과제

장유선

2023.08.24

1. 문제 정의

Quad Tree를 만들고, 주변에 존재하는 점의 개수를 측정하는 프로그램을 만들어 봅시다.

- Quad Tree

1. Root Node는 -1024, -1024, +1024, +1024의 범위를 표현합니다.

2. Quad Tree는 Insert, Get 기능을 지원하며, 2차원 Point 정보를 저장합니다 (lon, lat)

3. 각 Node는 가지고 있는 데이터가 16개를 초과 하는 경우 자식을 생성합니다.

- Test Program

1. 랜덤으로 -1024, -1024, +1024, +1024범위의 점 10,000개를 생성하여 List에 저장합니다.

2. List에 저장된 점에서 (127, 37)에서 거리가 50인 점의 개수를 출력하고 소요 시간을 측정합니다.

3. Quad Tree에 List에 저장된 점을 모두 추가하고, 동일하게 (127,37)에서 거리가 50인 점의 개수를 출력하고 소요 시간을 측정합니다

2. Quad Tree

- 자식 노드를 4개를 지니는 트리

- 2차원 데이터에 대한 Index을 수행 할 때 가장 간단하게 사용 할 수 있는 구조

- Insert

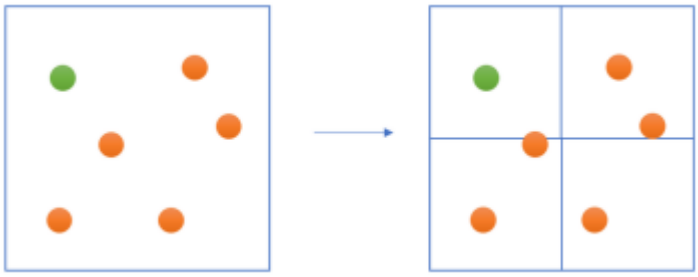
1. Root Node를 선택

2. 선택된 Node가 자식 노드를 지닌 경우

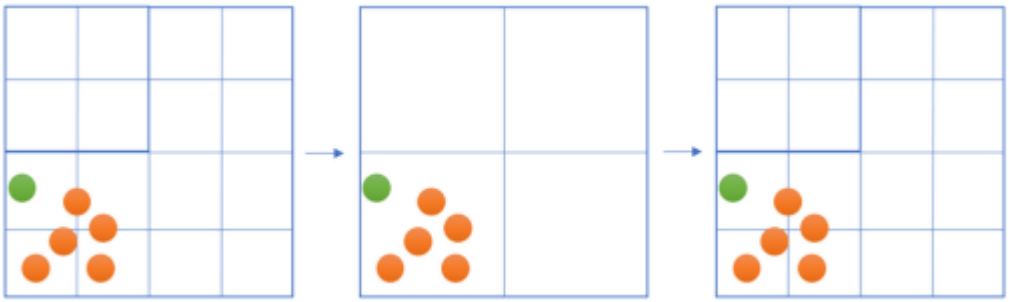
    a. 데이터가 해당되는 자식 노드를 현재 노드로 선택하고 2.로 돌아감

3. 데이터(Point)를 현재 Node에 삽입

4. 삽입 후 Node 데이터의 개수가 일정 수치 이상 인 경우, 4개의 자식 노드를 생성하고, 영역에 따라 데이터를 이동



a. 새로 생성된 자식 Node의 데이터의 개수가 일정 수치가 이상 인 경우, 현재 Node를 해당 자식 Node로 변경하고 b.로 돌아감



- GET

1. Root Node를 선택

2. 선택된 Node가 자식 노드를 지닌 경우

a. 선택할 범위에 해당하는 자식 노드를 선택하고 2.로 돌아감

3. Node 내의 데이터와 해당 값을 확인

3. Python Code Hard Copy

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rectangle:
```

```
    def __init__(self, x, y, w, h):
        self.x = x
        self.y = y
        self.w = w
        self.h = h

    def contains(self, point):
        return (self.x - self.w <= point.x <= self.x + self.w and
                self.y - self.h <= point.y <= self.y + self.h)

    def intersects(self, other):
        return not (other.x - other.w > self.x + self.w or
                    other.x + other.w < self.x - self.w or
                    other.y - other.h > self.y + self.h or
                    other.y + other.h < self.y - self.h)

