Algorytmy geometryczne
Dokumentacja

Projekt

Otoczka wypukła dla zbioru punktów w przestrzeni dwuwymiarowej

Grupa 4 – czwartek, 11:20-12:50, tydzień B
Adam Naumiec, Michał Kuszewski

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

Kraków, grudzień MMXXII, styczeń MMXXIII

Spis treści

1. Wstęp 5

1.1. Projekt 5
1.2. Informacje dotyczące dokumentacji projektu 5
1.3. Organizacja dokumentu 6
1.4. Cel projektu 6
1.5. Realizacja projektu 6

2. Część techniczna 7

2.1. Opis programu i podstawowych modułów 7
2.2. Wymagania techniczne 7
2.3. Dane domyślne 8

3. Cześć użytkownika 9

3.1. Sposób uruchamiania programu 9
3.2. Korzystanie z funkcji 9
3.3. Wywołanie funkcji 10

4. Zbiory testowe 12

4.1. Generowanie liczb losowych 12
4.2. Zbiór A 12
4.3. Zbiór B 13
4.4. Zbiór C 14
4.5. Zbiór D 15
4.6. Własny zbiór 15

5. Generowanie punktów 16

5.1. generate_points 16
5.2. gen_set_a 16
5.3. gen_set_b 17
5.4. gen_set_c 17
5.5. gen_set_d 18

6. Funkcje pomocnicze 19

6.1. det 19
6.2. distance 19
6.3. measure_time 19
6.4. are_collinear 20
6.5. convert_points_to_lines 20
6.6. find_minimum_point 20
6.7. find_maximum_point 21
6.8. find_lowest 21

7. Algorytm dziel i zwyciężaj (dziel i rządź) 22

7.1. brute_convex_hull 22
7.2. convex_hull 23
7.3. find_leftmost 24
7.4. find_rightmost 24
7.5. merge 24
7.6. divide_and_conquer 28
7.7. divide_and_conquer_visualisation 29
7.8. convex_hull_visualisation 29
7.9. merge_visualisation 30

8. Algorytm przyrostowy 31

8.1. incremental 31
8.2. find_left_tangent 32
8.3. find_right_tangent 33
8.4. incremental_visualisation 34
8.5. compare 35
8.6. filter_points 36

9. Algorytm Quickhull 37

9.1. line_point_distance 37
9.2. part_quickhull 38
9.3. quickhull 39
9.4. part_quickhull_visualisation 40
9.5. quickhull_visualisation 41

10. Algorytm górnej i dolnej otoczki 42

10.1. upper_lower_hull 42
10.2. upper_lower_hull_visualisation 43

11. Algorytm Jarvisa 44

11.1. jarvis 44
11.2. jarvis_visualisation 45

12. Algorytm Grahama 46

12.1. graham 46
12.2. graham_visualisation 47
12.3. find_tangent 47
12.4. find_extreme_point 49
12.5. find_next_point 49

13. Algorytm Chana 50

13.1. partial_hull 50
13.2. chan 51
13.3. make_segments 52
13.4. partial_hull_visualisation 52

13.5. chan_visualisation....53
14. Sprawozdanie....54
14.1. Kryterium klasyfikacyjne – wyznacznik....54
14.2. Tolerancja precyzji obliczeń....54
14.3. Opis problemów....55
14.4. Wykonane testy....56
14.5. Analiza wyników testów....56
14.6. Uzyskane wyniki....57
15. Teoretyczne złożoności algorytmów....58
16. Twórcy algorytmów i data znalezienia algorytmów....58
17. Wnioski i podsumowanie....59
18. Bibliografia....60

1. Wstęp

1.1. Projekt

Projekt składa się z:

1. Dokumentacji projektu,
2. Programu wraz z algorytmami i wizualizacjami,
3. Prezentacji.

1.2. Informacje dotyczące dokumentacji projektu

Informacje do dokumentacji:

- Wszystkie tworzone i pośrednie otoczki są zorientowane przeciwnie do ruchu wskazówek zegara.
- Otoczki wypukłe reprezentowane są przez listę kolejnych punktów na otoczce reprezentowanych przez krotkę dwóch liczb rzeczywistych.
- Wizualizacje reprezentowane są przez listę kolejnych scen, na które mogą składać się kolekcje punktów i linii.
- Do wizualizacji danych na wykresach wykorzystano narzędzie dostarczone na laboratoriach.
- Punkt reprezentowany jest jako krotka dwóch liczb rzeczywistych odpowiadających wartościom kolejno współrzędnej x i y.
- Linia reprezentowana jest jako lista dwóch krotek dwóch liczb rzeczywistych odpowiadających wartościom kolejno współrzędnej x i y, krotki stanowią reprezentację dwóch punktów będących kolejno początkiem i końcem linii.
- Ustalona została tolerancja związana z dokładnością obliczeń. Wyrażona jest współczynnikiem epsilon, którego domyśla w projekcie wartość to 10^{-12}. Kwestia dokładności obliczeń została poruszona w dalszej części dokumentacji.
- Wykorzystane zostały cztery różne zbiory testowe o różnych właściwościach, dodatkowo program pozwala generować zbiory testowe o różnych zadanych parametrach.

- W sekcji każdego algorytmu generowania otoczki wypukłej przy opisie funkcji realizującej dany algorytm został dodatkowo opisany sposób jego działania.

1.3. Organizacja dokumentu

Organizacja dokumentacji:

1. Cześć techniczna (opis programu, podstawowych modułów, wymagania techniczne),
2. Część użytkownika (sposób uruchamiania programu i korzystania z jego funkcji),
3. Sprawozdanie (opis problemu, wykonane testy, uzyskane wyniki).

Dokumentacja składa się z 60 stron i podzielna jest na 18 sekcji.

1.4. Cel projektu

Celem projektu było zaimplementowanie najpopularniejszych algorytmów wyznaczania otoczki wypukłej punktów na płaszczyźnie dwuwymiarowej w przestrzeni $\mathbb{R}^2$, wizualizacja kroków działania algorytmów oraz ich analiza, przygotowanie dokumentacji do projektu wraz ze sprawozdaniem, przetestowanie algorytmów na różnych zbiorach danych oraz prezentacja otrzymanych rezultatów na laboratorium.

1.5. Realizacja projektu

Projekt realizowany był przez dwie osoby, każdy przygotowywał ustalony wcześniej zakres materiału do projektu. Realizację zadań poprzedził wykład wprowadzający zagadnienie otoczki wypukłej oraz laboratoria zorientowane pod kątem analizy algorytmu Jarivisa i Grahama dla różnych zbiorów danych wraz z przygotowaniem sprawozdania z laboratorium. Podczas realizacji projektu korzystano z materiałów wymienionych w sekcji *Bibliografia*.

2. Część techniczna

2.1. Opis programu i podstawowych modułów

Program został przygotowany w języku Python z wykorzystaniem narzędzia Jupyter Notebook oraz dodatkowego narzędzia do wizualizacji kroków działania algorytmów dostarczonego na laboratorium. Wykorzystano także biblioteki dostępne dla języka Python: matplotlib, numpy, random, math, sortedcontainers, json, time, functools. Kod programu został podzielony na sekcje wraz z funkcjami realizującymi dany algorytm. Przygotowano także funkcje pomocnicze, które wykonywały określone zadania. Dokumentacja techniczna funkcji zaimplementowanych w projekcie znajduje się w dalszej części.

2.2. Wymagania techniczne

Do uruchomienia programu potrzebny jest interpreter języka Python oraz skonfigurowany Jupyter Notebook.

Przygotowany program napisany został w Pythonie 3.11, wykorzystane IDE to PyCharm w wersji 2022.3 oraz narzędzie Jupyter Notebook działające w przeglądarce internetowej Google Chrome w wersji 108, 64-bit.

Prezentacja została przygotowana w programie Microsoft PowerPoint i zapisana w formacie pdf. Dokumentacja została przygotowana w programie Microsoft Word i zapisana w formacie pdf. Do otworzenia pliku w formacie pdf potrzebny jest program taki jak np. Google Chrome lub Adobe Acrobat Reader.

2.3. Dane domyślne

```
EPSILON = 10 ** -12

A_NUM = 100
A_MIN_RNG = -100
A_MAX_RNG = 100

B_NUM = 100
B_RADIUS = 10
B_CENTER = (0, 0)

C_NUM = 100
C_V_1 = (-10, 10)
C_V_2 = (-10, -10)
C_V_3 = (10, -10)
C_V_4 = (10, 10)

D_NUM_SIDE = 25
D_NUM_DIAGONAL = 20
D_V_1 = (0, 0)
D_V_2 = (10, 0)
D_V_3 = (10, 10)
D_V_4 = (0, 10)
```

- EPSILON – tolerancja precyzji obliczeń
- A_NUM – liczba punktów w zbiorze A
- A_MIN_RNG – minimalna wartość współrzędnej punktu w zbiorze A
- A_MAX_RNG – maksymalna wartość współrzędnej punktu w zbiorze A
- B_NUM – liczba punktów w zbiorze B
- B_RADIUS – promień okręgu w zbiorze B
- B_CENTER – środek okręgu w zbiorze B
- C_NUM – liczba punktów w zbiorze C
- C_V_1 – współrzędna pierwszego wierzchołka w zbiorze C
- C_V_2 – współrzędna drugiego wierzchołka w zbiorze C
- C_V_3 – współrzędna trzeciego wierzchołka w zbiorze C
- C_V_4 – współrzędna czwartego wierzchołka w zbiorze C
- D_NUM_SIDE – liczba punktów na dwóch bokach prostokąta w zbiorze D
- D_NUM_DIAGONAL – liczba punktów na przekątnych prostokąta w zbiorze D
- D_V_1 – współrzędna pierwszego wierzchołka w zbiorze D
- D_V_2 – współrzędna drugiego wierzchołka w zbiorze D
- D_V_3 – współrzędna trzeciego wierzchołka w zbiorze D
- D_V-4 – współrzędna czwartego wierzchołka w zbiorze D

Program umożliwia modyfikację danych domyślnych, ale nie jest to zalecane. Preferowane jest wywołanie funkcji generujących zbiory

testowe z własnymi parametrami. Dodatkowe opisy funkcji generujących zbiory testowe podane są w dalszej części dokumentacji.

3. Cześć użytkownika

3.1. Sposób uruchamiania programu

By skorzystać z programu należy uruchomić narzędzie Jupyter Notebook poprzez IDE z interpreterem języka Python i wybrać uruchomienie kolejnych zaimplementowanych funkcji. Należy pamiętać o tym, że uruchomienie wszystkich funkcji programu w kolejności ich umiejscowienia w pliku z kodem jest konieczne do poprawnego działania programu.

3.2. Korzystanie z funkcji

By z korzystać z funkcji należy dla wybranego algorytmu wywołać odpowiednią funkcję wraz z danymi testowymi. Program pozwala na generowanie danych testowych zgodnych z własnymi wymaganiami. By uruchomić wizualizację działania algorytmu należy wybrać odpowiednią funkcję, która dodatkowo generuje kolejne wykonywane w algorytmie kroki i przekazać wygenerowaną wizualizację do funkcji rysującej z dostarczonego narzędzia. Należy pamiętać, że do testowania funkcji należy skorzystać z funkcji, które nie generują wizualizacji.

