Szymon Nowak-Trzos 411246
Informatyka rok II, gr.2 środa nieparzysta 13:00-14:30

Algorytmy Geometryczne – Ćwiczenie 2 Sprawozdanie

1. Wstęp

1.1. Cel Ćwiczenia

Celem ćwiczenia była implementacja dwóch algorytmów generujących otoczkę wypukłą oraz ich analiza na podstawie zbiorów danych podanych w treści zadania

1.2. Użyte Algorytmy

Do generacji otoczek wypukłych użyto dwóch algorytmów:

- Algorytm Grahama,
- Algorytm Jarvisa.

1.3. Użyte Narzędzia

Ćwiczenie zostało wykonane dzięki zmodyfikowanemu narzędziu graficznemu załączonego na stronie UPEL. Do niezbędnych obliczeń został wykorzystany język programowania Python oraz biblioteka numpy. Ponadto wykresy zostały wygenerowano przy pomocy biblioteki matplotlib, a czas został wyznaczony poprzez bibliotekę time. Do wygenerowania gifów użyto bibiloteki Pillow. Obliczenia, których wyniki zostały uwzględnione w tabelach, oraz wszystkie wykresy zostały wygenerowane na procesorze Intel Core i5-1135G7 2.40GHz.

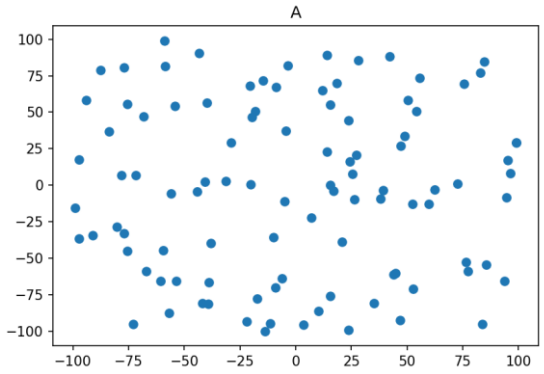
2. Szczegóły wykonywania ćwiczenia

2.1. Zbiory Danych

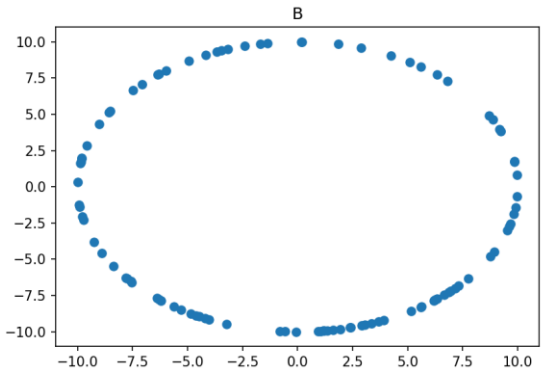
Wygenerowane zostały cztery zbiory danych:

- **A.** 100 losowych punktów o współrzędnych z przedziału [-100, 100]
- **B.** 100 losowych punktów leżących na okręgu o środku (0,0) i promieniu R=10,
- **C.** 100 losowych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10, -10), (10, -10), (10, 10),
- **D.** Zbiór zawierający: 4 wierzchołki kwadratu o lewym dolnym rogu w (-10, 10) i o boku a=10,
  Po 25 punktów na lewym i dolnym boku,
  Po 20 punktów leżących na przekątnych kwadratu

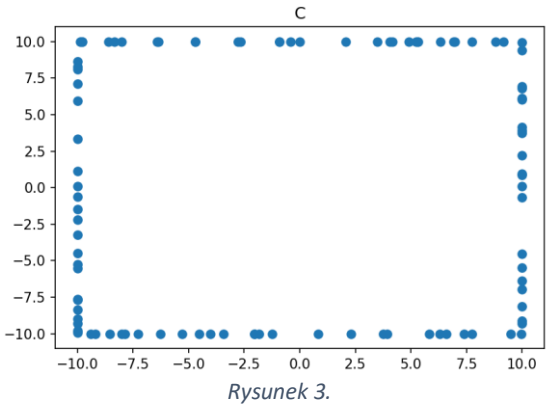
Do wygenerowania zbiorów wykorzystana została funkcja random.uniform z biblioteki random języka Python. Zbiór B uzyskany został przez parametryzację okręgu o zadanym środku. W kodzie znajduje się funkcja ‘generate_datasets’, dzięki której można wygenerować zbiory danych o innych parametrach. Wszystkie zbiory zostały zapisywane oraz odczytywane z plików .json.



