**KD-Tree**

Opis klas w plikach modułu **kdtree**

Moduł zawiera 3 pliki: **kdtreeAuxClasses.py, kdtree.py, kdtreevis.py**

1. **kdtreeAuxClasses.py** implementuje 3 klasy pomocnicze dla KD-Tree:
   1. **Range –** obiekty tej klasy reprezentują prostokąt. Wykorzystuje się go przy przeszukiwaniu drzewa. Obszar, z którego punkty chcemy wydobyć, reprezentowany jest jako obiekt klasy Range. Obszar, za który „odpowiedzialny” jest dany węzeł w drzewie, również reprezentowany jest obiektem tej klasy.

(**UWAGA nr 1**: oba te użycia różnią się nieznacznie tym, że obszar przeszukiwania jest domknięty, a obszar węzła jest domknięty na dolnej i prawej linii, lecz otwarty na górnej i lewej. Zostało to uwzględnione w metodach **isContainedIn** oraz **intersects** opisanych niżej)

Klasa ta zawiera atrybuty:

* + - **x1, x2, y1, y2 –** współrzędne linii, którymi ograniczony jest prostokąt (**UWAGA nr 2:** dla uproszczenia x1 <= x2 oraz y1 <= y2)

**Metody:**

* **isContainedIn(self, other)** – sprawdza, czy dany prostokąt zawiera się w podanym (uwzględnia uwagę nr 1)
  + Zwraca wartość typu **bool**
* **isPointInRange(self, point) –** sprawdza,czy podany punkt zawiera się w prostokącie
  + Zwraca wartość typu **bool**
* **intersects(self, other) –** sprawdza,czy obiekt Range przecina się z danym (uwzględnia uwagę nr 1)
  + Zwraca wartość typu **bool**
* **returnSplit(self, axis, line)** - na podstawie danej linii przecięcia oraz numeru osi zwraca lewy i prawy prostokąt nią przecięty
  + Parametry: **self –** obiekt Range, **axis** – nr osi (0 – oś x, 1 – oś y), **line –** współrzędna linii
  + Zwraca dwa obiekty typu **range –** lewe przecięcie i prawe przecięcie
* **returnSplit(self, axis, line)** - na podstawie danej linii przecięcia oraz numeru osi zwraca lewy i prawy prostokąt nią przecięty

**Metody pomocnicze:**

* **\_\_str\_\_(self)** – zwraca łańcuchową reprezentację obiektu
* **\_\_copy\_\_(self) –** zwraca kopię obiektu
* **\_\_corrdValidation(self) –** sprawdza, czy warunek z uwagi 2 jest spełniony
  1. **LeafNode -** liść KD-tree - przechowuje jeden punkt (klasę tę można pominąć w implementacji KD-tree, jednak pozwala ona lepiej zwizualizować KD-drzewo) Atrybuty:
     + **point –** tupla z koordynatami punktu, który przechowuje liść
  2. **Node –** węzeł KD-tree. Atrybuty:
     + **splitCoord –** współrzędna podziału (**Uwaga 3:** nie zapisujemy, której osi współrzędnych dotyczy współrzędna, ponieważ przechodząc drzewo, możnawydedukować to na podstawie ich głębokości)
     + **left –** wskaźnik na kolejny węzeł bądź liść zawierający punkty o mniejszej bądź równej współrzędnej na odpowiedniej osi niż wartość wskazana przez linię podziału
     + **right –** jak wyżej, leczwspółrzędne punktu/punktów przechowywanych przez wskazany liść/węzeł są większe

1. **kdtree.py –** główny plik modułu zawierający jedynie jedną klasę – implementację KD-tree.
   1. **KDTree –** dokładny opis znajduje się w prezentacji

**Atrybuty:**

* **maxRange –** obiekt klasy Range, który przechowuje najmniejszy prostokąt zawierający wszystkie dane punkty.
* **kdTreeRoot –** wskaźnik na pierwszy węzeł KD-Tree

