6주차 : Rotating Calipers

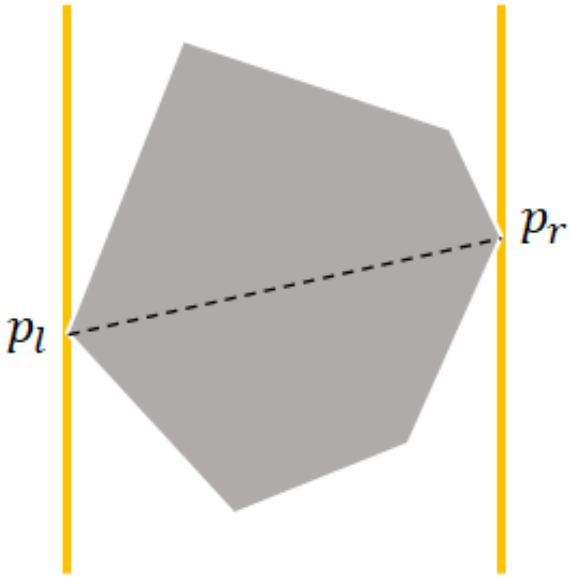
회전하는 캘리퍼스 알고리즘

> **회전하는 캘리퍼스** : 볼록 껍질(convex hull)의 최대 직경을 구하는 알고리즘.

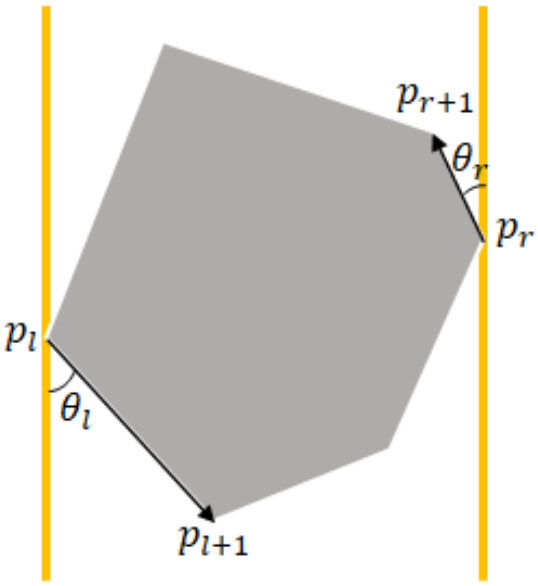
- 기본 개념 : 캘리퍼스로 무언가를 재는 과정을 알고리즘화 한 것. 평행한 두 직선 사이에 다각형을 끼우고, 다각형을 따라 두 직선을 한 바퀴 돌리며 맞닿는 꼭짓점들 사이의 거리를 구한다.
- 풀이 방법 : 가장 왼쪽에 위치한 점과 가장 오른쪽에 위치한 점에 y축에 수직인 두 직선을 각각 붙이고 시작, 볼록껍질의 점들이 나열되어있는 방향으로 어느 직선이 먼저 다각형의 다른 점을 만나는지 각도를 계산하여 두 각도 중 더 작은쪽으로 두 직선을 회전 후 닿은 점간의 거리를 계산하는 과정을 반복, 반바퀴 돌아 두 직선의 위치가 서로 바뀌면 종료. 여기서의 최대길이가 볼록껍질의 최대 직경이 된다.
- 그림 설명

