**Executive summary:**

This program is designed to solve a classification problem in two-dimension coordinates using K-mean clustering algorithm.

Users are allowed to input numbers of points as many as they want, and the numbers of groups they want to classify as well. They could also select the method when calculating distance.

The program uses two lists A1 and A2 to store the x-variable data and y-variable data respectively. C and C0 are two lists to store old and new centroids data respectively. When old centroids and new centroids are not the same, the algorithm is triggered.

Firstly, the program calculates the distance between each point to each centroid and store the data in list D, then classify the points into groups according to the distances. At last, re-calculate the centroid of each group. For the centroid of each group, its x variable equals to the sum of x variables of all the centroid in the group divided by the number of points in the group, and similarly the program calculates the y variable. If the old centroids and the new ones are not the same, the program will repeat the previous steps. If the old centroids and the new ones are the same, the program will print the results.

**Source code:**

import math

#initialization

N=eval(input('numbers of points: '))

K=eval(input('numbers of groups: '))

print('choose the method of calculating distance')

print('1 for manhattan')

print('2 for euclidean')

I=eval(input())

#put x-variable in A1 and y-variable in A2

A1,A2=[],[]

for i in range (0,N):

x=eval(input('input x['+str(i)+']:'))

y=eval(input('input y['+str(i)+']:'))

A1.append(x)

A2.append(y)

#initialize C and C0, which are used to store old and new

#centroid coordinate respectively

C=[[0 for col in range(2)]for row in range(K)]

C0=[[1 for col in range(2)]for row in range(K)]

#chose intial centroids

for i in range (0,K):

C0[i]=[A1[i],A2[i]]

#when old and new coordinate are not equal, go to while loop.

while (C0!=C):

for i in range (0,K):

C[i]=C0[i]

#initialize an K x N matrix to store the distance from each point

#to each centroid.

D=[[0 for col in range(N)]for row in range(K)]

for j in range(0,K):

for i in range (0,N):

if I==2:

D[j][i]=math.sqrt((A1[i]-C[j][0])\*\*2+\

(A2[i]-C[j][1])\*\*2)

else:

D[j][i]=abs(A1[i]-C[j][0])+abs(A2[i]-C[j][1])

#initialize an 1 x N matrix to store the group information,

#ie. G[2]=1 means the second point belongs to group 1.

G=[1]\*N

for j in range (0,N):

m=D[0][j]

for i in range (0,K):

if D[i][j]<m:

m=D[i][j]

G[j]=i+1

#re-calculate the centroid.

Count=[0]\*K

SumX,SumY=[0]\*K,[0]\*K

for n in range (0,N):

for x in range (0,K):

if G[n]==x+1:

Count[x]+=1

SumX[x]+=A1[n]

SumY[x]+=A2[n]

for i in range (0,K):

C0[i]=[SumX[i]/Count[i],SumY[i]/Count[i]]

#print result.

if I==1:

print('this program is using Manhattan distance')

else:

print('this program is using Euclidean distance')

for i in range (0,N):

print('('+str(A1[i])+','+str(A2[i])+') belongs to group'\

+str(G[i]))

**Example 1:**

The first example I choose is from the lecture.

Classify (1,1)、(2,1)、(4,3)、(5,4) into 2 groups.

Result of running the source code:

>>> ================================ RESTART ================================

>>>

numbers of points: 4

numbers of groups: 2

choose the method of calculating distance

1 for manhattan

2 for euclidean

2

input x[0]:1

input y[0]:1

input x[1]:2

input y[1]:1

input x[2]:4

input y[2]:3

input x[3]:5

input y[3]:4

this program is using Euclidean distance

(1,1) belongs to group1

(2,1) belongs to group1

(4,3) belongs to group2

(5,4) belongs to group2

**Example 2:**

Example 2 is from KMeansExample\_(2)[1].docx in the distribution folder.

Cluster the following eight points (with (x, y) representing locations) into three clusters A1(2, 10) A2(2, 5) A3(8, 4) A4(5, 8) A5(7, 5) A6(6, 4) A7(1, 2) A8(4, 9).

The result of running the source code:

>>> ================================ RESTART ================================

>>>

numbers of points: 8

numbers of groups: 3

choose the method of calculating distance

1 for manhattan

2 for euclidean

1

input x[0]:2

input y[0]:10

input x[1]:2

input y[1]:5

input x[2]:8

input y[2]:4

input x[3]:5

input y[3]:8

input x[4]:7

input y[4]:5

input x[5]:6

input y[5]:4

input x[6]:1

input y[6]:2

input x[7]:4

input y[7]:9

this program is using Manhattan distance

(2,10) belongs to group1

(2,5) belongs to group2

(8,4) belongs to group3

(5,8) belongs to group1

(7,5) belongs to group3

(6,4) belongs to group3

(1,2) belongs to group2

(4,9) belongs to group1

**Conclusion:**

Example 1 is a problem of classifying 4 points into two groups using Euclidean distance.

Example 2 is a problem of classifying 8 points into three groups using Manhattan distance.

Some other examples are also tested using my program, and they all correctly handled and solved.

The results of running my code in example 1 and example 2 are agree with the results shown in the lecture and .docx document, which means the program is correct and can handle the problem of classifying N points into K groups (N and K are determined by user) using Euclidean distance method or Manhattan distance method.