Algorytmy geometryczne

Sprawozdanie - ćwiczenie 2.

Dominik Adamczyk

Gr. 2

1. Specyfikacja techniczna urządzenia, na którym wykonano ćwiczenie

- System: Windows 11 Home x64
- Procesor: Intel Core i5-1135G7 2.4MHz – 4.2 MHz
- Pamięć ram: 16 Gb
- Język programowania: Python 3.10 z bibliotekami numpy, matplotlib, time, random
- Środowisko: Jupyter Notebook z użyciem narzędzia załączonego na potrzeby laboratoriów
- Narzędzie online do tworzenia animacji gif – ezgif.com

2. Treść i cel ćwiczenia

Ćwiczenie polegało na implementacji dwóch algorytmów znajdowania otoczki wypukłej i porównaniu ich. Aby to zrobić na początku konieczne było wygenerowanie odpowiednich zbiorów punktów, a następnie stworzenie funkcji, które mogą generować analogiczne zbiory na podstawie parametrów wejściowych. Następnie zaimplementowane zostały dwa algorytmy znajdowania otoczki: Algorytm Grahama oraz Algorytm Jarvisa. Obydwa algorytmy zostały porównane pod kątem poprawności i wydajności dla różnych danych wejściowych.

3. Zbiory danych

Na potrzeby sprawdzenia poprawności algorytmów oraz ich porównania, wygenerowane zostały 4 podstawowe zbiory punktów (punkty są liczbami rzeczywistymi):

- Dane A: 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100;100]
- Dane B: 100 losowo wygenerowanych punktów leżących na okręgu o środku w punkcie (0;0) i promieniu R = 10
- Dane C: 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10)
- Dane D: wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) wraz z losowo wygenerowanymi punktami, tak że po 25 punktów leży na bokach kwadratu pokrywających się z osiami OX i OY, a po 20 punktów leży na przekątnych kwadratu

Punkty zostały wygenerowane przy pomocy funkcji random.uniform z biblioteki random. Do generowania punktów ze zbioru B skorzystano z parametryzacji okręgu jako $(\cos t\,, \sin t)$. Funkcje trygonometryczne pochodzą z biblioteki numpy.

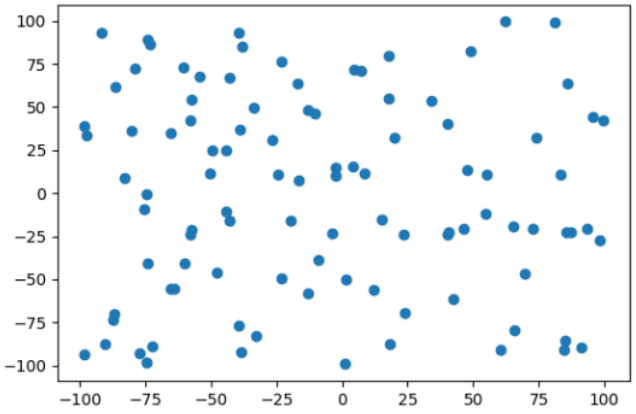
Kolejnym etapem było stworzenie funkcji generujących zbiory o analogicznej budowie, jak poprzednio wymienione zbiory punktów, ale z możliwością podania parametrów. W taki sposób powstały cztery funkcje:

- genDataA(quantity, minX, maxX, minY, maxY), gdzie quantity to ilość punktów, a pozostałe cztery parametry określają prostokąt [minX; maxX] x [minY; maxY], w którym są zawarte
- genDataB(quantity, centreX, centreY, rad), gdzie quantity to ilość punktów, centreX, centreY określają położenie środka okręgu, a rad to jego promień
- genDataC(quantity, vertices), gdzie quantity to ilość punktów, a vertices to tablica z wierzchołkami prostokąta
- genDataD(axisQuantity, diagonalQuantity, vertices), gdzie axisQuantity to ilość punktów na lewej i dolnej ścianie kwadratu, diagonalQuantity to ilość punktów na przekątnych kwadratu, a vertices to wierzchołki kwadratu.