# QuadTree 클래스
class QuadTree:
    def __init__(self, boundary, capacity):
        self.boundary = boundary
        self.capacity = capacity
        self.points = []
        self.divided = False

    def subdivide(self):
        x, y, w, h = self.boundary.x, self.boundary.y, self.boundary.w,
self.boundary.h
        half_w = w / 2
        half_h = h / 2
        ne = Rectangle(x + half_w, y - half_h, half_w, half_h)
        self.northeast = QuadTree(ne, self.capacity)
        nw = Rectangle(x - half_w, y - half_h, half_w, half_h)
        self.northwest = QuadTree(nw, self.capacity)
        se = Rectangle(x + half_w, y + half_h, half_w, half_h)
        self.southeast = QuadTree(se, self.capacity)
        sw = Rectangle(x - half_w, y + half_h, half_w, half_h)
        self.southwest = QuadTree(sw, self.capacity)
        self.divided = True

    def insert(self, point):
        if not self.boundary.contains(point):
            return False

        if len(self.points) < self.capacity:
            self.points.append(point)
            return True
        else:
```

```
            if not self.divided:
                self.subdivide()
            if (self.northeast.insert(point) or
                self.northwest.insert(point) or
                self.southeast.insert(point) or
                self.southwest.insert(point)):
                return True

    def get(self, range, found=None):
        """
        주어진 범위 내의 모든 점을 검색하는 메서드
        range: 검색할 범위를 나타내는 Rectangle 객체
        found: 검색된 점을 저장할 리스트 (재귀 호출 시 사용)
        """
        if found is None:
            found = []

        if not self.boundary.intersects(range):
            return found
        else:
            for p in self.points:
                if range.contains(p):
                    found.append(p)
            if self.divided:
                found = self.northwest.get(range, found)
                found = self.northeast.get(range, found)
                found = self.southwest.get(range, found)
                found = self.southeast.get(range, found)

        return found

# 두 점 사이의 유클리드 거리를 계산
def calculate_distance(point1, point2):
    return math.sqrt((point1.x - point2.x) ** 2 + (point1.y - point2.y) **
2)

import time
import random
import math

# 1. 랜덤으로 -1024, -1024, +1024, +1024 범위의 점 10,000 개 생성
points_list = [Point(random.uniform(-1024, 1024), random.uniform(-1024,
1024)) for _ in range(10000)]

# 중심점과 거리 설정
center = Point(127, 37)
radius = 50
```

```
# 2. 리스트
# 점 개수 검색 및 시간 측정
start_time_l = time.time()
count_l = 0
for point in points_list:
    if calculate_distance(center, point) <= radius:
        count_l += 1
end_time_l = time.time()
elapsed_time_l = end_time_l - start_time_l

print(f"리스트에 저장된 점의 개수: {len(points_list)}")
print(f"({center.x}, {center.y})에서 거리 {radius}인 점의 개수: {count_l}")
print(f"소요 시간: {elapsed_time_l}초")

# 3. Quad Tree
quad_tree = QuadTree(Rectangle(0, 0, 2048, 2048), 16)
# 점 삽입
for point in points_list:
    quad_tree.insert(point)

