Algorytmy Geometryczne

Sprawozdanie – Ćwiczenie 2: Otoczka wypukła

Krzysztof Chmielewski

Data wykonania: 21.10.2024
Data oddania: 04.11.2024

Spis treści

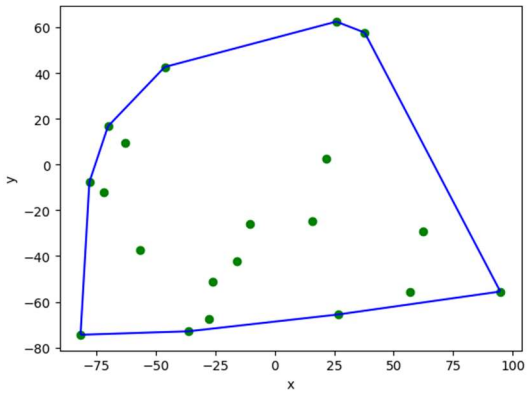
WSTĘP

CEL ĆWICZENIA

Ćwiczenie ma za zadanie wprowadzić pojęcie otoczki wypukłej zbioru punktów na płaszczyźnie dwuwymiarowej oraz algorytmów jej wyznaczania. Do tych algorytmów należą algorytm Grahama i Jarvisa, celem ćwiczenia jest także ich porównanie pod względem wydajności czasowej.

TEORIA

Otoczka wypukła $CH(S)$ dowolnego niepustego zbioru punktów S to najmniejszy zbiór wypukły zawierający S. Do otoczki zaliczamy punkty „ekstremalne” bez punktów współliniowych. Rys. 1 przedstawia przykładową otoczkę.



Rys. 1 Ilustracja otoczki wypukłej CH(S)

ALGORYTM GRAHAMA

1. Wybieramy punkt p_0 o najmniejszej wartości współrzędnej y, a jeśli takich jest wiele to z pośród nich wybieramy ten o najmniejszej współrzędnej x.
2. Pozostałe punkty należy posortować względem kąta jaki tworzy wektor $[p_0, p_i]$ z dodatnim kierunkiem osi OX. Należy także usunąć duplikaty (ze względu na ten kąt) i zostawić jedynie punkt najbardziej oddalony od p_0.
3. Tworzymy stos i dodajemy na niego pierwsze 3 punkty z listy posortowanych.
4. Algorytm wykorzystuje poniżej opisaną funkcję orient patrząc na dwa ostatnie punkty stosu s_{-1}, s_{-2} i rozważany aktualnie punkt p_i (orient (s_{-1}, s_{-2}, p_i)). Dopóki tworzą one układ prawoskrętny to usuwamy punkt ze stosu, jeśli tworzą lewoskrętny to dodajemy rozważany punkt na stos. Na koniec stos będzie tworzył otoczkę.

Funkcja orient określa skręt układu kolejnych punktów a,b,c:

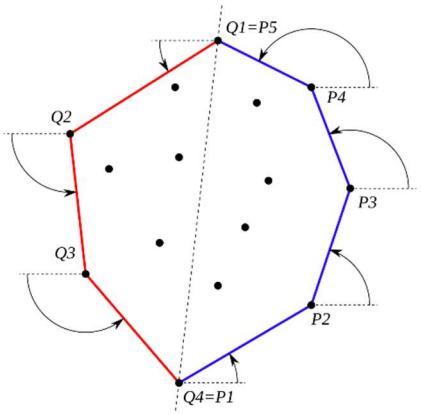
- $\text{orient}(a, b, c) > 0$; punkty tworzą układ lewoskrętny
- $\text{orient}(a, b, c) < 0$; punkty tworzą układ prawoskrętny
- $\text{orient}(a, b, c) = 0$; punkty są współliniowe

Złożoność czasowa algorytmu: O(nlogn)

ALGORYTM JARVISA

Algorytm jest także nazywany owijaniem prezentu. Podobnie jak w algorytmie Grahama szukamy minimalnego punktu. Drugi punkt otoczki znajdujemy poprzez określenie dla niego kąta między wektorem od punktu minimalnego do drugiego oraz osią OX. Następnie szukamy względem tego punktu, kolejnego punktu, który tworzy najmniejszy kąt między poprzednim punktem patrząc przeciwnie do ruchu wskazówek zegara. Jeśli dotrzemy do punktu wyjścia, czyli minimalnego punktu to znaczy, że wyznaczyliśmy otoczkę. Na Rys. 2. możemy zobaczyć wizualizację tego algorytmu.