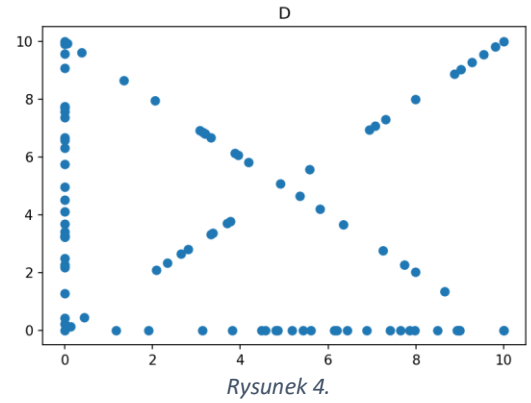
Rysunek 1.



Rysunek 2.

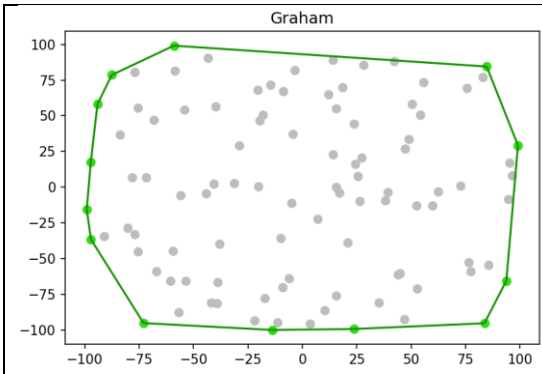


Rysunek 3.

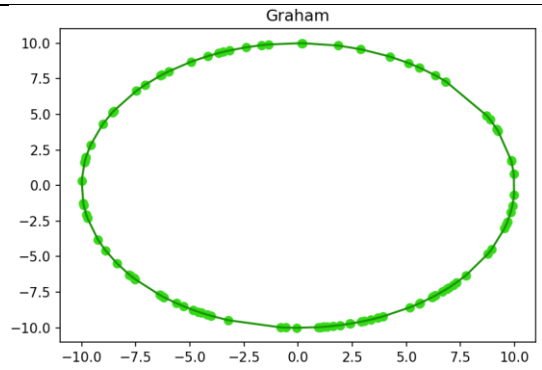


Rysunek 4.

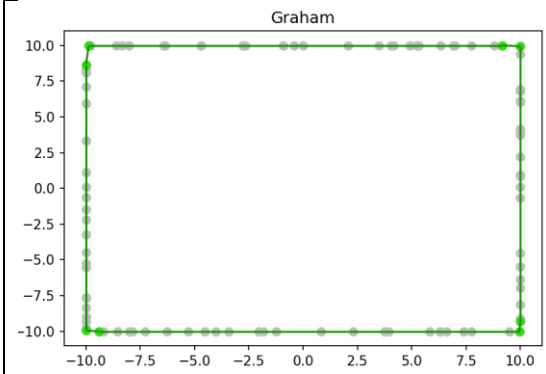
Algorytm Grahama we wszystkich zbiorach dobrze wyznaczył otoczkę, zatem dalej zostały umieszczone wykresy wygenerowane za jego pomocą.



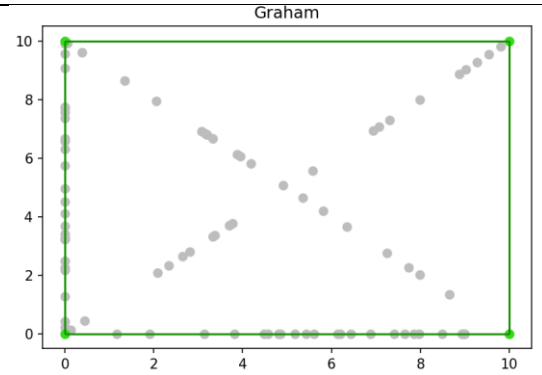
Rysunek 5.



