2차원 계산 기하

CCW, 선분 교차 판정, 볼록 껍질

신지섭

오늘 할 내용은?

1. 벡터, CCW: 계산 기하를 위한 기본적인 개념을 잡고 갑시다

2. 선분 교차 판정 Line Segment Intersection Check

3. 볼록 껍질 convex hull

    - 회전하는 캘리퍼스 Rotating Calipers

    - 내부 점 판정 Point in Convex Polygon Check

주제가 광범위하다 보니 할 내용도 많다

vector 벡터

- 자료구조 vector를 뜻하는 것이 아니라

  기하에서의 벡터(euclidean vector)를 말함

- **std::pair**를 이용해서 2차원 벡터를 표현하자

vector 벡터

- 벡터의 내적(Inner Product): $a \cdot b = a_x b_x + a_y b_y = |a||b|\cos\theta$

⇒ 벡터의 사이각 구할 때 쓰임: $\theta = \arccos(\frac{a \cdot b}{|a||b|})$

⇒ 벡터의 직각 여부 확인: 직각이면 $a \cdot b = 0$

- **벡터의 외적(Cross Product)**: $a \times b = a_x b_y - a_y b_x = |a||b|\sin\theta$

⇒ 삼각형의 넓이를 계산: 넓이는 $\frac{1}{2}||a| \cdot |b|\sin\theta|$

⇒ 두 벡터의 방향을 판별: 양수이면 반시계, 음수이면 시계

외적을 이용해 방향을 판별하는 경우가 가장 많아욧 ~\(≧▽≦)/~

벡터 문제풀이

벡터를 이용해
면적을 구해봅시다

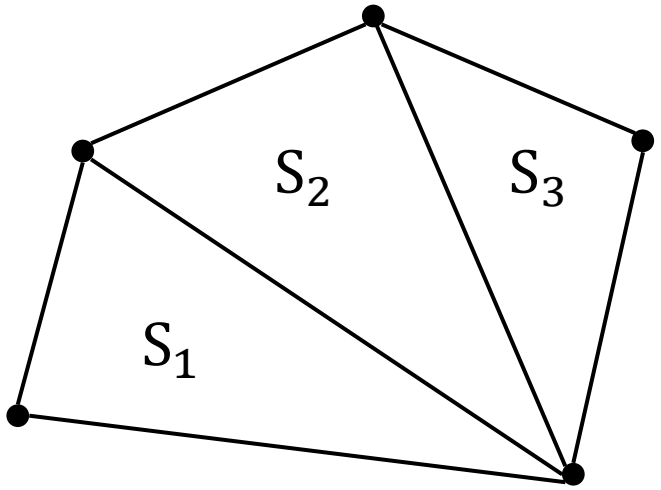
백준 2166
다각형의 면적

2166 다각형의 면적

문제: 2차원 상의 다각형의 넓이를 구하는 문제

⇒ 다각형을 삼각형으로 쪼개자

⇒ 쪼개진 삼각형의 넓이의 합이 다각형의 넓이이다.



다각형의 면적 코드

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
using Point = pair<long long, long long>; //point 정의

int n;
Point point[10005];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n;
    double an = 0;
    for (int i = 0; i < n; i++) {
        int a, b; cin >> a >> b;
        point[i] = { a, b };
    }
    for (int i = 1; i < n - 1; i++) {
        Point vec1 = { point[i].first - point[0].first, point[i].second - point[0].second };
        Point vec2 = { point[i + 1].first - point[0].first, point[i + 1].second - point[0].second };
        an += ((double)(vec1.first * vec2.second - vec1.second * vec2.first) / 2);
        // 외적을 이용한 삼각형 면적 구하기
    }
    cout << fixed;
    cout.precision(1);
    cout << abs(an); //넓이는 양수
    return 0;
}
```

CCW

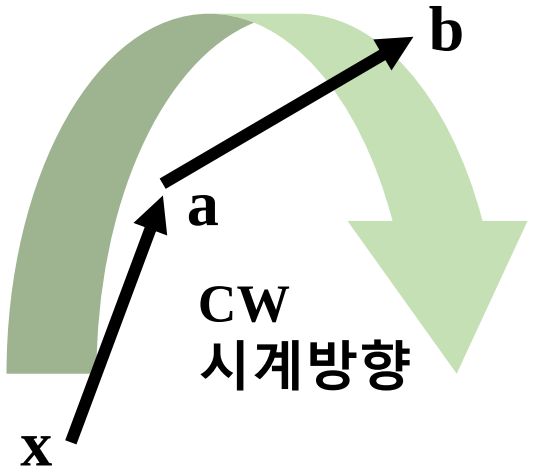
CCW: Counter Clock Wise의 약자

세 점으로 만든 두 벡터의 방향이 시계인지 반시계인지 판단

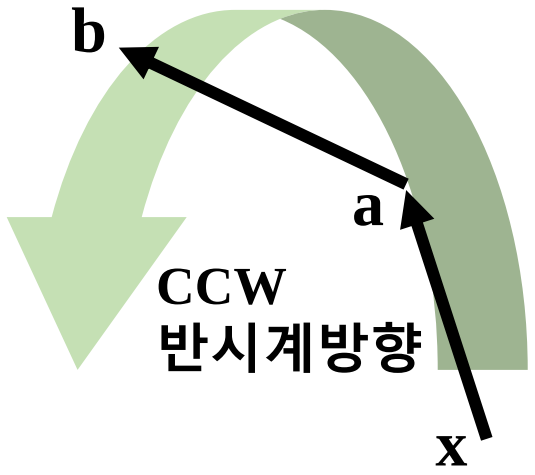
※ ccw가 0이면 세 점은 평행

```cpp
using Point = pair<long long, long long>;
int ccw(Point x, Point a, Point b) {
    Point vec1 = { a.first - x.first, a.second - x.second }; // 첫 번째 벡터
    Point vec2 = { b.first - x.first, b.second - x.second }; // 두 번째 벡터
    long long det = vec1.first * vec2.second - vec1.second * vec2.first; //determinant
    if (det > 0) // 양수이면 반시계
        return 1;
    else if (det == 0)
        return 0;
    else return -1; // 시계
}
```

▲ CCW 코드



ccw(x, a, b) > 0



ccw(x, a, b) < 0

CCW 문제풀이

사실 앞의 코드를 복붙하면...?

백준 11758
CCW

11758 CCW

문제: CCW를 구하는 문제

앞의 내용을 그대로 넣으면 되니 설명은 생략!

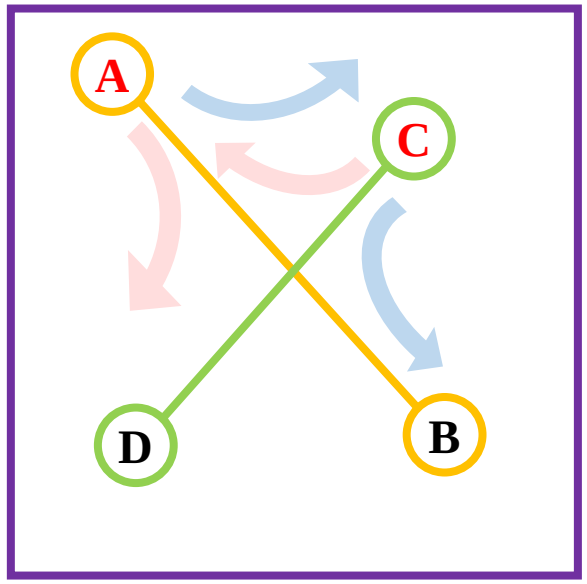
CCW 코드

```cpp
#include <iostream>
using namespace std;
using Point = pair<long long, long long>; // point자료형 정의

int ccw(Point x, Point a, Point b) {
    // 외적을 위한 벡터 계산
    Point vec1 = { a.first - x.first, a.second - x.second };
    Point vec2 = { b.first - x.first, b.second - x.second };
    // determinant
    long long det = vec1.first * vec2.second - vec1.second * vec2.first;
    if (det > 0)
        return 1;
    else if (det == 0)
        return 0;
    else return -1;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    Point point[3];
    for (int i = 0; i < 3; i++) {
        long long a, b; cin >> a >> b;
        point[i] = { a, b };
    }
    cout << ccw(point[0], point[1], point[2]);
    return 0;
}
```

선분 교차 판정 Line Segment Intersection Check

선분 교차: 두 선분이 만나는 상황

단, A < B 이고 C < D

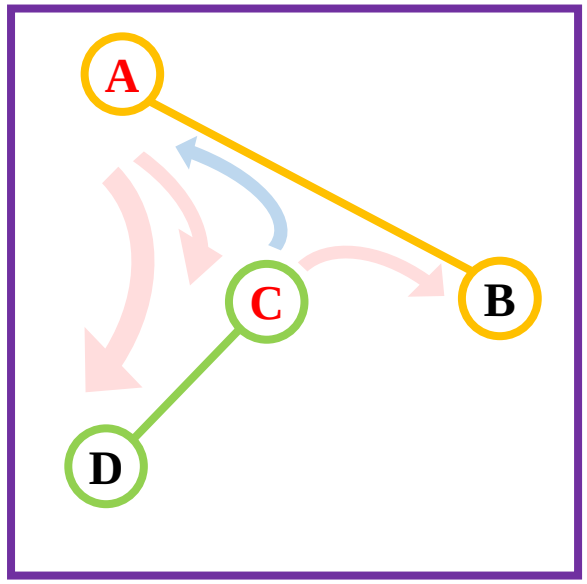


1. 선분이 교차하는 경우

$ccw(A, B, C) * ccw(A, B, D) \le 0$

$ccw(C, D, A) * ccw(C, D, B) \le 0$

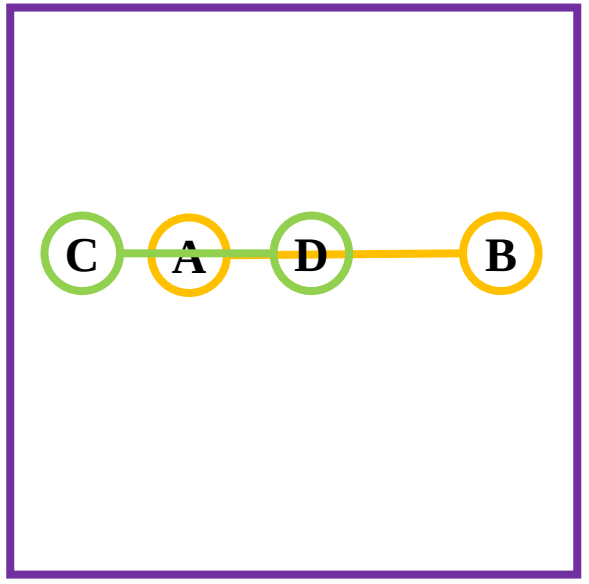
교차!



2. 선분이 교차하지 않는 경우

$ccw(A, B, C) * ccw(A, B, D) < 0$

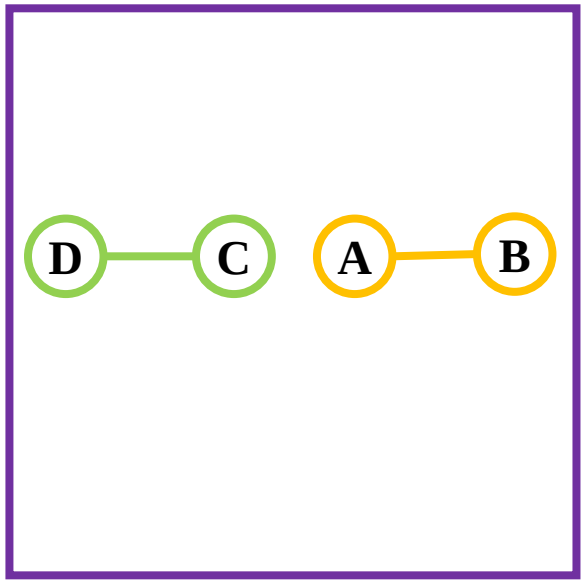
$\| \; ccw(C, D, A) * ccw(C, D, B) < 0$



3. 선분이 여러 지점에서 만남

$A \le D \;\&\&\; C \le B$

교차!



4. 선분이 평행하지만 안 만남

$(D < A \;\&\&\; C < A) \;\|$
$(B < D \;\&\&\; B < C)$

선분 교차 판정 문제풀이

선분 교차 판정 기본 문제!
선분 교차 1은 세 점이 일직선인 경우를 생각하지 않아요!

백준 17387
선분 교차 2

17387 선분 교차 2

문제: 두 선분이 교차하는지 확인하는 문제

$-1{,}000{,}000 \leq x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4 \leq 1{,}000{,}000$

$\Rightarrow$ 자료형을 long long으로 하자

선분 교차 2 코드

