ALGORYTMY GEOMETRYCZNE – PROJEKT

Lokalizacja punktu w przestrzeni dwuwymiarowej

Metoda trapezowa

Marta Stanisławska

Mateusz Wójcicki

Spis treści

1. Wstęp
2. Wstawianie odcinków
   a. uaktualnienie sąsiadów
   b. uaktualnienie grafu przeszukiwań

Lokalizacja punktu na płaszczyźnie

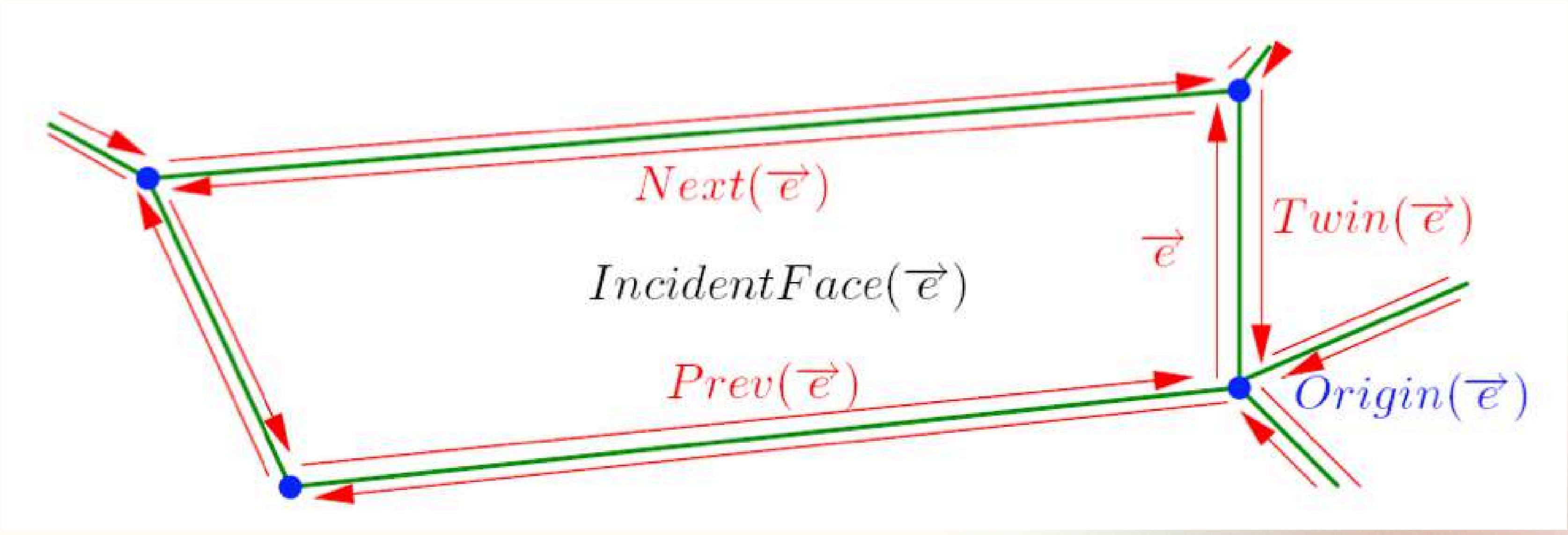
definicja ogólna

Dane: poligonowy podział płaszczyzny (podział planarny) S, zadany np. w postaci grafu krawędzi i wierzchołków, który należy przetworzyć, zapisując wyniki w odpowiedniej strukturze danych tak, aby umożliwić efektywne osiągnięcie celu.

Cel: odszukanie wielokąta (ściany) zawierającego zadany punkt.

Podział planarny

Struktura: podwójnie łączona lista krawędzi



Elementy:

- Wierzchołek:
  - współrzędne
  - incydenta krawędź
- Półkrawędź:
  - 3 krawędzie
  - wierzchołek
  - ściana
- Ściana:
  - półkrawędź z brzegu obszaru
  - półkrawędź z każdej ściany wewnątrz

Położenie ogólne

Założenia dla podziału planarnego

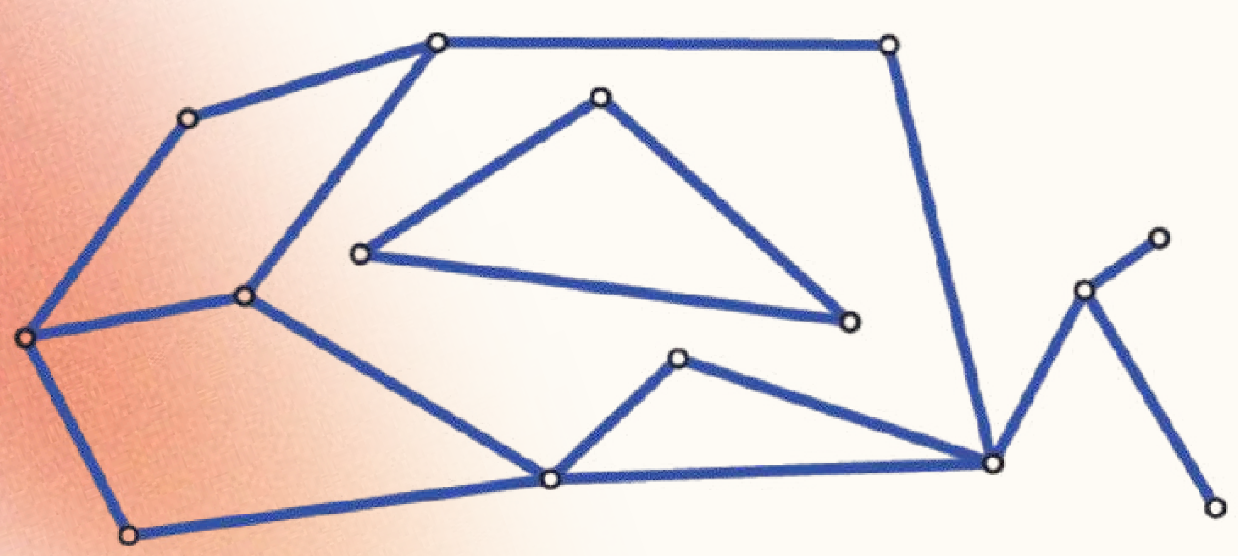
Żaden odcinek nie jest pionowy

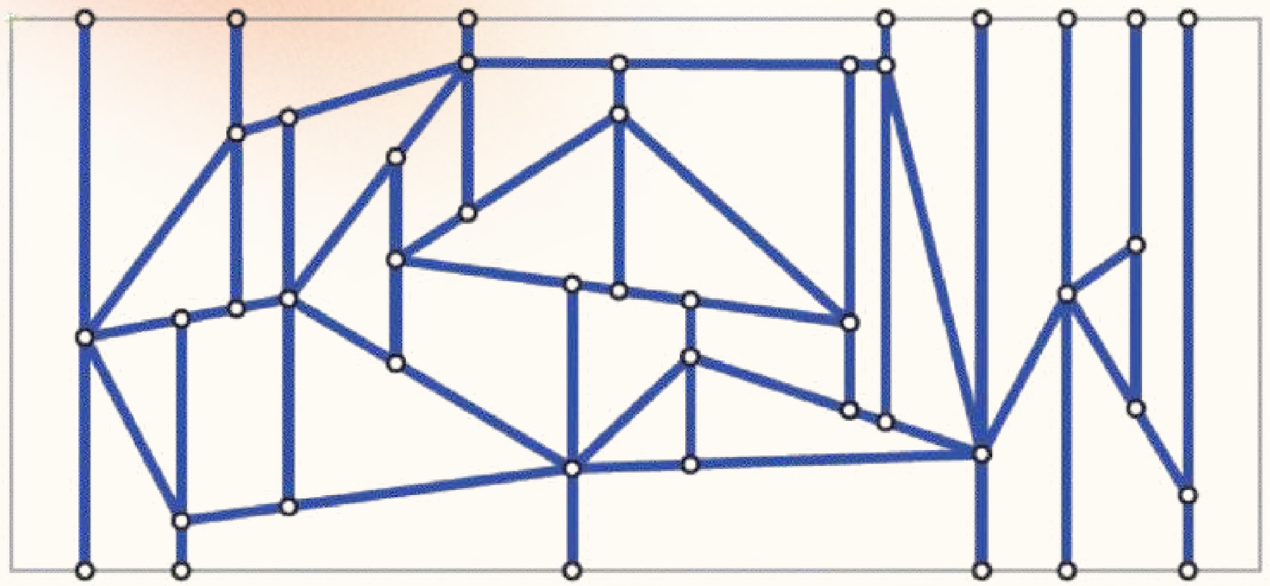
Wierzchołki żadnych dwóch odcinków nie mają takiej samej współrzędnej x

- poza końcami połączonych odcinków

Mapa trapezowa

- Jest to podział S na wielokąty wypukłe (trapezy lub trójkąty) otrzymany poprzez poprowadzenie dwóch rozszerzeń (odcinków) pionowych z każdego końca odcinka w S. Rozszerzenia kończą się, gdy napotkają inny odcinek S lub brzeg prostokąta.
- Mapa reprezentowana jest w postaci zbioru odcinków, przy czym żadne dwa odcinki nie przecinają się, poza ewentualnie wierzchołkami.

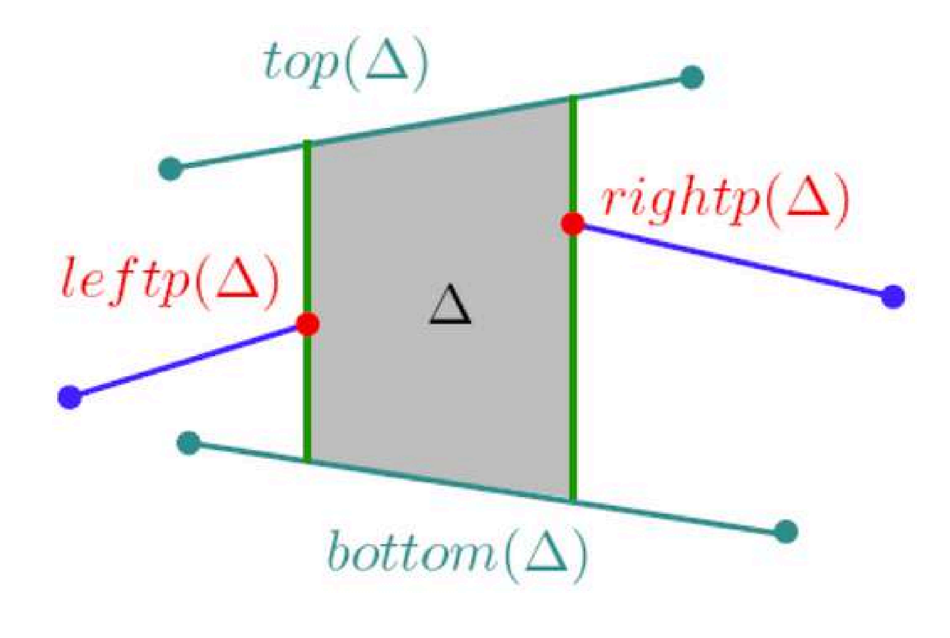




Mapa trapezowa

Trapez

Każdy trapez jest definiownany przez cztery elementy podziału

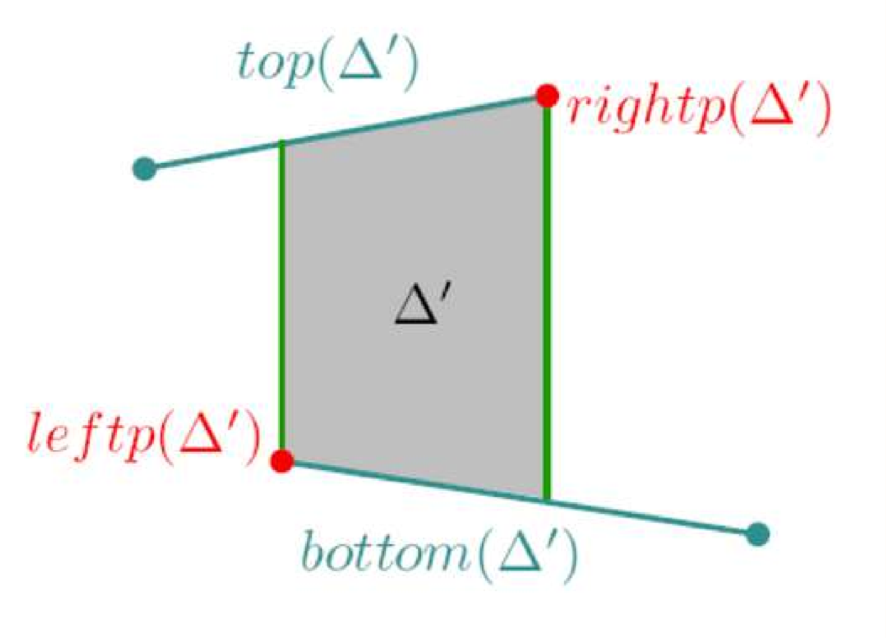


Odcinek dolny bottom(Δ)

