Algorytmy Geometryczne

Obliczanie grafu widoczności

Dokumentacja

Maciej Pięta

Szymon Twardosz

Grupa 6

Spis treści

**Dane techniczne1**

**Dokumentacja4**

Część użytkownika5

Util.py – opis funkcji6

Obstacle.py – opis klas oraz funkcji6

Graph.py – opis klas6

dijkstra.py – opis funkcji6

visibility.py – opis funkcji6

Visualiser.py – opis klas6

1. **Dane techniczne**

Projekt wykonywany został w języku Python. Korzystaliśmy również z narzędzia jupyter notebook. Do rysowania wizualizacji zostały użyte funkcję udostępnione na platformie UPEL. Do projektu podłączone zostały zewnętrzne biblioteki: numpy, matplotlib, sortedcontainers, bisect, math oraz copy.

Projekt składa się z plików:

* main.py
* visibility.py
* util.py
* graphic\_tool.py
* Visualiser.py
* dijkstra.py
* Graph.py
* Obstacle.py

lub w pliku visibile\_graph.ipynb, który zawiera kod wszystkich powyższych plików.

1. **Dokumentacja**
2. **Część użytkownika**

Użytkownik może korzystać z programu na dwa sposoby.

Pierwszy z nich to włączenie programu uruchamiając plik main.py. Po uruchomieniu programu na ekranie wyświetli się puste ,,płótno” na którym należy wprowadzić: punkt startowy, punkt końcowy ( poprzez naciśnięcie przycisku Dodaj punkt), oraz przeszkód ( poprzez naciśnięcie przycisku Dodaj figurę). Następnie okno to należy zamknąć. Po chwili na ekranie pojawi się pierwsza wizualizacja. Pokazuje ona sposób w jaki sposób tworzony jest graf. Po zamknięciu okna pojawi się druga wizualizacja. Ta reprezentować będzie działanie algorytmu dijkstry na utworzonym grafie oraz wyświetli najkrótszą ścieżkę.

Innym sposobem jest korzystanie z programu poprzez narzędzie jupyter notebook. W pliku tym znajdują się przykładowe wizualizację, które użytkownik może wykorzystać w celu analizy algorytmu. Oprócz tego dostępne jest również czyste ,,płótno” na którym można zadawać własne przeszkody oraz punkty.

1. **Utli.py – opis funkcji**

Plik ten implementuje funkcję odpowiedzialne za wczytywanie informacji umieszczanych przez użytkownika w układzie współrzędnych. W szczególności:

* **get\_obstacle\_from\_linesCollection**(lcol: LinesCollection, pointcount: int, obscount: int)

**Argumenty:** lcol – obiekt typu LinesCollection, który służy do przechowywania linii

pointcount – zmienna typu int przechowująca liczbę wczytanych już punktów

obscount– zmienna typu int przechowująca liczbę wczytanych już przeszkód

**Działanie:** Funkcja tworzy obiekt typu Obstacle() do którego zapisuje linie tworzące daną przeszkodę

**Zwraca:** Obiekt typu Obstacle()

* **get\_added\_elements**(plot1: Plot)

**Argumenty:** plot1 – obiekt typu Plot(), przechowuje on wszystkie informację, które użytkownik wprowadził na płótnie (punkty, linie, figury)

**Działanie:** Funkcja (korzystając z plot1) pobiera informację o położeniu wszystkich punktów oraz przeszkód

**Zwraca:** Listę punktów (typu Point) oraz listę przeszkód (typu Obstacle)

1. **Obstacle.py – opis klas i funkcji**

W pliku znajduje się implementacja trzech klas, które reprezentującą Punkt, Linię oraz Przeszkodę. Ich nazwy to kolejno Point, Line i Obstacle. Oprócz tego znajdują się w nim dwie funkcje. W szczególności są to:

* **orient**(p1: Point, p2: Point, p3: Point)

**Argumenty:** p1,p2,p3 – zmienne typu Point reprezentujące punkty na płaszczyźnie

**Działanie:** Funkcja na podstawie wyliczonego wyznacznika stwierdza czy punkt p3 leży po lewej/prawej stronie prostej p1p2, czy może jest współliniowy

**Zwraca:** Zmienną int w zależności od pozycji punktu p3

* **cross\_prod**(l1: Line, l2: Line)