3.3. Wywołanie funkcji

Wywołanie funkcji realizującej dany algorytm powinno wyglądać następująco (w miejsce *[nazwa]* należy wpisać odpowiednie dane):

- W przypadku, gdy chcemy jedynie wygenerować otoczkę bez jej wizualizacji lub przetestować działanie funkcji (zwracana jest lista punktów na otoczce):
    - Zbiór testowy:

```
%matplotlib notebook

otoczka_wypukła = [algorytm]([zbiór_testowy])
```

    - Generowanie zbioru testowego z własnymi parametrami:

```
%matplotlib notebook

otoczka_wypukła = [algorytm]([funkcja_generująca_zbiór_testowy](
[parametry_funkcji_lub_ich_brak]))
```

- W przypadku, gdy chcemy wygenerować wizualizację działania algorytmu (zwracana jest wizualizacja, w jednej komórce należy generować wizualizację tylko jednego algorytmu):
    - Zbiór testowy:

```
%matplotlib notebook

wizualizacja = Plot(scenes = [algorytm]([funkcja_generująca_zbiór_testowy](
[parametry_funkcji_lub_ich_brak])))
wizualizacja.draw()
```

    - Generowanie zbioru testowego z własnymi parametrami:

```
%matplotlib notebook

wizualizacja = Plot(scenes = [algorytm]([funkcja_generująca_zbiór_testowy](
[parametry_funkcji_lub_ich_brak])))
wizualizacja.draw()
```

Oznaczenia:

- [algorytm] – nazwa funkcji realizującej dany algorytm spośród:

```
- divide_and_conquer – algorytm dziel i rządź
- incremental – algorytm przyrostowy
- quickhull – algorytm quickhull
- upper_lower_hull – algorytm górnej i dolnej otoczki
- chan – algorytm Chana
- jarvis – algorytm Jarvisa
- graham – algorytm Grahama
```

- [zbiór_testowy] – nazwa zbiory testowego spośród:

```
- set_a – zbiór A
- set_b – zbiór B
- set_c – zbiór C
- set_d – zbiór D
```

- [funkcja_generująca_zbiór_testowy] – nazwa funkcji generującej zbiór testowy spośród:

```
- gen_set_a – zbiór A
- gen_set_b – zbiór B
- gen_set_c – zbiór C
- gen_set_d – zbiór D
```

- [parametry_funkcji_lub_ich_brak] – odpowiednie parametry funkcji generującej zbiór testowy zgodne z dokumentacją danej funkcji lub brak parametrów – parametry domyślne. Funkcje przyjmują różne parametry, należy zapoznać się z dokumentacją by dowiedzieć się jak zmodyfikować wygenerowane zbiory.

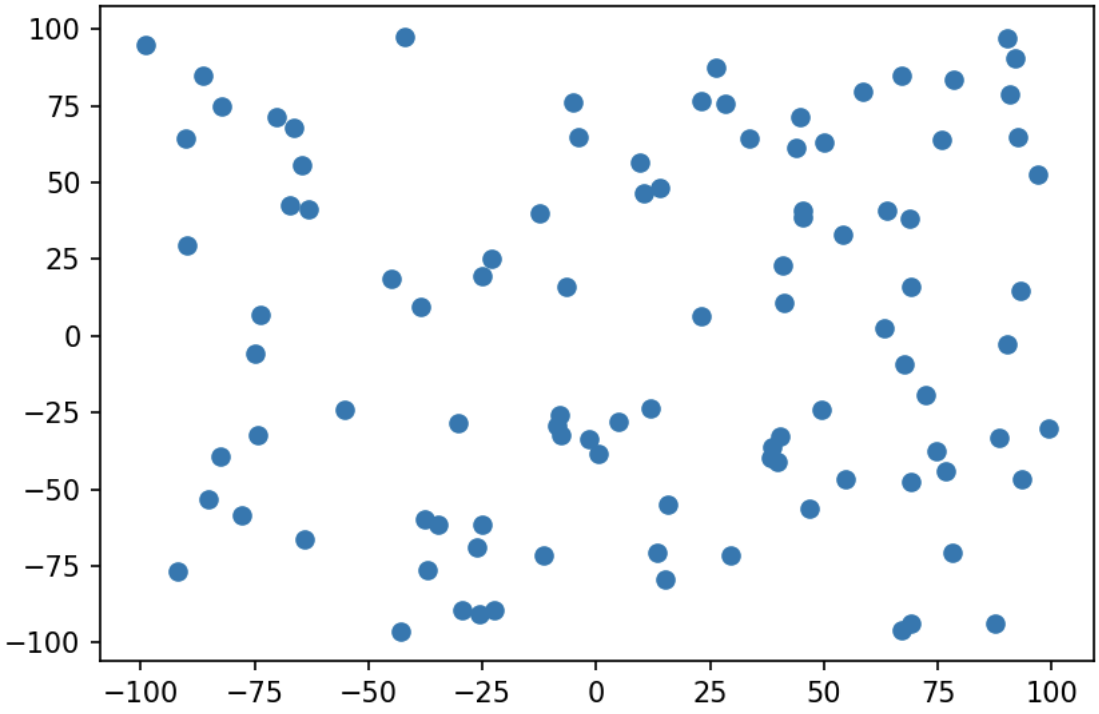
4. Zbiory testowe

4.1. Generowanie liczb losowych

Losowe liczby generowano za pomocą funkcji *uniform* oraz *randint* z biblioteki *random*. Funkcja ta generuje liczby pseudolosowe z podanego zakresu.

4.2. Zbiór A

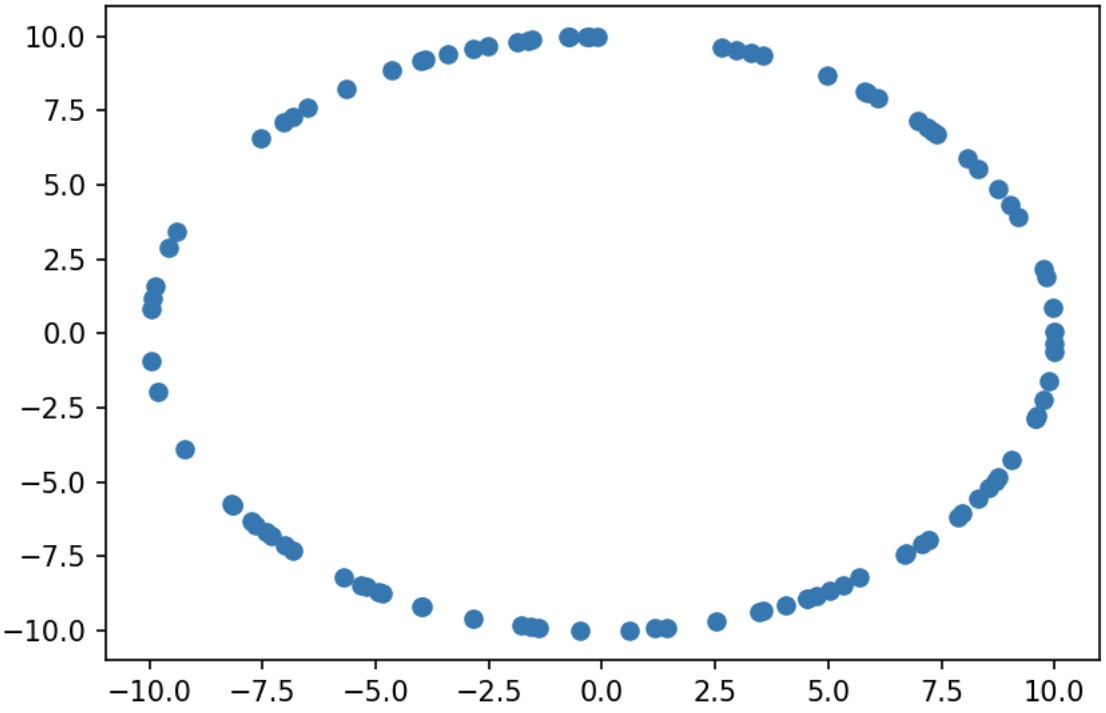
Zbiór $\mathcal{A}$ zawiera wygenerowane punkty zwarte na płaszczyźnie kartezjańskiej z punktami z zakresu $[-100, 100]$.



Rysunek 4.2. Zbiór $\mathcal{A}$

4.3. Zbiór B

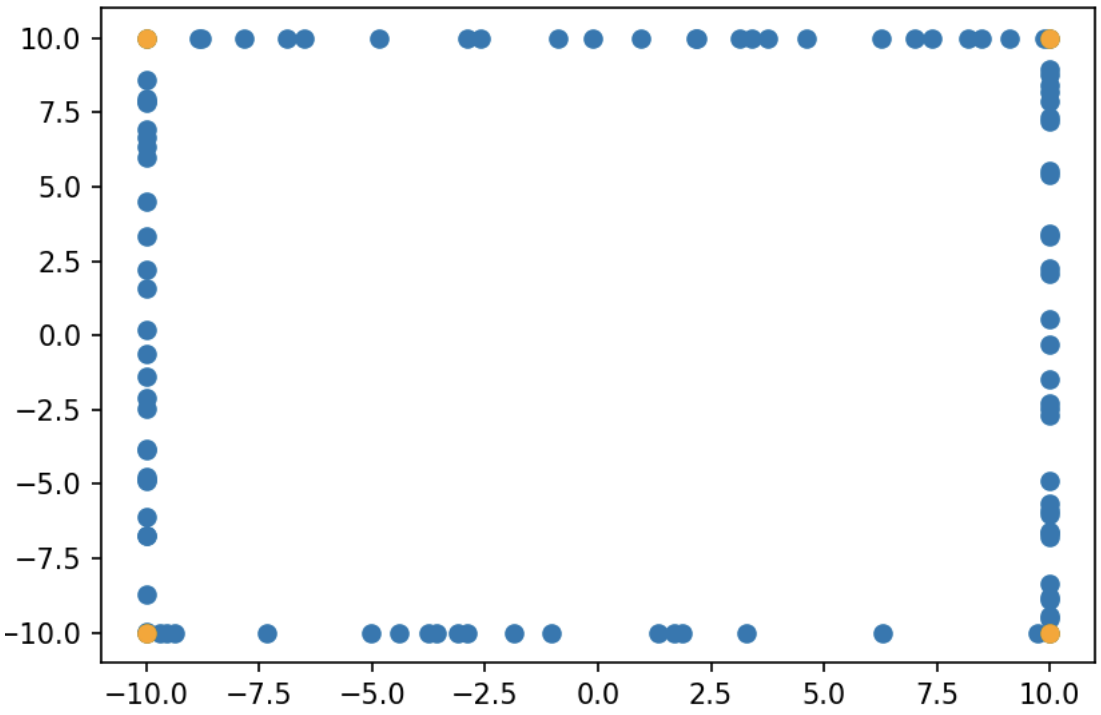
Do generacji punktów w zbiorze $\mathcal{B}$ wykorzystano z biblioteki *math* funkcje *sin* do obliczania sinusa i *cos* do obliczania cosinusa. Funkcje te zwracają wartości funkcji trygonometrycznych dla podanych argumentów. Wykorzystanie funkcji trygonometrycznych do generowania punktów znajdujących się na okręgu pozwoliło na ich równomierne rozmieszczenie, ponieważ prędkość krzywej jest stała. Wykorzystano także biblioteczną wartość liczby π.



