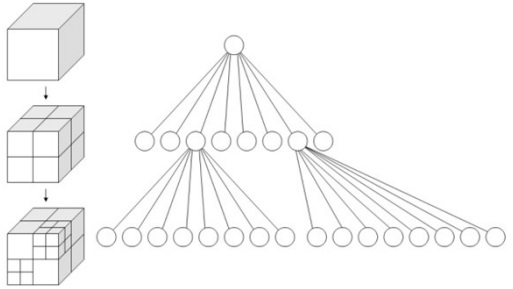
QuadTree – przykładowe zastosowania

- Wykrywanie kolizji w dwóch wymiarach
- Kompresja bitmap dwukolorowych



QuadTree – zalety

- Łatwe skalowanie na wyższe wymiary (OctTree)



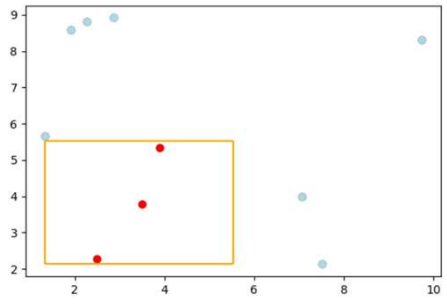
QuadTree – budowanie drzewa

- Budowanie drzewa zaczynamy od określenia maksymalnego prostokąta zawierającego cały zbiór punktów, który jest ustawiany jako korzeń naszej struktury

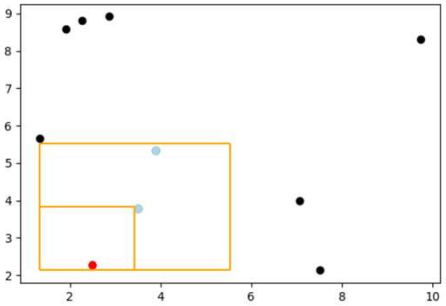
- Następnie rekurencyjnie każdy z węzłów dzielimy na cztery równe ćwiartki, który przypisujemy kolejne węzły aż do momentu w którym w danym węźle będzie nie więcej niż zadane w konstruktorze drzewa k punktów.

wizualizacja budowania QuadTree

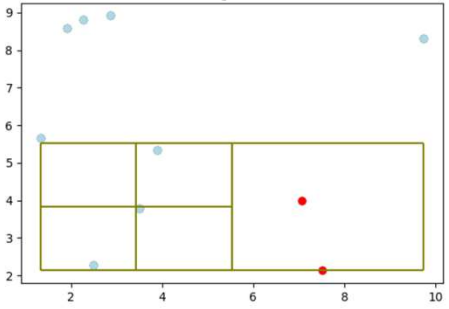
1



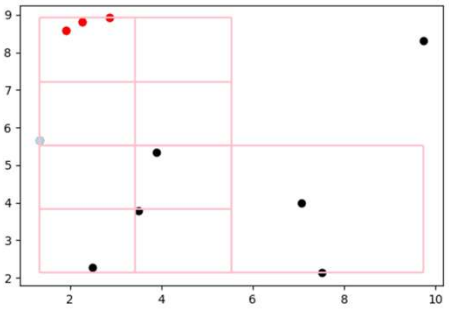
2



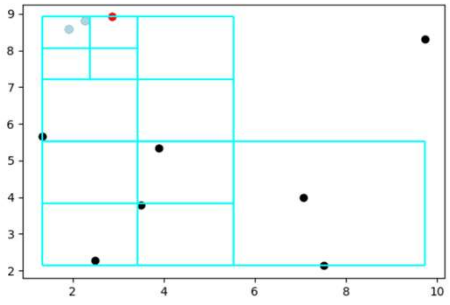
3



4

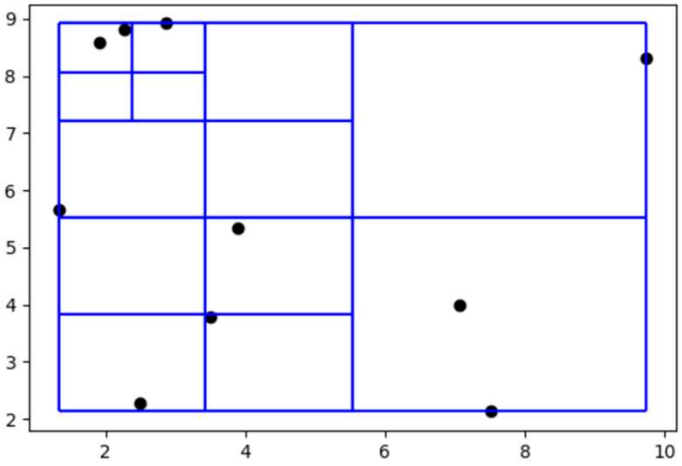


5



. . .