Odcinek górny top(Δ)

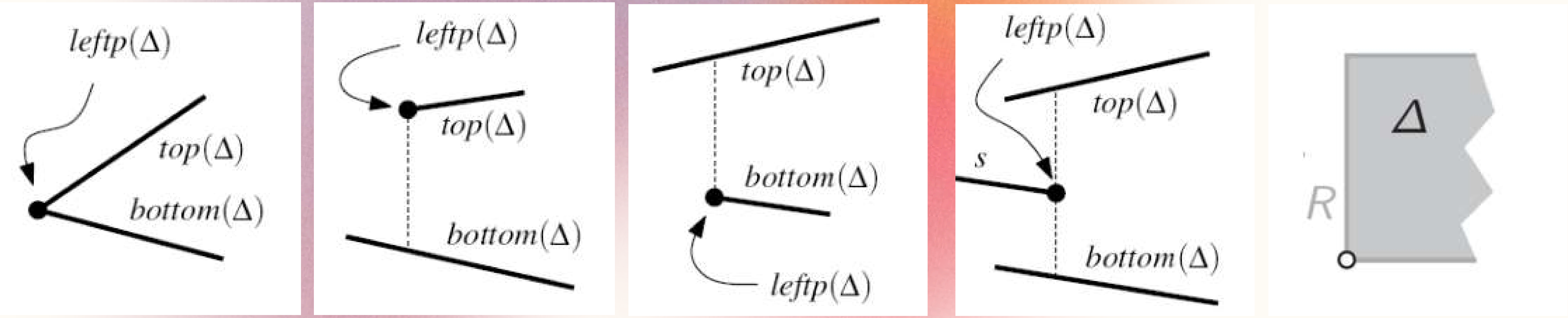
Wierzchołek lewy leftp(Δ)

Wierzchołek prawy rightp(Δ)

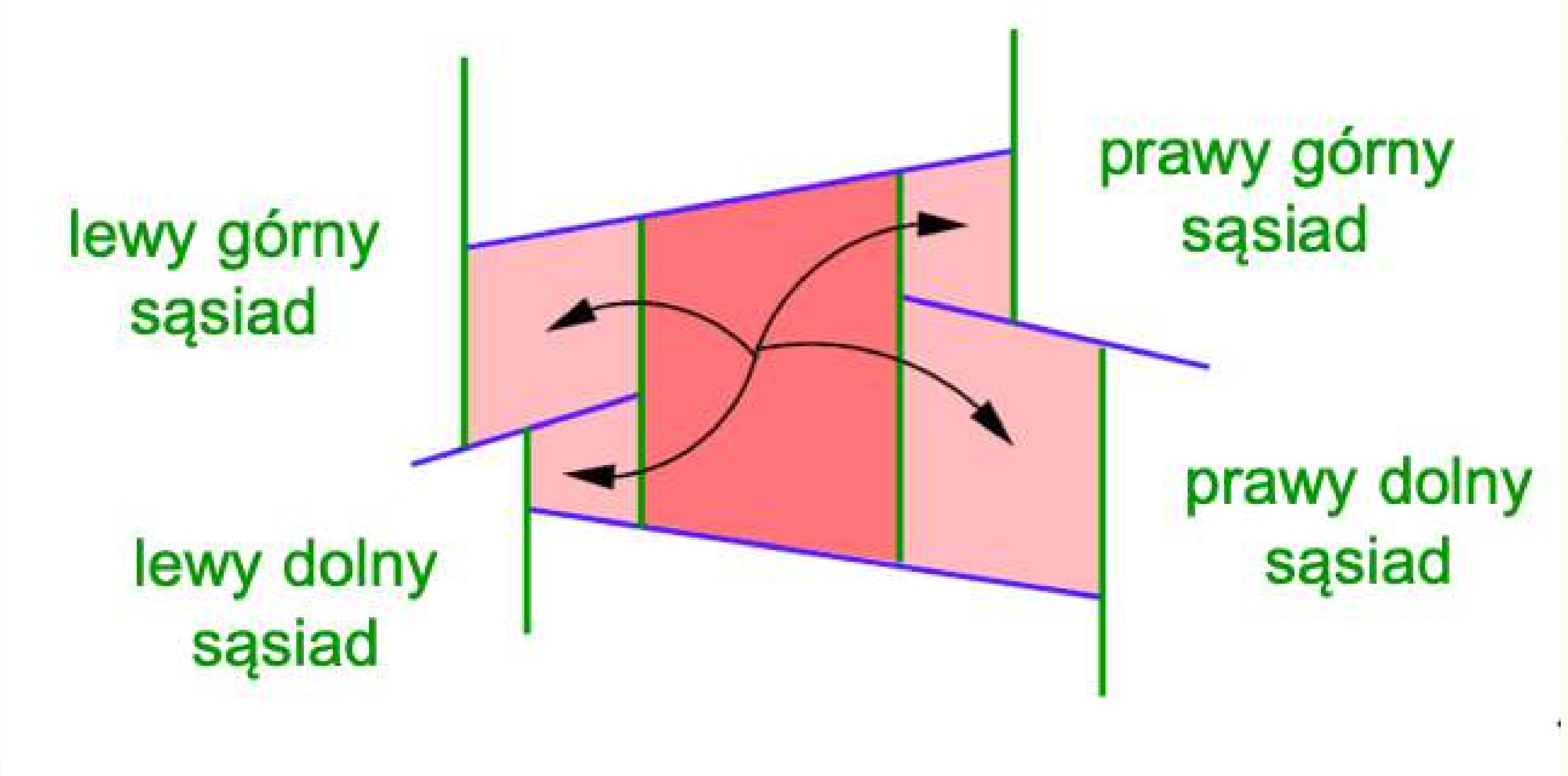


Mapa trapezowa

Możliwe przypadki dla lewego boku trapezu



Dla prawego boku trapezu przypadki są analogiczne.



Mapa trapezowa

Sąsiedztwo trapezów

Dwa trapezy są sąsiadami wtedy i tylko wtedy, gdy mają wspólną krawędź pionową.

- Dla podziału planarnego w położeniu ogólnym każdy trapez ma co najwyżej czterech sąsiadów.

Struktury danych użyte w algorytmie

- Reprezentuje punkt w przestrzeni 2D.
- Metoda __eq__ sprawdza, czy dwa punkty są równe (porównując ich współrzędne).
- Metoda __gt__ porównuje punkty na podstawie współrzędnej x.

POINT

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __gt__(self, other):
        return self.x >other.x

    def __hash__(self):
        return hash((self.x,self.y))
```

Struktury danych użyte w algorytmie

- Reprezentuje odcinek w przestrzeni 2D.
- Przyjmuje ona argumenty left_end i right_end, które odpowiadają dwóm skrajnym punktom odcinka.
- Konstruktor wylicza współczynniki a i b równania prostej y=ax+by = ax + b, jeśli odcinek nie jest pionowy.
- Metoda above_point sprawdza, czy dany punkt leży poniżej lub powyżej odcinka.
- Metoda get_point_at_x zwraca współrzędne punktu na odcinku dla podanego x.

SECTION

```
class Section:
    def __init__(self, left_end, right_end):

        self.L = left_end
        self.R = right_end

        if self.L.x == self.R.x: self.a, self.b = None, None
        else:
            self.a = (self.R.y - self.L.y) / (self.R.x - self.L.x)
            self.b = self.L.y - self.a * self.L.x


    def __hash__(self):
        return hash((self.L, self.R))

    def above_point(self,point):
        if not self.a or not self.b: return None
        return self.a * point.x + self.b > point.y

    def get_point_at_x(self, x):
        if not self.L.x <= x <= self.R.x: return None
        return Point(x, self.a * x + self.b)
```

Struktury danych użyte w algorytmie

- Reprezentuje trapez w przestrzeni 2D.
- Atrybuty left_p i right_p typu Point przechowują współrzędne punktów znajdujących się na lewej i prawej krawędzi trapezu.
- Atrybuty right_upper_neighbour, right_lower_neighbour, left_upper_neighbour i left_lower_neighbour typu Trapezoid odnoszą się do sąsiadujących trapezów
- Atrybut leaf zawiera odwołanie do wierzchołka w grafie przeszukiwań.

TRAPEZOID

```
class Trapezoid:
    def __init__(self, top, bottom, left_point, right_point):
        self.top = top
        self.bottom = bottom
        self.left_p = left_point
        self.right_p = right_point

        self.right_upper_neighbour = None
        self.right_lower_neighbour = None
        self.left_lower_neighbour = None
        self.left_upper_neighbour = None

        self.leaf = None
```

Struktury danych użyte w algorytmie

- Reprezentuje węzeł w grafie wyszukiwania.
- Atrybut type wskazuje typ węzła (leaf, xnode lub ynode).
- label przechowuje informację specyficzną dla typu węzła (Point, Section lub Trapezoid).
- Zawiera wskaźniki na dzieci węzła (left, right).

NODE

```python
class Node:
    def __init__(self, type, label):
        self.type = type
        self.label = label
        self.left = None
        self.right = None
```

Struktury danych użyte w algorytmie

- Reprezentuje graf wyszukiwania.
- Metoda find wyszukuje odpowiedni liść (trapez) w grafie dla danego punktu i obecnie badany odcinek w mapie trapezowej (nie jest używany, gdy mapa trapezowa jest już gotowa, a trapez jest poszukiwany).
- Logika wyszukiwania różni się w zależności od typu węzła (xnode, ynode, leaf).

SEARCHGRAPH

```python
class SearchGraph:
    def __init__(self, root):
        self.root=root

    def find(self, node, point, segment=None):

        if node.type == 'leaf': return node.label

        elif node.type == 'xnode':
            if point < node.label: return self.find(node.left, point, segment)
            else: return self.find(node.right, point, segment)

        else:
            if node.label.above_point(point): return self.find(node.right, point, segment)

            elif segment and node.label.get_point_at_x(segment.L.x) == point:
                if segment.a > node.label.a: return self.find(node.left,point,segment)
                else: return self.find(node.right,point,segment)

