기하 (Geometry)
중급 9회차

서강대학교 전해성(seastar105)

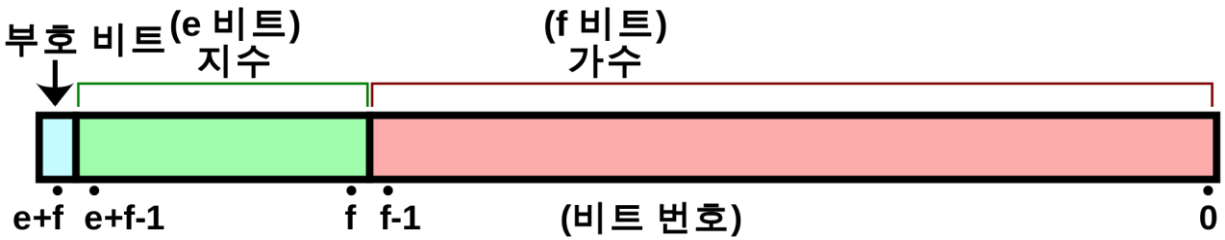
기하문제는 어떤 부분에서 어려운가?

1. 실수 오차에 민감하다.
2. 예외 처리를 해줘야 되는 경우가 많다.
3. 계산기하는 독립적인 분야에 가깝다.

컴퓨터에서의 실수 표현

컴퓨터는 2진법의 세계에서 살고 있기 때문에 실수를 온전히 표현할 수 없습니다.

주로 사용하는 방식은 IEEE 754 부동소수점 표현방식



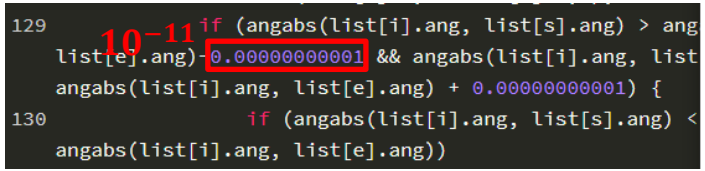
double형은 적당한 실수 2^{64}개를 잡아서 실수들은 이와 가까운 수로 저장합니다.

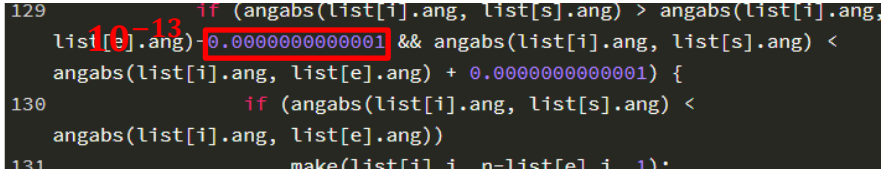
그래서 계산을 할 때 오차는 필연적이고 오차가 누적되기 때문에 오차를 관리하는 것은 어렵습니다.

컴퓨터에서의 실수 표현

그래서 정수만을 사용해서 풀 수 있는지 잘 고민해야 하고 가능하다면 정수로만 푸는 것이 좋습니다.

WA를 받았을 때 알고리즘이 틀린 것인지 오차때문에 틀리는 것인지 확실치가 않기 때문에 힘듭니다.





15023번 문제에서 왼쪽은 틀린 코드고 오른쪽은 맞는 코드입니다.

실수가 같은지 비교할 때는 "적절히" 작은 숫자를 잡고 차이가 그거보다 작으면 같다고 간주하는 것으로 오차를 줄이는데 이 "적절히"가 힘듭니다.

컴퓨터에서의 실수 표현

기하 문제를 풀 때는 정수만을 사용해서 풀 수 있는지 고민하고 그럴 수 있다면 정수만을 사용하는 것이 중요합니다.

예를 들어서 2차원 좌표가 실수로 주어지되 소수점 아래 3번째 자리까지만 주어진다면 전부 1000을 곱해서 정수로 만듭니다.

기하에서 기본 연산들

기하 문제를 풀 때 기본 도구들은 점, 선, 선분, 다각형, 원과 같은 도형들이 됩니다.

그리고 이들 간에 교차, , 거리, 크기 비교, 시계방향을 이루는지, 포함 관계같은 것이 기본 연산들이 됩니다.

이런 것들을 코딩하는 데에 도움을 주는 것이 벡터입니다.

벡터의 연산

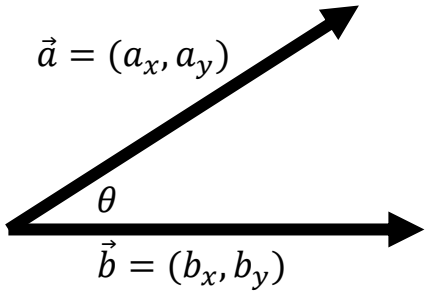
벡터는 두 점이 있으면 그 상대적 위치를 나타내는 데에 유용합니다.

벡터 간의 연산에는 덧셈, 뺄셈, 내적, 외적이 있는데 내적과 외적은 상당히 많이 사용됩니다.

특히 외적은 두 벡터가 시계방향인지를 확인하는 데에 자주 사용됩니다.

벡터로 할 수 있는 것들

내적을 이용하면 두 벡터 간의 각도를 직접적으로 구할 수 있습니다.



두 벡터의 내적

$= \vec{a} \cdot \vec{b}$

$= a_x b_x + a_y b_y$

$= |\vec{a}| * |\vec{b}| cos\theta$

arc cosine을 이용하면 각도를 구할 수도 있습니다.

CCW

벡터의 외적은 원래 3차원 벡터에 정의 되지만 2차원 벡터 (x, y)를 (x, y, 0)이라고 놓고 외적을 합시다.

두 벡터 a, b의 외적은 아래와 같이 계산할 수 있습니다.

$$\vec{a} \times \vec{b} = \hat{n}|a||b| \sin\theta$$

여기서 θ는 두 벡터 a, b가 이루는 각입니다.

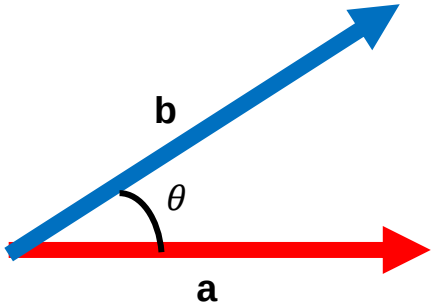
$\hat{n}$은 a와 b의 방향성 관계에 따라 결정되는 부호입니다.

CCW

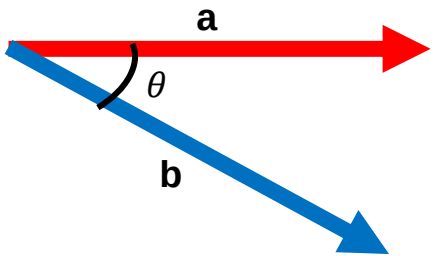
$$\vec{a} \times \vec{b} = \hat{n}|a||b| \sin\theta$$

저 부호는 벡터 a에서 b로 오른손으로 감아쥘 때의 방향으로 알아볼 수 있습니다.

또, 벡터 a와 b가 이루는 각 중에 더 작은 각이 a에서 부터 반시계방향으로 벌어지는 각이라면 양입니다. 시계방향이라면 음입니다.



