Computational Geometry

(CLRS Chapter 33)

김동진
(NHN NEXT)

참고 교재

- Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. *Computational Geometry: Algorithms and Applications*, 3rd Edition.
  - Google 검색하면 PDF 구할 수 있음.

분야 소개

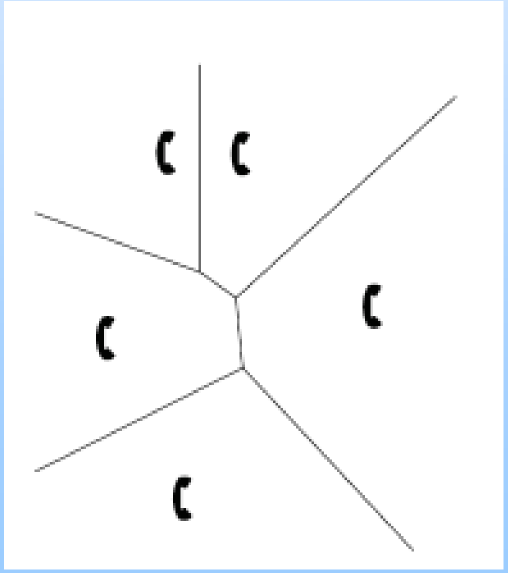
- 기하 문제를 해결하는 알고리즘을 연구하는 분야
- 응용 분야
  - 컴퓨터 그래픽스
  - 로보틱스
  - Geographic information systems
  - 등등

Computer Graphics 응용

- 주어진 상황
  - 2차원 화면에 n개의 도형이 그려져 있다.
  - 각 도형의 꼭지점 위치 정보는 주어져 있다.
- 문제
  - 겹치는 도형 쌍을 추출하라.
- 해법
  - 주어진 도형 형태와 정밀도 등에 따라서 다양한 알고리즘이 존재함.

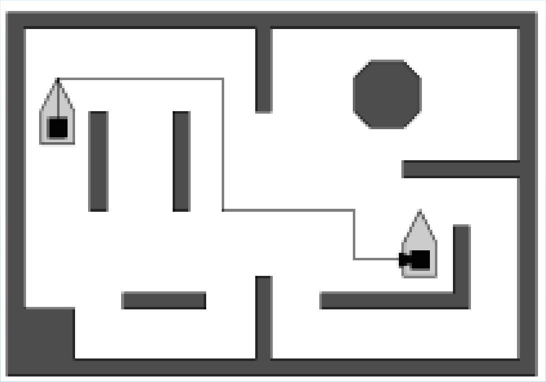
가장 가까운 위치 찾기 문제

- 주어진 상황
  - 특정 지역에서 치킨 체인점 1,000개를 운영하고 있다.
  - 주문은 본점에서 일괄적으로 받는다.
  - 주문이 들어오면 주문한 사람과 가장 가까운 체인점에서 배달하려고 한다.
- 문제
  - 전화 주문이 들어왔을 때 가장 가까운 지점을 찾는 효율적인 알고리즘을 제시하라.
- 해법
  - Voronoi diagram 사용
    - 참고 교재 chapter 7
  - 각 체인이 할당하는 영역을 미리 구축
  - 주문이 들어오면 해당 체인점에서 배송



Robot Motion Planning 응용

- 주어진 상황
  - 로봇이 주어진 지점 s에서 다른 지점 e로 이동하려고 한다.
  - 장애물들이 있어서 두 점을 연결하는 직선 경로로 이동할 수 없다.
  - 지도와 장애물의 위치 정보는 주어져 있다고 가정하자.
- 문제
  - 로봇이 장애물들을 피하면서 목표 지점으로 이동하는 경로를 구하라.



- 해법
  - 참고 교재 chapter 13
  - 빈 공간을 미리 자료구조로 구축
  - BFS 기법을 이용해서 탐색

Geographic Information Systems 응용

- 주어진 상황
  - 위치에 따른 상품 판매 수량을 저장한 지도 정보가 있다.
  - 위치에 따른 이동 인구수를 저장한 지도 정보가 있다.
- 문제
  - 특정 지역의 상품 판매 수량과 이동 인구수를 추출하라.
  - 상품 판매 수량과 이동 인구수의 상관 관계를 구하라.
- 해법
  - 주어지는 영역의 형태에 따라서 다양한 해법이 존재한다.
  - 예를 들어, 영역이 사각형이면 사각형 내의 포함되는 점의 개수 측정 방법 적용

목 차

- Line-segment properties
- 교차하는 선분 존재 여부 검사 알고리즘
- Convex Hull(볼록 다각형) 여부 검사 알고리즘
- The closest pair of points
- 참고 교재 요약

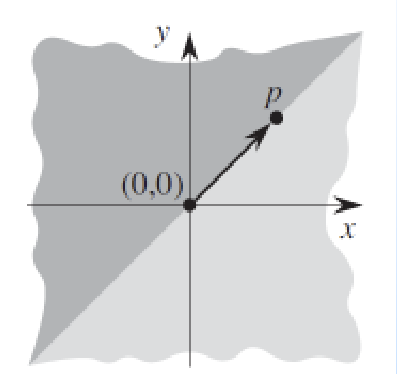
Line-segment Properties (1)

- Line-segment(선분) 표시 방법
  - p1 = (x1, y1), p2 = (x2, y2)이 주어져 있다.
  - 두 점을 연결하는 선분 위의 점 p3 = (x3, y3) 표현 방법
    - $x_3 = \alpha x_1 + (1 - \alpha)x_2 \; for \; 0 \leq \alpha \leq 1$ and
    - $y_3 = \alpha y_1 + (1 - \alpha)y_2 \; for \; 0 \leq \alpha \leq 1$
    - 즉, $p_3 = \alpha p_1 + (1 - \alpha)p_2 \; for \; 0 \leq \alpha \leq 1$

- Directed segment 표현 방법
  - $\overrightarrow{p_1 p_2}$: p_1에서 p_2로 향하는 선분
  - $\overrightarrow{p_2 p_1}$: p_2에서 p_1로 향하는 선분
  - $\overrightarrow{p_1 p_2}$에서 p_1이 원점이라면 $\overrightarrow{p_1 p_2}$는 vector p_2가 된다.

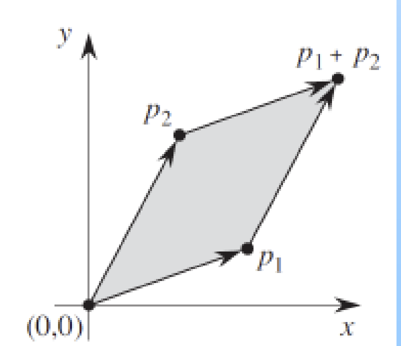
Clockwise 여부 검사

- 문제 설명
  - 두 개의 directed segments $\overrightarrow{p_0p_1}$와 $\overrightarrow{p_0p_2}$가 주어져 있다.
  - $\overrightarrow{p_0p_2}$가 $\overrightarrow{p_0p_1}$로부터 시계방향에 있는지 여부를 검사하라
    - 어두운 부분이 p의 반시계 방향 영역
    - 환한 부분이 시계 방향 영역



(b)

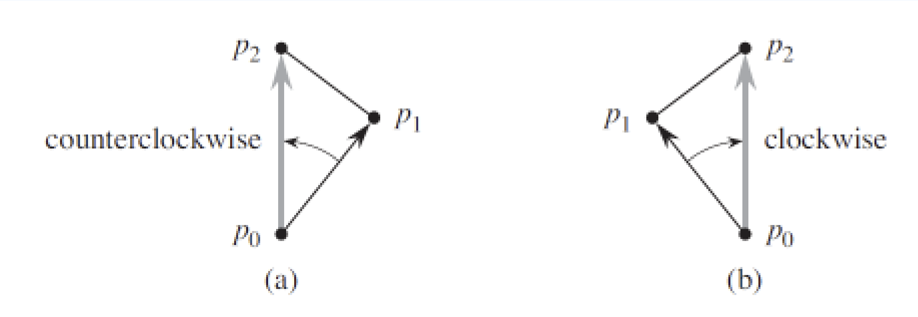
- Solution
  - Cross product
    - $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$일 때
    - $p_1 \times p_2 = \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = -p_1 \times p_2 = |p_1||p_2|sin\theta$
    - signed area를 의미
  - Cross product 결과가 양수이면 반시계방향
  - Cross product 결과가 음수이면 시계방향



(a)

연속된 두 개 선분에서 꺾인 방향

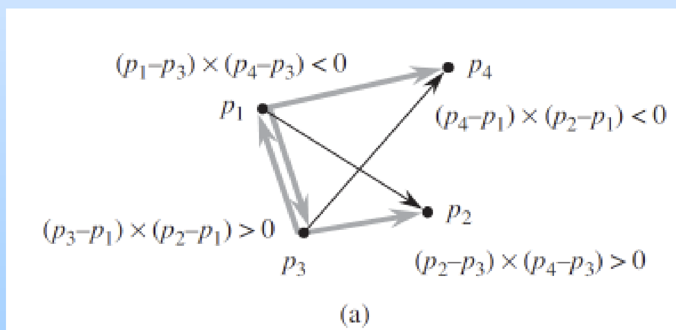
- 문제 설명
  - 두 개의 directed segments $\overrightarrow{p_0p_1}$와 $\overrightarrow{p_1p_2}$가 주어져 있다.
  - 연속된 두 개의 선분이 시계방향으로 방향이 바뀌었는지 검사하라.



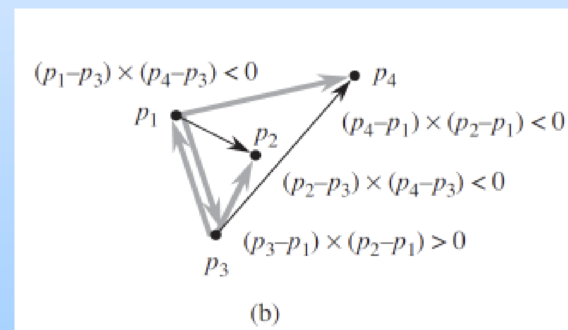
- Solution
  - $(p_1 - p_0) \times (p_2 - p_0)$의 결과로 판단
    - 양수이면 반시계 방향
    - 음수이면 시계 방향

두 개 선분의 교차 여부 검사 방법 (1)

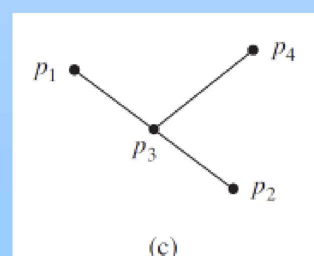
- 아이디어
  - 한 선분의 양 끝점이 다른 선분의 연장선으로 나누어지는 공간에서 서로 반대편에 있는지 검사 (cross product를 이용)
  - 그 반대의 경우도 확인
  - 서로간에 반대편에 있으면 교차
  - 반대편에 있지 않은 경우에 선분 끝이 다른 선분의 중간에 있으면 교차
    - Cross product 결과가 0이고
    - 점이 다른 선분을 포함하는 axis-aligned rectangle에 포함되는 경우
  - 예제 그림
    - 교차: a, c
    - 미교차: b, d



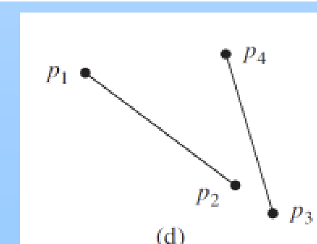
(a)



(b)



(c)



(d)

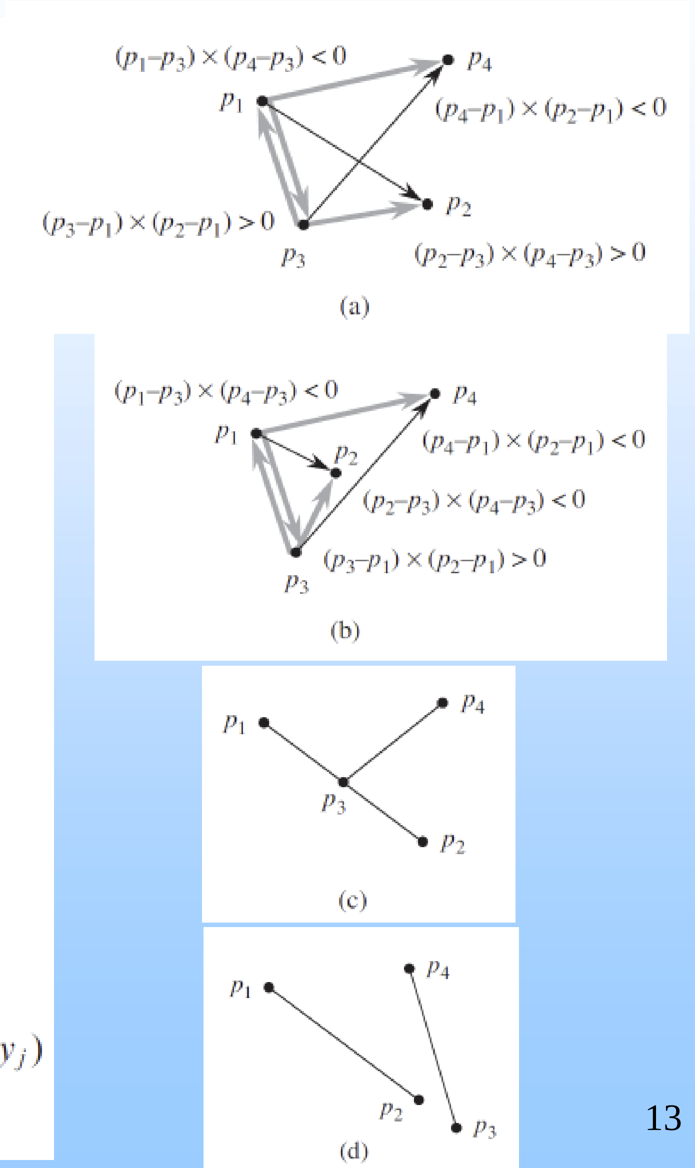
두 개 선분의 교차 여부 검사 방법 (2)