            else: return self.find(node.left,point,segment)
```

Algorytm konstrukcji mapy

Randomizowany przyrostowy algorytm konstrukcji T(S)

Dane wejściowe:

- zbiór odcinków S w położeniu ogólnym.

Wynik:

- mapa trapezowa T(S),
- struktura przeszukiwań D dla T(S).

Algorytm konstrukcji mapy

Wstęp

- Wyznaczenie losowej permutacji odcinków.

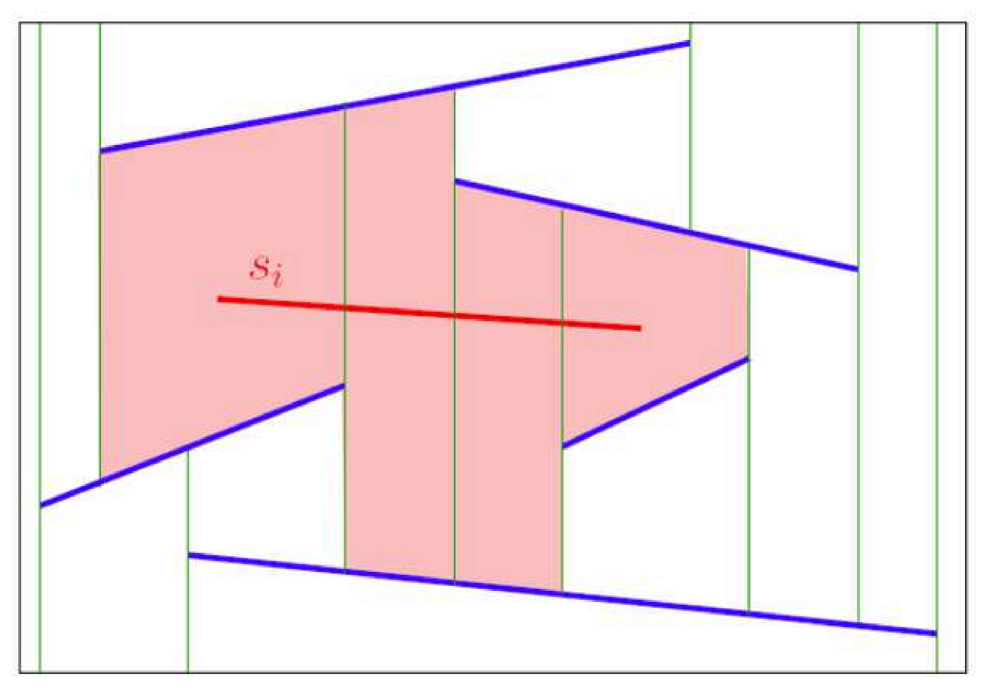
- Znalezienie prostokąta zewnętrznego:
  - wyznaczenie “ramki” na podstawie współrzędnych punktów o największej i najmniejszej współrzędnej x oraz największej i najmniejszej współrzędnej y.

- Inicjalizacja struktury danych dla znalezionego prostokąta zewnętrznego:
  - przypisanie wyznaczonego prostokąta zewnętrznego jako trapezu i węzła typu “Leaf”,
  - inicjalizacja grafu przeszukiwań poprzez przypisanie węzła prostokąta zewnętrznego jako korzenia.

Algorytm konstrukcji mapy

Wstawianie odcinka

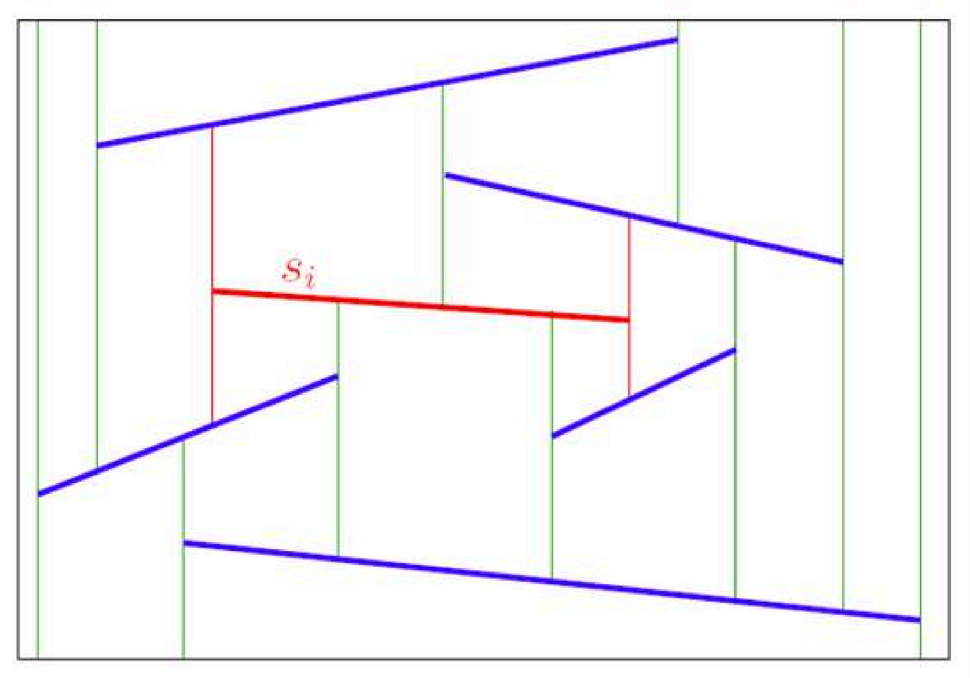
Głównym elementem algorytmu wyznaczania mapy jest pętla, w której każdy odcinek po kolei (zgodnie z kolejnością wyznaczoną przy permutacji) jest przetwarzany i powoduje modyfikację aktualnego stanu mapy.



Znajdź zbiór $\Delta_0, \Delta_1, \dots, \Delta_k$ trapezów przeciętych przez s_i.

Usuń $\Delta_0, \Delta_1, \dots, \Delta_k$ i zastąp je przez nowe trapezy.

Uaktualnij T_i oraz D_i.



- Gdy wstawiamy odcinek s_i, struktury T_{i-1}, D_{i-1} są już utworzone.

Algorytm konstrukcji mapy- wstawianie odcinka

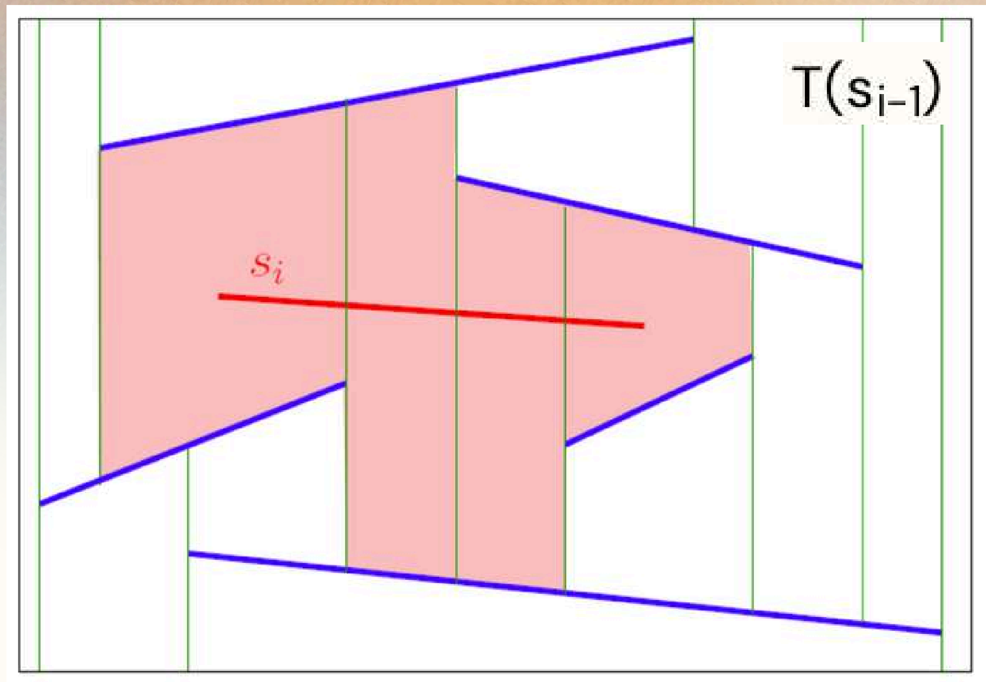
Wyznaczenie trapezów przeciętych przez wstawiany odcinek (strefa tego odcinka)

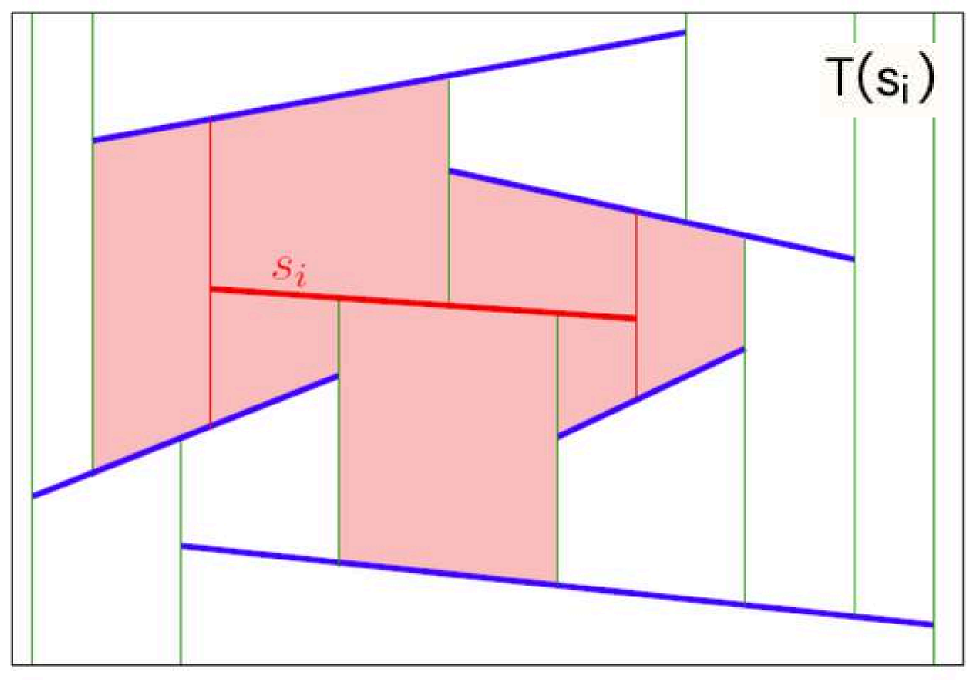
Strefę dla odcinka s_i w $T(s_{i-1})$ i $T(s_i)$ tworzą wszystkie trapezy przecinające s_i. Każdy taki trapez może zostać podzielony na maksymalnie cztery trapezy.

Dla $T(s_{i-1})$ jest to suma wszystkich trapezów, które zostaną usunięte.
Dla $T(s_i)$ jest to suma wszystkich trapezów, które zostaną stworzone.

$T(s_{i-1})$ i $T(s_i)$ są więc identyczne pod względem kształtu i rozmiaru. Różnią się jedynie podziałem na trapezy- ich ilością i rozmieszczeniem.

Na rysunkach obok, kolorem różowym zaznaczona została strefa dla wstawianego odcinka s_i.

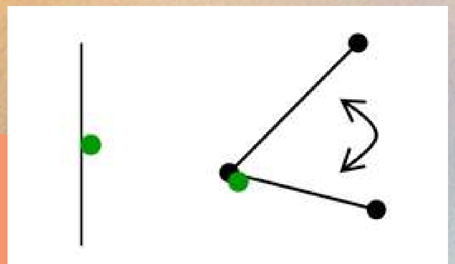




Algorytm konstrukcji mapy- wstawianie odcinka

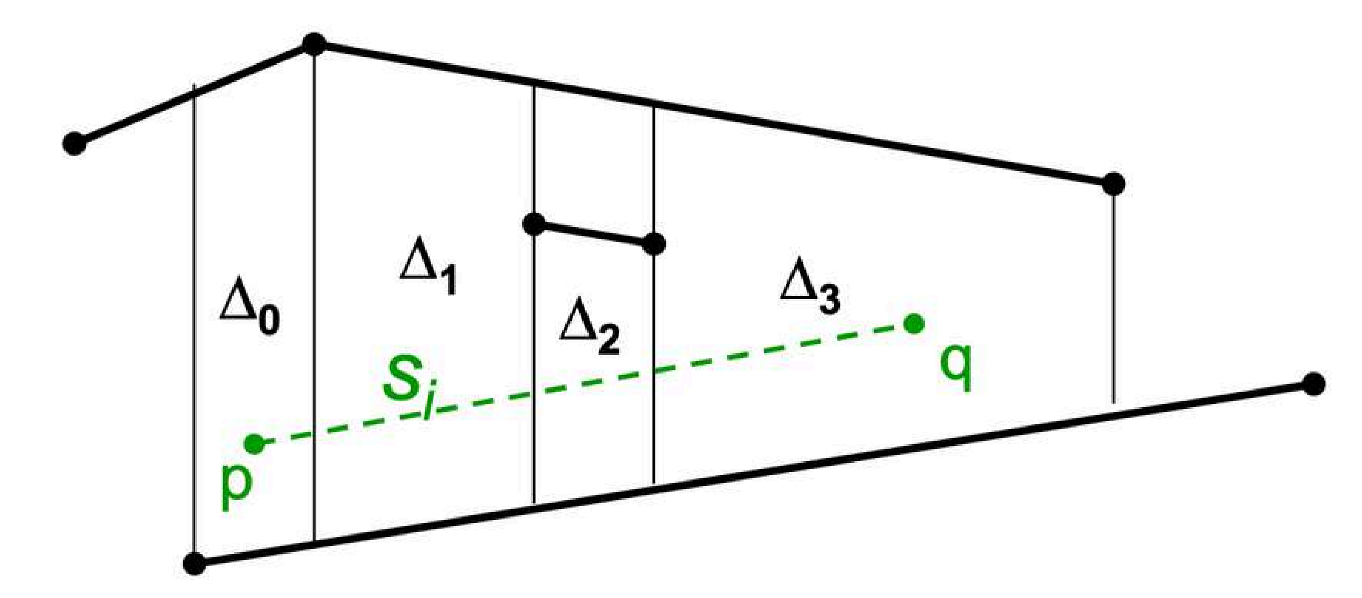
Wyznaczenie trapezów przeciętych przez wstawiany odcinek (strefa tego odcinka)

1. Znajdź w strukturze D trapez Δ zawierający lewy koniec odcinka s_i.
2. $j \leftarrow 0$
   while q leży na prawo od rightp(Δ_j)
