Michał Szczurek
Informatyka II rok, grupa wtorek 12:50 B

Algorytmy geometryczne, ćwiczenie 1- Sprawozdanie

1. Wprowadzenie

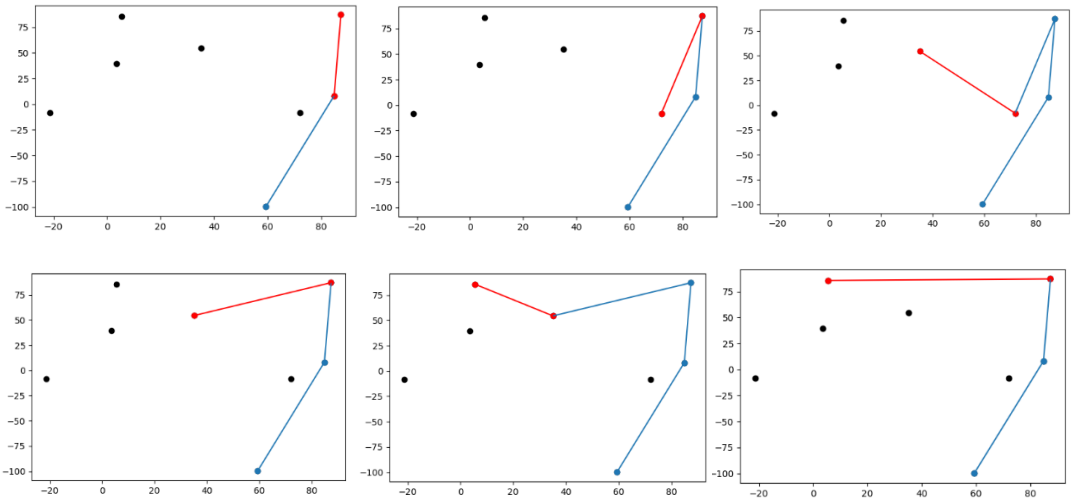
Celem ćwiczenia była implementacja i przetestowanie algorytmów obliczających otoczkę wypukłą- najmniejszy (w sensie inkluzji) zbiór wypukły zawierający podzbiór punktów. Do tego celu wykorzystano dwa algorytmy:

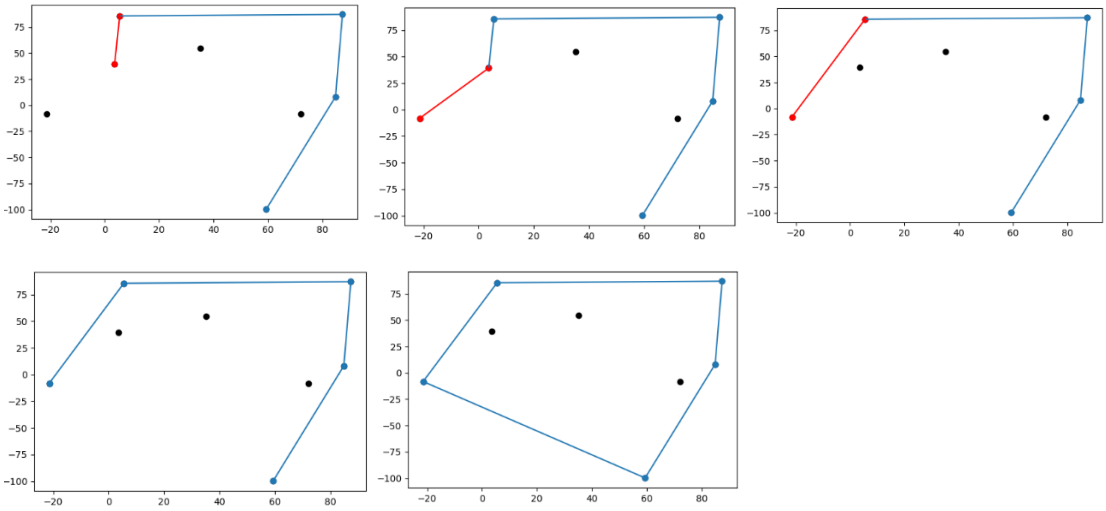
1.1 Algorytm Grahama

Algorytm, którego pomysłodawcą jest Ronald Graham, o złożoności czasowej O(n log n), gdzie n jest liczbą punktów należących do zbioru, dla którego otoczkę wypukłą chcemy wyznaczyć. Poniżej przedstawiono algorytm w postaci listy kroków:

1. Wybranie punktu p_0- punktu o najmniejszej współrzędnej y, spośród wszystkich punktów (W przypadku remisów wybrany zostaje punkt o najmniejszej współrzędnej x).
2. Posortowanie punktów względem kąta α pomiędzy wektorem $(p_0 p_i)$, a dodatnią osią OX układu współrzędnych.
3. Usunięcie punktów tworzących taki sam kąt α z wyjątkiem punktu oddalonego najbardziej od p_0.
4. Odłożenie punktów p_0, p_1, p_2 na stosie.
5. Przeglądanie posortowanej listy punktów zaczynając od p_3:
   - Ściąganie wierzchołków ze stosu tak długo jak punkt p_i nie jest położony na lewo względem prostej $(p_{t-1} p_t)$, gdzie t- indeks stosu.
   - Dodanie p_i na szczyt stosu.
6. Zwrócenie stosu jako wyniku.

