休講とレポート課題のお知らせ

- 来週6/1(水)は休講にします
- 代わりにレポート課題を出します
- 近日中にポータルサイトに掲示します

生産情報システム工学

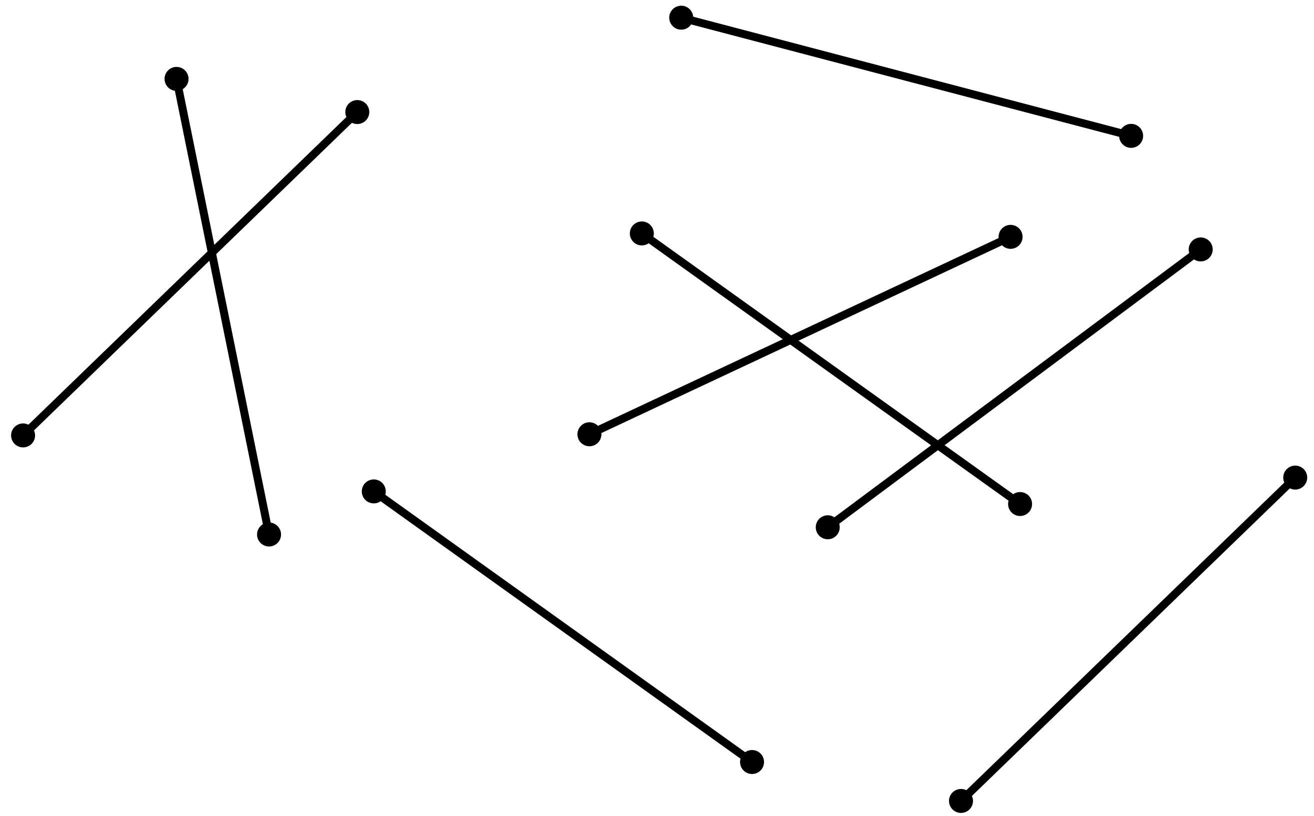
#06 交差(2)

2015/05/27(水)

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

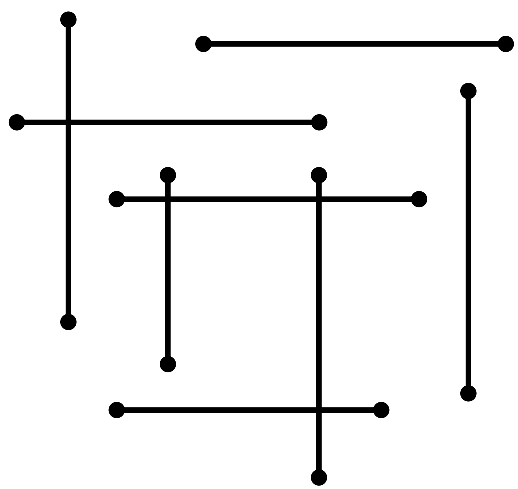
交差

- 問題：線分の交差はいくつあるか？

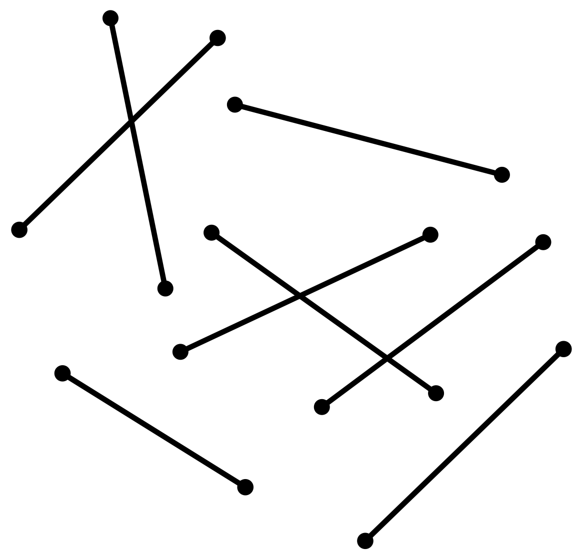


2.2 n本の線分の交差

水平・垂直な線分



一般の線分



今日の内容

- ヒープ (復習)
- 2分探索木 (復習+α)
- 2線分の交差判定 (復習)
- n本の一般の線分の交差判定 (メイン)

1.4.3 ヒープ

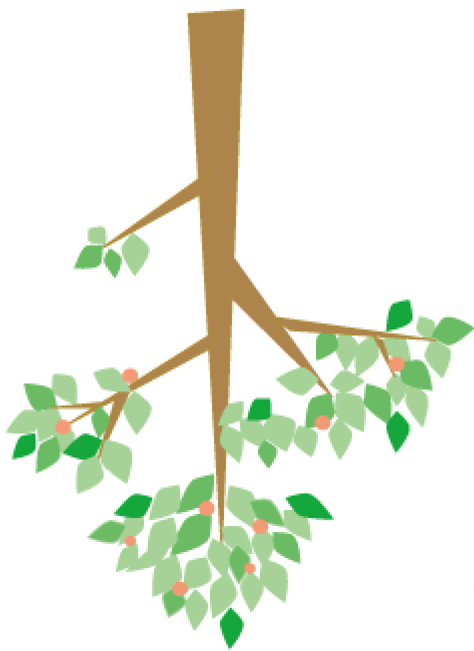
- ■ スタック・キュー
  - 取り出す順番：挿入された順番で決まる
    - スタック：最後に入ったもの
    - キュー：最初に入ったもの

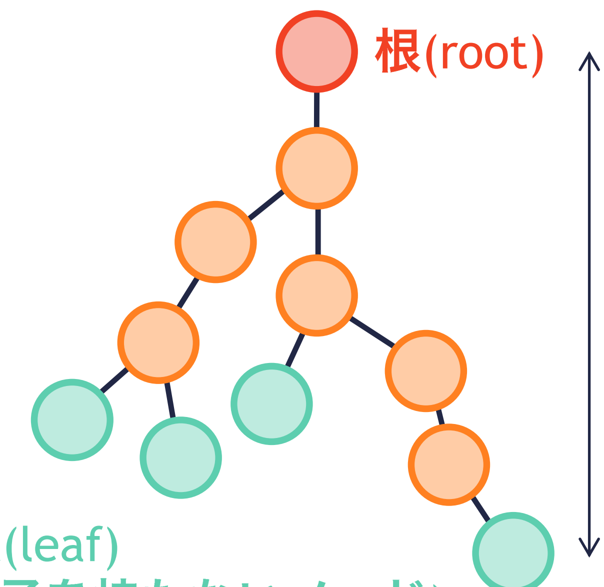
- ■ ヒープ(順位付きキュー，Priority Queue)
  - 取り出す順番：挿入された順番と無関係
  - 最大，または最小のものを取り出す

1.4.3 ヒープ

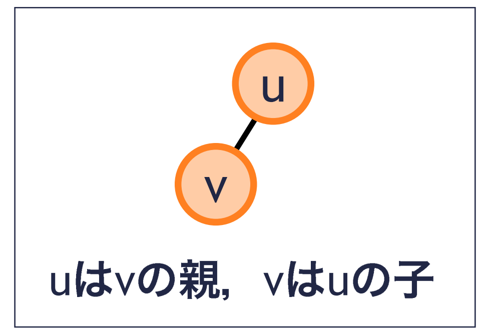
■ 木(Tree)

- いくつかの**ノード(頂点，節点)**とそれらを結ぶ**エッジ(枝，辺)**から構成される





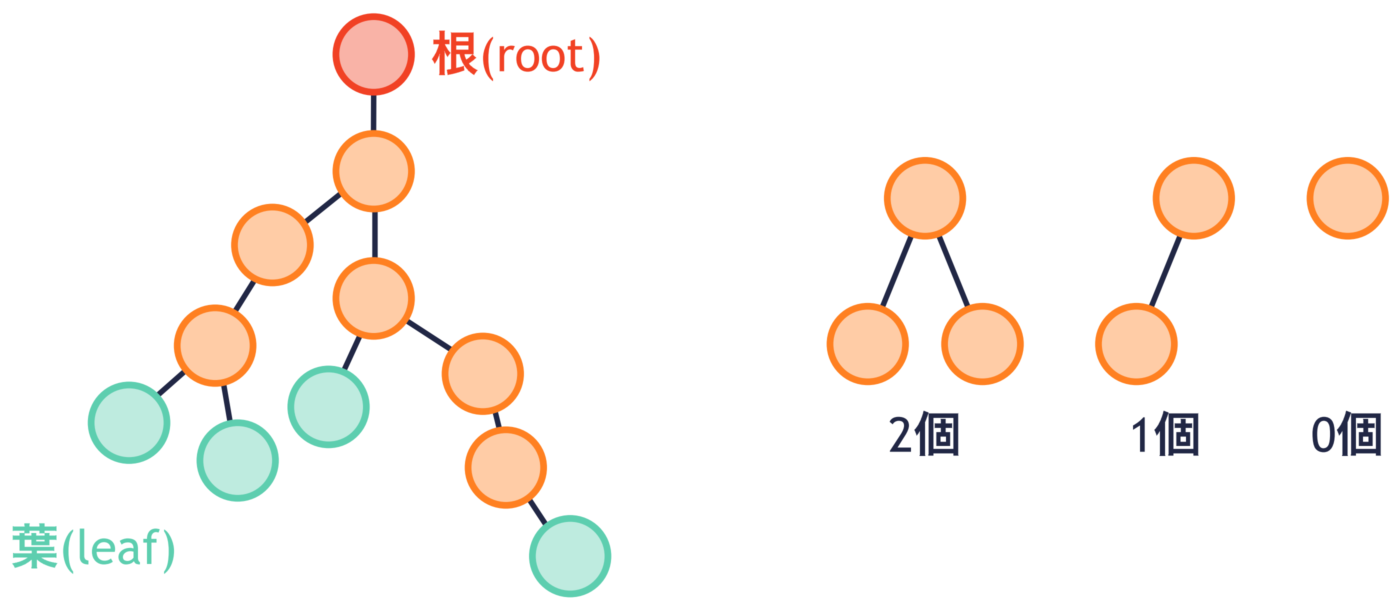
葉(leaf)
(=子を持たないノード)