Rysunek 6.



Rysunek 7.



Rysunek 8.

2.2. Algorytmy

2.2.1. Algorytm Grahama

Przekazana tablica punktów sortowana jest ze względu na kąt używając bibliotecznego sortowania:

array.sort(key=functools.cmp_to_key(lambda x, y: det(m_point, x, y)))

gdzie funkcja **det(a, b, c)** zwraca -1 jeśli punkt **c** leży po lewej stronie prostej **ab**, 1 jeśli leży po prawej lub różnicę odległości punktów **(a do c) – (a do b)** gdy punkt jest współliniowy. Jest to uzyskane metodą wyznacznika 3x3 z tolerancją ε = 10^(-12).

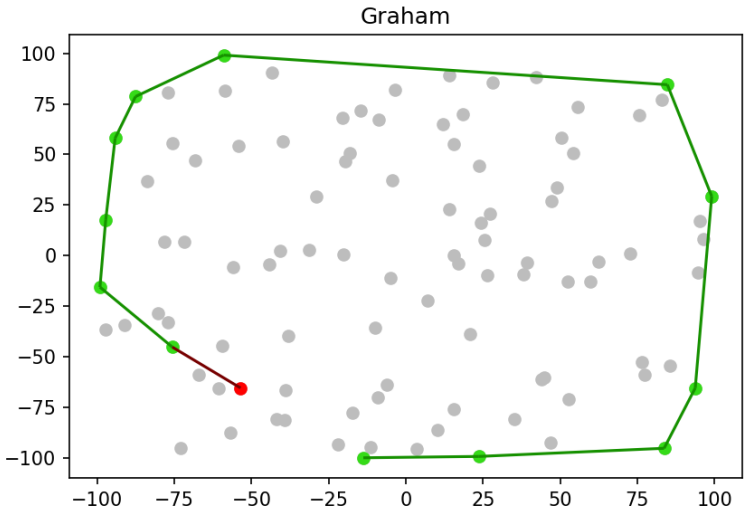
Po posortowaniu algorytm dodaje trzy pierwsze punkty do otoczki, a następnie przechodzi przez każdy punkt w tablicy sprawdzając, czy nowy punkt leży po lewej stronie. W takim wypadku dodaje go do otoczki. W przeciwnym zaś, jeśli punkt jest współliniowy to usuwa ostatni punkt z otoczki i sprawdza dalej, jeśli jest po prawej stronie nie bierze nowego punktu do otoczki.

2.2.2. Algorytm Jarvisa

Algorytm Jarvisa znajduje wpierw minimalny element **m_point** ze zbioru punktów ze względu na y. Drugi punkt otoczki wyznaczany jest poprzez jednokrotne przejście przez tablicę punktów i wyznaczenia punktu o najmniejszym kącie względem prostej przechodzącej przez punkty **(m_point.x-100, m_point.y)** oraz **m_point.** Prosta ta jest równoległa do osi **OX** i przechodzi przez najniższy punkt w zbiorze, więc mamy pewność, że nie przeczina otoczki w żadnym punkcie (albo należy do krawędzi otoczki). Mając dwa pierwsze punkty w otoczce algorytm wykonuje pętle dopóki punkt, który przerabiamy nie jest punktem **m_point.** Co powtórzenie pętli znajdujemy element o minimalnym kącie względem dwóch ostatnich punktów otoczki. Jest to wykonane za pomocą funkcji **det(a, b, c, 1)** opisanej wyżej. Ostatni argument tej funkcji sprawia, że dla współliniowych punktów funkcja zwraca 0, a nie różnicę odległości. Jeśli właśnie punkt rozpatrywany jest współliniowy to usuwamy ostatni punkt otoczki i wykonujemy dalej pętle.

2.3. Animacje

Animacje zostały załączone w postaci łączy do strony giphy.com.
Kolorem szarym zostały zaznaczone wszystkie punkty należące do danego zbioru punktów, na zielono punkty oraz krawędzie należące do otoczki. Czerwonym kolorem oznaczono rozpatrywaną krawędź i punkt.



