

계산기하학




학습 목표


- ✎ 기하의 기본 연산들과 그의 응용 문제를 학습한다.
- ✎ 볼록 외피 알고리즘에 대해 학습한다.
- ✎ 최근접쌍 문제에 대한 분할 정복 알고리즘을 학습한다.
- ✎ 평면 소거법과 그의 응용 문제를 학습한다.



차례


 계산 기하학 소개와 기하 물체의 표현

 기본 기하 연산

 다각형

✓ 다각형 면적 계산


✓ 다각형 포함 문제

 볼록 외피(Convex Hull)

 최근접쌍(Closest Pair of Points)

 평면 소거법(Plane Sweeping)

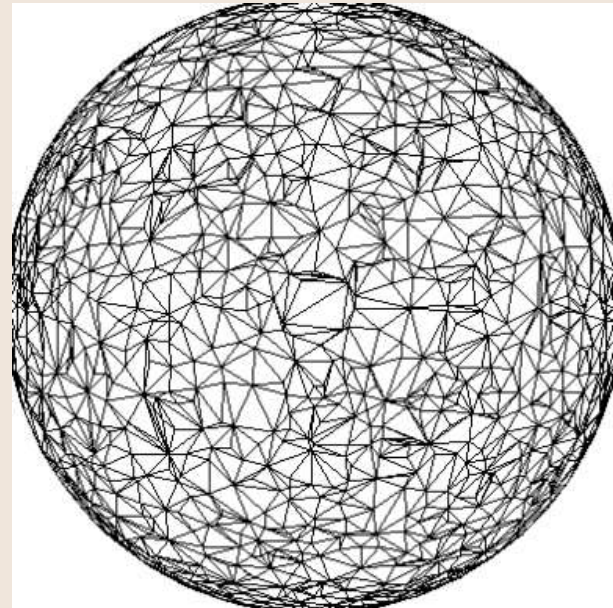
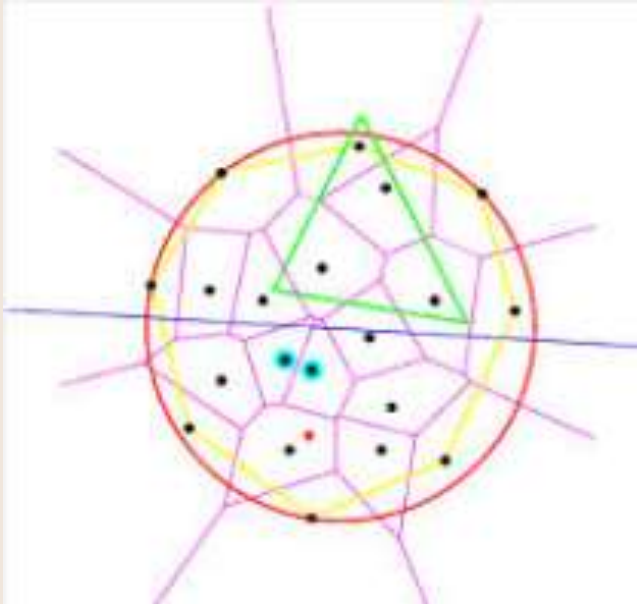
✓ 직사각형 합집합

 응용 문제



계산기하학(Computational Geometry)

- ✎ 컴퓨터 알고리즘의 한 분야
- ✎ 기하 물체를 대상으로 하는 문제를 조합론 및 알고리즘적인 해법으로 해결함



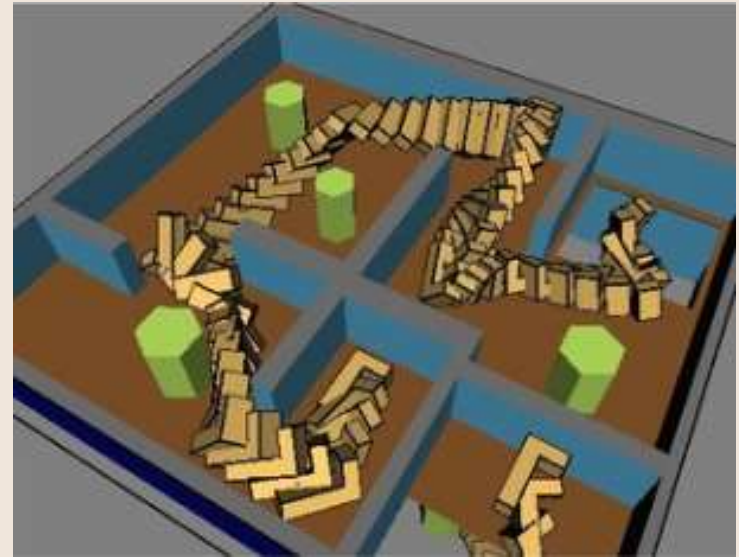
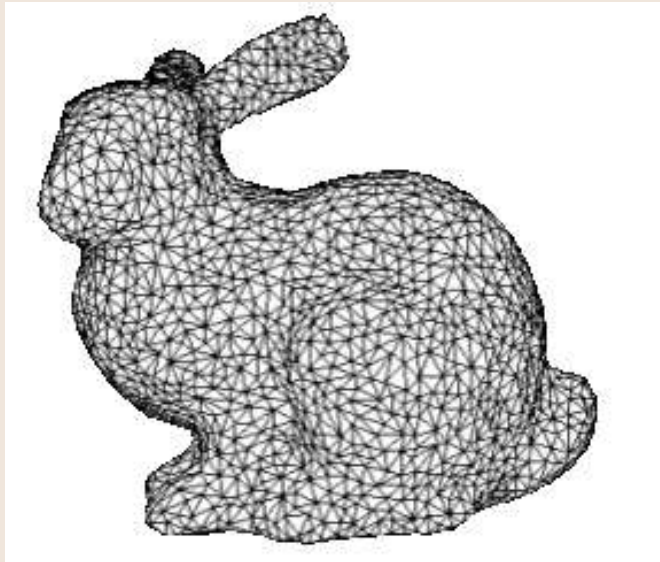


계산기하학(Computational Geometry)



응용 분야

✓ 컴퓨터 그래픽스, 로봇틱스, VLSI 디자인, CAD 등





기하 물체



0차원 : 점



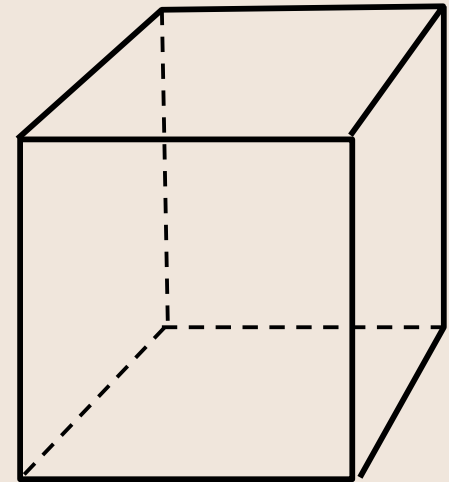
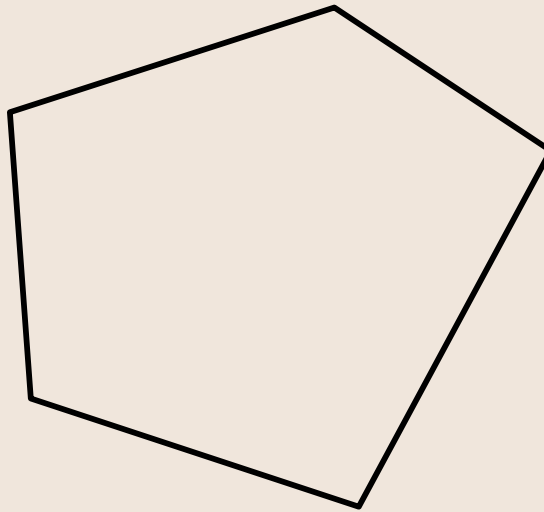
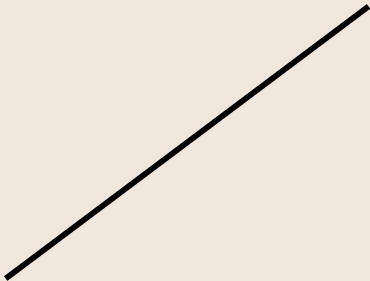
1차원 : 직선, 선분, 곡선



2차원 : 다각형, 원, 곡면



3차원 : 다면체, 구



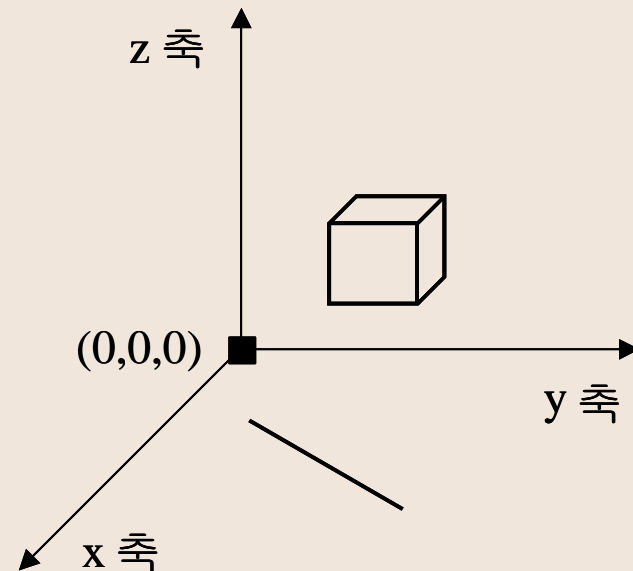
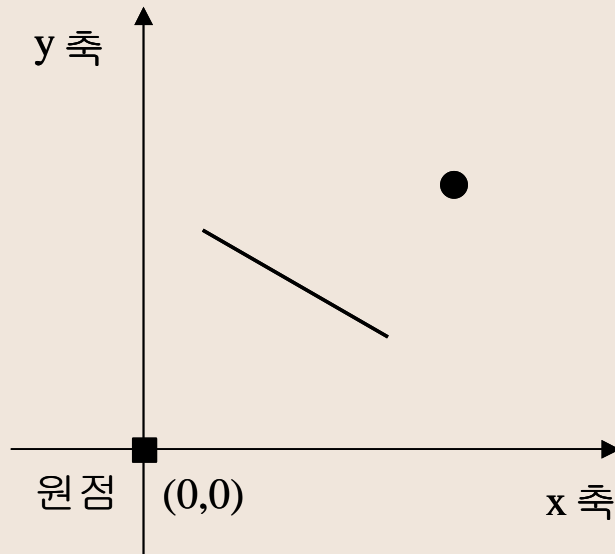


기하 물체의 표현



공간과 좌표계

- ✓ 2차원 공간(평면)의 좌표계
- ✓ 3차원 공간의 좌표계

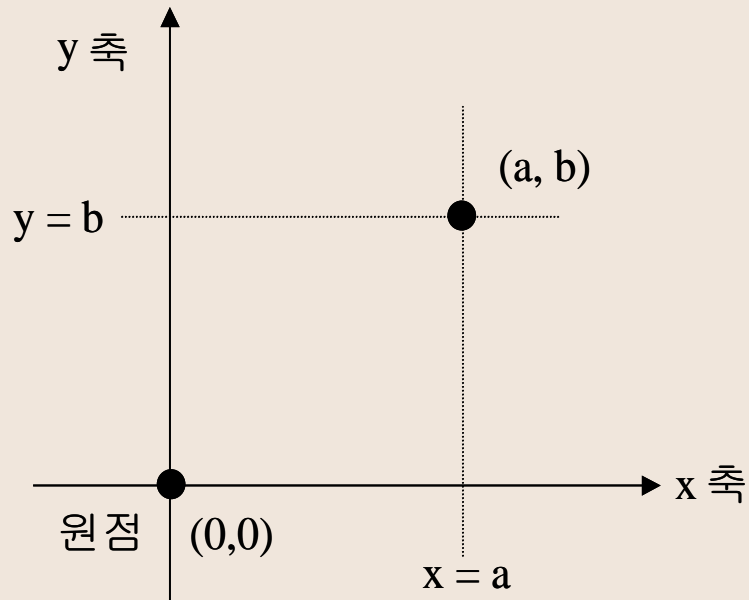




기하 물체의 표현



2차원 좌표계에서의 점



프로그램에서 점의 표현

```
typedef struct point{  
    int x;  
    int y;  
} Point;
```




기하 물체의 표현

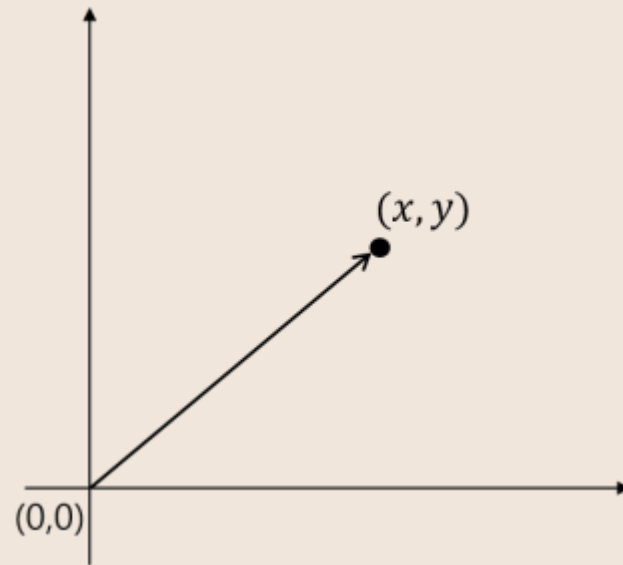
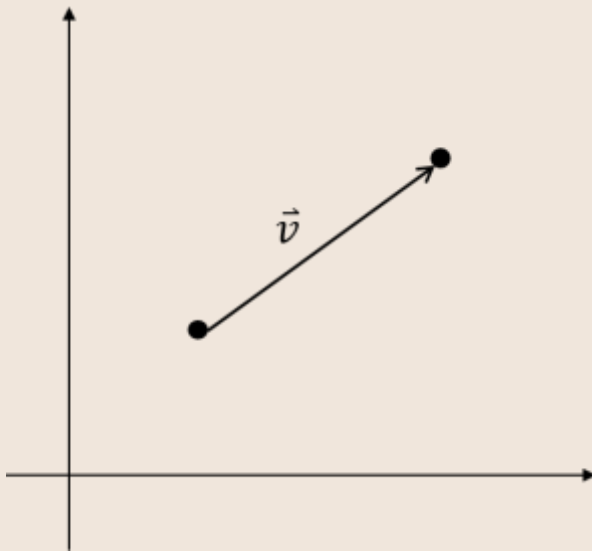


벡터에 의한 점의 표현

- ✓ 벡터(vector) : 두 점 사이의 상대적 위치 표시

시작점에서 끝점으로의 화살표로 표시

- ✓ 점의 좌표 (x, y) : 원점에서 끝점 (x, y) 로의 벡터

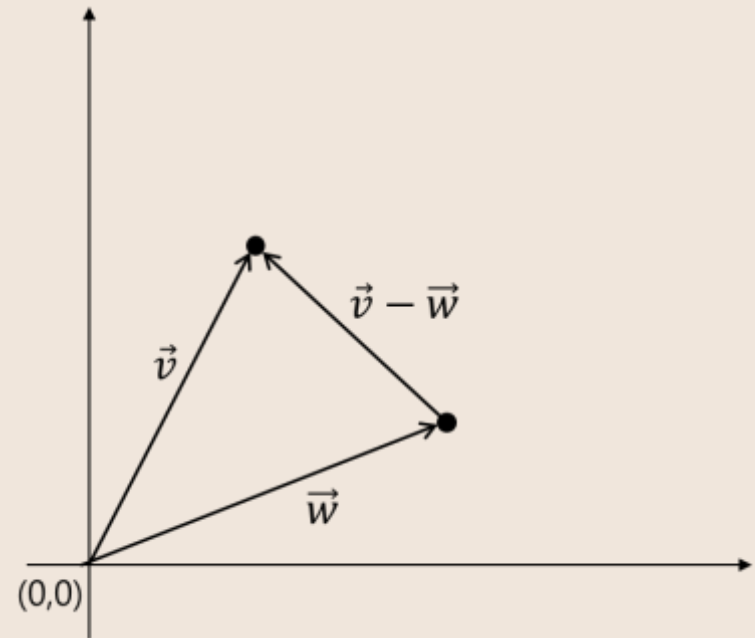
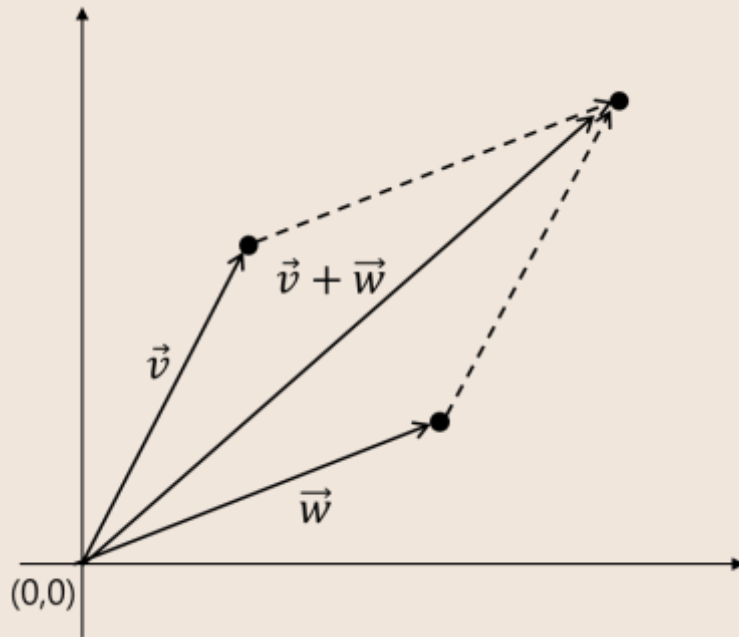




기하 물체의 표현



벡터의 덧셈과 뺄셈





기하 물체의 표현

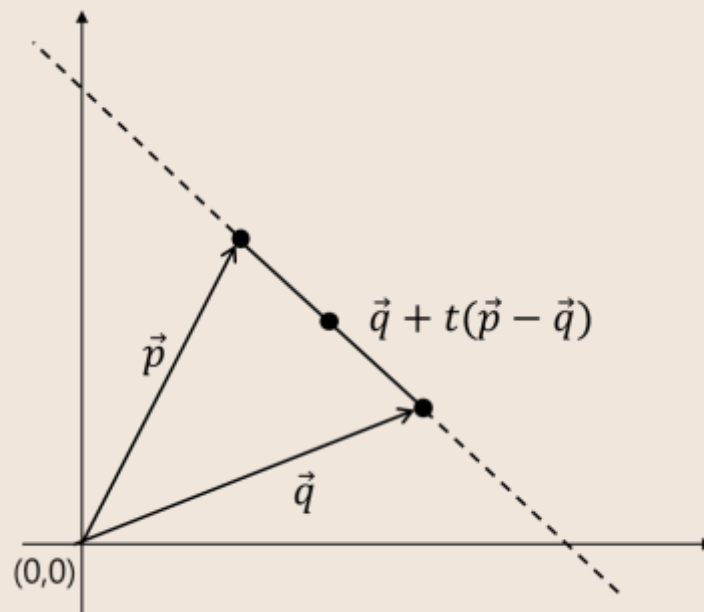


벡터에 의한 선분과 선의 표현

✓ 두 점 $p = (x_1, y_1)$ 과 $q = (x_2, y_2)$ 를 지나는 선분과 선 L

$$L = \vec{q} + t(\vec{p} - \vec{q}) = t\vec{p} + (1 - t)\vec{q} \quad \text{if } 0 \leq t \leq 1 \quad (\text{선분})$$

$$\text{또는 } -\infty \leq t \leq \infty \quad (\text{선})$$





기본 기하 연산

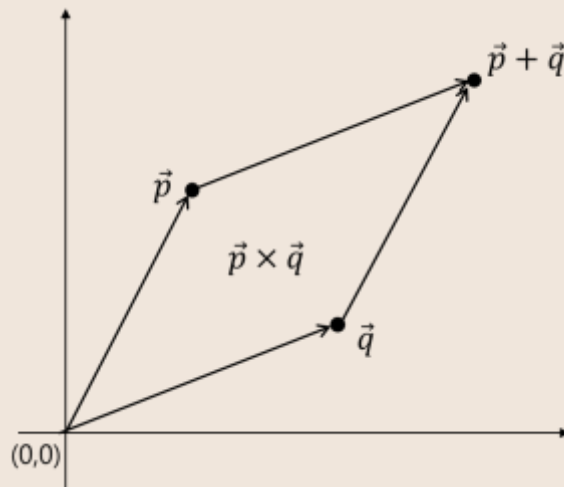


벡터의 외적

- 두 벡터 $\vec{p} = (p_x, p_y)$ 와 $\vec{q} = (q_x, q_y)$ 에 대해서,

$$\vec{p} \times \vec{q} = \det \begin{pmatrix} p_x & p_y \\ q_x & q_y \end{pmatrix} = p_x q_y - p_y q_x = -\vec{q} \times \vec{p}$$

- 외적 $\vec{p} \times \vec{q}$ 은 점 $(0, 0)$, \vec{p} , \vec{q} , $\vec{p} + \vec{q}$ 을 꼭지점으로 갖는 평행 사변형의 부호화된 면적을 나타냄





기본 기하 연산



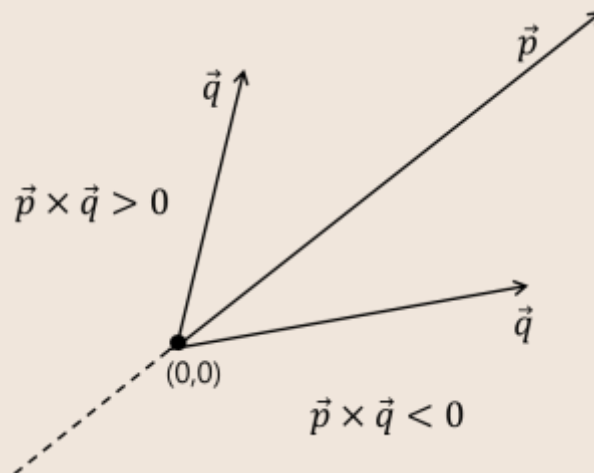
벡터의 외적

✓ 두 벡터 $\vec{p} = (p_x, p_y)$ 와 $\vec{q} = (q_x, q_y)$ 에 대해서,

$\vec{p} \times \vec{q} > 0$ 이면, \vec{q} 가 \vec{p} 로부터 반시계 방향으로 180도 이내에 있음

$\vec{p} \times \vec{q} < 0$ 이면, \vec{q} 가 \vec{p} 로부터 시계 방향으로 180도 이내에 있음

$\vec{p} \times \vec{q} = 0$ 이면, \vec{q} 는 \vec{p} 와 일직선 상에 있음





기본 기하 연산



ccw (counter colokwise) 함수

```
// 원점에서 벡터 q가 벡터 p에 반시계 방향이면 양수, 시계방향 (cw: clockwise)이면 음수
```

```
// 평행이면, 0 을 반환한다
```

```
int ccw2(Point p, Point q){  
    return p.x*q.y - p.y*q.x  
}
```

```
// 점 r을 기준으로 벡터  $\vec{rq}$ 가 벡터  $\vec{rp}$ 에 반시계 방향이면 양수,
```

```
// 시계방향이면 음수, 평행이면, 0 을 반환한다
```

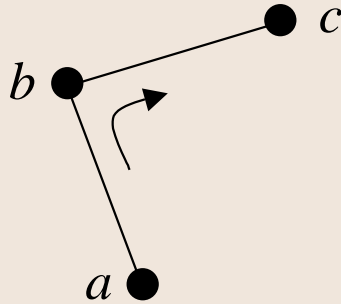
```
int ccw(Point r, Point p, Point q){  
    return ccw2(p - r, q - r)  
}
```



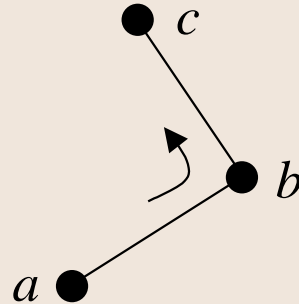
기본 기하 연산



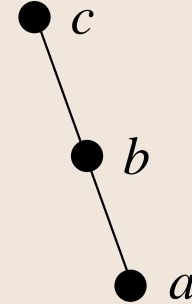
세 점 $a = (a_x, a_y)$, $b = (b_x, b_y)$, $c = (c_x, c_y)$ 의 회전방향



rightTurn
 $ccw(a, b, c) < 0$



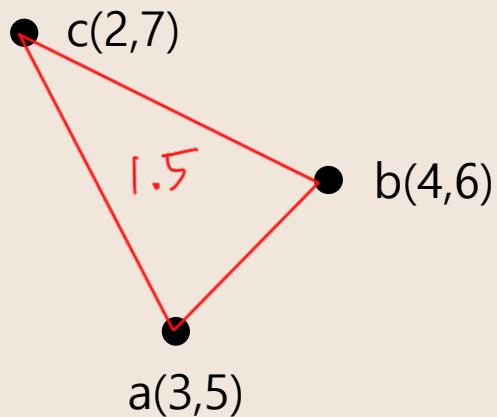
leftTurn
 $ccw(a, b, c) > 0$



collinear
 $ccw(a, b, c) = 0$



기본 기하 연산

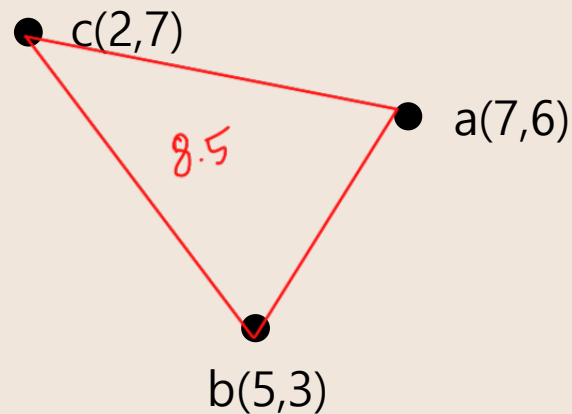


$$\overrightarrow{ab} = \vec{p} = (1,1)$$

$$\overrightarrow{ac} = \vec{q} = (-1, 2)$$

$$ccw(a, b, c) = ccw2(\vec{p}, \vec{q}) \quad \begin{vmatrix} 1 & 1 \\ -1 & 2 \end{vmatrix} = 2 - (-1) = 3$$

$$\begin{vmatrix} 3 & 5 & 1 \\ 4 & 6 & 1 \\ 2 & 7 & 1 \end{vmatrix} = 18 + 28 + 10 - 12 - 20 - 21 = 3$$



$$\overrightarrow{ab} = \vec{p} = (-2,-3)$$

$$\overrightarrow{ac} = \vec{q} = (-5, 1)$$

$$ccw(a, b, c) = ccw2(\vec{p}, \vec{q}) \quad \begin{vmatrix} -2 & -3 \\ -5 & 1 \end{vmatrix} = -2 - 15 = -17$$

$$\begin{vmatrix} 7 & 6 & 1 \\ 5 & 3 & 1 \\ 2 & 7 & 1 \end{vmatrix} = 21 + 35 + 12 - 6 - 30 - 49 = -17$$



기본 기하 연산



프로그램

```
// 세 점 a, b, c의 회전 방향이 좌회전인가를 검사
bool leftTurn(Point a, Point b, Point c){
    return ccw(a, b, c) > 0
}

// 세 점 a, b, c의 회전 방향이 우회전인가를 검사
bool rightTurn(Point a, Point b, Point c){
    return ccw(a, b, c) < 0
}

// 세 점 a, b, c의 회전 방향이 일직선인가를 검사
bool collinear(Point a, Point b, Point c){
    return ccw(a, b, c) == 0
}
```



기본 기하 연산



중간검사 프로그램

// 점 c가 선분 (a, b)에 위치하는가를 검사

bool between(Point a, Point b, Point c)

{

IF !collinear(a, b, c) THEN false; *Case 1*

IF a.x != b.x // 수직선이 아니면 *Case 2*

return a.x <= c.x && c.x <= b.x ||

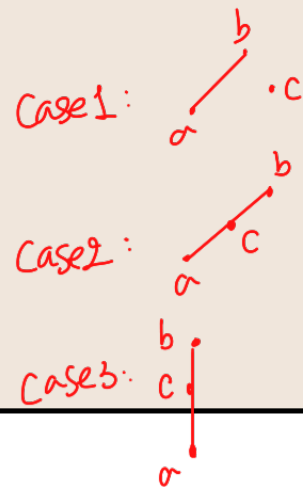
b.x <= c.x && c.x <= a.x

ELSE

return a.y <= c.y && c.y <= b.y ||

b.y <= c.y && c.y <= a.y

}



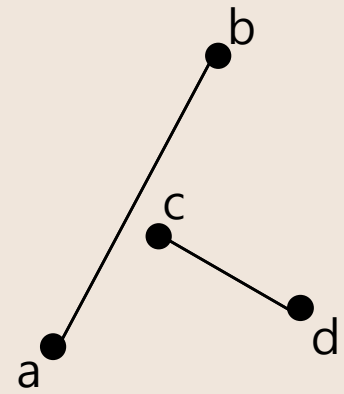
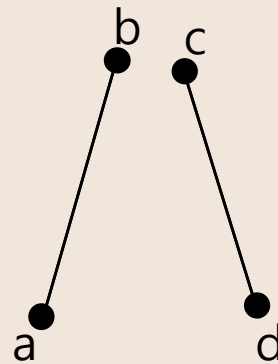
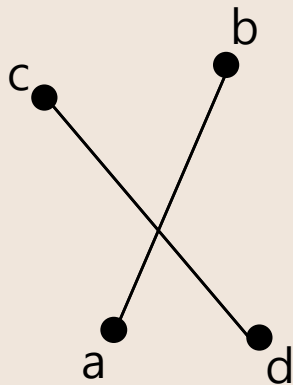
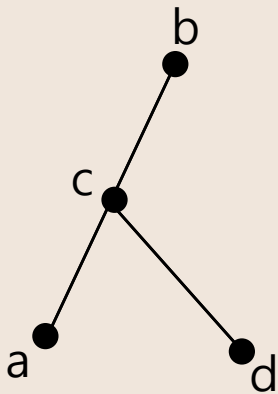


기본 기하 연산



선분 교차 검사

- ✓ 두 선분 (a, b) 와 (c, d) 의 교차 여부를 검사





기본 기하 연산



프로그램

```
// 세 점 a, b, c의 회전 방향 검사에서 좌회전이면 1,  
// 우회전이면 -1, 일직선이면 0을 반환  
int direction(Point a, Point b, Point c)  
{  
    IF (ccw(a, b, c) < 0) THEN return -1  
    IF (ccw(a, b, c) > 0) THEN return 1  
    return 0;  
}
```

```
// 선분의 끝점이 교차점이 되는 경우를 제외하는 경우  
bool intersectProp(Point a, Point b, Point c, Point d)  
{  
    return direction(a, b, c)*direction(a, b, d) < 0 &&  
           direction(c, d, a)*direction(c, d, b) < 0  
}
```



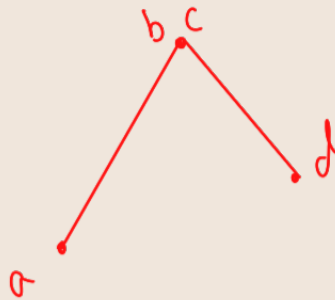


기본 기하 연산



프로그램

```
// 선분의 끝점이 교차점이 되는 경우를 허용하는 경우
bool intersect(Point a, Point b, Point c, Point d)
{
    return    direction(a, b, c)*direction(a, b, d) <= 0 &&
              direction(c, d, a)*direction(c, d, b) <= 0
}
```



다각형

다각형(polygon)

