공간 데이터 관리 proj#1

2019-12-09

20151575 윤제형

교수명 : 정성원

목차

1.프로젝트 개요

- 공간 정보를 저장하기 위해선 공간 정보에 특화된 자료구조를 사용하고 이를 위한 최적화된 query algorithm 본 프로젝트에선 이를 위한 자료구조인 Rtree , KDtree 를 사용하여 보고 Range Query 와 Knn Query를 해당 자료구조에 질의 해봄으로써 수업 시간에 배운 내용들을 실습해 보는데 목적을 둔다.

  해당 자료구조를 적절히 탐색, Best first search 방법과 pruning 기법을 사용하여 본 프로젝트의 결과 분석에서는 object 에 대한 접근을 최소한으로 줄이고 시간을 brute Force search 와 비교하여 획기적으로 시간을 단축시켜 이를 비교해 보고자 한다.

2.프로젝트 계획

A. Pseudocode

i. Kdtree_range

```
For temp in nodeQ:
        If dist(temp, target_cir) < r:
                Push(result,temp)
        If(temp -> left): -> right 에 대해서도
                If(left 의 공간안에 원이 있다면)
                        Push(nodeQ,temp ->left)
                If(left 의 공간 안에 원이 닿는 다면)
                        Push(nodeQ,temp ->left)
```

ii. Kdtree_knn

```
For temp in nodeQ:
        If(priority Q 길이 < k)
                Push(priority,temp)
        Elseif( max(priority) <= temp 와 원의 거리)
                Pruning
        Else
                Delete_tail(priorityQ)
                Push(priorityQ,temp);
        If(temp -> left): -> right 에 대해서도
                If(left 의 공간안에 원이 있다면)
                        Push(nodeQ,temp ->left)
                If(left 의 공간 안에 원이 닿는 다면)
                        Push(nodeQ,temp ->left)
```

iii. Rtree_range(node)

```
If(node is not leaf)
        For child in node->childs:
                Double mindist = mindist(child,cir)
                If(mindist <= r)
                        Rtree_range(r -> child);
Else
        For data in node->childs:
                Double dist = dist(data,cir)
                If(dist <= r)
                        Push(result,data);
```

iv. Rtree_knn

```
For temp in nodeQ//nodQ is minheap
        If(node is not leaf)
                For child in node->childs:
                        Double mindist = mindist(child,cir)
                        If(len(result) < k ||mindist <= tail(result))
                                Heap_insert(nodeQ,child,mindist
        Else
                For data in node->childs:
                        Double dist = dist(data,cir)
                        If(len(result)<k )
                                Push(result,data);
                        Elseif(dist <= tail(result))
                                Delete_tail(result)
                                Push(result,data);
```

3.프로젝트 결과 및 실험

오른쪽 사진은 프로젝트 결과 도출된 프로그램을 실행 시켜 3 가지 쿼리가 모두 같은 결과를 출력 하는 지에 대한 결과이다. 이 사진을 보면 모두 일치하는 결과를 보였다.

```
Rtree 10nn KDtree 10nn 일치
now: 70nn
brute_force 70nn
KDtree 70nn
Rtree 70nn
brute_force 70nn KDtree 70nn 일치
KDtree 70nn Rtree 70nn 일치
Rtree 70nn brute_force 70nn 일치
now: 60nn
brute_force 60nn
KDtree 60nn
Rtree 60nn
brute_force 60nn KDtree 60nn 일치
KDtree 60nn Rtree 60nn 일치
Rtree 60nn brute_force 60nn 일치
now: 80nn
brute_force 80nn
KDtree 80nn
Rtree 80nn
brute_force 80nn KDtree 80nn 일치
KDtree 80nn Rtree 80nn 일치
Rtree 80nn brute_force 80nn 일치
now: 20nn
Rtree 20nn
brute_force 20nn
KDtree 20nn
Rtree 20nn brute_force 20nn 일치
brute_force 20nn KDtree 20nn 일치
KDtree 20nn Rtree 20nn 일치
now: 5.000000
Rtree 5.000000 range
brute_force 5.000000 range
KDtree 5.000000 range
Rtree 5.000000 range brute_force 5.000000 range 일치
brute_force 5.000000 range KDtree 5.000000 range 일치
KDtree 5.000000 range Rtree 5.000000 range 일치
now: 40nn
brute_force 40nn
KDtree 40nn
Rtree 40nn
brute_force 40nn KDtree 40nn 일치
KDtree 40nn Rtree 40nn 일치
Rtree 40nn brute_force 40nn 일치
now: 50nn
KDtree 50nn
Rtree 50nn
brute_force 50nn
KDtree 50nn Rtree 50nn 일치
Rtree 50nn brute_force 50nn 일치
brute_force 50nn KDtree 50nn 일치
now: 50.000000
KDtree 50.000000 range
Rtree 50.000000 range
brute_force 50.000000 range
KDtree 50.000000 range Rtree 50.000000 range 일치
Rtree 50.000000 range brute_force 50.000000 range 일치
brute_force 50.000000 range KDtree 50.000000 range 일치
now: 10.000000
KDtree 10.000000 range
Rtree 10.000000 range
brute_force 10.000000 range
KDtree 10.000000 range Rtree 10.000000 range 일치
Rtree 10.000000 range brute_force 10.000000 range 일치
brute_force 10.000000 range KDtree 10.000000 range 일치
```

