Algorytm Fortune'a - opis implementacji

Łukasz Jezapkowicz

1 Struktury Danych

1.1 Klasa Coordinate

Klasa Coordinate służy do reprezentacji zwykłego punktu w przestrzeni dwuwymiarowej. Posiada metody *init* oraz *str* (oczywiste zastosowanie).

1.2 Klasa Arc

Klasa Arc służy do reprezentacji łuku (paraboli) w przestrzeni dwuwymiarowej. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Oprócz tego posiada metodę *get_plot*, która dla podanego przedziału x-ów generuje odpowiednie współrzędne y-owe pozwalające na wizualizację paraboli.

1.3 Klasa Breakpoint

Klasa Breakpoint służy do reprezentacji punktu załamania w przestrzeni dwuwymiarowej. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Oprócz tego posiada metody *does_intersect* oraz *get_intersection* (bazowane na algorytmie użytkownika mbrubeck) znajdujące punkt załamania.

1.4 Klasa HalfEdge

Klasa HalfEdge służy do reprezentacji pół-krawędzi w przestrzeni dwuwymiarowej. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Oprócz tego posiada metodę *get_origin* zwracająca początek krawędzi, *set_next* ustawiającą podaną pół-krawedź jako *next* oraz getter i setter dla pola *_twin*.

1.5 Klasa Point

Klasa Point służy podobnie jak klasa Coordinate do reprezentacji punktu w przestrzeni dwuwymiarowej , poszerzona o wskaźnik do pierwszej krawędzi $first\_edge$. Dziedziczy po klasie Coordinate. Posiada metody *init* oraz *str* (oczywiste zastosowanie).

1.6 Klasa Polygon

Klasa Polygon służy do reprezentacji wielokąta w przestrzeni dwuwymiarowej. Używana w algorytmie do reprezentacji obramowania zamykającego diagram. Posiada metodę *init* (oczywiste zastosowanie). Oprócz tego posiada metodę *order_points* ustawiająca wierzchołki wielokąta w porządku zgodnym z ruchem wskazówek zegara, metodę *get_closest_point* znajdującą najbliższy wierzchołek względem podanego, metodę $finish\_edges$ zamykającą krawędzie w obramowaniu oraz pomocnicze metody *delete_edge* usuwająca krawędź ze struktury, $finish\_edge$ zamykającą pojedyńczą krawędź, *inside* sprawdzająca czy punkt leży wewnątrz wielokąta oraz *get_intersection_point* znajdującą punkt przecięcia krawędzi z obramowaniem.

1.7 Klasa Vertex

Klasa Vertex służy podobnie jak klasa Coordinate do reprezentacji punktu w przestrzeni dwuwymiarowej, poszerzona o krawędzie do niego wchodzące. Reprezentuje wierzchołek diagramu. Dziedziczy po klasie Coordinate. Posiada metody *init* oraz *str* (oczywiste zastosowanie).

2 Struktura zdarzeń

2.1 Klasa Event

Klasa Event służy do reprezentacji zdarzenia w strukturze zdarzeń. Posiada metodę *init* oraz metody nadpisujące *eq*, *ne* oraz *lt* (oczywiste zastosowanie).

2.2 Klasa CircleEvent

Klasa CircleEvent służy do reprezentacji zdarzenia kołowego w strukturze zdarzeń. Dziedziczy po klasie Event. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Posiada również metodę *remove* zaznaczającą zdarzenie jako "fałszywe", metodę statyczną *create_circle_event* tworzącą nowe zdarzenie kołowe na podstawie 3 podanych łuków, metodę *create_circle* tworzącą koło z podanych 3 punktów oraz dwie właściwości x oraz y ułatwiające porównanie obiektu z innym zdarzeniem (tutaj przyjęto najniższy punkt okręgu).

2.3 Klasa SiteEvent

Klasa SiteEvent służy do reprezentacji zdarzenia punktowego w strukturze zdarzeń. Dziedziczy po klasie Event. Posiada metody *init* oraz *str* (oczywiste zastosowanie) oraz dwie właściwości x oraz y ułatwiające porównanie obiektu z innym zdarzeniem (tutaj przyjęto po prostu podany punkt).