- ✓ 볼록 다각형(convex polygon) : 모든 내각이 180도 미만인 다각형
- ✓ 오목 다각형(concave polygon): 180도가 넘는 내각을 갖는 다각형
- ✓ 단순 다각형(simple polygon) : 다각형의 경계가 스스로를 교차하지 않는 다각형

볼록 다각형의 특별한 성질

- ✓ 볼록 다각형 내부의 임의의 두 점을 연결하는 선분은 볼록 다각형의 테두리를 절대 교차하지 않는다
- ✓ 두 볼록 다각형의 교집합은 항상 볼록 다각형이다

다각형의 구현

- ✓ 다각형의 꼭지점을 나타내는 점들의 리스트 $(p_0, p_1, \dots, p_{n-1})$

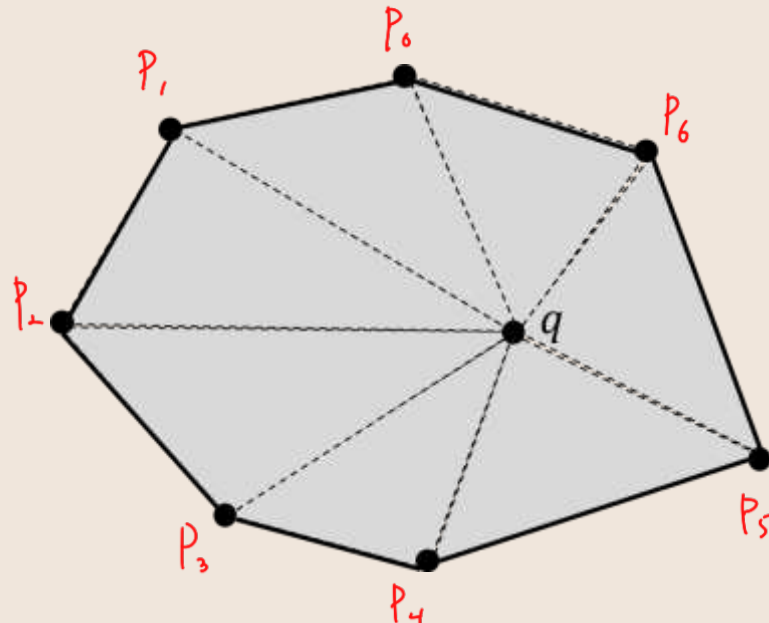


다각형의 면적



다각형 면적 구하기

- ✓ 다각형 P 가 n 개의 정점들을 반시계 방향으로 나열해서 p_0, p_1, \dots, p_{n-1} 로 주어진다
- ✓ 볼록 다각형의 경우
 - ✱ 다각형 내부의 한 점 q 를 잡아서 아래 그림과 같이 q 와 인접한 두 정점들로 이루어진 삼각형들로 분리할 수 있음
 - ✱ 삼각형의 면적은 벡터의 외적으로 구함



$$\frac{\sum |ccw(q, p_i, p_{i+1})|}{2}$$



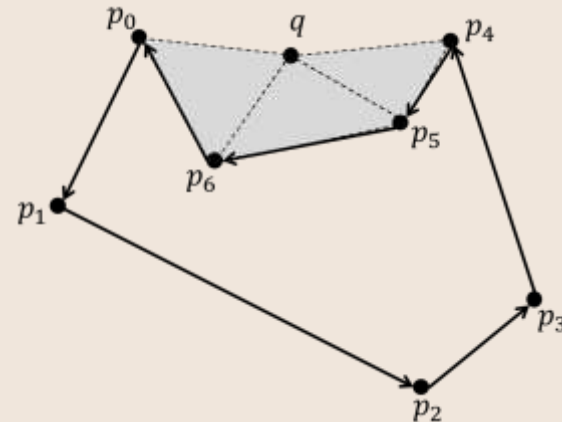
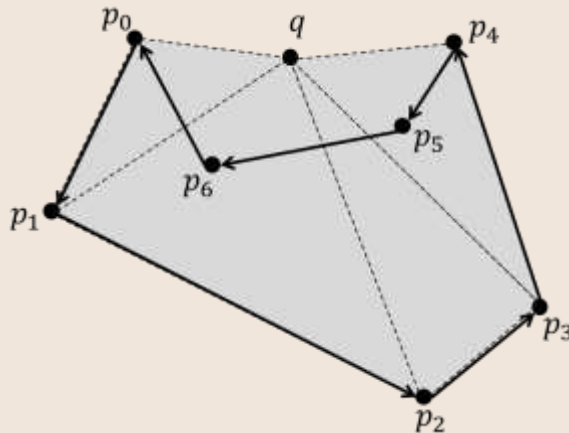
다각형의 면적



다각형 면적 구하기

✓ 오목 다각형의 경우

- ✧ 평면상의 한 점 q 를 잡아보자
- ✧ q 와 다각형의 인접한 두 정점 p_i 와 $p_{(i+1)\%n}$ 를 꼭지점으로 갖는 삼각형의 넓이를 생각
- ✧ 넓이 = $\frac{1}{2}(p_i - q) \times (p_{(i+1)\%n} - q)$
- ✧ 이 넓이는 $q, p_i, p_{(i+1)\%n}$ 가 좌회전할 때 양수이고 우회전할 때 음수이다





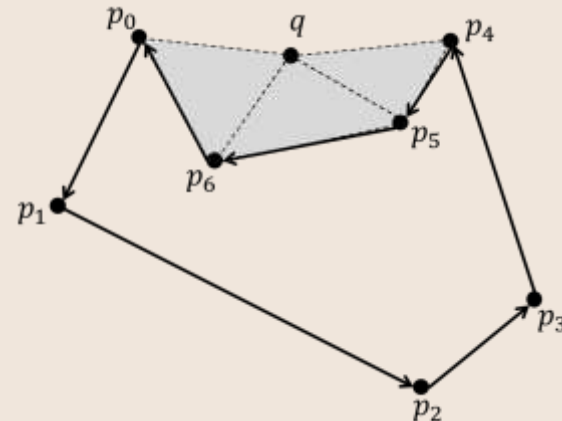
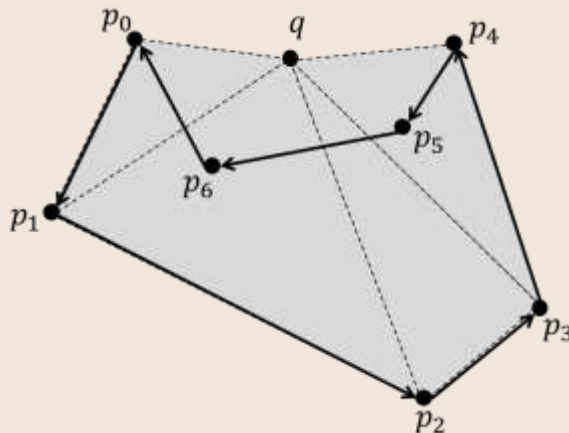
다각형의 면적



다각형 면적 구하기

✓ 오목 다각형의 경우

- ✦ 아래 왼쪽 그림에서 정점 p_0, p_1, \dots, p_4 에서는 좌회전을 함으로 삼각형들의 넓이는 양수
- ✦ 아래 오른쪽 그림에서 정점 p_4, p_5, \dots, p_0 에서는 우회전을 함으로 삼각형들의 넓이는 음수
- ✦ 따라서 모든 삼각형들의 합을 하면 오른쪽 그림의 흰색 부분인 다각형의 내부 면적을 구할 수 있음
- ✦ 다각형의 넓이 $= \frac{1}{2}(p_0 \times p_1 + p_1 \times p_2 + \dots p_{n-2} \times p_{n-1} + p_{n-1} \times p_0)$: q가 원점일때





다각형의 면적



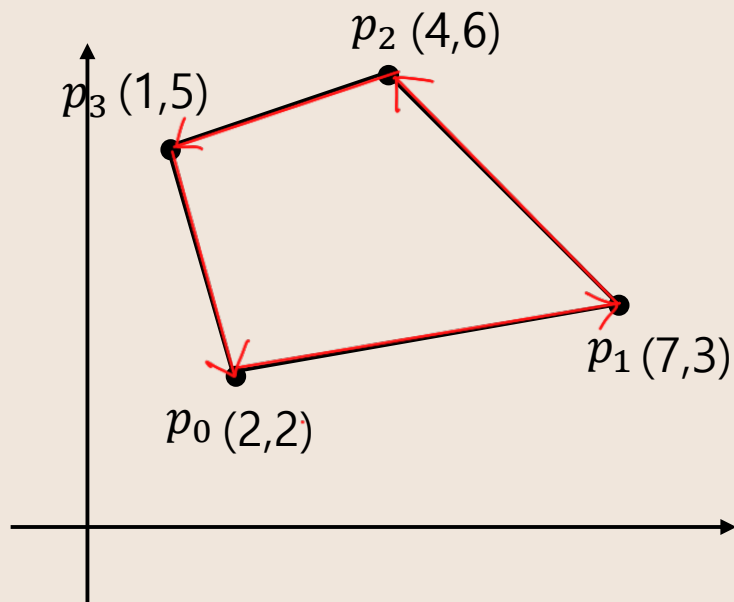
다각형 면적 구하기

```
// 다각형 p는 정점의 위치 벡터의 배열
// n은 다각형의 정점의 개수
double area(Polygon p, int n)
{
    ret ← 0
    FOR i in 0 → n-1
        j ← (i + 1)%n
        ret ← ret + (p[i].x * p[j].y - p[i].y * p[j].x)

    return abs(ret)/2.0
}
```



다각형의 면적



$$\begin{aligned} S &= \frac{1}{2} \times \text{abs}\left(\begin{vmatrix} 2 & 2 \\ 7 & 3 \end{vmatrix} + \begin{vmatrix} 7 & 3 \\ 4 & 6 \end{vmatrix} + \begin{vmatrix} 4 & 6 \\ 1 & 5 \end{vmatrix} + \begin{vmatrix} 1 & 5 \\ 2 & 2 \end{vmatrix}\right) \\ &= \frac{1}{2} \times \text{abs}(2 \times 3 - 2 \times 7 + 7 \times 6 - 3 \times 4 + 4 \times 5 - 6 \times 1 + 1 \times 2 - 2 \times 5) \\ &= \frac{1}{2} \times 28 = 14 \end{aligned}$$

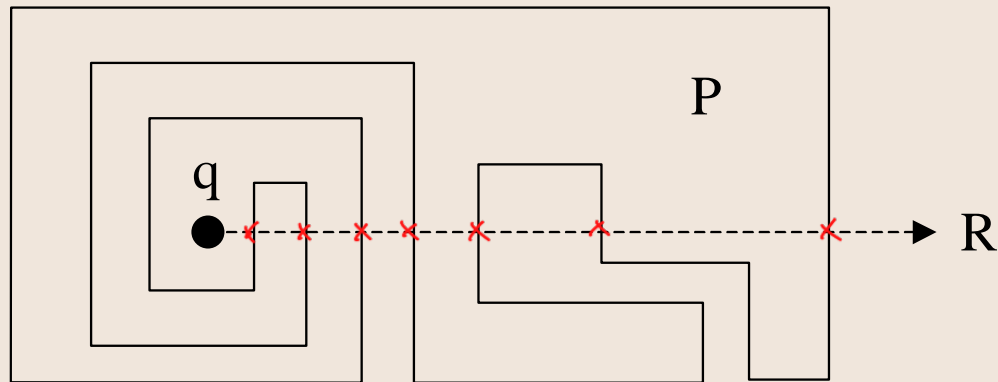


다각형 포함



다각형 포함 문제

- ✓ 다각형 P 와 점 q 가 주어졌을 때, q 가 P 의 내부에 위치하고 있는지 여부를 검사하시오



- ✓ 관찰 : q 를 지나는 반직선 R 과 P 의 경계선과의 교차점의 수가
 - * 홀수이면 q 는 P 의 내부에 있음
 - * 짝수이면 q 는 P 의 외부에 있음



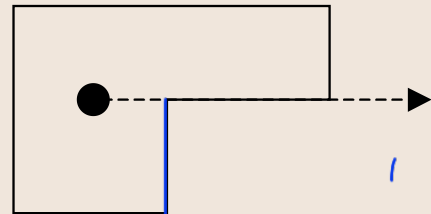
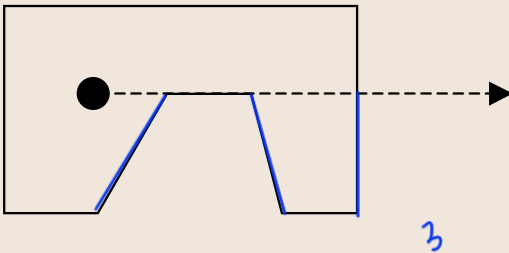
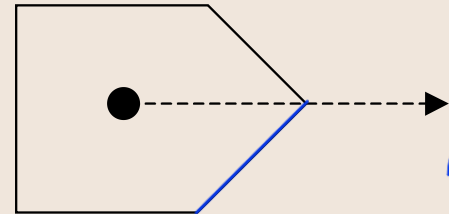
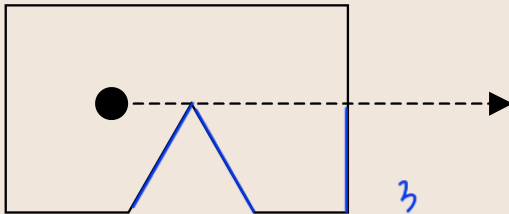
다각형 포함



다각형 포함 문제

✓ 예외 상황

- ✱ 반직선이 정점을 지나는 경우
- ✱ 반직선이 에지를 지나는 경우





다각형 포함



다각형 포함 문제

✓ 해결 방안

- * 반직선 R과 교차하는 에지에 대해서, 에지의 한 쪽 끝점이 R에 위치하면 다른 쪽 끝점이 R의 아래쪽에 있는 경우만 교차점으로 인정한다



교차점 인정



교차점 불인정



다각형 포함

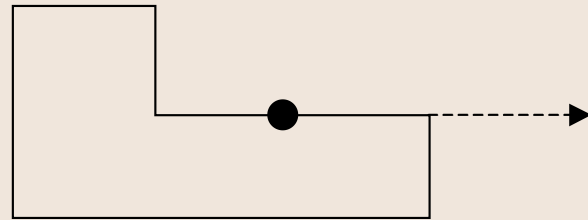
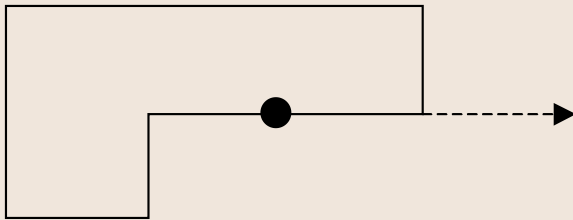


다각형 포함 문제



다른 예외 상황

✦ 점 q 가 P 의 경계선에 있는 경우



해결 방안

✦ q 가 P 의 각 에지에 위치하는지를 검사한다

프로그램

```
bool insidePolygon(Point q, Polygon P, int n) // n은 다각형 p의 정점의 개수
{
    crossings ← 0
    origin ← {0, 0} // 원점
    // q가 원점에 오도록 다각형을 이동시킨다 // 반직선은 양의 x축
    FOR i in 0 → n-1
        P[i].x ← P[i].x - q.x
        P[i].y ← P[i].y - q.y
    FOR i in 0 → n-1
        iPlus1 ← (i+1) % n // i = n-1이면 iPlus1 = 0
        IF between(P[i], P[iPlus1], origin)
            return false // q가 p의 에지에 놓여 있다
```



```
// 에지가 양의 x축과 교차하면 crossings에 1 증가
```

```
IF P[i].y < 0 && P[iPlus1].y >= 0 &&
```

```
    leftTurn(P[i], P[iPlus1], origin) ||
```

```
    P[iPlus1].y < 0 && P[i].y >= 0 &&
```

```
    leftTurn(P[iPlus1], P[i], origin)
```

```
THEN crossings++
```

```
IF crossings % 2
```

```
    return true
```

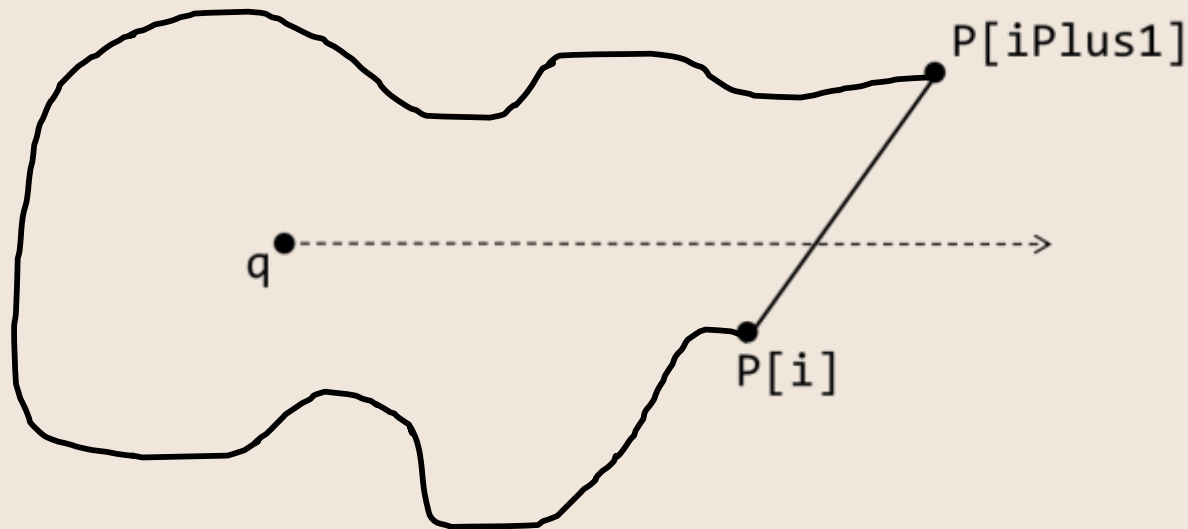
```
// crossings가 홀수이면
```

```
return false
```

```
}
```

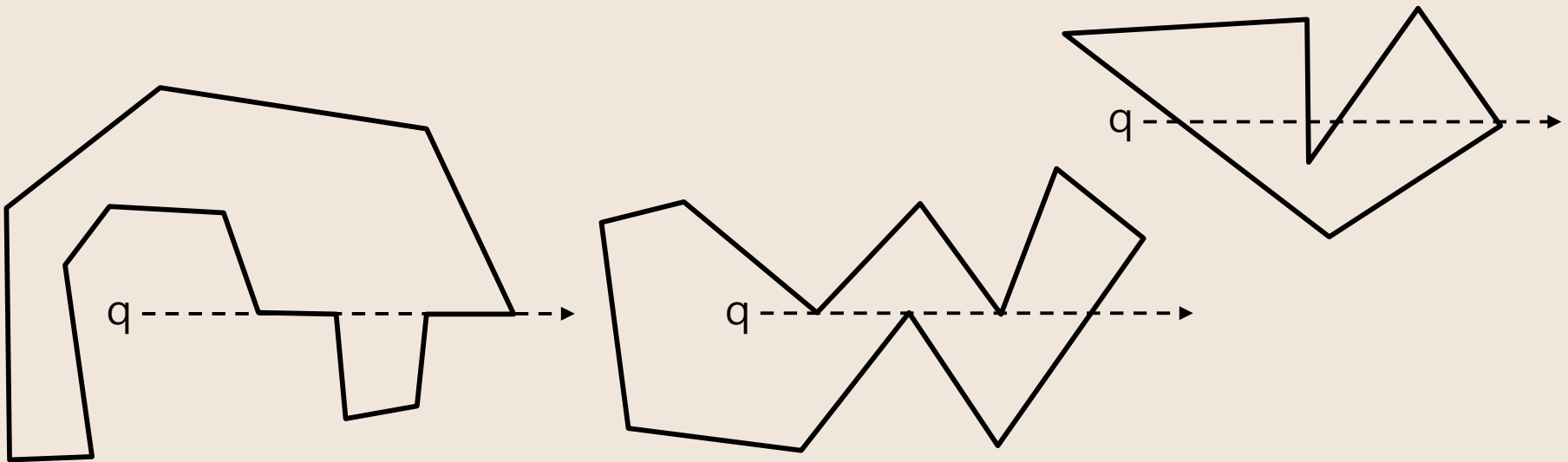
프로그램

```
// 에지가 양의 x축과 교차하면 crossings에 1 증가  
IF P[i].y < 0 && P[iPlus1].y >= 0 &&  
    leftTurn(P[i], P[iPlus1], origin) ||  
    P[iPlus1].y < 0 && P[i].y >= 0 &&  
    leftTurn(P[iPlus1], P[i], origin)  
THEN crossings++
```



프로그램

```
// 에지가 양의 x축과 교차하면 crossings에 1 증가  
IF P[i].y < 0 && P[iPlus1].y >= 0 &&  
    leftTurn(P[i], P[iPlus1], origin) ||  
    P[iPlus1].y < 0 && P[i].y >= 0 &&  
    leftTurn(P[iPlus1], P[i], origin)  
THEN crossings++
```



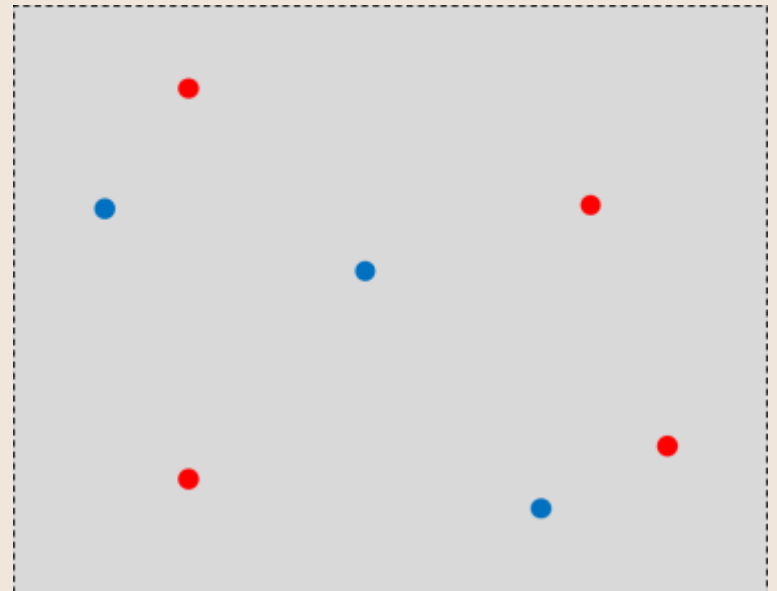
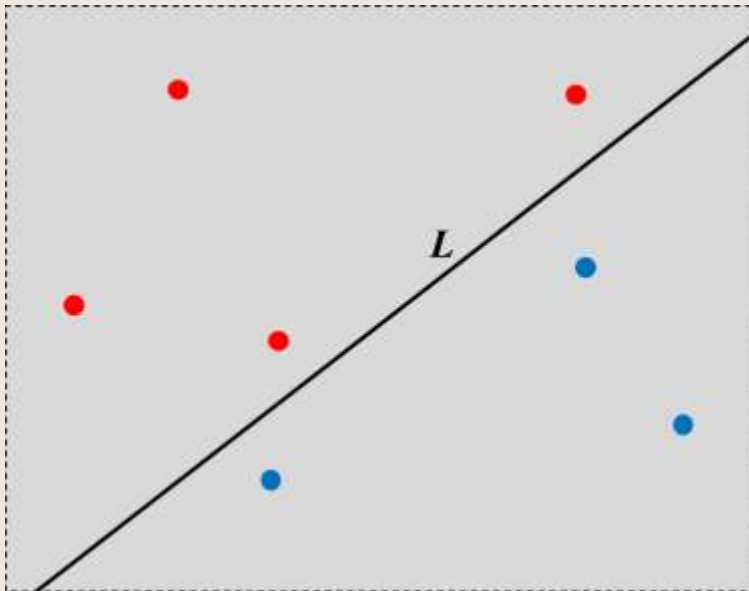


볼록 외피(Convex Hull)



문제

- ✓ 평면 상에 빨간색, 파란색 점들이 주어진다
- ✓ 직선 L 을 그어서 평면을 두 영역으로 나눌 때 각 영역에 같은 색의 점만 포함하도록 직선 L 이 존재하는가?
- ✓ 아래 오른쪽 그림은 직선 L 이 존재하지 않는 경우



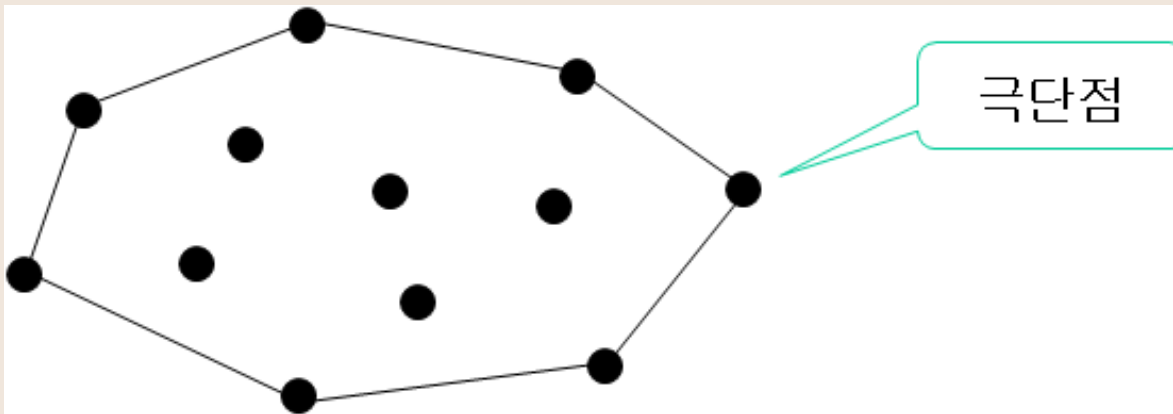


볼록 외피

✎ 앞의 문제를 해결하기 위해 볼록 외피를 이용할 수 있음

✎ 볼록 외피(convex hull)

- ✓ 주어진 점들을 모두 둘러싸는 가장 작은 볼록 다각형
- ✓ 극단점(extreme point) : 볼록 외피의 정점이 되는 점



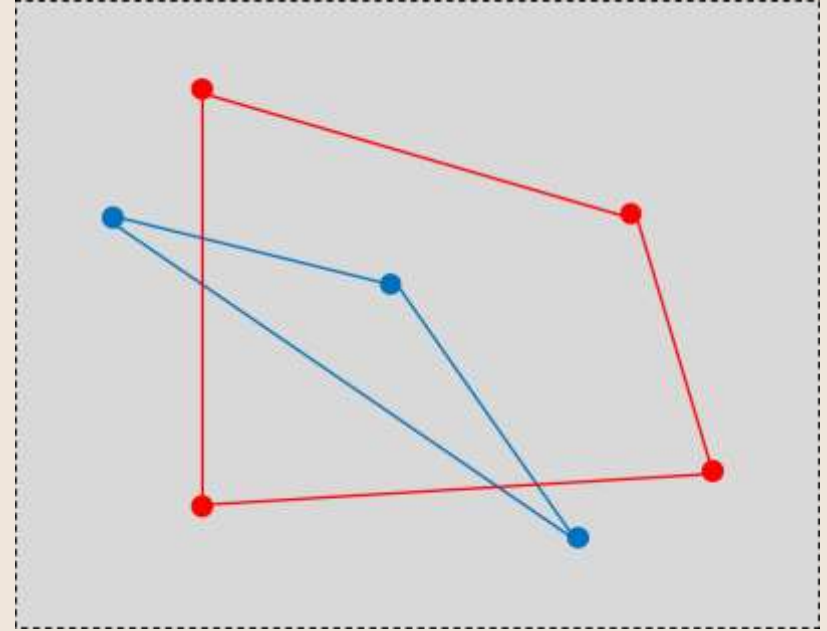
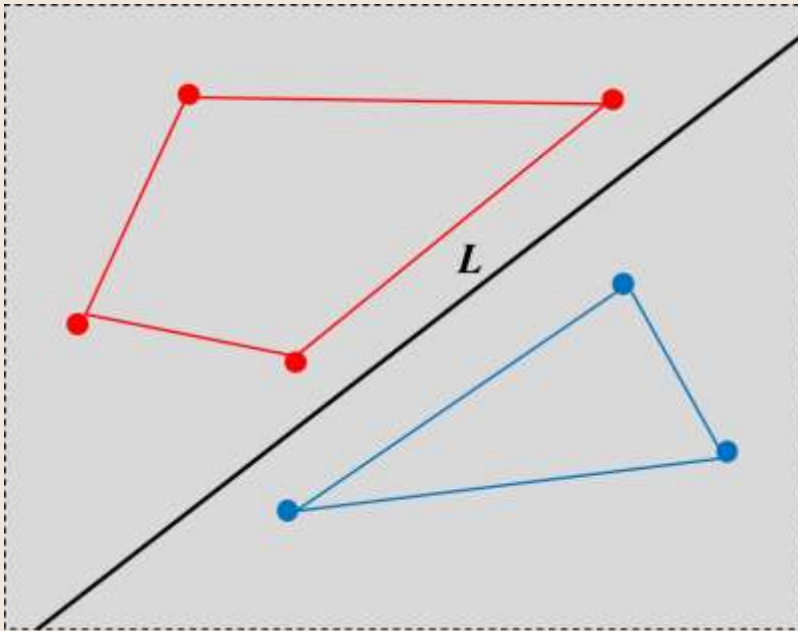


볼록 외피



문제 풀이

- ✓ 빨간색 점들의 볼록 외피와 파란색 점들의 볼록 외피를 구함
- ✓ 두 볼록 다각형이 겹치거나 닿아 있지 않으면 직선 L 이 존재





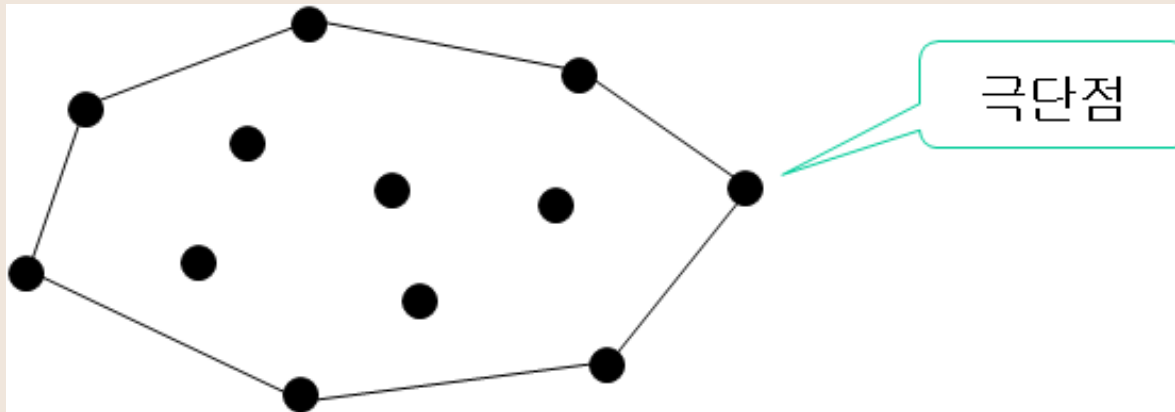
볼록 외피



볼록 외피 알고리즘(Jarvis's march)

✓ 극단점 하나를 찾는 방법은?