6

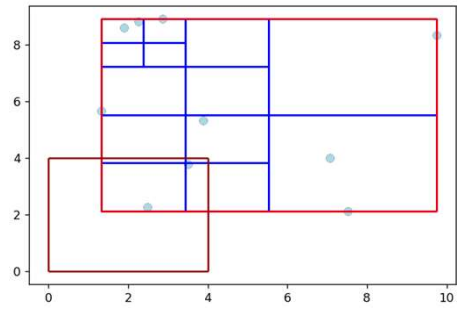


QuadTree – przeszukiwanie przestrzeni

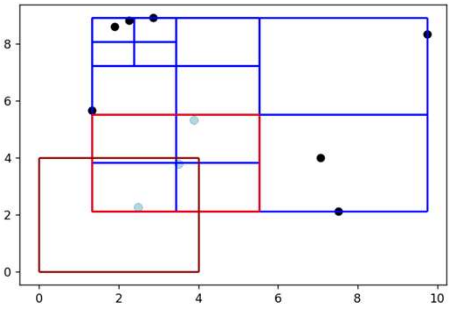
- Przeszukiwanie drzewa zaczynamy od korzenia i następnie rekurencyjnie przechodzimy po poddrzewach.
- Węzły możemy podzielić na cztery typy:
- Liść - sprawdzamy wszystkie punkty w liściu czy należą do obszaru.
- Węzeł całkowicie zawierający się w przeszukiwanym obszarze – wszystkie punkty dodajemy do wyniku.
- Węzeł całkowicie poza przeszukiwanym obszarem - kończymy przeszukiwanie w tym poddrzewie.
- Węzeł częściowo zawarty w przeszukiwanym obszarze ale nie liść - wywołujemy rekurencyjnie przeszukiwanie wszystkich dzieci/poddrzew danego węzła

wizualizacja przeszukiwania QuadTree

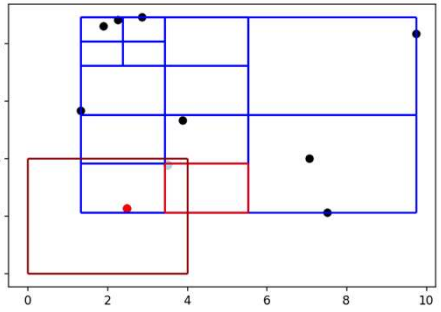
1



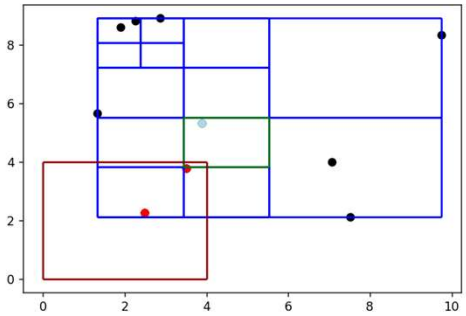
2



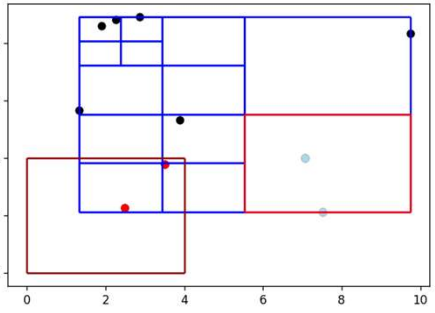
3



4

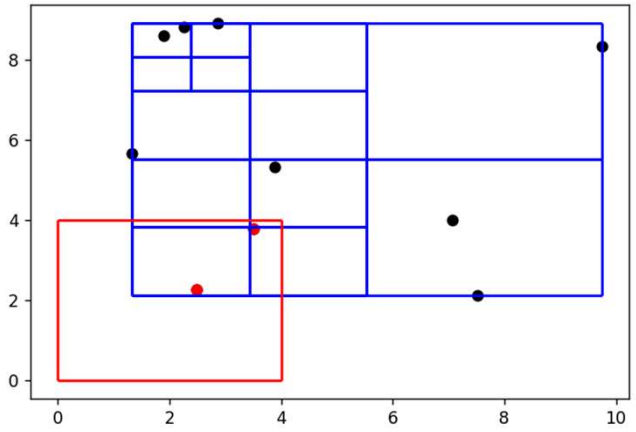


5



. . .

