Assignment_DBScan

2014004066 서왕규

<development enviroment>

Window 10

Visual studio code

Python 3.8

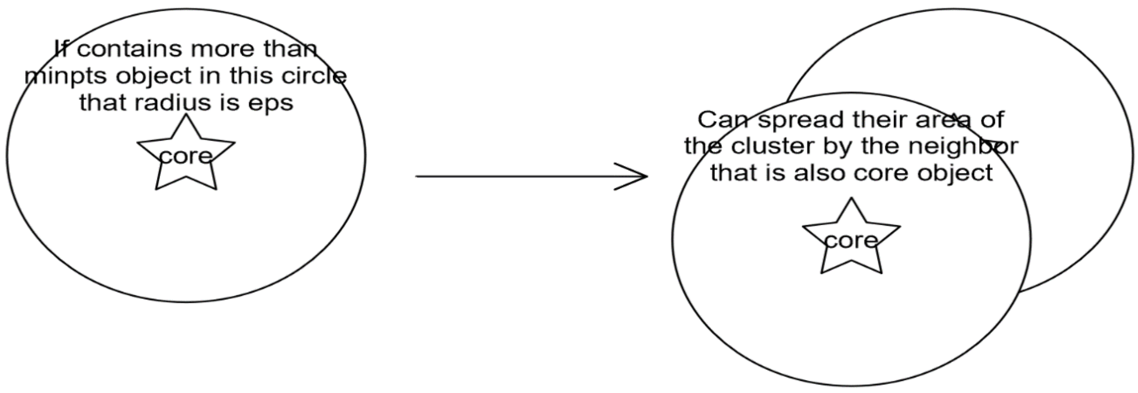
1. Algorithm

<Workload>

1. Object has two numeric attribute : x-coordinate, y-coodinate

2. input file has object id, and it is not always sorted by id order and can jump some id

<Algorithm for DBscan>


If contains more than minpts object in this circle that radius is eps
core
Can spread their area of the cluster by the neighbor that is also core object
core

1. Pick a object that is not visited

2. Check that object visited

3. If it is core point, the object's neighbor will be candidate set

4. If it is core point, it's neighbors are included to candidate set

5. If it is core point, it's neighbors are included to candidate set

6. If candidate is not in any cluster, add to new cluster

7. Repeat this process until, all point visited

By doing this sequence, density connected objects will be in to same cluster.

- Density connected : if p, q are density-connected, there is a object O that is density-reachable to p and q

- Density reachable : if p, q are density-reachable, this two objected connected by directly density-reachable object

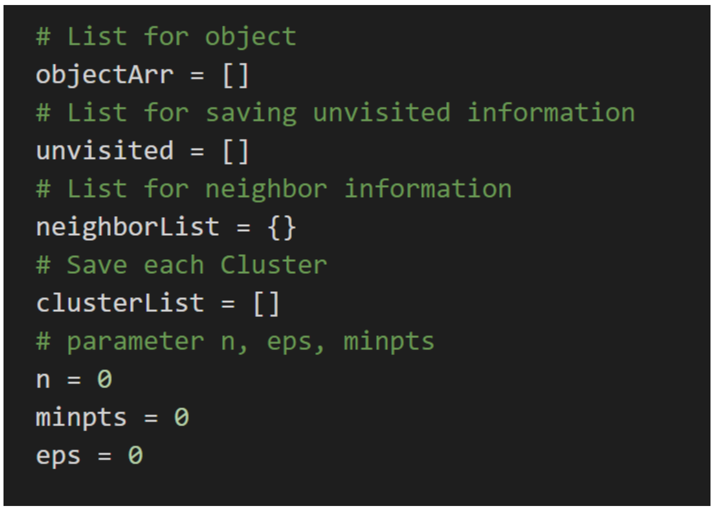
<Find core point?>

- The object be a core object that has neighbor more or same than minpts that I given.