**Argumenty:** l1, l2 – zmienne typu Line reprezentujące linie na płaszczyźnie

**Działanie:** Funkcja oblicza iloczyn wektorowy zmiennych l1,l2

**Zwraca:** Iloczyn wektorowy

Klasy zaimplementowane w tym pliku to:

* **Point –** obiekt reprezentujący punkt na płaszczyźnie
* **Atrybuty: x –** współrzędna x

**y –** współrzędna y

**ind –** indeks punktu

**oind –** indeks przeszkody do której ten punkt należy

**origin –** (\*\*\*)

**max\_X –** atrybut statyczny mówiący jaka jest maksymalna odcięta wszystkich punktów

* **Metody:**
* **\_\_init**\_\_(self, p: (float, float), pointIndex: int, obstacleIndex: int)

**Atrybuty:** p – krotka przechowująca dwie wartości, które są kolejno współrzędną x oraz y punktu

pointIndex – indeks punku

obstacleIndex – indeks przeszkody do której ten punkt należy

**Działanie**: Tworzy nowy obiekt typu Point

**Zwraca**: Obiekt typu Point

* **update\_origin**(og)

**Argumenty:** og (\*\*\*)

**Działanie:** Aktualizuje zmienną statyczną origin

**Zwraca:** brak

* **distance(**self, other**)**

**Argumenty:** other – zmienna typu Point

**Działanie:** oblicza dystans w metryce euklidesowej między jaki dzieli punkt od punktu other

**Zwraca:** dystans w metryce euklidesowej

* **dist\_sqr**() (\*\*\*)
* **findAngle (\*\*\*) – POWINNO BYĆ ZMIENIONE NA SORTOWANIE PRZEZ WYZNACZNIK**
* **\_\_eq\_\_**(self,other)

**Argumenty:** other – zmienna typu Point

**Działanie:** Sprawdza czy punkt jest taki sam jak other

**Zwraca:** Zmienną typu bool

* **\_\_gt\_\_**(self, other)

**Argumenty:** other – obiekt typu Point

**Działanie:** Sprawdza czy punkt jest większy od punktu other

**Zwraca:** Zmienna typu bool

* **Line –** klasa reprezentująca linię na płaszczyźnie
* **Atrybuty:** p1 – obiekt klasy Point, reprezentuje jeden koniec odcinka

p2 – obiekt klasy Point, reprezentuje drugi koniec odcinka

m – współczynnik kierunkowy prostej, którą wyznacza dany odcinek

b – wyraz wolny prostej opisanej wyżej

xIntercept – (\*\*\*)

seenCount – (\*\*\*)

sweepDistance – (\*\*\*)

* **Metody:**
* **\_\_init\_\_**(self, p1: Point, p2: Point)

**Argumenty:** p1, p2 – obiekty typu Point reprezentujące końce odcinka

**Działanie:** Tworzy obiekt typu Line

**Zwraca:** Obiekt typu Line

* **half\_line\_orientation**(self, halfline)

**Argumenty:** halfline – obiekt typu Line

**Działanie:** Sprawdza czy linia leży zgodnie z ruchem zegara względem linii halfline

**Zwraca:** Zmienną typu int

* **intersect\_line**(self, other)

**Argumenty:** other- obiekt typu Line

**Działanie:** Sprawdza czy linie się przecinają

**Zwraca:** Zmienna typu bool

* **\_\_eq\_\_**(other)

**Argumenty:** other – obiekt typu Line

**Działanie:** Sprawdza czy odcinek jest równy other

**Zwraca:** Zmienna typu bool

* **get\_len(**self)

**Argumenty:** brak

**Działanie:** Zwraca długość odcinka

**Zwraca:** Zmienna typu float

* **update\_sweep\_len**(self, new)

**Argumenty:** new – zmienna typu float

**Działanie:** Aktualizują atrybut sweepDistance

**Zwraca:** brak

* **Obstacle –** klasa reprezentujący przeszkodę
* **Atrybuty:** points – lista punktów typu Point, które należą do przeszkody

edges – lista linii typu Line, które należą do przeszkody

ind – indeks przeszkody

pointIndices – zbiór indeksów punktów które należą do przeszkody

minVertex – minimalny punkt należący do przeszkody w metryce określonej w klasie punkt

