

Geometry (1)

나정휘

<https://justicehui.github.io/>

목차

- 계산 기하
- 2차원 기하 물체의 표현
- CCW
- 선분 교차 판별
- 다각형 내부 판별
- 볼록 껍질
- 볼록 다각형 내부 판별
- 가장 먼 두 점
- 가장 가까운 두 점
- 볼록 다각형의 접선을 이용한 최적화

계산 기하

계산 기하

- 기하학적 도형을 대상으로 하는 문제를 알고리즘적 방법으로 해결하는 분야
 - 점
 - 직선, 반직선, 선분, 곡선
 - 다각형, 원
 - 다면체, 구

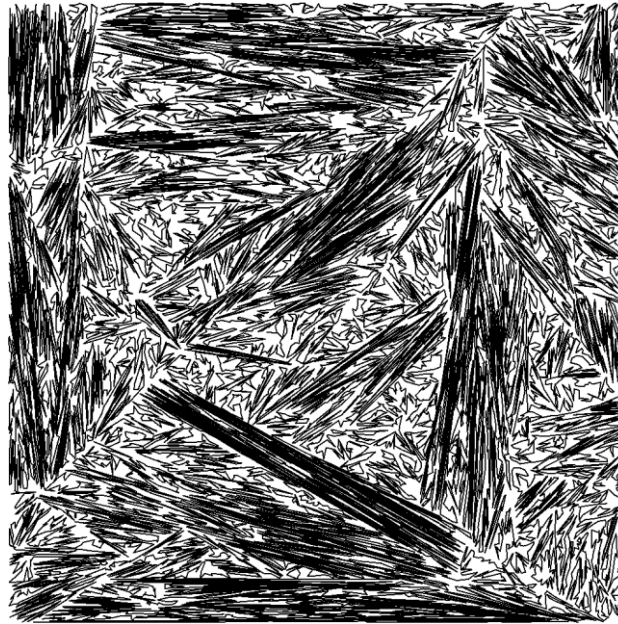
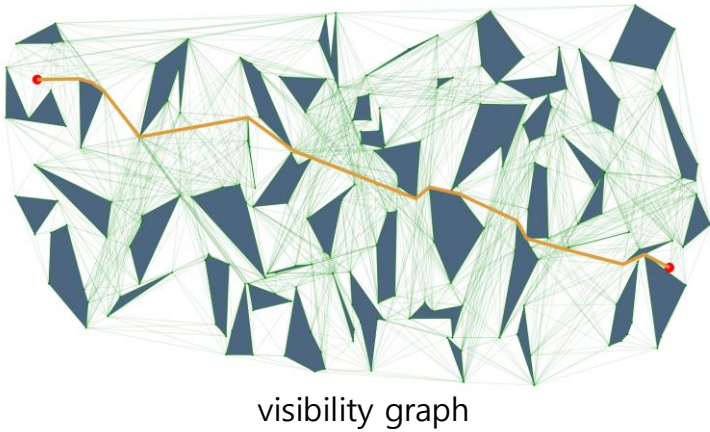
계산 기하

- 기하학적 도형을 대상으로 하는 문제를 알고리즘적 방법으로 해결하는 분야
 - 점
 - 직선, 반직선, 선분, 곡선
 - 다각형, 원
 - 다면체, 구

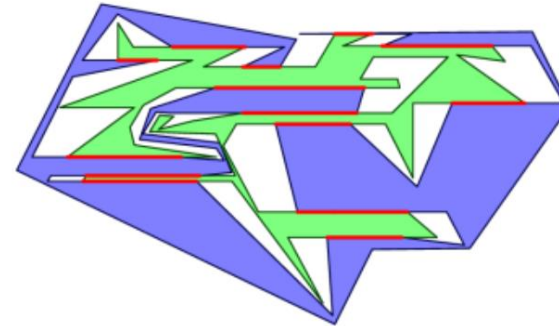


계산 기하

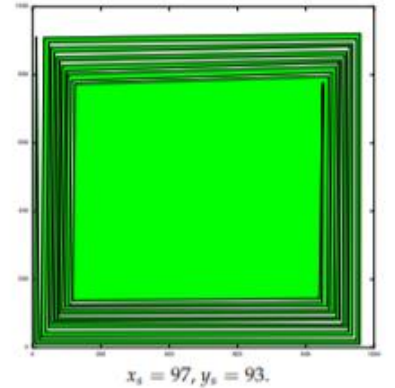
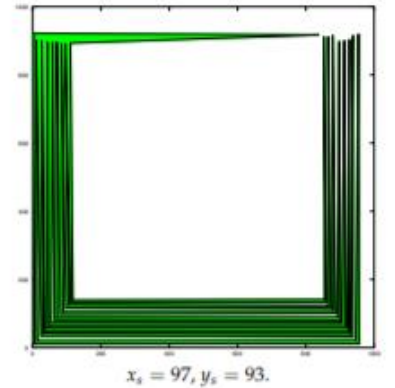
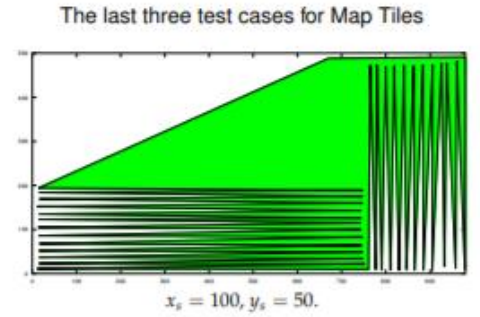
- 구현하기 귀찮음
- 수많은 예외 처리
- 쉬운 것만 다룰 예정



놀랍게도 다각형



두 "단순" 다각형의 접선



ㅎㅎ;;

2차원 물체의 표현

2차원 물체의 표현

- 3차원 부터는 웬만하면 올림피아드에 안 나옴
 - 왜?
 - 3차원은 많이 어렵고 귀찮다.
 - 2차원으로 충분히 어려운 문제를 많이 낼 수 있다.
- 0~2차원에서 사용하는 물체들
 - 0차원 : 점
 - 1차원 : 직선, 선분, 곡선
 - 2차원 : 다각형, 원, 곡면

2차원 물체의 표현

- 점
 - 좌표 평면 위에 있는 점
 - (x, y)
 - std::pair

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;

using Point = pair<int, int>;

