<Programming Assignment #3>

2017029716 박혜정

1. Environment

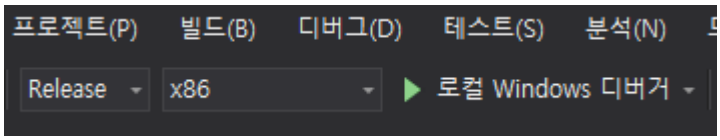
OS : Window

Language : C++

Tool : Visual Studio Community 2019

2. 컴파일 및 실행방법

1) 컴파일 방법



- Visual studio의 도구창에서 Debug -> Release로 변경해주세요.
- Crtl + Shift + B를 눌러 빌드해주세요.
- 솔루션 폴더의 Release 폴더에 clustering.exe가 생성된 걸 확인할 수 있습니다.

2) 실행방법

```
C:₩Users₩HB₩Desktop₩clustering₩Release>clustering.exe input1 8 15 22
```

소스코드는 /clustering/clustering 폴더 안에,

실행파일(clustering.exe)과 데이터셋(input1.txt, input2.txt, input3.txt) 및 평가파일과 평가 프로그램(PA3.exe)는 /clustering/Release폴더 안에 있습니다

따라서 /clustering/Release 폴더에서 cmd창에

[실행파일명] [Input data file name] [n] [Eps] [MinPts]

을 입력하여 프로그램을 실행시킬 수 있습니다.

3. 코드 설명

1) Summary

맨처음 파일을 읽어 데이터들을 Object 객체로 만들어 배열에 저장합니다. 이 후 각 Object별로 neiborhood point를 찾고, core point인지 확인합니다. 이후 dfs를 통해 clustering을 수행합니다.

2) 자료구조

```
class Object {
private:
    int id;
    double x;
    double y;
    int clusterNum;
    bool isCoreObject;
    vector<int> NEps;

public:
    Object(string iId, string iX, string iY) {
        id = stoi(iId);
        x = stod(iX);
        y = stod(iY);
        clusterNum = -1;
        isCoreObject = false;
    }
```

Object는 id, x좌표, y좌표, 속한 cluster의 번호, core point인지 확인하는 bool 변수와 neighborhood points를 담는 NEps를 가지고 있으며, 생성자를 통해 초기화됩니다. 또한 각 변수에 대한 getter와 setter를 가지고 있습니다.

```
void makeDisicionIfisCore(int minPts) {
    isCoreObject = NEps.size() >= minPts ? true : false;
}
```

Neighborhood point의 개수가 MinPts 이상인지를 확인하여 core point인지를 결정하는 메소드도 갖고 있습니다.

3) 파일 읽기

```
vector<Object*> readFile(string fileName) {
    ifstream openFile(fileName.data());
    vector<Object*> objects;

    if (!openFile.eof()) {
        string line;
        while (getline(openFile, line)) {
            vector<string> datas;

            int pre = 0;
            for (int i = 0; i < line.size(); i++) {
                if (line[i] == '₩t') {
                    string data = line.substr(pre, i - pre);
                    datas.push_back(data);
                    pre = i + 1;
                }
            }
            string word = line.substr(pre, line.size() - pre);
            datas.push_back(word);

            Object* newObject = new Object(datas[0], datas[1], datas[2]);
            objects.push_back(newObject);
        }
    }
    openFile.close();

    return objects;
}
```

파일을 읽고 '₩t'을 기준으로 파싱하여 Object에 값을 넣어줍니다.

4) dfs

```
void dfs(vector<Object*> objects, int now, int cluster) {
    objects[now]->setClusterNum(cluster);
    if (!objects[now]->isCore()) return;

    for (int i = 0; i < objects[now]->getNEpsSize(); ++i) {
        int next = objects[now]->getNEps()[i];
        if (objects[next]->getClusterNum() == -1) {
            dfs(objects, next, cluster);
        }
    }
}
```

Clustering을 진행하는 함수입니다. Object가 Core일 때 neighborhood points를 타고 들어가며 cluster의 번호를 입력해줍니다. 계속 진행하다 Core point가 아닌 Object를 만나면 return하며, 이 과정을 통해 cluster가 만들어집니다.

5) main

```
for (int i = 0; i < objects.size(); ++i) {
    for (int j = 0; j < objects.size(); ++j) {
        if (i == j) continue;

        double iX = objects[i]->getX(), jX = objects[j]->getX();
        double iY = objects[i]->getY(), jY = objects[j]->getY();
        double dist = (iX - jX) * (iX - jX) + (iY - jY) * (iY - jY);

        if (dist < eps * eps) {
            objects[i]->addNEps(j);
        }
    }
    objects[i]->makeDisicionIfisCore(minPts);
}
```

점들 사이의 거리를 계산하여 neighborhood point인지 확인한 후 배열에 넣어줍니다.

그 후 각 object가 core point인지 확인해줍니다.

```
for (int i = 0; i < objects.size(); i++) {
    if (objects[i]->getClusterNum() == -1) {
        if (objects[i]->isCore()) {
            dfs(objects, i, ++clusterIndex);
        }
        else {
            objects[i]->setClusterNum(-2);
        }
    }
}
```

이 후 모든 객체에 대해 dfs를 실행하며 clustering을 진행합니다.

Core point가 아닌 경우 noise로 표시해줍니다.

```
clusters.resize(clusterIndex + 1);
for (int i = 0; i < objects.size(); i++) {
    if (objects[i]->getClusterNum() != -2) {
        clusters[objects[i]->getClusterNum()].push_back(i);
    }
}
```

정해진 cluster의 번호에 맞게 cluster에 object id들을 넣어줍니다.

```
sort(clusters.begin(), clusters.end(), compare);
while (clusters.size() > n) {
    clusters.pop_back();
}
```

Cluster가 정해진 cluster의 개수보다 많이 생성되었을 경우, cluster안에 포함된 point의 수가 작은 작은 것부터 삭제하여 cluster의 수를 맞추어줍니다.

```
writeFile(clusters, fileName);
```

생성된 cluster들을 기반으로 output 파일을 만들어줍니다.

6) 결과 출력

```
void writeFile(vector<vector<int> > clusters, string filename) {
    for (int i = 0; i < clusters.size(); ++i) {
        string outputFileName = makeOutputFileName(filename, i);
        ofstream writeFile(outputFileName);

        for (int j = 0; j < clusters[i].size(); ++j) {
            string str = to_string(clusters[i][j]) + '\n';
            writeFile.write(str.c_str(), str.size());
        }

        writeFile.close();
    }
}
```

cluster별로 그 안에 든 object의 id를 출력해줍니다.

```
string makeOutputFileName(string filename, int clusterNum) {
    string inputFileNum = filename.substr(0, 6);
    return inputFileNum + "_cluster_" + to_string(clusterNum) + ".txt";
}
```

다음과 같이 출력 파일의 이름을 만들어줍니다.

4. 테스트 결과

다음과 같은 결과를 확인할 수 있었습니다.