計算幾何入門

北海道大学 B4
知能ソフトウェア研究室
山本 大聖 (tada)

本日の流れ

1. 事前知識

2. 典型アルゴリズム（点，ベクトル，線分）

3. 平面走査法

本日の流れ

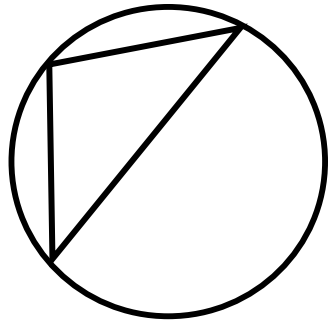
1. 事前知識

2. 典型アルゴリズム（点，ベクトル，線分）

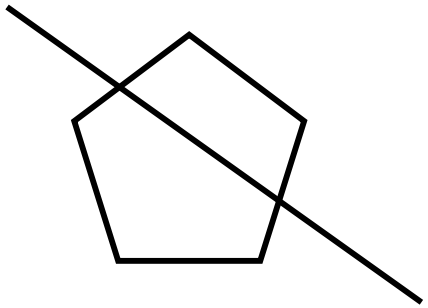
3. 平面走査法

計算幾何とは？

- 平面・空間上の図形を扱う問題



- 問題のバラエティが広く，実装が重い傾向がある
- 今回取り扱うのは平面幾何のみ



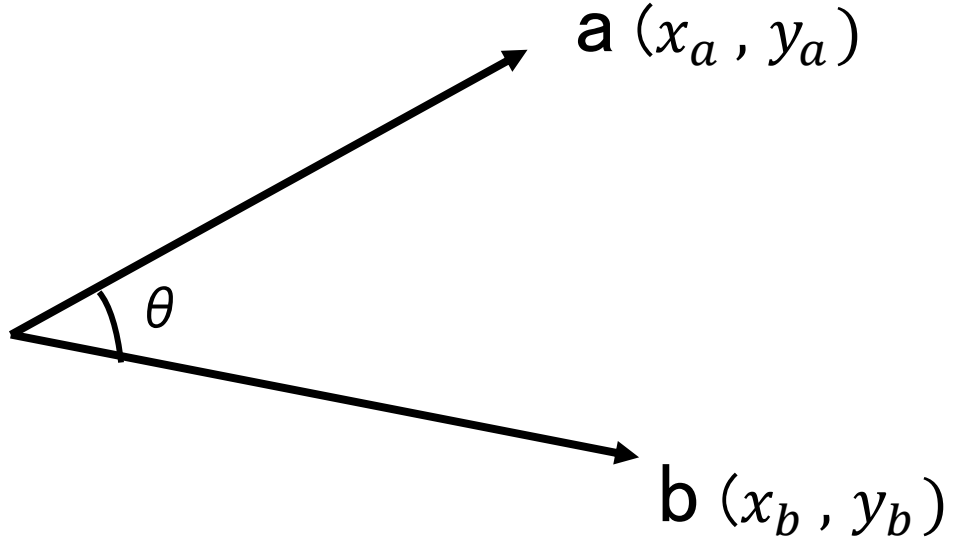
内積

- 2つのベクトルa, bの内積

$$dot(a,b) = x_a * x_b + y_a * y_b$$

- a, bのなす角をθとすると,

$$dot(a,b) = |a| * |b| * cos\,\theta$$



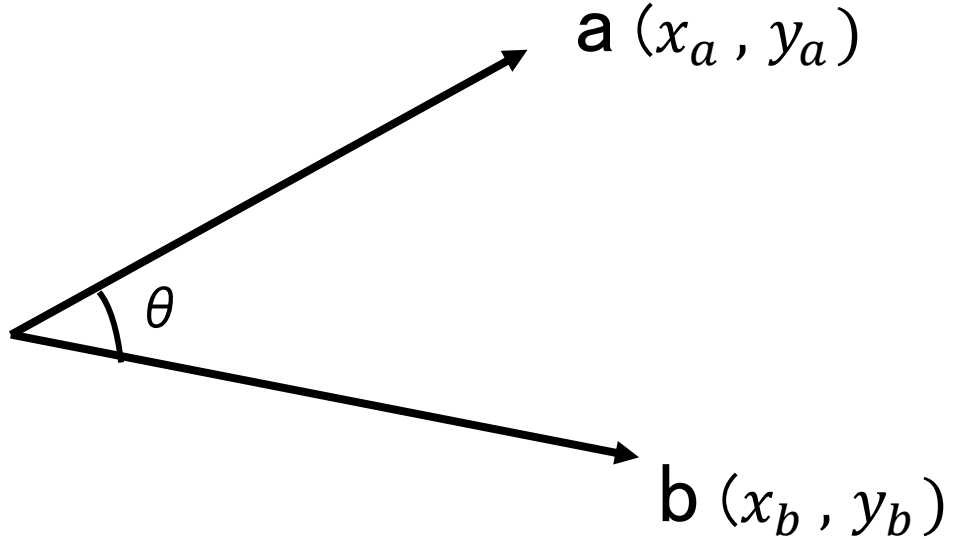
外積

- 2つのベクトルa, bの外積

$$cross(a,b) = x_a * y_b - y_a * x_b$$

- a, bのなす角をθとすると,

$$cross(a,b) = |a| * |b| * sin\,\theta$$



実装上の注意点

- 浮動小数点数型で計算するときには，誤差が生じる

  例) 100000000000.0f * 2.0f = 199999995904.0 など

そのため，非常に小さい値(EPSILON)を用いてその誤差を埋める必要がある

$$a == 0 \quad \rightarrow \quad abs(a) < EPS$$

$$a \geq 0 \quad \rightarrow \quad a > -EPS$$

$$a > 0 \quad \rightarrow \quad a > EPS$$

本日の流れ

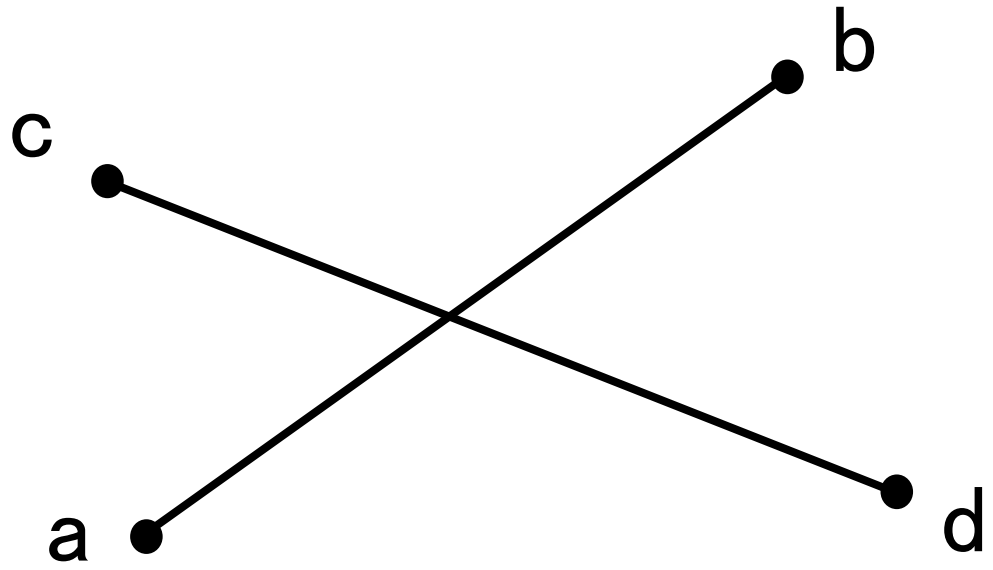
1. 事前知識

2. 典型アルゴリズム（点，ベクトル，線分）

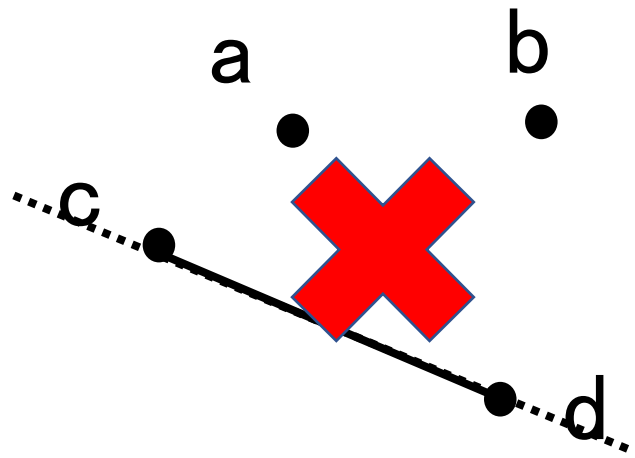
3. 平面走査法

線分と線分の交差判定

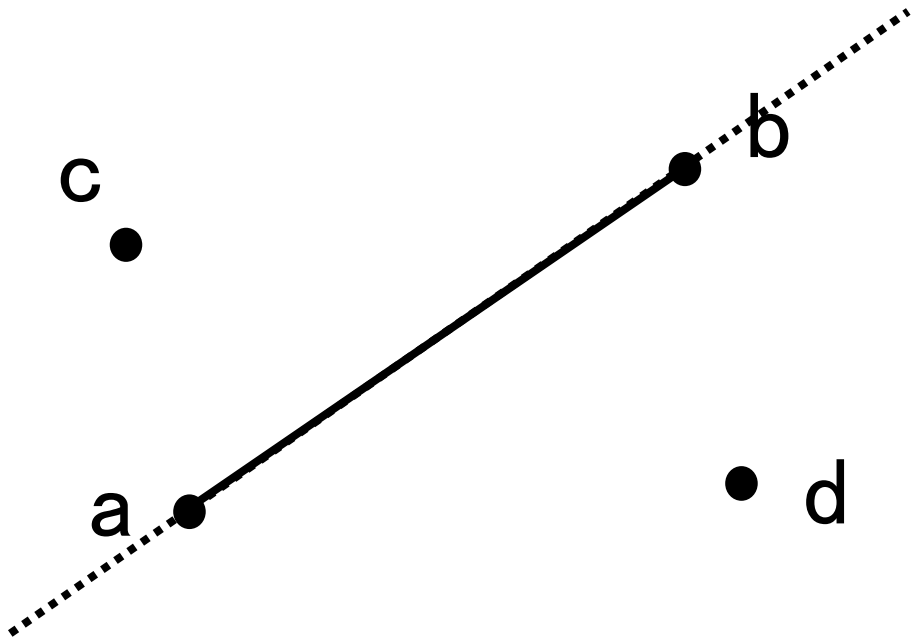
- 線分abと線分cdは交差しているかどうか

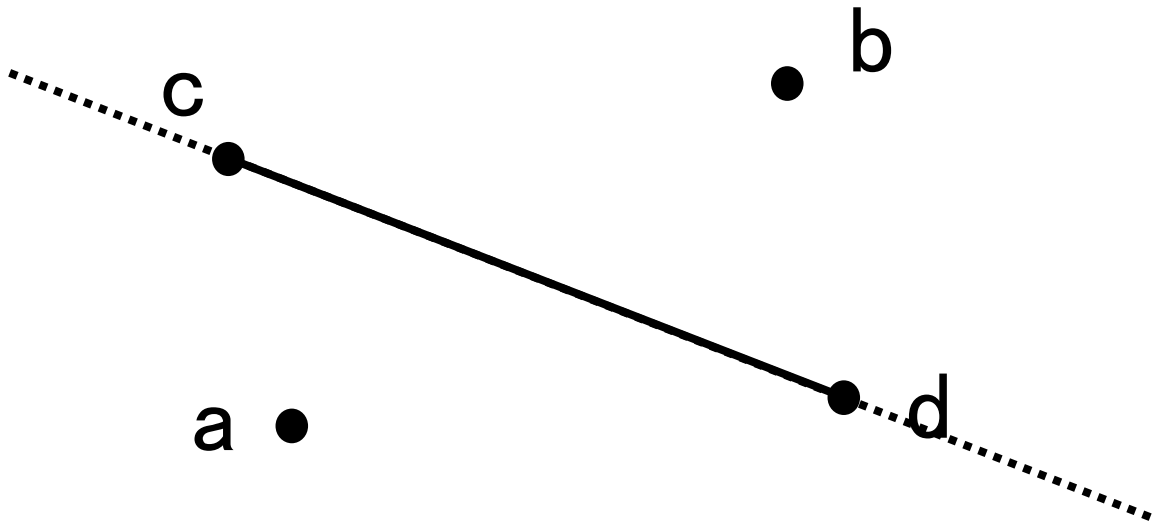


線分と線分の交差判定



- 下記の2つを満たすときに線分abと線分cdは交差している
  - 点c，点dは直線abからみて両側にある
  - 点a，点bは直線cdからみて両側にある

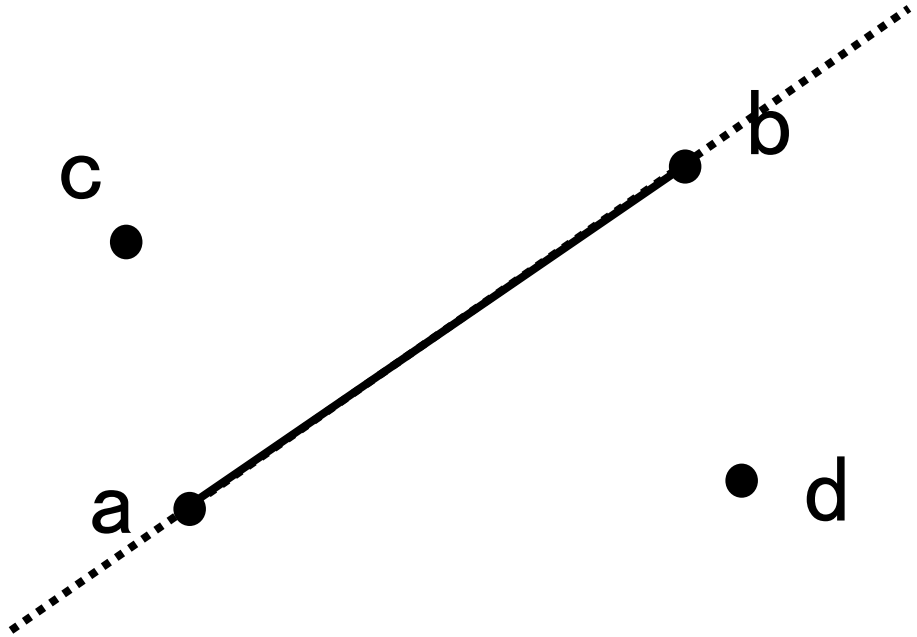


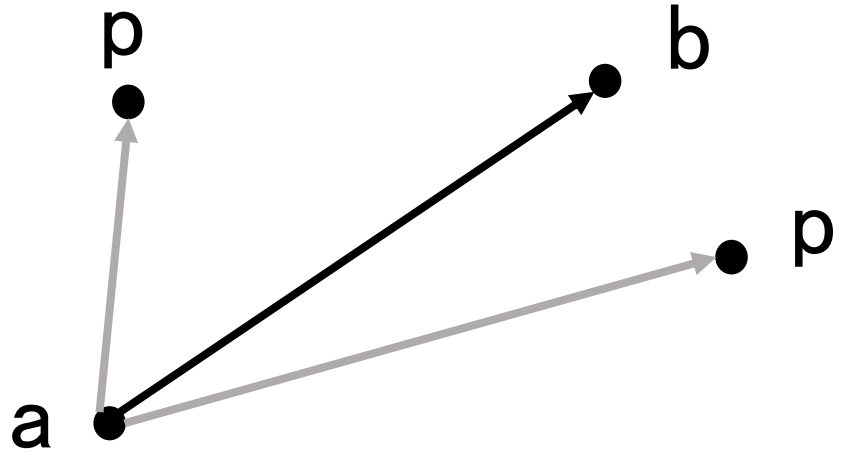