# 점 개수 검색 및 시간 측정
start_time_QT = time.time()
points_in_range = quad_tree.get(Rectangle(center.x, center.y, radius,
radius))
count_QT = len(points_in_range)
end_time_QT = time.time()
elapsed_time_QT = end_time_QT - start_time_QT
print("================================================")
print(f"쿼드트리에 삽입된 점의 개수: {len(points_list)}")
print(f"({center.x}, {center.y})에서 거리 {radius}인 점의 개수: {count_QT}")
print(f"소요시간: {elapsed_time_QT}초")
```

4. Code 설명

4-1. Point , Rectangle 클래스

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rectangle:
    def __init__(self, x, y, w, h):
        self.x = x
        self.y = y
```

```
        self.w = w
        self.h = h

    def contains(self, point):
        return (self.x - self.w <= point.x <= self.x + self.w and
                self.y - self.h <= point.y <= self.y + self.h)

    def intersects(self, other):
        return not (other.x - other.w > self.x + self.w or
                    other.x + other.w < self.x - self.w or
                    other.y - other.h > self.y + self.h or
                    other.y + other.h < self.y - self.h)
```

- Point 클래스는 하나의 점을 나타낸다. (x , y)좌표를 받아서 저장한다.
- Rectangle 클래스는 중심 x, y 좌표와 가로 w, 세로 h 크기를 받아서 사각 영역을 정의한다.
- Contains 메서드는 주어진 점이 이 사각 영역 내에 있는지를 확인하기 위해 사각 영역의 중심으로부터 가로와 세로 크기만큼 떨어진 범위 내에 점이 있으면 True를 반환한다.
- Intersects 다른 사각 영역 other와 현재 사각 영역이 교차하는지 확인하는 메서드이다.

4-2. QuadTree 클래스

```
class QuadTree:
    def __init__(self, boundary, capacity):
        self.boundary = boundary
        self.capacity = capacity
        self.points =
        self.divided = False
```

- Boundary는 현재 노드의 영역을 나타내고, capacity는 저장할 수 있는 최대 점의 개수이다.

4-3. subdivide 메서드

```
    def subdivide(self):
        x, y, w, h = self.boundary.x, self.boundary.y, self.boundary.w,
self.boundary.h
        half_w = w / 2
        half_h = h / 2
        ne = Rectangle(x + half_w, y - half_h, half_w, half_h)
        self.northeast = QuadTree(ne, self.capacity)
        nw = Rectangle(x - half_w, y - half_h, half_w, half_h)
        self.northwest = QuadTree(nw, self.capacity)
        se = Rectangle(x + half_w, y + half_h, half_w, half_h)
        self.southeast = QuadTree(se, self.capacity)
        sw = Rectangle(x - half_w, y + half_h, half_w, half_h)
```

```
        self.southwest = QuadTree(sw, self.capacity)
        self.divided = True
```

- 현재 노드를 4개의 하위 노드(ne, nw, se, sw)로 분할하는 메서드이다.
- ne노드는 현재 노드의 중심 좌표에서 half_w만큼 오른쪽으로 이동하고, half_h만큼 위로 이동한 지점에서 시작하며, 가로와 세로 길이는 half_w와 half_h이다.
- 나머지 사분면 (nw, se, sw)에 대해서도 유사하게 계산한다.

4-4. insert 메서드

```
    def insert(self, point):
        if not self.boundary.contains(point):
            return False

        if len(self.points) < self.capacity:
            self.points.append(point)
            return True
        else:
            if not self.divided:
                self.subdivide()
            if (self.northeast.insert(point) or
                self.northwest.insert(point) or
                self.southeast.insert(point) or
                self.southwest.insert(point)):
                return True
```

- 삽입하려는 point가 현재 노드의 영역 boundary 안에 있는지 확인. point가 영역 밖에 있다면 삽입을 종료하고 False를 반환합니다.
- 현재 노드에 저장된 점의 개수가 노드가 수용할 수 있는 최대 점의 개수 capacity보다 작은 경우, 해당 노드에 직접 점을 추가
- 노드에 저장된 점의 개수가 capacity를 초과하면, 노드를 분할하기 전에 먼저 self.divided 속성을 확인한다. => 노드가 아직 분할되지 않은 경우 (self.divided가 False), 노드를 분할하여 4개의 하위 노드를 생성한다. => 분할한 노드 중에서 point가 속하는 노드를 찾아 해당 하위 노드로 point를 삽입한다.