- Pseudo code

```
SEGMENTS-INTERSECT(p1, p2, p3, p4)
  d1 = DIRECTION(p3, p4, p1)
  d2 = DIRECTION(p3, p4, p2)
  d3 = DIRECTION(p1, p2, p3)
  d4 = DIRECTION(p1, p2, p4)
  if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
          ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
          return TRUE
  elseif d1 == 0 and ON-SEGMENT(p3, p4, p1)
          return TRUE
  elseif d2 == 0 and ON-SEGMENT(p3, p4, p2)
          return TRUE
  elseif d3 == 0 and ON-SEGMENT(p1, p2, p3)
          return TRUE
  elseif d4 == 0 and ON-SEGMENT(p1, p2, p4)
          return TRUE
  else return FALSE

DIRECTION(pi, pj, pk)
  return (pk − pi) × (pj − pi)

ON-SEGMENT(pi, pj, pk)
  if min(xi, xj) ≤ xk ≤ max(xi, xj) and min(yi, yj) ≤ yk ≤ max(yi, yj)
        return TRUE
  else return FALSE
```



목 차

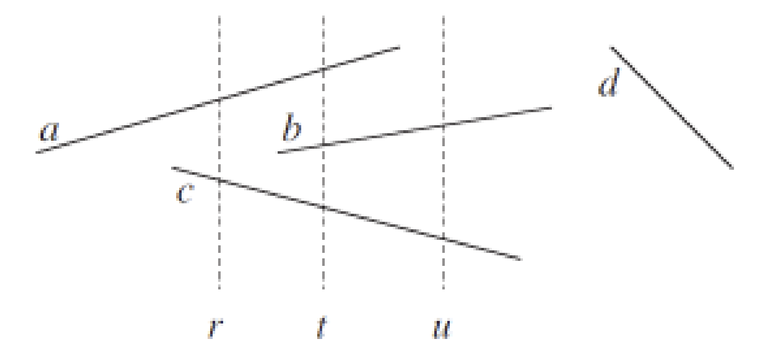
- Line-segment properties
- 교차하는 선분 존재 여부 검사 알고리즘
- Convex Hull(볼록 다각형) 여부 검사 알고리즘
- The closest pair of points
- 참고 교재 요약

문제 설명

- N개의 선분이 주어져 있다.
- 주어진 선분 중 교차하는 서로 교차하는 선분이 한 쌍이라도 존재하는지 여부를 검사하는 O(n lg n) 알고리즘을 제시하라.
  - 교차 쌍을 출력하지 않아도 됨.
- 가정
  - 세 개 이상 선분이 한 점에서 만나는 경우는 없다

특성 관찰 (1)

- 교차할 가능성이 있는 선분 쌍은 어떤 쌍들인가?
  - 선분의 x 좌표 값의 범위가 겹치지 않는 선분은 교차하지 않는다.
    - 아래 그림에서 d는 다른 선분들과 겹칠 가능성이 없음

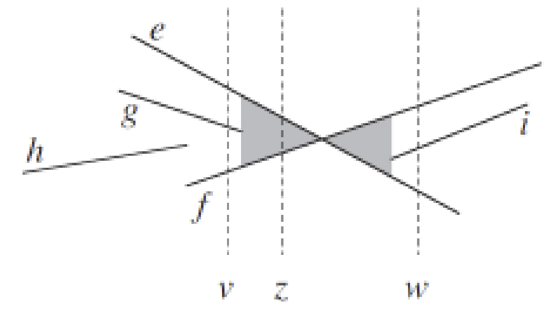


(a)

  - 따라서, x 좌표 값의 범위가 겹치는 선분으로 검사 대상을 제한할 수 있음.

특성 관찰 (2)

- 수직선을 $-\infty$에서 ∞까지 scan할 때 두 선분 교차점 주변 상황의 특징
  - 수직선이 교차점 직전일 때와 직후일 때 두 선분과 수직선이 만나는 교차점들(y값)의 대소가 바뀐다.
    - 아래 그림에서 z가 v에서 w로 움직일 때 선분 e와 선분 f가 수직선 z와 교차하는 점들을 비교하면 대소가 바뀜

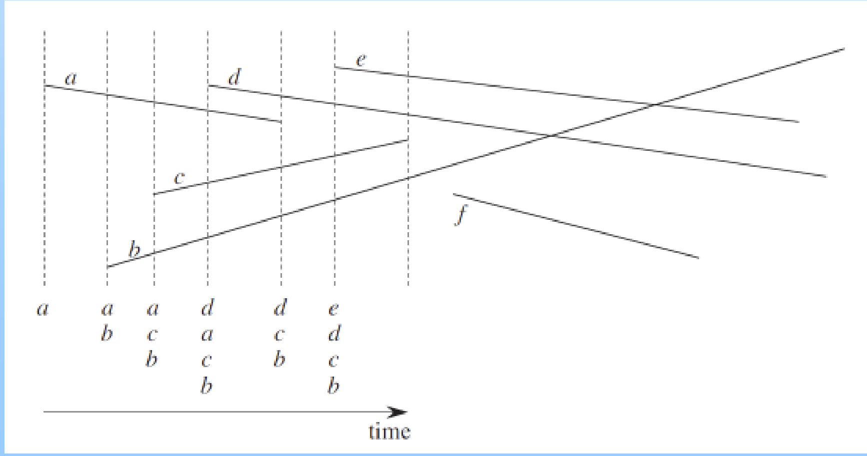


(b)

  - 수직선이 두 선분의 교차점을 지나가는 시점에 수직선과 두 선분이 만나는 두 개의 점 사이에 제 3의 선분과 수직선이 만나는 점은 없다.
    - 가정: 동시에 세 개의 선분이 만나는 경우는 없음
  - 수직선과 만나는 점들 기준으로 인접한 선분들 사이에서만 교차 검사를 수행하면 교차점을 찾을 수 있다.

핵심 아이디어

- 선분들의 양 끝점들을 x값 기준으로 오름차순으로 정렬 후 scan하자.
- 교차할 가능성이 있는 선분들 집합 S를 동적으로 유지한다.
  - 즉, scan 과정 중 만난 점이 선분의 왼쪽 점이면 해당 선분을 T에 추가
  - scan 과정 중 만난 점이 선분의 오른쪽 점이면 해당 선분을 T에서 제거
- T에 선분 추가 시
  - 추가된 선분과 추가된 선분의 바로 위 선분과 교차 여부 검사
  - 추가된 선분과 추가된 선분의 바로 아래 선분과 교차 여부 검사
- T에서 선분 제거 시
  - 제거된 선분의 바로 위 선분과 바로 아래 선분 사이의 교차 여부 검사
- 이 작업을 제일 오른쪽 점까지 진행



고려할 사항

- 수직 선분의 처리
  - y값이 낮은 점을 왼쪽 점으로 처리
  - (x, e, y) 기준으로 점들을 정렬.
    - 왼쪽 점이면 e = 0, 오른쪽 점이면 e = 1로 지정 후 정렬
- 교차 가능성이 있는 선분들의 집합 T에서 수직선과 만나는 점들 사이의 대소 관계를 효율적으로 관리하는 방법
  - 선분 삽입 시
    - Cross product 방법을 이용해서 이미 T에 있는 각각의 선분에 대해서 선분보다 점이 위에 있는지 아래에 있는지 검사
    - 검사 결과를 추가할 선분과 다른 선분의 위 아래 관계로 적용
    - 위 아래 관계를 기준으로 Red-black tree 구성
  - 선분 삭제 시
    - Red-black tree에서 선분 삭제

Pseudo Code

```
ANY-SEGMENTS-INTERSECT(S)
T = ∅
sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
for each point p in the sorted list of endpoints
        if p is the left endpoint of a segment s
                INSERT(T, s)
                if (ABOVE(T, s) exists and intersects s)
                        or (BELOW(T, s) exists and intersects s)
                        return TRUE
        if p is the right endpoint of a segment s
                if both ABOVE(T, s) and BELOW(T, s) exist
                        and ABOVE(T, s) intersects BELOW(T, s)
                        return TRUE
                DELETE(T, s)
return FALSE
```

- INSERT(T, s): insert segment s into T.
- DELETE(T, s): delete segment s from T.
- ABOVE(T, s): return the segment immediately above segment s in T.
- BELOW(T, s): return the segment immediately below segment s in T.

교차점 찾기 알고리즘의 Correctness (1)

- Correctness를 보이기 위해서 증명할 사항
  - 함수가 true를 return하면 교차점이 존재한다.
  - 교차점이 존재하면 true를 return한다.

- "함수가 true를 return하면 교차점이 존재한다."의 증명
  - 교차가 확인된 경우에만 true를 return
  - 따라서, true를 return하면 교차점이 존재한다.

- "교차점이 존재하면 true를 return한다."의 증명
  - 다음 장에 상세하게…

교차점 찾기 알고리즘의 Correctness (2)

- "교차점이 존재하면 true를 return한다."의 증명
  - Loop Invariant
    - Tree T는 점 p를 지나는 수직선과 만나는 점의 y값을 기준으로 선분들을 정렬한 binary search tree이다.

```
ANY-SEGMENTS-INTERSECT(S)
T = Ø
sort the endpoints of the segments in S from left to right,
      breaking ties by putting left endpoints before right endpoints
      and breaking further ties by putting points with lower
      y-coordinates first
for each point p in the sorted list of endpoints
      if p is the left endpoint of a segment s
            INSERT(T, s)
            if (ABOVE(T, s) exists and intersects s)
                  or (BELOW(T, s) exists and intersects s)
                  return TRUE
      if p is the right endpoint of a segment s
            if both ABOVE(T, s) and BELOW(T, s) exist
                  and ABOVE(T, s) intersects BELOW(T, s)
                  return TRUE
            DELETE(T, s)
return FALSE
```

  - 루프를 완료했다는 것은 교차점이 없다는 것을 의미함.
  - 따라서, 교차점이 존재하면 루프 중간에 return되어야 함.

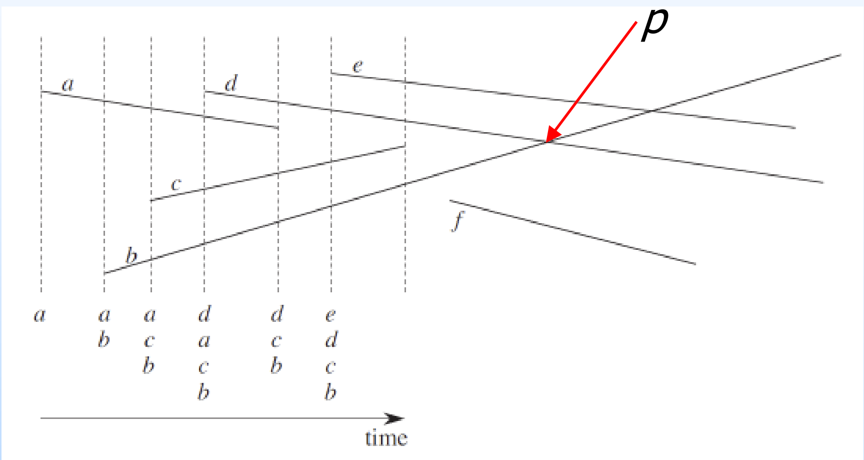
교차점 찾기 알고리즘의 Correctness (3)

- "교차점이 존재하면 true를 return한다."의 증명 (Cont.)
  - 초기 상태
    - T는 empty이므로 loop invariant는 만족한다.

```
ANY-SEGMENTS-INTERSECT(S)
T = ∅
sort the endpoints of the segments in S from left to right,
      breaking ties by putting left endpoints before right endpoints
      and breaking further ties by putting points with lower
      y-coordinates first
for each point p in the sorted list of endpoints
      if p is the left endpoint of a segment s
            INSERT(T, s)
            if (ABOVE(T, s) exists and intersects s)
                  or (BELOW(T, s) exists and intersects s)
                  return TRUE
      if p is the right endpoint of a segment s
            if both ABOVE(T, s) and BELOW(T, s) exist
                  and ABOVE(T, s) intersects BELOW(T, s)
                  return TRUE
            DELETE(T, s)
return FALSE
```

교차점 찾기 알고리즘의 Correctness (4)

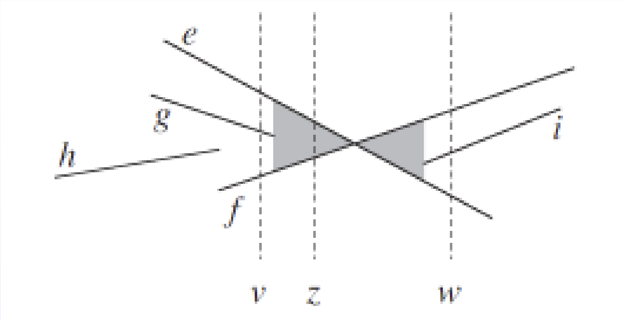
- "교차점이 존재하면 true를 return한다."의 증명 (Cont.)
  - Maintenance
    - 교차점 중에 가장 왼쪽에 있는 교차점을 p라 하자.