→ x좌표 또는 y좌표로 최대 또는 최소인 정점





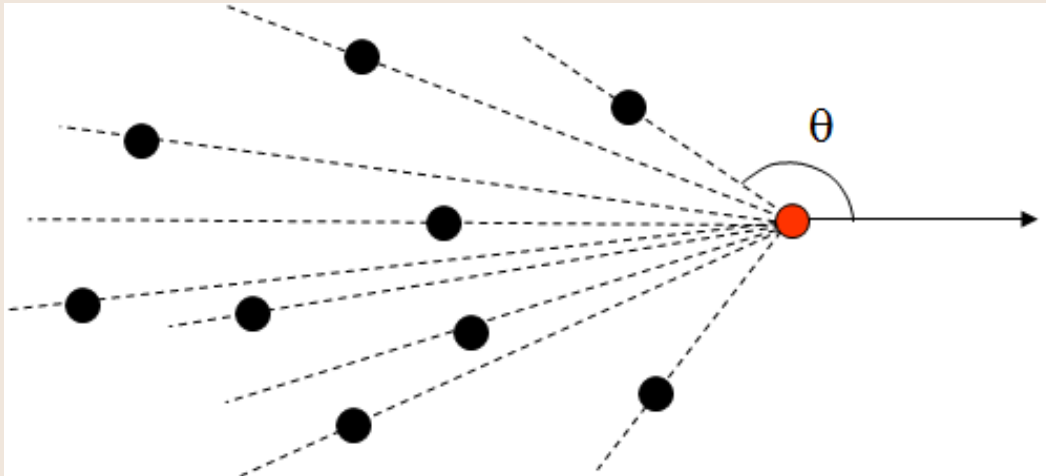
볼록 외피



Jarvis's march 알고리즘

- ✓ 극단점 p 에 연결된 볼록 외피의 한 에지를 찾는 방법은?

→ p 에서 각 정점으로 가는 벡터들 중 가장 왼쪽 또는 오른쪽에 있는 정점이 p 와 에지로 연결된 극단점



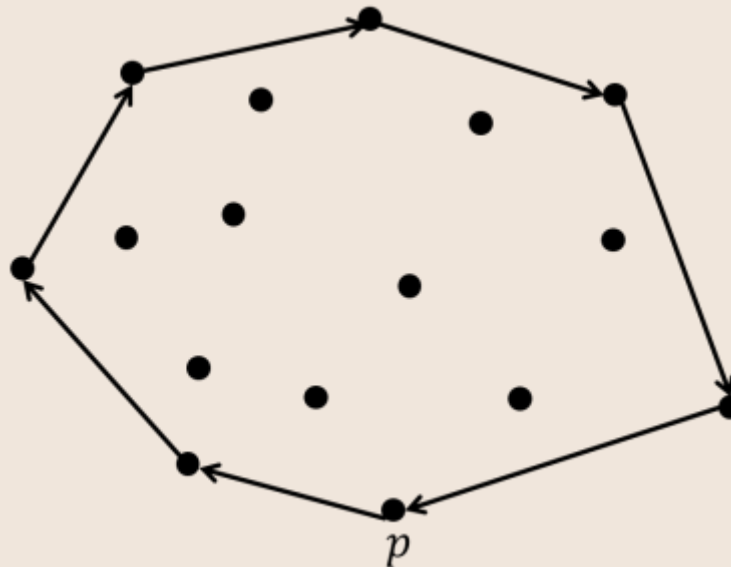


볼록 외피



Jarvis's march 알고리즘

- ✓ 볼록 외피의 한 극단점 p 를 찾는다
- ✓ 시계방향 또는 반시계 방향으로 p 에 인접한 볼록 외피의 한 에지를 찾아서 볼록 외피의 다음 정점을 찾고, 다음 정점에서 이 과정을 반복하면서 볼록 외피의 모든 극단점을 찾는다

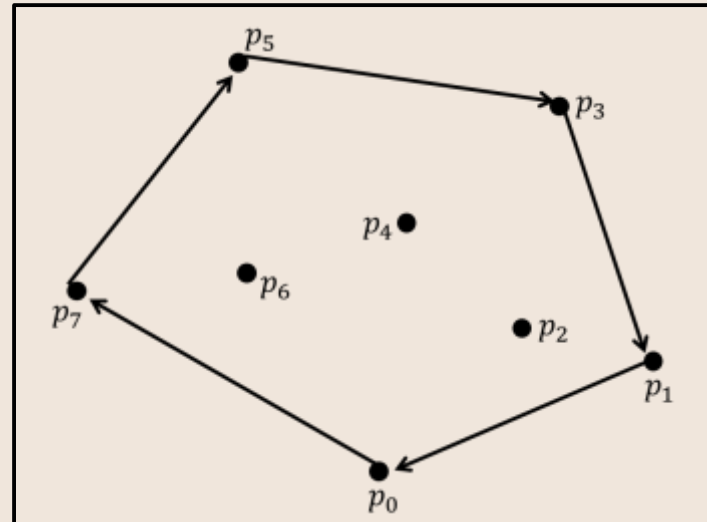
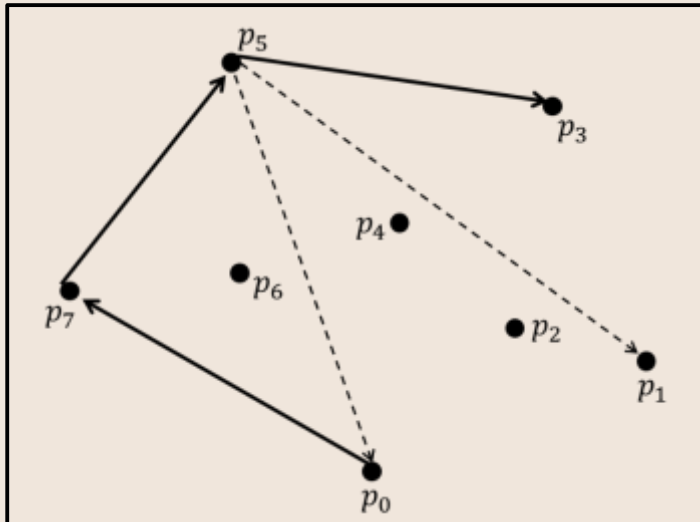
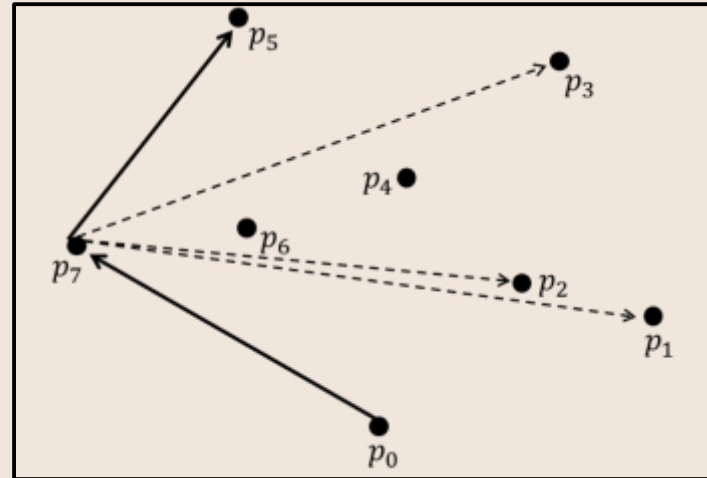
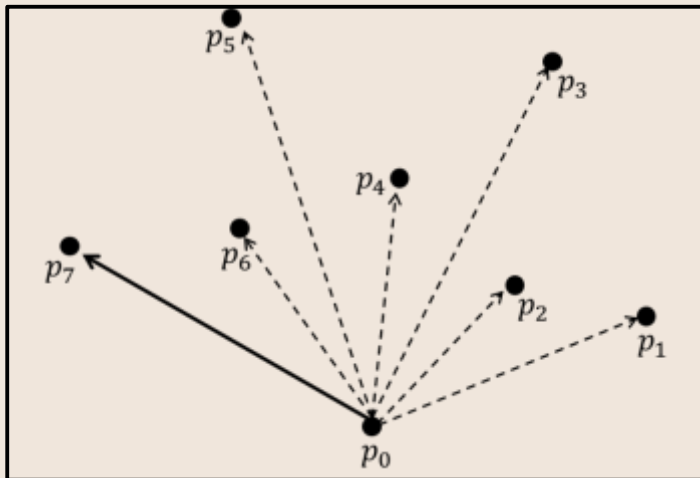




블록 외피



Jarvis's march 알고리즘



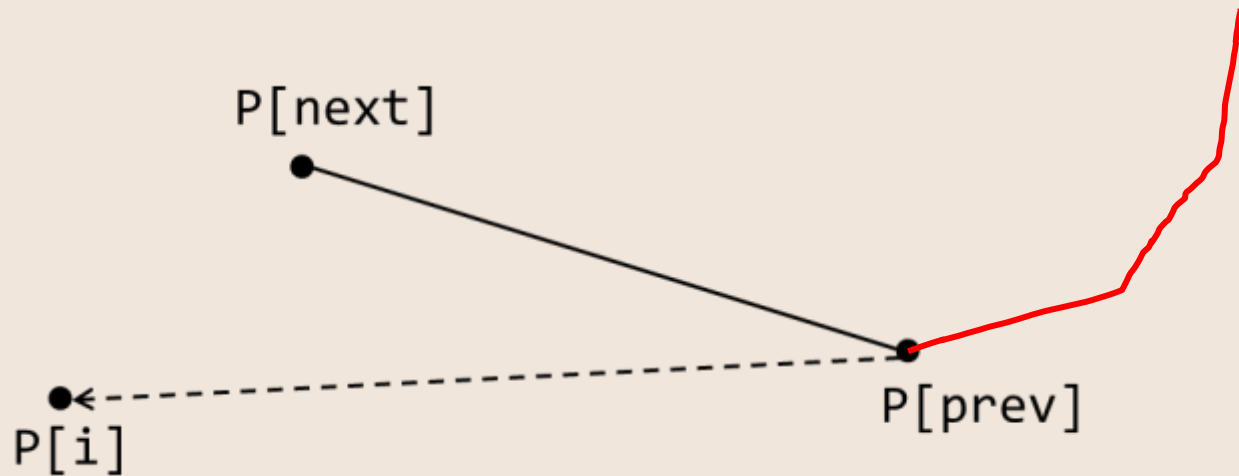


Jarvis's march 알고리즘

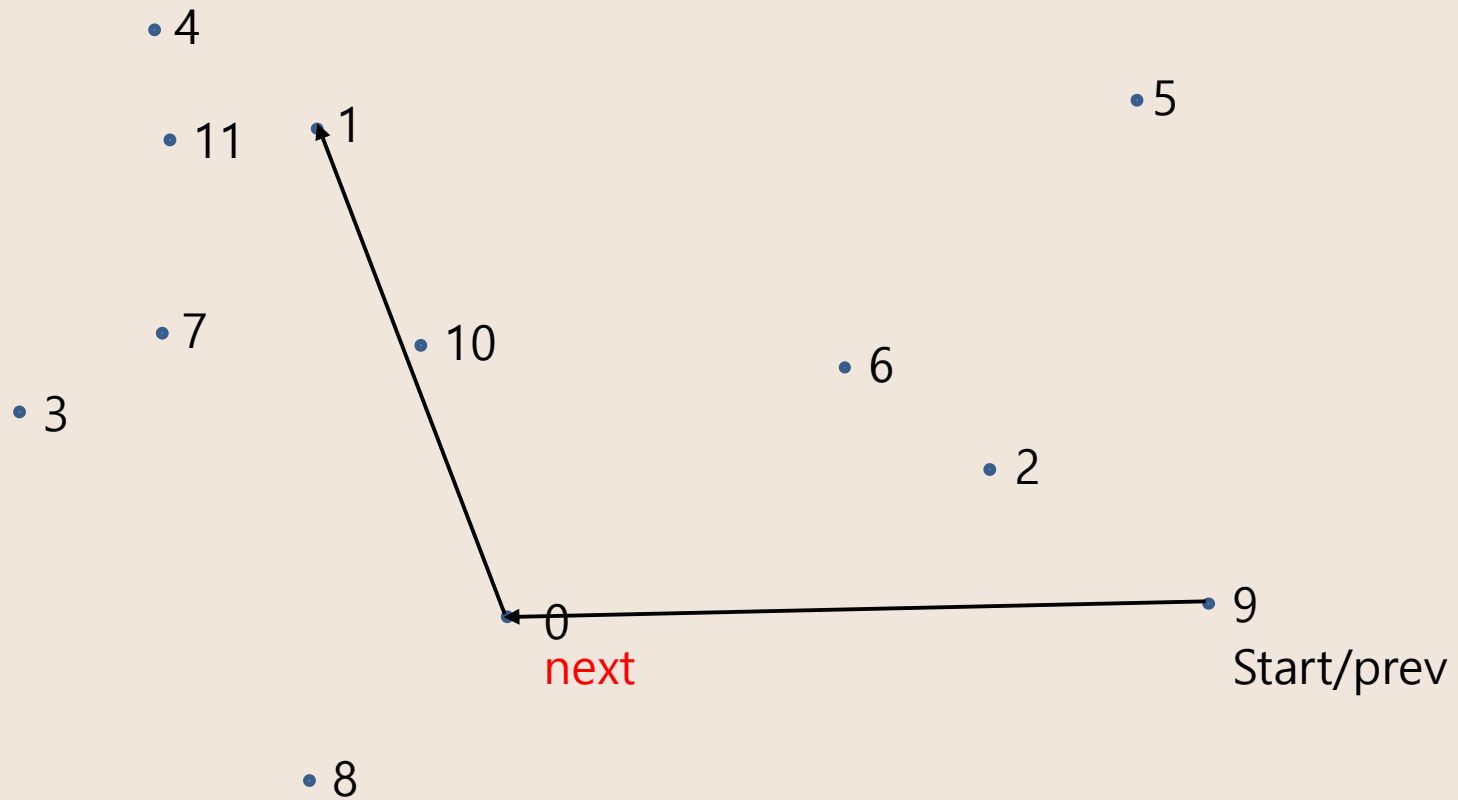
```
convexHull_Javis(Point P[], int n, Polygon C)
{
    start ← getExtreme(P, n)           // 하나의 극단점 탐색
    prev ← start
    next ← -1
    k ← 0
    WHILE next != start
        C[k++] ← P[prev]               // 극단점 저장
        // 다음 극단점 탐색
        next ← 0
        FOR i in 0 → n-1
            IF i != prev && i != next &&
                leftTurn(P[prev], P[next], P[i])
            THEN next ← i
        prev ← next
}
```

Jarvis's march 알고리즘

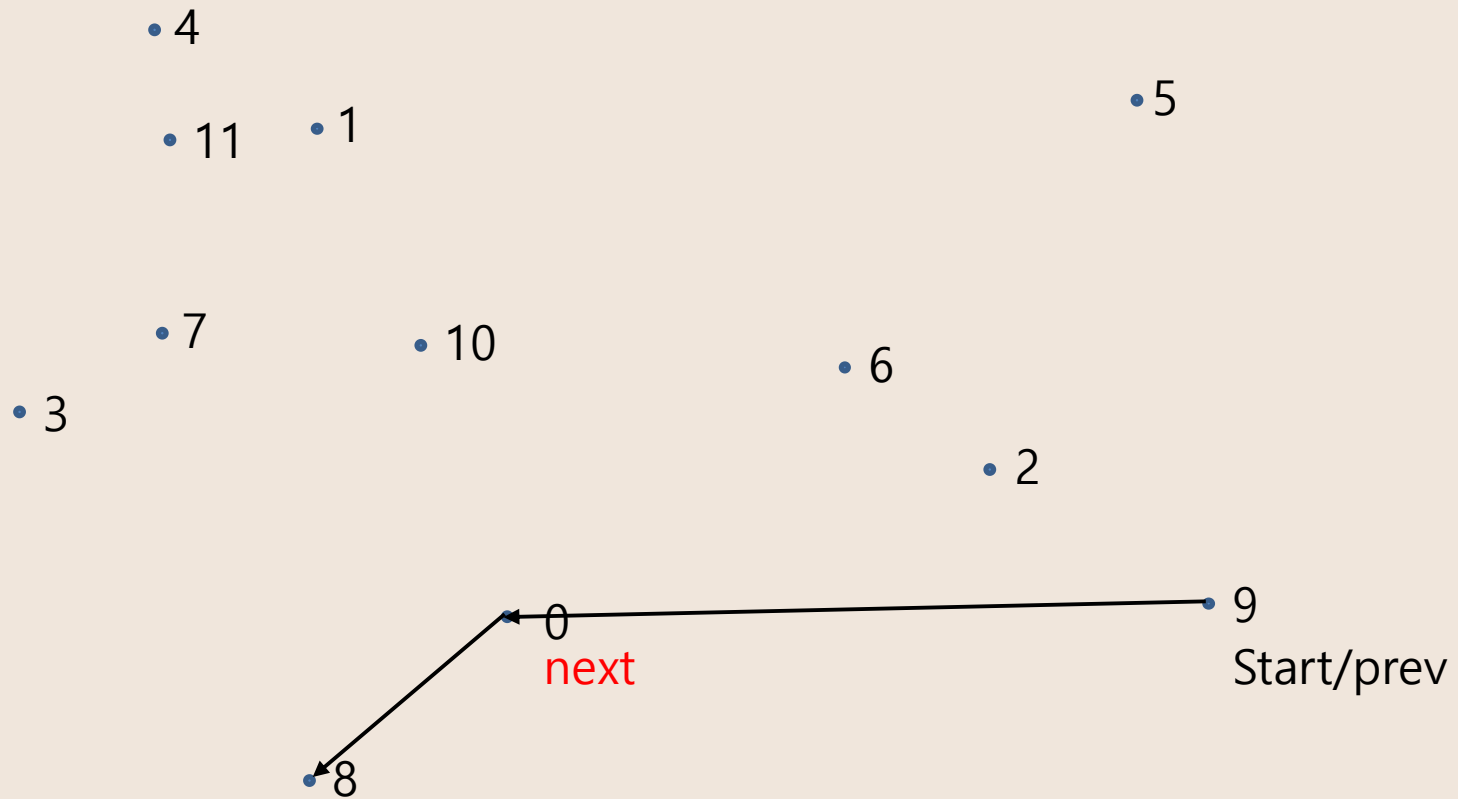
```
FOR i in 0 → n-1
  IF i != prev && i != next &&
    leftTurn(P[prev], P[next], P[i])
  THEN next ← i
```



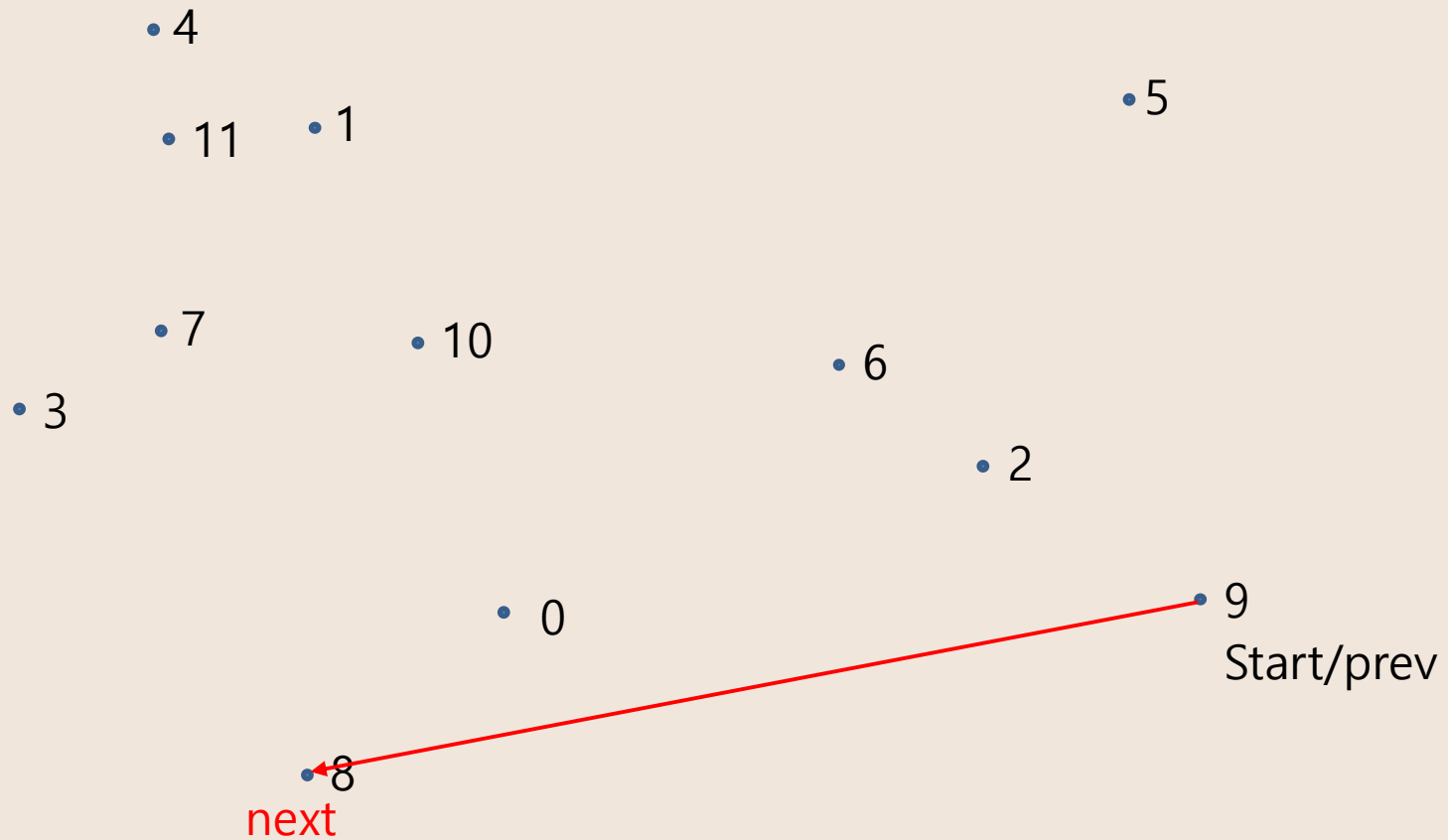
볼록 외피 알고리즘 (Jarvis's march 알고리즘)



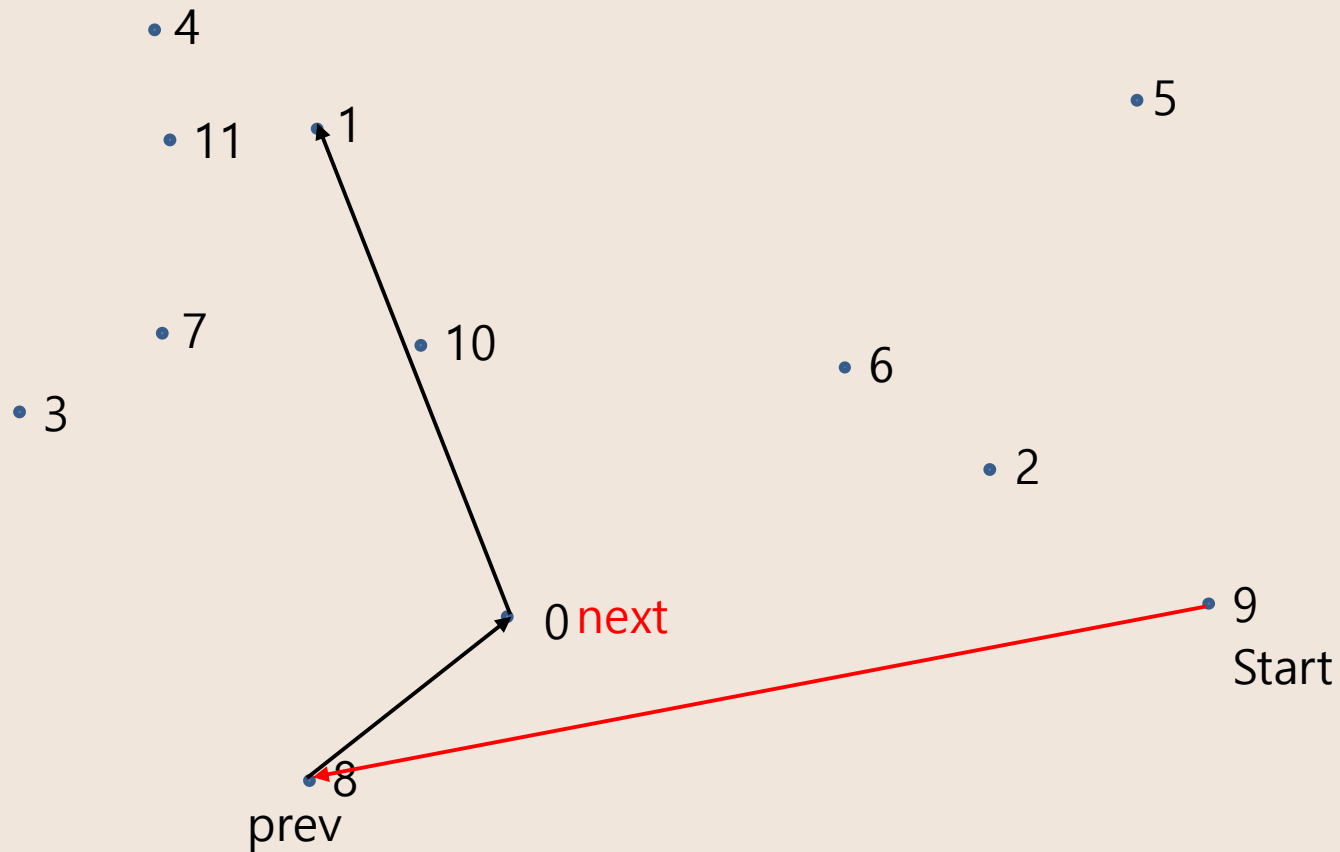
볼록 외피 알고리즘 (Jarvis's march 알고리즘)



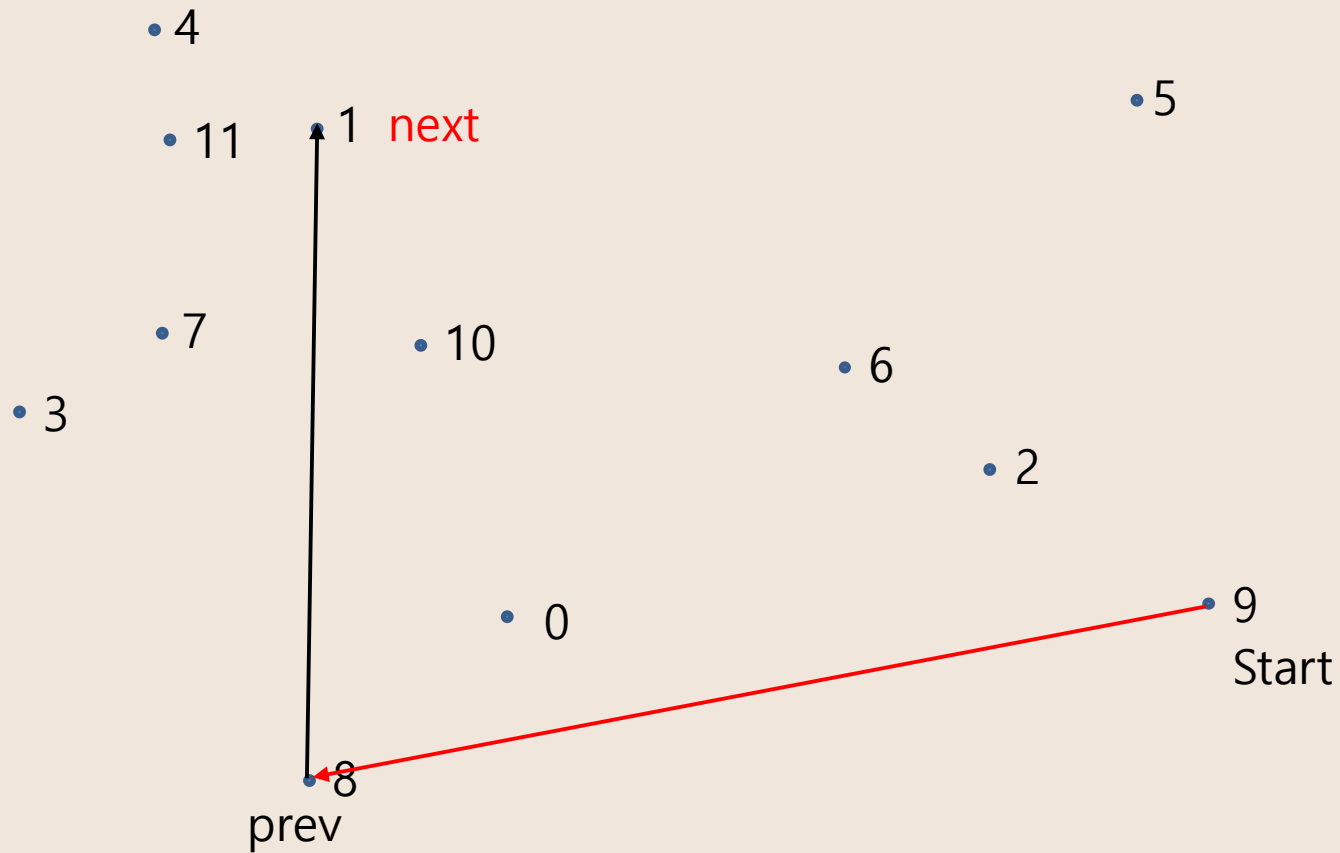
볼록 외피 알고리즘 (Jarvis's march 알고리즘)



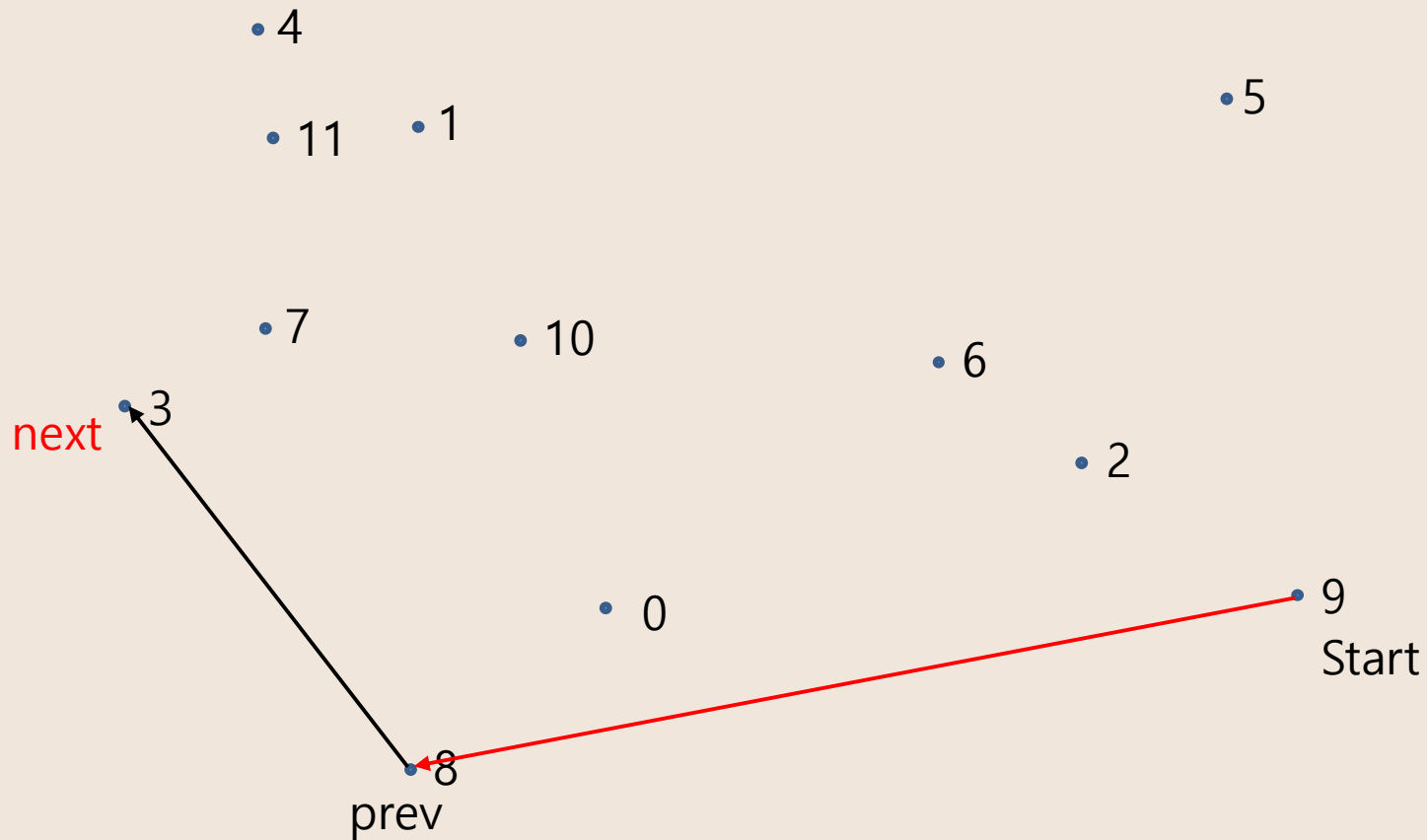
볼록 외피 알고리즘 (Jarvis's march 알고리즘)



