Term Project

CSE5601 & CSEG601
공간데이터 관리 및 응용

Range query및 kNN query processing 성능 비교

프로젝트 목표

공간 인덱스인 kd-tree, R-tree를 활용하여 range query 및 kNN query processing algorithm을 설계하고 질의 처리 성능을 비교 분석한다. 구체적으로는 다음 세 가지 방법을 사용하여 질의를 처리하고 **질의 처리에 걸리는 시간** 및 **질의 처리 과정에서 검사되는 객체의 수**를 비교 측정한다.

1. **Brute-force 알고리즘 :** 전탐색을 통해 질의 조건에 부합하는 객체를 검색한다.
2. **kd-tree :** kd-tree 인덱스를 생성하고 DFS, 혹은 BFS(best-first search) 탐색을 통해 질의 조건에 부합하는 객체를 검색한다.
3. **R-tree** : R-tree 인덱스를 생성하고 DFS, 혹은 BFS(best-first search) 탐색을 통해 질의 조건에 부합하는 객체를 검색한다.

제출물

각각의 소스코드, makefile(필요한 경우), 보고서

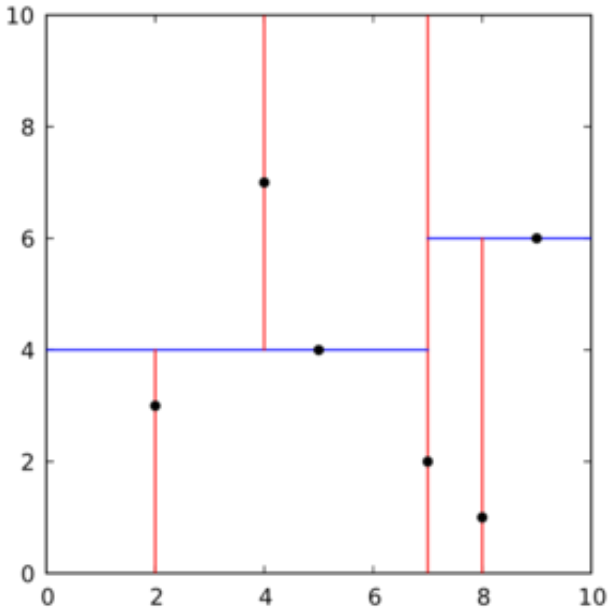
마감일

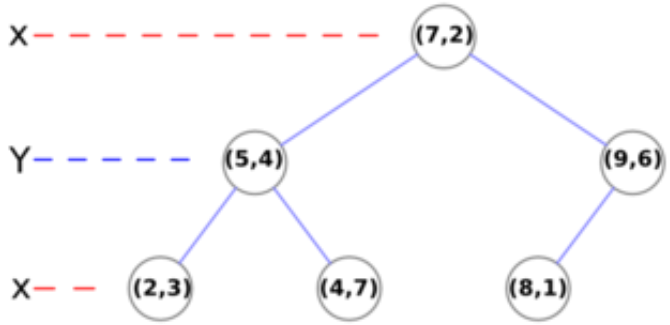
12월 13일 16:30분까지

kd Tree

Kd-tree

- 각 포인트를 기준으로 x축, y축으로 번갈아가면서 공간을 분할
- Time complexity : n개의 point로 kd tree를 만들때,
  - 빌드 : O($n \log^2 n$) or O($n \log n$)
  - Range query : O($n^{1-1/k}+m$), m은 range 질의 처리 결과 집합의 크기
  - NN query : O(log n)





Construct Kd-tree

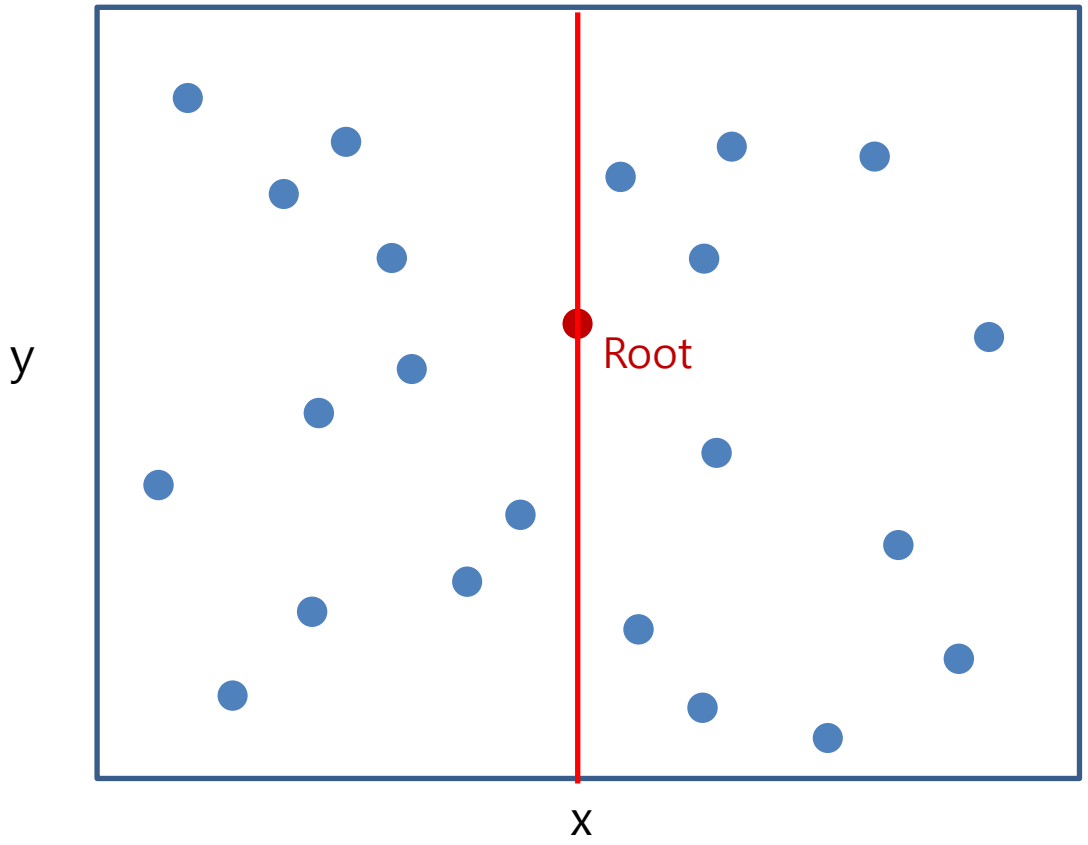
Kd-tree의 노드 구조

```
struct kd_node_t{
        double x[MAX_DIM];          //차원별 좌표
        struct kd_node_t *left;     //왼쪽 자식 노드에 대한 포인터
        struct kd_node_t *right;    //오른쪽 자식 노드에 대한 포인터
}
```