$\vec{a} \times \vec{b} = |a||b| \sin\theta > 0$



$\vec{a} \times \vec{b} = |a||b| \sin\theta < 0$

$\vec{a} \times \vec{b} = |a||b| \sin\theta = 0$

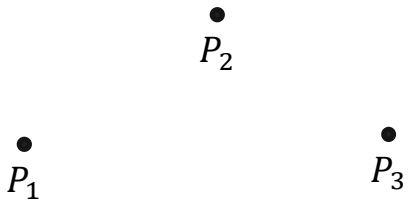
CCW

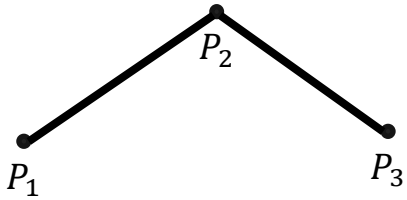
두 2차원 벡터 a, b의 외적값은 아래와 같은 식을 통해서도 계산할 수 있습니다

$$\vec{a} = (a_1, a_2)$$
$$\vec{b} = (b_1, b_2)$$
$$\vec{a} \times \vec{b} = a_1 b_2 - a_2 b_1$$

백준 11758번 CCW

2차원 상의 점 3개가 주어지고 점 3개를 이은 선분의 방향성을 묻는 문제입니다.



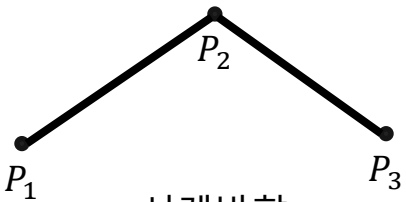


백준 11758번 CCW

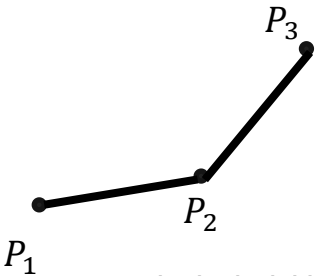
여기서 두 선분의 방향성이란 선분 P_1P_2와 P_2P_3가 시계방향을 이루는지 반시계 방향을 이루는지 일직선인지를 묻습니다.

이는 선분을 벡터로 보고, 두 벡터의 외적으로 확인할 수 있습니다.

이렇게 세점의 관계를 판별할 때 사용한 외적 값은 CCW라고 자주 부르는거 같습니다.



시계방향



반시계방향

백준 11758번 CCW

P_1에서 P_2로 가는 벡터를 a, P_1에서 P_3으로 가는 벡터를 b라고 하면 a, b의 외적 값의 부호를 통해서 시계인지 반시계인지 결정할 수 있습니다.

```
int x1,x2,x3;
int y1,y2,y3;
pair<int,int> v1,v2;
int main() {
    cin >> x1 >> y1;
    cin >> x2 >> y2;
    cin >> x3 >> y3;
    v1.first = x2-x1; v1.second = y2-y1;
    v2.first = x3-x1; v2.second = y3-y1;
    int a = v1.first * v2.second;
    int b = v1.second * v2.first;
    if(a-b > 0) cout << 1 << '\n';
    else if(a-b == 0) cout << 0 << '\n';
    else cout << -1 << '\n';
    return 0;
}
```

```
struct Point {
  ll x, y;
  Point() {};
  Point(ll x, ll y) : x(x), y(y) {};
  Point operator+(const Point &rhs) const { return Point(x+rhs.x, y+rhs.y); };
  Point operator-(const Point &rhs) const { return Point(x-rhs.x, y-rhs.y); };
  bool operator<(Point &rhs) {
    if (x == rhs.x) return y < rhs.y;
    return x < rhs.x;
  }
};

ll cross(const Point &a, const Point &b) {
    ll ret = a.x*b.y - a.y*b.x;
    if(ret > 0) return 1;
    else if(ret == 0) return 0;
    else return -1;
}

ll cross(const Point &a, const Point &b, const Point &c) {
  return cross(b-a, c-a);
}

int main() {
    Point a, b, c;
    cin >> a.x >> a.y;
    cin >> b.x >> b.y;
    cin >> c.x >> c.y;
    if(cross(a, b, c) > 0) cout << 1 << '\n';
    else if(cross(a, b, c) == 0) cout << 0 << '\n';
    else cout << -1 << '\n';
    return 0;
}
```

선분 교차

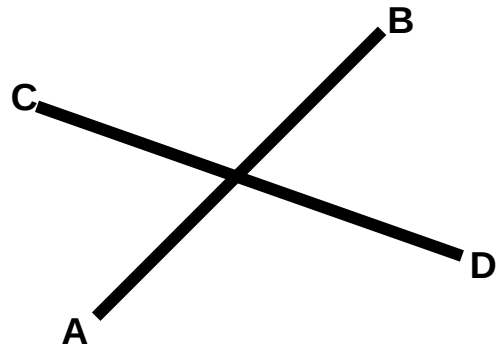
두 선분이 주어졌을 때 선분이 교차하는지 판단하는 문제는 간단해 보이지만 좌표만 가지고 판단하려면 상당히 까다롭습니다.

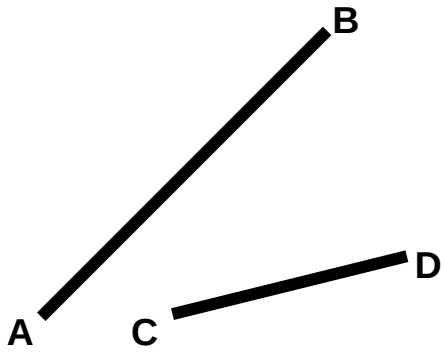
이 때 벡터와 CCW를 이용하면 꽤 간단해집니다.

선분 교차

선분 AB와 선분 CD의 교차 관계와 CCW를 연관지어 언제 교차하는지를 생각해봅시다.

AB와 CD가 교차한다면 점 C는 직선 AB의 위쪽에 있고 D는 직선 CD의 아래쪽에 있는 경우 혹은 그 반대인 경우일 것입니다.

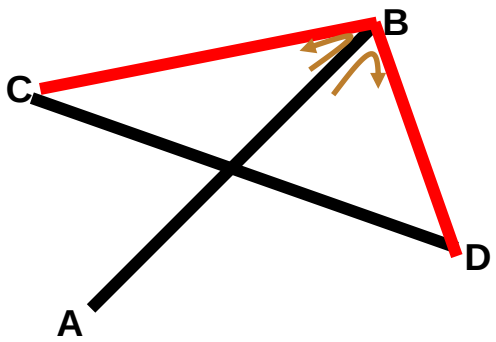


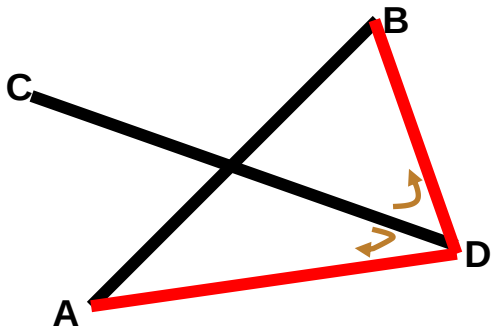


선분 교차

이는 점 A, B, C의 관계와 A, B, D의 관계가 다르다는 뜻이 됩니다.

그리고 C, D, A 그리고 C, D, B의 관계도 다릅니다.





선분 교차

관계가 다르다는 것은 어떻게 판단할까요?

CCW 값의 부호로 방향성을 판단했으니 이 부호가 다르냐로 판단할 수 있습니다.

A, B, C의 CCW값과 A, B, D의 CCW 값을 곱한 것이 음수라면 서로 관계가 다른 것입니다.

따라서 두 선분 AB와 CD가 교차하기 위해선 CCW(A, B, C)와 CCW(A, B, D)를 곱한 것이 음수여야 하고 CCW(C, D, A)와 CCW(C, D, B)를 곱한 것이 음수여야 합니다.

선분 교차

여기까지의 내용을 알면 BOJ 17386번 문제를 풀 수 있습니다.

