2023-여름방학 스터디 4주차

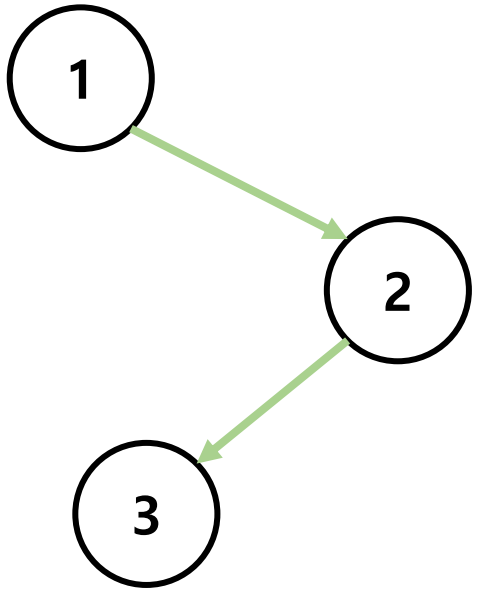
CCW, Convex Hull, 회전하는 캘리퍼스

2016133 이유진

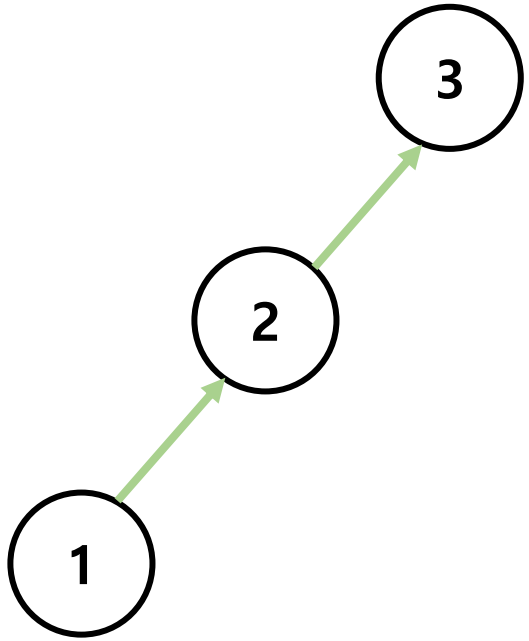
CCW(Counterclockwise) algorithm

- Counterclockwise : 시계 반대 방향
- **세 개의 점을 이은 선의 방향성**을 알 수 있는 알고리즘

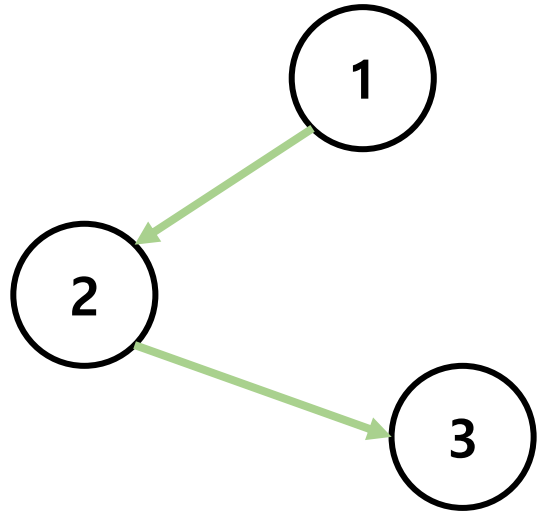
1) 시계 방향



2) 평행(일직선)

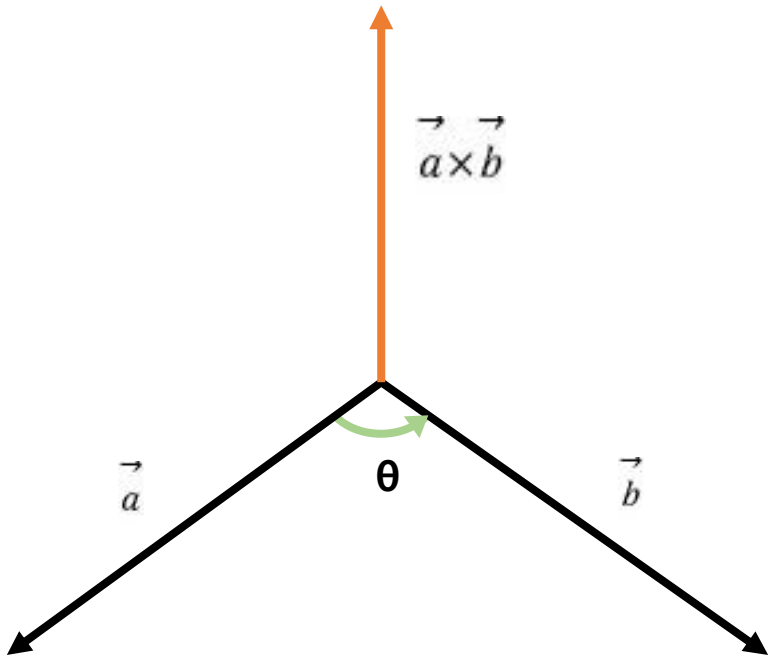


3) 반시계 방향



CCW(Counterclockwise) algorithm

- 3차원 좌표계에서 두 벡터의 외적 (cross product) 이용 : $\vec{a} \times \vec{b}$



두 벡터 $\vec{a} = \langle a_1, a_2, a_3 \rangle$, $\vec{b} = \langle b_1, b_2, b_3 \rangle$ 의 외적은

$$\vec{a} \times \vec{b} = \langle a_2 b_3 - a_3 b_2, a_3 b_1 - a_1 b_3, a_1 b_2 - a_2 b_1 \rangle$$

외적 https://blog.naver.com/mindo1103/90103361104

CCW(Counterclockwise) algorithm

참고!

- 추가) 행렬식을 이용하는 경우 : $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$ (2×2 행렬의 행렬식)

3x3 행렬 $A = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$ 의 행렬식은 $\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}$

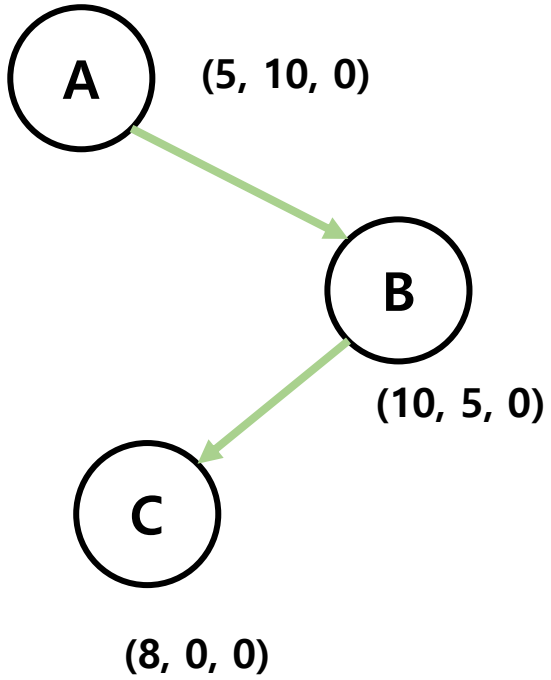
→ 단위벡터 $\vec{i} = \langle 1,0,0 \rangle$, $\vec{j} = \langle 0,1,0 \rangle$, $\vec{k} = \langle 0,0,1 \rangle$ 이용하면 외적은

$\vec{a} \times \vec{b} = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \vec{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \vec{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \vec{k}$, 즉 $\vec{a} \times \vec{b} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$ 와 같음!