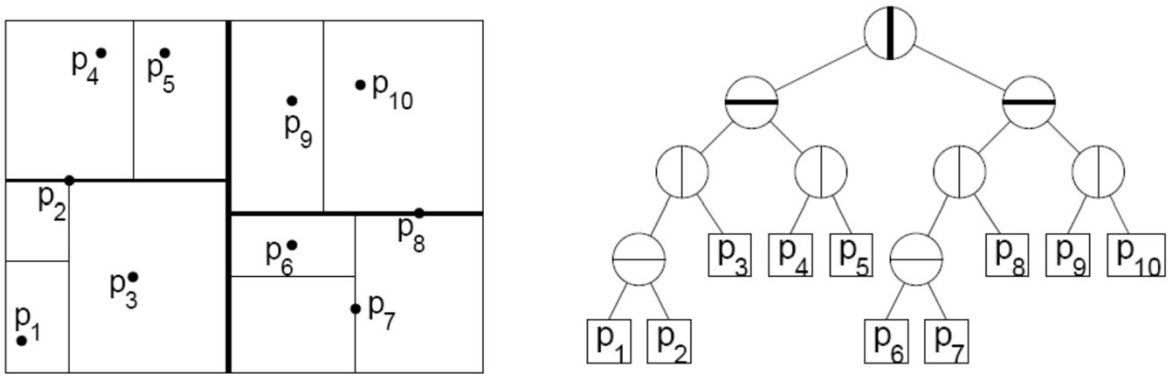
6



KDTree

KDTree – opis struktury

Drzewo k-wymiarowe (ang. KDTree) – wariant drzewa binarnego, struktura służąca do przechowywania i organizacji punktów przestrzeni k-wymiarowej.



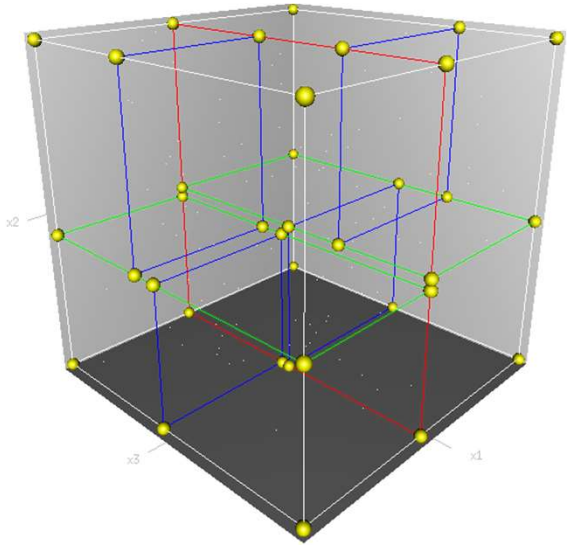
W każdym węźle dochodzi do podziału punktów wg. jednej współrzędnej. Liście przechowują punkty.

KDTree – przykładowe zastosowanie

- Znajdowanie najbliższych sąsiadów;
- Wyszukiwanie punktów w obszarach, w tym np. filtrowanie danych.

KDTree – zalety

- KDTree w łatwy sposób można przenieść na przestrzenie wielowymiarowe



KDTree – budowanie drzewa

Algorytm budowania drzewa polega na wielokrotnym podziale punktów względem jednej z współrzędnych.

W naszej implementacji:

- Na parzystych poziomach punkty zostały podzielone względem współrzędnej y, a na nieparzystych względem współrzędnej x.
- Dla zbalansowania: Każdorazowo punkty były sortowane względem danej współrzędnej, a punkt przedziału był wybierany jako średnia k-tego punktu i k+1-ego punktu, gdzie k to [liczba punktów]//2-1.
- Następnie algorytm rekurencyjnie tworzy poddrzewa dla zbiorów punktów, otrzymanych w danym podziale.
- Jeżeli zbiór punktów składa się z jednego punktu, tworzymy liść, do którego przypisujemy ten punkt.