R5	1	2	3	평균
rtree	0.000695	0.000514	0.001511	0.000907
kdtree	0.000667	0.00475	0.001361	0.002259
bruteForce	0.006927	0.007885	0.010304	0.008372

R10	1	2	3	평균
rtree	0.002124	0.001736	0.00487	0.00291
kdtree	0.002519	0.001835	0.00509	0.003148
bruteForce	0.007821	0.007673	0.012961	0.009485

R20	1	2	3	평균
rtree	0.008816	0.007266	0.019046	0.011709
kdtree	0.00949	0.007969	0.01844	0.011966
bruteForce	0.011807	0.010876	0.020026	0.014236

R30	1	2	3	평균
rtree	0.020066	0.0172	0.03926	0.025509
kdtree	0.021	0.01916	0.036259	0.025473
bruteForce	0.018088	0.0765	0.02844	0.041009

R40	1	2	3	평균
rtree	0.03723	0.033541	0.059083	0.043285
kdtree	0.0367	0.034586	0.055062	0.042116
bruteForce	0.026523	0.024452	0.033723	0.028233

R50	1	2	3	평균
rtree	0.055641	0.052377	0.074253	0.060757
kdtree	0.053185	0.05016	0.068819	0.057388
bruteForce	0.032122	0.030914	0.030775	0.03127

k1	1	2	3	평균
rtree	0.055641	0.000091	0.000216	0.018649
kdtree	0.000007	0.000012	0.00001	9.67E-06
bruteForce	0.01414	0.014592	0.014868	0.014533

k10	1	2	3	평균
rtree	0.000232	0.000121	0.000292	0.000215
kdtree	0.000015	0.000021	0.000014	1.67E-05
bruteForce	0.055093	0.056099	0.59056	0.233917

k20	1	2	3	평균
rtree	0.000337	0.000174	0.000531	0.000347
kdtree	0.000034	0.000039	0.000043	3.87E-05
bruteForce	0.121762	0.12678	0.13023	0.126257

k30	1	2	3	평균
rtree	0.000484	0.000259	0.000728	0.00049
kdtree	0.000056	0.000062	0.000069	6.23E-05
bruteForce	0.189708	0.190597	0.204548	0.194951

k40	1	2	3	평균
rtree	0.00073	0.00036	0.001063	0.000718
kdtree	0.00008	0.000104	0.000097	9.37E-05
bruteForce	0.257885	0.258455	0.248338	0.254893

k50	1	2	3	평균
rtree	0.000938	0.000439	0.001297	0.000891
kdtree	0.000098	0.000146	0.000137	0.000127
bruteForce	0.37096	0.362797	0.410644	0.381467