 Neighbor objects define the distance between two objects are less or same than eps that I given

2. detailed description of my codes

[1] global variable



```
# List for object
objectArr = []
# List for saving unvisited information
unvisited = []
# List for neighbor information
neighborList = {}
# Save each Cluster
clusterList = []
# parameter n, eps, minpts
n = 0
minpts = 0
eps = 0
```

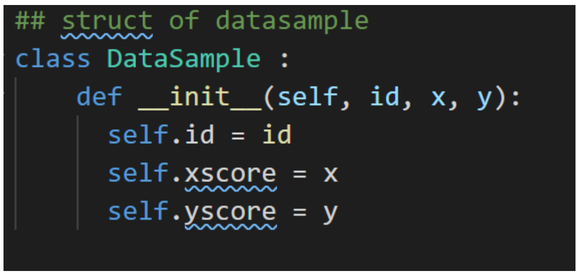
- objectArr is list for object

- unvisited is list for saving unvisited information

- neighborList is list for neighbor information

- n, eps, minpts is parameter given already for using in dbscan

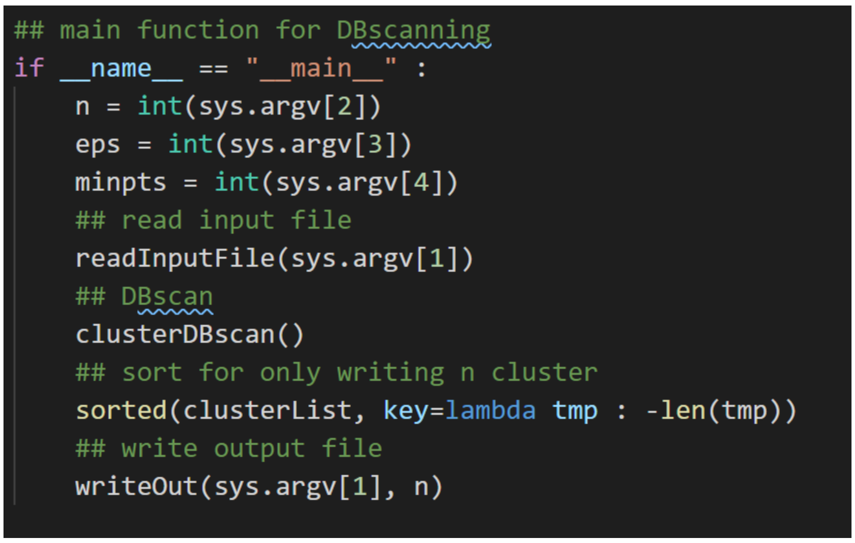
[2] Object class



```
## struct of datasample
class DataSample :
    def __init__(self, id, x, y):
        self.id = id
        self.xscore = x
        self.yscore = y
```