외적 https://blog.naver.com/mindo1103/90103361104

CCW(Counterclockwise) algorithm

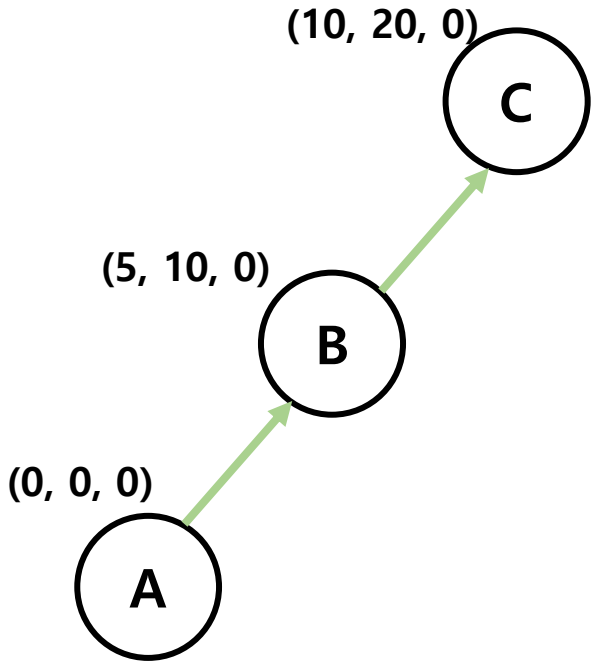
- 시계 방향 (clockwise)



  - 2차원 상의 점이므로 z값을 0으로 두고 계산
    - $\overrightarrow{AB}$ = (10-5, 5-10, 0-0) = (5, -5, 0)
    - $\overrightarrow{BC}$ = (8-10, 0-5, 0-0) = (-2, -5, 0)
  - $\overrightarrow{AB}$와 $\overrightarrow{BC}$의 외적 계산 : $\vec{a}\times\vec{b}=\langle a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1\rangle$
    - 위 식에서 a_3, b_3이 0이므로 (0, 0, a_1b_2-a_2b_1)
    - $\overrightarrow{AB}$ x $\overrightarrow{BC}$ = (0, 0, 5*(-5)-(-5)*(-2)) = (0, 0, -35)
  - 시계 방향의 경우 외적의 결과가 음수

CCW(Counterclockwise) algorithm

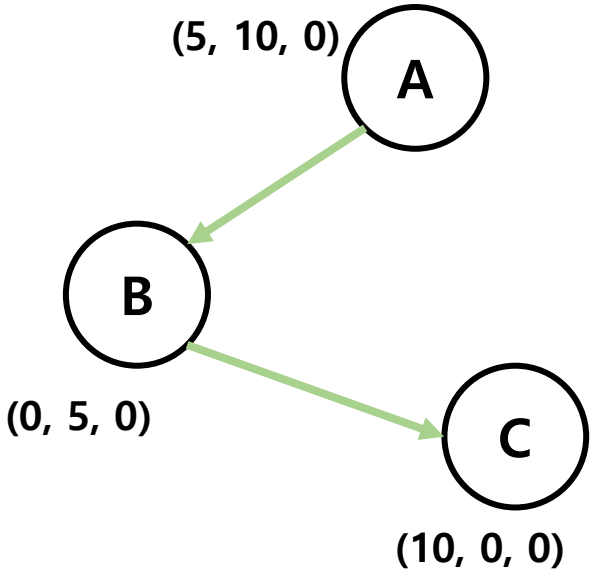
- 일직선 (collinear)



  - 2차원 상의 점이므로 z값을 0으로 두고 계산
    - $\overrightarrow{AB}$ = (5-0, 10-0, 0-0) = (5, 10, 0)
    - $\overrightarrow{BC}$ = (10-5, 20-10, 0-0) = (5, 10, 0)
  - $\overrightarrow{AB}$와 $\overrightarrow{BC}$의 외적 계산 : $\vec{a}\times\vec{b}=\langle a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1\rangle$
    - 위 식에서 a_3, b_3이 0이므로 (0, 0, a_1b_2-a_2b_1)
    - $\overrightarrow{AB}$ x $\overrightarrow{BC}$ = (0, 0, 5*10-5*10) = (0, 0, 0)
  - 일직선의 경우 외적의 결과가 0

CCW(Counterclockwise) algorithm

- 반시계 방향 (counterclockwise)



  - 2차원 상의 점이므로 z값을 0으로 두고 계산
    - $\overrightarrow{AB}$ = (0-5, 5-10, 0-0) = (-5, -5, 0)
    - $\overrightarrow{BC}$ = (10-0, 0-5, 0-0) = (10, -5, 0)
  - $\overrightarrow{AB}$와 $\overrightarrow{BC}$의 외적 계산 : $\vec{a}\times\vec{b}=\langle a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1\rangle$
    - 위 식에서 a_3, b_3이 0이므로 (0, 0, a_1b_2-a_2b_1)
    - $\overrightarrow{AB}$ x $\overrightarrow{BC}$ = (0, 0, (-5)*(-5)-(-5)*10) = (0, 0, 75)
  - 반시계 방향의 경우 외적의 결과가 양수

CCW(Counterclockwise) algorithm code

```cpp
#include <iostream>
using namespace std;

struct Point {
   private:
   public:
    int x;
    int y;
};

int CCW(int x1, int y1, int x2, int y2, int x3, int y3) {
    return (x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1);
}
```

```cpp
int main() {
    Point P1, P2, P3;

    cin >> P1.x >> P1.y;
    cin >> P2.x >> P2.y;
    cin >> P3.x >> P3.y;

    int result = CCW(P1.x, P1.y, P2.x, P2.y, P3.x, P3.y);

    if (result > 0)
        cout << "counterclockwise";
    else if (result < 0)
        cout << "clockwise";
    else
        cout << "collinear";

    return 0;
}
```