- 데이터를 각 축에 대해 번갈아가면서 정렬
- 각 축에 대해 중간값을 갖는 데이터를 선택해서 split node로 삼는다

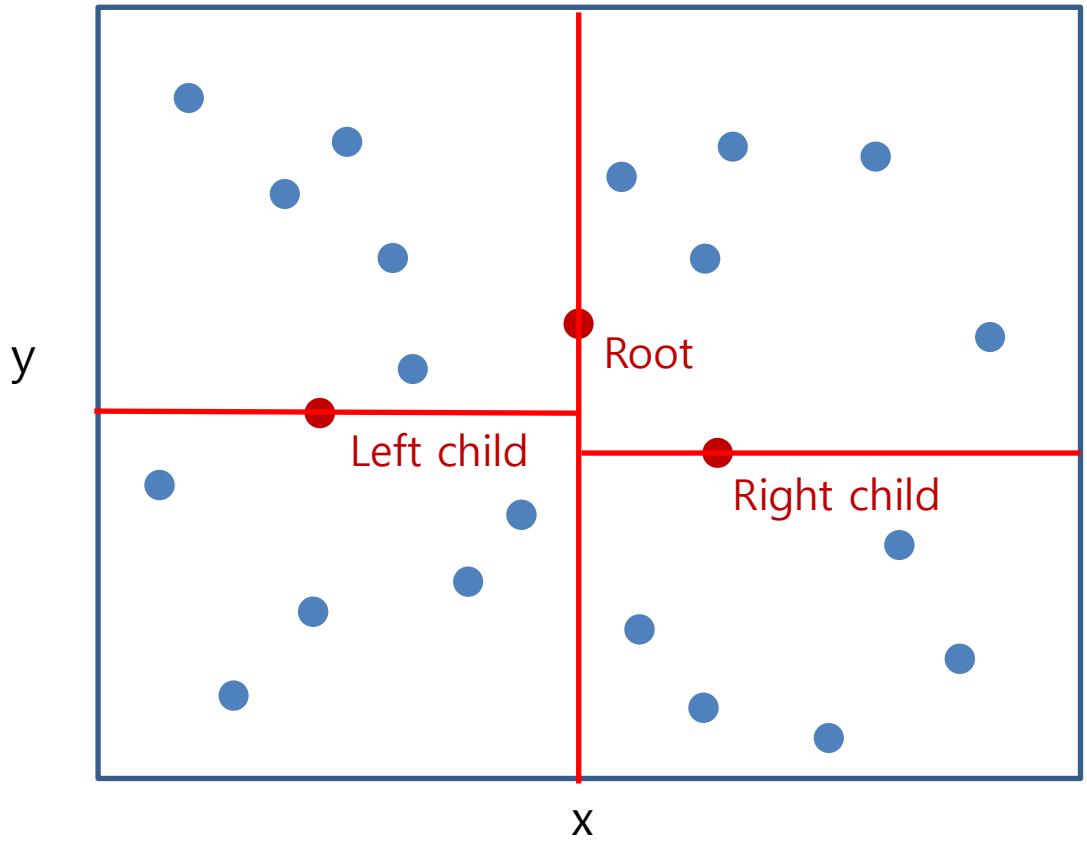
Construct kd tree

- x축의 중간값을 갖는 데이터로 공간을 분할



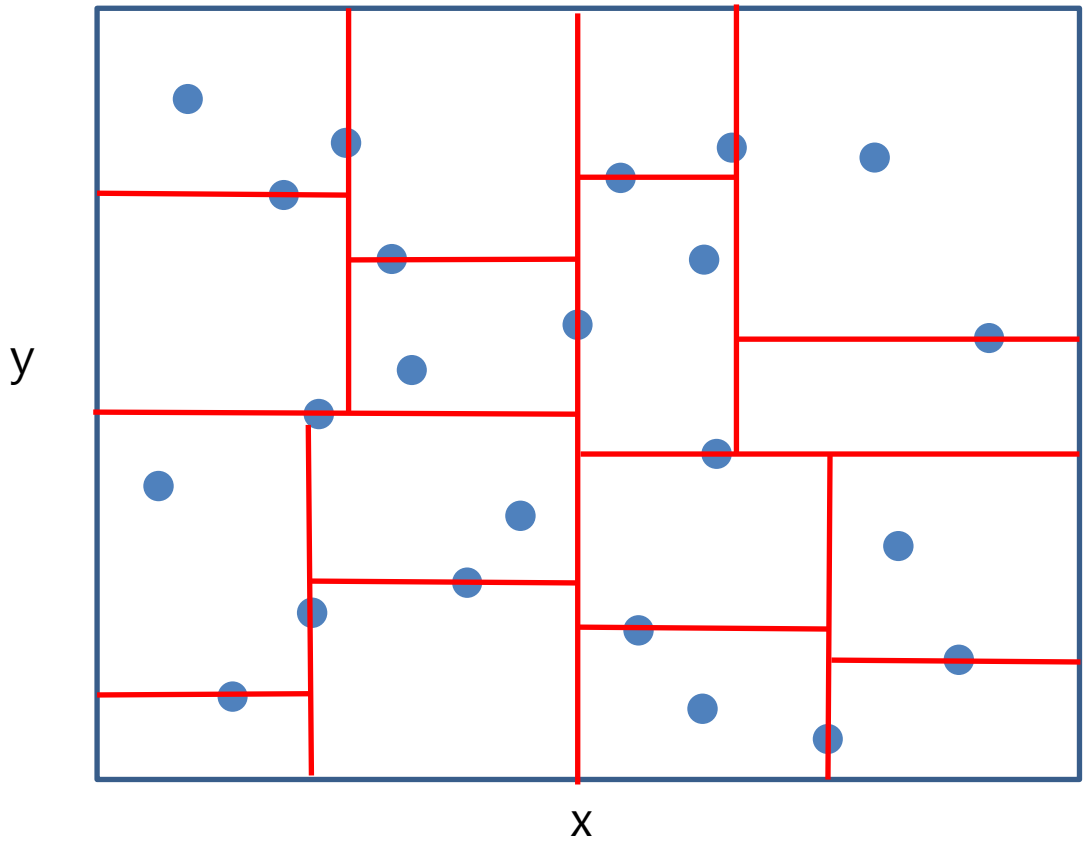
Construct kd tree

- 분할된 각각의 공간에서, y축의 중간값을 갖는 데이터로 공간을 분할함



Construct kd tree

- 모든 포인트가 포함될 때 까지 이 과정을 반복



필요한 자료구조 선언

다음과 같은 자료구조를 사용하는것을 추천

```
struct point{
        double x;
        double y;
}

struct Rect{
        double min_x, min_y;
        double max_x, max_y;
}

struct candidate_node{
        struct kd_node_t current_node;
        Rect rec;
}
```

- range query의 경우 candidate_node에 대한 stack, 혹은 queue 정의
- kNN query의 경우 candidate_node에 대한 <distance, candidate_node> pair에 대한 priority queue 정의(distance가 작은 준서대로 정렬)

함수 작성 예시

기타 필요한 함수 작성 예시
distance(point p, Rect rec) //포인트와 사각형 까지의 최소 거리

rangeQuery(point query_p, double radi)
//range query의 질의 조건인 질의 포인트와 질의 반경

kNNQuery(point query_p, int k)
//kNN query의 질의조건인 질의 포인트와 최근접이웃 개수

kd Tree에서의 Range query

Range query using Kd-tree

질의 조건으로 포인트 ***q*** (x,y 좌표)와 질의 범위 ***r*** 이 주어진다. 대략적인 질의 처리 프로세스는 다음과 같다.