This is class for representing a object

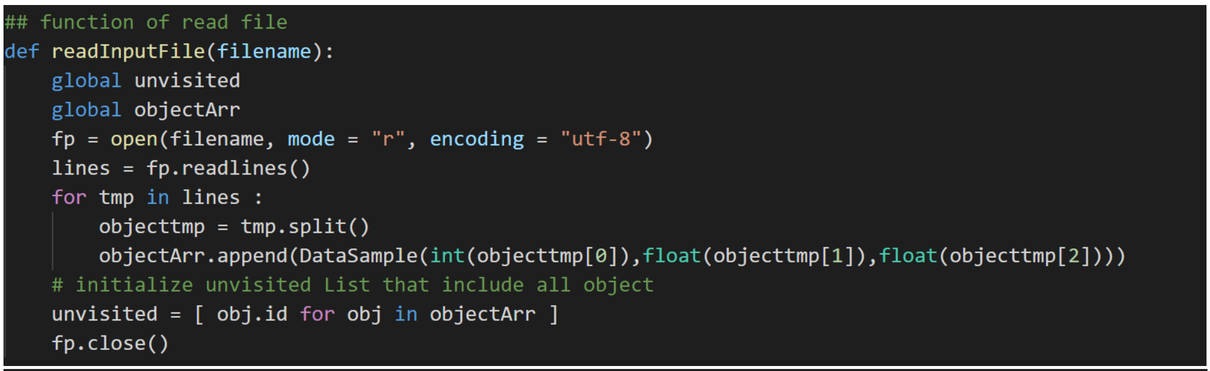
[3] Main function



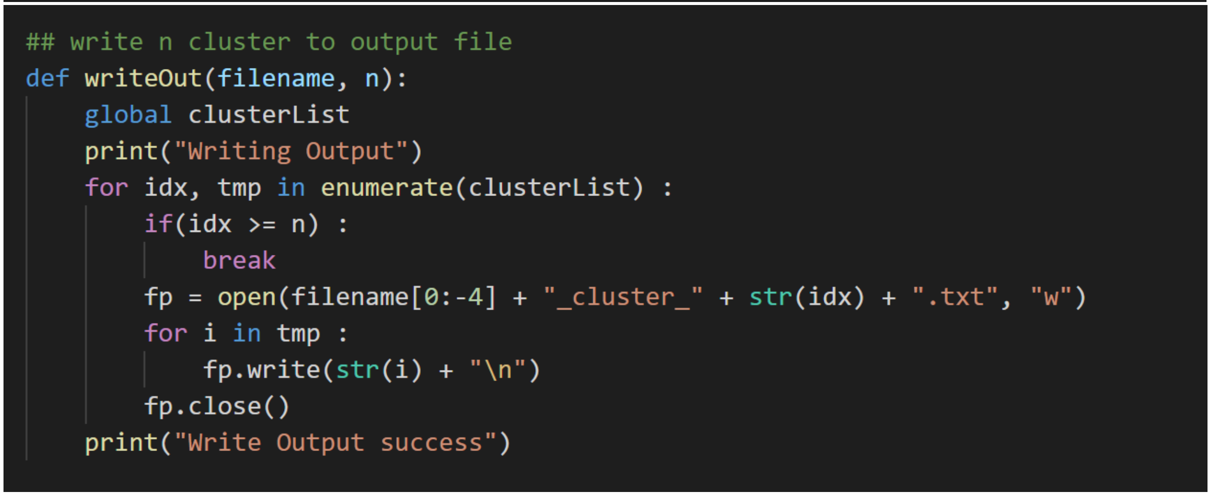
```python
## main function for DBscanning
if __name__ == "__main__" :
    n = int(sys.argv[2])
    eps = int(sys.argv[3])
    minpts = int(sys.argv[4])
    ## read input file
    readInputFile(sys.argv[1])
    ## DBscan
    clusterDBscan()
    ## sort for only writing n cluster
    sorted(clusterList, key=lambda tmp : -len(tmp))
    ## write output file
    writeOut(sys.argv[1], n)
```

Main function initialize argument, and call those functions that are readfile, clustering by dbscan, sorted result of clustering by using cluster length for print only N cluster, write file

[4] Read and Write file



```python
## function of read file
def readInputFile(filename):
    global unvisited
    global objectArr
    fp = open(filename, mode = "r", encoding = "utf-8")
    lines = fp.readlines()
    for tmp in lines :
        objecttmp = tmp.split()
        objectArr.append(DataSample(int(objecttmp[0]),float(objecttmp[1]),float(objecttmp[2])))
    # initialize unvisited List that include all object
    unvisited = [ obj.id for obj in objectArr ]
    fp.close()
```



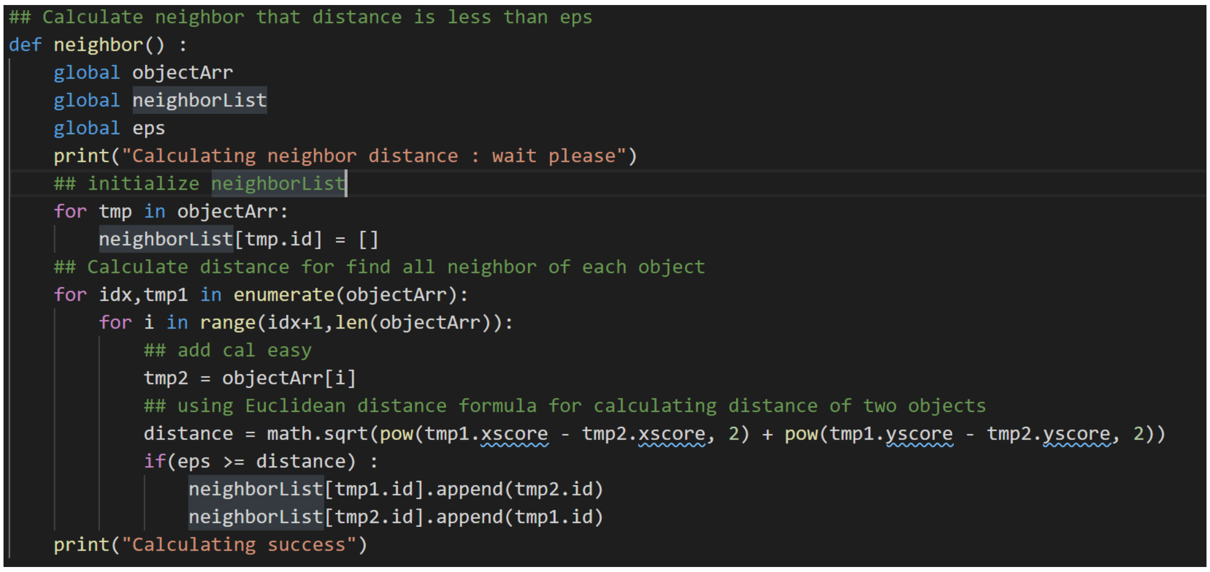
```python
## write n cluster to output file
def writeOut(filename, n):
    global clusterList
    print("Writing Output")
    for idx, tmp in enumerate(clusterList) :
        if(idx >= n) :
            break
        fp = open(filename[0:-4] + "_cluster_" + str(idx) + ".txt", "w")
        for i in tmp :
            fp.write(str(i) + "\n")
        fp.close()
    print("Write Output success")
```