Rysunek 4.3. Zbiór $\mathcal{B}$

4.4. Zbiór C

Zbiór $\mathcal{C}$ zawiera punkty rozmieszczone na bokach prostokąta oraz po jednym punkcie na każdym wierzchołku.

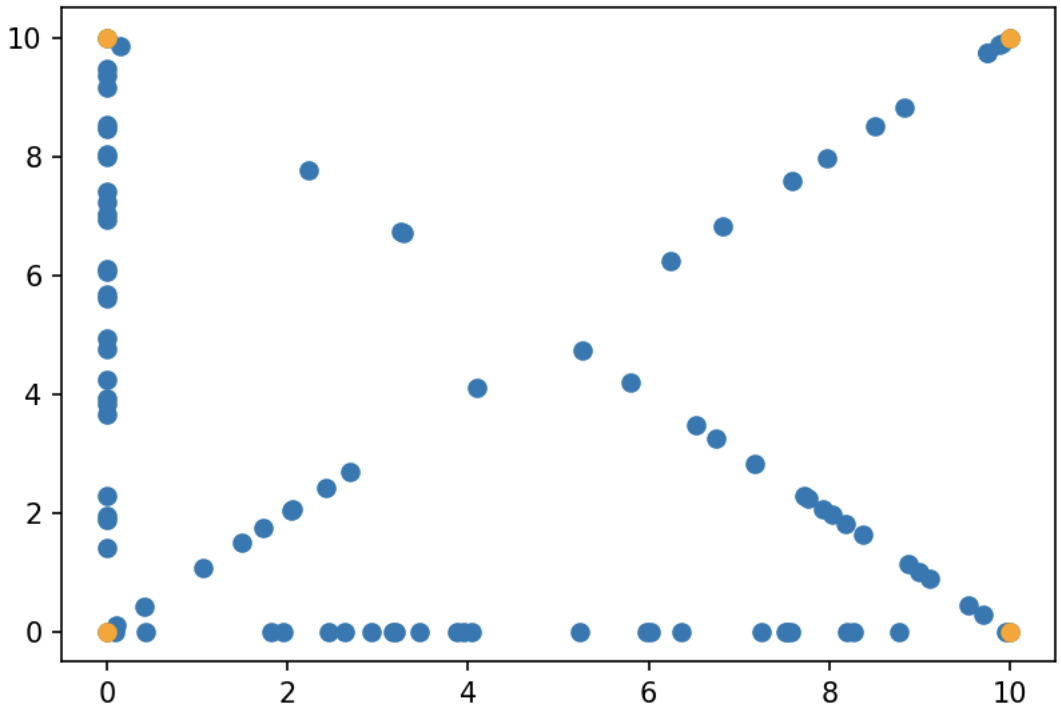


Rysunek 4.4. Zbiór $\mathcal{C}$

(na pomarańczowo zaznaczone punkty w wierzchołkach)

4.5. Zbiór D

W zbiorze $\mathcal{D}$ generwano punkty leżące na bokach, które pokrywały się z osiami OX i OY, przekątnych kwadratu oraz po jednym punkcie na każdym wierzchołku.



Rysunek 4.5. Zbiór $\mathcal{D}$

(na pomarańczowo zaznaczone punkty w wierzchołkach)

4.6. Własny zbiór

Zbiór losowych punktów można także wygenerować za pomocą funkcji generate_points, pozwala ona na określenie liczby punktów i minimalnej i maksymalnej wartości współrzędnej x oraz y.

5. Generowanie punktów

5.1. generate_points

generate_points(amount, min_x, max_x, min_y, max_y) – funkcja tworząca losowy zbiór punktów.

Parametry:

- amount – liczba losowanych punktów,
- min_x – minimalna możliwa do wylosowania współrzędna x punktu,
- max_x – maksymalna możliwa do wylosowania współrzędna x punktu,
- min_y – minimalna możliwa do wylosowania współrzędna y punktu,
- max_y – maksymalna możliwa do wylosowania współrzędna y punktu.

Wartość zwracana: (list) lista krotek reprezentujących punkty.

5.2. gen_set_a

gen_set_a(A_NUM=100, A_MIN_RNG=-100, A_MAX_RNG=100) – funkcja generująca losowy zbiór zadanej liczby punktów w zadanym przedziale. Domyślnie jest to 100 punktów na płaszczyźnie o minimalnej wartości x i y równej -100 i maksymalnej wartości równej 100.

Parametry:

- A_NUM (tuple) – zadana liczba losowych punktów (posiada wartość domyślną równą 100),
- A_MIN_RNG (tuple) – minimalna wartość współrzędnych x i y (posiada wartość domyślną równą -100),
- A_MAX_RNG (tuple) – maksymalna wartość współrzędnej x i y (posiada wartość domyślną równą 100).

Wartość zwracana: (list) lista krotek reprezentujących punkty.

5.3. gen_set_b

gen_set_b(B_NUM=100, B_RADIUS=10, C_CENTER=(0,0)) – funkcja generalcy losowy zbiór punktów na okręgu o zadanej liczbie punktów, promieniu okręgu i środku okręgu. Domyślnie jest to 100 punktów leżących na okręgu o promieniu równym 10 i środku w punkcie (0, 0).

Parametry:

- B_NUM – zadana liczba losowych punktów (posiada wartość domyślną równą 100),
- B_RADIUS – promień okręgu (posiada wartość domyślną równą 10),
- B_CENTER – środek okręgu w postaci punktu zadanego krotką dwóch liczb rzeczywistych (posiada wartość domyślną równą (0 ,0)).

Wartość zwracana: (list) lista krotek reprezentujących punkty.

5.4. gen_set_c

gen_set_c(C_NUM=100, C_V_1=(-10,10), C_V_2=(-10,-10), C_V_3=(10,-10), C_V_4=(10,10)) – funkcja generująca losowy zbiór zadanej liczby punktów leżących na bokach prostokąta zadanego czterema punktami oraz 4 punkty po 1 na każdym wierzchołu tego prostokąta. Domyślnie jest to 100 punktów na bokach prostokąta o wierzchołkach w punktach (-10, 10), (-10, -10), (10, -10), (10, 10) oraz 4 punkty po jednym na każdym z wierzchołków.

Parametry:

- C_NUM – liczba losowych punktów (posiada wartość domyślną równą 100),
- C_V_1 – pierwszy wierzchołek prostokąta (posiada wartość domyślną równą (-10, 10)),
- C_V_2 – drugi wierzchołek prostokąta (posiada wartość domyślną równą (-10, -10)),
- C_V_3 – trzeci wierzchołek prostokąta (posiada wartość domyślną równą (10, -10)),
- C_V_4 – czwarty wierzchołek prostokąta (posiada wartość domyślną równą (10, 10)).

Wartość zwracana: (list) lista krotek reprezentujących punkty.

5.5. gen_set_d

gen_set_d(D_NUM_SIDE=25, D_NUM_DIAGONAL=20, D_V_1=(0,0), D_V_2=(10,0), D_V_3=(10,10), D_V_4=(0,10)) – funkcja generująca losowy zbiór zadanej liczby punktów na przekątnych zadanego prostokąta i zadanej liczbie punktów na dolnym oraz lewym boku tego prostokąta oraz 4 punkty po 1 na każdym wierzchołku tego prostokąta. Domyślnie jest to 25 punktów na przekątnych i 20 punktów na bokach prostokąta zadanego punktami (0, 0), (10, 0), (10, 10), (0, 10) oraz 4 punkty po 1 na każdym z wierzchołków.

Parametry:

- D_NUM_SIDE – liczba losowych punktów na obu bokach prostokąta (posiada wartość domyślną równą 25),
- D_NUM_DIAGONAL – liczba losowych punktów na obu przekątnych prostokąta (posiada wartość domyślną równą 20),
- D_V_1 – pierwszy wierzchołek prostokąta (posiada wartość domyślną równą (0, 0)),
- D_V_2 – drugi wierzchołek prostokąta (posiada wartość domyślną równą (10, 0)),
- D_V_3 – trzeci wierzchołek prostokąta (posiada wartość domyślną równą (10, 10)),
- D_V_4 – czwarty wierzchołek prostokąta (posiada wartość domyślną równą (0, 10)).

Wartość zwracana: (list) lista krotek reprezentujących punkty.

6. Funkcje pomocnicze

6.1. det

det(a, b, c) – funkcja obliczająca wyznacznik z podanych 3 punktów. Wyznacznik liczony jest ze wzoru: $\det(a, b, c) = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}$.

Parametry:

- a – punkt w postaci krotki z dwoma wartościami (x, y),
- b – punkt w postaci krotki z dwoma wartościami (x, y),
- c – punkt w postaci krotki z dwoma wartościami (x, y).

Wartość zwracana: (float) wartość wyznacznika.

6.2. distance

distance(a, b) – funkcja obliczająca odległość między dwoma punktami.

Parametry:

- a – punkt w postaci krotki z dwoma wartościami (x, y),
- b – punkt w postaci krotki z dwoma wartościami (x, y).

Wartość zwracana: (float) odległość między punktami.

6.3. measure_time

measure_time(function, points) – funkcja mierząca czas wykonania funkcji obliczającej otoczkę wypukłą. Funkcja zwraca czas wykonania zaokrąglony do 3 miejsc po przecinku.

Parametry:

- function – funkcja obliczająca otoczkę wypukłą, przyjmująca zbiór punktów,
- points – zbiór punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (float) czas wykonania.

6.4. are_collinear

are_collinear(points) – funkcja sprawdzająca czy wszystkie punkty w podanym zbiorze punktów są współliniowe. Używana jako funkcja pomocnicza w algorytmie dziel i rządź.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (bool) informacja o współliniowości zbioru punktów.

6.5. convert_points_to_lines

convert_points_to_lines(points) – funkcja tworząca listę krawędzi na podstawie listy kolejnych punktów (pomiędzy ostatnim a pierwszym punktem w liście też istnieje krawędź).

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (list) lista linii zadanych dwoma punktami.

6.6. find_minimum_point

find_minimum_point(points) – funkcja wyszukująca i zwracająca najmniejszy leksykograficznie punkt w zbiorze punktów.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (tuple) współrzędne najmniejszego leksykograficznie punktu.

6.7. find_maximum_point

find_maximum_point(points) – funkcja wyszukująca i zwracająca największy leksykograficznie punkt w zbiorze punktów.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (tuple) współrzędne największego leksykograficznie punktu

6.8. find_lowest

find_lowest(points) – funkcja wyszukuje i zwraca indeks punktu o najmniejszej współrzędnej y. Jeśli współrzędne y są równe, to sprawdzane są wartości współrzędnej x i wybierany jest punkt z mniejszą współrzędną.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (tuple) współrzędne punktu najniżej położonego.