```cpp
// 교차 판정 함수
int check(Point a, Point b, Point c, Point d) {
    int ab = ccw(a, b, c) * ccw(a, b, d);
    int cd = ccw(c, d, b) * ccw(c, d, a);
    if (ab == 0 && cd == 0) { //직선이거나 적어도 하나의 끝점이 같으면
        if (a > b) swap(a, b);
        if (c > d) swap(c, d);
        if (a <= d && c <= b) // 공통 직선이 있으면
            return 1;
        return 0;
    }
    if (ab <= 0 && cd <= 0) // 교차하면
        return 1;
    return 0;
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    pair<Point, Point> l1;
    pair<Point, Point> l2;
    long long a, b, c, d;
    cin >> a >> b >> c >> d;
    l1 = { {a, b}, {c, d} };
    cin >> a >> b >> c >> d;
    l2 = { {a, b}, {c, d} };
    cout << check(l1.first, l1.second, l2.first, l2.second);
    return 0;
}
```

※ CCW 함수는 앞서 본 CCW 함수와 같습니다!

Convex Hull 볼록 껍질

- 볼록 껍질: n개의 점들을 포함하는 가장 작은 볼록 다각형
- 볼록 다각형: 모든 내각이 180도보다 같거나 작은 단순 다각형

1. X값이 가장 작은 점을 기준으로 하여 다른 점들을 반시계방향으로 정렬
2. 0, 1번째 점을 stack에 넣고 스택 제일 위의 점 2개와 새로운 점을 CCW

   ccw > 0이면 새로운 점 넣음, **ccw ≤ 0**이면 스택에서 하나 빼고 다시 반복

   ※ stack를 이용해 **볼록껍질에 해당하는 점**만 넣자!

   ※ stack의 크기가 2보다 작으면 새로운 점 넣기
3. 모든 점들에 대해 반복

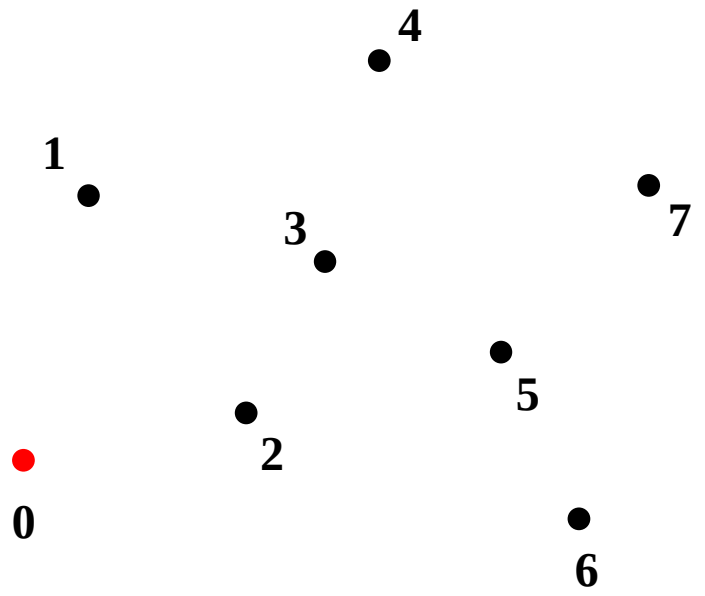
- 시간 복잡도 O(N log N)

Convex Hull 볼록 껍질

Step1. 정렬: x값을 기준으로 정렬

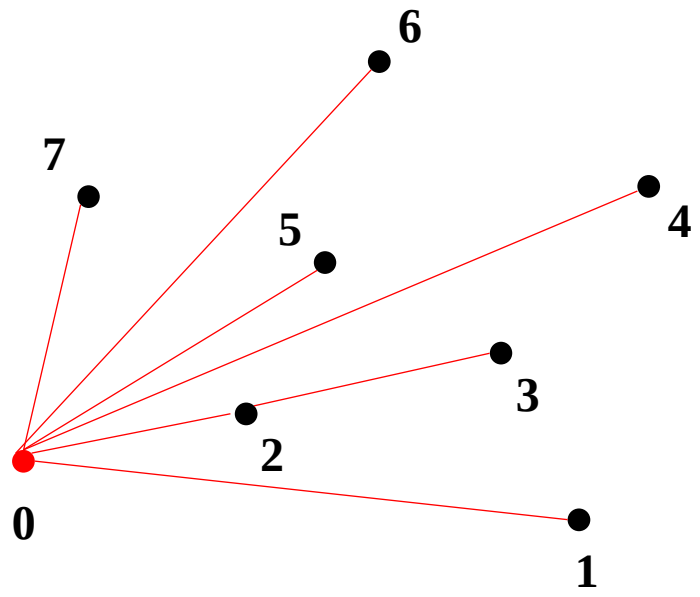
1. x값이 작은 점 먼저
2. x값이 같다면 y값이 작은 점 먼저

⇒ 사실 가장 작은 점 하나만 찾으면 됨



Convex Hull 볼록 껍질