Rysunek 9. Przykładowa klatka animacji wykonani algorytmu Grahama na zbiorze A

3. Analiza Danych

3.1. Zbiory z treści zadania

Oba algorytmy działały poprawnie na zbiorach z treści zadania.

3.1.1. Zbiór A

Zbiór ten jest złożony z punktów wygenerowanych losowo, więc jest bardzo małe prawdopodobieństwo wystąpienia wyjątków (np. trzech punktów współliniowych). Większość punktów występuje wewnątrz otoczki.

3.1.2. Zbiór B

Otoczka tegoż zbioru zawiera wszystkie jego punkty, więc algorytm Grahama sprawdza się najlepiej. Złożoność algorytmu Jarvisa wynosi O(n^2).

3.1.3. Zbiór C

Do otoczki tego zbioru należą po 2 punkty leżące na bokach prostokąta, reszta punktów jest współliniowa. Tym samym jest to dobry zbiór do sprawdzania zachowania algorytmów w przypadku współliniowości. Algorytm Jarvisa zachowuje się tu najlepiej, gdyż nie musi sprawdzać wszystkich punktów.

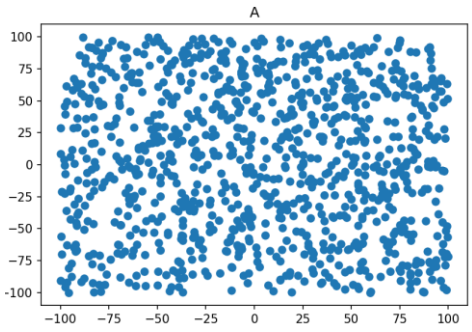
3.1.4. Zbiór D

Otoczka składa się tylko z 4 wierzchołków. Znów algorytm Jarvisa okazuje się być tym sprawniejszym, bo tylko 4 razy musi znaleźć punkt o minimalnym kącie.

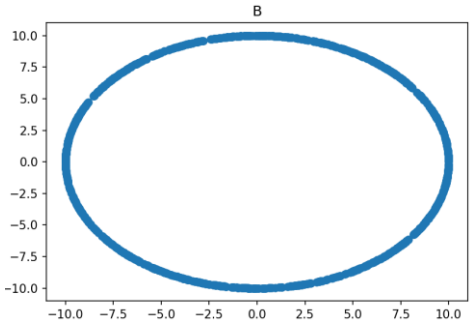
3.2. Zbiory większe

Wygenerowane zostały cztery zbiory danych:

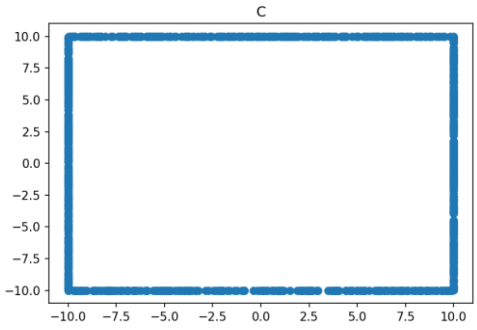
- **A.** 1000 losowych punktów o współrzędnych z przedziału [-100, 100]
- **B.** 1000 losowych punktów leżących na okręgu o środku (0,0) i promieniu R=10,
- **C.** 1000 losowych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10, -10), (10, -10), (10, 10),
- **D.** Zbiór zawierający: 4 wierzchołki kwadratu o lewym dolnym rogu w (-10, 10) i o boku a=10,
  Po 250 punktów na lewym i dolnym boku,
  Po 200 punktów leżących na przekątnych kwadratu



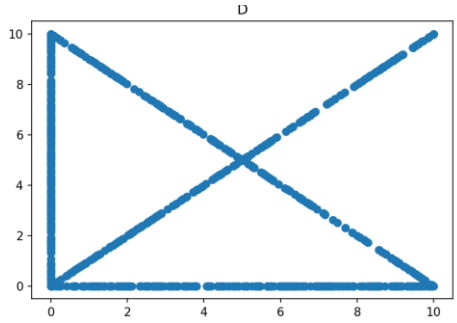
Rysunek 10.



