알고리즘 6장 – 기하 알고리즘

멀티미디어공학과
조 미 경 교수

기하 요소 표현

- 점, 선, 다각형 등의 도형을 다루는 분야에서 필요한 기초적인 기하 알고리즘
- 점

  x축과 y축의 좌표로 표현

```
struct point {
   int x;                  /* 점의 x 좌표 */
   int y;                  /* 점의 y 좌표 */
} ;  /* point 형의 변수들 */

point FirstPoint, SecondPoint;
```

기하 요소 표현

- 선

  2개의 점으로 표현되며 이들 2점이 직선으로 연결되었다고 가정

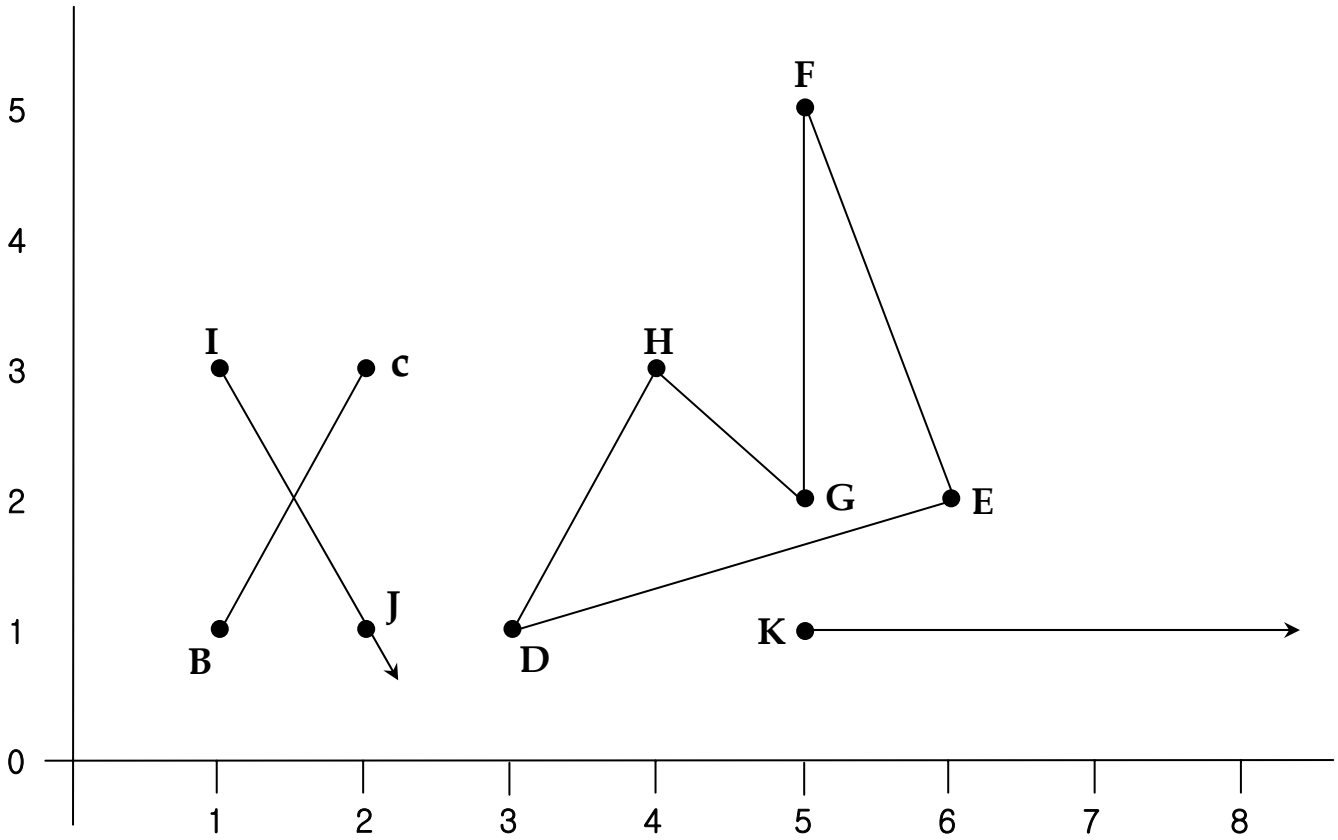
```
struct Line {
    point  p1;
    point  p2;
  };
Line line;
```

기하 요소 표현

- 다각형

  점의 집합으로 표현되며, 이웃한 점들은 직선으로 연결되어 있고 처음 점과 끝점도 직선으로 연결된 닫힌 모양의 도형으로 가정

```
point  Polygon[n];
```



