5주차 : Convex Hull

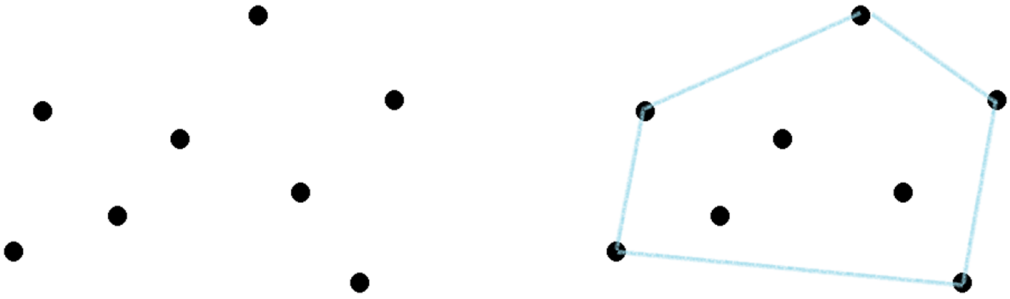
볼록 껍질 알고리즘(Convex Hull Algorithm)

```
convex: 볼록한, hull: 겉껍질
다양한 객체에 볼록 껍질을 만드는 알고리즘
```

간단히 말해서

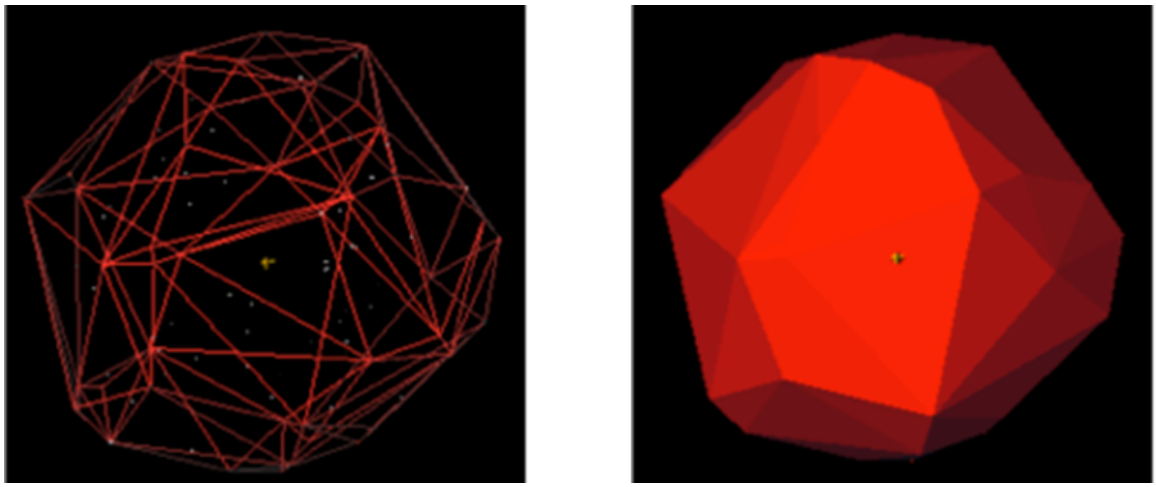
공간 상에서 모든 점들을 감쌀 수 있는 껍질을 찾는 알고리즘

▼ 평면 상에서의 예시



Source: https://www.crocus.co.kr/1288

▼ 공간 상에서의 예시



Source: https://en.wikipedia.org/wiki/Convex_hull

여기서 말하는 볼록이란?

- 평면 상에서, 볼록 다각형은 “경계의 두 점을 잇는 어떤 선분도 다각형 외부로 나가지 않는 단순 다각형(자기교차 하지 않는 것)이다.”
    - 단순하게 생각하면, 어떤 꼭지점의 각도 180도를 초과하지 않는 다각형

Convex Hull Algorithm의 종류

▼ Convex hull algorithm(이하 컨벡스헐)의 종류는 다양하게 존재한다.

