Dokumentacja

1. Część użytkownika

Informacje o programie

Program uruchamiany jest przez m.in. program jupyter notebook, format pliku w którym znajduje się program to .ipynb. Działanie algorytmu zawartego w programie ilustruje dołączona do programu prezentacja.

Punktem określana jest para współrzędnych zmiennoprzecinkowych x i y, natomiast odcinkiem para punktów, tj. jego początek oraz koniec.

Praca z programem

Do poprawnego działania programu niezbędne jest uruchomienie po kolei wszystkich modułów w pliku main.ipynb. Należy również zdefiniować z użyciem interfejsu graficznego zbiór odcinków, dla którego algorytm ma się wykonać. W tym celu należy je wprowadzić z użyciem komórki „Wczytywanie zbioru odcinków", klikając na przycisk „dodaj linie". Następnie każde dwa kliknięcia spowodują dodanie odcinka o końcach w miejscach kliknięć.

Po zakończeniu wprowadzania odcinków można uzyskać wizualizację tworzenia mapy trapezowej przez algorytm, w tym celu należy uruchomić komórkę zatytułowaną „Wizualizacja tworzenia mapy trapezowej". Kolejne etapy jej wytwarzania można obserwować przy użyciu przycisku „następny" i „poprzedni".

Po jej wykonaniu można z użyciem kolejnych dwóch komórek dodać punkty, których położenie określić ma algorytm. W tym celu należy uruchomić komórkę „Wczytywanie punktów" i dodać je z użyciem przycisku „dodaj punkty". Każde kliknięcie na wykresie spowoduje dodanie pojedynczego punktu.

Następnie możliwe jest obserwowanie wizualizacji algorytmu wyszukiwania punktu w przestrzeni. W tym celu należy uruchomić komórkę „Wizualizacja odnajdywania położenia punktu." Wizualizacja zostaje przedstawiona dla wszystkich wprowadzonych przez użytkownika punktów, dokładnie w kolejności ich wprowadzania.

2. Część techniczna

A) Opisy struktur danych

Point:

Klasa reprezentująca punkt/wierzchołek w algorytmie.

Atrybuty:

x – wartość reprezentująca współrzędną x,

y – wartość reprezentująca współrzędną y,

segment – pole przechowuje wskaźnik na odcinek, do którego należy,

jeśli nie należy do żadnego, to wartością pole jest None.

Metody w klasie:

__init__(self, x, y) – konstruktor klasy.

Argumenty:

self – wskaźnik na samego siebie,

x – wartość reprezentująca współrzędną x,

y – wartość reprezentująca współrzędną y.

Zwraca:

Metoda nie zwraca nic.

__repr__(self) i __str__(self) - metody służące do reprezentacji punktu w formie tekstu.

Argumenty:

self – wskaźnik na samego siebie.

Zwraca:

String będący reprezentacją tekstową punkty.

__hash__(self) – metoda hashująca, która pozwala na jednoznaczną identyfikacje punktu w formie ciągu znaków.

Argumenty:

self – wskaźnik na samego siebie,

Zwraca:

Indywidualny ciąg znaków, jednoznacznie wskazujący na podany punkt. Jeżeli punkty mają, takie same parametry x i y to posiadają ten sam ciąg znaków.

__eq__(self, other) – metoda porównująca dwa punkty, jeśli dwa punkty znajdują się w odległości mniejszej niż ustalony epsilon, to uznaje, że to są te same punkty.

Argumenty:

self – wskaźnik na samego siebie,

other – wskaźnik na drugi punkt

Zwraca:

Wartość 1, jeśli są to te same punkty oraz 0, jeśli nie są one tym samym punktem.

set_segment(self, segment) – metoda przepisuje do punktu odcinek do którego należy.

Argumenty:

self – wskaźnik na samego siebie

segment – odcinek do którego należy

Zwraca:

Ta metoda nic nie zwraca

Segment:

Klasa reprezentująca odcinek w algorytmie.

