[공간데이터 관리 및 응용 프로젝트 2]

20161575 김연요

1. 프로젝트 파일 분류:

README 파일에도 기술해놓았지만, 폴더를 압축 해제하시면 20161575 Code 폴더 안에 해당 보고서 외에도 3 가지 dataset 들과 BruteForce, KDTree, RTree 폴더 세 개가 존재합니다. 각 폴더에서 각 해당하는 방법의 Query 를 실행해보실 수 있습니다. 컴파일 방법과 실행 방법 역시 각 폴더안의 README 파일을 확인하시면 좋을 것 같습니다.

2. 각 질의 처리 구현 함수와 자료구조

세 개의 방법 모두에서 main 함수에서는 Query 를 입력받고, dataset 종류를 입력 받고, 각 각 선택한 종류의 Query 에 따라 해당 query 들을 실행합니다.

1) Bruteforce.c

브루트포스의 경우, 이름과 마찬가지로 전체 탐색을 통해 질의 조건에 맞는 객체를 탐색합니다.
사용한 자료구조의 경우 2 차원 좌표값을 저장할 수 있는 **point** 구조체와 knn query 에서 배열을 top-k 개 저장할 수 있는 구조체 **knn** 또한 정의했습니다. 그 외에도 두 개의 point 구조체들 사이의 유클리디언 distance 를 return 하는 함수인 **euclid_dist** 또한 정의했습니다.

2) KDTree.c

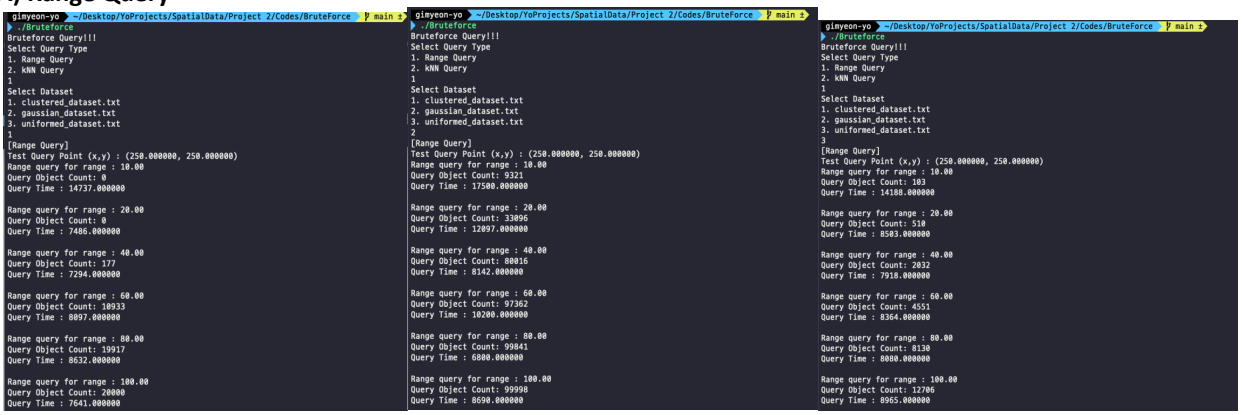
KDTree 에서는 기본적으로 제공된 구조체와 함수들을 적극 활용했습니다.
그 외에도, knn query 를 할 때 사용한 **knn_node** 구조체와 resultQ 라는 priority queue 를 직접 구현해서 그 기능들을 활용하여 kNN 의 후보가 되는 오브젝트 값들을 정렬해서 저장했습니다. priority queue 구조체를 선언함과 동시에 해당 priority queue 가 사용하는 함수들 (push, pop, empty 등) 을 직접 구현했습니다.

각 코드를 구현한 방식에 대해서는 주석을 통해 상세히 적어놨습니다.