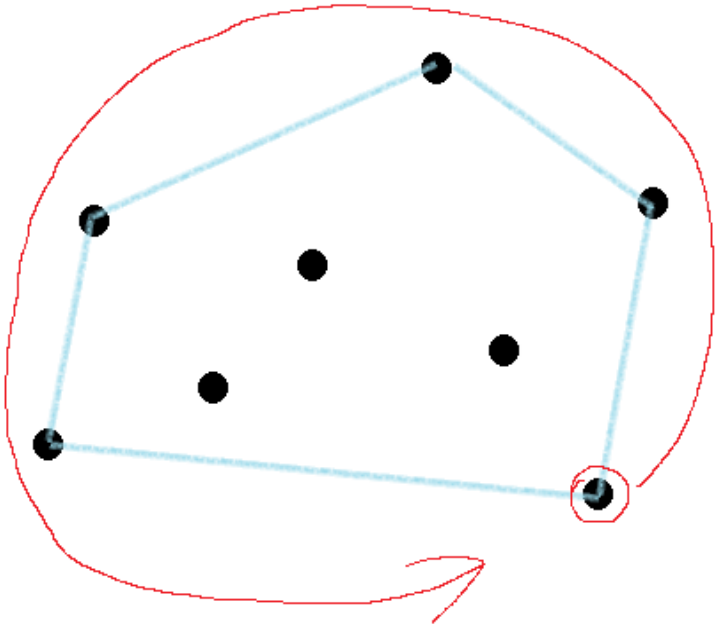
- Graham scan
- Quickhull
- Gift wrapping algorithm or Jarvis march
- Chan's algorithm
- Kirkpatrick–Seidel algorithm

여기서는 **Graham scan**을 기준으로 설명한다.

Graham Scan

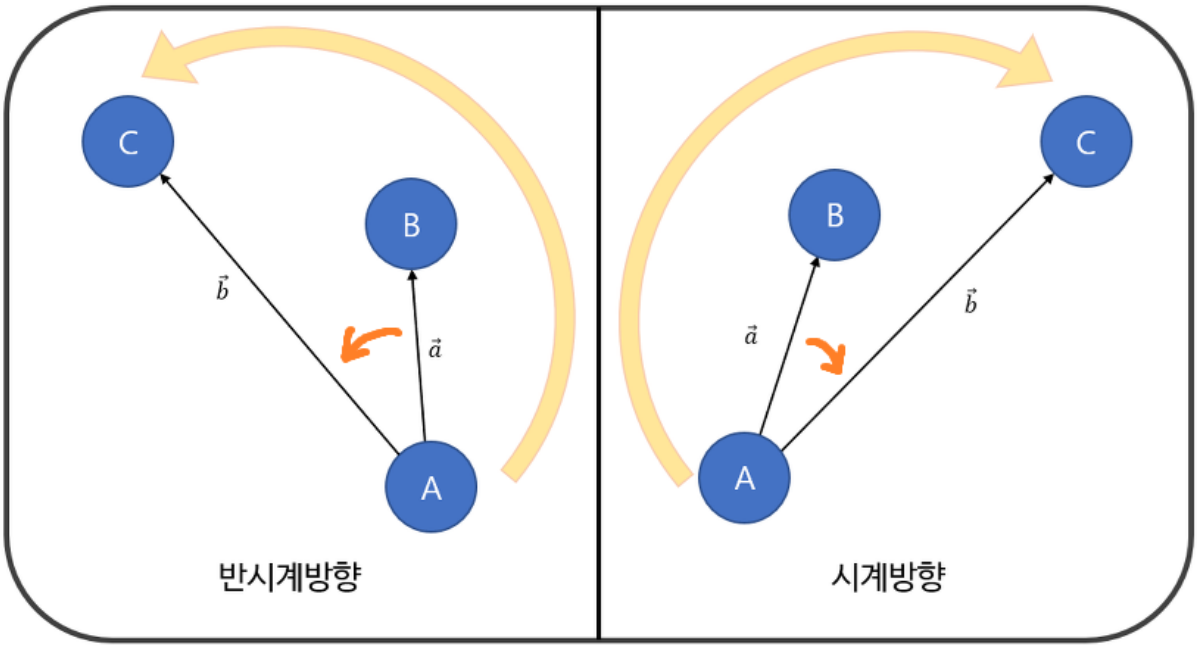
Key Idea

평면 볼록 껍질의 한 꼭짓점에서, 시계방향 혹은 반시계방향으로 순회할 수 있다.
반대로, 한 방향으로 순회할 수 있다면 이는 볼록하다.
모든 점을 포함하는 볼록 껍질은 오로지 하나 존재한다.



시계 방향? 반시계 방향?

두 점을 기준으로 다른 점의 방향을 시계방향 혹은 반시계 방향으로 판단할 수 있다.