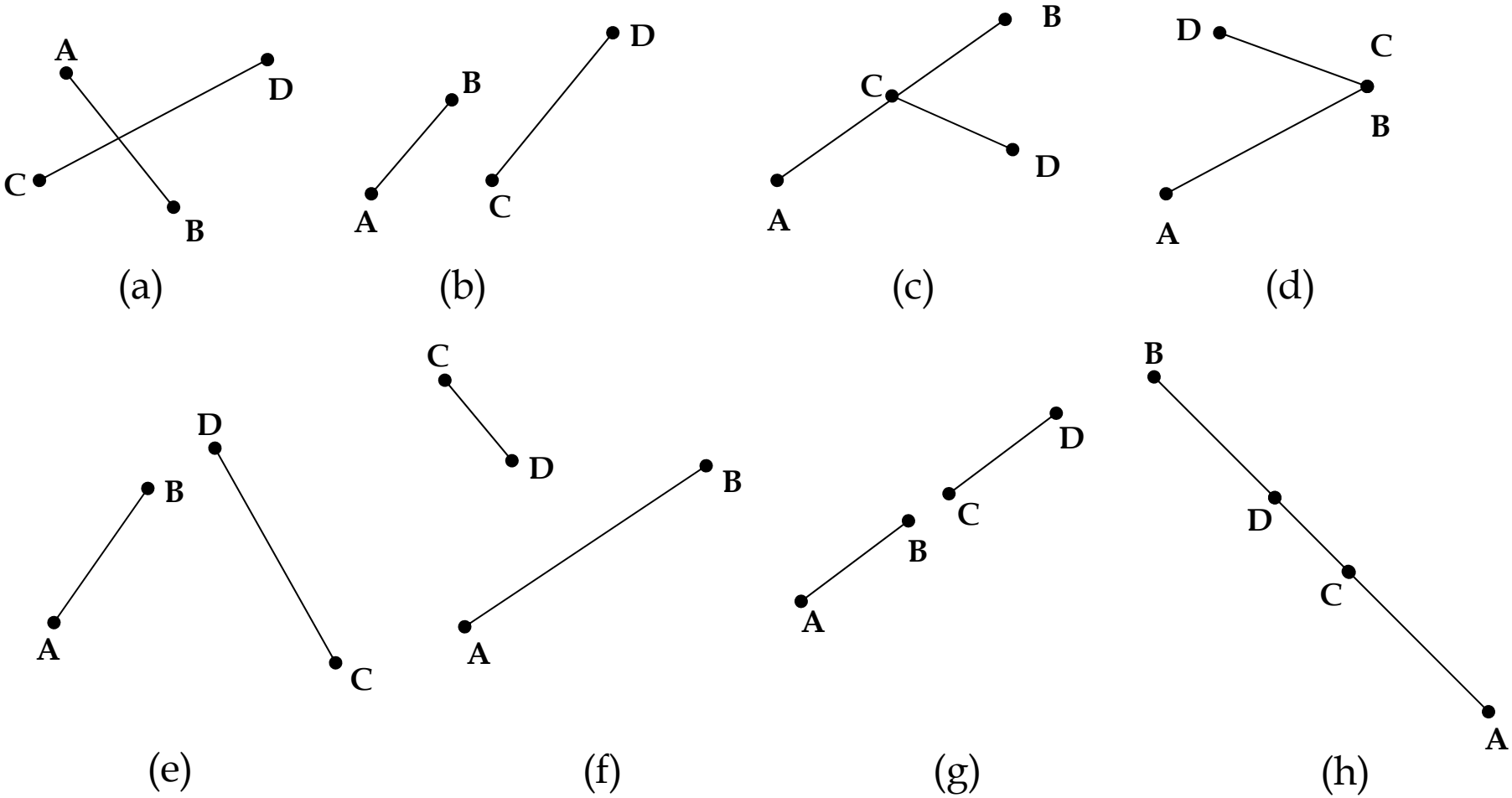
직교 좌표 평면 위에 놓인 기하 요소들의 예

두 선분의 교차 검사

- 두 선분이 주어졌을 때 이들이 서로 교차하는지 그렇지 않은지를 검사

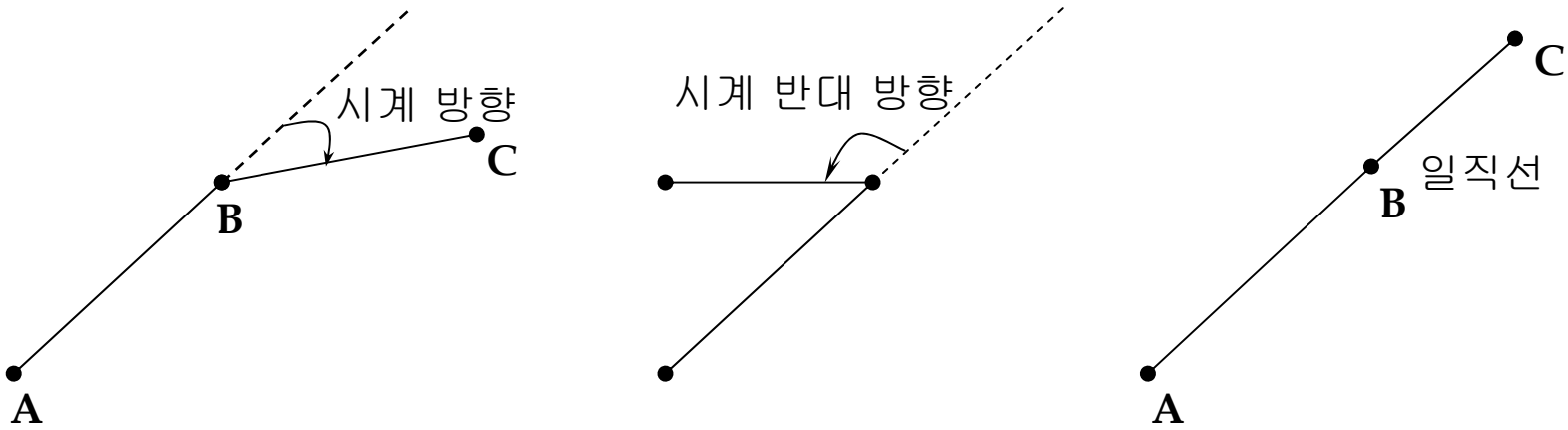
- 선분 AB, CD가 교차할 조건
  - 점 C와 D가 선분 AB를 기준으로 서로 반대편에 있을 때
  - 한 선분의 끝점이 다른 선분상에 있을 때
  - 두 선분이 일직선상에 있을 때

두 선분의 교차 검사



평면 위에 있는 두 선분의 상대 위치

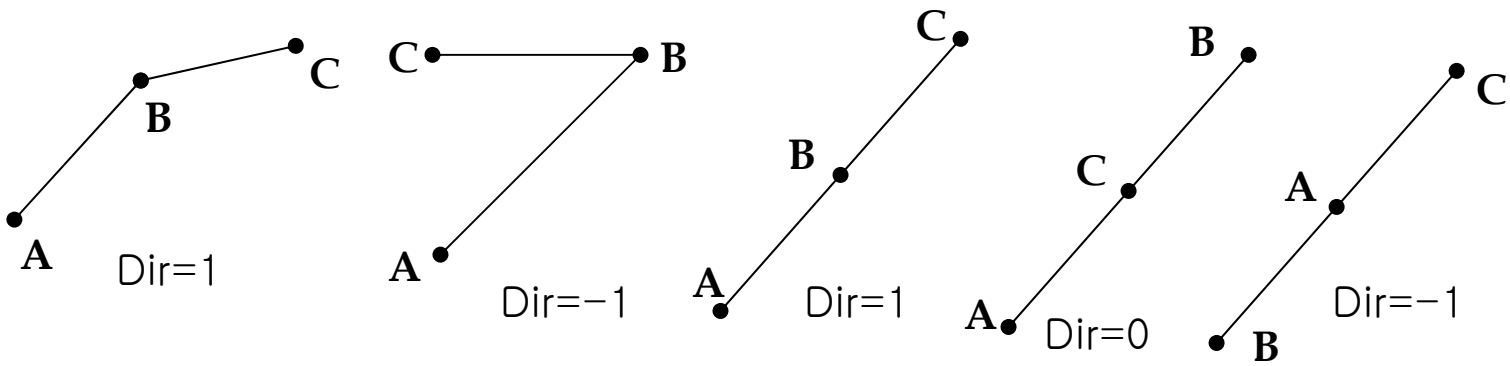
두 선분의 교차 검사



(a) (b) (c)

꺾은선 ABC의 꺾이는 방향

두 선분의 교차 검사



(a) (b) (c)

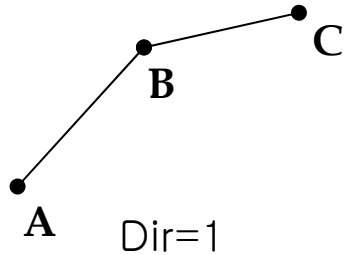
꺾은선 ABC의 꺾이는 방향

Direction(A,B,C)

(1) A, B, C가 일직선상에 있지 않을 때, 꺾은선 ABC의 방향이

① 시계 방향인 경우 : 1

② 시계 반대 방향인 경우 : -1

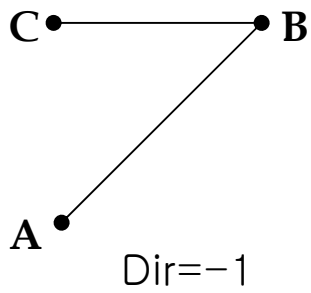


(2) 점 A, B, C가 일직선상에 있을 때,

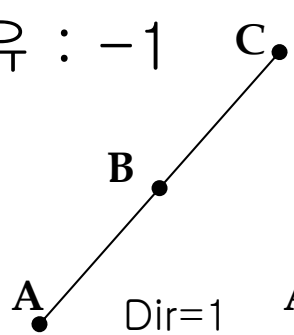
① C가 가운데 있는 경우 : 0

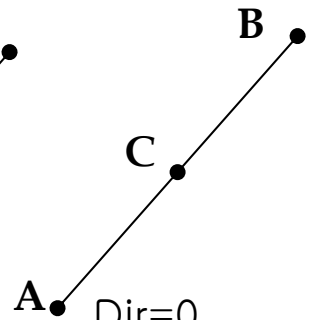
② B가 가운데 있는 경유 : 1

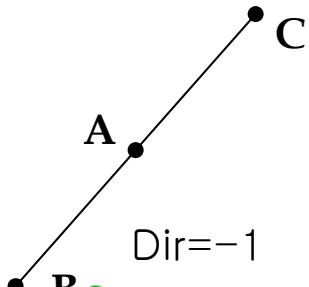
③ A가 가운데 있는 경유 : -1



함수 Direction이 복귀시키는 값



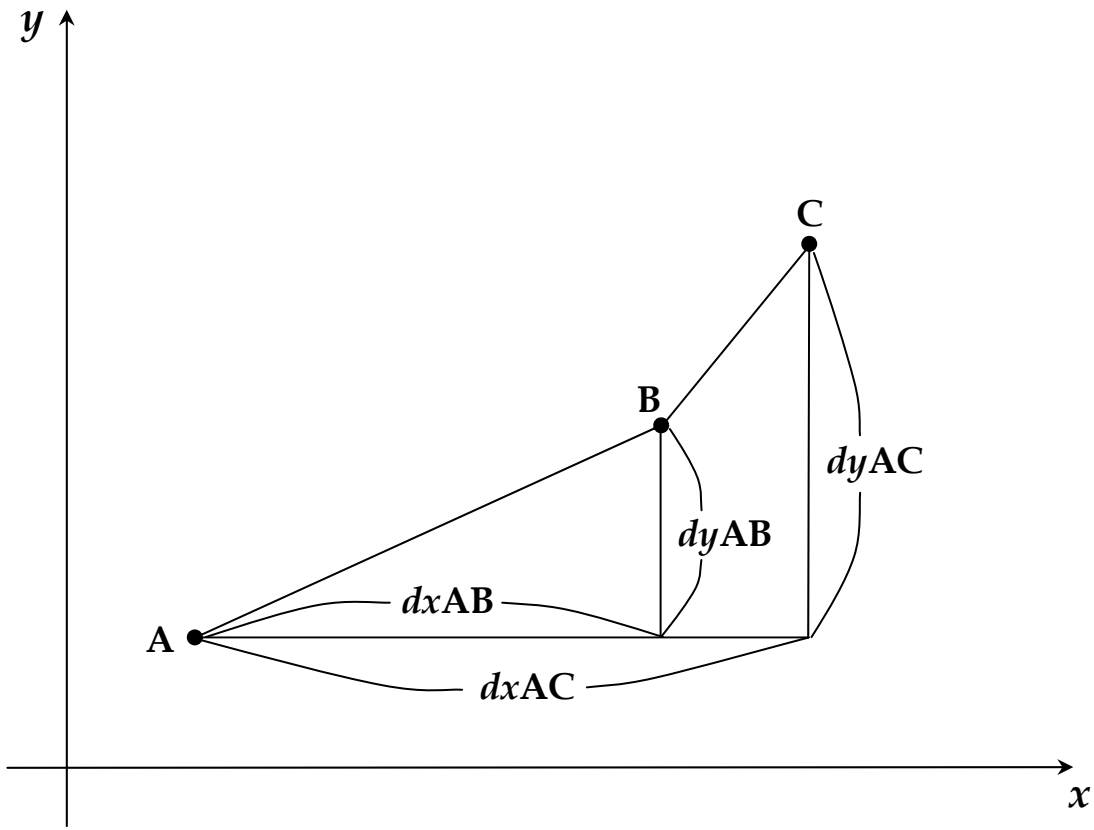




```
int Intersection (line AB,  line CD)  {
    // 입력 : AB, CD : 교차 상태를 검사할 두 선분
    //출력 : AB, CD가 서로 교차하면 TRUE, 아니면 FALSE.
  int LineCrossing;

  if ( (Direction (AB.A,  AB.B,  CD.C)  *
        Direction (AB.A,  AB.B,  CD.D) <= 0)  &&
       (Direction (CD.C,  CD.D,  AB.A) *
        Direction (CD.C,  CD.D,  AB.B) <= 0) )
        LineCrossing = TRUE;
  else LineCrossing = FALSE;
  return LineCrossing;
}
```