Złożoność czasowa algorytmu: O(n^2)



Rys. 2 Wizualizacja algorytmu Jarvisa

DANE TECHNICZNE

Ćwiczenie zostało wykonane z użyciem narzędzia graficznego dostarczonego przez Koło Naukowe Bit (**https://github.com/aghbit/Algorytmy-Geometryczne)**, które umożliwia wizualizację wykresów i kształtów geometrycznych wykorzystujące różne biblioteki języka Python (np. numpy, pandas, matplotlib) oraz Anacondę do stworzenia odpowiedniego jądra dla Jupyter Notebook. Kod zawarty w pliku ***chmielewski_kod_2.ipynb*** został napisany w języku Python właśnie przy użyciu Jupyter Notebook.

Do utworzenia zbiorów A,B,C,D napisano odpowiedne funkcje wykorzystujące biblioteki random oraz zawartą w niej metodę uniform(), a także wykorzystano funkcje sin, cos, pi z biblioteki math. Obliczenia zostały wykonane z wykorzystaniem sprzętu o następujących parametrach:

Komputer -> wirtualna maszyna VirtualBox:

- **Procesor:** AMD Ryzen 5 5600X 6 rdzeniowy, podstawowe taktowanie: 3,70 GHz -> w wirtualnej maszynie wykorzystano jedynie 5 z 12 wątków
- **Karta Graficzna:** NVIDIA GeForce RTX 3060 Ti 8GB GDDR6X
- **RAM:** 32GB DDR4 3000MHz CL16 -> w wirtualnej maszynie wykorzystano jedynie 16GB
- **OS:** Linux Ubuntu 24.04

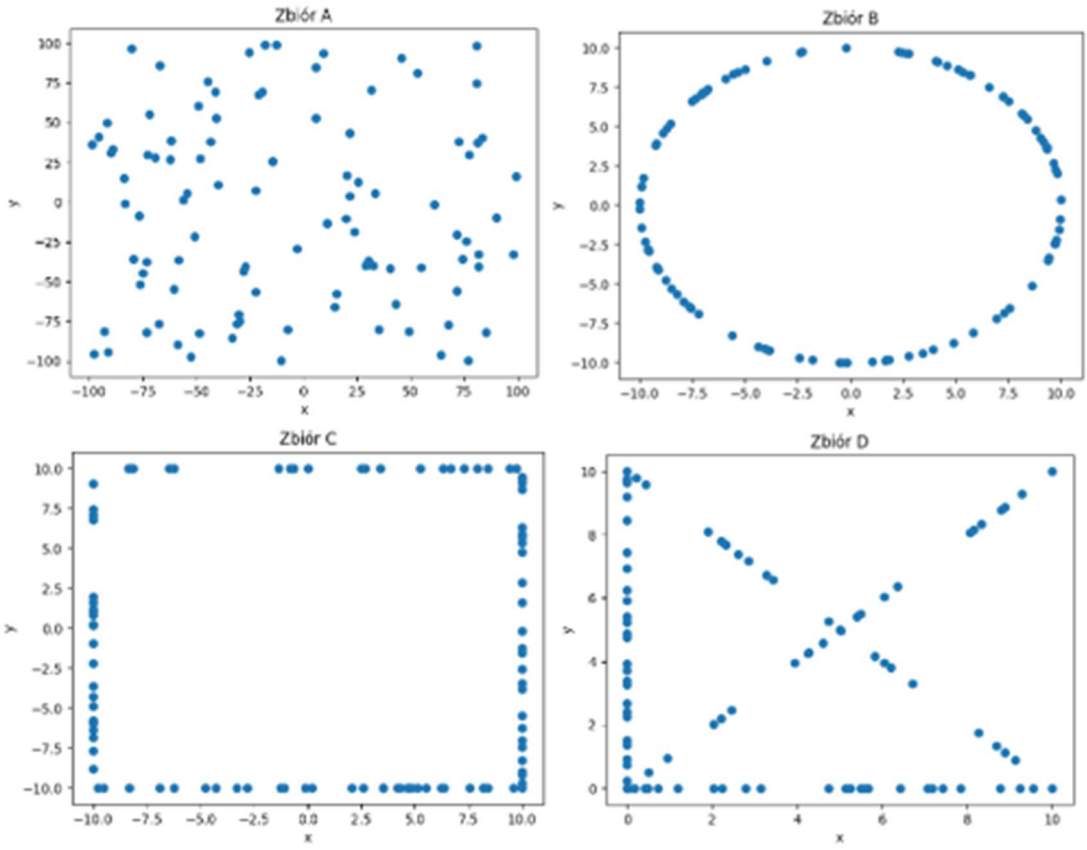
Laptop:

