Algorytmy 2
Laboratorium: powłoka wypukła, algorytm Grahama

Przemysław Klęsk

6 stycznia 2020

1 Cel

Celem zadania jest wykonanie implementacji *algorytmu Grahama* (ang. *Graham scan*), który dla podanego zbioru punktów na płaszczyźnie znajduje jego *powłokę wypukłą*[1] — czyli minimalny zbiór wypukły obejmujący wszystkie punkty. Rys. 1 ilustruje dwa przykłady tego zadania. Obwiednie wynikowych powłok wypukłych są zaznaczone kolorem czerwonym.

Jako konwencję przyjmuje się, że algorytm Grahama powinien zwrócić jako wynik pewną kolekcję zawierającą indeksy punktów, które biorą udział w powłoce wypukłej, w porządku lewoskrętnym (przeciwnym do ruchu wskazówek zegara), poczynając od punktu leżącego najniżej (punkt wyróżniony na rysunku kolorem zielonym). W przypadku remisu, tj. gdy istnieje kilka punktów najniższych, należy jako punkt początkowy wybrać skrajny lewy spośród najniższych.

Ważnym narzędziem geometrycznym w ramach algorytmu Grahama jest *iloczyn wektorowy*. Dla dowolnej pary punktów $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, traktowanych jako wektory zaczepione w środku układu współrzędnych, iloczyn wektorowy pozwala rozstrzygnąć czy P_1 znajduje się w sensie kątowym „na lewo" względem P_2 (ściślej mówiąc: przeciwnie do ruchu wskazówek zegara) czy też „na prawo" (zgodnie z ruchem wskazówek zegara) przy pokonywaniu drogi kątowej nie większej niż 180°. Tego typu elementarne sprawdzenie można sprowadzić do obliczenia wyrażenia

$$y_1 x_2 - y_2 x_1 \tag{1}$$

i sprawdzenia jego znaku. Wyrażenie to odpowiada *skierowanej* długości iloczynu wektorowego tzn.: $\pm\|P_1\|\|P_2\| \sin \angle(P_1, P_2)$. Istotnym jest fakt, że w powyższym wyrażeniu unikamy dzielenia oraz użycia funkcji trygonometrycznych, co spowalniałoby obliczenia.

Na wysokim poziomie (z zaniedbaniem szczegółów) algorytm Grahama można wypowiedzieć następująco:

1. Znajdź punkt poczatkowy (skrajny lewy spośród najniższych) i potraktuj go jako nowy środek układu współrzędnych.
2. Posortuj pozostałe punkty rosnąco według ich kąta względem wektora $(1, 0)$.

[1] inne nazwy: otoczka wypukła, otoczenie wypukłe

3. Do powłoki wypukłej dodaj indeks punktu początkowego (nowy środek układu współrzędnych) oraz pierwszego punktu z posortowanego porządku.

4. Przebiegaj pozostałe punkty w posortowanym porzadku (poczynając od drugiego):

   4.1 Dodaj indeks aktualnego punktu do powłoki wypukłej.

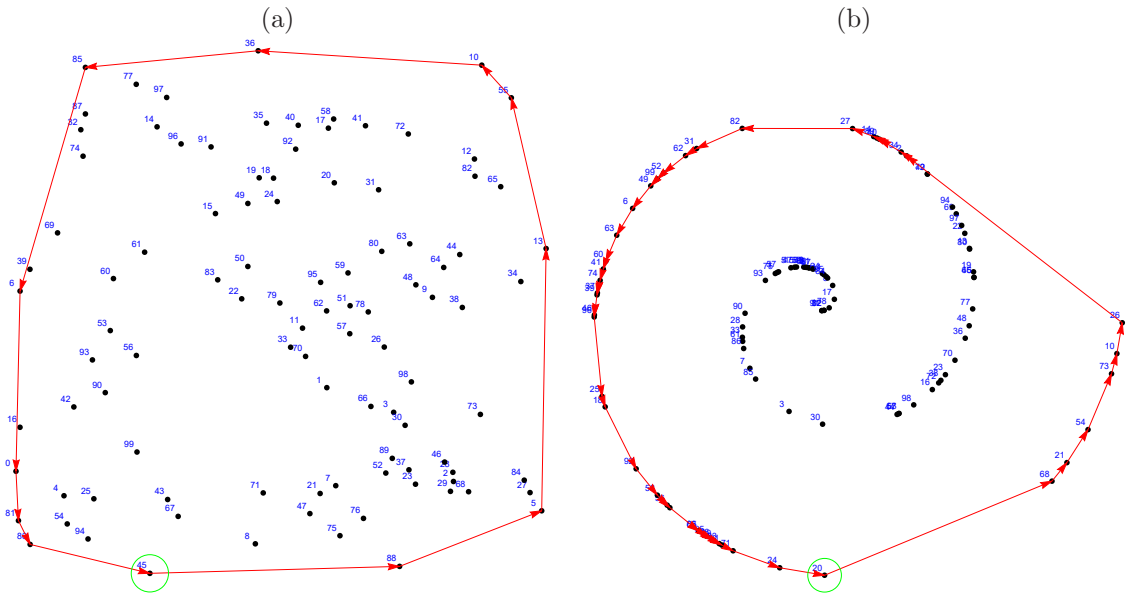
   4.2 Dopóki dwa wektory wyznaczone przez trzy ostatnie punkty leżące na aktualnej powłoce wypukłej tworzą „zakręt w prawo” (tzn. wektor ostatni „skręca w prawo” względem wektora przedostatniego):

      4.2.1 Usuń indeks przedostatniego punktu z aktualnej powłoki wypukłej.

5. Zwróć powłokę wypukłą.

Zarówno w kroku 2 (sortowanie punktów) jak i kroku 4.2 (sprawdzenie orientacji „zakrętu”) należy wykorzystać iloczyn wektorowy.