Step1. 정렬: 0번 점을 기준으로 **반시계 순**으로 정렬



※ 기울기가 같다면 0번째 점과 가까운 점 먼저

Convex Hull 볼록 껍질

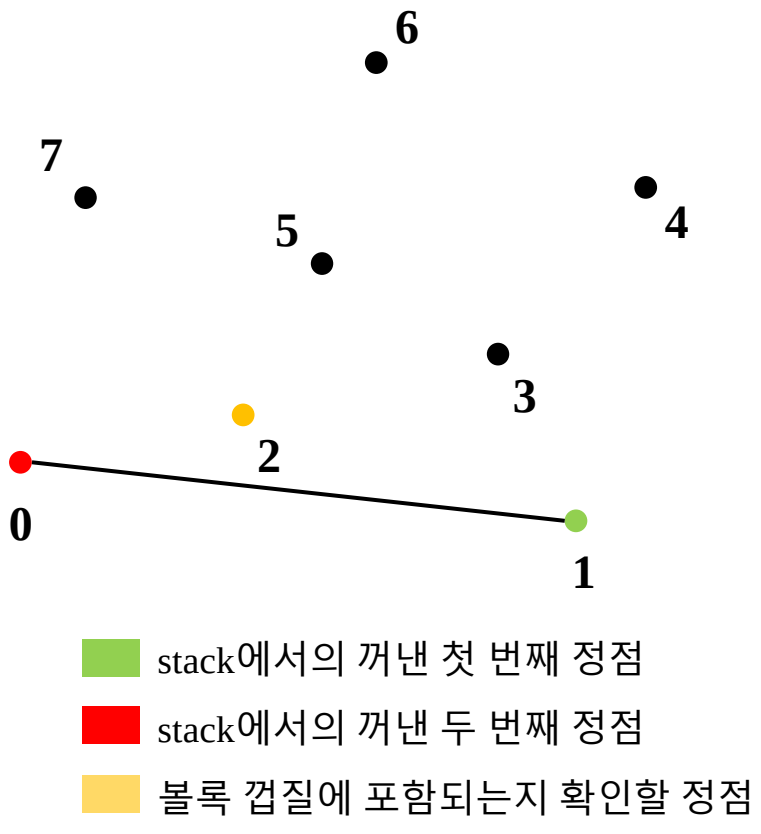
Step2. 0번과 1번 stack에 넣기

first:	**not started**
second:	**not started**
point:	**not started**

0번과 1번 stack에 넣기

stack

0 1



Convex Hull 볼록 껍질

Step2. ccw 값에 따라 second를 빼거나 패스

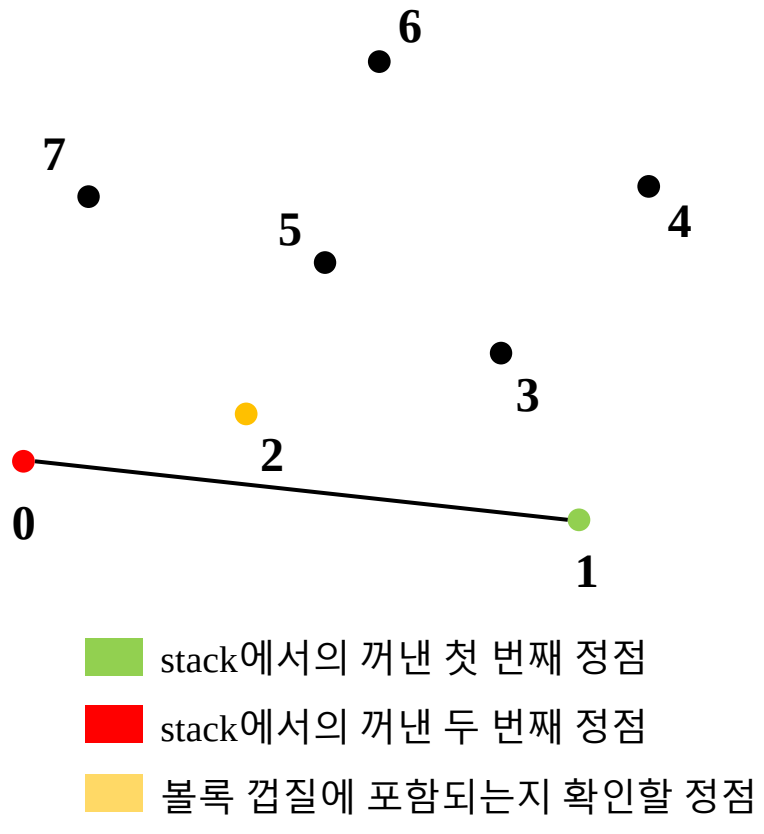
first: **0**
second: **1**
point: **2**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

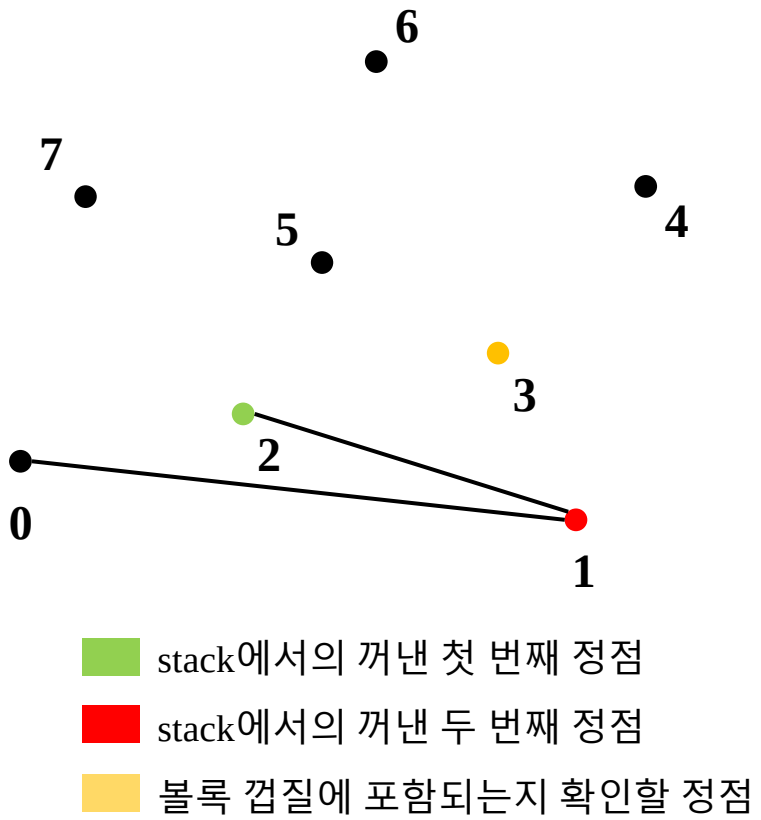
first: **1**
second: **2**
point: **3**

ccw(first, second, point) < 0

⇒ stack에서 하나 빼기

stack

0 **1** **2**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

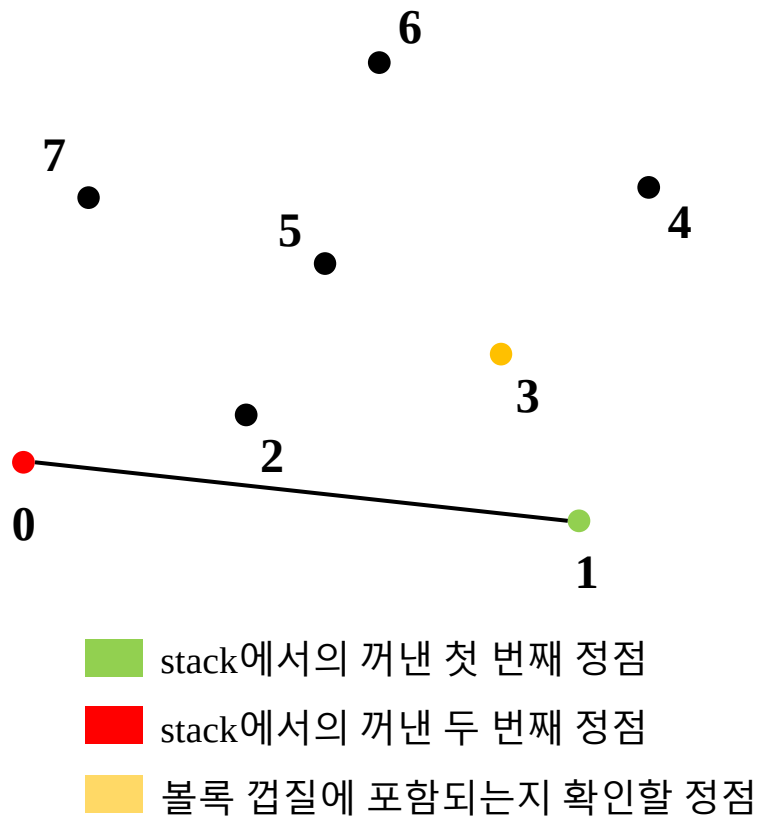
first: **0**
second: **1**
point: **3**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

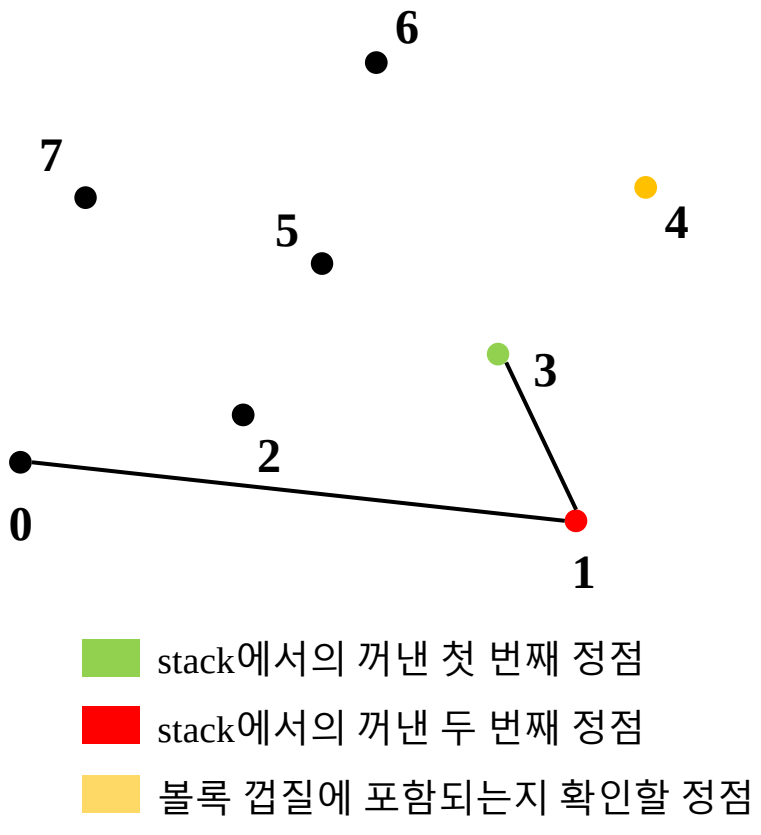
first:	**1**
second:	**3**
point:	**4**

ccw(first, second, point) < 0

⇒ stack에서 하나 빼기

stack

0 1 3



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

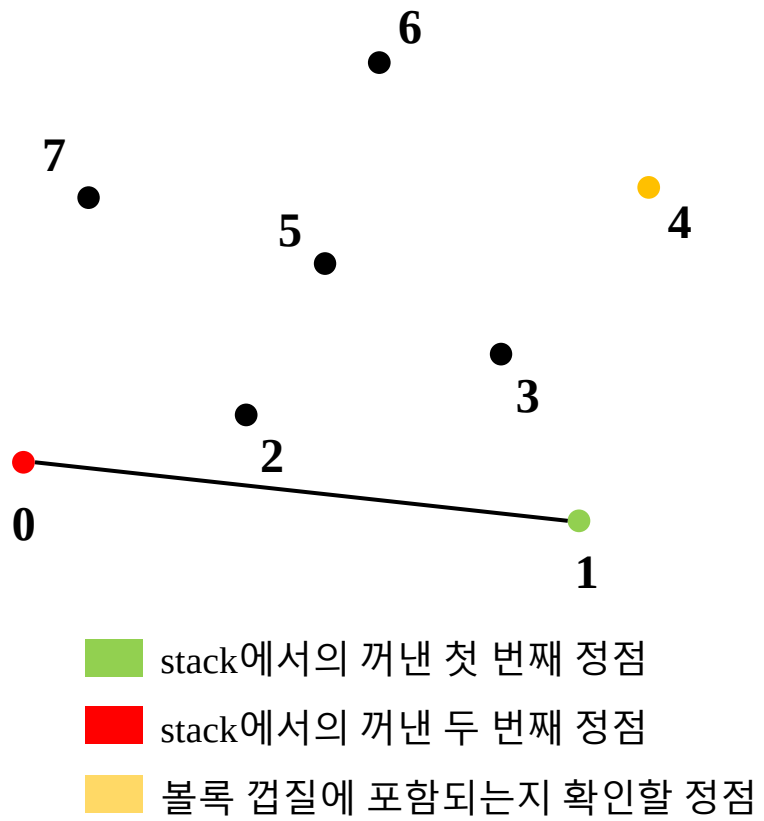
first: **0**
second: **1**
point: **4**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

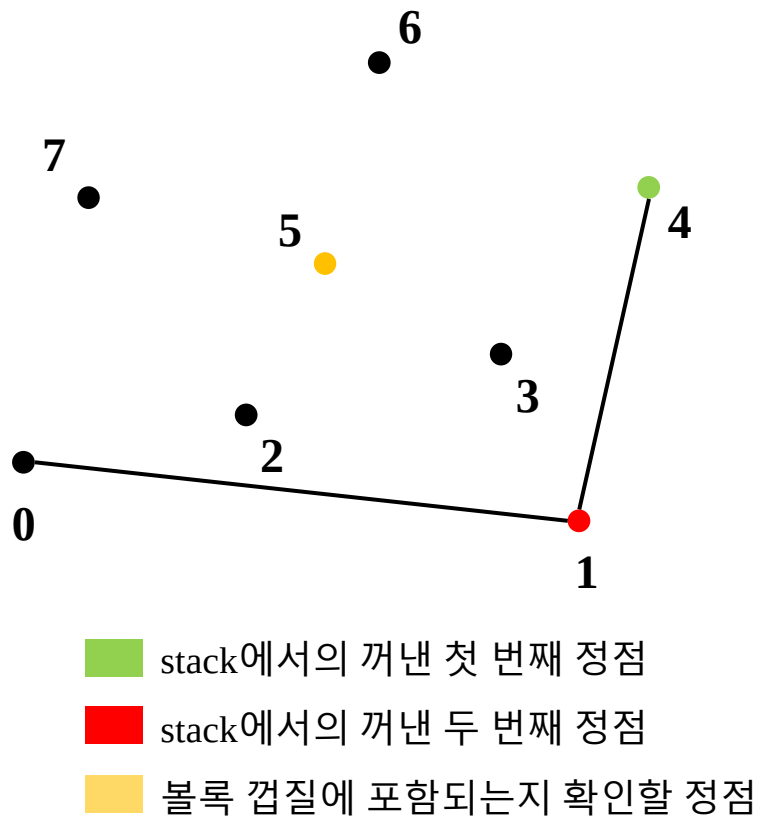
first: **1**
second: **4**
point: **5**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1** **4**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

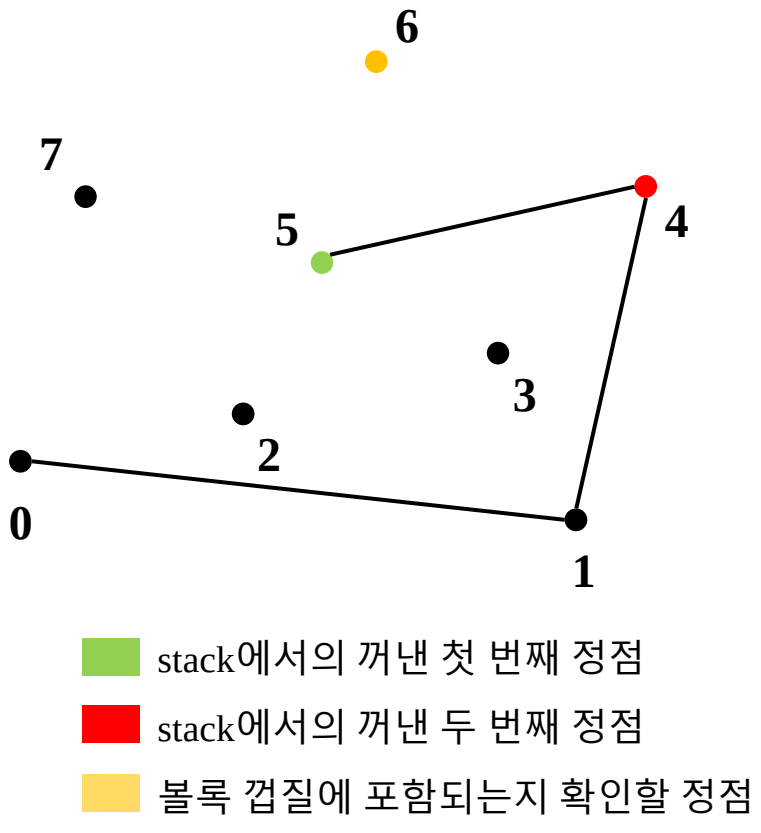
first:	**4**