高さ(=枝の数)
根から葉への経路の中で，
最も長いものの長さ
(この例の場合5)

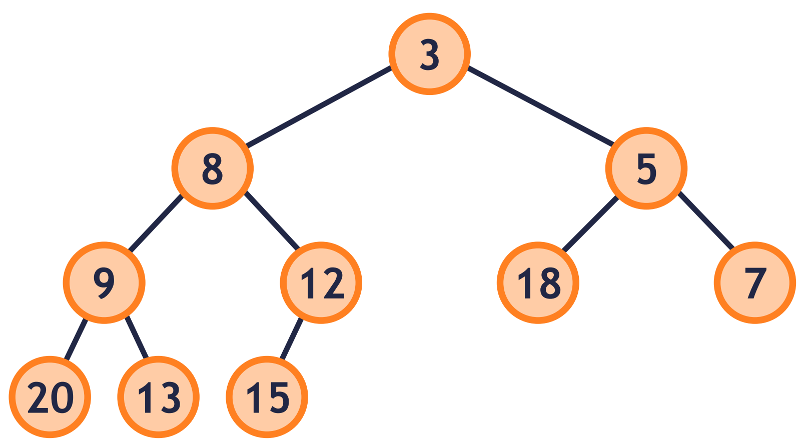
1.4.3 ヒープ

- 2分木(Binary Tree)
  - どのノードも2個以下の子を持つ木(3個以上はだめ)



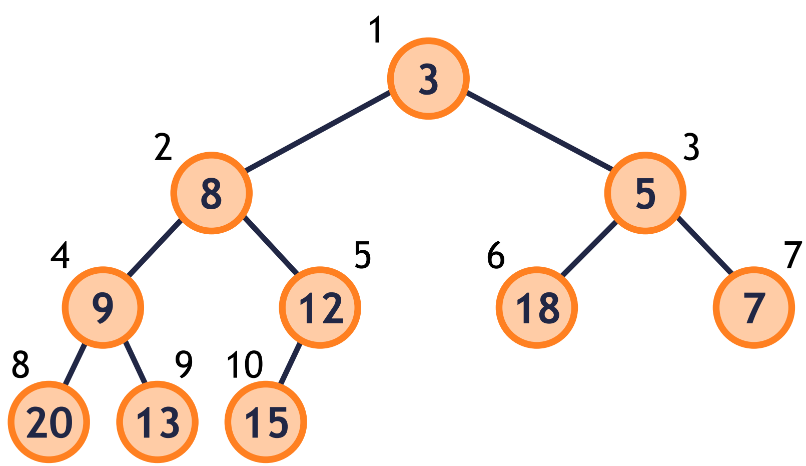
1.4.3 ヒープ

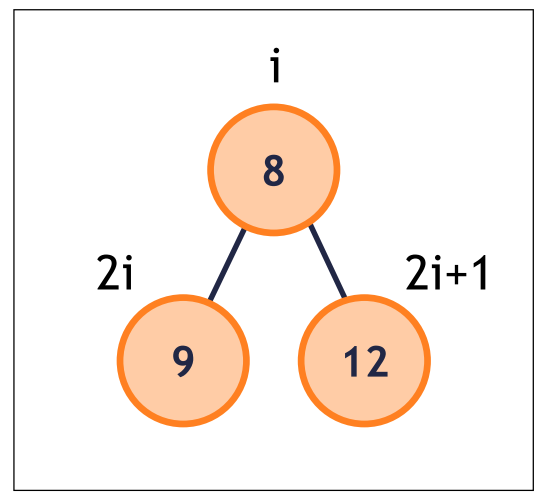
- ヒープは2分木の一種
  - 各ノードの要素がその全子孫より小さいか等しい
  - 最小の要素は常に木の根に蓄えられる



1.4.3 ヒープ

- ヒープは2分木の一種
  - 一般に，頂点vに割り当てられた要素が配列のi番目ならば，左の子は2i番目，右の子は2i+1番目に入る





	1	2	3	4	5	6	7	8	9	10			
heap	3	8	5	9	12	18	7	20	13	15			

1.4.3 ヒープ

- 主な基本操作
  - 挿入(データの追加)
  - 削除(データの取り出し)

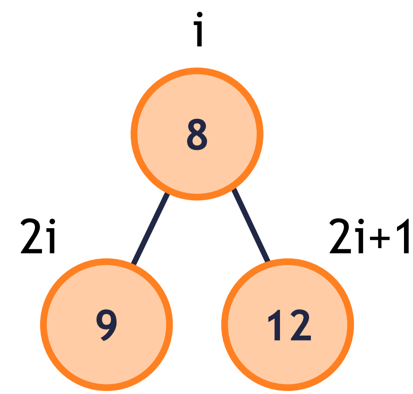
  - スタックの場合，プッシュとポップ
  - キューの場合，エンキューとデキュー

1.4.3 ヒープ

■ ヒープを実現する構造体

```
#define hmax 100

struct heap {
    int box[hmax+1];   //  データ
    int size;          //  データの個数
};
```



	0	1	2	3	4	5	6	7	8	9	10	11		
heap	×	12	15	24	18	30	61	43	55	82	32	33		

↑ 配列の添字0の要素は使わない

1.4.3 ヒープ

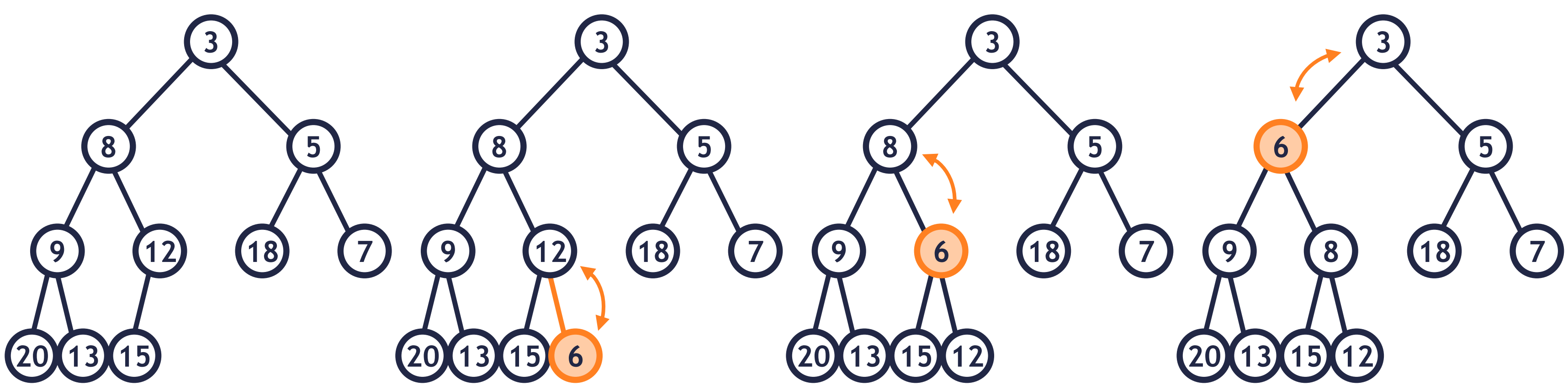
1	2	3	4	5	6	7	8	9	10			
3	8	5	9	12	18	7	20	13	15			

1	2	3	4	5	6	7	8	9	10	11		
3	8	5	9	12	18	7	20	13	15	6		

■ 挿入：insert(6)

1. 新たに挿入する要素を配列の末尾に入れる
2. 挿入要素とその親(12)で，ヒープ条件が成立するか調べる
3. もし成立しなければ，親と子を入れ替える
4. 終了条件：①ヒープ条件が成立，②挿入要素が根になる

繰返し



初期状態

6を入れる
12と6で条件を
調べる

12と6を入れ替える
8と6で条件を
調べる

8と6を入れ替える
3と6で条件が成立
終了

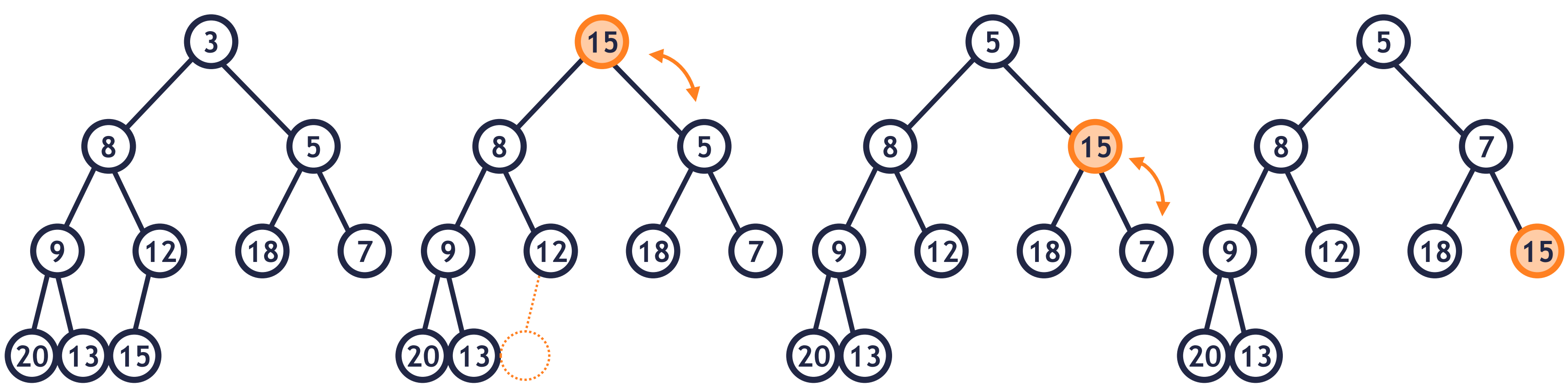
1.4.3 ヒープ

1	2	3	4	5	6	7	8	9	10			
3	8	5	9	12	18	7	20	13	15			

1	2	3	4	5	6	7	8	9				
15	8	5	9	12	18	7	20	13				

- 削除：deletemin()
  1. 末尾の要素を先頭に書き込む(最小要素の削除)
  2. 書き込んだ要素とその子でヒープ条件が成立するか調べる
  3. もし成立しなければ，左右の子の小さい方と入れ替える
  4. 終了条件：①ヒープ条件成立，②書き込んだ要素が葉になる