int main(){
    Point pt(1, 2);
    cout << pt.x << " " << pt.y << endl;
}
```

2차원 물체의 표현

- 두 점 $(x_1, y_1), (x_2, y_2)$ 를 지나는 직선
- $y = ax + b$
 - 일차 함수로 표현
 - 기울기 $a = (y_2 - y_1) / (x_2 - x_1)$
 - 절편 $b = y_1 - ax_1$
 - 장점 : 교점을 구하기 쉬움, 점과 동일하게 취급할 수 있음
 - 단점 : 기울기가 무한대일 때 예외 발생, 선분을 표현하기 힘들
- $ax + by + c = 0$ 으로 표현하는 경우도 존재
- 기울기 무한대를 표현할 수 있지만 수식이 더러워짐

2차원 물체의 표현

- 두 점 (x_1, y_1) , (x_2, y_2) 를 지나는 직선
- `pair<Point, Point>`
 - 두 점을 지나는 직선이니까 두 점을 저장한다는 아이디어
 - 장점 : 선분을 표현할 때 편함
 - 단점 : 직선을 표현할 때 불편함

2차원 물체의 표현

- 두 점 (x_1, y_1) , (x_2, y_2) 를 지나는 직선
- $x(t)$, $y(t)$
 - 매개변수로 표현
 - $x(t) = x_1 + (x_2 - x_1)t$
 - $y(t) = y_1 + (y_2 - y_1)t$
 - $-\infty < t < \infty$ 면 직선을 표현
 - $0 \leq t \leq 1$ 이면 선분을 표현
 - 장점 : 직선 위의 점을 깔끔하게 표현, 예외처리가 적음
 - 단점 : 익숙하지 않으면 실수하기 쉬움

2차원 물체의 표현

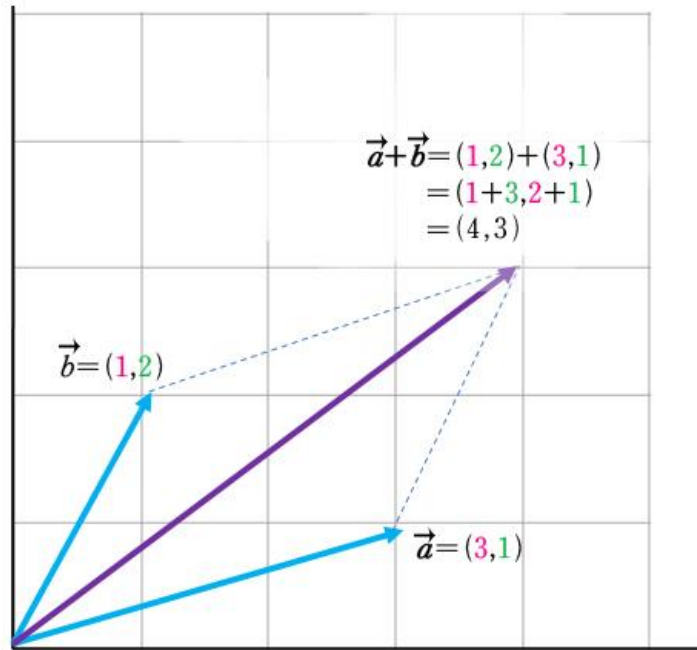
- 다각형
 - 다각형에 속한 점 $p_1, p_2, p_2, \dots, p_n$ 을 순서대로 배열에 저장
 - 어떤 순서대로?
 - 보통 반시계 방향으로 저장함

질문

CCW

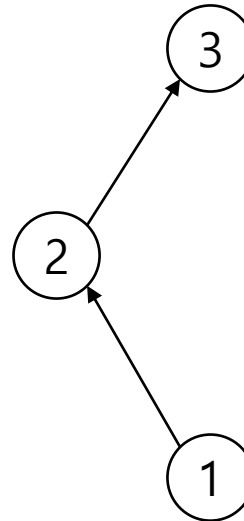
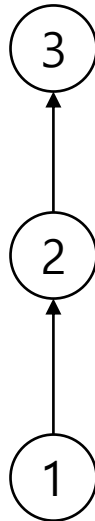
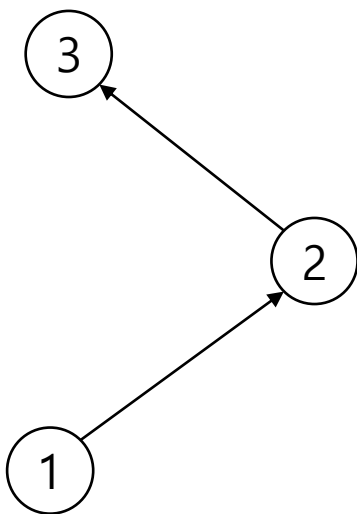
벡터

- 사실 점은 벡터로 표현할 수 있음 (방향, 크기)
- 벡터의 덧셈, 뺄셈
- 벡터의 내적, 외적 (?)



CCW

- CCW (CounterClockWise)
 - 두 벡터가 시계 방향인지 반시계 방향인지 판별하는 방법
 - 시계 방향이면 음수, 반시계 방향이면 양수, 일직선이면 0 반환



CCW

- 신발끈 공식
 - 세 점 $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ 을 꼭짓점으로 하는 삼각형의 넓이
 - $\frac{1}{2} | (x_1y_2 + x_2y_3 + x_3y_1) - (x_2y_1 + x_3y_2 + x_1y_3) |$
 - 절댓값을 붙이는 이유는?
- 세 점이 시계 방향이면 음수가 나오고, 반시계 방향이면 양수가 나옴
- 저 식을 그대로 사용하면 된다.
 - 또는 $(x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1)$

CCW

- 오버플로우 주의

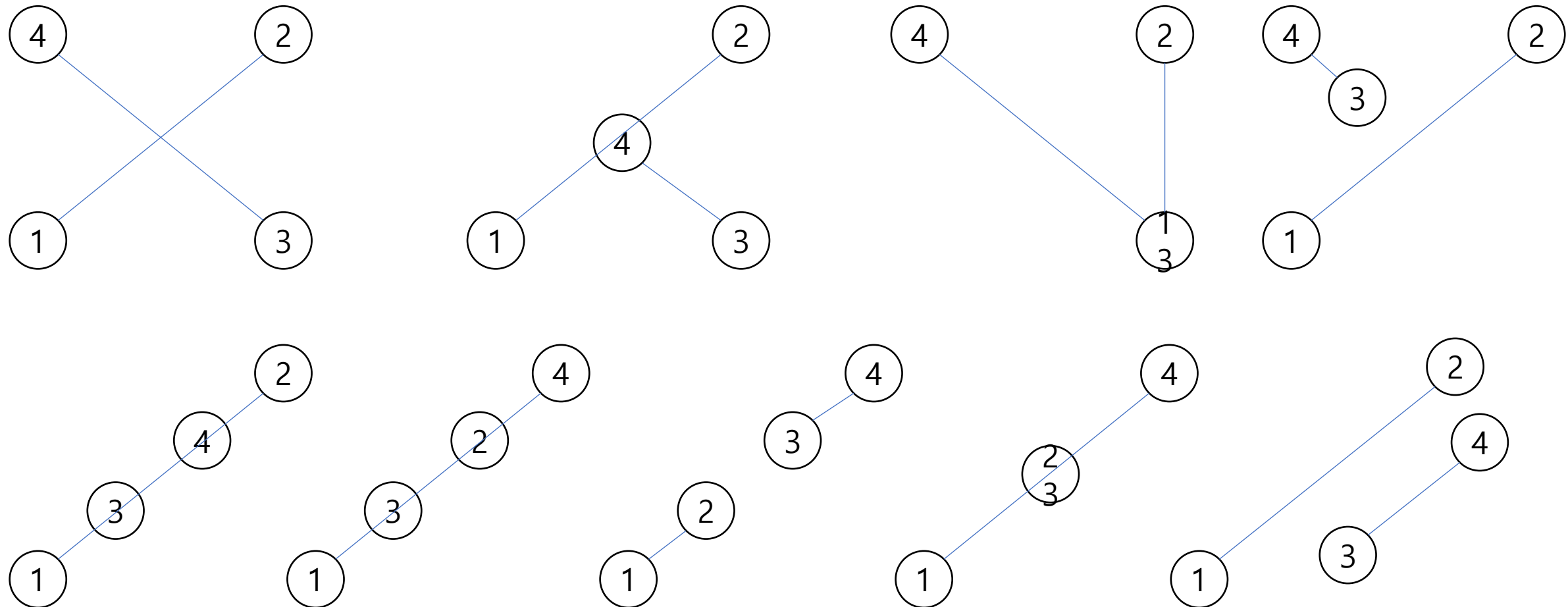
```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;

using ll = long long;
using Point = pair<ll, ll>;

int CCW(const Point &p1, const Point &p2, const Point &p3){
    ll res = p1.x*p2.y + p2.x*p3.y + p3.x*p1.y;
    res -= p2.x*p1.y + p3.x*p2.y + p1.x*p3.y;
    return (res > 0) - (res < 0);
}
```

선분 교차 판별

- 두 선분이 만나는지 판별



선분 교차 판별

- 예외가 너무 많아서 실수하기 쉬우므로 코드를 외우는 것이 편함

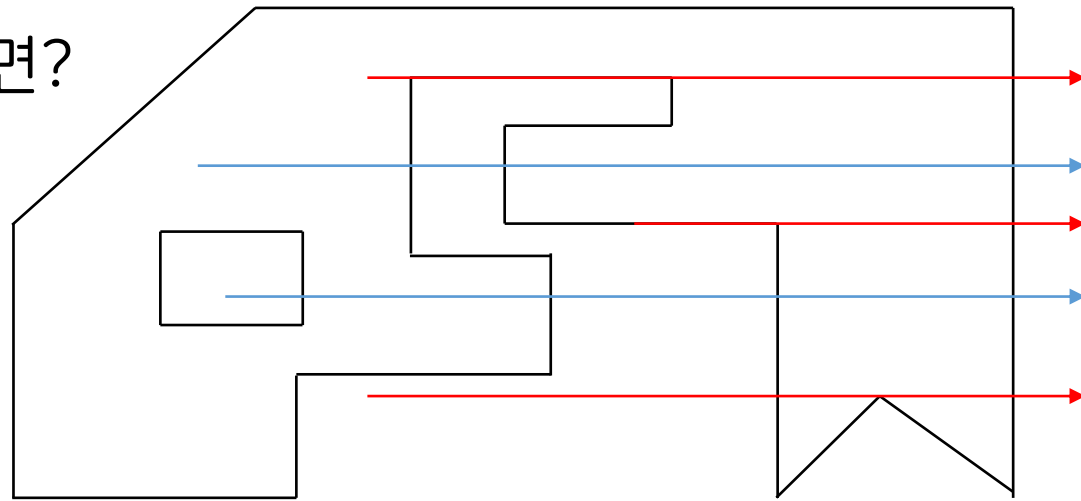
```
bool Cross(Point a, Point b, Point c, Point d){
    int ab = CCW(a, b, c) * CCW(a, b, d);
    int cd = CCW(c, d, a) * CCW(c, d, b);
    if(ab == 0 && cd == 0){
        if(a > b) swap(a, b);
        if(c > d) swap(c, d);
        return !(b < c || d < a);
    }
    return ab <= 0 && cd <= 0;
}
```

질문

다각형 내부 판별

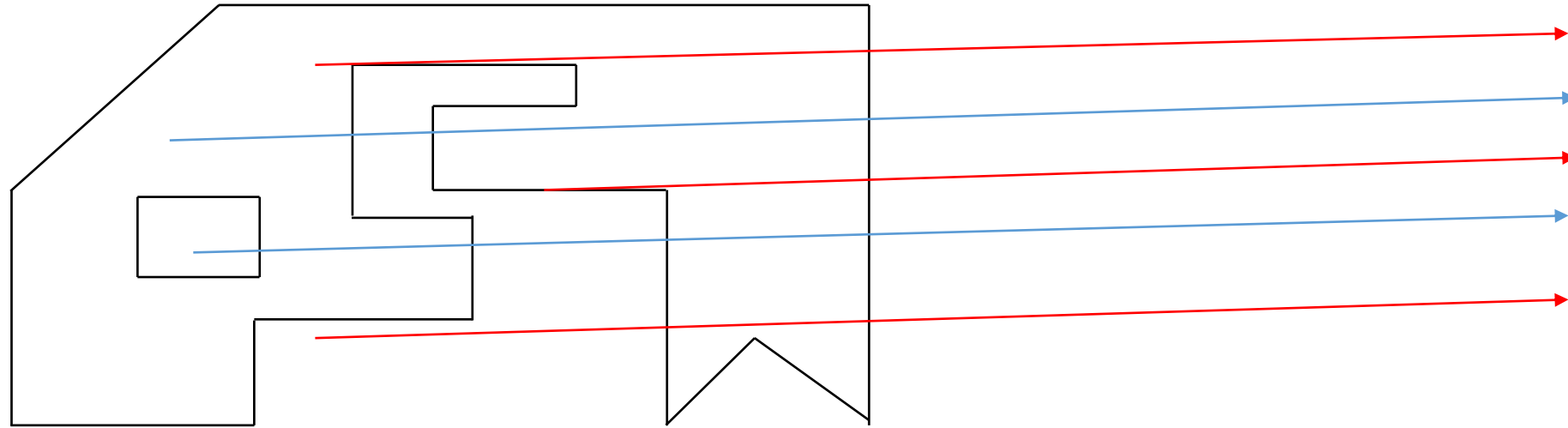
다각형 내부 판별

- 다각형과 어떤 점이 주어지면, 그 점이 다각형 내부에 있는지 판별
- 그 점에서 시작하는 반직선을 그린 뒤, 다각형과의 교점 개수를 세면 됨
 - 반직선은 (x, y) 와 (inf, y) 로 표현 가능
 - 홀수 : 내부
 - 짝수 : 외부
- 변/꼭짓점과 겹치면?



다각형 내부 판별

- 반직선을 (x, y) 와 $(\text{inf}, y+1)$ 로 표현하면 변/꼭짓점과 겹치지 않음
 - 반직선의 기울기가 $1/\text{inf}$ 가 되어서 (x, y) 를 제외한 어떠한 격자점에서도 다각형과 만나지 않음



다각형 내부 판별

- 반직선을 (x, y) 와 $(\text{inf}, y+1)$ 로 표현하면 변/꼭짓점과 겹치지 않음
 - 반직선의 기울기가 $1/\text{inf}$ 가 되어서 (x, y) 를 제외한 어떠한 격자점에서도 다각형과 만나지 않음