Atrybuty:

p – wskaźnik na punkt będący lewym końcem odcinka

q – wskaźnik na punkt będący prawym końcem odcinka

a – współczynnik kierunkowy w równaniu prostej, na której leży odcinek

b – wyraz wolny w równaniu prostej, na której leży odcinek

a i b przyjmują wartości None jeśli p.x == q.x (taka sytuacja występuje tylko w odcinkach tworzonych w ramach wizualizacji poszczególnych kroków algorytmu)

Metody w klasie:

__init__(self, p, q) – konstruktor klasy

Argumenty:

self – wskaźnik na samego siebie

p – wskaźnik na jeden z punktów

q – wskaźnik na drugi punkt

Zwraca:

Metoda nic nie zwraca

__eq__(self, other) – metoda porównująca, czy odcinki mają te same końce, co implikuje ich równość.

Argumenty:

self, – wskaźnik na samego siebie

other – wskaźnik na drugi punkt

__hash__(self) – metoda hashująca, pozwala na jednoznaczną identyfikację odcinka w formie ciągu znaków

Argumenty:

self – wskaźnik na samego siebie

Zwraca:

Ciąg znaków jednoznacznie wskazujący na dany odcinek

__repr__(self) i __str__(self) - metody służące do reprezentacji odcinka w formie tekstu.

Argumenty:

self – wskaźnik na samego siebie.

Zwraca:

String będący reprezentacją tekstową odcinka.

point_position(self, point) – funkcja zwracająca położenie punktu względem prostej.

Argumenty:

self – wskaźnik na samego siebie

point – punkt, którego położenie względem odcinka ma zostać sprawdzone

Zwraca:

Wartość dodatnią, jeśli punkt znajduje się nad odcinkiem, wartość ujemną jeśli punkt znajduje się pod odcinkiem. Jeśli punkt znajduje się poza przedziałem x odcinka, zwraca 0.

y(self, x) – określanie wartości y dla danego x dla prostej, w której zawiera się odcinek

Argumenty:

Self – wskaźnik na samego siebie

x – wartość typu float

Zwraca:

Wartość y, równą y = ax + b, czyli wartość y dla prostej na której leży odcinek dla danego x

Trapezoid:

Klasa reprezentująca trapez w algorytmie.

Atrybuty:

bottom – odcinek ograniczający trapez od dołu (segment)

top – odcinek ograniczający trapez od góry (segment)

leftp – lewy punkt ograniczający trapez

rightp – prawy punkt ograniczający trapez

upper_right – wskaźnik na prawego-górnego sąsiada trapezu

upper_leftt – wskaźnik na lewego-górnego sąsiada trapezu

lower_right – wskaźnik na prawego-dolnego sąsiada trapezu

lower_left – wskaźnik na lewego-dolnego sąsiada trapezu

self.node – wskaźnik do node reprezentującego trapez w grafie

Metody w klasie:

__init__(self): - konstruktor klasy

Argumenty:

self – wskazanie na samego siebie

Zwraca:

Funkcja nic nie zwraca.

__repr__(self) i __str__(self) - metody służące do reprezentacji trapezu w formie tekstu.

Argumenty:

self – wskazanie na samego siebie

NodeArea:

Klasa reprezentująca trapez w grafie przeszukiwania.

Atrybuty:

delta – trapez, do jakiego odnosi się dany node

parents – lista wszystkich node'ów, które posiadają krawędzie skierowane do tego node'a

Metody w klasie:

__init_(self, delta):

Argumenty:

self – wskazanie na samego siebie

delta – trapez, do jakiego będzie odnosić się dany node

Zwraca:

Nic.

NodeVertex:

Klasa reprezentująca wierzchołek w grafie przeszukiwania.

Atrybuty:

left_side – odnośnik do lewego dziecka (krawędź skierowana) w grafie (to, co znajduje się po lewej stronie wierzchołka)

right_side – odnośnik do prawego dziecka (krawędź skierowana) w grafie (to, co znajduje się po prawej stronie wierzchołka)

vertex – wierzchołek, do jakiego odnosi się dany node

Metody w klasie:

__init__(self, left_side, right_side, vertex) – konstruktor klasy

Argumenty:

self – wskazanie na samego siebie

left_side – odnośnik do node, który zostanie lewym dzieckiem danego node'a

right_side – odnośnik do node, który zostanie prawym dzieckiem danego node'a

vertex – wierzchołek, który będzie reprezentowany przez dany node w grafie

NodeSegment:

Klasa reprezentująca odcinek w grafie przeszukiwania.

Atrybuty:

upper_side – trapez (ściana), znajdująca się bezpośrednio nad odcinkiem

lower_side – trapez (ściana), znajdująca się bezpośrednio pod odcinkiem

segment – odcinek reprezentowany przez dany node w grafie

Metody w klasie:

__init__(self, upper_side, lower_side, segment) – konstruktor klasy

Argumenty:

self – wskazanie na samego siebie

upper_side – odnośnik do node, który zostanie prawym dzieckiem danego node'a

lower_side – odnośnik do node, który zostanie lewym dzieckiem danego node'a

segment – odcinek, który będzie reprezentowany przez dany node w grafie

Zwraca:

Nic.