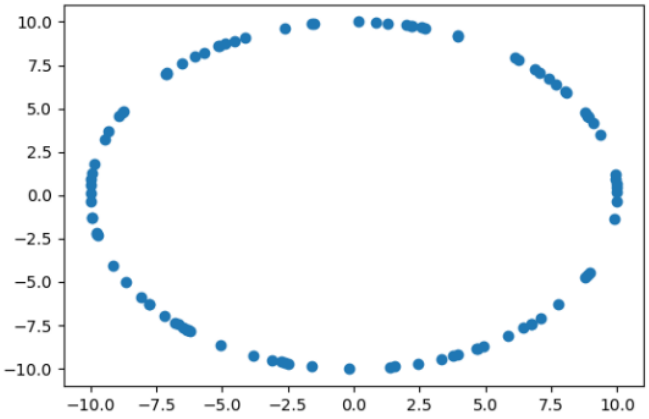
W przypadku powyższych funkcji przyjęte jest założenie, że dane wejściowe są poprawne (to znaczy, że np. tablica vertices ma poprawnie określone wierzchołki), oraz że zarówno kwadrat jak i prostokąt z funkcji genDataC i genDataD mają ściany prostopadłe/równoległe do osi układu współrzędnych. W przeciwnych przypadkach działanie funkcji może skutkować błędami.

Poniższe grafy prezentują podstawowe zbiory punktów, na których sprawdzane będą poprawności implementowanych algorytmów.

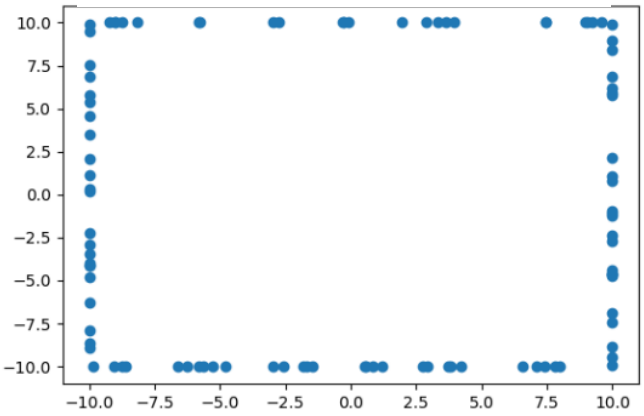
Rys. 1. Zbiór danych A



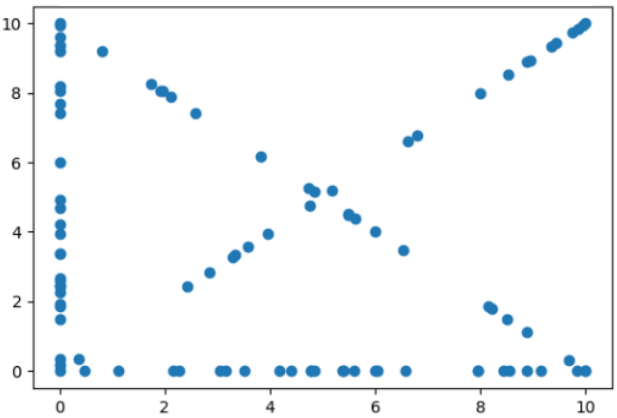
Rys. 2. Zbiór danych B



Rys. 3. Zbiór danych C



Rys. 4. Zbiór danych D



4. Funkcje pomocnicze

Do implementacji badanych algorytmów konieczne będzie użycie funkcji pomocniczych. Algorytm Grahama będzie wymagał posortowania punktów rosnąco względem kąta między wektorami [1, 0] (zaczepionym w punkcie O) oraz OP, gdzie O to zadany punkt początkowy, a P to punkt sortowany. Dla algorytmu Jarvisa konieczne będzie znajdowanie punktu R o najmniejszym kącie pomiędzy wektorami [1, 0] (zaczepionym w punkcie S), a wektorem SR. Na potrzeby tych wyliczeń korzystać będę z funkcji liczącej wyznacznik, przyjmującej 3 punkty (a, b, c) i zwracającej informację o orientacji punktu c względem prostej ab. Wyznacznik jest liczony przy pomocy wzoru $det(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$. Przyjęta tolerancja dla zera to 1e-12. Wzór oraz tolerancja podyktowane są wynikami poprzedniego laboratorium – wyznacznik zapewniał oczekiwaną dokładność dla tej tolerancji oraz nie zwracał

nieoczekiwanych wyników, co miało miejsce w przypadku wyznacznika 2x2. Dla obydwu algorytmów wyznaczających otoczkę, w przypadku punktów współliniowych następuje ich sortowanie pod względem odległości pomiędzy punktem początkowym, a punktem sortowanym.

