Aleksander Jóźwik gr. 2, Pon. godz. 15:00 A

Data wykonania: 21.10.2024 Data oddania: 06.11.2024

Algorytmy Geometryczne - laboratorium 2

Otoczka wypukła

1. Dane techniczne

- System operacyjny: Fedora Linux (x86-64)
- Procesor: Intel Core i5-8350U (1.70 - 3.60 GHz)
- Pamięć RAM: 16GB (2133 MHz)
- Środowisko: Jupyter Notebook
- Język: Python 3.9.20

W realizacji ćwiczenia wykorzystano biblioteki: *numpy*, *pandas*, *random*, *functools* oraz narzędzie wizualizacji stworzone przez koło naukowe *BIT*. Użyta precyzji przechowywania zmiennych i obliczeń w ćwiczeniu to *float64*.

2. Cel ćwiczenia

Celem ćwiczenia jest implementacja dwóch algorytmów wyznaczania otoczki wypukłej tj. algorytmu Grahama oraz algorytmu Jarvisa. Zadanie obejmuje także porównanie ich złożoności czasowej, wykorzystanie ich do znalezienia otoczek wypukłych dla zadanych zbiorów punktów, a także wizualizację poszczególnych kroków algorytmów oraz analizę uzyskanych danych.

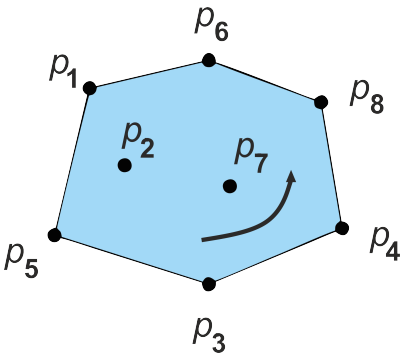
3. Wstęp teoretyczny

3.1. Definicja otoczki wypukłej

Otoczka wypukła *CH(S)* (dowolnego niepustego zbioru punktów) to najmniejszy zbiór wypukły zawierający *S*.

Na płaszczyźnie dwuwymiarowej otoczka wypukła sprowadza się do najmniejszego wielokąta wypukłego zawierającego *S*.

Listę punktów otoczki z *S* podaje się wg konwencji w porządku przeciwnym do ruchu wskazówek zegara (układ prawoskrętny).



Rysunek 1: Przykładowa otoczka wypukła $\mathcal{L} = (p_3, p_4, p_8, p_6, p_1, p_5)$ ze zbioru punktów $\{p_1, p_2, ..., p_8\}$

W przypadku punktów współliniowych do otoczki należeć mogą jedynie punkty najbardziej oddalone od siebie na płaszczyźnie.

3.2. Algorytmy znajdowania otoczki wypukłej

3.2.1. Algorytm Grahama

Algorytm Grahama funkcjonuje poprzez regularne usuwanie wierzchołków wklęsłych. Realizowane jest to poprzez utrzymywanie stosu wierzchołków „kandydujących" na punkty otoczki oraz iterację po liście punktów posortowanych ze względu na kąt.

1. Wybieramy z podanego zbioru S punkt początkowy p_0, posiadający najniższą współrzędną y. W przypadku kilku takich punktów, wybieramy ten o najmniejszej współrzędnej x.
2. Sortujemy pozostałe punkty według kąta utworzonego między wektorem (p_0, p) a dodatnią osią OX. Jeżeli kilka punktów tworzy identyczny kąt, zachowujemy tylko ten najbardziej oddalony od p_0, a resztę usuwamy, otrzymując uporządkowany ciąg $\{p_1, p_2, ...p_m\}$.
3. Tworzymy pusty stos S na który wkładamy punkty p_0, p_1, p_2. Ustanawiamy t jako indeks stosu oraz i równe 3.
4. Dopóki i jest mniejsze od m, sprawdzamy czy punkt p_i leży na lewo od prostej utworzonej przez punkty p_{t-1} i p_t. Jeśli tak, dodajemy p_i na stos i zwiększamy i o 1, w przeciwnym razie zdejmujemy punkt ze stosu.