Zbiór rysunków 1. Przykładowy przebieg algorytmu Grahama:





Linie niebieskie, to obecny wygląd otoczki wypukłej, czerwone- krawędź rozważana.

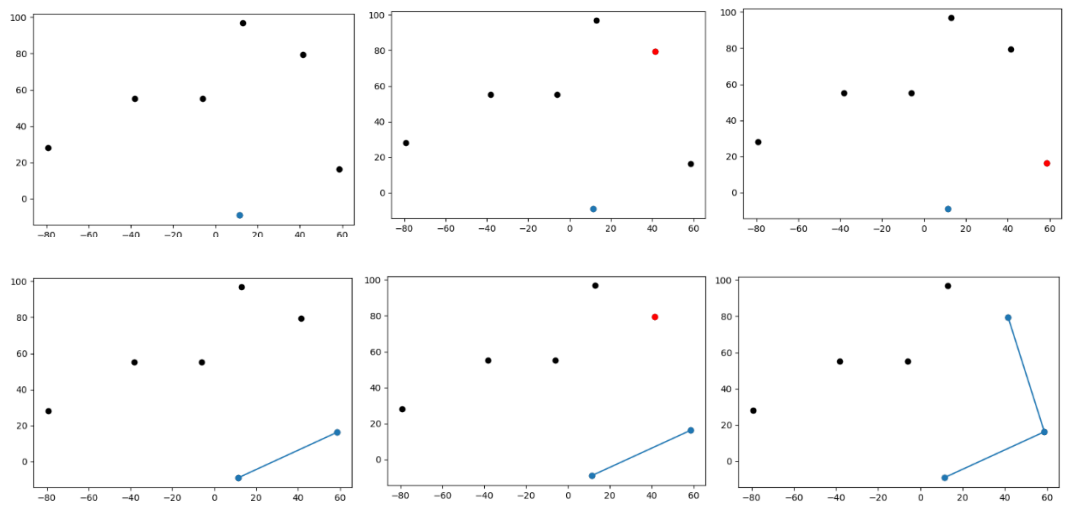
1.2 Algorytm Jarvisa

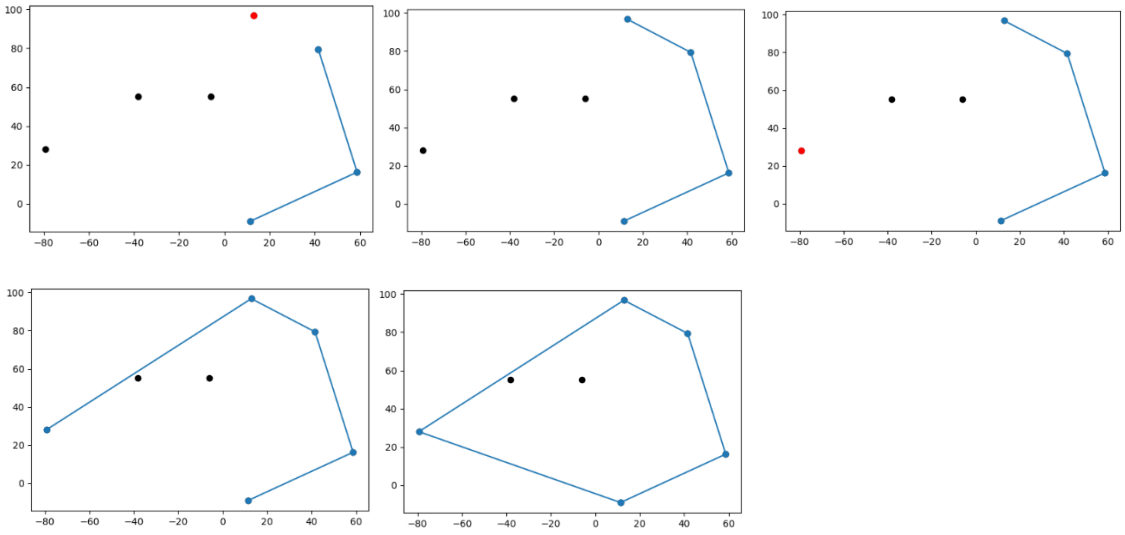
Algorytm, opracowany niezależnie przez Donalda Chanda i Shama Kapura w 1970r. oraz R. Jarvisa w 1973 r. przypadku dla płaszczyzny. Złożoność czasowa algorytmu to O(k*n), gdzie k to ilość punktów należących do otoczki wypukłej, a n jest liczbą wszystkich punktów należących do zbioru.

Poniżej przedstawiono algorytm w postaci listy kroków:

1. Wybranie punktu p_0- punktu o najmniejszej współrzędnej y, spośród wszystkich punktów (W przypadku remisów wybrany zostaje punkt o najmniejszej współrzędnej x).
2. Znalezienie punktu N dla którego kąt liczony przeciwnie do wskazówek zegara w odniesieniu do ostatniej krawędzi otoczki jest najmniejszy i dodanie go do listy punktów otoczki wypukłej. (A więc takiego, który z ostatnim wybranym punktem N' wyznaczy prostą, taką, że żaden punkt nie znajduje się po jej prawej stronie).
3. Powtarzanie punktu 2. dopóki N != p_0.