3 Struktura stanu

3.1 Klasa Node

Klasa Node służy do reprezentacji węzła w drzewie struktury stanu. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Posiada właściwości $left$, $right$, $grandparent$, $height$, $balance$, $successor$, $predecessor$ zwracające odpowiednio lewego syna, prawego syna, dziadka, wysokość, różnicę wysokości lewego i prawego poddrzewa, następnika, poprzednika węzła. Posiada również metodę *get_key* zwracającą dane węzła, settery $left$ i $right$ ustawiające lewego i prawego syna węzła, metody *calculate_height*, *update_height*, *update_heights* zajmujące się wysokością węzła w drzewie, metody is_left_child, is_right_child, is_leaf odpowiadające na pytanie czy węzeł jest lewym synem, prawym synem, liściem, metody *minimum* oraz *maximum* zwracające minimum i maksimum poddrzewa o korzeniu w węźle oraz metodę $replace\_leaf$ rozbijająca liść na poddrzewo.

3.2 Klasa LeafNode

Klasa LeafNode służy do reprezentacji liścia w drzewie struktury stanu. Dziedziczy po klasie Node. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Posiada również metody *get_key* oraz *get_value* zwracające punkt przypisany paraboli oraz parabolę.

3.3 Klasa InternalNode

Klasa InternalNode służy do reprezentacji wewnętrznego węzła w drzewie struktury stanu. Dziedziczy po klasie Node. Posiada metody *init* oraz *str* (oczywiste zastosowanie). Posiada również metody *get_key* oraz *get_value* zwracające współrzędne punktu załamania oraz punkt załamania.

3.4 Klasa Tree

Klasa Tree służy do reprezentacji drzewa. Zawiera statyczne metody $find\_value$ znajdującą węzeł o podanych danych, $find\_leaf\_node$ znajdującą liść o podanym kluczu, *insert* wstawiającą węzeł do drzewa, *delete* usuwającą węzeł z drzewa, *balance_and_propagate* oraz *balance* balansujące drzewo, $rotate\_left$ oraz *rorate_right* naprawiające drzewo.

4 Klasa Math

Klasa Math dostarcza metody pozwalające obliczenie pewnych wartości. Posiada statyczne metody *distance* obliczającą odległość między dwoma punktami, *magnitude* obliczającą długość wektora, *norm* obliczającą normę wektora, *line_ray_intersection_point* oraz *get_intersection* znajdujące (jeśli istnieje) punkt przecięcia dwóch odcinków, *calculate_angle* obliczającą kąt pomiędzy dwoma punktami oraz *check_clockwise* sprawdzające czy punkty ułożone są zgodnie z kierunkiem ruchu wskazówek zegara.

5 Klasa Voronoi

Klasa Voronoi służy do znajdywania diagramu Voronoi dla danej chmury punktów. Posiada metody *init*, *initialize* (oczywiste zastosowanie), *create_diagram* tworzącą diagram Voronoi, *handle_site_event* obsługującą zdarzenie punktowe, $handle_circle_event$ obsługującą zdarzenie kołowe, *check_circles* sprawdzającą czy nie zachodzi zdarzenie kołowe oraz *update_breakpoints* aktualizującą punkty załamania. Metody działają w sposób zgodny z tym co pojawia się w książce Berga "Computational Geometry".

6 Wizualizacja

Wizualizację algorytmu Fortune'a można przeprowadzić używając pliku $Wizualizacja.ipynb$ oraz narzędzia Jupyter Notebook. Należy podać listę punktów w $2D$ (jako tuple) *points* oraz użyć kolejno metod $Voronoi()$ oraz $create\_diagram(points)$ by dostać listę obiektów klasy *Scene scenes*. Następnie należy te sceny zwizualizować używając kolejno $plot = Plot(scenes)$ oraz $plot.draw()$.

7 Inspiracje

Link 1
Link 2
Link 3
Link 4
Link 5
Link 6