Lokalizacja punktu w przestrzeni dwuwymiarowej – metoda trapezowa

Krzysztof Kwiecień
Paweł Żurawski

2 Styczeń 2021

1. Wprowadzenie

W ramach projektu przygotowano struktury reprezentujące mapę trapezową oraz graf wyszukiwania dla danego obszaru z podziałem poligonowym. Struktura mapy trapezowej pozwala dla zadanego punktu P określić jego położenie na płaszczyźnie dwuwymiarowej. Zaimplementowany program ma służyć jako narzędzie dydaktyczne, które za pomocą interfejsu graficznego wyświetla poszczególne kroki tworzenia mapy trapezowej.

2. Dokumentacja:

I. Część użytkownika

1. Ogólne informacje o programie

Program znajduje się w pliku .ipynb i jest możliwy do uruchomienia przez program jupyter notebook. Dodatkowo do programu dołączona jest prezentacja ilustrująca algorytm zawarty w programie.
Punkty w programie prezentowane są poprzez dwie liczby x, y.
Odcinki zawierają dwa punkty pokazujące początek i koniec odcinka

2. Reprezentacja linii w programie

Program jako parametr wejściowy przyjmuje zbiór odcinków $S = \{s_1, s_2, s_3, ..., s_n\}$
w położeniu ogólnym. Odcinki mogą być wczytywanie do programuna trzy sposoby:

- ręczne wprowadzanie odcinków
- pobieranie narysowanych odcinków z wykresu
- wczytywanie odcinków z pliku .json

3. Sposób uruchomienia programu

Aby uruchomić program należy skompilować po kolei wszystkie moduły w pliku main.ipynb oraz w funkcji main należy podać zbiór odcinków dla którego algorytm ma się wykonać.

4. Przykłady uruchomienia programu

Ręczne wprowadzanie odcinków:
Należy uruchomić komórkę zatytułowaną “Odczyt odcinków z płótna, zapis do pliku”, następnie wybrać opcję “Dodaj linię”, za pomocą myszki można dodawać kolejne odcinki na płótnie. Po zakończeniu należy uruchomić następną komórkę, odpowiedzialną za zapis odcinków do pliku. Jego nazwę można zmienić wewnątrz funkcji open.

Generowanie odcinków:
Należy uruchomić wszystkie komórki pod nagłówkiem “Generator odcinków”, następnie w komórce “Generacja odcinków, zapis do pliku” jako parametry funkcji generateLines przekazujemy kolejno - liczbę odcinków, minimalne oraz maksymalne wartości współrzędnych dla generowanych odcinków, np.
generateLines(10, 0, 0, 100, 100) - oznacza, że chcemy wygenerować 10 odcinków wewnątrz prostokąta wyznaczonego przez punkty (0, 0) i (100, 100).
W tej samej komórce odbywa się zapis do pliku, którego nazwę można zmienić wewnątrz funkcji open.

Wczytywanie odcinków z pliku:
Aby wczytać i wyświetlić odcinki należy uruchomić komórkę zatytułowaną “Odczyt linii z pliku”

Tworzenie mapy trapezowej i wyświetlanie:
Należy uruchomić komórkę zatytułowaną “Tworzenie mapy, wizualizacja” przy wcześniejszym wczytaniu odcinków z pliku

Poszukiwanie trapezu, w którym znajduje się punkt:
Należy uruchomić komórkę zatytułowaną “Poszukiwanie punktu” przy wcześniejszym utworzeniu mapy trapezowej. Współrzędne punktu należy podać wewnątrz wywołania Point().
np. point - Point(-20, 10) - punkt o współrzędnych (-20, 10)

II. Część techniczna

1. Spis modułów w programie:

- numpy - moduł odpowiadający za operacje na tablicach
- matplotlib - moduł odpowiadający za obsługę interfejsów wykresów
- json - moduł pozwalający na obsługiwanie plików w formacie .json
- random - moduł pozwalający wygenerować losowe liczby