Zbiór rysunków 2. Przykładowy przebieg algorytmu Jarvisa:





Linie niebieskie, to obecny wygląd otoczki wypukłej, a czerwone punkty- punkty będące obecnie punktami tworzącymi najmniejszy kąt z ostatnią krawędzią spośród przeglądniętych.

2. Specyfikacja środowiska, w którym wykonano ćwiczenie

Ćwiczenie zostało wykonane przy pomocy jupyter notebook i Python 3 na komputerze z systemem Windows 10 (64 bitowym) i procesorem Intel Core i-5- 9300H (64 bitowy). Wykresy wygenerowano przy pomocy dostarczonego na laboratoria narzędzia graficznego i bibliotek matplotlib, random i time.

3. Sposób wykonania ćwiczenia

Pierwszym etapem ćwiczenia było wygenerowanie zestawów danych, które można podzielić na cztery rodzaje:

- tab1- zawierający losowo wygenerowane punkty o współrzędnych z przedziału [-100, 100],
- tab2- zawierający losowo wygenerowane punkty leżące na okręgu o środku (0,0) i promieniu R= 10,
- tab3- zawierający losowo wygenerowane punkty leżące na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10),
- tab4- zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo na bokach i przekątnych kwadratu.

Do wygenerowania powyższych zestawów użyto funkcji uniform z biblioteki random, która generuje zmienne pseudolosowe typu float64.

Następnie zaimplementowano algorytmy Grahama i Jarvisa, które poza wynikiem będącym listą punktów należących do otoczki wypukłej umożliwiały również tworzenie wykresów[1] obrazujących działanie algorytmów. Zaimplementowano również wersje algorytmów pozbawione możliwości rysowania wykresów. Na tych wersjach przetestowano czas działania algorytmów dla wygenerowanych zbiorów danych- miało to na celu zminimalizowanie wpływu funkcji niezwiązanych z działaniem algorytmu, takich jak dodanie sceny do wykresu, na czas wykonywania się.

Do określania wzajemnego położenia punktów użyto wyznacznika 3x3, i tolerancji błędu 10^{-11}.

4. Czas działania algorytmów dla poszczególnych zbirów danych

Poniższe tabele zawierają kolumny: n- określającą liczbę punktów w zbiorze, k- określającą liczbę punktów należących do otoczki wypukłej, t_G- czas działania algorytmu Grahama, t_J- czas działania algorytmu Jarvisa, $t_G/\ln(n)$- czas działania algorytmu Grahama podzielony przez ln(n) oraz t_J/k- czas działania algorytmu Jarvisa podzielony przez wartość k. Dwie ostatnie kolumny mają na celu sprawdzenie czy czasy działania algorytmów pokrywają się z teoretyczną złożonością czasową. Dzieląc t_J i t_G przez wspomniane wartości zgodnie z założeniem teoretycznym otrzymamy liczby, które powinny znajdować się na prostej przechodzącej przez punkt (0,0). Wynika to z faktu, że złożoność algorytmu Grahama- $O(nlog(n))$ po podzieleniu przez ln(n) da iloczyn n i stałego współczynnika. Podobnie dla algorytmu Jarvisa po podzieleniu czasu przez wartość k, wynikiem będzie n przemnożone przez stałą.

Liczba n dla zestawu tab4 oznacza, że w zestawie zawiera się (n-4) * 5/9 punktów leżących na bokach kwadratu, (n-4) * 4/9 punktów leżących na przekątnej kwadratu oraz 4 wierzchołki kwadratu. (Wartości zaokrąglano do najbliższych liczb naturalnych.)

Tabela 1. Czasy uzyskane dla zestawów tab1.

n	k	t_G	t_J	t_J/k	t_G/ ln(n)
1000	19	0,0125	0,0247	0,0013	0,0018
2000	21	0,0316	0,0452	0,0022	0,0042
3000	23	0,0471	0,0744	0,0032	0,0059
4000	29	0,0614	0,0996	0,0034	0,0074
5000	22	0,0740	0,1191	0,0054	0,0087
6000	22	0,0962	0,1386	0,0063	0,0111
7000	23	0,1267	0,1787	0,0078	0,0143

[1] Należy zwrócić uwagę na fakt, że w przypadku algorytmu Grahama rysowane są również punkty odrzucone w 3. Kroku algorytmu- punkty te nie są rozpatrywane przez algorytm. Są jednak rysowane dla zwiększenia czytelności algorytmu.

Tabela 2. Czasy uzyskane dla zestawów tab2.

n	k	t_G	t_J	t_J/k	t_G/ ln(n)
1000	1000	0,0114	1,0716	0,0011	0,0017
2000	2000	0,0326	4,6323	0,0023	0,0043
3000	2993	0,0413	11,5666	0,0039	0,0052
4000	3986	0,0529	22,0800	0,0055	0,0064
5000	4978	0,0755	30,4447	0,0061	0,0089
6000	5976	0,0817	46,0291	0,0077	0,0094
7000	6939	0,1071	65,1565	0,0094	0,0121

Tabela 3. Czasy uzyskane dla zestawów tab3.