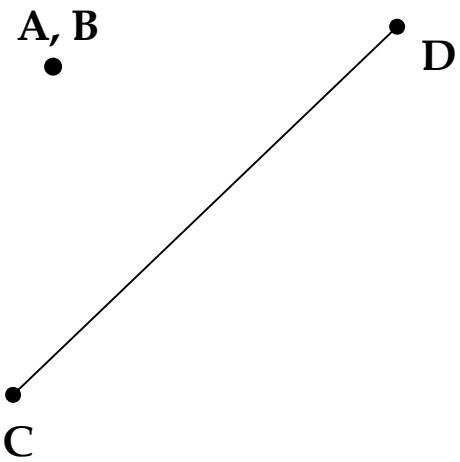
두 선분의 교차 검사 알고리즘

선분의 기울기

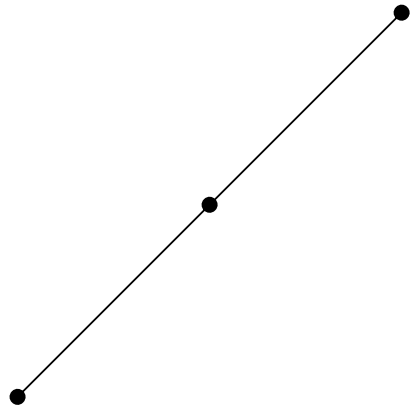


```
int Direction (point A, point B, point C) {
  //입력 : A,B, C : 세 점의 좌표
  //출력 : Dir : <그림 6-4>에서 정의된 값
int dxAB, dxAC, dyAB, dyAC, Dir;
1       dxAB = B.x – A.x;
2       dyAB = B.y – A.y;
3       dxAC = C.x – A.x;
4       dyAC = C.y – A.x;
5       if (dxAB * dyAC < dyAB * dxAC)  Dir = 1;  /* 시계 방향 */
6       if (dxAB * dyAC > dyAB * dxAC)  Dir = -1; /* 반시계 */
7       if ( (dxAB * dyAC == dyAB * dxAC) )  { /* 일직선 */
8           if (dxAB == 0  && dyAB == 0)  Dir = 0;
9           if ( (dxAB * dxAC < 0) || (dyAB * dyAC < 0) ) Dir = -1;
10          else if ( (dxAB * dxAB + dyAB * dyAB) ) >=
                    (dxAC * dxAC + dyAC * dyAC) )
11              Dir = 0;
12          else Dir = 1;
13      }
14      return Dir;
    }
```

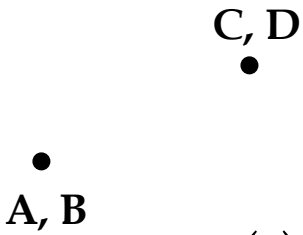
세 점의 상대 위치를 결정하는 알고리즘



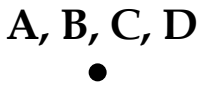
(a)



(b)



(c)

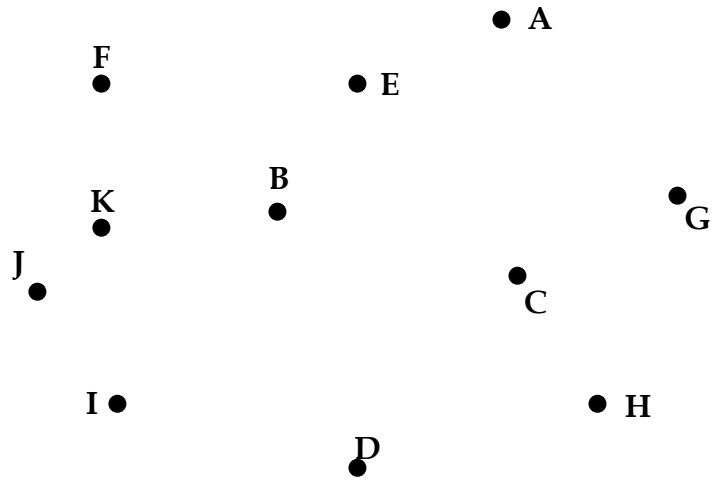


(d)

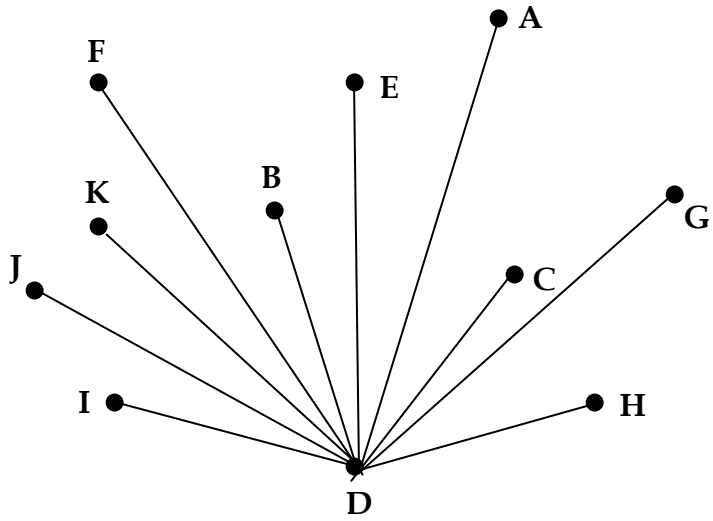
선분이 점이 되는 경우

단순 폐쇄 경로

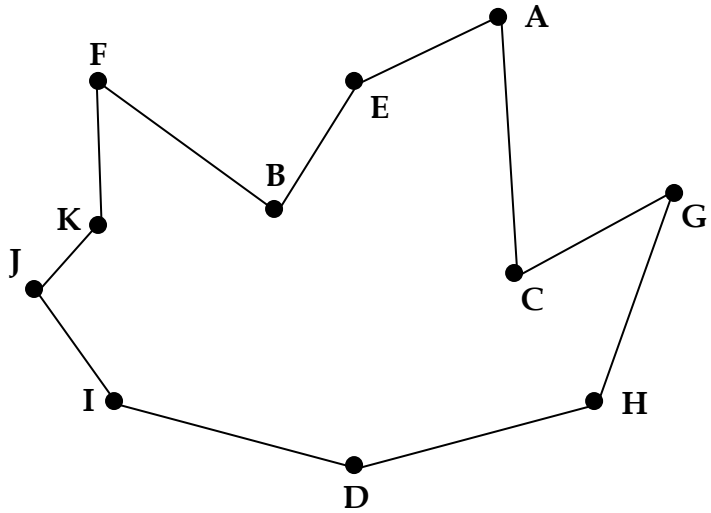
- N개의 점이 주어졌을 때 이 점들을 모두 경유하고 출발점에 되돌아오는 비교차 경로.
- 단순 다각형: 단순 폐쇄 경로에 의해 만들어지는 다각형
- 방법: 기준점과의 각도 계산과 정렬
  - 기준점에 따라 단순 폐쇄 경로도 다르게 됨



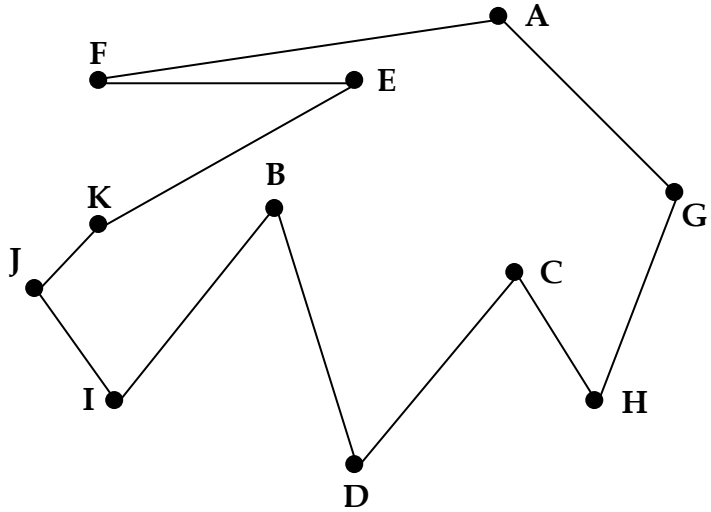
(a)



(b)



