

# 第13回のキーワード

1

## アルゴリズム関係

- 2分探索木  
(binary search tree)
- 平衡木 (balanced tree)
- AVL木 (Adelson-Velsky and Landis' tree) ← 2人
- 木の回転 (tree rotation)
- 1重回転 / 2重回転  
(single/double rotation)
- $O(\log n)$

## Java関係 (前回)

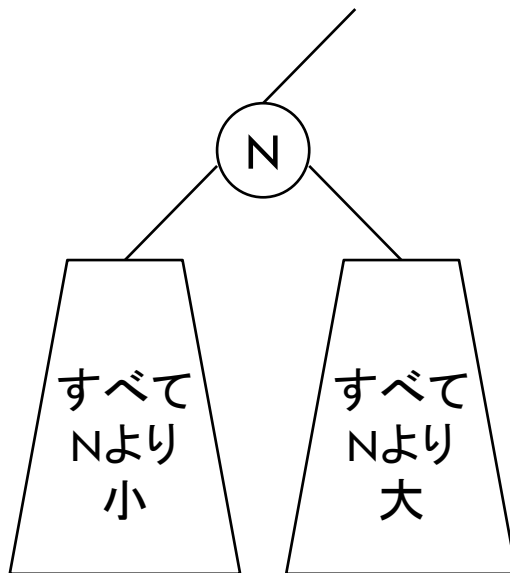
- 3項演算子
  - ▣ 条件式 ? 値1 : 値2
- `class X<E extends Y> { ... }`
  - ▣ 型パラメータ(E)の限定条件
  - ▣ Eは必ずクラスYを継承するか  
インターフェースYを実装する
- `<? extends T>`
  - ▣ ジェネリクスの変数宣言
  - ▣ TまたはTの子クラス(子インターフェース)を許容する
- `<? super T>`
  - ▣ TまたはTの親クラス(親インターフェース)を許容する