      if si leży powyżej rightp(Δ_j)
         else niech Δ_{j+1} będzie dolnym prawym sąsiadem Δ_j
         then niech Δ_{j+1} będzie górnym prawym sąsiadem Δ_j
      $j \leftarrow j + 1$
   return $\Delta_0, \Delta_1, \dots, \Delta_k$



Uwaga!
Jeśli p jest już w strukturze:

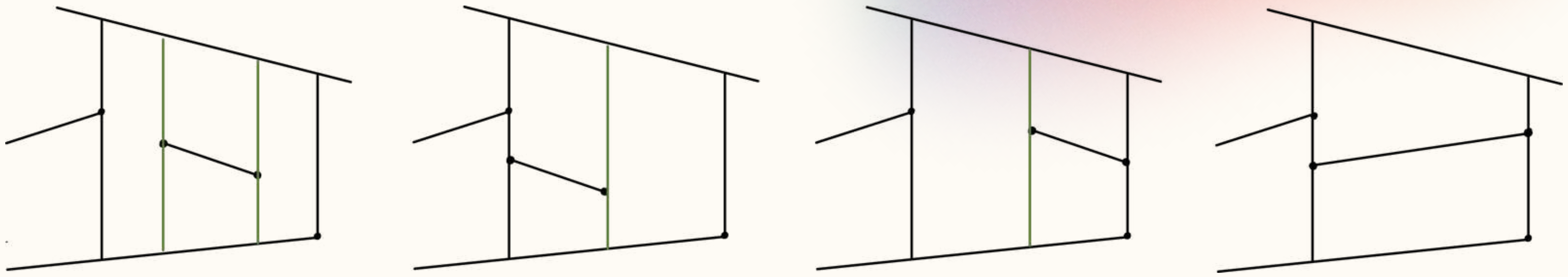
- jeśli p leży na prostej pionowej, to przyjmujemy, że leży po prawej stronie,
- jeśli p jest wspólnym początkiem z innym odcinkiem s, to jeśli nachylenie s_i jest mniejsze od nachylenia s, to p leży poniżej s.



Algorytm konstrukcji mapy- wstawianie odcinka

Przypadek 1. Wstawiany odcinek przecina tylko jeden trapez

Uaktualnienie sąsiadów



Ogólna idea postępowania:

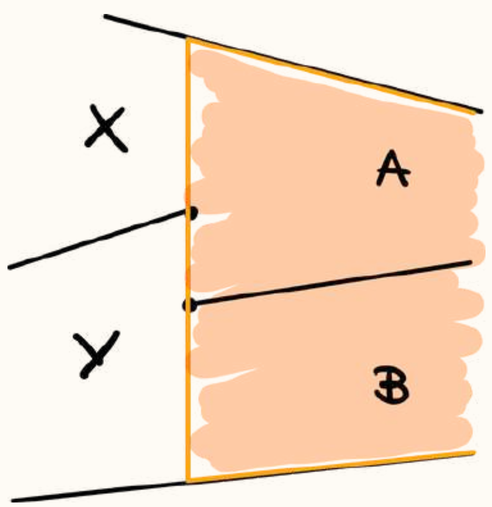
- usuwamy Δ z T,
- zastępujemy przez 2, 3 lub 4 trapezy,
- aktualizujemy informacje dla trapezów o sąsiadach, bottom(Δ), top(Δ) leftp(Δ), rightp(Δ).

Przypadki zostaną przedstawione dla położenia lewego końca odcinka względem leftp trapezu- dla prawego końca postępowanie jest analogiczne.

Algorytm konstrukcji mapy- wstawianie odcinka

Przypadek 1. Wstawiany odcinek przecina tylko jeden trapez

Uaktualnienie sąsiadów



Przypadek 1.1.: współrzędna x lewego końca odcinka pokrywa się z współrzędną x leftp trapezu.

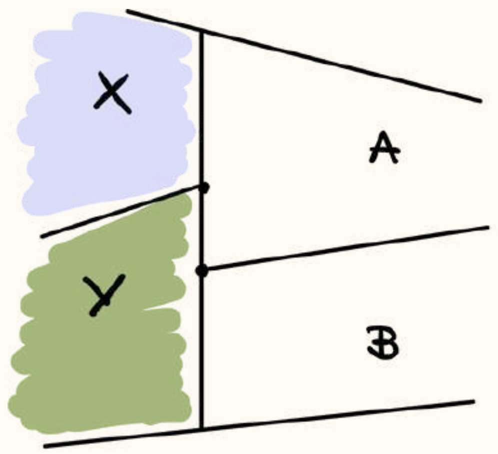
Lewym dolnym sąsiadem trapezu B staje się Y.
Prawym dolnym sąsiadem Y (o ile istnieje) staje się B.

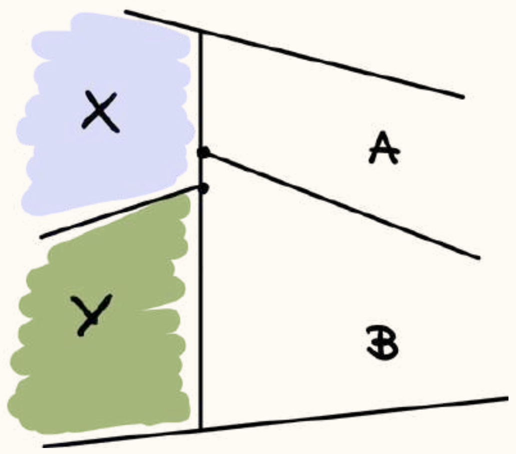
Lewym górnym sąsiadem trapezu A staje się X.
Prawym górnym sąsiadem X (o ile istnieje) staje się A.

Przypadki w zalezności od położenia lewego końca wstawianego odcinka względem leftp trapezu na osi y:

Lewy koniec odcinka leży poniżej leftp trapezu:

Lewym dolnym sąsiadem A staje się Y.
Prawym górnym sąsiadem Y (o ile istnieje) staje się A.





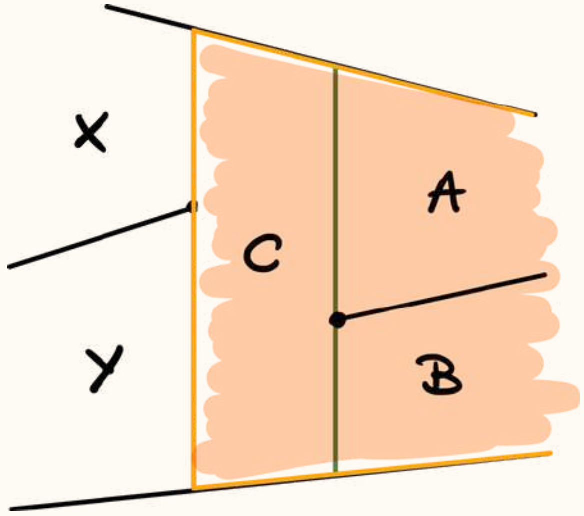
Lewy koniec odcinka leży powyżej leftp trapezu:

Lewym górnym sąsiadem B staje się X.
Prawym dolnym sąsiadem X (o ile istnieje) staje się B.

Algorytm konstrukcji mapy- wstawianie odcinka

Przypadek 1. Wstawiany odcinek przecina tylko jeden trapez

Uaktualnienie sąsiadów



Przypadek 1.2.: współrzędna x lewego końca odcinka jest większa niż współrzędna x leftp trapezu.

Tworzymy nowy trapez C. Jego top(C) jest top trapezu przecinanego przez wstawiany odcinek, botom(C)- wstawiany odcinek, leftp(C)- leftp trapezu przecinanego przez wstawiany odcinek, a right(C)- lewy koniec wstawianego odcinka.

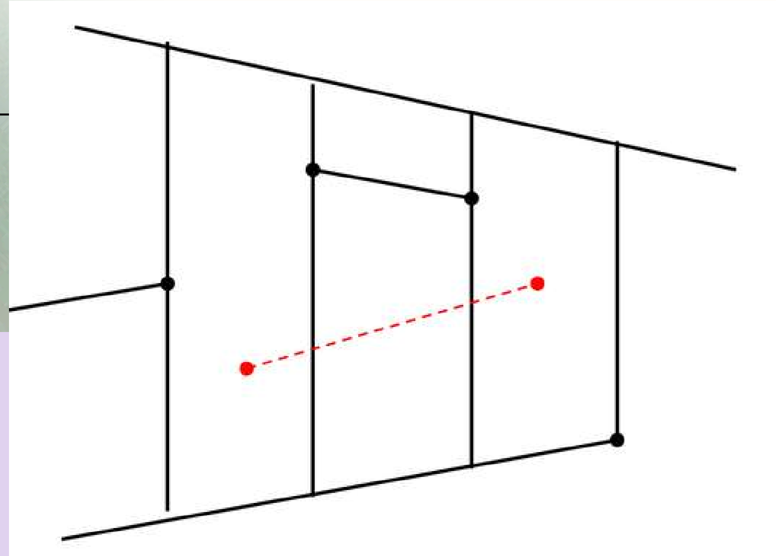