1. 루트노드로부터 탐색을 시작

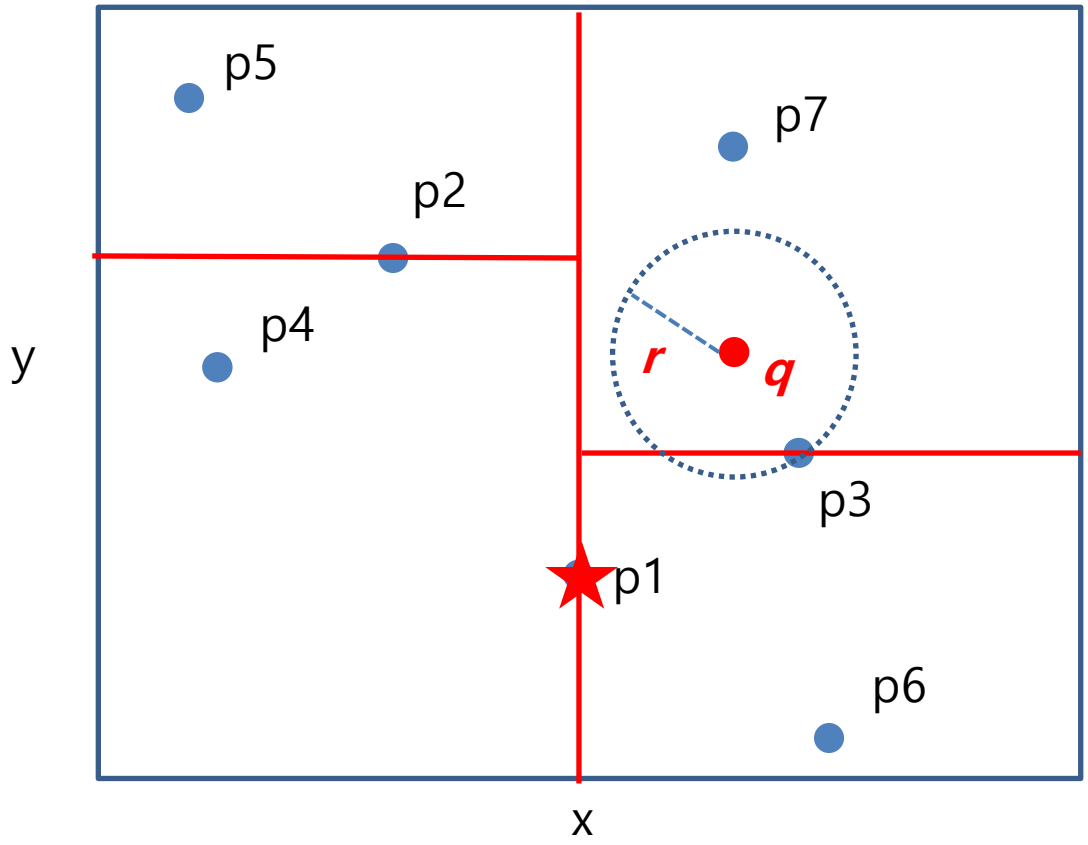
2. 노드에 속한 포인트 ***p*** 와 질의 포인트 ***q*** 와의 거리를 측정하여 질의 조건에 부합하는지 검사한다.

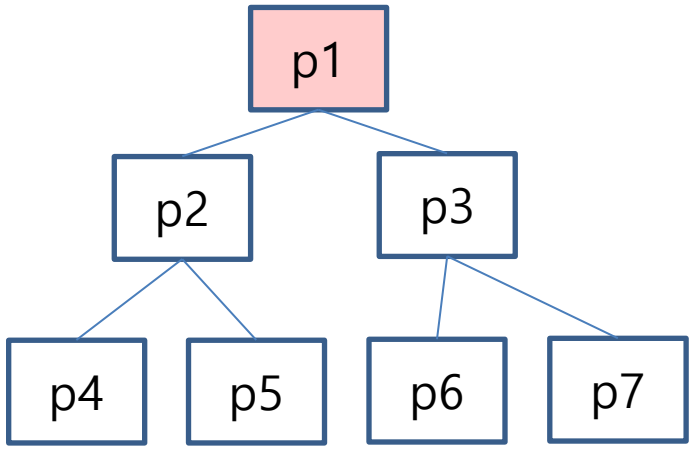
3. ***p*** 로 인해 분할된 왼쪽, 오른쪽 자식들이 속한 영역이 질의 반경과 겹치는 부분이 있는지 검사한 후, 겹치는 경우 해당 영역을 탐색한다(stack에 삽입).

4. 더 이상 탐색할 노드가 없을 때 까지 2~3번의 작업을 반복한다(stack이 empty인 경우 중단).

Example -Range Query using KD tree

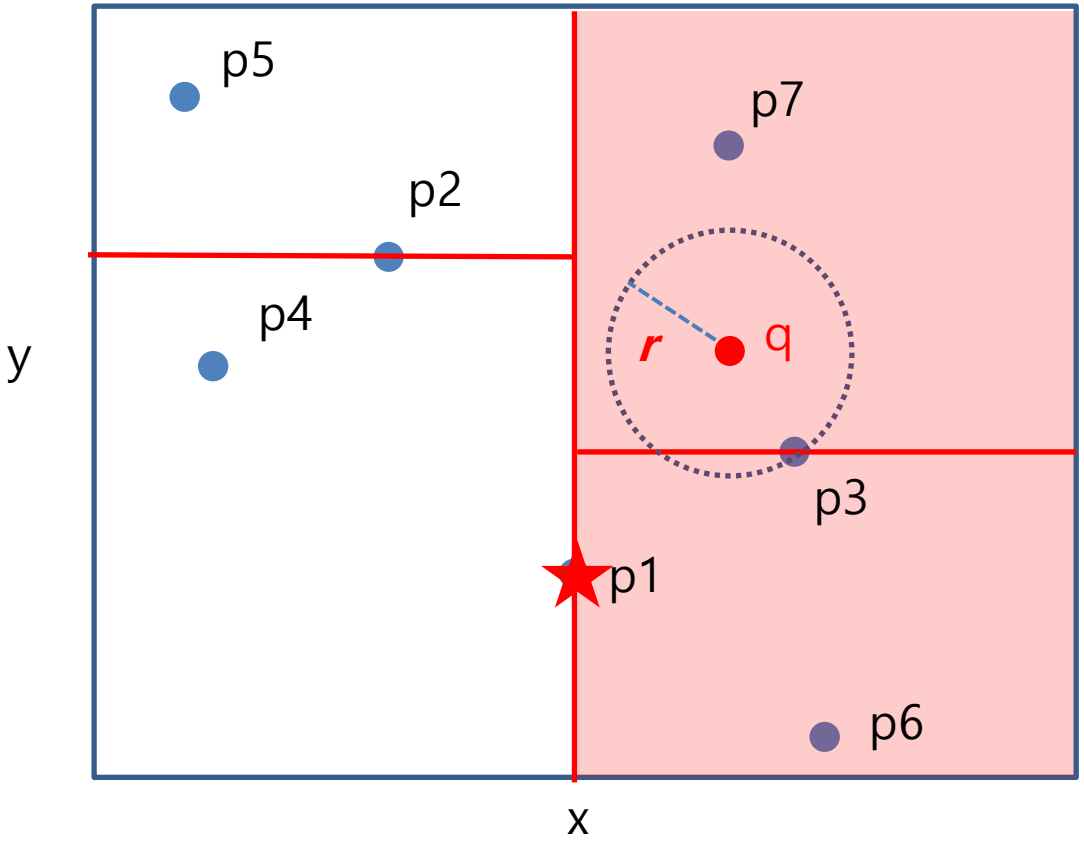
- 질의 포인트 ***q*** 와 범위 ***r*** 이 다음과 같이 주어졌다고 하자.
- 스택에 root node를 삽입한다. 최초 영역은 (min_x=0, min_y=0, max_x= ∞, max_y=∞)
- 스택에 저장된 노드를 pop하고, 질의 조건에 부합하는지 검사한다.