線分と線分の交差判定

- 点c，点dは直線abからみて両側にある

- 外積を用いることで判定できる





cross(b − a, p − a) < 0 なら
点pは線分abからみて右側にある

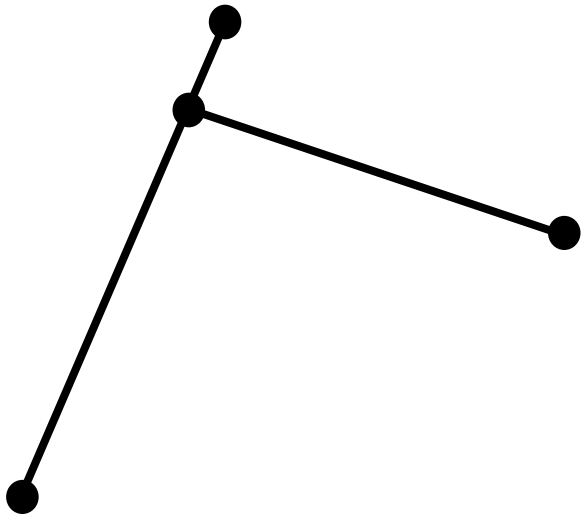
cross(b − a, p − a) > 0 なら
点pは線分abからみて左側にある

だが、cross(b − a, c − a) * cross(b − a, d − a) < 0 では不十分

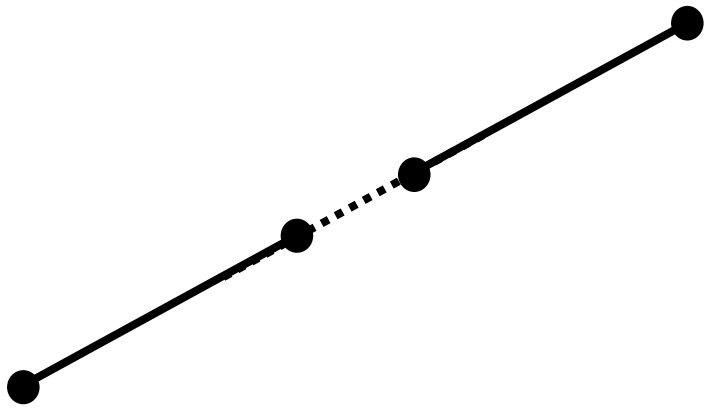
線分と線分の交差判定

- コーナーケースに注意する必要がある

線分上にもう一つの線分の端点がある



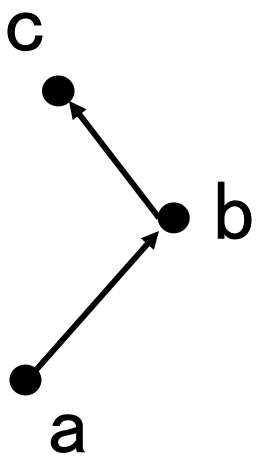
2つの線分が同一直線上にある



CCW関数（CounterClockWise）

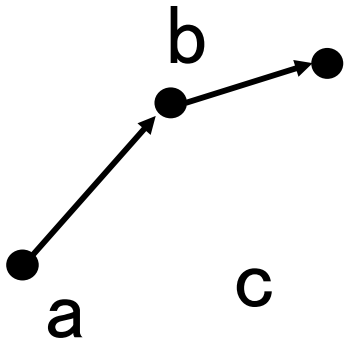
- 3点の位置関係に関する関数
- 5つのパターンに応じて値を返す（実装によりけり）

① a→bに対し，b→cが反時計回りに進む



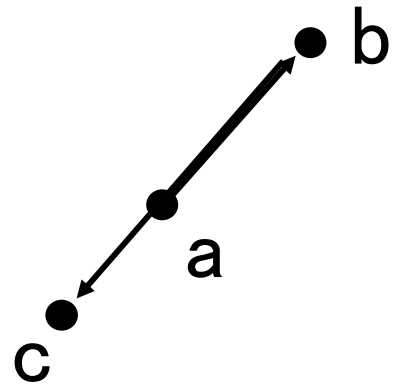
return +1

② a→bに対し，b→cが時計回りに進む



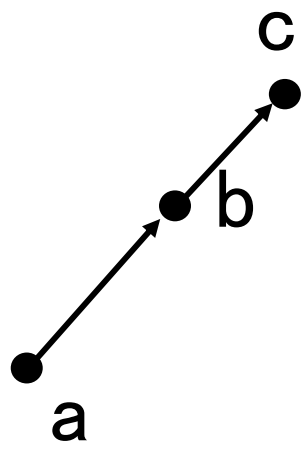
return −1

③ a→bに対し，b→cが逆方向に進む



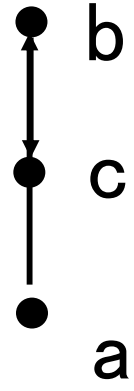
return +2

④ a→bに対し，b→cが同一方向に進む



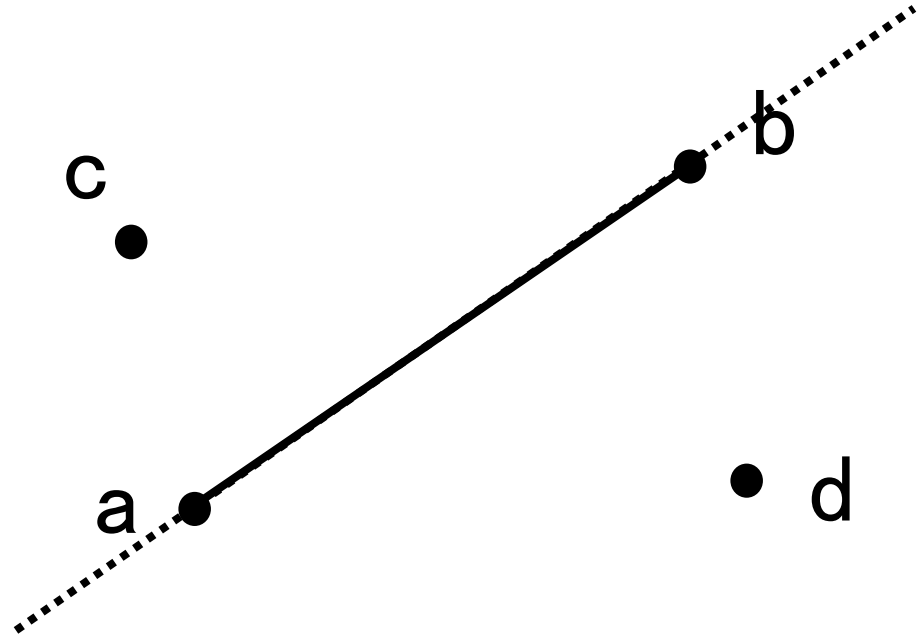
return −2

⑤



return 0

CCW関数（CounterClockWise）



- 点c，点dは直線abからみて両側にある

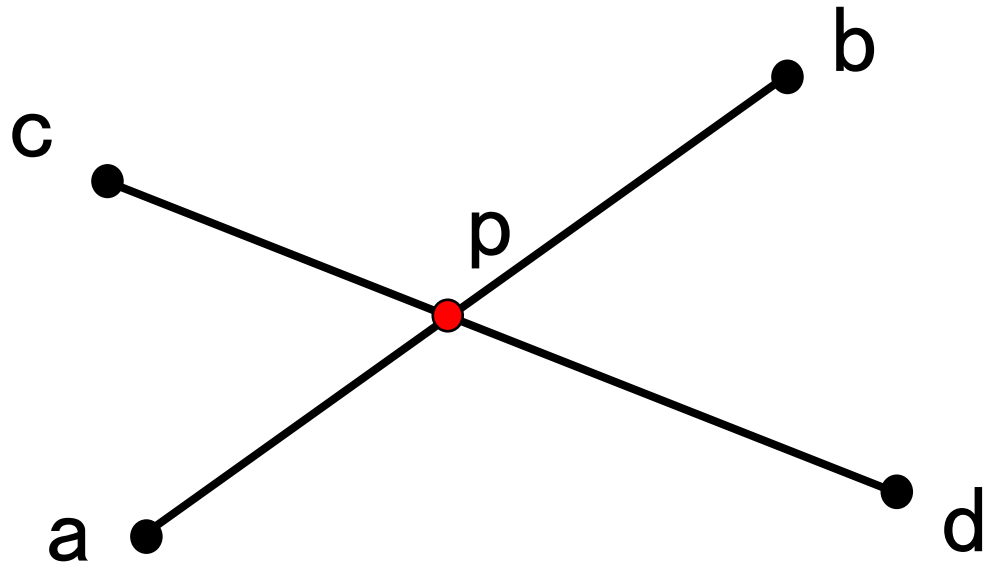
- CCWを用いると，

CCW(a, b, c) * CCW(a, b, d) <= 0 で綺麗に求めることができる

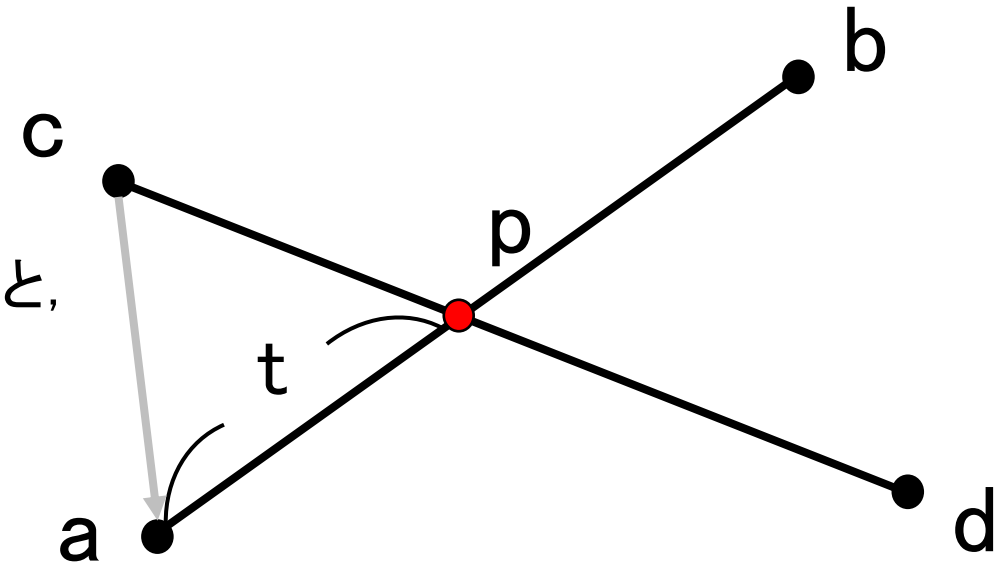
```
int ccw(const Point& a, const Point& b, const Point& c) {
    if (cross(b - a, c - a) > EPS) return +1;
    if (cross(b - a, c - a) < -EPS) return -1;
    if (dot(b - a, c - a) < -EPS) return +2;
    if (norm(b - a) + EPS < norm(c - a)) return -2;
    return 0;
}
```