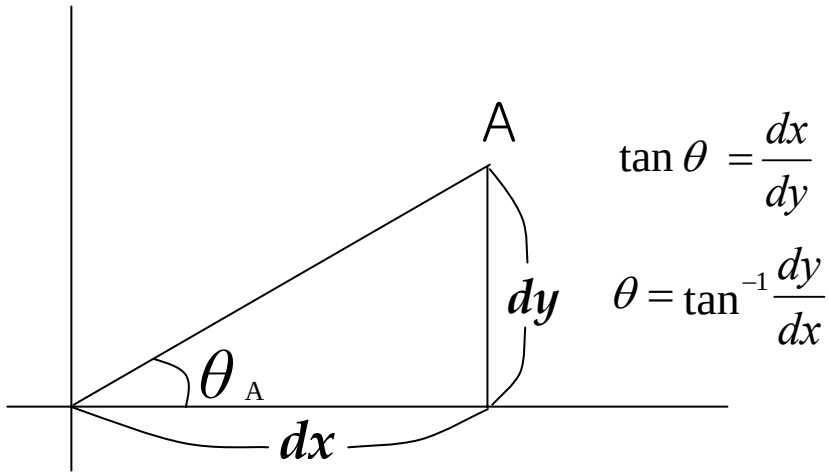
(c)



(d)

점집합의 단순 다각형 구하는 예

점의 각도 계산

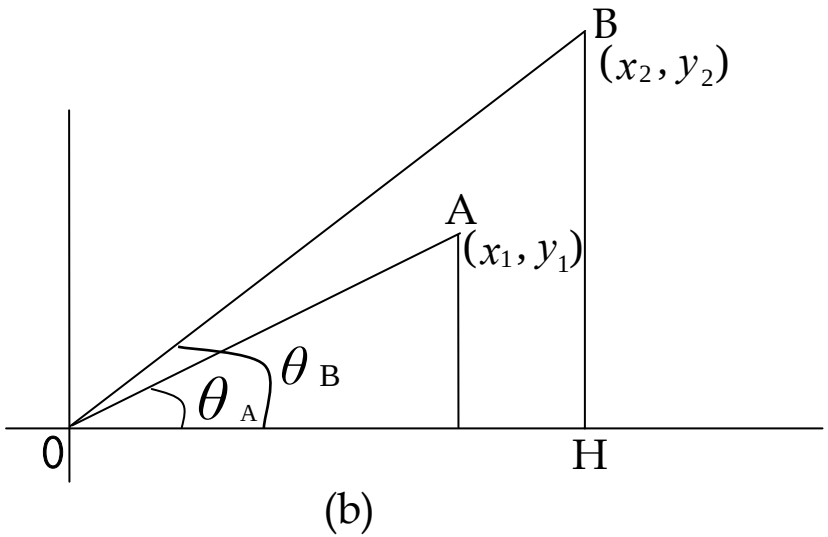


(a)

√ $\tan^{-1}$함수 호출에 많이 시간이 소요됨
√ 정확한 각도는 구할 필요가 없음

점의 각도 계산

- 상대 각도
  - √ 두 점 A, B의 실제 각도 $\theta_A \theta_B$, 상대 각도 T_A, T_B
  - √ $\theta_A \leq \theta_B$ 이면 $T_A \leq T_B$



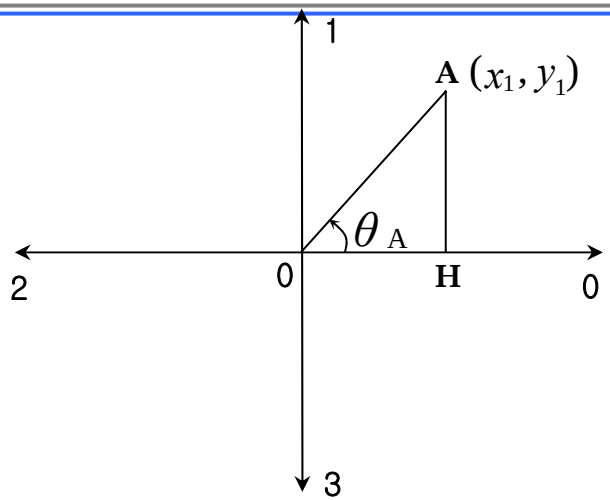
(b)

$$\tan\theta_A < \tan\theta_B$$

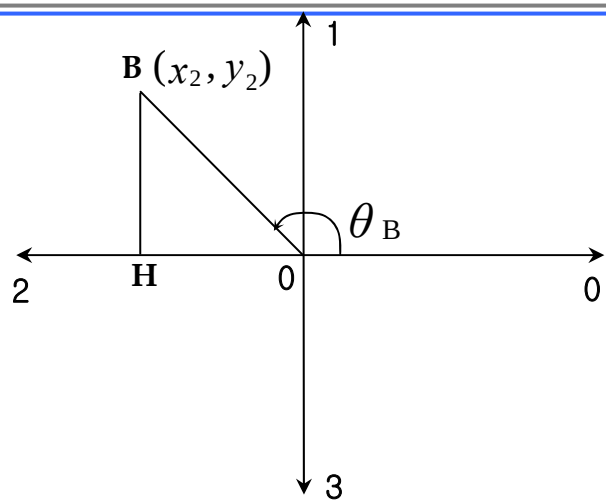
$$\rightarrow \frac{dy_1}{dx_1} < \frac{dy_2}{dx_2}$$

$$\rightarrow \frac{dy_1}{dx_1 + dy_1} < \frac{dy_2}{dx_2 + dy_2}$$

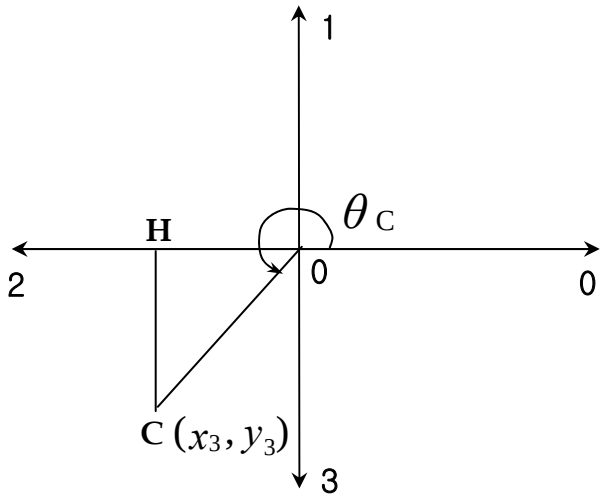
$0 \leq T_A \leq 1$ $1 \leq T_A \leq 2$ $2 \leq T_A \leq 3$ $3 \leq T_A \leq 4$



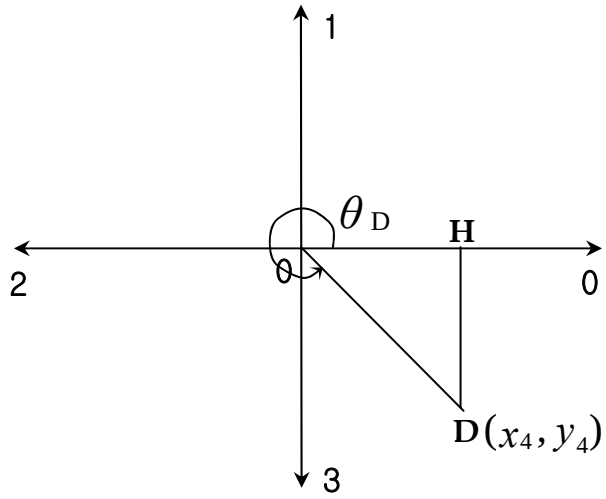
(a)T_A=dy1/(dx1+dy1)



(b)T_B=2-dy2/(|dx1|+dy2)



(c)T_C=2+|dy3|/(|dx3|+|dy3|)



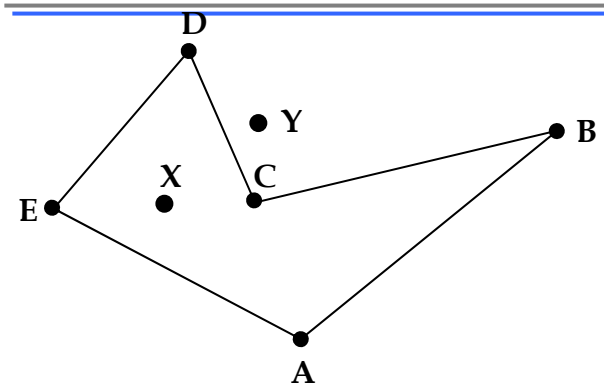
(d) T_D=4-|dy3|/(|dx3|+|dy3|)

```
Float ComputeAngle (point A,   point B) {
    // 입력 : A, B : 두 점
    //     출력 : Angle = T_B * 90.0, 즉 선분 AB와 반직선 Ax가 이루는 상대 각도.
    //              Angle의 값은 0.0~360.0사이이다
        int Dx,  Dy;
        float Angle;
1       Dx = B.x – A.x;
2       Dy = B.y – A.y;
3       if ( (Dx >= 0) && (Dy == 0) ) Angle = 0;                    /* Ax상에 있는 점*/
4       else  {
5               Angle = abs (Dy) / (abs (Dx) + abs (Dy) );          /* 1사분면의 점 */
6               if ( (Dx < 0) && (Dy >= 0) ) Angle = 2-Angle;       /* 2사분면의 점 */
7               else if ( (Dx <=0) && (Dy < 0) ) Angle = 2 + Angle  /* 3사분면의 점 */
8               else if ( (Dx > 0) && (Dy < 0) ) Angle = 4 – Angle; /* 4사분면의 점 */
9       }
10      return ( Angle * 90.0 );
    }
```