Graph:

Klasa reprezentująca graf przeszukiwań.

Atrybuty:

root – korzeń drzewa (graf jest DAGiem)

Metody w klasie:

__init__(self, root) – konstruktor klasy

Argumenty:

self – wskazanie na samego siebie

root – węzęł, który ma zostać korzeniem.

Zwraca:

Nic.

AddSegmentScene:

Klasa reprezentująca zbiór odcinków obecnych w mapie trapezowej w konkretnym kroku jej kreacji.

Atrybuty:

lines – zbiór odcinków obecnych w danym kroku tworzenia mapy trapezowej, które były obecne również w poprzednim kroku (domyślnie zawiera wszystkie odcinki przekazane do konstruktora)

new_lines – zbiór odcinków obecnych w danym kroku, które nie były obecne w poprzednim kroku (domyślnie puste)

deleted_lines – zbiór odcinków nieobecnych w danym kroku, które były obecne w poprzednim kroku (domyślnie puste)

Metody w klasie:

__init__(self, segments) – konstruktor klasy

Argumenty:

self – wskazanie na samego siebie

Segments – lista odcinków, które znajdują się w danym kroku tworzenia mapy trapezowej. Elementami listy są dwuelementowe listy, gdzie każdy z tych elementów reprezentuje jeden punkt na płaszczyźnie 2D. Punkty reprezentowane są przez dwuelementowe krotki zawierające liczby zmiennoprzecinkowe typu float.

Zwraca:

Nic.

set_all_segments_new(self) – ustawia wszystkie odcinki obecne w danym kroku jako nowe.

Argumenty:

self – wskazanie na samego siebie

Zwraca:

Nic.

compare_scenes(self, other) – Dokonuje porówania zbiorów odcinków pomiędzy dwoma krokami. Odcinki obecne w self a nieobecne w other ustawiane są jako nowe. Odcinki obecne w other, a nieobecne w self ustawiane są jako usunięte. Odcinki obecne w obu krokach ustawiane są w self jako obecne również w poprzednim (zbiór lines).

Argumenty

self – wskazanie na samego siebie

other – inny obiekt AddSegmentScene

Zwraca:

Nic.

generate_normal_scene(self) – tworzy obiekt typu Scene, zawierający wszystkie odcinki obecne w danym kroku.

Argumenty

self – wskazanie na samego siebie

Zwraca:

Obiekt typu Scene.

generate_deleted_scene(self) – tworzy obiekt typu scene, zawierający wszystkie odcinki, które były obecne w poprzednim kroku, nieobecne w danym kroku.

Argumenty

self – wskazanie na samego siebie

Zwraca:

Obiekt typu Scene.

generate_added_scene(self) – tworzy obiekt typu scene, zawierający wszystkie odcinki, które nie były obecne w poprzednim kroku, obecne w tym kroku oraz odcinki obecne w obu krokach.

Argumenty

self – wskazanie na samego siebie

Zwraca:

Obiekt typu Scene.

GraphSearchScene:

Klasa reprezentująca kolejne kroki przeszukiwania drzewa w postaci odnośników do aktualnie badanych w trakcie przeszukiwania Node'ów . Korzysta ona z ostatniej sceny z listy scen CREATE_SCENESET, która przez całe działanie programu reprezentuje aktualny stan mapy trapezowej. Obiekty tej klasy tworzone są podczas przeszukiwania grafu przeszukiwań w funkcji find in graph.

Atrybuty:

lines – zbiór odcinków oraz boków trapezów znajdujących się aktualnie w mapie trapezowej

root – węzeł w grafie, do którego odnosi się dany krok przeszukiwania (węzeł, w którym aktualnie jesteśmy)

searched_point – punkt, którego położenie określa algorytm

left – odnośnik do lewego dziecka roota w grafie (jeśli root jest AreaVertex to left jest None)

right – odnośnik do prawego dziecka roota w grafie (analogicznie)

Metody klasy:

__init__(self, root, searched_point) – konstruktor klasy

Argumenty:

Self – wskaźnik na samego siebie

root – węzeł w grafie, w którym aktualnie znajdujemy się podczas przeszukiwania

searched_point – punkt, którego trapezu szukamy w grafie

Zwraca:

Nic.

generate_node_view(self, choose) – zwraca reprezentację element u przypiętego do danego node'a (lub jednego z jego dzieci) w postaci zbioru punktów i linii kompatybilnych z LinesCollection. W całym programie funkcja używana jest jedynie przez metodę create_scene(self) należącą do tej samej klasy.