線分と線分の交点

- 線分abと線分cdの交点の座標を求める



線分と線分の交点



- 線分abの長さに対する線分apの長さを t とすると,
交点pの座標は

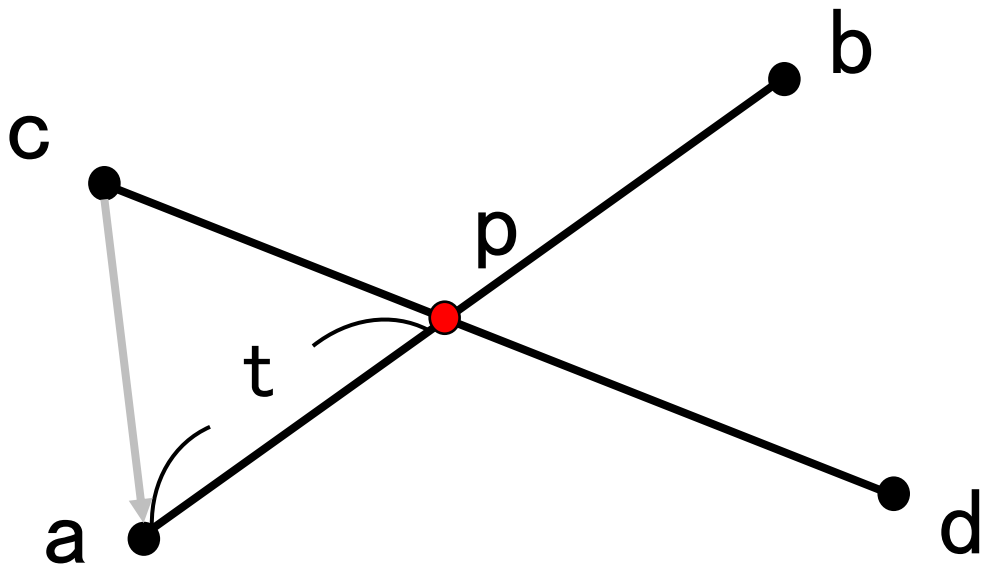
$$p = a + t * (b - a)$$ となる

- 交点pは線分cd上にあるので,
ベクトルcpとベクトルcdの外積は0となる

$$cross(p - c, d - c) = 0$$

$$cross(a - c + t * (b - a), d - c) = 0$$

線分と線分の交点



$$cross(a - c + t * (b - a), d - c) = 0$$

$$cross(a - c, d - c) + t * cross(b - a, d - c) = 0$$

$$t = -\frac{cross(a - c, d - c)}{cross(b - a, d - c)}$$

$$t = \frac{cross(a - c, d - c)}{cross(d - c, b - a)}$$

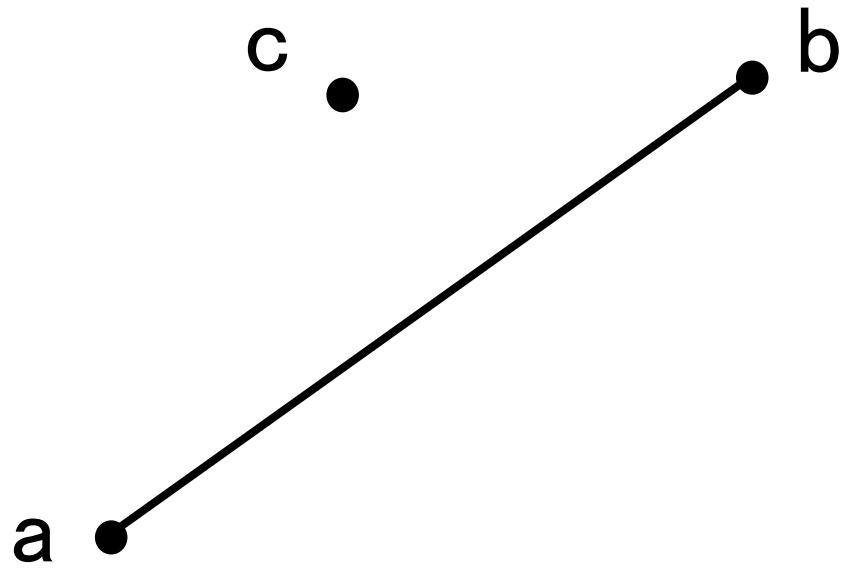
以上より

$$p = a + \frac{cross(a - c, d - c)}{cross(d - c, b - a)} * (b - a)$$

（面積比の考えで導出する方法もあります）

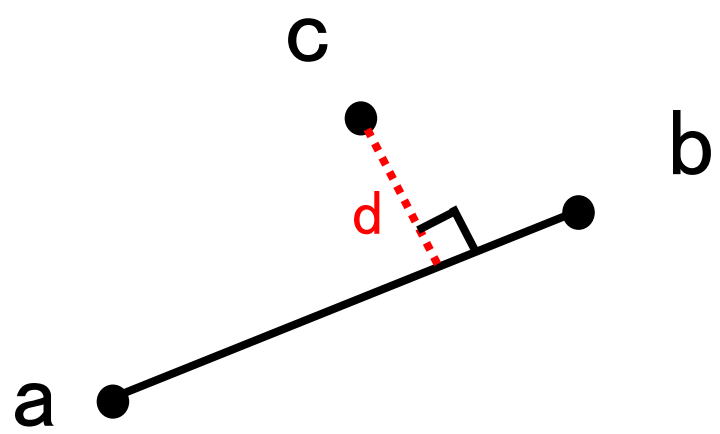
点と線分の距離

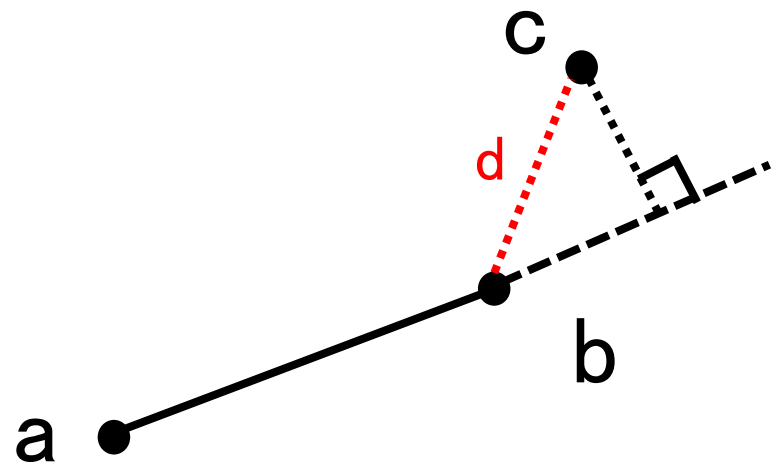
- 点cから線分abまでの距離を求める



点と線分の距離

- 基本的な考え
  - 点cから線分ab上に垂直に下せる点があるならそれが最短距離
  - それがないなら点cと線分の端点への距離の小さい方が最短距離

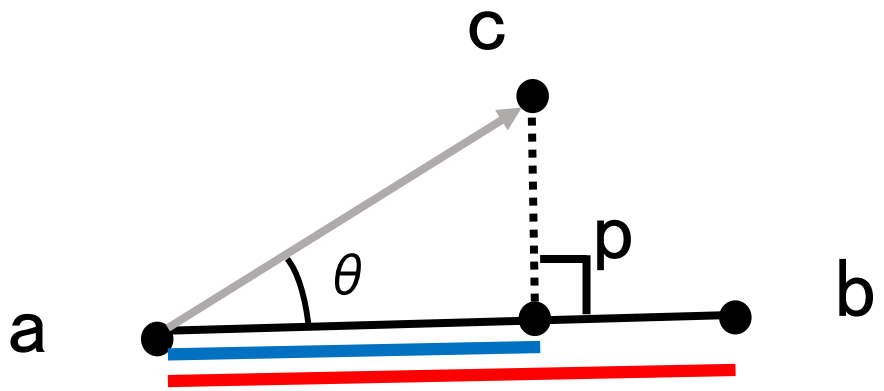




点と線分の距離

- 点cから直線abに垂直に下ろした点pを求める

内積を利用（正射影）



$$dot(c-a, b-a) = |\boldsymbol{b-a}| * |\boldsymbol{c-a}| * \boldsymbol{\cos\theta}$$

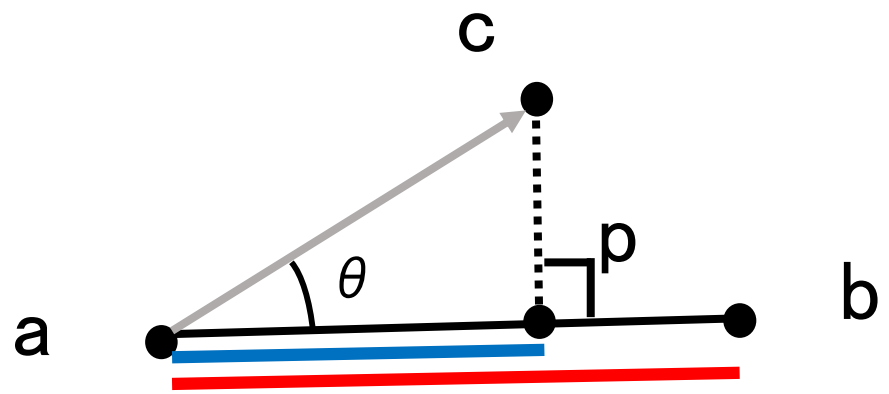
$$|\boldsymbol{c-a}| * \boldsymbol{\cos\theta} = \frac{dot(c-a,b-a)}{|\boldsymbol{b-a}|}$$

点と線分の距離

- 点cから直線abに垂直に下ろした点pを求める

線分abに対する線分apの長さを計算できるので、

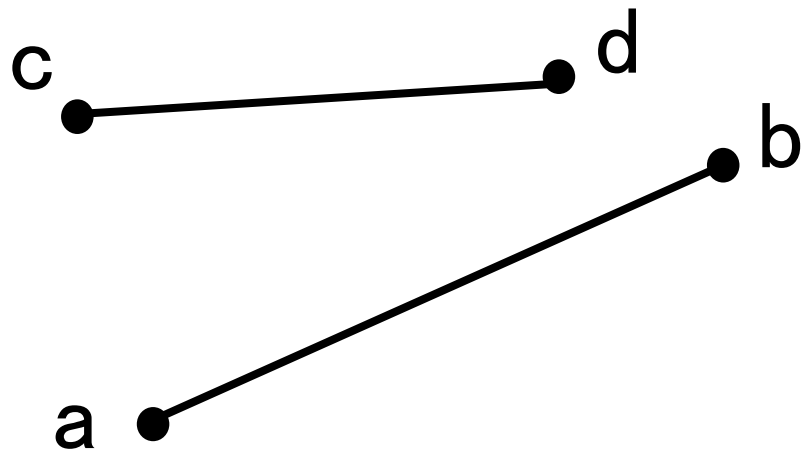
$$p = \mathrm{a} + (\mathrm{b} - a) * \frac{dot(c-a, b-a)}{|b-a|^2}$$