Lewym dolnym sąsiadem C staje się X. Prawym dolnym sąsiadem X (o ile istnieje) staje się C.
Lewym górnym sąsiadem C staje się Y. Prawym górnym sąsiadem Y (o ile istnieje) staje się C.

Prawym górnym sąsiadem C staje się A. Lewym górnym sąsiadem A staje się C.
Prawym dolnym sąsiadem C staje się B. Lewym dolnym sąsiadem B staje się C.

Algorytm konstrukcji mapy- wstawianie odcinka

Przypadek 2. Wstawiany odcinek przecina wiele trapezów

Uaktualnienie sąsiadów



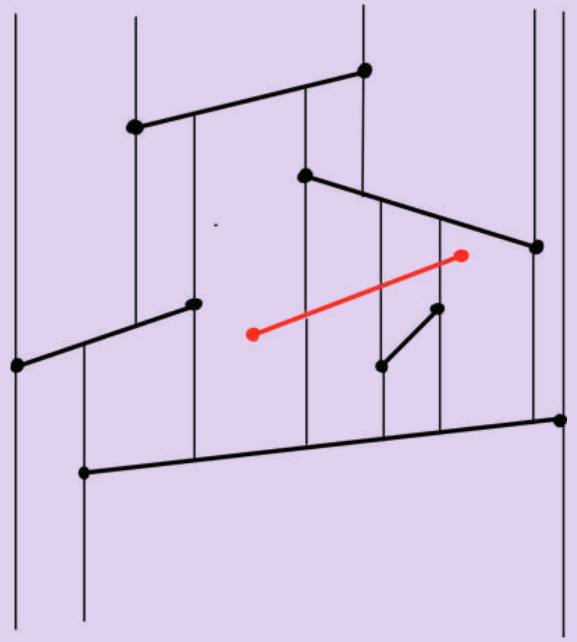
Analizujemy po kolei, od lewej strony, każdy trapez, jaki przecinany jest przez wstawiany odcinek. Dla pierwszego i ostatniego przeciętego trapezu postępujemy tak, jak w przypadku wstawiania odcinka do pojedynczego trapezu.

Należy jednak zapamiętywać, czy przechodzimy do dolnego, czy górnego sąsiada trapezu, aby poprawnie utworzyć nowe trapezy. W zależności od dwóch kolejnych takich przejść odpowiednie trapezy są łączone oraz tworzone są nowe trapezy.

Algorytm konstrukcji mapy- wstawianie odcinka
Przypadek 2. Wstawiany odcinek przecina wiele trapezów

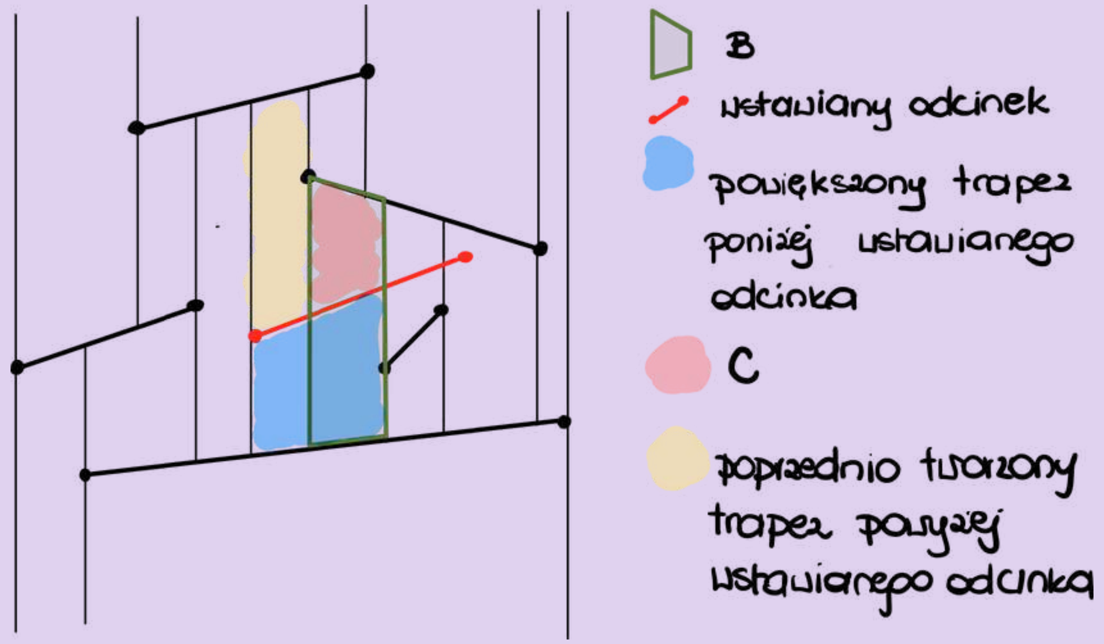
Uaktualnienie sąsiadów

Załóżmy, że wstawiany odcinek przechodzi, między innymi, kolejno przez trapezy A i B. A jest poprzednio rozpatrzonym trapezem, natomiast B- aktualnie rozpatrywanym.
Załóżmy również, że odcinek przechodzi z B do jego górnego sąsiada, jak również z A odcinek przechodził do górnego sąsiada (którym jest B).



Postępowanie w tym przypadku:

1. Powiększamy tworzony trapez znajdujący się poniżej wstawianego odcinka- przedłużamy do współrzędnej x rightp(B).
2. Tworzymy nowy trapez C znajdujący się powyżej wstawianego odcinka. top(C) to top(B), botom(C) to wstawiany odcinek, leftp(C) to rightp poprzednio tworzonego trapezu znajdującego się powyżej wstawianego odcinka, a rightp(C) to rightp(B).
3. Uaktualniamy sąsiedztwa: prawym dolnym sąsiadem poprzednio tworzonego trapezu powyżej odcinka jest C (i odwrotnie), a górnymi sąsiadami C są górni sąsiedzi B (i odwrotnie- jeśli istnieli).



Graf wyszukiwania

Wstęp

Graf wyszukiwania to struktura danych, która wspiera efektywne określanie położenia punktu w podziale płaszczyzny, realizowanym za pomocą mapy trapezowej. Struktura ta działa jako skierowany, acykliczny graf, którego liście odpowiadają trapezom w mapie trapezowej T(S). Każdy liść w grafie zawiera wskaźnik do odpowiedniego trapezu, a każdy trapez w T(S) ma wskaźnik do odpowiadającego mu liścia w grafie.

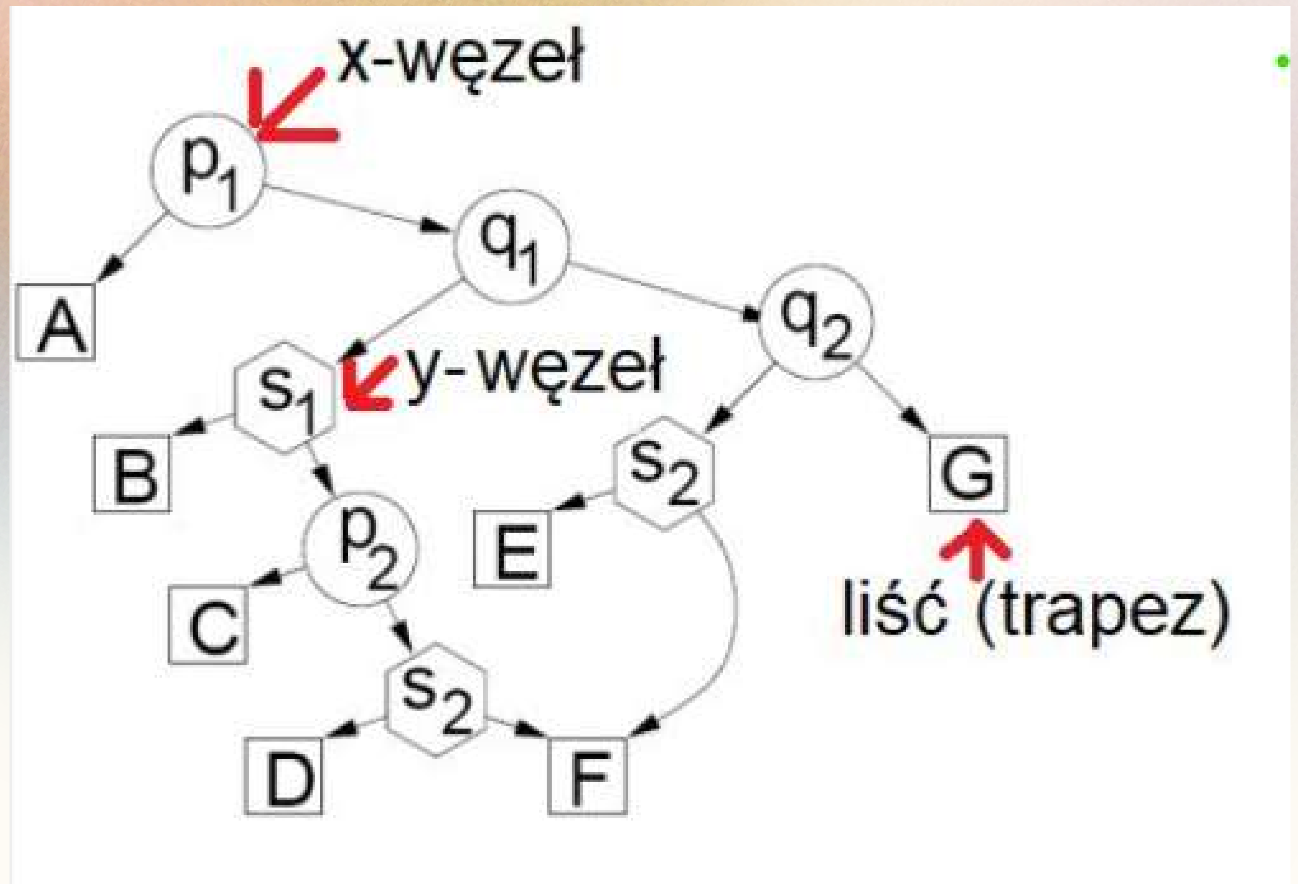
Węzły grafu wyszukiwania

