生産情報システム工学

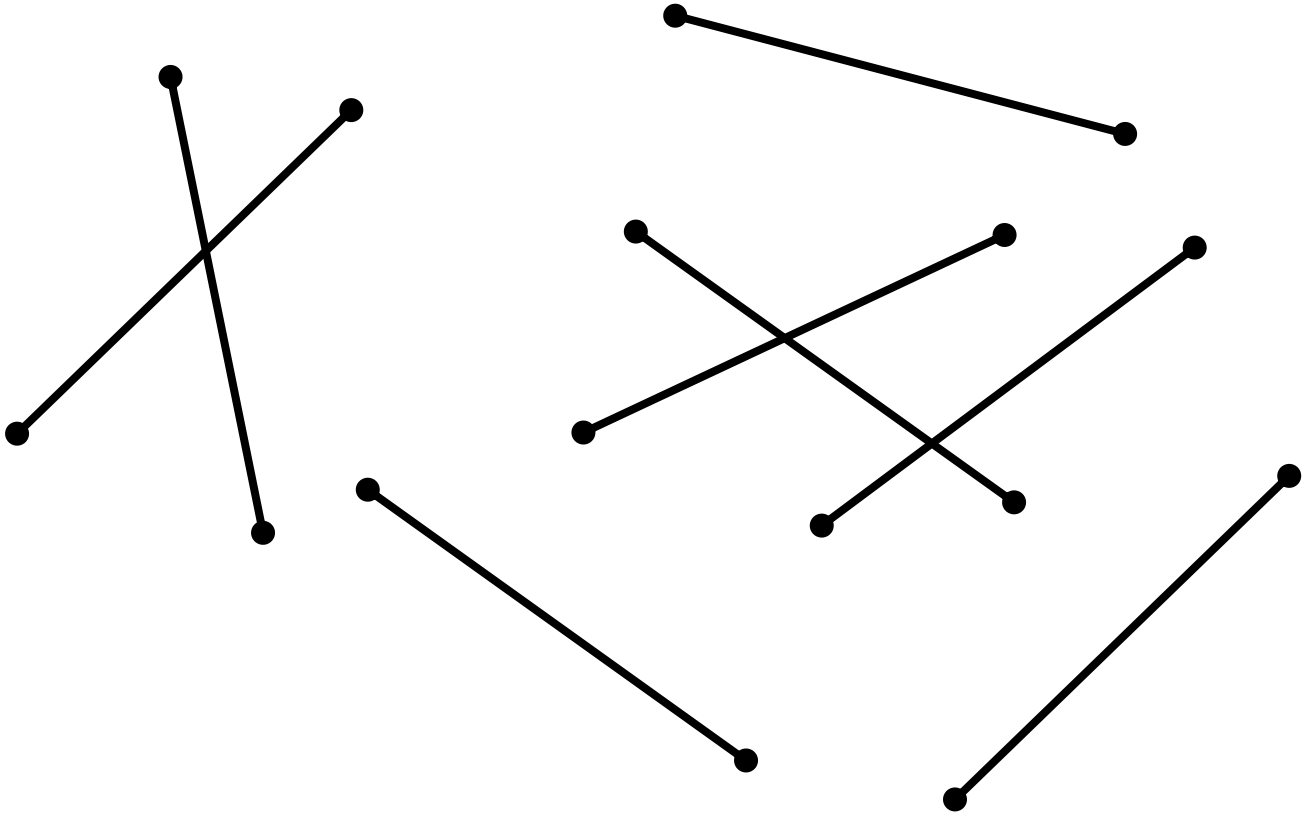
#05 交差(1)

2015/05/20(水)

溝口 知広　准教授(居室：61-408室)
mizo@cs.ce.nihon-u.ac.jp

2.0 交差

■ 問題：線分の交差はいくつあるか？



2.0 交差

■ 交差の検出が必要な分野

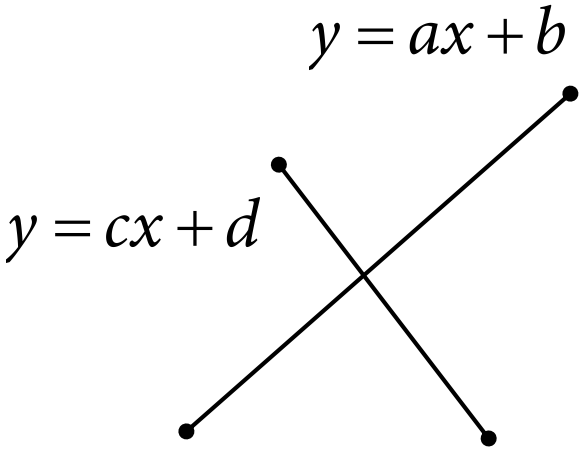
- VLSI回路の配線
- CG：隠面除去
- ロボット：障害物との接触判定
- …
- など多数

■ 交差を検出する効率的なアルゴリズムが必要

2.1 2線分の交差

■ 交差を判定する簡単な方法

- 連立方程式から交点を求める



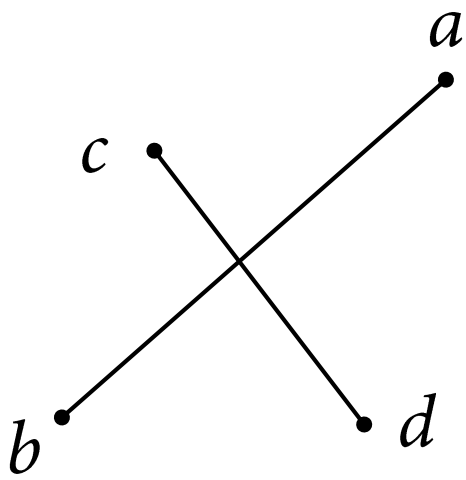
交点： $x = \dfrac{d-b}{a-c}$, $y = \dfrac{ad-bc}{a-c}$

- 交差判定ができない，もしくは困難な場合がある：
  1. 傾きが等しい場合(a=c) → 交点数が0または無限
  2. 垂直の場合 → 傾きが無限大(yの係数が0)
  3. 傾きが近い場合(a≒c) → 計算誤差が生じる