```
const ll MAX_COORD = 1e9;
bool Check(const vector<Point> &v, Point p1){
    int n = v.size(), cnt = 0;
    Point p2(MAX_COORD+1, p1.y+1);
    for(int i=0; i<n; i++){
        int j = i+1 == n ? 0 : i+1;
        cnt += Cross(v[i], v[j], p1, p2);
    }
    return cnt & 1;
}
```

질문

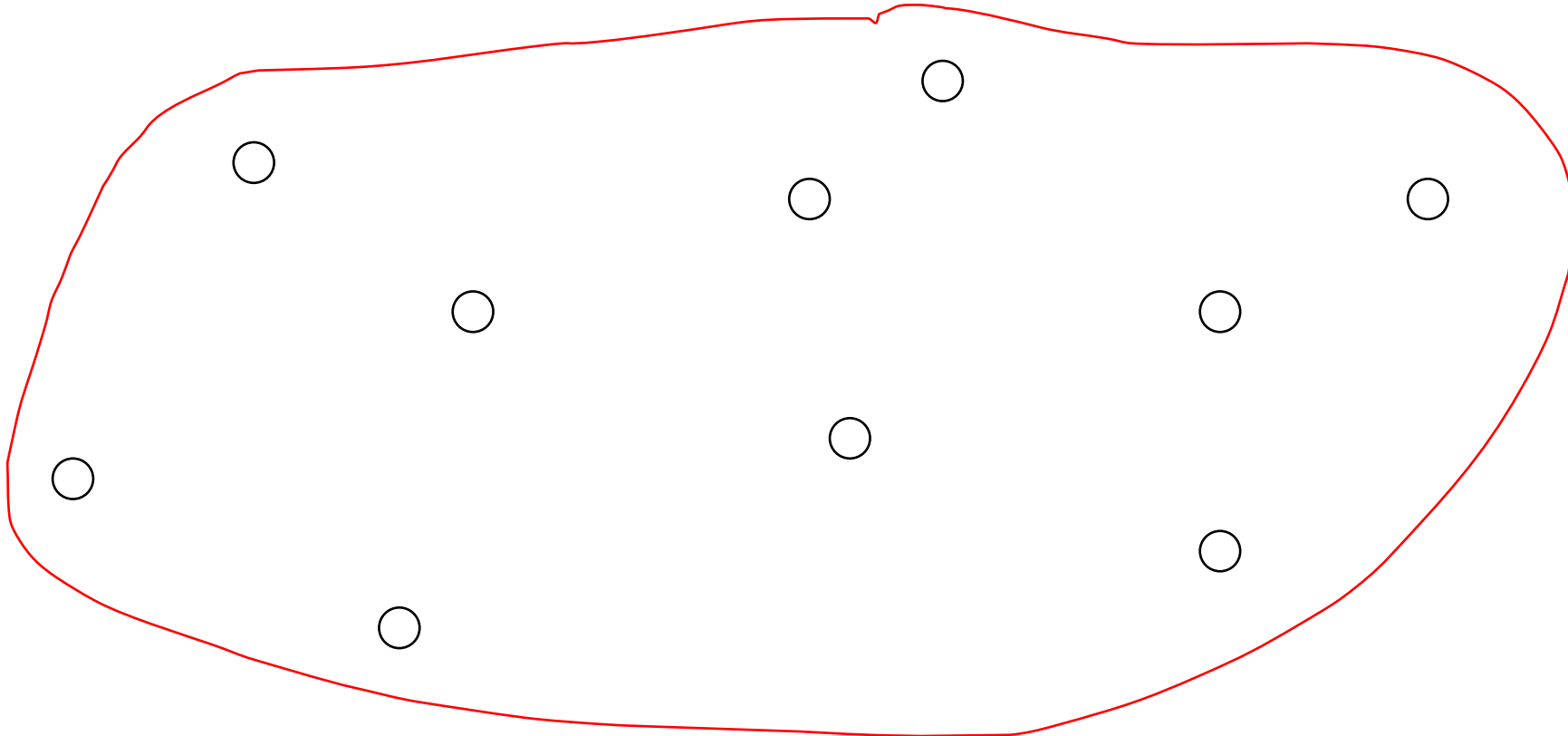
볼록 꺾질

볼록 껍질

- 볼록 다각형 : 모든 내각이 180도 미만인 다각형
- 볼록 껍질 : 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형
 - $O(N \log N)$ 에 구할 수 있다!

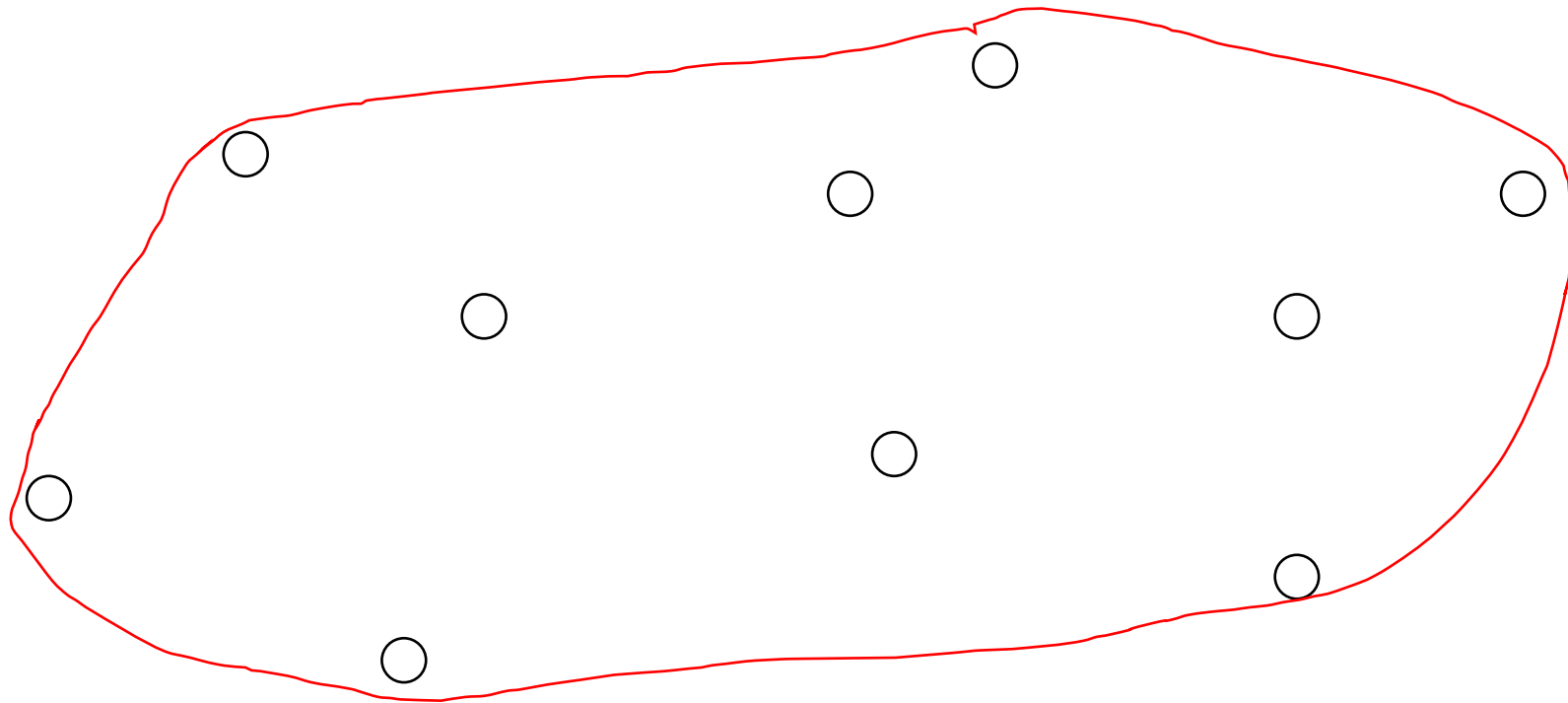
볼록 껍질

- 볼록 다각형 : 모든 내각이 180도 미만인 다각형
- 볼록 껍질 : 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형



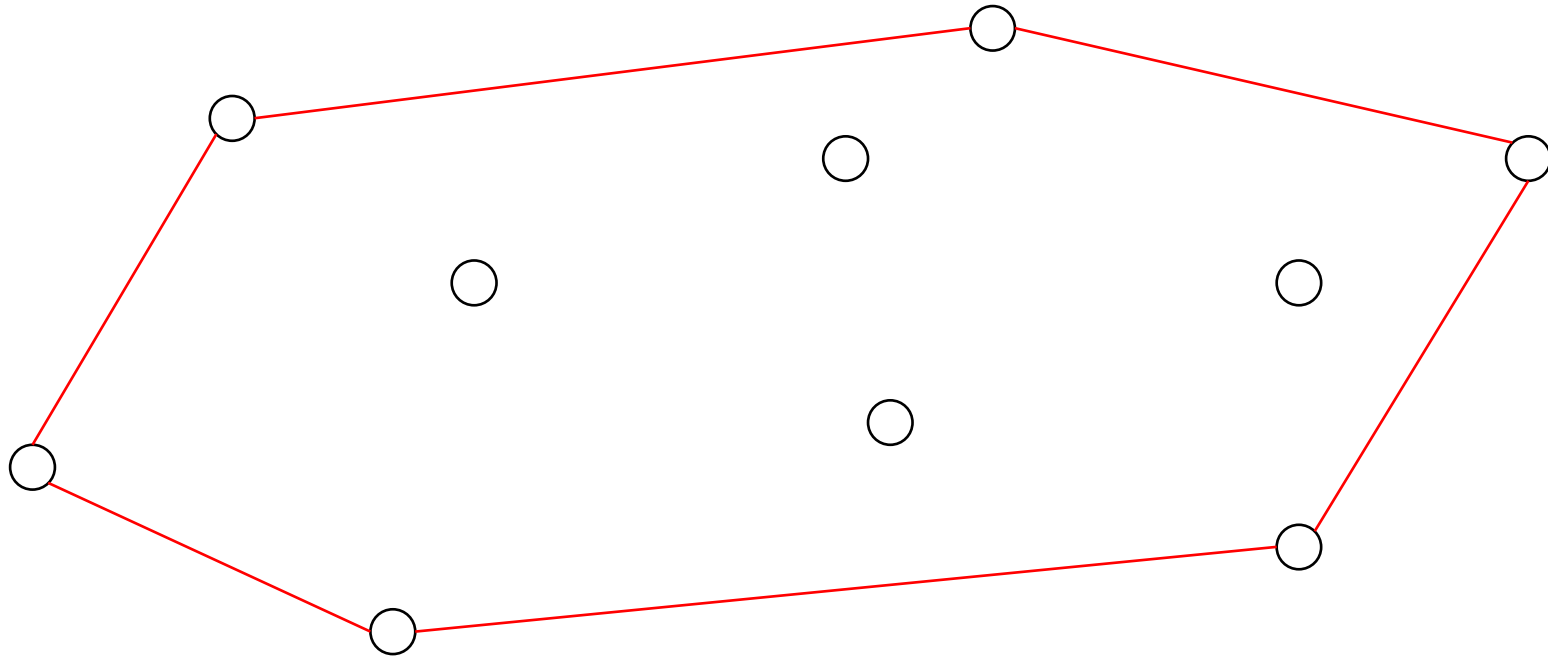
볼록 껍질

- 볼록 다각형 : 모든 내각이 180도 미만인 다각형
- 볼록 껍질 : 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형