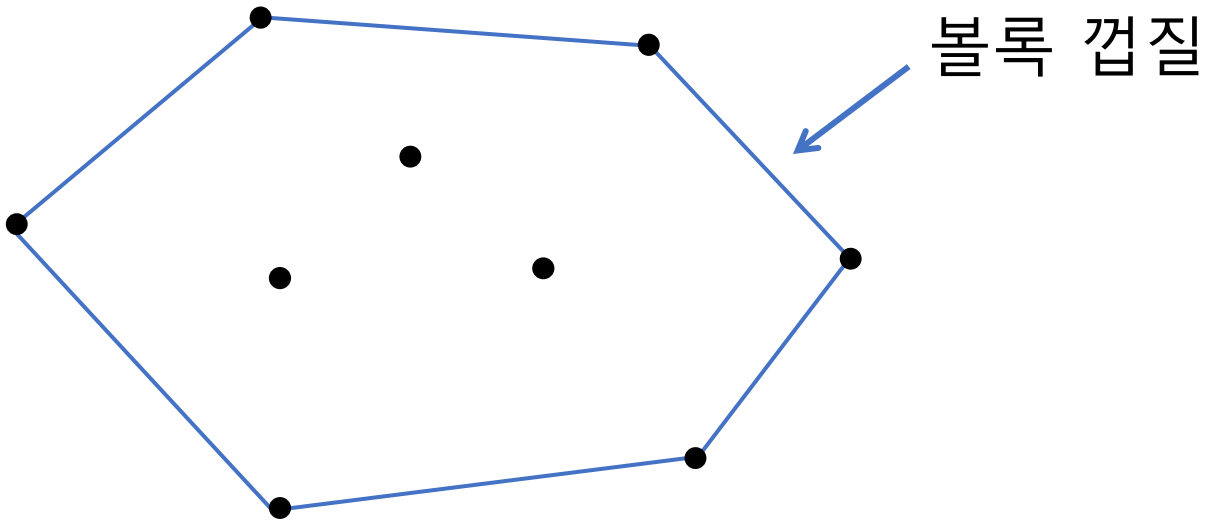
https://snowfleur.tistory.com/98

Convex Hull algorithm

- Convex hull : 2차원 평면 상에 여러 점이 있을 때

  1) 그 점 중 일부를 이용해 볼록한 다각형을 만들되

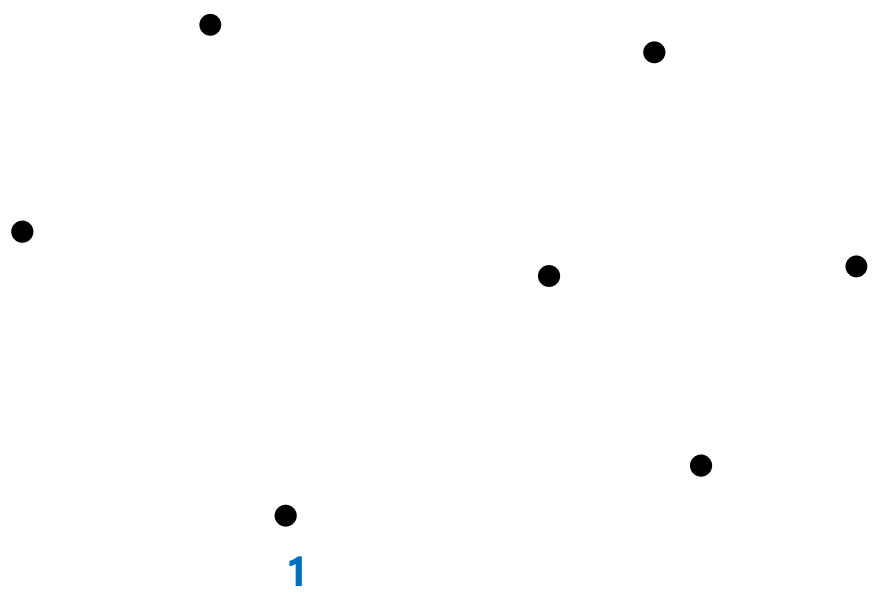
  2) 다각형 내부에 모든 점을 포함시키는 경우



Convex Hull algorithm

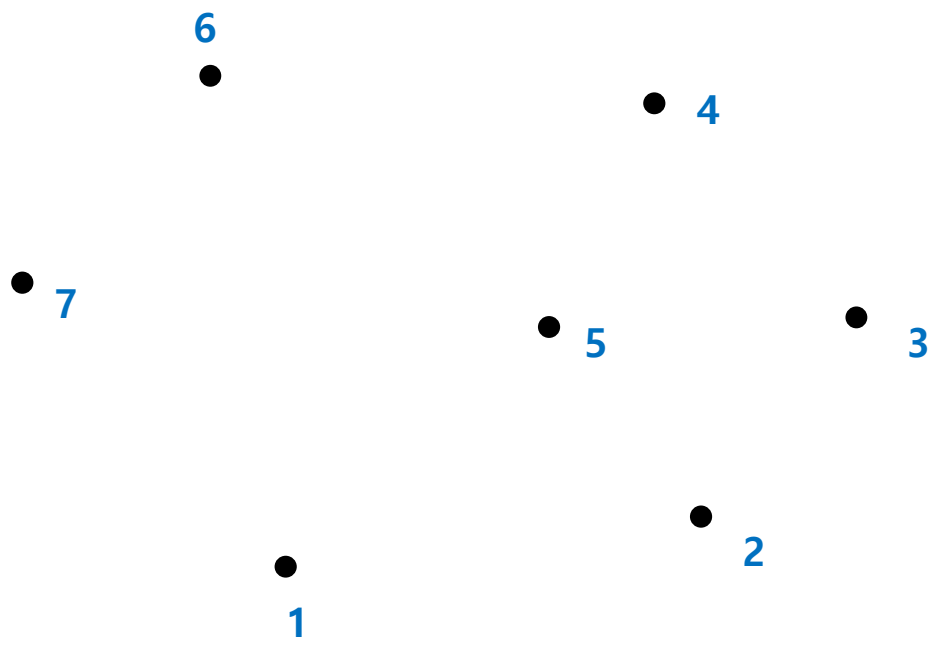
1) 기준점 잡기

- y좌표가 가장 작은 것
- y좌표가 같은 점이 있다면 그 중 x좌표가 가장 작은 것



Convex Hull algorithm

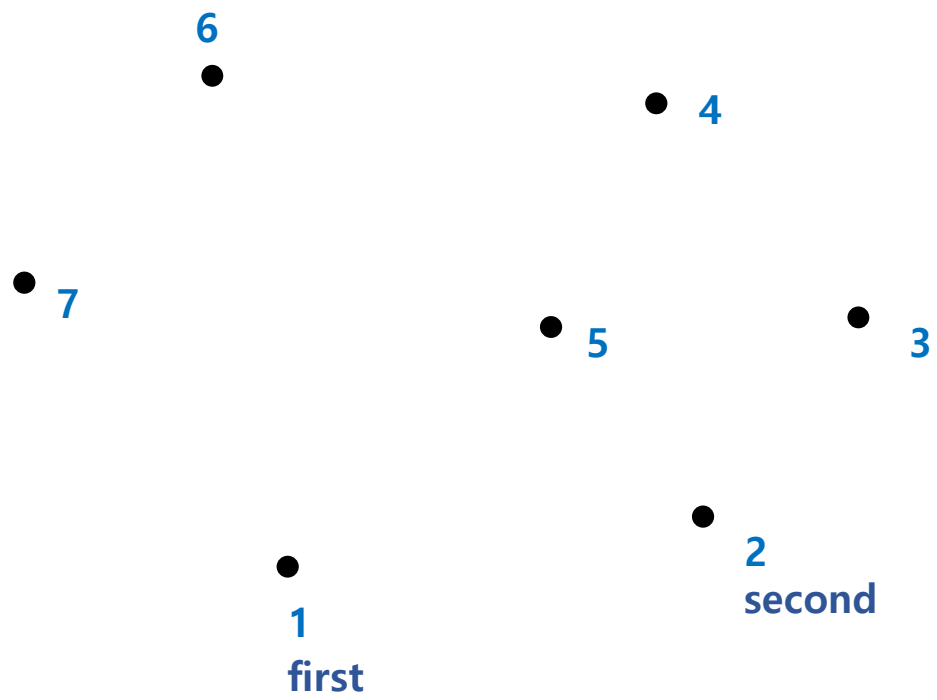
2) 기준점을 시작으로 점들을 반시계 방향 정렬 (CCW 알고리즘 이용)



Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
2
1

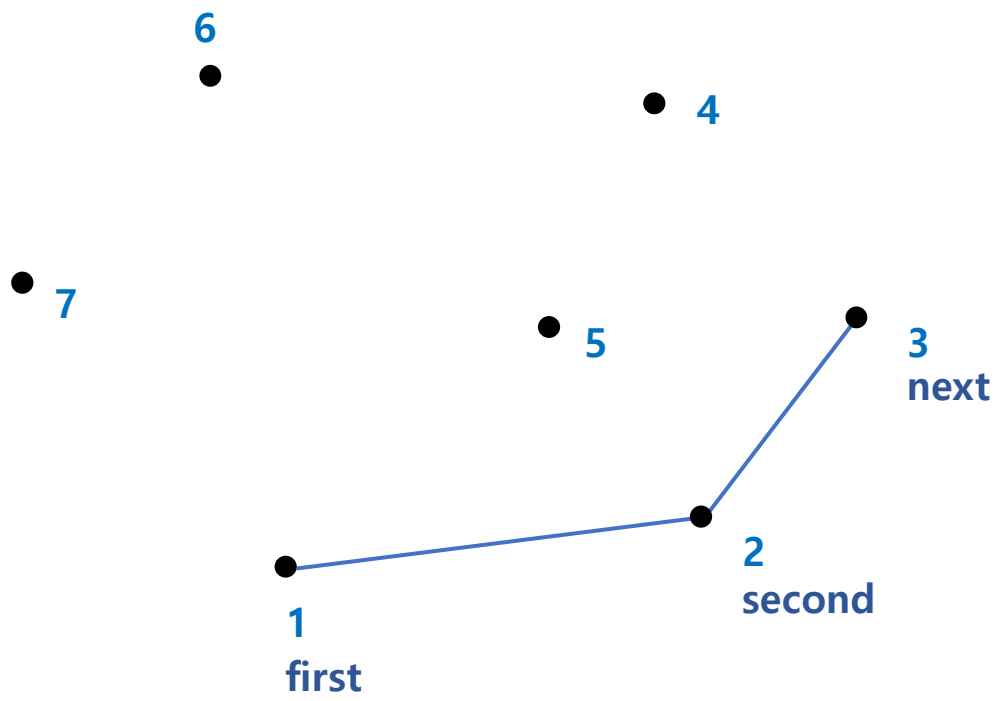


- 스택에 1, 2번 점을 넣고 시작

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
3
2
1

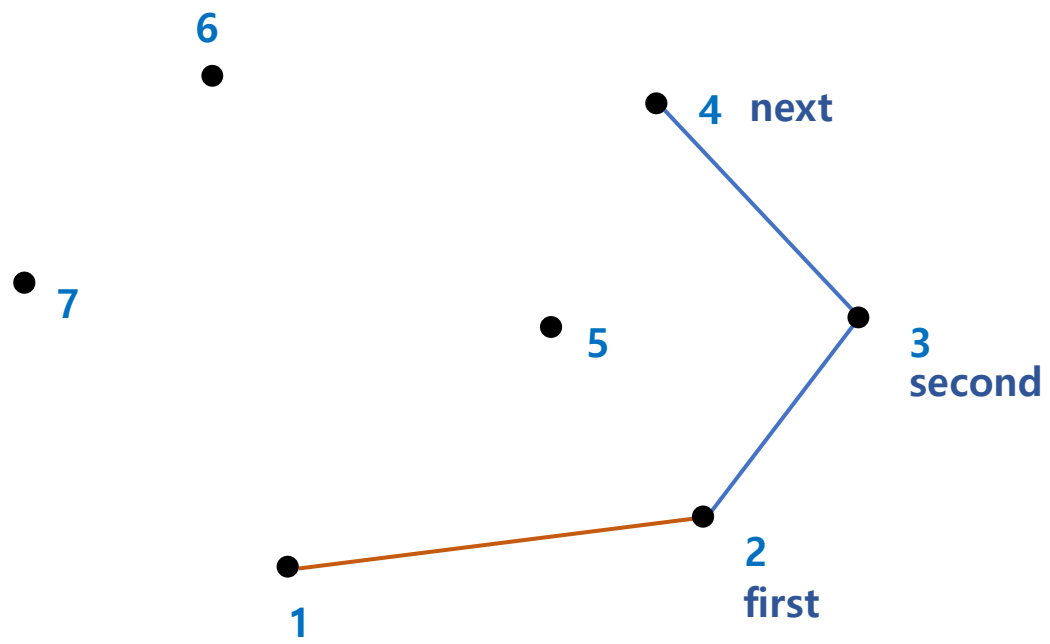


- 3번째 점부터 탐색 시작
- 스택에서 first, second를 pop, next와 CCW
  - 1, 2번 점을 꺼내 3번과 CCW
- counterclockwise인지(좌회전) 확인
  - 양수라면 좌회전 → 볼록 껍질 가능!
  - first, second, next를 스택에 push

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
4
3
2
1

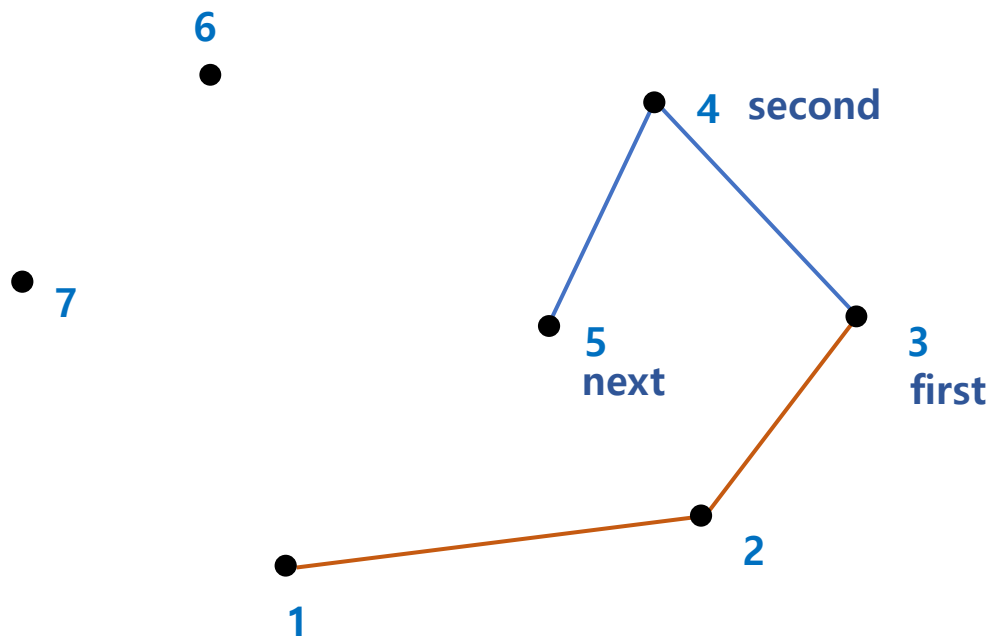


- 스택에서 first, second를 pop, next와 CCW
  - 2, 3번 점을 꺼내 4번과 CCW
- counterclockwise인지(좌회전) 확인
  - 양수라면 좌회전 → 볼록 껍질 가능!
  - first, second, next를 스택에 push

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
5
4
3
2
1

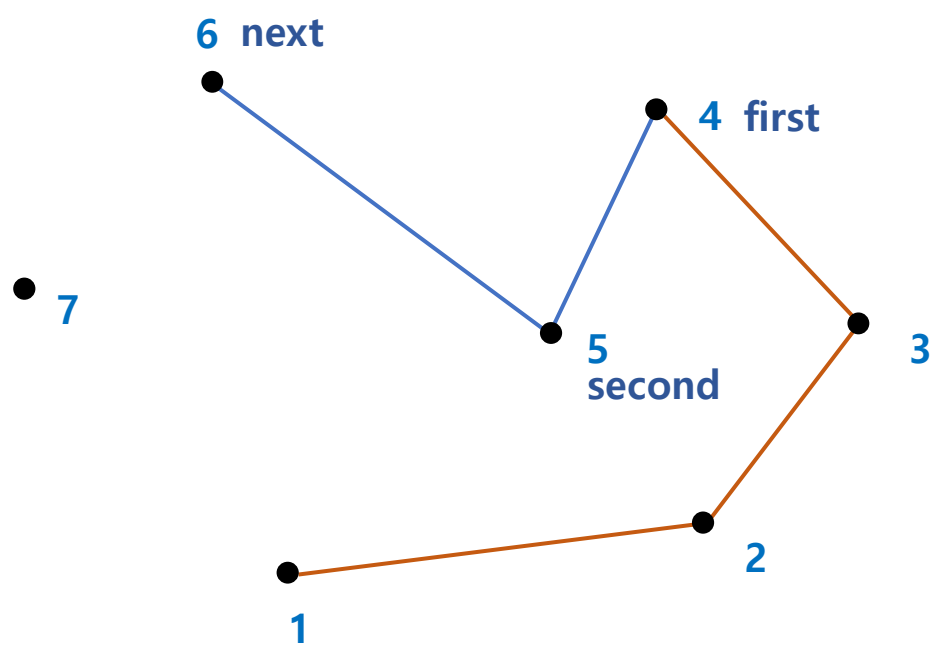


- 스택에서 first, second를 pop, next와 CCW
  - 3, 4번 점을 꺼내 5번과 CCW
- counterclockwise인지(좌회전) 확인
  - 양수라면 좌회전 → 볼록 껍질 가능!
  - first, second, next를 스택에 push