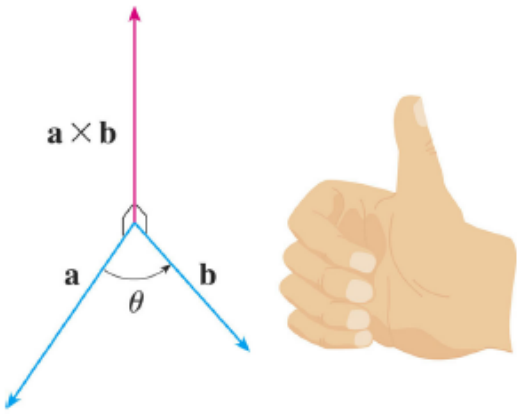
Source: https://degurii.tistory.com/47

세 점의 방향성을 판단하는 알고리즘, CCW Algorithm

세 점의 방향성은 두 점 A, B를 잇는 벡터 ***a***와 두 점 A, C를 잇는 벡터 ***b***의 외적을 통해 판단할 수 있다.

▼ 외적에 대한 기본 내용

- 외적을 취했을 때 방향성은 오른손 법칙을 따른다.



- 외적의 기본 수식은 다음과 같다.

$$\vec{u} \times \vec{v} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{vmatrix} = \begin{vmatrix} m_2 & m_3 \\ n_2 & n_3 \end{vmatrix} \hat{i} - \begin{vmatrix} m_1 & m_3 \\ n_1 & n_3 \end{vmatrix} \hat{j} + \begin{vmatrix} m_1 & m_2 \\ n_1 & n_2 \end{vmatrix} \hat{k}$$

$$= (m_2n_3 - m_3n_2,\ m_3n_1 - m_1n_3,\ m_1n_2 - m_2n_1)$$

외적에 대한 기본 내용을 참고했을 때, 세 점의 좌표를

$$A = (x_1,\ y_1,\ 0), B = (x_2,\ y_2,\ 0), C = (x_3,\ y_3,\ 0)$$

라고 한다면, 벡터 ***a***와 벡터 ***b***는 다음과 같이 둘 수 있다.

$$\vec{a} = \vec{AB} = (x_2 - x_1,\ y_2 - y_1,\ 0),\ \vec{b} = \vec{AC} = (x_3 - x_1,\ y_3 - y_1,\ 0)$$

이를 외적하면

$$\vec{a} \times \vec{b} = (0,\ 0,\ (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)$$

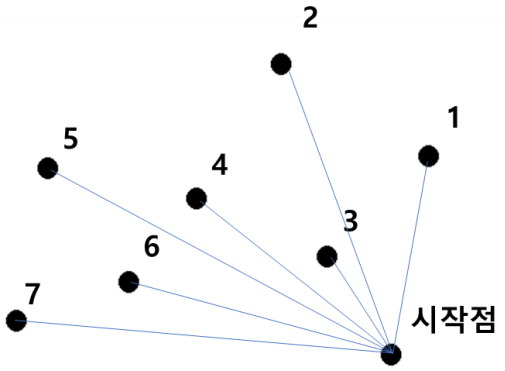
$$= x_1y_2 + x_2y_3 + x_3y_1 - (y_1x_2 + y_2x_3 + y_3x_1)$$

와 같이 나오며, 오른손법칙에 따라 해당 식이 양수면 반시계방향, 음수면 시계방향이다.

Graham Scan

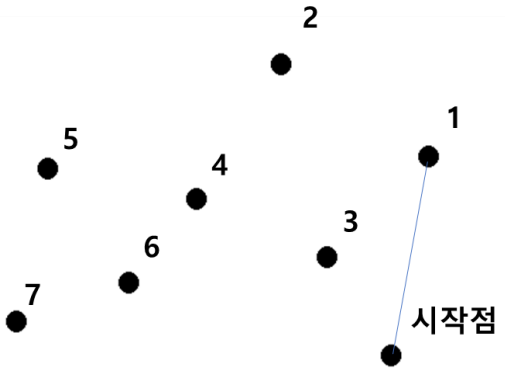
~~여기까지 봤으면 그냥 여기 보는게 이해가 빠름~~

1. y 좌표가 가장 낮은 점을 찾는다. 이를 시작점이라 부른다.
2. 시작점으로부터 반시계방향으로 다른 점들을 정렬한다. 더 정확하게는 x축과, 시작점과 해당 점을 이은 선이 이루는 각을 오름차순으로 정렬한다.

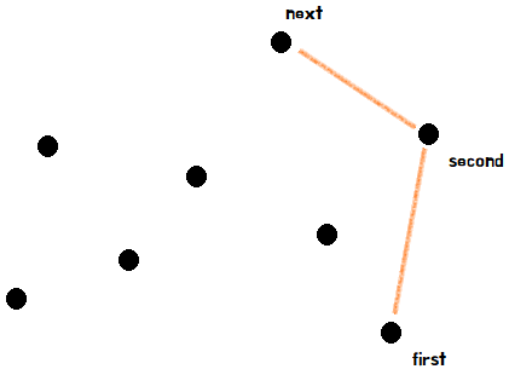


   앞으로는 이 정렬된 순서가 기본적으로 **볼록 껍질이라고 가정하고** 다음 정점이 반시계 방향이라면 받아들이고, 아니라면 한 정점 전에서 다시 판단한다.