    - 점 p보다 앞에 있는 선분의 끝점들을 처리하는 경우를 고려해 보자.
    - 루프 내부를 수행하기 전에 Tree T가 binary search tree라면 루프 내부를 한번 더 수행해도 Tree T는 binary search tree이다.
    - 왜냐하면 두 선분이 수직선과 교차하는 값들의 대소가 바뀌는 경우는 없기 때문이다.

교차점 찾기 알고리즘의 Correctness (5)

- Maintenance (Cont.)
  - (code에 따르면) 교차 선분의 교차 검사는 두 가지 경우에 수행

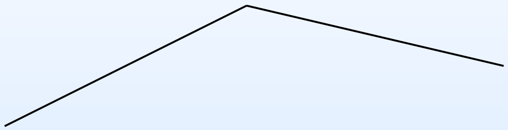


(b)

    - e와 f중 한 개가 T에 insert된 상태에서 다른 한 개가 추가되는 경우
    - 다른 선분의 끝점이 delete되면서 e와 f가 인접한 선분이 되는 경우
  - 다른 한 개가 추가될 때 교차 여부 검사하는 경우
    - e와 f는 교차하므로 true를 return하면서 루프를 빠져 나옴
  - 다른 선분의 끝점에서 선분이 delete되면서 검사하는 경우
    - 교차점 p보다 앞에 있는 선분의 끝점 중 맨 마지막 끝점에서 이 상황이 발생함.
    - 이 시점에 true를 return하고 루프 종료
    - 추가 고려해야 할 special case는 없나?

교차점 찾기 알고리즘의 Correctness (6)

- Maintenance (Cont.)
  - 추가 고려해야 할 special case
    - 두 개 선분이 끝점에서 만나는 경우



  - 추가 고려하지 않을 경우 버그가 발생할 수 있는 상황은?
    - 앞에 있는 선분의 끝점이 뒤에 있는 선분의 시작점보다 정렬 결과의 앞에 있는 경우
    - 이 경우 뒤에 오는 선분의 추가 전에 앞 선분이 T에서 삭제됨
    - 두 선분의 교차 검사가 이루어지지 않음
  - 처리 방법
    - Vertex들을 정렬할 때 (x, y) 대신 (x, e, y)로 정렬.
    - 여기서, (x, y)가 선분의 시작점이면 e = 0, 끝점이면 e = 1
    - 위치가 같으면 끝점이 앞에 오도록 정렬됨.
    - 따라서, 앞 선분 삭제 전에 뒤에 오는 선분이 추가됨

교차점 찾기 알고리즘의 Correctness (7)

- Termination
  - 교차점이 있다면 중간에 return됨.
  - 따라서, 루프를 정상 종료했다면 교차점이 없다는 것을 의미함.
- 증명 완료
  - 교차점이 존재하면 true를 return함
  - True를 return하면 교차점이 존재함

Time Complexity

- O(n lg n)
  - Line 1
    - O(1)
  - Line 2
    - O(n lg n)
  - Line 3-11
    - O(n lg n)

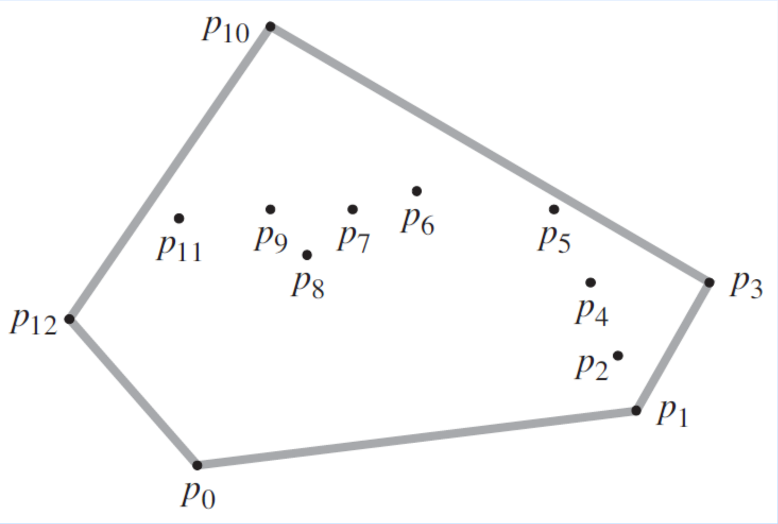
```
ANY-SEGMENTS-INTERSECT(S)
1  T = ∅
2  sort the endpoints of the segments in S from left to right,
       breaking ties by putting left endpoints before right endpoints
       and breaking further ties by putting points with lower
       y-coordinates first
3  for each point p in the sorted list of endpoints
4      if p is the left endpoint of a segment s
5          INSERT(T, s)
6          if (ABOVE(T, s) exists and intersects s)
               or (BELOW(T, s) exists and intersects s)
7              return TRUE
8      if p is the right endpoint of a segment s
9          if both ABOVE(T, s) and BELOW(T, s) exist
               and ABOVE(T, s) intersects BELOW(T, s)
10             return TRUE
11         DELETE(T, s)
12 return FALSE
```

목 차

- Line-segment properties
- 교차하는 선분 존재 여부 검사 알고리즘
- Convex Hull(볼록 다각형) 여부 검사 알고리즘
- The closest pair of points
- 참고 교재 요약

Convex Hull(볼록다각형) 찾기 알고리즘 (1)

- 문제
  - 2차원 좌표가 n개 주어져 있다.
  - 주어진 모든 점들을 포함하는 최소 convex hull을 구하라.
    - 벽에 못이 박혀있을 때 고무줄을 이용해서 모든 못들을 둘러싼 모양



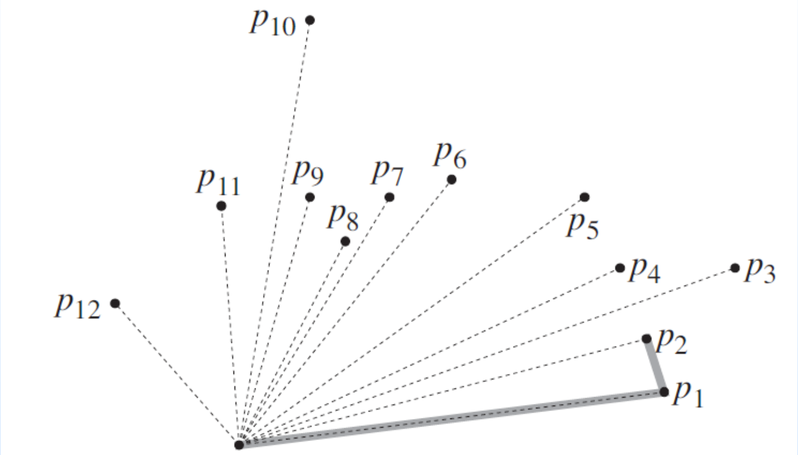
- 방법은?

Convex Hull(볼록다각형) 찾기 알고리즘 (2)

- 다양한 방법 존재
  - Graham’s scan algorithm
  - Jarvis’s march algorithm
  - Incremental method
  - Divide-and-conquer method

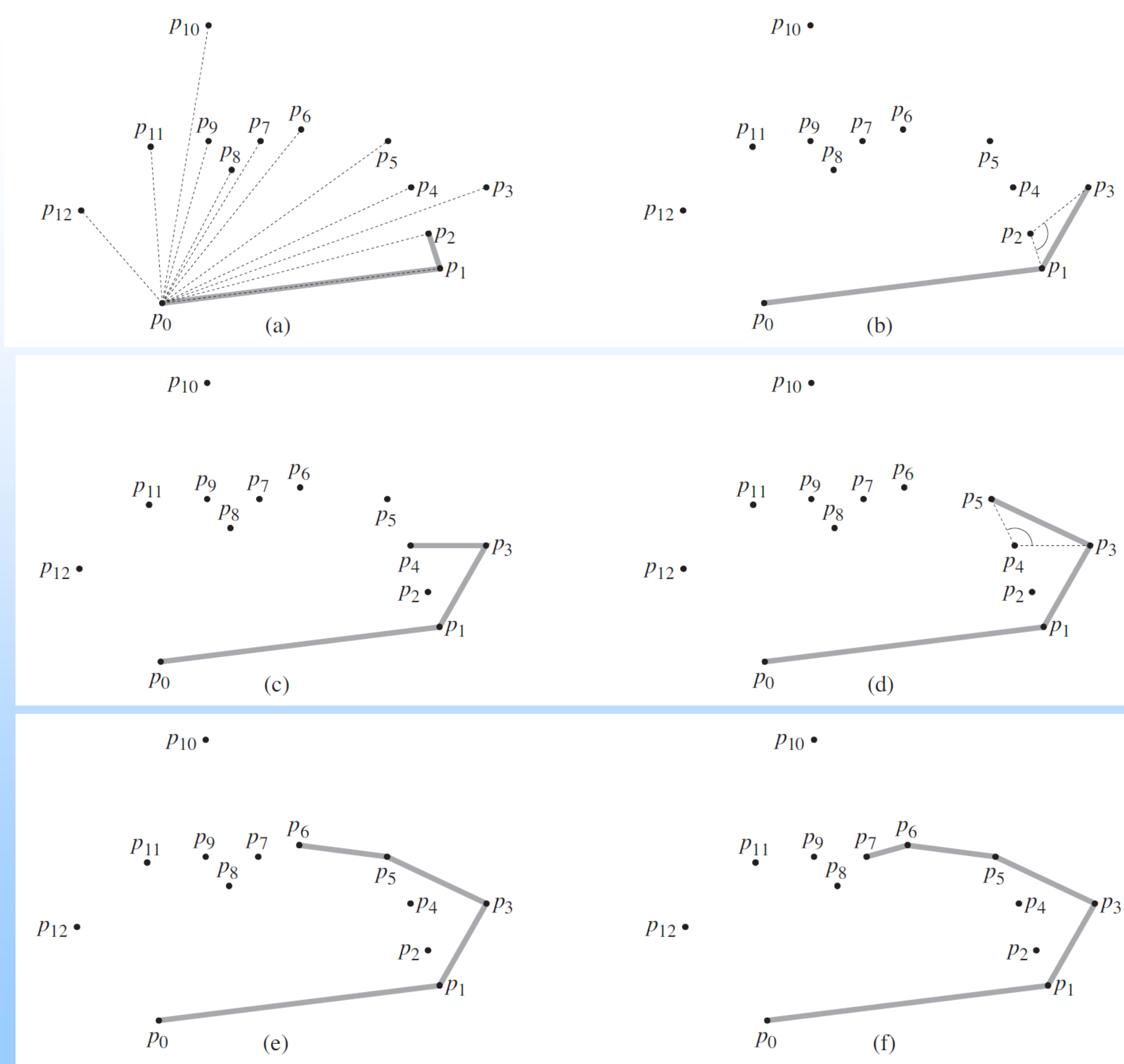
Graham's Scan (1)