- **Procesor:** Intel Core I5-1235u 10 rdzeniowy, podstawowe taktowanie: 1,30 GHz
- **Karta Graficzna:** zintegrowana z procesorem Intel UHD
- **RAM:** 8GB DDR4 3200MHz
- **OS:** Linux Ubuntu 24.04

REALIZACJA ĆWICZENIA

Obliczenia były wykonywane na czterech różnych zbiorach: A, B, C i D:

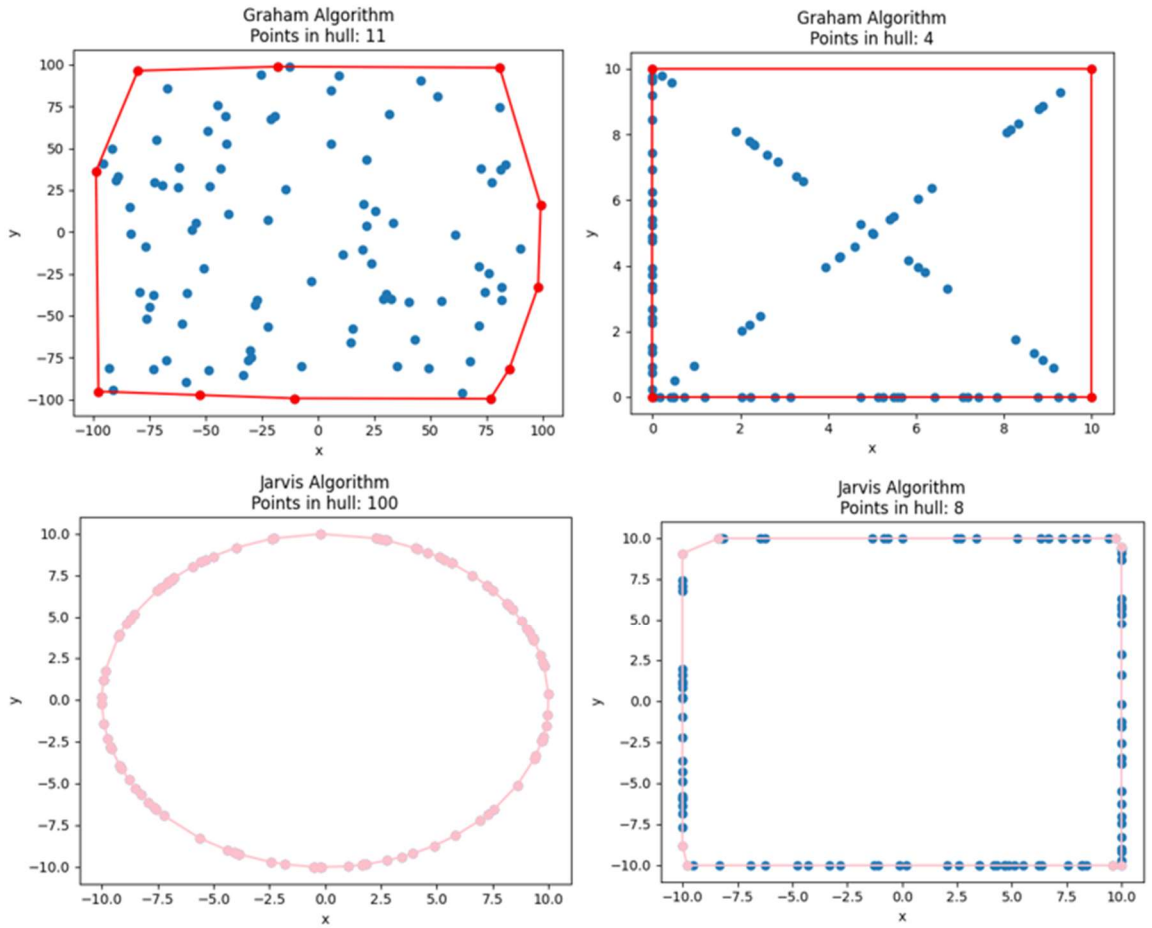
- **A**: zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100]
- **B**: zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10
- **C:** zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10)
- **D:** zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo w sposób następujący: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu.