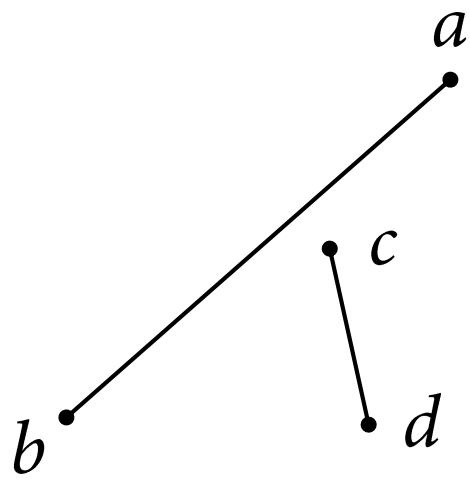
2.1 2線分の交差

■ 三角形の符号付き面積を利用する方法：

- 基本的な考え方：
  - 2本の線分abとcdが互いに交わるならば，端点cと端点dが線分abを含む直線によって分離される
  - 同様に，aとbもcdを含む直線によって分離される



交差する



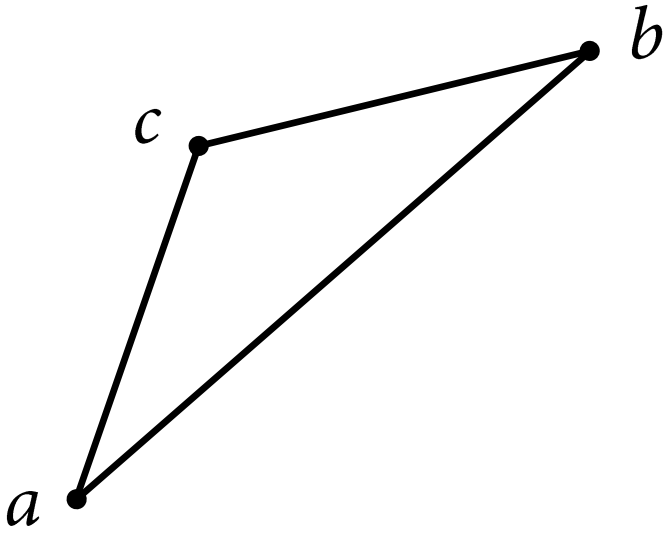
交差しない

2.1 2線分の交差

■ 三角形の符号付き面積を利用する方法：

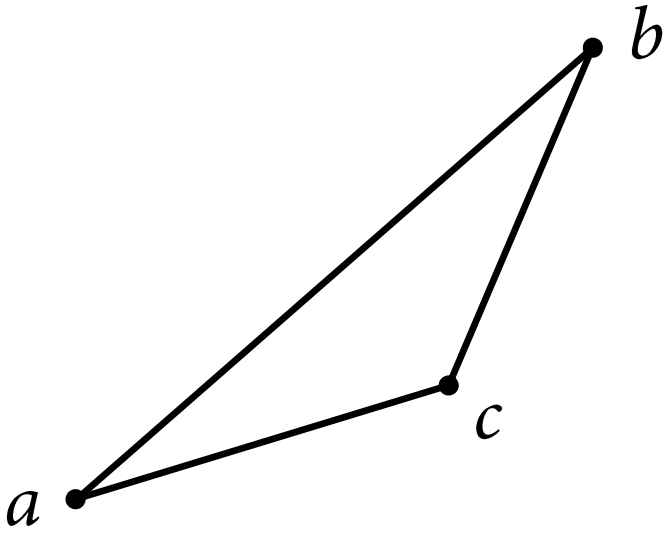
- 面積の符号は3点(a, b, c)の順序で決まる

反時計回り



面積は正

時計回り



面積は負

2.1 2線分の交差

■ 符号付き面積の算出

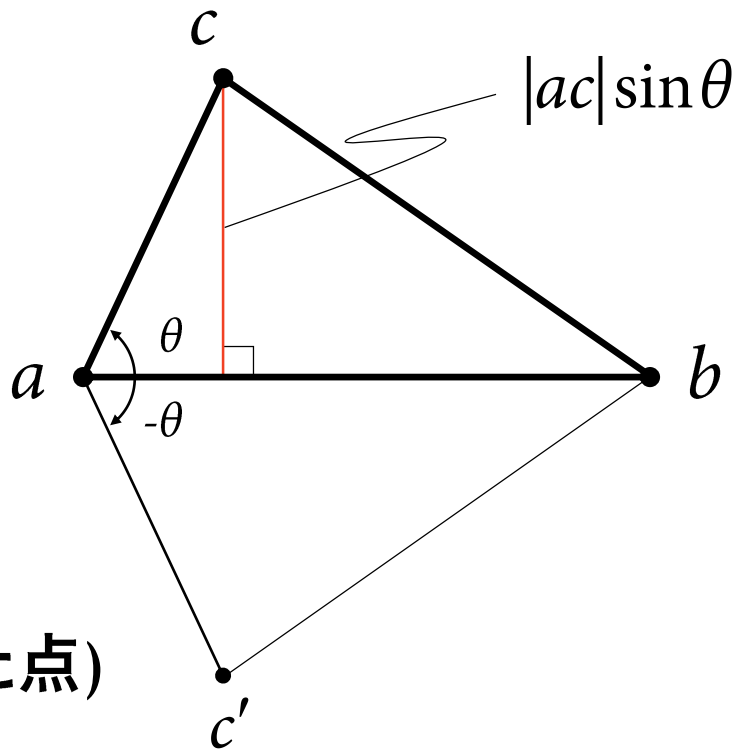
- 三角形*abc*の面積

$$S_{abc} = \frac{1}{2}|ab||ac|\sin\theta$$

- 三角形*abc'*の面積
  (*c'*は*ab*に対して反射させた点)