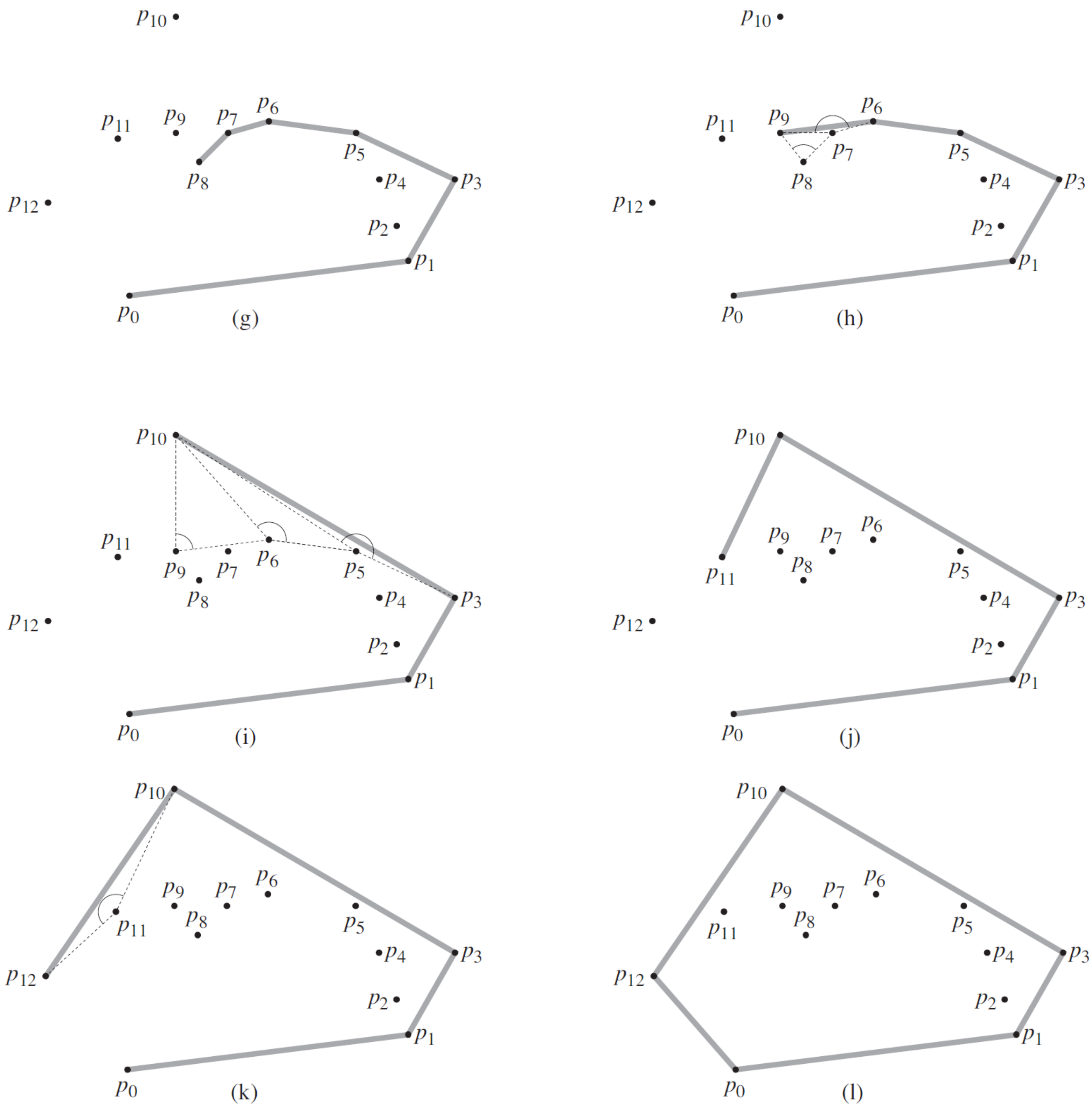
- 아이디어
  - y 좌표 값이 제일 작은 점 p0을 선택
  - p0을 지나는 수평선과 이루는 각도를 기준으로 다른 점들을 정렬



(a)

  - 정렬된 순서대로 scan하면서 convex hull을 구성할 후보 점들을 stack에 저장
  - Stack의 top에 있는 점이 convex hull을 구성하는 점이 아닌 것으로 확인되면 pop
  - 새로운 점이 후보 점이면 push
  - 모든 점들을 scan한 후 stack에 있는 점들이 convex hull을 구성하는 점들임


p_{10}
p_{11} p_9 p_7 p_6
p_8 p_5
p_4 p_3
p_{12}
p_2
p_1
p_0
(a)
(b)
(c)
(d)
(e)
(f)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0
(g)
p_{10}
p_{11}
p_9
p_6
p_7
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0
(h)
p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0
(i)
p_{10}
p_9
p_7
p_6
p_{11}
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(j)
p_{10}
p_9
p_7
p_6
p_{11}
p_8
p_5
p_{12}
p_4
p_3
p_2
p_1
p_0
(k)
p_{10}
p_9
p_7
p_6
p_{11}
p_8
p_5
p_{12}
p_4
p_3
p_2
p_1
p_0
(l)

Graham's Scan (4)

- Pseudo Code

```
GRAHAM-SCAN(Q)
let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
let ⟨p1, p2, ..., pm⟩ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p0
        (if more than one point has the same angle, remove all but
        the one that is farthest from p0)
let S be an empty stack
PUSH(p0, S)
PUSH(p1, S)
PUSH(p2, S)
for i = 3 to m
        while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                    and pi makes a nonleft turn
                POP(S)
        PUSH(pi, S)
return S
```

Graham's Scan (5)

- Theorem 33.1 (Correctness of Graham's scan)

  If GRAHAM-SCAN executes on a set Q of points, where $|Q| \geq 3$, then at termination, the stack S consists of, from bottom to top, exactly the vertices of CH(Q) in counterclockwise order.

  - CH(Q): 점들의 집합 Q의 convex hull을 구성하는 vertices

- 증명
  - 다음 장에 이어서…

Graham's Scan (6)

- 증명
  - Loop invariant를 이용해서 증명
  - Line 2를 수행하고 나면 vertex들의 sequence <p_1, …, p_m>이 생성됨
  - Line 4~6까지 수행한 후 stack에는 p0, p1, p2의 순서대로 쌓여 있음.

```
GRAHAM-SCAN(Q)
1   let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
2   let ⟨p1, p2, ..., pm⟩ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p0
        (if more than one point has the same angle, remove all but
        the one that is farthest from p0)
3   let S be an empty stack
4   PUSH(p0, S)
5   PUSH(p1, S)
6   PUSH(p2, S)
7   for i = 3 to m
8       while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and pi makes a nonleft turn
9           POP(S)
10      PUSH(pi, S)
11  return S
```

Graham's Scan (7)

- 증명 (cont.)
  - Loop invariant
    - 루프의 i번째 반복 시작 시점에 stack S에는 CH(Q_{i-1})를 저장
    - 여기서, CH(Q_{i-1})는 0번부터 i-1번째 vertex들이 주어졌을 때 구한 convex hull을 구성하는 vertex들을 의미
    - Stack의 bottom부터 top까지 vertex들이 반시계 방향 순서로 저장되어 있음

```
for i = 3 to m
    while the angle formed by points NEXT-TO-TOP(S), TOP(S),
            and p_i makes a nonleft turn
        POP(S)
    PUSH(p_i, S)
return S
```

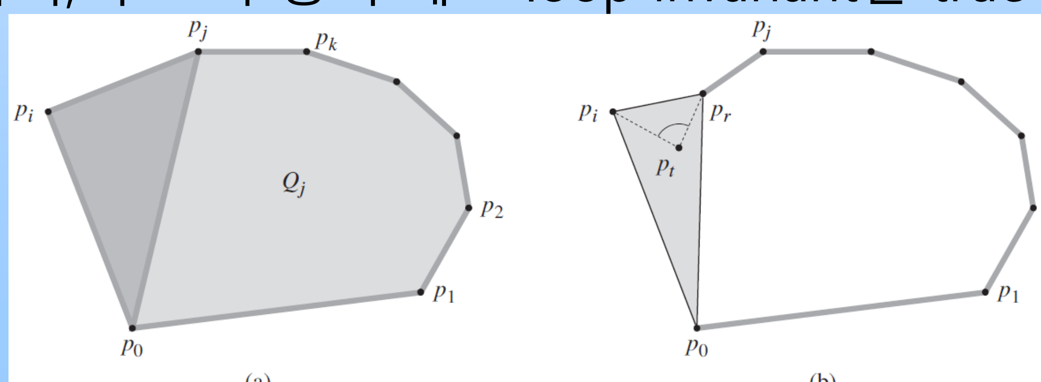
  - Initialization
    - i=3에서 시작하고 stack에는 p0, p1, p2가 있음
    - CH(Q2)는 세 개의 vertex들로 구성된 convex hull임.
    - 이 vertex들은 stack에 반시계 방향으로 쌓여 있음.
    - 따라서, 초기 상태에 loop invariant는 true

Graham's Scan (8)

- 증명 (cont.)
  - Maintenance

```
7   for i = 3 to m
8       while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and p_i makes a nonleft turn
9           POP(S)
10      PUSH(p_i, S)
11  return S
```

- i-1번까지 루프를 수행했을 때 loop invariant가 true라고 가정하자.
- 8번에서 left turn이 아니면 stack의 top에 있는 vertex는 convex hull을 구성하는 vertex가 될 수 없다. 그림 (b)의 p_t는 pop된다.
- 더 이상 pop되지 않을 때 stack에 있는 vertex들은 convex hull.
- p_i를 추가해도 convex hull(그림 a)이고 반시계 방향 유지.
- 따라서, 루프 수행 후에도 loop invariant는 true



Graham's Scan (9)

- 증명 (cont.)
  - Termination

```
for i = 3 to m
    while the angle formed by points NEXT-TO-TOP(S), TOP(S),
            and p_i makes a nonleft turn
        POP(S)
    PUSH(p_i, S)
return S
```

- $i = m + 1$에서 루프 종료
- 루프 invariant에 의해서 stack에는 $CH(Q_m)$이 반시계 방향으로 쌓여 있음.
- 이 때 Q_m은 전체 vertex 집합임.
- 따라서, 루프를 종료하면 stack에는 모든 vertex들을 포함하는 convex hull을 구성하는 vertex들이 반시계 방향 순서로 쌓여 있음.

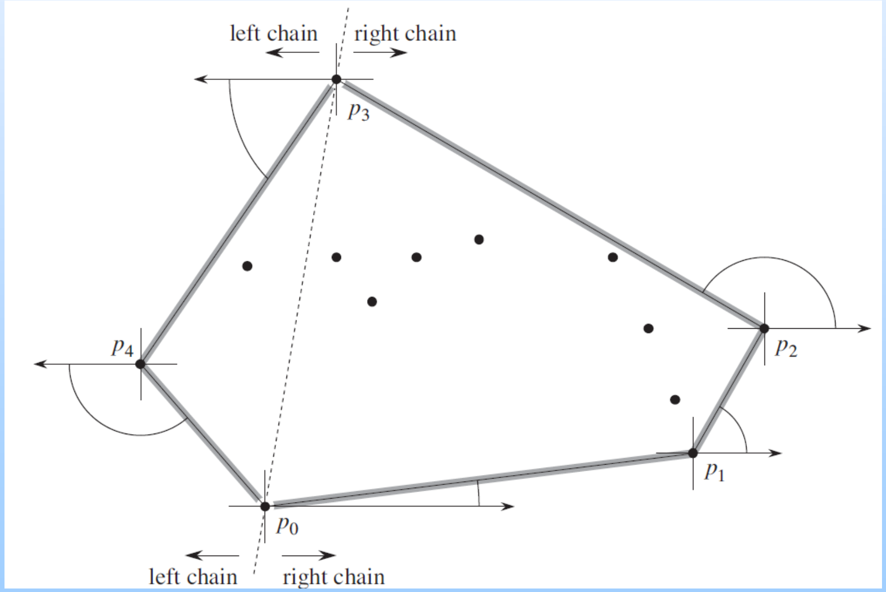
Graham's Scan (10)

- 시간 복잡도
  - 전체
    - O(n lg n)
  - Line 1
    - O(n)
  - Line 2
    - O(n lg n)
  - Lin3 3~6
    - O(1)
  - Line 7~10
    - 복잡도는?
    - O(n)
    - Why O(n)?
    - For-loop 수행 시 pop 수행 최대 누적 횟수는 n-3

```
GRAHAM-SCAN(Q)
1   let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie
2   let ⟨p1, p2, ..., pm⟩ be the remaining points in Q,
        sorted by polar angle in counterclockwise order around p0
        (if more than one point has the same angle, remove all but
        the one that is farthest from p0)
3   let S be an empty stack
4   PUSH(p0, S)
5   PUSH(p1, S)
6   PUSH(p2, S)
7   for i = 3 to m
8       while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                and pi makes a nonleft turn
9           POP(S)
10      PUSH(pi, S)
11  return S
```

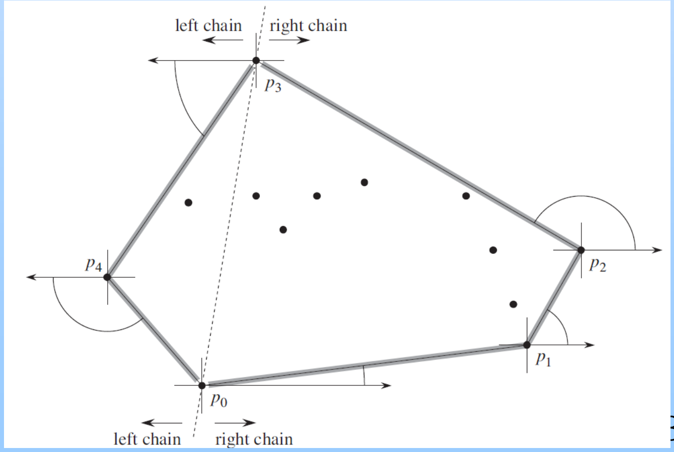
Jarvis's March (1)

- 아이디어
  - 선물을 포장지로 둘러싸는 방법과 유사
  - y값이 제일 낮은 vertex 선택
  - 종이를 팽팽하게 해서 오른쪽 방향부터 감쌀 때 첫 번째 만나는 점은 convex hull의 한 점이 된다.
  - 이 과정을 전체를 둘러쌀 때까지 반복



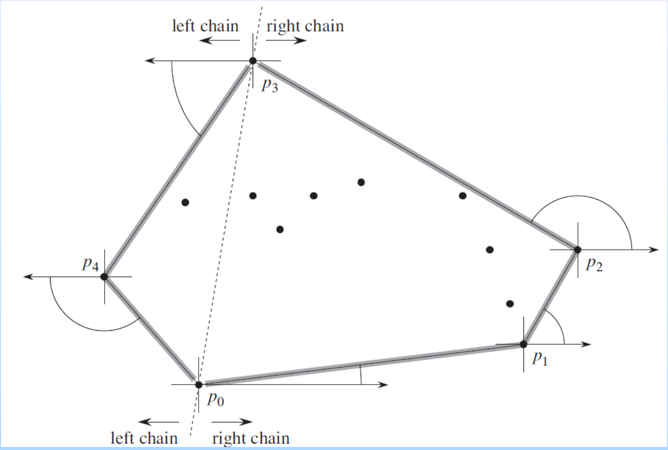
Jarvis's March (2)

- 목표
  - Convex hull을 구성하는 sequence H = $\{p_0, p_1, \ldots, p_{h-1}\}$ 생성
- 방법
  - y값이 제일 작은 vertex p_0를 찾는다.
    - p_0는 convex hull을 구성하는 점이 된다.
  - 찾은 점을 지나는 오른쪽 방향선과 이루는 각도가 제일 작은 vertex를 찾는다.
    - 각도가 같으면 제일 먼 vertex를 선택
  - 위 작업을 y값이 제일 큰 점을 찾을 때까지 반복
  - 동일한 작업을 왼쪽 방향선과 이루는 각도 기준으로 수행
  - 위 작업을 찾은 점이 p_0가 될 때까지 반복
- 질문
  - 각도가 제일 큰 점을 찾는 방법은?
- 답변
  - Cross product 사용



Jarvis's March (3)

- 시간 복잡도
  - O(nh)
    - h는 convex hull을 구성하는 vertex 개수
    - 수평선과 이루는 각도가 제일 작은 vertex 찾는 시간: O(n)
    - 위 작업 반복 횟수: O(h)



Incremental Method

- 아이디어
  - x좌표 기준으로 정렬해서 <p_1, p_2, …, p_n> 생성
  - i번째 단계에서 CH({p_1, p_2, …, p_{i-1}}) 갱신해서 CH({p_1, p_2, …, p_i}) 생성

- 시간 복잡도
  - O(n lg n)

- 문제
  - 갱신하는 방법을 기술하라.