Rys. 3 Wizualizacje zbiorów A,B,C,D

Na Rys. 3 można zobaczyć ilustracje wygenerowanych zbiorów.

WYNIKI I ANALIZA



Rys. 4 Przykładowe wyniki algorytmów

Ilustracje na Rys. 4. pokazują wyznaczone otoczki dla każdego ze zbiorów przy czym dla dwóch pierwszych wybrano algorytm Grahama, dla dwóch kolejnych algorytm Jarvisa. Wszystkie ilustracje tj. dla każdego zbioru A,B,C i D wyznaczone zarówno za pomocą algorytmu Grahama jak i Jarvisa można zobaczyć w pliku z kodem (chmielewski_kod_2.ipynb).

Tabela 1. Porównanie algorytmów

Liczba punktów w otoczce wyznaczonej przez algorytm	Grahama	Jarvisa
Zbiór A	11	11
Zbiór B	100	100
Zbiór C	8	8
Zbiór D	4	4

Oba algorytmy wyznaczają tą samą otoczkę dla każdego zbioru (co można zobaczyć w pliku z kodem) oraz taką samą liczbę punktów w każdej z nich, co widać na **Tabeli 1.** i zgromadzonych w niej wynikach.

NOWE ZBIORY I PORÓWNANIE CZASÓW

Głównym zagadnieniem jest porównanie czasów wykonania tych algorytmów na różnych zbiorach o różnych wielkościach, w tym celu stworzymy nowe zbiory bazujące na poprzednich, jednak ze zwiększoną liczbą punktów (ograniczoną dla B do czterech pierwszych).

Liczby punktów kolejno generowana dla każdego zbioru: 100, 1000, 2500, 5000, 10^4, 10^5.

- **A**: punkty o współrzędnych z przedziału [-1000, 1000]
- **B**: punkty leżące na okręgu o środku (1000,1000) i promieniu R=500
- **C:** punkty leżące na bokach prostokąta o wierzchołkach -500,0), (400,0), (400,400), (-500,400)
- **D:** punkty zawierające wierzchołki kwadratu (0,0), (100,0), (100,100), (0,100) oraz punkty wygenerowane losowo w sposób następujący: po x punktów na dwóch bokach kwadratu leżących na osiach i po x punktów na przekątnych kwadratu.