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
3
2
1

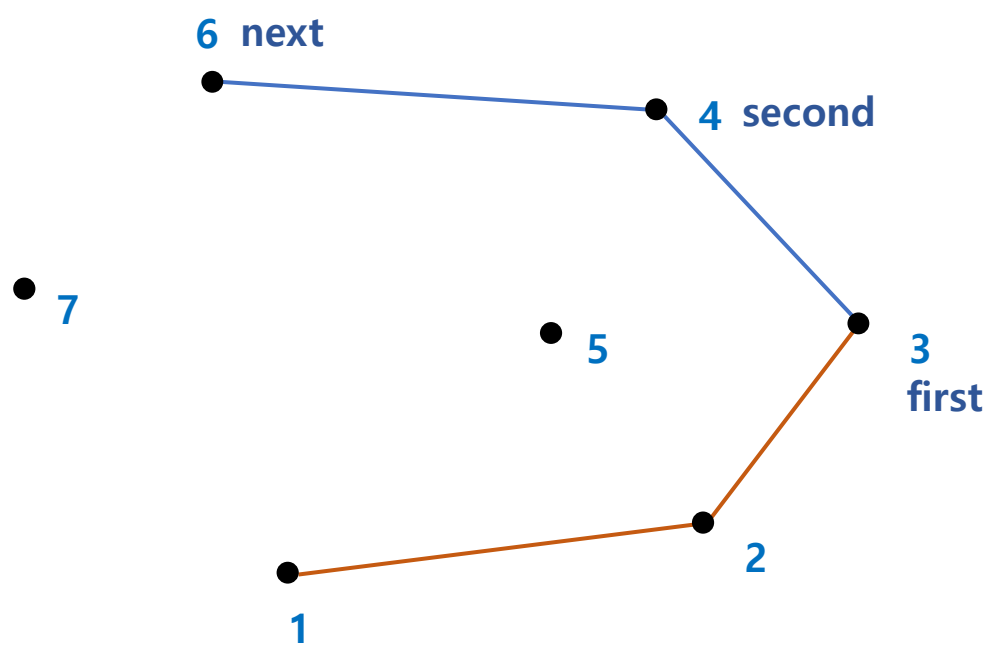


- 스택에서 first, second를 pop, next와 CCW
  - 4, 5번 점을 꺼내 6번과 CCW
- counterclockwise인지(좌회전) 확인
  - 우회전이므로 스택에서 다시 pop
  - 다시 진행

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
6
4
3
2
1

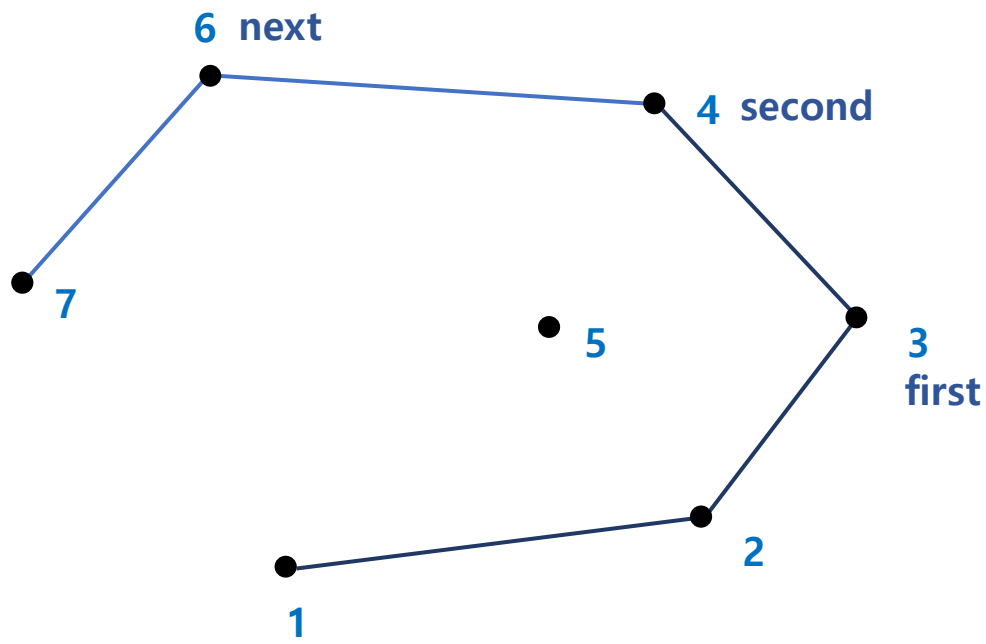


- 스택에서 first, second를 pop, next와 CCW
  - 3번 점을 꺼내 4, 6번과 CCW
- counterclockwise인지(좌회전) 확인
  - 좌회전 → 볼록 껍질 가능!
  - first, second, next를 스택에 push

Convex Hull algorithm

3) Convex hull 알고리즘 이용

stack
7
6
4
3
2
1



- 스택에서 first, second를 pop, next와 CCW
  - 4, 6번 점을 꺼내 7번과 CCW
- counterclockwise인지(좌회전) 확인
  - 좌회전 → 볼록 껍질 가능!
  - first, second, next를 스택에 push
- 모든 점을 계산했으므로 끝!
- 스택에 있는 점들이 볼록 껍질

Convex Hull algorithm code

```
#define MAX 100002
typedef long long ll;

struct INFO {
    int x, y;
    int p, q;
    INFO(int x1 = 0, int y1 = 0) : x(x1), y(y1), p(1), q(0) {}
};
INFO p[MAX];

bool comp(const INFO &A, const INFO &B) {
    if (1LL * A.q * B.p != 1LL * A.p * B.q) return 1LL * A.q * B.p < 1LL * A.p * B.q;

    if (A.y != B.y) return A.y < B.y;

    return A.x < B.x;
}

ll ccw(const INFO &A, const INFO &B, const INFO &C) {
    return 1LL * (A.x * B.y + B.x * C.y + C.x * A.y - B.x * A.y - C.x * B.y - A.x * C.y);
}
```

```
int main() {
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int x, y;
        scanf("%d %d", &x, &y);

        p[i] = INFO(x, y);
    }
    sort(p, p + n, comp);  // y좌표, x좌표가 작은 순으로 정렬

    for (int i = 1; i < n; i++) {  // 기준점으로부터 상대 위치 계산
        p[i].p = p[i].x - p[0].x;
        p[i].q = p[i].y - p[0].y;
    }
    sort(p + 1, p + n, comp);  // 반시계 방향으로 정렬(기준점 제외)

    stack<int> s;
    s.push(0);  // 스택에 first, second 넣기
    s.push(1);

    int next = 2;
    while (next < n) {
        while (s.size() >= 2) {
            int first, second;
            second = s.top();
            s.pop();
            first = s.top();

            if (ccw(p[first], p[second], p[next]) > 0) {  // 좌회전인 경우
                s.push(second);
                break;
            }
        }
        s.push(next++);  // next push
    }
    printf("%d", s.size());
    return 0;
}
```