Na złożoność algorytmu Grahama wpływ mają szukanie minimum, sortowanie, inicjalizacja stosu oraz porównywanie współrzędnych i badanie położenia punktu względem prostej. Ze względu na to, że każdy z punktów analizowany jest tylko raz, to dominujący wpływ na złożoność ma sortowanie punktów. W efekcie złożoność algorytmu wynosi *O(nlogn)*, gdzie n - liczba punktów zbioru początkowego.

3.2.2. Algorytm Jarvisa

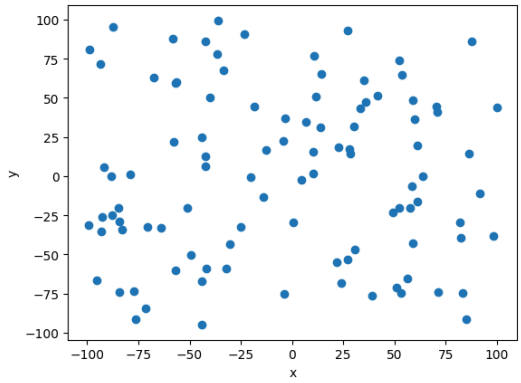
Algorytm Jarvisa działa podobnie do owijania prezentu papierem (stąd jego nazwa). W każdym kroku podejmuje lokalnie optymalne decyzje, wybierając punkt tworzący najmniejszy możliwy kąt z aktualną krawędzią, bez uwzględniania przyszłych wyborów. Ostatecznie prowadzi to do znalezienia globalnie optymalnego rozwiązania, czyli poprawnej otoczki wypukłej.

1. Znajdujemy punkt p_0 z S o najmniejszej współrzędnej y, w przypadku wielu takich punktów wybieramy ten o najmniejszej współrzędnej x. Ten punkt dodajemy do otoczki.
2. Dla każdego kolejnego punktu ze zbioru szukamy punktu, którego kąt liczony przeciwnie do wskazówek zegara względem ostatniej krawędzi otoczki jest najmniejszy.
3. Jeżeli nie ma punktu o mniejszym kącie to dodajemy go do otoczki
4. Powtarzamy procedurę do momentu powrotu do punktu p_0.

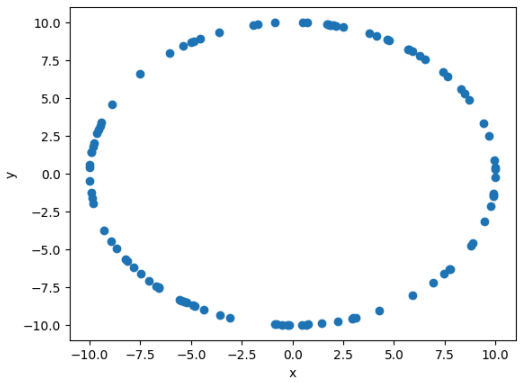
Złożoność algorytmu Jarvisa wynosi *O(nk)*, gdzie k to liczba punktów otoczki, a n to liczba punktów całego zbioru. Dla specyficznych zbiorów punktów, takich jak punkty ułożone na prostokącie, liczba k pozostaje rzędu *O(1)*, co sprawia, że algorytm wykazuje liniową złożoność czasową względem liczby punktów wejściowych.

4. Realizacja ćwiczenia

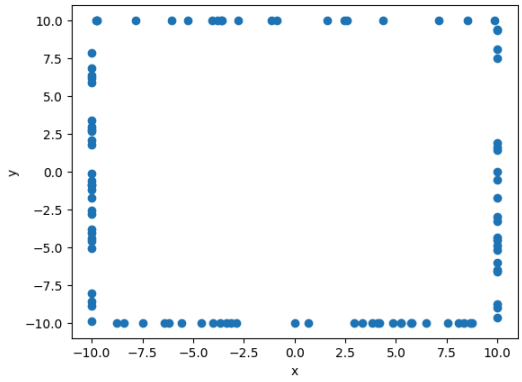
Na początku zostały wygenerowane (przy pomocy *numpy*: `numpy.random.uniform()` oraz *random*: `random.choice()`) 4 zbiory punktów (typu double) na dwuwymiarowej przestrzeni euklidesowej.