**Metody publiczne (interfejs):**

* **Konstruktor –** przyjmuje listę punktów. Tworzy dwie listy na bazie danej listy punktów – jedną posortowaną wg współrzędnej x oraz drugą po współrzędnej y – i na ich podstawie oblicza maxRange oraz buduje drzewo (szczegóły w prezentacji)
* **printTree(self) –** wypisuje drzewo (funkcja służąca do debugowania programu)
* **search(self, searchRange,** node=None, nodeRange=None, depth=0**)** – rekurencyjna funkcja, wyszukująca wszystkie punkty znajdujące się w zadanym przedziale
  + **Parametry:**
    - **searchRange –** obiekt typu Range, reprezentujący zadany przez użytkownika przedział;
    - **node** (domyślna wartość: None) **–** obecnie przetwarzany węzeł/liść drzewa. Jeśli node jest Nonem, to przypisuje się mu wartość self.kdTreeRoot;
    - **nodeRange** (domyślna wartość – None) – przedział, za który „odpowiada” dany węzeł
    - **depth** (domyślna wartość – 0) – obecnagłębokość drzewa
  + **Zwraca:** listę punktów zawartych w zadanym przedziale

**Metody prywatne i „protected” (pomocnicze):**

* **\_\_initAux(self, pointsXSorted, pointsYSorted, depth=0)** – rekurencyjna funkcja tworząca KD-tree. Dokładny opis działania znajduje się w prezentacji (link)
  + **Parametry:**
    - **pointsXSorted, pointsYSorted** – punkty posortowane względem współrzędnej x i y (**UWAGA 4** – obie listy przechowują te same punkty, lecz w różnej kolejności)
    - **depth** (domyślna wartość – 0) – obecnagłębokość drzewa
  + **Zwraca:**
    - **Obiekt klasy LeafNode** jeżeli dane tablice były jednoelementowe (zwracany obiekt zawiera zawarty w tablicach punkt)
    - **Obiekt klasy Node** w przeciwnym wypadku, zawierający linię przecięcia, lewy oraz prawy węzeł/liść
* **\_split(self, x\_sorted, y\_sorted, axis) –** funkcja obliczająca punkt przecięcia dwóch tablic i dzieląca je wg tej granicy.
  + **Parametry:**
    - **x\_sorted, y\_sorted** – posortowane punkty odpowiednio wg x i y
    - **axis** – numer osi przecięcia
  + **Zwraca:**
    - Wartości tablicy **result –** 4 listy wynikowe podziału danych tablic wg obliczonej granicy
    - **splitPointCoordinate –** obliczona granica podziału
* **\_\_findMaxRange(self, pointsXSorted, pointsYSorted)–** funkcja zwracająca atrybut maxRange na bazie dwóch posortowanych tablic.
* **\_reportSubtree(self, node)** – funkcja zwracająca listę punktów znajdujących się w liściach danego węzła KD-tree

1. **kdtreevis.py –** plik odpowiedzialny za wizualizację inicjalizacji oraz przeszukiwania KD-tree. Zawiera 2 klasy:
   1. **Visualizer –** wizualizator drzewa, zapamiętujący jego obecny stan tak, aby mozna było w łatwy sposób stworzyć sceny do narzędzia graficznego oraz na koniec je zwrócić.

**Atrybuty:**

* **\_\_colors –** słownik zawierający kolory używane przy wizualizacji (atrybut prywatny)
* **setOfPoints** – początkowy zbiór punktów
* **maxRange –** przechowuje atrybut maxRange obiektu KDTreeVis (opisanego niżej). Domyślnie: None
* **searchRange –** przechowuje obiekt klasy Range reprezentujący obszar przeszukiwania KD- tree. Domyślnie: None
* **lines –** linie podziału, przechowywane w węzłach obiektu klasy KDTreeVis. Domyślnie: pusta lista
* **reportedPoints** – punkty znalezione w trakcie przeszukiwania drzewa. Domyślnie: pusta lista
* **initScenes –** sceny pokazujące inicjalizację KD- tree. Domyślnie: scena pokazująca jedynie punkty z parametru **setOfPoints**
* **searchScenes** – sceny pokazujące przeszukiwanie KD-tree. Domyślnie: pusta lista