7. Algorytm dziel i zwyciężaj (dziel i rządź)

7.1. brute_convex_hull

brute_convex_hull(points) – funkcja wyznaczająca otoczkę wypukłą zbioru punktów. Używana jako funkcja pomocnicza algorytmu dziel i rządź. Wykorzystywany jest algorytm krawędzi skrajnych.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Każda para dwóch różnych punktów tworzy odcinek (potencjalną krawędź otoczki). Dla każdego odcinka sprawdzane jest, ile pozostałych punktów leży na lewo od niego. Jeśli wszystkie punkty (oprócz tych, które wyznaczają daną krawędź) leżą na lewo od odcinka, to krawędź ta należy do otoczki.
2. Mając zbiór krawędzi należących do otoczki wypukłej, wyznaczana jest kolejność punktów otoczki, by były one ustawione przeciwnie do ruchu wskazówek zegara.

Wartość zwracana: (list) otoczka wypukła.

7.2. convex_hull

convex_hull(points) – funkcja wyznaczająca otoczkę wypukłą zbioru punktów. Używana jako funkcja rekurencyjna algorytmu dziel i rządź.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja sprawdza liczbę podanych jako parametr punktów. Jeśli liczba punktów jest mniejsza lub równa 5, to wywoływana jest funkcja brute_convex_hull. Otoczka wypukła zwrócona przez funkcję brute_convex_hull jest zwracana przez funkcję convex_hull.
2. Jeśli liczba podanych punktów jest większa lub równa 6, to zbiór punktów dzielony jest na 2 osobne zbiory: lewy, do którego trafia $\left\lfloor \frac{n}{2} \right\rfloor$ pierwszych punktów i prawy, do którego trafiają pozostałe $\left\lceil \frac{n}{2} \right\rceil$ punkty.
3. Otoczki wypukłe zbiorów lewego i prawego są obliczane rekurencyjnie funkcją convex_hull.
4. Obliczone już otoczki tych zbiorów są scalane przy pomocy funkcji merge.
5. Scalona otoczka wypukła jest zwracana przez funkcję convex_hull.

Wartość zwracana: (list) otoczka wypukła.

7.3. find_leftmost

find_leftmost(points) – funkcja wyszukująca i zwracająca indeks najbardziej wysuniętego na lewo punktu w podanym zbiorze punktów. Używana jako funkcja pomocnicza w algorytmie dziel i rządź.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (tuple) współrzędne punktu wysuniętego najbardziej na lewo.

7.4. find_rightmost

find_rightmost(points) – funkcja wyszukująca i zwracająca indeks najbardziej wysuniętego na prawo punktu w podanym zbiorze punktów. Używana jako funkcja pomocnicza w algorytmie dziel i rządź.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (tuple) współrzędne punktu wysuniętego najbardziej na prawo.

7.5. merge

merge(left, right) – funkcja łącząca dwie otoczki wypukłe w jedną.

Parametry:

- left – lewa otoczka wypukła, lista punktów w postaci krotek z dwoma wartościami (x, y),
- right – prawa otoczka wypukła, lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja sprawdza czy wszystkie punkty lewej i prawej otoczki są współliniowe (do sprawdzania współliniowości zbioru punktów

wykorzystywana jest funkcja are_collinear). Jeśli tak, to jako górny i dolny punkt lewej otoczki zostaje ustawiony skrajnie lewy punkt lewej otoczki (do znajdowania skrajnie lewego / prawego punktu w zbiorze punktów wykorzystywane są funkcje find_leftmost / find_rightmost), a górnym i dolnym punktem prawej otoczki jest skrajnie prawy punkt z prawej otoczki.

2. Jeśli punkty tylko lewej otoczki są współliniowe, to górnym i dolnym punktem lewej otoczki jest skrajnie lewy punkt lewej otoczki. Górny i dolny punkt prawej otoczki są wyznaczane następująco:
    a. Pomocniczy punkt jest ustawiany na skrajnie lewy punkt prawej otoczki. Następnie, dopóki wyznacznik obliczony z punktów: górny punkt lewej otoczki, punkt pomocniczy prawej otoczki, górny sąsiad punktu pomocniczego prawej otoczki jest większy lub równy 0, ustawiam punkt pomocniczy prawej otoczki na jego górnego sąsiada. Po zakończeniu wykonywania pętli, punkt pomocniczy jest górnym punktem prawej otoczki.
    b. Pomocniczy punkt jest ustawiany na skrajnie lewy punkt prawej otoczki. Następnie, dopóki wyznacznik obliczony z punktów: dolny punkt lewej otoczki, punkt pomocniczy prawej otoczki, dolny sąsiad punktu pomocniczego prawej otoczki jest mniejszy lub równy 0, ustawiam punkt pomocniczy prawej otoczki na jego dolnego sąsiada. Po zakończeniu wykonywania pętli, punkt pomocniczy jest dolnym punktem prawej otoczki.
3. Jeśli punkty tylko prawej otoczki są współliniowe, to górnym i dolnym punktem prawej otoczki jest skrajnie prawy punkt prawej otoczki. Górny i dolny punkt lewej otoczki są wyznaczane następująco:
    a. Pomocniczy punkt jest ustawiany na skrajnie prawy punkt lewej otoczki. Następnie, dopóki wyznacznik obliczony z punktów: górny punkt prawej otoczki, punkt pomocniczy lewej otoczki, górny sąsiad punktu pomocniczego lewej otoczki jest mniejszy lub równy 0, ustawiam punkt pomocniczy na jego górnego sąsiada. Po zakończeniu wykonywania pętli, punkt pomocniczy jest górnym punktem lewej otoczki.

b. Pomocniczy punkt jest ustawiany na skrajnie prawy punkt lewej otoczki. Następnie, dopóki wyznacznik obliczony z punktów: dolny punkt prawej otoczki, punkt pomocniczy lewej otoczki, dolny sąsiad punktu pomocniczego lewej otoczki jest większy lub równy 0, ustawiam punkt pomocniczy na jego dolnego sąsiada. Po zakończeniu wykonywania pętli, punkt pomocniczy jest dolnym punktem lewej otoczki.

4. Jeśli punkty obydwu zbiorów nie są współliniowe, to wyznaczane są górne i dolne punkty obydwu otoczek wypukłych:
   a. Punktem pomocniczym lewej otoczki zostaje skrajnie prawy punkt lewej otoczki. Punktem pomocniczym prawej otoczki zostaje skrajnie lewy punkt prawej otoczki. Następnie w nieskończonej pętli wykonywane są dwie pętle wyznaczające górne punkty lewej i prawej otoczki:
      i. Dopóki wyznacznik obliczony z punktów: prawy punkt pomocniczy, lewy punkt pomocniczy, górny sąsiad lewego punktu pomocniczego jest mniejszy lub równy 0, lewym punktem pomocniczym staje się górny sąsiad lewego punktu pomocniczego.
      ii. Dopóki wyznacznik obliczony z punktów: lewy punkt pomocniczy, prawy punkt pomocniczy, górny sąsiad prawego punktu pomocniczego jest większy lub równy 0, prawym punktem pomocniczym staje się górny sąsiad prawego punktu pomocniczego. Jeśli pętla ta nie wykona się ani razu w danej iteracji pętli nieskończonej, to pętla nieskończona jest przerywana. Po przerwaniu iteracji, lewy punkt pomocniczy jest górnym punktem lewej otoczki, a prawy punkt pomocniczy jest górnym punktem prawej otoczki.
   b. Punktem pomocniczym lewej otoczki zostaje skrajnie prawy punkt lewej otoczki. Punktem pomocniczym prawej otoczki zostaje skrajnie lewy punkt prawej otoczki. Następnie w nieskończonej pętli wykonywane są dwie pętle wyznaczające dolne punkty lewej i prawej otoczki.
      i. Dopóki wyznacznik obliczony z punktów: lewy pomocniczy, prawy pomocniczy, dolny sąsiad prawego

pomocniczego jest mniejszy lub równy 0, prawym punktem pomocniczym staje się jego dolny sąsiad.

ii. Dopóki wyznacznik obliczony z punktów: prawy punkt pomocniczy, lewy punkt pomocniczy, dolny sąsiad lewego punktu pomocniczego jest większy lub równy 0, lewym punktem pomocniczym staje się jego dolny sąsiad. Jeśli ta pętla nie wykona się ani razu w danej iteracji pętli nieskończonej, to pętla nieskończona jest przerywana. Po przerwaniu iteracji, lewy punkt pomocniczy jest dolnym punktem lewej otoczki, a prawy punkt pomocniczy jest dolnym punktem prawej otoczki.

5. Mając wyznaczone górne i dolne punkty obydwu otoczek, tworzona jest otoczka wynikowa. Algorytm zaczyna przechodzenie po punktach od górnego punktu lewej otoczki, dodając wszystkie punkty po drodze do dolnego punktu lewej otoczki (włącznie). Następnie do otoczki wynikowej dodawane są wszystkie punkty z prawej otoczki od dolnego punktu do górnego punktu (włącznie).

Wartość zwracana: (list) złączenie dwóch otoczek wypukłych.

7.6. divide_and_conquer

divide_and_conquer(points) – funkcja wyznaczająca otoczkę wypukłą korzystając z algorytmu dziel i rządź. Funkcja zwraca indeksy punktów należących do otoczki wypukłej w kolejności przeciwnej do ruchu wskazówek zegara.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Początkowo program sprawdza, czy liczba podanych punktów jest mniejsza niż 3. Jeśli tak, to otoczka wypukła nie istnieje, a funkcja zwraca None.
2. Następnie tworzone jest odwzorowanie każdego punktu na odpowiadający mu indeks w liście.
3. Podany zbiór punktów jest sortowany według współrzędnej x (jeśli współrzędne x są równe, to sortowanie przebiega po współrzędnej y). Posortowanie punktów ułatwi podział punktów według ich mediany, przy podziale na lewą i prawą otoczkę w funkcji convex_hull.
4. Wywoływana jest funkcja convex_hull zwracająca punkty należące do otoczki wypukłej.
5. Korzystając z utworzonego odwzorowania, punkty zamieniane są na ich indeksy w tablicy wejściowej. Lista z indeksami punktów jest zwracana przez funkcję.
6. Całkowita złożoność algorytmu: $O(n \log n)$.

Wartość zwracana: (list) otoczka wypukła.

7.7. divide_and_conquer_visualisation

divide_and_conquer_visualisation(points) – funkcja wizualizująca działanie algorytmu dziel i rządź. Zwraca sceny możliwe do odczytania przez narzędzie graficzne.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji divide_and_conquer.
2. Wszystkie punkty w zbiorze są przedstawiane na niebiesko.
3. Punkty należące do otoczki wypukłej są przedstawiane na czerwono.

Wartość zwracana: (list) wizualizacja.