2. Opis klas (struktur danych) w programie:

a) Point

Moduł implementuje funkcje:

- __init__(self,x,y):

  Argumenty:

  - x - współrzędna reprezentująca punkt na osi x
  - y - współrzędna reprezentująca punkt na osi y

  Wartość zwracana: brak

  Opis: Funkcja jest konstruktorem klasy i przypisuje współrzędne liczbowe obiektowi

Przechowywane dane

- x - współrzędna reprezentująca punkt na osi x
- y - współrzędna reprezentująca punkt na osi y

b) Segment

Moduł implementuje funkcje:

- __init__(self,p,q):

  Argumenty:

  - p - punkt reprezentujący lewy koniec odcinka
  - q - prawy reprezentujący koniec odcinka

  Wartość zwracana: brak

  Opis:

  Funkcja jest konstruktorem klasy i przypisuje odcinkowi lewy i prawy koniec. Funkcja weryfikuje czy współrzędna x-owa punktu p jest mniejsza od współrzędnej x-owej punktu q. Jeśli tak to lewy koniec jest równy punktowi p, a prawy koniec jest równy punktowi q. W przeciwnym wypadku końce są zamieniane. Lewy koniec odcinka zawsze ma mniejszą współrzędną x od prawego końca.

- isPointAbove (self,point):

  Argumenty:

  - point - punkt, który badany jest pod kątem położenia powyżej prostej

  Wartość zwracana: wartość logiczna True/False

  Opis:

  Gdy punkt leży powyżej prostej zwracana jest wartość True. W przeciwnym zwraca wartość to False

- isPointAbove (self,x):

  Argumenty:

  - x - punkt, dla którego określamy wartość y danego odcinka

  Wartość zwracana: współrzędna y dla punkt x

  Opis:

  Jeśli x zawiera się pomiędzy początkiem i końcem prostej, wtedy funkcja wylicza dla niego wartość y w punkcie x. Gdy x nie zawiera się w prostej zwracana wartość to None

Przechowywane dane

- leftPoint - współrzędna reprezentująca lewy koniec odcinka
- rightPoint - współrzędna reprezentująca punkt na osi y
- slope - współczynnik nachylenia prostej
- const - stała prostej

c) Trapezoid

Moduł implementuje funkcje:

- __init__(self,topSegment,bottomSegment, leftPoint, rightPoint):

  Argumenty:

  - topSegment - odcinek reprezentujący górny bok trapezu
  - bottomSegment - odcinek reprezentujący dolny bok trapezu
  - leftPoint - wierzchołek reprezentujący lewy bok trapezu
  - rightPoint - wierzchołek reprezentujący prawy bok trapezu

  Wartość zwracana: brak

  Opis: funkcja nie zwraca wartości

  Funkcja inicjalizuje trapez, poprzez przypisanie mu odcinków i wierzchołków reprezentujących go w przestrzeni 2D. Dodatkowo funkcja ustawia jego wszystkich sąsiadów na None oraz ustawia odwołanie do grafu wyszukiwania na None

Przechowywane dane

- topSegment - górny bok trapezu
- bottomSegment - dolny bok trapezu
- leftPoint - wierzchołek leżący na lewym boku trapezu
- rightPoint - wierzchołek leżący na prawym boku trapezu
- topLeft - lewy górny sąsiad
- bottomLeft - lewy dolny sąsiad
- topRight - prawy górny sąsiad
- bottomRight - prawy górny sąsiad
- node - odwołanie do liścia (trapezu) w grafie wyszukiwania

d) SearchGraph

Moduł implementuje funkcje:

- __init__(self,root):

  Argumenty:

  root - węzeł będący korzeniem grafu

  Wartość zwracana: brak

  Opis: Funkcja jest konstruktorem klasy

- updateRoot(self,root):

  Argumenty:

  root - węzeł będący nowym korzeniem grafu

  Wartość zwracana: brak

  Opis: Funkcja służy aktualizacji korzenia grafu

- query(self,point,node):

  Argumenty:

  point - punkt, którego lokalizacji poszukujemy
  node - węzeł, który chcemy rozważyć

  Wartość zwracana: trapez w którym znajduje się poszukiwany punkt

  Opis: Rekurencyjna funkcja służąca znalezieniu trapezu w którym zawiera się punkt.