Krok pierwszy algorytmu to poszukiwanie pewnego minimum, i tym samym złożoność obliczeniowa rzędu $\Theta(n)$, gdzie n to liczba punktów podanych na wejście algorytmu. Krok drugi sortujący punkty wg kąta ma złożoność obliczeniową $\Theta(n \log n)$. Główna pętla (krok 4) może pozornie wskazywać na złożonść $\Theta(n^2)$, ze względu na zagnieżdżenie w niej dodatkowej pętli 4.2. Dokładna analiza pokazuje jednak, że efektywny koszt głównej pętli jest tylko liniowy — $\Theta(n)$, ze względu na usuwanie punktów w kroku 4.2.1. Tym samym złożoność obliczeniowa całego algorytmu Grahama wynika z kroku najwolniejszego — sortowania — i jest rzędu $\Theta(n \log n)$.



Rysunek 1: Przykładowe zbiory punktów na płaszczyźnie i ich powłoki wypukłe (kolor czerwony, porządek lewoskrętny wg strzałek). Punkt początkowy wyróżniony zielonym okręgiem.

2 Instrukcje, wskazówki, podpowiedzi

1. W implementacji należy przewidzieć typ pomocniczy (strukturę lub klasę) reprezentujący **punkt na płaszczyźnie** o współrzędnych (x, y).

2. Wspomniany **iloczyn wektorowy** może zostać zaimplementowany w formie **komparatora** pary punktów, czyli funkcji zwracającej `bool` lub `int`, na którą wskaźnik będzie używany dalej podczas sortowania punktów lub wykrywania orientacji zakrętów. Alternatywnie zamiast komparatora, można w ramach typu punkt przeciążyć jeden z operatorów porównania np.: `>`. Wówczas zapis `p1 > p2` powinien zwrócić wartość `true`, jeżeli punkt `p1` leży „na lewo" względem `p2` (tj. poruszając się przeciwnie do ruchu wskazówek zegara i pokonując kąt mniejszy niż 180°), a `false` w przeciwnym razie.

3. Można wykorzystać pewne elementy wykonane w poprzednich zadaniach: sortowanie przez kopcowanie oraz własne kolekcje np. listę lub tablicę dynamiczną (poprzez dołączenie odpowiednich plików nagłówkowych `.h`). Kolekcja może przydać się do zwrócenia wyniku algorytmu Grahama: indeksy punktów, które należą do powłoki wypukłej (jako podzbiór indeksów wszystkich punktów w podanej na wejście kolekcji).

4. Właściwa funkcja realizująca **algorytm Grahama** może mieć zatem następujący **nagłówek**:
   `linked_list<int> graham_scan(point** points, int n)`
   lub
   `linked_list<int> graham_scan(linked_list<point*> points)`.

5. Funkcja ta powinna pracować na kopii podanych punktów — nie może dopuścić do przesortowania się tych punktów „na zewnątrz" funkcji, czy też do translacji punktów w związku z przesunięciem środka układu współrzędnych.

6. Należy zadbać o prawidłową obsługę sytuacji szczególnych (np.: małe tablice wejściowe, punkty leżące wzdłuż jednej linii).

3 Eksperymenty

Wewnątrz spakowanego pliku `pklesk_graham_scan.zip`, który towarzyszy tej instrukcji, znajduje się pięć plików tekstowych: `points1.txt`, ..., `points5.txt`, zawierających wejściowe zbiory punktów, dla których chcemy znaleźć powłoki wypukłe. Format: n — liczba punktów (wiersz pierwszy), współrzędne (x, y) punktów oddzielone spacją (kolejne n wierszy). Należy przygotować funkcję, która potrafi odczytać plik tekstowy w tym formacie i zamienić go na kolekcję punktów (listę lub tablicę).

Dla każdego z pięciu problemów należy **wykonać algorytm Grahama raportując** na ekran:

- liczbę punktów biorących udział w znalezionej powłoce wypukłej,
- indeksy tych punktów,
- łączny czas obliczeń (wraz z rozdziałem na czas sortowania i czas głównej pętli).

4 Sprawdzenie antyplagiatowe — przygotowanie wiadomości e-mail do wysłania

1. Kod źródłowy programu po sprawdzeniu przez prowadzącego zajęcia laboratoryjne musi zostać przesłany na adres *algo2@zut.edu.pl*.

2. Plik z kodem źródłowym musi mieć nazwę wg schematu: `nr_albumu.algo2.nr_lab.main.c` (plik może mieć rozszerzenie `.c` lub `.cpp`). Przykład: `123456.algo2.lab06.main.c` (szóste zadanie laboratoryjne studenta o numerze albumu 123456). Jeżeli kod źródłowy programu składa się z wielu plików, to należy stworzyć jeden plik, umieszczając w nim kody wszystkich plików składowych.

3. Plik musi zostać wysłany z poczty ZUT (zut.edu.pl).

4. Temat maila musi mieć postać: `ALGO2 IS1 XXXY LAB06`, gdzie `XXXY` to numer grupy (np. `ALGO2 IS1 210C LAB06`).

5. W pierwszych trzech liniach pliku z kodem źródłowym w komentarzach muszą znaleźć się:

   - informacja identyczna z zamieszczoną w temacie maila (linia 1),
   - imię i nazwisko autora (linia 2),
   - adres e-mail (linia 3).

6. Mail nie może zawierać żadnej treści (tylko załącznik).

7. W razie wykrycia plagiatu, wszytkie uwikłane osoby otrzymają za dane zadanie ocenę 0 punktów (co jest gorsze niż ocena 2 w skali $\{2, 3, 3.5, 4, 4.5, 5\}$).