선분 교차 1 성공

시간 제한	메모리 제한	제출	정답
0.25 초 (추가 시간 없음)	512 MB	2909	1049

문제

2차원 좌표 평면 위의 두 선분 L_1, L_2가 주어졌을 때, 두 선분이 교차하는지 아닌지 구해보자.

L_1의 양 끝 점은 (x_1, y_1), (x_2, y_2), L_2의 양 끝 점은 (x_3, y_3), (x_4, y_4)이다.

입력

첫째 줄에 L_1의 양 끝 점 x_1, y_1, x_2, y_2가, 둘째 줄에 L_2의 양 끝 점 x_3, y_3, x_4, y_4가 주어진다. 세 점이 일직선 위에 있는 경우는 없다.

출력

L_1과 L_2가 교차하면 1, 아니면 0을 출력한다.

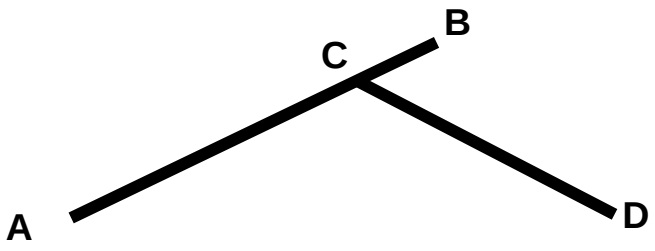
```
ll cross(const Point &a, const Point &b) {
    ll ret = a.x*b.y - a.y*b.x;
    if(ret > 0) return 1;
    else if(ret == 0) return 0;
    else return -1;
}

ll cross(const Point &a, const Point &b, const Point &c) {
  return cross(b-a, c-a);
}

bool intersect(Point a, Point b, Point c, Point d) {
    ll ab = cross(a, b, c) * cross(a, b, d);
    ll cd = cross(c, d, a) * cross(c, d, b);
    return ab < 0 && cd < 0;
}
```

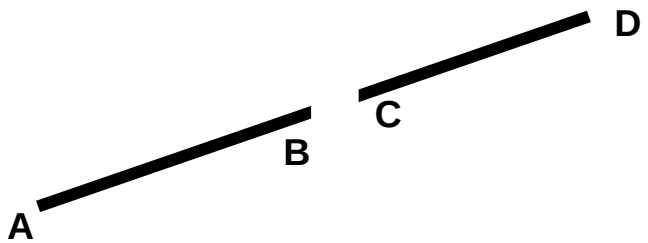
선분 교차

문제의 조건에서 세 점이 일직선 위에 있는 경우는 없다고 했습니다.

이 조건 아래처럼 다른 선분 위에 한 선분의 끝이 올라와 있는 경우가 없다는 것을 뜻합니다.

그리고 두 선분이 하나의 직선의 일부분이 아니란 뜻입니다.

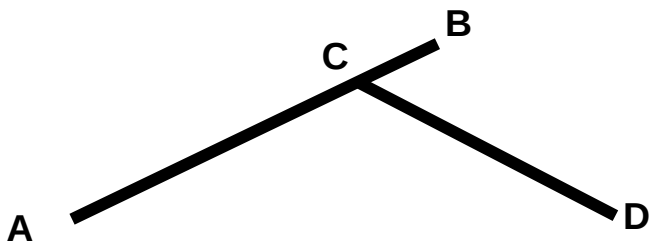




선분 교차

한 쪽 끝만 다른 선분에 올라와 있는 경우라면 교차 판정을 사용하는 데에 사용되는 4개의 CCW 값 중에 하나가 0이 될 것입니다.

따라서 두 곱이 전부 음수인지 판단하는 게 아니라 0 이하인지로 판별하면 됩니다.

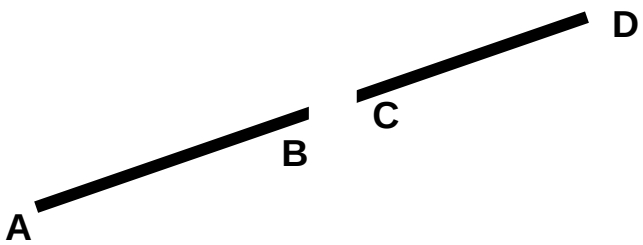


선분 교차

이제 두 선분이 하나의 직선으로부터 나온 경우를 생각해봅시다.

이 때는 두 CCW의 곱이 전부 0입니다.

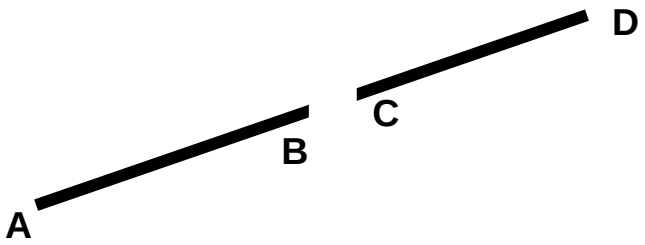
두 CCW의 곱이 전부 0인지를 통해서 이 경우를 판별할 수 있다는 뜻입니다.



선분 교차

이 경우는 각 선분의 끝이 겹치는지를 통해서 판단해주면 됩니다.

하나의 직선에서 나왔기 때문에 AB의 x구간과 CD의 x구간이 겹치는지, AB의 y구간과 CD의 y구간이 겹치는지 확인해주면 됩니다.



선분 교차

이걸로 선분이 교차하는지를 판단할 수 있습니다.

다만 좌표가 실수로 주어질 땐 항상 조심합시다.

```
bool disjoint(ll s, ll e, ll l, ll r){
    if(s > e) swap(s, e);
    if(l > r) swap(l, r);
    return e < l || s > r;
}

bool intersect(Point a, Point b, Point c, Point d) {
    ll ab = cross(a, b, c) * cross(a, b, d);
    ll cd = cross(c, d, a) * cross(c, d, b);
    if(ab == 0 && cd == 0) {
        return !disjoint(a.x, b.x, c.x, d.x) && !disjoint(a.y, b.y, c.y, d.y);
    }
    return ab <= 0 && cd <= 0;
}
```

선분 교차

선분을 점 두개로 나타냈습니다. 벡터를 통해서 나타내는 것도 가능합니다.

직선은 점 하나와 벡터 하나로도 나타낼 수 있습니다.

$$\vec{u} + k\vec{v}$$

여기서 u는 직선이 지나는 점 중의 하나를 나타내며 v는 직선의 방향벡터입니다.

실수 k를 곱해서 둘을 더하는 것으로 직선 위의 모든 점을 나타낼 수 있습니다.

선분 AB와 CD의 교차 판정은 각각을 위와 같은 방식으로 나타내고 교점을 구한 뒤에 k가 0이상 1이하인지 보는 것으로도 가능합니다.

백준 16491번 대피소 찾기

대피소 찾기 성공 스페셜 저지 출처

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	512 MB	271	111	85	46.448%

문제

지구에서 보낸 화성표면 탐사로봇은 2032년 현재 100개 이상이고, 그 개수가 빠르게 증가하고있다. 그 이유는 지구에는 없는 귀중한 금속 자원이 화성표면에서 속 속 발견되고 있기 때문이다.

화성 궤도에는 모선이 돌고 있어서, 화성 표면에서 임무를 수행하는 수많은 로봇과 수시로 데이타와 명령을 주고 받도록 되어있다.

이 모선이 하는 일 중 하나는 화성 대기에 먼지 폭풍이 예상되었을 때, 표면 탐사 로봇을 대피소로 대피시키는 것이다. 모선에서는 화성 표면의 로봇에게 대피해야 할 대피소를 지정해주고 대피 명령을 내리는 것이다.