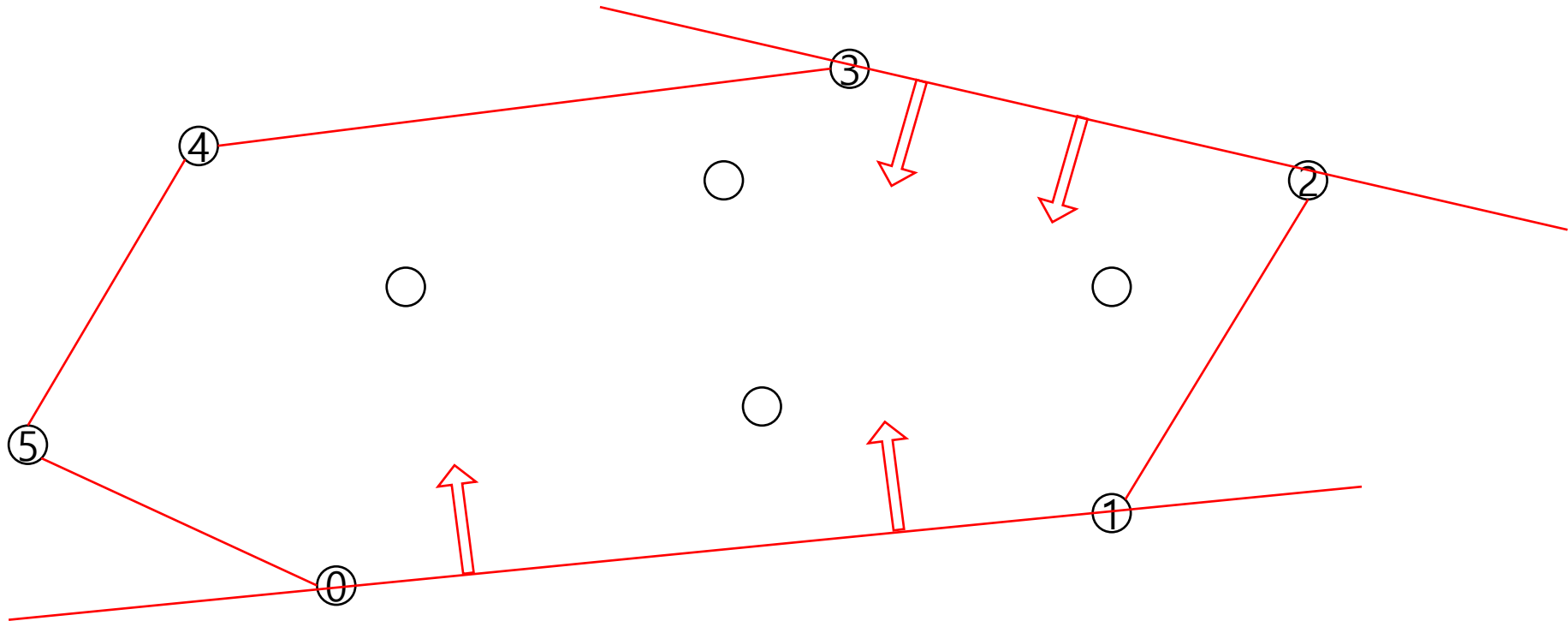
볼록 껍질

- 볼록 다각형 : 모든 내각이 180도 미만인 다각형
- 볼록 껍질 : 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형



볼록 껍질

- 볼록 다각형 : 모든 내각이 180도 미만인 다각형
- 볼록 껍질 : 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형

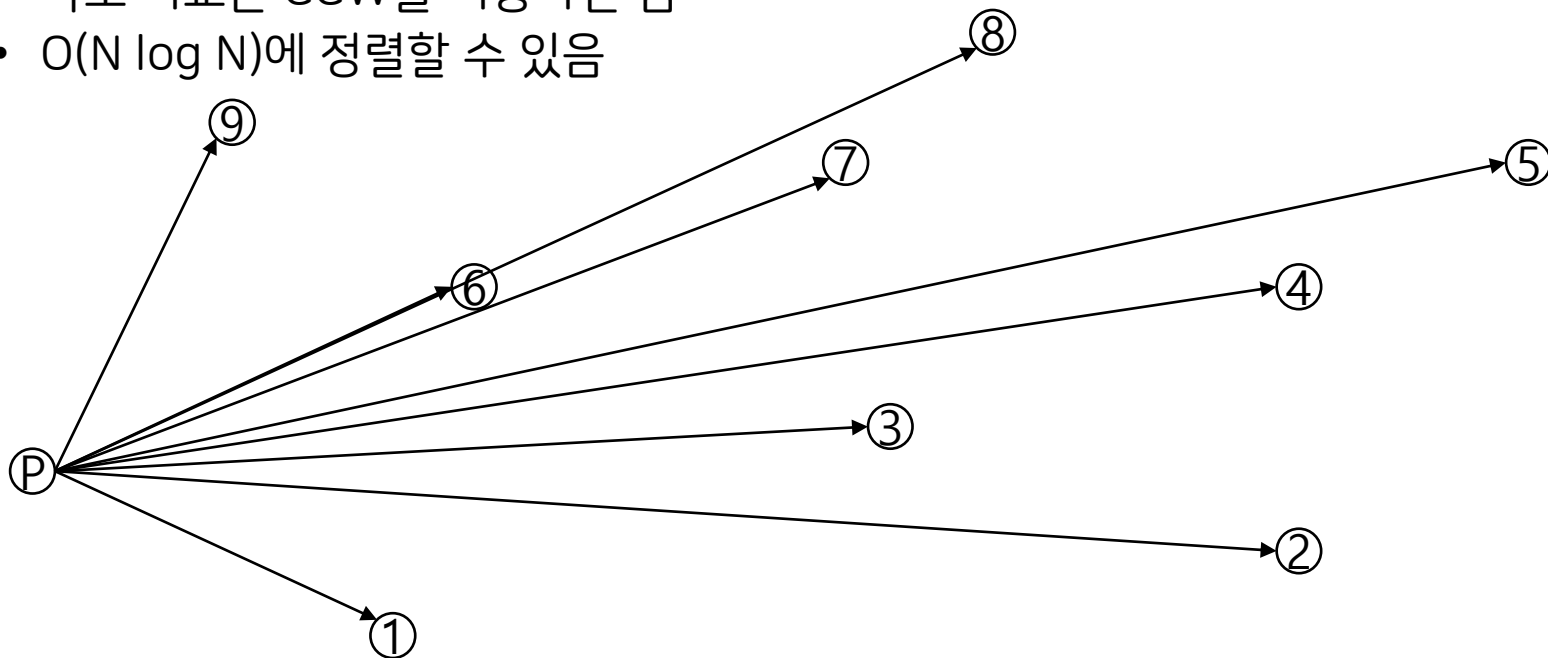


볼록 꺾질

- $O(N^3)$ 알고리즘
 - 모든 점 쌍을 보면서
 - 다른 모든 점들이 한 방향에 있으면 볼록 꺾질의 변이 됨
 - 너무 느림

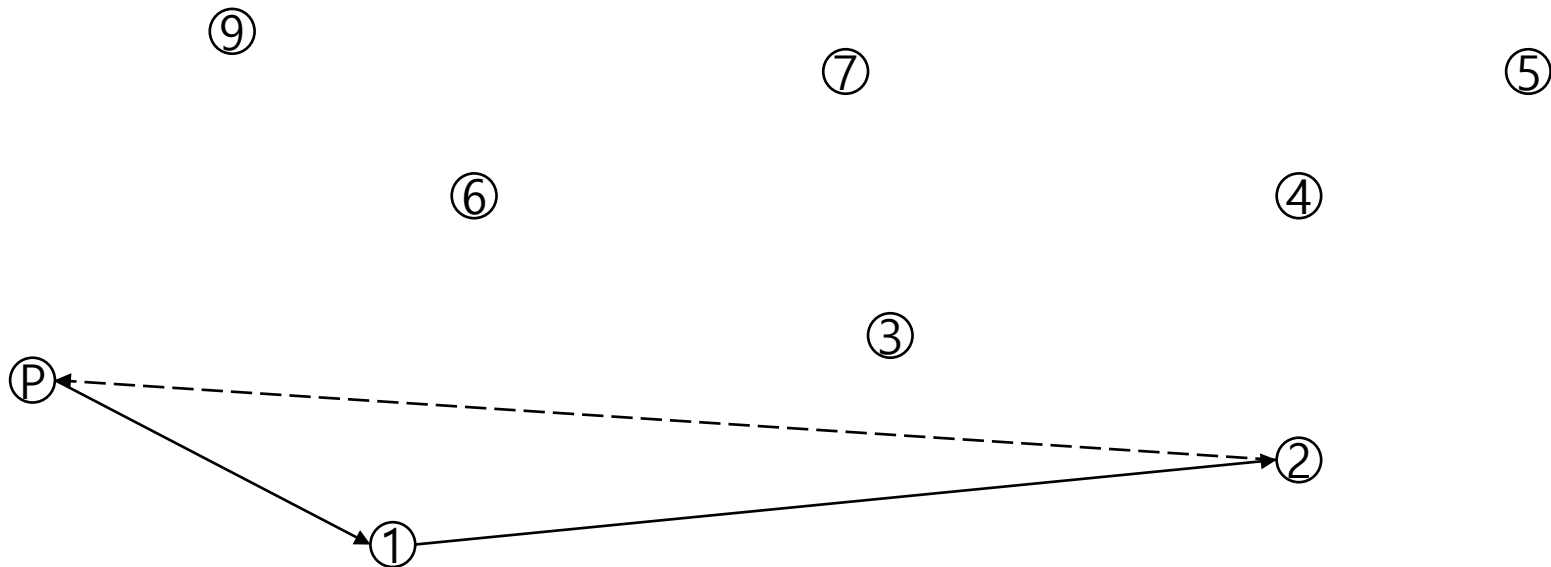
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 1. x좌표가 가장 작은 점(여러 개라면 y좌표가 가장 작은 점) P를 찾는다.
 - P는 항상 볼록 껍질의 꼭짓점, $O(N)$ 에 찾을 수 있음
 2. 다른 모든 점을 P를 기준으로 **각도 순서**대로(일직선이면 가까운 점이 먼저 오도록) 정렬한다.
 - 모든 점을 반시계 방향으로 정렬
 - 각도 비교는 CCW를 이용하면 됨
 - $O(N \log N)$ 에 정렬할 수 있음



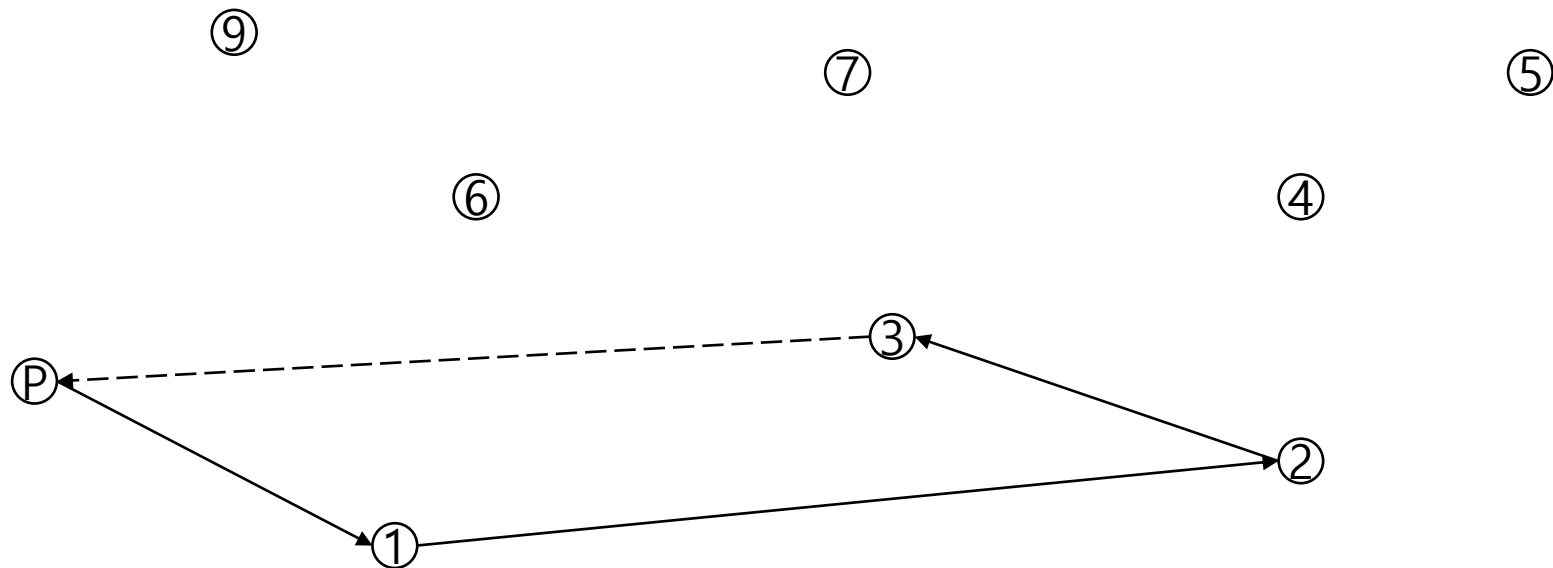