KDTree – złożoność budowania drzewa

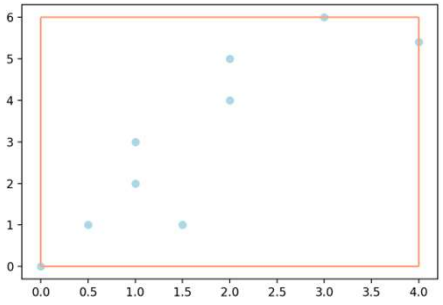
W budowaniu KDTree ważne jest ustalenie mediany dla każdego węzła, aby zapewnić głębokość O(log n), jednak jest to kosztowny krok:

- Sortowanie O(n log n)
- Ustalenie mediany O(1)
- Podział punktów O(n)

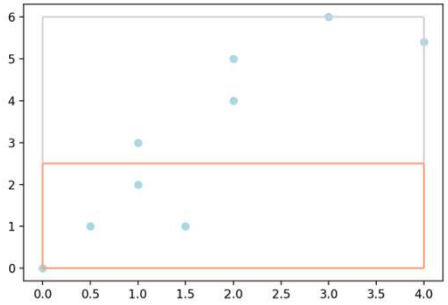
Stąd złożoność czasowa budowy drzewa to $O(nlog^2 n)$

Wizualizacja budowania KDTree

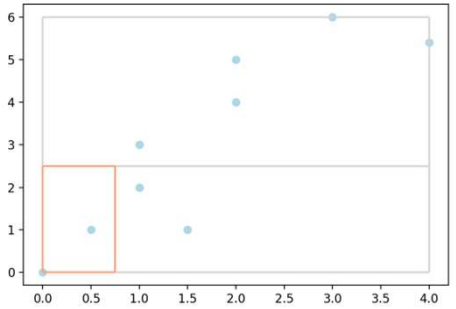
1



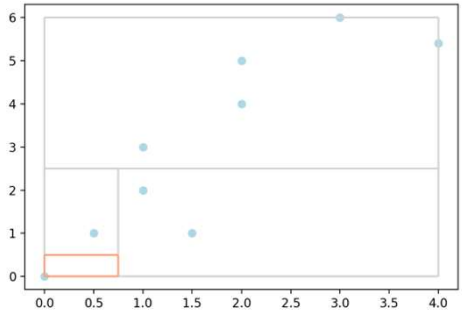
2



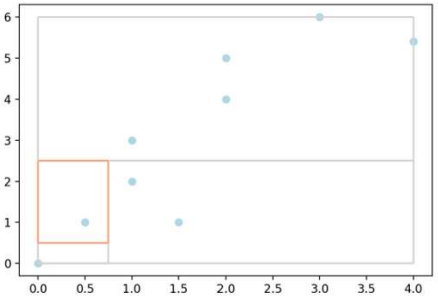
3



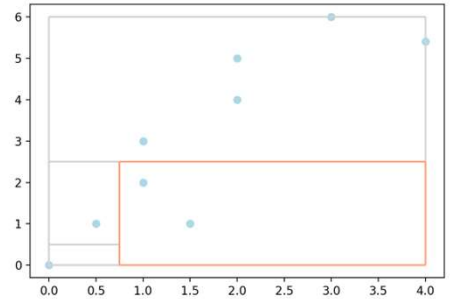
4



5

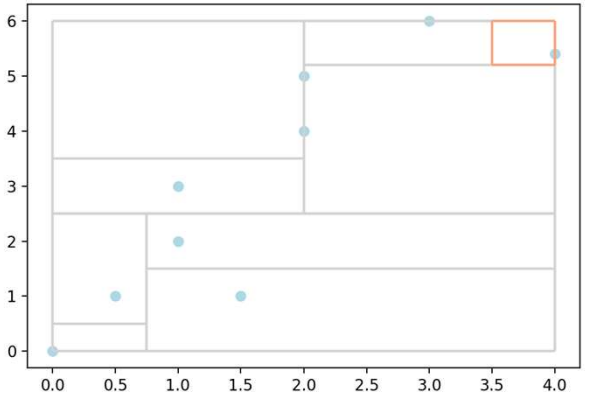


6



...