하나의 대피소에는 하나의 로봇만 수용할 수 있고, 대피 명령을 받은 로봇은 직선경로로 대피소를 향해 전속력으로 이동하게 되어있다. 문제는 로봇이 대피소로 이동중에 다른 로봇과 충돌할 수 있다는 점이다. 충돌할 일이 없도록 각각의 로봇에게 대피소를 할당해 주는 프로그램을 작성하라.

화성은 2차원 평면이다. 위도와 경도는 왼쪽 위를 (0,0)으로 오른쪽 아래는 (100,100)으로 한다.

입력

첫째 줄에 로봇과 대피소의 수 N ($1 \le N \le 10$)이 주어진다. 둘째 줄부터 N개의 줄에 로봇의 위도와 경도가 1번 로봇부터 N번 로봇까지 한 줄에 하나씩 주어진다. 다음 N개의 줄에 대피소의 위도와 경도가 1번 대피소부터 N번 대피소까지 한 줄에 하나씩 주어진다.

로봇과 대피소의 위치가 같은 경우는 없으며, 로봇과 대피소는 항상 화성 표면 위에 있다.

백준 16491번 대피소 찾기

각 로봇마다 대피소를 하나 골라서 배정해줘야 합니다.

최대 10개까지 밖에 없기 때문에 모든 경우(10!)을 시도해봐도 충분합니다.

하나의 경우에 대해서 가능한지의 여부는 로봇과 배정된 대피소를 이어주는 선분 N개 중에 교차하는 선분이 있는지 판단하는 것으로 충분합니다.

Convex Hull(볼록 껍질)

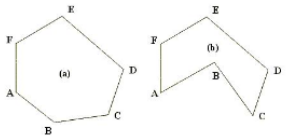
볼록 껍질 성공

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	16281	4704	2259	25.630%

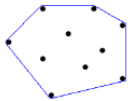
문제

다각형의 임의의 두 꼭짓점을 연결하는 선분이 항상 다각형 내부에 존재하는 다각형을 볼록 다각형이라고 한다. 아래 그림에서 (a)는 볼록 다각형이며, (b)는 볼록 다각형이 아니다.



조금만 생각해 보면 다각형의 모든 내각이 180도 이하일 때 볼록 다각형이 된다는 것을 알 수 있다. 편의상 이 문제에서는 180도 미만인 경우만을 볼록 다각형으로 한정하도록 한다.

2차원 평면에 N개의 점이 주어졌을 때, 이들 중 몇 개의 점을 골라 볼록 다각형을 만드는데, 나머지 모든 점을 내부에 포함하도록 할 수 있다. 이를 볼록 껍질 (CONVEX HULL)이라 한다. 아래 그림은 N=10인 경우의 한 예이다.



점의 집합이 주어졌을 때, 볼록 껍질을 이루는 점의 개수를 구하는 프로그램을 작성하시오.

Convex Hull(볼록 껍질)

점이 N개 주어졌을 때 이 점들의 볼록 껍질은 주어진 점들을 꼭짓점으로 사용하며 점을 모두 포함하는 볼록 다각형입니다.

볼록껍질은 기하문제를 풀 때 꽤 자주 등장합니다.

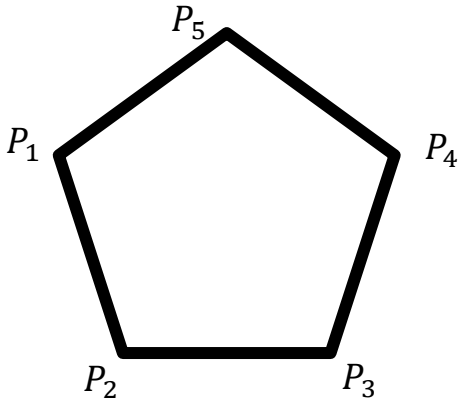
Convex Hull(볼록 껍질)

볼록 다각형의 성질 중 하나를 통해서 $O(N^3)$알고리즘을 먼저 살펴보겠습니다.

볼록 다각형이 있을 때 꼭짓점들을 반시계 방향으로 나열해봅시다.

그러면 모든 점들이 P_iP_{i+1}을 기준으로 반시계 방향으로 존재합니다.

이를 통해서 모든 점의 쌍들에 대해서 다른 모든 점들이 반시계 방향으로 있는지 확인함으로 볼록 껍질을 구할 수 있습니다.



Convex Hull(볼록 껍질)

물론 이거보다 훨씬 빠른 $O(NlogN)$만에 볼록껍질을 구하는 방법이 있습니다.

Graham Scan과 Monotone Chain이란 방식이 있는데 두 방식 비슷한 과정을 가진 알고리즘이니 편한 것을 사용하시면 되겠습니다.

Convex Hull(볼록 껍질) – Graham Scan

점 N개가 주어지면 다음과 같은 과정을 통해서 볼록껍질을 구합니다.

1. 주어진 점 중에서 y좌표가 가장 작은 점을 구합니다. 그러한 점이 여러 개 있다면 x좌표가 가장 작은 것을 구합니다.

2. 1. 에서 구한 점을 원점으로 생각하고 다른 점들을 반시계 방향으로 정렬합니다.

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다. 다만 스택에 넣을 때는 스택의 제일 위 점 두개와 넣을 점과의 CCW를 통해서 새로 넣을 점과 스택의 마지막 두 점이 반시계를 이루도록 해줘야 합니다.

4. 모든 점에 대해서 3.의 과정을 끝냈으면 스택의 점들이 convex hull입니다.

Convex Hull(볼록 껍질) – Graham Scan

1. 주어진 점 중에서 y좌표가 가장 작은 점을 구합니다. 그러한 점이 여러 개 있다면 x좌표가 가장 작은 것을 구합니다.

위 과정은 주어진 점들을 훑어보면서 O(N)만에 가능합니다.

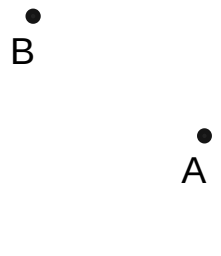
Convex Hull(볼록 껍질) – Graham Scan

2. 1.에서 구한 점을 원점으로 생각하고 다른 점들을 반시계 방향으로 정렬합니다.

원점으로 생각한다는 것은 모든 점들을 평행이동하여 생각하는 것과 동일합니다.

이 때, 모든 점들은 원점 기준으로 하나의 반평면에 존재하기 때문에 직접적으로 각을 구하지 않아도 반시계로 정렬하는 것이 가능합니다. 그리고 각이 같다면 거리를 기준으로 정렬합니다.

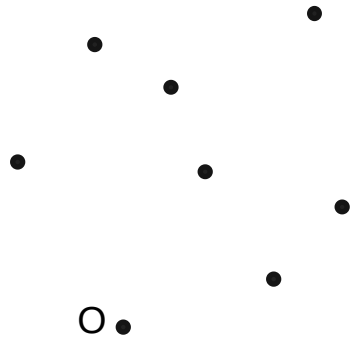
O, A, B의 CCW를 구했을 때 아래와 같이 반시계라면 CCW가 양수고 시계라면 음수일 것입니다.

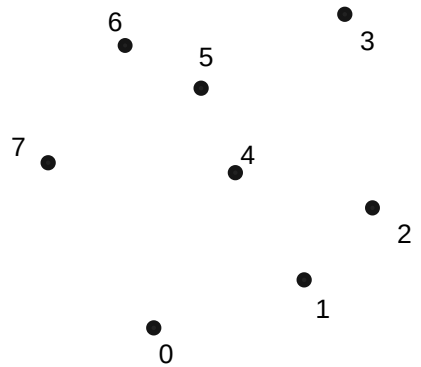


Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

아래와 같이 점이 주어졌고 정렬한 결과가 오른쪽이라고 합시다.