Przechowywane dane

- root - węzeł będący nowym korzeniem grafu

e) XNode

Moduł implementuje funkcje:

- __init__(self,point,left,right):

  Argumenty:

  point - punkt związany z węzłem
  left - węzeł po lewej stronie
  right - węzeł po prawej stronie

  Wartość zwracana: funkcja nie zwraca wartości

  Opis: Funkcja jest konstruktorem klasy, w celu przypisania podanych węzłów wykonuje funkcje setLeft oraz setRight

- setLeft(self,node):

  Argumenty:

  left - nowy węzeł po lewej stronie

  Wartość zwracana: brak

  Opis: Funkcja służy aktualizacji lewego dziecka węzła, dodatkowo pod dodaniu t-węzła aktualizuje jego listę rodziców, tak aby zawierała ten węzeł

- setRight(self,node):

  Argumenty:

  right - nowy węzeł po prawej stronie

  Wartość zwracana: brak

  Opis: Funkcja służy aktualizacji prawego dziecka węzła, dodatkowo pod dodaniu t-węzła aktualizuje jego listę rodziców, tak aby zawierała ten węzeł

Przechowywane dane
- type - typ węzła, dla x-węzła jest to ‘xnode’
- point - punkt związany z węzłem
- left - węzeł po lewej stronie
- right - węzeł po prawej stronie

f) YNode

Moduł implementuje funkcje:

- __init__(self,point,above,below):

  Argumenty:

  segment - odcinek związany z węzłem

  above - węzeł prowadzący powyżej odcinek

  below - węzeł prowadzący poniżej odcinek

  Wartość zwracana: brak

  Opis: Funkcja jest konstruktorem klasy, w celu przypisania podanych węzłów wykonuje funkcje setAbove oraz setBelow

- setAbove(self,node):

  Argumenty:

  above - nowy węzeł prowadzący powyżej odcinek

  Wartość zwracana: brak

  Opis: Funkcja służy aktualizacji dziecka węzła prowadzącego powyżej odcinek, dodatkowo pod dodaniu t-węzła aktualizuje jego listę rodziców, tak aby zawierała ten węzeł

- setBelow(self,node):

  Argumenty:

  below - nowy węzeł prowadzący poniżej odcinek

  Wartość zwracana: brak

  Opis: Funkcja służy aktualizacji dziecka węzła prowadzącego poniżej odcinek, dodatkowo pod dodaniu t-węzła aktualizuje jego listę rodziców, tak aby zawierała ten węzeł

Przechowywane dane
- type - typ węzła, dla y-węzła jest to ‘ynode’
- point - punkt związany z węzłem
- above - węzeł prowadzący powyżej odcinek
- below - węzeł prowadzący poniżej odcinek

g) TNode

Moduł implementuje funkcje:

- __init__(self,trapezoid):

  Argumenty:

  trapezoid - trapez związany z węzłem

  Wartość zwracana: brak

  Opis: Funkcja jest konstruktorem klasy, przypisuje danemu trapezowi siebie jako powiązany z nim węzeł

- replaceNode(self,searchGraph,node):

  Argumenty:

  searchGraph - graf wyszukiwania
  node - węzeł mający zastąpić obecny

  Wartość zwracana: brak

  Opis: Funkcja aktualizuje wszystkich rodziców tego węzła aby zamiast na niego wskazywały na nowo podany węzeł, dzięki czemu jest on przez niego zastępowany w strukturze D

Przechowywane dane
- type - typ węzła, dla t-węzła jest to ‘tnode’
- trapezoid - trapez związany z węzłem
- parents - lista węzłów będących rodzicami tego węzła