点pが線分ab上の点なら点cは
線分abに垂直に下ろせる点があることがわかる

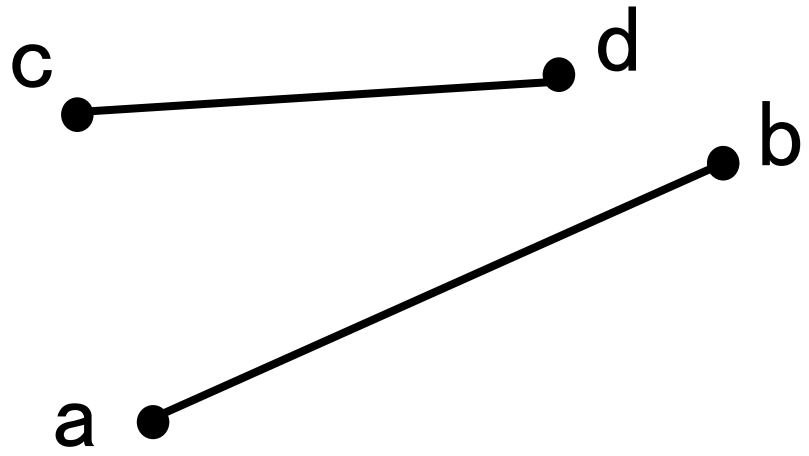
線分と線分の距離

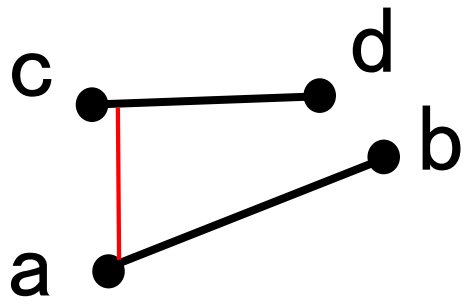
- 線分abから線分cdまでの距離を求める

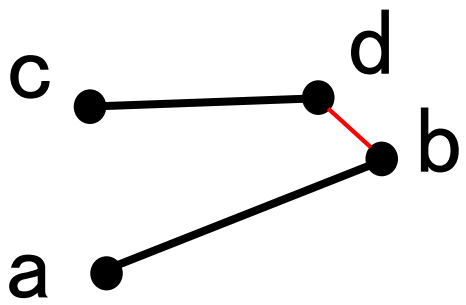


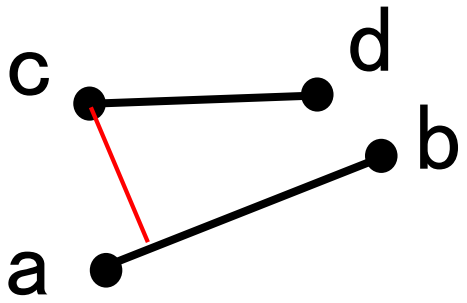
線分と線分の距離

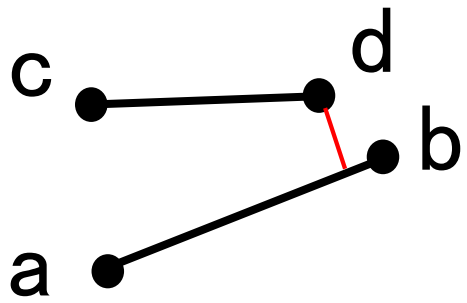
- 基本的な考え
  - 線分同士が交差しているなら距離は0
  - 公差していないなら線分の端点と線分の距離を全4通り求めて最も小さいのが最短距離





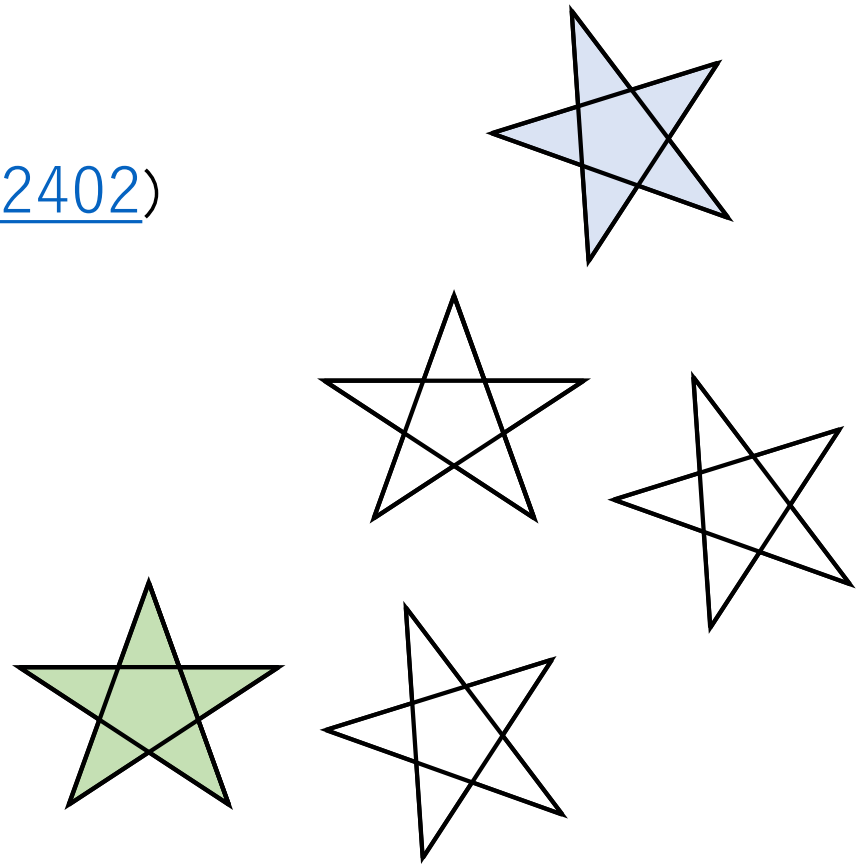






AOJ 2402 : 天の川

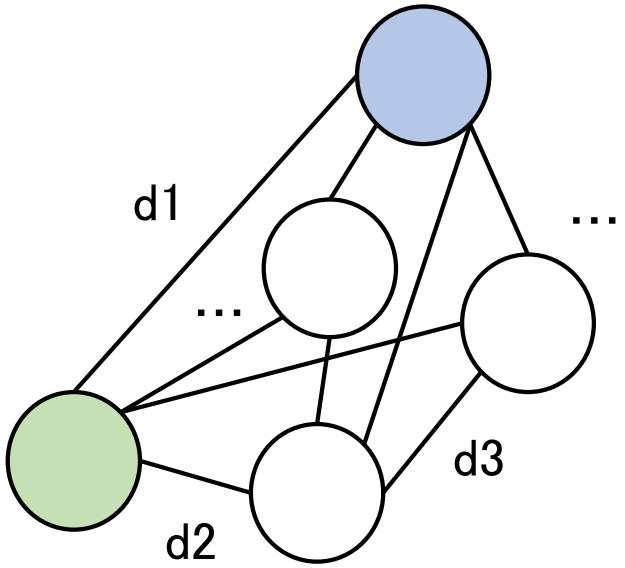
URL (https://onlinejudge.u-aizu.ac.jp/problems/2402)

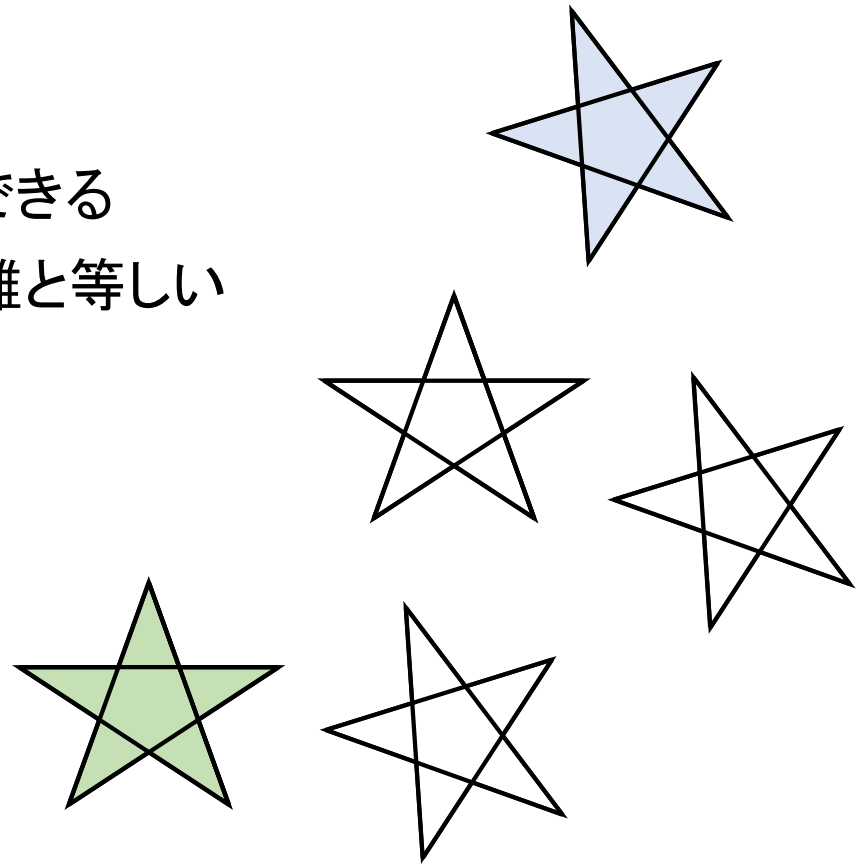


- 5つの線分を持つn個の星が与えられる
- M番目の星からL番目の星へ移動するのに
  必要な移動距離の総和の最小値を出力せよ
- ただし，線分上は距離としてカウントされない
- 1 <= n <= 100

AOJ 2402 : 天の川

- 星から星への移動距離を求めるとグラフ問題へと帰着できる
- 今回の星と星の距離はそれぞれの線分同士の最短距離と等しい
- 辺に重みをつけて最短経路を見つける

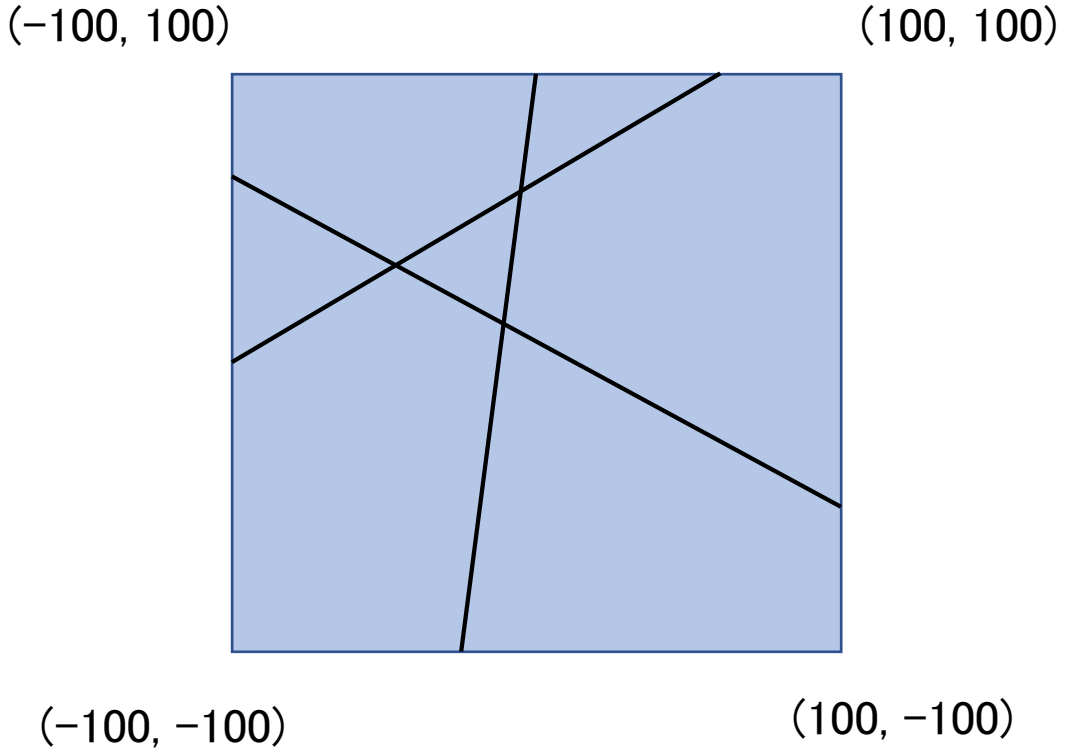




AOJ 2009 : Area Separation

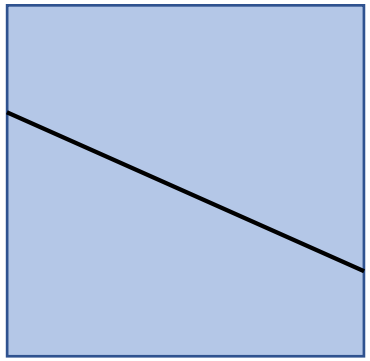
URL (https://onlinejudge.u-aizu.ac.jp/problems/2009)

- 正方形領域が与えられる
- 与えられるn本の直線によって領域が
  いくつに分割されるか出力せよ
- 1 <= n <= 100

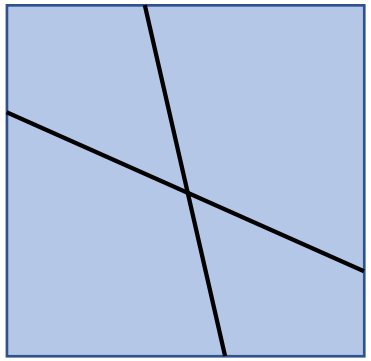


AOJ 2009 : Area Separation

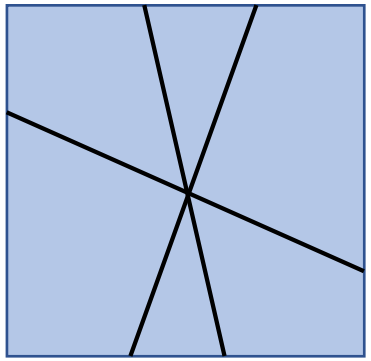
- 線分の公差数などを確かめると規則性が見える
- 交点の重複にも気をつける必要がある



