Algorytmy wyznaczania otoczki wypukłej

Dokumentacja techniczna projektu

Adam Dyda
Norbert Wolniak
Grudzień 2020

- # Wprowadzenie.

  W ramach projektu zaimplementowaliśmy algorytmy do wyznaczania otoczki wypukłej w przestrzeni dwuwymiarowej. Algorytmy : Przyrostowy , Górna i dolna otoczka, Grahama, Jarvisa, Dziel i rządź, QuickHull i Chana.

- # Informacje techniczne.

  Projekt został napisany w oprogramowania Jupyter Notebook przy użyciu proponowanego narzędzia graficznego, które napisane jest w języku Python3 i wykorzystuje bibliotekę MatPlotLib. Poszczególne są implementowane poprzez odpowiadające im funkcje do każdego algorytmu funkcje w dwóch wersjach, zwracającej otoczkę oraz zwracającej wykres i kolejne kroki wykonania algorytmu.

- # Funkcje pomocnicze.

  - collection_to_points(collection)
    - Funkcja wykorzystywana do zamiany punktów zawierających się w obiekcie klasy PointsCollection na listę tych punktów
    - Argumenty:
      - collection - kolekcja punktów, klasy PointsCollection
    - Wartość zwracana:
      - lista punktów

  - det(p,q,r)
    - Wyznacznik 3 x 3
    - Argumenty:
      -p- pierwszy punkt w przestrzeni dwuwymiarowej
      -q- drugi punkt w przestrzeni dwuwymiarowej
      -r- trzeci punkt w przestrzeni dwuwymiarowej
    - Wartość zwracana:
      Funkcja zwraca wartość wyznacznika

- orientation(p, q, r, e)
  - Orientacja punktu względem wektora <p, q>
  - Argumenty:
    -p- pierwszy punkt w przestrzeni dwuwymiarowej
    -q- drugi punkt w przestrzeni dwuwymiarowej
    -r- trzeci punkt w przestrzeni dwuwymiarowej
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca wartość -1 gdy r jest cw względem wektora <p,q>,
    wartość 1 gdy r jest ccw względem wektora <p,q>
    i wartość 0 gdy r jest współliniowy względem wektora <p,q>

- alpha(p, q, r, e)
  - Funkcja pomocnicza do wyznaczania kolejności punktów potrzebnych do sortowania ccw
  - Argumenty:
    -p- pierwszy punkt w przestrzeni dwuwymiarowej
    -q- drugi punkt w przestrzeni dwuwymiarowej
    -r- trzeci punkt w przestrzeni dwuwymiarowej
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca wartości -1, 0, 1 zależnie od orientacji punktu r

- sorted_ccw(convex_hull, point, e)
  - Funkcja sortująca punkty należące do otoczki w kolejności ccw
  - Argumenty:
    -convex_hull- otoczka wypukła
    -point- punkt w przestrzeni dwuwymiarowej względem którego następuje sortowanie
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca listę posortowanych punktów

- distance(p,q)
  - Funkcja wyznacza kwadrat dystansu między dwoma punktami (służy do porównywania odległości)
  - Argumenty:
    -p- pierwszy punkt w przestrzeni dwuwymiarowej
    -q- drugi punkt w przestrzeni dwuwymiarowej
  - Wartość zwracana:
    - kwadrat odległości między punktami

- Funkcje pomocnicze: Algorytm Przyrostowy
  - lies_inside(convex_hull, point, e)
    - Funkcja sprawdzająca czy punkt leży wewnątrz zbioru
    - Argumenty:
      -convex_hull- otoczka wypukła w kolejności punktów ccw
      -point- punkt w przestrzeni dwuwymiarowej
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca wartość logiczną True/False
  - compute_tangent_points(convex_hull, point, e)
    - Funkcja obliczająca indeksy punktów leżących na dwóch stycznych punktu do zbioru
    - Argumenty:
      -convex_hull- otoczka wypukła w kolejności ccw
      -point- punkt w przestrzeni dwuwymiarowej względem którego obliczane są styczne
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca dwuelementową listę indeksów