繰返し



初期状態

15を根に入れる
15と5で条件を
調べる

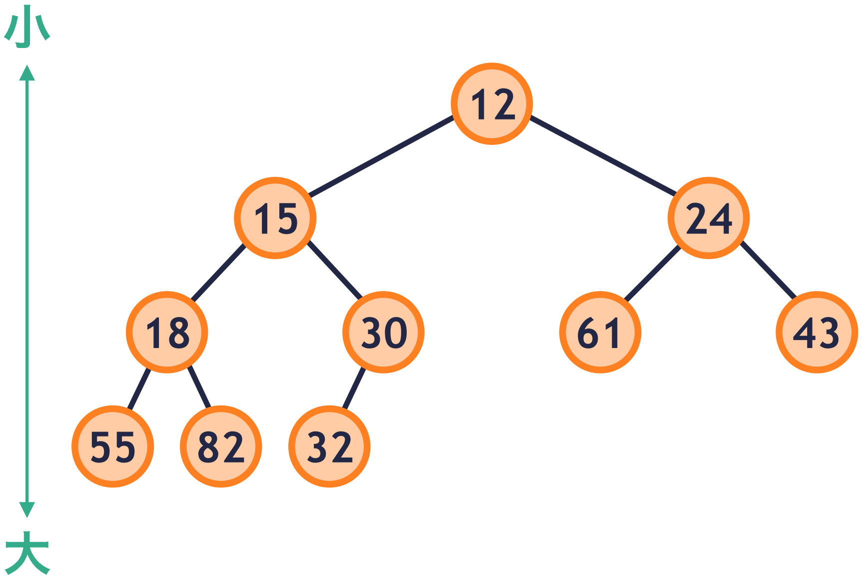
15と5を入れ替える
15と7で条件を
調べる

15と7を入れ替える
15が葉なので条件が
成立，終了

1.4.4 2分探索木

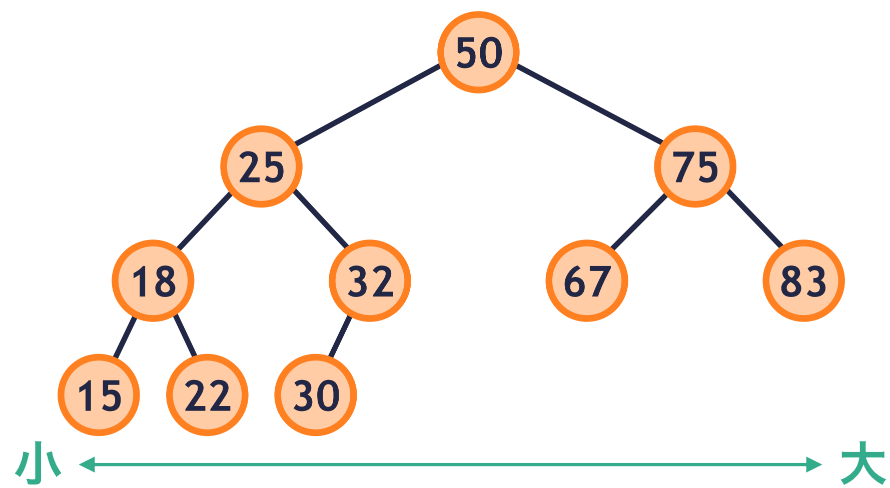
■ ヒープと2分探索木

ヒープ	2分探索木



各頂点の要素はすべての子孫の要素よりも小さいか等しい

<u>**最小要素を取り出す**</u>



各頂点の要素は左部分木のすべての要素より大きい

<u>**指定された要素を取り出す**</u>

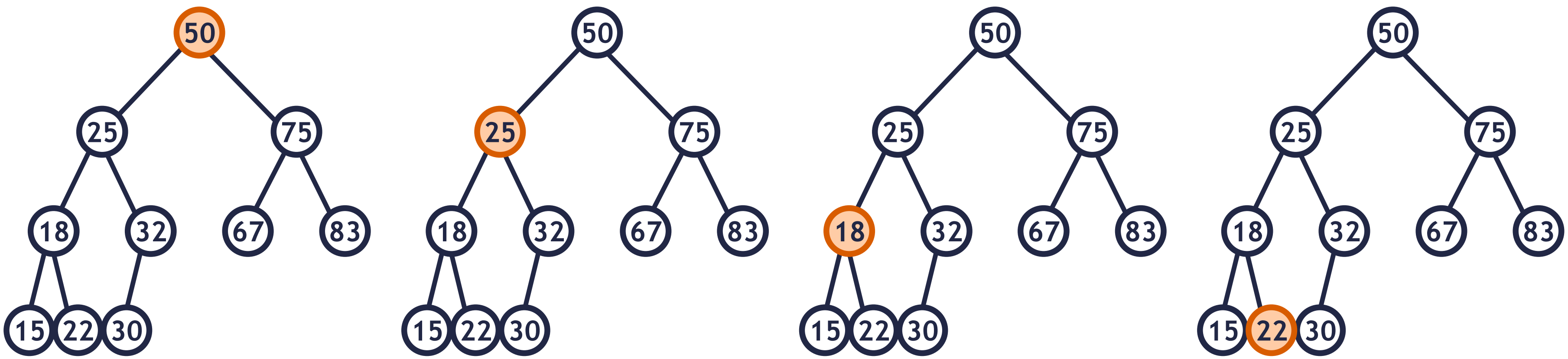
1.4.4 2分探索木

- 復習：2分探索(データ構造入門で学習済み)
  - あらかじめデータを整列させておき，配列の中央要素との比較と探索範囲を縮小を繰り返し行う
  - 探索は高速に行える($O(\log N)$)
  - データの挿入・削除に時間がかかる($O(N)$)

- 2分探索木
  - データの挿入・削除も高速に行える($O(\log N)$)
  - アルゴリズム理論における基本的なデータ構造

1.4.4 2分探索木

■ 探索の例1 (木に含まれるkey=22を探索する場合)



根からスタートする
key<50，左部分木へ

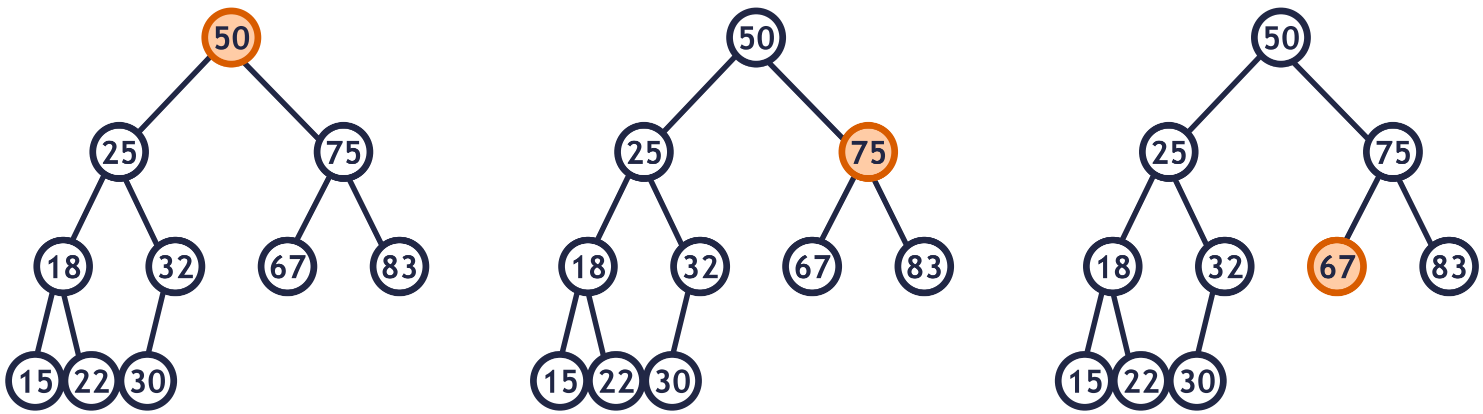
key<25，左部分木へ

18<key，右部分木へ

key==22
見つかったので終了

1.4.4 2分探索木

■ 探索の例2 (木に含まれないkey=68を探索する場合)



根からスタートする
50<key，右部分木へ

key<75，左部分木へ

67<key，葉に到達しても
見つからないので終了

1.4.4 2分探索木

■ 探索の例

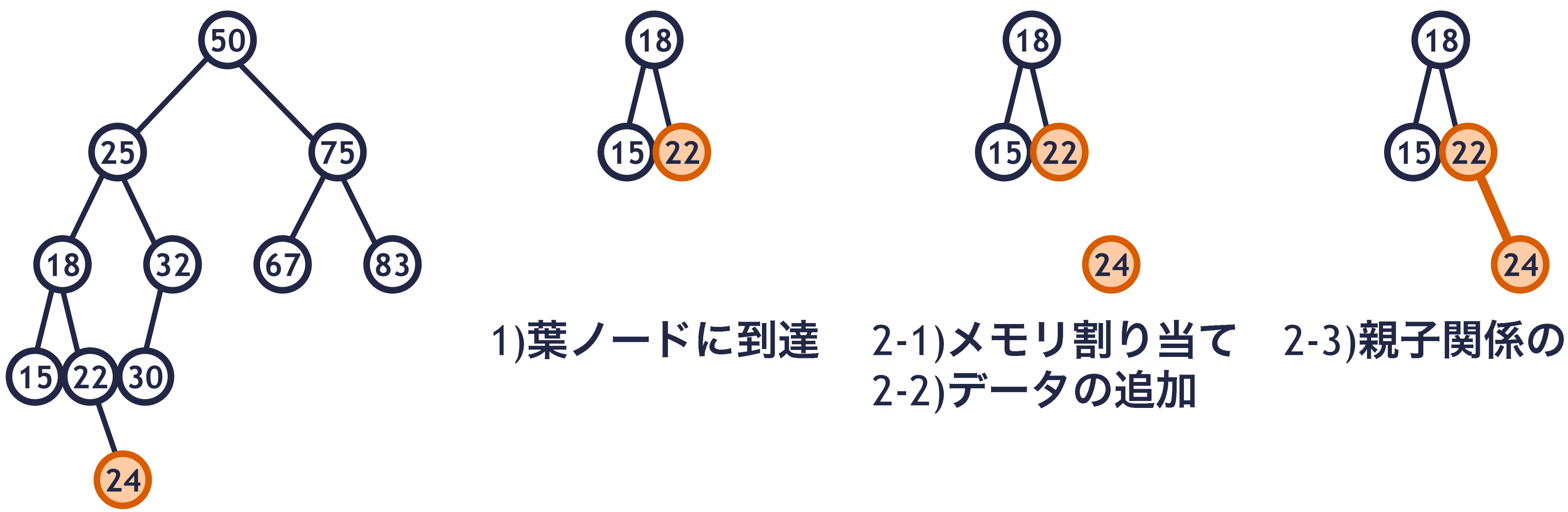
1. 初期化：根からスタートする
2. 比較：探索するデータkeyを現在訪れているノードの要素と比較する

繰返し

3. 移動：keyの方が小さければ左部分木へ，keyの方が大きければ右部分木へ移動する
4. 終了条件(1)：keyと等しい要素が見つかれば終了する
5. 終了条件(2)：葉に到達しても見つからなければ，この木にkeyは含まれないので終了する

1.4.4 2分探索木

- 挿入の例 (24を新たに追加する場合)
  1. 探索の場合と同様に，根から比較と移動を繰り返す
  2. 最後に訪れたノードにxを新たな子として追加する
     1. 新たな子のためのメモリを割り当てる
     2. データを追加する
     3. 親子関係を更新する