* **Metody:**
  + **\_\_init\_\_**(self, index)

**Argumenty:** index – zmienna typu int, reprezentuje indeks przeszkody

**Działanie:** Tworzy obiekt typu Obstacle

**Zwraca:** Obiekt typu Obstacle

* **add\_point**(self, p)

**Argumenty:** p – obiekt typu Point, punkt należący do przeszkody

**Działanie:** Dodaje punkt do listy points

**Zwraca:** Brak

* **add\_edge**(self, e)

**Argumenty:** e – obiekt typu Line, linia należąca do przeszkody

**Działanie**: Dodaje linię do listy edges

**Zwraca**: Brak

* **get\_incident\_lines**(self, vertex)

**Argumenty:** vertex – zmienna typu Point, punkt należący do przeszkody

**Działanie:** Szuka linii których końcem jest zadany punkt vertex.

**Zwraca:** Dwa obiekty typu Line

* **get\_intersecting\_edges**(self, line)

**Argumenty:** line – zmienna typu Line

**Działanie:** Szuka linii które przecinają obiekt line

**Zwraca:** Listę obiektów typu Line

* **same\_line**(point1, point2)

**Argumenty:** point1, point2 – obiekty typu Point należące do przeszkody

**Działanie:** Sprawdza czy punkty te posiadają wspólną krawędź

**Zwraca:** Zmienną typu bool

1. **Graph.py – opis klas**

W pliku znajduję się implementacja dwóch klas. W szczególności są to:

* Node – klasa reprezentująca wierzchołek grafu
* **Atrybuty:**

index – zmienna typu int, reprezentująca indeks wierzchołka

point – krotka, złożona z dwóch zmiennych typu float, reprezentuje współrzędne punktu na płaszczyźnie

edges – słownik reprezentujący połączenia pomiędzy punktem a jego sąsiadami

* **Metody**
* **\_\_init\_\_**(self, index, point)

**Argumenty:** index – zmienna typu int,

point – krotka, złożona z dwóch zmiennych typu float

**Działanie:** Tworzy obiekt typu Node

**Zwraca:** Obiekt typu Node

* **add\_edge**(self, other, weight)

**Argumenty:** other – zmienna typu int, sąsiad wierzchołka

weight – zmienna typu float, waga krawędzi między wierzchołkiem i jego sąsiadem

**Działanie:** Dodaje krawędź do słownika

**Zwraca:** Brak

* **Graph –** klasa reprezentująca graf
* **Atrybuty:**

nodeList – lista obiektów typu Node, reprezentuje wszystkie wierzchołki grafu

node\_coord – lista krotek złożonych z typów float, które są współrzędnymi wierzchołków na płaszczyźnie

edges – słownik reprezentujący krawędzie

edges\_coord – lista list złożonych z krotek, które są współrzędnymi linii na płaszczyźnie

* **Metody**
* **\_\_init\_\_**(self, v)

**Argumenty:** v – lista krotek, które są współrzędnymi kolejnych punktów

**Działanie:** Tworzy obiekt typu Graph

**Zwraca:** Obiekt typu Graph

* **add\_edge**(self, n1, n2, weight)

**Argumenty:** n1, n2 – indeksy wierzchołków których łączy krawędź

weight – zmienna typu float, waga krawędzi między wierzchołkiem i jego sąsiadem

**Działanie:** Dodaje krawędź do słownika

**Zwraca:** Brak

* **add\_node**(self, n)

**Argumenty:** n – obiekt typu Node, reprezentuje wierzchołek grafu

**Działanie:** Dodaje wierzchołek do listy

**Zwraca:** Brak

* **\_\_len\_\_**(self)

**Argumenty:** Brak

**Działanie:** Zwraca liczbę wierzchołków

**Zwraca:** Zmienna typu int – liczbę wierzchołków

1. **dijkstra.py – opis funkcji**

W pliku znajduję się implementacja algorytmu Dijkstry

* **dijkstra**(graph: Graph, s: int, t: int, visualise\_flag: bool)