7.8. convex_hull_visualisation

convex_hull_visualisation(points, all_points, scenes) – funkcja wizualizująca działanie funkcji convex_hull.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y),
- all_points – lista wszystkich punktów przekazanych do funkcji divide_and_conquer_visualisation (wykorzystywane do dodawania nowych scen),
- scenes – lista scen wizualizacji.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji convex_hull.

Wartość zwracana: (list) wizualizacja.

7.9. merge_visualisation

merge_visualisation(left, right, all_points, scenes) – funkcja wizualizująca działanie funkcji merge.

Parametry:

- left – lewa otoczka wypukła, lista punktów w postaci krotek z dwoma wartościami (x, y),
- right – prawa otoczka wypukła, lista punktów w postaci krotek z dwoma wartościami (x, y).
- all_points – lista wszystkich punktów przekazanych do funkcji divide_and_conquer_visualisation.
- scenes – lista scen wizualizacji.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji merge.
2. Wszystkie punkty zbioru przedstawiane są na niebiesko.
3. Punkty należące do lewej otoczki wypukłej zaznaczane są na czerwono.
4. Punkty należące do prawej otoczki wypukłej zaznaczane są na żółto.
5. Przy wyznaczaniu górnej i dolnej stycznej między otoczkami, styczna robocza jest przedstawiana na zielono.
6. Już wyznaczona styczna górna ma kolor pomarańczowy.

Wartość zwracana: (list) wizualizacja.

8. Algorytm przyrostowy

8.1. incremental

incremental(points) – funkcja wyznaczająca otoczkę wypukłą korzystając z algorytmu przyrostowego. Funkcja zwraca indeksy punktów tworzących otoczkę w kolejności przeciwnej do ruchu wskazówek zegara.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja importuje strukturę SortedList z biblioteki sortedcontainers. SortedList umożliwia przechowywanie elementów w liście, z dodawaniem, usuwaniem i wstawianiem elementów w czasie $O(\log n)$.
2. Program sprawdza, czy liczba podanych punktów jest mniejsza niż 3. Jeśli tak, to otoczka wypukła nie istnieje, a funkcja zwraca None.
3. Tworzone jest odwzorowanie każdego punktu na odpowiadający mu indeks w liście.
4. Podany zbiór punktów jest sortowany według współrzędnej x (jeśli współrzędne x punktów są równe, to sortowanie przebiega względem współrzędnej y). Posortowanie punktów ułatwi sprawdzanie, czy rozpatrywany punkt należy do otoczki wypukłej – przy przechodzeniu po kolei po posortowanych punktach, każdy kolejny punkt nie należy do otoczki.
5. Tworzona jest otoczka wypukła z 3 pierwszych punktów w liście.
6. Dla każdego punktu (oprócz 3 początkowych) wykonywana jest iteracja pętli. W każdej iteracji znajdowane są punkty otoczki będące stycznymi rozpatrywanego punktu do otoczki za pomocą funkcji find_left_tangent i find_right_tangent. Wszystkie punkty znajdujące się obecnie w otoczce pomiędzy punktami otoczki będącymi stycznymi są usuwane. W ich miejsce umieszczany jest rozpatrywany punkt.

7. Po ukończeniu głównej pętli funkcji, wykorzystywane jest utworzone wcześniej odwzorowanie, by listę punktów zamienić na listę indeksów. Lista z indeksami jest zwracana przez funkcję.
8. Całkowita złożoność algorytmu: $O(n \log n)$.

Wartość zwracana: (list) otoczka wypukła.

8.2. find_left_tangent

find_left_tangent(point, hull) – funkcja znajdująca binarnie lewą styczną z punktu do otoczki.

Parametry:

- point – punkt w postaci krotki z dwoma wartościami (x, y),
- hull – obiekt klasy SortedList zawierający otoczkę wypukłą.

Działanie algorytmu:

1. Lewy zakres wyszukiwania binarnego ustawiany jest na pierwszy punkt otoczki.
2. Prawy zakres wyszukiwania binarnego ustawiany jest na ostatni punkt otoczki.
3. Wykonywana jest pętla, działająca, dopóki lewy koniec zakresu jest mniejszy lub równy prawemu końcowi zakresu.
4. Środkowy punkt wyszukiwania binarnego ustawiany jest na punkt leżący na środku między lewym, a prawym końcem zakresu.
5. Rezultat obliczeń ustawiany jest początkowo na indeks równy -1.
6. Obliczanie są 3 wyznaczniki (dla każdego z punktów: lewy koniec, prawy koniec, środek) według następującego schematu punktów: rozpatrywany punkt, lewy / prawy / środek, następnik lewego / prawego / środkowego.
7. Jeśli wyznacznik powiązany z punktem środkowym jest mniejszy lub równy 0, to rezultat ustawiany jest na punkt środkowy, a prawy koniec zakresu na poprzednik punktu środkowego.
8. W przeciwnym wypadku, jeśli:
    a. Wyznacznik powiązany z lewym końcem zakresu jest mniejszy lub równy 0, to prawy koniec zakresu jest ustawiany jako poprzednik punktu środkowego.

   b. Wyznacznik powiązany z prawym końcem zakresu jest mniejszy lub równy 0, to lewy koniec zakresu jest ustawiany na następnika punktu środkowego.
   c. Poprzednie wyznaczniki są większe od 0, to obliczany jest kolejny wyznacznik powiązany z punktami: rozpatrywany punkt, środkowy punkt, skrajnie lewy punkt. Jeśli wyznacznik ten jest większy od 0, to prawy koniec zakresu ustawiany jest na poprzednika punktu środkowego. W przeciwnym wypadku lewy koniec zakresu ustawiany jest na następnika punktu środkowego.
9. Funkcja zwraca indeks punktu stycznego zapisanego jako rezultat.

Wartość zwracana: (float) współrzędne punktu stycznego po lewej.

8.3. find_right_tangent

find_right_tangent(point, hull) – funkcja znajdująca binarnie prawą styczną z punktu do otoczki.

Parametry:

- point – punkt w postaci krotki z dwoma wartościami (x, y),
- hull – obiekt klasy SortedList zawierający otoczkę wypukłą.

Działanie algorytmu:

1. Lewy zakres wyszukiwania binarnego ustawiany jest na pierwszy punkt otoczki.
2. Prawy zakres wyszukiwania binarnego ustawiany jest na ostatni punkt otoczki.
3. Wykonywana jest pętla, działająca, dopóki lewy koniec zakresu jest mniejszy lub równy prawemu końcowi zakresu.
4. Środkowy punkt wyszukiwania binarnego ustawiany jest na punkt leżący na środku między lewym, a prawym końcem zakresu.
5. Rezultat obliczeń ustawiany jest początkowo na indeks równy -1.
6. Obliczanie są 3 wyznaczniki (dla każdego z punktów: lewy koniec, prawy koniec, środek) według następującego schematu punktów:

rozpatrywany punkt, lewy / prawy / środek, następnik lewego / prawego / środkowego.

7. Jeśli wyznacznik powiązany z punktem środkowym jest mniejszy lub równy 0, to rezultat ustawiany jest na punkt środkowy, a lewy koniec zakresu jest ustawiany na następnik punktu środkowego.
8. W przeciwnym wypadku, jeśli:
    a. Wyznacznik powiązany z lewym końcem zakresu jest mniejszy lub równy 0, to prawy koniec zakresu ustawiany jest na poprzednika punktu środkowego.
    b. Wyznacznik powiązany z prawym końcem zakresu jest mniejszy lub równy 0, to lewy koniec zakresu jest ustawiany na następnika punktu środkowego.
    c. Poprzednie wyznaczniki są większe od 0, to obliczany jest kolejny wyznacznik powiązany z punktami: rozpatrywany punkt, punkt środkowy, lewy koniec zakresu. Jeśli wyznacznik ten jest większy od 0, to prawy koniec zakresu jest ustawiany na poprzednika punktu środkowego. W przeciwnym wypadku lewy koniec zakresu ustawiany jest na następnika punktu środkowego.
9. Funkcja zwraca indeks punktu stycznego będącym następnikiem punktu zapisanego jako rezultat.

Wartość zwracana: (float) współrzędne punktu stycznego po prawej.

8.4. incremental_visualisation

incremental_visualisation(points) – funkcja wizualizująca działanie algorytmu przyrostowego. Funkcja zwraca sceny możliwe do odczytania przez narzędzie graficzne.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji incremental.
2. Wszystkie punkty przekazane do funkcji prezentowane są na niebiesko.

3. Punkty należące do utworzonej otoczki prezentowane są na czerwono.
4. Obecnie rozpatrywany punkt wraz z wyznaczonymi stycznymi prezentowane są na żółto.

Wartość zwracana: (list) wizualizacja.

8.5. compare

compare(a, b) – funkcja określająca kolejność dwóch punktów przy sortowaniu względem kąta, między odcinkiem łączącym dany punkt z punktem zbioru zawierającym najmniejszą współrzędną y, a dodatnim kierunkiem osi x. Wykorzystywana jako funkcja pomocnicza funkcji graham.

Parametry:

- a – punkt w postaci krotki z dwoma wartościami (x, y),
- b – punkt w postaci krotki z dwoma wartościami (x, y).

Działanie algorytmu:

1. Używana jest zmienna globalna p, zawierająca punkt zbioru z najmniejszą współrzędna y. Obliczany jest wyznacznik z punktów: punkt p, punkt a, punkt b.
2. Jeśli wyznacznik jest równy 0, to obliczane są odległości punktów a i b od punktu p. Jeśli odległość p-a jest większa lub równa odległości p-b, to zwracana jest liczba 1. W przeciwnym wypadku zwracana jest liczba -1.
3. Jeśli wyznacznik jest większy od 0, to zwracana jest wartość -1.
4. W przypadku wartości wyznacznika mniejszej od 0, zwracana jest wartość 1.

Wartość zwracana: (int) wartość porównania {-1, 0, 1}.

8.6. filter_points

filter_points(points) – funkcja filtrująca punkty po ich wcześniejszym posortowaniu względem kąta przy wykorzystaniu funkcji compare. Funkcja tworzy nowy zbiór punktów, w którym znajdują się tylko te punkty z wejściowego zbioru, które nie są współliniowe względem punktu p o najmniejszej współrzędnej y.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Jeśli liczba punktów jest większa niż 0 i wszystkie punkty są współliniowe, to funkcja zwraca listę składającą się z dwóch punktów: najmniejszego i największego leksykograficznie.
2. W przeciwnym przypadku, jeśli liczba punktów jest większa niż 1, to w pętli do nowego zbioru punktów przepisywane są tylko te punkty, które dla wyznacznika składającego się z punktów: punkt p o najmniejszej współrzędnej y, obecnie rozpatrywany punkt, kolejny rozpatrywany punkt przyjmują wartości różne od 0.
3. Zwracany jest nowo utworzony zbiór punktów.

Wartość zwracana: (list) lista punktów po filtrowaniu.