Argumenty:

self – wskazanie na samego siebie

choose – liczba całkowita, implikuje z jakiego node'a zostanie pobrany element do wizualizacji (0 – self.root, > 0 , self.right, < 0 self.left)

Zwraca:

Listę punktów oraz listę linii składających się na konkretny element.

create_scene(self) – generuje i zwraca obiekt Scene na który składają się: mapa trapezowa oraz aktualnie badany element i jego dzieci (po wyświetleniu sceny elementy te przyjmują kolory odpowiednio żółty, czerwony i zielony (lewe i prawe dziecko)).

Argumenty:

self – wskazanie na samego siebie

Zwraca:

Obiekt klasy Scene.

Funkcje

find_in_graph(root, point) – znajduje ścianę, do której należy przekazany jako argument punkt w grafie przeszukiwań. Funkcja zapisuje również do globalnej tablicy kolejne etapy przeszukiwania grafu.

Argumenty:

root – wskazanie na korzeń drzewa, które ma zostać przeszukane

point – Obiekt klasy point. Reprezentuje punkt, dla którego obszar w mapie trapezowej chcemy znaleźć.

Zwraca:

Obiekt typu Trapezoid.

replace_in_node(delta_node, new_node) – funkcja podmienia obszar w grafie (NodeArea) na wierzchołki, przez które został zastąpiony dla wszystkich jego rodziców.

Argumenty:

delta_node – stary obszar w grafie

new_node – wierzchołki, przez który stary obszar został zastąpiony

Zwraca:

Nic.

make_new_area_node(delta) – generuje nowy NodeArea dla przekazanego trapezu.

Argumenty:

delta – obiekt typu Trapezoid, trapez, który zostanie podpięty pod nowoutworzony Node.

Zwraca:

Obiekt typu NodeArea.

search_graph_dfs(root) – funkcja przechodzi przez graf przy użyciu algorytmu DFS, zwraca zbiór odcinków (zarówno zwykłych, jak i pionowych ścian trapezów) w postaci obiektów typu Segment (obiekty typu segment posiadające dwa punkty o równych x tworzone są tylko na potrzeby wizualizacji i tylko wewnątrz tej funkcji, nie są obecne podczas tworzenia mapy trapezowej).

Argumenty:

root – wskazanie wierzchołek w grafie, z którego ma się rozpocząć przeszukiwanie

Zwraca:

Zbiór obiektów Segment.

creation_update(graph) – tworzy obiekt typu AddSegmentScene na podstawie elementów obecnych w grafie przeszukiwania. Funkcja służy do generowania wizualizacji kolejnych kroków listy trapezowej. Obiekt ten zostaje dodany do globalnej listy CREATE_SCENESET().

Argumenty:

graph – graf, który ma zostać przeszukany

Zwraca:

Nic.

construct_map() – funkcja dokonuje budowy mapy trapezowej na podstawie wczytanych przez użytkownika odcinków (w przypadku braku wczytania odcinków bądź błędnej listy odcinków funkcja może działać błędnie lub nie działać wcale). Funkcja zwraca graf przeszukiwań.

Argumenty: brak.

Zwraca:

Obiekt typu Graph.

get_input_lines() – odpowiada za pobranie wprowadzonych przez użytkownika odcinków

Argumenty:

Brak.

Zwraca:

Listę odcinków w postaci listy list reprezentujących pojedyncze odcinki.

convert_to_segments(tuples_list) – zamienia listę krotek na listę obiektów typu segment (konwersja odcinków)

Argumenty:

tuples_list – lista odcinków w postaci krotek

Zwraca:

Listę obiektów typu Segment.

get_delta_to_delete(all_graph, segment) – funkcja znajduje i zwraca listę wszystkich trapezów na mapie trapezowej, które przecina przekazany do funkcji odcinek

Argumenty:

graph – graf przeszukiwania

segment – odcinek, dla którego chcemy znaleźć przecinające się z nim obszary

Zwraca:

Listę obiektów typu Trapezoid.

update_T(all_delta, segment) – dokonuje aktualizacji grafu przeszukiwań oraz mapy trapezowej przy wstawieniu nowego odcinka.

Argumenty:

all_delta – lista wszystkich odcinków, które przecina odcinek

segment – odcinek, który wstawiamy do mapy i do grafu

Zwraca:

Nic.

make_main_area() – funkcja inicjalizuje strukturę mapy trapezowej

Argumenty:

Brak.