Wewnętrzne węzły dzielą się na dwa typy:

- x-węzeł - przechowuje współrzędne wierzchołka
- y-węzeł - przechowuje wskaźnik do odcinka

Zasady poruszania się w drzewie wyszukiwania:

1. x-węzły (punkty):
    - Jeśli punkt znajduje się na lewo od pionowej prostej przechodzącej przez wierzchołek reprezentowany w węźle, przechodzimy do lewego dziecka węzła.
    - W przeciwnym razie, kierujemy się do prawego dziecka.
2. y-węzły (odcinki):
    - Jeśli punkt znajduje się powyżej odcinka reprezentowanego w węźle, przechodzimy do lewego dziecka.
    - Jeśli punkt jest poniżej odcinka, kierujemy się do prawego dziecka.

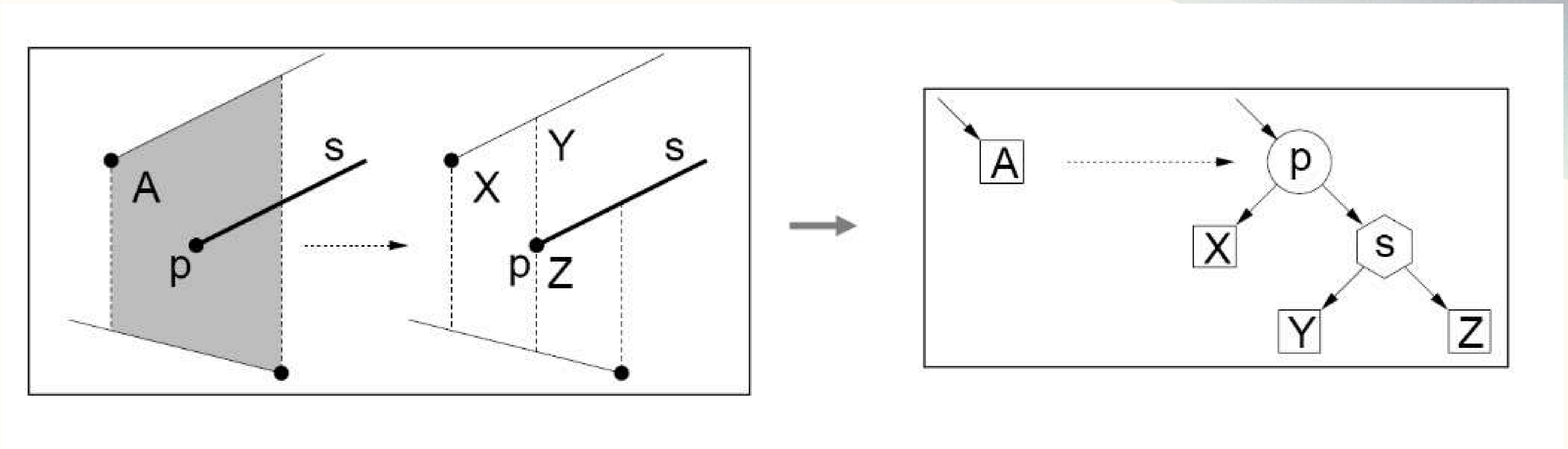


Algorytm konstrukcji grafu wyszukiwań

Podczas usuwania trapezu z grafu wyszukiwania liść reprezentujący ten trapez jest zastępowany nową częścią drzewa, która odzwierciedla podział na nowo utworzone trapezy. Sposób zastąpienia zależy od liczby końców odcinka s, które znajdują się w usuwanym trapezie

Przypadek 1

Gdy usuwany trapez **A** zawiera jeden koniec odcinka **s**, zastępujemy go trzema nowymi trapezami: **X**, **Y** i **Z**. Nowe trapezy są wstawiane do struktury zgodnie z modelem poruszania się w drzewie opisanym wcześniej. W procesie tym dodajemy odpowiednie węzły: y-węzły, reprezentujące odcinek oraz liście reprezentujące nowe trapezy

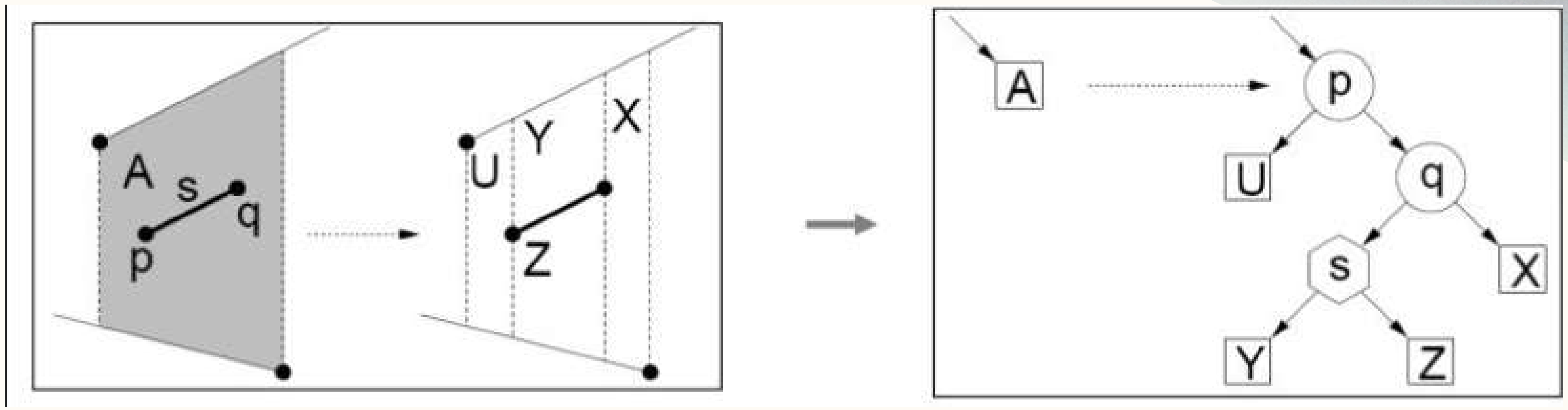


Algorytm konstrukcji grafu wyszukiwań

Podczas usuwania trapezu z grafu wyszukiwania liść reprezentujący ten trapez jest zastępowany nową częścią drzewa, która odzwierciedla podział na nowo utworzone trapezy. Sposób zastąpienia zależy od liczby końców odcinka s, które znajdują się w usuwanym trapezie

Przypadek 2

Gdy usuwany trapez **A** zawiera w sobie odcinek **s** (czyli zawiera oba końce odcinka), zastępujemy go czterema nowymi trapezami: **U**, **X**, **Y** i **Z**.

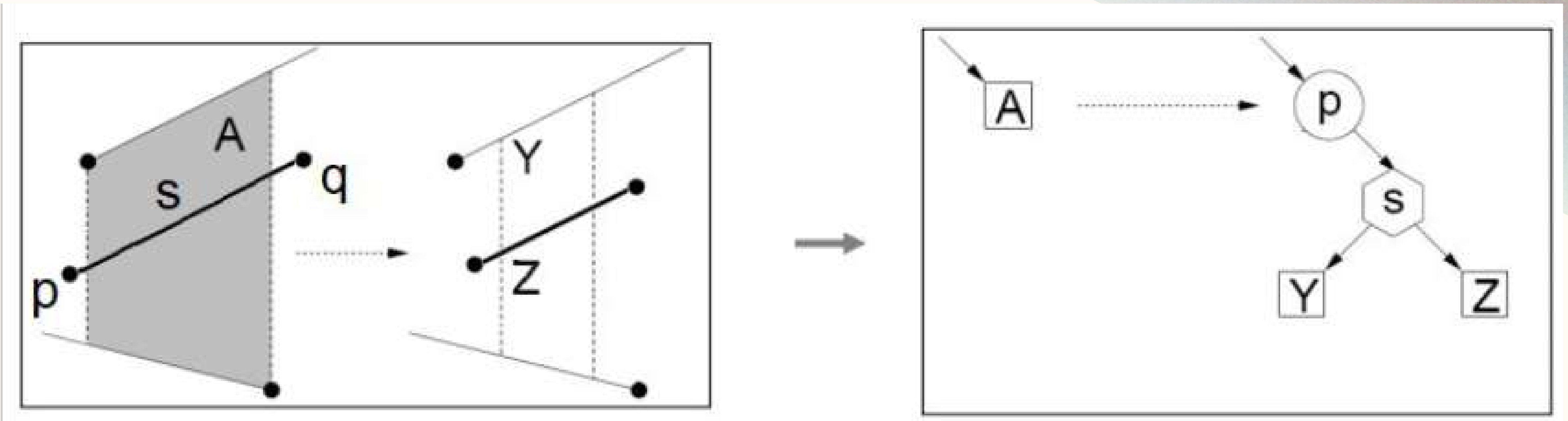


Algorytm konstrukcji grafu wyszukiwań

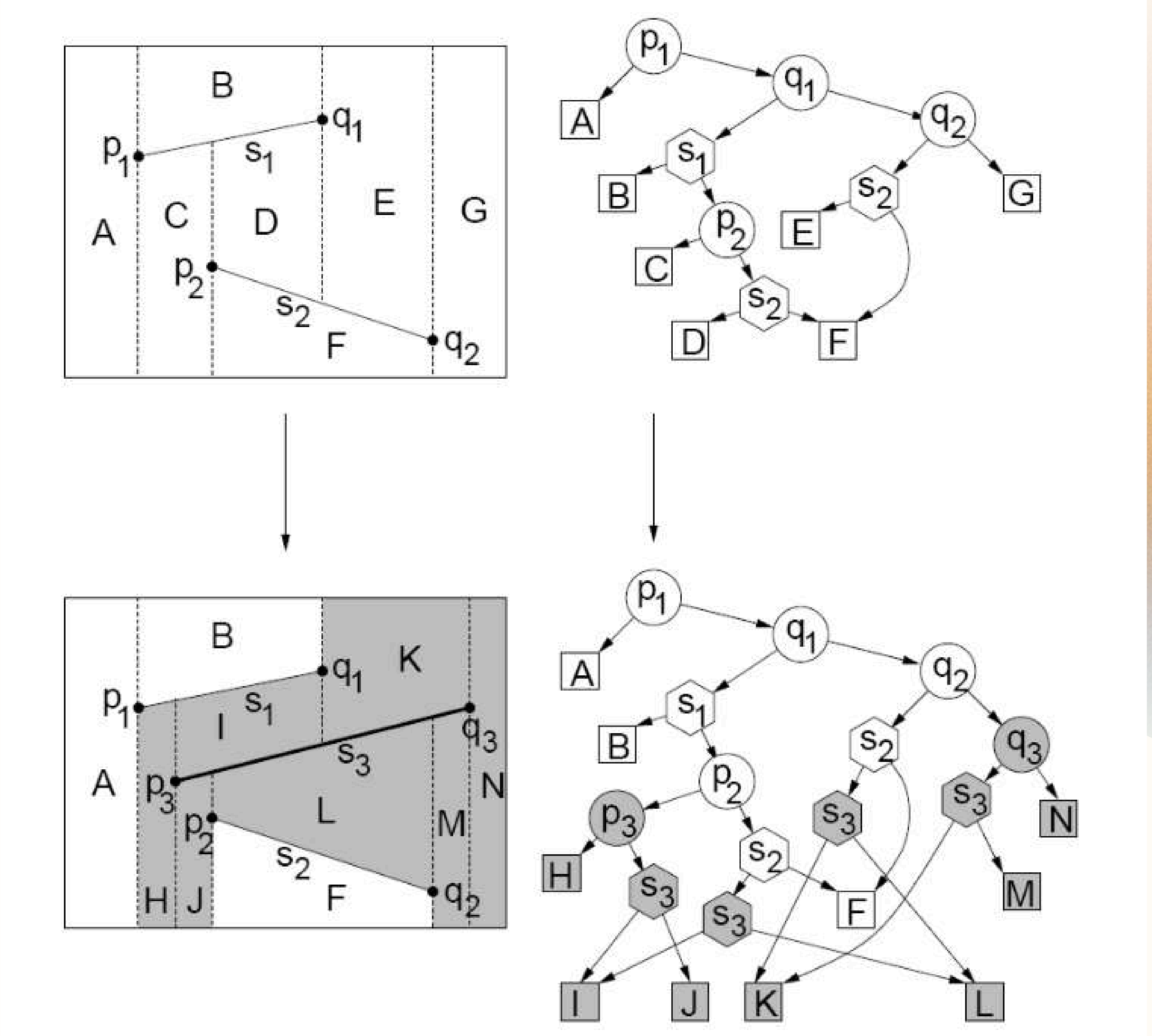
Podczas usuwania trapezu z grafu wyszukiwania liść reprezentujący ten trapez jest zastępowany nową częścią drzewa, która odzwierciedla podział na nowo utworzone trapezy. Sposób zastąpienia zależy od liczby końców odcinka sktóre znajdują się w usuwanym trapezie

Przypadek 3

Gdy usuwany trapez **A** nie zawiera ani jednego końca odcinka **s** zastępujemy go dwoma nowymi trapezami **Y** i **Z**



Przykład konstrukcji grafu wyszukiwania



Dodatkowe informacje o grafie wyszukiwań

Złożoność tworzenia grafu wyszukiwań

