Opis projektu – Algorytmy Geometryczne

Mateusz Kocot

1. Cel projektu

Celem projektu było znalezienie i zaimplementowanie odpowiednich algorytmów rozwiązujących następujący problem:

Mając daną chmurę punktów na płaszczyźnie dwuwymiarowej, wyznaczyć:

- minimalny okrąg zawierający tę chmurę,
- prostokąt o minimalnym polu zawierający tę chmurę,
- prostokąt o minimalnym obwodzie zawierający tę chmurę.

2. Wstęp

Paczka zawiera oprócz tego pliku:

- *projekt.ipynb* – główny plik projektu zawierający cały kod. Umieszczono w nim aplikację służącą do wizualizacji, funkcje generujące zbiory, algorytmy rozwiązujące zadany problem oraz procedury umożliwiające wizualizację poszczególnych zagadnień. Plik ten zawiera także liczne komentarze ułatwiające zrozumienie programu. Należy go otworzyć przez Jupyter Notebook.
- *sets.json* – zbiór zbiorów punktów przydatny w trakcie wizualizacji.
- *prezentacja.pdf* – prezentacja służąca do objaśnienia projektu.

W dalszej części dokumentu zostanie opisany plik z kodem: *projekt.ipynb*.

3. Aplikacja graficzna

W celu wizualizacji działania algorytmów, wykorzystano aplikację graficzną[1] pozwalającą wyświetlanie punktów i odcinków. Istnieje także możliwość wprowadzania punktów.

4. Zbiory danych

W celu sprawdzenia działania późniejszych algorytmów, napisano funkcje generujące zbiory odpowiednio według następnych kryteriów:

1) punkty generowane losowo w podanym zakresie,
2) punkty generowane losowo na zadanym okręgu,
3) punkty generowane losowo na bokach zadanego prostokąta,
4) punkty generowane losowo na prostej przechodzącej przez dane punkty A i B.

[1] Źródło: https://github.com/Podsiadlo

Z wykorzystaniem aplikacji graficznej można także wprowadzić interaktywnie zbiór punktów. Po wylosowaniu i ew. wprowadzeniu zbiorów punktów, istnieje możliwość zapisania ich do pliku w formacie JSON. Można także odczytać zapisane wcześniej w formacie JSON pliki ze zbiorami.

5. Algorytmy znajdujące minimalny okrąg

Problem znajdowania najmniejszego okręgu, tj. okręgu o najmniejszym promieniu, zawierającego wszystkie punkty z zadanego zbioru jest dość powszechny i można łatwo podać przykłady z życia codziennego, gdzie problem ten jest rozważany. Można rozważyć budowę szpitalu (środek okręgu) z ograniczonym zasięgiem. Istotnym jest, by jak najwięcej domów (punkty ze zbioru) znalazło się w zasięgu (w kole). Z drugiej strony, można także znaleźć optymalne miejsce zrzucenia bomby, tak by zniszczenia były jak największe.

Istnieje wiele algorytmów rozwiązujących problem minimalnego okręgu. Znany jest algorytm o złożoności $O(n)$ – algorytm Megidda. Jednakże stała kryjąca się za tą złożonością jest bardzo duża. W tym projekcie zdecydowałem się na przedstawienie trzech algorytmów:

- algorytm Brute Force o złożoności czasowej $O(n^4)$,
- algorytm Skyuma[2] o oczekiwanej złożoności czasowej $O(n \log n)$,
- algorytm Welzla[3] o oczekiwanej złożoności oczekiwanej $O(n)$.

W mojej implementacji wszystkie algorytmy wykorzystują funkcję $build\_circle$, która zwraca okrąg opisany na maksymalnie trzech punktach:

- Dla 0 punktów, funkcja zwraca $None$.
- Dla 1 punktu, funkcja zwraca okrąg o środku w tym punkcie i o promieniu $r = 0$.
- Dla 2 punktów, funkcja zwraca okrąg o środku w odcinku łączącym te dwa punkty i o promieniu równym połowie długości tego odcinka.
- Dla 3 punktów, jeżeli tworzą trójkąt rozwartokątny, rozpatrujemy tylko punkty łączące najdłuższy bok (patrz poprzedni przypadek); w przeciwnym przypadku, funkcja zwraca okrąg opisany na trójkącie utworzonym z tych trzech punktów.

5.1 Algorytm Brute Force

Najprostszy algorytm o najgorszej złożoności czasowej: $O(n^4)$. Algorytm polega na budowaniu okręgu opisanego (*build_circle*) na każdych trzech punktach z zadanego zbioru i spośród tych okręgów – znalezienie najmniejszego zawierającego wszystkie punkty. Algorytm nie ma praktycznego zastosowania, gdyż już dla 200 punktów wykonuje się w niecałe 10 sek., a dla okręgu – prawie minutę.