1)葉ノードに到達

2-1)メモリ割り当て
2-2)データの追加

2-3)親子関係の更新

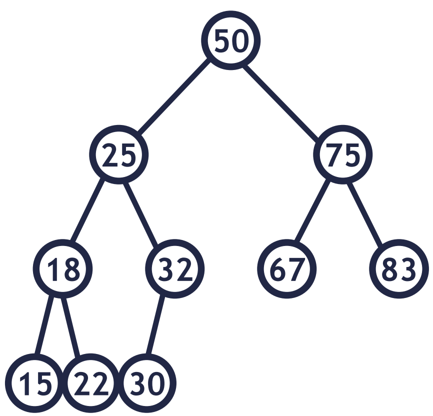
1.4.4 2分探索木

- **削除の例**
  1. 探索の場合と同様に，削除するノードへ移動する
  2. ノードを削除する

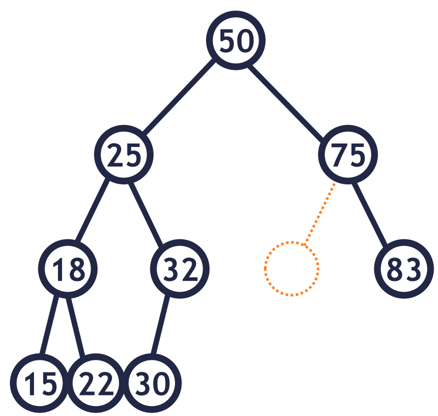
     Case1：削除するノードが葉の場合 → 葉を削除する

     Case2：葉ではなく，1つの子を持つ場合 → 子で置き換える

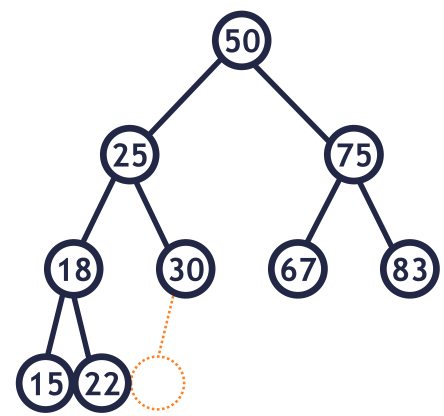
初期状態



Case1：delete(67)



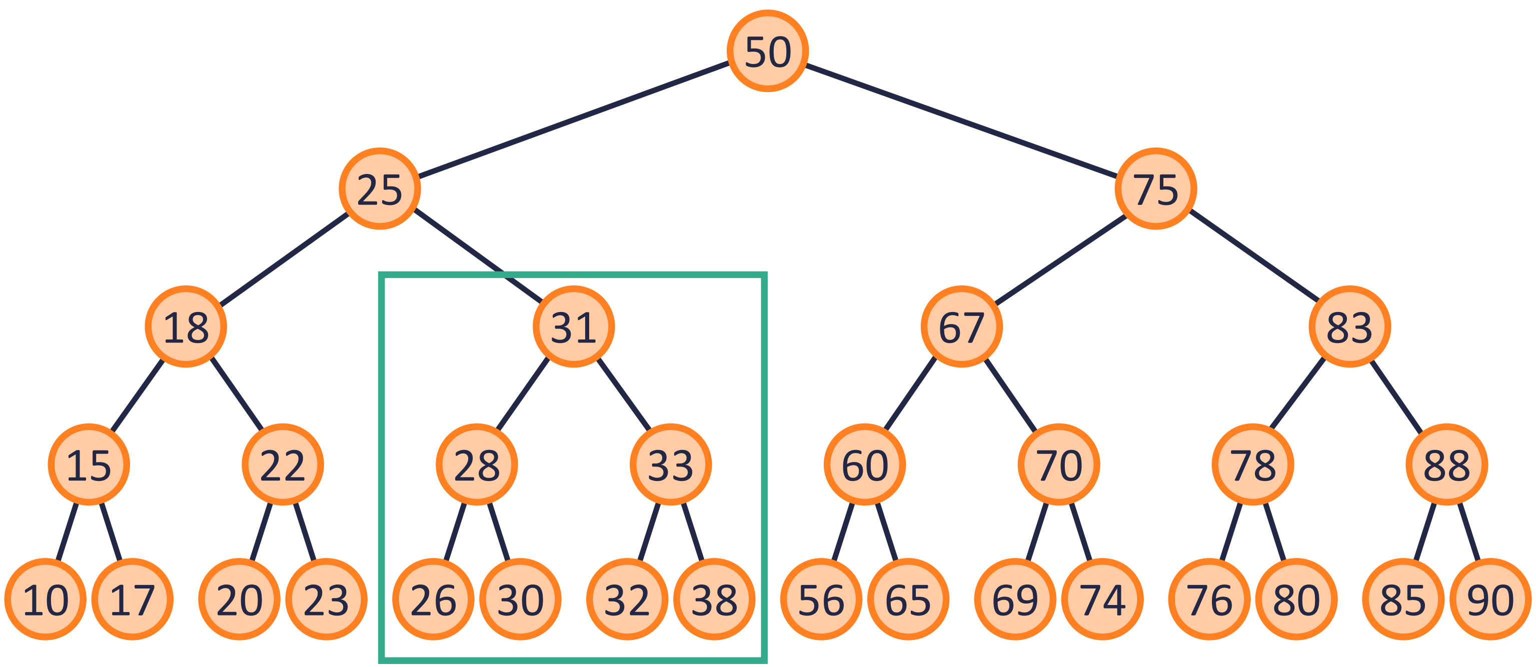
Case2：delete(32)



1.4.4 2分探索木

■ 削除の例

- あるノードの要素の次に大きな要素は，右部分木の左を繰り返し辿った先のノードの要素
- 例：25の次に大きな値は，その右部分木の左端の26



1.4.4 2分探索木

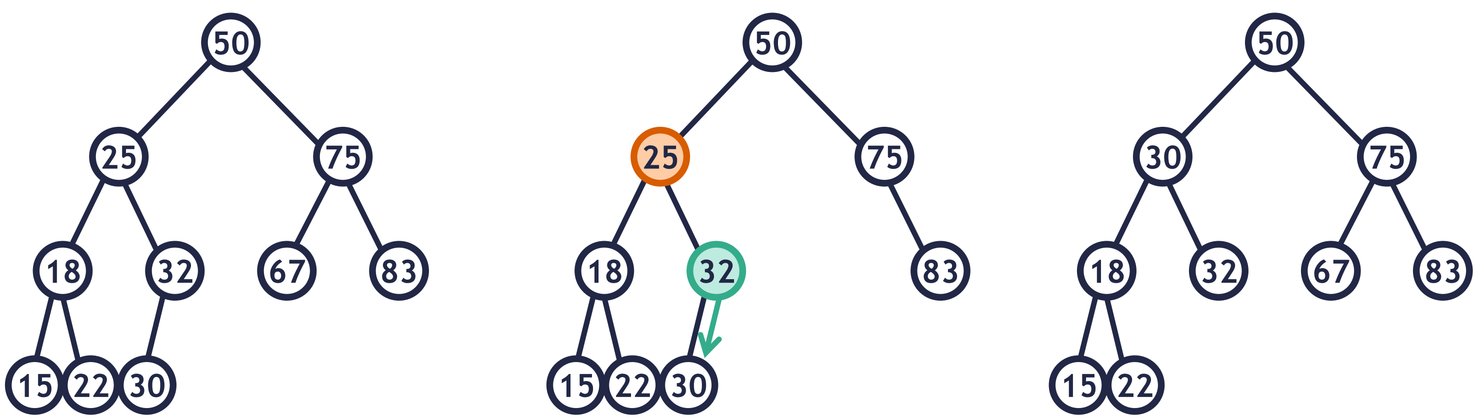
■ 削除の例

1. 探索の場合と同様に，削除するノードへ移動する
2. ノードを削除する

   Case3：葉ではなく，2つの子を持つ場合

   1. 削除ノードの右の子(32)を出発点とし，左の子を繰り返し辿る (削除ノードの次に大きな要素を見つける)
   2. 到達したノードの要素(30)を削除ノードに上書き

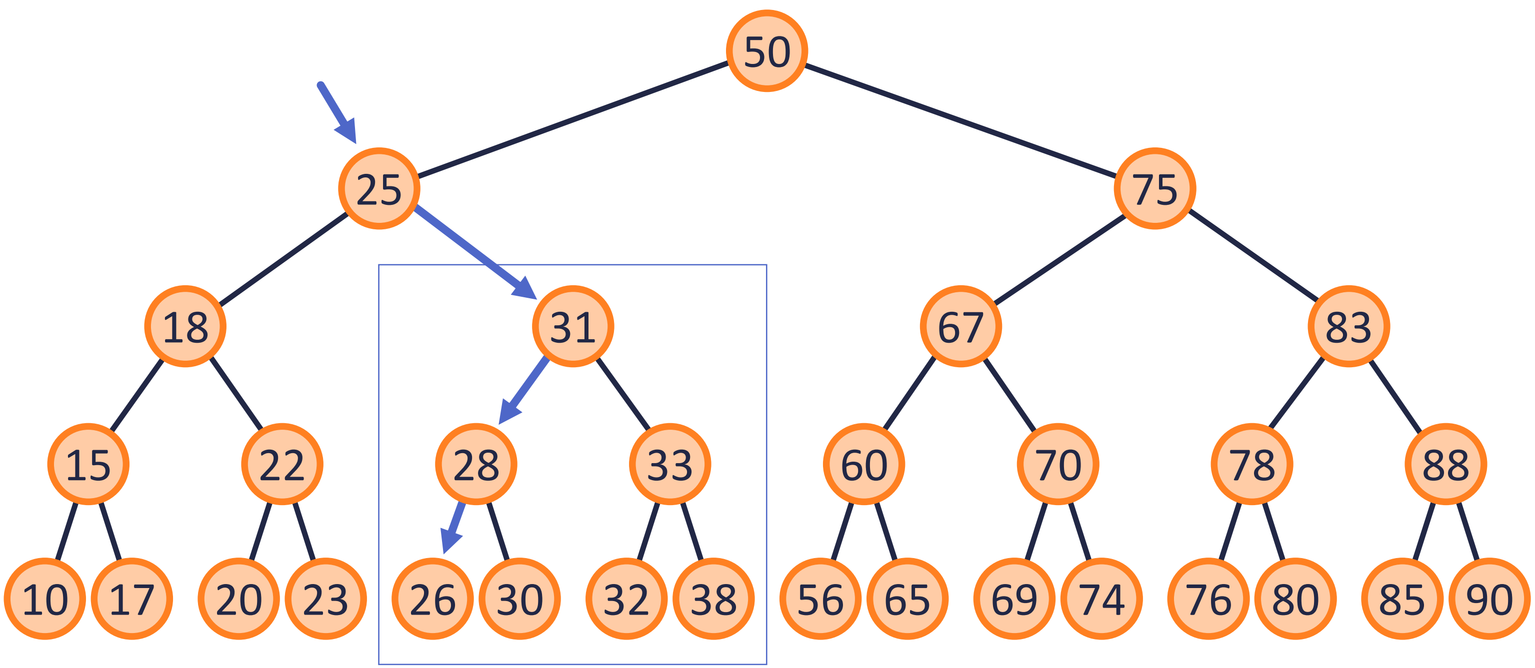
初期状態　　Case3：delete(25)