Tworzenie grafu wyszukiwań w metodzie trapezowej wymaga podziału przestrzeni na trapezy na podstawie dostarczonych segmentów linii, co wiąże się z szeregiem operacji geometrycznych i konstrukcji pomocniczych.
Czas tworzenia grafu:

1. Konstrukcja grafu: O(nlogn) - Podczas wprowadzania segmentów do grafu wyszukiwań, każdy nowy segment powoduje aktualizację drzewa wyszukiwań trapezowych i potencjalnie dzieli istniejące trapezy na nowe.
    - Operacje te odbywają się w czasie logarytmicznym na poziomie drzewa dla każdego segmentu, co prowadzi do całkowitej złożoności O(nlogn).
    - Dla dobrze rozłożonych danych przestrzennych algorytm zachowuje swoją optymalną złożoność.
2. Aktualizacje struktury: O(n) -Gdy segment przecina istniejące trapezy, struktura grafu musi być zaktualizowana. Może to obejmować dodawanie nowych trapezów i tworzenie nowych wierzchołków w grafie. Każda aktualizacja jest lokalna i proporcjonalna do liczby przeciętych trapezów.
3. Złożoność pamięciowa:

- O(n) – Liczba trapezów jest liniowo zależna od liczby segmentów.
- Chociaż teoretycznie liczba trapezów może wzrosnąć do $O(n^2)$, w praktyce graf rzadko osiąga taką złożoność dzięki zastosowaniu efektywnych algorytmów podziału przestrzeni, takich jak algorytmy inkrementalne.

Dodatkowe informacje o grafie wyszukiwań

- **Spójność struktury:** Położenie punktu w grafie D jest określane jednoznacznie na podstawie lokalnych decyzji podejmowanych w kolejnych węzłach.
- **Obsługa sytuacji granicznych:** W przypadku, gdy punkt leży dokładnie na prostej lub odcinku, reguły decyzyjne mogą być rozszerzone o dodatkowe kryteria, takie jak rozróżnienie na podstawie kierunku odcinka lub jego nachylenia.

Wyszukiwanie punktu w mapie

Dane wejściowe:

- Szukany punkt q
- Mapa Trapezowa

Wynik:

- Trapez, w którym znajduje się szukany punkt

Algorytm wyszukiwania punktu w mapie

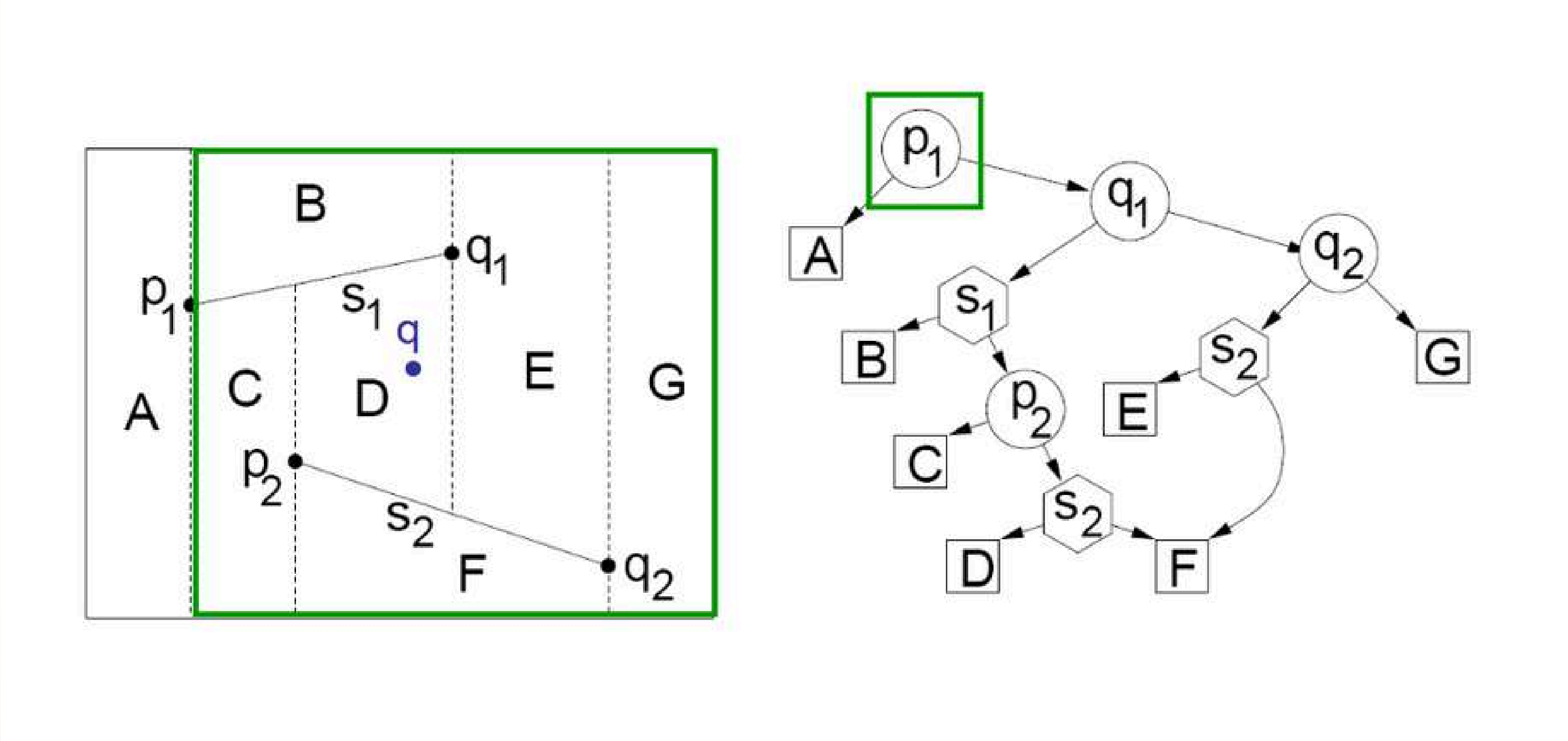
Algorytm lokalizacji punktu w strukturze wyszukiwania oparty na drzewie wykonuje następujące kroki:

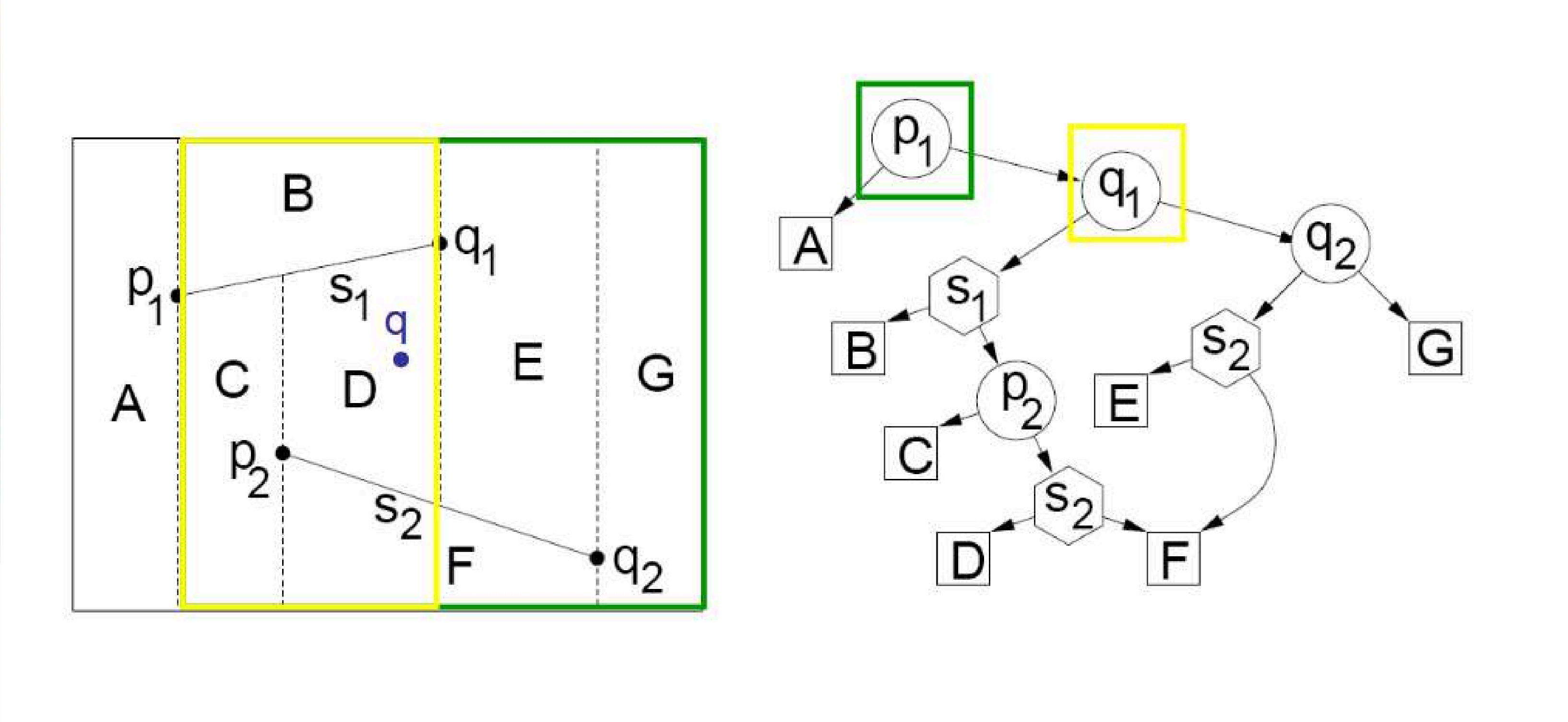
1. x-węzeł (punkt):
- Jeśli szukany punkt leży po lewej stronie pionowej prostej przechodzącej przez wierzchołek w x-węźle, przechodzimy do lewego potomka.
- W przeciwnym przypadku, przechodzimy do prawego potomka.

2. y-węzeł (odcinek):
- Jeśli szukany punkt znajduje się poniżej odcinka reprezentowanego przez węzeł, przechodzimy do lewego potomka.
- W przeciwnym przypadku, przechodzimy do prawego potomka.

3. Liść (trapez):