9. Algorytm Quickhull

9.1. line_point_distance

line_point_distance(point, line_beginning, line_end) – funkcja obliczająca odległość punktu od prostej. Punkt reprezentowany jest jako krotka dwóch liczb rzeczywistych odpowiadająca kolejno wartości współrzędnej x i y, prosta zadana jest jako dwa punkty – początek i koniec odcinka zawierającego się w prostej, względem której liczymy odległość. W przypadku tej funkcji zadanie punktów odcinka zawierającego się w prostej w odwrotnej kolejności (to jest najpierw punktu końcowego, a następnie początkowego) nie wpłynie na uzyskany wynik, kolejność ta ma jednak znaczenie przy obliczaniu wyznacznika i wyznaczaniu orientacji punktów (w przypadku podania punktów w odwrotnej kolejności możemy uzyskać odwrotną orientację). W przypadku, gdy punkt leży na prostej, funkcja zwraca wartość 0 (zgodnie z definicją matematyczną) nie wykonując dodatkowych obliczeń (funkcja jest zabezpieczona przed próbą dzielenia przez 0).

Parametry:

- point (tuple) – punkt zadany krotką dwóch liczb rzeczywistych (odpowiednio współrzędnej x i y),
- line_beginning (tuple) – pierwszy punkt zadający prostą wyrażony krotką dwóch liczb rzeczywistych (odpowiednio współrzędnej x i y),
- line_end (tuple) – drugi punkt zadający prostą wyrażony krotką dwóch liczb rzeczywistych (odpowiednio współrzędnej x i y).

Wartość zwracana: (float) odległość między punktem a prostą.

9.2. part_quickhull

part_quickhull(line_beginning, line_end) – funkcja pomocnicza dla funkcji quickhull.

Parametry:

- points_set – wejściowy zbiór punktów,
- line_beginning – początek odcinka zawierającej się w prostej dzielącej zbiór na górny i dolny,
- line_end – koniec odcinka zawierającej się w prostej dzielącej zbiór na górny i dolny,
- subset – podzbiór punktów (górny lub dolny) wyznaczony przez poprzedni podział punktów na górne i dolne względem prostej przechodzącej przez dwa punkty,
- scenes – kolejne sceny prezentujące kolejne kroki wykonania algorytmu,
- hull_points – punkty, które już zostały dodane do otoczki,
- hull_lines – obecny stan krawędzi znajdujących się w otoczce.

Działanie algorytmu:

1. Funkcja stanowi funkcję pomocniczą dla funkcji quickhull.
2. Algorytm znajduje w przekazanym do funkcji podzbiorze punkt najdalej położony od przekazanej do funkcji prostej w przekazanym podzbiorze.
3. Algorytm łączy znaleziony punkt z końcami odcinka przekazanymi do funkcji.
4. Linie łączące punkty są odcinkami zawierającymi się w prostych, względem których algorytm dzieli otrzymany wcześniej podzbiór na punkty znajdujące się powyżej i poniżej prostych.
5. Dla wyznaczonych w podziale podzbiorów punktów powyżej i poniżej prostych funkcja wywołuje się rekurencyjnie i działa, dopóki przekazany podzbiór nie jest pusty.
6. Algorytm łączy wszystkie wyznaczone rekurencyjnie fragmenty otoczki i tak wyznaczoną część wynikowej otoczki przekazuje je do pierwotnej funkcji quickhull, która łącząc wszystkie otrzymane fragmenty z wywołań funkcji pomocniczej zwraca wynikową otoczkę wypukłą.

Wartość zwracana: (list) wyznaczony rekurencyjnie fragment otoczki.

9.3. quickhull

quickhull(points_set) – funkcja wyznaczająca otoczkę wypukłą zbioru punktów na płaszczyźnie dwuwymiarowej za pomocą algorytmu quickhull.

Parametry:

- points_set – zbiór punktów, dla którego algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

1. Algorytm wyszukuje punkt położony najbardziej na lewo i najbardziej na prawo (kolejno najmniejsza i największa wartość współrzędnej x).
2. Te dwa punkty wyznaczają prostą, względem której dzielimy punkty na te położone wyżej i niżej prostej (po prawej lub po lewej odcinka o początku i końcu w wyznaczonych punktach).
3. Algorytm wyznacza punkt najdalej położony od prostej za pomocą funkcji max i line_point_distance.
4. Tak podzielony zbiór wejściowych punktów na zbiory punktów powyżej i poniżej oraz wyznaczony punkt położony najdalej od prostej przekazywane są do funkcji pomocniczej part_quickhull, która rekurencyjnie wyznacza kolejne fragmenty otoczki postępując w analogiczny sposób.
5. Funkcja otrzymuje od funkcji pomocniczej fragmenty wynikowej otoczki wypukłej i łączy je w całość uzyskując wynikową otoczkę wypukłą, którą zwraca na wyjście w postaci listy kolejnych punktów.

Wartość zwracana: (list) otoczka wypukła.

9.4. part_quickhull_visualisation

part_quickhull_visualisation(points_set, line_beginning, line_end, subset, scenes, hull_points, hull_lines) – funkcja pomocnicza dla funkcji quickhull_visualisation.

Parametry:

- points_set – wejściowy zbiór punktów,
- line_beginning – początek odcinka zawierającej się w prostej dzielącej zbiór na górny i dolny,
- line_end – koniec odcinka zawierającej się w prostej dzielącej zbiór na górny i dolny,
- subset – podzbiór punktów (górny lub dolny) wyznaczony przez poprzedni podział punktów na górne i dolne względem prostej przechodzącej przez dwa punkty,
- scenes – kolejne sceny prezentujące kolejne kroki wykonania algorytmu,
- hull_points – punkty, które już zostały dodane do otoczki,
- hull_lines – obecny stan krawędzi znajdujących się w otoczce.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji part_quickhull.
2. Algorytm analogicznie wyznacza kolejne fragmenty wynikowej otoczki wypukłej oraz prezentuje swoje działania.
3. Na obecny etap wyznaczania otoczki składają się linie i punkty pokolorowane na limonkowo.
4. Dotychczasowe etapy mają kolor czarny.

Wartość zwracana: (list) znaleziony fragment otoczki wypukłej oraz dodane sceny kolejnych etapów działania algorytmu.

9.5. quickhull_visualisation

quickhull_visualisation(points_set) – funkcja prezentująca działanie algorytmu quickhull. Zwraca listę kolejnych scen ukazujących kolejne kroki algorytmu oraz odpowiednio koloruje punkty i dodaje kolorowe krawędzie otoczki oraz wskazuje sprawdzane w danym kroku dane.

Parametry:

- points_set – lista stanowiąca zbiór punktów, dla których algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji quickhull.
2. Wszystkie punkty w zbiorze na początku reprezentowane są kolorem błękitnym.
3. Kolejne etapy wyznaczania otoczki wypukłej mają kolor czarny.
4. Obecny etap ma kolor limonkowy.
5. Wynikowa otoczka wypukła ma kolor czerwony.

Wartość zwracana: (list) wizualizacja.

10. Algorytm górnej i dolnej otoczki

10.1. upper_lower_hull

upper_lower_hull(points_set) – funkcja wyznaczająca otoczkę wypukłą algorytmem górnej i dolnej otoczki.

Parametry:

- points_set – zbiór punktów, dla którego algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

1. Funkcja sortuje punkty leksykograficznie (rosnąco względem współrzędnej x, a w drugiej kolejności – w przypadku kilku punktów o tej samej współrzędnej x – względem współrzędnej y).
2. Algorytm tworzy listę z punktami powyżej i poniżej. Do listy powyżej dodaje dwa pierwsze punkty z posortowanej listy z punktami.
3. Następnie przechodzi po wszystkich punktach i dodaje każdy kolejny punkt do listy.
4. Algorytm sprawdza, czy teraz w liście trzy ostatnie punkty mają ujemny wyznacznik, jeśli nie to usuwa ostatni punkt w liście, dopóki w liście są punkty i wyznacznik jest nieujemny – czyli ostatni punkt znajduje się na prawo od prostej wyznaczonej przez przedostatni i ostatni za nim.
5. Po przejściu przez wszystkie punkty, w liście upper zapisana jest górna otoczka, algorytm działa analogicznie dla dolnej otoczki operując na liście lower, w której znajdzie się dolna otoczka. Do obliczeń również wykorzystywany jest wyznacznik.
6. Funkcja łączy obie otoczki, które w wyniku dają właściwą otoczkę.

Wartość zwracana: (list) otoczka wypukła.

10.2. upper_lower_hull_visualisation

upper_lower_hull_visualisation(points_set) – funkcja prezentująca działanie algorytmu górnej i dolnej otoczki. Zwraca listę kolejnych scen ukazujących kolejne kroki algorytmu oraz odpowiednio koloruje punkty i dodaje kolorowe krawędzie otoczki oraz wskazuje sprawdzane w danym kroku dane.

Parametry:

- points_set – zbiór punktów, dla którego algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji upper_lower_hull.
2. Funkcja wyznacza otoczkę wypukłą i prezentuje kolejne etapy działania.
3. Punkty przekazane do funkcji, które nie leżą na otoczce wypukłej, ani nie są obecnie na nich przeprowadzane obliczenia mają kolor błękitny.
4. Obecnie sprawdzany punkt i krawędź do niego poprowadzona mają kolor limonowy.
5. Wynikowa otoczka wypukła oraz etapy jej wyznaczania w danym kroku mają kolor czerwony.

Wartość zwracana: (list) wizualizacja.

11. Algorytm Jarvisa

11.1. jarvis

jarvis(points) – funkcja wyznaczająca dla zbioru punktów na płaszczyźnie dwuwymiarowej otoczkę wypukłą za pomocą algorytmu Jarvisa.

Parametry:

- points (list) – lista punktów reprezentowanych w liście jako krotka dwóch liczb rzeczywistych będących kolejno wartościami współrzędnej x i y, dla których algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

1. Funkcja szuka punktu najmniejszego leksykograficznie (o najmniejszej współrzędnej x, a w przypadku kilku takich punktów wybiera ten o najmniejszej współrzędnej y) jest to punkt początkowy.
2. Algorytm znajduje kolejny punkt, który tworzy z ostatnim punktem w otoczce największy kąt w odpowiednim kierunku, a jeśli takich punktów jest kilka to wybiera ten oddalony najdalej od ostatniego punktu w otoczce, wykorzystując do obliczeń wyznacznik i funkcję obliczającą odległość między punktami.
3. Algorytm wyszukuje kolejny punkt spośród wszystkich punktów, do momentu aż wróci do punktu początkowego.
4. Algorytm zwraca listę kolejnych punktów tworzących otoczkę wypukłą.

Wartość zwracana: (list) otoczka wypukła.

11.2. jarvis_visualisation

jarvis_visualisation(points) – funkcja prezentująca wizualizację działania algorytmu Jarvisa, generuje kolejne etapy wyznaczania otoczki wypukłej, przedstawia kolejne punkty i krawędzie otoczki, obecnie sprawdzaną krawędź i punkt oraz je koloruje.

Parametry:

- points (list) – lista punktów reprezentowanych w liście jako krotka dwóch liczb rzeczywistych będących kolejno wartościami współrzędnej x i y, dla których algorytm wyznacza otoczkę wypukłą.