3. **Opis funkcji w programie:**

   a) **permutate(S)**

   Argumenty:

   S - zbiór odcinków reprezentowanych jako obiekty klasy Segment

   Wartość zwracana: lista odcinków S

   Opis: Funkcja zwraca losową permutacja odcinków

   b) **updateLeftNeighbors(oldTrapezoid, newTrapezoid):**

   Argumenty:

   oldTrapezoid - trapez usuwany z mapy trapezowej
   newTrapezoid - trapez wstawiany do mapy trapezowej

   Wartość zwracana: brak

   Opis: Funkcja aktualizuje odwołania lewych sąsiadów starego trapezu tak, aby wskazywały na nowego

c) **updateRightNeighbors(oldTrapezoid, newTrapezoid):**

Argumenty:

oldTrapezoid - trapez usuwany z mapy trapezowej

newTrapezoid - trapez wstawiany do mapy trapezowej

Wartość zwracana: brak

Opis: Funkcja aktualizuje odwołania prawych sąsiadów starego trapezu tak, aby wskazywały na nowego

d) **findIntersectingTrapezoids(startTrapezoid, endTrapezoid, segment, intersectingTrapezoids):**

Argumenty:

startTrapezoid - trapez będący początkiem strefy

endTrapezoid - trapez będący końcem strefy

segment - wstawiany odcinek
intersectingTrapezoids - tablica, która zostanie wypełniona przecinanymi trapezami

Wartość zwracana: brak

Opis: Funkcja znajduje wszystkie trapezy przecinane przez dany odcinek. Znajdowane są one z wykorzystaniem relacji sąsiedzkich, od lewej do prawej. Podana na wejściu lista wypełniania jest kolejno znajdowanymi trapezami.

e) **main(lines)**

Argumenty:

lines - lista nieprzetworzonych odcinków (odczytanych z pliku)

Wartość zwracana: graf wyszukiwania oraz lista scen do wizualizacji

Opis: Funkcja tworzy mapę trapezową oraz graf wyszukiwania. Na początku tworzy pierwotny trapez (obejmujący wszystkie odcinki), następnie dodaje pojedynczo kolejne segmenty do mapy i aktualizuje ją oraz strukturę D.

f) **insertSegmentInOneTrapezoid(searchGraph, trapezoid, segment)**

Argumenty:

searchGraph - graf wyszukiwania

trapezoid - trapez, do którego wstawiamy odcinek

segment - wstawiany odcinek

Wartość zwracana: lista dodanych trapezów

Opis: Funkcja dodaje odcinek zawierający się całkowicie w istniejącym trapezie do mapy, aktualizuje przy tym wszystkie powiązania oraz strukturę D.

g) **insertSegmentInManyTrapezoids(searchGraph, intersectingTrapezoids, segment)**

Argumenty:

searchGraph - graf wyszukiwania

intersectingTrapezoids - lista trapezów, które przecina wstawiany odcinek

segment - wstawiany odcinek

Wartość zwracana: lista dodanych trapezów

Opis: Funkcja dodaje odcinek do mapy, aktualizuje przy tym wszystkie powiązania oraz strukturę D. Wewnętrznie rozważa wszystkie możliwe przypadki położenia odcinka względem przecinanych trapezów.

h) **visualise((segment, addedTrapezoidsLines, allLinesAdded, sphereForSegment, addedTrapezoids, scenes):**

Argumenty:

segment - Obiekt typu Segment (odcinek)

addedTrapezoidsLines - lista krawędzi wszystkich dodanych trapezów
allLinesAdded - lista krawędzi dodanych trapezów dla strefy danego odcinka
sphereForSegment - lista odcinków reprezentujący strefę dla odcinka
addedTrapezoids - lista dodanych trapezów do strefy
scenes - lista scen