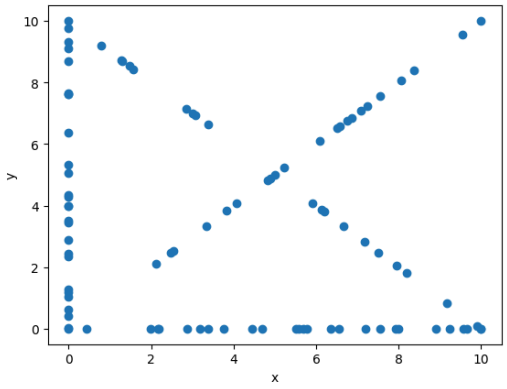
Rysunek 2: Zbiór A: 100 losowo wygenerowanych punktów o współrzędnych z przedziału $[-100, 100]$



Rysunek 3: Zbiór B: 100 losowo wygenerowanych punktów leżących na okręgu o środku $(0, 0)$ i promieniu $R = 10$



Rysunek 4: Zbiór C: 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach $(-10, 10), (-10, -10), (10, -10), (10, 10)$



Rysunek 5: Zbiór D: wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu

Następnie zaimplementowano algorytmy Grahama oraz Jarvisa do znajdowania otoczki wypukłej, unikając funkcji wyznaczającej kąt. Zamiast tego użyto własnej implementacji wyznacznika macierzy 3×3 w celu określenia położenia punktów względem siebie.

W algorytmie Grahama wyznacznik posłużył jako kryterium porównania we wbudowanej funkcji sortującej `sorted()`, która wykorzystuje `cmp_to_key()` z modułu *functools*. Dla współliniowych punktów porządek określano na podstawie odległości od punktu początkowego p_0 (nie usuwano ich na tym etapie). Sortowanie po odległości gwarantuje, że punkty te są ułożone od najbliższego do najdalszego. Podczas iteracji są one przetwarzane w takiej kolejności, co zapewnia poprawne usuwanie punktów wewnętrznych oraz zachowywanie punktów ekstremalnych na stosie reprezentującym otoczkę (w jej skład wchodzi ostatni, czyli najbardziej odległy). Doświadczalnie próbowano pominąć sortowanie punktów współliniowych, co przy obecnej konstrukcji pętli powodowało problemy z niewłaściwym określaniem punktów ekstremalnych.

W algorytmie Jarvisa wyznacznik umożliwił wyszukiwanie kolejnych punktów otoczki. Zamiast wybierać punkt o najmniejszym kącie względem ostatniej krawędzi otoczki, wybierano taki, dla którego nie istnieje żaden punkt znajdujący się po jego prawej stronie względem krawędzi otoczki, którą tworzy.
W przypadku punktów współliniowych porównywana jest ich odległość od poprzedniego punktu otoczki i ostatecznie wybierany jest ten najbardziej odległy - eliminacja punktów wewnętrznych.

Niepewność dla zera została ustalona na $\varepsilon = \mathbf{10^{-24}}$ (większe wartości powodowały błędną klasyfikację punktów na okręgu jako współliniowych).

Dla każdego ze zbiorów uruchomiono oba algorytmy i zwizualizowano ich kroki oraz efekty działania. Następnie zmodyfikowano zbiory w celu przeprowadzenia testów czasowych. Zebrane dane przeanalizowano oraz przedstawiono na wykresach.

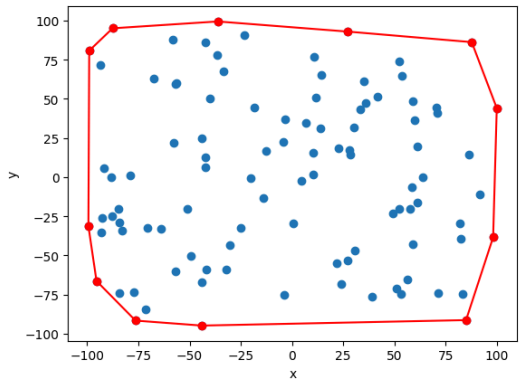
5. Analiza wyników

5.1. Zbiory oryginalne

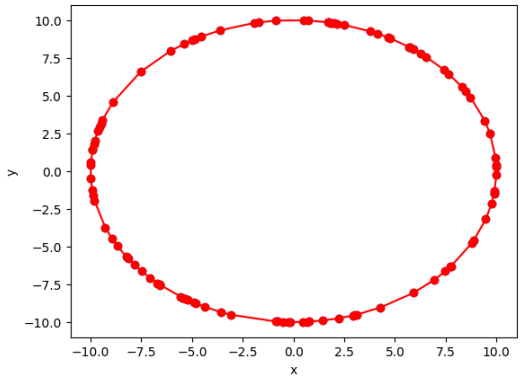
Wizualizacje przedstawione poniżej zostały pokolorowane wg przyjętej konwencji:

- na **niebiesko** zaznaczono punkty ze zbioru,
- na **czerwono** zaznaczono punkty oraz krawędzie należące do otoczki.