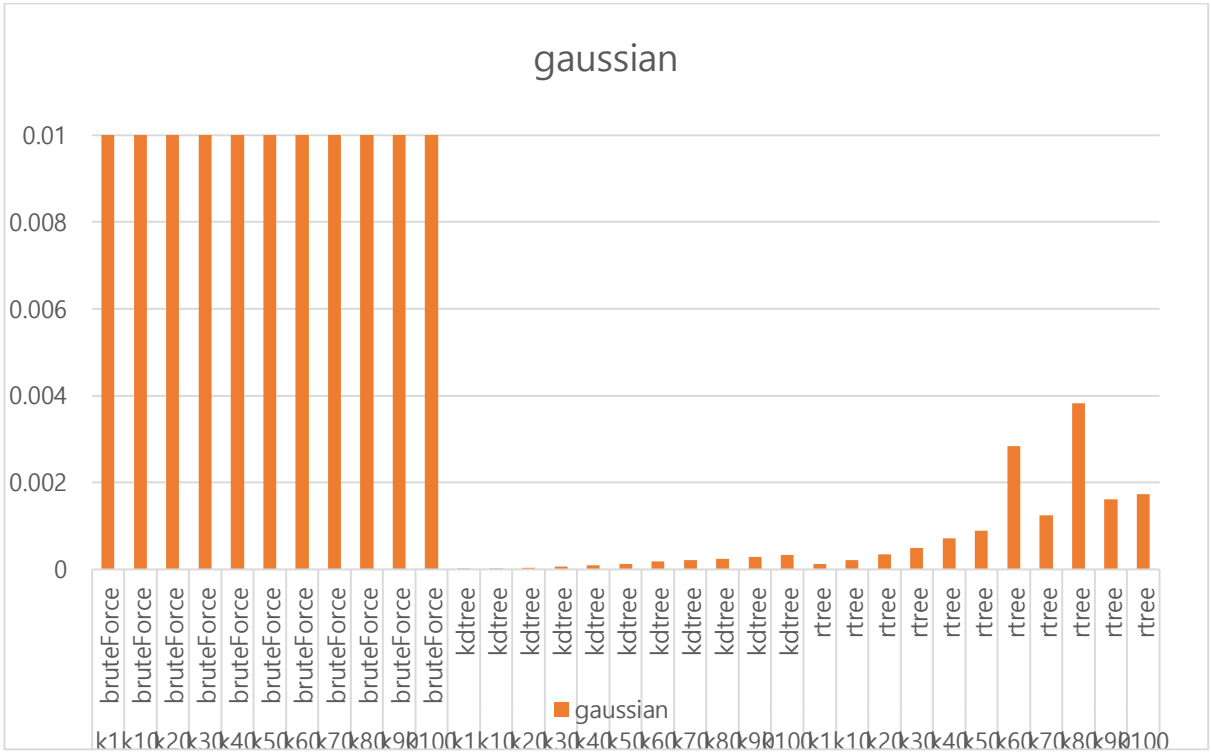
k60	1	2	3	평균
rtree	0.001077	0.000582	0.006839	0.002833
kdtree	0.000153	0.000207	0.000177	0.000179
bruteForce	0.441044	0.42375	0.479329	0.448041