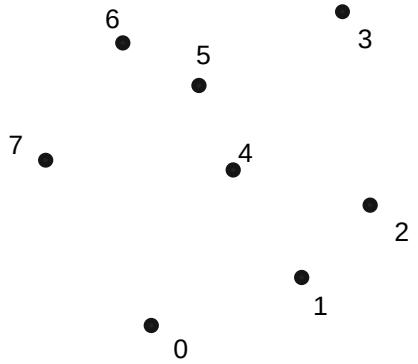




Convex Hull(볼록 껍질) – Graham Scan

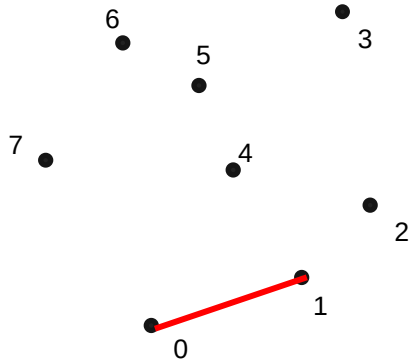
3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

O부터 시작해서 스택에 넣습니다.



Convex Hull(볼록 껍질) – Graham Scan

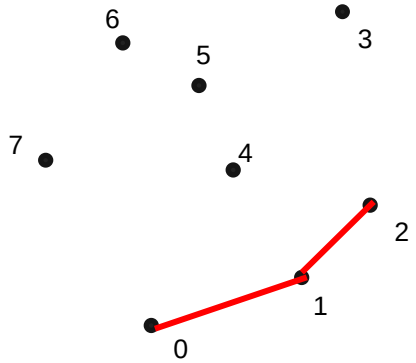
3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다



Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

스택의 원소가 2개가 됐습니다. 제일 위의 점인 0, 1과 2가 반시계를 이루는지 확인합니다. 반시계니까 스택에 넣습니다.

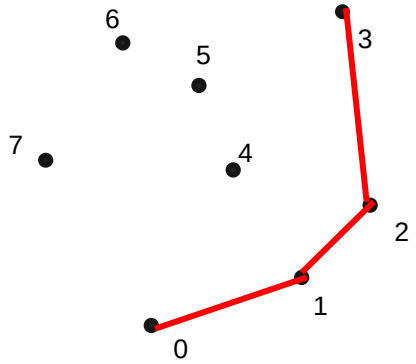


2
1
0

Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

1, 2, 3은 반시계를 이룹니다. 3을 넣어줍니다.

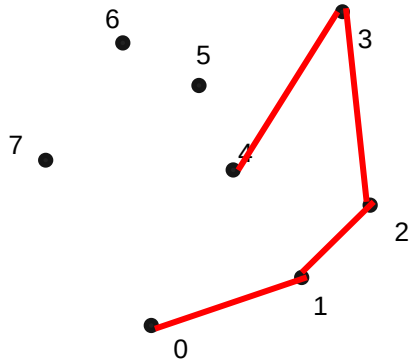


3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

2, 3, 4는 반시계 방향을 이룹니다. 넣어줍니다.

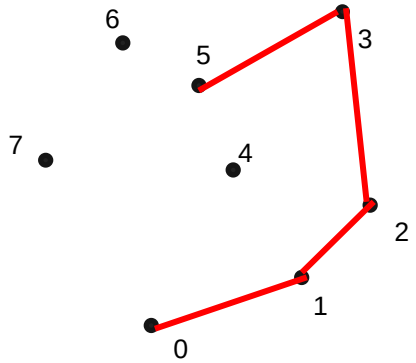


4
3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

3, 4, 5는 시계방향을 이룹니다. 4를 빼줍니다.
2, 3, 5는 반시계방향을 이룹니다. 5를 넣습니다.

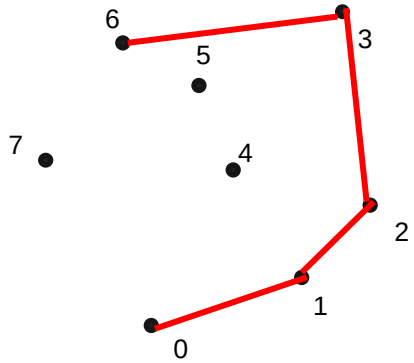


5
3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

3, 5, 6은 시계방향입니다. 5를 빼줍니다.
2, 3, 6은 반시계입니다. 6을 넣습니다.

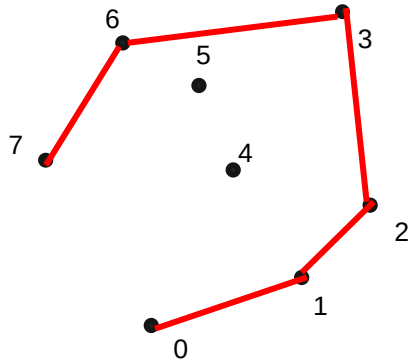


6
3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

3. 정렬한 순서대로 원점부터 시작하여 스택에 넣습니다.
   다만 스택은 넣을 점과 제일 위 두 점이 항상 반시계가 되도록 합니다

3, 6, 7은 반시계 방향입니다. 7을 넣어줍니다.
더 이상 넣을 점이 없습니다.

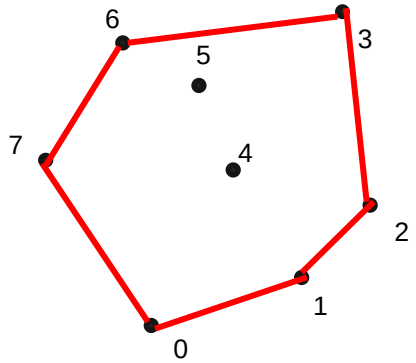


7
6
3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

4. 모든 점에 대해서 3.의 과정을 끝냈으면 스택의 점들이 convex hull입니다.

0, 1, 2, 3, 6, 7이 convex hull입니다.



7
6
3
2
1
0

Convex Hull(볼록 껍질) – Graham Scan

```
vector<Point> graham_scan(vector<Point> a) {
  if (a.size() <= 2) return a;
  int N = a.size();
  swap(a[0], *min_element(a.begin(), a.end(), [] (Point a, Point b) {
      if(a.y == b.y) return a.x < b.x;
      return a.y < b.y;
  }));
  for(int i=1;i<N;++i) a[i] = a[i] - a[0];
  a[0].x = a[0].y = 0;
  sort(a.begin()+1, a.end(), [] (Point a, Point b) {
      ll c = cross(a, b);
      if(c == 0) return (a.x*a.x + a.y*a.y) < (b.x*b.x + b.y*b.y);
      return c > 0;
  });
  vector<Point> ret;
  for(int i=0;i<N;++i) {
      while(ret.size() >= 2 && cross(ret[ret.size()-2], ret[ret.size()-1], a[i]) <= 0) ret.pop_back();
      ret.push_back(a[i]);
  }
  return ret;
}
```

Convex Hull(볼록 껍질) – Monotone Chain

Graham Scan외에 Monotone Chain이라는 방식도 있습니다.

이 방식은 볼록껍질의 윗 껍질과 아래 껍질을 구해서 합치는 방식으로 알고리즘이 진행됩니다.

이 알고리즘에서 중요한 것은 점들을 x좌표 기준으로 정렬하고 x좌표가 같을 땐 y좌표를 기준으로 정렬했을 때 항상 정렬 순으로 처음 점과 끝점은 항상 convex hull에 포함된다는 것입니다.

이 점을 이용해서 convex hull을 구합니다.

Convex Hull(볼록 껍질) – Monotone Chain

점 N개가 주어지면 다음과 같은 과정을 통해서 볼록껍질을 구합니다.

1. x좌표 순으로 점을 정렬합니다. x좌표가 같다면 y좌표 순으로 정렬합니다.