Rysunek 11.



Rysunek 12.



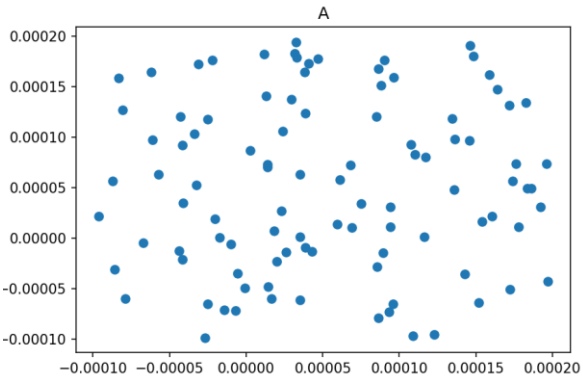
Rysunek 13.

Nie ma zauważalnych różnic w zachowaniu się algorytmów pomiędzy zbiorami większymi a standardowymi.
Zbiór A: Algorytm Jarvisa; Zbiór D: Algorytm Grahama, Algorytm Jarvisa

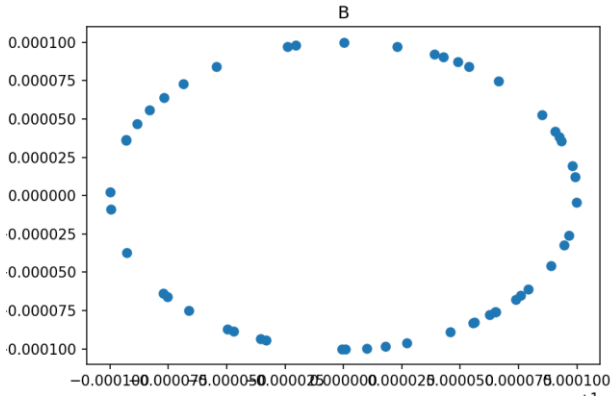
3.3. Zbiory Mniejsze

Wygenerowane zostały cztery zbiory danych:

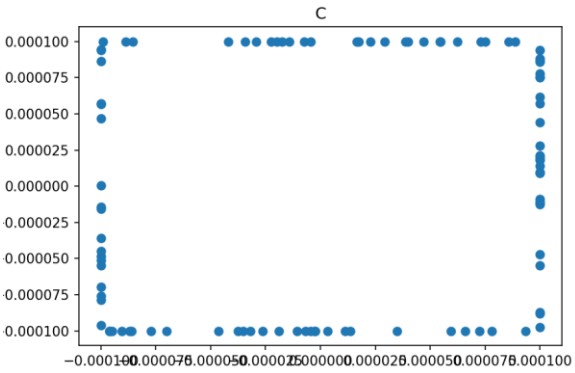
- **A.** 100 losowych punktów o współrzędnych z przedziału [-10^(-4), 10^(-4)]
- **B.** 50 losowych punktów leżących na okręgu o środku (1,0) i promieniu R=10^(-4),
- **C.** 100 losowych punktów leżących na bokach prostokąta o wierzchołkach (-10^(-4), 10^(-4)), (-10^(-4), -10^(-4)), (10^(-4), -10^(-4)), (10^(-4), 10^(-4)),
- **D.** Zbiór zawierający: 4 wierzchołki kwadratu o lewym dolnym rogu w (1, 0), boku a=10^(-4),
  Po 25 punktów na lewym i dolnym boku,
  Po 20 punktów leżących na przekątnych kwadratu



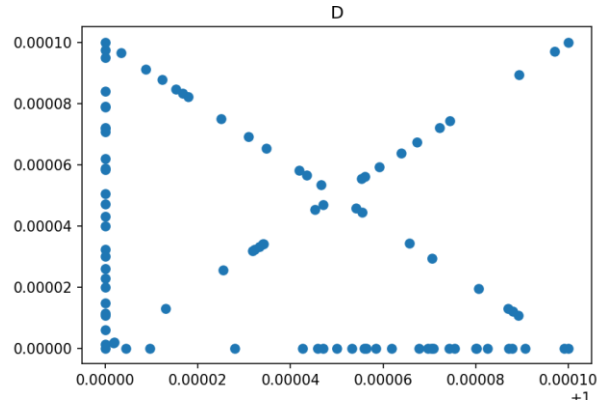
Rysunek 13.