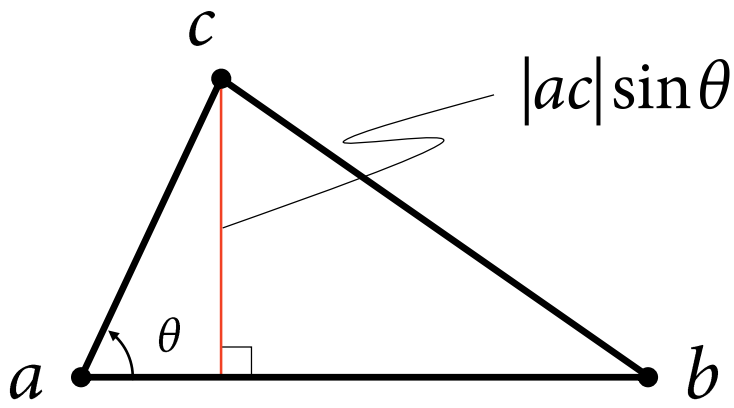
$$S_{abc'} = \frac{1}{2}|ab||ac'|\sin(-\theta)$$

$$= -\frac{1}{2}|ab||ac'|\sin\theta$$



2.1 2線分の交差

■ 外積を使った符号付き面積の算出



$$S_{abc} = \frac{1}{2} ab \times ac$$

$$= \frac{1}{2}\left(\left(x_b - x_a\right)\cdot\left(y_c - y_a\right) - \left(x_c - x_a\right)\cdot\left(y_b - y_a\right)\right)$$

2.1 2線分の交差

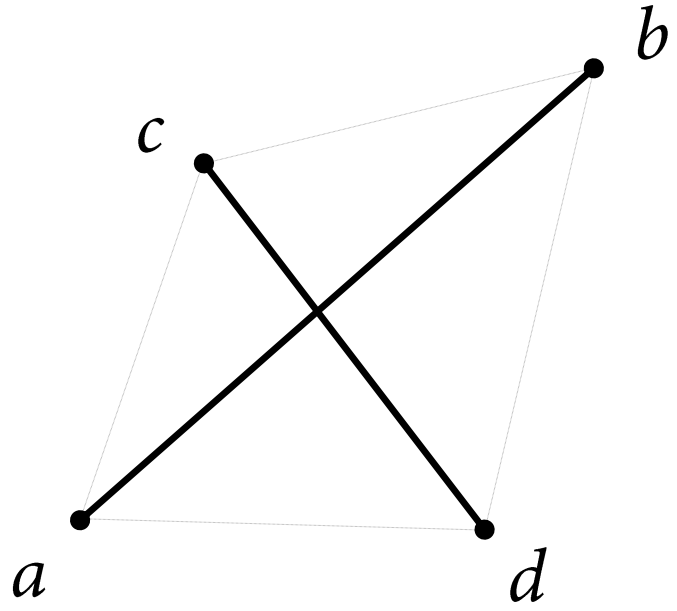
■ 三角形の符号付き面積を利用する方法：

- 三角形△*abc*と三角形△*abd*の符号付き面積が異なる符号を持てば交差すると判定する

<u>交差する場合</u>

- 3点*a*,*b*,*c*の順は反時計回り
  → 符号付き面積は正

- 3点*a*,*b*,*d*の順は時計回り
  → 符号付き面積は負



2.1 2線分の交差

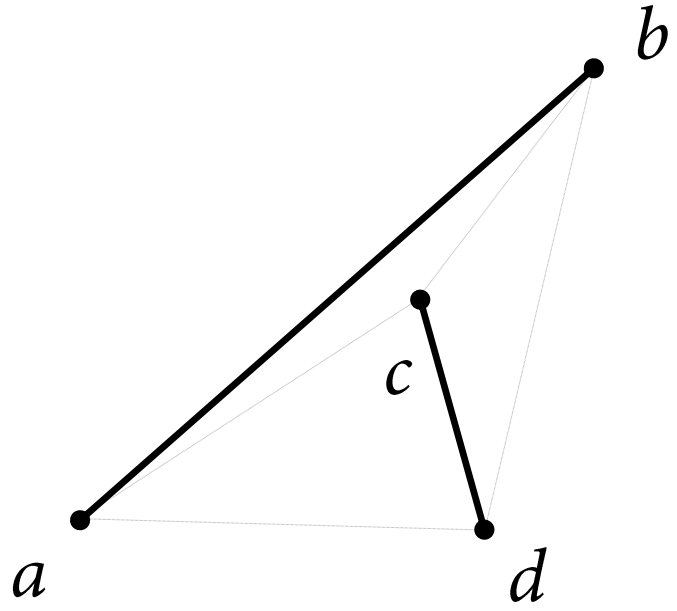
■ 三角形の符号付き面積を利用する

- 三角形△abcと三角形△abdの符号付き面積が異なる符号を持てば交差する

<u>交差しない場合</u>

- 3点a,b,cの順は時計回り
  → 符号付き面積は負

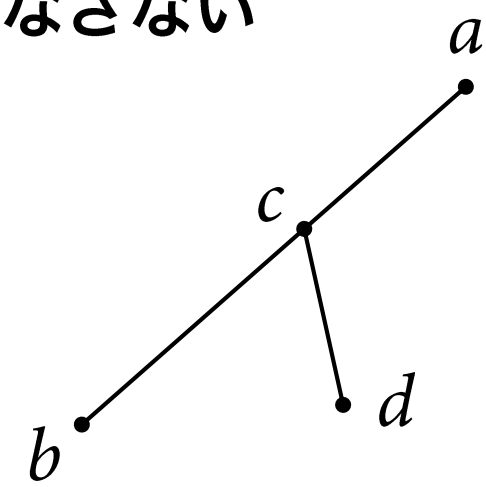
- 3点a,b,dの順は時計回り
  → 符号付き面積は負



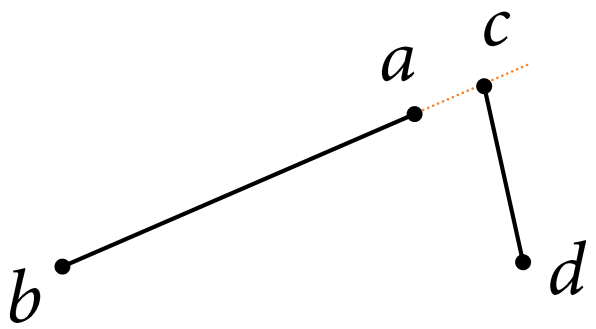
2.1 2線分の交差

■ 特殊な場合：

- 一方の線分の端点が，他方の線分上にある場合も，交差していると判定する
  - 下図の例では，三角形abcの符号付き面積はゼロになる
- 線分の延長線上にある場合は，交差しているとはみなさない