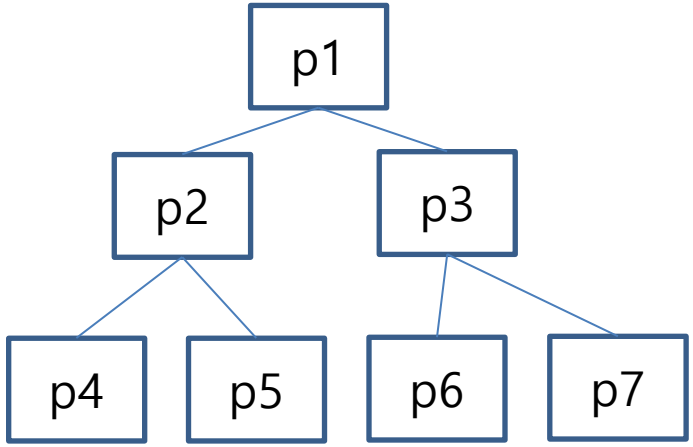




Example -Range Query using KD tree

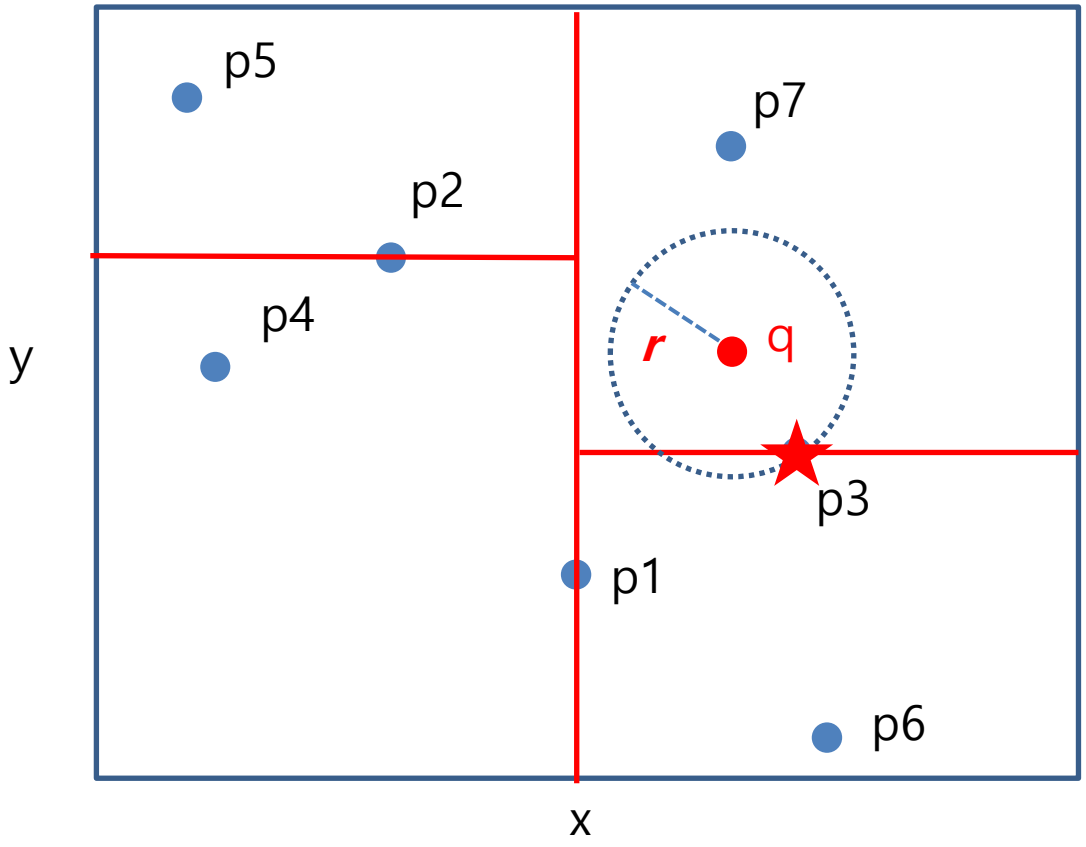
- Root에 의해 분할된 양쪽 영역이 질의 범위 안에 들어오는지 검사한다
- 왼쪽 자식이 커버하는 영역 : (min_x=0, min_y=0, max_x=p1_x, max_y=∞)
- 오른쪽 자식이 커버하는 영역 : (min_x=p1_x, min_y=0, max_x= ∞, max_y=∞)
- 질의 조건에 부합하는 자식이 있는 경우 해당 영역에 대한 Rect와 함께 stack에 삽입.

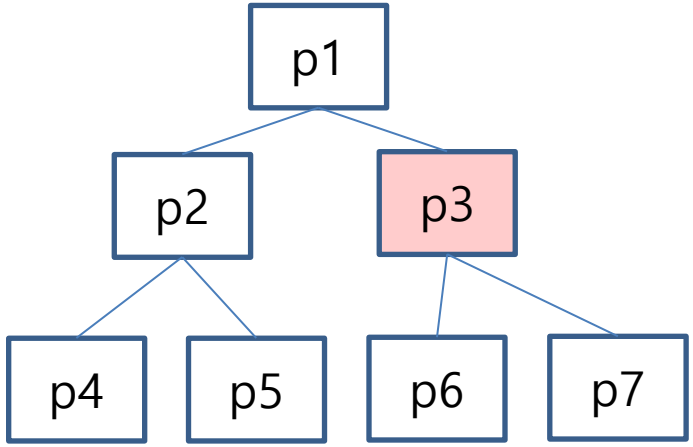




Example -Range Query using KD tree

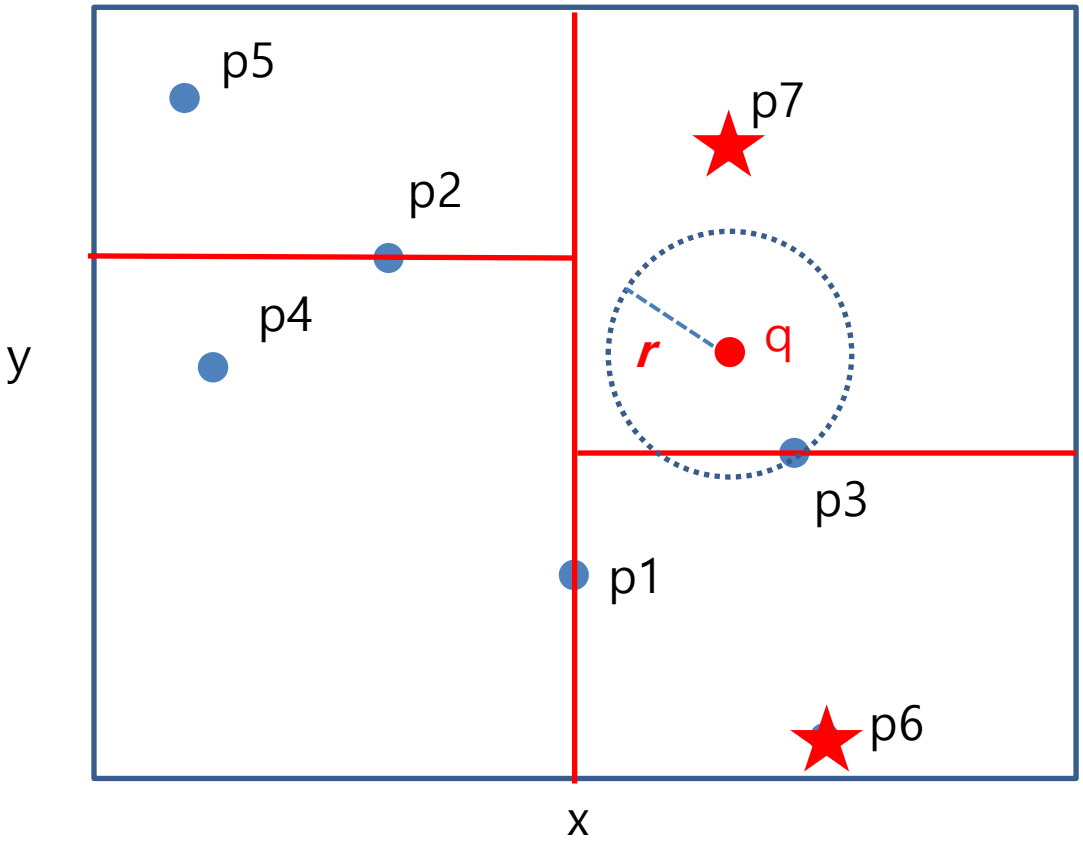
- stack이 비어있지 않다면 후보 노드를 하나 pop 시키고 질의 조건을 검사한다. 예제에서는 p3를 검사한 후 p3를 결과 집합에 포함시킨다.
- 앞선 검사와 마찬가지로, 해당 노드의 자식노드들의 영역이 질의 범위 안에 들어가는지 검사한 후 질의 조건에 부합한다면 스택에 삽입한다.