**Argumenty:** graph – obiekt typu Graph, graf widoczności

s – zmienna typu int, indeks punktu, który jest punktem początkiem

t – zmienna typu int, indeks punkty, który jest punktem końcowym

visualise\_flag – flaga, sterująca wizualizacją

**Działanie:** Funkcja zgodnie z algorytmem Edsgera Dijkstry wyznacza najkrótszą drogę z punktu startowego do końcowego

**Zwraca:** Zmienną typu float – dystans z punktu s do t; listę zmiennych typu int – indeksy poprzedników każdego wierzchołka; listę obiektów typu Scene – sceny wizualizacji

1. **visibility.py – opis funkcji**

W pliku znajduje się implementacja funkcji, które tworzą graf widoczności. W szczególności są to:

* **visible**(w, pw, obstacles, i, w\_list, BroomT, visible\_list)

**Argumenty:** w – obiekt typu Point, aktualnie rozpatrywany punkt

pw – obiekt typu Line, miotła

obstacles – lista obiektów typu Obstacle, zawiera wszystkie rozpatrywane przeszkody

i – indeks rozpatrywanego punktu w tablicy w\_list

w\_list – lista obiektów typu Point, wszystkie punkty wprowadzone przez użytkownika

BroomT – obiekt typu SortedList(), aktualny stan miotły

visible\_list – lista zmiennych bool, daje informację czy i-ty punkt jest widoczny dla rozpatrywanego punktu

**Działanie:** Sprawdza czy punkt w jest widoczny przy zadanym stanie miotły i dla zadanego punktu

**Zwraca:** Zmienną typu bool

* **visible\_vertices**(point, obstacles, graph, vertices, visualiser)

**Argumenty:** point – obiekt klasy Point, punkt dla którego funkcja szukać będzie widocznych punktów

obstacles – lista obiektów klasy Obstacle, wszystkie przeszkody zadane przez użytkownika

graph – obiekt typu Graph, graf widoczności

vertices – lista obiektów klasy Point, wszystkie punkty zadane przez użytkownika

visualiser – obiekt klasy VisibilityVisualiser, wizualizuje przebieg algorytmu

**Działanie:** Dla zadanego punktu szuka innych, które są ,,widoczne” oraz dodaje je do grafu widoczności wraz z wyliczonymi wagami ( według normy Euklidesowej)

**Zwraca:** Brak

* **compute\_graph**(points, obstacles, vis\_flag)

**Argumenty**: points – lista obiektów klasy Point, wszystkie punkty zadane przez użytkownika

obstacles – lista obiektów klasy Obstacle, wszystkie przeszkody zadane przez użytkownika

vis\_flag – zmienna typu bool, odpowiedzialna za sterowanie wizualizacją

**Działanie:** Tworzy graf widoczności

**Zwraca:** Graf widoczności

* **find\_and\_remove**(T, val)

**Argumenty:** T – obiekt klasy SortedList, aktualny stan miotły

val – krotka, element do usunięcia

**Działanie:** usuwa val z SortedList’y

**Zwraca:** Zmienną typu bool

1. **visualiser.py – opis klas**

W pliku znajduje się implementacja klas, które tworzą wizualizację tworzenia grafu oraz algorytmu dijkstry. W szczególności są to klasy:

* **DijkstraVisualiser** – klasa służąca to wizualizacji przebiegu algorytmu Dijkstry
* **Atrybuty:** points – lista krotek złożonych z zmiennych typu float, punkty na płaszczyźnie

edges – lista list, w których znajdują się krotki złożone z zmiennych typu float, linie na płaszczyźnie

proccessed\_points – lista krotek złożonych z zmiennych typu float, współrzędne przetworzonych wierzchołków w algorytmie Dijkstry

scenes – lista obiektów klasy Scene, przechowuje sceny potrzebne do wizualizacji

destiny – krotka zawierająca zmienne typu float, współrzędne punktu końcowego

* **Metody**
* **\_\_init\_\_**(self, points, edges, destiny)

**Argumenty:** points – lista krotek złożonych z zmiennych typu float, współrzędne punktów na płaszczyźnie

edges – lista list, w których znajdują się krotki złożone z zmiennych typu float, współrzędne linii na płaszczyźnie

destiny – krotka zawierająca zmienne typu float, współrzędne punktu końcowego

**Działanie:** Tworzy obiekt klasy DijkstraVisualiser

**Zwraca:** Obiekt klasy DijkstraVisualiser

* **create\_start\_scene**(self)