Wartość zwracana: lista krawędzi wszystkich dodanych trapezów

Opis: Funkcja przyjmuje szereg wejściowych tablic, które pozwalają na wizualizację krok po kroku działania algorytmu. Wszystkim tablicom przypisywane są różne kolory w wizualizacji, tak aby łatwo można rozróżnić reprezentujące przez nie zbiory danych.

i) **minAndMaxFromLines(lines):**

Argumenty:

lines - zbiór odcinków

Wartość zwracana: tupla zawierająca 4 wartości liczbowe

Opis: Funkcja przyjmuje zbiór odcinków dla których wylicza minimalne i maksymalne wartości dla osi x i y. Dodatkowo funkcja dodaje obramowanie dla tych wartości wynoszące 10 % z różnicy pomiędzy wartościami minimalnymi i maksymalnymi z osi x i y.

j) **trapezoidLines(trapezoid):**

Argumenty:

trapezoid - konkretny jeden trapez

Wartość zwracana: zbiór linii reprezentujących graficznie trapez

Opis: Funkcja oblicza i przypisuje do wynikowej listy zestaw linii reprezentujących trapez, obliczanych na podstawie atrybutów argumentu trapezoid.

k) **cutVerticalLines(lines, trapeze):**

Argumenty:

lines - zbiór odcinków

trapeze - obiekt typu trapez

Wartość zwracana: zbiór linii niezawierających się w trapezie

Opis: Funkcja bada czy odcinki z argumentu lines zawierają się w trapezie. Jeśli się zawierają odpowiednio modyfikuje je, tak by nie zawierały się w badanym trapezie. Każdy odcinek nie zawierający się w trapezie oraz każdy zmodyfikowany odcinek zawierający się w trapezie umieszczany jest w liście result, która jest wartością zwracaną

l) **createOuterTrapezoid(lines):**

Argumenty:

lines - zbiór odcinków

Wartość zwracana: Obiekt typu Trapez

Opis: Funkcja inicjuje argumenty x1,x2,y1,y2 jako wynik działania funkcji minAndMaxFromLines(lines). Następnie tworzy odcinki które reprezentują dolny i górny bok trapezu oraz tworzy punkty reprezentujące lewy i prawy bok trapezu. Tak stworzony trapez powierzchniowo obejmuje wszystkie odcinki z argumentu wejściowego lines

m) **orient(A, B, C, epsilon=10**-10):**

Argumenty:

A - punkt o współrzędnych x i y

B - punkt o współrzędnych x i y

C - punkt o współrzędnych x i y

epsilon - tolerancja dla wyznacznika

Wartość zwracana: wartość liczbowa 1 lub 2

Opis: Funkcja zwraca wartość 1 gdy punkt C znajduje się w stronę przeciwną do ruchu wskazówek zegara (CCW) względem linii AB lub wartość 2 w pozostałych przypadkach (CW, COLL)

n) **intersect(line1, line2):**

Argumenty:

line1 - odcinek

line2 - odcinek

Wartość zwracana: wartość logiczna True/False

Opis: Funkcja sprawdza czy dane odcinki się przecinają

o) **generateLines(quantity=5, minX=0, minY=0, maxX=100, maxY=100):**

Argumenty:

quantity - ilość linii do wygenerowania
minX - minimalna wartość na osi X
minY - minimalna wartość na osi Y
maxX - maksymalna wartość na osi X
maxY - maksymalna wartość na osi Y

Wartość zwracana: lista odcinków

Opis: Funkcja generuje losowe odcinki w położeniu ogólnym

III. Sprawozdanie

1. Opis problemu

Projekt polegał na przygotowaniu implementacji algorytmu określania położenia punktu P w podziale poligonowym odcinków. Aby program wykonywał się efektywnie oraz pozwalał dotrzymać oczekiwanych czasów działanie algorytmu należało wybrać odpowiednie struktury danych.
W naszej implementacji użytko do tego mapy trapezowej T reprezentowanej jako struktury powiązań sąsiedzkich między trapezami oraz struktury przeszukiwań D jako grafu wyszukiwania. Graf wyszukiwania zawierał 3 rodzaje węzłów. Węzły wewnętrzne dzieliły się na dwa rodzaje:

- x-węzeł - reprezentujący obiekt typu punkt
- y-węzeł - reprezentujący odcinek

Liście natomiast zawierały tylko wyłącznie aktualnie istniejące trapezy,
z mapy trapezowej.
Zaimplementowane struktury były ze sobą wzajemnie powiązane, ponieważ trapez z T(S) ma wskaźnik do odpowiadającego mu liścia w D, a liść w D ma wskaźnik do odpowiadającemu mu trapezu w T(S).
Struktury musiały także poprawnie obsługiwać wydarzenia dla dowolnych odcinków zaczynających się w tych samych punktach. W takim przypadku niezbędne było prawidłowe określenie położenia punktu w strukturze przeszukiwań D.

W algorytmie należało także poprawnie określić sąsiedztwo trapezów. Dzięki położeniu ogólnym wierzchołków trapez mógł posiadać tylko 4 sąsiadów. Jednak mimo tak małej ilości sąsiadów, należało rozważyć kilka możliwości ich występowania. Przy usuwania trapezu i wstawianiu nowych trapez z wyznaczonej dla odcinka sfery należało poprawnie zaktualizować sąsiedzkie dowiązania trapezów oraz pamiętać aby skracać rozszerzenia pionowe trapezów, które wystawały ponad bok wstawianych trapezów.

2. Wykonane testy

W celu weryfikacji poprawności algorytmu określenia pozycji punktu p w podziale poligonowym, przeprowadzono szereg testów. Testy składały się z różnych zestawów odcinków (w położeniu ogólnym) o różnym stopniu trudności. Odcinki mogły być wprowadzenie na kilka sposobów. Mogły być generowane ręcznie lub przez generator. Generator umożliwia modyfikacje liczby odcinków oraz zakres obszaru występowania odcinków.

Testy miały za zadanie:

- przetestować poprawność układania węzłów w strukturze grafu wyszukiwania,
- sprawdzić poprawność konstruowania mapy trapezowej,
- przetestować różne przypadki mapy trapezowej, w celu określenia poprawności zaimplementowanej wizualizacji step-by-step algorytmu
- przetestować poprawność tworzenia nowych trapezów

3. Uzyskane wyniki

Zaimplementowany algorytm poprawnie określa pozycji punktu p w podziale planarnym. Posiada poprawną wizualizację przedstawiającą krok po kroku jego działanie oraz zwraca prawidłowe wartości dla wszystkich wykonanych testów. W implementacji udało się zachować wszystkie oszacowanie założenia odnośnie złożoności pamięciowej oraz czasowej algorytmu. Po zaprojektowaniu mapy trapezowej w czasie rzędu O (*nlogn*), wykorzystując graf wyszukiwań, możemy określić położenie dowolnego punktu w czasie rzędu O(*logn*).

Wydajność algorytmu została przetestowana dla różnych rozmiarów danych wejściowych. Wszystkie z nich zostały wygenerowane losowo. Podczas mierzenia czasu wyłączono całkowicie obsługę wizualizacji, aby nie wpływała ona na wyniki.

Uzyskane czasy tworzenia mapy trapezowej przedstawiono w tabeli poniżej oraz na wykresie.

Tabela 1. Czas trwania algorytmu w zależności od liczby odcinków

Ilość odcinków	Czas trwania algorytmu [s]
10	0.00099897
25	0.00200081
50	0.00399804
100	0.00599790
150	0.00899792
200	0.01399541
300	0.01999331
400	0.02999020
500	0.03498864
1000	0.09097195
1500	0.14195037
2000	0.18593645
3500	0.34288692
5000	0.50983167
10000	1.09464383
20000	2.24492788

Wykres 1. Czas trwania algorytmu w zależności od liczby odcinków