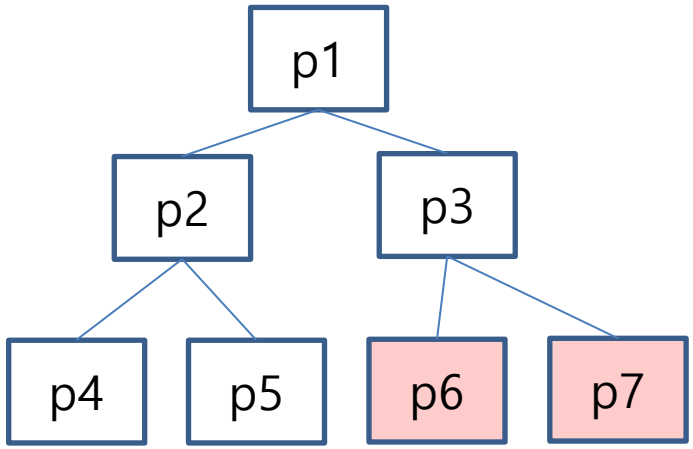




Example -Range Query using KD tree

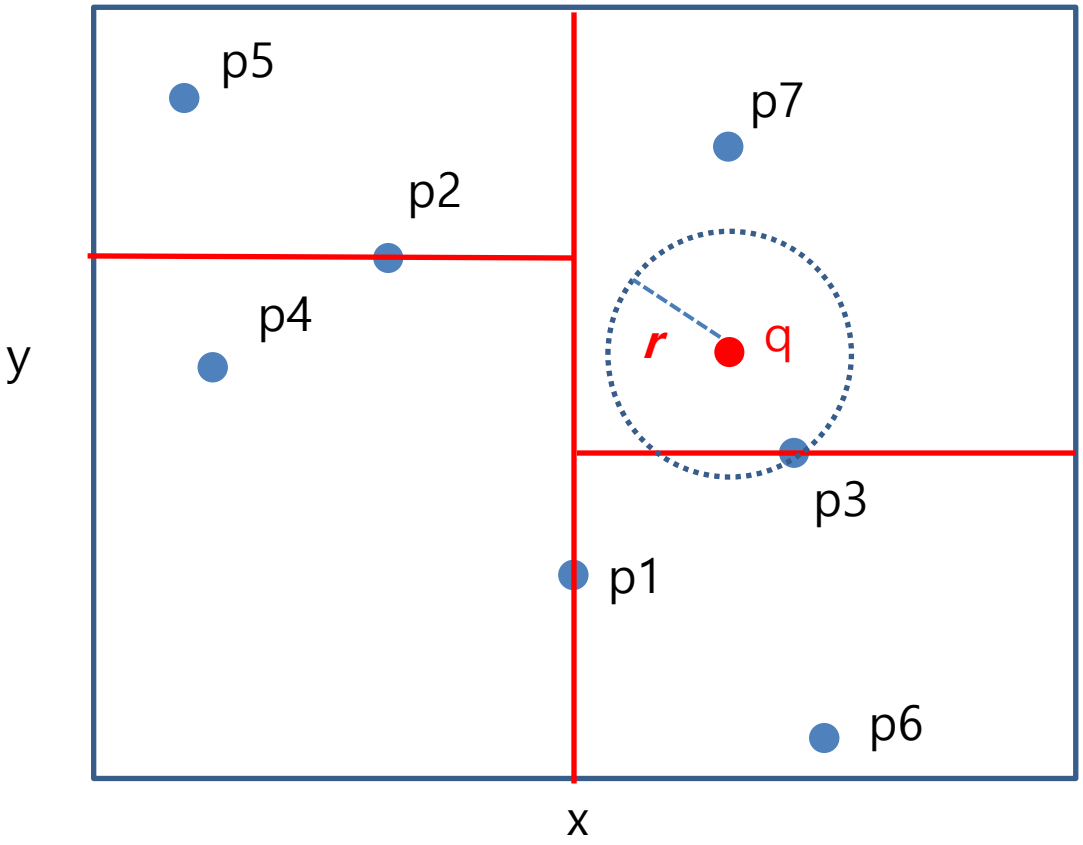
- 양쪽 영역이 전부 질의 범위 안에 포함되기 때문에 양쪽 branch를 모두 방문하고 검사함.

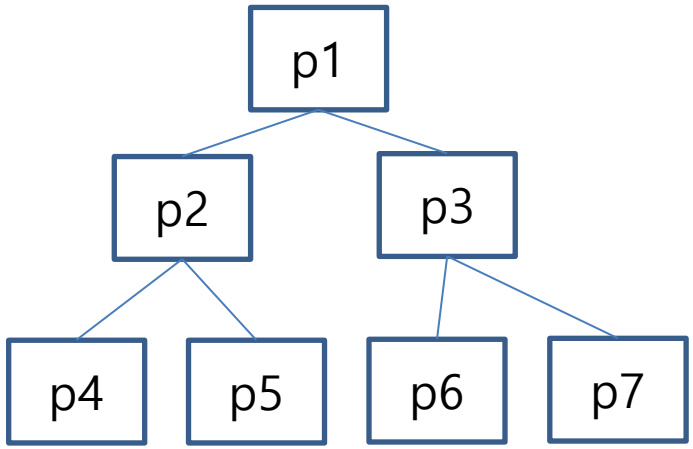




Example -Range Query using KD tree

- 두 점 모두 질의 범위에 포함되지 않으므로 결과 집합에 포함시키지 않음.
- 더 이상 탐색할 노드가 없기 때문에 알고리즘은 종료.
- p3만이 결과로 반환됨





kd Tree에서의 NN query

kNN query using Kd-tree

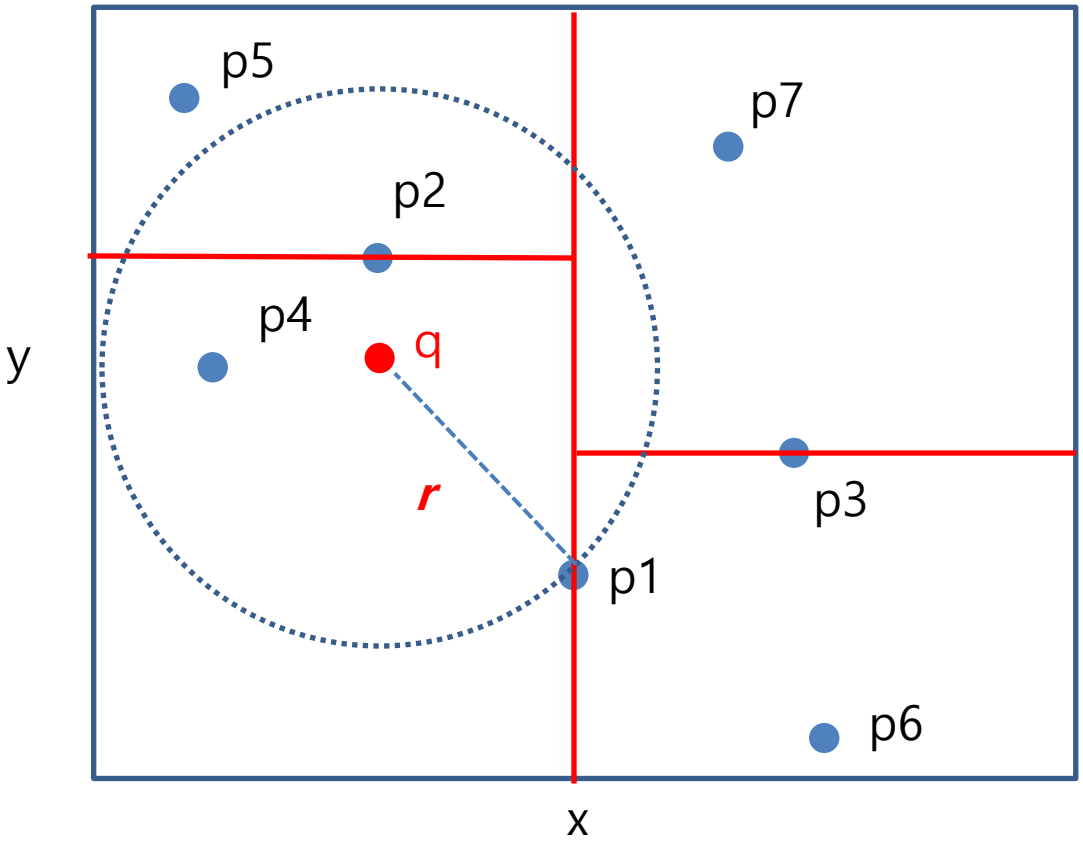
질의 조건으로 포인트 ***q*** (x,y 좌표)와 최근접 이웃의 개수 ***k*** 가 주어진다. 대략적인 질의 처리 프로세스는 다음과 같다.