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



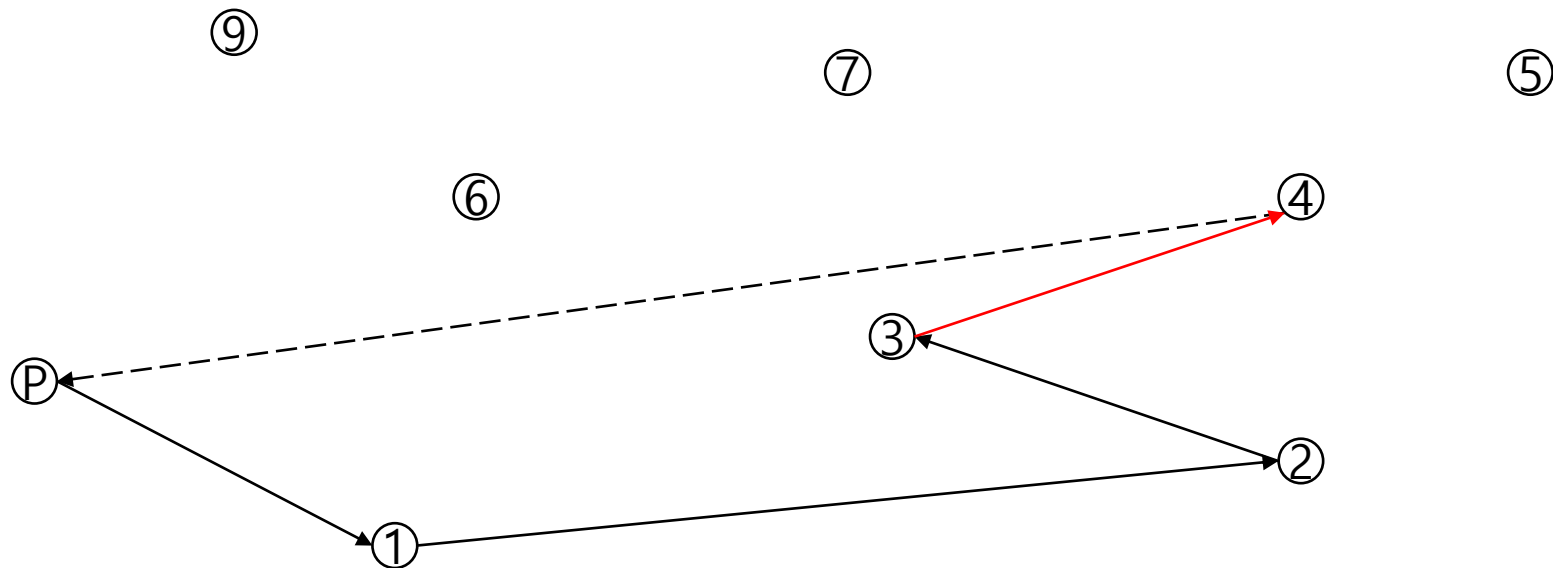
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



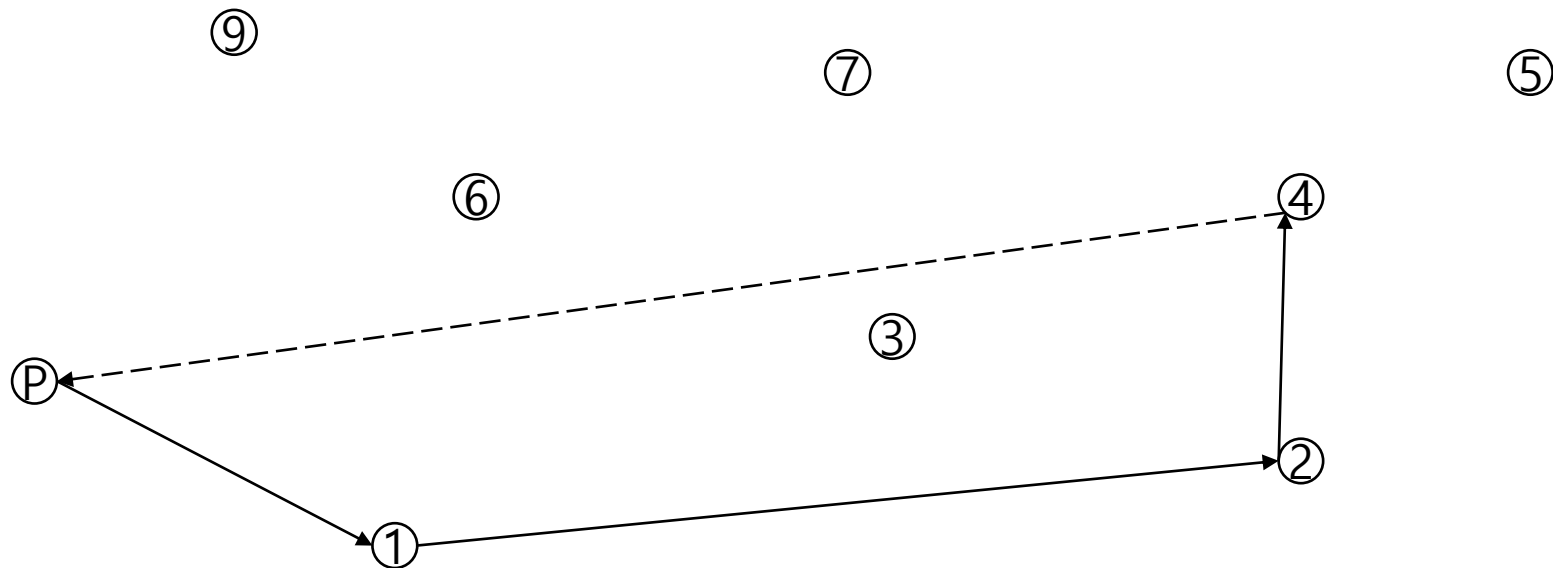
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



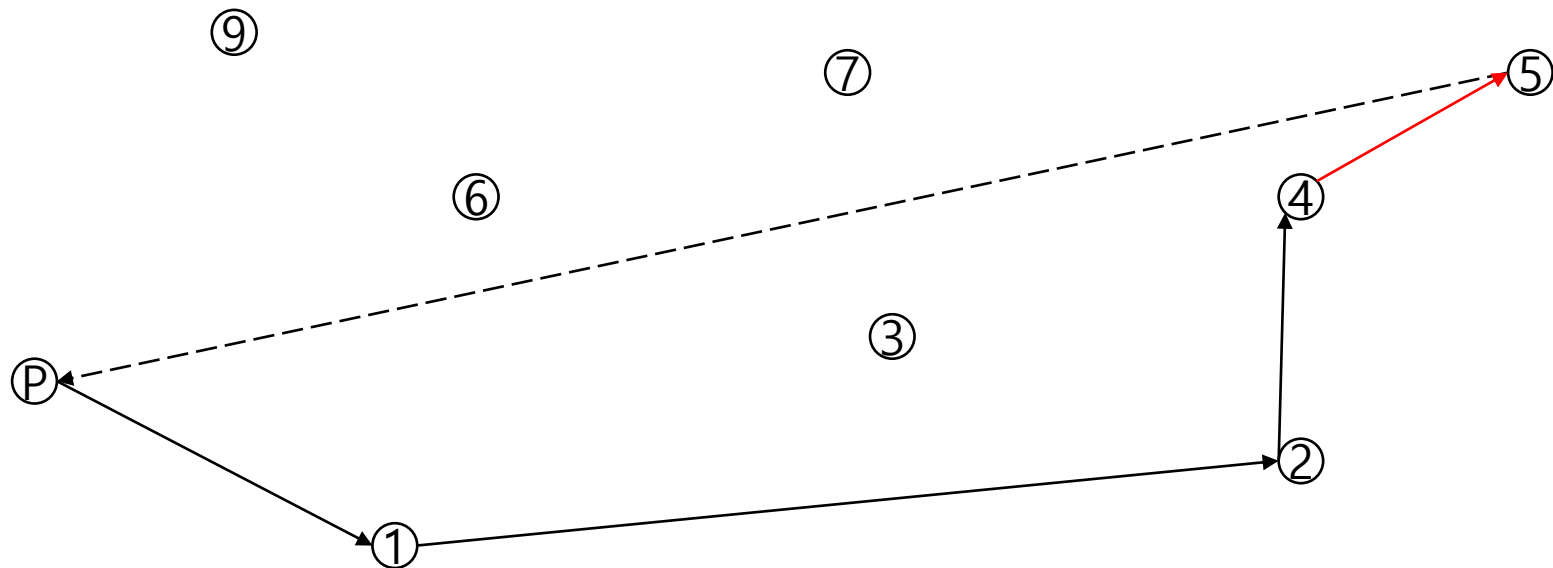
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



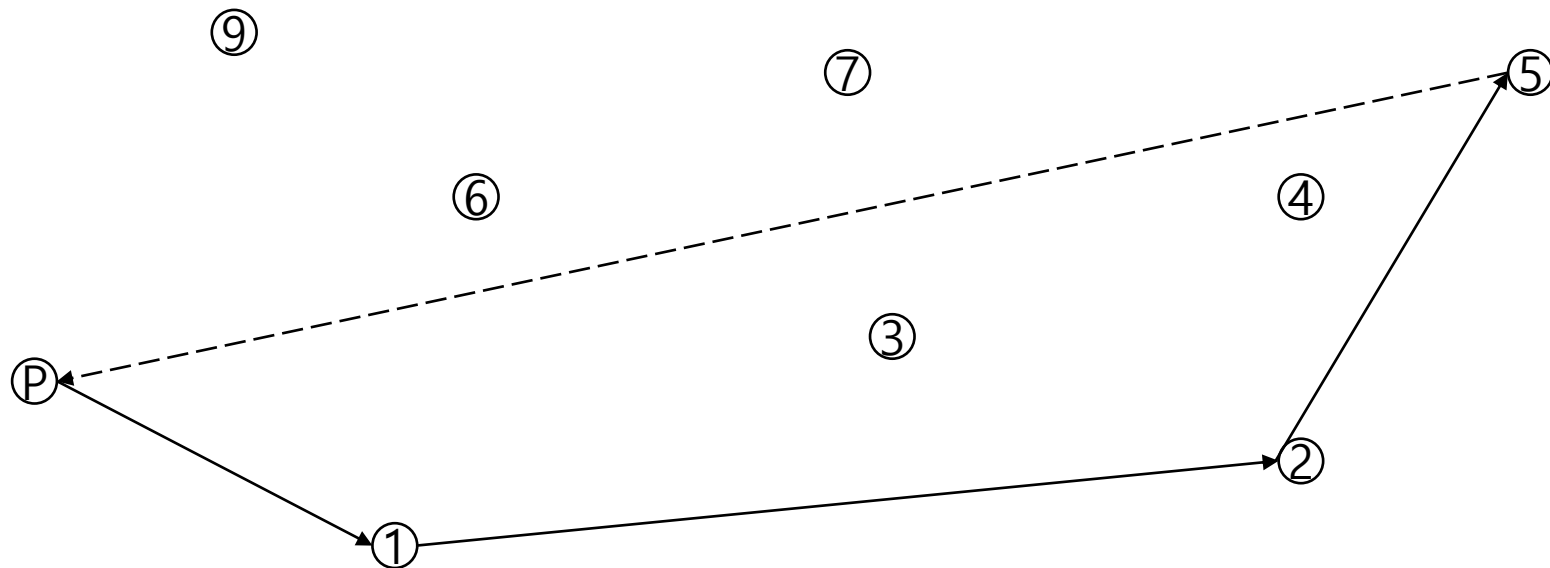
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



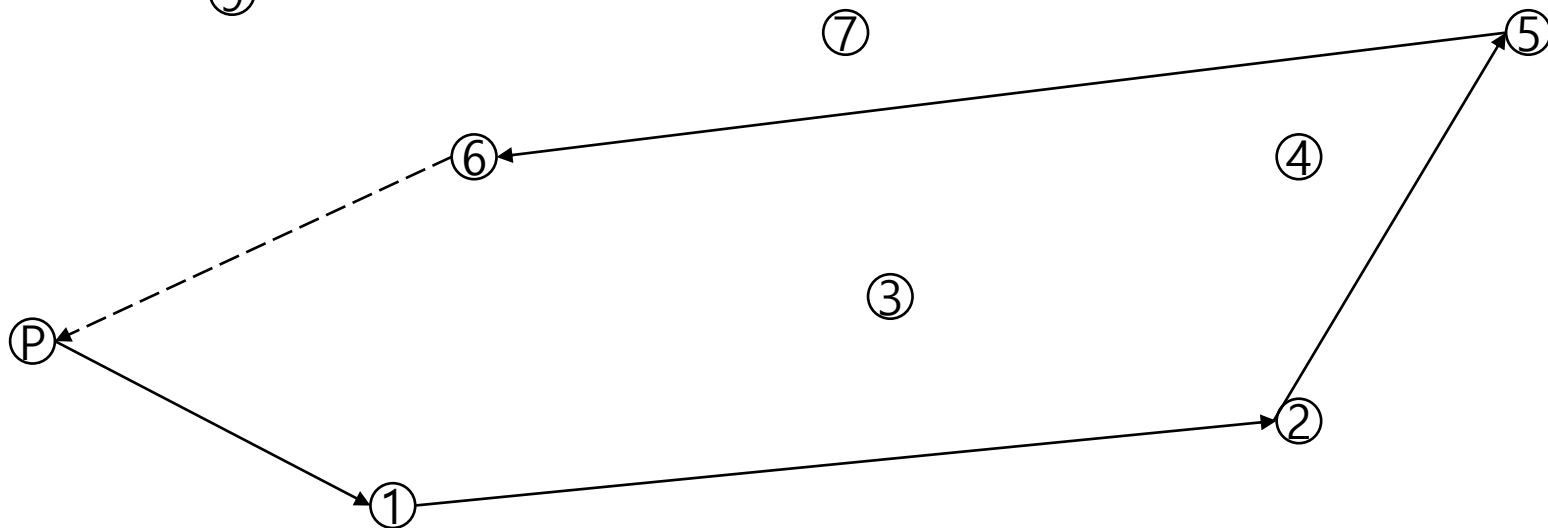
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



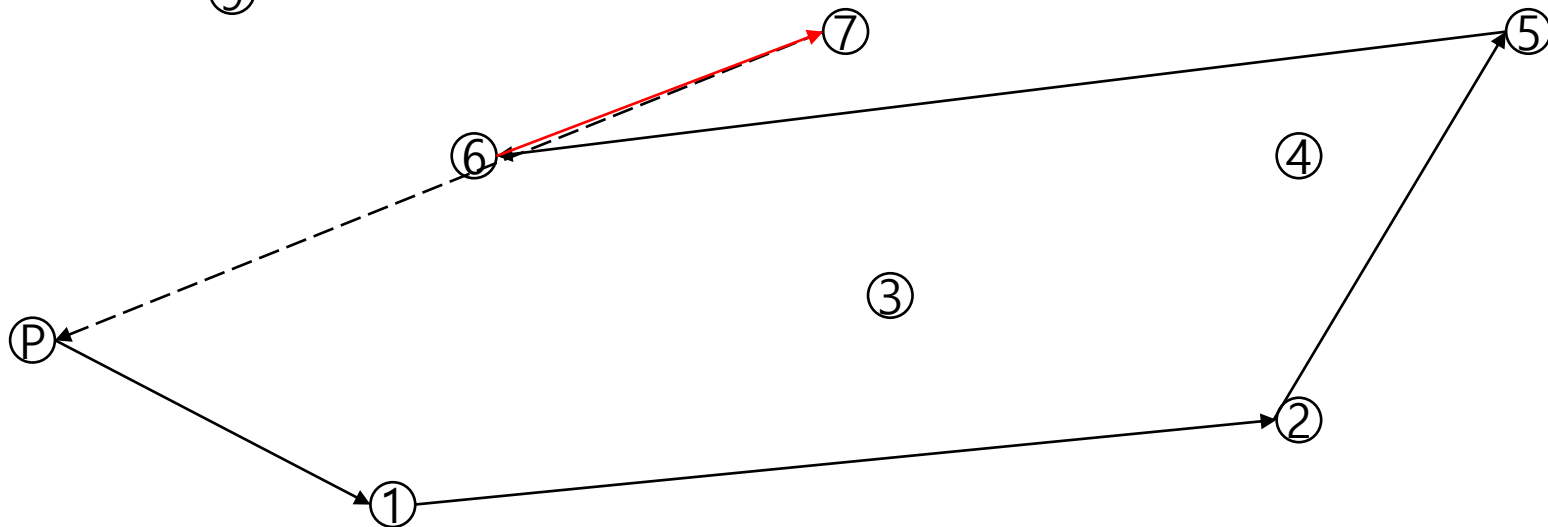
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



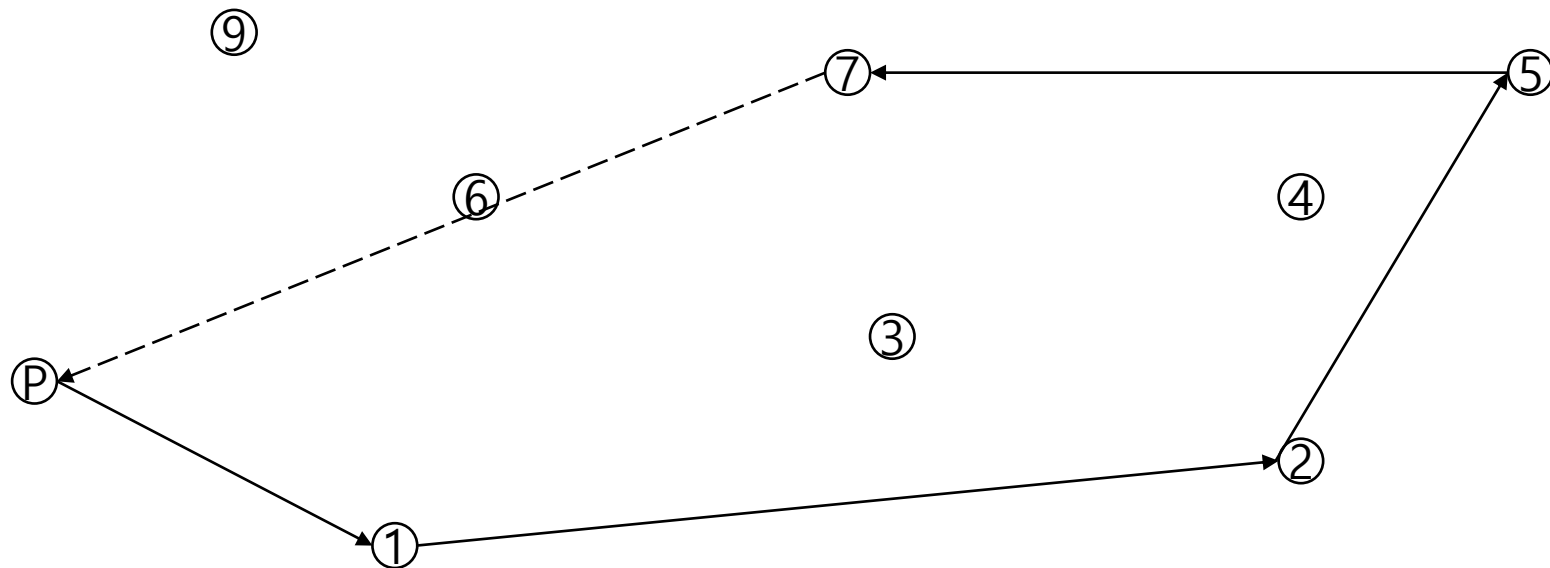
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



