Lokalizacja punktu w przestrzeni dwuwymiarowej Metoda trapezowa

Dokumentacja techniczna projektu

Konrad Pękala

Jan Stobnicki

Grudzień 2020

# 1. Wprowadzenie

W ramach projektu zaliczeniowego przygotowaliśmy algorytm lokalizujący punkt w przestrzeni dwuwymiarowej metodą trapezową. W ramach inicjalizacji budowany jest graf DAG (directed acyclic graph), dzięki któremu sam algorytm lokalizacji zachowuje złożoność logarytmiczną.

# 2. Podstawowe informacje techniczne

Implementacje algorytmu przygotowano w języku Python. Główna część algorytmu znajduje się w pliku algo.py. Struktury danych znajdują się pliku structures.py. W pliku visualizer.py znajduje klasa Visualizer, dzięki której program będzie wyświetlał wyniki działania algorytmu. Wykorzystaliśmy zaproponowaną bibliotekę graficzną, która znajduje się w pliku lib.py.

Projekt znajduje się na githubie: <https://github.com/kpekala/PointLocation>

## 2.1 Wymagane pakiety

Dla poprawnego działania oprogramowania wymagane jest uruchomienie modułów w środowisku uruchomieniowym zawierającym poniższe moduły.

1. algo:
   1. ui.lib
   2. ui.plot\_utils
   3. ui.visualizer
   4. structures
2. main:
   1. algo
   2. ui.plot\_utils
   3. ui.visualizer
   4. structures
3. structures:
   1. utils
4. utils:
   1. typing
5. ui.plot\_utils:
   1. structures
   2. random
6. ui.visualizer
   1. ui.lib
   2. utils

# 3. Struktury

Moduł: structures

## 3.1 Struktura modułu

Moduł implementuje klasy:

* Point – reprezentująca punkt na mapie
* Segment – reprezentująca odcinek na mapie
* XNode – wierzchołek drzewa DAG reprezentujący punkt
* YNode - wierzchołek drzewa DAG reprezentujący odcinek
* YNode - wierzchołek drzewa DAG reprezentujący odcinek
* TrapezoidNode - wierzchołek drzewa DAG reprezentujący trapez
* Dag – klasa trzymająca referencje do korzenia grafu

A także funkcje:

* createTrapezoid(topSegment, bottomSegment, leftPoint, rightPoint)

Argumenty:

* + topSegment: górny odcinek trapezu
  + bottomSegment: dolny odcinek trapeze
  + leftPoint: punkt na trapezie o najmniejszej współrzędnej x
  + rightPoint: punkt na trapezie o największej współrzędnej x

Wartość zwracana: obiekt klasy TrapezoidNode

## 3.2 Klasa Point

### 3.2.1 Implementowane metody

* \_\_init\_\_(self, x, y)

Argumenty:

* + x: współrzędna x
  + y: współrzędna y

Wartość zwracana: brak

* toList(self)

Argumenty: brak

Wartość zwracana: Lista dwu-elementowa gdzie pierwszy element to współrzędna x a drugi to współrzędna y

* \_\_str\_\_(self)

Argumenty: brak

Wartość zwracana: wartość punktu w formie tekstowej

## 3.3 Klasa Segment

### 3.3.1 Implementowane metody

* \_\_init\_\_(self, p, q)

Argumenty:

* + p: lewy punkt odcinka
  + q: prawy punkt odcinka

Wartość zwracana: brak

* isPointAbove(self, point)

Argumenty:

* + point: punkt na mapie

Wartość zwracana: wartość bool

* getY(self, x)

Argumenty:

* + x: wartość x

Wartość zwracana: wartość y punktu na odcinku o współrzędnej x

* toList(self)

Argumenty: brak

Wartość zwracana: Lista dwu-elementowa gdzie pierwszy element to lewy punkt odcinka a drugi to prawy punkt odcinka

* \_\_str\_\_(self)

Argumenty: brak

Wartość zwracana: wartość odcinka w formie tekstowej

## 3.4 XNode

### 3.4.1 Implementowane metody

* \_\_init\_\_(self, point, left = None, right = None)

Argumenty:

* + point: punkt na mapie reprezentowany przez XNode
  + left: lewy wierzchołek danego XNoda w grafie DAG
  + right: prawy wierzchołek danego XNoda w grafie DAG

Wartość zwracana: brak

* setLeft(self, node)

Argumenty:

* + node: nowy wierzchołek do dodania w grafie DAG jako lewe dziecko danego XNoda

Wartość zwracana: brak

* setRight(self, node)

Argumenty:

* + node: nowy wierzchołek do dodania w grafie DAG jako prawe dziecko danego XNoda

Wartość zwracana: brak

## 3.5 YNode

### 3.5.1 Implementowane metody

* \_\_init\_\_(self, segment, above = None, below = None)

Argumenty:

* + segment: odcinek na mapie reprezentowany przez YNode
  + above: wierzchołek będący geometrycznie powyżej danego odcinka
  + below: wierzchołek będący geometrycznie poniżej danego odcinka

Wartość zwracana: brak

* setBelow(self, node)

Argumenty:

* + node: nowy wierzchołek do dodania w grafie DAG jako prawe dziecko danego YNoda

Wartość zwracana: brak

* setAbove(self, node)