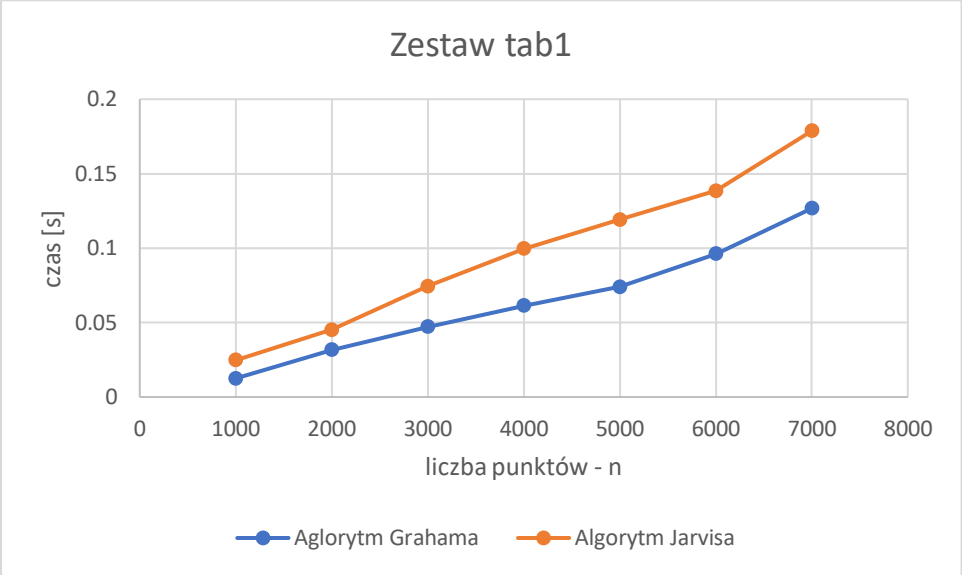
n	k	t_G	t_J	t_J/k	t_G/ ln(n)
1000	8	0,0141	0,0145	0,0018	0,0020
2000	8	0,0312	0,0228	0,0029	0,0041
3000	8	0,0463	0,0349	0,0044	0,0058
4000	8	0,0614	0,0480	0,0060	0,0074
5000	8	0,0779	0,0572	0,0072	0,0091
6000	8	0,0925	0,0760	0,0095	0,0106
7000	8	0,1217	0,0831	0,0104	0,0137

Tabela 4. Czasy uzyskane dla zestawów tab4.

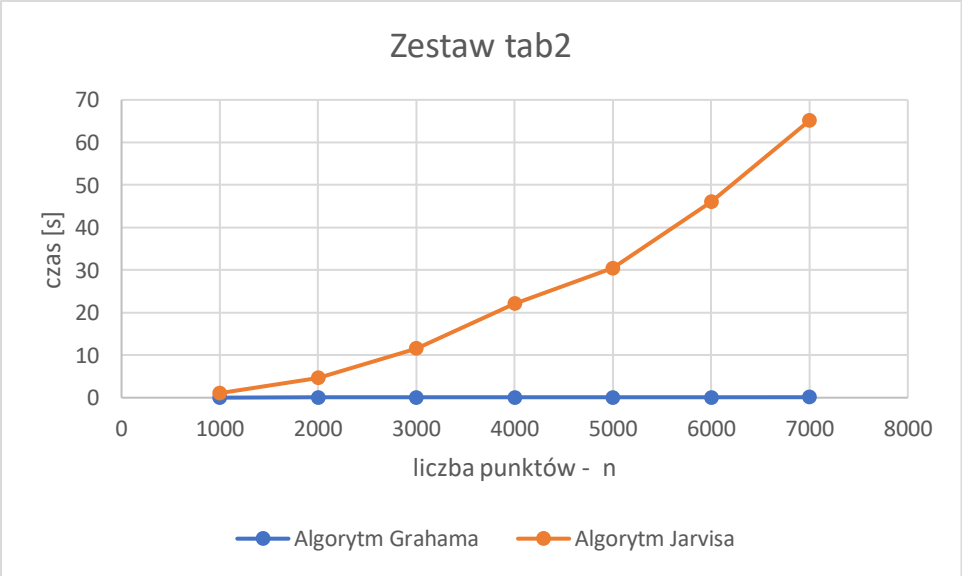
n	k	t_G	t_J	t_J/k	t_G/ ln(n)
1000	4	0,0132	0,0061	0,0015	0,0019
2000	4	0,0351	0,0124	0,0031	0,0046
3000	4	0,0484	0,0178	0,0045	0,0060
4000	4	0,0625	0,0235	0,0059	0,0075
5000	4	0,0713	0,0293	0,0073	0,0084
6000	4	0,0879	0,036	0,0090	0,0101
7000	4	0,1131	0,0431	0,0108	0,0128

Dane z powyższych tabel zostały zilustrowane na wykresach:

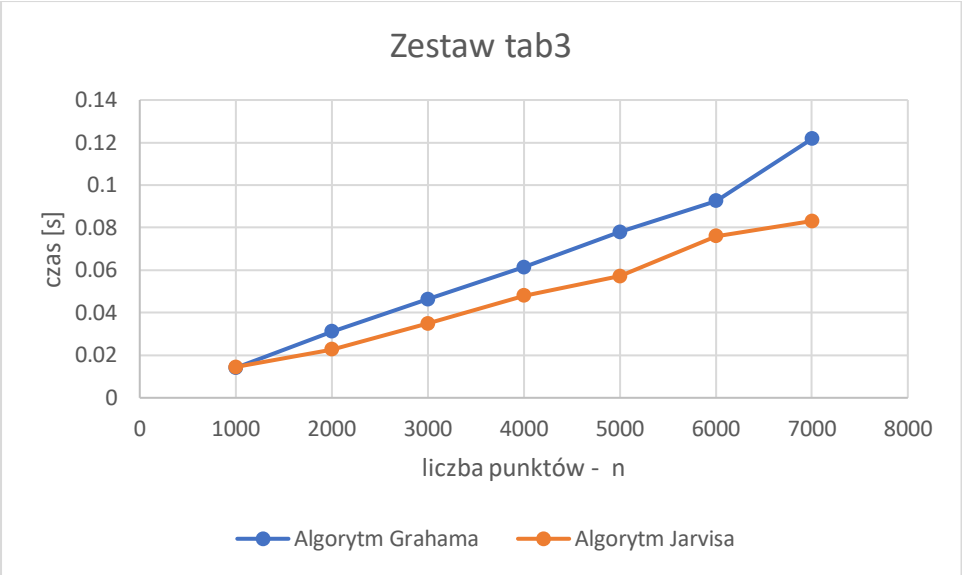
Wykres 1. Czasy uzyskane dla zestawów tab1.



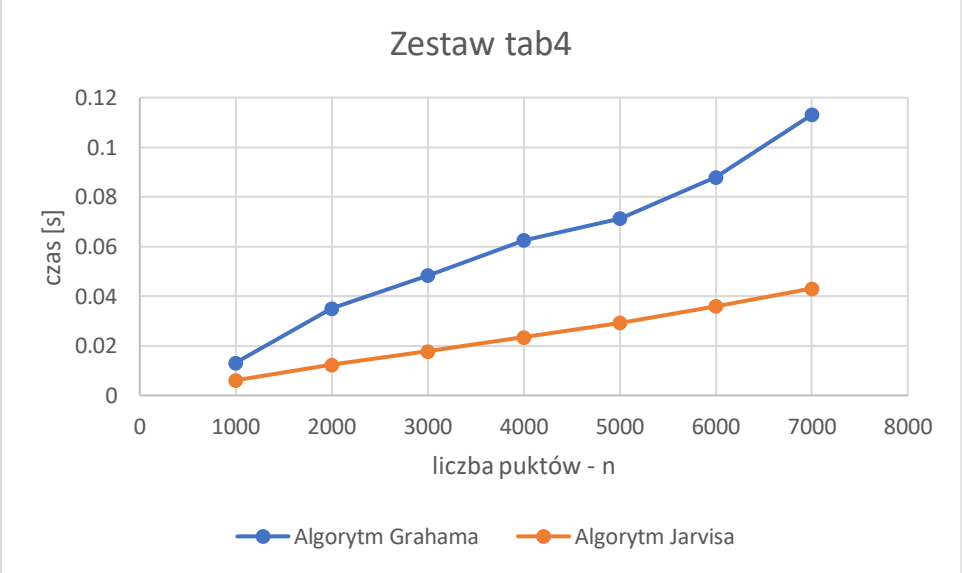
Wykres 2. Czasy uzyskane dla zestawów tab2.



Wykres 3. Czasy uzyskane dla zestawów tab3.

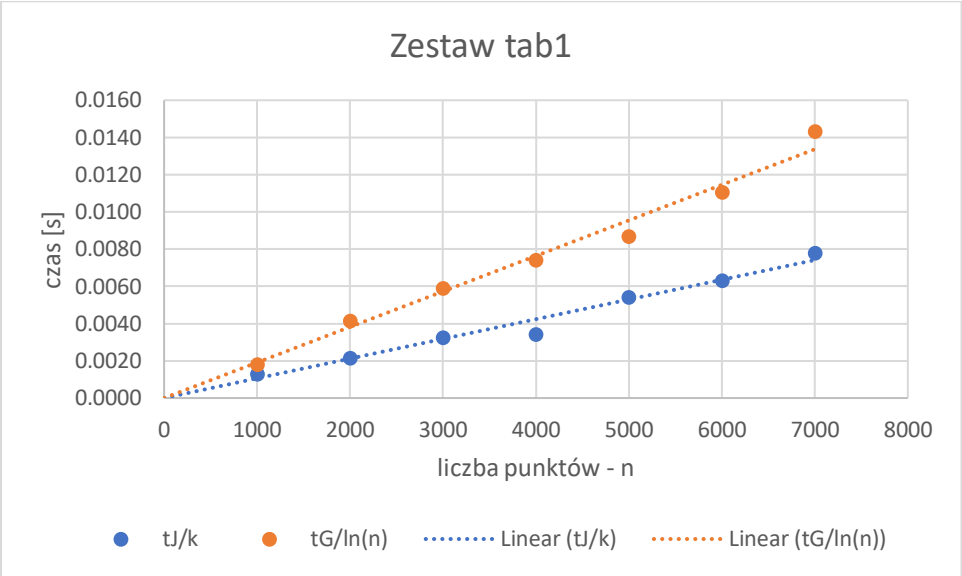


Wykres 4. Czasy uzyskane dla zestawów tab4.