5. Algorytm Grahama – implementacja

W przypadku algorytmu Grahama pierwszym krokiem jest wyznaczenie punktu O o najmniejszej współrzędnej y i x (dla punktów z tą samą współrzędną y). Ten wyznaczony punkt staje się pierwszym należącym do otoczki. Następnie wykonywane jest sortowanie punktów po ich kącie, tak jak jest to opisane w poprzednim punkcie. W celu sortowania zaimplementowałem algorytm quicksort, w taki sposób, aby dla dwóch punktów sprawdzał, który z nich tworzy mniejszy kąt (na zasadach opisanych w poprzednim punkcie), a dla tych samych kątów, sortował po odległości. Dla posortowanej tabeli punktów wykonywany jest główny algorytm. Pierwsze trzy punkty umieszczane są na stosie. Następnie wykonywana jest pętla, która dla każdego z pozostałych punktów sprawdza, czy punkt ten jest położony po lewej stronie prostej utworzonej przez ostatnie dwa punkty ze stosu. Jeżeli punkt leży po prawej stronie, bądź jest współliniowy, to wykonywana jest operacja zdejmowania ostatniego elementu ze stosu, aż rozważany punkt nie będzie po lewej stronie prostej – wtedy jest kładziony na stos. Po wykonaniu tej pętli dla wszystkich punktów na stosie będą znajdować się elementy otoczki w kolejności umożliwiającej połączenie kolejnych punktów ze sobą. Złożoność tego algorytmu jest zależna jedynie od ilości n punktów w zbiorze i wynosi O(nlogn).

6. Algorytm Jarvisa – implementacja

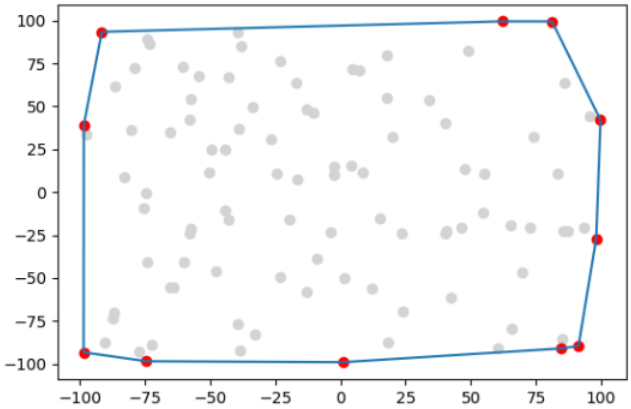
Dla algorytmu Jarvisa tak samo, jak w przypadku algorytmu Grahama wyznaczany jest punkt O o najmniejszej współrzędnej y, a dla tych samych wartości y – o najmniejszej współrzędnej x. Kolejnym etapem jest znalezienie takiego punktu P, że nie istnieje punkt, który byłby po prawej stronie prostej OP. Ten proces jest zapętlony w taki sposób, że po znalezieniu punktu P szukany jest punkt R, taki że nie istnieje punkt który byłby po prawej stronie prostej PR itd. Pętla ta jest powtarzana, dopóki punkt O nie zostanie ponownie wybrany, jako punkt, przez który przechodzi prosta o powyższej własności – wtedy otoczka zostanie znaleziona. Szukanie punktu konstruującego prostą o zadanej własności odbywa się poprzez liniowe przeglądanie zbioru punktów i sprawdzanie orientacji przeglądanego punktu względem prostej wyznaczonej przez punkt należący do otoczki i kandydata na punkt należący do otoczki. Jeżeli przeglądany punkt leży po prawej stronie prostej, to należy zamienić kandydata na właśnie ten punkt. Złożoność tego algorytmu jest zależna od ilości punktów h w otocze oraz ilości n punktów w zbiorze i wynosi O(nh)