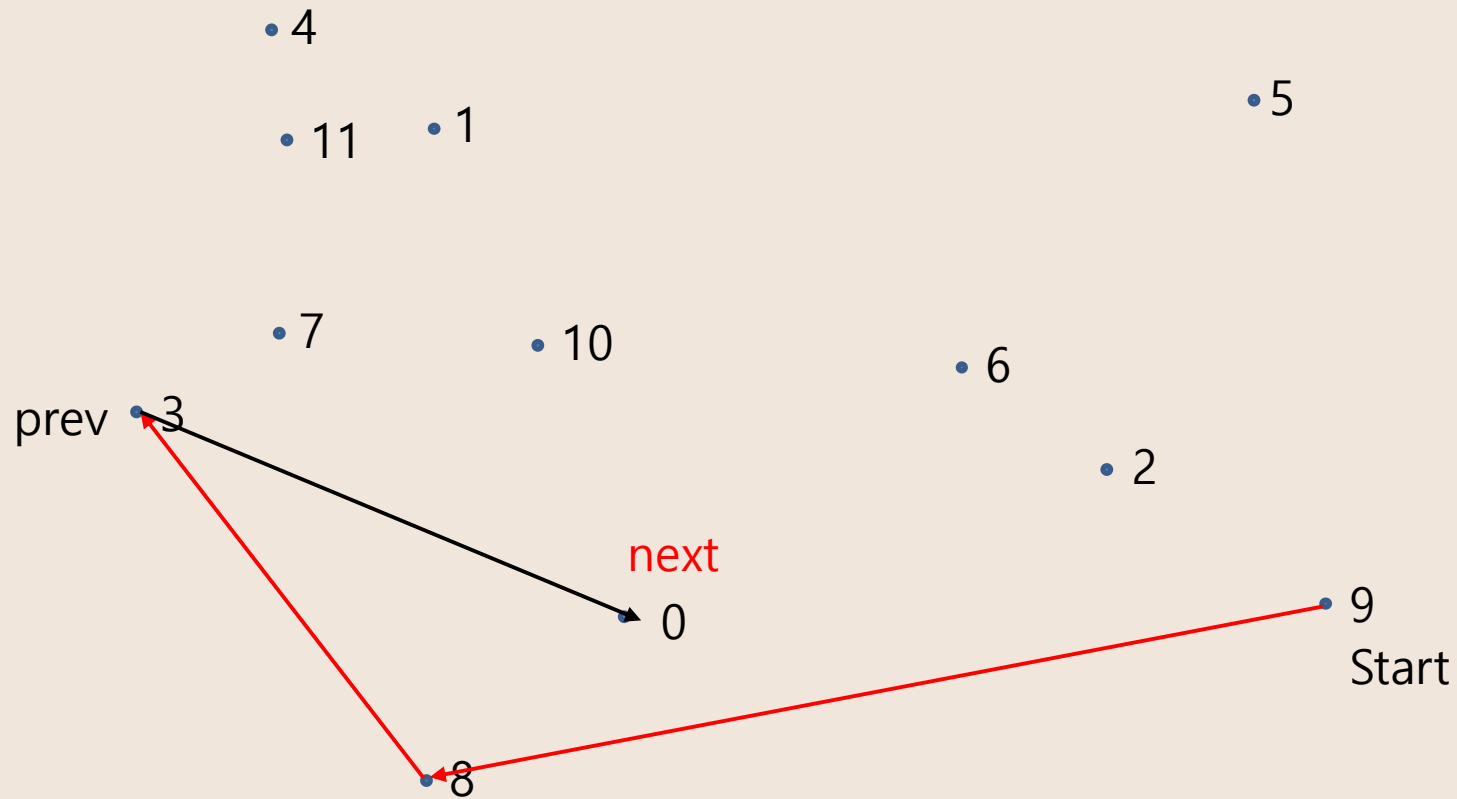
볼록 외피 알고리즘 (Jarvis's march 알고리즘)



볼록 외피 알고리즘 (Jarvis's march 알고리즘)



볼록 외피 알고리즘 (Jarvis's march 알고리즘)



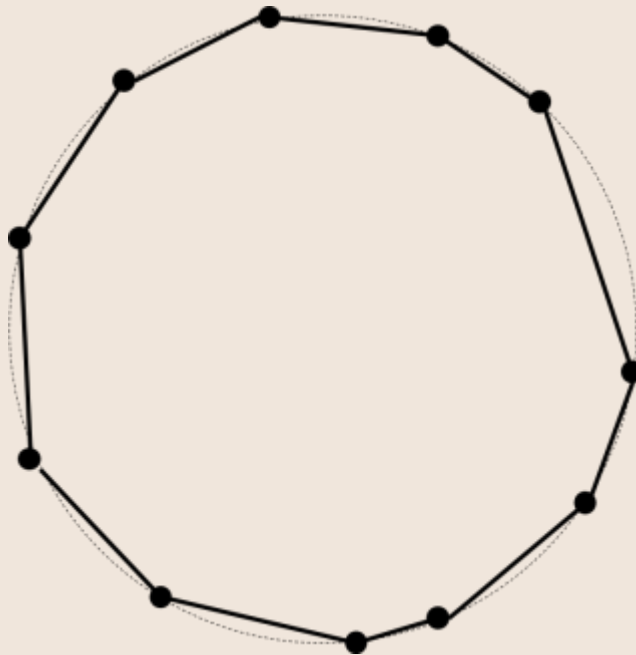


볼록 외피



Jarvis's march 알고리즘의 시간복잡도

- ✓ N개의 점들이 주어 질 때, 최악의 경우에 $O(N^2)$
- ✓ 주어진 모든 점들이 볼록 외피의 극단점이 되는 경우에 각 점에서 인접한 볼록 외피의 에지를 결정하기 위해서 $\Omega(N)$ 시간이 사용될 수 있다



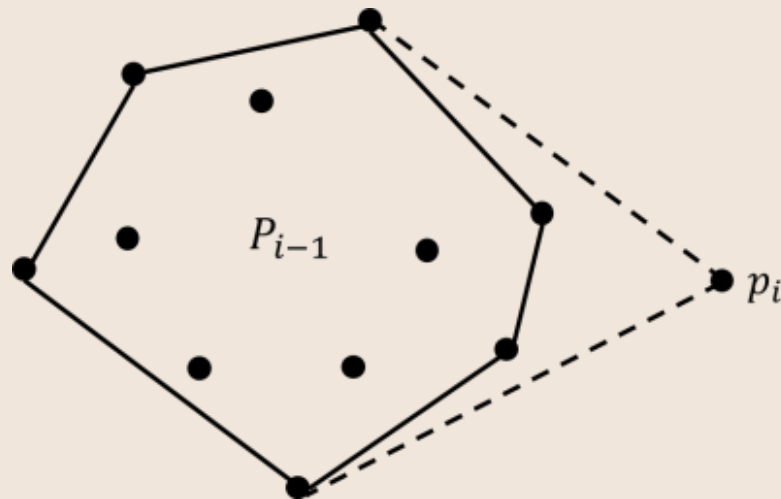


볼록 외피



$O(N \log N)$ 볼록 외피 알고리즘

- ✓ 점들을 x 좌표에 대해서 정렬한다
- ✓ 정렬된 점들을 p_0, p_1, \dots, p_{N-1} 라고 하자
- ✓ 먼저 p_0, p_1, p_2 를 꼭지점으로 하는 삼각형을 P_2 라고 하면, P_2 는 세 점 p_0, p_1, p_2 의 볼록 외피이다
- ✓ 점들 p_0, p_1, \dots, p_{i-1} 의 볼록 외피를 P_{i-1} 이라고 할 때, 점 p_i 를 지나는 P_{i-1} 의 두 접선을 찾아서 p_0, p_1, \dots, p_i 의 볼록 외피를 P_i 를 구한다

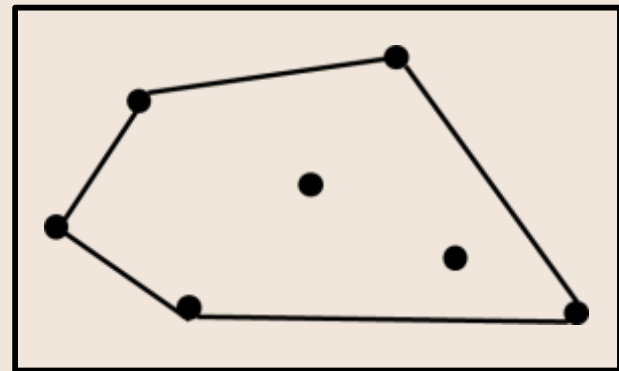
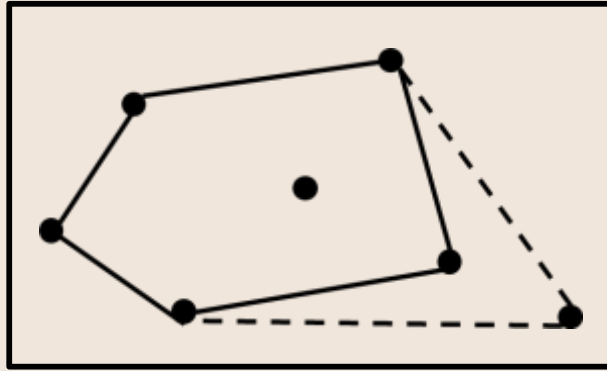
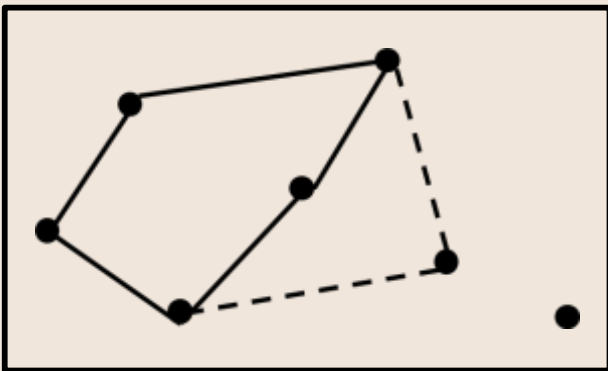
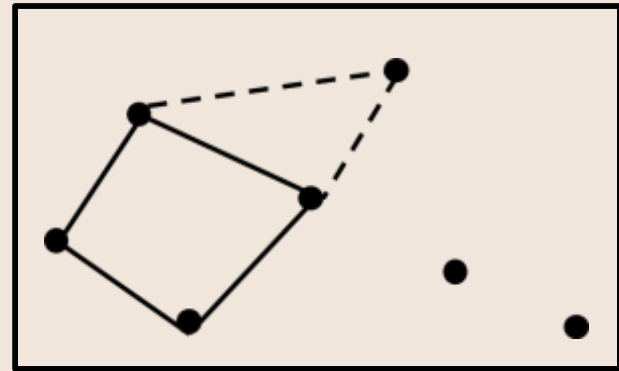
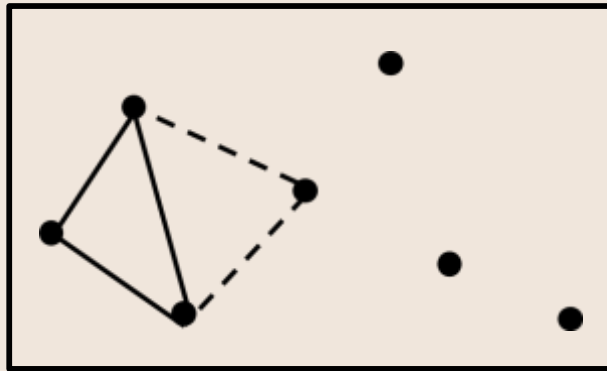
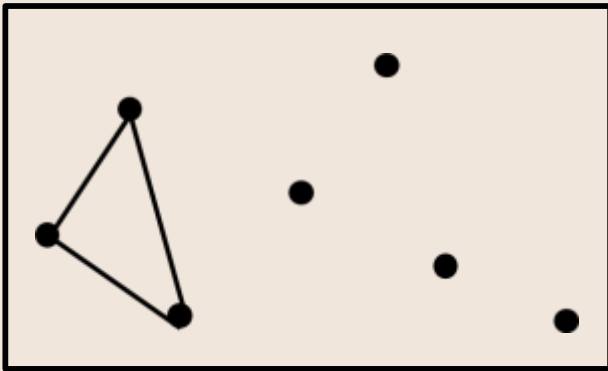




볼록 외피



$O(N \log N)$ 볼록 외피 알고리즘



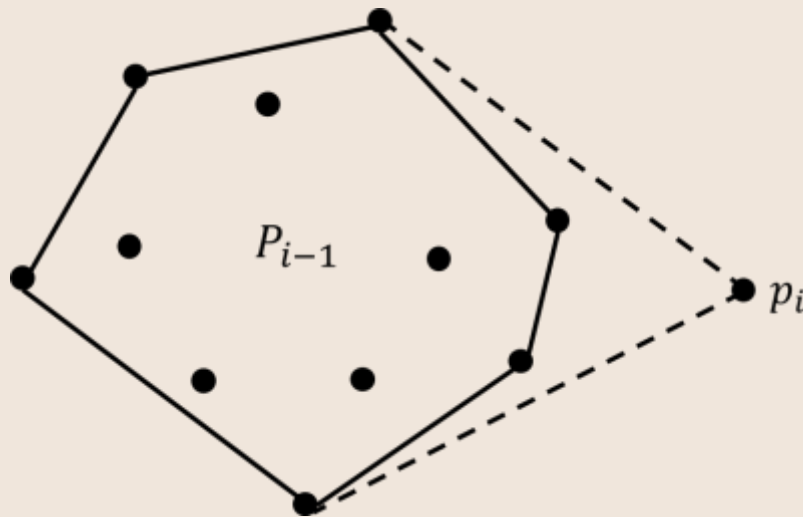


볼록 외피



$O(N \log N)$ 볼록 외피 알고리즘

- ✓ 점들 p_0, p_1, \dots, p_{i-1} 의 볼록 외피를 P_{i-1} 이라고 할 때, 점 p_i 를 지나는 P_{i-1} 의 두 접선을 어떻게 찾을 수 있는가?
- ✓ 접선이 통과하는 두 접점을 찾으시오.



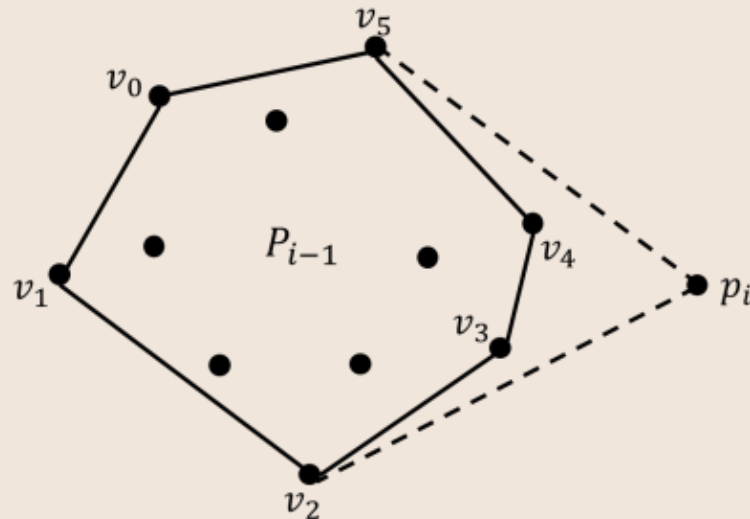


볼록 외피



$O(N \log N)$ 볼록 외피 알고리즘

- ✓ 볼록 외피를 P_{i-1} 의 꼭지점들을 반시계 방향으로 v_0, v_1, \dots, v_h 라고 할 때, p_i, v_i, v_{i+1} 은 좌회전 또는 우회전한다
- ✓ 접점 v_k 는 p_i, v_i, v_{i+1} 의 방향이 바뀌는 꼭지점
- ✓ p_i, v_{k-1}, v_k : 좌회전 $\rightarrow p_i, v_k, v_{k+1}$: 우회전 또는
 p_i, v_{k-1}, v_k : 우회전 $\rightarrow p_i, v_k, v_{k+1}$: 좌회전



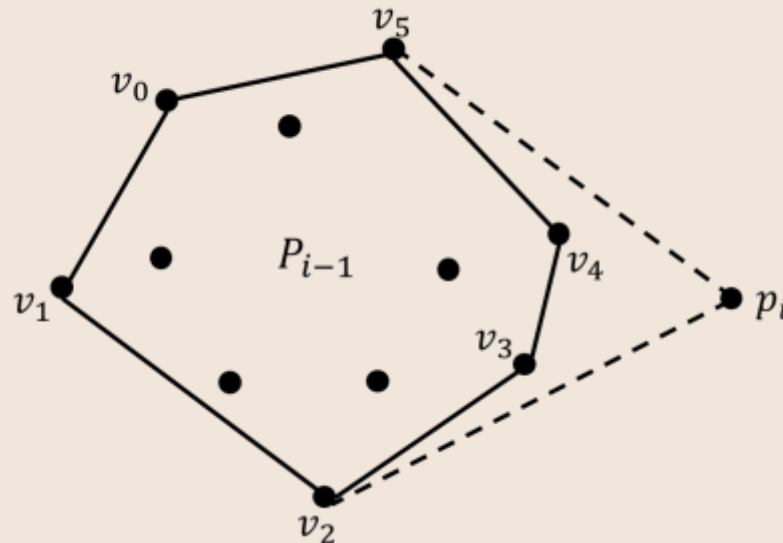


볼록 외피



$O(N \log N)$ 볼록 외피 알고리즘

- 아래 그림에서 p_i, v_1, v_2 : 좌회전 $\rightarrow p_i, v_2, v_3$: 우회전이고
 p_i, v_4, v_5 : 우회전 $\rightarrow p_i, v_5, v_0$: 좌회전이므로 v_2 와 v_5 이 접점

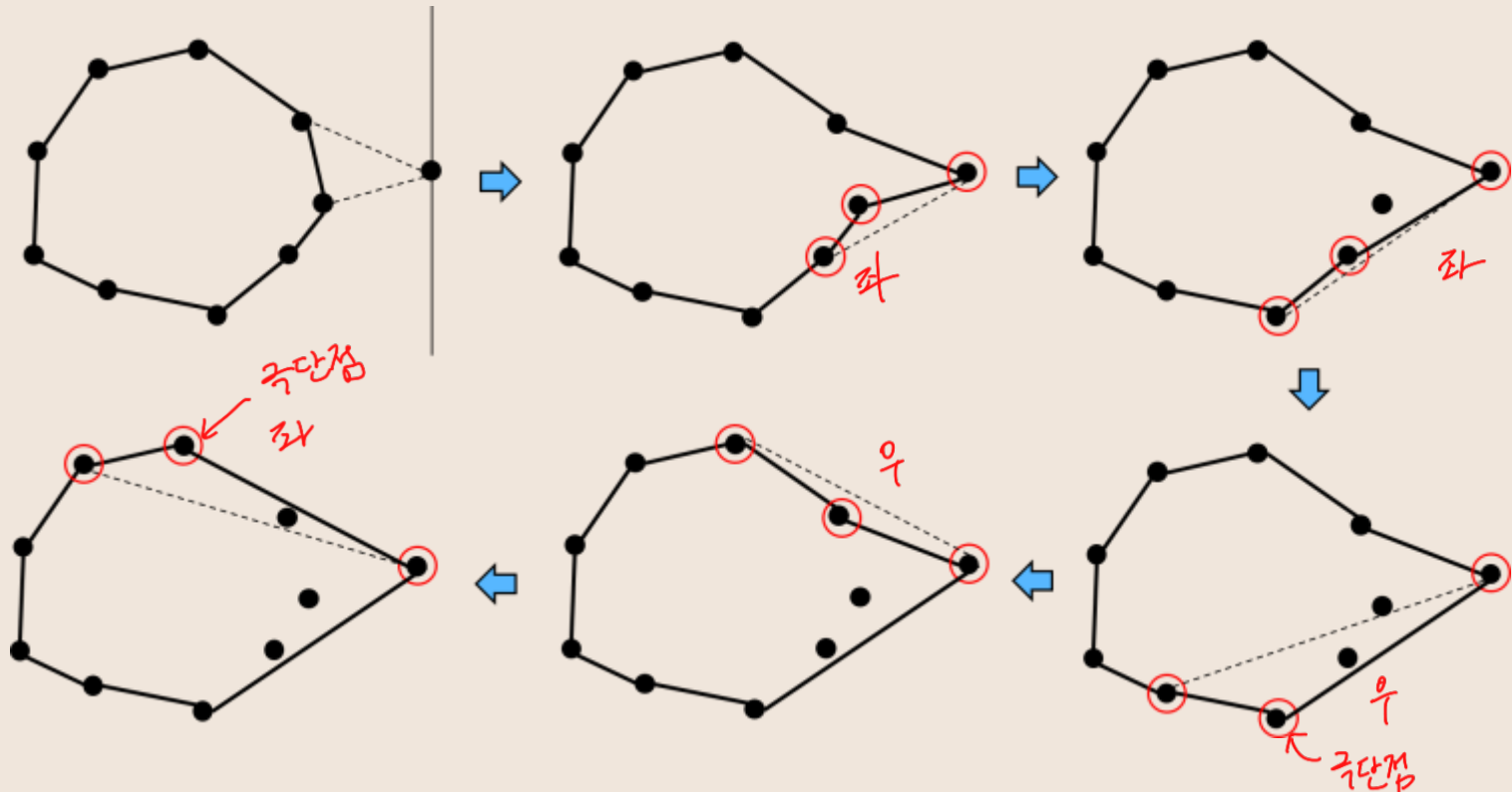




볼록 외피

$O(N \log N)$ 볼록 외피 알고리즘

- ✓ 바로 이전 고려했던 점 p_{i-1} 은 볼록 외피를 p_{i-1} 의 한 극단점이다
- ✓ 점 p_{i-1} 에서 시계방향과 반시계방향으로 인접한 세 극단점을 고려해서 오목 정점을 제거한다



$O(N \log N)$ 볼록 외피 알고리즘

```
convexHull(Point P[], int n)
{
    // Represent the convex hull as a circular list C[] of vertices
    // in counterclockwise order
    // next[i] is index of the next vertex of i in C[] counterclockwise
    // prev[i] is index of the next vertex of i in C[] clockwise

    Sort the points of P[] in the x-coordinates
    next[0] ← 1; prev[1] ← 0
    next[1] ← 0; prev[0] ← 1
}
```

$O(N \log N)$ 볼록 외피 알고리즘

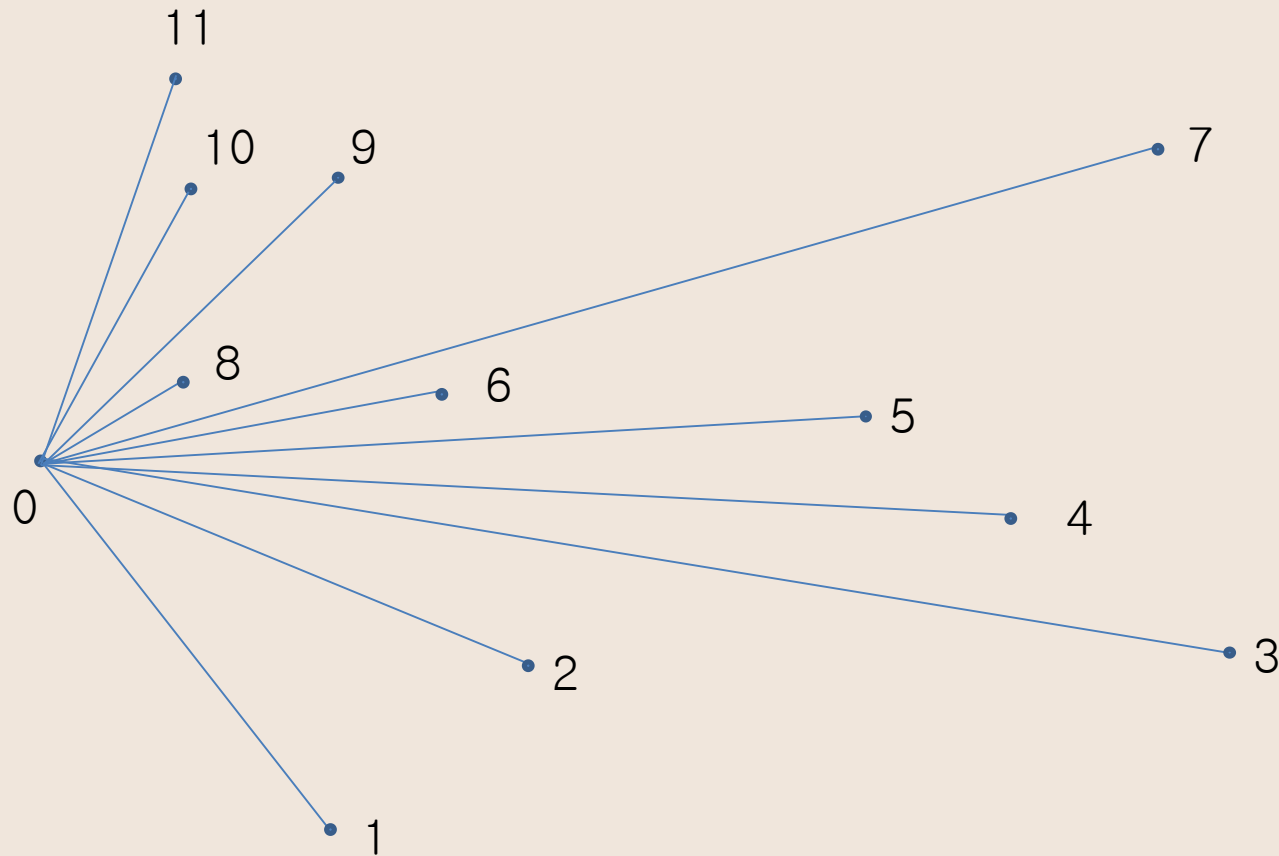
```
FOR i in 2 → n-1
    IF P[i].y > P[i-1].y
        prev[i] ← i-1; next[i] ← next[i-1]
    ELSE
        next[i] ← i-1; prev[i] ← prev[i-1]
    next[prev[i]] ← i; prev[next[i]] ← i
    WHILE leftTurn(i, prev[i], prev[prev[i]])
        next[prev[prev[i]]] ← i
        prev[i] ← prev[prev[i]]
    WHILE rightTurn(i, next[i], next[next[i]])
        prev[next[next[i]]] ← i
        next[i] ← next[next[i]]
}
```

볼록 외피 알고리즘 (Graham's Algorithm)

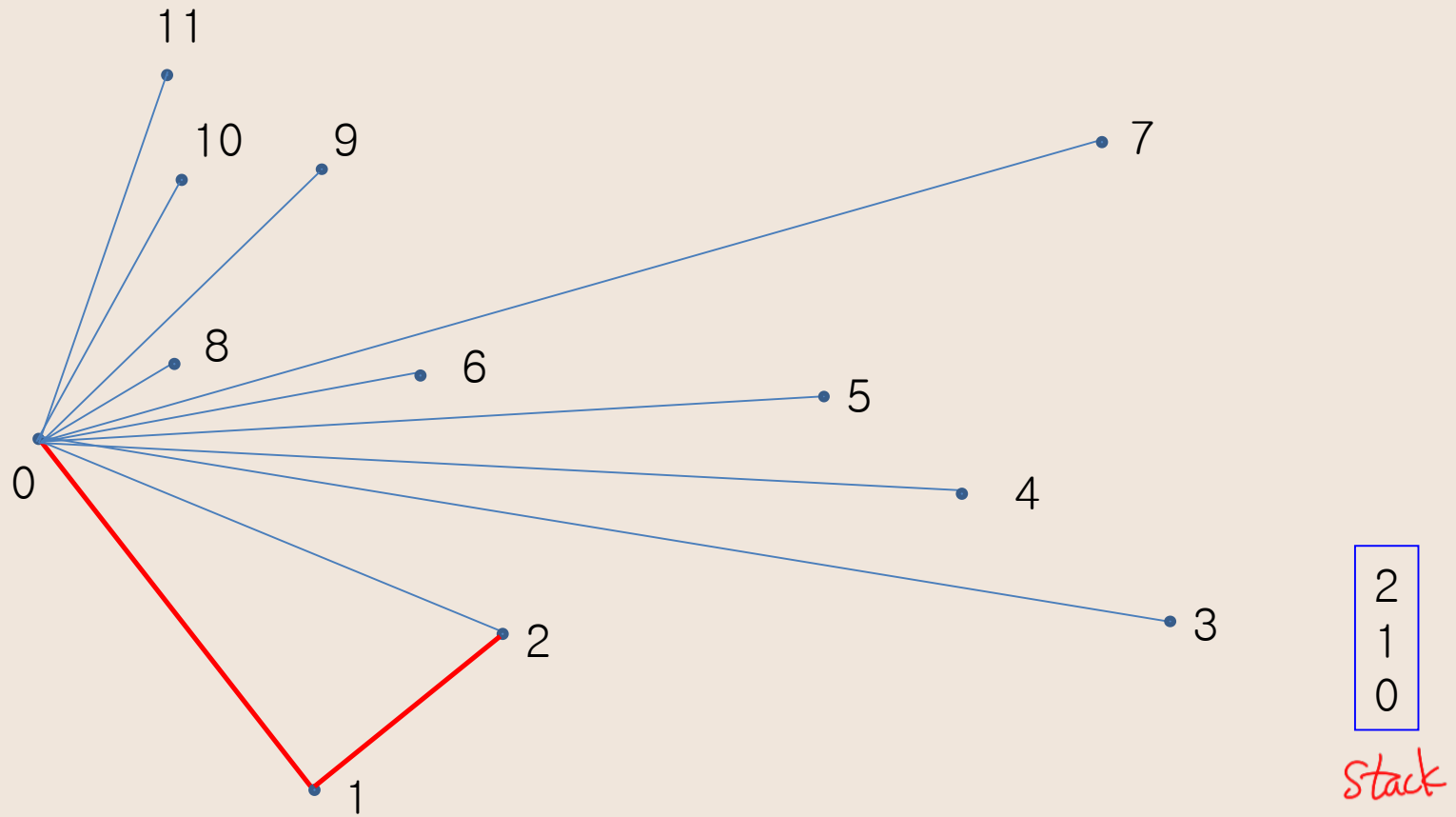


볼록 외피 알고리즘 (Graham's Algorithm)