---

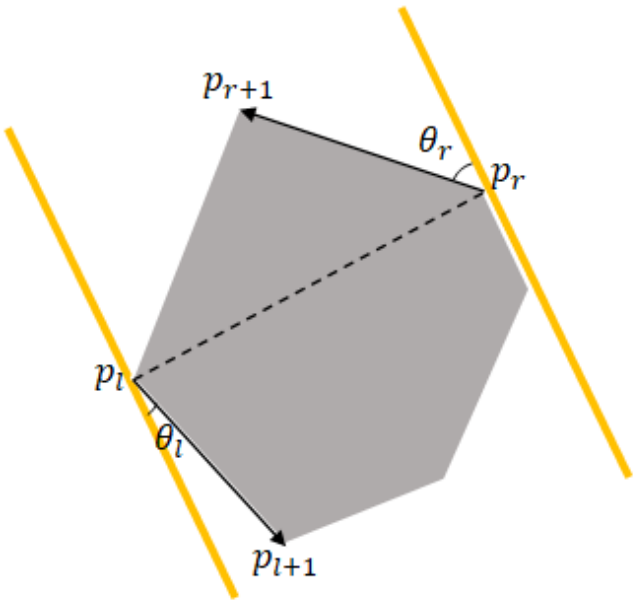
1. y축에 평행하는 두 직선을 각각을 볼록껍질의 가장 좌측, 가장 우측점에 접하도록 놓는다. 이때 두점의 거리를 계산해 최댓값(최대직경)으로 갱신한다.



2. 볼록껍질의 점들이 나열되어있는 방향(설명에서는 반시계 방향으로 가정)으로 다음 점(Pl+1,Pr+1)을 잡는다. 현재점에서 다음점으로 이동하는 선과 캘리퍼스(노란선) 간의 두 사잇각 중 크기가 작은쪽으로 탐색을 이어나간다.

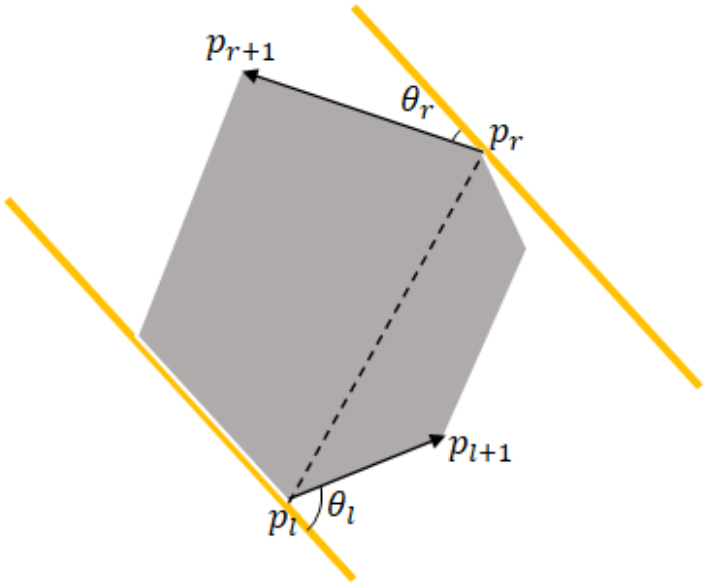


3. θl > θr 이므로 Pr+1을 Pr로 바꿔준후 캘리퍼스(노란선)을 선분PrPr+1과 평행하게 놓는다. 새롭게 지정된 Pr과 Pl의 거리를 계산해 최댓값이면 갱신하고 다음각을 비교한다.