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

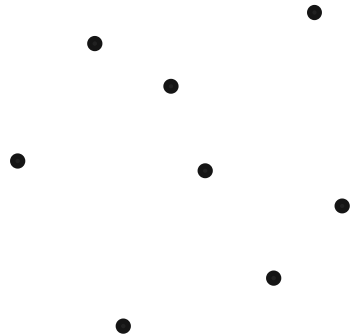
3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

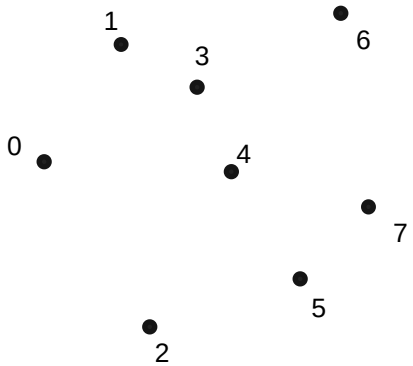
4. upper hull과 lower hull을 합쳐주면 convex hull이 나옵니다.

Convex Hull(볼록 껍질) – Monotone Chain

1. x좌표 순으로 점을 정렬합니다. x좌표가 같다면 y좌표 순으로 정렬합니다.
   정렬순으로 처음 점을 A, 마지막 점을 B라고 합시다.

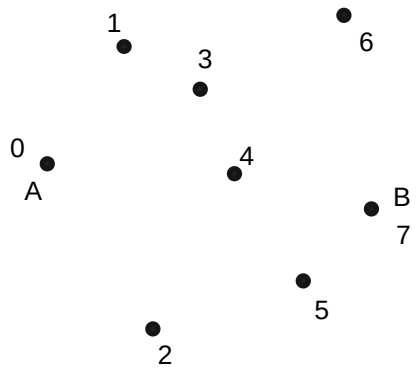
아래와 같이 점이 주어졌고 정렬한 결과가 오른쪽이라고 합시다.





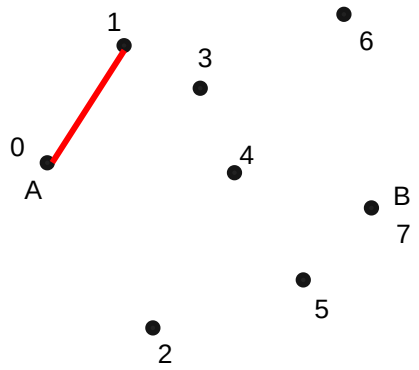
Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.



Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

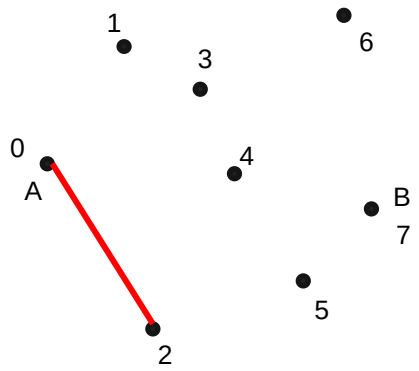


lower

1
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

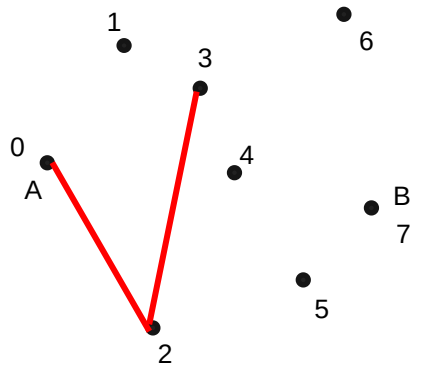


lower

2
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.



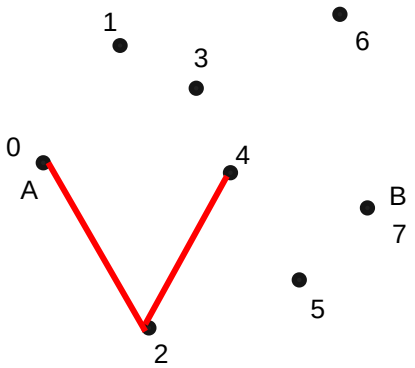
lower

3
2
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

2, 3, 4는 시계방향입니다. 3을 빼고 4를 넣습니다



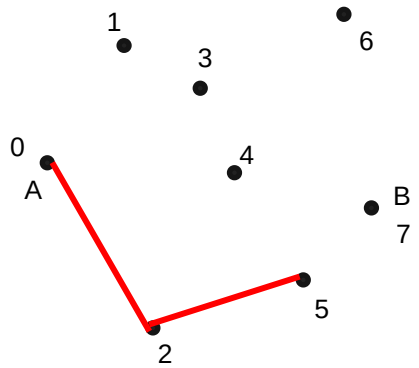
lower

4
2
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

2, 4, 5는 시계방향입니다. 4를 빼고 5를 넣어줍니다.

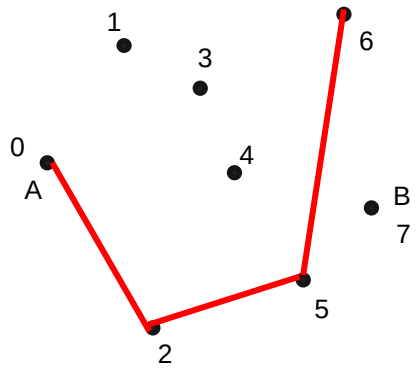


lower

5
2
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.



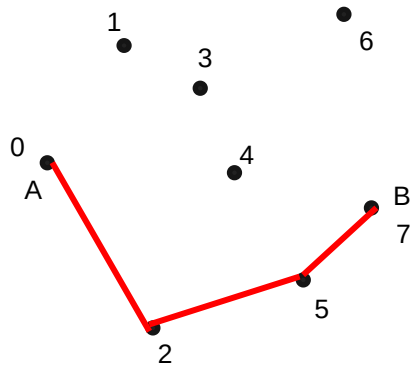
lower

6
5
2
0

Convex Hull(볼록 껍질) – Monotone Chain

2. 정렬순으로 점을 순회합니다. Graham scan에서 스택에 점을 넣듯이 스택에 점을 넣습니다. 이것이 lower hull이 됩니다.

5, 6, 7은 시계방향입니다. 6을 빼고 7을 넣습니다.

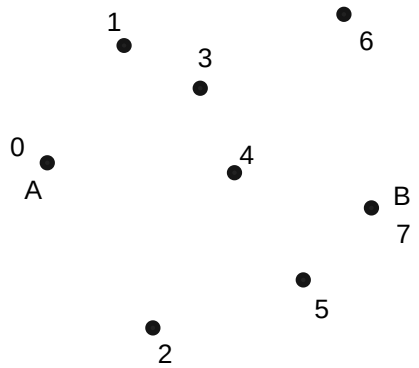


lower

7
5
2
0

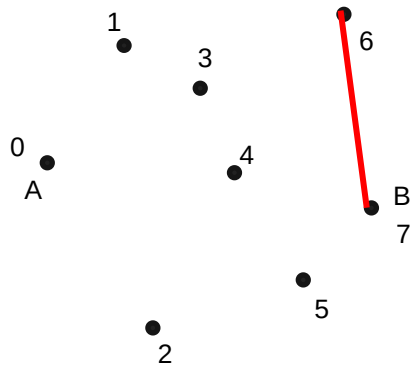
Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.



Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

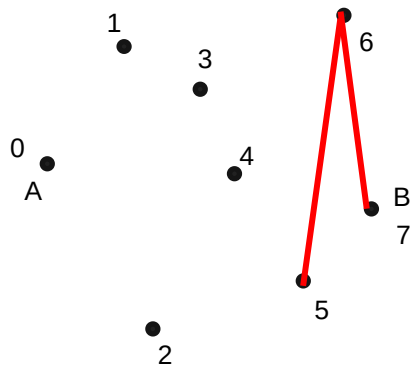


upper

6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.