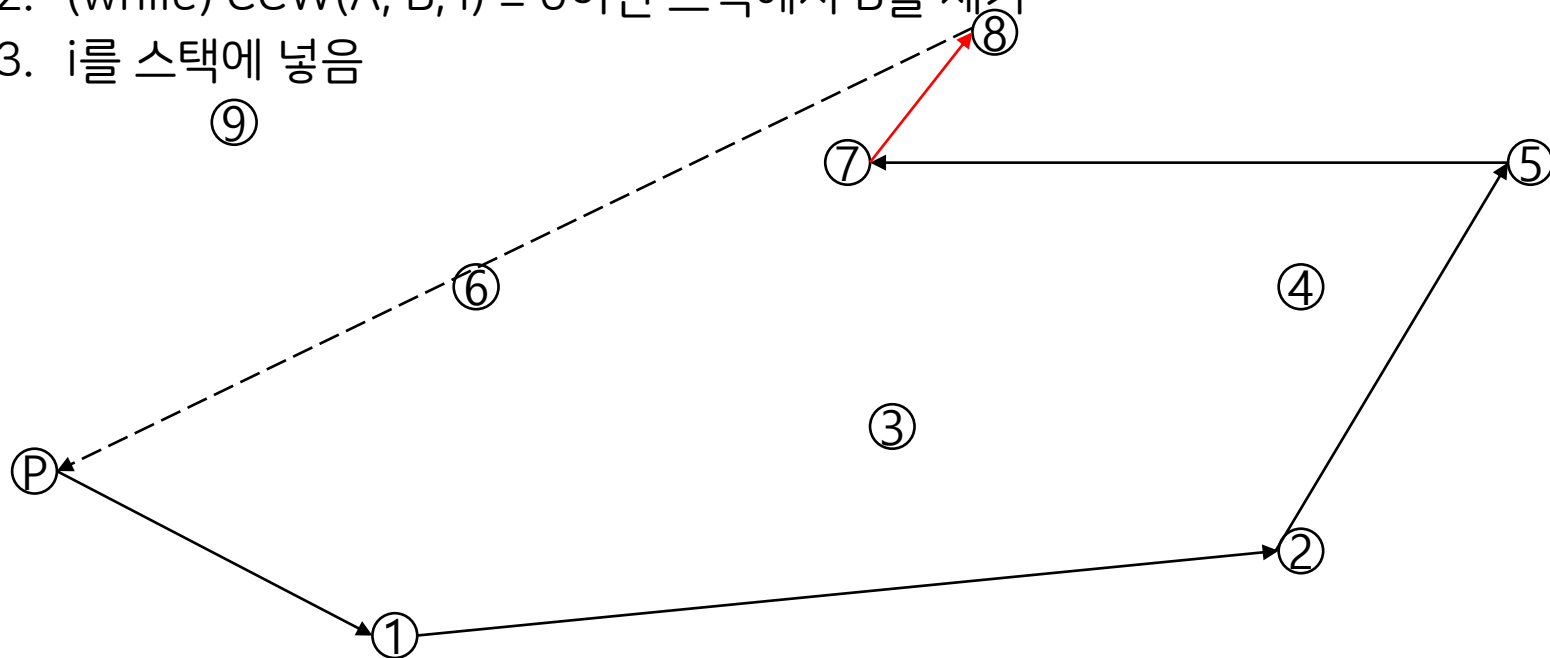
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



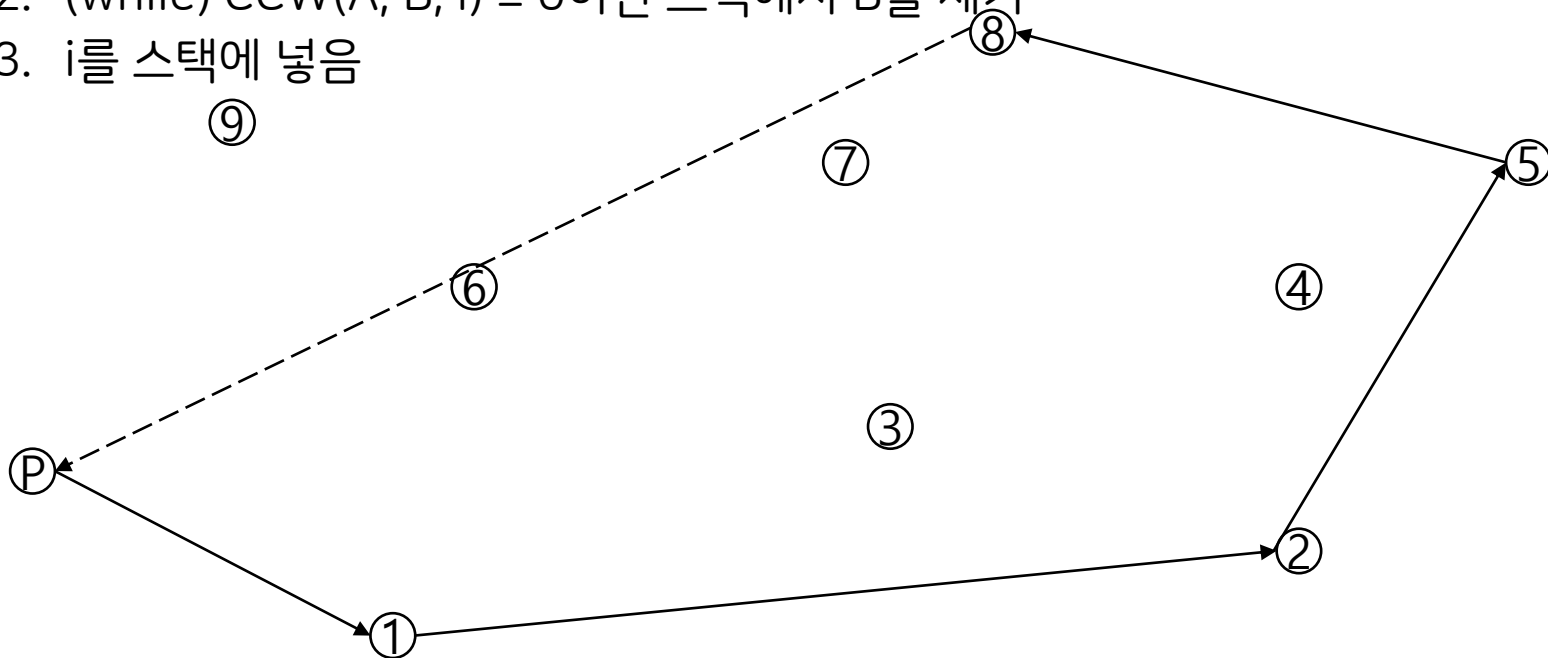
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



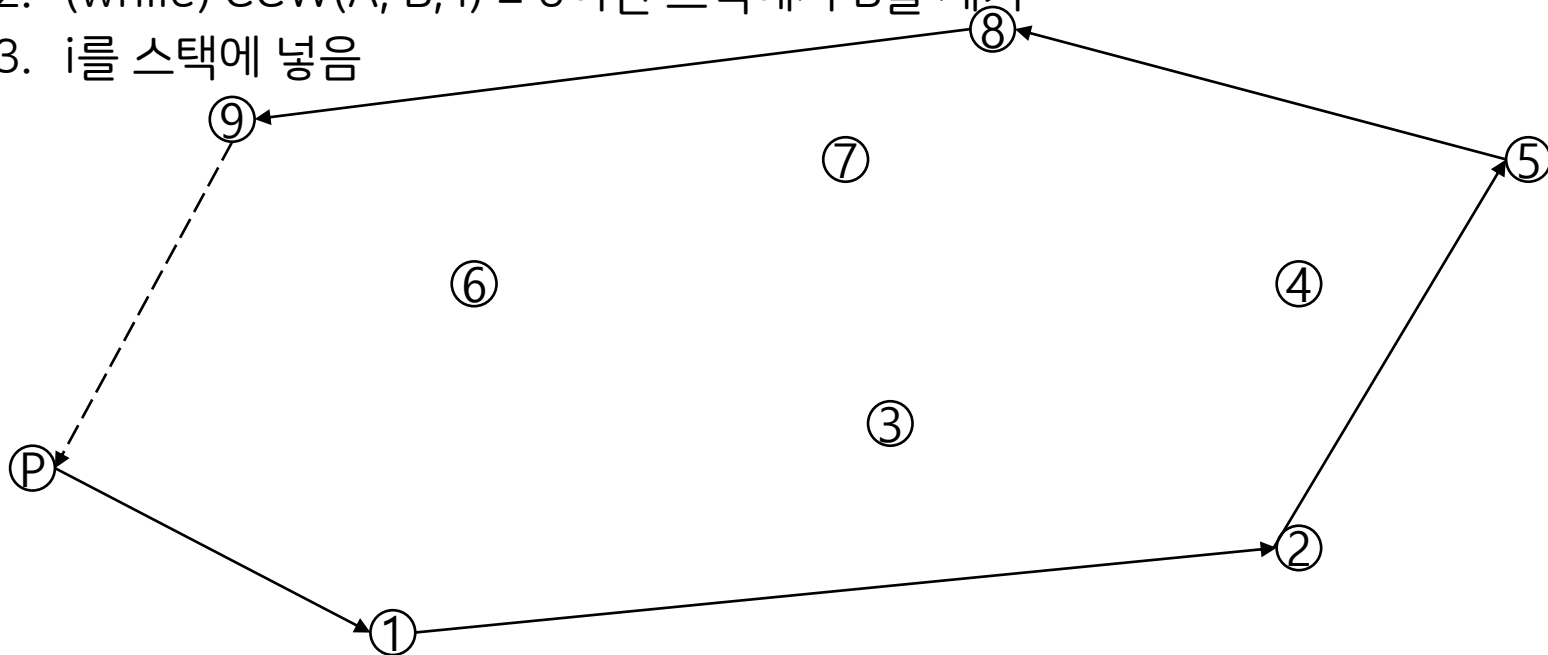
볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



볼록 껍질

- Graham's Scan : 0..i번째 점의 Convex Hull을 관리
 - 점을 차례대로 볼록 껍질에 넣는 방식으로 진행, 이때 볼록 껍질은 스택으로 관리
 - 처음에는 P, 1, 2번 점으로 이루어진 볼록 껍질에서 시작
 - 3번 점부터 차례대로 보면서 아래 과정을 수행
 1. 스택의 맨 뒤에 있는 점 B, 두 번째로 뒤에 있는 점 A, 현재 점 i
 2. (while) $CCW(A, B, i) \leq 0$ 이면 스택에서 B를 제거
 3. i를 스택에 넣음



볼록 껍질

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;

int CCW(Point p1, Point p2, Point p3){