**Argumenty:** Brak

**Działanie:** Tworzy obiekt klasy Scene oraz zapisuje go do listy scenes

**Zwraca:** Brak

* **process\_point**(self, p)

**Argumenty:** p – zmienna typu int, indeks punktu

**Działanie:** Zapisuje indeks p w tablicy proccesed\_points

**Zwraca:** Brak

* **create\_scene**(self, start\_point, end\_point)

**Argumenty:** start\_point – zmienna typu int, indeks punktu do którego znamy najkrótszą ścieżkę

end\_point – zmienna typu int, indeks punktu dla którego szukamy najkrótszej ścieżki ( jest połączony z start\_point)

**Działanie:** Tworzy obiekt klasy Scene oraz zapisuje go do listy scenes

**Zwraca:** Brak

* **create\_end\_scene**(self, parent, s, t)

**Argumenty:** parent – lista zmiennych typu int, poprzednicy każdego wierzchołka w ich najkrótszych ścieżkach z wierzchołka s do t

s – zmienna typu int, indeks punktu startowego

t – zmienna typu int, indeks punktu końcowego

**Działanie:** Tworzy obiekt typu Scene i zapisuje go do listy scenes

**Zwraca:** Brak

* **get\_scenes**(self)

**Argumenty:** Brak

**Działanie:** Brak

**Zwraca:** Listę obiektów klasy Scene

* **VisibilityVisualiser** – klasa służąca to wizualizacji przebiegu funkcji tworzącej graf widoczności
* **Atrybuty:** lines– lista list, w których znajdują się krotki złożone z zmiennych typu float, linie na płaszczyźnie

points – lista krotek złożonych z zmiennych typu float, punkty na płaszczyźnie

start\_point – krotka zmiennych typu float, współrzędne punktu startowego na płaszczyźnie

end\_point – krotka zmiennych typu float, współrzędne punktu końcowego na płaszczyźnie

broom\_X – zmienna typu float, współrzędna x miotły równoległej do osi X

scenes – lista obiektów klasy Scene, przechowuje sceny potrzebne do wizualizacji

* **Metody**
* **\_\_init\_\_**(self, lines, points, start\_point, end\_point, x)

**Argumenty:** lines– lista list, w których znajdują się krotki złożone z zmiennych typu float, linie na płaszczyźnie

points – lista krotek złożonych z zmiennych typu float, punkty na płaszczyźnie

start\_point – krotka zmiennych typu float, współrzędne punktu startowego na płaszczyźnie

end\_point – krotka zmiennych typu float, współrzędne punktu końcowego na płaszczyźnie

broom\_X – zmienna typu float, współrzędna x miotły równoległej do osi X

visible\_vert – lista krotek zmiennych typu float, współrzędne widocznych punktów

broom\_line – lista krotek zmiennych typu float, współrzędne końców miotły

**Działanie:** Tworzy obiekt klasy VisibilityVisualiser

**Zwraca:** Obiekt klasy VisibilityVisualiser

* **create\_start\_scene**(self)

**Argumenty:** Brak

**Działanie:** Tworzy obiekt klasy Scene oraz zapisuje go do listy scenes

**Zwraca:** Brak

* **create\_broom\_scene**(self, broom)

**Argumenty:** broom – obiekt klasy Line, miotła

**Działanie –** Tworzy obiekt klasy Scenes i zapisuje go do listy scenes

**Zwraca:** Brak

* **intersecting\_scenes**(self, intersecting\_lines)

**Argumenty:** intersecting\_lines – lista obiektów klasy Line, linie znajdujące się aktualnie w strukturze stanu miotły

**Działanie:** Tworzy obiekt klasy Scene i zapisuje go do listy scenes

**Zwraca:** Brak

* **change\_broom\_scene**(broom, lines, visible\_flag)

**Argumenty:** broom – obiekt klasy Line, miotła na płaszczyźnie

**Działanie:** Tworzy dwa obiekty klasy Scene i zapisuje je do listy scenes

**Zwraca:** Brak

* **graph\_connection\_scene**(self, point)

**Argumenty:** point – obiekt klasy Point, aktualnie rozpatrywany punkt dla którego szukamy połączeń z innymi punktami

**Działanie:** Tworzy obiekt klasy Scene i zapisuje go do listy scenes

**Zwraca:** Brak