7. Działanie algorytmów – podstawowe przypadki

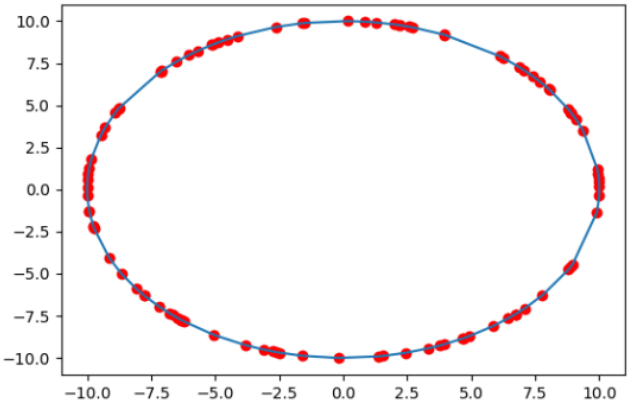
Zaimplementowane algorytmy Grahama i Jarvisa zostały sprawdzone dla podstawowych zbiorów opisanych w punkcie 3. Obydwa algorytmy zwróciły takie same wyniki, które pokrywały się z przewidywaniami: dla danych B liczba punktów otoczki wypukłej była równa liczbie punktów w zbiorze, dla danych C liczba punktów otoczki była równa 8 (prawdopodobieństwo otrzymania punktu na rogu jest zerowe), a dla danych D, gdzie do zbioru punktów zaliczały się wierzchołki kwadratu, liczba punktów otoczki wyniosła 4. Poniżej

prezentowana jest graficzna reprezentacja otoczki wypukłej tych zbiorów. Punkty otoczki zaznaczone są na czerwono.

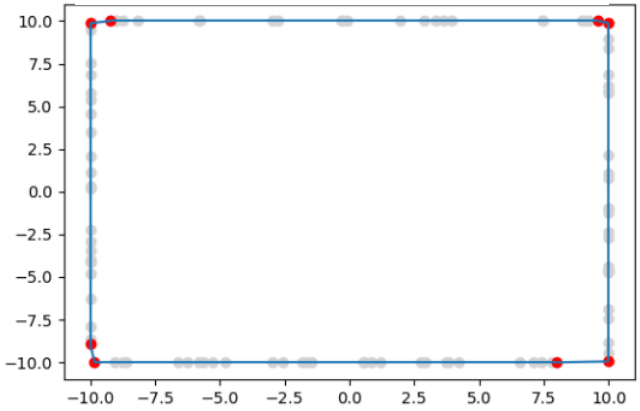
Rys. 5. Zbiór danych A – otoczka wypukła



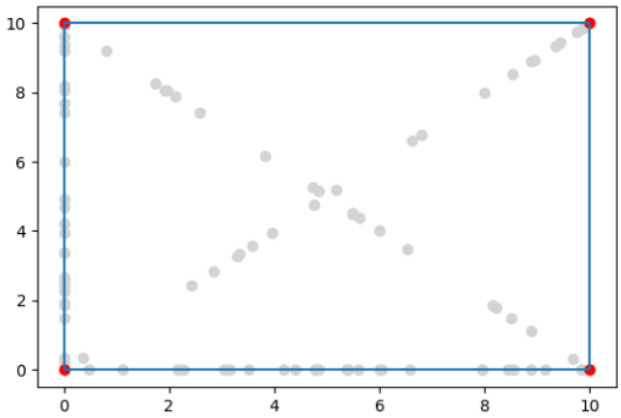
Rys. 6. Zbiór danych B – otoczka wypukła



Rys. 7. Zbiór danych C – otoczka wypukła



Rys. 8. Zbiór danych D– otoczka wypukła



Dodatkowo do sprawozdania załączone zostały wizualizacje działania algorytmów Grahama i Jarvisa dla powyższych danych. Znajdują się one w pliku .zip i są one animacjami zapisanymi w formacie .gif stworzonymi przy pomocy narzędzia online i biblioteki matplotlib. Wśród animacji nie pojawiła się wizualizacja algorytmu Jarvisa dla zbioru danych B, ze względu na dużą ilość kroków który wykonuje algorytm w tym przypadku.