    ll res = (p2.x - p1.x) * (p3.y - p2.y) - (p3.x - p2.x) * (p2.y - p1.y);
    return (res > 0) - (res < 0);
}

ll Dist(Point p1, Point p2){
    ll dx = p2.x - p1.x, dy = p2.y - p1.y;
    return dx*dx + dy*dy;
}

vector<Point> ConvexHull(vector<Point> points){
    swap(points[0], *min_element(points.begin(), points.end()));
    sort(points.begin()+1, points.end(), [&](auto a, auto b){
        int dir = CCW(points[0], a, b);
        if(dir != 0) return dir > 0;
        return Dist(points[0], a) < Dist(points[0], b);
    });

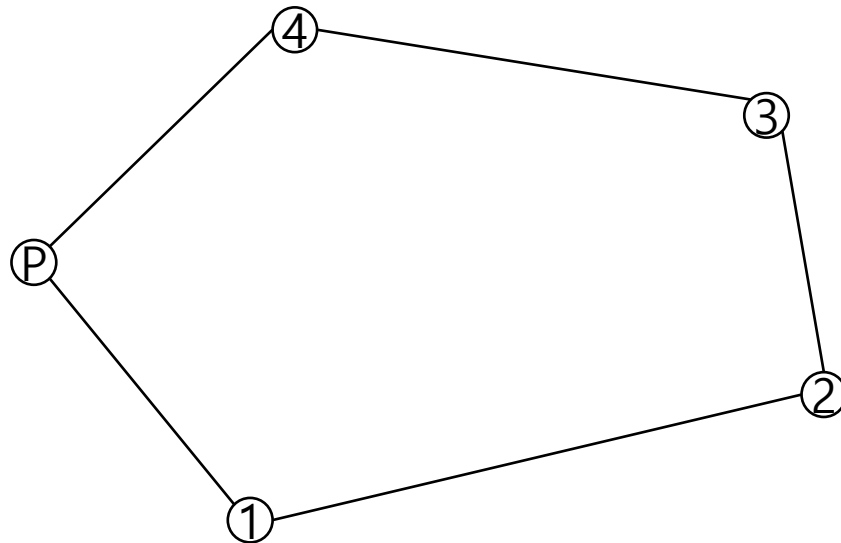
    vector<Point> stk;
    for(auto i : points){
        while(stk.size() >= 2 && CCW(stk[stk.size()-2], stk.back(), i) <= 0) stk.pop_back();
        stk.push_back(i);
    }
    return stk;
}
```