交差している

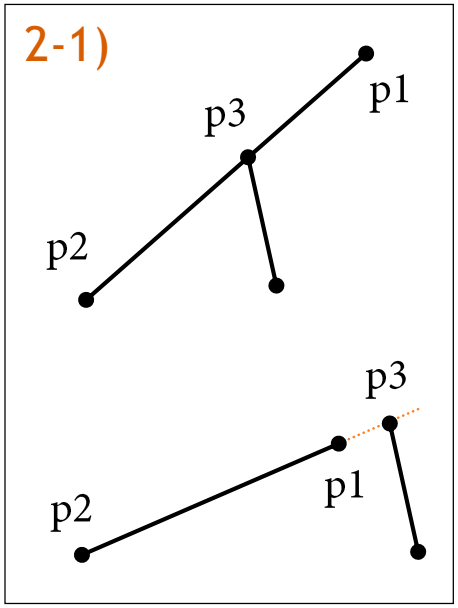


交差していない

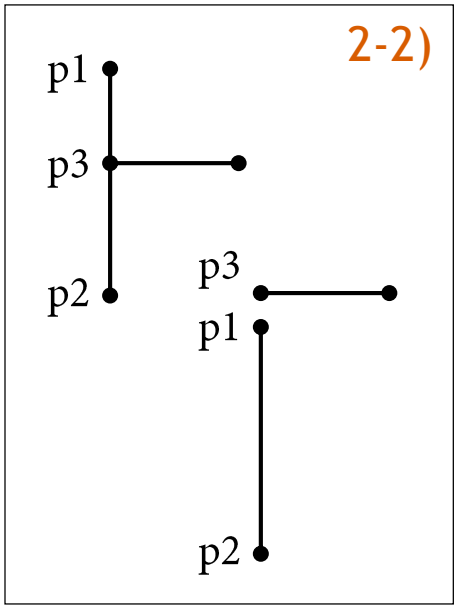
2.1 2線分の交差

```
//  3点で決まる三角形の符号付き面積を計算する
double area( struct point p1, struct point p2, struct point p3 )
{
  double  area =
    ( p1.x - p3.x )*( p2.y - p3.y ) - ( p2.x - p3.x )*( p1.y - p3.y );
  return  area;
}
```

2.1 2線分の交差

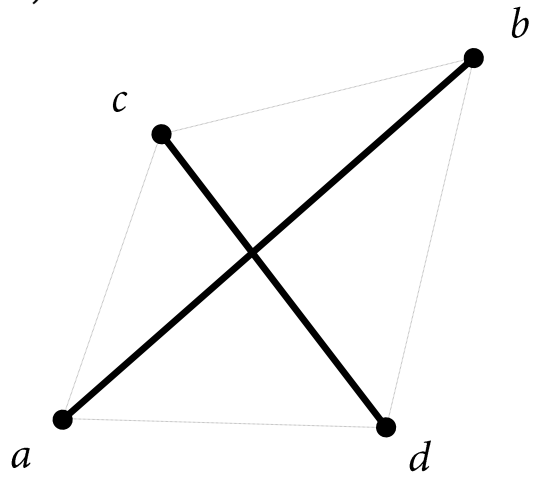



```
//  線分p1p2に点p3が乗っているかどうかを判断する
int between( struct point p1, struct point p2, struct point p3 )
{
  double tmp = area( p1, p2, p3 );          //  面積の計算

  //  1) 面積がゼロでなければ線分上にはない
  if( tmp != 0.0 )      return 0;

  //  2) 面積がゼロの場合には，直線上にはのっているので，線分上にあるかを判定する
  //  2-1) 線分p1p2が垂直でなければ，x座標を評価して判定する
  if( p1.x != p2.x ){
    if( p1.x <= p3.x && p3.x <= p2.x )                  return 1;
    else if( p2.x <= p3.x && p3.x <= p1.x )             return 1;
    else                                                return 0;
  }
  //  2-2) 線分p1p2が垂直であれば，y座標を評価して判定する
  else{
    if( p1.y <= p3.y && p3.y <= p2.y )                  return 1;
    else if( p2.y <= p3.y && p3.y <= p1.y )             return 1;
    else                                                return 0;
  }
}
```



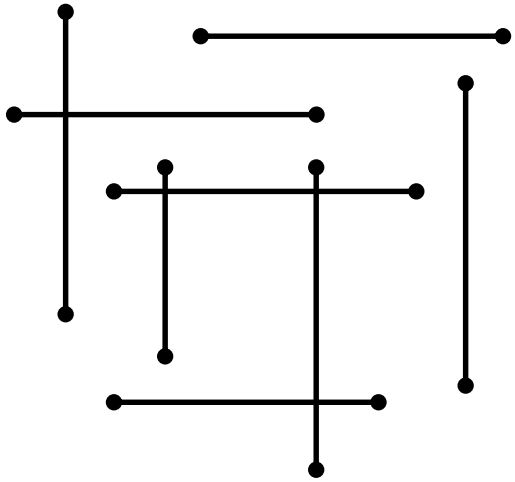
2.1 2線分の交差

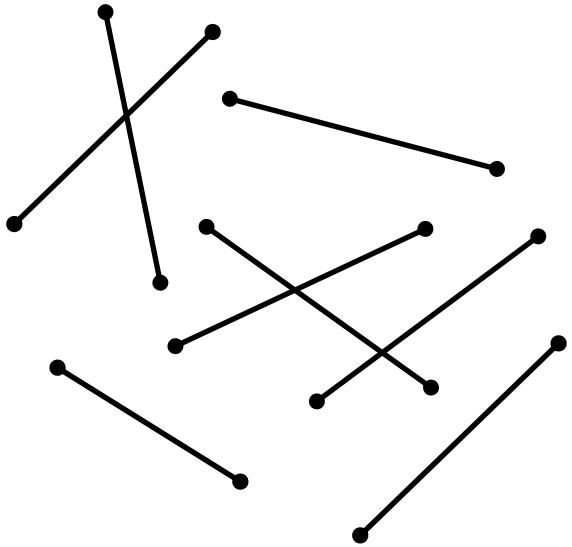
```
// 線分abとcdが交差するときは1を返し，交差しないときは0を返す
int intersect( struct point a, struct point b, struct point c, struct point d )
{
  // 1) 一方の線分上に，他方の線分のいずれかの端点が乗っている場合
  int  b1 = between( a, b, c );          // 線分ab上に点cがあるかどうか
  int  b2 = between( a, b, d );          // 線分ab上に点dがあるかどうか
  int  b3 = between( c, d, a );          // 線分cd上に点aがあるかどうか
  int  b4 = between( c, d, b );          // 線分cd上に点bがあるかどうか
  if( b1 == 1 || b2 == 1 || b3 == 1 || b4 == 1 )      return 2;

  // 2) 交差する場合
  double  a1 = area( a, b, c );
  double  a2 = area( a, b, d );
  double  a3 = area( c, d, a );
  double  a4 = area( c, d, b );
  if( a1*a2 < 0.0 && a3*a4 < 0.0 )          return 1;

  // 3) 交差しない場合
  else       return 0;
}
```