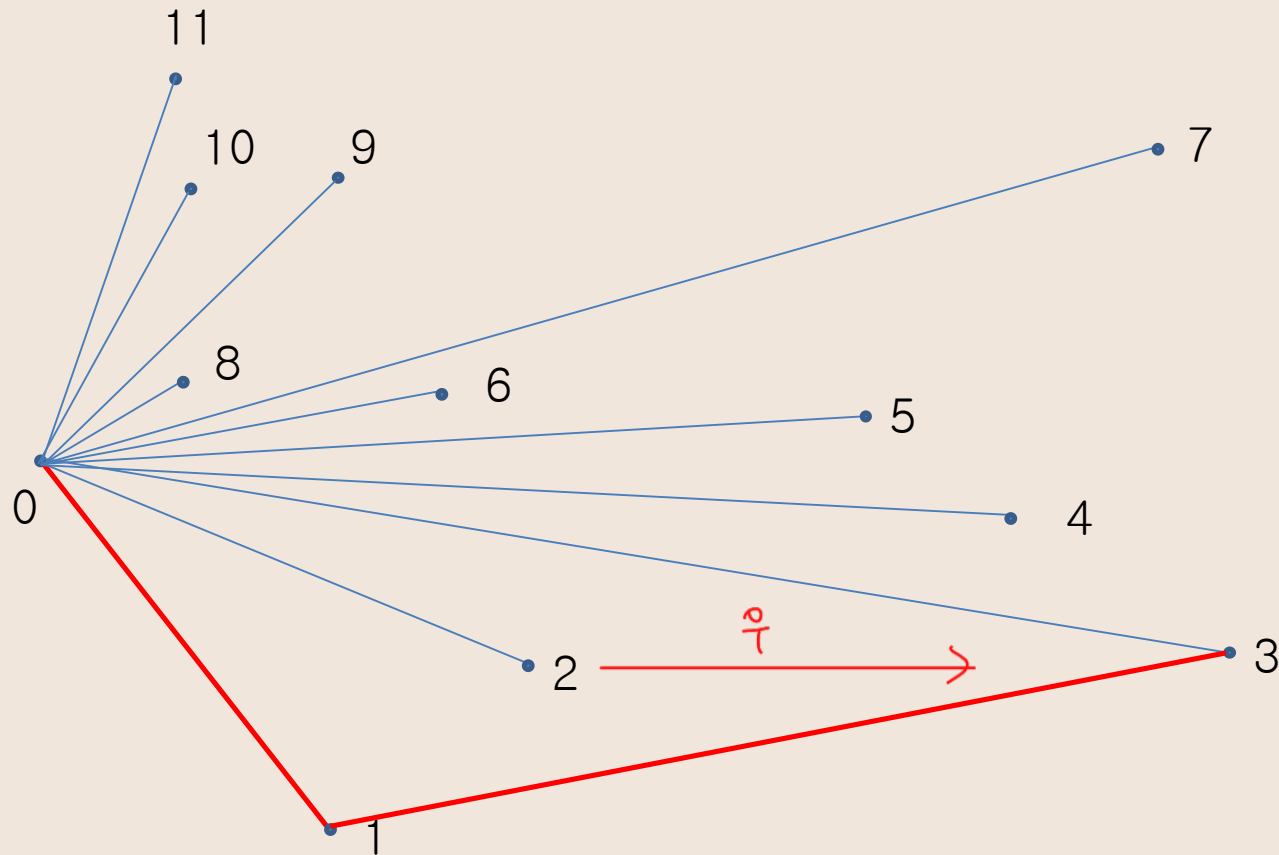
direction 함수로
각에 따라 Sorting



볼록 외피 알고리즘 (Graham's Algorithm)



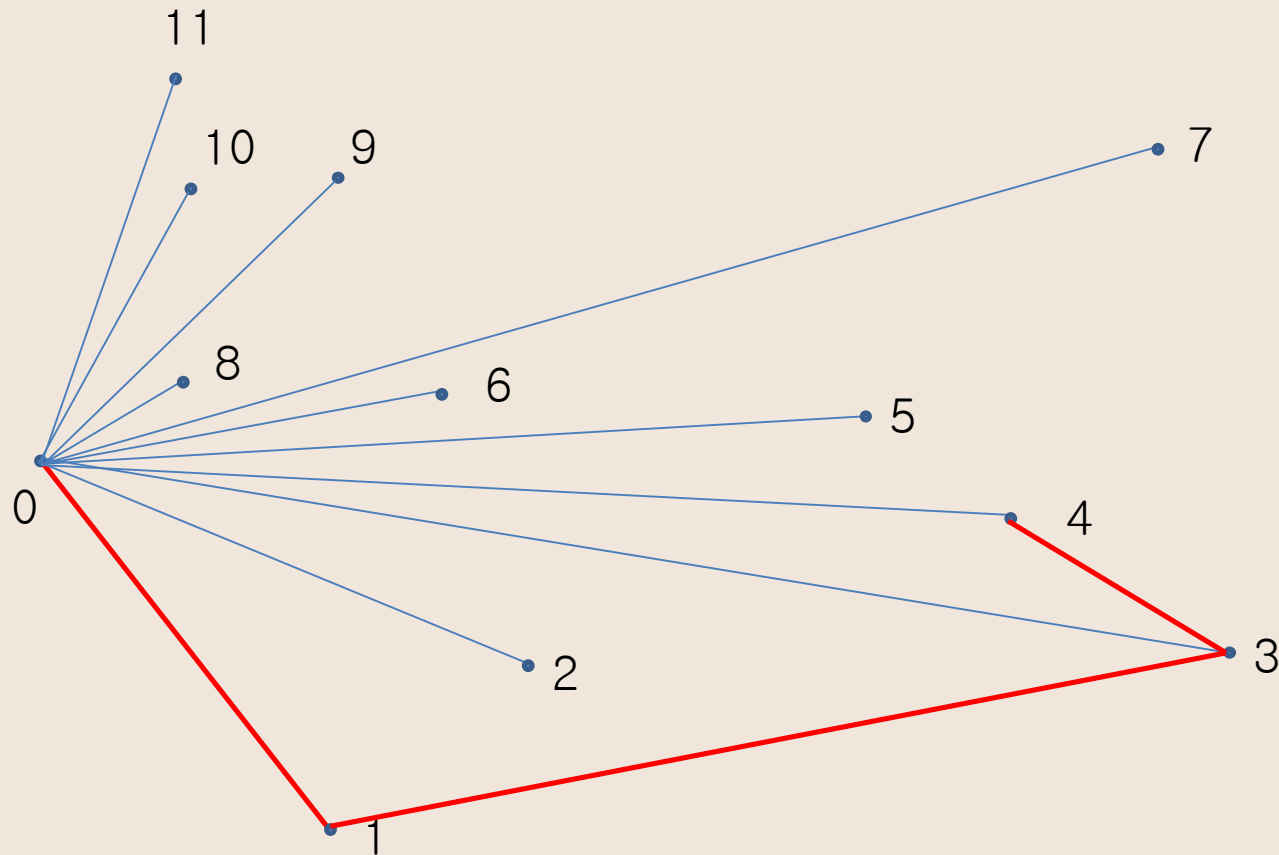
볼록 외피 알고리즘 (Graham's Algorithm)



3
1
0

↻라

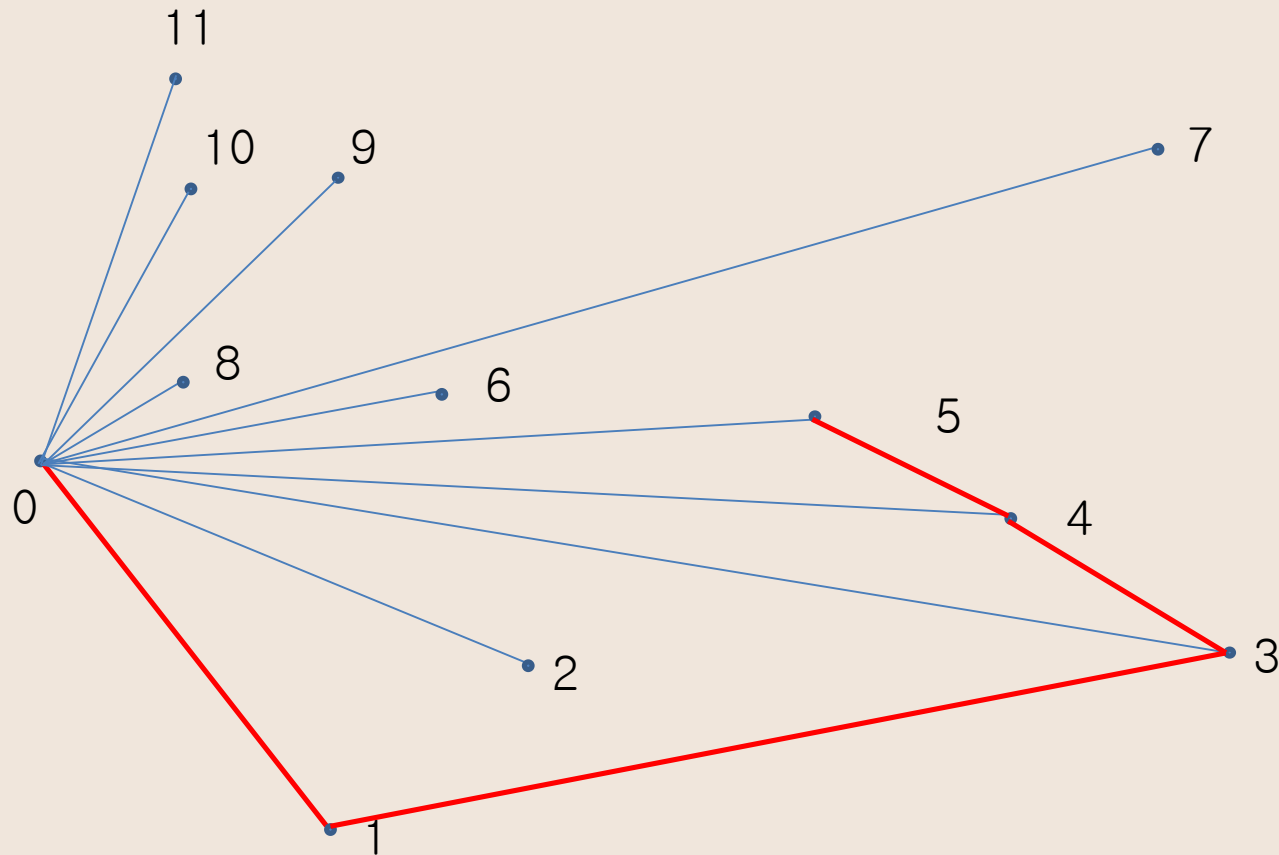
볼록 외피 알고리즘 (Graham's Algorithm)



4
3
1
0

} 34

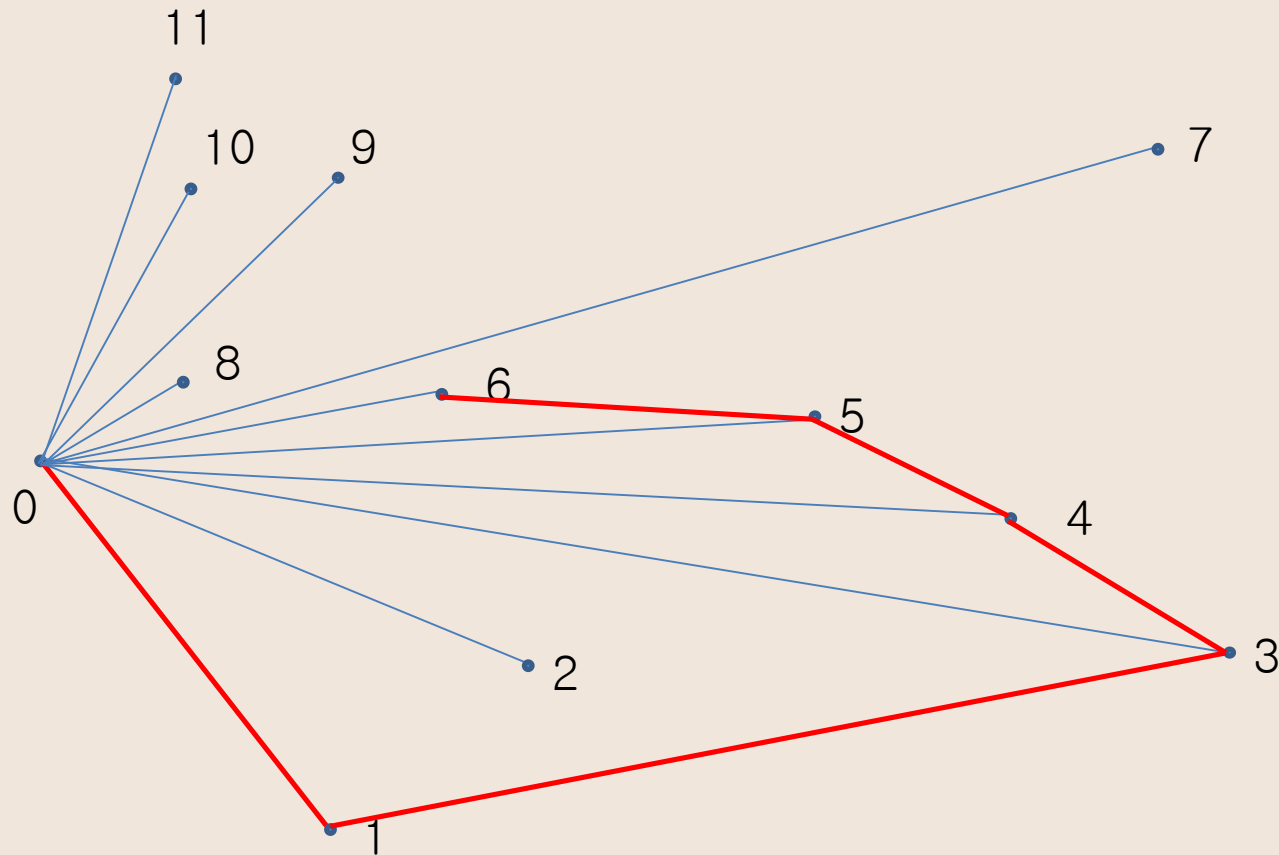
볼록 외피 알고리즘 (Graham's Algorithm)



5
4
3
1
0

} 24

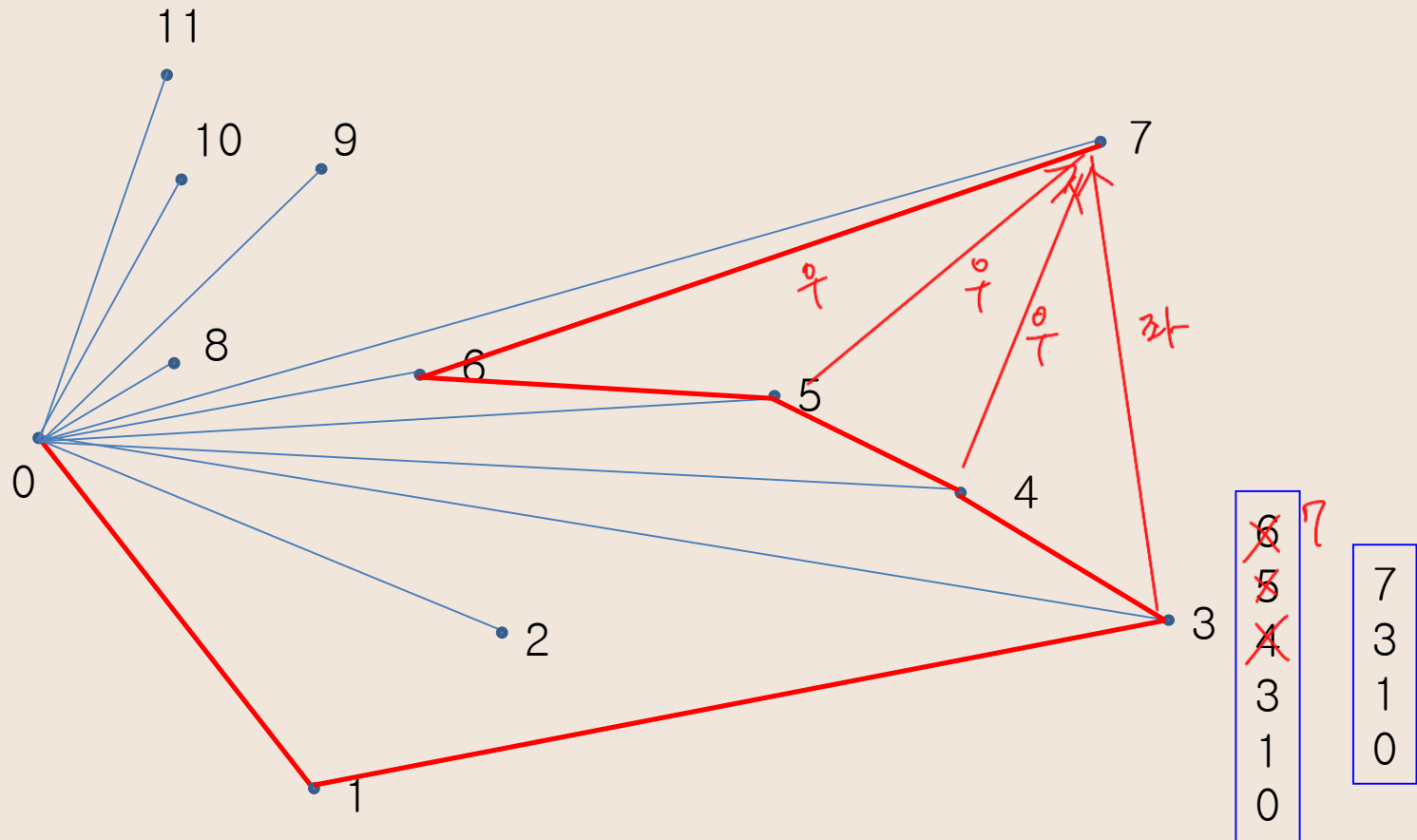
볼록 외피 알고리즘 (Graham's Algorithm)



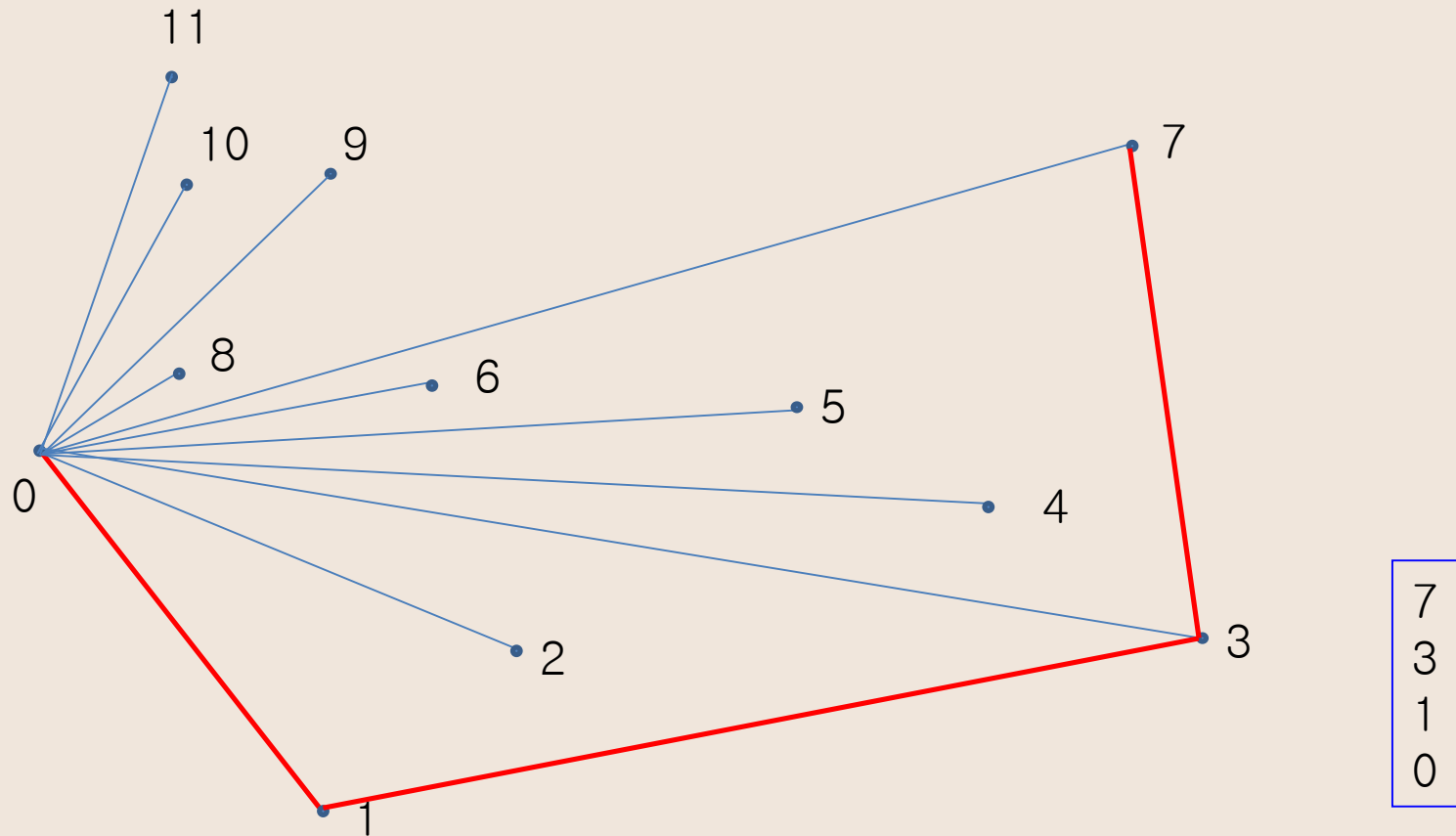
6
5
4
3
1
0

ok


볼록 외피 알고리즘 (Graham's Algorithm)



볼록 외피 알고리즘 (Graham's Algorithm)



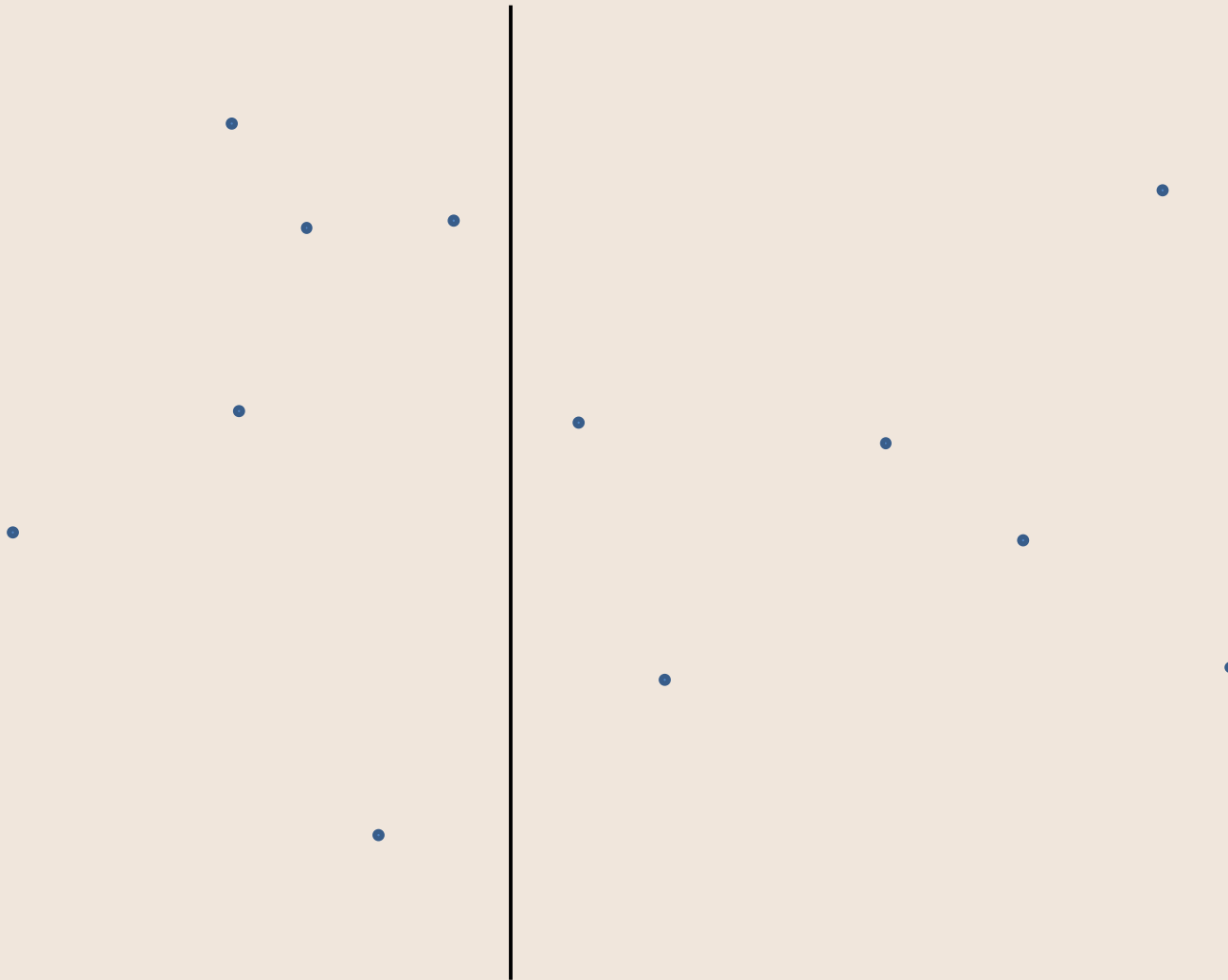
볼록 외피 알고리즘 (Graham's Algorithm)

 이 알고리즘의 시간복잡도는?

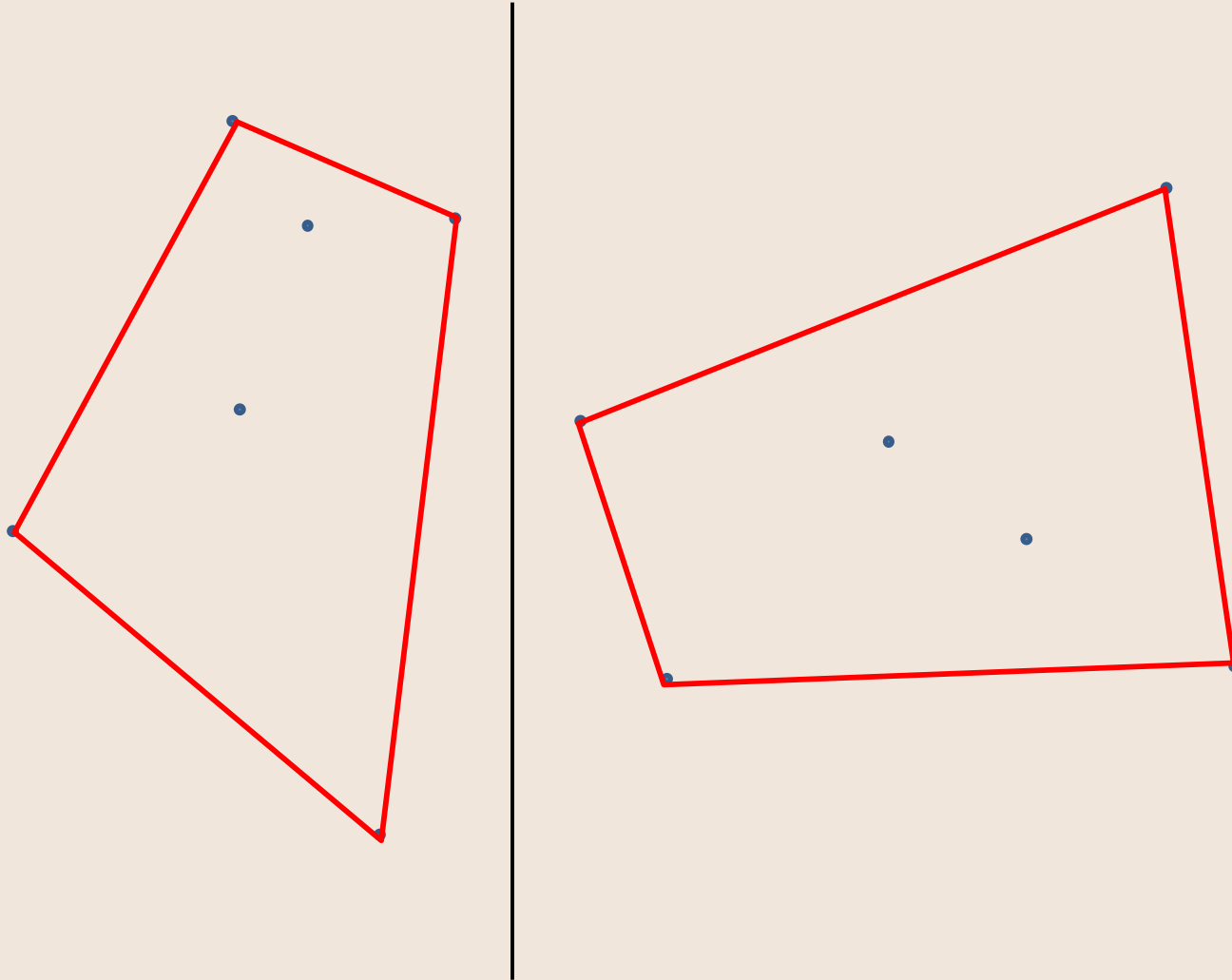
- ✓ 좌측 극단점 찾기: $O(N)$
- ✓ 극단점을 중심으로 부채살 모양을 갖는 점들의 순서 구하기: $O(N \lg N)$
- ✓ 부채 모양을 반시계 방향으로 돌면서 볼록외피 극단점 찾기: $O(N)$

- ✎ 볼록 외피 알고리즘 (분할정복법)
- ✎ N 개의 점을 x 좌표에 따라 정렬 한 후, 2등분
- ✎ 각 분할에 대해 recursive 하게 볼록외피를 구함
- ✎ 두 볼록외피를 통합

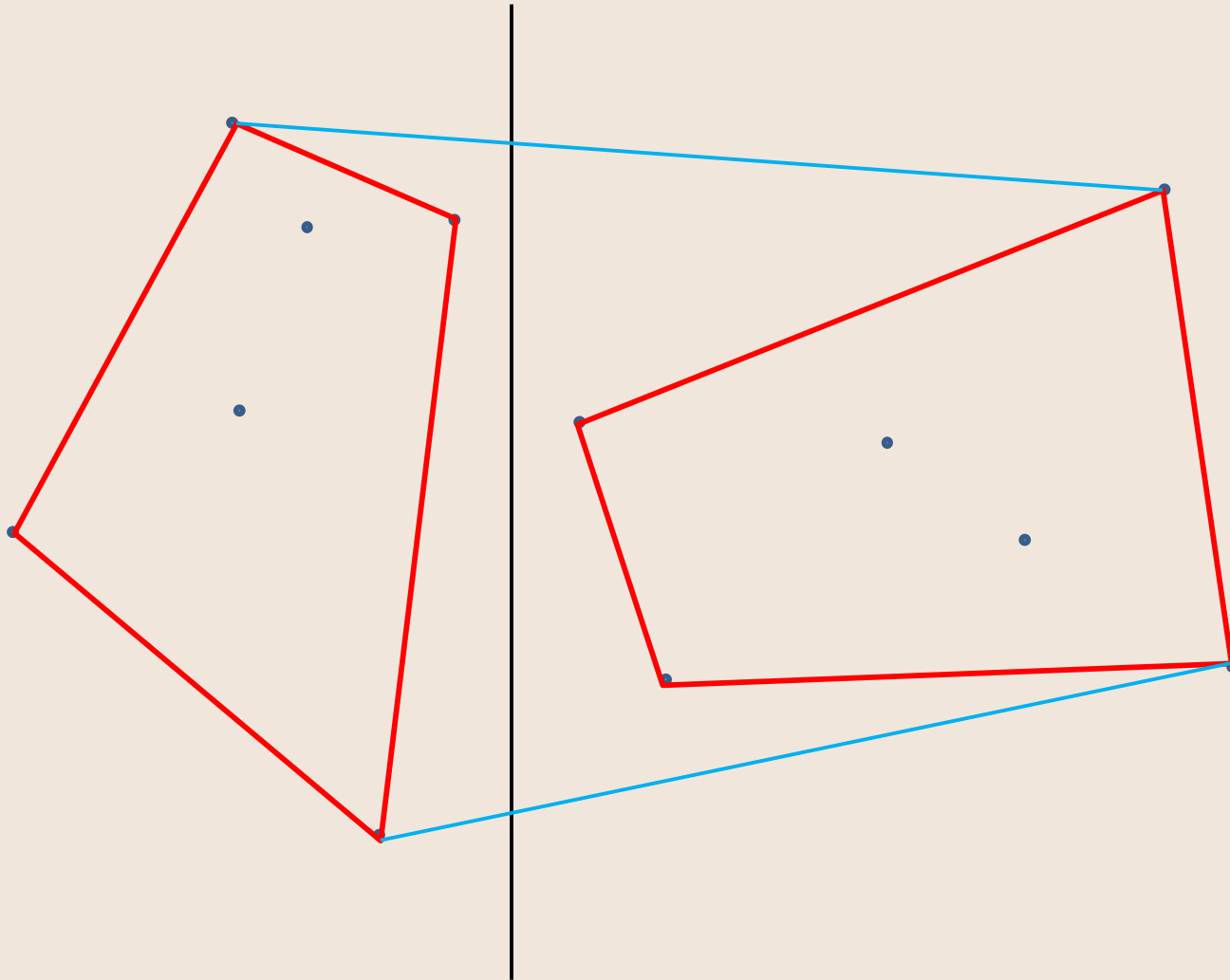
볼록 외피 알고리즘 (분할정복법)




볼록 외피 알고리즘 (분할정복법)





볼록 외피 알고리즘 (분할정복법)




볼록 외피 알고리즘 (분할정복법)

 $T(N)$: 분할정복법으로 N 개의 점에 대한 볼록외피 구하는 시간

 N 개의 점을 x 좌표에 따라 정렬 한 후, 2등분 $\rightarrow O(N \lg N)$


 각 분할에 대해 볼록외피 구하는 시간: $2T\left(\frac{N}{2}\right)$

 두 볼록외피를 통합 $\rightarrow O(N)$


 전체적인 시간 복잡도 $\rightarrow T(N) = 2T\left(\frac{N}{2}\right) + O(N) \rightarrow T(N) = O(N \lg N)$



최근접쌍(Closest pair of points)

 최근접쌍 문제

- ✓ 평면상에 N개의 점들이 주어 질 때, 가장 가까운 두 점을 찾으시오

 점들의 거리(Euclidean distance)

- ✓ 두 점 $p=(x_1, y_1)$ 과 $q=(x_2, y_2)$ 사이의 거리 $d(p, q)$

$$d(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

 Brute-force 알고리즘

- ✓ 모든 두 점 쌍들, 다시 말해서, $\binom{n}{2} = O(n^2)$ 개의 쌍들에 대해서 거리를 구해서 최소값을 찾는다



최근접쌍



분할 정복 알고리즘

✓ 분할 과정

- ✧ N 개 점들의 집합 S 를 각각 $N/2$ 개의 점들의 집합 S_1 과 S_2 로 나눈다
- ✧ 점들의 x 좌표의 중간값(median)을 이용해서 S_1 과 S_2 로 나눌 수 있다

✓ 정복 과정

- ✧ 집합 S_1 과 S_2 에 재귀적으로 최근접쌍 문제를 푼다
- ✧ δ_1 과 δ_2 를 각각 S_1 과 S_2 의 최근접쌍의 거리

✓ 통합 과정

- ✧ S_1 의 한 점과 S_2 의 한 점 사이의 거리의 최소값 δ_3 을 구한다
- ✧ $\min(\delta_1, \delta_2, \delta_3)$ 가 최근접쌍의 거리



최근접 쌍



분할 정복 알고리즘

- ✓ 통합 과정에서 S1의 한 점과 S2의 한 점의 모든 쌍들에 대해서 거리를 구하면 적어도 $N^2/4$ 개의 계산이 필요
- ✓ $T(N) = 2T\left(\frac{N}{2}\right) + O(N^2) \rightarrow T(N) = O(N^2)$
- ✓ 통합 과정을 $O(N)$ 시간 안에 수행할 수 있을까?