질문

볼록 다각형 내부 판별

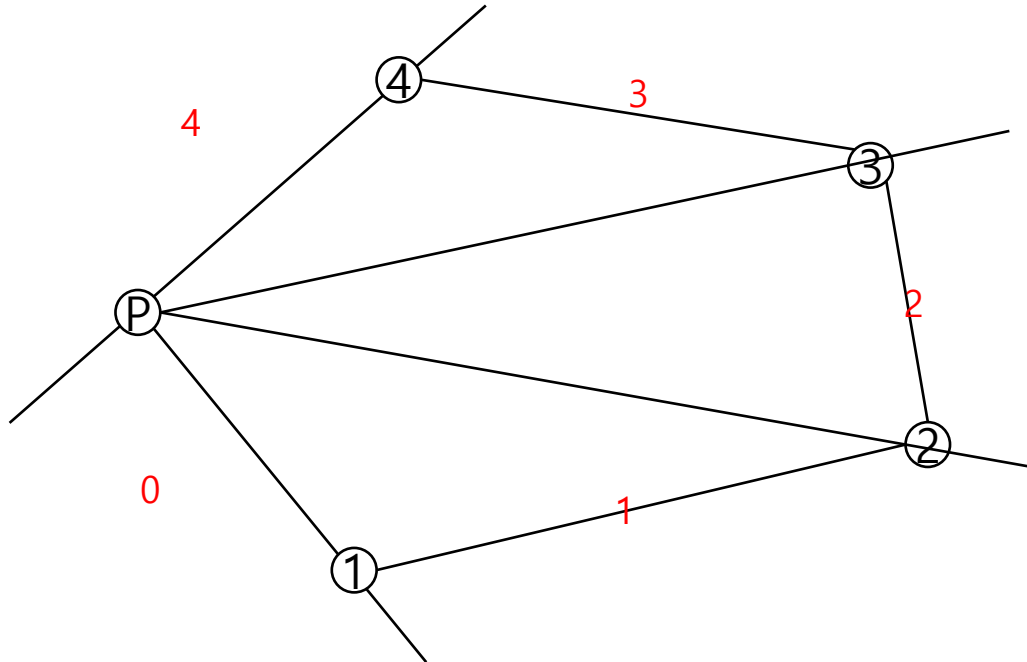
볼록 다각형 내부 판별

- 반직선 그려서 교점 세는 방법은 $O(N)$
 - 볼록 다각형이면 교점 최대 2개 밖에 없는데 굳이 이렇게 해야 할까?



볼록 다각형 내부 판별

- 반직선 그려서 교점 세는 방법은 $O(N)$
 - 볼록 다각형이면 교점 최대 2개 밖에 없는데 굳이 이렇게 해야 할까?
 - 볼록 N 각형은 $N-2$ 개의 삼각형으로 나타낼 수 있다.
 - 어떤 점이 삼각형 내부에 있는지는 $O(1)$ 에 판별 가능 (CCW 3번)
 - 몇 번 영역에 속하는지 찾는 건 이분 탐색을 이용하면 됨



볼록 다각형 내부 판별

```
bool Check(const vector<Point> &v, const Point &pt){  
    if(CCW(v[0], v[1], pt) < 0) return false;  
    int l = 1, r = v.size() - 1;  
    while(l < r){  
        int m = l + r + 1 >> 1;  
        if(CCW(v[0], v[m], pt) >= 0) l = m;  
        else r = m - 1;  
    }  
    if(l == v.size() - 1)  
        return CCW(v[0], v.back(), pt) == 0 && v[0] <= pt && pt <= v.back();  
    return CCW(v[0], v[l], pt) >= 0  
        && CCW(v[l], v[l+1], pt) >= 0  
        && CCW(v[l+1], v[0], pt) >= 0;  
}
```

질문