2.2 n本の線分の交差

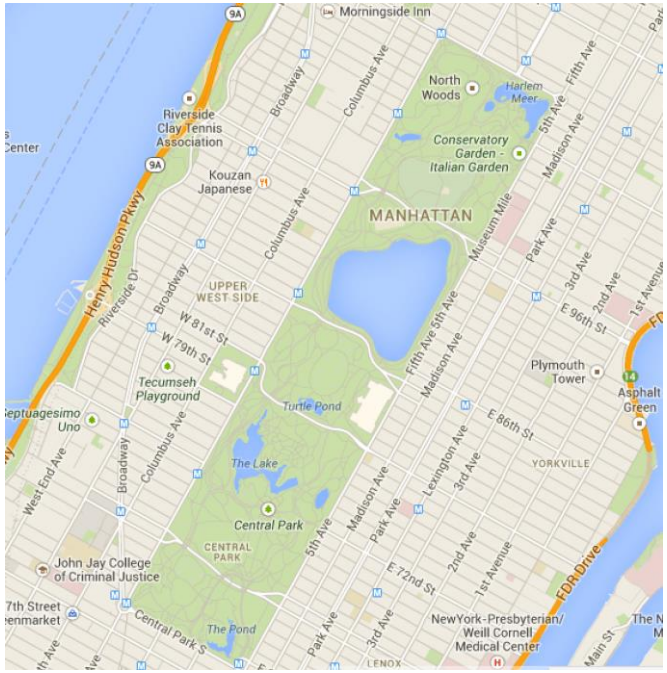
水平・垂直な線分	一般の線分





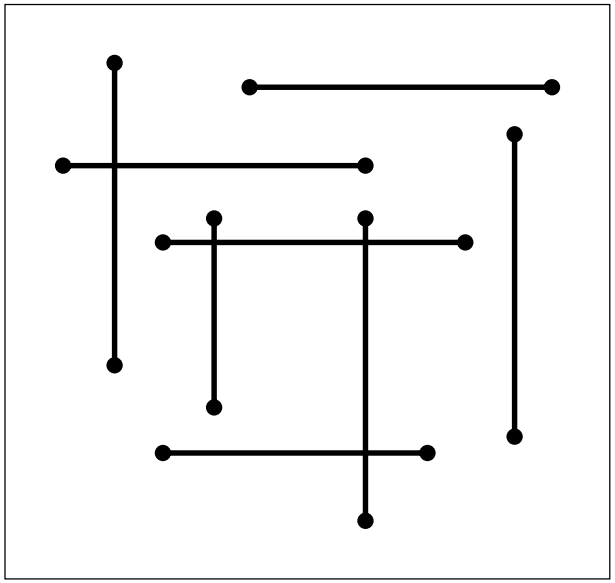
2.2.1 水平・垂直な線分

■ 「マンハッタン幾何学」とも呼ばれる



2.2.1 水平・垂直な線分

■ 問題：水平・垂直なn本の線分の中に，交差はいくつあるか？

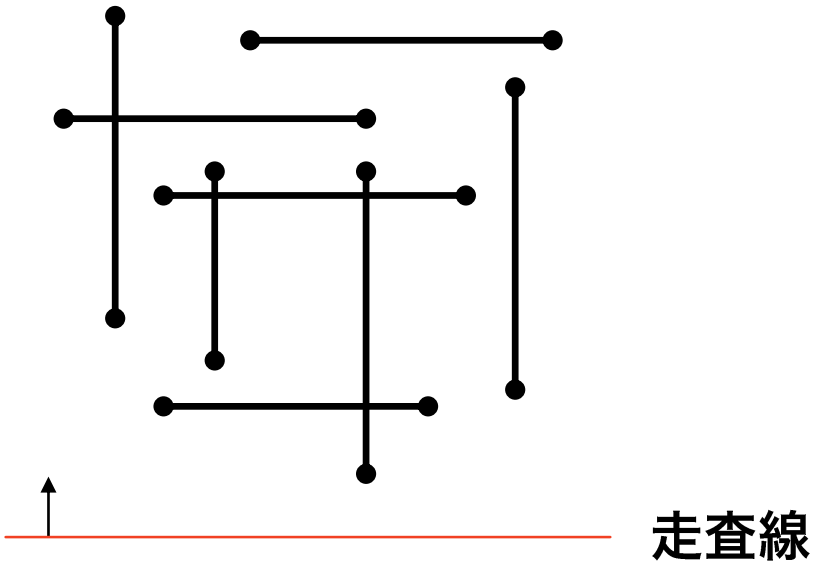


■ すぐに思いつく簡単な方法

- n本の線分から2本ずつ選び，それらの交差を調べる
- 計算量：$O(n^2)$
- nが大きい場合には使えない
  - 参考：バブルソート $O(n^2)$ vs クイックソート $O(n\log n)$
- もっと効率的な方法は？

2.2.1 水平・垂直な線分

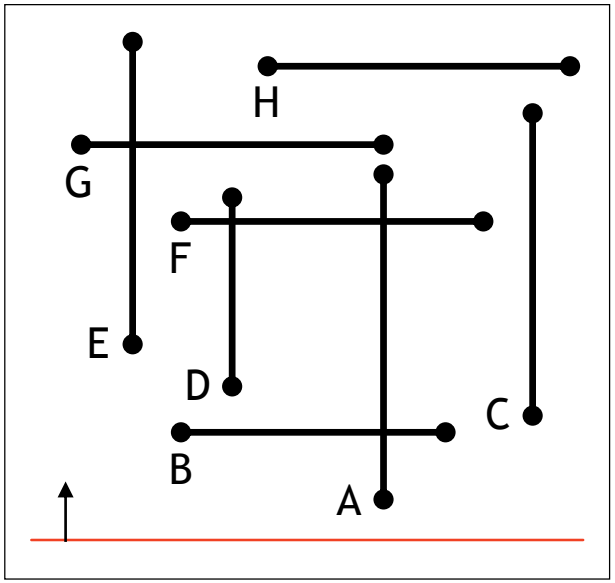
- 平面走査法(plane sweep)
  - 1本の水平，または垂直な直線(=走査線)を平面上を移動させながら，線分の交差を見つける