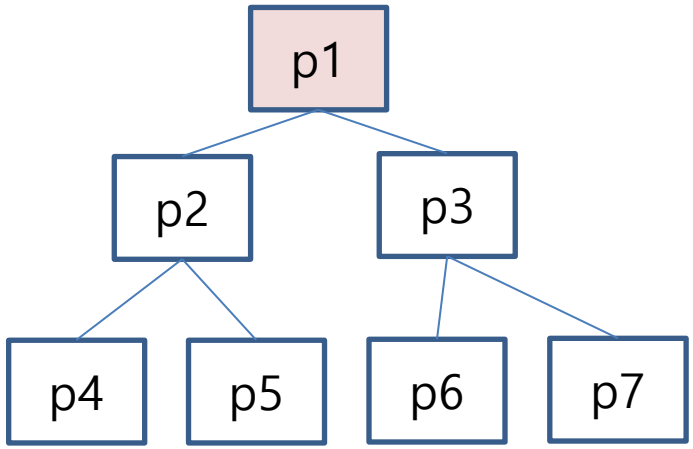
1. 루트노드로부터 탐색을 시작
2. 아주 큰 범위(***r*** =MAX_DOUBLE)를 갖는 range query로 시작한다.
3. 탐색과정에서 질의 범위 내에 포함되는 포인트를 ***k*** 개 찾은 경우, 이것을 NN의 후보로 삼고 가장 먼 포인트 까지의 거리(dist(***q***, k^{th} neighbor))로 질의 범위 ***r*** 을 업데이트한다.
4. 아직까지 탐색되지 않은 노드들은 업데이트된 질의 범위 ***r*** 을 사용해서 검사한다.
5. 지금까지 찾아낸 kNN의 후보군보다 더 가까운 포인트를 찾아냈다면, 후보군을 업데이트하고 질의 범위 r을 후보군 중 가장 먼 포인트까지의 거리로 업데이트 한다.
6. 더 이상 탐색할 노드가 없을 때 까지 3~5의 과정을 반복한다.

- kNN query의 처리 과정은 range query와 유사하다. 차이점은 범위 ***r*** 이 검색 과정에서 동적으로 업데이트 된다는 것이다.
- 두 개의 priority queue인 resultQ, nodeQ을 선언한다.
- resultQ은 kNN의 후보가 되는 오브젝트들을 질의 포인트와의 거리가 먼 순서로 정렬한다.
- nodeQ는 다음으로 탐색할 branch들을 질의 포인트와의 거리가 가까운 순서대로 정렬한다.

Example - NN Query using KD tree

- 예제에서는 **BFS** 탐색을 통한 NN 질의(***k*** =1)를 처리한다.
- ***r*** =MAX_DOUBLE인 범위 질의로 시작. 루트노드인 p1을 검사한다.
- 현재 NN의 후보가 없기 때문에 p1을 NN의 후보로 삼고 범위 질의의 범위를 p1까지의 거리로 업데이트한다.

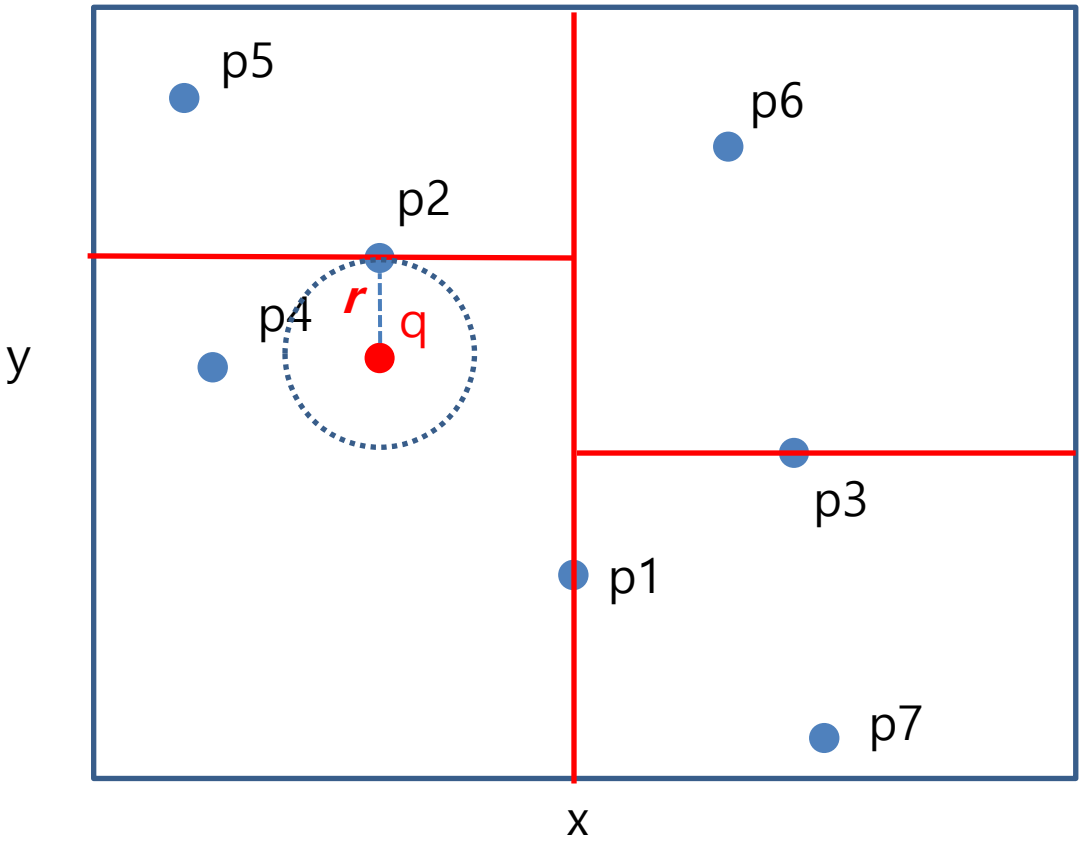


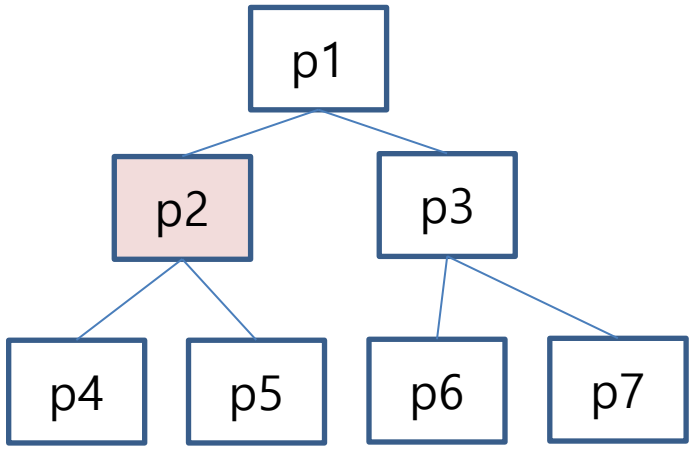


현재 NN의 후보 : p1

Example - NN Query using KD tree

- p1의 양쪽 자식노드의 영역이 모두 범위 ***r*** 내에 들어오기 때문에, 양쪽 branch를 모두 탐색한다. 두 자식 노드가 커버하는 영역과 질의 포인트간의 거리를 계산한 후 nodeQ에 삽입한다
- p2가 현재 후보보다 더 가깝기 때문에 후보는 p2로 변경. 질의 반경은 p2까지의 거리로 업데이트된다.