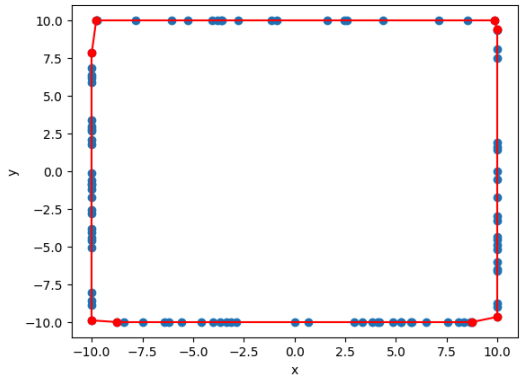
Wynikowe ciągi wierzchołków otoczek oraz wizualizacje funkcjonowania algorytmów krok po kroku, zostały wypisane i zamieszczone w pliku *Jupyter* wraz z kodem źródłowym.



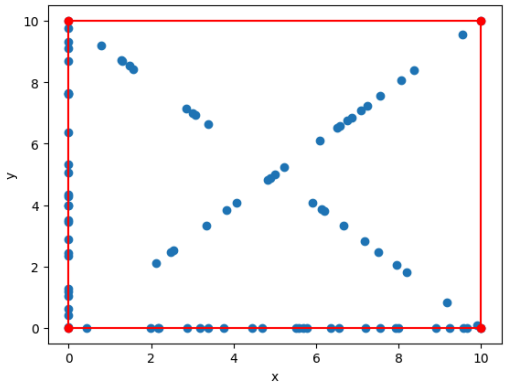
Rysunek 6: Wizualizacja otoczki wypukłej dla zbioru A



Rysunek 7: Wizualizacja otoczki wypukłej dla zbioru B



Rysunek 8: Wizualizacja otoczki wypukłej dla zbioru C



Rysunek 9: Wizualizacja otoczki wypukłej dla zbioru D

Zbiór punktów	**Liczba punktów otoczki**	
	Algorytm Grahama	Algorytm Jarvisa
A	12	12
B	100	100
C	8	8
D	4	4

Tabela 1: Wyniki działania algorytmów dla oryginalnych zbiorów

Jak można zauważyć w Tabeli 1, oraz w pliku *Jupyter*, liczba wraz z ciągami punktów wyznaczonych otoczek jest taka sama dla każdego z algorytmów. Oznacza to, że oba z nich działają poprawnie dla wszystkich zbiorów danych.

Dla zbioru A widoczne jest na Rysunku 6, że otoczka jest poprawna i zawarte są w niej pozostałe punkty zbioru. W przypadku otoczki B (Rysunek 7) wszystkie punkty zostały skategoryzowane jako należące do otoczki. Każdy punkt okręgu jest najbardziej oddalonym punktem w pewnym kierunku (wzdłuż prostej przechodzącej przez środek okręgu), więc musi należeć do otoczki wypukłej. Dla zbiorów C (Rysunek 8) i D (Rysunek 9) istniało ryzyko wystąpienia problemów dla punktów współliniowych, jednak użyte implementacje poradziły sobie z nimi poprzez odpowiednią obsługę tychże przypadków. Jak można zauważyć w Tabeli 1, liczba punktów otoczki dla zbioru C wynosi 8. Jest to maksymalna możliwa wartość dla losowych punktów leżących na bokach prostokąta. Na Rysunku 8 widać, że dla żadnego z boków nie został wygenerowany punkt, który byłby wierzchołkiem prostokąta (jest na to znikoma szansa). W efekcie sprawia to, że dla każdego boku muszą istnieć dwa punkty krańcowe włączone do otoczki. Odmienna sytuacja wystąpiła dla zbioru D, gdzie jak widać na Rysunku 9, obecne są 4 wierzchołki kwadratu i to właśnie one tworzą otoczkę wypukłą dla tej figury.