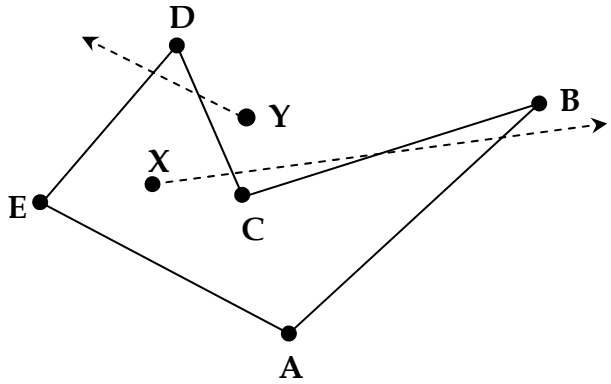
점의 상대 각도를 구하는 알고리즘

점과 다각형의 상대 위치 검사

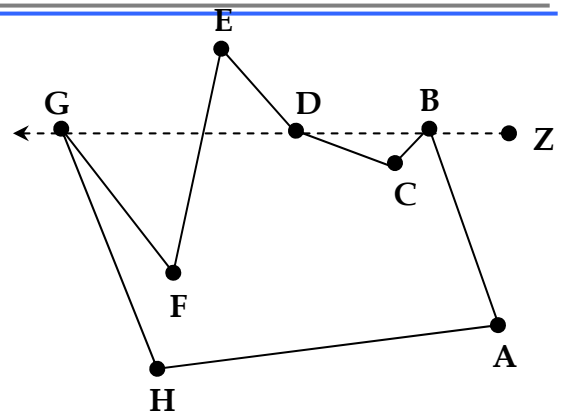
- 어떤 점의 위치가 다각형의 내부인지 외부인지를 결정하는 문제
- 점에서 그은 반직선이 다각형의 변과 교차하는 개수를 조사
  - 꼭지점과 교차하는 경우



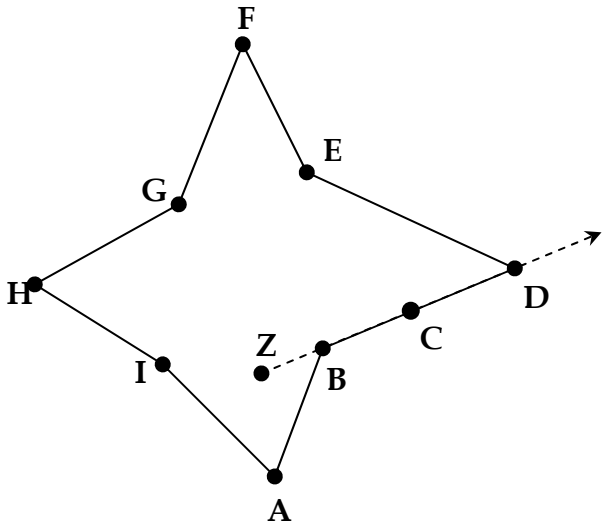
(a)



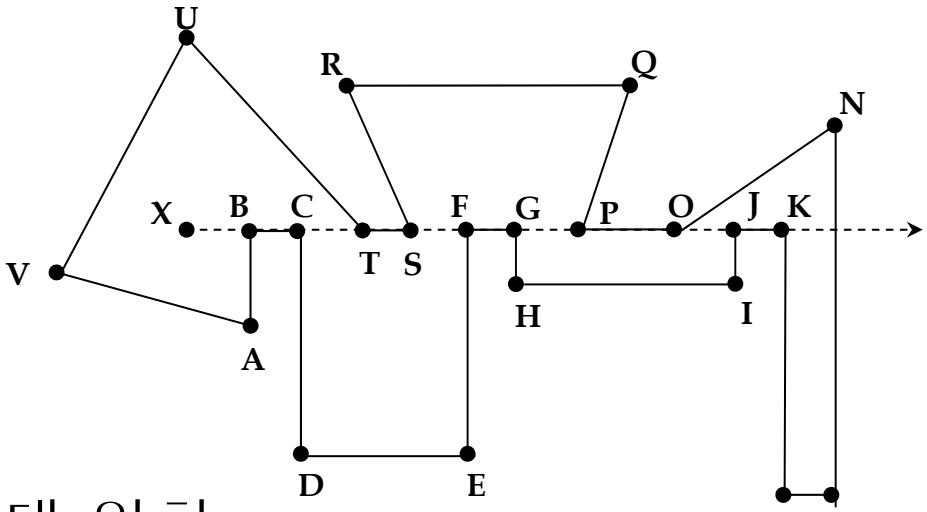
(b)



(c)



(d)



(e)

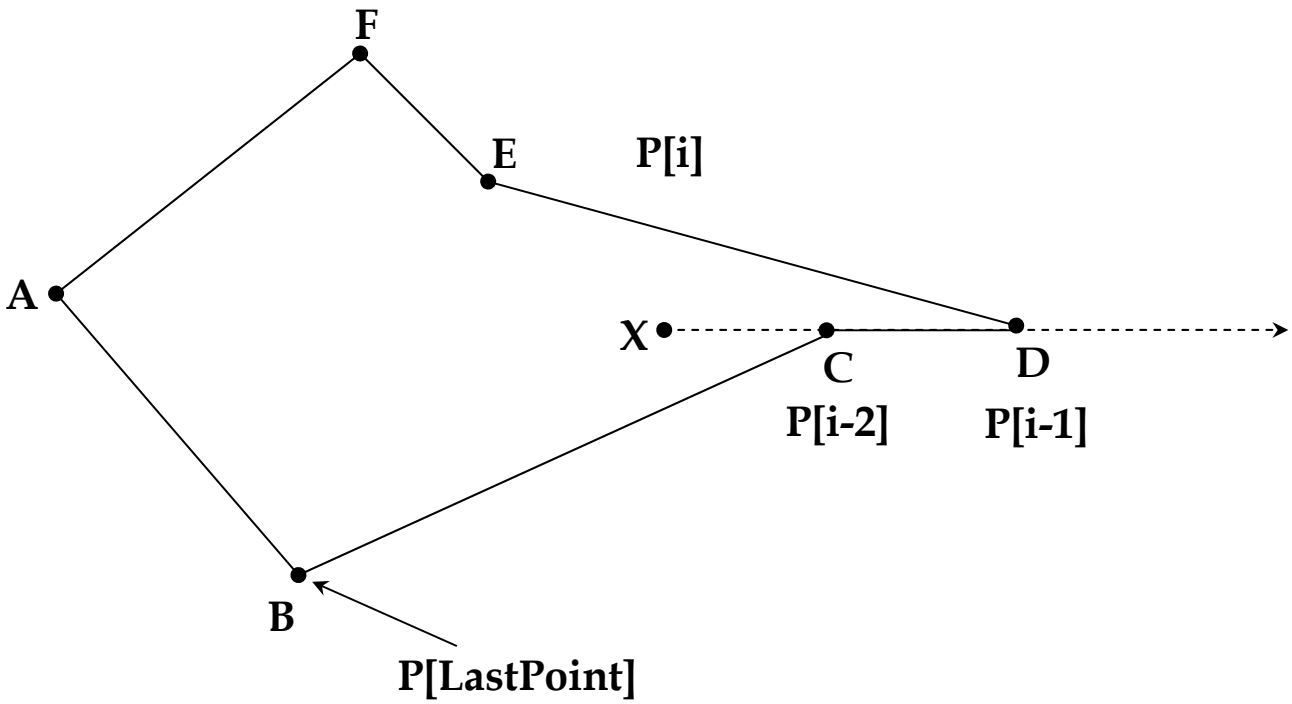
점과 다각형의 상대 위치

```
int IsPointInside ( point A, point P[ ], int n) {
/* 입력 : A : 점, P[0 : n] : 다각형, n : 다각형의 꼭지점의 수.
   가정 : P[n] = P[n]이다. 점 A는 다각형의 꼭지점이 아니다.  P[0].y ≠ A.y
   출력 : TRUE : 점이 다각형의 내부에 있거나 다각형의 한 변 위에 있을 때
          FALSE : 점이 다각형의 외부에 있을 때 */

    int  Count, i, LastPoint;
    line TestLine, PolygonLine;
    int PointOnTestLine;

1    Count = LastPoint = 0;  PointOnTestLine = FALSE;
2    TestLine.p1 = A;  TestLine.p2 = A;  Testline.p2.x = MAXINT;
3    for (i = 1;  i <= n;  i++)  {
4        PolygonLine.p1 = PolygonLine.p2 = P[i];
5        if (Intersection (TestLine,  PolygonLine) )
6            pointOnTestLine = TRUE;
```

```
7     else {
8           polygonLine.p2 = P[LastPoint];
9           LastPoint = i;
            if ( !PointOnTestLine ) {
11              if (Intersection (PolygonLine,  TestLine) );
12                  Count++;
13          } else {
14                  if (Direction (TestLine.p1,  TestLine.p2,  PolygonLine.p1) *
15                      Direction (TestLine.p1,  TestLine.p2,  PolygonLine.p2) < 0)
16                              Count++;
17                  PointOnTestLine = FALSE;
18          }
19      }
20  } //end of for
21  return( (Count mod 2) == 1 );
    }
```

