2018008304 컴퓨터소프트웨어학부 박경하

알고리즘 요약

우선, input file에서 데이터를 받아와 그것을 x, y좌표와 object id를 가지고 있는 벡터 안에 저장합니다. 그 뒤, 랜덤하게 포인터 p 하나를 찍은 뒤에, p가 만약 core point라면 주변에 있는 모든 점들을 클러스터 안에 저장합니다. 그리고 클러스트에 담았던 모든 점들에 대해서 direct density-reachable인지 체크한 뒤, 조건이 성립하면 border 일 때까지 재귀적으로 반복하여 클러스터에 저장합니다. 그리고 모두 끝나면, 다시 포인터 p를 찍고 위와 같은 과정을 n번 반복합니다.

코드 상세 설명

```
15    struct object {
16        int object_id;
17        double x_coordinate;
18        double y_coordinate;
19    };
```

Input 데이터를 담을 구조체를 선언하였습니다.

```
173       for(int i=0; i<n; i++) {
174
175           GetPoint(&point);
176           GetCluster(point, Eps, MinPts);
177
```

Input 파일을 위의 구조체에 담은 뒤에, 위의 과정을 n번 반복합니다. 먼저 랜덤으로 p를 뽑기 위해 GetPoint() 함수를 불러오게 됩니다.

```
29    void GetPoint(int *p){
30        /* data_set에서 랜덤으로 포인트값 뽑기 */
31
32        srand(time(NULL));
33        bool isExit = true;
34
35        /* 이미 클러스터에 있는 포인터값이라면, 다시 난수추출 */
36        while(isExit) {
37            *p = rand() % data_set.size(); //0~data_set.size 의 범위에서 난수 생성
38
39            if (cluster.size() == 0)
40                isExit = false;
41            else {
42                isExit = false;
43                for(vector<object>::iterator it = used_data.begin(); it!= used_data.end(); it++) {
44
45                    if(*p == it->object_id) {
46                        isExit = true;
47                        break;
48                    }
49                }
50            }
51        }
52    }
```

Getpoint()함수에서는, 먼저 srand를 이용하여 랜덤하게 데이터 사이즈 만큼 난수를 불러오게 되는데, 만약 랜덤하게 부른 p가 클러스터에 이미 저장되어 있다면, 혹은 다른 클러스터에 사용되었다면 다시 뽑게 합니다.

```
100  void GetCluster(int p, int eps, int minpts){
101      /* point가 core-point 조건을 만족하는지 확인 */
102      //현재 p의 좌표에서 Eps값을 +- 한 내역 안에 있는 모든 점들의 개수를 구함. 그 개수가  minPts 이상이어야함.
103
104      cluster.clear();
105
106      int neighborhood_n = 0;
107      vector<object> temp_cluster;
108
109
110      for(vector< object>::iterator it = data_set.begin(); it!=data_set.end(); ++it) {
111          if((it->x_coordinate >= data_set[p].x_coordinate - eps) && (it->x_coordinate <= data_set[p].x_coordinate + eps)) {
112              if((it->y_coordinate >= data_set[p].y_coordinate - eps) && (it->y_coordinate <= data_set[p].y_coordinate + eps)) {
113                  neighborhood_n++;
114                  cluster.push_back(*it);
115                  used_data.push_back(*it);
116              }
117          }
118      }
119
120      if(neighborhood_n >= minpts) { //core-point 조건성립
121          //temp_cluster 내에 있는 object도 DDR을 체크하여 cluster을 확장해나감
122          int size = cluster.size();
123          for(int i=0; i<size; i++)
124              CheckDDR(cluster[i].object_id, eps, minpts);
125      }
```

그 후, 랜덤하게 부른 p가 core-point의 조건을 만족하는지 확인합니다. 그 뒤, 클러스터에 eps 내에 있는 점들을 모두 넣어주고, 사용한 데이터에도 값들을 모두 넣어줍니다. Core-point라면, 클러스터 내에 있는 모든 Object가 direct density-reachable인지 확인하는 재귀함수를 부릅니다.

```
54   void CheckDDR(int p, int eps, int minpts){
55      int neighborhood_n = 0;
56       vector<object> temp_cluster;
57
58       for(vector<object>::iterator it = data_set.begin(); it!=data_set.end(); ++it) {
59           if((it->x_coordinate >= data_set[p].x_coordinate - eps) && (it->x_coordinate <= data_set[p].x_coordinate + eps)) {
60               if((it->y_coordinate >= data_set[p].y_coordinate - eps) && (it->y_coordinate <= data_set[p].y_coordinate + eps)) {
61                   neighborhood_n++;
62                   temp_cluster.push_back(*it);
63               }
64           }
65       }
66
67       //기존의 클러스터 뒤에 삽입
68
69       vector<int> erase_num;
70
71       for(vector<object>::iterator it = temp_cluster.begin(); it!=temp_cluster.end(); ++it) {
72               for(vector<object>::iterator it_ = cluster.begin(); it_!=cluster.end(); ++it_) {
73                   if(it->object_id == it_->object_id) {
74                       erase_num.push_back(it->object_id);
75                   }
76               }
77       }
78       for(int i=0; i<erase_num.size(); i++) {
79           for(vector<object>::iterator it = temp_cluster.begin(); it!=temp_cluster.end(); ++it) {
80               if(it->object_id == erase_num[i]) {
81                   temp_cluster.erase(it);
82                   break;
83               }
84           }
85       }
```

CheckDDR에서도 위 함수와 마찬가지로 eps내에 있는 object의 개수가 minpts 이상인지 확인하는데, 이 때 eps 내에 있는 object가 이미 클러스터 내에 있는 object일 수 있으므로 erase_num을 이용하여 걸러줍니다.

```
87       if(temp_cluster.size() != 0){
88           cluster.insert(cluster.end(), temp_cluster.begin(), temp_cluster.end());
89           used_data.insert(used_data.end(), temp_cluster.begin(), temp_cluster.end());
90
91
92           if(neighborhood_n >= minpts){ //DDR 조건성립
93               for(vector<object>::iterator it = temp_cluster.begin(); it!=temp_cluster.end(); ++it) {
94                   CheckDDR(it->object_id, eps, minpts);
95               }
96           }
97       }
98   }
```

그 뒤, 클러스터와 사용한 데이터에 각각 넣어준 뒤, 만약 이 점이 direct density-reachable 조건을 성립하게 되면 eps내에 있던 점들을 모두 재귀함수로 인자로 넣어줍니다.

컴파일 설명

```
➜  Assignment3 git:(master) ✗ make
g++ -o clustering clustering.o
```

Makefile을 이용하여 해당 .cpp 파일을 컴파일 하였습니다.

```
➜  Assignment3 git:(master) ✗ ./clustering input3.txt 4 5 5
➜  Assignment3 git:(master) ✗
```

구현 및 테스트 사양

구현은 MAC의 visual code에서 하였으며, 테스트는 .exe파일을 실행하기 위해 window10에서 진행하였습니다

```
C:₩Users₩dudgk₩Documents₩카카오톡 받은 파일₩test-3>PA3.exe input1
85.59477점
C:₩Users₩dudgk₩Documents₩카카오톡 받은 파일₩test-3>PA3.exe input2
91.55122점
C:₩Users₩dudgk₩Documents₩카카오톡 받은 파일₩test-3>PA3.exe input3
99.97736점
C:₩Users₩dudgk₩Documents₩카카오톡 받은 파일₩test-3>
```