5.2. Zbiory zmodyfikowane

5.2.1. Modyfikacja zbiorów testowych

W celu przeprowadzenia testów wydajnościowych algorytmów zmodyfikowano poprzednie zbiory:

1. **Zbiór A**: zawiera losowo wygenerowane punkty o współrzędnych z przedziału $[-1000, 1000]$,
2. **Zbiór B**: zawiera losowo wygenerowane punkty leżące na okręgu o środku $(200, 200)$ i promieniu $R = 2500$,
3. **Zbiór C**: zawiera losowo wygenerowane punkty leżące na bokach prostokąta o wierzchołkach $(-300, 100), (100, 100), (100, 500), (-300, 500)$,
4. **Zbiór D**: zawiera wierzchołki kwadratu $(0, 0), (300, 0), (300, 300), (0, 300)$ oraz punkty wygenerowane losowo na przekątnych kwadratu i dwóch bokach kwadratu (leżących na osiach), których stosunek to $\frac{2}{3}$.

Zbiory te wygenerowano dla zadanych kolejno liczności:

- [1000, 2500, 5000, 10000, 25000, 50000, 60000, 70000, 85000, 100000] $\rightarrow$ dla zbiorów A, C, D,
- [100, 500, 1000, 2000, 3000, 5000, 7000, 8000, 10000, 12000] $\rightarrow$ dla zbioru B.

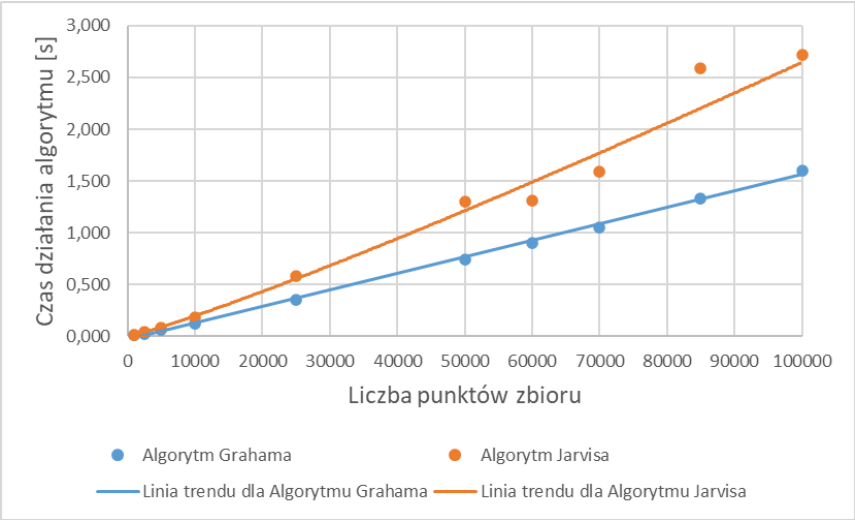
Dla zbiorów zmodyfikowanych istnieje możliwość wypisania listy punktów tworzących otoczkę wypukłą, jednak celowo pominięto ten aspekt, aby nie zaciemniać wyników pomiarów czasowych.

Podczas testów weryfikowano zgodność liczby punktów otoczek wynikowych zwracanych przez oba algorytmy, co miało szczególne znaczenie w potwierdzeniu poprawności implementacji, zwłaszcza dla przypadku badającego zmodyfikowany zbiór B

5.2.2. Wyniki testów dla zmodyfikowanego zbioru A

Liczność zbioru		Czas działania [s]	
Wszystkie punkty	**Punkty otoczki**	**Algorytm Grahama**	**Algorytm Jarvisa**
1000	21	0.010	0.016
2500	21	0.026	0.042
5000	21	0.059	0.086
10000	24	0.13	0.19
25000	26	0.35	0.58
50000	33	0.75	1.30
60000	28	0.91	1.31
70000	29	1.06	1.59
85000	38	1.33	2.59
100000	34	1.60	2.72

Tabela 2: Wyniki pomiaru czasu wykonania dla zmodyfikowanego zbioru A przy różnych licznościach