領域数：+1



領域数：+2



領域数：+2

補足

- 他にも多角形，凸包，円などに関する典型アルゴリズムがありますが，
  HCPCの過去スライドで既に紹介されていたのでそちらをご覧ください．

本日の流れ

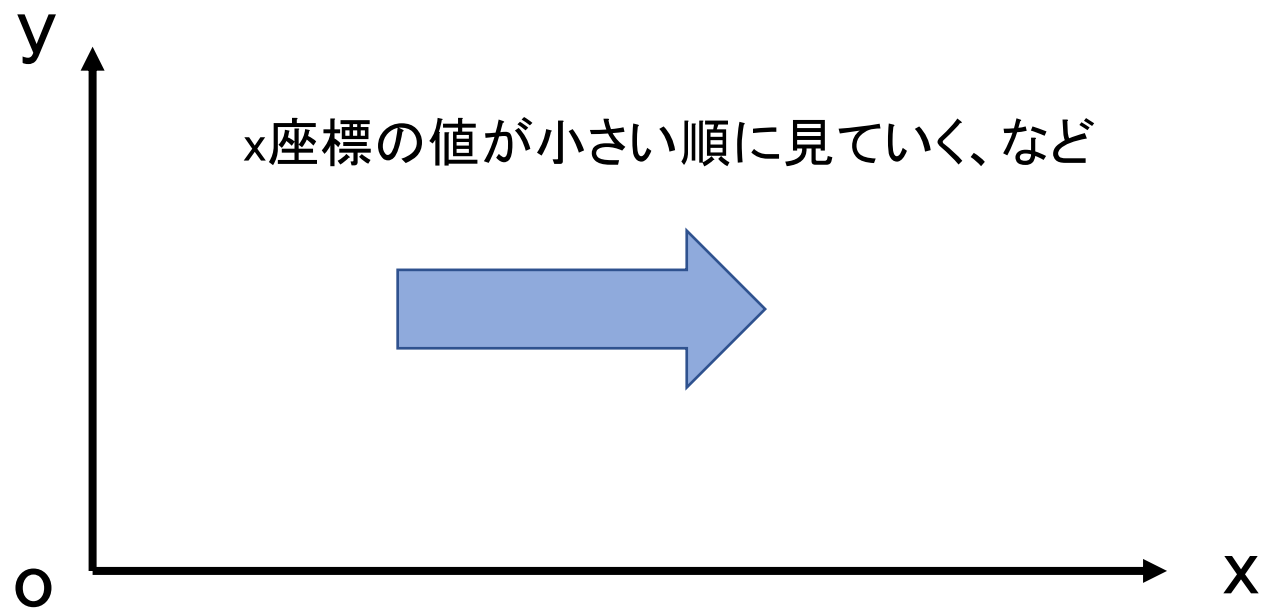
1. 事前知識

2. 典型アルゴリズム（点，ベクトル，線分）

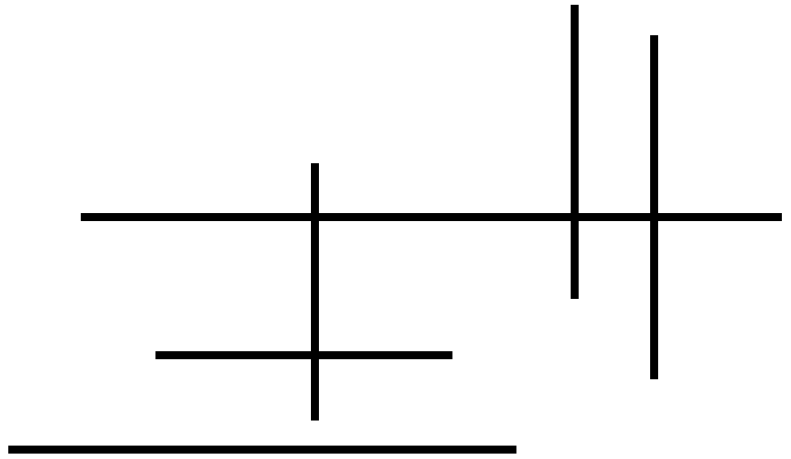
3. 平面走査法

平面走査法とは？

- 図形をある一定方向から走査し、計算量を落とすテクニック



線分交差列挙



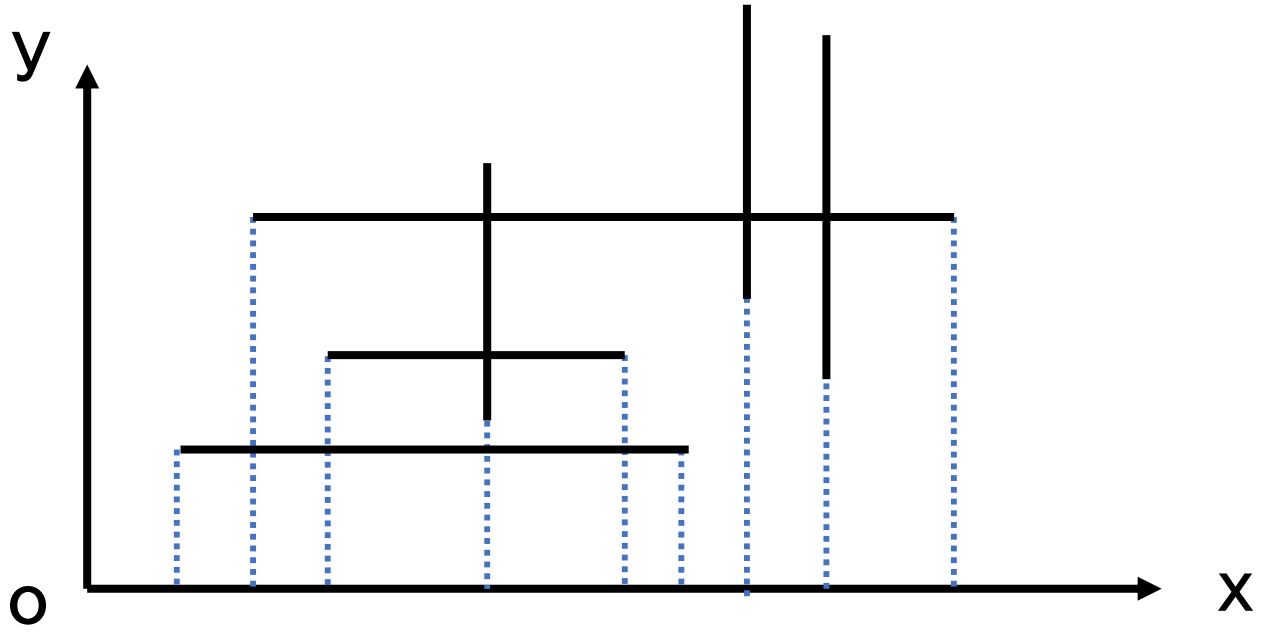
- **X軸またはy軸に平行**な線分がn本与えらえる
- それらの交点の数を出力せよ
- 1 <= n <= 100,000
- 互いに平行な2つの線分が重なることはない
- ただし，交点の数kは k <= 100,000 とする

→ 愚直に計算するとO(n^2)