Figure 1 : 테스트 케이스당 3번 실시하여 평균을 낸 모습

R5	gaussian	clustering	uniformed	k1	gaussian	clustering	uniformed
rtree	0.000907	0.001904	0.000061	rtree	0.01864933	9.567E-05	7.13333E-05
kdtree	0.002259	0.001709	6.56667E-05	kdtree	9.6667E-06	2.767E-05	6.33333E-06
bruteForce	0.008372	0.007338	0.00706	bruteForce	0.01453333	0.0142123	0.014625333
R10				k10			
rtree	0.002124	0.005449	0.000147333	rtree	0.000215	0.0001692	0.000103333
kdtree	0.002519	0.005389	0.000163333	kdtree	1.6667E-05	1.533E-05	1.96667E-05
bruteForce	0.007821	0.009156	0.005111533	bruteForce	0.23391733	0.055773	0.056845333
R20				k20			
rtree	0.008816	0.013596	0.000488667	rtree	0.00034733	0.0001727	0.000183667
kdtree	0.00949	0.012237	0.000555667	kdtree	3.8667E-05	0.000034	3.86667E-05
bruteForce	0.011807	0.013222	0.007208333	bruteForce	0.12625733	0.1237867	0.126640333
R30				k30			
rtree	0.020066	0.015348	0.000876667	rtree	0.00049033	0.0003827	0.000258333
kdtree	0.021	0.014343	0.001085333	kdtree	6.2333E-05	5.567E-05	7.26667E-05
bruteForce	0.018088	0.014873	0.007446333	bruteForce	0.194951	0.1925827	0.195794
R40				k40			
rtree	0.03723	0.014609	0.001359333	rtree	0.00071767	0.0009777	0.000359333
kdtree	0.0367	0.014183	0.001819333	kdtree	9.3667E-05	8.167E-05	0.000107333
bruteForce	0.026523	0.014979	0.007810333	bruteForce	0.25489267	0.2606827	0.265927667
R50				k50			
rtree	0.055641	0.014203	0.001905333	rtree	0.00089133	0.0010617	0.000431333
kdtree	0.053185	0.014142	0.002722	kdtree	0.000127	0.0001107	0.000140667
bruteForce	0.032122	0.014901	0.254846667	bruteForce	0.381467	0.3723303	0.383867

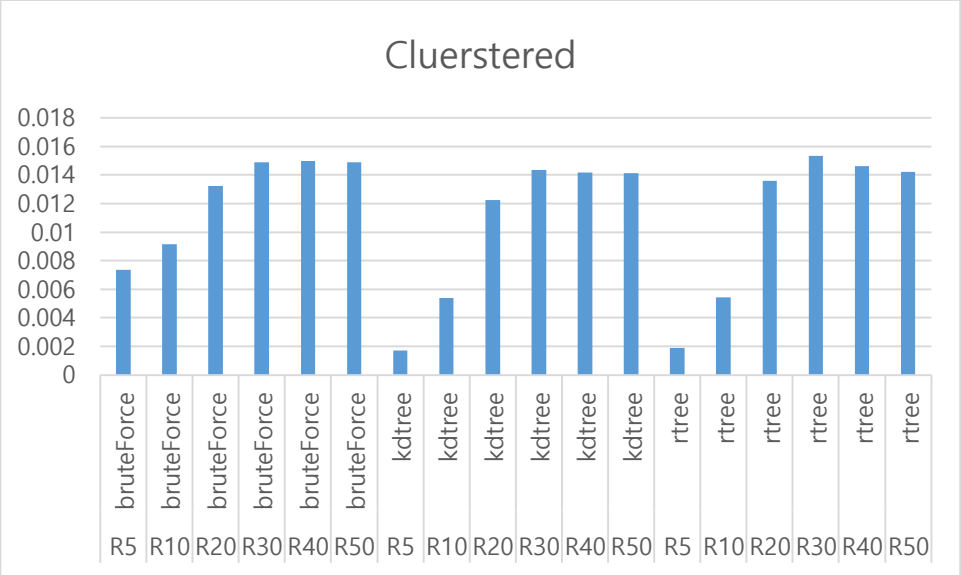
Figure 2: 각 테스트 케이스 마다 평균 낸 데이터 값을 모은 모습