Na animacjach algorytmu Grahama przyjęte są następujące oznaczenia kolorów: niebieski – punkty nie sprawdzone przez algorytm, szary – punkty sprawdzone przez algorytm, ale niezaklasyfikowane do otoczki, czerwony – punkty należące do otoczki, pomarańczowy – aktualnie rozpatrywany punkt, który będzie należał do otoczki, gdy znajdzie się po lewej stronie prostej.

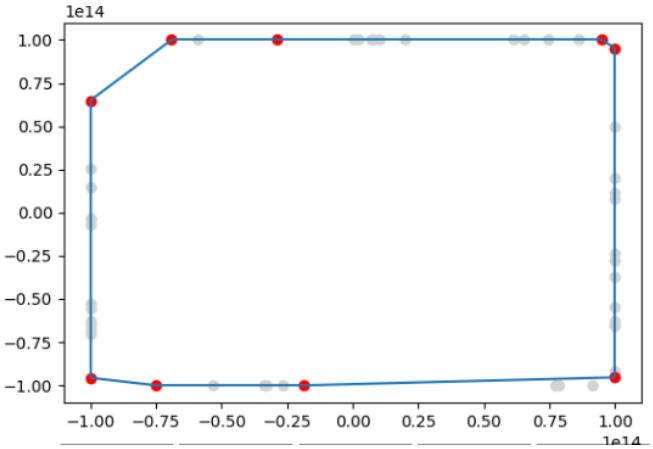
Dla animacji algorytmu Jarvisa przyjęte są następujące oznaczenia: niebieski – punkty zbioru, czerwony – punkty należące do otoczki, pomarańczowy – aktualny kandydat na następny punkt otoczki, żółty – punkt aktualnie porównywany z pomarańczowym, jeżeli punkt żółty leży po prawej stronie prostej wyznaczonej przez ostatni punkt na otoczce i punkt pomarańczowy, to pozycja punktu pomarańczowego zmienia się na pozycję punktu żółtego.

8. Działanie algorytmów – błędy w wynikach

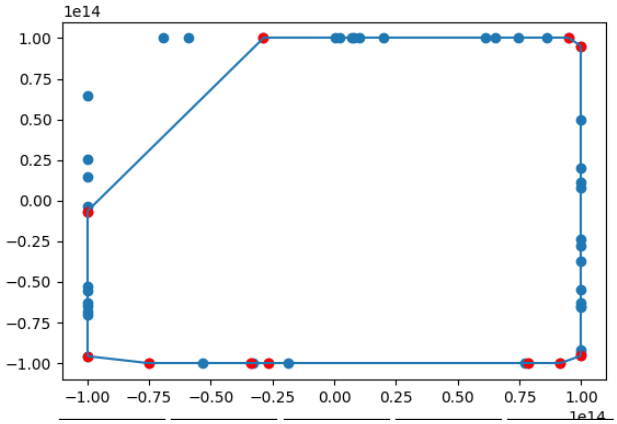
Przy odpowiednio dobranych wartościach parametrów wejściowych możliwe jest uzyskanie błędnych wyników dla tych implementacji algorytmów znajdowania otoczki. Przypadki źle wyznaczonej otoczki wypukłej możliwe są wtedy, gdy operacje przeprowadzane są dla dużych wartości, albo gdy zagęszczenie punktów jest zbyt duże.

Dla zestawów danych wygenerowanych na bazie danych C i danych D można zobaczyć błędy wynikające ze skończonej precyzji obliczeń, co przekłada się na złe określanie współliniowości punktów – w rezultacie niektóre punkty są pomijane jako część otoczki – w szczególności widać to na rys 10. Błędy dotyczą zarówno algorytmu Grahama jak i Jarvisa.