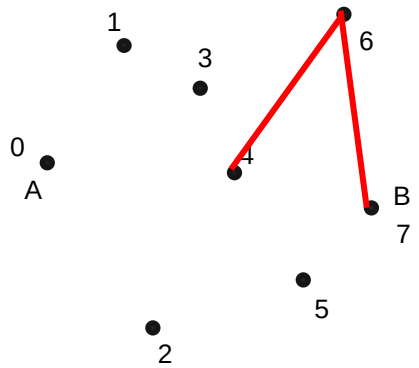
upper

5
6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

6, 5, 4는 시계방향입니다. 5를 빼고 4를 넣습니다.



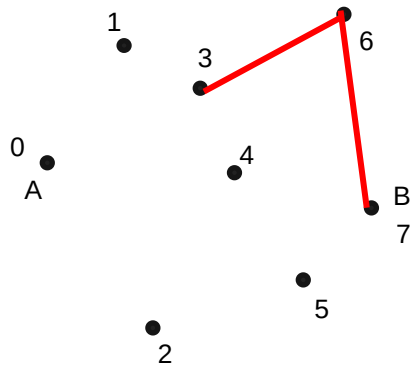
upper

4
6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

6, 4, 3은 시계방향입니다. 4를 빼고 3을 넣습니다.

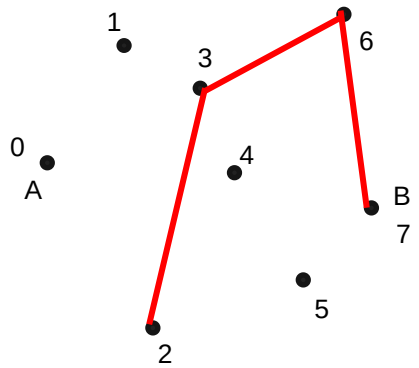


upper

3
6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.



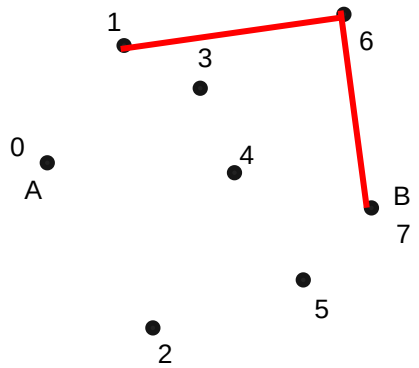
upper

2
3
6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

3, 2, 1은 시계방향입니다. 2를 빼줍니다.
6, 3, 1도 시계방향입니다. 3을 빼줍니다.

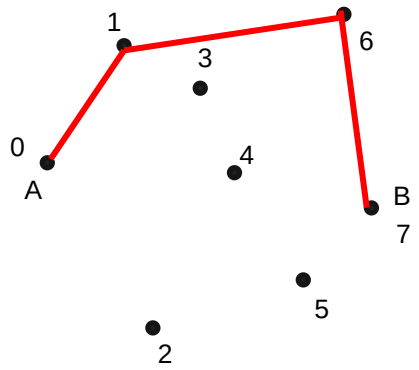


upper

1
6
7

Convex Hull(볼록 껍질) – Monotone Chain

3. 정렬의 역순으로 순회하며 Graham scan에서 스택에 점을 넣듯이 점을 넣습니다. 이것이 upper hull이 됩니다.

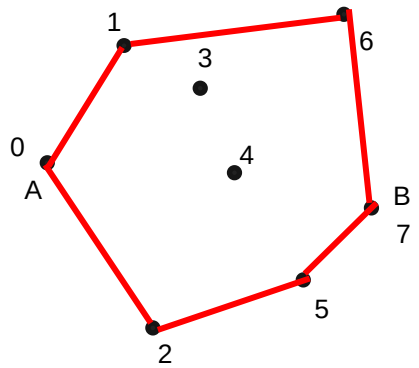


upper

0
1
6
7

Convex Hull(볼록 껍질) – Monotone Chain

4. upper hull과 lower hull을 합쳐주면 convex hull이 나옵니다.



upper	lower
0	7
1	5
6	2
7	0

Convex Hull(볼록 껍질) – Monotone Chain

```cpp
vector<Point> monotone_chain(vector<Point> a) {
  if (a.size() <= 2) return a;
  int N = a.size();
  sort(a.begin(), a.end(), [] (Point a, Point b) {
      if(a.x == b.x) return a.y < b.y;
      return a.x < b.x;
  });
  vector<Point> upper, lower;
  for(int i=0;i<N;++i) {
      while(upper.size()>=2 && cross(upper[upper.size()-2], upper[upper.size()-1], a[i]) <= 0) upper.pop_back();
      upper.push_back(a[i]);
  }
  for(int i=N-1;i>=0;--i) {
      while(lower.size()>=2 && cross(lower[lower.size()-2], lower[lower.size()-1], a[i]) <= 0) lower.pop_back();
      lower.push_back(a[i]);
  }
  vector<Point> ret;
  for(int i=0;i<upper.size();++i) ret.push_back(upper[i]);
  for(int i=1;i<lower.size()-1;++i) ret.push_back(lower[i]);
  return ret;
}
```

Convex Hull(볼록 껍질) – Special Case

기하 문제들의 경우 항상 예외 처리를 할 것이 많다고 했습니다.

주어지는 세 점이 한 직선 위에 존재할 때 그 직선 위에 점들을 전부 convex hull에 포함해야 되는 경우가 convex hull을 만들 때 특수한 경우입니다.

이를 위해서는 스택에서 점들을 뺄 때, 일직선일때는 빼지 않는 것입니다.

Monotone Chain은 이것으로 충분하지만 Graham Scan은 정렬 했을 때 각도가 제일 큰 점들의 순서를 뒤집는 것까지 필요합니다.

백준 2254번 감옥 건설

감옥 건설 성공

☆

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	1417	436	296	28.380%

문제

소들의 반란이 있은 뒤, 이 소들은 포로로 잡은 인간들을 감시해야 했다.

소들은 (Px, Py)의 위치에 감옥을 짓고, 감옥 둘레에 가능한 한 여러 겹으로 담을 쌓아 포로들이 도망가기 힘들도록 하려 한다. 감옥은 하나의 점으로 표현된다.

이러한 목적을 달성하기 위해, 소들은 감옥 주변에 N개의 담 기둥을 세웠다. 각각의 담은 감옥을 완전히 감싸야 하고, 담 안에 (부분적으로라도) 포함되는 담이 있다면 이러한 담도 완전히 감싸야 한다. 즉, 담벼락이 교차하거나 한 점에서 만나서는 안 된다. 감옥과 담 기둥 중 어느 세 점도 일직선상에 있지 않다.

이러한 담 기둥들이 주어졌을 때, 겹치지 않는 최대의 중첩된 담의 겹 수를 구하는 프로그램을 작성하시오.

담은 여러 개의 담벼락이 연결된, 닫힌 다각형을 의미하고, 각각의 담벼락의 두 끝 점은 담 기둥 이어야 한다. 이러한 담 사이에는 반드시 약간이라도 공간이 있어야 한다. 즉, 서로 다른 두 담이 하나의 담벼락이나 담 기둥을 공유해서는 안 된다.

입력

첫째 줄에 N(1≤N≤1,000), Px, Py (-100,000≤Px, Py≤100,000)이 주어진다. 다음 N개의 줄에는 차례로 담 기둥의 좌표가 주어진다. 각각의 좌표는 절댓값이 100,000을 넘지 않는 정수이다.