second:	**5**
point:	**6**

ccw(first, second, point) < 0

⇒ stack에서 하나 빼기

stack

0 1 4 5



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

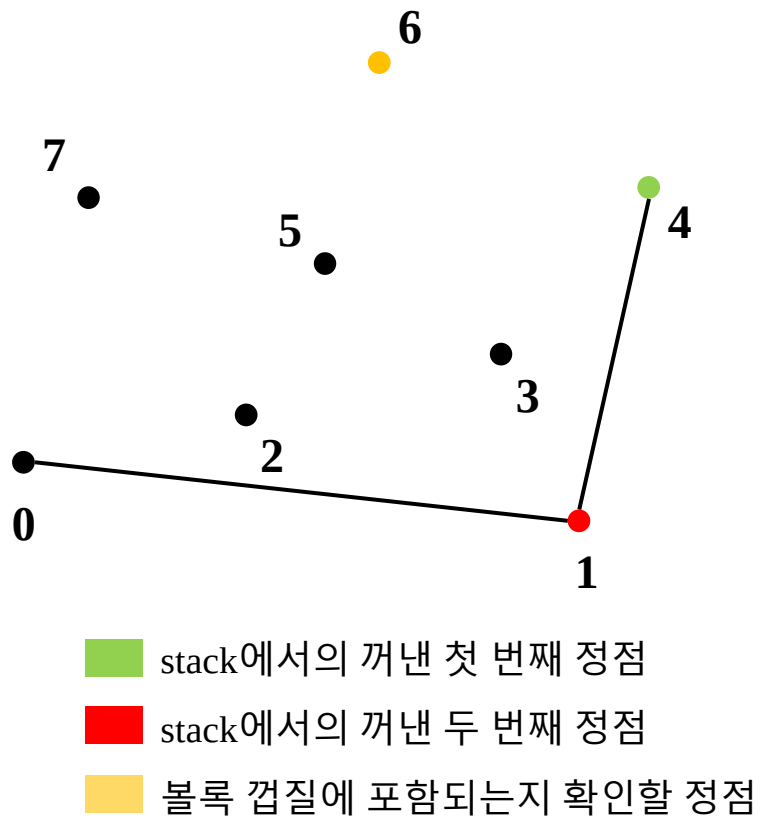
first: **1**
second: **4**
point: **6**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1** **4**



Convex Hull 볼록 껍질

Step3. 모든 점에 대해 step2 반복

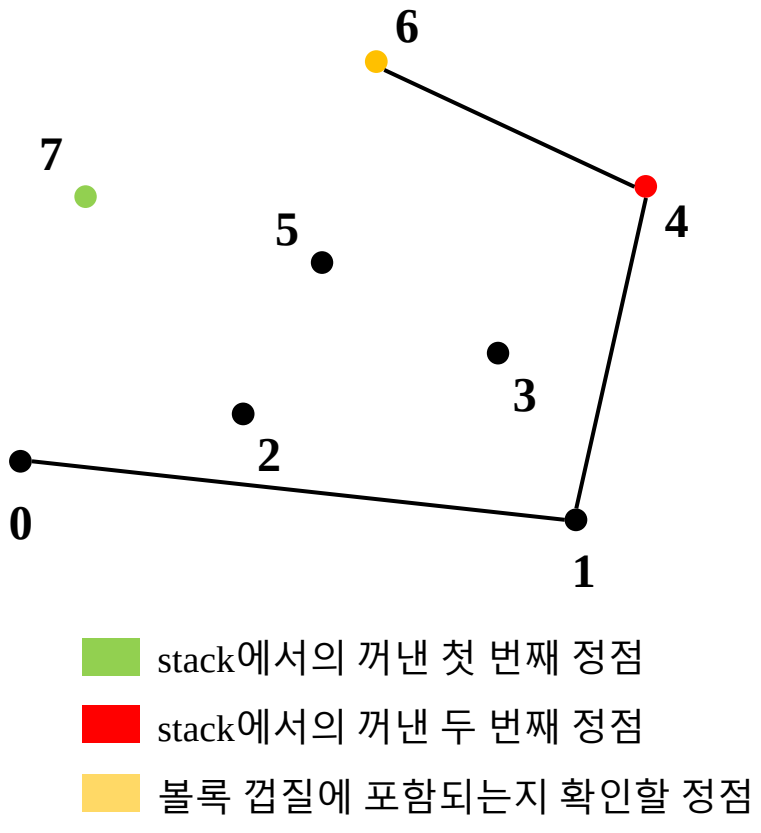
first: **4**
second: **6**
point: **7**

ccw(first, second, point) > 0

⇒ **point**를 stack에 넣기!

stack

0 **1** **4** **6**

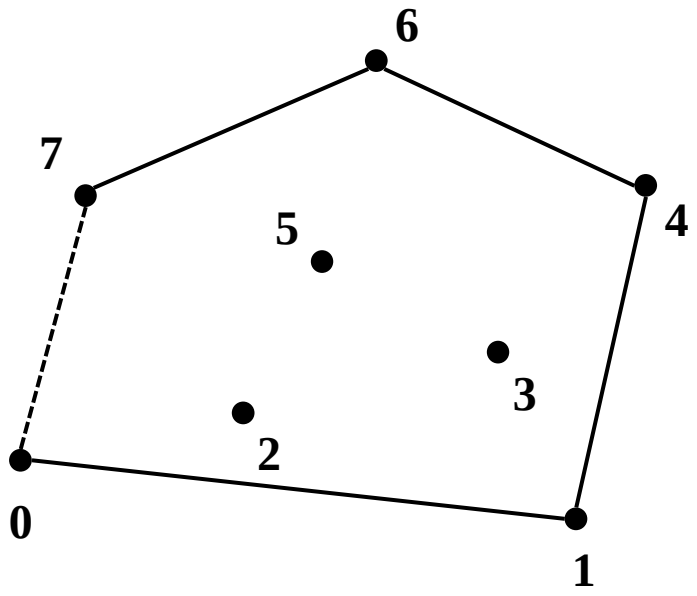


Convex Hull 볼록 껍질

종료!

1. 점을 반시계 방향으로 **정렬** → O(N log N)
2. 모든 점들이 **최대 1번** stack에 push, pop → O(N)

⇒ 시간 복잡도 **O(N log N)** (N = 점의 개수)



stack

0 1 4 6 7

Convex Hull 문제 풀이

볼록 껍질 기본 문제!

백준 1708
볼록 껍질

1708 볼록 껍질

문제: 볼록 껍질을 이루는 점의 개수를 구하는 문제

⇒ 볼록껍질을 구성하는 점을 stack(vector)에 넣어 구하자

⇒ 다각형의 모든 점이 일직선을 이루는 경우는 없으므로 안심하고 구하자!

볼록 껍질 코드

CCW와 전역변수

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
using Point = pair<long long, long long>;

Point point[100005];
int n;



int ccw(Point x, Point a, Point b) {
    Point vec1 = { a.first - x.first, a.second - x.second };
    Point vec2 = { b.first - x.first, b.second - x.second };
    long long det = vec1.first * vec2.second - vec1.second * vec2.first;
    if (det > 0)
        return 1;
    else if (det == 0)
        return 0;
    else return -1;
}
```