Rysunek 13.



Rysunek 14.



Rysunek 15.

Algorytm Jarvisa niedokładnie wyznacza minimalny element zbioru, przez co generuje błędy w otoczce. Na wygenerowanych danych w zbiorze C algorytm Jarvisa wchodził w nieskończoną pętlę, gdyż po wygenerowaniu już całkowitej otoczki nie powracał do początkowego punktu przez to, że nie wykrywał, że to on jest wierzchołkiem tworzącym następny najmniejszy kąt. Jest to zapewne wynik niedokładności obliczeń przy tej skali. Przy testowaniu innych zbiorów algorytm Grahama zawsze dobrze wskazywał otoczkę, a algorytm Jarvisa potrafił dawać błędny wynik w zbiorach B oraz C.

Zbiór A: Algorytm Grahama, Algorytm Jarvisa, Zbiór B: Algorytm Grahama, Algorytm Jarvisa
Zbiór C: Algorytm Grahama, Algorytm Jarvisa, Zbiór D: Algorytm Grahama, Algorytm Jarvisa

4. Pomiary czasowe funkcji

Czas algorytmów został zmierzony za pomocą funkcji time.time oraz jest on średnią ze 100 powtórzeń działania danego algorytmu na nieposortowanym zbiorze. Do pomiarów czasu przyjęto zbiory pochodne od podstawowego, aczkolwiek ze zmienioną licznością punktów.

	Zbiór 10^1			Zbiór 10^2			Zbiór 10^3			Zbiór 3*10^3	
	Graham	Jarvis		Graham	Jarvis		Graham	Jarvis		Graham	Jarvis
A	2,21E-05 [s]	4,85E-05 [s]	A	3,49E-04 [s]	5,23E-04 [s]	A	5,32E-03 [s]	8,01E-03 [s]	A	9,86E-02 [s]	1,81E-01 [s]
B	1,99E-05 [s]	5,98E-05 [s]	B	2,91E-04 [s]	5,09E-03 [s]	B	5,34E-03 [s]	5,59E-01 [s]	B	9,72E-02 [s]	2,23E+01 [s]
C	1,99E-05 [s]	4,98E-05 [s]	C	4,09E-04 [s]	5,17E-04 [s]	C	6,68E-03 [s]	5,39E-03 [s]	C	9,66E-02 [s]	5,82E-02 [s]
D	2,09E-04 [s]	1,196E-04 [s]	D	3,28E-03 [s]	1,23E-03 [s]	D	4,92E-02 [s]	1,10E-02 [s]	D	2,87E-01 [s]	8,37E-02 [s]

Można zauważyć, że algorytm Grahama jest zawsze szybszy w zbiorach A i B oraz w zbiorach o liczności 10^1 i 10^2. Algorytm Jarvisa jest szybszy w zbiorze D, gdyż ma on do znalezienia tylko 4 punkty. W zbiorze C o liczności 3*10^3 Jarvis wykonuje się aż 22s.

5. Wnioski

Zadanie to prezentuje różnice w sposobach rozwiązywania jednego problemu. Dwa różne podejścia mogą zwracać ten sam prawidłowy wynik, aczkolwiek zupełnie różnie się zachowywać. W treści zadania wybrano takie zbiory punktów by zaznaczyć wady i zalety danych algorytmów. Algorytm Grahama zawsze sortuje listę punktów, potem przechodzi po niej liniowo. Podejście to dobrze nadaje się do zbiorów danych, o których nic nie wiemy. Złożoność obliczeniowa to zawsze O(nlogn). Algorytm Jarvisa zaś ma złożoność O(n*k), więc jeśli jesteśmy wcześniej przybliżyć ilość punktów oczekiwanej otoczki oraz będzie ona wystarczająco mała to powinniśmy skorzystać z tego algorytmu.