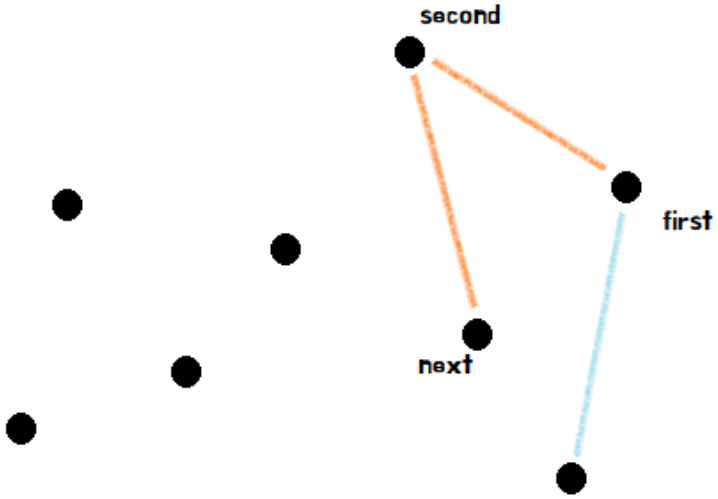
3. 시작점과 1번 점을 기본으로 두고 시작한다.



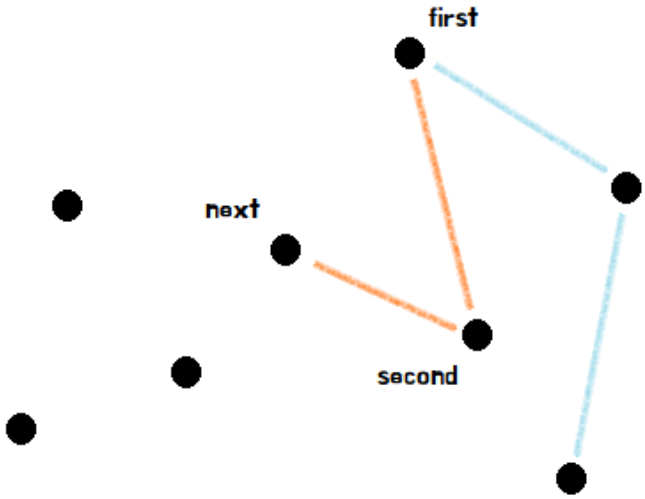
4. 다음 점을 찾은 후 가장 최근에 확인한 두 점을 기준으로 다음 점이 반시계방향인지 판단한다.

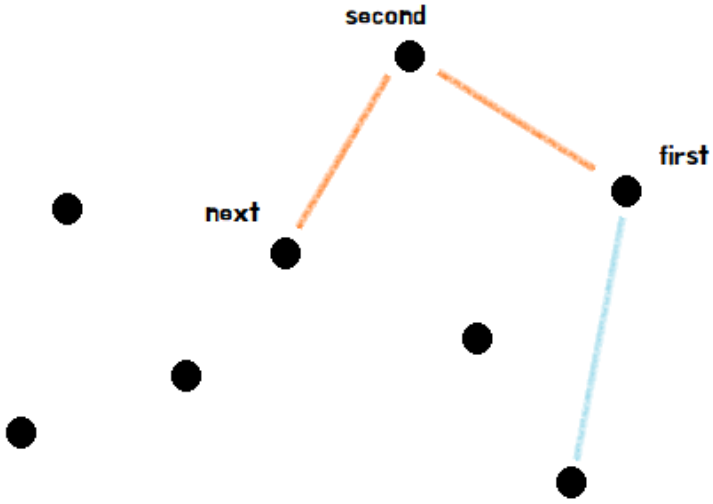


4-1. 반시계방향이라면 받아들이고, 반복한다.



4-2. 반시계방향이 아니라면 버리고 하나 전 정점으로 돌아간 후 반복한다.





시간복잡도

Graham Scan의 시간 복잡도는 `O(n log n)`이다. 사실 이는 정렬 알고리즘의 시간 복잡도와 같으며, 정점을 순회하는데에는 `O(n)`의 시간 복잡도를 가진다.

