**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. Poniżej znajduje się kilka przeprowadzonych testów oraz wynik (wizualizacja) jaki otrzymano.

TUTAJ BĘDZIE SPOKOJNIE

**Dokumentacja**

1. **Część użytkowa**
2. **Część techniczna**
3. 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 odcinek. Jeżeli odcinki mają, takie same parametry x i y to posiadają ten sam ciąg znaków.

\_\_eq\_\_(self, other) – metoda porównywująca 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 punkty 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

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

1. Opisy funkcji

**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).