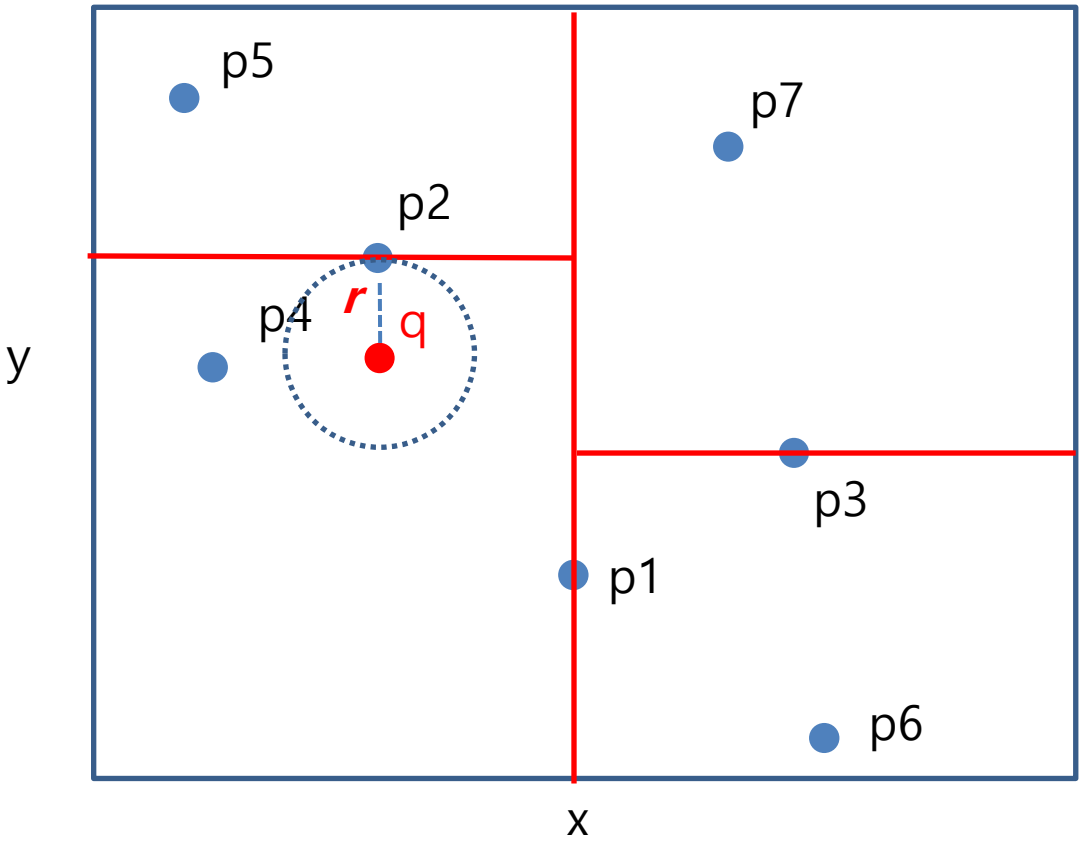


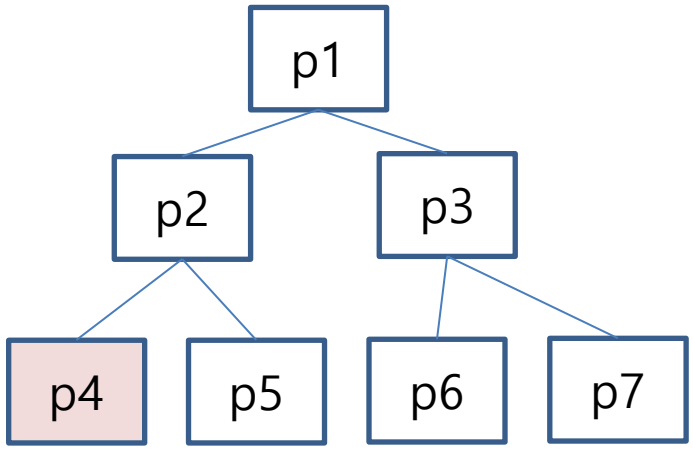


현재 NN의 후보 : p2

Example - NN Query using KD tree

- p2의 왼쪽 branch가 질의 영역안에 들어오기 때문에 p4를 검색한다. 그러나 현재 질의 범위 바깥에 있기 때문에 후보군과 질의 반경은 업데이트 되지 않는다.
- p5가 포함된 영역은 질의 반경보다 먼곳에 위치하기 때문에 탐색하지 않는다.