백준 2254번 감옥 건설

문제에서 눈치 채야 하는 점은 담 기둥을 사용해 담을 만들면 그 모양은 담 기둥을 이용해 만든 convex hull이라는 점입니다.

이를 통해서 여러 겹의 담을 만드는 것은 담 기둥을 통해서 convex hull을 만들고 convex hull에 속하는 담 기둥은 제외하고 제외한 담 기둥으로 convex hull을 만드는 과정임을 알 수 있습니다.

백준 2254번 감옥 건설

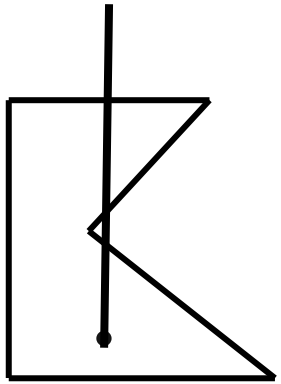
하지만 문제에서 convex hull 안에 항상 감옥이 존재해야한다고 했습니다.

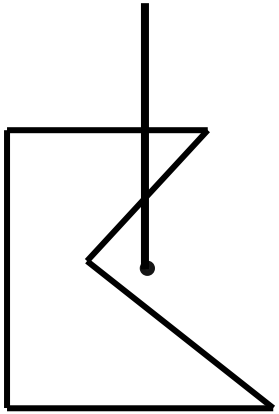
다각형이 하나 있고 어떤 점이 그 다각형 안에 존재하는지는 어떻게 확인해야 할까요?

백준 2254번 감옥 건설

다각형 안에 어떤 점이 있는지 확인하는 것은 점에서 적당한 반직선을 하나 그어서 이 반직선과 다각형이 만나는 횟수가 홀수 번인지 확인하면 됩니다.

홀수번이라면 안에 있는 것이고 아니라면 다각형 밖에 존재합니다.

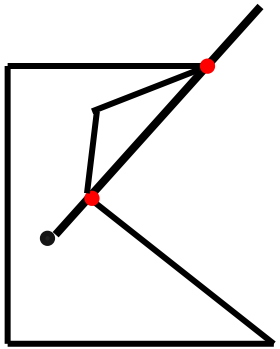




백준 2254번 감옥 건설

여기서 반직선을 그을 때 조심해야 되는 것이 다각형의 꼭짓점을 지나는 반직선 긋게 되면 이 판별이 제대로 안될 때도 있습니다.

아래와 같은 경우 두 번 만나지만 다각형 안에 있습니다.



백준 2254번 감옥 건설

이 방법 말고 볼록 다각형에서 쓸 수 있는 방법이 있습니다.

볼록 다각형의 점들을 반시계 방향으로 나열한 것을 P라고 합시다.

그러면 어떤 점 X가 다각형 안에 존재한다는 것은 모든 CCW(X, P_i, P_{i+1})가 동일한 방향이라는 뜻입니다.

백준 2254번 감옥 건설

우리가 만드는 convex hull은 볼록 다각형이기 때문에 점이 볼록 다각형 안에 있는지를 판단하는 것을 통해서 감옥이 담 안에 있는지 판단할 수 있습니다.

이를 통해서 감옥이 안에 존재하는 convex hull을 더 이상 만들 수 없을 때까지 convex hull을 만들면 답을 구할 수 있습니다.

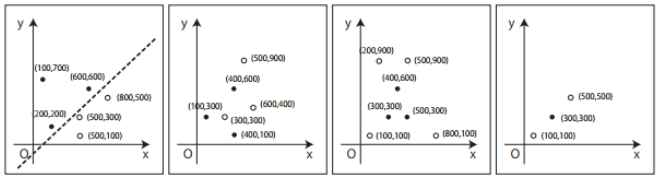
convex hull을 더 이상 만들 수 없는 경우는 모든 점이 한 직선 위에 있을 때 입니다.

백준 3878번 점 분리

문제

평면 위에 여러 개의 검정 점과 흰 점이 있다. 이때, 길이가 무한대인 직선을 그어 흰 점과 검은 점을 분리하려고 한다. 직선은 어떤 점과도 만나면 안 된다. 직선으로 인해서 나누어지는 두 그룹 중 한 그룹에는 흰 점만 있어야 하고, 다른 그룹에는 검은 점만 있어야 한다.

아래 그림에서 제일 왼쪽 예제는 점선으로 표시된 직선으로 두 점을 나눌 수 있다. 하지만 나머지 예제는 직선으로 점을 분리할 수 없다.



흰 점과 검은 점의 좌표가 주어졌을 때, 직선으로 점을 분리할 수 있는지 없는지를 알아내는 프로그램을 작성하시오.

입력

첫째 줄에는 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스의 첫째 줄에 검정 점의 개수 n과 흰 점의 개수 m이 공백으로 구분되어 주어진다. n과 m은 100보다 작거나 같다. 다음 줄부터 n개의 줄에는 검정 점의 좌표가 공백으로 구분되어 주어진다. 그 다음 m개의 줄에는 흰 점의 좌표가 주어진다.

모든 점의 x, y좌표는 0보다 크거나 같고, 10000보다 작거나 같은 정수이다. 또한, 같은 위치에 점이 2개 이상 있는 경우는 없다.

출력

각각의 테스트 케이스에 대해서, 점을 문제의 설명대로 분리할 수 있으면 YES를, 아니면 NO를 출력한다.

백준 3878번 점 분리

흰색 점과 검은색 점을 하나의 직선으로 분리가 가능한지 판단해야 하는 문제입니다.

이를 판단하는 방법은 흰색 점들로 만든 convex hull과 검은색 점들로 만든 convex hull이 교차하는가로 판별할 수 있습니다.

백준 3878번 점 분리

만약 어떤 직선이 흰 점과 검은 점을 분리할 수 있다면 당연히 두 점으로 만든 convex hull은 교차하지 않습니다.

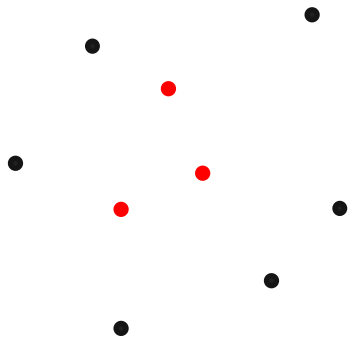
그리고 convex hull끼리 교차하지 않는다면 당연히 분리할 수 있는 직선이 존재합니다.

따라서 분리 가능한 직선이 존재한다는 것과 두 convex hull이 교차하지 않는다는 것은 동치입니다.

백준 3878번 점 분리

사실 방금 전의 슬라이드의 말을 틀렸습니다.

교차하지 않지만 하나의 convex hull이 다른 convex hull을 포함한다면 분리가 불가능하기 때문입니다. 이 경우를 제외하면 성립합니다.



백준 3878번 점 분리

convex hull끼리 교차하는지 판단한다고 했는데 말은 쉽지만 코드로 나타내기엔 꽤 어렵습니다.

먼저 convex hull을 구하고 각 convex hull의 점이 다른 convex hull 안에 존재하는지를 판단합니다.

교차하지 않으려면 어떠한 점도 다른 convex hull에 포함되어선 안됩니다.

백준 3878번 점 분리

그 다음엔 하나의 convex hull의 선분이 다른 convex hull의 선분들과 교차하는지 판별해줘야 합니다.

이걸로 끝입니다만 점이 한 개일 때, 직선일 때 등 생각해줄 케이스가 꽤 많으니 주의해서 코딩을 합시다.

Problem Set

- 필수 문제
  - 16491 대피소 찾기
  - 2254 감옥 건설
  - 3878 점 분리

“Any Question?”