→ 交差する可能性の高い線分を絞り込む

→ 平面走査を行う

線分交差列挙



- 線分の端点のx座標が小さい順に走査する
- 線分のどの端点を見ているかで操作内容を変更する

線分交差列挙

① 線分が水平で左端点のとき

→ 線分を登録する

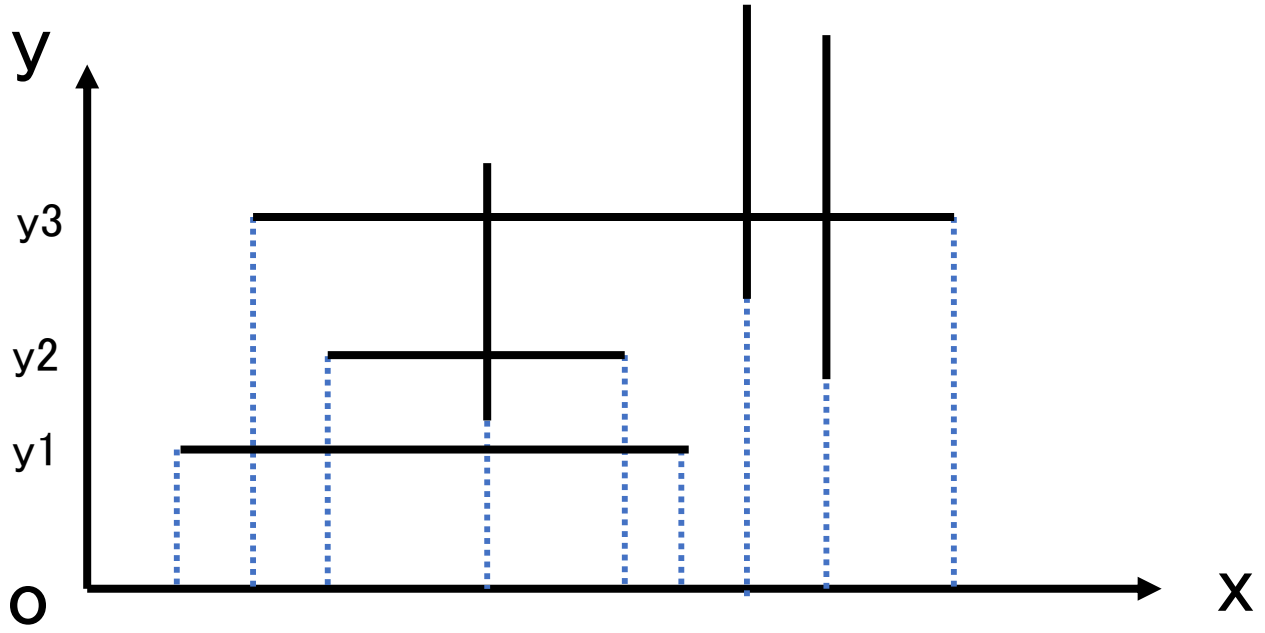
② 線分が水平で右端点のとき

→ 線分を削除する

③ 線分が垂直のとき

→ 他の線分と交差して
いるか確認する

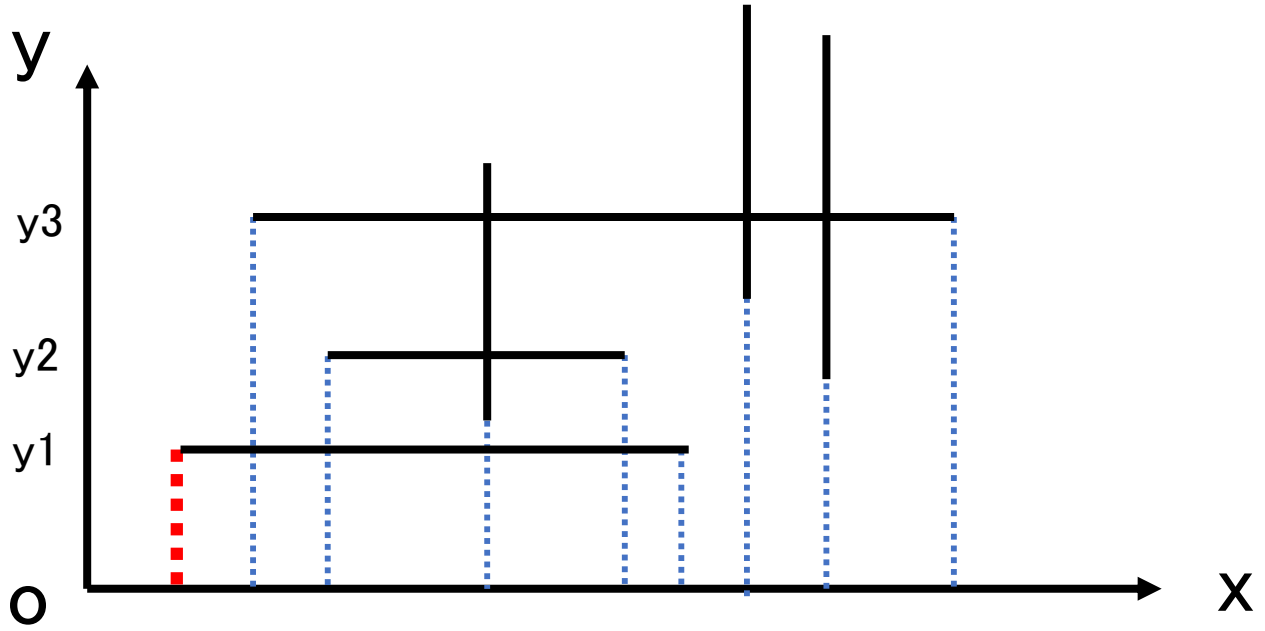
線分交差列挙



- 線分を登録するときは、端点のy座標を平衡二分探索木(C++ならsetなど)に渡す

→ 登録、削除を O(log n) で行うことができ、二分探索も行える

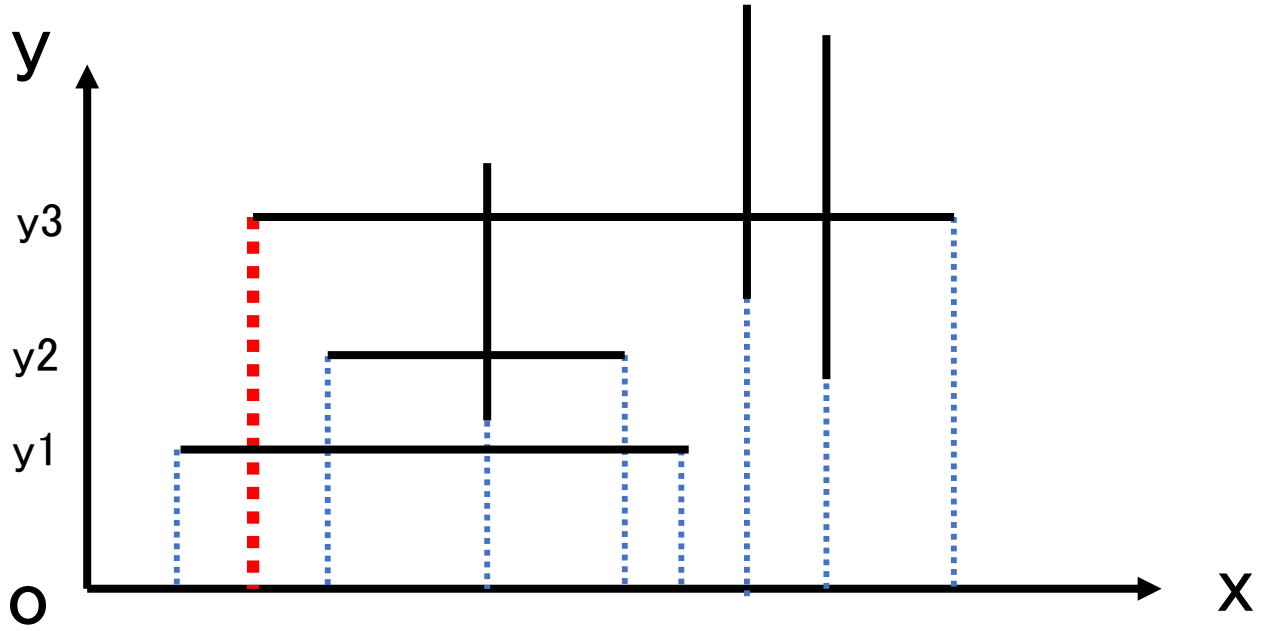
線分交差列挙



- 線分1のy座標を登録

  Set : [y1]

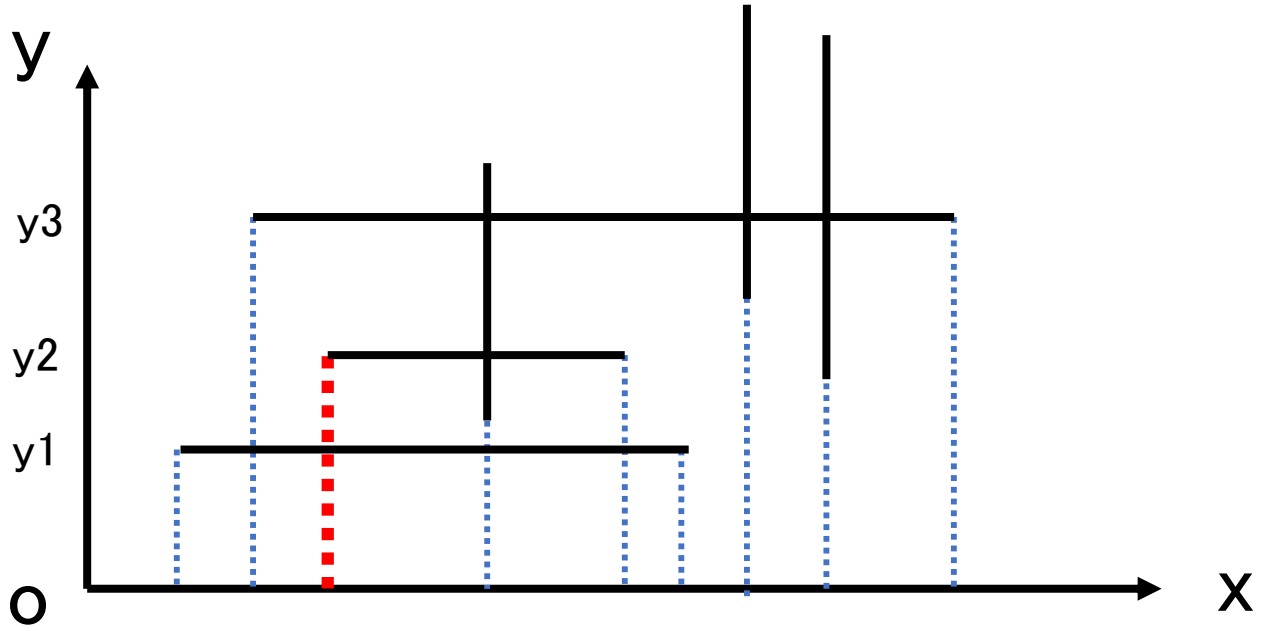
線分交差列挙



- 線分3のy座標を登録

  Set : [y1, y3]

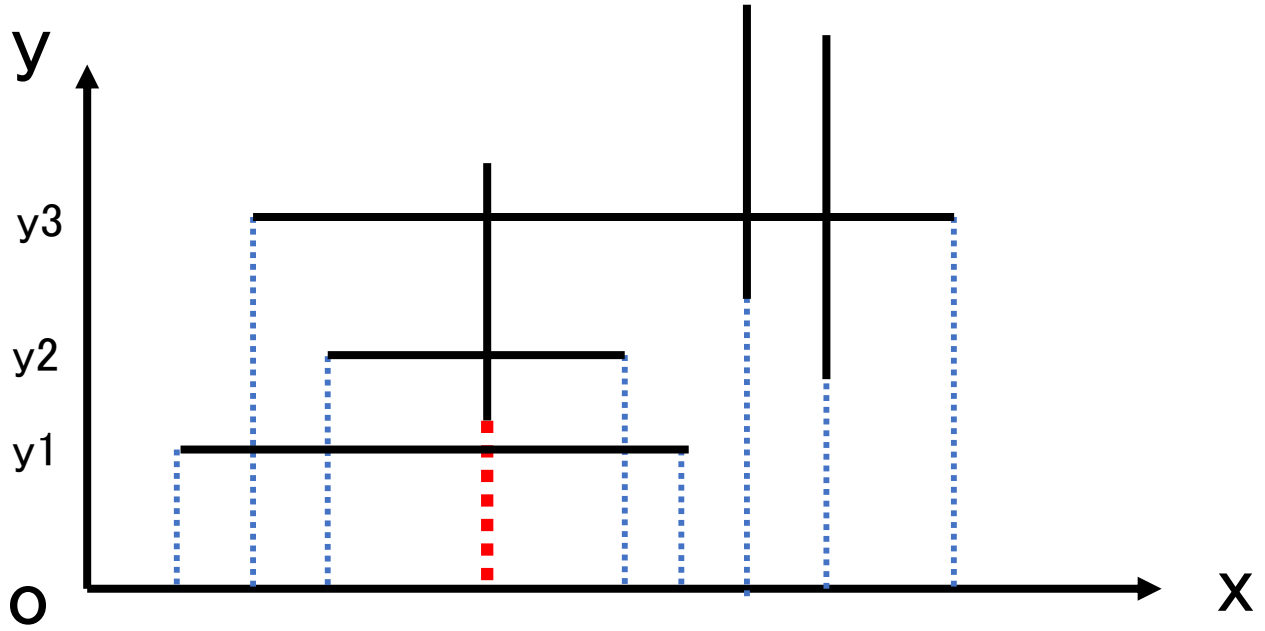
線分交差列挙



- 線分2のy座標を登録（二分探索木で管理しているので、y座標の小さい順に管理できる）

  Set : [y1, y2, y3]

線分交差列挙



- 線分4の端点のy座標範囲に該当する要素数を二分探索で取得

  Set : [y1, y2, y3]

(以降省略)

線分交差列挙

- 垂直線分イベント時の計算量について

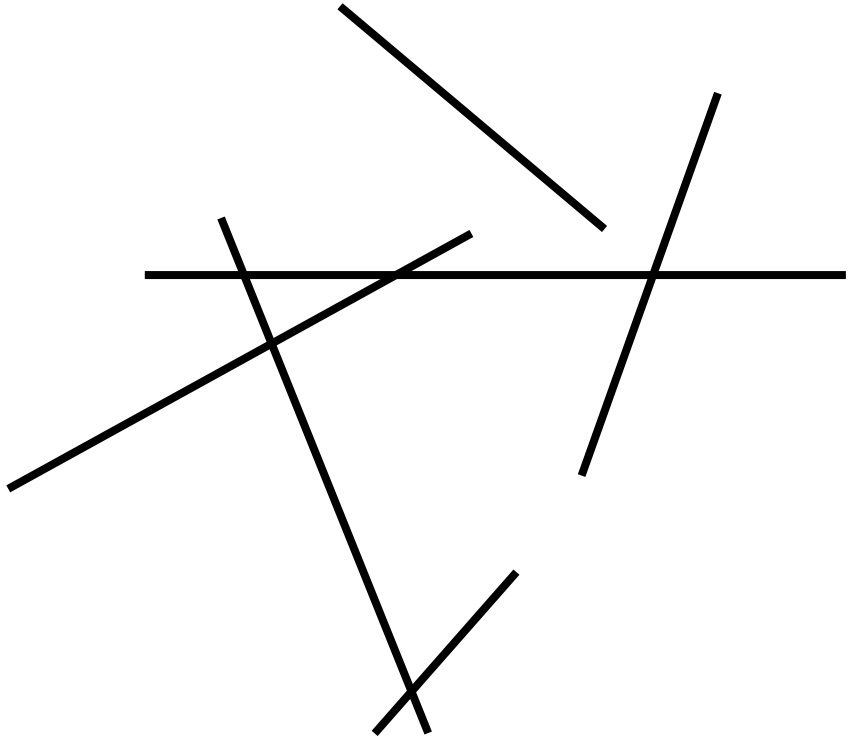
C++標準の平衡二分探索木を用いた場合は，
二分探索　と　該当範囲(交点数分)の探索　の計算時間がかかる
　　→ 最終的な計算量は O((交点数) + n log n) となる
　　（平衡二分探索木の実装によっては O(n log n) で済むらしい）

また，先程までの説明とは異なり，線分情報をセグメントツリーで管理した場合は
交点数を O(log n) で取得でき，最終的な計算量は O(n log n) となる　（端点のy座標の圧縮が必要）