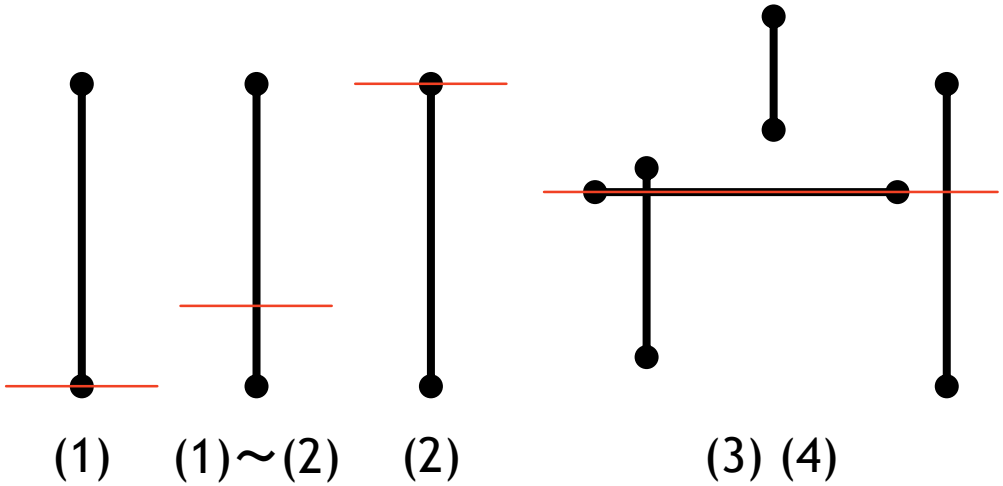


2.2.1 水平・垂直な線分

■ 平面走査法(plane sweep)の基本的な考え方

1. 垂直線分と出会う時，垂直線分が走査線上に現れる
2. 垂直線分と離れる時，上記の点が走査線から消える
3. 水平線分と出会う時，捜査線と一瞬だけ重なる
4. 水平線分と出会う時，この水平線分の区間内に垂直線分の点があるか？　あるならばそれが交差である！

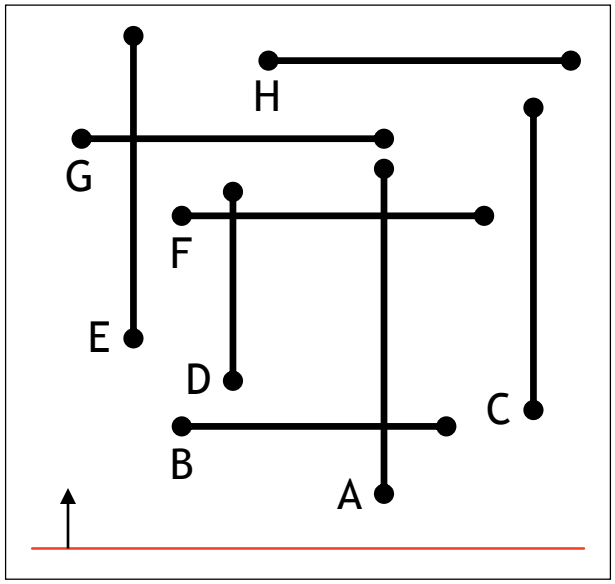




2.2.1 水平・垂直な線分

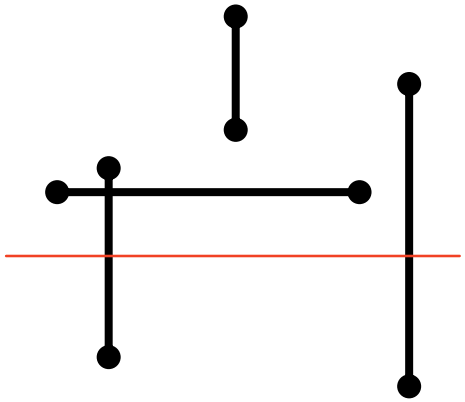
■ イベント計画(event schedule)

- 走査線のイベントポイント(停止位置)を順に教えてくれる
- y座標順に線分の端点を整列させる
  → A, B, C, D, E, F, D, A, G, C, H, E
- 垂直線分は2回(上下の端点)，水平線分は1回
- リストで実現する



2.2.1 水平・垂直な線分

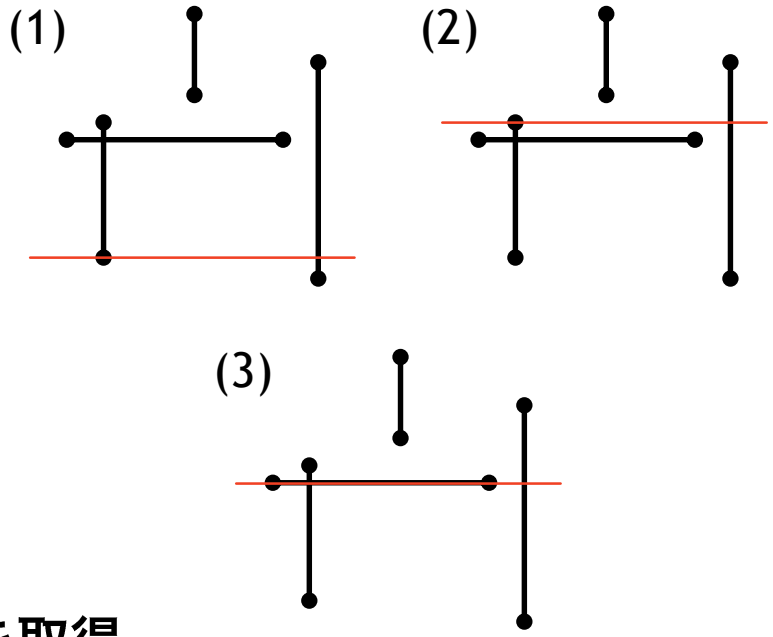
- **走査線計画**(sweep-line schedule)
  - 走査線と垂直成分との交差状況を適切に表現するデータ構造
  - 水平線分と出会った時に，それと交差する垂直線分を迅速に見つけ出すことができる構造
  - 走査線計画は垂直線分の各イベントポイントで変更され，水平線分の各イベントポイントで交差を見つけるために使う