Divide-and-Conquer Method

- 아이디어
  - 주어진 vertex들을 x 좌표의 median을 기준으로 두 그룹으로 분할
    - 왼쪽 그룹에 $\lceil n/2 \rceil$개
    - 오른쪽 그룹에 $\lfloor n/2 \rfloor$개
  - 각각의 그룹에 속한 vertex들로 구성되는 convex hull 구성
  - 두 개 그룹을 merge해서 convex hull 구성

- 시간 복잡도
  - T(n) = 2T(2/n) + O(n)
  - O(n lg n)

- 문제
  - Merge하는 방법을 기술하라.

주어진 vertex들 사이의 최장 거리 (1)

- 문제
  - n개의 vertex가 주어져 있다.
  - 가장 거리가 먼 vertex 쌍을 찾는 O(n lg n) 알고리즘을 제시하라.

- 방법
  - 주어진 vertex들을 둘러싸는 convex hull을 구한다.
    - O(n lg n)
  - Convex hull 내에서 가장 거리가 먼 vertex 쌍을 찾는다.
    - O(n)

- 질문
  - 가장 거리가 먼 두 개의 점은 반드시 convex hull을 구성하는 vertex로 이루어져 있는가?
  - Convex hull 내에서 가장 거리가 먼 vertex 쌍을 찾는 O(n) 알고리즘은 어떻게 되는가?

주어진 vertex들 사이의 최장 거리 (2)

- 문제
  - 가장 거리가 먼 두 개의 점은 반드시 convex hull을 구성하는 vertex로 이루어진다는 것을 증명하시오.
- 증명
  - 가장 거리가 먼 두 점 중 최소 한 점이 convex hull을 구성하는 vertex가 아니라고 가정하자.
  - 두 점을 연결하는 선분 p_1p_2를 확장하면 convex hull을 구성하는 edge e와 만나게 된다.
    - 왜냐하면 convex hull을 구성하지 않은 모든 vertex들은 convex hull 내부에 존재하기 때문이다.
  - Edge e의 양 끝점 중 하나와 p_1 또는 p_2를 연결한 선분은 p_1p_2보다 길다.
  - 모순 발생
  - 따라서, 가장 거리가 먼 두 점은 convex hull을 구성하는 vertex들이다.

주어진 vertex들 사이의 최장 거리 (3)

- 문제
  - Convex hull 내에서 가장 거리가 먼 vertex 쌍을 찾는 O(n) 알고리즘을 제시하시오.

- 아이디어
  - Convex hull을 구성하는 한 점 p_s을 선택한다.
  - p_s와 가장 먼 점 p_e를 찾는다.
  - p_s를 지나고 선분 p_sp_e에 수직한 직선을 그린다.
  - p_e를 지나고 선분 p_sp_e에 수직한 직선을 그린다.
  - Convex hull을 돌리면서 두 개의 수직선 간격이 커지는 경우를 찾는다.
  - Convex hull을 한 바퀴 돌리면서 두 개 수직선 사이의 최대 거리 찾는다.
  - 최대 거리를 만든 선분의 양 끝점이 가장 멀리 떨어져 있는 두 점이다.

- 알고리즘은?

목 차

- Line-segment properties
- 교차하는 선분 존재 여부 검사 알고리즘
- Convex Hull(볼록 다각형) 여부 검사 알고리즘
- The closest pair of points
- 참고 교재 요약

Closest Pair 찾기 (1)

- 문제
  - 2차원 평면 상에 2개 이상의 vertices가 주어져 있다.
  - 가장 가까운 pair를 찾는 O(n lg n) 알고리즘을 제시하시오.

- 아이디어
  - Brute-force method를 사용하면 $O(n^2)$에 구할 수 있다.
  - O(n lg n)에 하려면 divide-and-conquer 사용

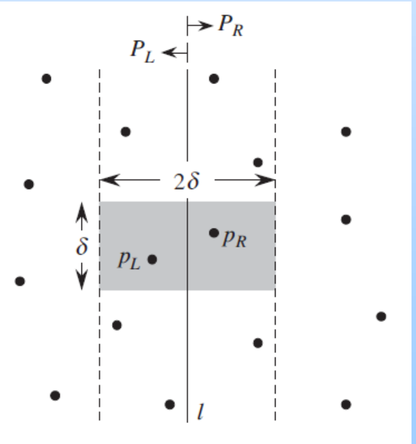
- O(n lg n) 알고리즘은?

Closest Pair 찾기 (2)

- Divide-and-conquer 기반 방법의 알고리즘 구조
  - Vertex들을 x좌표 기준으로 정렬
  - Divide
    - 정렬된 vertex들을 median을 기준으로 두 개의 그룹으로 분할
    - 왼쪽 vertex 그룹을 P_L, 오른쪽 vertex 그룹을 P_R이라 하자.
  - Conquer
    - 각 그룹에 있는 vertex들 사이의 최소 거리를 재귀적으로 구한다.
    - Vertex 개수가 3개 이하이면 brute-force method로 최소 거리 계산
    - 구한 최소 거리를 각각 δ_L, δ_R이라 하자.
    - $\delta = \min(\delta_L, \delta_R)$이라 하자.
  - Combine
    - (왼쪽 그룹의 점, 오른쪽 그룹의 점) 쌍 중 최소 거리를 구한다.
    - 구한 결과한 $\delta = \min(\delta_L, \delta_R)$ 중 작은 값이 최소 거리이다.
    - (왼쪽 그룹의 점, 오른쪽 그룹의 점) 쌍 중 최소 거리를 구하는 방법은?

Closest Pair 찾기 (3)

- (왼쪽 그룹의 점, 오른쪽 그룹의 점) 쌍 중 최소 거리를 구하는 방법은?
  - Brute-force 방법
    - 모든 쌍의 거리를 구하고 그 중 최소값을 구한다.
    - 최소 거리를 구하는데 소요되는 시간: $O(n^2)$
    - 전체 recurrence relation $T(n) = 2T(n/2) + O(n^2)$
    - 전체 시간 복잡도(master theorem 적용): $O(n^2)$
  - 경계선 근처에 있는 점들 사이의 거리만 계산하는 방법
    - conquer 단계에서 구한 최소 거리는 $\delta = \min(\delta_L, \delta_R)$이다.
    - 경계선과 거리가 δ이내인 점들만 고려한다.
    - 모든 점이 δ이내에 있으면? $O(n^2)$
    - 개선 방법은?
    - y좌표 기준으로 정렬 후 증가순으로 비교
    - $O(n \lg n)$
    - 전체 recurrence relation $T(n) = 2T(n/2) + O(n \lg n)$
    - 전체 시간 복잡도(Exercises 4.6-2): $O\left(n(\lg n)^2\right)$
    - $O(n \lg n)$가 되도록 하는 방법은?



(a)

Closest Pair 찾기 (4)

- 경계선 근처의 점들 사이의 거리를 효율적으로 계산하는 방법
  - 초기에 한번 전체 vertex들을 y좌표 기준으로 정렬
  - 입력으로 y좌표 기준으로 정렬된 정보를 전달
  - 재귀 호출 시에 왼쪽, 오른쪽 각각 y값 기준으로 정렬된 정보를 전달
    - 전달 정보 생성 방법: merge의 반대 개념 적용

```
let Y_L[1..Y.length] and Y_R[1..Y.length] be new arrays
Y_L.length = Y_R.length = 0
for i = 1 to Y.length
    if Y[i] ∈ P_L
        Y_L.length = Y_L.length + 1
        Y_L[Y_L.length] = Y[i]
    else Y_R.length = Y_R.length + 1
        Y_R[Y_R.length] = Y[i]
```

    - 전달 정보 생성 시간: $O(n)$
  - 경계선과 거리가 δ이내인 점들만 고려
  - 이미 y좌료 기준으로 정렬되어 있으므로 추가 정렬할 필요 없음
  - y좌표 기준으로 순차적으로 비교해서 최소 거리 계산

Closest Pair 찾기 (5)

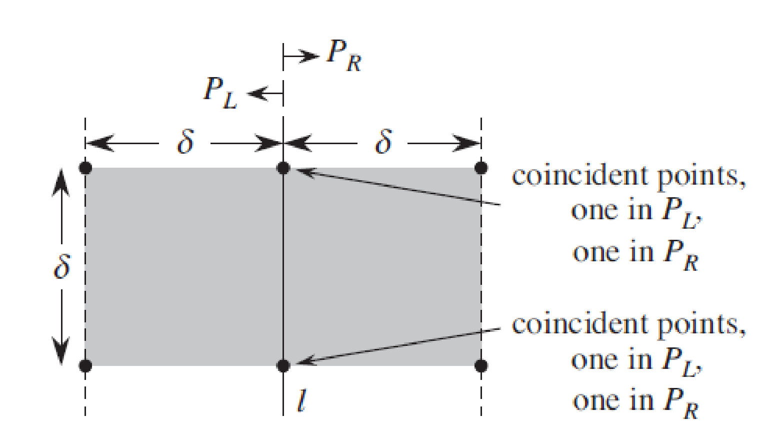
- Divide-and-conquer algorithm
  - Vertex들을 x좌표 기준으로 정렬한 sequence X 저장
  - Vertex들을 y좌표 기준으로 정렬한 sequence Y 저장
  - 첫 번째 호출 입력: vertex 집합 P와 X, Y
  - Divide
    - 정렬된 vertex들을 median을 기준으로 두 개의 그룹으로 분할
    - 왼쪽 vertex 그룹을 P_L, 오른쪽 vertex 그룹을 P_R이라 하자.
    - X_L, Y_L, X_R, Y_R 구성 (X_L은 X값 기준으로 왼쪽 점들 정렬 결과 의미)
  - Conquer
    - P_L, X_L, Y_L을 입력으로 재귀호출해서 최소 거리 δ_L 계산
    - P_R, X_R, Y_R을 입력으로 재귀호출해서 최소 거리 δ_R 계산
    - $\delta = \min(\delta_L, \delta_R)$이라 하자.
  - Combine
    - 경계선 근처의 점들 사이의 거리 중 최소값 계산
    - 구한 최소값과 δ 중 작은 값이 최소 거리이다.