볼록 껍질 코드

사용자 정의 정렬 함수

```
23  // 거리 제곱 리턴
24  long long dist(Point a, Point b) {
25      return (a.first - b.first) * (a.first - b.first) + (a.second - b.second) * (a.second - b.second);
26  }
27
28  // 첫 번째 정렬 -> x값 작은 순
29  bool com(Point a, Point b) {
30      if (a.first == b.first)
31          return a.second < b.second;
32      return a.first < b.first;
33  }
34
35  // 두 번째 정렬 -> 기울기 순
36  bool com2(Point a, Point b) {
37      int temp = ccw(point[0], a, b);
38      if (temp != 0)
39          return temp > 0;
40      return dist(point[0], a) < dist(point[0], b);
41  }
```

볼록 껍질 코드

볼록 껍질 구현

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n;
    for (int i = 0; i < n; i++) {
        long long a, b; cin >> a >> b;
        point[i] = { a, b };
    }
    // 정렬
    sort(point, point + n, com);
    sort(point + 1, point + n, com2);
    vector<Point> convex;
    // 기본적으로 두개 넣기
    convex.push_back(point[0]);
    convex.push_back(point[1]);
    for (int i = 2; i < n; i++) {
        while (convex.size() >= 2) { // 2개보다 많으면
            Point second = convex.back();
            convex.pop_back();
            Point first = convex.back();
            if (ccw(first, second, point[i]) > 0) { // second가 point[i]보다 바깥쪽
                convex.push_back(second);
                break;
            }
        }
        convex.push_back(point[i]);
    }
    cout << convex.size();
    return 0;
}
```

볼록 다각형의 성질

Q. 어떤 점들이 볼록 다각형을 이루고 있다고 한다면...?

⇒ 볼록 다각형 내부(경계 포함)의 **가장 먼** 두 점 간의 거리를 **빠르게** 구할 수 있다

⇒ 회전하는 캘리퍼스 Rotating Calipers

⇒ 어떤 점이 볼록 다각형 내부에 있는지 없는지 **빠르게** 구할 수 있다

⇒ 내부 점 판정 Point in Convex Polygon Check

Rotating Calipers 회전하는 캘리퍼스