# 2分探索木

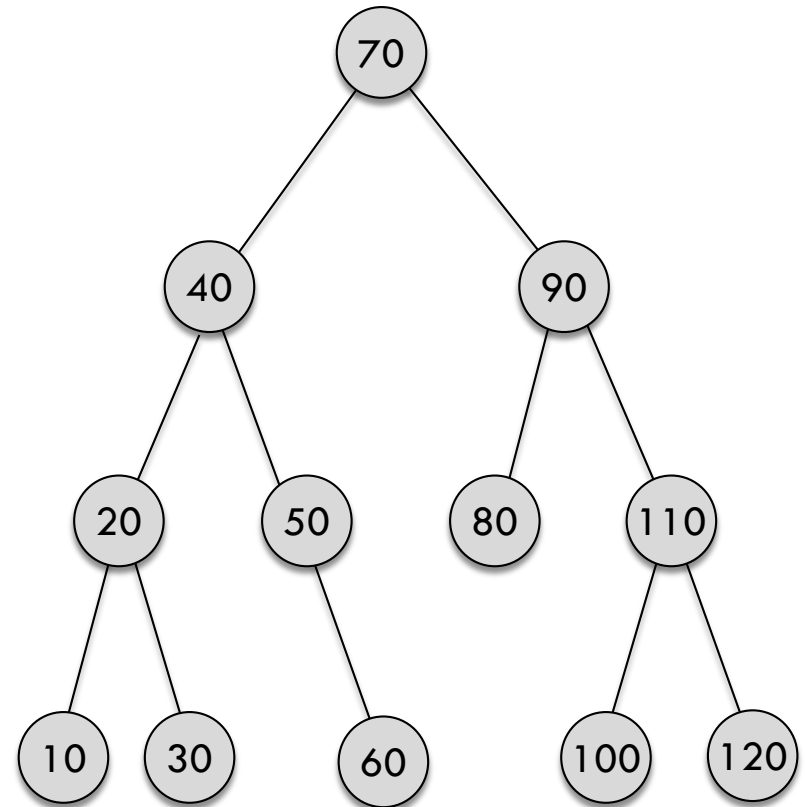
2

## □ 定義

- 任意のノードNについて
- Nの左の子孫(左の部分木の要素)は、すべてNより小さい
- Nの右の子孫(右の部分木の要素)は、すべてNより大きい



## □ 例



# 要素の探索と追加

3

## □ 要素の探索

- 根から要素を比較していく
- ノードとキーとの大小(前後)関係により, 右または左の子をたどっていく

## □ 要素の追加

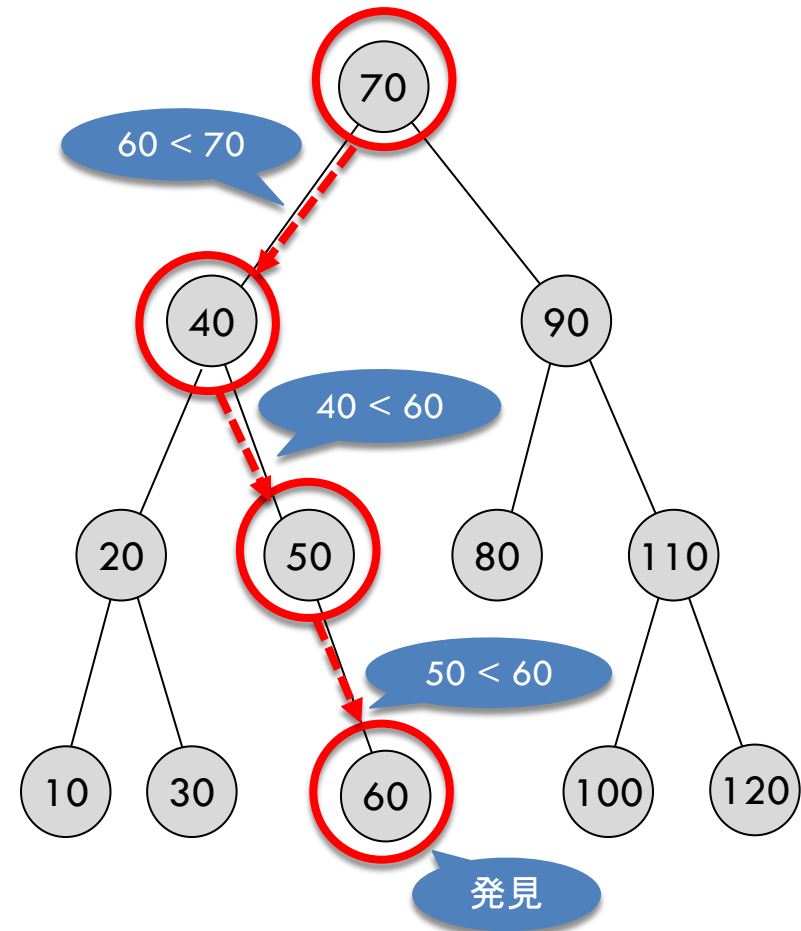
- 探索によって要素を発見できなかったら, その位置に新しいノードを追加する

## □ 木のバランス

- 理想的な2分探索木は?
- そのときの平均計算量は?