Closest Pair 찾기 (6)

- Time Complexity
  - 초기 정렬
    - $O(n \lg n)$
  - Divide
    - $O(n)$
  - Combine
    - $O(n)$ ← correct?
  - Recurrence relation
    - $T(n) = 2T(n/2) + O(n)$ if $n > 3$
    - $T(n) = O(1)$ if $n \leq 3$
  - 시간 복잡도
    - $T(n) = O(n \lg n)$

Closest Pair 찾기 (7)

- Combine 시간 복잡도가 $O(n)$ 인 이유
  - 재귀 호출로 구한 subproblem에서 최소 거리가 $\delta = \min(\delta_L, \delta_R)$
  - 경계선 중심으로 좌우로 δ 이내인 점들을 대상으로 거리 계산
  - 각 점에 대해서 거리를 계산하게 되는 다른 점들 개수는 최대 7개
    - $\delta \times \delta$ 사각형의 각 꼭지점에 점들이 위치하는 경우가 최대
    - 중간에 다른 점이 있으면 최소 거리가 δ 보다 작아지므로 모순
  - 따라서 모든 점들이 2δ 폭 영역 내에 있어도 $O(n)$ 시간에 최소 거리 계산 가능



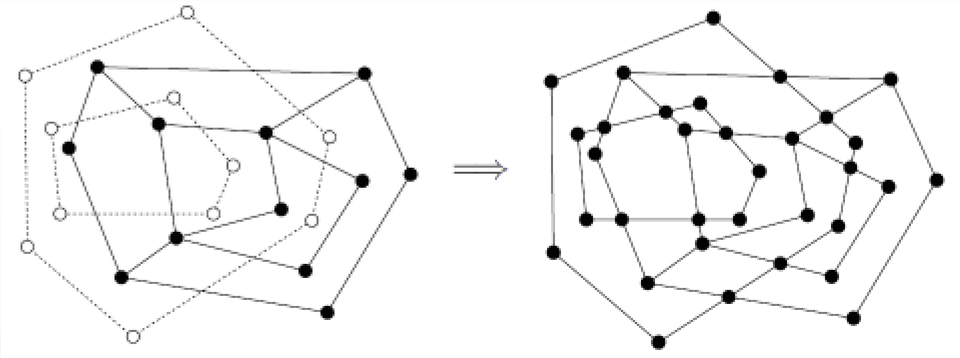
(b)

목 차

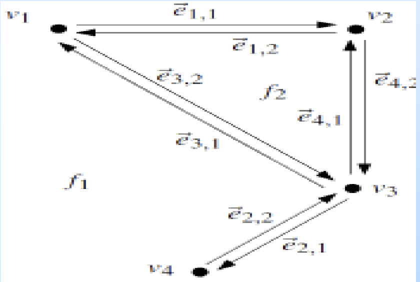
- Line-segment properties
- 교차하는 선분 존재 여부 검사 알고리즘
- Convex Hull(볼록 다각형) 여부 검사 알고리즘
- The closest pair of points
- 참고 교재
  - 관련 Problem들 고찰

Overlay of Two Subdivisions

- 문제
  - 두 개의 구획 정보가 있다.
    - 점선 영역과 실선 영역
  - 겹치는 부분을 찾아라.



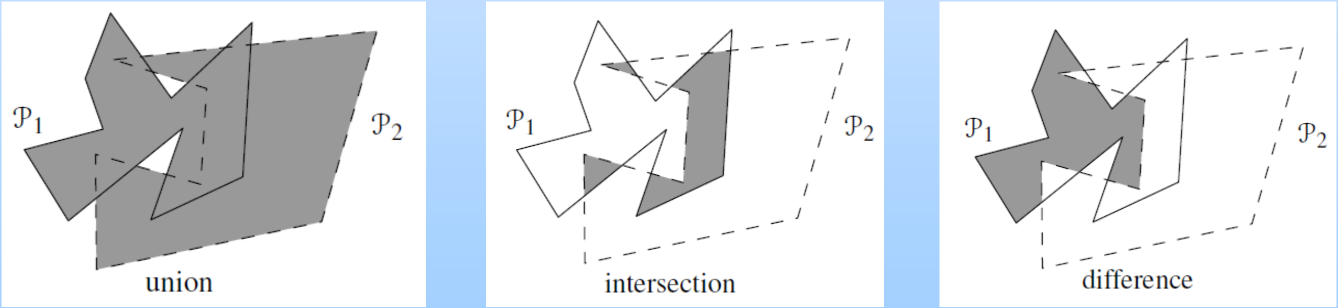
- 방법
  - 교차점을 찾아서 vertex를 추가한다.
  - 각 면이 원래 영역에서 내부인지 외부인지 판단한다.
    - 두 개 영역에서 모두 내부이면 겹치는 부분이다.
  - 주어진 면이 내부인지 외부인지 판단하는 방법
    - Doubly connected edge list (half-edge) 자료구조 사용
    - Edge의 왼쪽이 edge를 사용하는 면이 되도록 표현하는 자료구조
    - 면을 구성하는 half-edge list가 clockwise이면 외부 아니면 내부
    - 가장 바깥 cycle과 hole이 clockwise.



- 참고 교재 chapter 2 참고

Boolean Operations

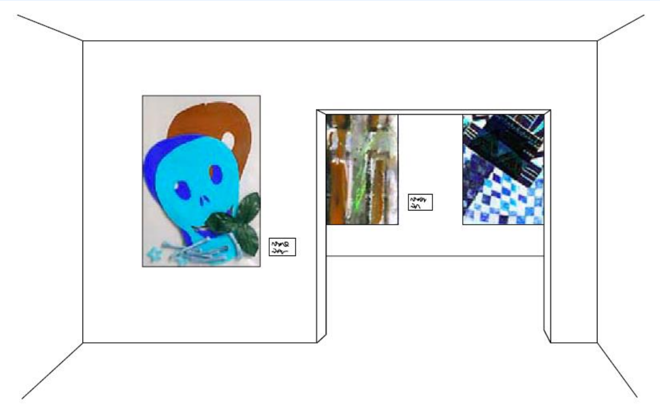
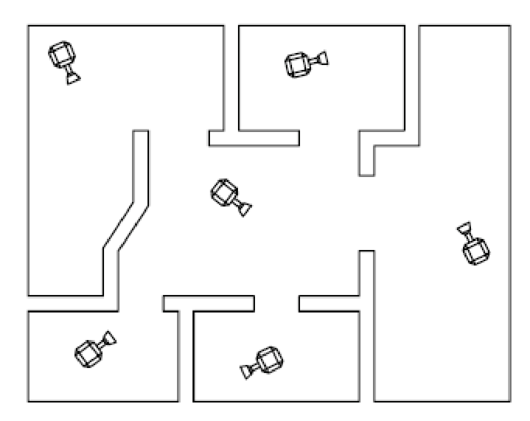
- 문제
  - 두 개의 다각형이 주어져 있다.
  - 두 다각형이 합쳐진 새로운 다각형을 구하라.
  - 공통 영역으로 구성되는 다각형을 구하라.
  - 한쪽 다각형에만 있는 영역으로 구성되는 다각형을 구하라.
- 방법
  - Union operation
  - Intersection operation
  - Difference operation



- 응용: 3차원 기본 입체들의 조합으로 새로운 입체 생성
- 참고 교재 chapter 2 참고

Art Gallery Problem (1)

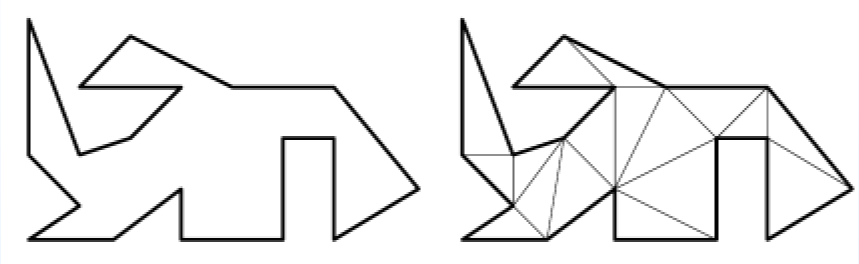
- 문제
  - 미술관에 도난방지용 CCTV 카메라를 설치하려고 한다.
  - 각 카메라 영상은 감시실의 모니터링 TV 화면에 출력된다.
  - 미술관 내의 모든 곳을 볼 수 있으려면 최소 몇 개의 카메라가 필요한가?
  - 그리고, 어디에 설치해야 하는가?

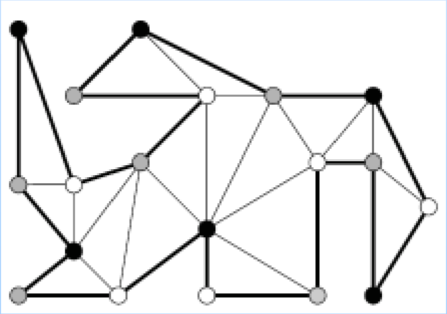
- 아이디어
  - 공간을 2차원 평면으로 매핑 후 삼각형으로 분할
  - 삼각형마다 하나씩 설치. 그러나, 최적은 아님
  - 카메라 개수를 줄이는 방법을 찾아보자 (다음 장...)

Art Gallery Problem (2)

- 알고리즘
  - 주어진 다각형을 삼각형으로 분할

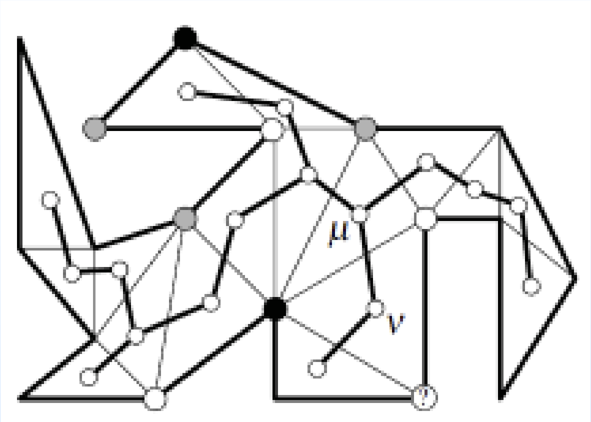


    - 다양한 삼각형 분할(triangulation) 방법은 참고 교재 Ch3, Ch9 참고
  - 3-coloring
    - 모든 삼각형의 꼭지점을 세 가지 색(white, black, gray)으로 칠함
    - 각 edge의 양 끝점의 색깔이 다르게 칠함
    - 3-coloring이 가능하다는 증명은 다음 장에.
  - Gray vertex에 카메라 설치
    - 각각의 삼각형은 gray vertex를 포함
    - 따라서 모든 삼각형 내부를 카메라로 촬영 가능
    - 최대 카메라 대수는 n/3



Art Gallery Problem (3)

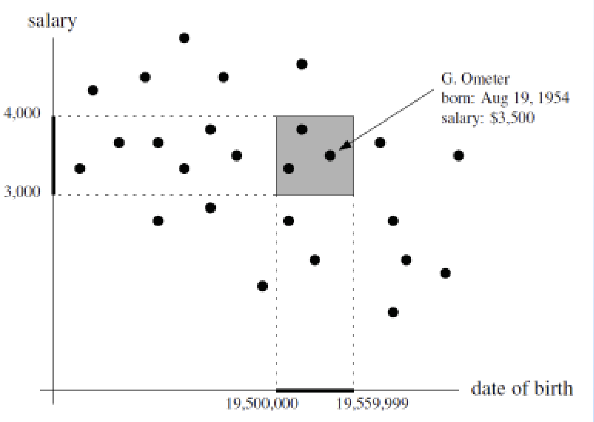
- 3-coloring
  - 각 삼각형을 vertex에 대응시키는 dual graph 생성



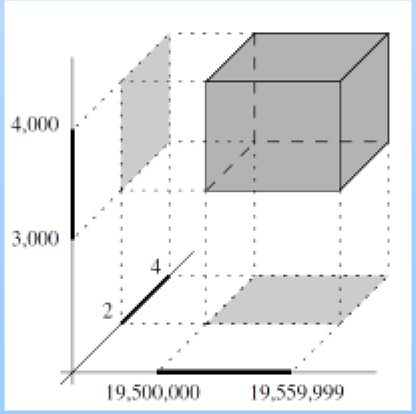
  - Dual graph는 tree이다. 왜냐하면 dual graph의 edge를 절단하면 다각형이 두 개로 분할되기 때문이다.
  - Dual graph의 tree 중 임의의 node 선택
  - 선택한 node에 해당하는 삼각형의 세 꼭지점을 black, gray, white로 색칠
  - 선택한 노드를 root로 시작해서 depth first search
  - 새로 접근하는 node에 해당하는 삼각형의 꼭지점 두 개는 이미 색칠된 상태. 나머지 vertex를 사용하지 않은 색으로 칠함.

Orthogonal Range Searching

- 문제
  - 주어진 Database에서 1950년~1955년 출생, 급여 $3,000~$4,000 추출
- 방법
  - Query를 기하적으로 해석
  - 2차원 평면에서 사각형 내에 있는 점에 추출

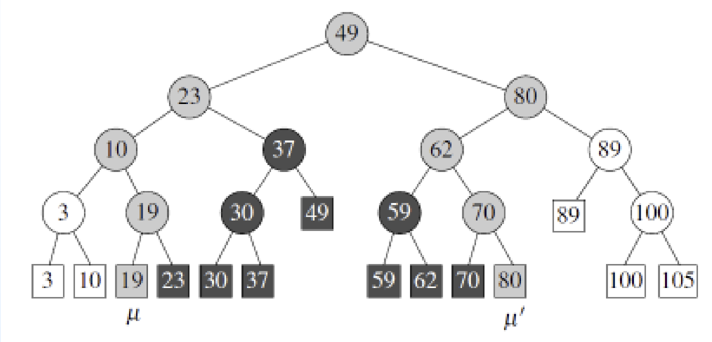


- 차원 확대
  - Query
    - 1950년~1955년 출생, 급여 $3,000~$4,000, 자녀가 2~4명
  - 결과
    - 3차원 공간에서 육면체 내에 있는 점에 해당



1-Dimensional Range Searching

- 문제
  - 직선상에 있는 점들 중 interval [x:x′] 있는 모든 점들을 찾아라

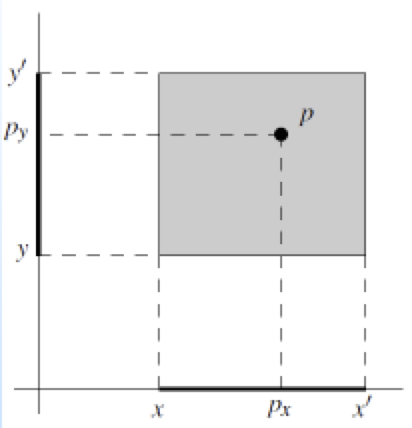


- 방법
  - Balanced binary search tree 사용
    - B-tree, B^+-tree 혹은 red-black tree
  - BST를 탐색하면서 x와 x′이 split되는 node 추출
    - 값이 x보다 크고 x′보다 작은 node
  - Split node의 left-subtree에서 x보다 크거나 같은 모든 node 출력
  - Split node의 right-subtree에서 x′보다 작거나 같은 모든 node 출력