Rotating Calipers: rotating(회전하는) + calipas(길이를 재는 도구)

⇒ 지름의 길이를 돌면서 구함

⇒ **Two pointer**를 이용한 지름 재기 $O(n)$

※ 알고 넘어가야할 사실

점들 중 가장 거리가 먼 두 점은 모두 볼록 껍질 위에 있다

Rotating Calipers 회전하는 캘리퍼스

- Rotating Calipers algorithm

1. 볼록껍질 위의 점들에 반시계 방향으로 $P_0, P_1, P_2, \dots, P_{n-1}$라고 하고

   $A = \overline{P_i P_{i+1}}$ $(i = 0, 1, 2, \dots, n-2)$, $B = \overline{P_j P_{j+1}}$ $(j = i+1, i+2, \dots, n-2)$ 라고 하자

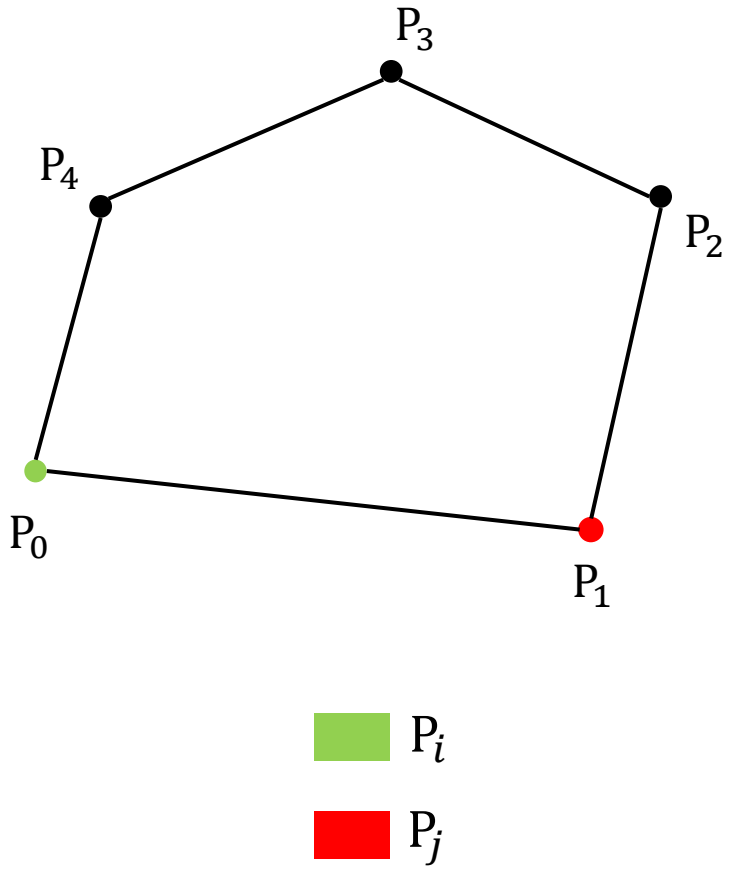
2. $\overline{P_i P_{i+1}}$ 의 길이를 재고, **CCW(A, B) > 0 이면 j 증가, CCW(A, B) < 0이면 i 증가**

3. j가 n - 1이 되기 전까지 반복

다음 페이지의 그림을 보면서 알고리즘을 익혀 봅시다!

Rotating Calipers 회전하는 캘리퍼스

Step1. 볼록 껍질 위의 점 찾기

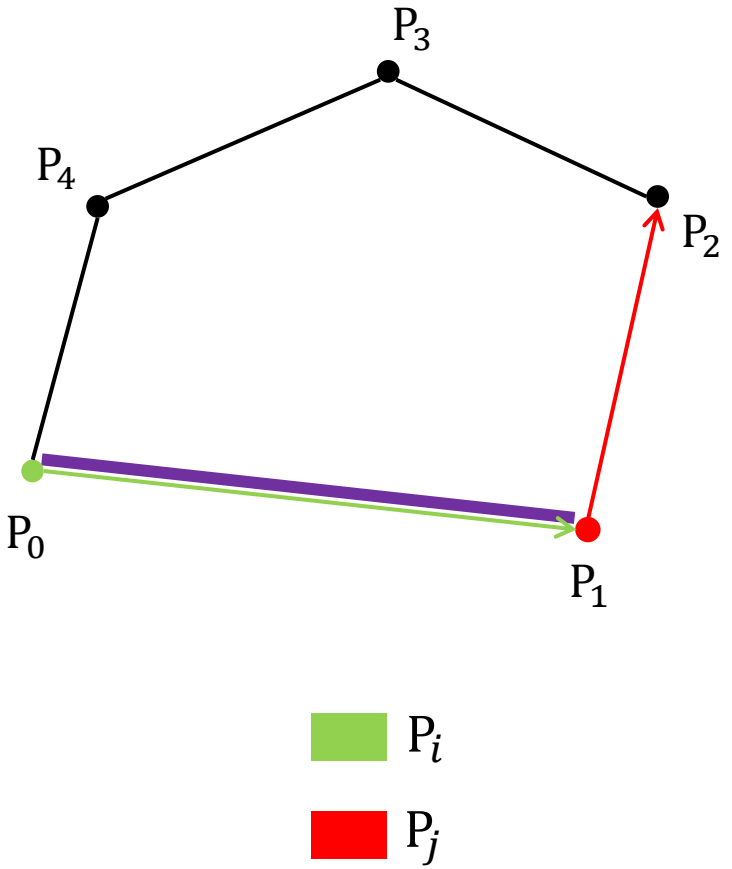


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) > 0

⇒ j 증가!

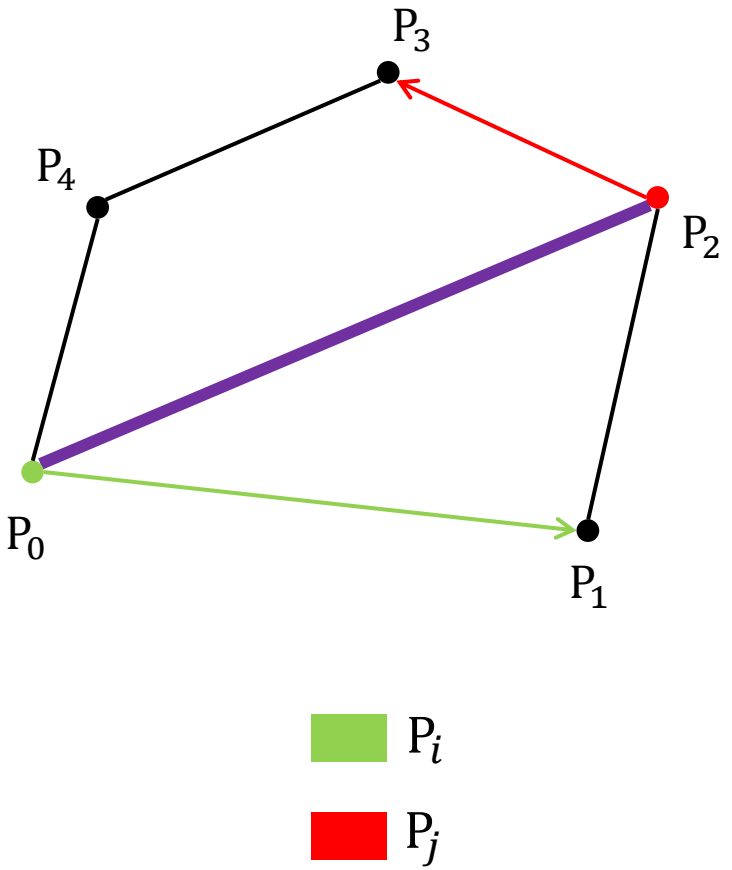


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) > 0

⇒ j 증가!

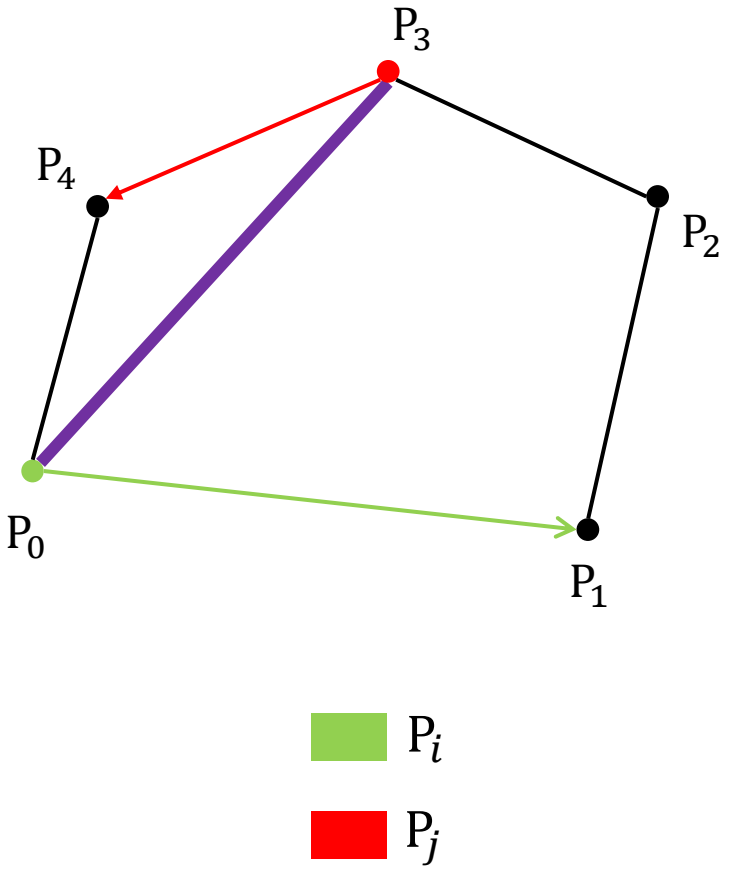


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) < 0

⇒ i 증가!

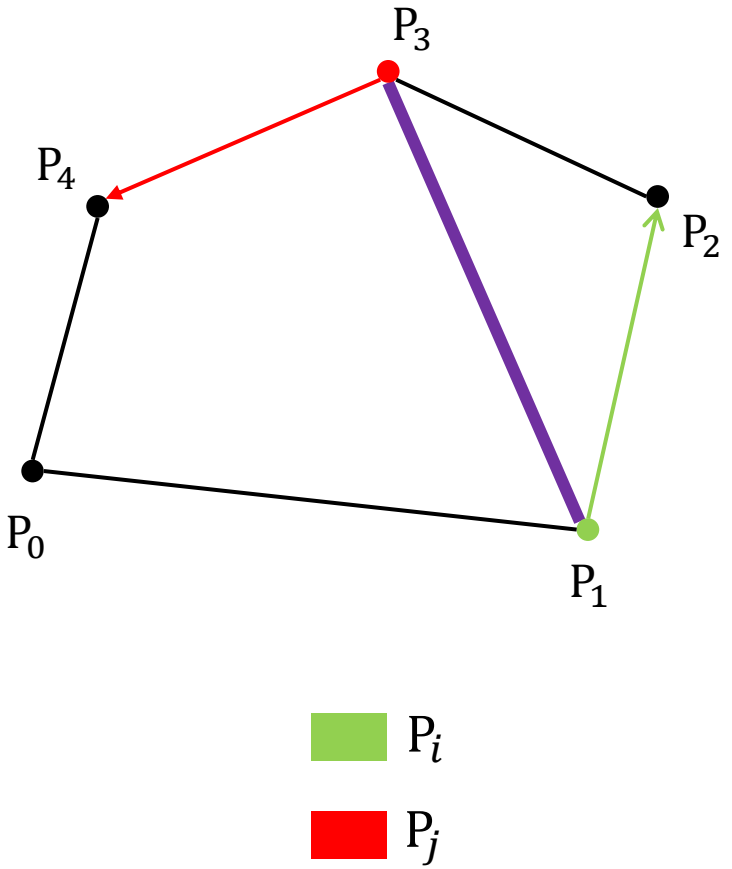


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) > 0

⇒ j 증가!

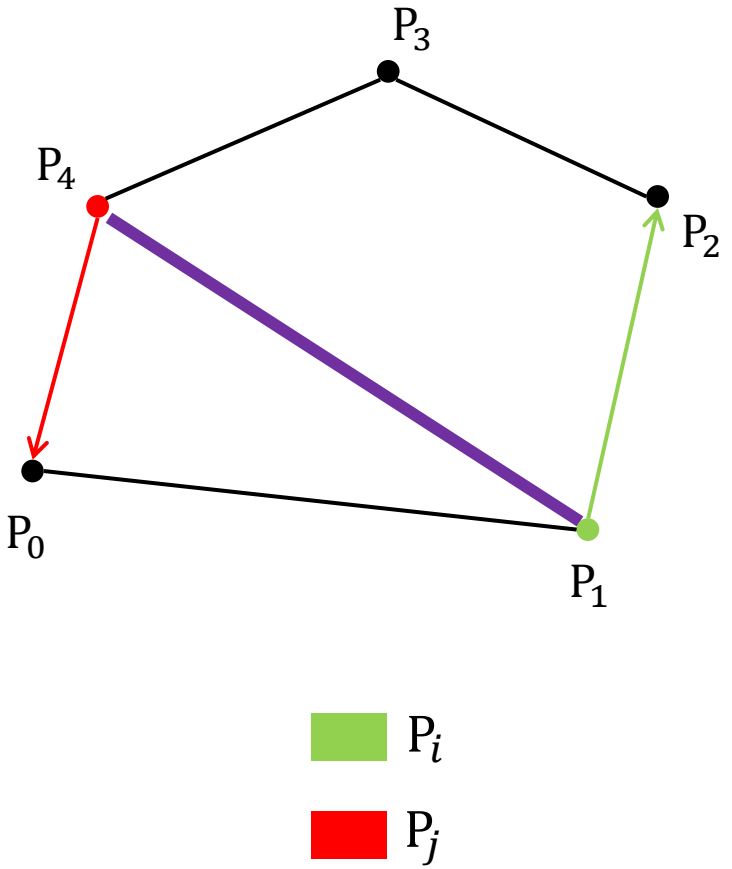


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) < 0

⇒ i 증가!

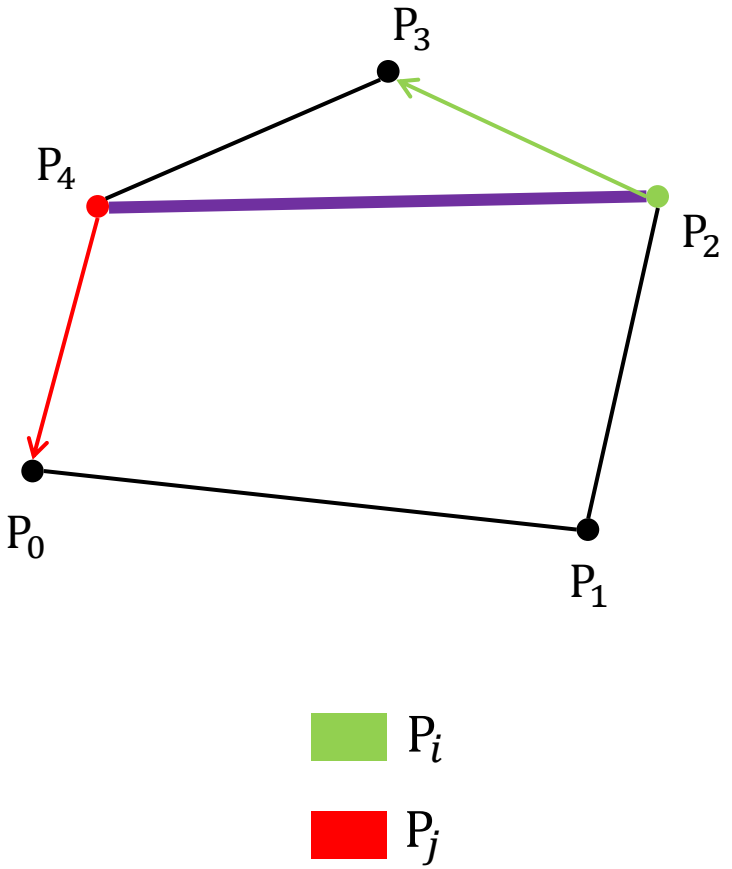


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

CCW(초록, 빨강) > 0

⇒ j 증가!

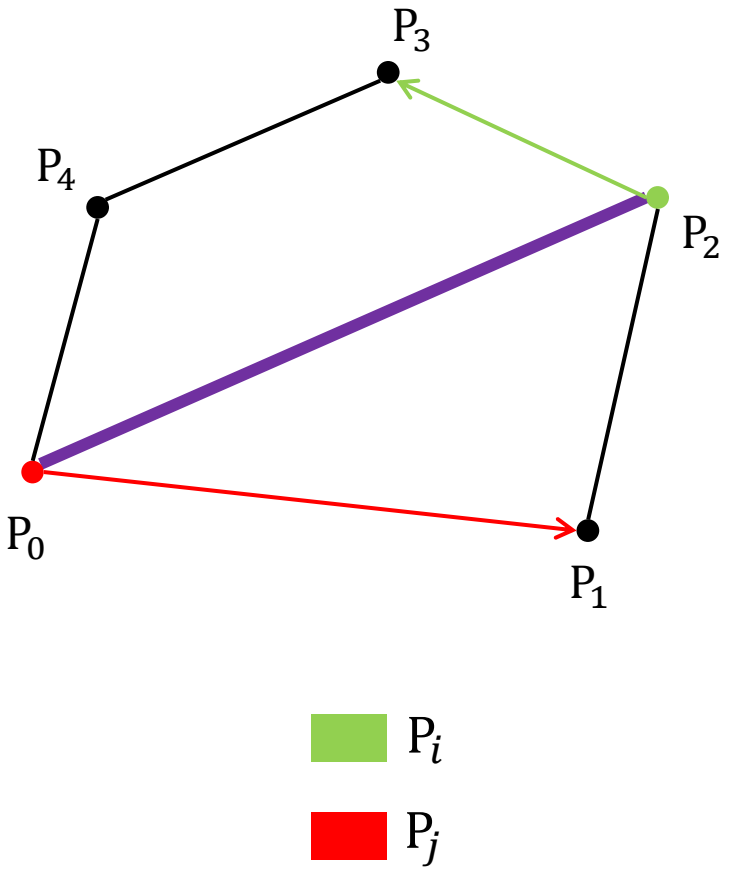


Rotating Calipers 회전하는 캘리퍼스

Step2. 거리 측정과 CCW

j가 n - 1을 넘음

⇒ 종료!



Rotating Calipers 문제풀이

백준 9240
로버트 후드

9240 로버트 후드

문제: 가장 먼 두 화살의 거리를 출력하는 문제

$\Rightarrow 2 \leq C \leq 100{,}000$ 이므로 브루트 포스로 풀면 $O(C^2)$로 **시간 초과**

$\Rightarrow$ **회전하는 캘리퍼스를 이용**해 시간복잡도를 $O(C \log C)$로 줄이자!

로버트 후드 코드

볼록 껍질 구하기(관련 함수는 생략)