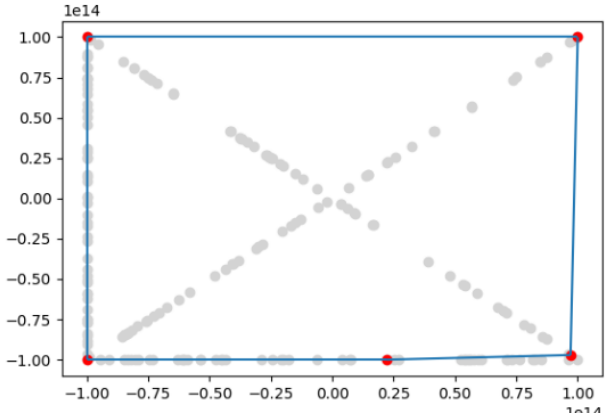
Rys. 9. Błędnie wyznaczona otoczka – Algorytm Grahama, dane C z przedziału [-1e14;1e14] oczekiwana ilość punktów w otoczce:8, otrzymano: 9



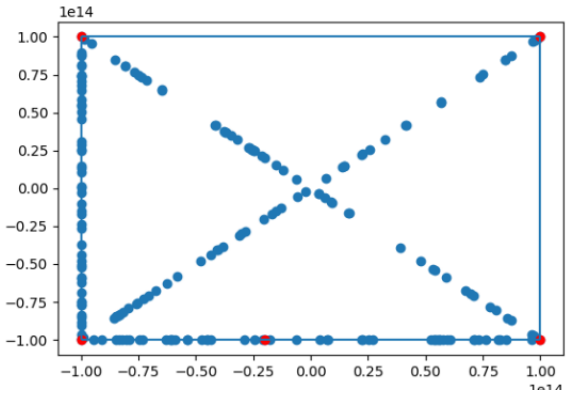
Rys. 10. Błędnie wyznaczona otoczka – Algorytm Jarvisa, dane C z przedziału [-1e14;1e14] oczekiwana ilość punktów w otoczce:8, otrzymano: 11



Rys. 11. Błędnie wyznaczona otoczka – Algorytm Grahama, dane D z przedziału [-1e14;1e14] oczekiwana ilość punktów w otoczce:4, otrzymano: 5



Rys. 12. Błędnie wyznaczona otoczka – Algorytm Jarvisa, dane D z przedziału [-1e14;1e14] oczekiwana ilość punktów w otoczce:4, otrzymano: 5



Kolejnym przykładem błędnego wyznaczenia otoczki jest przypadek z dużym zagęszczeniem nie współliniowych punktów na małym obszarze. Taki efekt można uzyskać np. poprzez wygenerowanie 1500 punktów na okręgu o promieniu 1. Wygenerowane w teście punkty były różne, więc teoretyczna otoczka powinna mieć 1500 punktów, jednakże algorytmy Grahama i Jarvisa zwracały otoczkę złożoną z 1494 punktów. Ponownie skończona precyzja obliczeń powoduje błędy – w tym wypadku zaliczając punkty jako współliniowe, chociaż te są rozmieszczone na okręgu.

9. Porównanie czasów działań algorytmów

Ostatnim etapem ćwiczenia jest porównanie wydajności algorytmów Grahama i Jarvisa dla różnych układów zbiorów wejściowych. W tym porównaniu dodatkowo uwzględniony został algorytm Grahama, który po etapie sortowania, a przed etapem wyznaczania otoczki, usuwa punkty współliniowe, które na pewno do otoczki nie będą należały (odbywa się to za pomocą liniowego przejścia po tablicy posortowanych punktów).

Testy wydajności przeprowadzone zostały dla czterech różnych schematów generowania danych – analogicznych do tych opisanych w punkcie 3., ale ze zmienionymi ilościami generowanych punktów. Badane ilości punktów wynoszą odpowiednio: 100, 1000, 5000, 10000, 15000, 30000, 50000 punktów. Dla zbiorów danych tworzonych na podstawie zbioru D liczba punktów jest cztery razy większa.

Punkty ze zbioru A rozmieszczono w przedziałach [-10, 10], dla zbioru B zamieniono promień na R=100, a punkty ze zbiorów C i D rozmieszczone są na planie kwadratu o wierzchołkach [(0, 0), (10, 0), (10, 10), (0, 10)]. Do wyznaczenia czasów użyto funkcji time z biblioteki time.