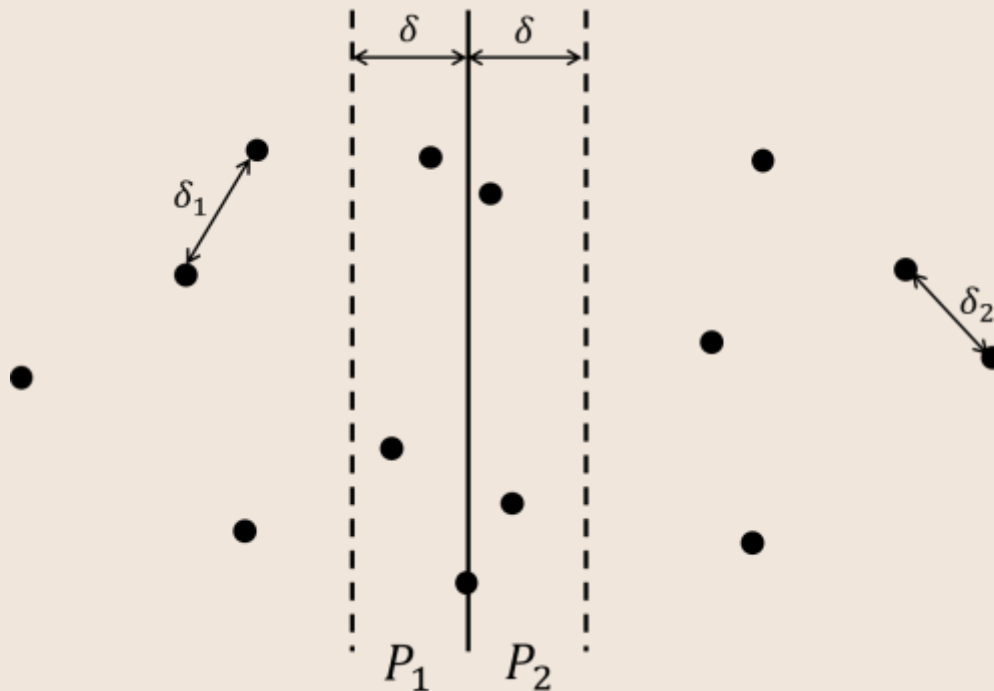


최근접쌍



분할 정복 알고리즘

- ✓ 집합 S_1 과 S_2 가 수직선 L 에 의해서 나뉜다고 하고 $\delta = \min(\delta_1, \delta_2)$ 라고 하면 수직선 L 에서 x 축 방향으로 거리 δ 안의 영역 P 를 생각하자
- ✓ 영역 P 에서 S_1 과 S_2 에 속하는 영역을 P_1 과 P_2 라고 하자



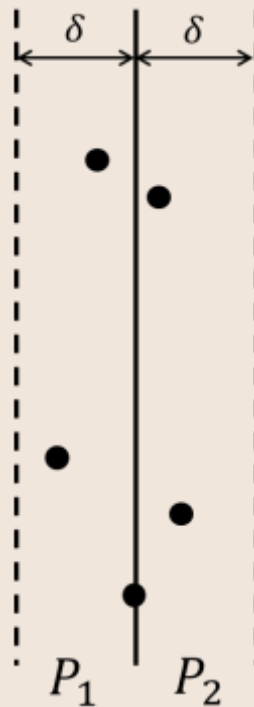


최근접쌍



분할 정복 알고리즘

- ✓ 통합과정에서 S_1 의 한 점 p 와 S_2 의 한 점 q 의 쌍을 고려할 때, P_1 과 P_2 에 속하지 않는 점들은 고려할 필요가 없음 ← P_1 과 P_2 에 속하지 않는 두 점 사이의 거리는 δ 보다 크기 때문에 최근접쌍이 될 수 없음



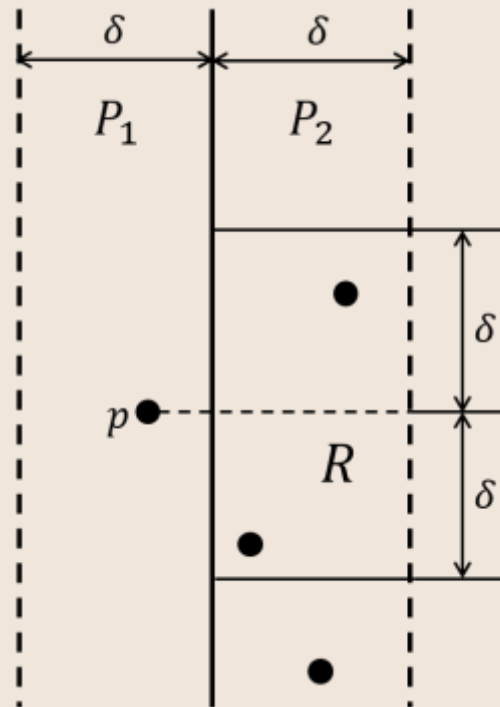


최근접쌍



분할 정복 알고리즘

- ✓ P1의 한 점 p 에 대해서 고려할 P2의 점은 아래 그림의 $\delta \times 2\delta$ 사각형 R 에 속하는 점들이다 ← R 에 속하지 않는 P2의 점은 p 로부터 거리가 δ 보다 크기 때문에 최근접쌍이 될 수 없음



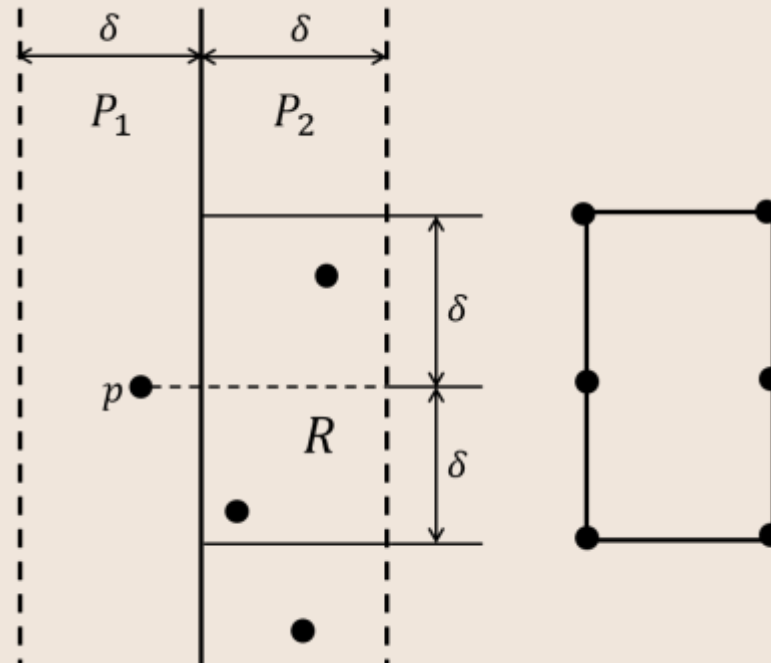


최근접쌍



분할 정복 알고리즘

- ✓ P1의 한 점 p 에 대해서 고려할 사각형 R 에 속하는 점들은 많아야 6개 있음 \leftarrow R 에 속하는 P2의 임의의 두 점은 서로 간의 거리가 δ 보다 크거나 같기 때문에 아래 그림과 같이 위치해야 한다



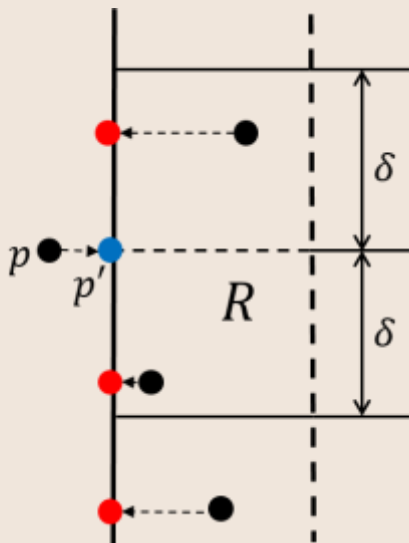


최근접쌍



분할 정복 알고리즘

- ✓ P1의 한 점 p 에 대해서 거리를 계산해야 할 R안의 점들은 많아야 6개임
은 알지만 그 점들이 무엇인지 어떻게 알 수 있는가?
- ✓ P2의 모든 점들을 y축에 사영(projection)해서 점들을 y좌표 값으로 정렬한다
- ✓ P1의 한 점 p 에 대해서, p 를 y축에 사영한 점 p' 의 주변 δ 안의 P2의 점들만을 고려하면 됨 (이 점들은 많아야 6개임)





최근접 쌍



분할 정복 알고리즘

CPair(S)

Partition S into two subset S1 and S2 about the vertical line L
Find the distances of the closest pairs of S1 and S2 recursively
Let δ_1 and δ_2 be such distances of S1 and S2, resp.
Let $\delta = \min(\delta_1, \delta_2)$
Let P1 be the set of points of S1 which are within δ of L and
let P2 be the corresponding subset of S2
Project P1 and P2 onto L and sort by y-coordinate
Let P'1 and P'2 be the sorted sequences
For each p in P'1, inspect the points of P'2 within δ from p
Let δ_3 be the smallest distance of any pair examined
Solution $\leftarrow \min(\delta, \delta_3)$



최근접쌍



분할 정복 알고리즘

Let P_1 be the set of points of S_1 which are within δ of L and
let P_2 be the corresponding subset of S_2
Project P_1 and P_2 onto L and sort by y -coordinate

- ✓ 만약 S_1 과 S_2 의 재귀 호출에서 매번 정렬을 한다면

$$T(N) = 2T\left(\frac{N}{2}\right) + O(N \log N) \rightarrow T(N) = O(N(\log N)^2)$$

- ✓ 재귀 호출 시 마다 정렬하지 않으려면 어떻게 하여야 하는가?



최근접쌍



분할 정복 알고리즘

- ✓ 모든 점들의 집합 S 에 대해서 y 좌표로 정렬한 배열 Y 를 생성
- ✓ S 를 $S1$ 과 $S2$ 로 분할 할 때, Y 역시 $S1$ 에 속하는 점들을 y 좌표로 정렬한 배열 $Y1$ 과 $S2$ 에 속하는 점들을 y 좌표로 정렬한 배열 $Y2$ 로 분할
- ✓ 이 분할은 다음과 같이 $O(N)$ 시간에 수행 가능

```
Examine the points in  $Y$  in order
```

```
IF  $Y[i]$  is in  $S1$ 
```

```
    Append  $Y[i]$  to the end of  $Y1$ 
```

```
ELSE
```

```
    Append  $Y[i]$  to the end of  $Y2$ 
```



평면 소거법(Plane Sweeping)

- ✎ 평면소거법은 평면을 수평선 또는 수직선(sweep-line 이라 부름)으로 아래에서 위 또는 왼쪽에서 오른쪽으로 쓸어가면서 주어진 문제를 해결하는 기법이다
- ✎ 기본적인 아이디어는 해결하기 복잡한 2차원의 문제를 단순한 1차원의 문제로 나누어 해결하고자 하는 것이다. Sweep-line이 1차원을 표현한다. 이를 위하여 다음 두 가지의 자료 구조를 이용한다



평면 소거법

- ✎ Event point의 순서를 결정하는 자료 구조
 - ✓ sweep-line의 움직임에 따라 그 상태를 변경하여야 하는데, 상태가 변화되는 sweep-line의 위치 (event-point 라 부름)의 순서를 가지는 자료 구조
- ✎ Sweep-line 의 상태를 나타내는 자료 구조
 - ✓ 해결하고자 하는 데이터와 현 위치의 sweep-line 이 교차하고 있는 상태를 표현함
- ✎ 이 두 자료 구조는 해결하고자 하는 문제에 따라 각기 달리 설정된다

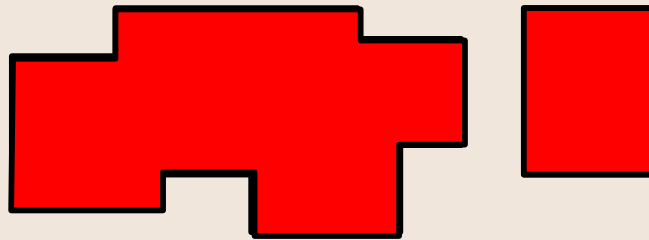


직사각형 합집합



문제

- ✓ X축과 Y축에 각 변이 평행한 직사각형들이 N개 주어질 때, 이들의 합집합의 면적을 구하시오



단순한 합

- ✓ 각 직사각형들의 면적을 구해서 합하면 겹치는 부분을 여러 번 합하게 되므로 답이 될 수 없다

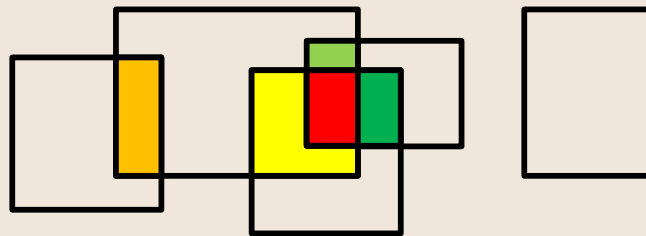


직사각형 합집합



포함-배제 원칙(inclusion-exclusion principle) 이용

- ✓ 각 직사각형들의 면적을 모두 더한 뒤, 두 직사각형들의 쌍에 대해 이들의 교집합의 면적을 구함. 이 면적은 두 번 더해졌으므로 결과에서 빼 줌. 그러면 세 개의 사각형이 겹치는 부분은 세 번 더해지고 세 번 빼졌으므로 이들을 다시 더하는 식으로 반복 → 모든 조합의 교집합을 한번씩 고려해야 함으로 2^n 개의 교집합을 계산해야 함





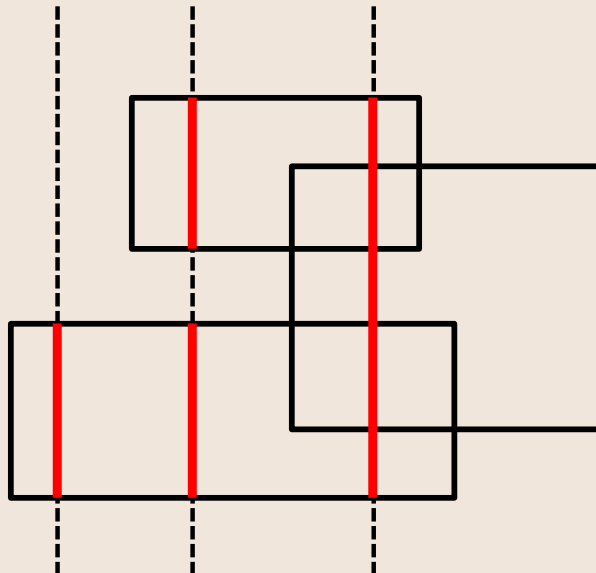
직사각형 합집합



평면 소거법 알고리즘 설계

- ✓ 왼쪽에서 오른쪽으로 쓸고 지나가는 수직선(sweep-line)을 상상해 보자
- ✓ sweep-line의 좌표 x 가 주어질 때, 사각형들을 수직선으로 잘랐을 때 단면의 길이를 반환하는 함수 `cutLength()`가 있다면 합집합의 넓이는 이 함수를 x 에 대해 정적분한 결과
- ✓ Don't mind : `cutLength()`를 실제로 적분할 필요는 없다

Sweep-line





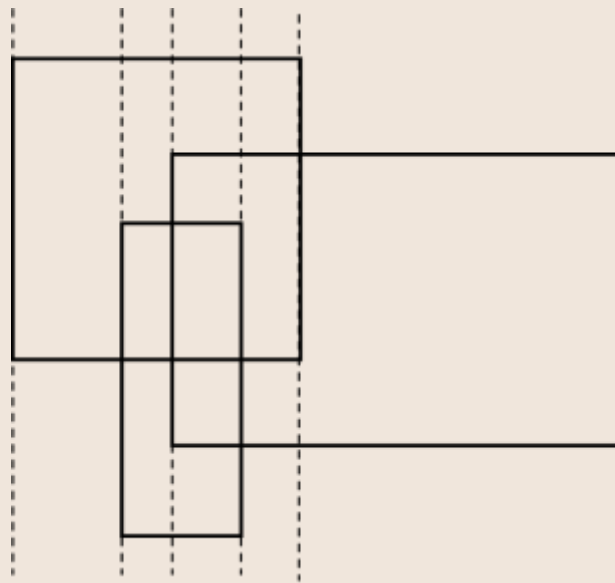
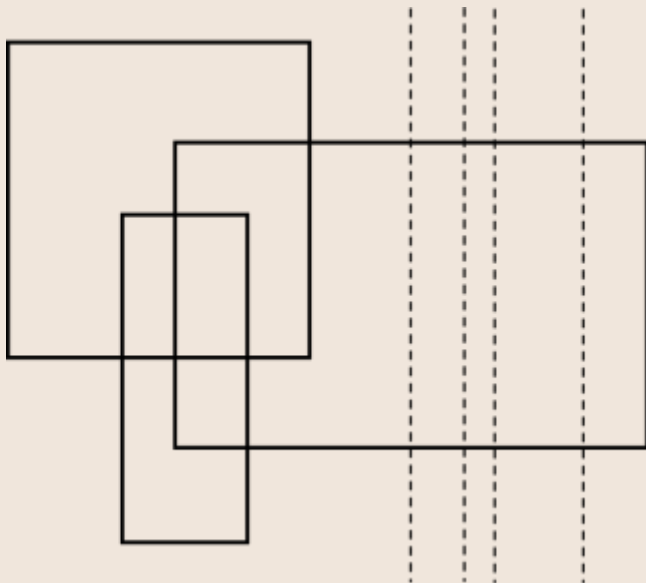
직사각형 합집합



평면 소거법 알고리즘 설계

✓ Event point의 순서를 결정하는 자료 구조

- ✱ 아래 왼쪽 그림에서 네 수직선에서 단면의 길이는 항상 같음
- ✱ 아래 오른쪽 그림처럼 단면의 길이가 변하는 지점은 직사각형의 왼쪽 끝이거나 오른쪽 끝점 뿐이다





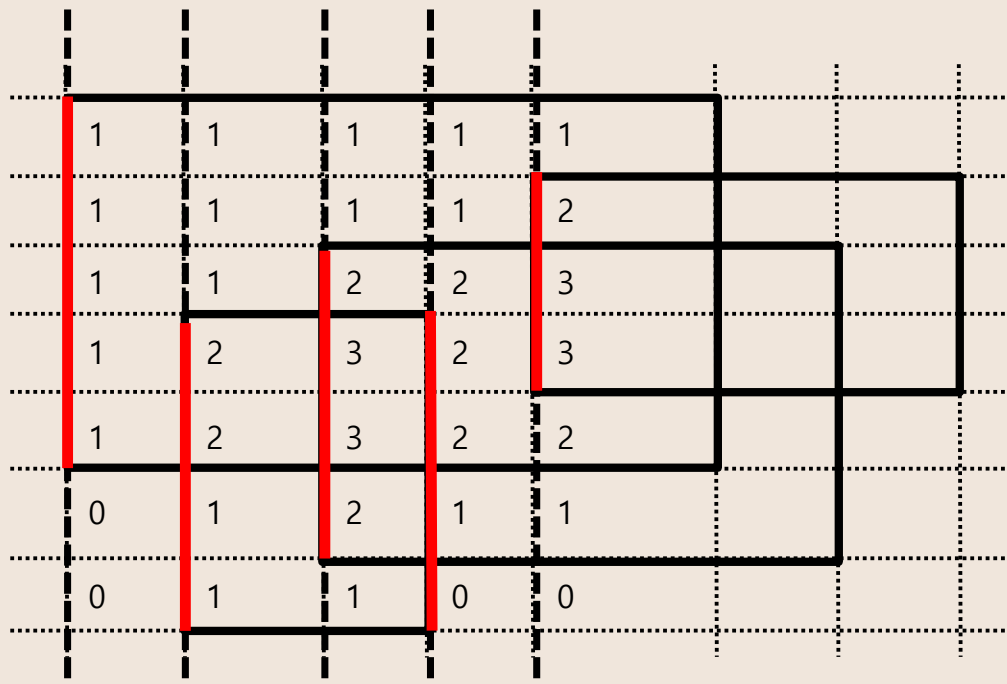
직사각형 합집합



평면 소거법 알고리즘 설계

✓ Sweep-line의 상태를 나타내는 자료 구조

- ✧ sweep-line 이 지나는 좌표 x의 단면에서 사각형이 겹치는 부분을 표시
- ✧ 사각형의 위/아래 변의 y좌표 값들로 단면의 부분들을 구분
- ✧ 배열 count를 유지해서 단면의 각 부분에서 겹치는 사각형의 개수를 저장





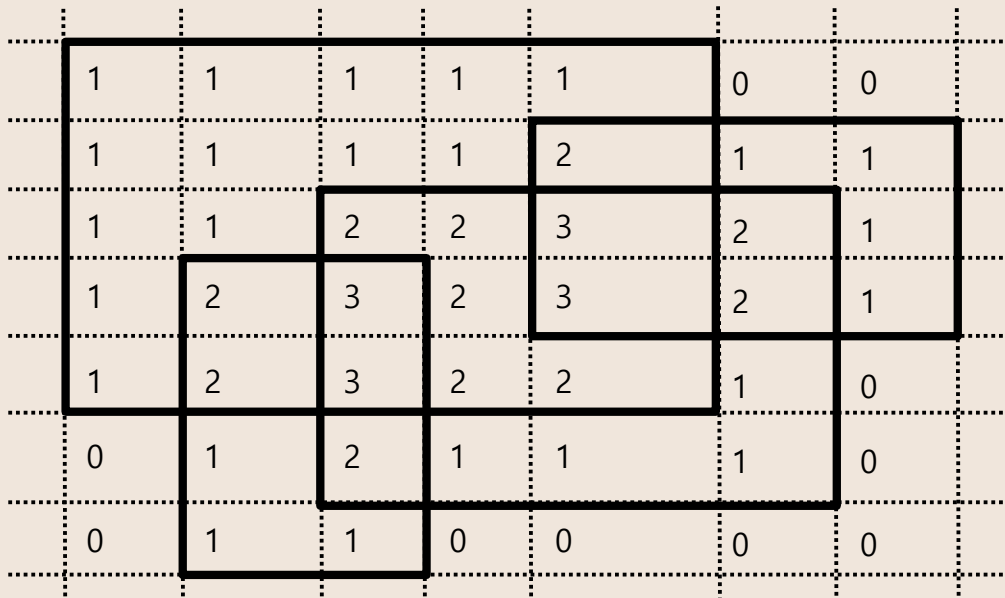
직사각형 합집합



평면 소거법 알고리즘 설계

✓ Sweep-line의 상태를 나타내는 자료 구조

- ✧ sweep-line 이 지나는 좌표 x의 단면에서 사각형이 겹치는 부분을 표시
- ✧ 사각형의 위/아래 변의 y좌표 값들로 단면의 부분들을 구분
- ✧ 배열 count를 유지해서 단면의 각 부분에서 겹치는 사각형의 개수를 저장





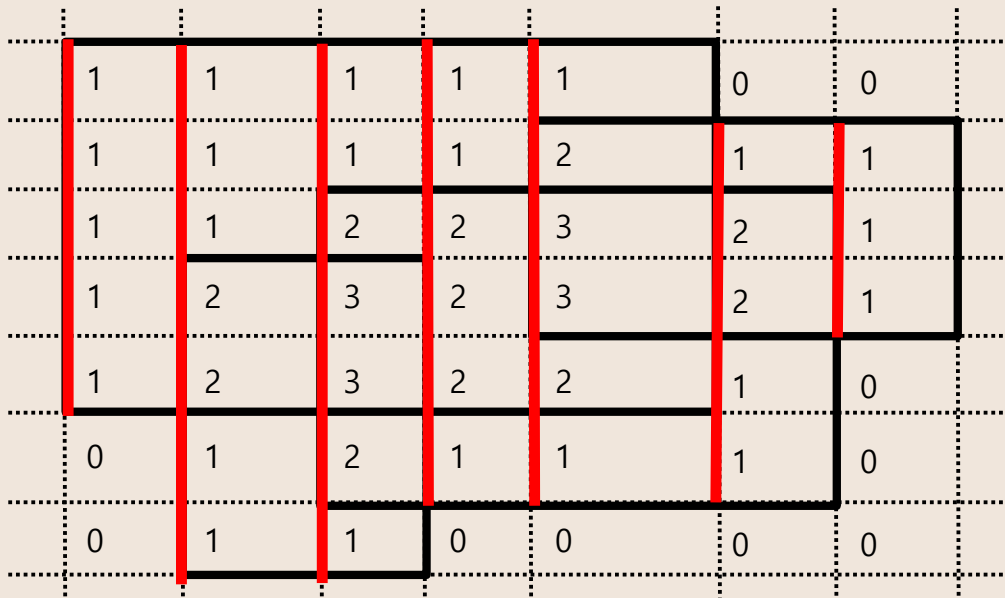
직사각형 합집합



평면 소거법 알고리즘 설계

✓ Sweep-line의 상태를 나타내는 자료 구조

- ✧ sweep-line 이 지나는 좌표 x의 단면에서 사각형이 겹치는 부분을 표시
- ✧ 사각형의 위/아래 변의 y좌표 값들로 단면의 부분들을 구분
- ✧ 배열 count를 유지해서 단면의 각 부분에서 겹치는 사각형의 개수를 저장





직사각형 합집합



평면 소거법 알고리즘

- ✓ 직사각형의 표현

: 왼쪽-아래 모서리 좌표 $(x1, y1)$ 와 오른쪽-위 모서리 좌표 $(x2, y2)$

```
struct Rectangle{  
    int x1, y1, x2, y2;    //x1 < x2, y1 < y2  
};
```

- ✓ 배열 $Q[i]$: event-point 들의 집합, 직사각형들의 왼쪽, 오른쪽 변의 x 좌표 들을 정렬해서 저장
- ✓ 배열 $S[i]$: sweep-line 에 의한 단면의 정보, 직사각형들의 위쪽, 아래쪽 변의 y 좌표들을 정렬해서 저장