Monotone Chain

https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain

문제 풀어보기

▼ 볼록 껍질 (Platinum 5) - https://www.acmicpc.net/problem/1708

```
# https://www.acmicpc.net/source/35758060
import sys
input = sys.stdin.readline


def ccw(x1, y1, x2, y2, x3, y3):
    return (x2-x1)*(y3-y1)-(x3-x1)*(y2-y1)


def compare(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    c = ccw(x0, y0, x1, y1, x2, y2)
    if c != 0:
        return c < 0
    else:
        if x1 == x2:
            return y1 > y2
        else:
            return x1 > x2


def asort(p):
    if len(p) <= 1:
        return p

    pivot = p[0]
    left = [x for x in p[1:] if compare(pivot, x)]
    right = [x for x in p[1:] if not compare(pivot, x)]
    return asort(left) + [pivot] + asort(right)


def convex(p):
    s = [(x0, y0)]
    i = 0
    for i in range(n-1):
        while len(s) >= 2 and ccw(*s[-2], *s[-1], *p[i]) <= 0:
            s.pop()
        s.append(p[i])
    if not ccw(*s[-2], *s[-1], x0, y0):
        s.pop()
    return s


n = int(input())

p = []
x0, y0 = int(1e9), 0
for _ in range(n):
    x, y = map(int, input().split())
    if x0 > x:
        if x0 != int(1e9):
            p.append((x0, y0))
        x0, y0 = x, y
    elif x0 == x and y0 > y:
        p.append((x0, y0))
        y0 = y
    else:
        p.append((x, y))
```

```
p = asort(p)
s = convex(p)
print(len(s))

# --------------------
# 이호형
import sys


def CCW(v1, v2, v3):
    (x1, y1), (x2, y2), (x3, y3) = v1, v2, v3
    return x1*y2 + x2*y3 + x3*y1 - y1*x2 - y2*x3 - y3*x1


def delete_grad(ver_tuple):
    ver_grad, ver_x, ver_y = ver_tuple
    return ver_x, ver_y


def distance(a, b):
    return abs(a[0]-b[0]) ** 2 + abs(a[1]-b[1]) ** 2


def larger_distance(a, b, c):
    return b if distance(a, b) > distance(a, c) else c


N = int(sys.stdin.readline())
vertexes = []
# y좌표가 최소인 정점을 찾는다. 여러 개라면 x좌표가 최대인 정점을 선택한다.
max_x, min_y = -40001, 40001
for _ in range(N):
    x, y = map(int, sys.stdin.readline().split())
    if y > min_y:
        vertexes.append((x, y))
    elif y == min_y:
        if x > max_x:
            vertexes.append((max_x, min_y))
            max_x = x
        else:
            vertexes.append((x, y))
    else:
        vertexes.append((max_x, min_y))
        max_x = x
        min_y = y

# 각도에 따라 정렬한다. 시작점으로부터 기울기로 판단한다.
# 기울기의 경우에는 0~inf -inf~0 순으로 변하므로 양수와 음수를 나눠서 정렬한다.
gradient_vers_pos = []
gradient_vers_neg = []
gradient_vers_inf = []
for i in range(1, N):
    x, y = vertexes[i]
    dx = x - max_x
    dy = y - min_y
    if dx == 0:
        gradient_vers_inf.append((None, x, y))
        continue
    gradient = dy / dx
    if gradient > 0:
        gradient_vers_pos.append((gradient, x, y))
    else:
        gradient_vers_neg.append((gradient, x, y))
gradient_vers_pos.sort(reverse=True)
```

```
gradient_vers_neg.sort(reverse=True)
gradient_vers = [(None, max_x, min_y)] + gradient_vers_neg + gradient_vers_inf + gradient_ver
s_pos

# Graham scan
stack = [(max_x, min_y), delete_grad(gradient_vers.pop())]
while gradient_vers:
    first = stack[-2]
    second = stack[-1]
    next_ver = delete_grad(gradient_vers[-1])
    CCW_value = CCW(first, second, next_ver)
    if CCW_value > 0:
        gradient_vers.pop()
        stack.append(next_ver)
    elif CCW_value == 0:
        gradient_vers.pop()
        stack[-1] = larger_distance(first, second, next_ver)
    else:
        stack.pop()

print(len(stack)-1)
```

추가 문제

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

▼ 코드

```
```

참고

- Convex hull의 정의_위키피디아(en)
- Graham Scan_위키피디아
- Graham Scan에 대한 기본적인 이해
- CCW 알고리즘
- Monotone Chain_위키(en)
- Monotone Chain 간단설명