Tabela 1. Czas (podany w sekundach) znajdowania otoczki wypukłej w zależności od zastosowanego schematu generowania zbioru danych, algorytmu i ilości punktów.

Schemat generowania danych	Algorytm	Ilość punktów						
		100	1000	5000	10000	15000	30000	50000
Dane A	Grahama	0.01	0.02	0.09	0.19	0.33	0.76	1.44
	Grahama z filtrowaniem	0.00	0.02	0.11	0.22	0.35	0.82	1.53
	Jarvisa	0.00	0.03	0.17	0.40	0.54	1.51	2.40
Dane B	Grahama	0.01	0.04	0.21	0.63	0.67	1.45	-
	Grahama z filtrowaniem	0.00	0.03	0.18	0.64	0.72	1.50	-
	Jarvisa	0.02	1.96	53.68	203.73	458.35	1892.10	-
Dane C	Grahama	0.01	0.03	0.11	0.28	0.53	1.06	1.77
	Grahama z filtrowaniem	0.00	0.02	0.15	0.32	0.51	1.03	1.66
	Jarvisa	0.00	0.02	0.08	0.15	0.23	0.49	0.73
Dane D	Grahama	0.02	0.14	0.93	2.11	2.92	6.73	12.15
	Grahama z filtrowaniem	0.01	0.14	0.87	1.91	2.85	6.45	11.80
	Jarvisa	0.01	0.03	0.16	0.29	0.43	0.88	1.48

Na początku warto zwrócić uwagę na porównanie dwóch metod implementacji algorytmu Grahama. Czasy działania między różnymi implementacjami nie różnią się znacząco – wynika to z wymaganego dla obydwu algorytmów posortowania tablicy (co ma złożoność O(nlogn), podczas gdy reszta algorytmu jest liniowa). Filtrowanie punktów, które stanowi liniowe przejście po tablicy jest niekorzystne w przypadku zestawów punktów, które nie mają elementów współliniowych – wtedy takie liniowe przejście nie jest rekompensowane przez część algorytmu wyznaczającą otoczkę – obrazują to nieznacznie dłuższe czasy wykonania algorytmu Grahama z filtrowaniem dla zbiorów danych A i B. Natomiast gdy w zbiorach występują z dużym nagromadzeniem punkty współliniowe filtrowanie punktów może przyczyniać się do nieznacznie szybszych rezultatów działania. Największe różnice (chociaż

dalej niewielkie) można zaobserwować w przypadku danych D. Ze względu na dystrybucję danych odfiltrowane zostaje ¾ punktów. Jak się okazuje nawet tak znaczące zmniejszenie badanego zbioru skutkuje wynikami mniejszymi o średnio 5%.

Kolejnym etapem jest porównanie algorytmu Jarvisa do algorytmu Grahama. Jako że różnice pomiędzy dwoma wersjami algorytmu Grahama są nieznaczne, w porównaniu będę korzystał z algorytmu Grahama podstawowego.

Algorytm Jarvisa jest uzależniony od ilości punktów w otoczce i w zbiorze, podczas gdy Grahama jest zależny tylko od ilości punktów w zbiorze. Te różnice są silnie widoczne, gdy porównane zostaną czasy działań algorytmów dla danych B. W okręgu wszystkie punkty stanowią otoczkę, co sprawia, że algorytm Jarvisa staje się algorytmem o złożoności kwadratowej. Powoduje to znaczne odstępstwa w czasie wykonania na niekorzyść metody Jarvisa. Algorytm Jarvisa osiąga również gorsze wyniki dla zestawu losowych punktów – w tym przypadku liczba punktów na otoczce wynikowej jest nieokreślona, co sprawia, że metoda Jarvisa, dla której nieistotny jest porządek punktów wypada gorzej. Inaczej jest w przypadku, gdy liczba punktów na otoczce wynikowej jest mała w porównaniu do ogólnej liczby punktów. Takie przypadki obecne są w zbiorach C i D, gdzie oczekiwana liczba punktów otoczki to kolejno 8 i 4. W tych przypadkach algorytm Jarvisa osiąga złożoność wręcz liniową (O(8*n), O(4*n)) i jest w stanie wyznaczyć otoczkę w zaledwie paru iteracjach po zadanej liście, co skutkuje o wiele lepszym czasem wykonania w porównaniu do metody Grahama, która zawsze wymaga sortowania.