TABELA PORÓWNAŃ

Tabela 2. Porównanie większych zbiorów

Set Name	Number of Points	Graham Time (s)	Jarvis Time (s)	Delta (s)	Faster Algorithm	Number of points in Graham's hull	Number of points in Jarvis' hull
Zbiór A	100	0.002250	0.005550	0.003300	Graham	12	12
Zbiór A	1000	0.022110	0.068420	0.046310	Graham	16	16
Zbiór A	2500	0.047030	0.221650	0.174620	Graham	22	22
Zbiór A	5000	0.109580	0.429260	0.319680	Graham	20	20
Zbiór A	10000	0.244140	1.301780	1.057640	Graham	29	29
Zbiór A	100000	2.533890	11.274480	8.740590	Graham	25	25
Zbiór B	100	0.001510	0.043930	0.042420	Graham	100	100
Zbiór B	1000	0.012380	4.290180	4.277800	Graham	1000	1000
Zbiór B	2500	0.033580	26.912660	26.879080	Graham	2500	2500
Zbiór B	5000	0.064440	106.002630	105.938190	Graham	5000	5000
Zbiór C	100	0.001680	0.003590	0.001910	Graham	8	8
Zbiór C	1000	0.016060	0.036420	0.020360	Graham	8	8
Zbiór C	2500	0.048580	0.100110	0.051530	Graham	8	8
Zbiór C	5000	0.096410	0.195320	0.098910	Graham	8	8
Zbiór C	10000	0.182370	0.414930	0.232560	Graham	8	8
Zbiór C	100000	2.191210	3.881670	1.690460	Graham	8	8
Zbiór D	100	0.002000	0.005810	0.003810	Graham	4	4
Zbiór D	1000	0.009210	0.042850	0.033640	Graham	4	4
Zbiór D	2500	0.023430	0.109300	0.085870	Graham	4	4
Zbiór D	5000	0.051620	0.207920	0.156300	Graham	4	4
Zbiór D	10000	0.100970	0.430170	0.329200	Graham	4	4
Zbiór D	100000	1.414050	4.347980	2.933930	Graham	4	4

Tabela 2. jest wynikiem modułu pandas i utworzenia dataframe z danych dla każdego zbioru. Jak widać w kolumnie „Faster Algorithm" oraz „Delta (s)" **Tabeli 2.** algorytm Grahama przewyższa znacznie algorytm Jarvisa pod względem czasu tworzenia otoczki wypukłej. Obserwacja ta jest tym bardziej wyraźna im większy jest zbiór punktów. Dla zbioru B liczba punktów została ograniczona jedynie do pierwszych czterech wielkości, ponieważ większa liczba sprawiła długi czas oczekiwania na wyniki ze względu na czas trwania algorytmu Jarvisa. Jak widać nawet dla 5000 punktów algorytm ten trwa ponad 106 sekund i jest o ponad 105 sekund wolniejszy od Grahama. Oba algorytmy wyznaczają taką samą liczbę punktów w otoczce dla każdego zbioru.

Średni czas wykonania dla algorytmu Grahama: 0,3277s

Średni czas wykonania dla algorytmu Jarvisa: 7,2876s

UWAGI DO IMPLEMENTACJI

ALGORYTM GRAHAMA

Do określania prawo- czy lewoskrętu układu została wykorzystana funkcja orient(a,b,c) tak jak wspomniano w sekcji TEORIA/ALGORYTM GRAHAMA. W implementacji funkcja ta opiera się o określenie wyznacznika macierzy 3x3 z użyciem funkcji bibliotecznej numpy. Aby posortować punkty w zależności od kąta rosnąco uwzględniając długości od punktu startowego p_0, dla każdego z punktów wpierw wyznaczono te wartości tj. cosinus kąta między danym punktem a p_0 oraz odległość danego punktu od p_0. W przedziale od 0 do π cosinus jest malejący, dlatego w funkcji sorted użyto argumentu reversed=True.