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

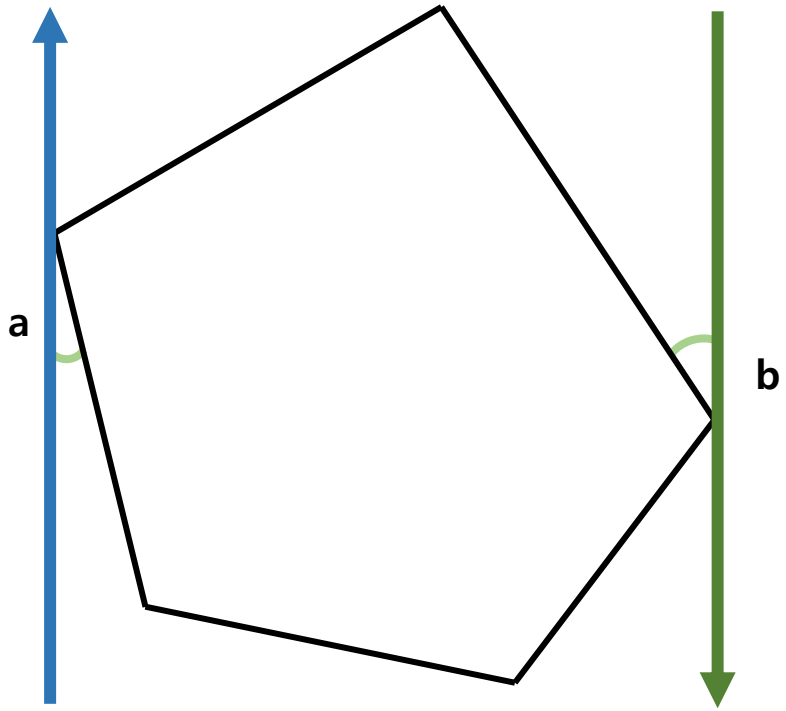
Rotating Calipers algorithm

- Calipers : 작은 물건의 지름, 너비 등을 측정하는 도구
  두 개의 평행한 변 사이에 물체를 끼워 측정함

- **볼록 다각형의 지름** 길이 구하기 (볼록 다각형을 완전히 포함하는 가장 긴 선분의 길이)
  1) 다각형을 평행한 두 직선 사이에 끼우고
  2) 다각형을 따라 두 직선을 (반시계/시계 방향으로) 한 바퀴 돌리면서
  3) 두 직선에 닿는 꼭짓점의 거리를 모두 계산 후
  4) 가장 긴 거리를 구함