**Metody publiczne (interfejs):**

* **konstruktor –** przyjmuje listę punktów, zwraca obiekt klasy Visualizer
* **setMaxRange(self, maxRange)** – ustawia atrybut maxRange na bazie danego parametru
* **setSearchRange(self, searchRange)** – ustawia atrybut searchRange na bazie danego parametru
* **addLine(self, splitCoord, smallerBound, biggerBound, axis)** – dodaje linie przecięcia na bazie współrzędnej przecięcia (**splitCoord**), osi (**axis**), oraz granic przedziału (**smallerBound, biggerBound)**
* **addPoint(self, point)** – dodaje punkt do reportedPoints
* **makeScene(self, currPoints=None, currRange=None)** – tworzy nową scenę na bazie danych parametrów oraz własnych atrybutów oraz dodaje ją do odpowiedniego atrybutu (**initScenes** lub **searchScenes**)
  + **Parametry:**
    - **currPoints –** obecnie przetwarzane punkty (w czasie inicjalizacji)
    - **currRange –** obecnie analizowany obszar (w czasie wyszukiwania
* **getInitScenes(self)** – zwraca sceny inicjalizacji
* **getSearchScenes(self) –** dodaje ostatnią scenę (jedynie punkty wraz z zaznaczonymi odnalezionymi punktami oraz przeszukiwany obszar) do scen przeszukiwania i je zwraca.
* **clear(self)** – ustawia wartości domyślne dla atrybutów tak, aby można było znów zwizualizować przeszukiwanie KD-tree

**Metody prywatne (pomocnicze):**

* **\_\_getRangeLines(self,rangeObject) –** zwraca listę odcinków danego prostokąta (reprezentowanego jako obiekt klasy Range) lub pustą listę jeśli parametr jest wartością None
  1. **KDTreeVis –** klasa dziedzicząca klasy KDTree. Zasadniczo różni się tym, że zawiera dodatkowe pole **vis**, zawierające obiekt klasy Visualizer potrzebny do graficznego przedstawienia działania. Większość klas z macierzystej klasy została nadpisana w taki sposób, aby obiekt klasy Visualizer zapisywał sceny przedstawiające najważniejsze etapy działania algorytmu inicjalizacji i przeszukiwania. Poza tym owe metody przyjmują, działają i zwracają to samo, co metody przez nie nadpisywane.

Do tych metod należą:

* + - **Konstruktor** – posiada stworzenie nowego obiektu Visualizer i przypisanie go do atrybutu **vis.** Tworzy również jedną scenę
    - **\_\_initAux(self, pointsXSorted, pointsYSorted, depth=0)** – tworzy scenę przed podziałem punktów (zaznaczając wszystkie punkty zawarte w drzewie danego węzła) oraz po podziale, z dorysowaniem linii.
    - **search(self, searchRange, node=None, nodeRange=None, depth=0)** -metoda ta:
      1. w przypadku pierwszego zejścia rekurencyjnego ustawia w wizualizatorze obszar wyszukiwania (metodą **setSearchRange**) oraz dodaje pierwszą scenę
      2. w przypadku liścia dodaje punkt do wizualizatora i dodaje nową scenę
      3. w przypadku węzła dodaje 2 sceny z zaznaczeniem punktów po odpowiednim podziale
    - **\_\_reportSubtree(self, node)** - metoda ta znajduje znalezione punkty do wizualizatora

Ponadto w owej klasie występują metody rozszerzające nadrzędną klasę. Są one związane przede wszystkim z obsługą wizualizatora:

* **getInitPlot(self)** – zwraca obiekt typu Plot na bazie scen inicjalizacji z wizualizatora
* **getSearchPlot(self)** - zwraca obiekt typu Plot na bazie scen przeszukiwania z wizualizatora
* **clearVis(self)** – czyści sceny przeszukiwania z wizualizatora (**UWAGA 5 –** zaleca się używanie tej metody po każdym wywołaniu getSearchPlot, w przeciwnym wypadku obiekt Plot będzie zawierał kolejne sceny z różnych przeszukiwań