ALGORYTM JARVISA

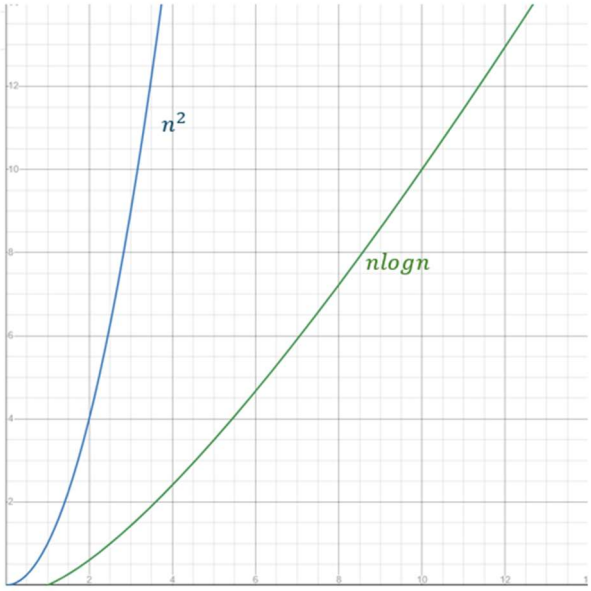
Aby wyznaczyć kąt między punktami a,b i c w algorytmie Jarvisa, użyto iloczynu skalarnego wyznaczonego za pomocą odpowiednich wektorów. Do znalezienia najmniejszego kąta wykorzystano słownik, w którym przechowywane są kąty i odpowiadające im indeksy punktów. W implementacji uwzględniono, aby wyniki dla tych samych kątów zachowywały jedynie punkty, dla których odległość od poprzedniego punktu otoczki jest większa.

UWAGI OGÓLNE

Do sprawozdania dołączono także cztery pliki .gif pokazujące tworzenie otoczki wypukłej na zbiorze A oraz D z wykorzystaniem obu algorytmów. Punkty oraz odcinki zaznaczone na różowo są rozpatrywanymi aktualnie przez algorytm, natomiast czerwone to te należące już do otoczki. Algorytm Grahama sprawdza trzy następne punkty ze stosu oraz ich skręt, algorytm Jarvisa wyznacza kolejno punkty o coraz mniejszych kątach.

WNIOSKI

Ćwiczenie pokazuje rzeczywistość jaką jest złożoność czasowa algorytmów komputerowych. Wiemy, że według teorii złożoność czasowa dla algorytmu Jarvisa to O(n^2), a dla Grahama O(nlogn), gdzie n jest ilością punktów w rozpatrywanym zbiorze.



Rys. 5 Porównanie funkcji n^2 i nlogn

Patrząc na Rys. 5 widzimy, że wraz ze wzrostem n funkcja n^2 przyjmuje znacznie większe wartości niż nlogn. Na podstawie danych zawartych w **Tabeli 2.** widzimy, że otoczkę w każdym ze zbiorów szybciej wyznaczył algorytm Grahama. Patrząc na kolumnę „Delta (s)" w **Tabeli 2.** jesteśmy w stanie także wyznaczyć średnią wartość o jaką algorytm Grahama był szybszy od algorytmu Jarvisa, średnia ta wynosi: 6,9599s.

Widzimy zatem, że algorytm Grahama jest o wiele szybszy od Jarvisa, dla zbioru, który składa się z samych punktów otoczki jest ponad 100 sekund szybszy. Dzieje się tak dlatego, że algorytm Jarvisa za każdym razem sprawdza każdy punkt w zbiorze, dopóki nie znajdzie tego, który tworzy najmniejszy kąt, nie dziwne więc, że dla okręgu jest to tak duży czas, gdyż nie ma w tym zbiorze punktów wewnątrz otoczki, lecz wszystkie punkty zbioru są także jej elementami.

Ponad to widzimy w Tabeli 2., że zbiór ten tj. zbiór B najwięcej posiada 5000 punktów co jest o 20 razy mniej niż najwięcej punktów np. w zbiorze A, jednak różnica czasowa, ze względu na kształt jaki tworzą punkty w tym zbiorze, jest duża.

UWAGI OSOBISTE

Patrząc na zbiory przygotowane na początku do testowania, widać, że zbiór A jest chmurą punktów, zatem wyznaczenie otoczki nie powinno sprawiać dużo kłopotów. Zbiór B stanowi okrąg, w związku z tym każdy punkt w tym zbiorze powinien należeć do otoczki, od razu można dojść do wniosku, że będzie to najbardziej czasochłonna otoczka do wyznaczenia. Najbardziej problematyczne zbiory stanowią C i D, ponieważ zawierają dużą liczbę punktów współliniowych co jest trudnością do wyeliminowania zważając na dokładność liczb zmiennoprzecinkowych i tego jak operuje na nich komputer.