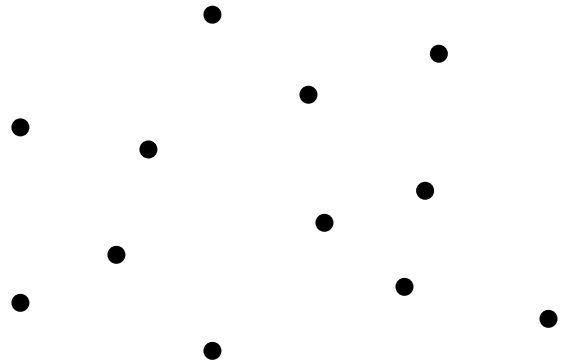
점이 다각형의 내부에 있는가를 검사하는 알고리즘

다각형의 변이 검사선을 통과하는 경우

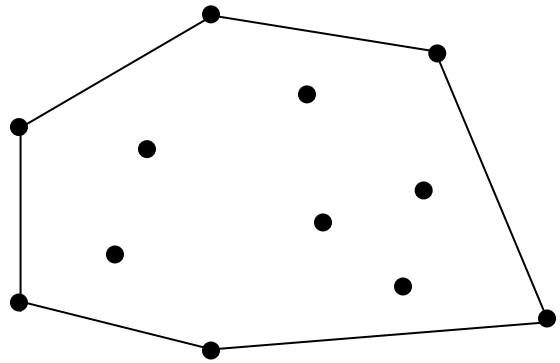


볼록 껍질 찾기

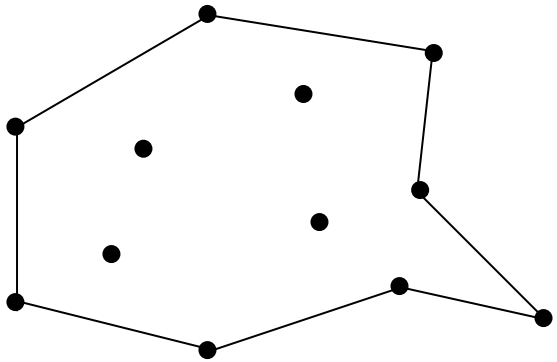
- 볼록 다각형(convex hull): 다각형 내부의 임의의 2점을 연결하는 선분이 반드시 다각형 내부에 존재하는 다각형.
- 볼록 껍질: 점집합의 모든 점을 포함하는 최소의 볼록 다각형. 즉 볼록 껍질은 점들을 모두 둘러싸는 최소길이 경로이다.
- 볼록 껍질 외부의 임의의 직선을 껍질 쪽으로 접근시킬 때 이 직선이 처음으로 만나는 점은 볼록 껍질의 꼭지점이다.



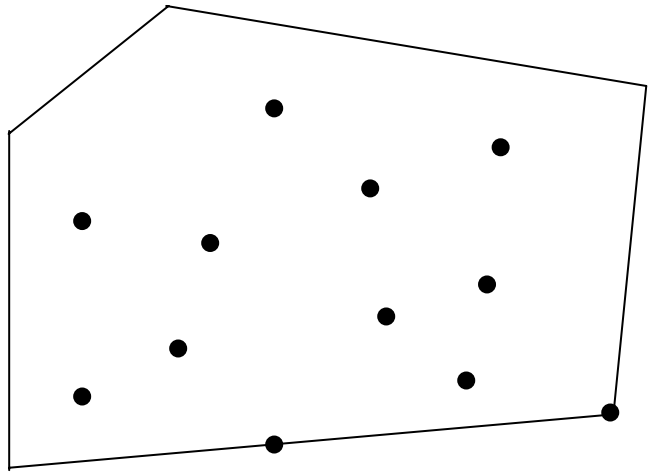
(a)



(b)



(c)

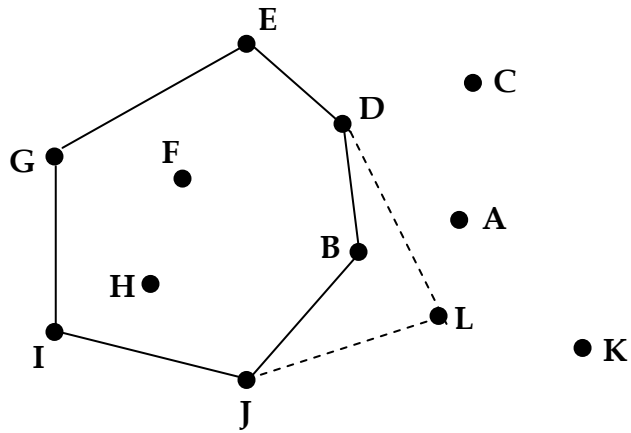


(d)

점 집합과 볼록 껍질

단순한 방법

- 점을 하나씩 추가해 나가면서 구하는 방법
- 최소각의 점과 최대각의 점을 구하고 이 두 점 사이의 점들은 제거하고 이 두 점과 새점을 연결한다.



볼록 껍질의 확장

단순한 방법의 분석

- 초기 정렬

  $O(n \log n)$

- 확장

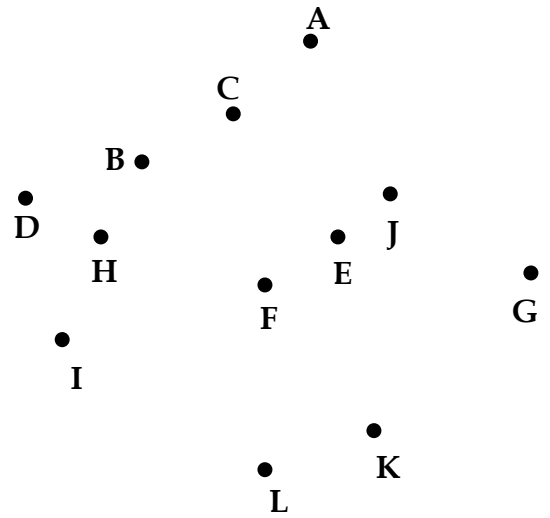
  $T(n)=T(n-1)+O(n),\ T(1)=O(1)$

  $T(n)=O(n^2)$

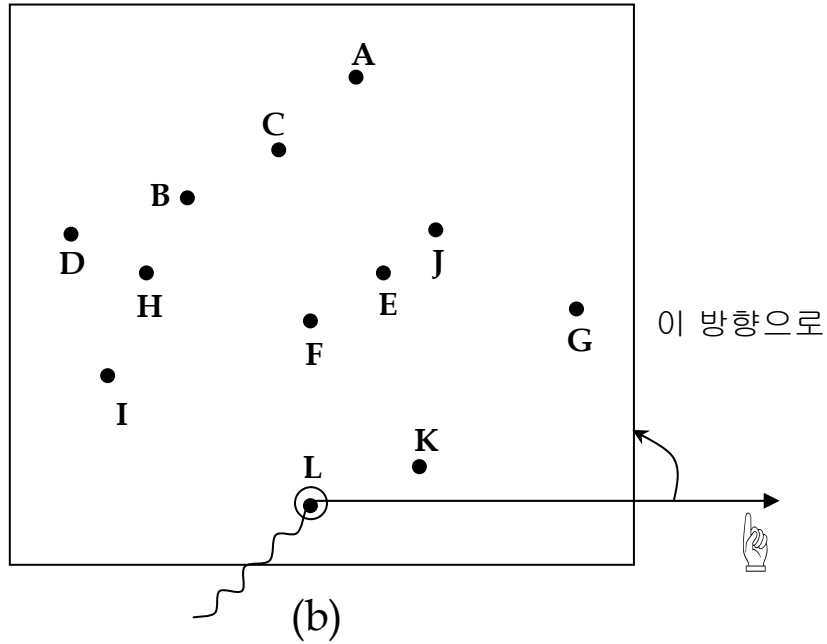
짐꾸리기 알고리즘

- 다양한 이름으로 불림

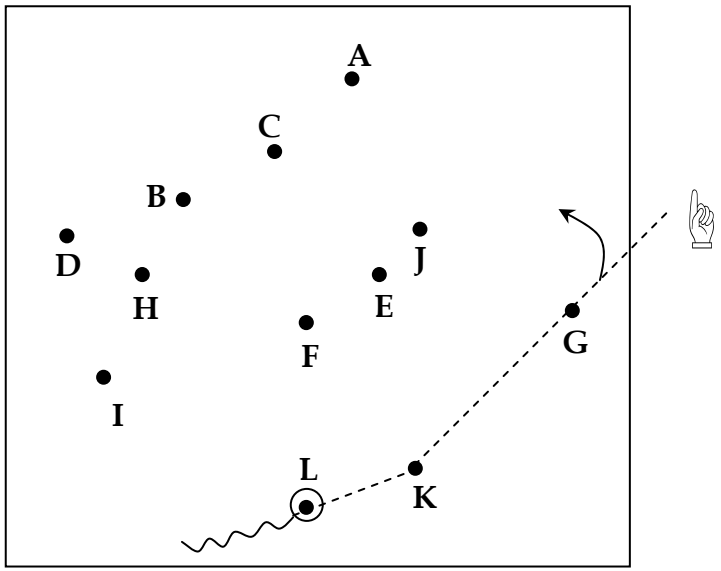
  package wrapping, gift wrapping, Jarvis' march
- 껍질 외부의 임의의 직선을 껍질 쪽으로 접근시킬 때 이 직선이 만나는 것은 볼록 껍질의 한 꼭지점이나 한 변.