何にせよ，平面走査を行うことで計算量を落とすことができた

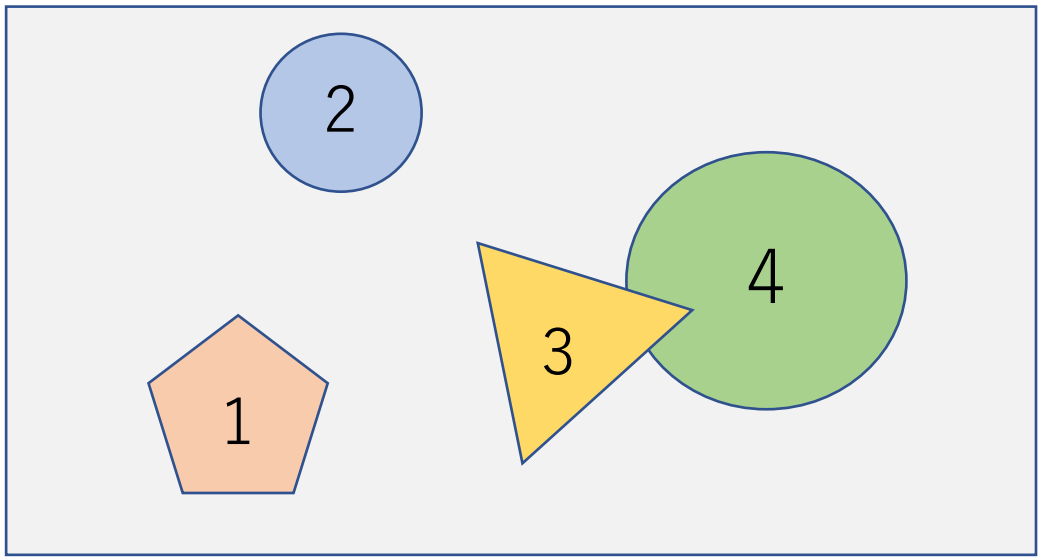
挑戦：線分交差列挙

- 線分がn本与えらえる
- それらの交点の数を出力せよ
- 1 <= n <= 100,000
- ただし，交点の数kは k <= 100,000 とする



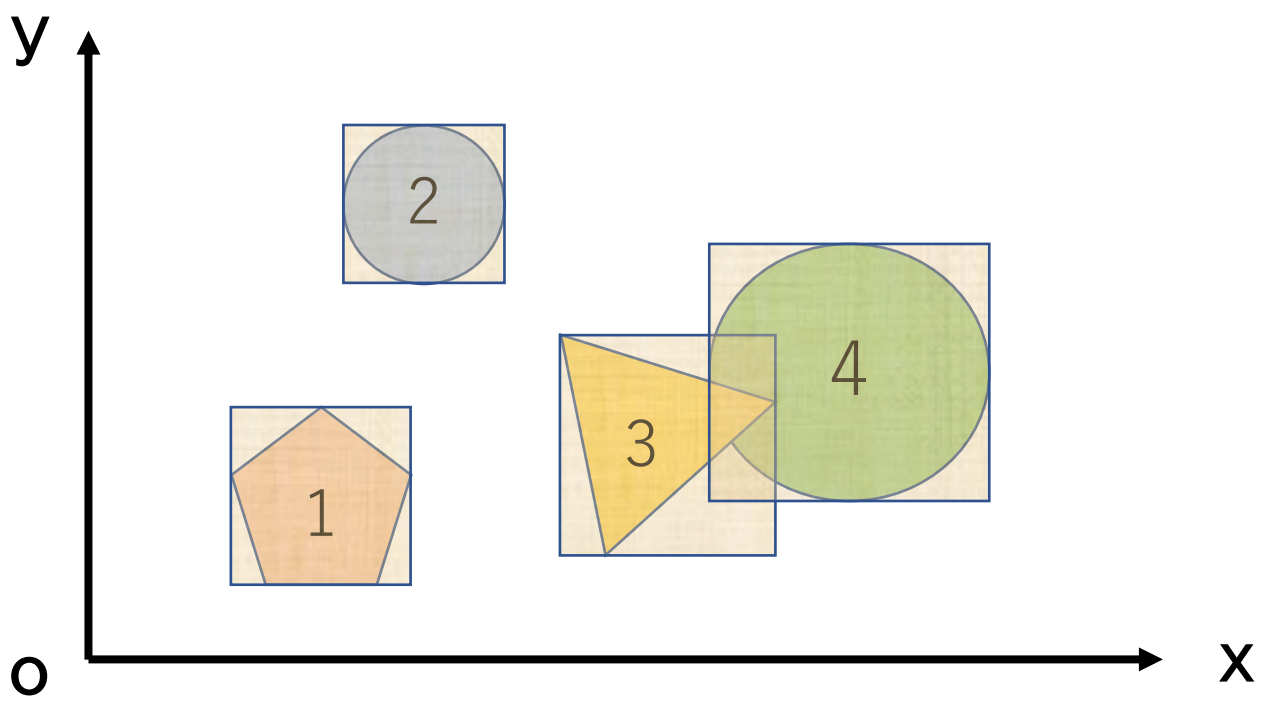
おまけ：Sweep and Prune

- オブジェクトの衝突判定の高速化に用いられるアルゴリズムのひとつ



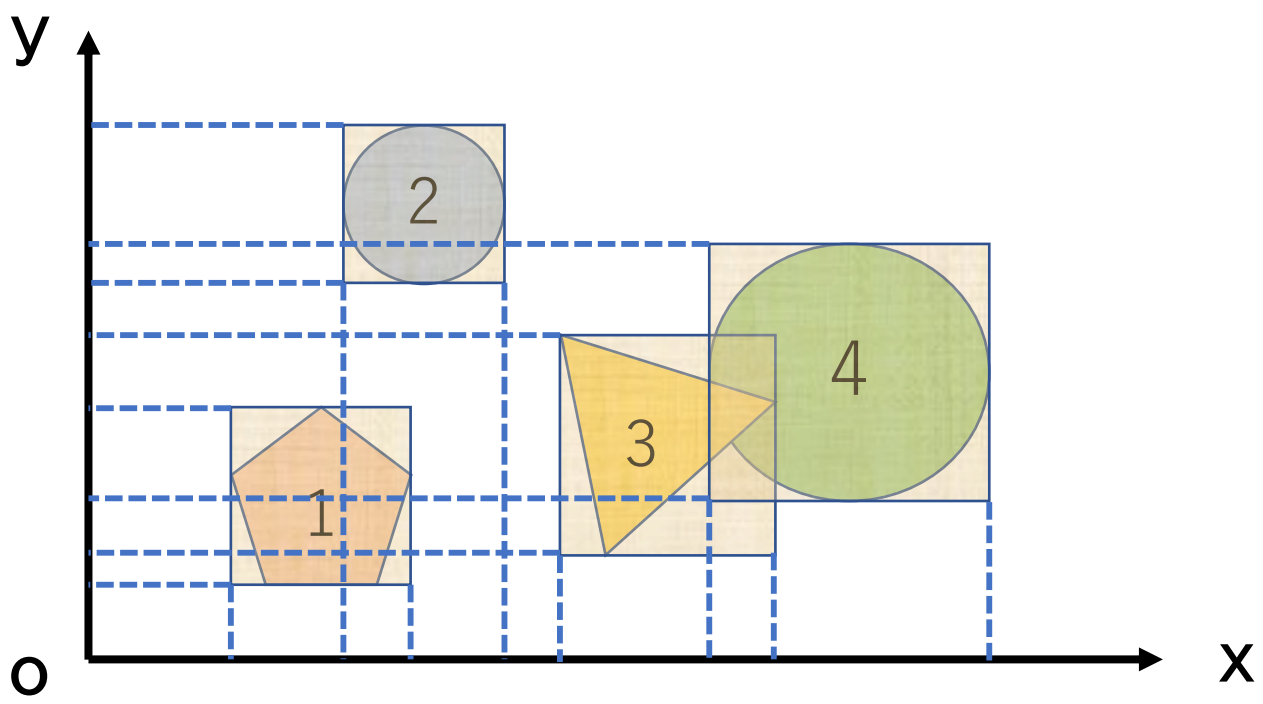
- オブジェクト同士が衝突してるかの判定を愚直に総当たりすると O(n^2) かかる
- 平面走査で衝突する可能性のあるオブジェクトのペアを探し出す

おまけ：Sweep and Prune



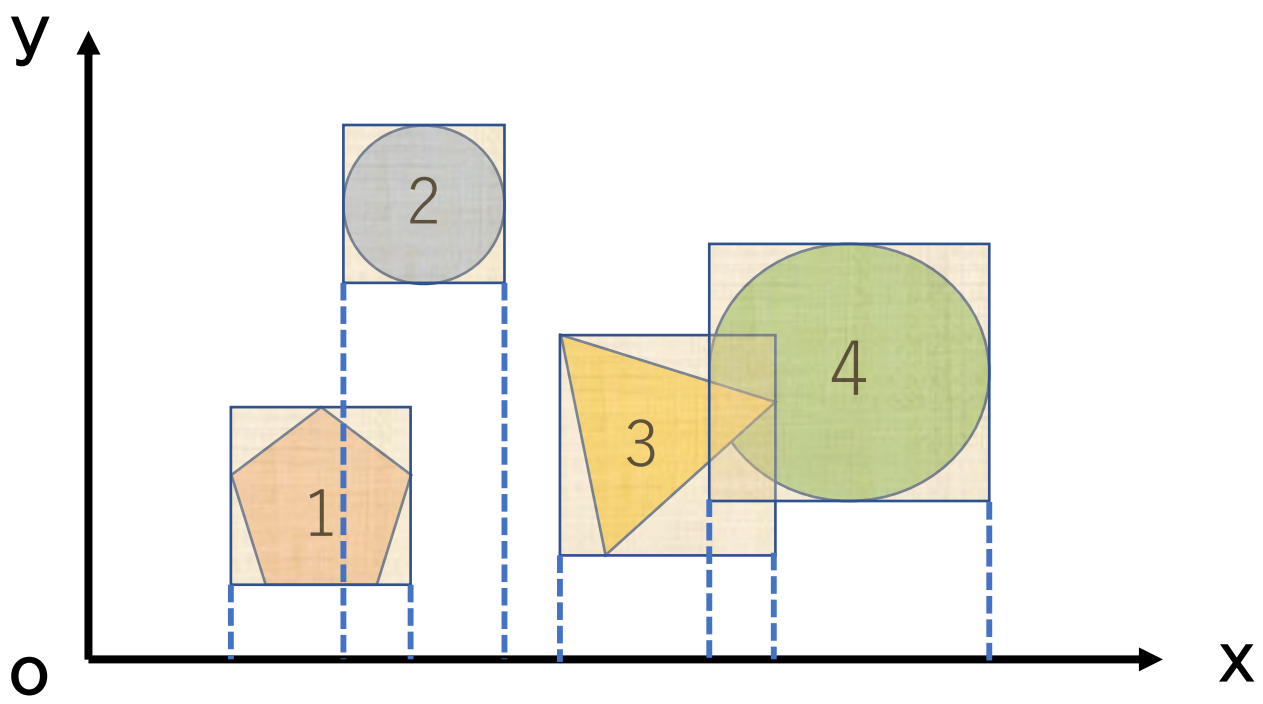
- 第一手順
  - オブジェクトのAABB(Axis-Aligned Bounding Box)を求める
  - つまり，オブジェクトを囲む，軸(今回はx軸，y軸)に平行な矩形を求める

おまけ：Sweep and Prune



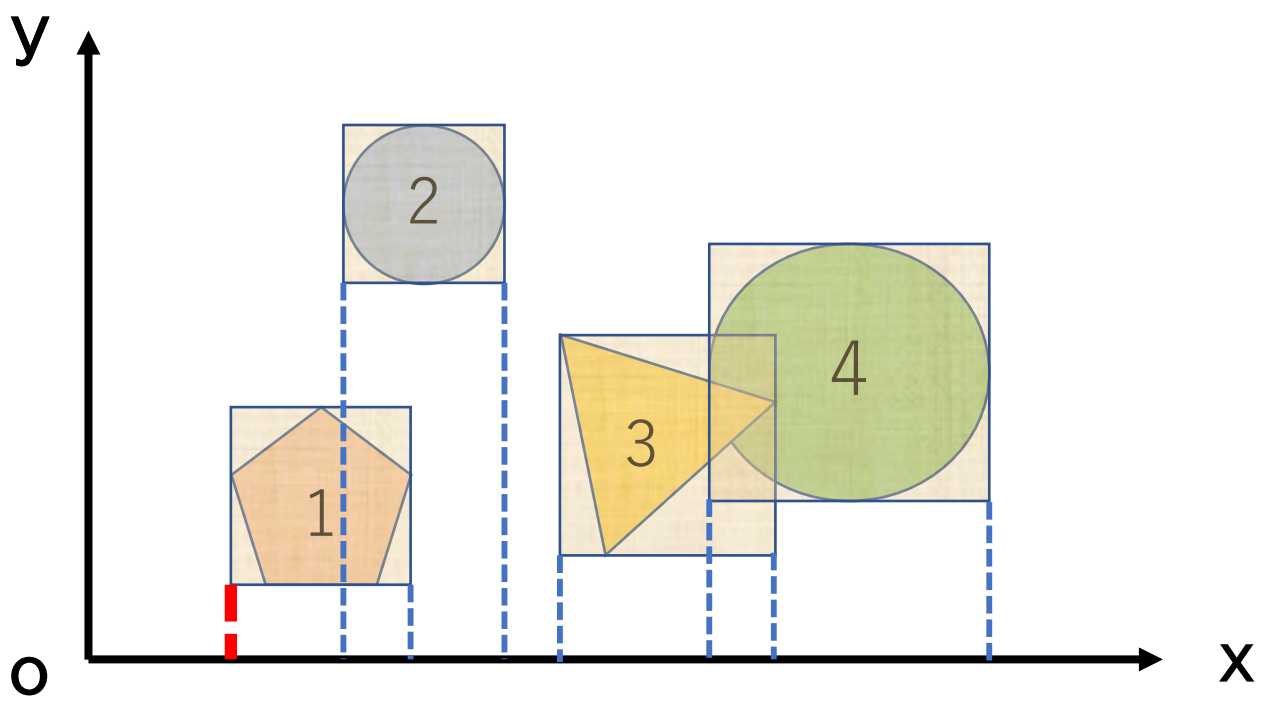
- 第二手順
  - AABBの端点を軸へと投影する

おまけ：Sweep and Prune



- 第三手順
  - x軸に投影された点を小さい順に走査する
  - AABBの開始地点ならリストにオブジェクトを登録し、
    終了地点ならリストからオブジェクトを削除する