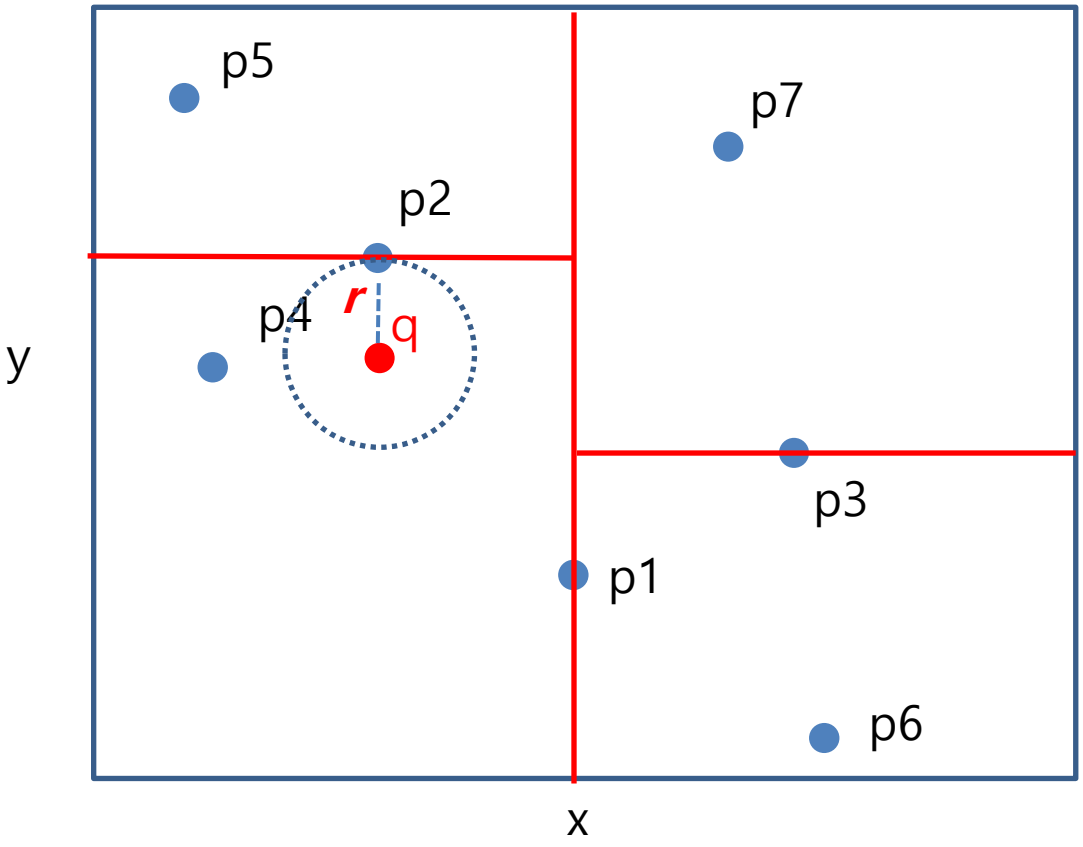


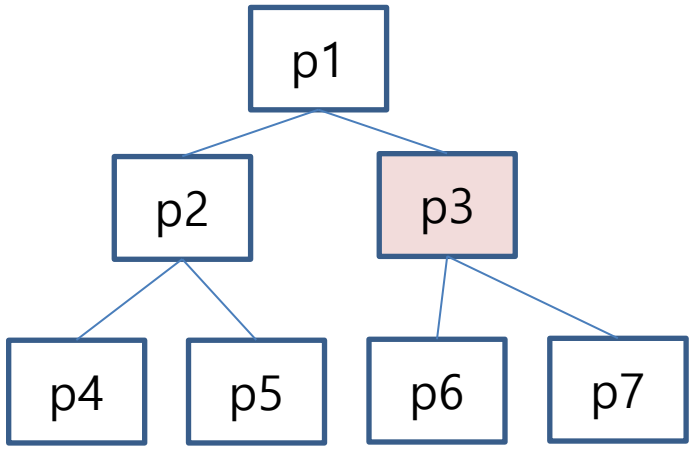


현재 NN의 후보 : p2

Example - NN Query using KD tree

- p3는 초기 단계에서는 질의 반경에 포함되었지만, 현재 업데이트 된 질의 반경보다 멀리 있기 때문에 후보군의 업데이트가 일어나지 않는다.
- p3의 두 자식 노드가 속한 영역 또한 업데이트된 질의 반경보다 멀리 있기 때문에 탐색되지 않는다.



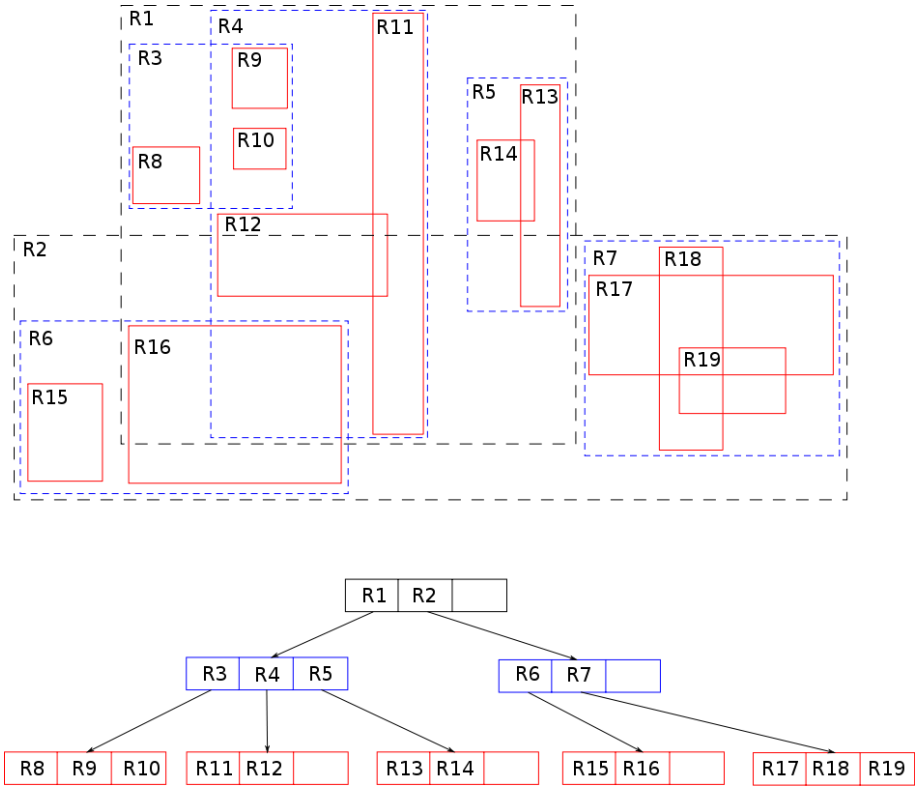


최종 NN : p2

R tree

R-tree

- R-tree는 인접한 객체를를 그룹화 하고 MBR(minimum bounding rectangle)을 사용해서 표현한다.



R-tree의 구성 요소

- **Non-leaf node(internal node)**
  - 자식노드의 MBR, 자식노드에 대한 pointer를 가짐
- **Leaf node**
  - MBR 안에 포함된 오브젝트들에 대한 pointer를 가짐

- **Split Threshold**
  - 모든 노드는 최대(M), 최소(m) 용량을 가짐
  - 노드가 가리키고 있는 object의 수, 혹은 자식 노드의 수가 M을 초과할 경우, 노드가 분할됨
  - 분할된 노드는 최소 m개의 object, 혹은 자식 노드를 가리키고 있어야 함.

Construct R tree

R tree의 노드 구조

```
typedef struct _RTREENODE{
        RTREEBRANCH branch[MAXCARD];
} RTREENODE;

typedef struct _RTREEBRANCH{
        RTREEMBR mbr;
        struct RTREENODE *child
} RTREEBRANCH;

typedef struct _RTREEMBR{
        REALTYPE bound[SIDES_NUMB]; /* xmin,ymin,...,xmax,ymax,... */
}RTREEMBR;
```

MAXCARD는 노드에 포함될 수 있는 최대 object의 개수로, 지정된 page size에 따라 계산된다(기본설정은 512).

R-tree의 Range, kNN search

- Rtree의 range query와 kNN query processing은 강의자료 참조

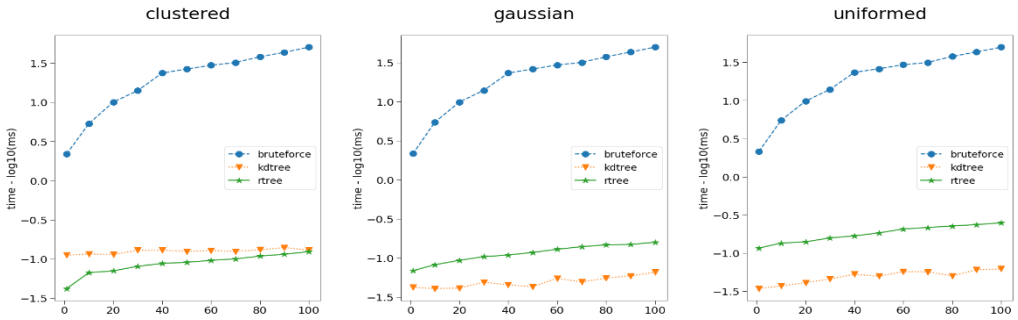
보고서 작성 예시

실험은 주어진 테스트셋에 대하여 질의 조건을 변화시켜 가며 응답시간과 검사된 오브젝트의 개수를 비교측정합니다.
범위 질의의 경우 범위를 증가시켜가며 실험 결과를 측정합니다.

- 변화시키는 범위의 크기는 축의 길이를 d라고 할때
  (d*0.01,d*0.02, d*0.04, d*0.06, d*0.08, d*0.1) 로 증가

- kNN질의의 경우 k 개수를 1부터 10씩 증가시켜가며(1~100) 실험합니다.

- 보고서에는 다음과 같은 내용이 필요합니다.
  - 각 질의 처리 구현한 함수 및 자료구조들의 내역과 기능에 대한 설명(소스코드 x)
  - 각 실험결과에 대한 분석(실험결과에 대한 그래프, 결과에 대한 이유 분석)



질문

관련 자료는 사이버캠퍼스에서 확인

- kd-tree 소스 코드
- R-tree 소스 코드
- 프로젝트 문서
- 테스트 데이터셋

담당 조교 서민준

연락처
Mail : sogang879@gmail.com
RM : AS816