Read and Write file to required format

Output file's name generate by input file's name

Output file's Object is not sorted in a cluster. Because It is not affected to result

[5] Find neighbor

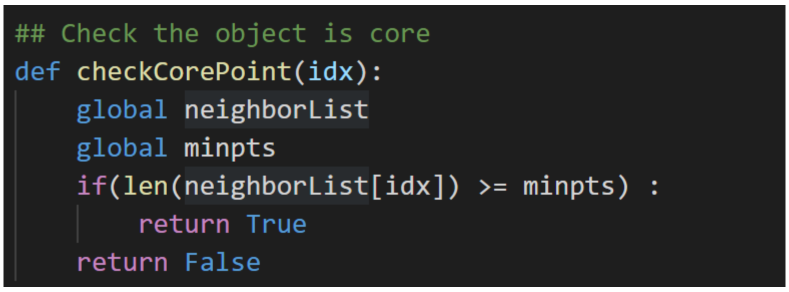


```python
## Calculate neighbor that distance is less than eps
def neighbor() :
    global objectArr
    global neighborList
    global eps
    print("Calculating neighbor distance : wait please")
    ## initialize neighborList
    for tmp in objectArr:
        neighborList[tmp.id] = []
    ## Calculate distance for find all neighbor of each object
    for idx,tmp1 in enumerate(objectArr):
        for i in range(idx+1,len(objectArr)):
            ## add cal easy
            tmp2 = objectArr[i]
            ## using Euclidean distance formula for calculating distance of two objects
            distance = math.sqrt(pow(tmp1.xscore - tmp2.xscore, 2) + pow(tmp1.yscore - tmp2.yscore, 2))
            if(eps >= distance) :
                neighborList[tmp1.id].append(tmp2.id)
                neighborList[tmp2.id].append(tmp1.id)
    print("Calculating success")
```

Find all neighbors of each objects by using Euclidean distance. If distance is less or same than eps, they will be neighbor.

This function needs some time that c * (n^2)/2(c is constant) for calculation if n is large. Especially, This done in python.

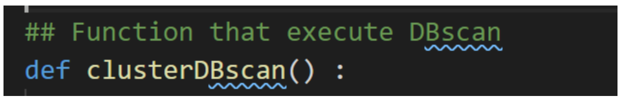
[6] Check core object



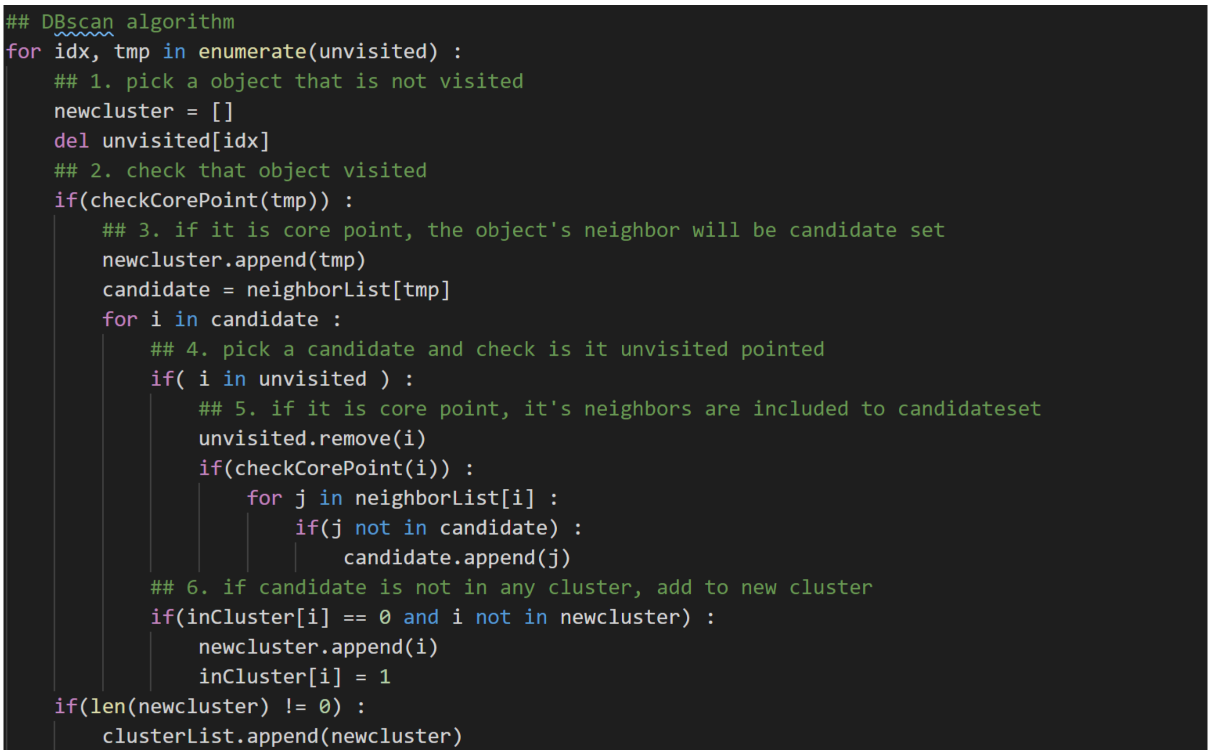
```python
## Check the object is core
def checkCorePoint(idx):
    global neighborList
    global minpts
    if(len(neighborList[idx]) >= minpts) :
        return True
    return False
```