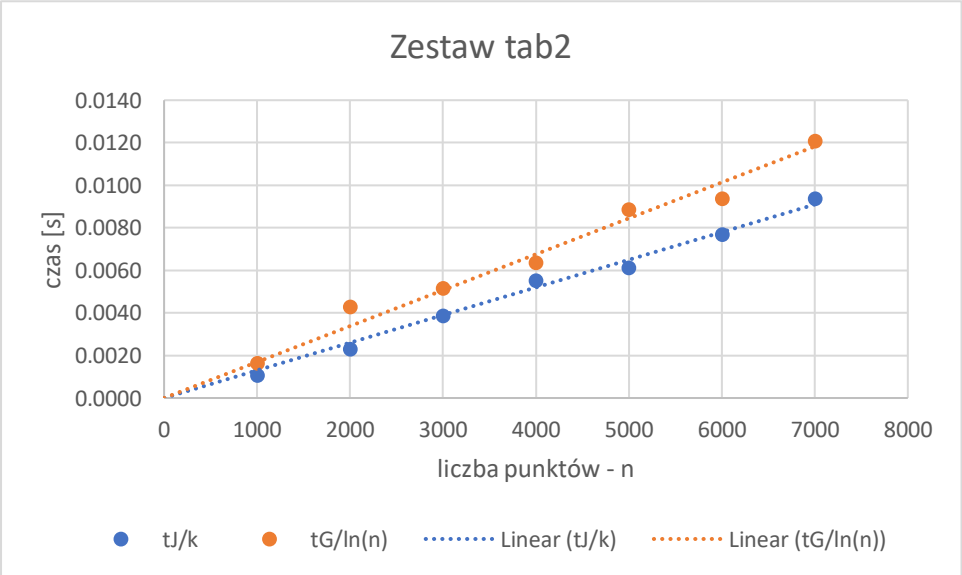


Sporządzono również wykresy ilustrujące wartości kolumn 3. i 4. Powyższych tabel. Do punktów dopasowano prostą przechodzącą przez punkt (0,0). Powyższe wykresy mają na celu pokazanie, że t_J/k i t_G/ln(n) są w sposób liniowy zależne od n. W szczególności nie należy wyciągać wniosków dotyczących relacji pomiędzy wartościami. (Przykładowo: czas algorytmu Grahama został podzielony przez ln(n). Gdyby zamiast tego wybrać log(n), to wartości byłyby inne, jednak kształt krzywej- zachowany.)

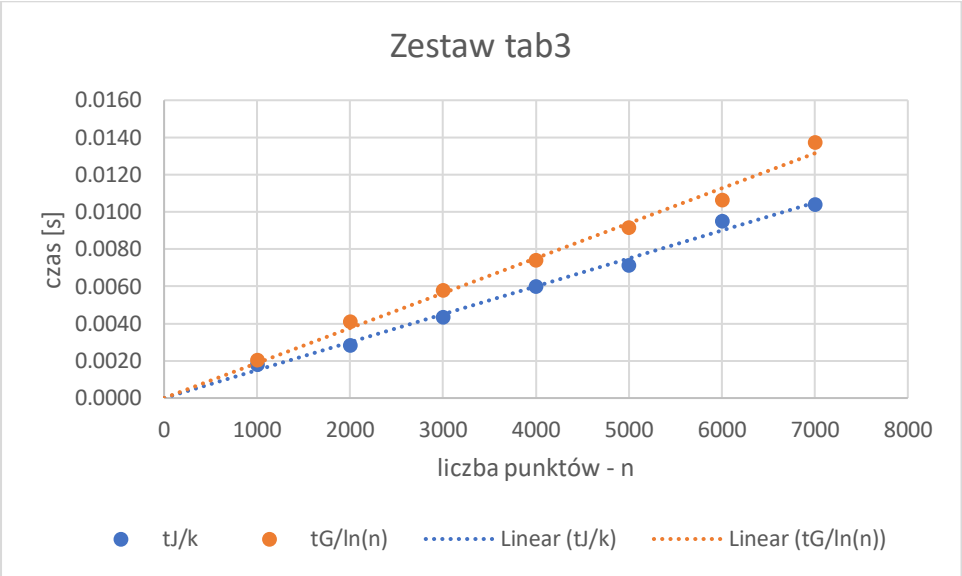
Wykres 5. t_J/k i t_G/ln(n) dla zestawów tab1.