- Gdy dotrzemy do liścia, oznacza to, że znaleźliśmy trapez, w którym znajduje się poszukiwany punkt, więc algorytm kończy działanie.

Złożoność tego algorytmu jest logarytmiczna, a czas lokalizacji punktu na mapie trapezowej wynosi O(logn) co sprawia, że algorytm jest bardzo wydajny, nawet dla dużych zbiorów danych.



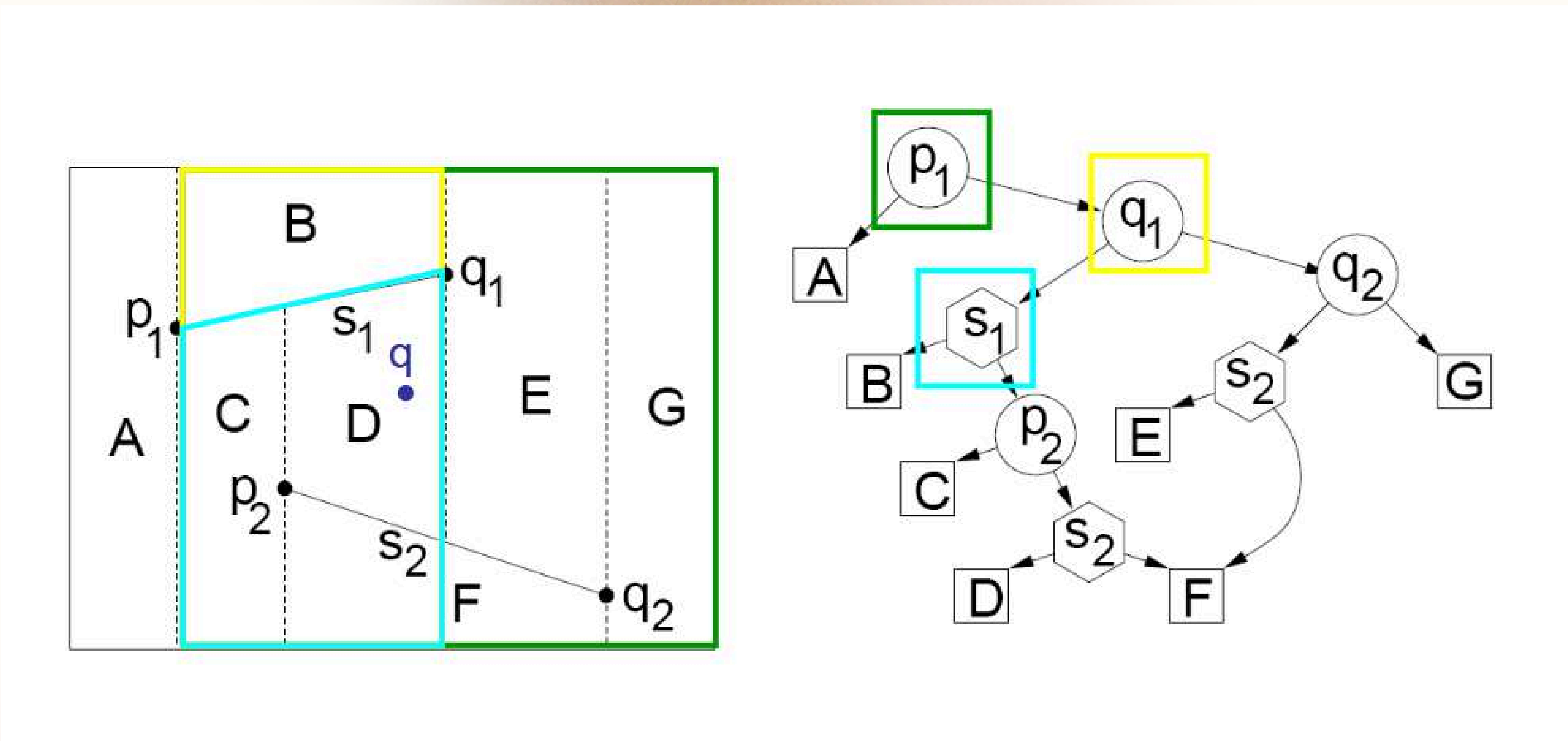


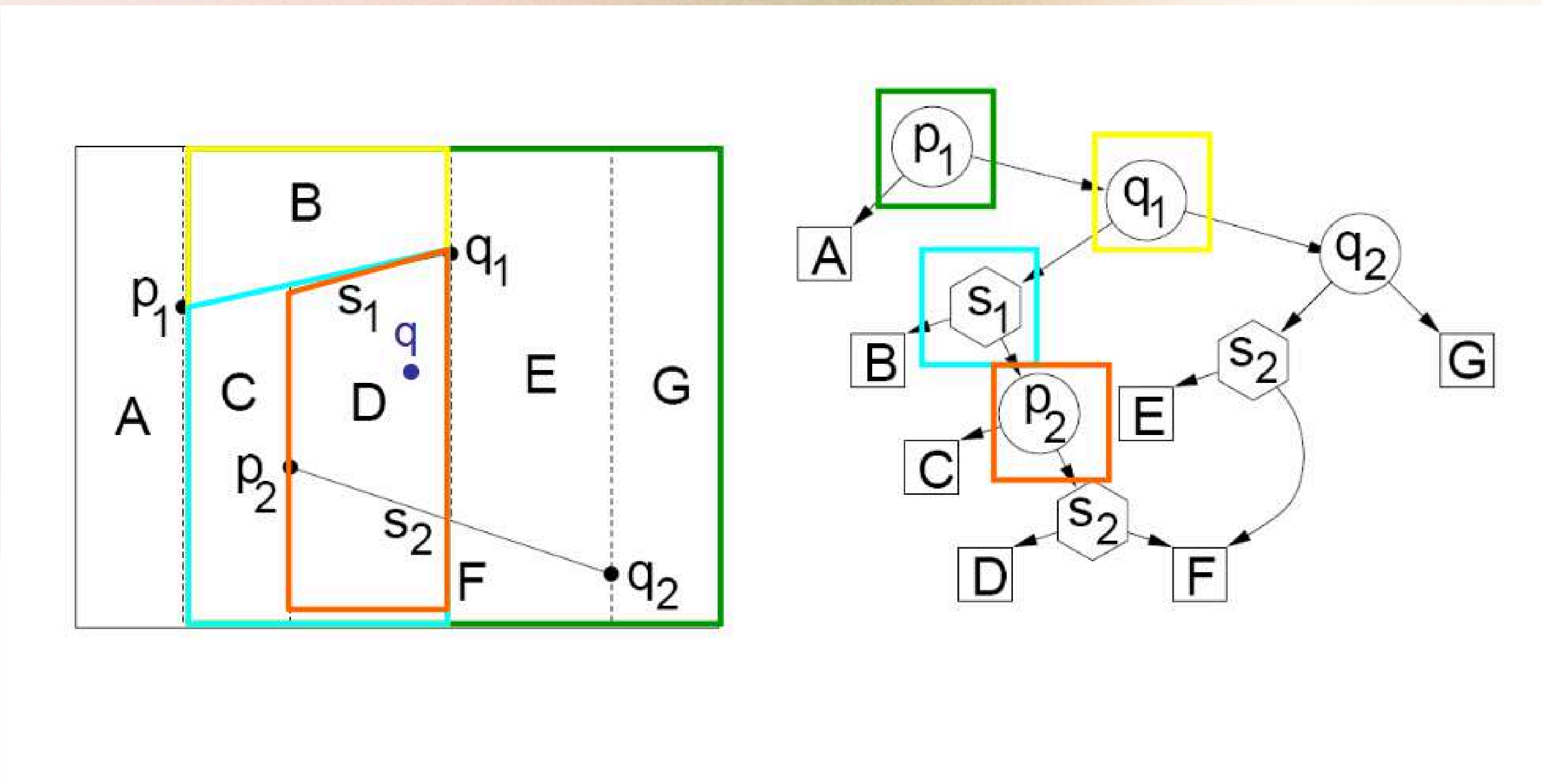
Algorytm wyszukiwania punktu w mapie

Algorytm lokalizacji punktu w strukturze wyszukiwania oparty na drzewie wykonuje następujące kroki:

1. x-węzeł (punkt):
   - Jeśli szukany punkt leży po lewej stronie pionowej prostej przechodzącej przez wierzchołek w x-węźle, przechodzimy do lewego potomka.
   - W przeciwnym przypadku, przechodzimy do prawego potomka.
2. y-węzeł (odcinek):
   - Jeśli szukany punkt znajduje się poniżej odcinka reprezentowanego przez węzeł, przechodzimy do lewego potomka.
   - W przeciwnym przypadku, przechodzimy do prawego potomka.
3. Liść (trapez):
   - Gdy dotrzemy do liścia, oznacza to, że znaleźliśmy trapez, w którym znajduje się poszukiwany punkt, więc algorytm kończy działanie.

Złożoność tego algorytmu jest logarytmiczna, a czas lokalizacji punktu na mapie trapezowej wynosi O(logn), co sprawia, że algorytm jest bardzo wydajny, nawet dla dużych zbiorów danych.



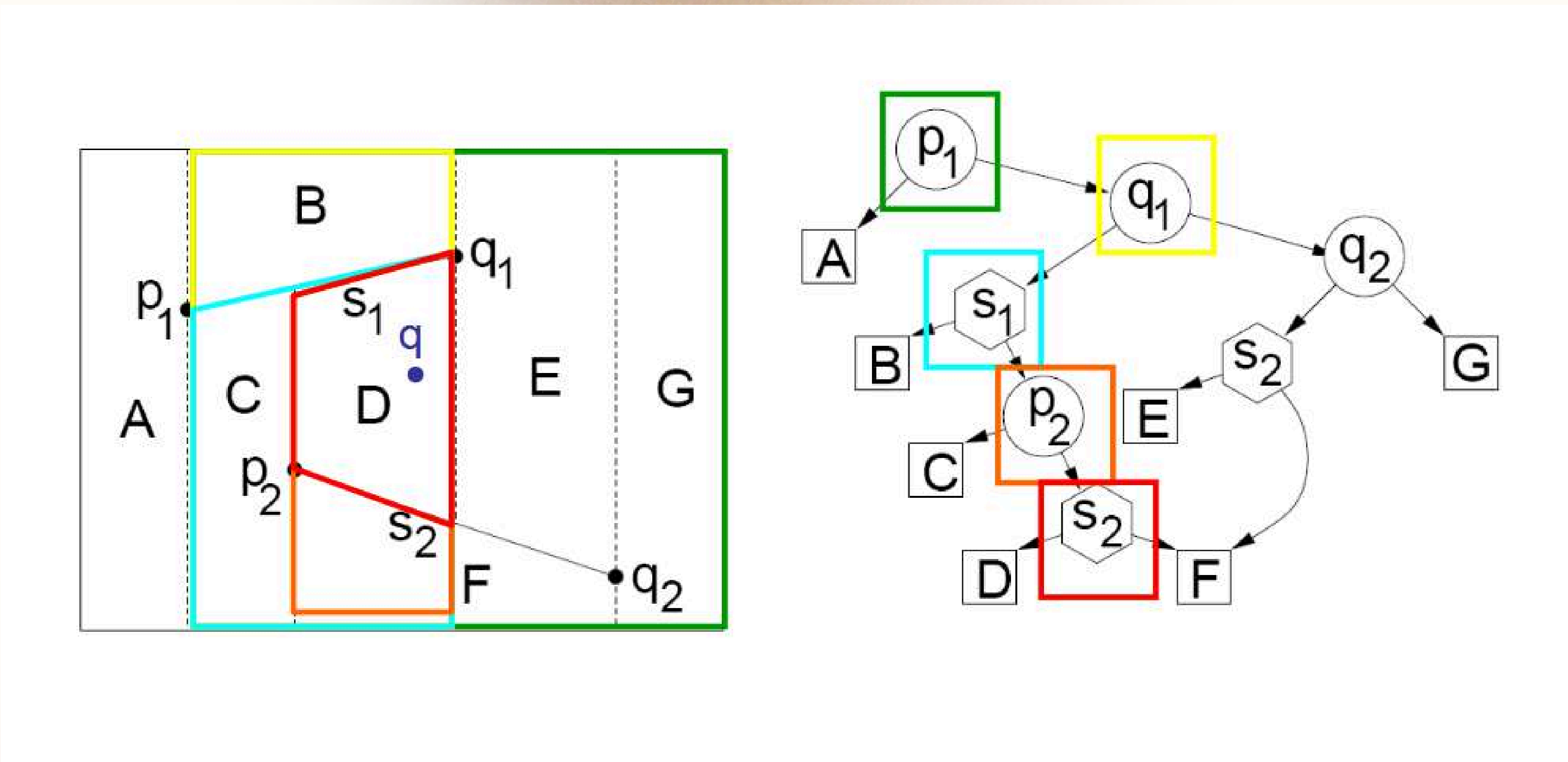


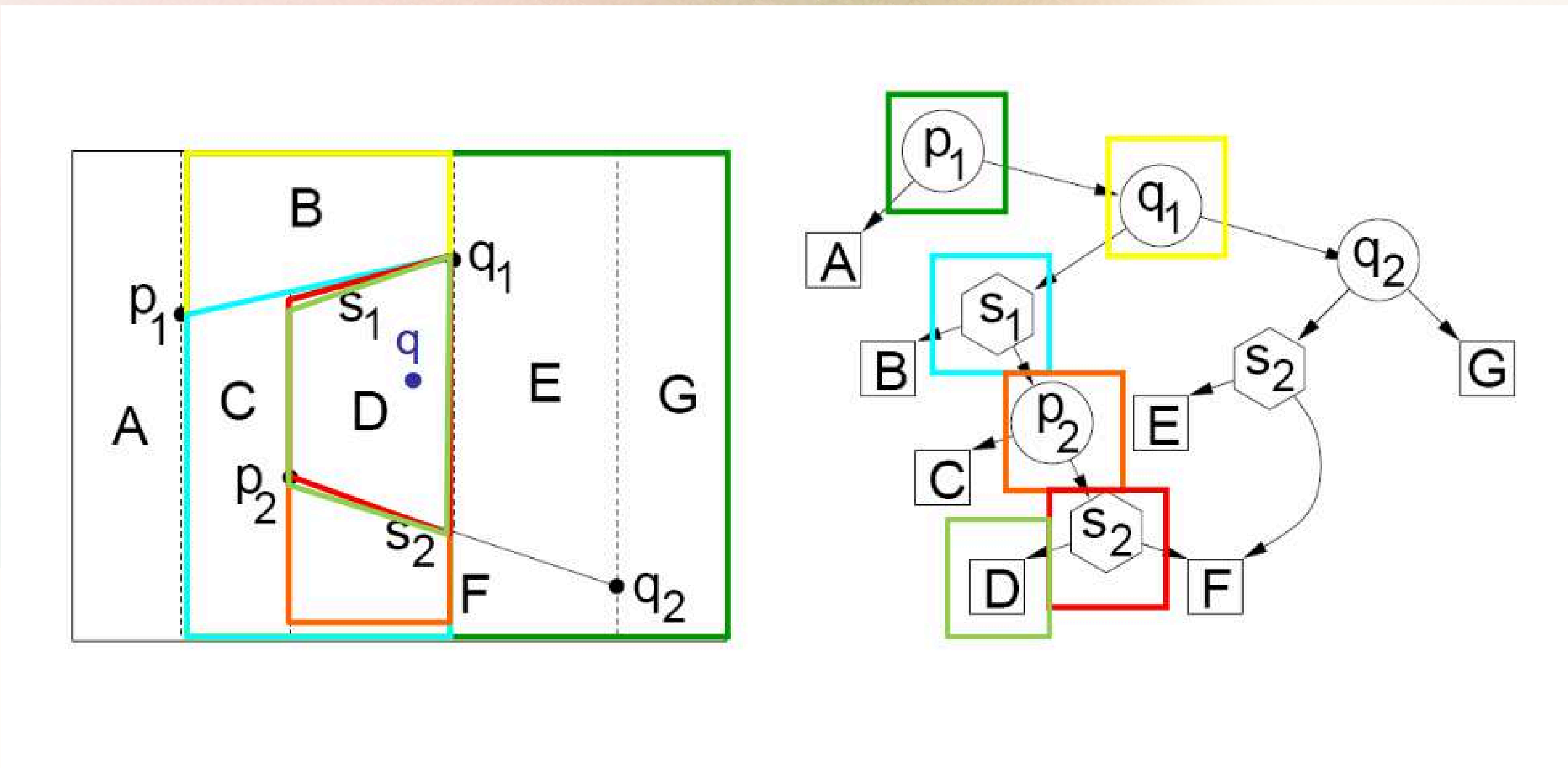
Algorytm wyszukiwania punktu w mapie

Algorytm lokalizacji punktu w strukturze wyszukiwania oparty na drzewie wykonuje następujące kroki:

1. x-węzeł (punkt):
- Jeśli szukany punkt leży po lewej stronie pionowej prostej przechodzącej przez wierzchołek w x-węźle, przechodzimy do lewego potomka.
- W przeciwnym przypadku, przechodzimy do prawego potomka.
2. y-węzeł (odcinek):
- Jeśli szukany punkt znajduje się poniżej odcinka reprezentowanego przez węzeł, przechodzimy do lewego potomka.
- W przeciwnym przypadku, przechodzimy do prawego potomka.
3. Liść (trapez):
- Gdy dotrzemy do liścia, oznacza to, że znaleźliśmy trapez, w którym znajduje się poszukiwany punkt, więc algorytm kończy działanie.

Złożoność tego algorytmu jest logarytmiczna, a czas lokalizacji punktu na mapie trapezowej wynosi O(logn), co sprawia, że algorytm jest bardzo wydajny, nawet dla dużych zbiorów danych.





Bibliografia

https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect08-trap-map.pdf
https://faculty.sites.iastate.edu/jia/files/inline-files/13.%20trapezoidal%20maps.pdf
https://youtube.com/playlist?list=PLubYOWSI9mIvTio-1bXWnhE9LdeXfox1z&si=Y4rtvJMhDcK7QziP