4-5. get 메서드

```
def get(self, range, found=None):
        """
        주어진 범위 내의 모든 점을 검색하는 메서드
        range: 검색할 범위를 나타내는 Rectangle 객체
        found: 검색된 점을 저장할 리스트 (재귀 호출 시 사용)
        return: 검색된 점을 포함하는 리스트
        """
```

```
        if found is None:
            found = []

        if not self.boundary.intersects(range):
            return found
        else:
            for p in self.points:
                if range.contains(p):
                    found.append(p)
            if self.divided:
                found = self.northwest.get(range, found)
                found = self.northeast.get(range, found)
                found = self.southwest.get(range, found)
                found = self.southeast.get(range, found)

        return found
```

- range: 검색할 범위

  found 검색된 점을 저장할 리스트

- 현재 노드의 영역 boundary와 주어진 검색 범위 range가 교차하는지 확인 => 교차하지 않으면 현재 노드에 속한 어떤 점도 검색 범위 내에 없으므로 검색을 종료하고 found 리스트를 반환

- 현재 노드에 저장된 점들 중에서 검색 범위 range에 속한 점들을 찾아 found 리스트에 추가 (range.contains(p)를 통해 각 점 p가 검색 범위에 속하는지 확인)

4-6. 유클리드 거리 계산

```
def calculate_distance(point1, point2):
    return math.sqrt((point1.x - point2.x) ** 2 + (point1.y - point2.y) ** 2)
```

- (point1.x - point2.x)는 두 점의 x 좌표 간의 차이

- (point1.y - point2.y)는 두 점의 y 좌표 간의 차이

- ** 2는 차이의 제곱

- math.sqrt() 함수는 제곱근을 계산하는 함수로, 두 점 사이의 직선 거리를 얻기 위해 사용

4-7. Test Program

```
import time
import random
import math

# 1. 랜덤으로 -1024, -1024, +1024, +1024 범위의 점 10,000 개 생성
points_list = [Point(random.uniform(-1024, 1024), random.uniform(-1024, 1024))
for _ in range(10000)]
```

```
# 중심점과 거리 설정
center = Point(127, 37)
radius = 50

# 2. 리스트
# 점 개수 검색 및 시간 측정
start_time_l = time.time()
count_l = 0
for point in points_list:
    if calculate_distance(center, point) <= radius:
        count_l += 1
end_time_l = time.time()
elapsed_time_l = end_time_l - start_time_l

print(f"리스트에 저장된 점의 개수: {len(points_list)}")
print(f"({center.x}, {center.y})에서 거리 {radius}인 점의 개수: {count_l}")
print(f"소요 시간: {elapsed_time_l}초")

# 3. Quad Tree
quad_tree = QuadTree(Rectangle(0, 0, 2048, 2048), 16)
# 점 삽입
for point in points_list:
    quad_tree.insert(point)

# 점 개수 검색 및 시간 측정
start_time_QT = time.time()
points_in_range = quad_tree.get(Rectangle(center.x, center.y, radius, radius))
count_QT = len(points_in_range)
end_time_QT = time.time()
elapsed_time_QT = end_time_QT - start_time_QT
print("==================================================")
print(f"쿼드트리에 삽입된 점의 개수: {len(points_list)}")
print(f"({center.x}, {center.y})에서 거리 {radius}인 점의 개수: {count_QT}")
print(f"소요시간: {elapsed_time_QT}초")
```

5. 결과

```
리스트에 저장된 점의 개수: 10000
(127, 37)에서 거리 50인 점의 개수: 16
소요 시간: 0.020941972732543945초
==================================================
쿼드트리에 삽입된 점의 개수: 10000
(127, 37)에서 거리 50인 점의 개수: 18
```

소요시간: 0.00099992752075195 31초

6. 결과 화면