3. 실험 결과 분석

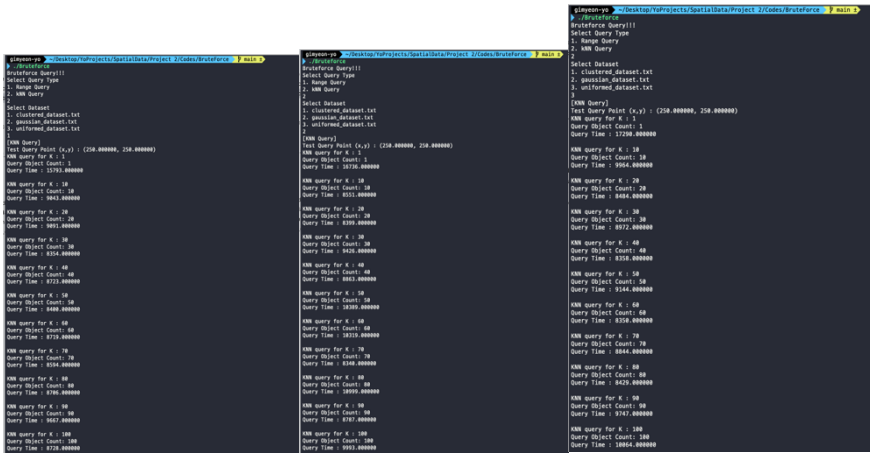
1) Bruteforce

A) Range Query



Brute Force 를 각각 Clustered, Gaussian, Uniform 한 dataset 에 대해 Range query 를 질의한 결과입니다.

B) KNN Query



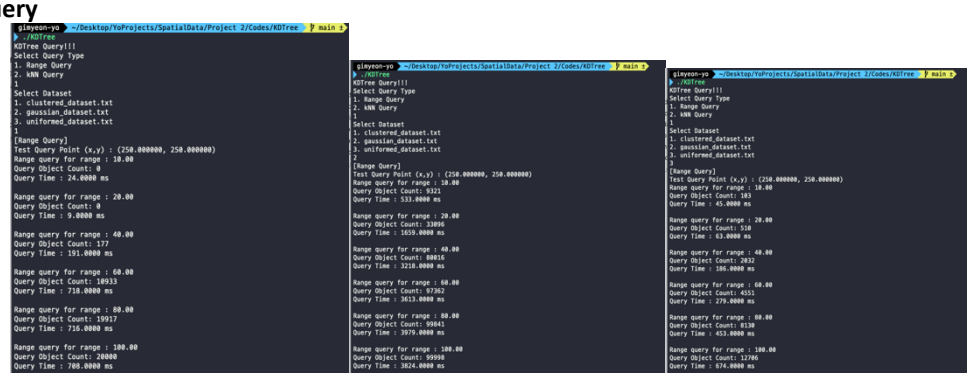
Brute Force 를 각각 Clustered, Gaussian, Uniform 한 dataset 에 대해 KNN query 를 질의한 결과입니다.

결과 분석 : Bruteforce 의 경우 모든 data 들에 대해 전체 탐색을 진행하기 때문에 Query 종류와 관계 없이 시간이 오래걸리는 비효율성을 보여준다. 그렇기 때문에 당연히 Range 값이나 K 값에도 시간 영향은 받지 않는 모습을 확인할 수 있었다.

이후에 KD tree 와의 비교를 통해 시간 효율성을 더 깊이 비교, 확인해보겠다.