```
48  int main() {
49      ios_base::sync_with_stdio(false);
50      cin.tie(NULL);
51      cout.tie(NULL);
52      cin >> c;
53      for (int i = 0; i < c; i++) {
54          int a, b; cin >> a >> b;
55          point[i] = { a, b };
56      }
57      // 볼록 껍질 구하기
58      sort(point, point + c, com);
59      sort(point + 1, point + c, com2);
60      convex.push_back(point[0]);
61      convex.push_back(point[1]);
62      for (int i = 2; i < c; i++) {
63          while (convex.size() >= 2) {
64              Point second = convex.back();
65              convex.pop_back();
66              Point first = convex.back();
67              if (ccw(first, second, point[i]) > 0) {
68                  convex.push_back(second);
69                  break;
70              }
71          }
72          convex.push_back(point[i]);
73      }
```

로버트 후드 코드

회전하는 캘리퍼스 구현 부분

```
74      double an = 0;
75      int cnt = convex.size();
76      int i = 0;
77      int j = 1;
78      if (cnt == 2) { // 볼록껍질이 직선일때
79          an = dist(convex[0], convex[1]);
80      }
81      else {
82          while (j < cnt && i < j) { //반바퀴 돌 때까지
83              Point vec1 = { convex[(i + 1) % cnt].first - convex[i % cnt].first, convex[(i + 1) % cnt].second - convex[i % cnt].second };
84              Point vec2 = { convex[(j + 1) % cnt].first - convex[j % cnt].first, convex[(j + 1) % cnt].second - convex[j % cnt].second };
85              an = min(an, dist(convex[i], convex[j % cnt])); // 거리구함
86              if (ccw({ 0,0 }, vec1, vec2) > 0) // 두 번째 벡터 이동할 때
87                  j++;
88              else
89                  i++;
90          }
91      }
92      cout << fixed;
93      cout.precision(12);
94      cout << an;
95  }
```

볼록 다각형 내부 점 판정 Point in Convex Polygon Check

이분 탐색을 이용한 볼록 다각형 내부 점 판정

1. 볼록껍질 위의 점들에 반시계 방향으로 $P_0, P_1, P_2, \dots, P_{n-1}$라고 하고 한점을 X라고 하자
2. $CCW(P_0, P_1, X) < 0 \,||\, CCW(P_0, P_{n-1}, X) > 0$ 이면 **X는 다각형 외부**에 있다
3. 1과 n-1사이에서 이분탐색을 하여 X가 내부에 있을 수 있는 삼각형 P_0, P_i, P_j를 구한다
4. $ccw(P_i, P_j, X) \geq 0$이면 **X는 다각형 내부**에 있다

※ 시간복잡도: $O(n \log n)$

다음 페이지의 그림을 보면서 알고리즘을 익혀 봅시다!

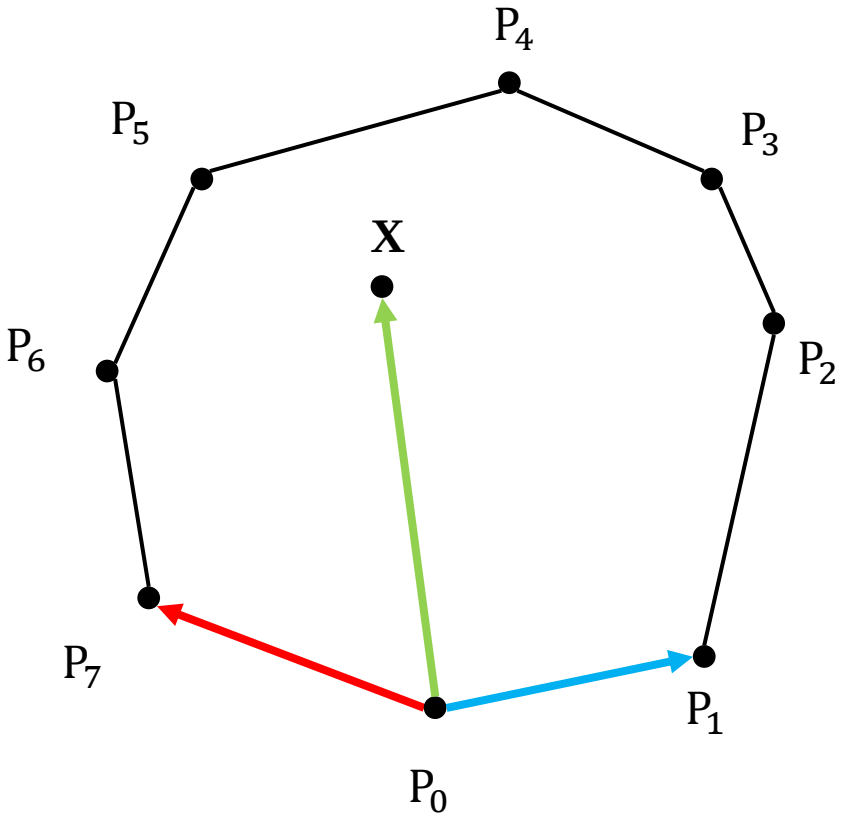
볼록 다각형 내부 점 판정 Point in Convex Polygon Check

Step1. CCW(P_0, P_1, X)와 CCW(P_0, P_{n-1}, X) 확인

$CCW(P_0, P_1, X) > 0$

$CCW(P_0, P_{n-1}, X) < 0$

$\Rightarrow$ **내부**에 있을 수 있다



볼록 다각형 내부 점 판정 Point in Convex Polygon Check

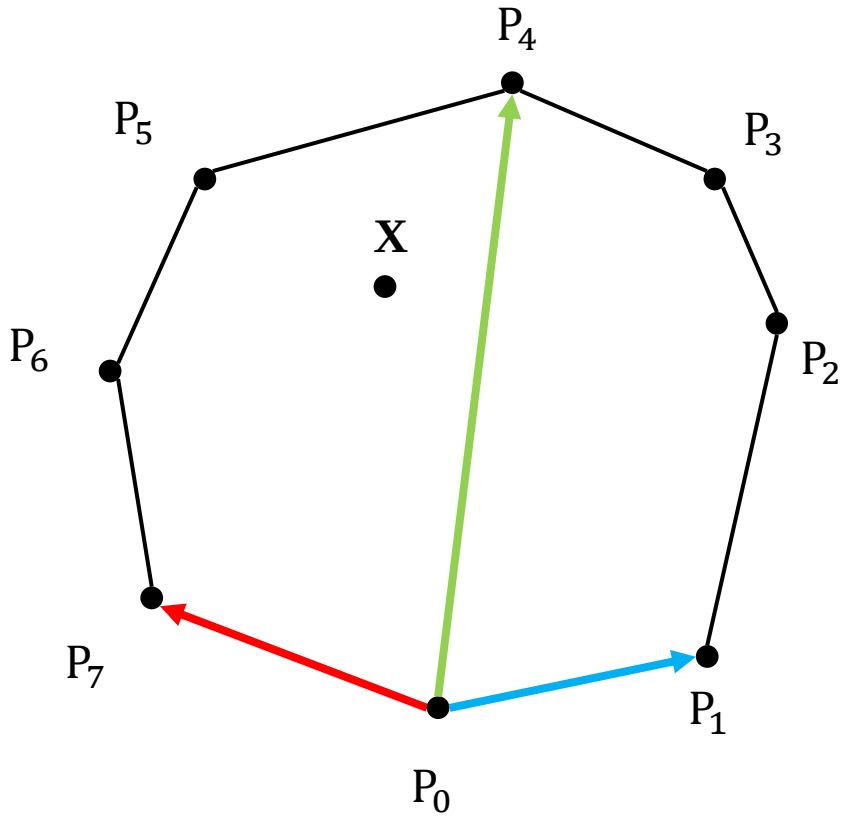
Step2. X가 내부에 있을 수 있는 삼각형 P_0, P_i, P_j를 구하기

※ start + 1 == end 이면 종료!

start: 1
end: 7
mid: **4**

CCW(P_0, P_{mid}, X) > 0