ostatni krok



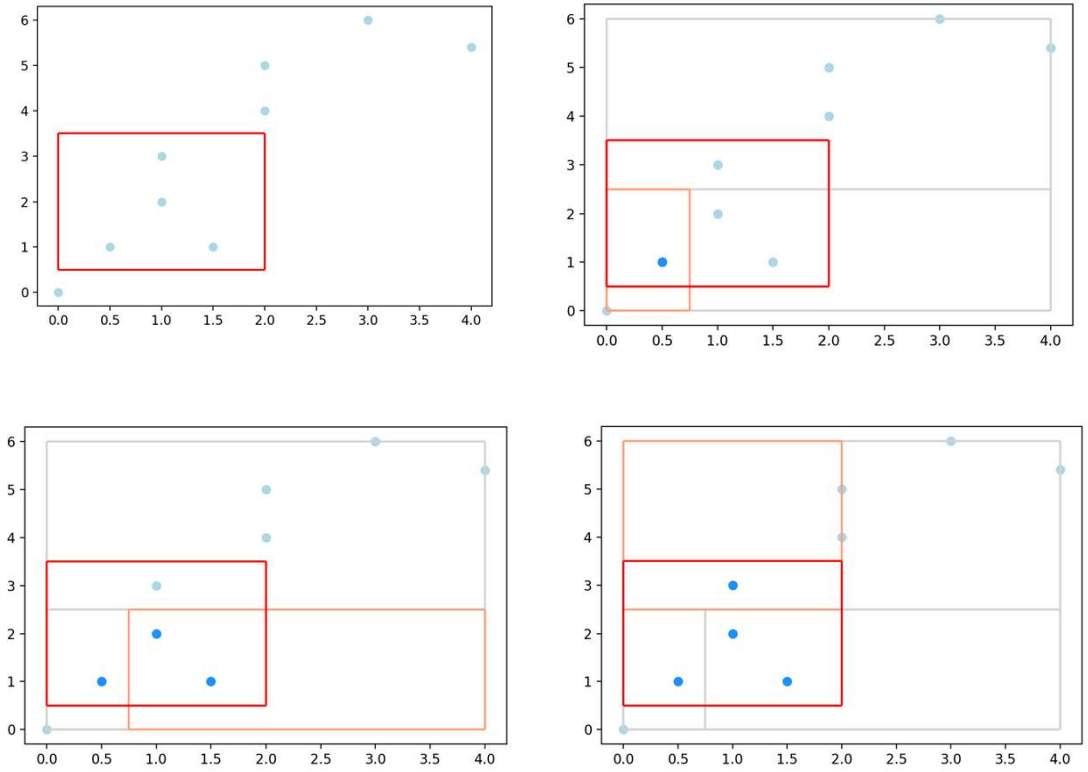
KDTree – przeszukiwanie przestrzeni

Algorytm polega na wyszukaniu wszystkich punktów z zdanego ortogonalnego przedziału D.

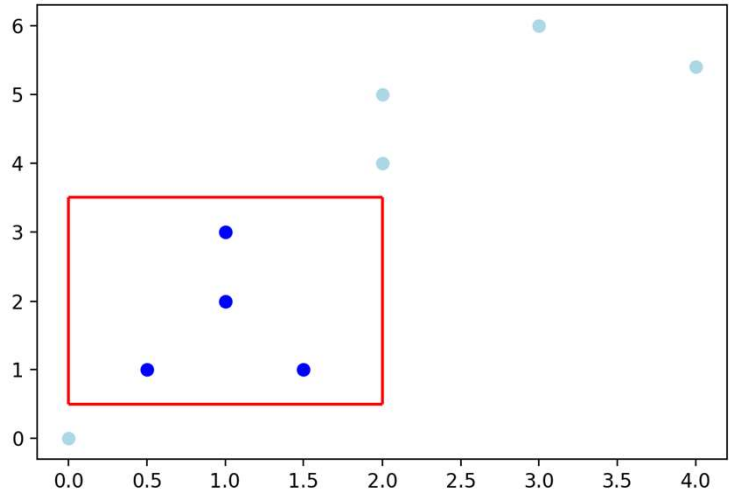
W naszej implementacji:

- Jeżeli przedział reprezentowany przez lewe lub prawe poddrzewo danego węzła:
  - zawiera się w D, to zwracam wszystkie liście z tego poddrzewa;
  - przecina się z D, to rekurencyjnie przeszukuję to poddrzewo.
- Warunek brzegowy to natrafienie w rekurencji na liść – w takim przypadku sprawdzam czy punkt z liścia należy do D i zwracam w zależności od wyniku.