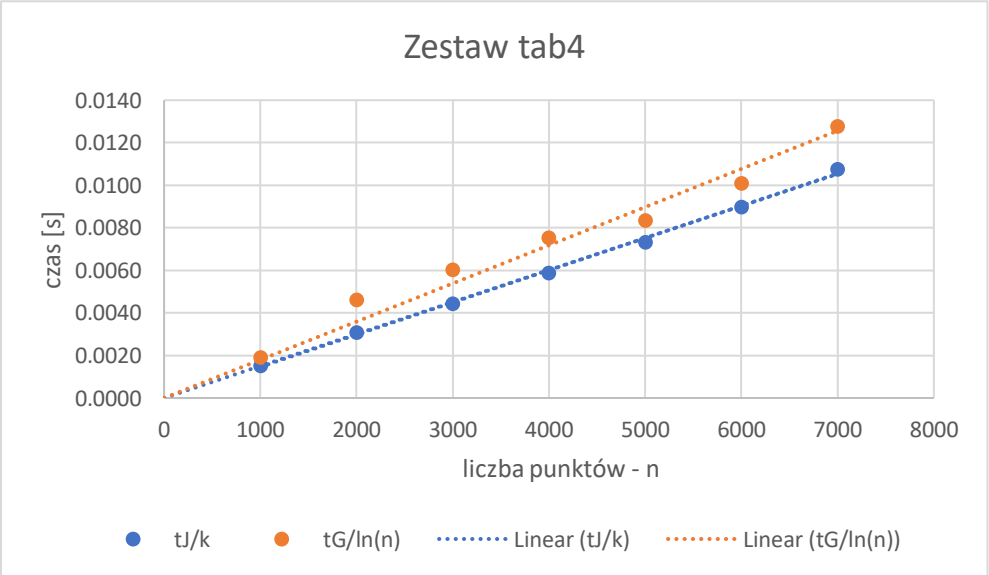
Wykres 6. t_J/k i t_G/ln(n) dla zestawów tab2.



Wykres 7. t_J/k i t_G/ln(n) dla zestawów tab3.



Wykres 8. t_J/k i t_G/ln(n) dla zestawów tab4.



5. Obserwacje i wnioski

5.1 Problemy napotkane w czasie implementacji algorytmów i wnioski dotyczące implementacji:

- Początkowo algorytm Grahama oparty był o sortowanie korzystające z algorytmu quicksort. Taka implementacja spowodowała jednak znaczne wydłużenie czasu pracy algorytmu, szczególnie zauważalne dla zestawów tab3 i tab4. Było to konsekwencją faktu, że w tych zestawach znaczna część punktów tworzy taki sam kąt z prostą OX. (punkty na bokach figur). Algorytm quicksort korzystający z metody wyznaczania pivotu Lomuto w takiej sytuacji często dokonywał niezrównoważonych podziałów punktów, a cały algorytm w rezultacie miał złożoność O(n^2). W celu poprawy działania algorytmu Grahama zamiast quicksort wykorzystano algorytm mergesort, który w każdym przypadku ma złożoność O(nlog(n)). Wadą takiego rozwiązania jest większa złożoność pamięciowa algorytmu. Nie jest to jednak bardzo istotny problem, jako że punkty reprezentowane są przez krotki, co czyni zestaw danych zajmującym stosunkowo niewiele pamięci. Alternatywą mogłaby być modyfikacja quicksorta dzieląca punkty na 3 grupy- mniejsze, większe i równe pivotowi. Wciąż jednak możliwa byłaby sytuacja, gdy punkty byłyby wstępnie posortowane, wtedy algorytm quicksort znów miałby złożoność O(n^2).

- Dla tolerancji ε = 10^{-12} zauważono następującą anomalię- czasami algorytm Grahama kończył się błędem wykonania wynikającym z wyjścia poza zakres tablicy. Było to spowodowane błędną klasyfikacją punktów o takim samym kącie α jako nie współliniowe (a nawet jako położone poniżej prostej wyznaczonej przez p_0 i oś OX), co skutkowało nie usunięciem ich w kroku 3 algorytmu. 2 z tych punktów były następnie odkładane na początkowy stos i brały udział w wyliczaniu wyznacznika warunkującego ściąganie punktów ze stosu. W szczególności te nieprawidłowości mogły doprowadzić do sytuacji, gdy ze stosu zostają zdjęte wszystkie punkty poza p_0, co powoduje, że następne obliczenie wyznacznika odwołuje się do pustego stosu. Problem rozwiązano dodając do warunku na ściągnie punktu ze stosu konieczności posiadania przez stos co najmniej dwóch elementów, co jest sprawdzane przed policzeniem wyznacznika. Wspomniany błąd był powodem, dla którego tolerancję ε zwiększono do 10^{-11}, dzięki czemu więcej punktów jest sklasyfikowane jako współliniowe- pomimo rozwiązania krytycznego problemu fakt, że punkt współliniowy był klasyfikowany jako leżący pod prostą nie był pożądany i mógł być źródłem innych problemów. Należy jednak podkreślić, że nowa wartość tolerancji, również nie jest idealnym wyborem- zdarza się, że punkty są klasyfikowane jako współliniowe, gdy nie jest to porządne. Można to zaobserwować w tabeli 2., gdzie nie wszystkie punkty zostały wybrane do otoczki, z racji, że niektóre z nich zostały uznane za współliniowe (szczególnie widać to dla większych zestawów danych, gdzie punkty są umieszczone bliżej siebie). Pomimo tego, ε = 10^{-11} uznano za lepszy wybór, jako, że błędy powodowane przez niego regularnie występują tylko dla jednego rodzaju zestawów danych (W przypadku bardziej losowych punktów, szanse, że dwa punkty będą tak umiejscowione, że algorytm błędnie uzna je za współliniowe jest znikoma.)