Zwraca:

Obiekt typu Trapezoid. Pierwszy element w mapie trapezowej.

update_one_delta(old_delta, segment, left_delta, right_delta, upper_delta, lower_delta) – dokonuje rozdzielenia pojedynczego trapezu, który w pełni zawiera nowo dodany odcinek na cztery trapezy

Argumenty:

old_delta – trapez do rozdzielenia

segment – odcinek zawierający się w trapezie

left_delta – trapez na lewo od odcinka

right_delta – trapez na prawo od odcinka

upper_delta – trapez nad odcinkiem

lower_delta – trapez pod odcinkiem

Zwraca:

Nic.

replace_left_node(old_delta, left, upper, lower, segment) – podmień w grafie trapez, który zawiera lewy koniec odcinka

Argumenty:

old_delta – trapez przed podziałem

left – trapez po lewej stronie po podziale trapezu old_delta

upper – trapez nad odcinkiem po podziale trapezu old_delta

lower – trapez nad odcinkiem po podziale trapezu old_delta

segment – odcinek, który podzielił trapez

Zwraca:

Nic.

replace_right_node(old_delta, right, upper, lower, segment) – podmień w grafie trapez, który zawiera prawy koniec odcinka

Argumenty:

old_delta – trapez przed podziałem

right – trapez po prawej stronie po podziale trapezu old_delta

upper – trapez nad odcinkiem po podziale trapezu old_delta

lower – trapez nad odcinkiem po podziale trapezu old_delta

segment – odcinek, który podzielił trapez

Zwraca:

Nic.

replace_mid_node(old_delta, upper, lower, segment) - podmień w grafie trapez, który zawiera środek odcinka (nie zawiera lewego ani prawego końca)

Argumenty:

old_delta – trapez przed podziałem

upper – trapez nad odcinkiem po podziale trapezu old_delta

lower – trapez nad odcinkiem po podziale trapezu old_delta

segment – odcinek, który podzielił trapez

Zwraca:

Nic.

Sprawozdanie

Opis problemu

Celem zadania było zaimplementowanie algorytmu wyznaczania położenia punktu (odszukiwania wielokąta zawierającego dany wierzchołek) na płaszczyźnie 2D przy podziale planarnym odcinków.

By algorytm wykonywał się efektywnie konieczne było skorzystanie z odpowiednich struktur danych.

Struktury jakie wykorzystaliśmy to mapa trapezowa T, reprezentowana przez strukturę powiązań sąsiedzkich konkretnych elementów (trapezów) oraz graf przeszukiwań. Zastosowany graf posiadał 3 rodzaje węzłów, NodeVertex, NodeSegment oraz NodeArea, reprezentujące odpowiednio wierzchołek, odcinek oraz ścianę (trapez). Każdy węzeł w grafie posiadał dowiązania do odpowiadającego mu elementu na mapie trapezowej. Węzły NodeVertex oraz NodeSegment zawsze posiadały dokładnie dwójkę dzieci (wierzchołki wewnętrzne), natomiast węzły NodeArea nie posiadały dzieci, będąc zawsze liściami.

Istotną częścią zadania było poprawne identyfikowanie sąsiadów każdego trapezu, ze względu na zastosowanie położenia ogólnego odcinków każdy trapez mógł zawierać maksymalnie 4 sąsiadów. Mimo tak niewielkiej ilości sąsiadów istniało dużo konfiguracji ich występowania, co należało odpowiednio rozważyć przy dodawaniu kolejnych odcinków.

Testy

Aby zweryfikować poprawność działania algorytmu przeprowadzono szereg testów weryfikujących działanie programu. Testy te miały szereg zadań, m.in. weryfikację poprawności tworzenia nowych trapezów, weryfikację poprawności tworzenia i edytowania mapy trapezowej oraz grafu przeszukiwania. Testy były wprowadzane przez interfejs graficzny użytkownika, zawierały różne konfiguracje odcinków o różnym stopniu złożoności.

Wyniki oraz wnioski

Algorytm dokonuje poprawnego określenia pozycji zadanego punktu w podziale planarnym, co zweryfikowano z pomocą dołączonej do algorytmu wizualizacji kolejnych etapów działania algorytmu (zarówno tworzenia samej mapy trapezowej jak i wyszukiwania na niej zadanych wierzchołków). Sama implementacja również wpisuje się w założenia co do użytej pamięci oraz złożoności obliczeniowej algorytmu (odpowiednio O(n) oraz O(nlogn) dla tworzenia mapy trapezowej oraz O(logn) dla lokalizacji punktu).