- Funkcje pomocnicze: Algorytm Grahama
  - delete_collinear(S,p,e)
    - Funkcja służy do usuwania punktów współliniowych wzgledem danego punkty
    - Argumenty:
      -S- zbiór punktów do sprawdzenia
      -q- punkt względem którego usuwamy punkty
    - Wartość zwracana:
      - zbiór S po usunięciu punktów

- Funkcje pomocnicze: QuickHull
  - distance_to_line(p,A,B)
    - Funkcja służy do wyznaczania odległości między punktem a odcinkiem
    - Argumenty:

-p - punkt którego odległość od prostej mierzymy
-A - punkt będący początkiem odcinka
-B - punkt będący końcem odcinka
- Wartość zwracana:
- odległość punktu p od odcinka AB

• Funkcje pomocnicze: Algorytm Chana

- min_angle(a,b,c)
  - Funkcja służy do sprawdzania który punkt po połączeniu z danym punktem tworzy mniejszy kąt
  - Argumenty:
    -a - punkt względem którego wyznaczamy kąt
    -b - pierwszy punkt testowy
    -B - drugi punkt testowy
  - Wartość zwracana:
    - jeden z punktów wejściowych b lub c
- split(P,m)
  - Funkcja służy do podziału zbioru punktów na zbiory o określonej ilości punktów
  - Argumenty:
    -P -zbiór punktów do podzielenie
    -m - ilość punktów w każdym wynikowym zbiorze
  - Wartość zwracana:
    - Lista zbiorów punktów
- jarvis_next_point(p,S):
  - Funkcja służy wyznaczenia punktu z danego zbioru tworzącego z danym punktem największy kąt
  - Argumenty:
    -p - punkt względem którego szukamy punktu
    -S - zbiór w którym szukamy punktu
  - Wartość zwracana:
    - punkt który tworzy z punktem wejściowym największy kąt

• Algorytm Przyrostowy

- incremental_algorithm(S, e)
  - Funkcja implementująca algorytm przyrostowy
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca listę punktów należących do otoczki

- # Algorytm Górna i dolna otoczka
  - upper_lower_algorithm(S)
    - Funkcja implementująca algorytm górnej i dolnej otoczki
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do otoczki

- # Algorytm Grahama
  - graham_algorithm(S,e):
    - Funkcja implementująca algorytm Grahama
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do otoczki

- # Algorytm Jarvisa
  - jarvis_algorithm(S, e)
    - Funkcja implementująca algorytm Jarvisa
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do otoczki
- # Algorytm Dziel i rządź
  - divide_and_conquer_algorithm(S,k):
    - Funkcja inicjalizująca algorytm Dziel i rządź
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
      -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do otoczki

  - divide_and_conquer(points, k):
    - Funkcja dzieląca zbiór na kolejne zbiory A i B
    - Argumenty:
      -points- zbiór punktów do podziału

-k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
- Wartość zwracana:
  Funkcja zwraca wartość zwracaną funkcji merge_convex_hulls(A,B)

- merge_convex_hulls(A,B):
  - Funkcja łącząca dwie otoczki wypukłe A i B
  - Argumenty:
    -A- zbiór punktów otoczki leżących po lewej stronie
    -B- zbiór punktów otoczki leżących po prawej stronie
  - Wartość zwracana:
    Funkcja zwraca listę punktów należących do wyznaczonej tymczasowej otoczki wypukłej

• QuickHull

- quickhull(S)
  - funkcja inicjalizująca algorytm quickhull do wyznaczania otoczki wypukłej
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej
- quickhull_step(A,B,S)
  - funkcja wyznaczające kolejne punkty otoczki wykonywana rekurencyjnie
  - Argumenty:
    -A - jeden ze skrajnych punktów w rozpatrywanym zbiorze
    -B - drugi ze skrajnych punktów w rozpatrywanych zbiorze
    -S - rozpatrywany zbiór
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej w rozpatrywanym zbiorze