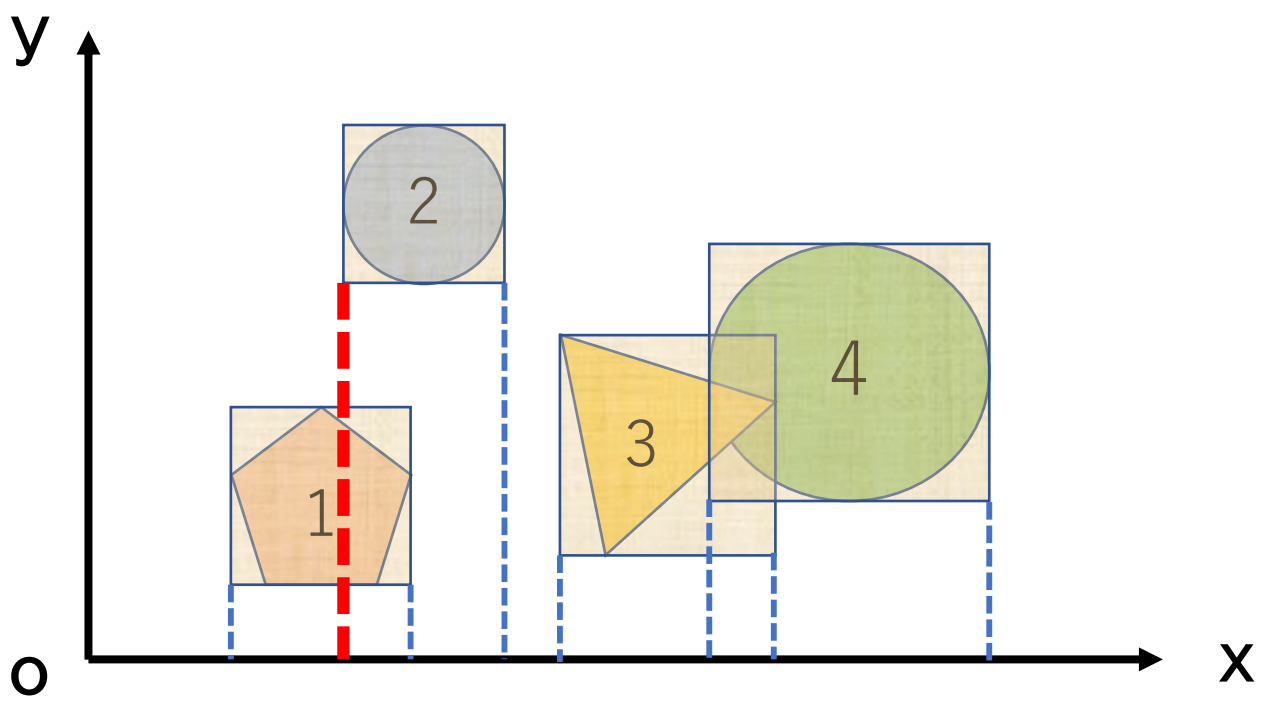
おまけ：Sweep and Prune



- リストにオブジェクト1を登録する

  List : [1]

おまけ：Sweep and Prune

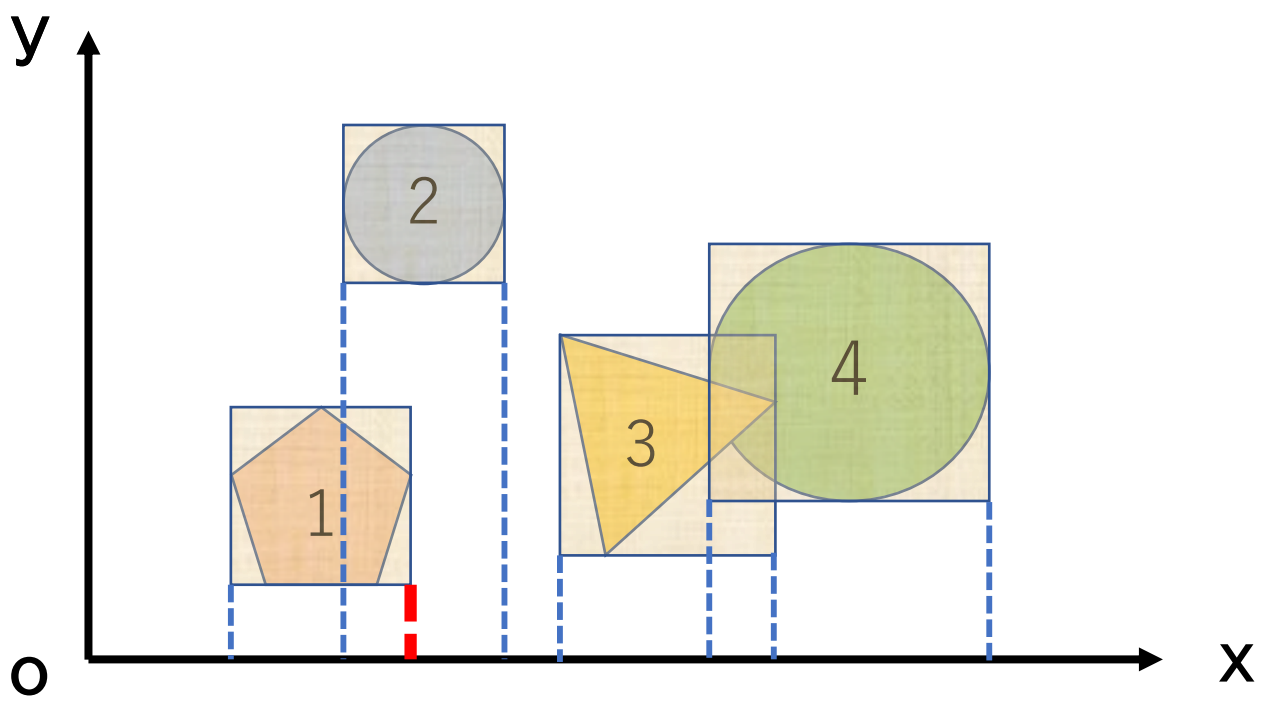


- リストにオブジェクト2を登録する

  List : [1, 2]

- オブジェクト2はオブジェクト1と衝突可能性ありとして登録する

おまけ：Sweep and Prune

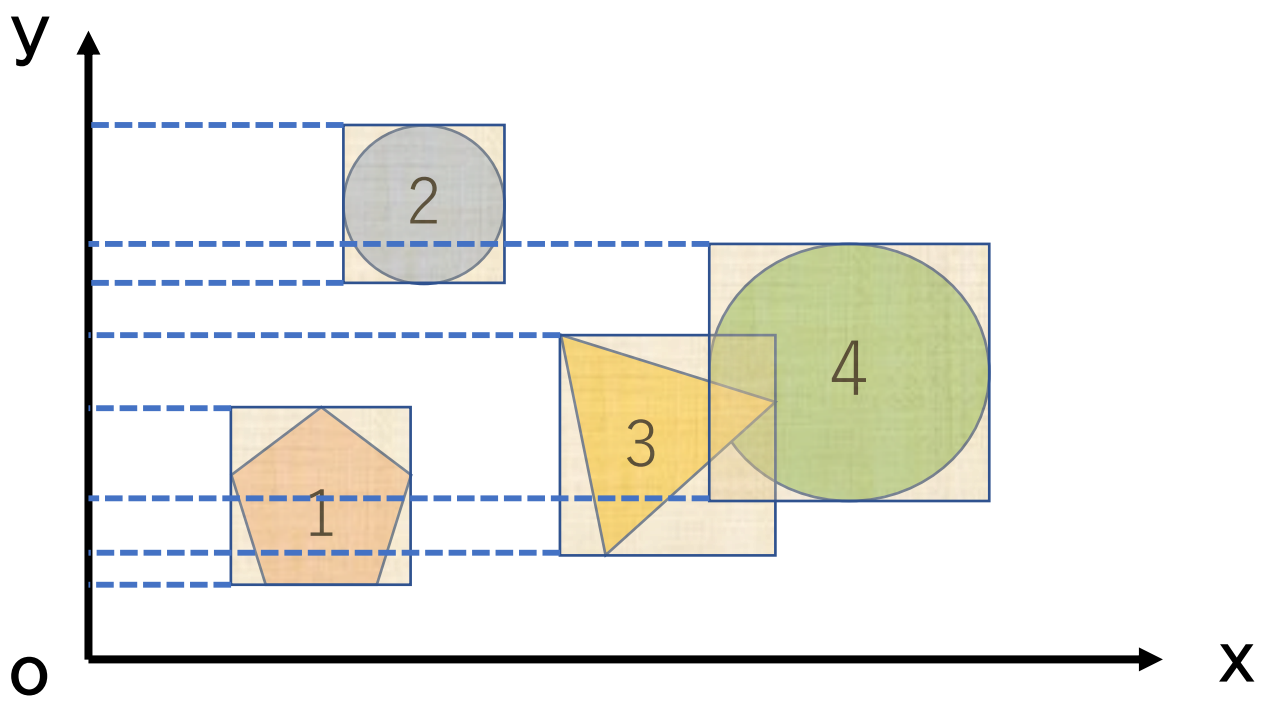


- リストからオブジェクト1を削除する

  List : [2]

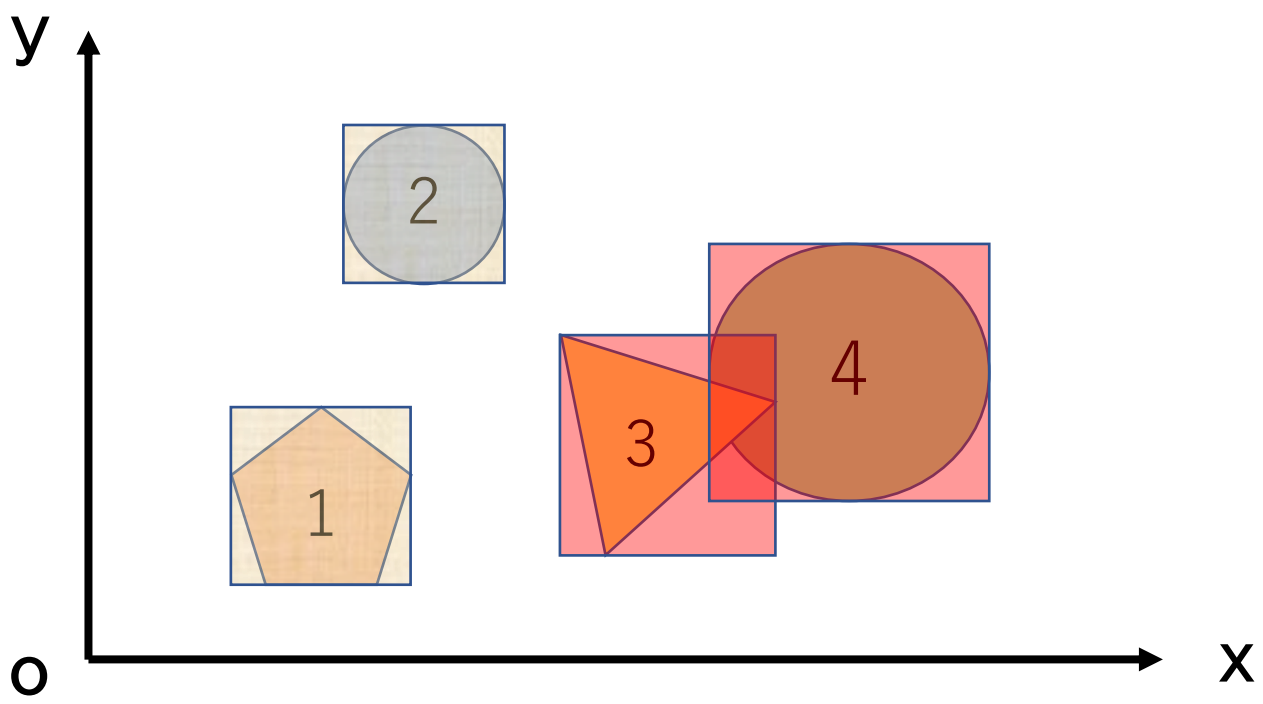
（以降省略）

おまけ：Sweep and Prune



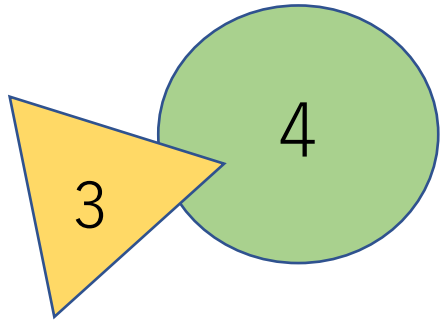
- 第四手順
  - y軸に投影された点も同様に小さい順に走査する

おまけ：Sweep and Prune



x軸方向とy軸方向の走査の両方で衝突可能性ありと登録された
オブジェクトのペアは衝突している可能性が高い

おまけ：Sweep and Prune



- 実際に衝突しているか計算する（三角形と円の衝突判定）
- AABB等を用いて大まかに衝突比較する段階のことをブロードフェーズと呼ぶ

おまけ：Sweep and Prune

- 走査する前段階のソートについて補足
  - ゲーム内で1フレーム経った後でもオブジェクトの並び順は変わりにくい
  - そのため，平均計算量が小さいO(n log n)のソートアルゴリズムよりも
    平均計算量が O(n^2) の**挿入ソート**を用いたほうが高速になる傾向がある

- 衝突判定の高速化アルゴリズムとして，他に空間分割があります
  （4分木空間分割，BSP，kd Tree など）

おしまい

- ICPCで幾何問題を担当する方はライブラリ設計頑張りましょう！
- ゲームプログラミングも楽しいですよ！計算幾何の知識を役立てよう！