1.4.4 2分探索木

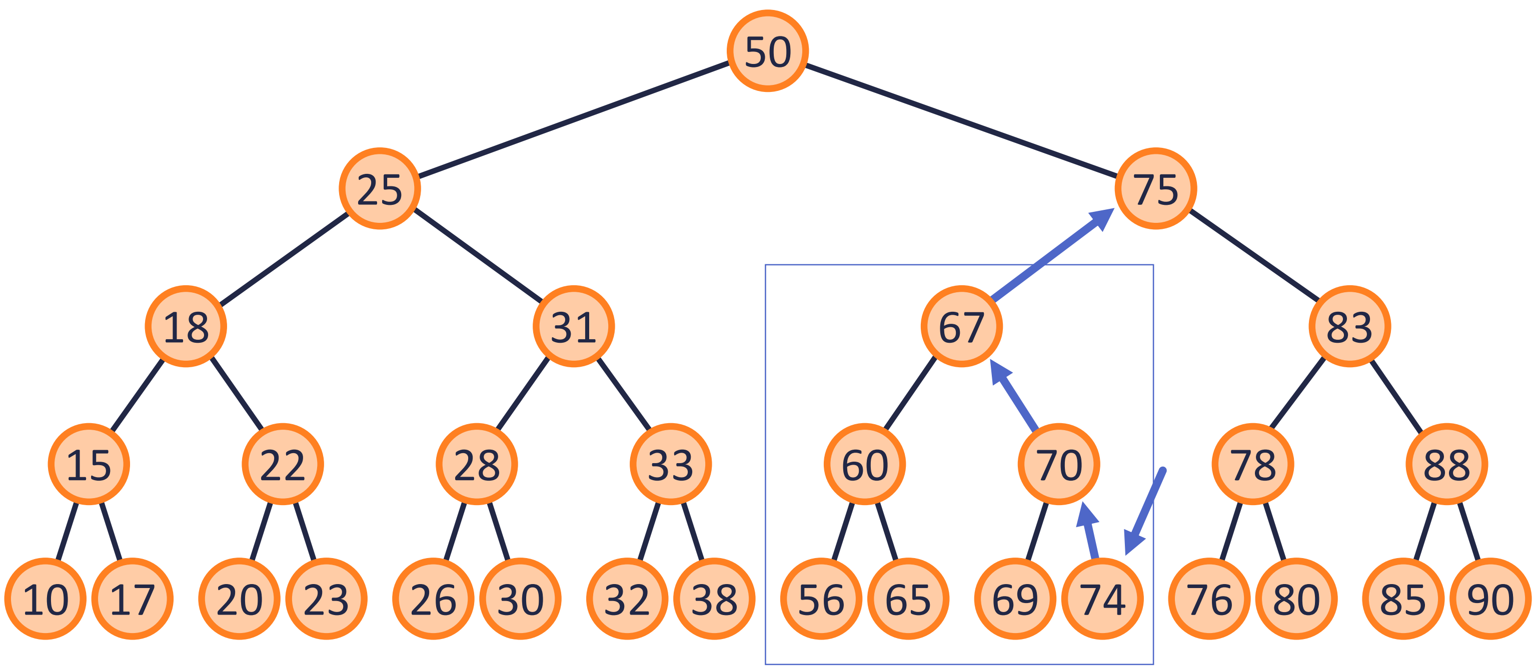
- 直後の要素の削除：delete_next(25)
  - 例1：25の直後の要素である26の削除(子を持つ場合)
  - 手順は削除の場合と同様

1. 要素25を持つノードを探す
2. 見つかれば，その右部分木の左端を探索して削除する



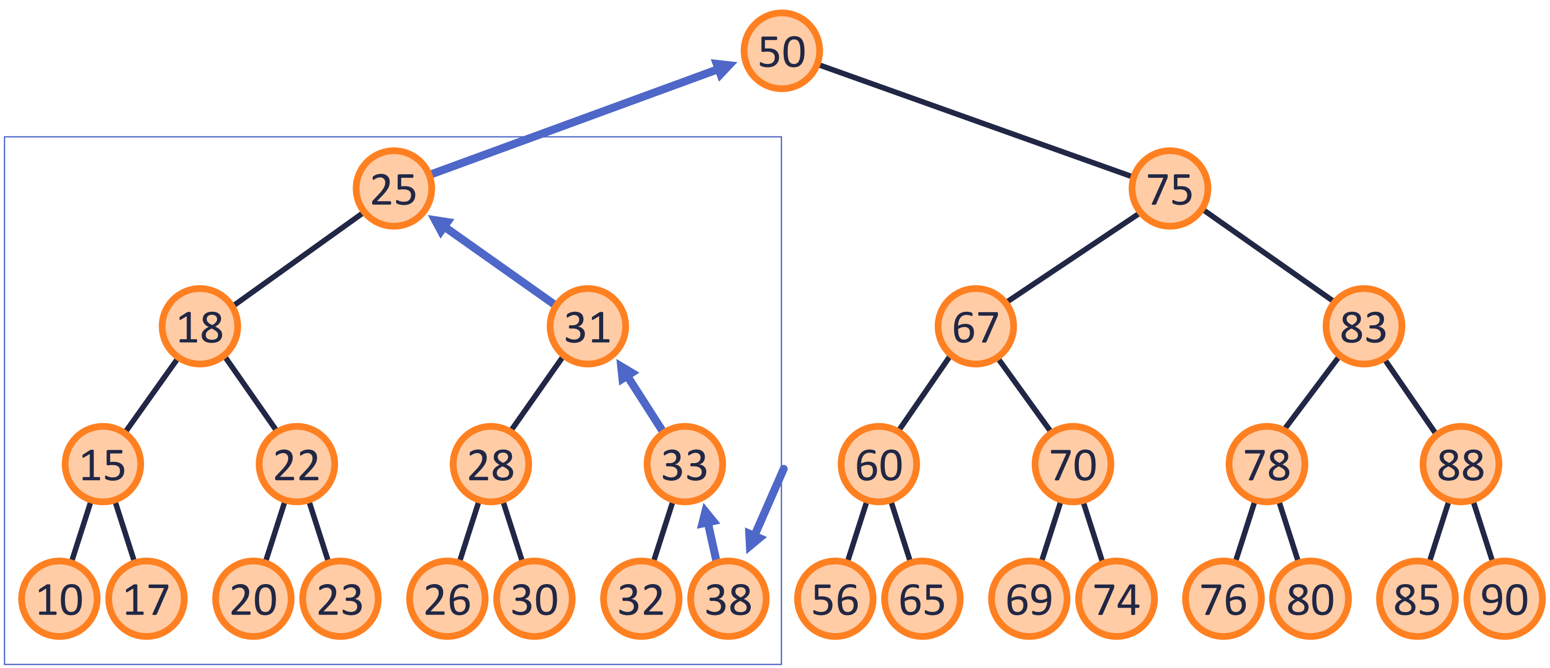
1.4.4　2分探索木

- 直後の要素の削除：delete_next(74)
  - 例2：74の直後の要素である75の削除(子を持たない場合)
  - 手順が削除の場合とは異なる
  - 要素74を持つノードを探す
  - 親を辿り，最初に左部分木へ移動した辺の親ノードを探す



1.4.4 2分探索木

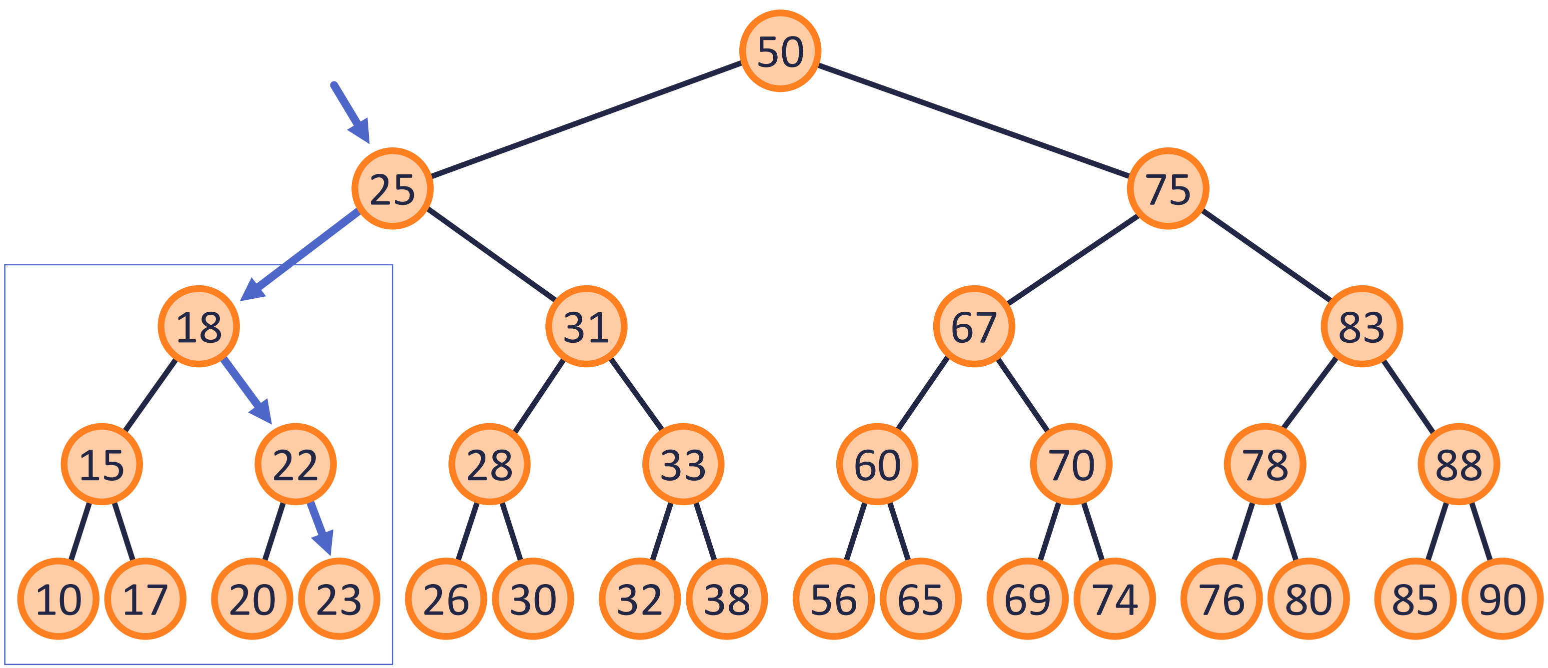
- 直後の要素の削除：delete_next(36)
  - 例3：38の直後の要素である50の削除(子を持たない場合)
  - 手順が削除の場合とは異なる
  - 要素38を持つノードを探す
  - 親を辿り，最初に左部分木へ移動した辺の親ノードを探す



1.4.4 2分探索木

- 直前の要素の削除：delete_prev(25)
  - 例1：25の直前の要素である23の削除(子を持つ場合)
  - 手順は直後の要素の削除の場合と同様

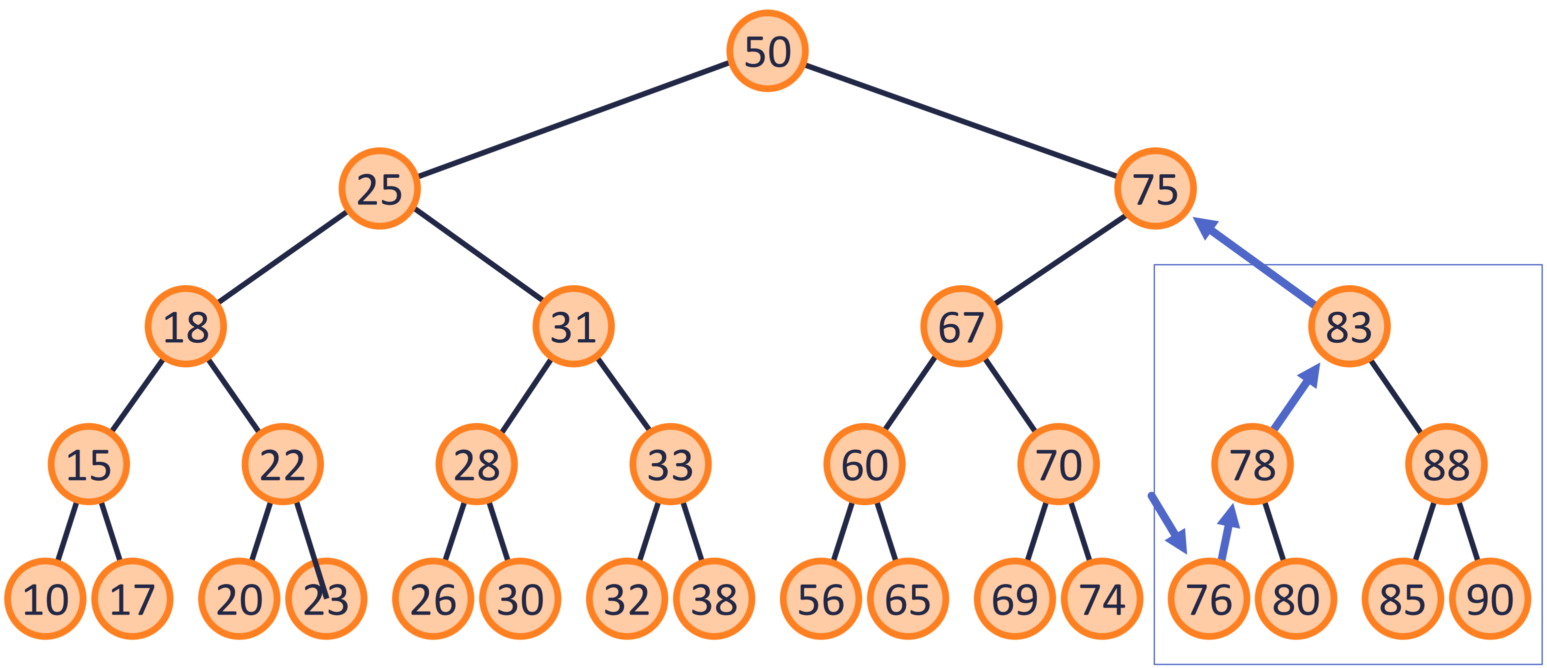
1. 要素25を持つノードを探す
2. 見つかれば，その左部分木の右端を探索して削除する