2.2.1 水平・垂直な線分

■ 走査線計画(sweep-line schedule)

- 2分探索木で実現する



1. 垂直線分の下端点
   → x座標を木に挿入
   (交差の候補とする)

2. 垂直線分の上端点
   → x座標を木から削除

3. 水平線分
   → 2端点のx座標(x1, x2)を取得
   → 区間探索(x1〜x2の範囲)
   (水平線分と交わる垂直線分(=交差)を木から探す)

2.2.1 水平・垂直な線分

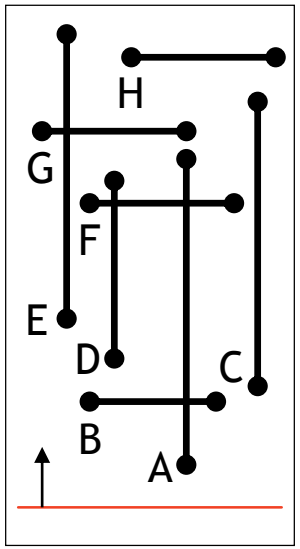
■ アルゴリズム

1. 線分の端点をy座標の順に整列させリスト*L*に入れる
2. 2分探索木*T*を空にする
3. 走査線を下から上に移動し(*L*から順に点を取り出し), *L*の各点に対して以下の操作を行う
   a. 走査線に出会う端点が垂直線分の下端点ならば，その線分を*T*に挿入する．上端点ならば，*T*から削除する．
   b. 捜査線が水平線分と出会うならば，*T*に対して線分の両端点の*x*座標を区間の両端として区間探索を行い，その水平線分と交わる垂直線分(=交差)を報告する

2.2.1 水平・垂直な線分

■ 実行例(1/3)

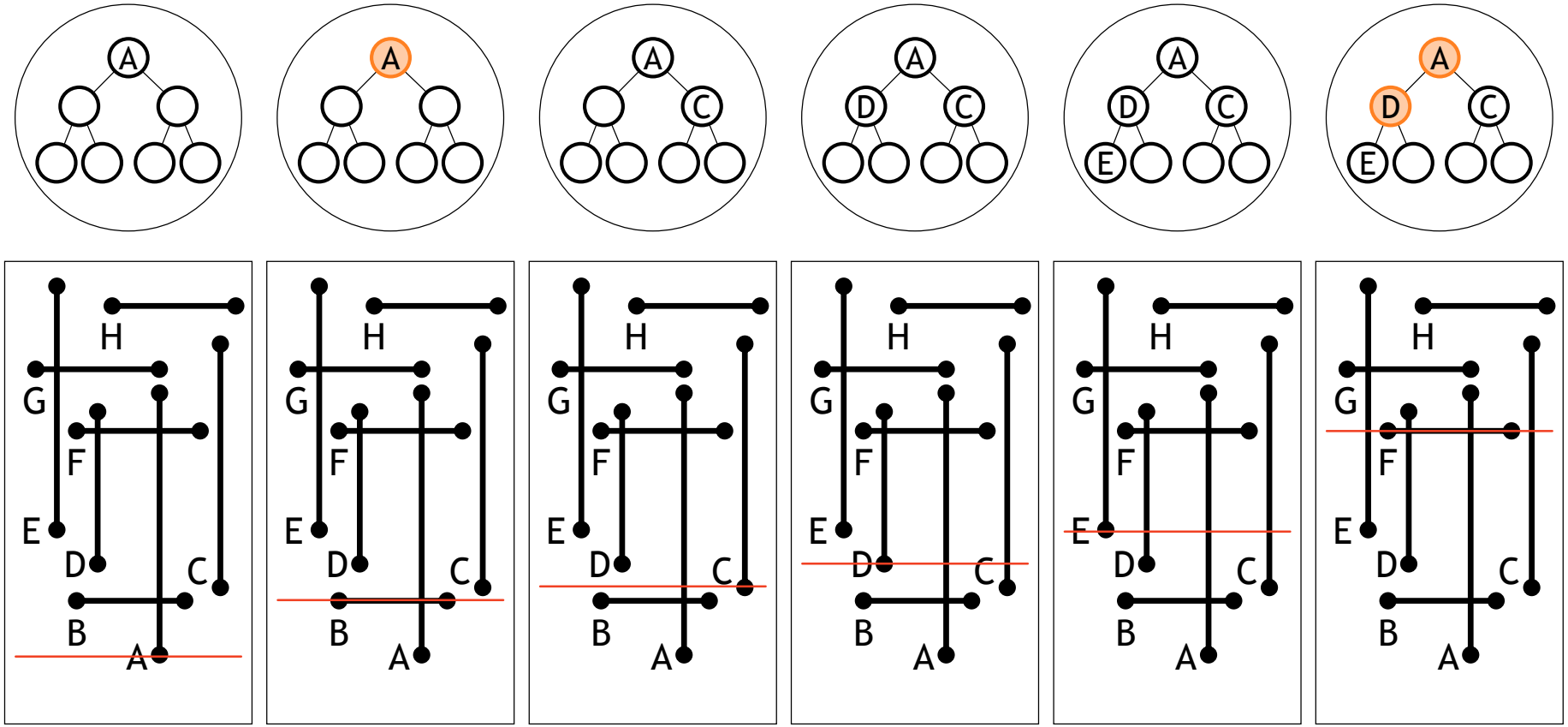
リストL	A	B	C	D	E	F	D	A	G	C	H	E



初期状態

2.2.1 水平・垂直な線分

■ 実行例(2/3)