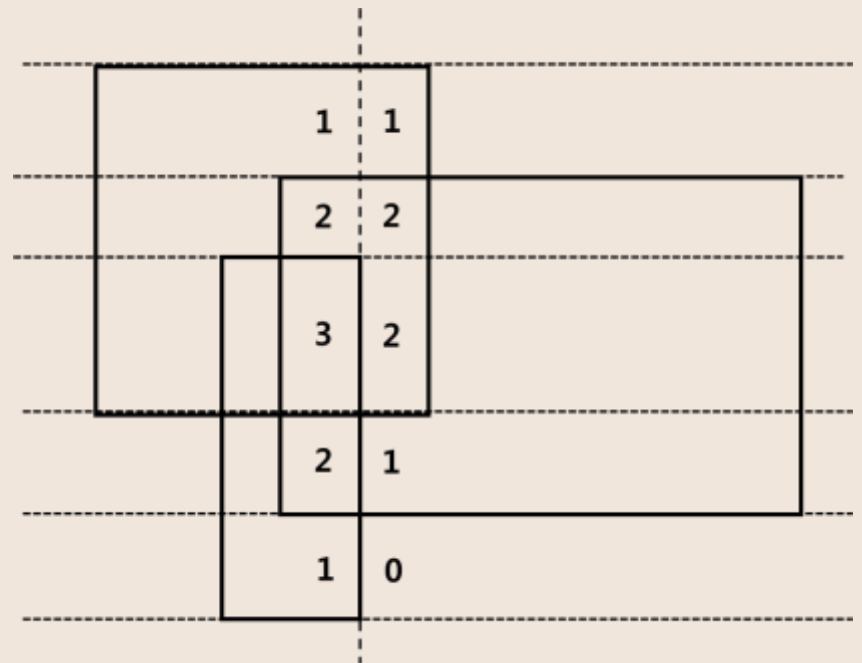
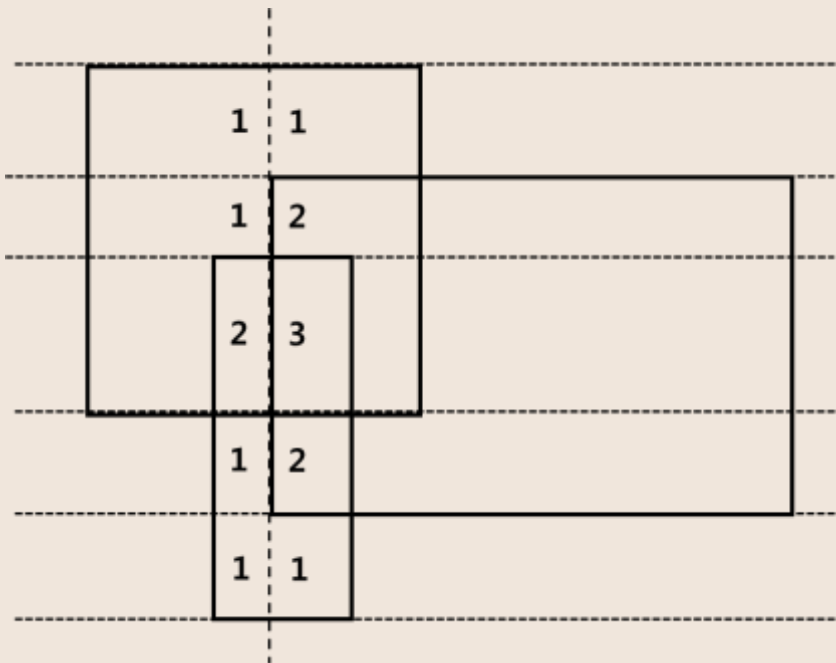
평면 소거법 알고리즘

UnionArea()

```
FOR i in 0 → m1-1                                //m1 : # of events
  x ← Q[i]
  IF sweep-line touches a left side of rectangle at Q[i]
    delta ← 1
  ELSE delta ← -1
  R ← the rectangle at Q[i]
  y1 ← R.y1    y2 ← R.y2
  area ← 0
  // count[i] = S[i] ~ S[i+1] 구간에 겹쳐진 사각형의 수
  FOR j in 0 → m2-1                                //m2 : size of the array S
    IF y1 ≤ S[j] AND S[j] < y2
      count[j] += delta
```

평면 소거법 알고리즘

```
// count[i] = S[i] ~ S[i+1] 구간에 겹쳐진 사각형의 수
FOR j in 0 → m2-1           //m2 : size of the array S
  IF y1 ≤ S[j] AND S[j] < y2
    count[j] += delta
```





평면 소거법 알고리즘

```
cutlength ← 0
FOR j in 0 → m2-1
    IF count[j] > 0
        cutLength += S[j+1] - S[j]
// the length between Q[i] and Q[i+1] * cutLength
IF i + 1 < m1
    area += cutLength * (Q[i+1] - x)

return area
```

평면 소거법 알고리즘

```
cutlength  $\leftarrow$  0
```

```
FOR j in 0  $\rightarrow$  m2-1
```

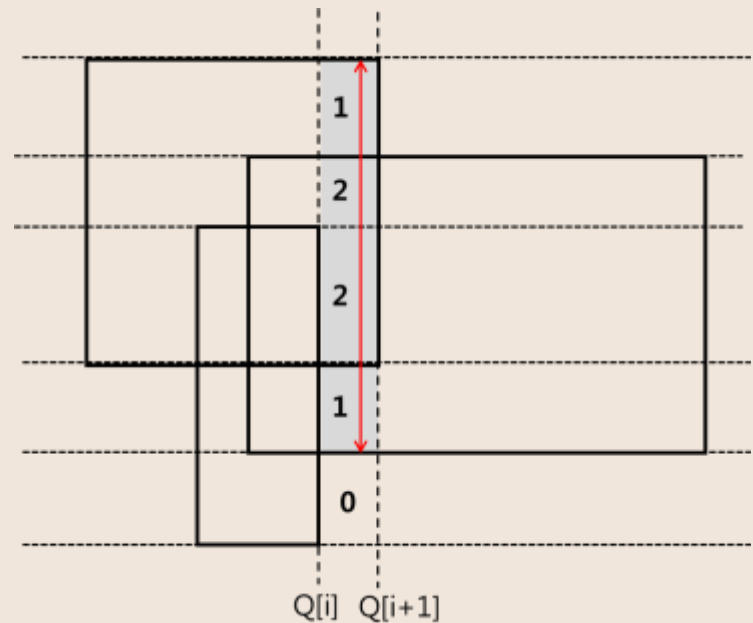
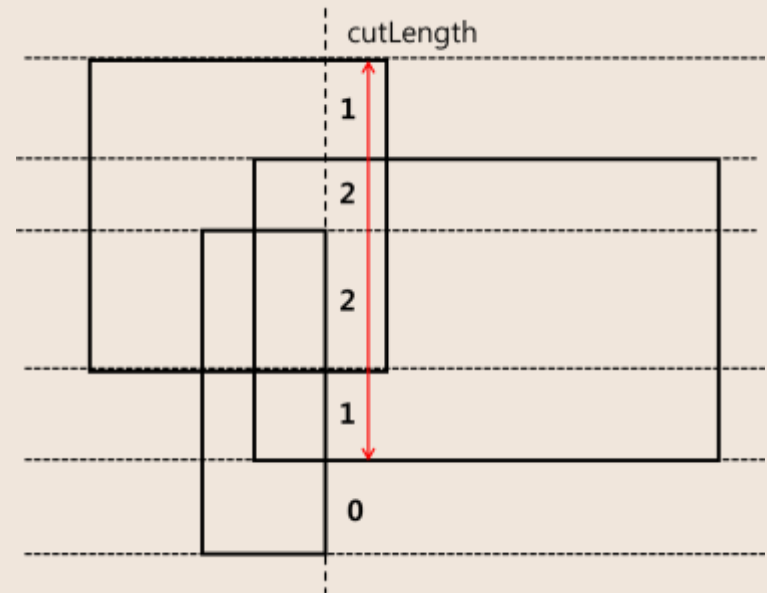
```
  IF count[j] > 0
```

```
    cutLength += S[j+1] - S[j]
```

```
area  $\leftarrow$  0
```

```
IF i + 1 < m1
```

```
  area += cutLength * (Q[i+1] - x)
```

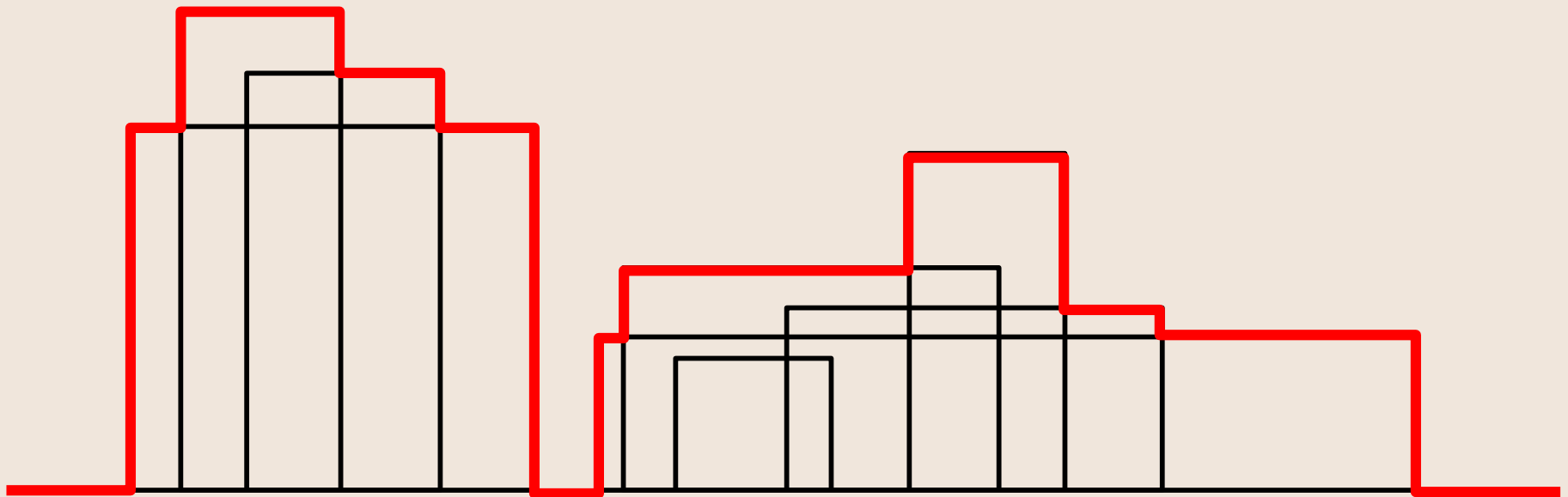




응용 문제



Skyline

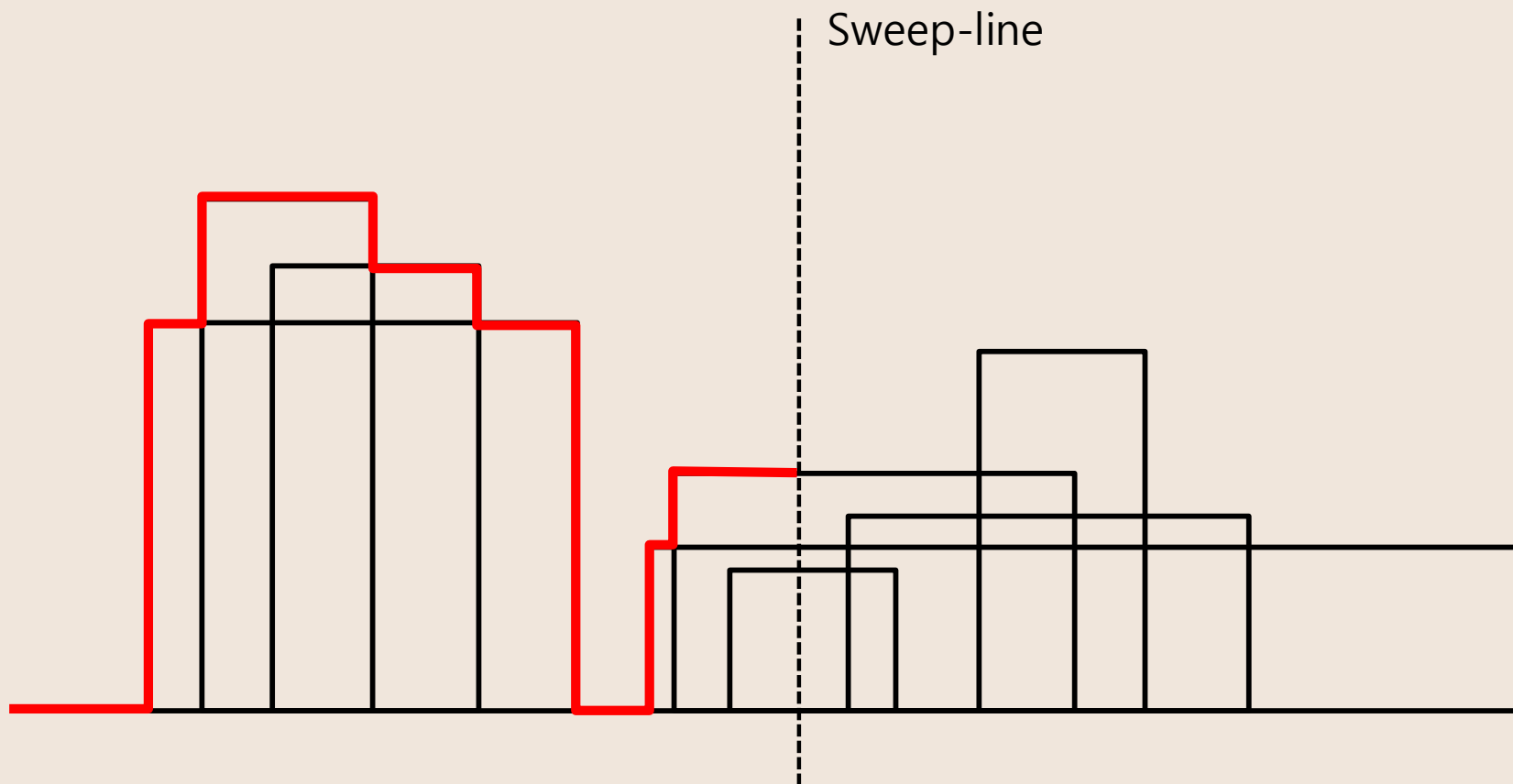




응용 문제



Skyline

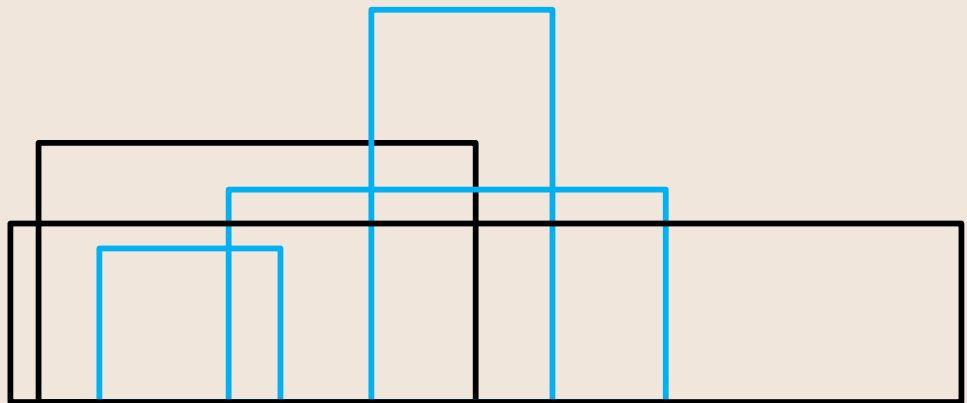
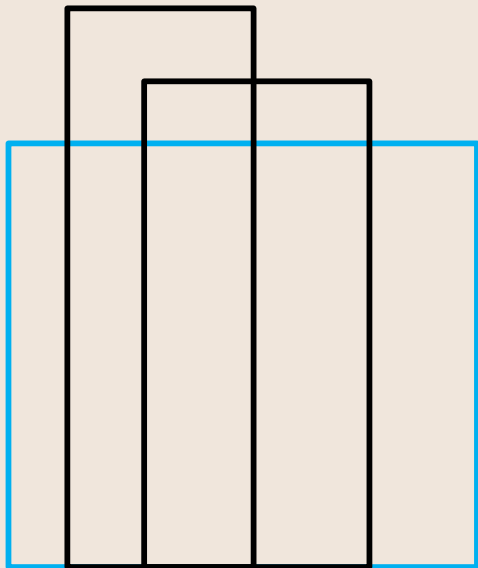




응용 문제



Skyline (분할정복법)

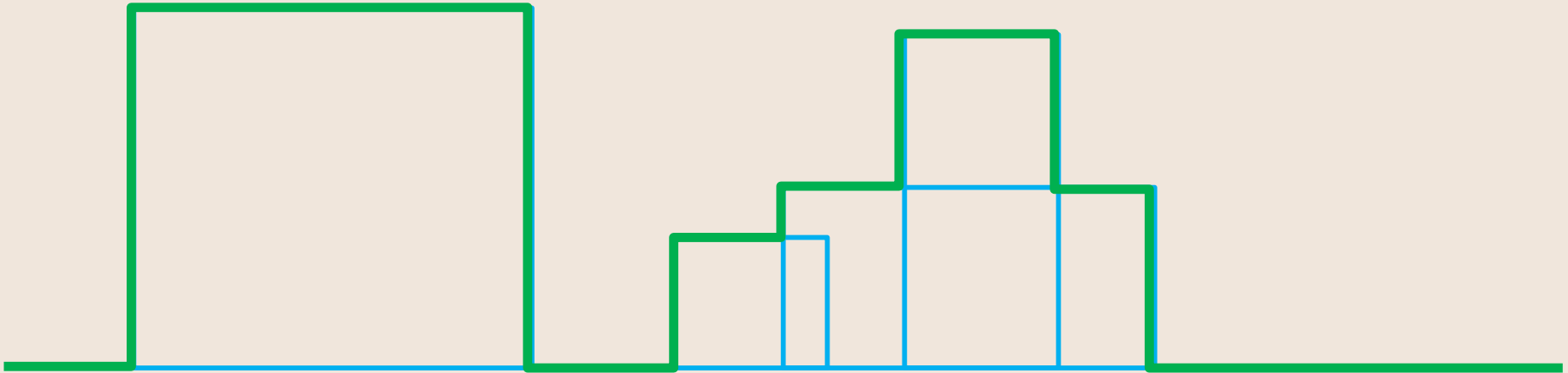




응용 문제



Skyline (분할정복법)

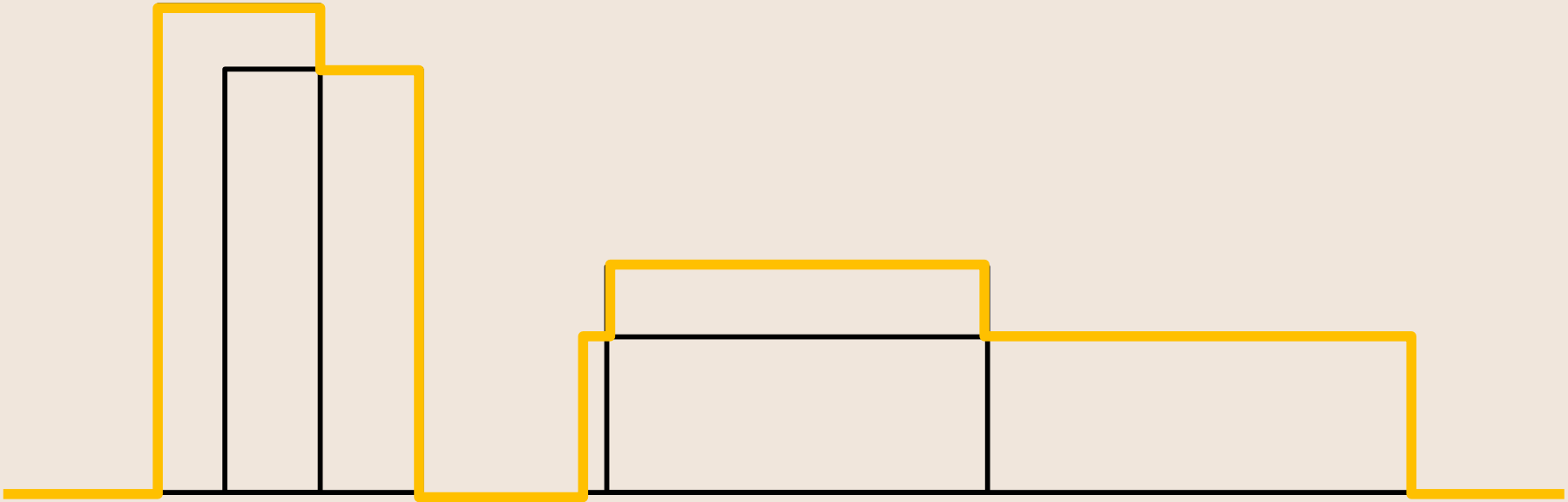




응용 문제



Skyline (분할정복법)

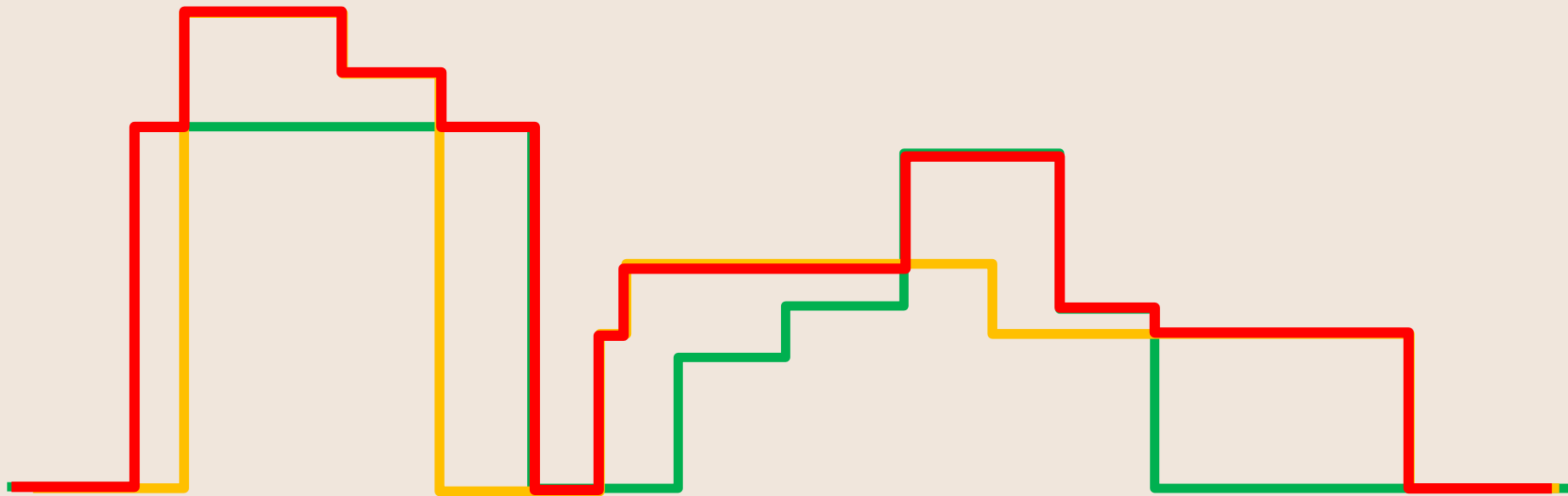




응용 문제



Skyline (분할정복법)





응용 문제



Skyline 시간 복잡도 분석

- ✓ By Sweep line : $O(n \lg n)$
- ✓ By Divide & Conquer : $O(n \lg n)$



응용 문제



룸메이트(ICPC2006)

- ✓ 철수는 기숙사 방을 같이 사용하는 룸메이트가 있다. 그들은 오랫동안 함께 살아왔기 때문에 헤어 드라이기, 면도기, 배터리 충전기 등의 생활 용품들을 공동으로 사용하고 있다. 따라서 용품들을 사용할 시간이 겹치지 않도록 사용 시간을 잘 배분해야만 한다
- ✓ 철수가 사용할 용품들이 시간 순서대로 $o_{11}, o_{12}, \dots, o_{1m}$ 으로 주어지고 룸메이트가 사용할 용품들은 $o_{21}, o_{22}, \dots, o_{2n}$ 로 주어진다
- ✓ 각 용품 o_{ij} 에는 용품을 사용하는데 걸리는 시간 p_{ij} 가 주어진다
- ✓ 철수와 룸메이트 모두가 용품들을 다 사용할 때까지 걸리는 시간이 최소가 되도록 용품들의 사용시간을 결정하시오



응용 문제



룸메이트

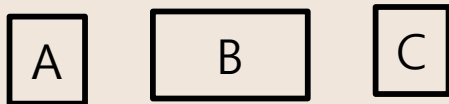
- ✓ 예) 공동으로 사용하는 용품 A, B, C가 주어지고 철수가 각 용품을 사용하는 시간이 2, 1, 3으로 룸메이트가 사용하는 시간이 1, 2, 1로 주어진다. 즉, 철수(룸메이트 역시)는 각 용품을 1번 이상 사용할 수 있는데, 일단 어떤 용품을 사용하면 앞에서 말한 시간만큼 그 용품을 사용한다.

철수가 A, B, A, C의 순서로 룸메이트가 A, C, A, B의 순서로 용품들을 사용하려고 한다면 어떻게 배분해야 할까?

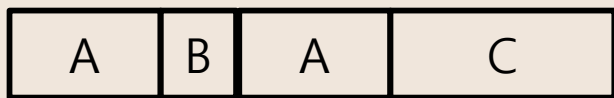
철수



룸메이트



철수



룸메이트



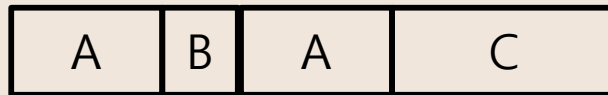


응용 문제

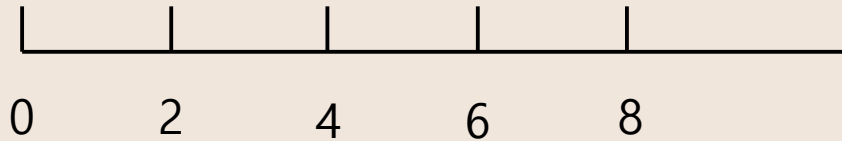


룸메이트

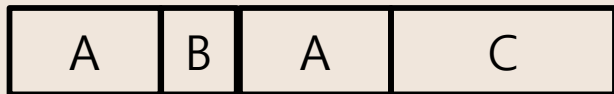
철수



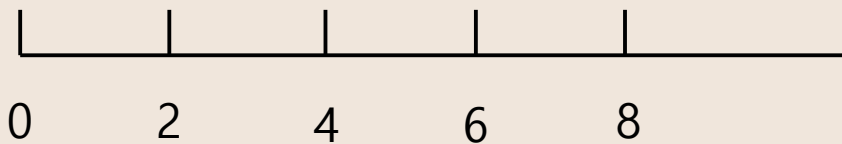
룸메이트



철수



룸메이트



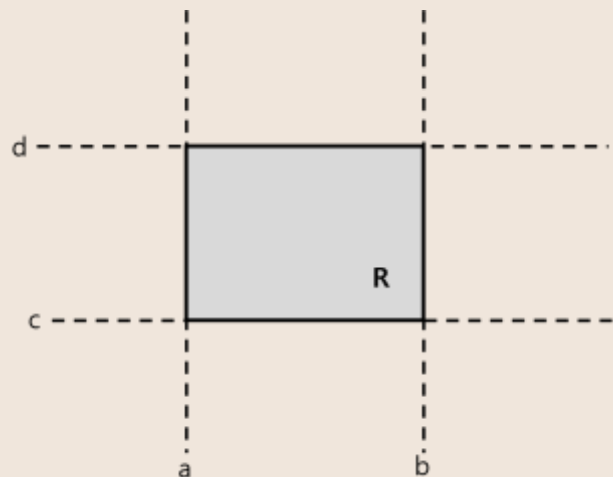


응용 문제



룸메이트

- ✓ 언뜻 보기에는 계산 기하학 문제가 아닌 것으로 보이지만 계산 기하학으로 풀 수 있다
- ✓ 철수가 사용할 용품들의 시간 구간들을 순서대로 x 축에 배치하고 룸메이트가 사용할 용품들의 시간 구간들을 y 축에 배치
- ✓ x 축 구간 $I = [a, b]$ 와 y 축 구간 $J = [c, d]$ 가 같은 용품을 사용하는 구간이면 이 구간들이 평면 상에서 겹치는 사각형 영역 R 을 장애물(obstacle)로 생각



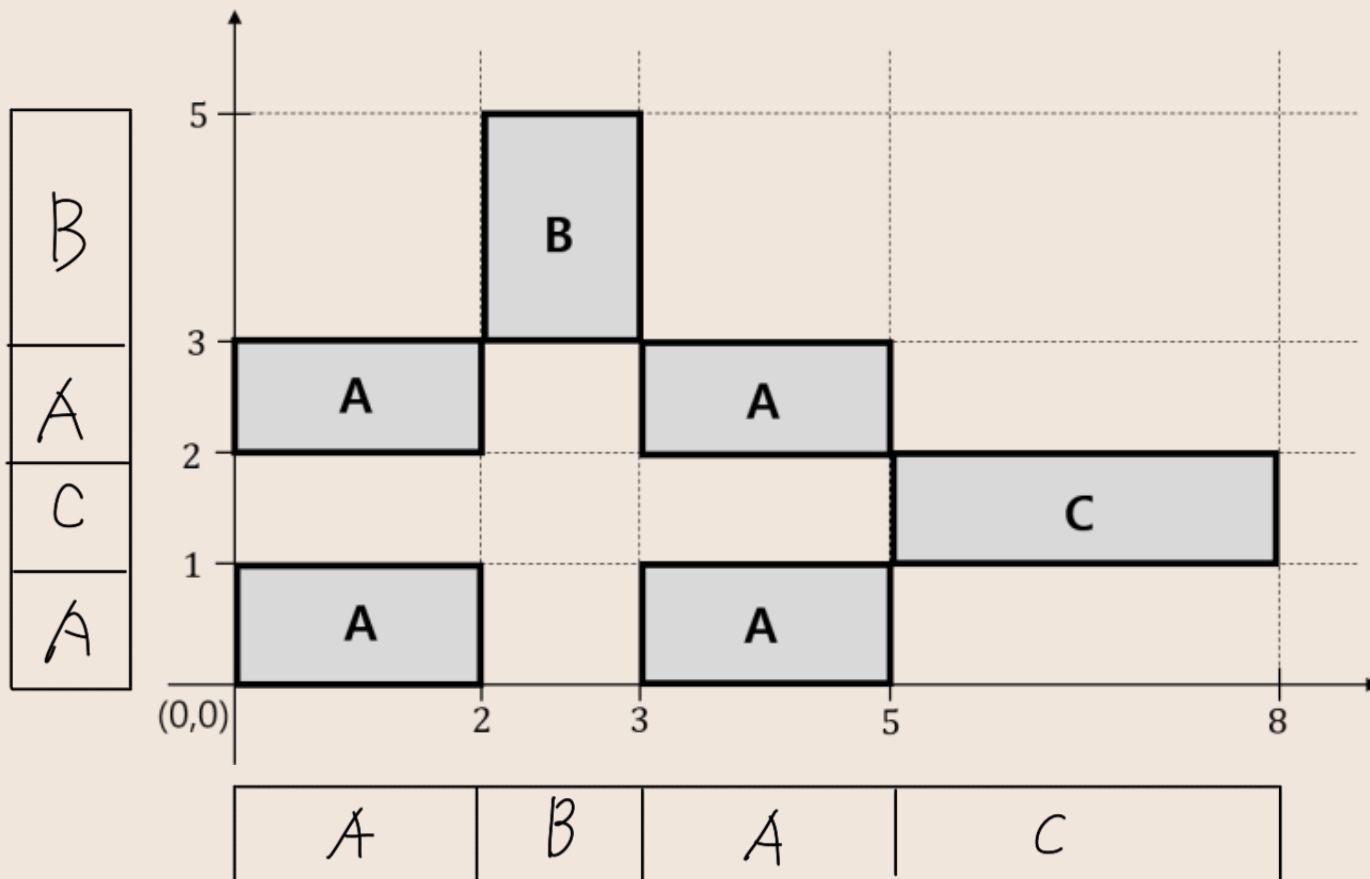


응용 문제



룸메이트

- ✔ 앞의 예제는 다음과 같은 장애물들이 존재하는 상황으로 변환됨





응용 문제



룸메이트

- ✓ 철수가 사용할 용품들의 시간의 합 $f_1 = p_{11} + p_{12} + \cdots + p_{1m}$ 과 룸메이트가 사용할 용품들의 시간의 합 $f_2 = p_{21} + p_{22} + \cdots + p_{2n}$ 에 대해서, 점 $F = (f_1, f_2)$ 를 생각하자
- ✓ 원 점 $(0, 0)$ 에서 시작해서 점 F까지의 장애물을 통과하지 않는 경로 P를 생각한다. 여기서, 경로 P는 수평, 수직 선분과 대각 선분(x축과 45도 각도의 기울기를 가진 선분)들로 구성된다
- ✓ 경로 P는 다음과 같은 의미를 가진다
 - ✧ 수평 선분 : 철수가 용품을 사용하고 룸메이트는 기다리는 경우
 - ✧ 수직 선분 : 룸메이트가 용품을 사용하고 철수가 기다리는 경우
 - ✧ 대각 선분 : 철수와 룸메이트가 (서로 다른) 용품을 동시에 사용하는 경우