1.4.4 2分探索木

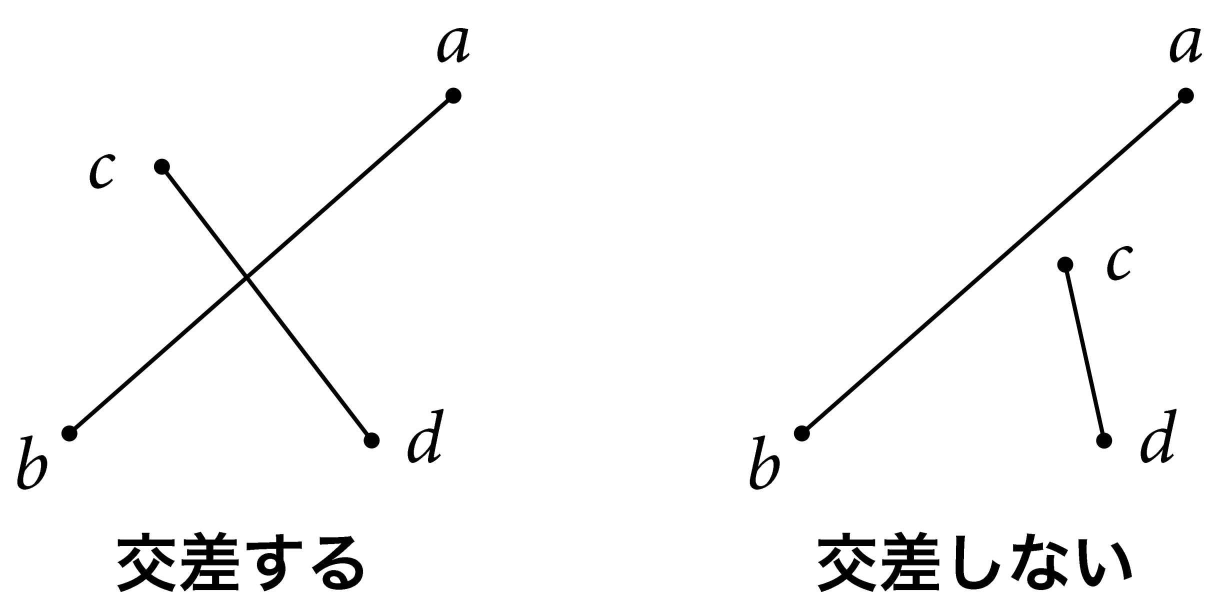
- 直前の要素の削除：delete_prev(76)
  - 例2：25の直前の要素である23の削除(子を持つ場合)
  - 手順は直後の要素の削除の場合と同様

1. 要素76を持つノードを探す
2. 親を辿り，最初に右部分木へ移動した辺の親ノードを探す



2.1 2線分の交差

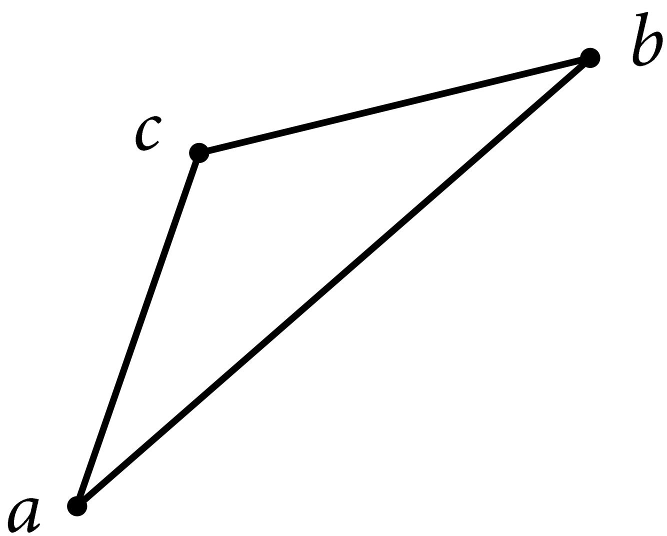
- 三角形の符号付き面積を利用する方法：
  - 基本的な考え方：
    - 2本の線分abとcdが互いに交わるならば，端点cと端点dが線分abを含む直線によって分離される
    - 同様に，aとbもcdを含む直線によって分離される



2.1 2線分の交差

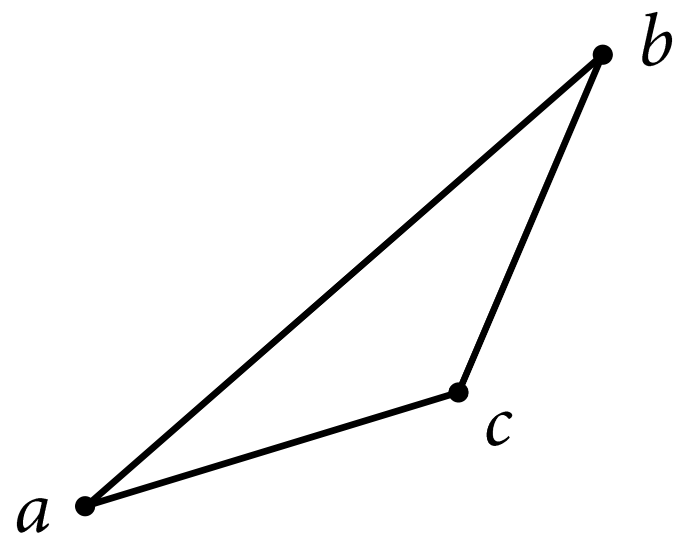
- 三角形の符号付き面積を利用する方法：
  - 面積の符号は<u>3点(a, b, c)の順序</u>で決まる

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

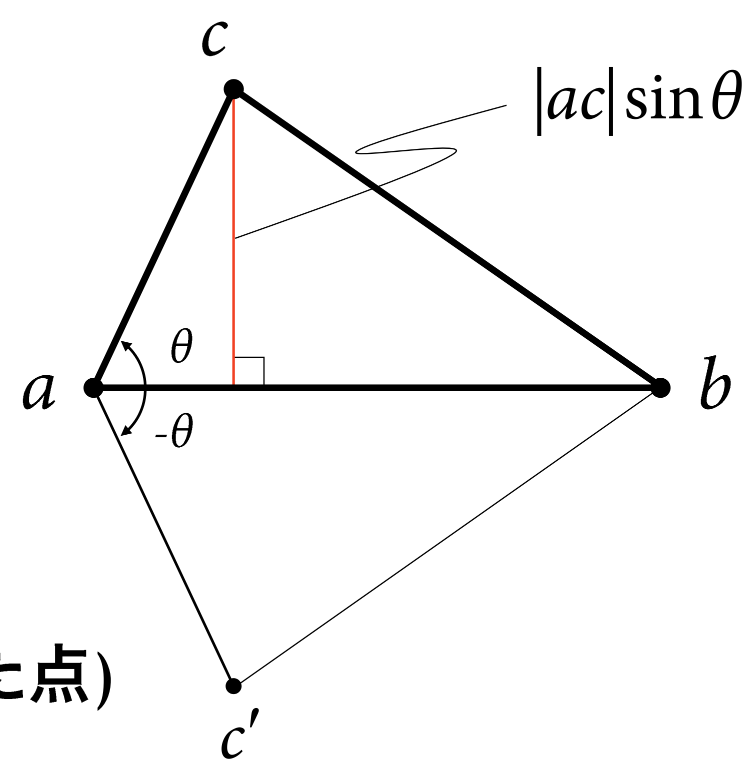
■ 符号付き面積の算出

- 三角形*abc*の面積

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

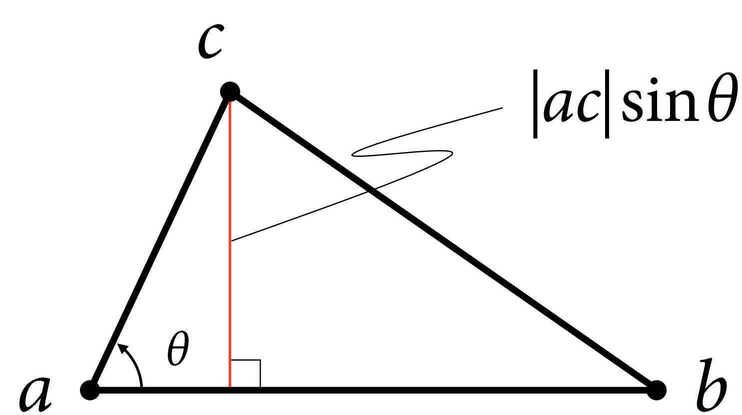
- 三角形*abc'*の面積
(*c'*は*ab*に対して反射させた点)

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



2.1 2線分の交差

■ 外積を使った符号付き面積の算出



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

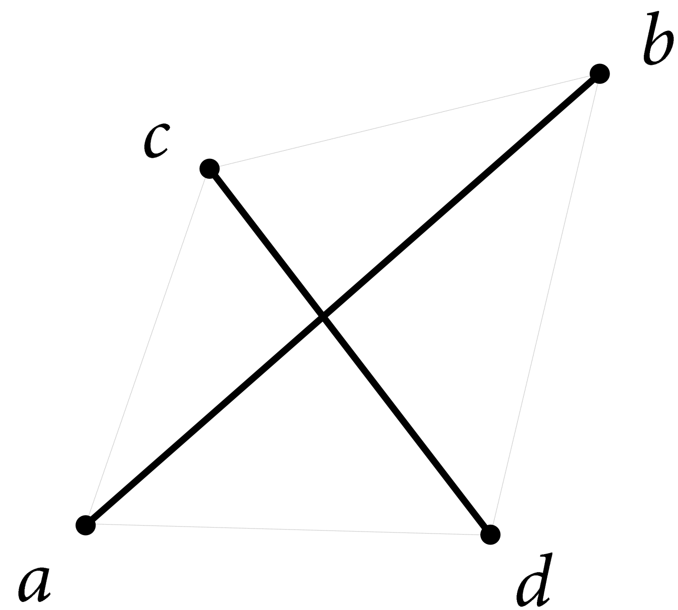
2.1 2線分の交差

■ 三角形の符号付き面積を利用する方法

- △abcと△abd，△cdaと△cdbの符号付き面積がいずれも異なる符号を持てば交差する

交差する場合

1. 3点a,b,cの順は反時計回り
   → 符号付き面積は正
2. 3点a,b,dの順は時計回り
   → 符号付き面積は負
3. 3点c,d,aの順は時計回り
   → 符号付き面積は負
4. 3点c,d,bの順は反時計回り
   → 符号付き面積は正