Check the object is core object or not. If that object has neighbors more than minpts that is a parameter already given, It will be core object

[7] Clustering by using DBscan>



```python
## Function that execute DBscan
def clusterDBscan() :
```



```python
## DBscan algorithm
for idx, tmp in enumerate(unvisited) :
    ## 1. pick a object that is not visited
    newcluster = []
    del unvisited[idx]
    ## 2. check that object visited
    if(checkCorePoint(tmp)) :
        ## 3. if it is core point, the object's neighbor will be candidate set
        newcluster.append(tmp)
        candidate = neighborList[tmp]
        for i in candidate :
            ## 4. pick a candidate and check is it unvisited pointed
            if( i in unvisited ) :
                ## 5. if it is core point, it's neighbors are included to candidateset
                unvisited.remove(i)
                if(checkCorePoint(i)) :
                    for j in neighborList[i] :
                        if(j not in candidate) :
                            candidate.append(j)
            ## 6. if candidate is not in any cluster, add to new cluster
            if(inCluster[i] == 0 and i not in newcluster) :
                newcluster.append(i)
                inCluster[i] = 1
    if(len(newcluster) != 0) :
        clusterList.append(newcluster)
```

1. Pick a object that is not visited

2. Check that object visited

3. If it is core point, the object's neighbor will be candidate set

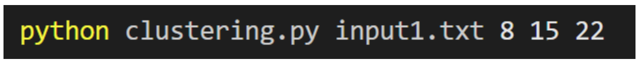
4. If it is core point, it's neighbors are included to candidate set

5. If it is core point, it's neighbors are included to candidate set

6. If candidate is not in any cluster, add to new cluster

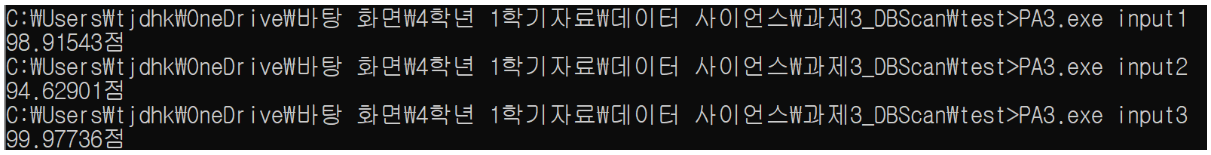
7. Repeat this process until, all point visited

3. Instruction for compiling and result



```
python clustering.py input1.txt 8 15 22
```

Compile instruction syntax> python clustering.py <inputfilename> <n> <eps> <minpts>



```
C:₩Users₩tjdhk₩OneDrive₩바탕 화면₩4학년 1학기자료₩데이터 사이언스₩과제3_DBScan₩test>PA3.exe input1
98.91543점
C:₩Users₩tjdhk₩OneDrive₩바탕 화면₩4학년 1학기자료₩데이터 사이언스₩과제3_DBScan₩test>PA3.exe input2
94.62901점
C:₩Users₩tjdhk₩OneDrive₩바탕 화면₩4학년 1학기자료₩데이터 사이언스₩과제3_DBScan₩test>PA3.exe input3
99.97736점
```

This is result of processing