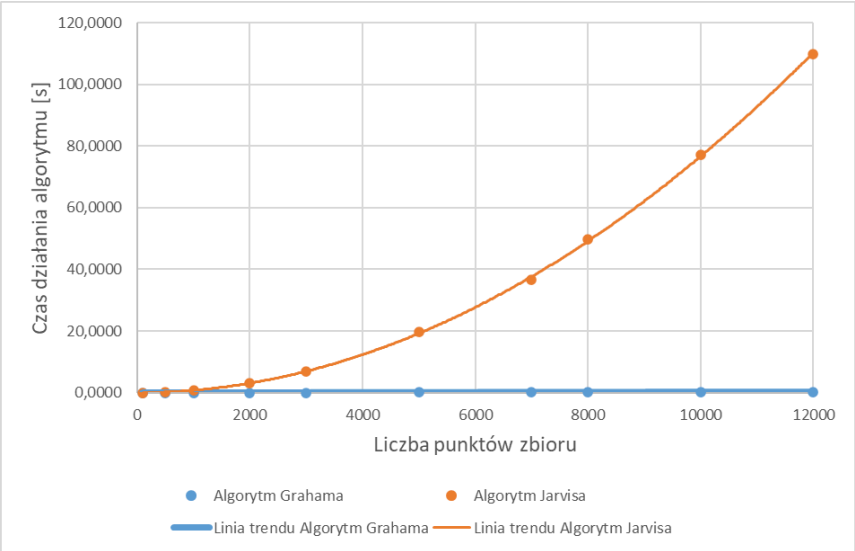
Rysunek 10: Porównanie graficzne czasu wyznaczania otoczki wypukłej przez algorytmy dla zmodyfikowanego zbioru A

Jak można zauważyć w Tabeli 2, algorytm Grahama wykazuje się lepszą efektywnością w stosunku do algorytmu Jarvisa dla zbioru losowych punktów z przestrzeni. Na Rysunku 10 widać, że linia trendu dla algorytmu Grahama jest zbliżona do liniowej, co ma związek z tym, że dla stosunkowo niewielkich wartości n, składnik $\log n$ rośnie bardzo wolno. Algorytm Jarvisa jest w tym przypadku wolniejszy, ale nie jest to aż tak diametralna różnica. Wynika to z liczby punktów, które znalazły się w otoczce.
Dla badanego rozkładu punktów okazało się, że $k \approx 3 \cdot \log n$ (nie jest to oczywiście uniwersalna reguła, a raczej kwestia przypadku), co jest bezpośrednim powodem takiego stanu rzeczy. Największą różnicę w czasie wykonania zauważyć można dla zbioru 85000 pkt, którego otoczka zawiera najwięcej punktów (co daje rezultat w postaci największego k). Algorytm Jarvisa jest wtedy niemal 2 razy wolniejszy od algorytmu Grahama.

5.2.3. Wyniki testów dla zmodyfikowanego zbioru B

Liczność zbioru		Czas działania [s]	
Wszystkie punkty	**Punkty otoczki**	**Algorytm Grahama**	**Algorytm Jarvisa**
100	100	0.0005	0.0076
500	500	0.0038	0.2077
1000	1000	0.0086	0.7788
2000	2000	0.02	3.06
3000	3000	0.03	6.77
5000	5000	0.05	19.66
7000	7000	0.07	36.57
8000	8000	0.09	49.68
10000	10000	0.11	77.22
12000	12000	0.14	109.88

Tabela 3: Wyniki pomiaru czasu wykonania dla zmodyfikowanego zbioru B przy różnych licznościach



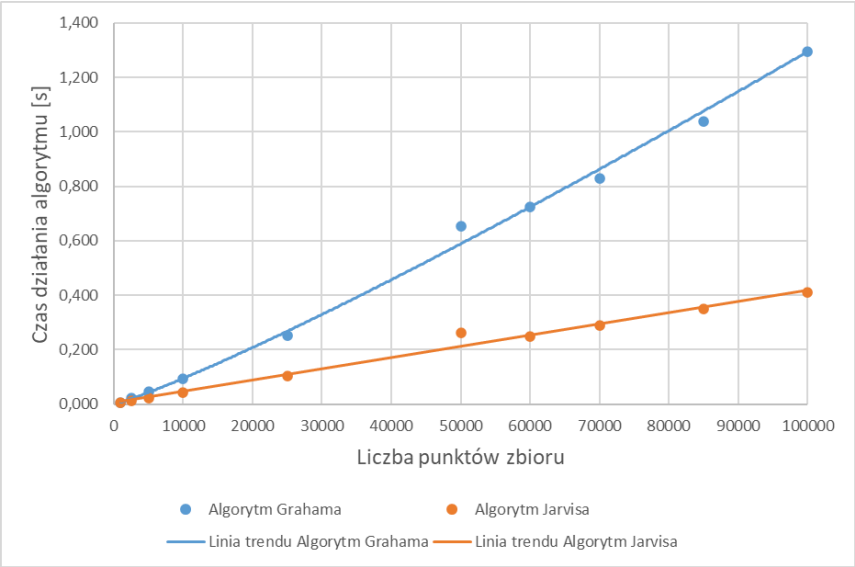
Rysunek 11: Porównanie graficzne czasu wyznaczania otoczki wypukłej przez algorytmy dla zmodyfikowanego zbioru B