Działanie algorytmu:

- Funkcja działa analogicznie do funkcji jarvis.
- Wszystkie punkty w zbiorze reprezentowane są kolorem błękitnym.
- Wynikowa otoczka wypukła składa się z linii i punktów reprezentowanych kolorem czerwonym.
- Obecnie sprawdzany punkt i krawędź będące kandydatami na znalezienie się na otoczce wypukłej reprezentowane są kolorem zielonym.

Wartość zwracana: (list) wizualizacja.

12. Algorytm Grahama

12.1. graham

graham(points) – funkcja wyznaczająca otoczkę wypukłą zbioru punktów, korzystając z algorytmu Grahama. Funkcja używana jako funkcja pomocnicza do obliczania otoczki wypukłej korzystając z algorytmu Chana.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Korzystając z funkcji find_lowest znajdowany jest punkt o najmniejszej współrzędnej y. Jest on umieszczany w zmiennej globalnej p i usuwany z listy punktów.
2. Punkty są sortowane korzystając z funkcji sort i komparatora compare.
3. Punkty współliniowe są filtrowane przy pomocy funkcji filter_points.
4. Jeśli liczba punktów po filtrowaniu jest mniejsza niż 3, to do listy punktów ponownie dołączany jest punkt p. Jeśli punkty te są współliniowe, to zwracana jest lista dwóch punktów składająca się z najmniejszego i największego leksykograficznie punktów. W przeciwnym przypadku zwracana jest zbiór punktów.
5. Jeśli liczba punktów po filtrowaniu jest większa lub równa 3, to tworzony jest stos składający się kolejno z: punktu o najmniejszej współrzędnej y i pierwszych dwóch punktów z przefiltrowanego zbioru punktów.
6. Wykonywana jest zewnętrzna pętla, przechodząca po wszystkich punktach zbioru z wyłączeniem dwóch pierwszych. W wewnętrznej pętli sprawdzane jest, czy 2 ostatnie punkty ze stosu tworzą z obecnie rozpatrywanym punktem zakręt w lewo: Jeśli tak, to pętla wewnętrzna jest przerywana. Jeśli nie, to usuwany jest punkt ze szczytu stosu. Po zakończeniu wykonywania pętli wewnętrznej, obecnie rozpatrywany punkt jest dodawany na stos.

7. Po zakończeniu wszystkich iteracji, stos zawiera kolejne punkty otoczki wypukłej, które są zwracane przez funkcję.

Wartość zwracana: (list) otoczka wypukła.

12.2. graham_visualisation

graham_visualisation(points) – funkcja wizualizująca działanie algorytmu Grahama. Zwraca listę scen możliwych do odczytania przez narzędzie graficzne.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji graham.
2. Wszystkie punkty w zbiorze prezentowane są na niebiesko.
3. Wynikowa otoczka wypukła prezentowana jest na czerwono.

Wartość zwracana: (list) wizualizacja.

12.3. find_tangent

find_tangent(hull, point) – funkcja wyszukująca binarnie styczną z punktu do otoczki.

Parametry:

- hull – lista punktów otoczki wypukłej w postaci krotek z dwoma wartościami (x, y),
- point – punkt w postaci krotki z dwoma wartościami (x, y).

Działanie algorytmu:

1. Lewy koniec zakresu wyszukiwania ustawiany jest na pierwszy punkt otoczki.
2. Prawy koniec zakresu wyszukiwania ustawiany jest na ostatni punkt otoczki.
3. Wykonywana jest pętla, działająca, dopóki lewy koniec zakresu jest mniejszy lub równy prawemu końcowi zakresu.

4. Środkowy punkt wyszukiwania binarnego ustawiany jest na punkt leżący na środku między lewym, a prawym końcem zakresu.
5. Rezultat obliczeń ustawiany jest początkowo na indeks równy -1
6. Obliczanie są 3 wyznaczniki (dla każdego z punktów: lewy koniec, prawy koniec, środek) według następującego schematu punktów: rozpatrywany punkt, lewy / prawy / środek, następnik lewego / prawego / środkowego.
7. Jeśli wyznacznik powiązany z punktem środkowym jest mniejszy lub równy 0, to rezultat ustawiany jest na punkt środkowy, a lewy koniec zakresu jest ustawiany na następnik punktu środkowego.
8. W przeciwnym wypadku, jeśli:
    a. Wyznacznik powiązany z lewym końcem zakresu jest mniejszy lub równy 0, to prawy koniec zakresu ustawiany jest na poprzednika punktu środkowego.
    b. Wyznacznik powiązany z prawym końcem zakresu jest mniejszy lub równy 0, to lewy koniec zakresu jest ustawiany na następnika punktu środkowego.
    c. Poprzednie wyznaczniki są większe od 0, to obliczany jest kolejny wyznacznik powiązany z punktami: rozpatrywany punkt, punkt środkowy, lewy koniec zakresu. Jeśli wyznacznik ten jest większy od 0, to prawy koniec zakresu jest ustawiany na poprzednika punktu środkowego. W przeciwnym wypadku lewy koniec zakresu ustawiany jest na następnika punktu środkowego.
9. Funkcja zwraca indeks punktu stycznego będącym następnikiem punktu zapisanego jako rezultat.

Wartość zwracana: (int) indeks punktu.

12.4. find_extreme_point

find_extreme_point(hulls) – zwraca indeks punktu z najmniejszą współrzędną y wśród punktów wszystkich otoczek.

Parametry:

- hulls – lista zawierająca obliczone już otoczki wypukłe. Każda z otoczek wypukłych jest listą punktów składających się na tę otoczkę.

Wartość zwracana: (int) indeks punktu.

12.5. find_next_point

find_next_point(hulls, coordinates) – funkcja wyznaczająca indeksykolejnego punktu należącego do otoczki wypukłej.

Parametry:

- hulls – lista zawierająca obliczone już otoczki wypukłe. Każda z otoczek wypukłych jest listą punktów składających się na tę otoczkę,
- coordinates – krotka z dwoma wartościami określającymi indeks otoczki wypukłej oraz indeks punktu w tej otoczce. Indeksy te określają punkt zbioru wejściowego.

Działanie algorytmu:

1. Rezultat ustawiany jest na następny punkt bieżącej otoczki.
2. Dla każdej otoczki (z wyjątkiem obecnej) wyznaczana jest styczna, korzystając z funkcji find_tangent.
3. Każda obliczona styczna porównywana jest z rezultatem za pomocą wyznacznika – jeśli styczna tworzy większy kąt niż obecny rezultat, lub tworzy taki sam kąt, ale odległość między punktami jest większa, to obecna styczna staje się rezultatem.
4. Po zakończeniu wszystkich iteracji zwracany jest rezultat – indeksy kolejnego punktu otoczki wypukłej.

Wartość zwracana: (int) indeks punktu.

13. Algorytm Chana

13.1. partial_hull

partial_hull(points, m) – funkcja obliczająca otoczkę wypukłą zbioru punktów dla danej aproksymacji liczby punktów w otoczce – m. Funkcja używana jako funkcja pomocnicza funkcji chan.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y),
- m – aproksymacja liczby punktów w wynikowej otoczce wypukłej.

Działanie algorytmu:

1. Obliczana jest liczba pomniejszych otoczek wypukłych ze wzoru $r = \left\lceil \frac{n}{m} \right\rceil$, gdzie n to liczba punktów w zbiorze wejściowym.
2. Wejściowy zbiór punktów dzielony jest na r zbiorów, w którym każdy (potencjalnie oprócz ostatniego) zawiera m punktów. Dla każdego z tych zbiorów obliczana jest otoczka wypukła korzystając z funkcji graham. Obliczone otoczki wypukłe przechowywane są w liście otoczek.
3. Tworzona jest lista zawierająca punkty obliczanej otoczki. Początkowo umieszcza się w niej indeksy punktu o najmniejszej współrzędnej y.
4. W pętli, która wykonuje się maksymalnie m razy obliczane są indeksy kolejnego punktu otoczki. Jeśli kolejny punkt jest jednocześnie punktem początkowym otoczki, to otoczka została znaleziona. Indeksy punktów zamieniane są na punkty i ich lista jest zwracana przez funkcję. Jeśli kolejny punkt nie jest punktem początkowym, to jest dodawany do tworzonej otoczki.
5. Jeśli po m iteracjach nie znaleziono otoczki wypukłej, to funkcja zwraca wartość None.

Wartość zwracana: (list) otoczka wypukła.

13.2. chan

chan(points) – funkcja wyznaczająca otoczkę wypukła za pomocą algorytmu Chana. Funkcja zwraca indeksy punktów należących do otoczki wypukłej w kolejności przeciwnej do ruchu wskazówek zegara.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Początkowo program sprawdza, czy liczba podanych punktów jest mniejsza niż 3. Jeśli tak, to otoczka wypukła nie istnieje, a funkcja zwraca None.
2. Następnie tworzone jest odwzorowanie każdego punktu na odpowiadający mu indeks w liście.
3. Wykonywana jest pętla z iteratorem t z początkową wartością 1. Dla każdej iteracji aproksymacja m równa jest $m = \min(2^{2^t}, n)$, gdzie n to liczba punktów w liście.
4. Dla danej wartości m wywoływana jest funkcja partial_hull.
5. Jeśli funkcja partial_hull znalazła otoczkę wypukłą, to korzystając z utworzonego odwzorowania, punkty zamieniane są na ich indeksy w tablicy wejściowej. Lista z indeksami punktów jest zwracana przez funkcję. W przeciwnym wypadku zwiększana jest wartość iteratora t o 1.
6. Całkowita złożoność: $O(n \log h)$.

Wartość zwracana: (list) otoczka wypukła.

13.3. make_segments

make_segments(points) – funkcja tworząca listę kolejnych krawędzi na podstawie listy kolejnych punktów. Pomiędzy ostatnim i pierwszym punktem nie jest tworzona krawędź.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Wartość zwracana: (list) lista krawędzi w postaci list dwóch krotek dwóch liczb rzeczywistych reprezentujących współrzędne dwóch punktów stanowiących końce krawędzi.

13.4. partial_hull_visualisation

partial_hull_visualisation(points, m, scenes) – funkcja wizualizująca działanie funkcji partial_hull.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y),
- m – aproksymacja liczby punktów w wynikowej otoczce wypukłej,
- scenes – lista scen wizualizacji.

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji partial_hull.
2. Wszystkie punkty zbioru prezentowane są na niebiesko.
3. Utworzone pomniejsze otoczki wypukłe są prezentowane kolorami: zielony, żółty, ciemno niebieski, brązowy, fioletowy, czarny.
4. Tworzona otoczka jest prezentowana na czerwono.

Wartość zwracana: (list) wizualizacja.

13.5. chan_visualisation

chan_visualisation(points) – funkcja wizualizująca działanie algorytmu Chana. Zwraca listę scen możliwych do odczytania przez narzędzie graficzne.

Parametry:

- points – lista punktów w postaci krotek z dwoma wartościami (x, y).

Działanie algorytmu:

1. Funkcja działa analogicznie do funkcji chan.
2. Wszystkie punkty w zbiorze prezentowane są na niebiesko.
3. Wynikowa otoczka wypukła prezentowana jest na czerwono.