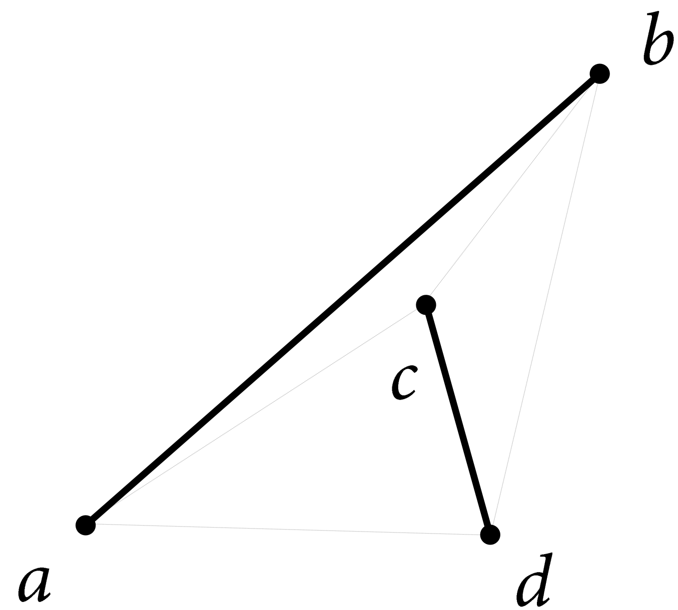
2.1 2線分の交差

■ 三角形の符号付き面積を利用する方法

- △abcと△abd，△cdaと△cdbの符号付き面積がいずれも異なる符号を持てば交差する

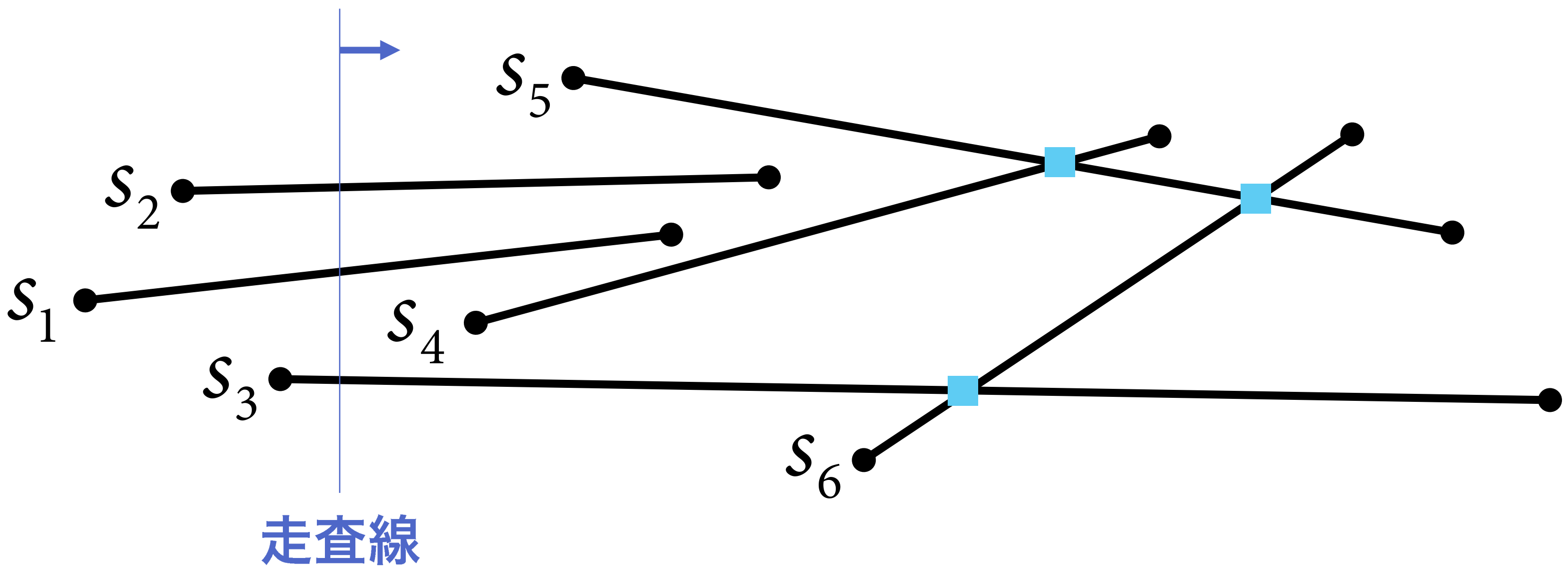
<u>交差しない場合</u>

1. 3点<u>a,b,cの順</u>は時計回り
   → 符号付き面積は負
2. 3点<u>a,b,dの順</u>は時計回り
   → 符号付き面積は負
3. 3点<u>c,d,aの順</u>は時計回り
   → 符号付き面積は負
4. 3点<u>c,d,bの順</u>は反時計回り
   → 符号付き面積は正

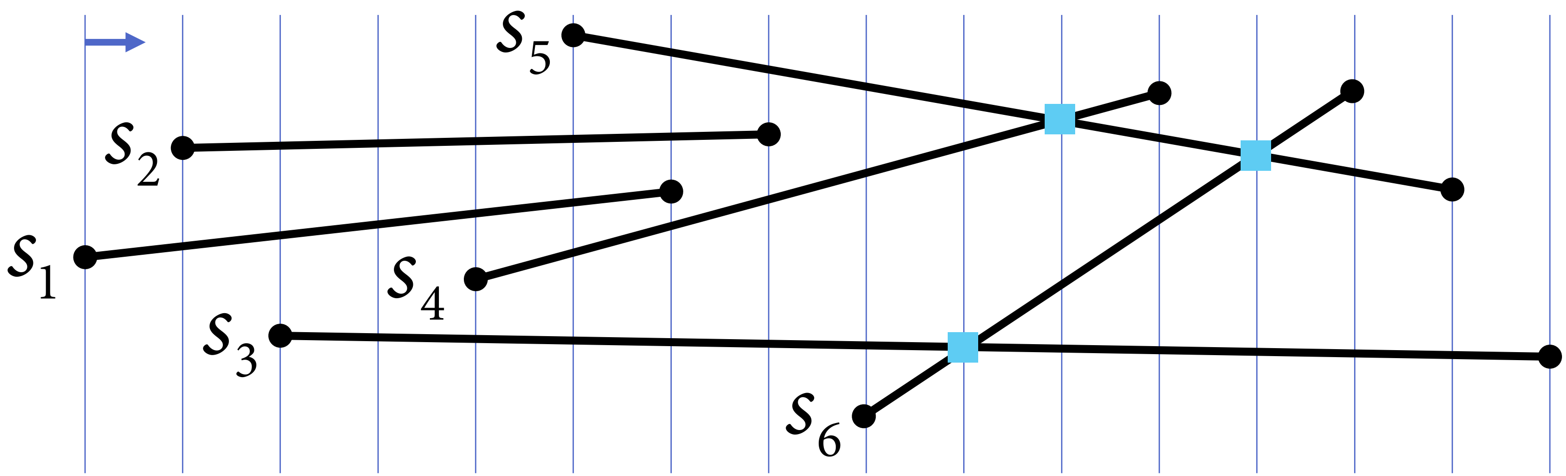


2.2.2 一般の線分

- 問題：平面上に線分がn本与えられたとき，それらの中に互いに交差するペアをすべて判定せよ
  - 仮定：垂直線，水平線は含まれないものとする
- 解法：走査線を水平方向(左→右)に移動させながら，垂直方向に隣接する線分ペアの交差を調べる



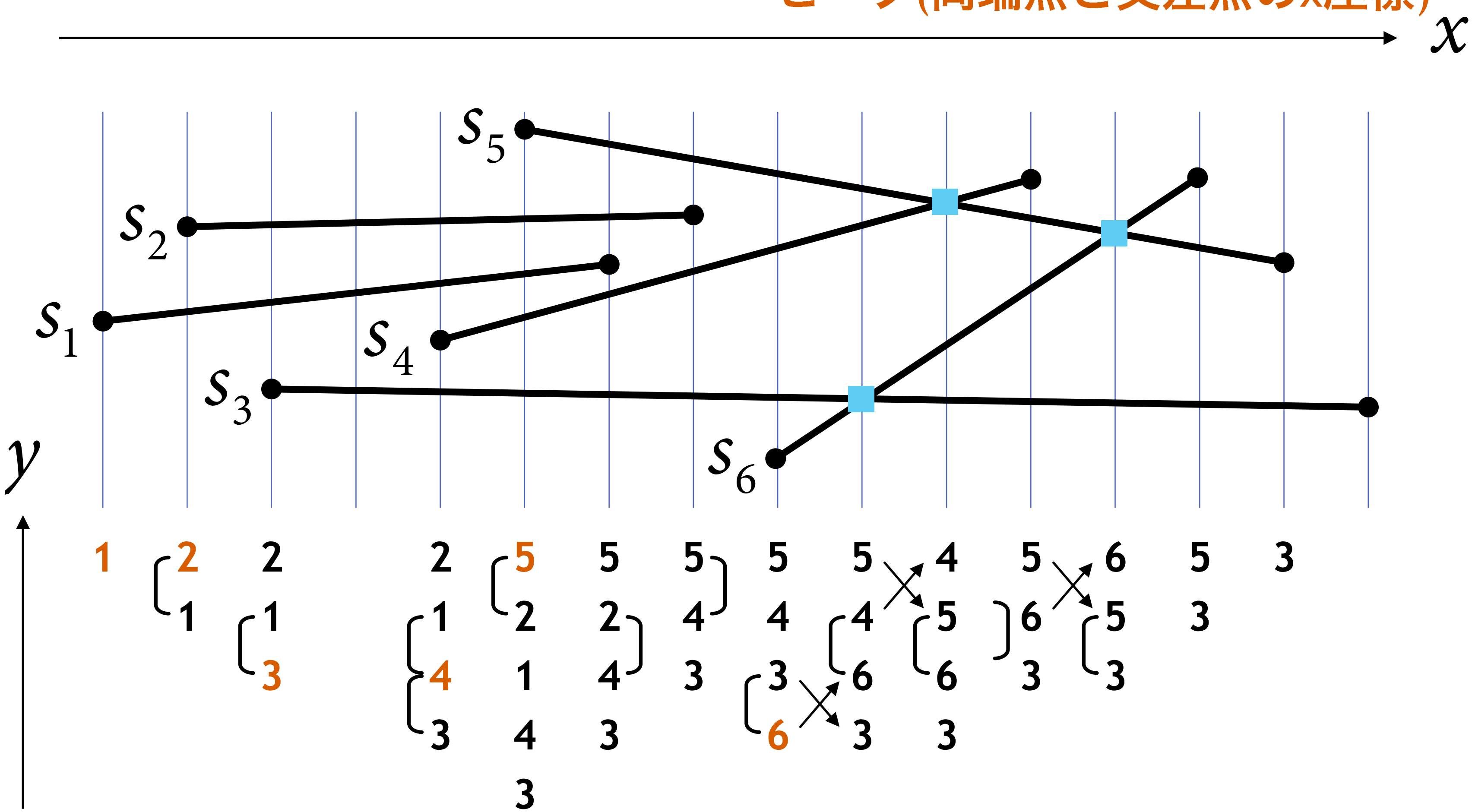
2.2.2 一般の線分



1	2	2		2	5	5	5	5	5	4	5	6	5	3	
	1	1		1	2	2	4	4	4	5	6	5	3		
		3		4	1	4	3	3	6	6	3	3			
				3	4	3		6	3	3					
					3										