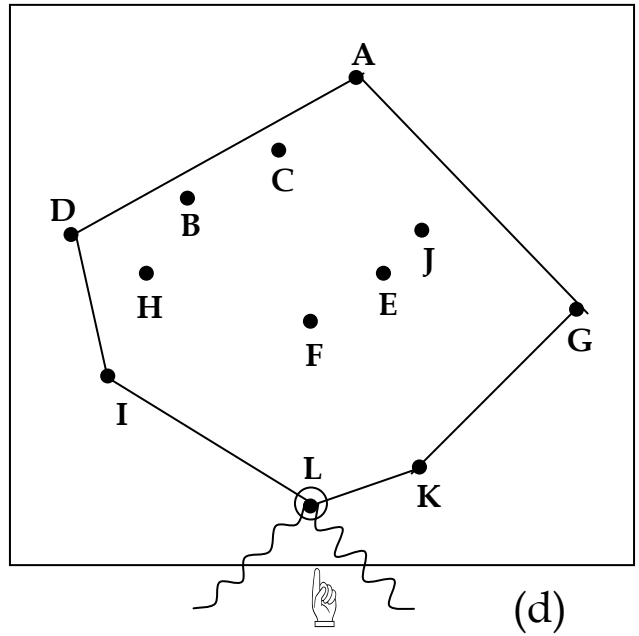
(a)



(b)



(c)



(d)

짐꾸리기 알고리즘의 적용 예

인덱스 / 단계	0	1	2	3	4	5	6	7	8	9	10	11	12
0		B	C	D	E	F	G	H	I	J	K	L	
1	(L)	B	C	D	E	F	G	H	I	J	K	A	L
2	(L)	(K)	C	D	E	F	G	H	I	J	B	A	L
3	(L)	(K)	(G)	D	E	F	C	H	I	J	B	A	L
4	(L)	(K)	(G)	(A)	E	F	C	H	I	J	B	D	L
5	(L)	(K)	(G)	(A)	(D)	F	C	H	I	J	B	E	L
6	(L)	(K)	(G)	(A)	(D)	(I)	C	H	F	J	B	E	L

12개의 점으로 구성된 점집합의 볼록 껍질 구하는 예

짐꾸리기 알고리즘

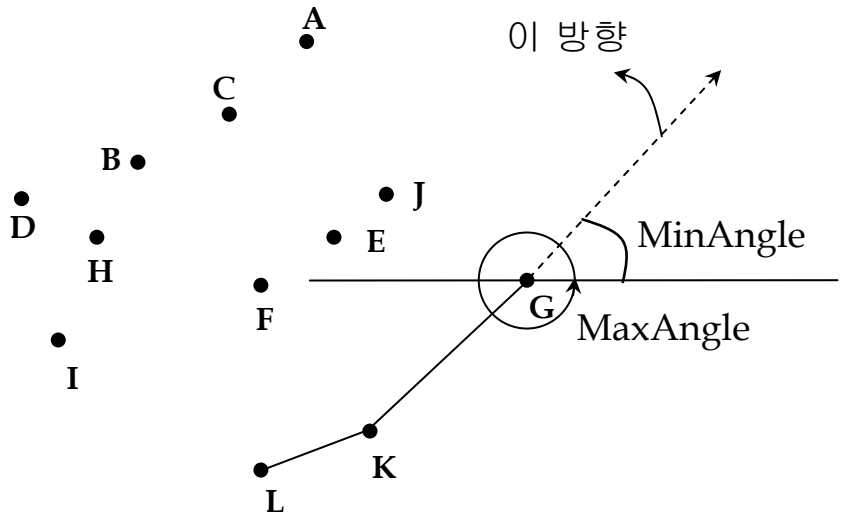
- 양의 x축 방향과 이루는 각도를 계산하여 최소의 각을 갖는 점을 다음의 꼭지점 및 다음의 시작점으로 취한다.

1. Y 좌표가 최소인 점을 최초의 꼭지점으로 선택한다.
2. 기준점으로부터 아직 선택이 안된 모든 점들에 대한 각도를 계산한다.
3. 최소각을 갖는 점을 다음의 볼록 껍질의 꼭지점으로 선택한다.
4. 지금 선택된 꼭지점이 최초의 기준점이면 계산 끝, 아니면 그 점을 기준점으로 하여 단계 2부터 반복한다.

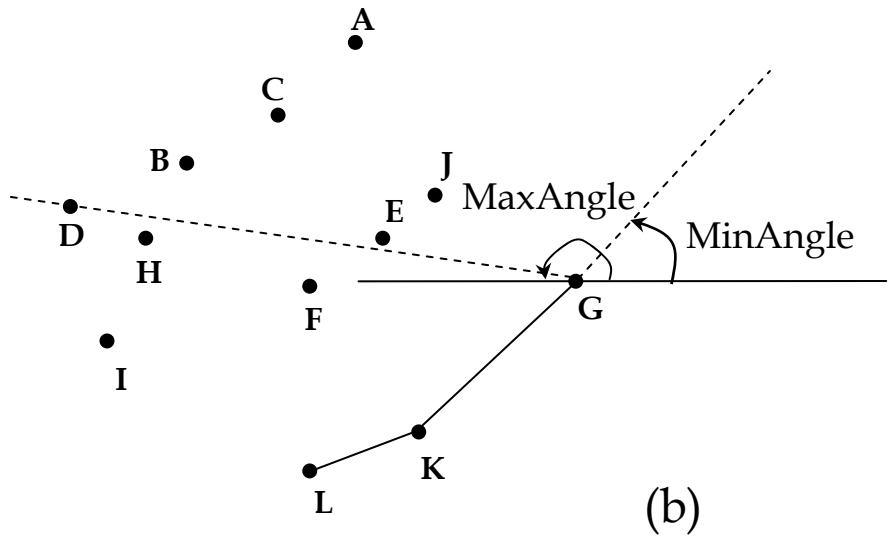
```
int  PointWrapping (struct point P[ ], int n)  {
    /* 입력 :  n : 점집합 대의 점의 개수.
                P[0 : n] : n개의 점은 P[0]   P[n- 1]에 저장되어 있음.
       출력 : P[0 : NumVertex - 1]에 볼록 껍질의 꼭지점들이 순서대로 저장됨. */

    int i,  NumVertex
    float MinAngle,  MaxAngle,  Angle;
    int FirstPoint,  NextPoint;
1   FirstPoint = 0;

2   for (i = 1;  i < n;  i++)            /* 볼록 껍질의 첫째 꼭지점 찾기 */
3       if (P[i].y < P[FirstPoint].y)  FristPoint = i;
4   NumVertex = -1;
5   P[n] = P[FirstPoint];                /* 첫째 꼭지점의 복사본을 P[n]에 저장 */
6   MaxAngle = 0.0;
7   NextPoint = FirstPoint;
```

```
8    do {                              /* 다음 꼭지점을 찾는 루프 */
9          NumVertex++;                /* 다음 꼭지점을 배열의 왼쪽으로 밀어둠 */
10         Swap (&P[NumVertex], &P[NextPoint]);
11         NextPoint = n;
12         MinAngle = MaxAngle;
13         MaxAngle = 360.0;
14         for (i = NumVertex + 1; i <= n; i++) {
15             Angle = ComputeAngle (P[NumVertex], P[i])
16             if (Angle > MinAngle && Angle < MaxAngle ) {
17                 NextPoint = i;
18                 MaxAngle = Angle;
19             }
20         }
21    } while (NextPoint == n)
22    return NumVertex;
   }
```

짐꾸리기 알고리즘



(a)



(b)