• Algorytm Chana

- chan_algorithm(S)
  - funkcja zwracająca otoczke wypukła za pomocą algorytmu Chana
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej

• Algorytm Przyrostowy Wizualizacja

- plot_incremental_algorithm(S, e):
  - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu

        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
          -e- tolerancja dla zera
        - Wartość zwracana:
          -Obiekt klasy Plot() służący do wizualizacji

- Algorytm Górna i dolna otoczka Wizualizacja
    - plot_upper_lower_algorithm(S):
        - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
        - Wartość zwracana:
          -Obiekt klasy Plot() służący do wizualizacji

- Algorytm Grahama Wizualizacja
    - plot_graham_algorithm(S)
        - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
        - Wartość zwracana:
          -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot

- Algorytm Jarvisa Wizualizacja
    - plot_jarvis_algorithm(S)
        - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
        - Wartość zwracana:
          -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot
- Algorytm Dziel i rządź Wizualizcja
    - plot_divide_and_conquer_algorithm(S,k):

        - Funkcja inicjalizująca algorytm Dziel i rządź do rysowania otoczki wypukłej
        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
          -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
        - Wartość zwracana:
          -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot

    - plot_divide_and_conquer(points, k, plot, S):
        - Funkcja dzieląca zbiór na kolejne zbiory A i B
        - Argumenty:
          -points- zbiór punktów do podziału
          -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
          -plot- obiekt klasy Plot()
          -S- zbiór początkowy punktów
        - Wartość zwracana:
          Funkcja zwraca wartość zwracaną funkcji plot_merge_convex_hulls(A,B)

    - plot_merge_convex_hulls(A,B, plot, S):
        - Funkcja łącząca dwie otoczki wypukłe A i B
        - Argumenty:
          -A- zbiór punktów otoczki leżących po lewej stronie
          -B- zbiór punktów otoczki leżących po prawej stronie
          -plot- obiekt klasy Plot()
          -S- zbiór początkowy punktów
        - Wartość zwracana:
          Funkcja zwraca listę punktów należących do wyznaczonej tymczasowej otoczki wypukłej

- QuickHull Wizualizacja
    - plot_quickhull(S)
        - funkcja inicjalizująca algorytm quickhull do rysowania otoczki wypukłej
        - Argumenty:
          -S - zbiór punktów na których wyznaczamy otoczkę
        - Wartość zwracana:
          -rysunek zboru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot
    - plot_quickhull_step(points,A,B,S,plot)
        - funkcja służąca do rysowania otoczki wypukłej wyznaczonej na danym zbiorze, wykonywana rekurencyjnie
        - Argumenty:
          -points- pierwotny zbiór punktów na których wyznaczmy otoczkę

-A - jeden ze skrajnych punktów w rozpatrywanym zbiorze
-B - drugi ze skrajnych punktów w rozpatrywanych zbiorze
-S - rozpatrywany zbiór
-plot - obiekt klasy Plot, rysunek na którym rysujemy wykres otoczki

- Wartość zwracana:
  -Lista punktów należących do otoczki w rozpatrywanym zbiorze

- Algorytm Chana Wizualizacja
  - plot_chan_algorithm(S)
    - funkcja służy do rysowania otoczki wyznaczonej na danym zbiorze punktów za pomocą algorytmu Chana
    - Argumenty:
      -S - zbiór punktów na którym wyznaczamy otoczkę
    - Wartość zwracana:
      -obiekt typu Plot, rysunek zawierający wykres wyznaczonej otoczki
- Porównanie Szybkości
- Bibliografia

  - https://www.youtube.com/watch?v=EzeYI7p9MjU
  - https://en.wikipedia.org/wiki/Chan%27s_algorithm
  - https://pl.wikipedia.org/wiki/Quickhull
  - https://cs.stackexchange.com/questions/97194/is-binary-search-really-required-in-chans-convex-hull-algorithm
  - Mark de Berg - “Computational Geometry - Algorithms and Applications”