5.2 Algorytm Skyuma

Algorytm przy odpowiedniej implementacji i z wykorzystaniem odpowiednich struktur danych ma złożoność $O(n \log n)$. W moim przypadku, pozwoliłem sobie na zaimplementowanie algorytmu na zwykłych listach, co w przypadku, gdy rozmiar otoczki wypukłej danego zbioru jest $\Theta(n)$ sprawia, że złożoność całego algorytmu ukwadratawia się. Jednakże w znacznej większości przypadków, gdy rozmiar otoczki wypukłej jest $o(n)$, algorytm działa zadowalającym czasie $O(n \log n)$. Algorytm wykorzystuje algorytm Grahama do znajdowania otoczki wypukłej (funkcja *convex_hull* - $O(n \log n)$). Algorytm wykorzystuje także funkcję *radius(A, B, C)*, która zwraca promień okręgu opisanego na

[2] Źródło: https://tidsskrift.dk/daimipb/article/view/6704
[3] Źródło: https://en.wikipedia.org/wiki/Smallest-circle_problem

trójkącie ABC oraz funkcję *angle(A, B, C, R)*, która zwraca kąt między odcinkami AB i BC, a oprócz odp. punktów przyjmuje także promień okręgu opisanego na ABC - R. Pseudokod algorytmu Skyuma został przedstawiony poniżej.

```
Skyum(set):
   S = convex_hull(set)
   if |S| ≠ 1:
      while True:
         znajdź p w S maksymalizując (radius(before(p), p, next(p)),
         angle(before(p), p, (next(p))) w kolejności leksykograficznej;
         if angle(before(p), p, (next(p))≤ π/2:
            return build_circle(before(p), p, (next(p))
         else:
            usuń p z S
```

W programie oprócz samej implementacji przedstawiono także z wykorzystaniem aplikacji graficznej, jak działa algorytm Skyuma.

5.3 Algorytm Welzla

Algorytm ma złożoność oczekiwaną $O(n)$, co zostało osiągnięte dzięki randomizacji. W programie znajduje się implementacja i wizualizacja działania, lecz ze względu na liczne wywołania rekurencyjne, dla odpowiednio dużego zbioru ($n > 2900$) maksymalna głębokość rekursji zostanie przekroczona i algorytm przestaje działać. Dla mniejszych zbiorów jednak, algorytm działa bez zastrzeżeń, choć ze względu na randomizację i niewystarczającą długość działania, nie mogłem potwierdzić złożoności oczekiwanej $O(n)$. Pseudokod algorytmu Welzla został przedstawiony poniżej:

```
Welzl(P, R = []):
   if |P| = 0 lub |R| = 3:
      return build_circle(R)
   wybierz losowo p z P
   D = Welzl(P – {p}, R}
   if p jest w środku D:
      return D
   else:
      return Welzl(P – {p}, R ∪ {p})
```

Mimo dość prostej implementacji, algorytm jest trudny do zrozumienia i mimo swojej złożoności, dla zbiorów $n < 2900$ działał wolniej od algorytmu Skyuma (za wyjątkiem zbioru generowanego na okręgu).

5.4 Podsumowanie algorytmów znajdujących minimalny okrąg

Algorytm Brute Force można pominąć milczeniem. Mimo, że algorytm Welzla ma lepszą złożoność, to działa wolniej od algorytmu Skyuma, a oprócz tego, nie działa dla liczby punktów $n > 2900$. Oczywiście, dotyczy to mojej implementacji tego algorytmu. Algorytm Welzla można zaimplementować bez wykorzystania rekurencji.

6. Algorytm znajdujący minimalne prostokąty

Podobnie jak problem najmniejszego okręgu, znalezienie minimalnego prostokąta zawierającego zbiór punktów także może być przydatne. Przykładem może być chęć ogrodzenia osiedla ogrodzeniem w kształcie prostokąta (patrząc od góry). Najpowszechniejszym pomysłem jest tutaj wykorzystanie podejścia rotating calipers[4], o czym piszę poniżej. Problem minimalnego prostokąta można rozgraniczyć na:

- prostokąt o minimalnym polu,
- prostokąt o minimalnym obwodzie.

Algorytm w moim programie znajduje oba, a oprócz tego pozwala na wizualizacje działania.

6.1 Algorytm według podejścia rotating calipers

Jeżeli zbiór, na którym chcemy zastosować algorytm, tworzy wielokąt wypukły, złożoność algorytmu jest $O(n)$. Jednakże, gdy trzeba jeszcze utworzyć otoczkę wypukłą, złożoność, w przypadku użycia algorytmu Grahama, wzrasta do $O(n \log n)$. Algorytm wykorzystuje następującą własność:

Jeden z boków minimalnego prostokąta co najmniej częściowo pokrywa się z jednym z boków wielokąta skonstruowanego na podstawie otoczki wypukłej danego zbioru.

Własność ta pozwala na znalezienie w prosty sposób minimalnego prostokąta. Poniżej przedstawiono pseudokod dla algorytmu znajdującego prostokąt o minimalnym polu albo minimalnym promieniu.

```
Min_bounding_rectangle(set):
   S = convex_hull(set)
   for każdy odcinek AB z S:
      znajdź taki prostokąt P1P2P3P4, że jeden z boków prostokąta co
      najmniej częściowo pokrywa się z AB, zawiera wszystkie punkty z S i
      jest możliwie najmniejszy (pokrywa się z A, B i jeszcze co najmniej 3
      punktami z S)
   Z powyższych prostokątów zwróć taki o najmniejszym polu/promieniu.
```

Pseudokod może wydawać się skomplikowany, natomiast chodzi po prostu o poruszanie się po otoczce wypukłej przeciwnie do ruchu wskazówek zegara i budowania prostokąta dla każdego boku wielokąta zbudowanego na otoczce wypukłej.

6.2 Czas działania

Ze względu na „poruszanie się" po otoczce wypukłej, algorytm znacznie wolniej wykonuje się na zbiorach generowanych na okręgach. Dla właśnie tych zbiorów, już dla 4000 punktów, algorytm wykonuje się przez $10\ sek$. Dla pozostałych zbiorów, czasy są zdecydowanie mniejsze (ok. $0{,}5\ sek$. Dla 32000 punktów).

[4] Źródło: https://en.wikipedia.org/wiki/Rotating_calipers