Dla każdej kolejnej liczności zmodyfikowanego zbioru B, liczba punktów otoczki jest równa liczbie wszystkich punktów, co było oczekiwane i dowodzi poprawności implementacji obu algorytmów.
Taki stan rzeczy wpływa szczególnie na wydajność algorytmu Jarvisa. Jak można zauważyć na Rysunku 11, linia trendu dla tego algorytmu ma postać kwadratową. Związane jest to właśnie z faktem, że $k = n$, co skutkuje osiągnięciem przez ten algorytm złożoności $O(n^2)$. Różnice między algorytmami są tutaj na tyle znaczące, że linia trendu algorytmu Grahama nie odbiega wyraźnie od osi OX. Jak widać w Tabeli 3, złożoność algorytmu Grahama sprawia, że jest on zdecydowanie lepszym wyborem dla tak spreparowanego zbioru. W skrajnym przypadku (12000 pkt), jest on szybszy od algorytmu Jarvisa o aż 785 razy. Różnica ta zwiększałaby się jeszcze bardziej dla coraz to większej liczności zbioru.

5.2.4. Wyniki testów dla zmodyfikowanego zbioru C

Liczność zbioru		Czas działania [s]	
Wszystkie punkty	**Punkty otoczki**	**Algorytm Grahama**	**Algorytm Jarvisa**
1000	8	0.006	0.004
2500	8	0.02	0.01
5000	8	0.04	0.02
10000	8	0.09	0.04
25000	8	0.25	0.10
50000	8	0.65	0.26
60000	8	0.73	0.25
70000	8	0.83	0.29
85000	8	1.04	0.35
100000	8	1.30	0.41

Tabela 4: Wyniki pomiaru czasu wykonania dla zmodyfikowanego zbioru C przy różnych licznościach



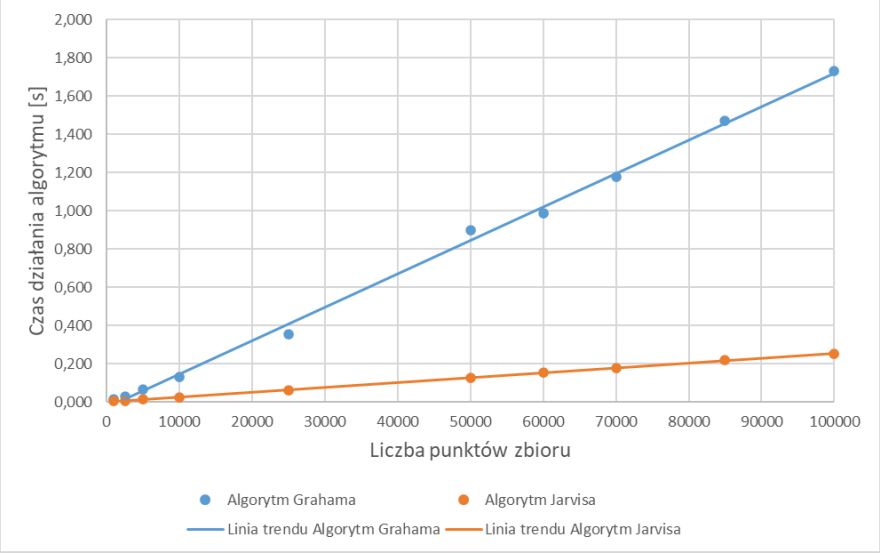
Rysunek 12: Porównanie graficzne czasu wyznaczania otoczki wypukłej przez algorytmy dla zmodyfikowanego zbioru C

