LAPORAN TUCIL 2

ALGORITMA DIVIDE AND CONQUER

Muhammad Naufal Satriandana (13520068)

1. Algoritma *Divide and Conquer*

Program ini adalah sebuah modul yang berfungsi untuk mencari dan menampilkan convex hull dari suatu data set yang diberikan dengan menggunakan algoritma divide and conquer. Divide and conquer adalah suatu algoritma yang menyelesaikan permasalahan dengan membagi persoalan tersebut menjadi bagian yang lebih kecil, Menyeselaikan permasalahan yang telah dibagi tersebut, lalu kemudian menyatukannya kembali untuk mendapatkan solusi akhir. Dalam penyelesaian permasalahan convex hull, terdapat suatu data set point yang kemudian dibagi-bagi sesuai letak point tersebut. Langkah-langkah penyelesaian convex hull dalam program ini adalah sebagai berikut:

1. Program menerima suatu tabel yang berbentuk array of points, yang dalam hal ini setiap point dalam array memiliki komponen x dan y
2. Program mencari point paling kiri dan point paling kanan, yakni mencari point yang memiliki komponen x minimum dan point yang memiliki x maksimum, kemudian memasukkan kedua titik tersebut ke himpunan penyelesaian
3. Program membentuk garis dari kedua point tersebut kemudian membagi array menjadi dua, yakni array yang mengandung seluruh point yang berada di atas garis dan array yang mengandung seluruh point yang berada di bawah garis
4. Untuk kedua array tersebut, program mencari point terjauh di array tersebut dari garis yang telah dibentuk, kemudian memasukkan point terjauh ke himpunan penyelesaian
5. Untuk tiap point terjauh yang ditemukan, program akan membuat dua buah garis; garis pertama adalah dari point awal ke point terjauh, garis kedua adalah dari point terjauh ke point akhir
6. Untuk tiap garis tersebut, program akan membagi array menjadi dua lagi, seperti pada langkah 3
7. Jika program sedang mencari ke arah atas, program akan melanjutkan pencarian point terjauh dari tiap garis yang telah dibentuk ke arah atas. Demikian pula sebaliknya untuk arah bawah. Untuk tiap point terjauh yang ditemukan, point tersebut dimasukkan ke himpunan penyelesaian
8. Program akan terus mengulangi langkah secara rekursif hingga himpunan point untuk pencarian titik tejauh sudah kosong dan tidak bisa lagi dilakukan pencarian titik terjauh
9. Setelah proses selesai, akan dibentuk garis dari tiap titik yang ditemukan. Garis yang dibentuk diekspresikan dalam array yang tiap elemennya menunjukkan hubungan antar titik (misal ada garis dari titik 1 ke titik 2, maka di dalam array tersebut ada elemen [1, 2]) dan mengembalikan array ini sebagai solusi akhir
10. Kode program

Program ini merupakan library untuk python. Berikut adalah source code program.

import math

import numpy as np

class myConvexHull:

    points = []

    outermostPointIndex = []

    simplices = []

    def \_\_init\_\_(self, paramPoints):

        self.myConvexHull(paramPoints)

    def myConvexHull(self, paramPoints):

        self.simplices = []

        self.points = paramPoints

        #just a list of indexes

        pointIndex = [i for i in range(len(self.points))]

        #find the leftmost and rightmost point and append them to an array

        self.outermostPointIndex = []

        leftmostidx = 0

        rightmostidx = 0

        for i in range(len(self.points)):

            if(self.points[i][0] <  self.points[leftmostidx][0]):

                leftmostidx = i

            if(self.points[i][0] >  self.points[rightmostidx][0]):

                rightmostidx = i

        self.outermostPointIndex.append(leftmostidx)

        self.outermostPointIndex.append(rightmostidx)

        #split points into two section

        x1 = self.points[leftmostidx][0]

        y1 = self.points[leftmostidx][1]

        x2 = self.points[rightmostidx][0]

        y2 = self.points[rightmostidx][1]

        upperSection = self.GetPointsInSection(pointIndex, x1, y1, x2, y2, "up")

        lowerSection = self.GetPointsInSection(pointIndex, x1, y1, x2, y2, "down")

        #answer is outermost points from left section and right section

        self.FindOuterIndex(upperSection, leftmostidx, rightmostidx, "up")

        self.FindOuterIndex(lowerSection, leftmostidx, rightmostidx, "down")

        #print(self.points[self.outermostPointIndex])