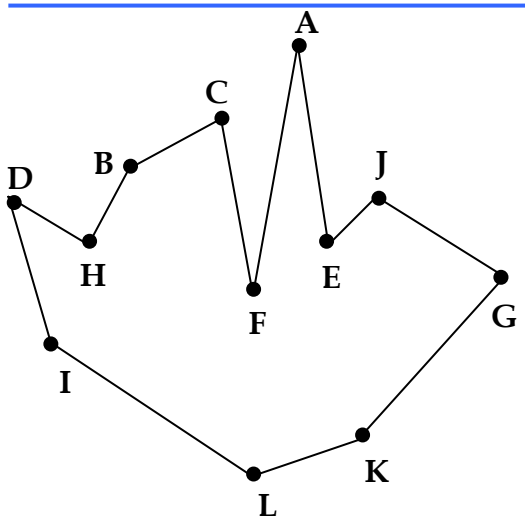
여러 꼭지점이 일직선상에 있는 경우

짐꾸리기 알고리즘 분석

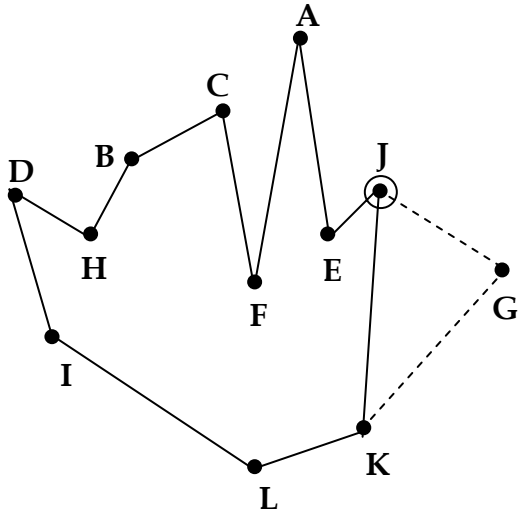
- 모든 점들이 볼록 껍질의 점으로 되는 경우가 최악의 경우이며 이때 수행시간은 $O(n^2)$

Graham 알고리즘

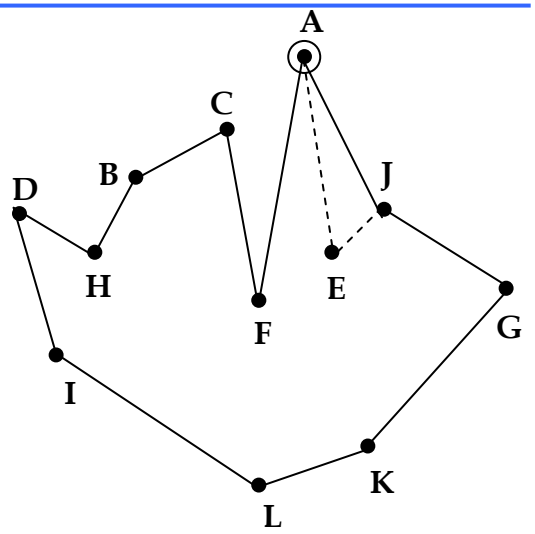
- 단순 폐쇄 경로를 구한 후 볼록 껍질의 꼭지점이 될 수 없는 것을 제거해 나가는 방법
  - 점 제거 기준



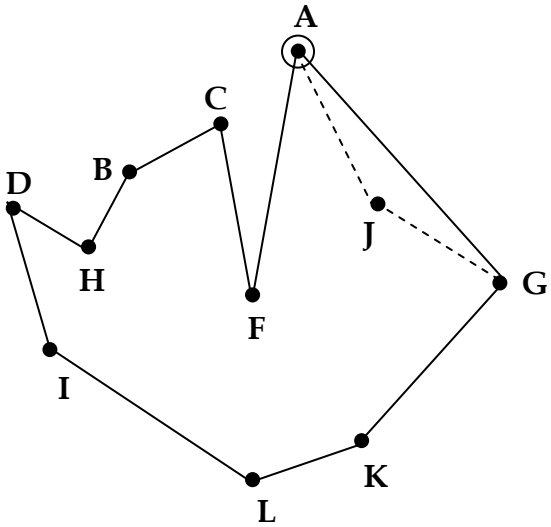
(a)



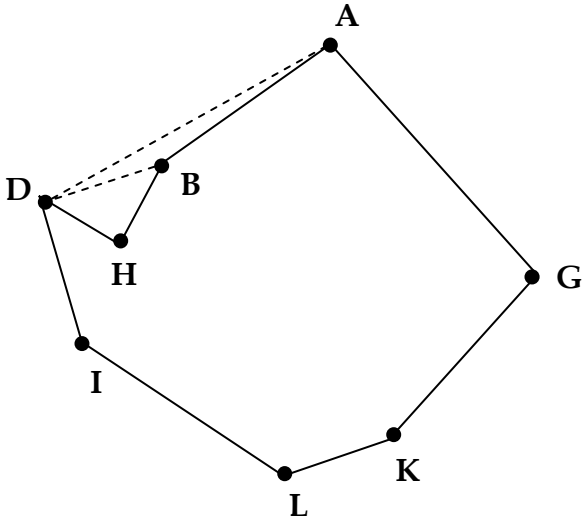
(b)



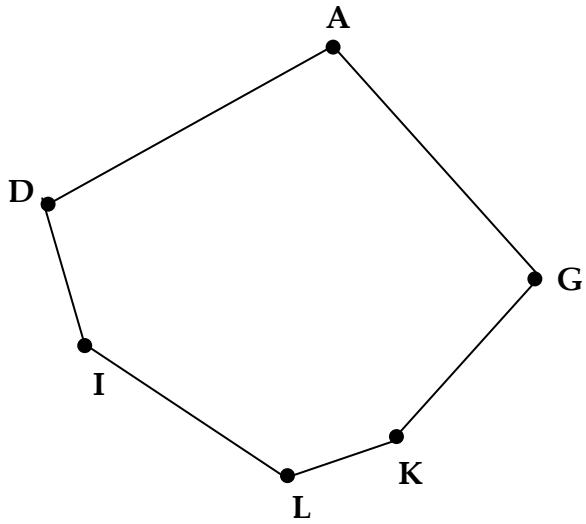
(c)



(d)



(e)



(f)

그레이엄 알고리즘의 적용 예

Graham 알고리즘의 근거

1. 점집합의 모든 점들을 조사한다.
2. 볼록 다각형의 꼭지점들을 어떤 기준점으로부터 반시계방향으로 따라나가면 항상 반시계 방향으로 꺾인다.
3. 단순 폐쇄 경로를 따라나가는 도중에 시계방향으로 꺾이어서 제거되는 점들은 그때까지 만들어진 볼록 다각형의 내부에 존재하는 점이다.

```
int GrahamScan (struct point P[ ], int n)  {
  /* 입력 : n : 점집합 내의 점의 개수, P[0 : n] : n개의 점은 P[1] …P[n]에 저장되어 있음.
     출력 : P[1 : NextPoint]에 볼록 껍질의 꼭지점들이 순서대로 저장됨. */
   int i, FirstPoint, NextPoint ;
1    FirstPoint = 1;
2    for (i = 2;  i <= n;  i++)                          /*  y좌표가 최소인 점 찾기  */
3        if (P[i].y < P[FirstPoint].y)  FirstPoint = i;
4    for (i = 1;  i < n;  i++)                           /*  x, y좌표가 최소인 점 찾기  */
5        if (P[I].y == P[FirstPoint].y  &&  P[i].x > P[FirstPoint].x)   FirstPoint = i;
6    Swap (&P[FirstPoint], &P[1]);
7    FindSimplePolygon (P, n);                  /* 단순 다각형 구하기 */
8    P[0] = p[n] ;
9    NextPoint = 3;
10   for (i = 4;  i <= n;  i++)  {              /* 볼록 껍질의 꼭지점이 될 수 없는 점 제거 */
11       while (Direction (P[NextPoint – 1], P[NextPoint], p[i]) >= 0)
12           NextPoint-- ;
13       NextPoint++;
14       Swap (&P[NextPoint], &P[i]);
     }
   return NextPoint ;                           /* 볼록 껍질의 꼭지점의 수를 복귀 */
}
```

그레이엄의 볼록 껍질 찾는 알고리즘

단계	0	1	2	3	4	5	6	7	8	9	10	11	12
0	I	L	K	G	J	E	A	F	C	B	H	D	I
1	(I)	(L)	(K)	(G)	J	E	A	F	C	B	H	D	I
2	(I)	(L)	(K)	(G)	(J)	E	A	F	C	B	H	D	I
3	(I)	(L)	(K)	(G)	(J)	(E)	A	F	C	B	H	D	I
4	(I)	(L)	(K)	(G)	(A)	△E	△J	F	C	B	H	D	I
5	(I)	(L)	(K)	(G)	(A)	(F)	△J	△E	C	B	H	D	I
6	(I)	(L)	(K)	(G)	(A)	(C)	△J	△E	△F	B	H	D	I
7	(I)	(L)	(K)	(G)	(A)	(B)	△J	△E	△F	△C	H	D	I
8	(I)	(L)	(K)	(G)	(A)	(B)	(H)	△E	△F	△C	△J	D	I
9	(I)	(L)	(K)	(G)	(A)	(D)	△H	△E	△F	△C	△J	△B	I
10	(I)	(L)	(K)	(G)	(A)	(D)	(I)	△E	△F	△C	△J	△B	△H

Graham 알고리즘의 분석

- 단순 폐쇄 경로를 구하는 데 O(n log n) 시간
- 점 제거 – O(n) 시간
  - 줄 11의 while 루프는 바깥쪽 for 루프 전체에 대하여 O(n) 시간 걸리기 때문