- 시간복잡도
  - O(k + lg n)
    - k는 찾은 node 개수

Kd-Tree (1)

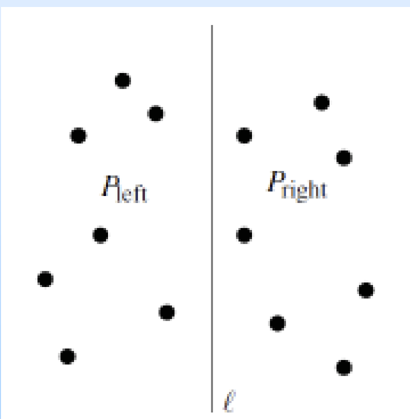
- 문제
  - 2차원 평면에서 사각형 영역에 속하는 모든 점들을 찾아라

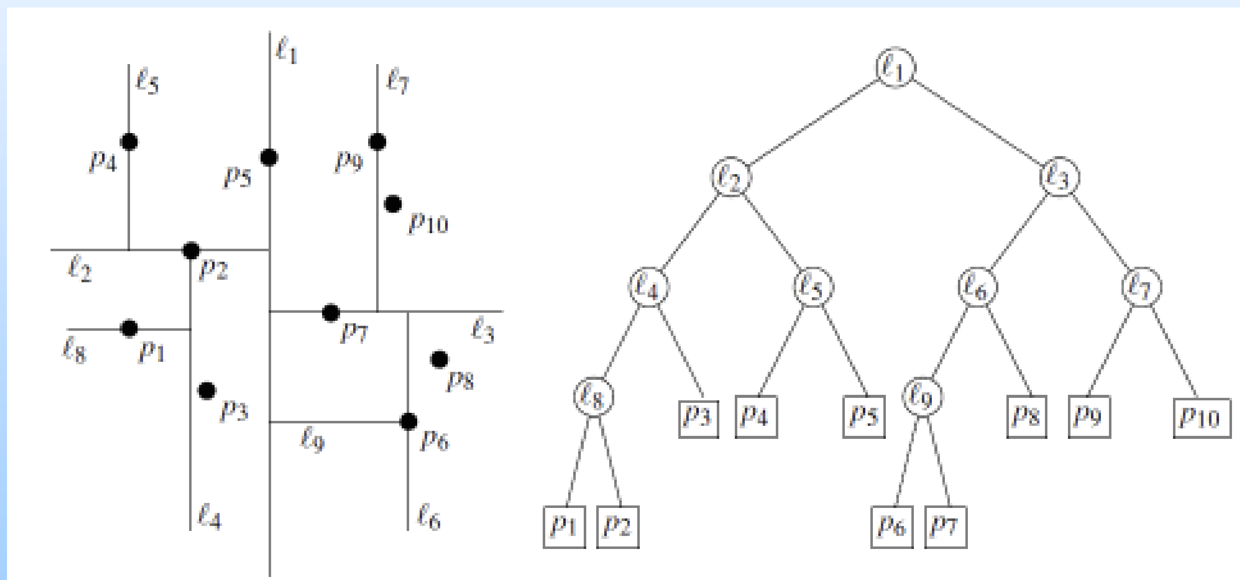


- 방법
  - kd-tree 구축 (상세 설명은 다음 장에…)
  - Kd-tree에서 search (상세 설명은 다음 장에…)

Kd-Tree (2)

- 2-dimension kd-tree 생성
  - 전체 점들을 수직선을 기준으로 왼쪽과 오른쪽으로 분할
  - 분할된 영역에서 수평선 기준으로 위와 아래로 구분
  - 분할된 영역에서 다시 수직선 기준으로 왼쪽과 오른쪽으로 분할
  - 위의 작업을 반복





Kd-Tree (3)

- 2-dimension kd tree 생성 알고리즘

Algorithm BUILDKDTREE(P, $depth$)
Input. A set of points P and the current depth $depth$.
Output. The root of a kd-tree storing P.
1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else** **if** $depth$ is even
4. **then** Split P into two subsets with a vertical line ℓ through the median x-coordinate of the points in P. Let P_1 be the set of points to the left of ℓ or on ℓ, and let P_2 be the set of points to the right of ℓ.
5. **else** Split P into two subsets with a horizontal line ℓ through the median y-coordinate of the points in P. Let P_1 be the set of points below ℓ or on ℓ, and let P_2 be the set of points above ℓ.
6. $v_{\text{left}} \leftarrow$ BUILDKDTREE(P_1, $depth + 1$)
7. $v_{\text{right}} \leftarrow$ BUILDKDTREE(P_2, $depth + 1$)
8. Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
9. **return** v

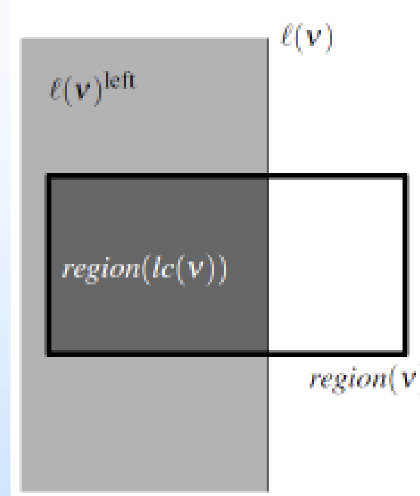
  - Time complexity: O(n lg n)

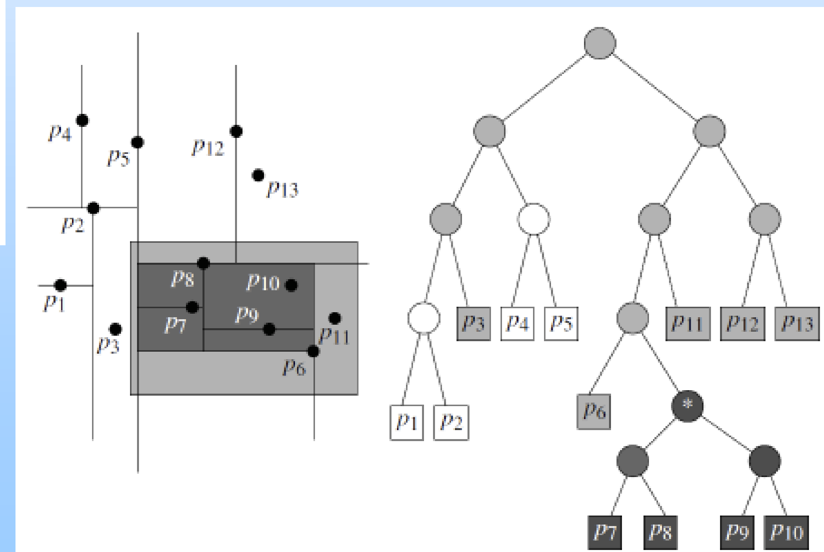
Kd-Tree (4)

- Kd tree 탐색 알고리즘

Algorithm SEARCHKDTREE(v, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below v that lie in the range.

```
if v is a leaf
   then Report the point stored at v if it lies in R.
   else if region(lc(v)) is fully contained in R
           then REPORTSUBTREE(lc(v))
           else if region(lc(v)) intersects R
                   then SEARCHKDTREE(lc(v), R)
        if region(rc(v)) is fully contained in R
           then REPORTSUBTREE(rc(v))
           else if region(rc(v)) intersects R
                   then SEARCHKDTREE(rc(v), R)
```





Kd-Tree (5)

- 2-dimension kd tree 탐색 알고리즘의 시간 복잡도
  - Basis step
    - root와 수직선이 주어지면 수직선으로 공간 분할된 경우
  - Recursive step
    - 수평선으로 나눈 후 다시 수직선으로 나누어짐
    - 각각 수직선으로 나누어지는 subproblem이 2개 생성
  - Recurrence relation

$$Q(n) = \begin{cases} O(1), & \text{if } n = 1, \\ 2 + 2Q(n/4), & \text{if } n > 1. \end{cases}$$

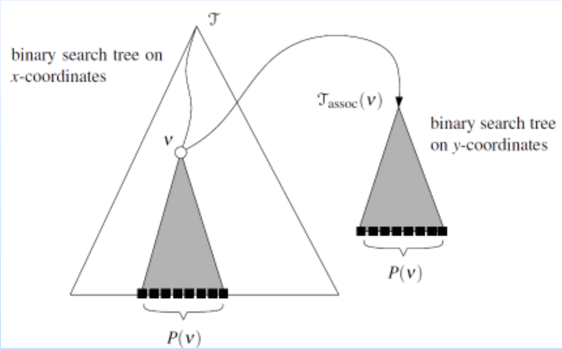
  - 시간 복잡도
    - $O(\sqrt{n} + k)$
    - k는 결과 개수

Algorithm SEARCHKDTREE(ν, R)
Input. The root of (a subtree of) a kd-tree, and a range R.
Output. All points at leaves below ν that lie in the range.

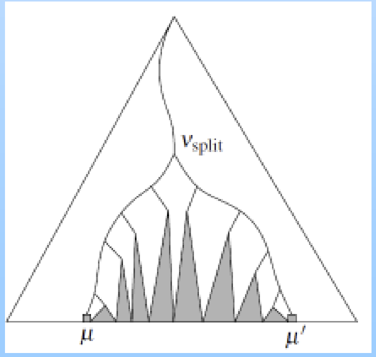
```
if ν is a leaf
  then Report the point stored at ν if it lies in R.
  else if region(lc(ν)) is fully contained in R
         then REPORTSUBTREE(lc(ν))
         else if region(lc(ν)) intersects R
                then SEARCHKDTREE(lc(ν), R)
       if region(rc(ν)) is fully contained in R
         then REPORTSUBTREE(rc(ν))
         else if region(rc(ν)) intersects R
                then SEARCHKDTREE(rc(ν), R)
```

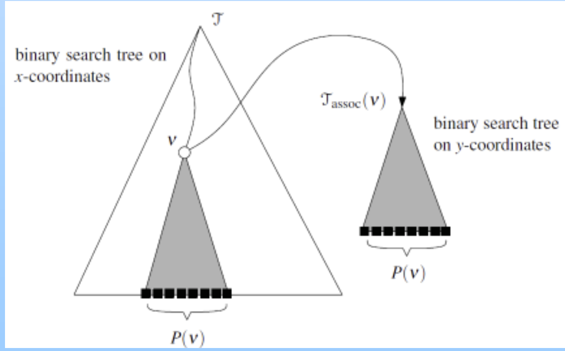
Range Trees

- 2-dimensional range tree
  - x좌표 balanced search tree를 생성
  - 각 node가 포함하는 정보
    - 자신의 descendants들을 y좌표 기준으로 BST 생성
    - Root pointer 저장



  - 탐색
    - 1-dimensional range search
    - 포함되는 subroot들을 대상으로 다시 1-D search (회색 영역)





    - O(n lg n) 공간복잡도, O(lg^{2}n + k) 시간복잡도

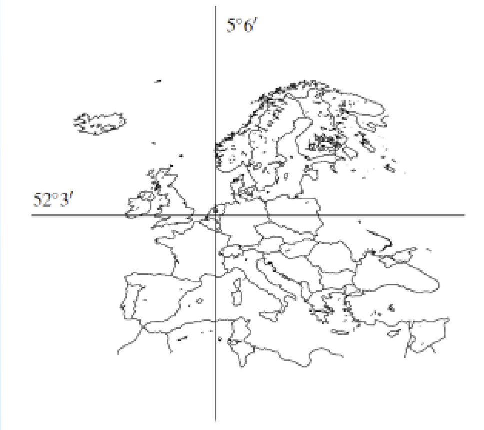
Point Location (1)

- 문제
  - 점 (x, y)가 정사각형 [xmin, xmax], [ymin, ymax] 내부에 있는지 여부를 검사하는 방법을 제시하라.

- 방법
  - $x_{min} \le x \le x_{max}$ 이고 $y_{min} \le y \le y_{max}$이면 내부.
  - 아니면 외부

Point Location (2)

- 문제
  - 유럽 국가 경계선이 다각형 정보로 주어져 있다.