Wizualizacja przeszukiwania przestrzeni z wykorzystaniem KDTree



rezultat:



Porównanie QuadTree i KDTree

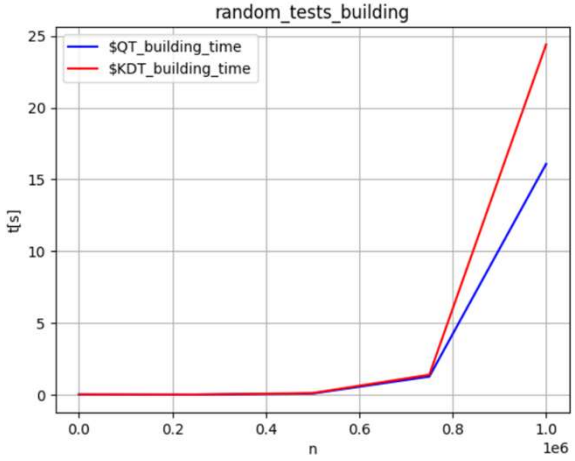
Cechy struktur

- Obie struktury realizują ideę podziału obszaru na mniejsze podobszary.
- KDTree można łatwo rozszerzyć na wyższe wymiary.
- W QuadTree można w łatwy sposób dodawać nowe punkty. W KDTree nie jest to takie łatwe.

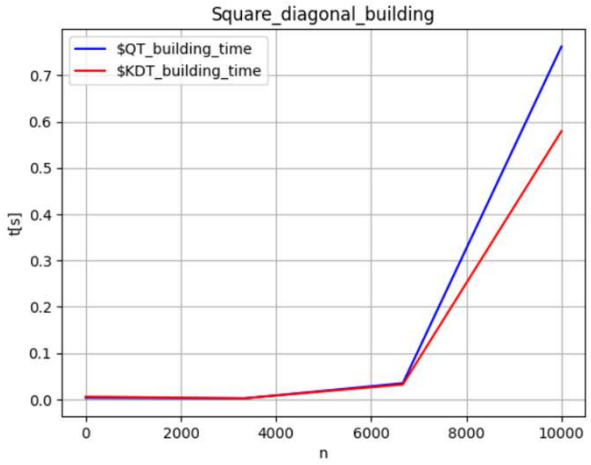
Wykorzystane zbiory danych

- Wszystkie zbiory zostały wygenerowane wewnątrz kwadratu 1000x1000 za pomocą funkcji random.uniform() oraz random.choice().
- Zostały przygotowane następujące zbiory do testowania:
- random_1eX – testy losowe , X oznacza ilość punktów. Łącznie stworzyliśmy 5 takich testów.
- linear_X_Y – testy typu liniowego w których punkty leżą na pionowych i poziomych liniach, których jest łącznie Y, punktów jest X. Łącznie stworzyliśmy 4 takie testy.
- square_diagonal_X – testy w których punkty leżą na dwóch bokach prostokąta a także na obu przekątnych, X oznacza osobno ilość punktów na bokach oraz osobno na przekątnych. Łącznie stworzyliśmy 4 takie testy.

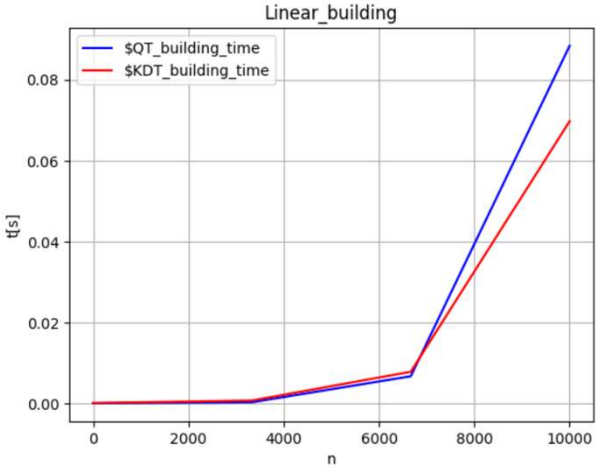
Czas budowy struktury



	collection	QT building time [s]	KDT building time [s]
0	random_1e2	0.015739	0.018770
1	random_1e3	0.005139	0.010189
2	random_1e4	0.083910	0.112884
3	random_1e5	1.264895	1.391615
4	random_1e6	16.080887	24.406131