2) KD Tree

A) Range Query



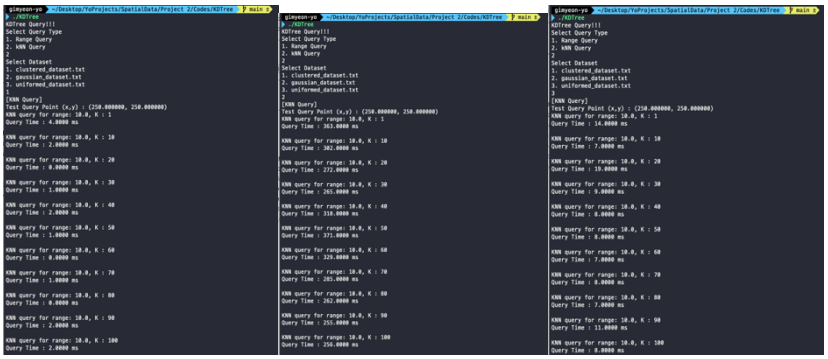
KD Tree 를 사용하여 각각 Clustered, Gaussian, Uniform 한 dataset 에 대해 Range query 를 질의한 결과입니다.

B) KNN Query

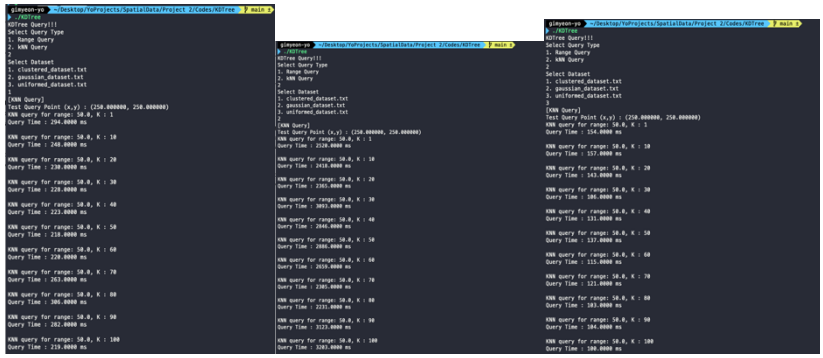
KD Tree 를 사용하여 Clustered, Gaussian, Uniform 한 dataset 에 대해 KNN query 를 질의한 결과입니다.

KNN query 의 경우, K 값을 {1,10,20,30,…100}뿐만 아니라 range 값 또한 radius = 10, 50, 100 으로 변화를 주면서 모두 비교를 해보았습니다.

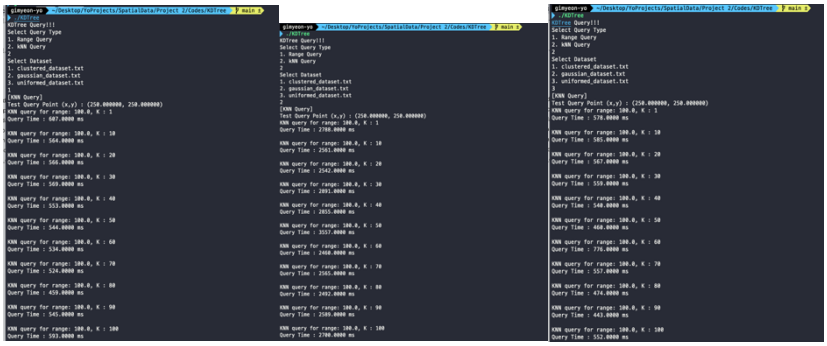
ㄱ. R = 10, (Clustered, Gaussian, Uniform 순서)



ㄴ. R = 50, (Clustered, Gaussian, Uniform 순서)



ㄷ. R = 100, (Clustered, Gaussian, Uniform 순서)



결과 분석 : 우선, 전체적으로 Bruteforce 에 비해서는 훨씬 효율성이 높은 결과를 보인다.
당연히 radius 가 높아질수록 더 오래 걸리는 결과를 확인할 수 있었고,
gaussian 같이 데이터가 불균형한 데이터에 대해서 가장 오래 걸리는 효율성을 확인했다.
그 반대로 uniform data 와 같이 균일한 데이터 분포에서 가장 빠른 시간을 보였는데, 이는 KD Tree 의 특성상 데이터가 균일할 수록 더 pruning 과정에서 많은 데이터들을 prune 할 수 있기 때문이다.

<감사합니다.>