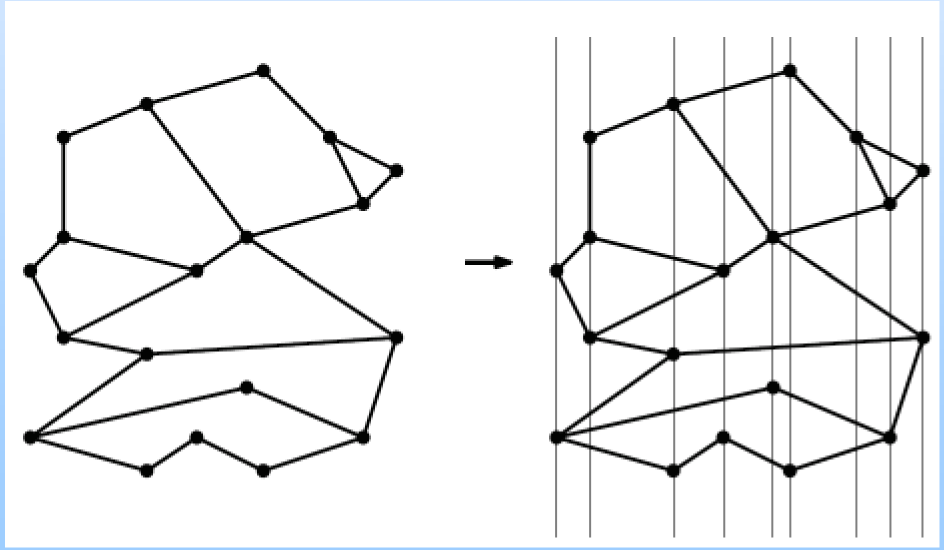


  - 휴대폰 사용자가 현재 어느 국가에 있는지를 알려주는 APP을 개발하려고 한다.
  - 휴대폰은 인공위성으로부터 GPS 정보를 받을 수 있다.
  - GPS 정보가 주어지면 사용자가 있는 국가 이름을 출력하라.

- 방법은?

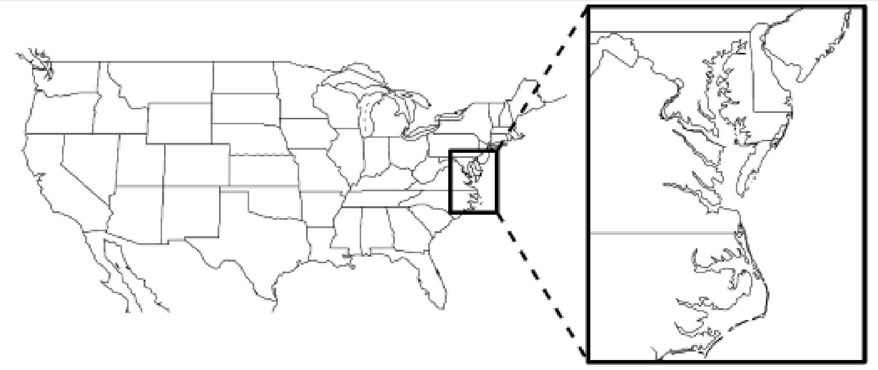
Point Location (3)

- 자료구조
  - 각 점을 지나는 수직선을 긋는다.
  - 그러면 모든 지역은 삼각형 혹은 사각형으로 분할된다.
  - 수직선과 주어진 선분이 만나는 점들을 찾아서 작은 영역들을 생성한다.
  - 각 점들을 x좌표 기준으로 정렬해서 저장한다.
  - 각 수직선별로 다른 선분과 만나는 점들의 y좌표를 정렬해서 저장한다.



Windowing Query

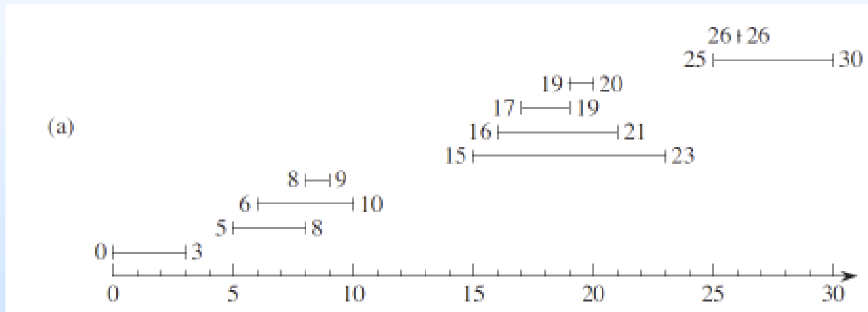
- 상황
  - 자동차 네비게이션 화면에 지도를 출력하고자 한다.
  - 지도 정보는 서버에 저장되어 있다.
  - 현재 위치는 GPS로 확인 가능하다.
  - 현재 위치에 해당하는 지도 정보를 화면에 출력하고자 한다



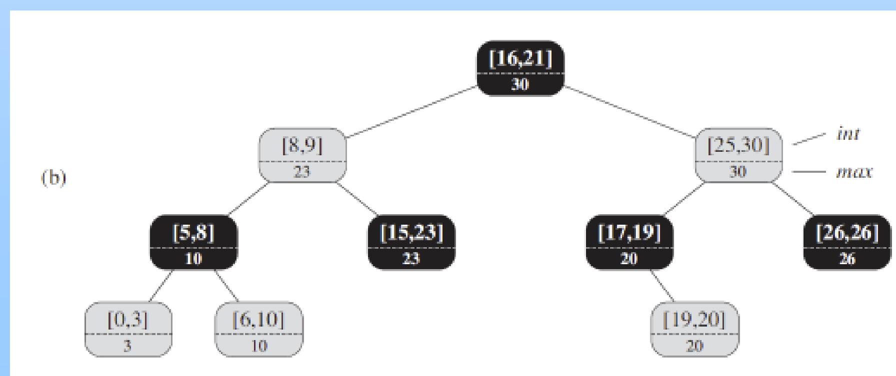
- 문제
  - 지도 정보는 고정되어 있지만 자동차의 위치는 변한다.
  - 화면에 출력할 영역을 찾는 효율적인 알고리즘을 제시하라.
- 방법
  - Interval tree와 range tree 사용

Interval Trees (1)

- 문제
  - x축에 평행한 n개의 선분이 주어져 있다.
  - 선분 $[x_{min}, x_{max}]$과 겹치는 선분을 구하는 효율적인 알고리즘을 제시하라.



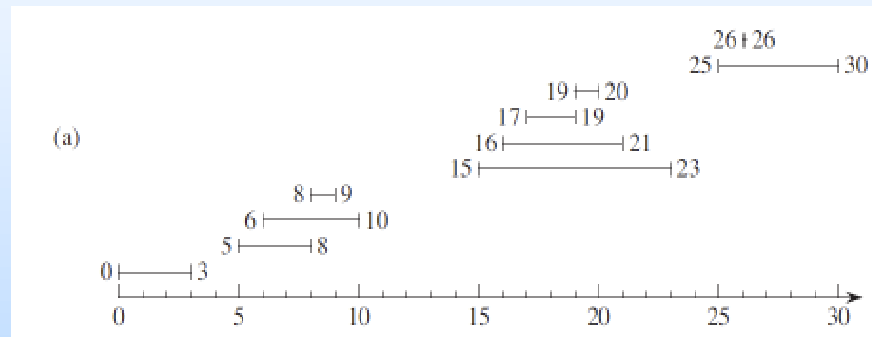
- 방법
  - 선분들을 binary search tree에 저장
    - node의 대소 관계는 시작점으로 판별
    - 각 node는 선분 시작점, 선분 끝점, subtree에 있는 선분의 최대값을 저장

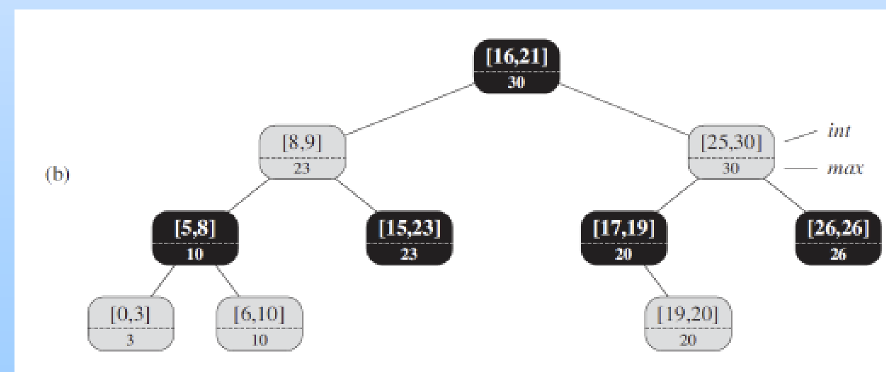


Interval Trees (2)

- 방법 (cont.)
  - 탐색 진행 조건
    - 주어진 선분과 노드에 저장된 선분이 겹치는지 여부
    - 노드에 저장된 max 값과 주어진 선분의 최대값 비교

```
INTERVAL-SEARCH(T, i)
x = T.root
while x ≠ T.nil and i does not overlap x.int
    if x.left ≠ T.nil and x.left.max ≥ i.low
        x = x.left
    else x = x.right
return x
```





- Introduction to Algorithms 14.3, 참고 교재 10.1 참고
- 사각형 영역과 겹치는 선분 찾기: 각 노드가 subtree 정보 유지 (range tree)

Delaunay Triangulations (1)

- 상황
  - 지구 표면을 높이를 반영하는 지형으로 표현하고자 한다.
  - 이를 위해서 여러 지점에서의 높이를 측정하였다.
  - 측정되지 않은 위치까지 반영하여 지형을 생성하라.

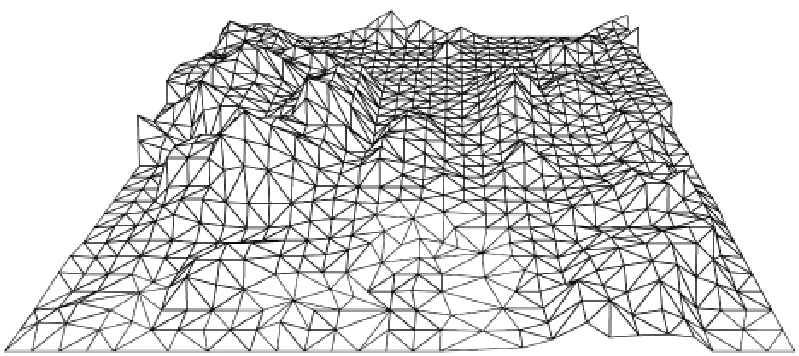
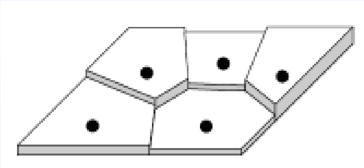


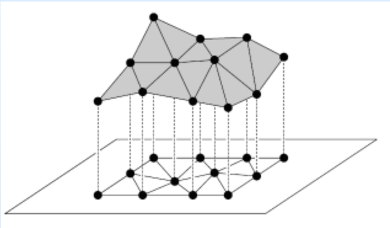
Figure 9.1
A perspective view of a terrain

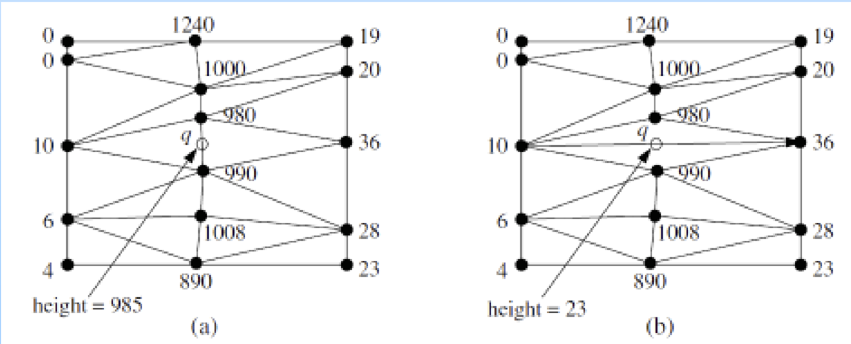
Delaunay Triangulations (2)

- 방법 1
  - 측정되지 않은 지점의 높이를 높이를 측정한 가장 가까운 지점의 높이로 지정하는 방법
  - 문제점
    - 계단식 형태의 결과 생성



- 방법 2
  - 주위의 점들을 사용해서 삼각형으로 표현
  - 고려 사항
    - 계곡에 해당하는 두 점을 연결하면 이상해 보임

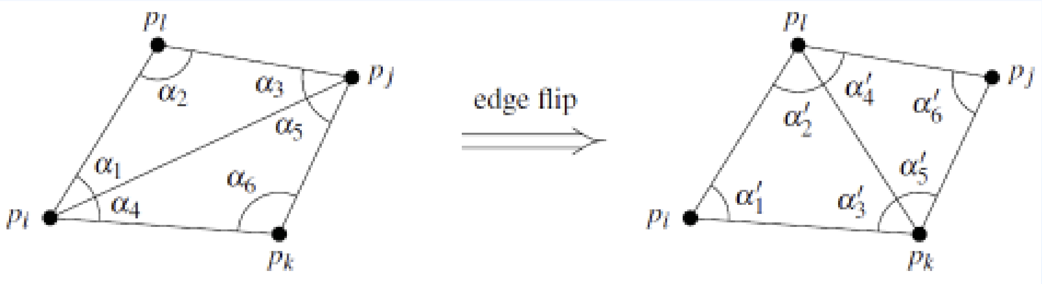




- 목표
  - 전체에서 제일 작은 각도가 크도록 삼각형 생성

Delaunay Triangulations (3)

- 각도가 작은 삼각형을 줄이는 방법
  - 대각선 edge를 제거하고 반대 대각선 edge 추가



Delaunay Triangulations (3)

- Delaunay Triangulation 생성 방법
  - 점들을 2차원에 투영해서 polygon 구성
  - Voronoi diagram 생성
  - Delaunay graph 생성
    - Voronoi cell마다 node 대응
    - 인접한 cell 사이에 edge 연결
  - 삼각형 구성
    - 3개의 점으로 구성된 원에 다른 점들이 포함되지 않도록 조정