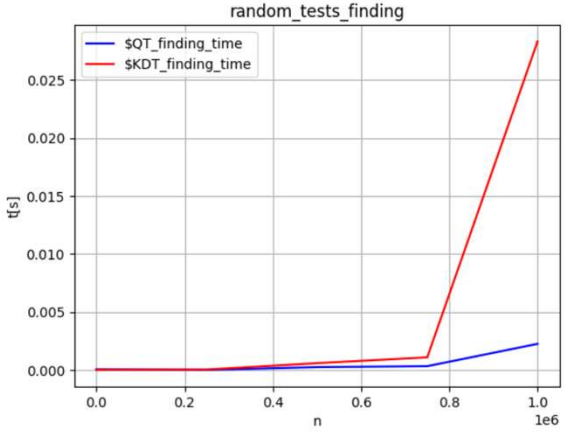


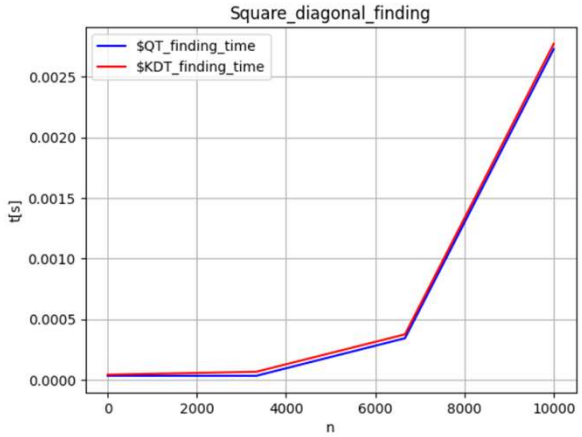
	collection	QT building time [s]	KDT building time [s]
0	square_diagonal_30	0.003463	0.005811
1	square_diagonal_100	0.002216	0.002374
2	square_diagonal_1000	0.034962	0.032159
3	square_diagonal_10000	0.761832	0.579034

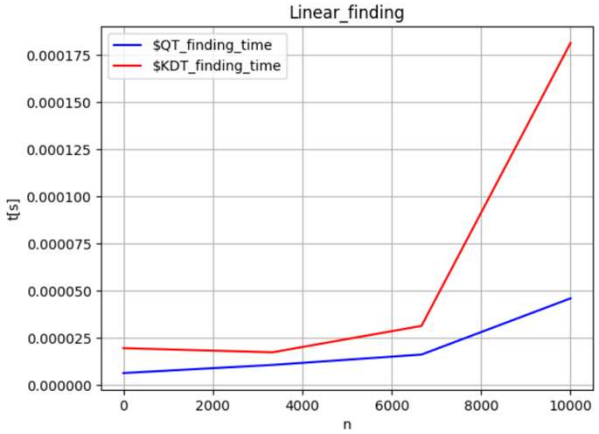


	collection	QT building time [s]	KDT building time [s]
0	linear_30_10	0.000117	0.000117
1	linear_100_10	0.000382	0.000382
2	linear_1000_10	0.006792	0.006792
3	linear_10000_10	0.088338	0.088338

Czas przeszukiwania







	collection	QT finding time [s]	KDT finding time [s]
0	random_1e2	0.000067	0.000033
1	random_1e3	0.000019	0.000052
2	random_1e4	0.000263	0.000602
3	random_1e5	0.000343	0.001104
4	random_1e6	0.002266	0.028310

	collection	QT finding time [s]	KDT finding time [s]
0	square_diagonal_30	0.000034	0.000043
1	square_diagonal_100	0.000034	0.000067
2	square_diagonal_1000	0.000344	0.000375
3	square_diagonal_10000	0.002729	0.002772

	collection	QT finding time [s]	KDT finding time [s]
0	linear_30_10	0.000006	0.000006
1	linear_100_10	0.000011	0.000011
2	linear_1000_10	0.000016	0.000016
3	linear_10000_10	0.000046	0.000046


Dziękujemy za uwagę!
Andrzej Karciński i Aleksandra Smela