        #print(self.outermostPointIndex)

        self.simplices = np.array(self.simplices)

    def GetPointsInSection(self, pointIndex, x1, y1, x2, y2, section): #returns index of points in desired section

        upperSection = []

        lowerSection = []

        for i in pointIndex:

            x3 = self.points[i][0]

            y3 = self.points[i][1]

            if (x1\*y2 + x3\*y1 + x2\*y3 - x3\*y2 - x2\*y1 - x1\*y3) > 0 and not (x3 == x1 and y3 == y1) and not (x3 == x2 and y3 == y2):

                upperSection.append(i)

            elif (x1\*y2 + x3\*y1 + x2\*y3 - x3\*y2 - x2\*y1 - x1\*y3) < 0 and not (x3 == x1 and y3 == y1) and not (x3 == x2 and y3 == y2):

                lowerSection.append(i)

        if section == "up":

            return upperSection

        else:

            return lowerSection

    def FindOuterIndex(self, pointIndex, idx1, idx2, direction): #adds outermost points

        p1 = self.points[idx1]

        p2 = self.points[idx2]

        if pointIndex != []: #recurrence

            farthestPoint = self.GetFarthestPoint(pointIndex, p1, p2)

            if farthestPoint >=0:

                self.outermostPointIndex.append(farthestPoint)

            x1 = p1[0]

            y1 = p1[1]

            x2 = p2[0]

            y2 = p2[1]

            xMax = self.points[farthestPoint][0]

            yMax = self.points[farthestPoint][1]

            s1 = self.GetPointsInSection(pointIndex, x1, y1, xMax, yMax, direction)

            s2 = self.GetPointsInSection(pointIndex, xMax, yMax, x2, y2, direction)

            self.FindOuterIndex(s1, idx1, farthestPoint, direction)

            self.FindOuterIndex(s2, farthestPoint, idx2, direction)

        else: #base : if pointIndex is empty then append idx1 and idx2

            if [idx1, idx2] not in self.simplices and idx1 != -1 and idx2 != -1:

                self.simplices.append([idx1, idx2])

    def GetFarthestPoint(self, pointIndex, p1, p2): #returns the index of the farthest point in points from line p1p2

        #finding the linear equation that satisfies p1 and p2: Ax + By + C = 0

        #using the formula (y1 – y2)x + (x2 – x1)y + (x1y2 – x2y1) = 0

        A = p1[1] - p2[1]

        B = p2[0] - p1[0]

        C = p1[0]\*p2[1] - p2[0]\*p1[1]

        maximum = 0

        maxidx = -1

        for i in pointIndex:

            distance = abs(A\*self.points[i][0] + B\*self.points[i][1] + C)/math.sqrt(A\*A + B\*B)

            if (distance > maximum):

                maximum = distance

                maxidx = i

        return maxidx

Berikut adalah driver untuk mencoba modul

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

#read input

print("Toy datasets:")

print("1. iris")

print("2. wine")

print("3. breast\_cancer")

num = int(input("input a number: "))

data = datasets.load\_iris()

if num == 2:

    data = datasets.load\_wine()

elif num == 3:

    data = datasets.load\_breast\_cancer()

#create a DataFrame

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['Target'] = pd.DataFrame(data.target)

print(df.shape)

df.head()

#visualisasi hasil ConvexHull

from scipy.spatial import ConvexHull

from myConvexHull import myConvexHull

#sepal

plt.figure(figsize = (10, 6))

colors = ['b','r','g']

plt.title(data.feature\_names[0] + " vs " + data.feature\_names[1])

plt.xlabel(data.feature\_names[0])

plt.ylabel(data.feature\_names[1])

for i in range(len(data.target\_names)):

    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[0,1]].values

    hull = myConvexHull(bucket) #bagian ini diganti dengan hasil implementasiConvexHull Divide & Conquer

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target\_names[i])

    for simplex in hull.simplices:

        plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i])

plt.legend()

plt.show()

#petal

plt.figure(figsize = (10, 6))

colors = ['b','r','g']

plt.title(data.feature\_names[2] + " vs " + data.feature\_names[3])

plt.xlabel(data.feature\_names[0])

plt.ylabel(data.feature\_names[1])

for i in range(len(data.target\_names)):

    bucket = df[df['Target'] == i]

    bucket = bucket.iloc[:,[2,3]].values

    hull = myConvexHull(bucket) #bagian ini diganti dengan hasil implementasiConvexHull Divide & Conquer

    plt.scatter(bucket[:, 0], bucket[:, 1], label=data.target\_names[i])

    for simplex in hull.simplices:

        plt.plot(bucket[simplex, 0], bucket[simplex, 1], colors[i])

plt.legend()

plt.show()

1. Skrinshut Input dan Output

Input yang digunakan merupakan dataset dari library yang sudah di hard code kedalam program.

1. iris

Text

Description automatically generated



Chart, scatter chart

Description automatically generated

Chart

Description automatically generated

1. wine

Text

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

3. breast\_cancer

Text

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, line chart

Description automatically generated

1. Link Google Drive

<https://drive.google.com/drive/folders/1lfPsyvwqMOB79tYfEsbgBDvaFHKkI0gK?usp=sharing>

<https://github.com/FaLzNaufal/Tucil2_13520068>

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Pustaka myConvexHull berhasil dibuat dan tidak ada kesalahan | √ |  |
| 2. Convex hull yang dihasilkan sudah benar | √ |  |
| 3. Pustaka myConvexHull dapat digunakan untuk menampilkan convex hull setiap label dengan warna yang berbeda. | √ |  |
| 4. Bonus: program dapat menerima input dan menuliskan output untuk dataset lainnya. | √ |  |