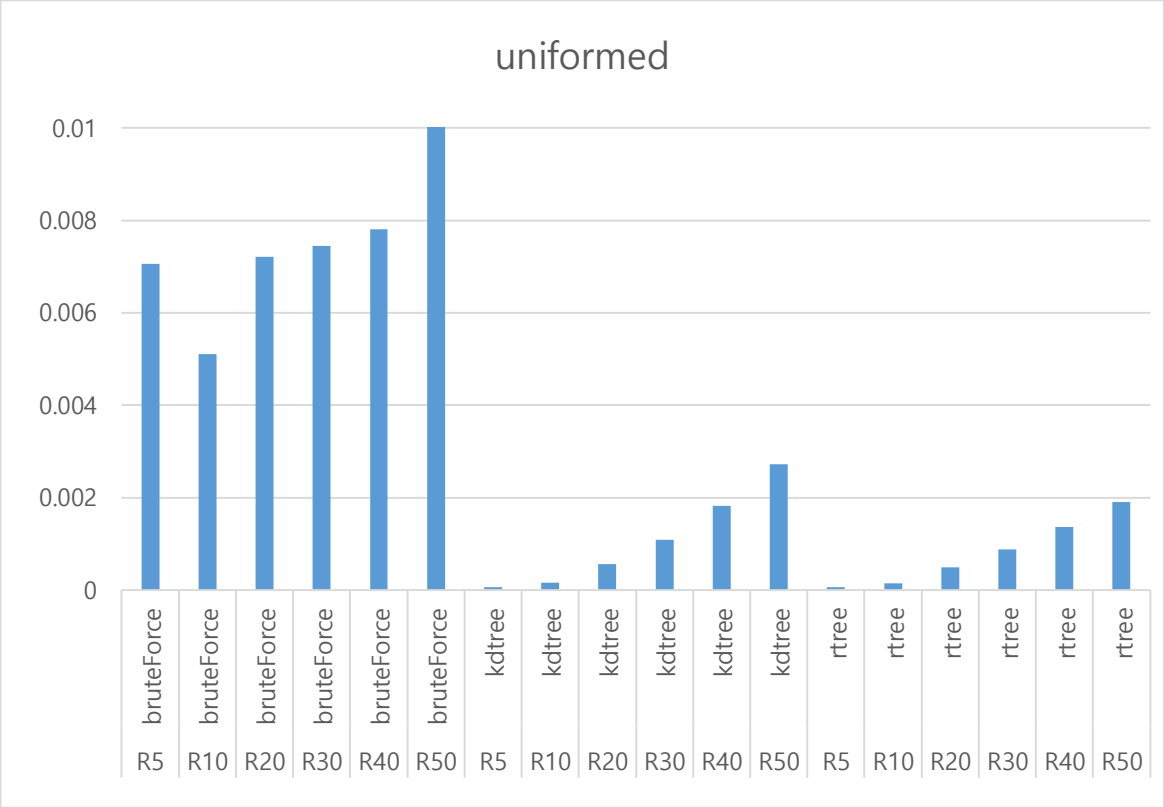
실험은 위의 그림처럼 각 테스트 케이스(Gaussian,Clustered,uniformed) 마다 3 번씩 수행하여 평균을 내었다. 이 값을 이용한 엑셀 함수를 보면



위 사진은 k 범위에 따른 Gaussian data 에 대한 실행 시간을 정리 한 표이다. Bruteforce 의 경우 pruning 을 진행하지 않고 전 검사를 하기 때문에 실행 시간이 매우 느리다. 반면에 pruning 의

효과가 가장 잘 나타나는 knn query 의 경우 kdtree 나 rtree 에서 매우 빠른 성능을 보였다 하지만 rtree 에서보다 kdtree 에서 더욱 빠른 성능을 보였다.





위 두개의 사진은 cluerrstered 된 data 에서의 range query 와 uniformed 된 data 에서의 range query 이다

cluerstered 된 data 에서는 3 가지 모두 비슷한 성능을 보인다. 왜냐하면 data 의 특성상 range 안에 몰려있을 가능성이 높고 그렇게 되면 pruning 이 잘 진행되지 않기 때문에 전검사를 하는 brute force 와 비슷한 성능을 보였다.

Uniformed data 의 경우 균등하게 분포한 data 이기 때문에 cluerstered data 와 달리 brute force 와 kdtree,rtree 간의 많은 성능차를 보였다. 이는 range 밖의 점들에 대해 pruning 을 잘 진행하여 필요한 점들만 search 하기 때문이다. 허나 knn 과 비교하여 pruning 요소가 적기 때문에 knn 만큼의 차이는 보이지 못했다. 또한.두개의 tree 가 비슷한 성능을 보였으나 미세하게 rtree 가 앞섰다.

총평: data 의 특성에 따라 brute force 와 다름없는 실행시간을 보이기도하며 매우 큰 차이를 보이기도 한다. 또한, 전체적인 query 성능은 kdtree 가 조금 앞선다고 볼 수 있다.