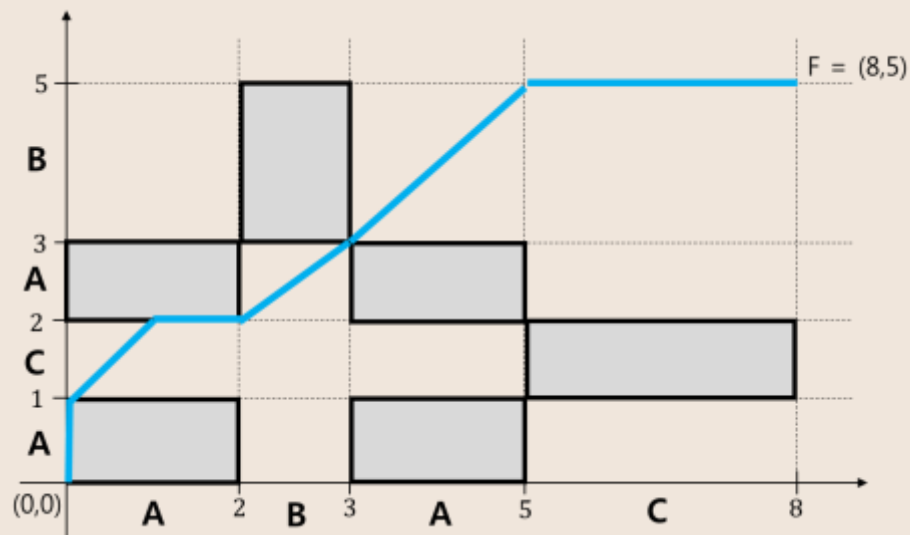


응용 문제

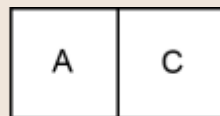


룸메이트

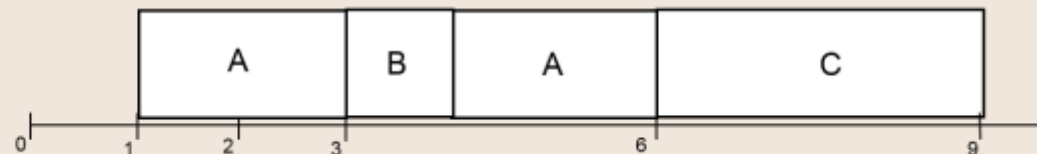
- ✔ 앞의 예제에서 그림의 파란 경로 P는 아래 스케줄을 나타낸다



룸메이트 :



철수 :



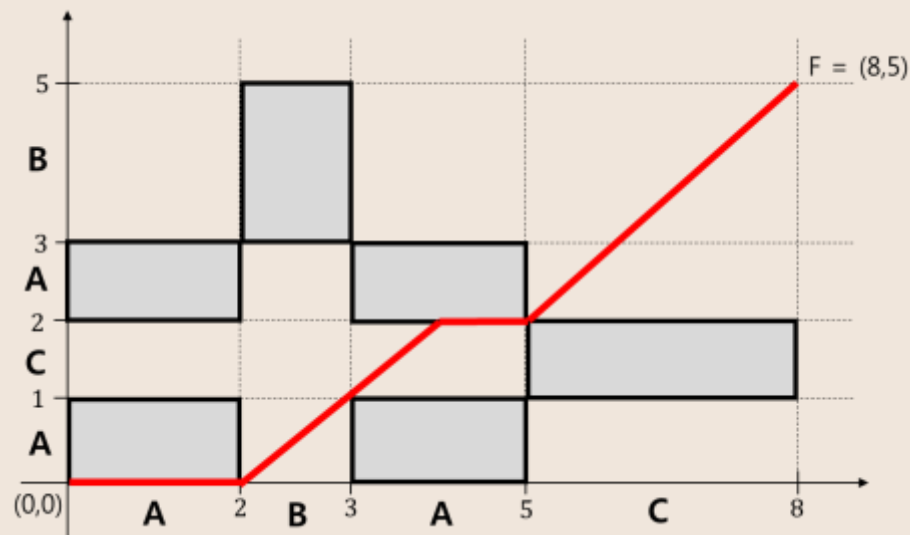


응용 문제



룸메이트

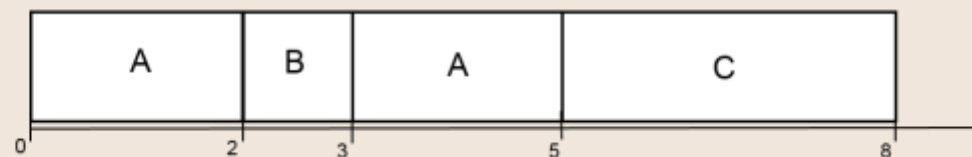
- ✔ 앞의 예제에서 그림의 빨간 경로 P는 아래의 최적 스케줄을 나타낸다



룸메이트 :



철수 :



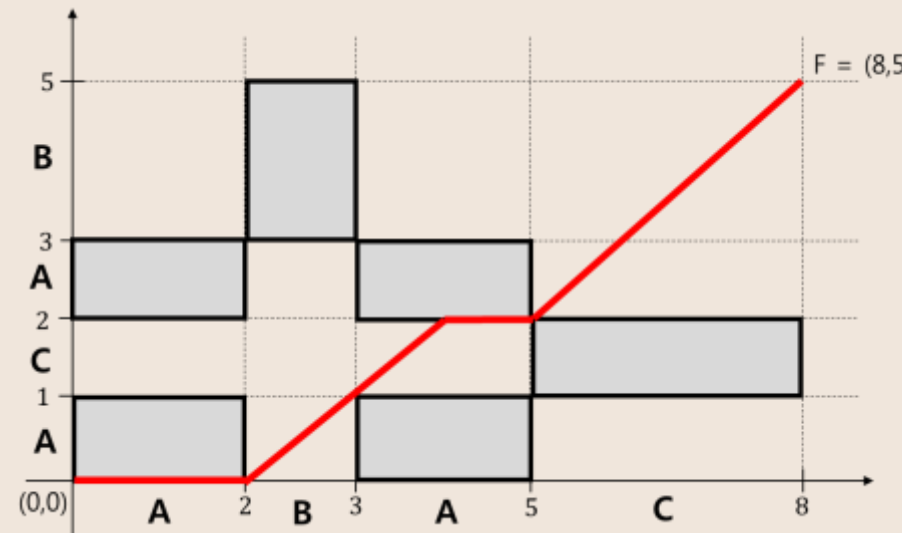
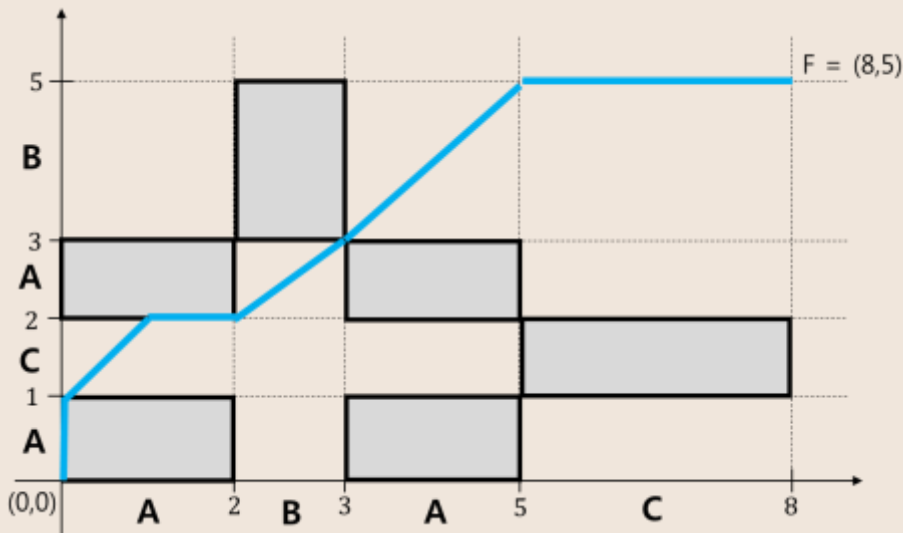


응용 문제



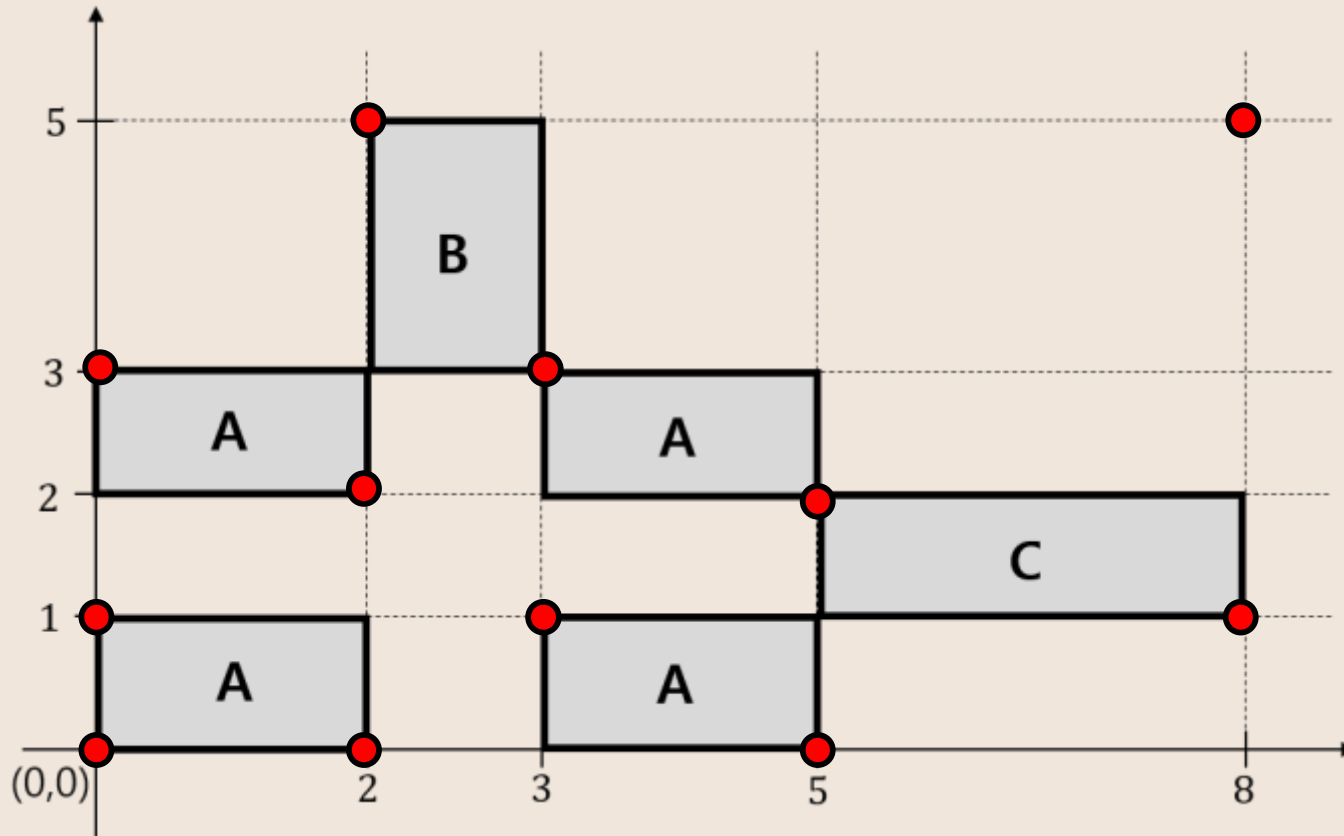
룸메이트

- ✓ 이제 문제는 수직/수평 변을 가진 사각형의 장애물들이 존재 할 때, 원점에서 점 F까지 장애물을 통과하지 않는 최단 경로를 찾는 문제가 되었다
- ✓ 이 최단 경로를 어떻게 찾을 수 있을까?



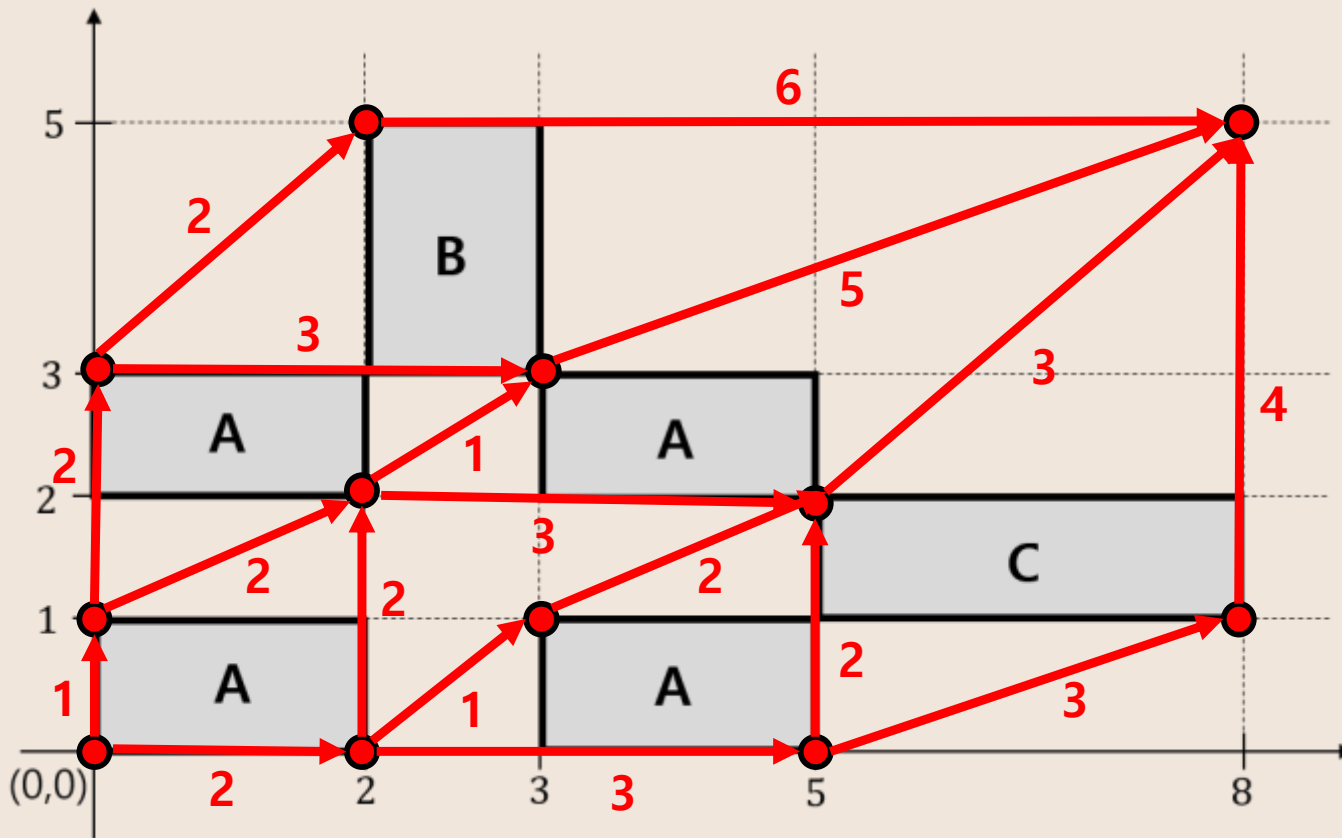


응용 문제



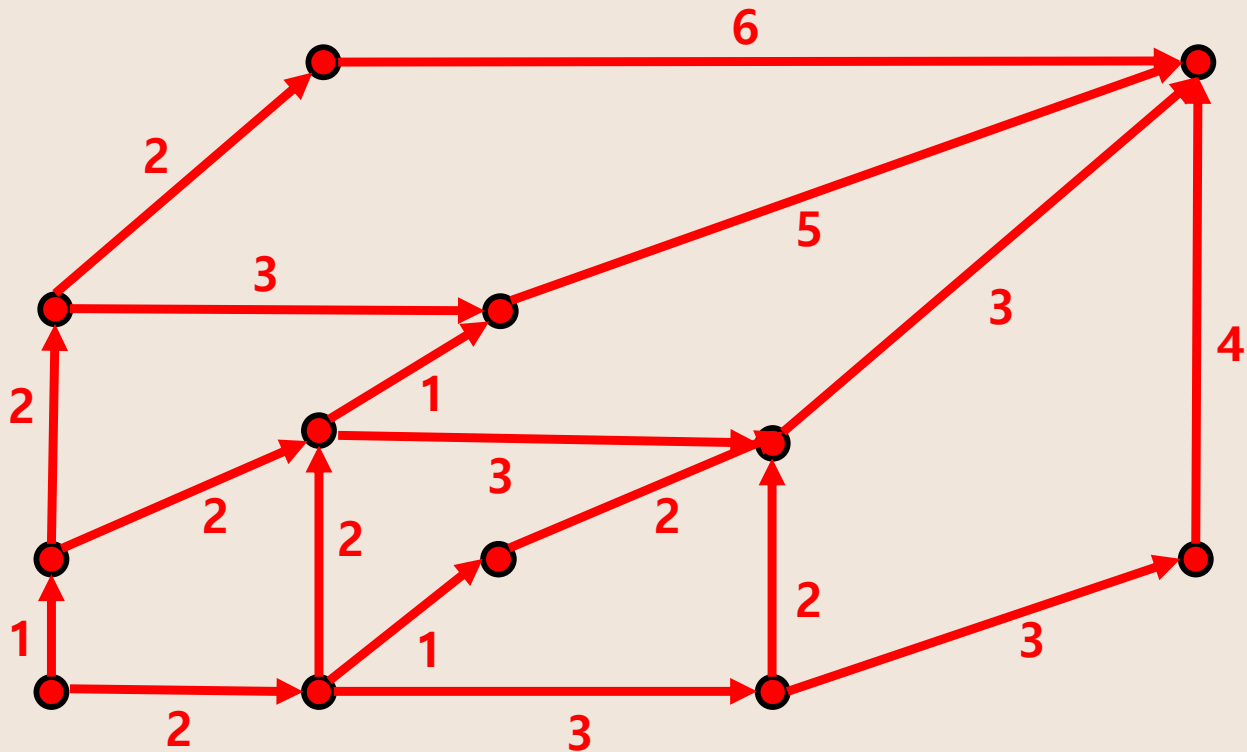


응용 문제



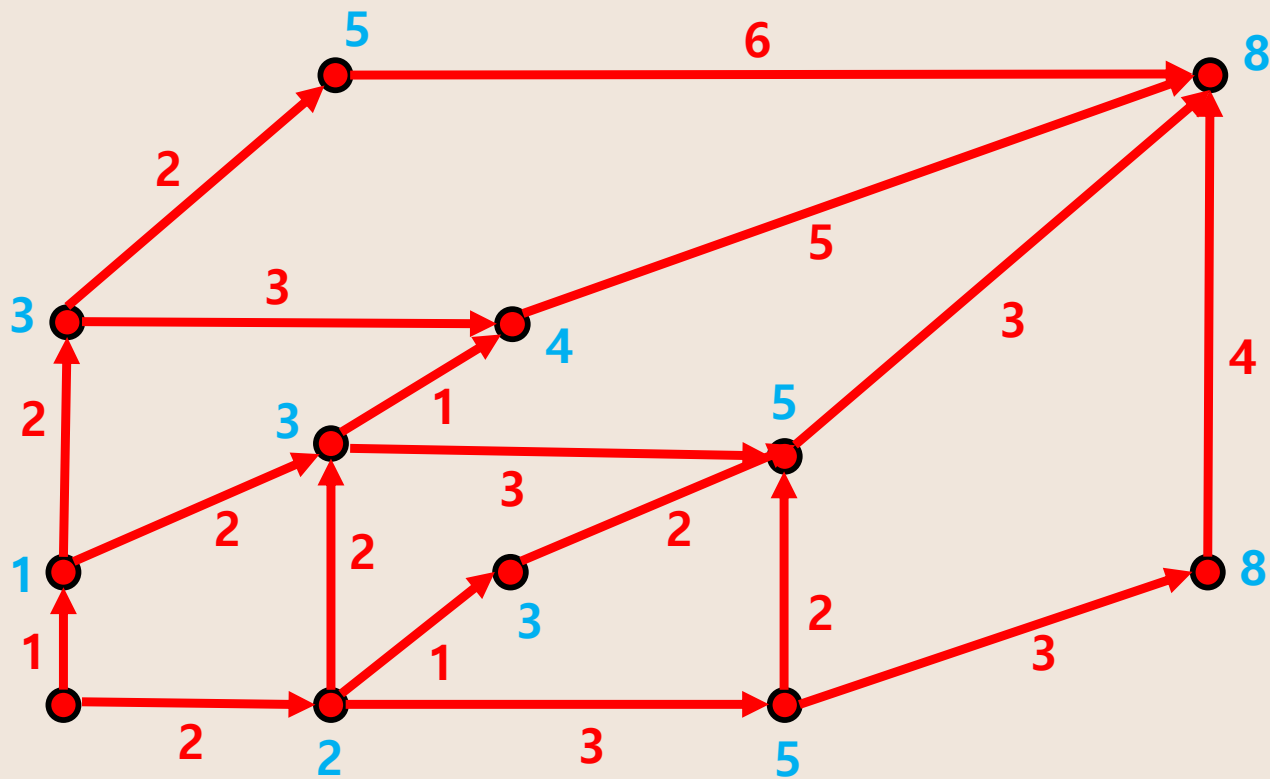


응용 문제



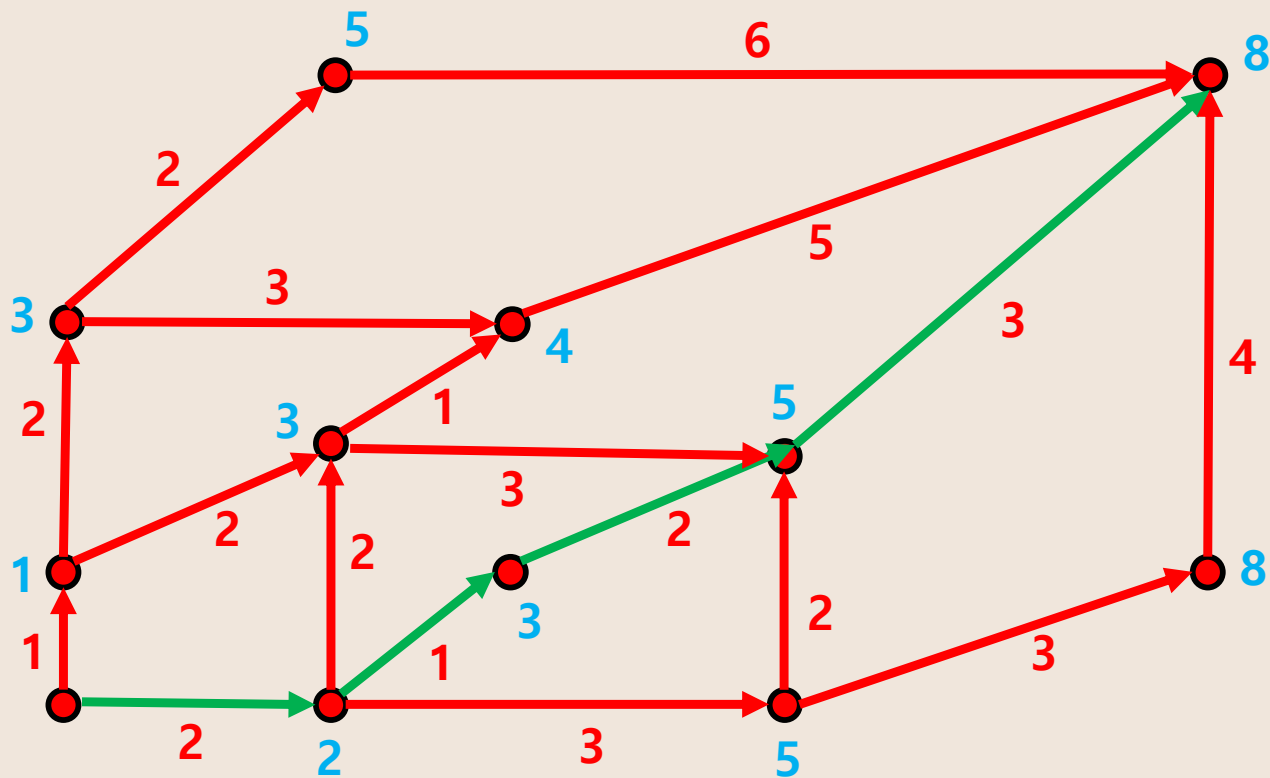


응용 문제





응용 문제



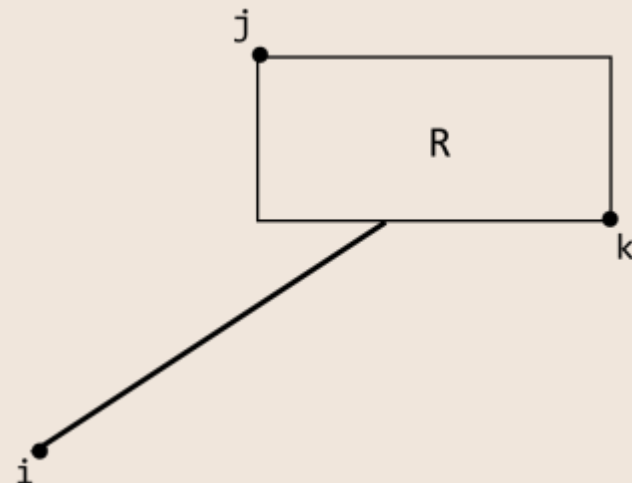


응용 문제



룸메이트

- ✓ 사각형의 장애물 R에 대해서, 사각형의 왼쪽 위 꼭지점(NW-꼭지점)과 오른쪽 아래 꼭지점(SE-꼭지점)을 생각한다
- ✓ 그래프 $N=(V, E)$ 이 다음과 같이 정의된다
 - * V 는 원점, F 그리고 모든 장애물들의 NW-꼭지점과 SE-꼭지점으로 구성된다
 - * 아래 그림에서와 같이 정점 i 에서 대각선 방향으로 선을 그으면 원점과 F로 정의된 큰 사각형과 만나거나 장애물의 테두리와 만난다





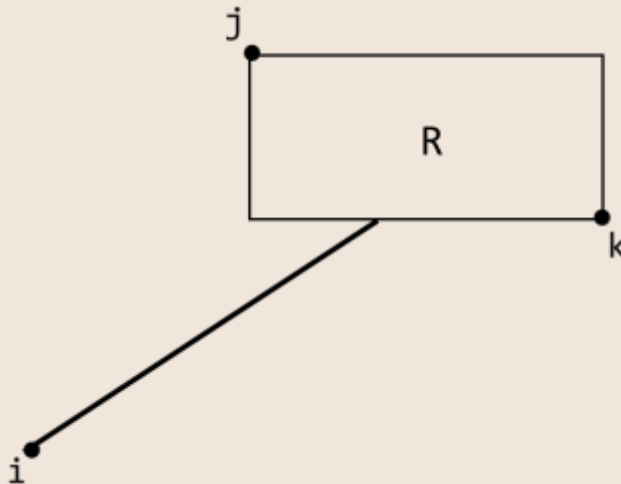
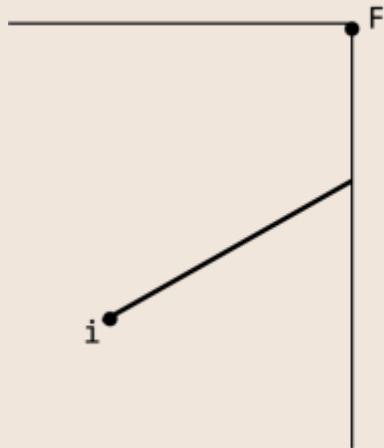
응용 문제



룸메이트

✓ 그래프 $N=(V, E)$ 이 다음과 같이 정의된다

- ✱ 정점 i 로부터 대각선이 원점과 F 로 정의된 큰 사각형과 만나면 간선 $e=(i, F)$ 가 추가된다
- ✱ 정점 i 로부터 대각선이 장애물 R 의 테두리와 처음으로 만나고 R 의 NW-꼭지점과 SE-꼭지점을 각각 j, k 라고 할 때, 간선 (i, j) 와 (i, k) 가 추가된다
- ✱ 따라서 각 정점 $i \in V - \{F\}$ 은 많아야 두 개의 간선과 인접하다





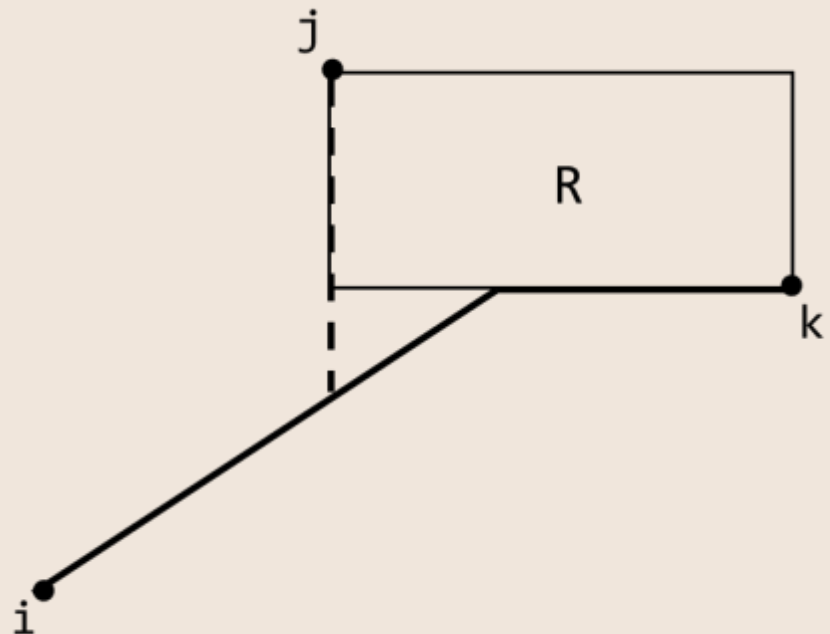
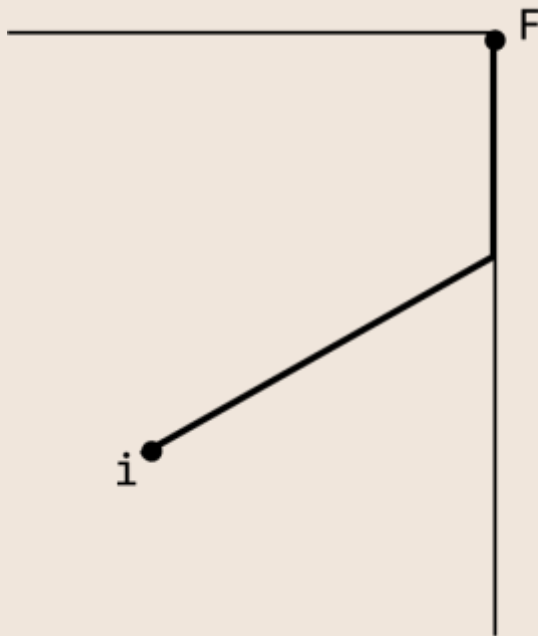
응용 문제



룸메이트

✓ 그래프 $N=(V, E)$ 이 다음과 같이 정의된다

- * 각 간선 (i, j) 의 길이(가중치)가 i 에서 j 로의 경로의 수직 또는 수평 부분의 길이와 대각 부분의 길이의 합으로 주어진다





응용 문제



룸메이트

- ✓ 원점에서 점 F까지 장애물을 통과하지 않는 최단 경로의 길이는 이 그래프 N에서 원점에서 F까지의 최단 경로의 길이와 같다
- ✓ N에서 최단 경로를 찾는 알고리즘을 이용해서 문제를 해결해보시오