2.2.2 一般の線分

イベント点(左端点，右端点，交差点)
→ ヒープ(両端点と交差点のx座標)



走査線と交わる線分 → 2分探索木(線分の左端点のy座標)

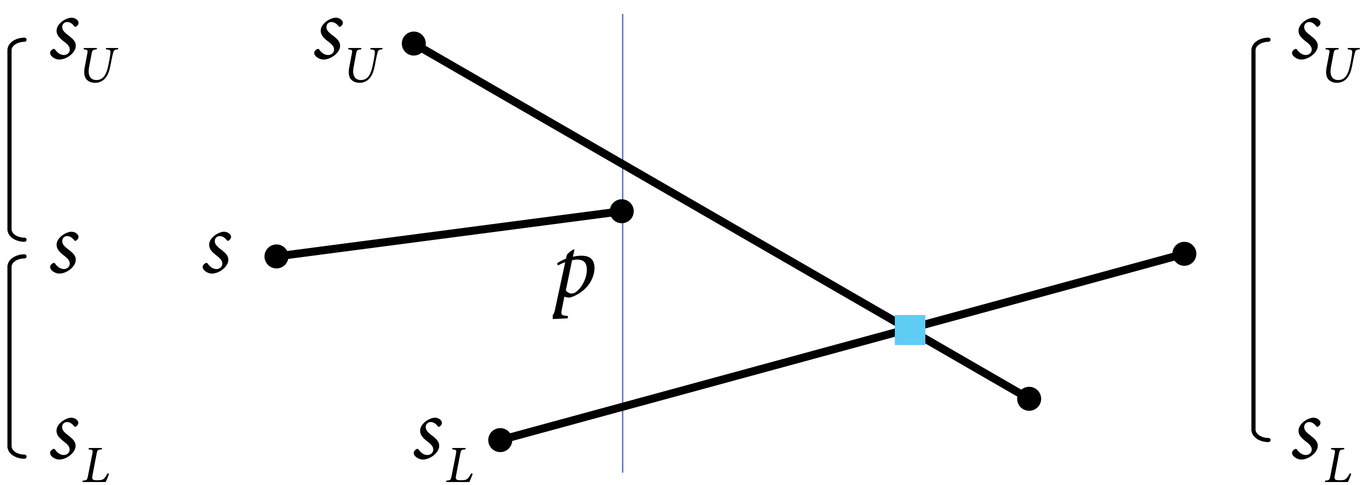
2.2.2 一般の線分

- 走査線と交わる線分
  - 走査線は左→右へ移動
  - 走査線と交わる線分の管理は2分探索木で実現
- イベント点
  - 線分の左右の端点 + 交差点(判定の途中で現れる)
  - リストでは追加・削除に時間がかかってしまう
  - ヒープで実現する

2.2.2 一般の線分

■ 右端点の場合

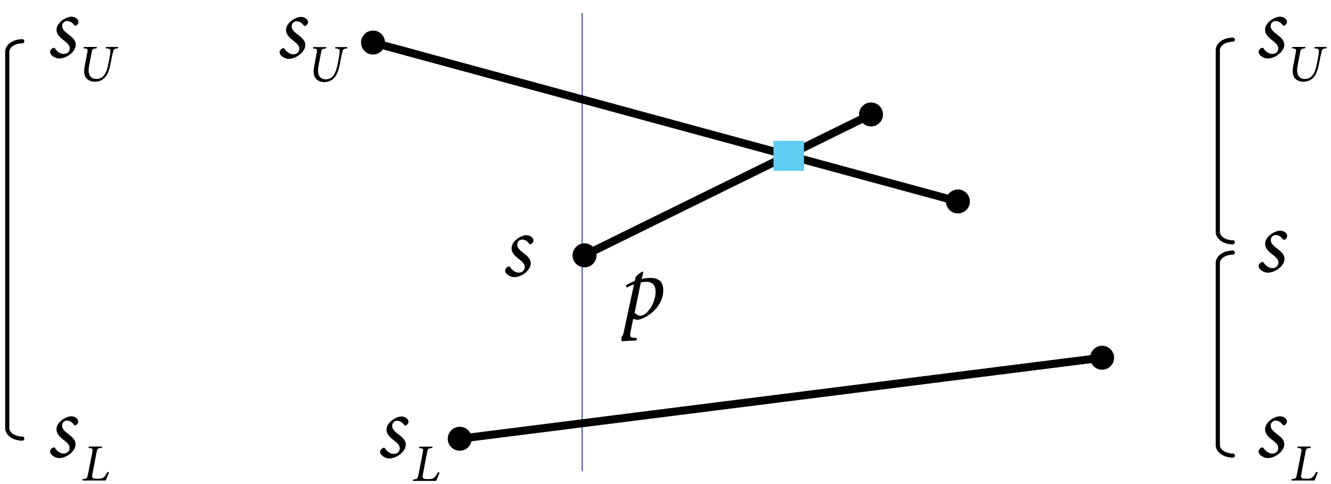
- Pの上下の線分の交差判定を行う $< s_U, s_L >$
- 交差する → 交差点をヒープに追加



2.2.2 一般の線分

■ 左端点の場合

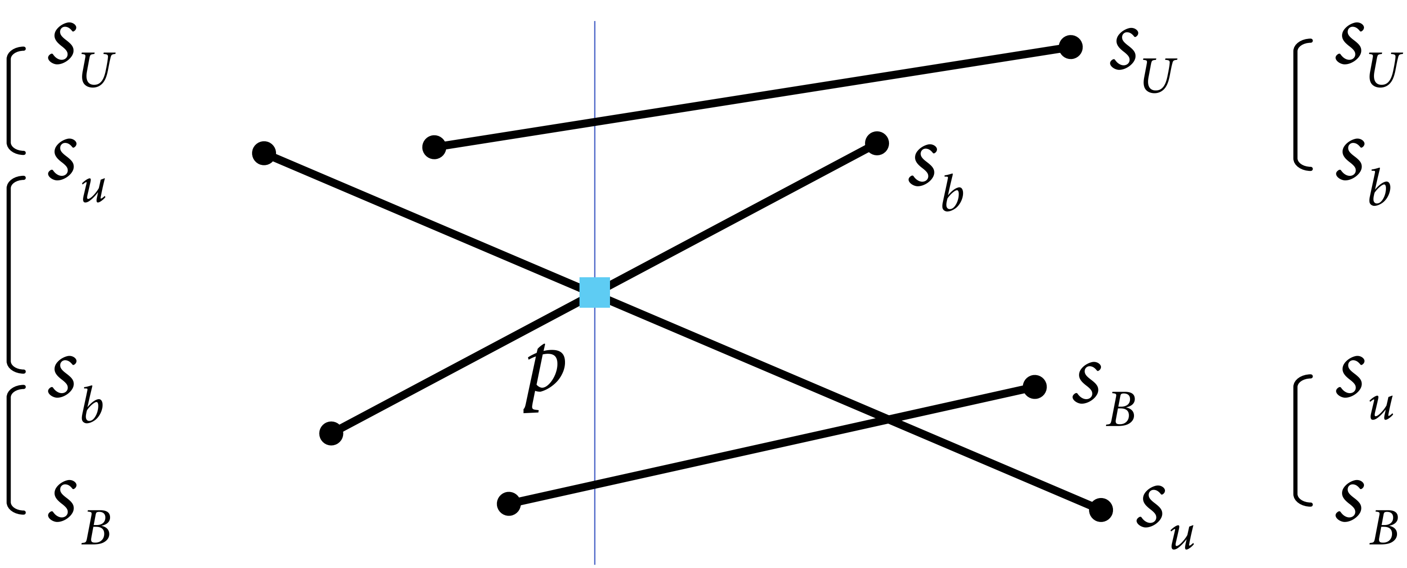
- Pを端点に持つ線分と，その上下の線分の2ペアに対して交差判定を行う $< s, s_U >$ $< s, s_L >$
- 交差する → 交差点をヒープに追加



2.2.2 一般の線分

■ 交差点の場合

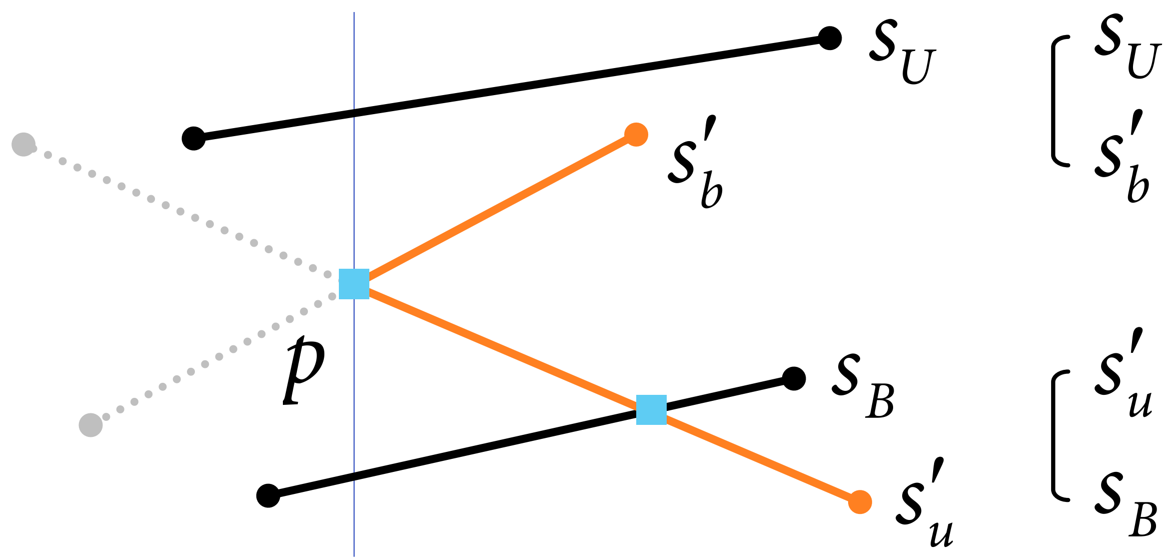
- 隣接関係が変化することにより生成される2ペアに対して交差判定を行う $< s_U, s_u >$ $< s_b, s_B >$
- 交差する → 交差点をヒープに追加



2.2.2 一般の線分

■ 交差点の場合

- 交差点pを新たな左端点とする2線分を2分探索木Tに挿入する $< s_U, s'_u >$ $< s_b, s'_B >$
- 新たに隣接する2ペアに対して交差判定を行う



2.2.2 一般の線分

■ 擬似コード

1. 線分の端点をx座標をキーとしてヒープHに挿入する
2. 2分探索木Tを空にする
3. While(Hが空でなければ)
   1. Hから最小要素pを取り出す(イベント点)
   2. pが線分lの左端点ならば
      - lをTに挿入する
      - lと隣接するT内の上下2本の線分とlとの間でそれぞれ交差判定を行う
      - 交点があれば，交点を左端点(イベント点)としてHに挿入
   3. pが線分lの右端点ならば
      - lをTから削除する
      - 新たに隣接する線分の交差判定を行う
      - 交点があれば，交点を左端点(イベント点)としてHに挿入