W przypadku zmodyfikowanego zbioru C sytuacja się zmieniła, co można zobaczyć w Tabeli 4 oraz na Wykresie 12. Ograniczona od góry liczba punktów otoczki sprawiła, że $k = 8$, przez co w efekcie złożoność algorytmu Jarvisa jest rzędu $O(n)$, podczas gdy dla algorytmu Grahama jest to $O(n \log n)$. Wynika z tego, że dla zbiorów o m.in. dużej zawartości punktów współliniowych (a w efekcie ograniczonym k), algorytm Jarvisa jest lepszym wyborem odnośnie czasu wykonania.

5.2.5. Wyniki testów dla zmodyfikowanego zbioru D

Liczność zbioru		Czas działania [s]	
Wszystkie punkty	**Punkty otoczki**	**Algorytm Grahama**	**Algorytm Jarvisa**
1000	4	0.014	0.004
2500	4	0.028	0.007
5000	4	0.07	0.01
10000	4	0.13	0.02
25000	4	0.35	0.06
50000	4	0.90	0.13
60000	4	0.99	0.15
70000	4	1.17	0.18
85000	4	1.47	0.22
100000	4	1.73	0.25

Tabela 5: Wyniki pomiaru czasu wykonania dla zmodyfikowanego zbioru D przy różnych licznościach



Rysunek 13: Porównanie graficzne czasu wyznaczania otoczki wypukłej przez algorytmy dla zmodyfikowanego zbioru D

Jak można zauważyć w Tabeli 5 oraz na Wykresie 13, algorytm Jarvisa wykazuje jeszcze większą przewagę nad algorytmem Grahama w porównaniu do poprzedniego zbioru. Dzieje się tak, ponieważ obecny zbiór zawiera wyłącznie wierzchołki kwadratu, które są jedynymi punktami tworzącymi otoczkę wypukłą. W efekcie wartość k (liczba punktów w otoczce) jest dwa razy mniejsza niż w przypadku zmodyfikowanego zbioru C, co dodatkowo przyspiesza działanie algorytmu Jarvisa.

6. Wnioski

Na podstawie przeprowadzonego ćwiczenia można wyciągnąć następujące wnioski:

1. Zarówno algorytm Grahama, jak i algorytm Jarvisa poprawnie wyznaczyły otoczki wypukłe dla wszystkich analizowanych zbiorów punktów, co świadczy o ich poprawnej implementacji i działaniu w różnych przypadkach testowych.

2. Dla zbiorów losowych (np. zbiór A), algorytm Grahama jest szybszy od algorytmu Jarvisa, szczególnie przy większych licznościach punktów. Wynika to z jego złożoności czasowej $O(n \log n)$, co czyni go bardziej efektywnym w przypadku dużych zbiorów.

3. Algorytm Jarvisa wykazuje wyraźną przewagę przy zbiorach, gdzie liczba punktów otoczki k jest mała i ograniczona (np. zbiory C i D, zawierające punkty współliniowe). W takich przypadkach jego złożoność jest praktycznie liniowa względem liczby punktów wejściowych, co przekłada się na lepszą wydajność w porównaniu do algorytmu Grahama.

4. Wyniki testów wskazują, że liczba punktów tworzących otoczkę wypukłą k wpływa bezpośrednio na czas wykonania algorytmu Jarvisa. W przypadku, gdy $k = n$ (jak w zbiorze B, gdzie wszystkie punkty należą do otoczki), złożoność algorytmu Jarvisa wynosi $O(n^2)$, co znacznie wydłuża czas jego działania.

5. Wybór odpowiedniego algorytmu zależy od charakterystyki zbioru punktów:
   - Algorytm Grahama jest bardziej uniwersalnym wyborem dla zbiorów o losowym rozkładzie punktów.
   - Algorytm Jarvisa sprawdza się lepiej w przypadkach, gdy większość punktów jest współliniowa lub liczba punktów otoczki wypukłej jest ograniczona.

Podsumowując, algorytmy Grahama i Jarvisa oferują skuteczne podejścia do problemu wyznaczania otoczki wypukłej, jednak ich wydajność różni się w zależności od specyfiki zbioru danych, co warto uwzględniać przy planowaniu rozwiązań w praktycznych zastosowaniach.