⇒ **start = mid**



볼록 다각형 내부 점 판정 Point in Convex Polygon Check

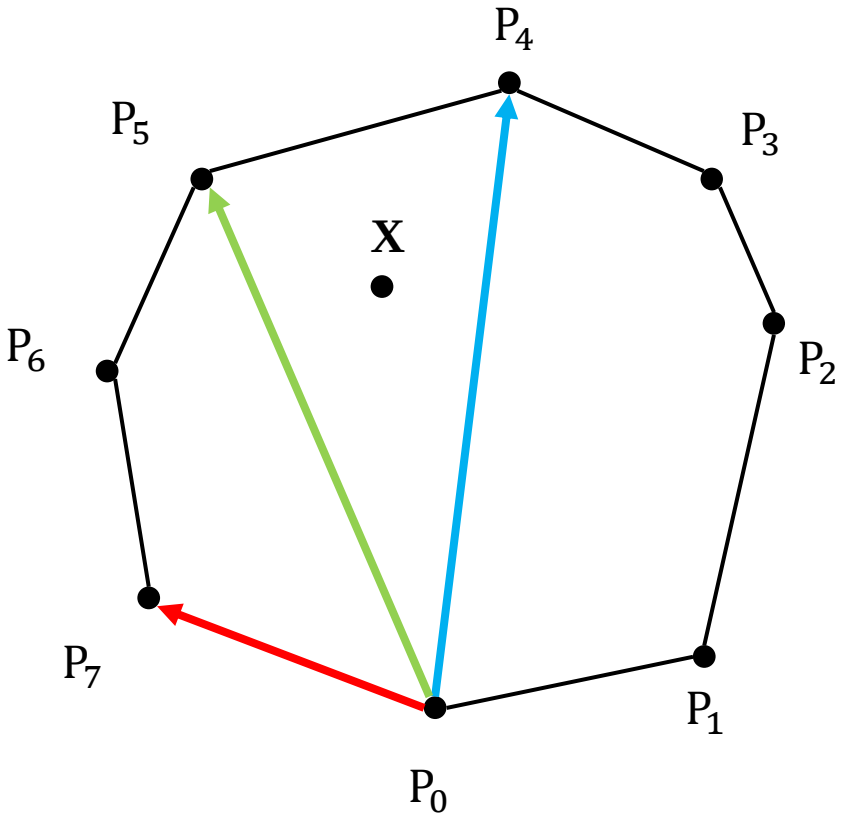
Step2. X가 내부에 있을 수 있는 삼각형 P_0, P_i, P_j를 구하기

※ start + 1 == end 이면 종료!

start: 4
end: 7
mid: **5**

$CCW(P_0, P_{mid}, X) < 0$

⇒ **end = mid**



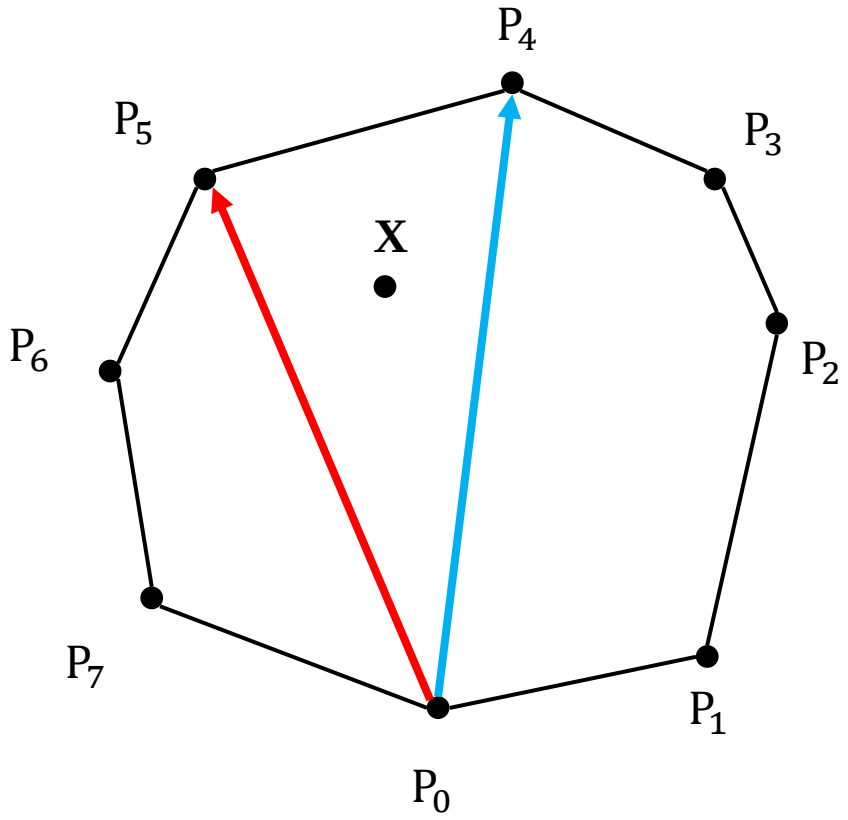
볼록 다각형 내부 점 판정 Point in Convex Polygon Check

Step2. X가 내부에 있을 수 있는 삼각형 P_0, P_i, P_j를 구하기

※ start + 1 == end 이면 종료!

start: 4
End: 5
mid: **4**

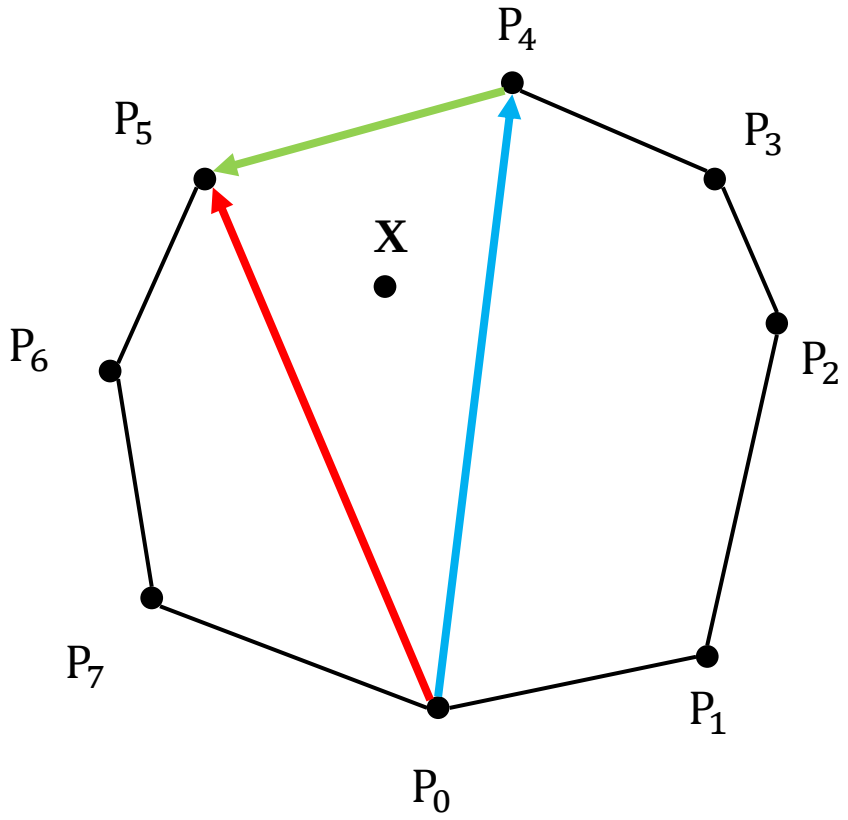
start + 1 == end이므로 종료!



볼록 다각형 내부 점 판정 Point in Convex Polygon Check

Step3. CCW(P_{start}, P_{end}, X) 확인

start: 4
end: 5
CCW(P_{start}, P_{end}, X) > 0



⇒ X는 볼록 다각형 **내부**에 있다!!!

볼록 다각형 내부 점 판정 문제풀이

백준 20670
미스테리 싸인

20670 미스테리 싸인

문제: 두 개의 볼록 껍질 A , B에 대해 A외부 또는 B 내부인 점의 개수를 구하는 문제

⇒ 점의 개수(k)만큼 A, B에 대해 점 내부 판정을 시도하자

⇒ $3 \leq n, m \leq 10{,}000$ $2 \leq K \leq 300{,}000$이므로 **이분탐색 이용!**

⇒ A, B는 **이미 반시계 방향** 순서로 주어지니 따로 정렬할 필요 없다!

미스테리 싸인 코드

볼록 다각형 내부 점 판정 함수

```
23  // 내부에 점 확인, 있으면 1 리턴
24  int check(Point* point, Point x, int cnt) {
25      // 밖에 있으면
26      if (ccw(point[0], point[1], x) < 0 || ccw(point[0], point[cnt - 1], x) > 0) {
27          return 0;
28      }
29      int start = 1;
30      int end = cnt - 1;
31      // 이분 탐색, 삼각형이 생길때까지
32      while (start + 1 < end) {
33          int mid = (start + end) / 2;
34          if (ccw(point[0], point[mid], x) > 0) {
35              start = mid;
36          }
37          else end = mid;
38      }
39
40      if (ccw(point[start], point[end], x) < 0)   //밖에 있으면
41          return 0;
42      return 1;
43  }
```

미스테리 싸인 코드

main

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n >> m >> k;
    for (int i = 0; i < n; i++) {
        long long a, b; cin >> a >> b;
        pointA[i] = { a, b };
    }
    for (int i = 0; i < m; i++) {
        long long a, b; cin >> a >> b;
        pointB[i] = { a, b };
    }
    for (int i = 0; i < k; i++) {
        long long a, b; cin >> a >> b;
        //A 밖 또는 B 안에 있으면 조건 위반
        if (check(pointA, { a ,b }, n) == 0 || check(pointB, { a ,b }, m) == 1)
            an++;
    }

    if (an == 0) {
        cout << "YES";
    }
    else cout << an;
    return 0;
}
```

Problem Set – 선분 교차 판정

1688 지민이의 테러

20149 선분 교차3

10255 교차점

Problem Set – Convex Hull

3679 단순 다각형

17403 가장 높고 넓은 성

10839 미술관

Problem Set – Rotating Calipers

1310 달리기 코스

10254 고속도로

13310 먼 별

Problem Set – 볼록 다각형 내부 점 판정

2254 감옥 건설

9875 Cow Curling

3878 점 분리

Problem Set – 도전!

1077 넓이

내부 점 판정, 교점 판정, 넓이 구하기 → 기하 종합 세트 문제

2125 좀

선분이 볼록 다각형 내부에 있는지 판정 → 그래프 **잘** 그리기