Argumenty:

* + node: nowy wierzchołek do dodania w grafie DAG jako lewe dziecko danego YNoda

Wartość zwracana: brak

## 3.6 TrapezoidNode

### 3.6.1 Implementowane metody

* \_\_init\_\_(self, segment, topSegment = None, bottomSegment = None, leftPoint = None, rightPoint = None)

Argumenty:

* + topSegment: odcinek będący górną częscią trapezu
  + bottomSegment: odcinek będący dolną częscią trapezu
  + leftPoint: punkt wyznaczający lewą część trapezu
  + rightPoint: punkt wyznaczający prawą część trapezu

Wartość zwracana: brak

* containsSegment(self, segment)

Argumenty:

* + segment: odcinek na mapie

Wartość zwracana: bool

* containsPoint(self, point)

Argumenty:

* + point: punkt na mapie

Wartość zwracana: bool

* replacePositionWith(self, dag, node)

Argumenty:

* + dag: graf reprezentujący mapę
  + node: wierzchołek

Wartość zwracana: brak

* toLines(self)

Argumenty: brak

Wartość zwracana: 4-elementowa tablica zawierająca odcinki trapezu w formie tablicowej

## 3.7 Dag

### 3.7.1 Implementowane metody

* \_\_init\_\_(self, root)

Argumenty:

* + root: korzeń grafu DAG reprezentującego mapę

Wartość zwracana: brak

* updateRoot(self, root)

Argumenty:

* + root: korzeń grafu DAG reprezentującego mapę

Wartość zwracana: brak

# 4. Algorytmy

Moduł: algo

Moduł implementuje funkcje:

* buildDag(lineSegments, visualizer = None)

Argumenty:

* + lineSegments: lista przechowująca wejściowe odcinki
  + visualizer: obiekt zapisujący kroki algorytmu potrzebne do wizualizacji

Wartość zwracana: graf DAG reprezentujący podział trapezowy mapy

* findArea(node, point)

Argumenty:

* + node: aktualny wierzchołek w przeszukiwaniu grafu
  + point: punkt na mapie dla którego funkcja znajdzie lokalizacje na mapie

Wartość zwracana: obszar na który znajduje się punkt

* simpleCase(dag, trapezoid, segment, visualizer = None)

Argumenty:

* + dag: graf reprezentujący mapę
  + trapezoid: rozpatrywany trapez
  + segment: rozpatrywany odcinek
  + visualizer: obiekt zapisujący kroki algorytmu potrzebne do wizualizacji

Wartość zwracana: brak

* hardCase(dag, intersectedTrapezoids, segment, visualizer = None)

Argumenty:

* + dag: graf reprezentujący mapę
  + intersectedTrapezoids: rozpatrywane trapezy
  + segment: rozpatrywany odcinek
  + visualizer: obiekt zapisujący kroki algorytmu potrzebne do wizualizacji

Wartość zwracana: brak

* findIntersectedTrapezoids(node, segment, intersectedTrapezoids):

Argumenty:

* + node: aktualnie rozpatrywany wierzchołek
  + segment: rozpatrywany odcinek
  + intersectedTrapezoids: rozpatrywane trapezy

Wartość zwracana: brak

* createBoundingBox():

Argumenty: brak

Wartość zwracana: Trapez reprezentujący ściany mapy

# 5. Wizualizacja

## 5.1 Struktura modułu

Moduł implementuje klasy:

* DagData – klasa przechowująca wszystkie elementy na mapie
* Visualizer - klasa zapisująca kroki algorytmu potrzebne do wizualizacji

## 5.2 Klasa DagData

### 5.2.1 Implementowane metody

* \_\_init\_\_(self)

## 5.3 Klasa Visualizer

### 5.3.1 Implementowane metody

* \_\_init\_\_(self, lines)

Argumenty:

* + lines: bazowe odcinki w wizualizacji

Wartość zwracana: brak

* addFigure(self, lines)

Argumenty:

* + lines: odcinki figury

Wartość zwracana: brak

* addDag(self, dag)

Argumenty:

* + dag: graf reprezentujący mapę

Wartość zwracana: brak

* traverseDag(self, dagData: DagData, node)

Argumenty:

* + dagData: obiekt w którym będzie się zapisywało wszystkie elementy z mapy
  + node: aktualnie rozpatrywany wierzchołek

Wartość zwracana: brak

* getScenes(self)

Argumenty: brak

Wartość zwracana: lista zawierająca sceny

# 6. Pozostałe funkcje

Moduły utils i plot\_utils

* coefficients(p1, p2)

Argumenty:

* + p1: lewy punt odcinka
  + p2: prawy punkt odcinka

Wartość zwracana: współczynniki funkcji liniowej opartej na podanym odcinku

* functionValue(seg, x)

Argumenty:

* + segment: odcinek
  + x: współrzędna x

Wartość zwracana: wartość y odcinka w punkcie x

* generateRandomLineSegments(n, min\_x, max\_x, min\_y, max\_y)

Argumenty:

* + n: liczba odcinków
  + min\_x: minimalna wartość x
  + max\_x: maksymalna wartość x
  + min\_y: minimalna wartość y
  + max\_y: maksymalna wartość y

Wartość zwracana: lista losowo wygenerowanych odcinków