Rotating Calipers algorithm

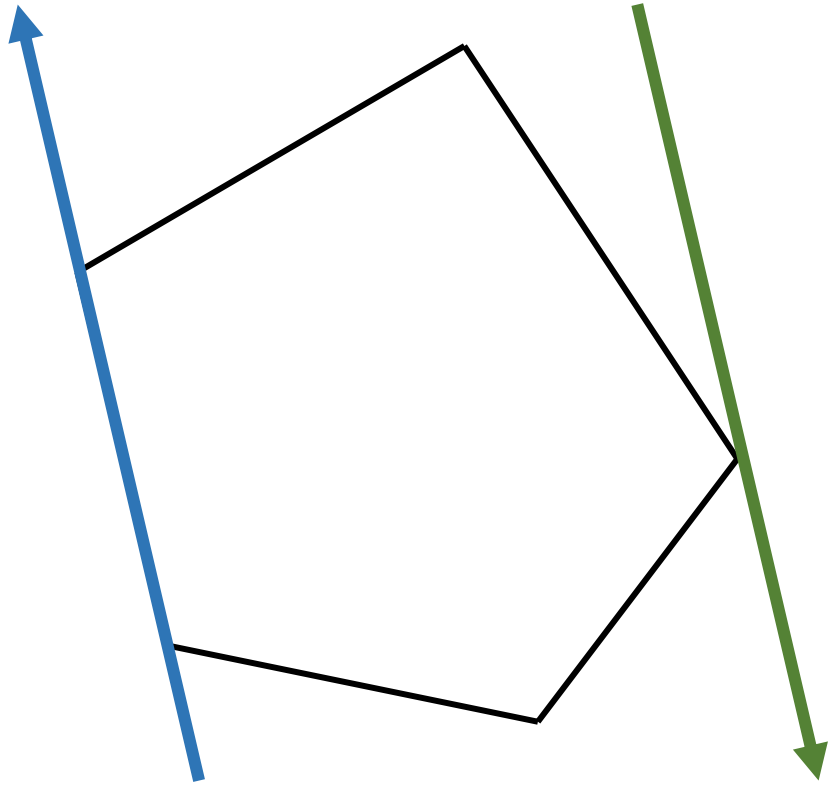
- 가장 왼쪽 점(x값이 가장 작은 점)부터 시작



- a보다 b가 작으므로 a만큼 반시계 방향으로 회전

Rotating Calipers algorithm

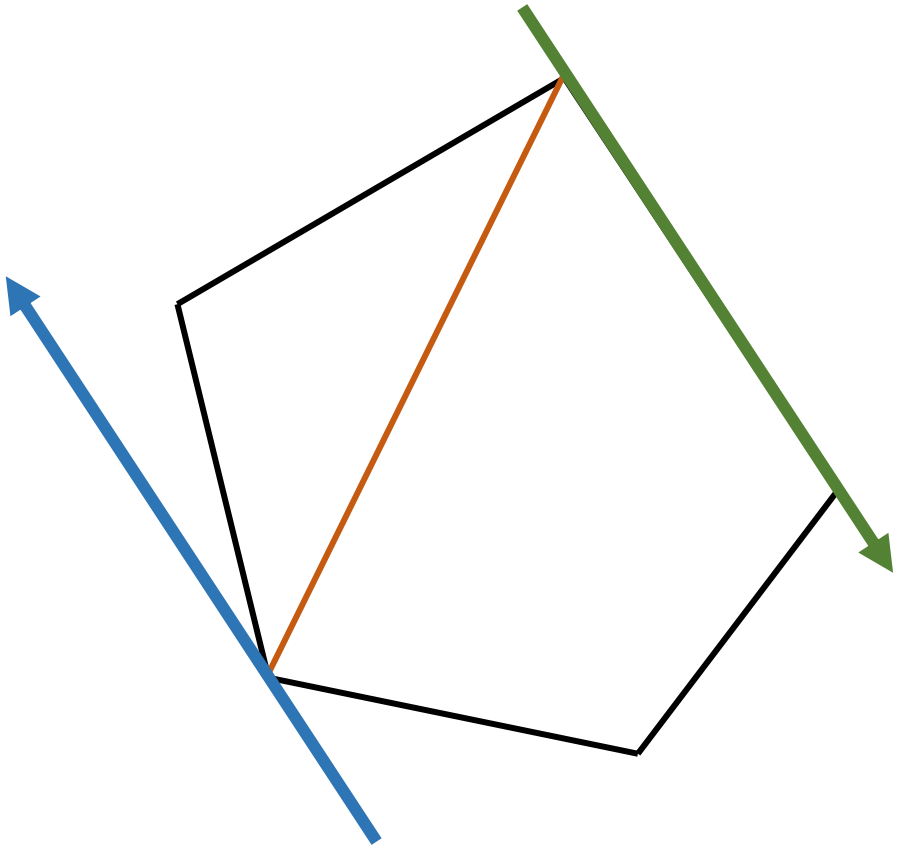
- 두 벡터가 자리를 바꿀 때까지 반복



- 반시계 방향 회전

Rotating Calipers algorithm

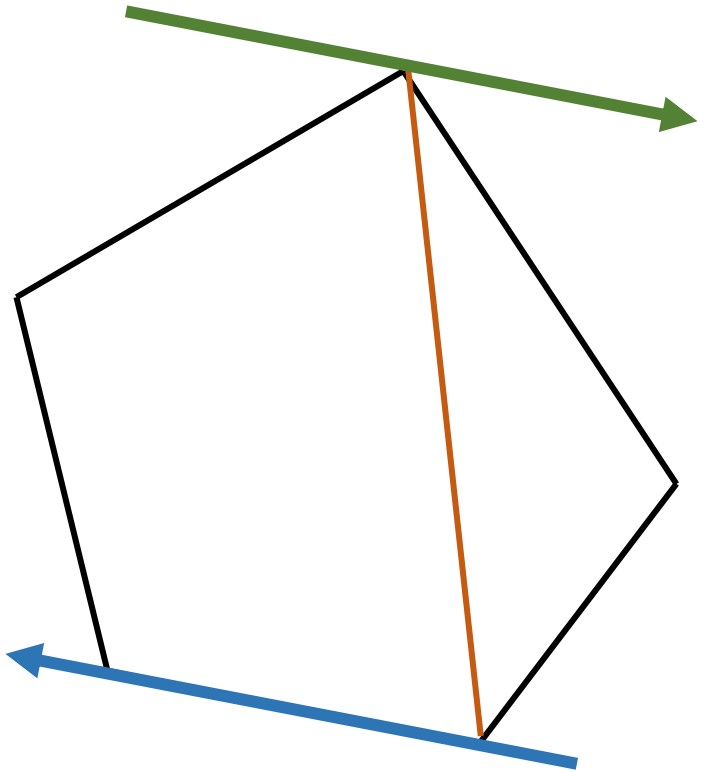
- 두 벡터가 자리를 바꿀 때까지 반복



- 반시계 방향 회전

Rotating Calipers algorithm

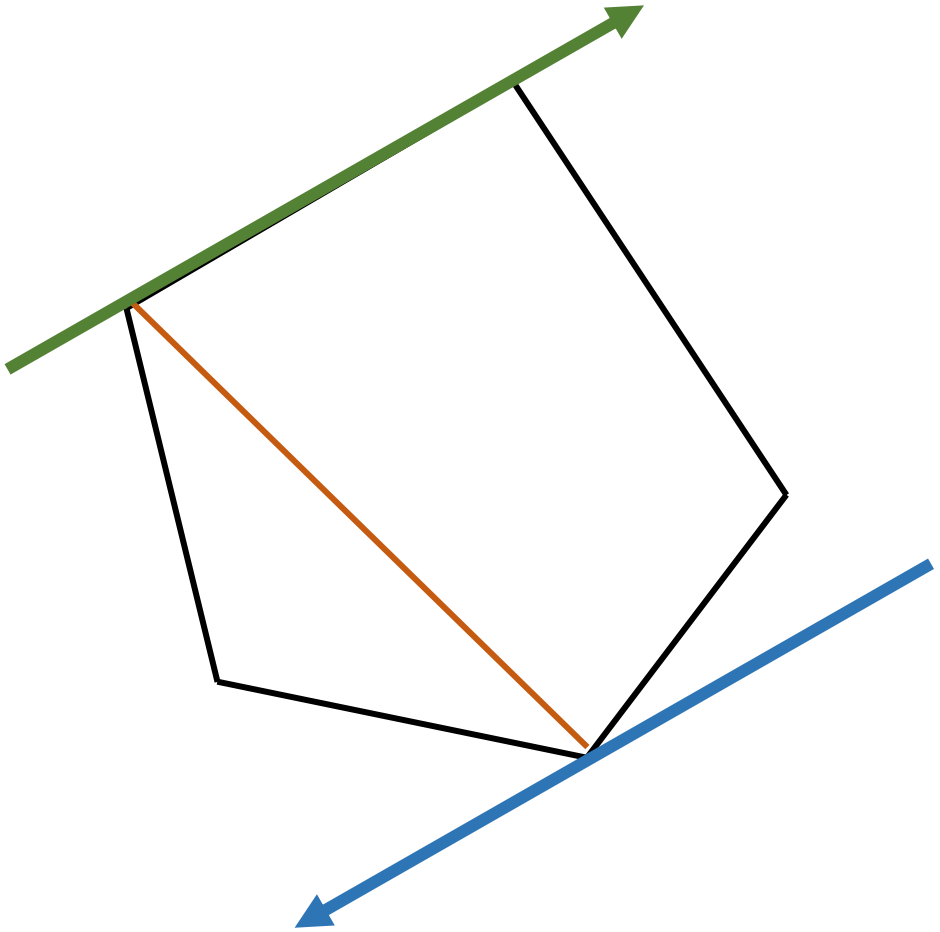
- 두 벡터가 자리를 바꿀 때까지 반복



- 반시계 방향 회전

Rotating Calipers algorithm

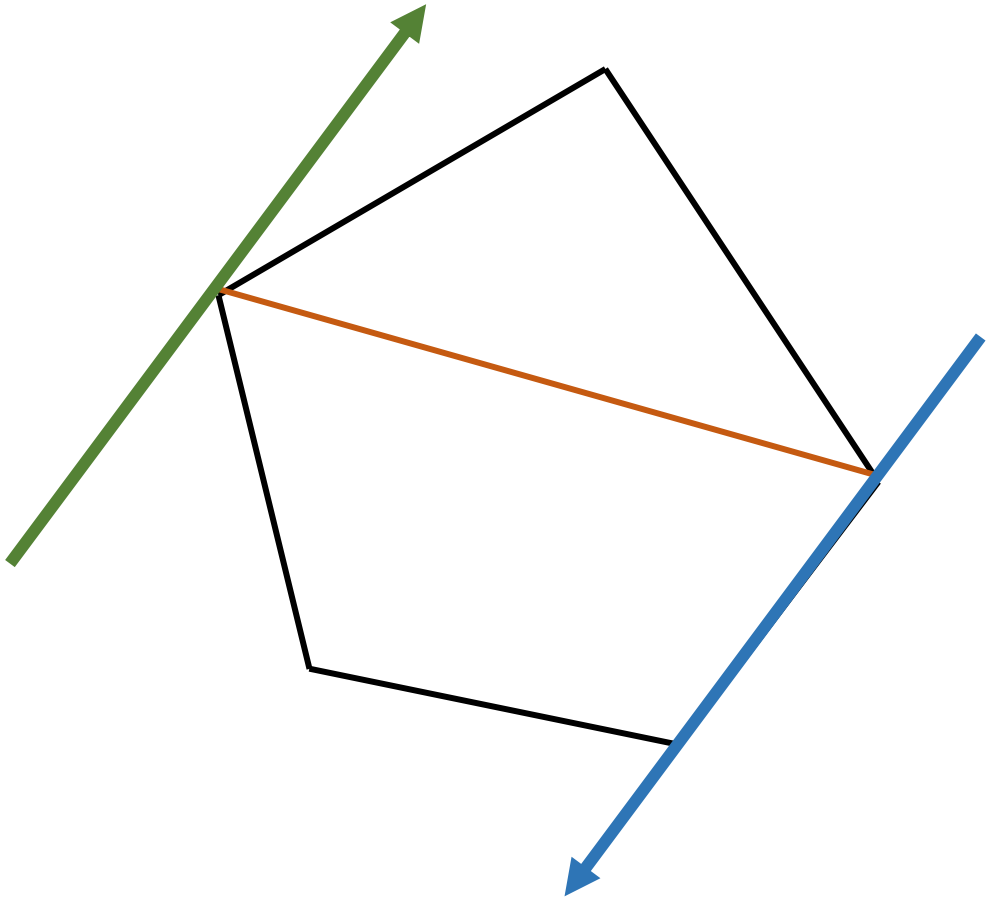
- 두 벡터가 자리를 바꿀 때까지 반복



- 반시계 방향 회전

Rotating Calipers algorithm

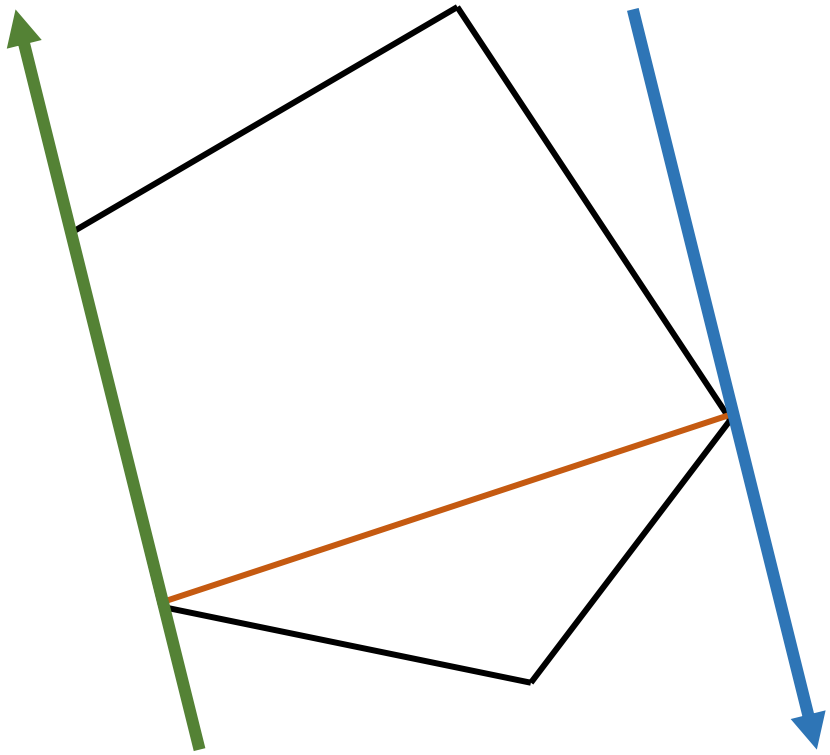
- 두 벡터가 자리를 바꿀 때까지 반복



- 반시계 방향 회전

Rotating Calipers algorithm

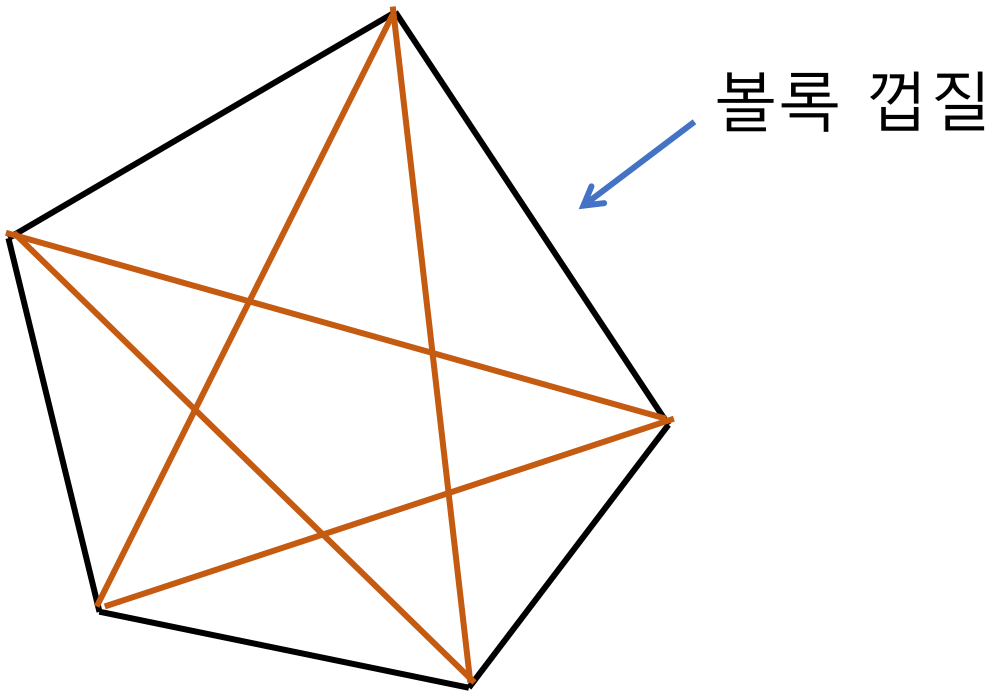
- 두 벡터가 자리를 바꿀 때까지 반복



- 두 벡터의 자리가 바뀜
- 한 바퀴를 다 돌았으므로 종료

Rotating Calipers algorithm 응용

- Convex hull로 구한 볼록 껍질에서 두 점의 길이를 계산



Rotating Calipers algorithm code

```
int len = convex_hull.size();
int left = 0, right = 0;

for (int i = 0; i < len; i++) {
    if (convex_hull[i] < convex_hull[left]) left = i;    // 가장 왼쪽 점
    if (convex_hull[right] < convex_hull[i]) right = i;  // 가장 오른쪽 점
}
vector2 calipers(0, 1);                                  // 가장 처음의 캘리퍼스는 | | 모양으로 설정