- Zasadniczo algorytm Grahama może działać z pominięciem punktu 3. W zaimplementowanej wersji nie można jednak po prostu usunąć wywołania punktu za to odpowiedzialnej, gdyż będzie to skutkować stworzeniem błędnej otoczki[2]. Zaimplementowany algorytm w przypadku, gdy rozważany punkt jest współliniowy z ostatnią krawędzią obecnie istniejącej otoczki zawsze uzna, że do otoczki powinien należeć punkt rozpatrywany a nie ten będący już w otoczce. Innym podejściem jest porównanie podejście oparte na porównaniu odległości punktów współliniowych od 2. punktu (licząc od wierzchu) na stosie, które nie wymaga zastosowania kroku 3. Pierwsze podejście również jest poprawne ponieważ:
  - Jeśli punkty o których mowa byłyby współliniowe ze sobą i z p_0- to został tylko ten najdalszy od p_0 – jest to zapewnione przez krok 3, a więc wybrano właściwy punkt. (Punkt bliższy znajdowałby się we wnętrzu otoczki).
  - Jeśli punkty o których mowa byłyby współliniowe ze sobą i nie były współliniowe z p_0, to wybrany zostanie punkt o największym kącie α, który równocześnie jest punktem bardziej oddalonym od 2. punktu (licząc od wierzchu), mającego kąt α najmniejszy spośród trzech omawianych punktów(a więc bliższego do punktu o mniejszym α jako, że wszystkie trzy punkty są współliniowe), na stosie- wybrano poprawnie.

5.2 Wnioski związane z czasem wykonywania i poprawnością algorytmów

- Nie zaobserwowano różnicy w poprawności algorytmów. Algorytmy działały poprawnie dla wybranej tolerancji $\varepsilon = 10^{-11}$ poza przypadkiem, gdy duża liczba punktów została umieszczona na odpowiednio małym okręgu (tab2). Przyczyna tego faktu jest szerzej opisane w 2. akapicie punktu 5.1.

- Na podstawie wykresów 5-8 można stwierdzić, że czas wykonywania się algorytmów był zgodny z przewidywaną złożonością czasowa wynoszącą O(k*n) dla algorytmu Jarvisa i O(n log(n)) dla algorytmu Grahama.

- Algorytm Grahama miał podobny czas wykonania niezależnie od typu zestawu danych.

- W przypadku, gdy punkty były generowane w sposób najmniej przewidywalny (tab1) lepszy okazał się być algorytm Grahama. Związane jest to z faktem, że funkcja log(n) rosła wolniej niż wartość k.

- Algorytm Jarvisa dla punktów na okręgu (tab2) w praktyce miał złożoność O(n^2), ponieważ wszystkie[3] punkty wchodziły w skład otoczki wypukłej. Spowodowało to, że był znacznie wolniejszy (ponad 600 razy dla n =7000) od algorytmu Grahama.

- Algorytm Jarvisa okazał się być lepszy dla zestawów, gdzie stosunkowo mała ilość punktów należała do otoczki (tab3 i tab4). W praktyce jego złożoność dla takich zestawów punktów można było traktować jako O(n).

[2] Należy również zastosować metodę wyboru punktów spośród współliniowych opartą o liczenie dystansu, opisaną w dalszej części akapitu.
[3] Z dokładnością do błędu spowodowanego niewłaściwą klasyfikacją punktów.

- Krok algorytmu Grahama polegający na usunięciu punktów o takim samym kącie α z p_0 był szczególnie istotny dla zestawów tab3 itab4. Pozwolił on na pominięcie analizy prawie[4] wszystkich punktów na osiach wykresu. W kodzie dostarczonym do ćwiczenia znajduje się implementacja funkcji porównującej czas działania algorytmu Grahama z usuwaniem i bez usuwania punktów. Wyniki porównania nie są jednak zawarte w sprawozdaniu- wielokrotnie zdarzało się, że dla takiego samego zestawu danych raz szybsza była jedna wersja algorytmu, a po chwili- druga, wobec tego nie można z nich wywnioskować, czy wprowadzona modyfikacja faktycznie jest użyteczna. Najprawdopodobniej wynika to z faktu, że sama operacja usuwania elementu z listy jest kosztowna czasowo i zdarza się, że usuwanie elementów trwa dłużej niż trwałoby przejrzenie ich w algorytmie Grahama.

A. G. z usuwaniem	A. G. bez usuwania
0.1680	0.1419
0.1409	0.1250
0.1300	0.1310
0.1306	0.1529
0.1369	0.1429
0.1420	0.1410

Tabela 5. Przykładowe czasy wykonywania się obu wersji algorytmu Grahama dla zestawu tab4 z 2500 punktami na brzegach i 2000 na przekątnych wyrażone w sekundach dla kilku prób.

- Zbiory typu tab1 pozwoliły na zobrazowanie działania algorytmów dla przypadkowych punktów. Zbiory typu tab2 obnażyły główną słabość algorytmu Jarvisa– pesymistyczną złożoność O(n^2) Zbiory typu tab3 i tab4 pozwoliły z kolei na zaobserwowanie przewag algorytmu Jarvisa nad algorytmem Grahama (algorytm Jarvisa był dla obu zestawów praktycznie liniowy). Dodatkowo zestawy te miały wiele punktów, które zostały usunięte w podstawowej wersji algorytmu jako punkty o tym samym kącie, co pozwoliło na porównanie dwóch wersji algorytmu Grahama (i stwierdzenie, że nie ma między nimi zależności szybszy- wolniejszy).

[4] Oprócz p_0 na osiach musiały zostać jeszcze punkty pełniące rolę wierzchołków kwadratu (tab3) lub będące najbliżej wierzchołkom prostokąta tab(4)