Aを挿入

Bの区間探索
Aとの交差検出

Cを挿入

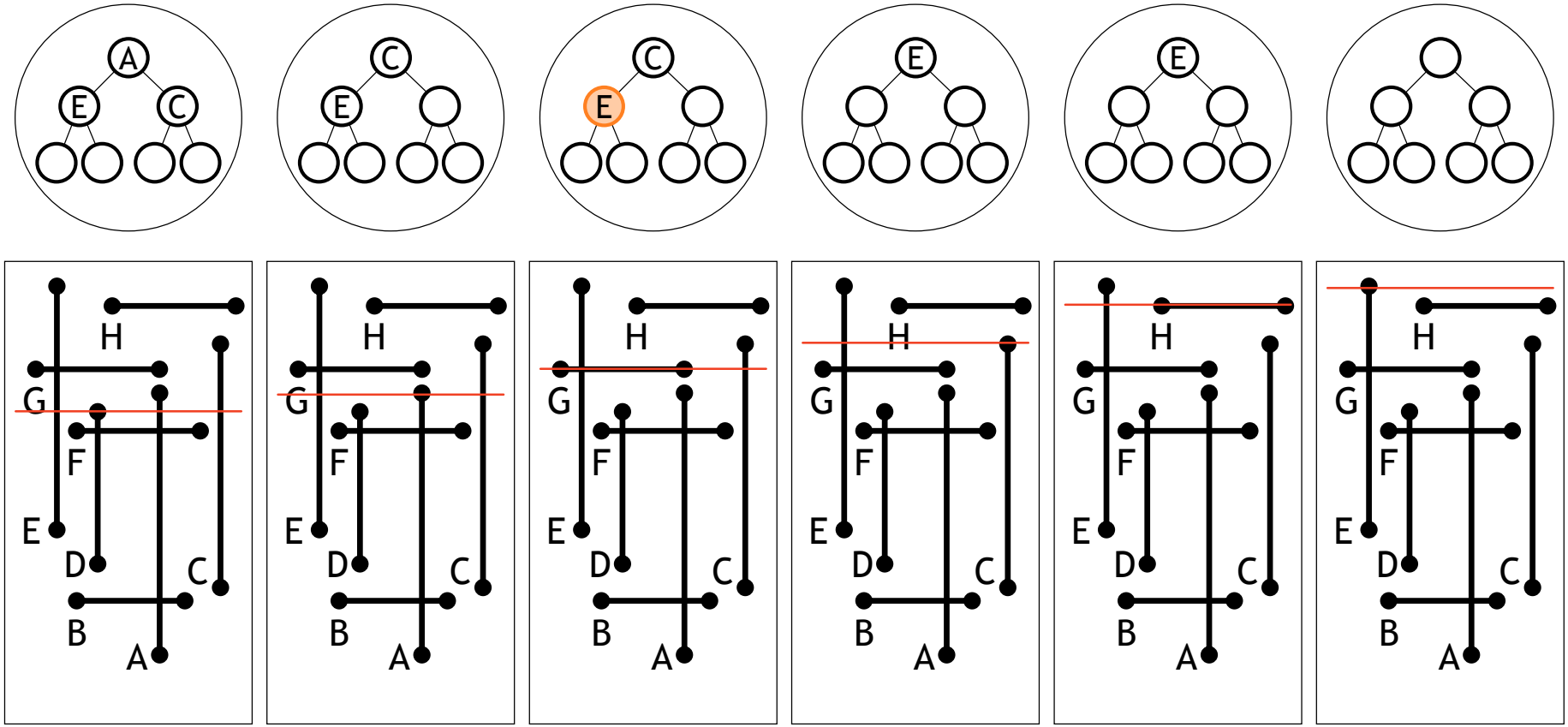
Dを挿入

Eを挿入

Fとの区間探索
A,Dとの交差検出

2.2.1 水平・垂直な線分

■ 実行例(3/3)



Dを削除　Aを削除　Gとの区間探索 Eとの交差検出　Cを削除　Hとの区間探索　Eを削除

2.2.1 水平・垂直な線分

```
#define NUM 8

// 端点を表す構造体
struct point{
  int x;
  int y;
};

// 根ノードの点
struct point dammy;

// 線分を表す構造体
struct line_segment{
  struct point p1;
  struct point p2;
};

// 線分
struct line_segment ls[NUM];
```

```
// リストの要素を表す構造体
struct element{
  struct point* p1;
  struct point* p2;
};

// リストの要素とその個数
struct element e[16];
int n_elements = 0;

// ノードを示す構造体
struct node{
  struct point* t;     // 垂直成分の下端点
  struct node* lson;   // 左の子
  struct node* rson;   // 右の子
};

// ノード
struct node nil;       // 葉ノードの子
struct node root;      // 根ノードの親
```

2.2.1 水平・垂直な線分

```
// リストの作成
// p1を下端点，p2を上端点とする
n_elements = 0;
for (int i = 0; i<NUM;i++ ){
  // 垂直成分
  if ( ls[i].p1.y != ls[i].p2.y ){
    e[n_elements].p1 = &(ls[i].p1);
    e[n_elements].p2 = &(ls[i].p2);
    n_elements++;
    e[n_elements].p1 = &(ls[i].p2);
    e[n_elements].p2 = &(ls[i].p1);
    n_elements++;
  }
  // 水平成分
  else{
    e[n_elements].p1 = &(ls[i].p1);
    e[n_elements].p2 = &(ls[i].p2);
    n_elements++;
  }
}
```

2.2.1 水平・垂直な線分

```
// 走査線を下から上に平面上を平行移動しリストの各点に対して以下の操作を行う
for (int i = 0; i < n_elements;i++){

    // 垂直線分の下端点の木への挿入
    if (e[i].p1->y < e[i].p2->y)
        insert(e[i].p1);

    // 垂直線分の下端点の木からの削除
    else if (e[i].p2->y < e[i].p1->y)
        remove(e[i].p2);

    // 水平線分の2端点のx座標による区間探索
    else{

        if ( e[i].p1->x < e[i].p2->x )
            tree_interval( &root, e[i].p1->x, e[i].p2->x);
        else
            tree_interval( &root, e[i].p2->x, e[i].p1->x);
    }
}
```

2.2.1 水平・垂直な線分

```
int tree_interval(struct node *p, int x1, int x2)
{
    if ( p != &nil){
        //  区間内の場合
        if ( x1 <= p->t->x && p->t->x <= x2 ){
            printf("下端点(%d, %d)の垂直成分はx座標の区間[%d, %d]の水平線分と交わる¥n",
            p->t->x, p->t->y, x1, x2);
            tree_interval( (p->lson), x1, x2);
            tree_interval( (p->rson), x1, x2);
        }
        //  区間外の場合
        else{
            //  区間の左ならば
            if ( p->t->x < x1 )              tree_interval( p->rson, x1, x2);
            //  区間の右
            else if ( x2 < p->t->x )         tree_interval( p->lson, x1, x2);
        }
    }
    return 1;
}
```