10. Wnioski

Zaimplementowane algorytmy Jarvisa i Grahama są w stanie wyznaczyć otoczkę wypukłą w poprawny sposób pod warunkiem, że pozwala na to precyzja obliczeń. Przykłady podane w treści zadania (a badane w punkcie 7) nie stwarzały problemów dla algorytmów. Dane A nie stanowiły problemu dla żadnego z algorytmów – w tym zbiorze nie było punktów współliniowych, które mogły zaburzyć wynik końcowy. Dla danych B sytuacja wyglądała podobnie, chociaż w tym przypadku odczuwalny był wolniejszy czas działania algorytmu Jarvisa (z powodów wymienionych w poprzednim punkcie). Algorytm Grahama natomiast znajdował poprawną otoczkę od razu – bez konieczności zdejmowania punktów ze stosu (można to zobaczyć na załączonej animacji), co jest rezultatem uporządkowania punktów i zaliczenia każdego z nich do otoczki końcowej. Dla zestawu danych B istniało ryzyko kwalifikacji punktów jako współliniowe (co jest zademonstrowane w punkcie z błędami), jednak przez niewielką ilość rozważanych punktów i odpowiednio dobrany wyznacznik taki błąd się nie pojawił. Obydwa algorytmy zwróciły oczekiwane wyniki dla zbioru danych C. W tym przypadku istniało ryzyko błędów związane ze współliniowością rozważanych punktów, jednak nie występowały one dla zadanego przedziału. Zbiór danych D również testował współliniowość punktów, ale także sprawdzał, czy w algorytmie Grahama poprawnie zaimplementowane jest zdejmowania punktów ze stosu otoczki. Ponownie jednak obydwa algorytmy poradziły sobie dobrze z tym zbiorem, wyznaczając otoczkę złożoną z czterech punktów będących wierzchołkami kwadratu. Wszystkie z powyższych testów obrazują dodatkowo, że dla algorytmu Jarvisa nie jest istotne rozłożenie punktów na płaszczyźnie, jeżeli nie wpływa ono na kształt otoczki wynikowej.

Zdarzają się również przypadki, gdy powyższe algorytmy zwracają wyniki błędne. Takie sytuacje (opisane w punkcie 8.) są skutkiem skończonej precyzji obliczeń i problemami z poprawnym wyznaczeniem punktów współliniowych dla wartości. Występują też problemy, gdy algorytm błędnie klasyfikuje punkty jako współliniowe, pomimo że takimi nie są. Aby

zapobiegać tym wypadkom należy odpowiednio dobierać tolerancje dla zera w zależności od badanego zbioru punktów.

Warto jest świadomie dobierać algorytm do zadanego zbioru punktów. W losowych przypadkach (takich jak dane A) dobrym rozwiązaniem może być algorytm Grahama, którego czas działania jest przewidywalny i zależny jedynie od ilości danych wejściowych. Gdy jednak podejrzewamy małą liczbę punktów w otoczce wynikowej, rozsądne będzie skorzystanie z algorytmu Jarvisa (szczególnie gdy liczba punktów w zbiorze jest znacznie większa), który osiąga zauważalnie lepsze wyniki wydajności. W przypadku algorytmu Jarvisa należy jednak uważać na przypadki, gdy ilość punktów otoczki jest zależna od ilości punktów zbioru wejściowego – wtedy staje się algorytmem o złożoności kwadratowej, który będzie osiągał znacznie niższe wyniki od metody Grahama. Uważam też, że dla algorytmu Grahama nie jest konieczne implementowanie filtrowania punktów. Wyniki wydajności uzyskane przy pomocy tej metody są bardzo zbliżone do tych z podstawowego algorytmu Grahama. Filtracja punktów przynosi jakiekolwiek rezultaty jedynie w przypadku bardzo specyficznych zbiorów danych, dla których i tak znacząco lepszym może okazać się algorytm Jarvisa.