double ret = (convex_hull[right] - convex_hull[left]).norm();  // 처음 측정한 길이

vector<vector2> toNext(len);  // 현재 점에서 다음 점으로 가는 벡터
for (int i = 0; i < len; i++) {
    toNext[i] = (convex_hull[(i + 1) % len] - convex_hull[i]).normalize();  // 볼록 껍질의 처음부터 끝까지
}
a = left, b = right;  // a, b는 돌아가는 점들의 현재 위치
int aa = a, bb = b;   // aa,bb 에는 가장 먼 거리의 점들 저장

while (a != right || b != left) {
    double A = calipers.dot(toNext[a]);
    double B = -calipers.dot(toNext[b]);
    if (A > B) {  // b보다 a의 각도가 더 작은 경우
        calipers = toNext[a];
        a = (a + 1) % len;  // a를 다음 점으로
    } else { // a보다 b의 각도가 더 작은 경우
        calipers = vector2(0, 0) - toNext[b];
        b = (b + 1) % len;  // b를 다음 점으로
    }

    long double dist = (convex_hull[a] - convex_hull[b]).norm();  // 거리 측정
    if (ret < dist) { // 더 긴 거리 저장
        ret = dist;
        aa = a;
        bb = b;
    }
}
```

https://blog.naver.com/PostView.naver?blogId=ehddml1229&logNo=220932792326&parentCategoryNo=&categoryNo=16&viewDate=&isShowPopularPosts=true&from=search

Boj

- 11758 CCW (G5) https://www.acmicpc.net/problem/11758
- 1708 볼록 껍질 (P5) https://www.acmicpc.net/problem/1708
- 9240 로버트 후드 (P3) https://www.acmicpc.net/problem/9240
- 10254 고속도로 (P2) https://www.acmicpc.net/problem/10254

감사합니다.