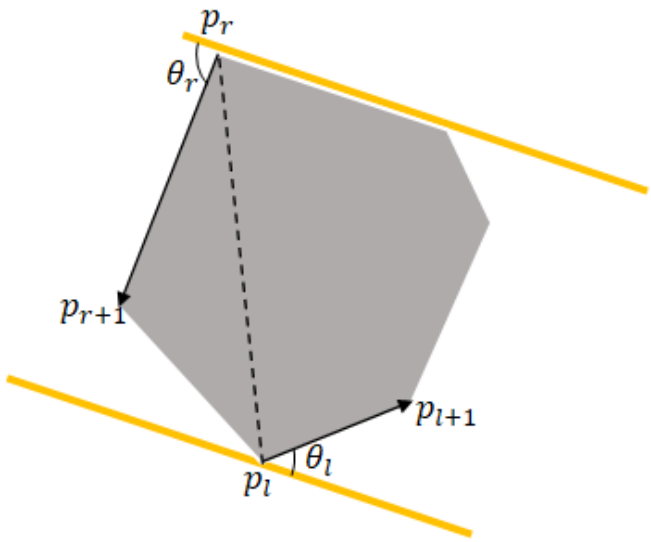


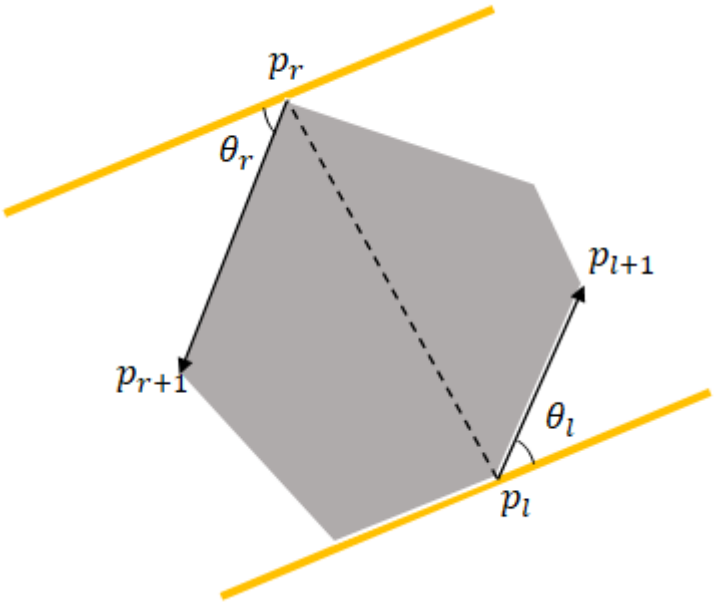
4. θr > θl 이므로 Pl+1을 Pl로 바꿔준후 캘리퍼스(노란선)을 선분PlPl+1과 평행하게 놓는다.

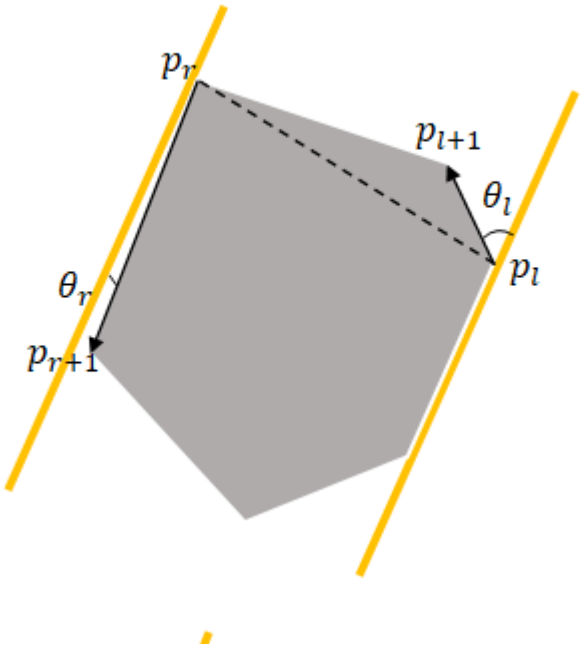
5. 새롭게 지정된 Pl과 Pr의 거리를 계산해 최댓값이면 갱신하고 다음각을 비교한다.

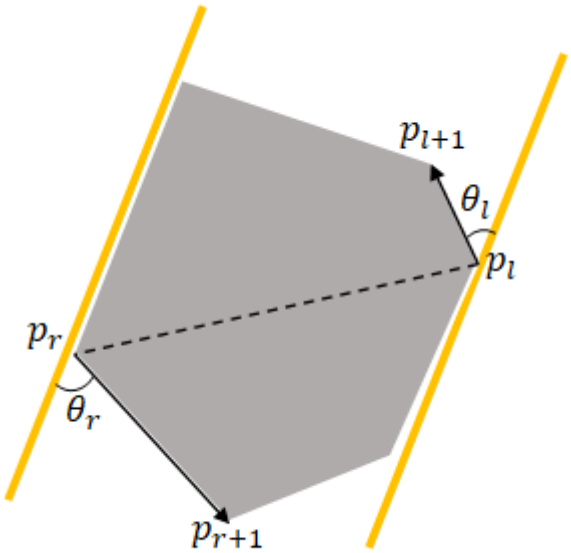


6. 같은 과정을 반복한다.