Wartość zwracana: (list) wizualizacja.

14. Sprawozdanie

14.1. Kryterium klasyfikacyjne – wyznacznik

Punkty w zbiorach klasyfikowano na te znajdujące się po lewej, po prawej oraz współliniowe względem dwóch punktów, przez które przechodzi prosta/prostej (znajdujące się na jednej prostej) na podstawie wartości obliczonego wyznacznika.

Klasyfikacji dokonano na podstawie wartości wyznacznika dla punktów a, b i badanego punktu c. Wykorzystano następującą zależność: Wykorzystano własny zaimplementowany wyznacznik 2x2.

Punkt c jest wobec punktów (prostej łączącej punkty) a i b:

$$\det(a,b,c)\begin{cases}(-\infty,0) \Rightarrow po\ prawej,\\ 0 \Rightarrow współliniowy,\\ (0,+\infty) \Rightarrow po\ lewej.\end{cases}$$

14.2. Tolerancja precyzji obliczeń

Wykorzystanie tolerancji związane jest z reprezentacją liczb rzeczywistych w komputerze i wynikającym z tego obarczeniem wyników obliczeń niedokładnością przy wykorzystaniu liczb zmiennoprzecinkowych (w języku *Python* typ zmiennej $float$ – 64-bitowa liczba zmiennoprzecinkowa).

Dokonano obliczeń dla tolerancji:

$$\varepsilon = 10^{-12}.$$

14.3. Opis problemów

Do testów przygotowano różne zbiory punktów. Zbiór $\mathcal{A}$ zawierał punkty losowo wygenerowane na płaszczyźnie, był to najbardziej podstawowy przypadek. Drugi zbiór to zbiór $\mathcal{B}$, w którym punkty rozłożone były na okręgu, w tym przypadku do otoczki należał każdy punkt ze zbioru. Zbiór ten przydatny był do zbadania algorytmów, których złożoność zależała od liczby punktów należących do otoczki, np. algorytm Jarvisa. Trzeci zbiór – zbiór $\mathcal{C}$ składał się z punktów rozłożonych na bokach zadanego prostokąta, dodatkowo w zbirze znalazły się cztery punkty, po jednym na każdym wierzchołku – to one powinny znaleźć się jako jedyne na otoczce wypukłej, znalezienie się na otoczce wypukłej innych punktów wynikało z niedokładności obliczeń na liczbach zmiennoprzecinkowych w komputerze. Ostatni zbiór – $\mathcal{D}$ to punkty znajdujące się na przekątnych i dwóch bokach prostokąta oraz cztery punkty, po jednym na każdym wierzchołku. Tutaj także w otoczce powinny znaleźć się tylko te cztery punkty na wierzchołkach. W zbiorze $\mathcal{D}$ szybszy okazał się algorytm Jarvis pomimo swojej teoretycznej większej złożoności obliczeniowej. Algorytm Jarvisa był tym przypadku szybszy ze względu na specyfikacje zbioru $\mathcal{D}$. W tym zbiorze algorytm Grahama kwalifikował tymczasowo punkty na przekątnych jako potencjalne punkty na otoczkach i dalej szukał kolejnych punktów potencjalnie należących do toczki, natomiast algorytm Jarvisa sprawdzał kolejne punkty i odrzucał punkty na przekątnych, które na otoczce znaleźć się nie mogły. Tymczasowe kwalifikowanie punktów na przekątnych jako potencjalnie znajdujących się na otoczce w algorytmie Grahama sprawiło, że dużo szybszy okazał się algorytm Jarvisa. Ukazuje to, zże algorytmy powinno się dobierać indywidualnie do danej sytuacji, nie kierując się jedynie teoretyczną złożonością „na papierze”.

14.4. Wykonane testy

Testy wykonano najpierw dla zbiorów testowych opisanych wcześniej a następnie dla zbiorów powiększonych, największe zbiory powiększone wykorzystane do zbadania działania algorytmów:

- Zbiór $\mathcal{A}$ powiększony – 100 000 punktów o współrzędnych w przedziale <-100, 100>.
- Zbiór $\mathcal{B}$ powiększony – 100 000 punktów na okręgu o promieniu 1 000 i środku w punkcie (0, 0).
- Zbiór $\mathcal{C}$ powiększony – 100 000 punktów na bokach prostokąta o wierzchołkach w punktach (0, 0), (1000, 0), (1000, 1000), (0, 1000).
- Zbiór $\mathcal{D}$ powiększony – 100 000 punktów na dwóch bokach i 1 000 punktów na przekątnych analogicznego prostokąta jak w poprzednim zbiorze.

14.5. Analiza wyników testów

W testach ogólnie najszybszym algorytmem okazał się algorytm górnej i dolnej otoczki – najszybszy we wszystkich testach, a najwolniejszymi okazały się ogólnie algorytmy Jarvisa i przyrostowy, jednak w zbiorze $\mathcal{D}$ i $\mathcal{C}$ wolny okazał się także algorytm Grahama, a w zbiorze $\mathcal{B}$ algorytm Chana.

14.6. Uzyskane wyniki

Uzyskane wyniki testów przedstawiono w tabeli.

Lp.	Obiekt testowy		Czas potrzebny na wykonanie algorytmu [s]						
	Zbiór	Parametry	Dziel i rządź	Przyrostowy	Quickhull	Górnej i dolnej otoczki	Chana	Jarvis	Grahama
1.	$\mathcal{A}$	100	0,002	0,005	0,001	0,001	0,003	0,001	0,001
2.	$\mathcal{A}$	1 000	0,017	0,067	0,005	0,004	0,028	0,02	0,009
3.	$\mathcal{A}$	10 000	0,183	0,687	0,061	0,045	0,395	0,322	0,132
4.	$\mathcal{A}$	100 000	1,398	11,833	0,953	0,533	3,89	4,408	1,68
5.	$\mathcal{B}$	100	0,003	0,01	0,002	0,001	0,004	0,017	0,001
6.	$\mathcal{B}$	1 000	0,017	0,129	0,028	0,003	0,067	1,385	0,008
7.	$\mathcal{B}$	10 000	0,219	3,531	0,31	0,035	0,779	145,843	0,113
8.	$\mathcal{B}$	100 000	1,968	72,584	4,781	0,571	14,681	300	1,555
9.	$\mathcal{C}$	100	0,003	0,007	0,001	0,001	0,002	0,001	0,001
10.	$\mathcal{C}$	1 000	0,018	0,047	0,008	0,004	0,015	0,008	0,013
11.	$\mathcal{C}$	10 000	0,554	1,462	0,159	0,097	0,316	0,185	0,667
12.	$\mathcal{C}$	100 000	1,286	16,625	0,866	0,548	1,436	0,771	2,239
13.	$\mathcal{D}$	(25, 20)	0,002	0,005	0,001	0,001	0,002	0,002	0,002
14.	$\mathcal{D}$	(10^3, 10^2)	0,06	0,076	0,014	0,009	0,032	0,019	0,048
15.	$\mathcal{D}$	(10^4, 10^3)	1,298	16,931	0,784	0,56	1,495	0,798	2,333
16.	$\mathcal{D}$	(10^5, 10^3)	2,93	97,176	1,326	1,016	3,061	1,657	8,186

Tabela 14.5. Wyniki testów

15. Teoretyczne złożoności algorytmów

Teoretyczne złożoności algorytmów to:

- Algorytm dziel i rządź (zwyciężaj): $O(n\ log\ n)$.
- Algorytm przyrostowy: $O(n\ log\ n)$.
- Algorytm Quickhull: $O(n\ log\ n)$, potencjalnie $O(n^2)$.
- Algorytm górnej i dolnej otoczki: $O(n\ log\ n)$.
- Algorytm Chana: $O(n\ log\ k)$.
- Algorytm Jarvisa: $O(nk)$, potencjalnie $O(n^2)$.
- Algorytm Grahama: $O(n\ log\ n)$.

Gdzie n to liczba wszystkich punktów w zbiorze, a k to moc zbioru punktów należących do otoczki.

16. Twórcy algorytmów i data znalezienia algorytmów

Algorytmy:

- dziel i rządź (zwyciężaj): Preparata, Hong, 1977.
- przyrostowy: Kallay; 1984.
- Quickhull: Eddy, 1977; Bykat, 1978.
- górnej i dolnej otoczki: Andrew, 1979.
- Chana: Chan, 1996.
- Jarvisa: Jarvis, 1973.
- Grahama: Graham, 1972.

17. Wnioski i podsumowanie

W projekcie udało się zaimplementować poprawnie działające popularne algorytmy wyznaczania otoczki wypukłej. Wniosek taki można wyciągnąć po przetestowaniu implementacji dla wielu zbiorów testowych o różnych parametrach i uzyskaniu poprawnych wyników.

Wizualizacja działania algorytmów pozwala dobrze zrozumieć i zapamiętać ideę za nimi stojącą oraz jest atrakcyjnym sposobem analizy algorytmów, bo pozwala na śledzenie każdego kroku wykonania algorytmu i pozwala na wizualną weryfikację poprawności działania algorytmu.

W projekcie zwrócono uwagę na wiele szczegółów implementacyjnych by funkcje poprawnie działały dla wszelkich typów danych. Przyjrzano się również zagadnieniu dokładności obliczeń na liczbach zmiennoprzecinkowych.

Obszerna dokumentacja pozwala na łatwe i dokładne zapoznanie się z realizacją projektu i poznanie wszelkich szczegółów działania.

Wykonane testy działania algorytmów pozwoliły zobrazować konieczność doboru odpowiednich algorytmów do danych. Różnice w czasie wykonania funkcji, szczególnie dla dużych danych, ujawniły wpływ implementacji na złożoność i wydajność algorytmów.

Niezbędnym do przeprowadzenia wartościowych testów było wcześniejsze przygotowanie odpowiednich danych testowych, które

Wnioski te są szczególnie cenne dla projektów informatycznych, w których precyzja obliczeń i identyfikacja pewnych danych z wykorzystaniem algorytmów geometrycznych są sprawą kluczową. Poruszone na przedmiocie tematy są ważnymi zagadnieniami związanymi z przetwarzaniem danych i geometrią obliczeniową, a wykonany projekt pozwolił dobrze zapoznać się z algorytmami geometrycznymi.

18. Bibliografia

1. Wykłady oraz laboratoria z przedmiotu „Algorytmy geometryczne” na 3. semestrze studiów na kierunku Informatyka w AGH w Krakowie, dr inż. Barbara Głut
2. „Computational Geometry: Algorithms and Applications”, Mark de Berg
3. “Wprowadzenie do algorytmów”, Thomas H. Cormen
4. Wikipedia:
    a. https://en.wikipedia.org/wiki/Convex_hull
    b. https://en.wikipedia.org/wiki/Convex_hull_algorithms
    c. https://en.wikipedia.org/wiki/Gift_wrapping_algorithm
    d. https://en.wikipedia.org/wiki/Graham_scan
    e. https://en.wikipedia.org/wiki/Quickhull
    f. https://en.wikipedia.org/wiki/Chan%27s_algorithm

* * *