p_r
θ_r
p_{r+1}
p_{l+1}
θ_l
p_l


p_r
θ_r
p_{l+1}
p_{r+1}
θ_l
p_l





7. 최대직경이 될 수 있는 모든 후보 길이들에 대해 최댓값 갱신이 끝나면 그 때의 최댓값이 최대 직경이 된다.
8. 이렇게 최대직경을 구하면, 캘리퍼스가 다각형의 각변에 한번씩만 평행하기 때문에 O(n)의 시간복잡도가 보장된다.

예제 및 풀이

로버트 후드 **(Platinum 4)** - https://www.acmicpc.net/problem/9240

- 활쏘기 대회에서 화살을 C발 발사했고 모두 과녁에 적중했을 때, 과녁에 맞은 화살 사이의 거리 중 최댓값이 가장 큰 사람이 승리
- 과녁을 2차원 평면, 화살을 점으로 생각하여 가장 먼 화살의 거리를 구하는 프로그램을 작성하는 문제

▼ 코드

```
# 양승열
from math import dist, atan2


class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return f"({self.x}, {self.y})"


def ccw(p1, p2, p3):
    return p1[0]*p2[1] + p2[0]*p3[1] + p3[0]*p1[1] - p1[1]*p2[0] - p2[1]*p3[0] - p
3[1]*p1[0]


def monotone_chain(dots):
    dots.sort()

    for d in dots:
        while len(lower) >= 2 and ccw(lower[-2], lower[-1], d) < 0:
            lower.pop()
        lower.append(d)

    for d in reversed(dots):
        while len(upper) >= 2 and ccw(upper[-2], upper[-1], d) < 0:
            upper.pop()
        upper.append(d)

    return lower[:-1] + upper[:-1]


def rotating_calipers():
    # x가 최소, 최대인 점을 각각 찾는다.
    pl = 0
    pr = 0
    for p in range(length):
        if convex_hull[pl][0] > convex_hull[p][0]:
            pl = p
        if convex_hull[pr][0] < convex_hull[p][0]:
            pr = p

    # calipers 벡터를 y축에 평행 시킨다.
```

```
    calipers = Point(0, 1)

    # 현재 두 점의 거리 계산
    ret = dist(convex_hull[pl], convex_hull[pr])

    # calipers 와 이루는 각 계산
    for _ in range(length):
        al = atan2(calipers.y - (convex_hull[pl][1] - convex_hull[(pl + 1) % lengt
h][1]), calipers.x - (convex_hull[pl][0] - convex_hull[(pl + 1) % length][0]))
        ar = atan2(calipers.y - (convex_hull[(pr + 1) % length][1] - convex_hull[p
r][1]), calipers.x - (convex_hull[(pr + 1) % length][0] - convex_hull[pr][0]))

        if al < ar:
            calipers.x = convex_hull[pl][0] - convex_hull[(pl + 1) % length][0]
            calipers.y = convex_hull[pl][1] - convex_hull[(pl + 1) % length][1]
            pl = (pl + 1) % length
        else:
            calipers.x = convex_hull[(pr + 1) % length][0] - convex_hull[pr][0]
            calipers.y = convex_hull[(pr + 1) % length][1] - convex_hull[pr][1]
            pr = (pr + 1) % length

        # 두 점의 거리 최댓값 갱신
        ret = max(ret, dist(convex_hull[pl], convex_hull[pr]))

    return ret


points = []
lower = []
upper = []

n = int(input())
for i in range(n):
    x, y = map(int, input().split())
    points.append((x, y))

convex_hull = monotone_chain(points)
length = len(convex_hull)

print(rotating_calipers())
```

- 시간복잡도
    - `O(nlogn)`

추가 문제 (선택적 풀이)

- 고속도로 (Platinum **2**) - https://www.acmicpc.net/problem/10254

참고

- https://aruz.tistory.com/entry/rotating-calipers
- https://hellogaon.tistory.com/40