## □ 例

- 60を探索する場合



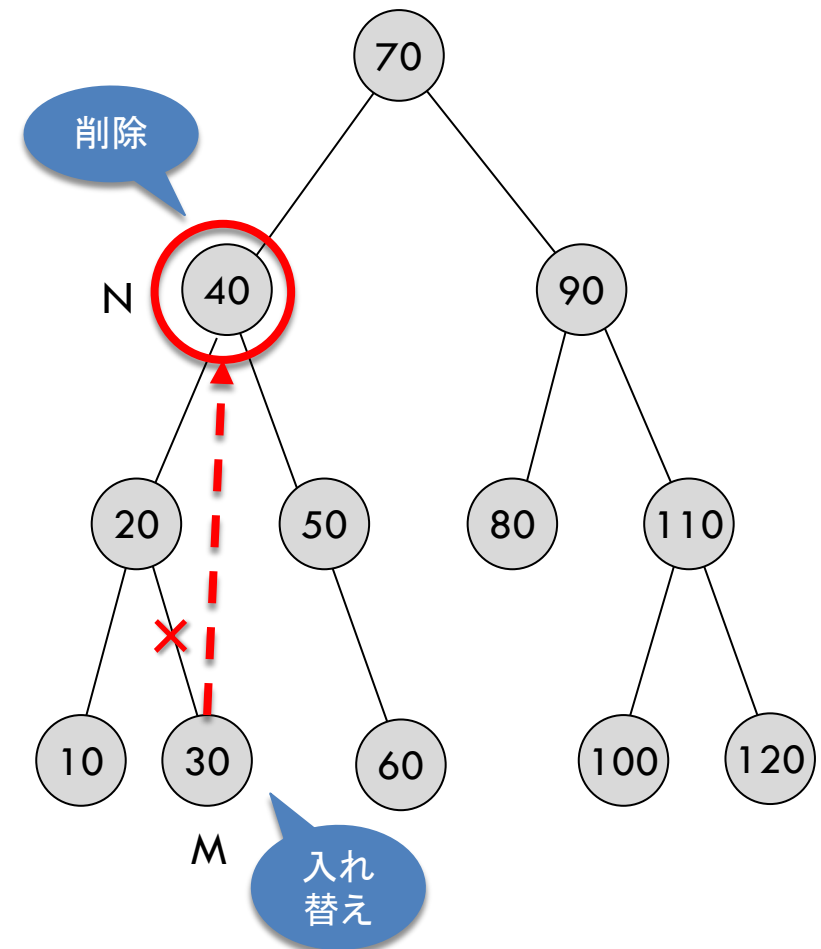
# 要素の削除

4

- 削除するノードをNとする
- 場合A: Nの子が0個なら
  - ▣ 単にNの親からNへの辺を切る
- 場合B: Nの子が1個なら
  - ▣ Nの親から, Nを飛ばしてそのひとりっ子につなげ直す
- 場合C: Nの子が2個なら
  - ▣ Nの左の子孫から最大値のノードMを求める(右の子孫から最小値でもよい)
  - ▣ いったんMを削除する  
その際, Mの子の数に応じて場合AまたはBの処理する
  - ▣ Nの位置にMを入れ替える  
(Nの子2個はMが引き継ぐ)

## □ 例

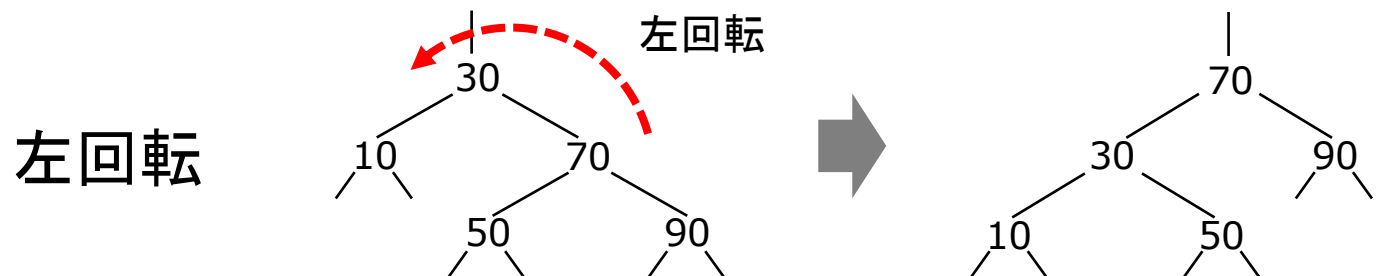
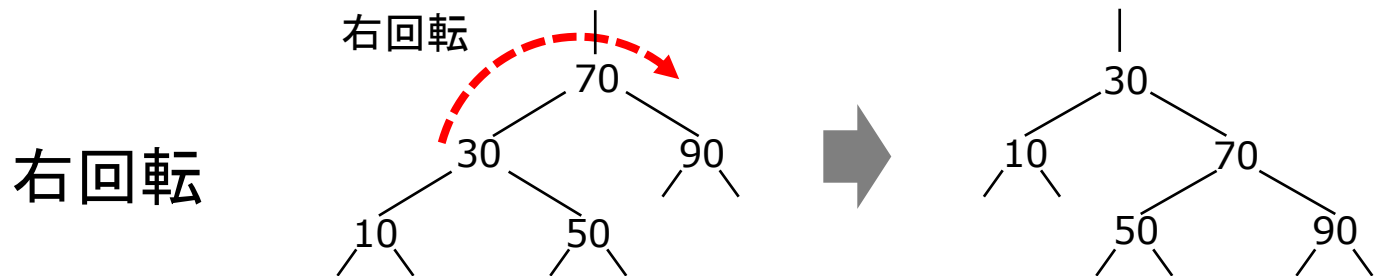
- ▣ 場合Cで「40」を削除する例



# AVL木(平衡木の一種)

5

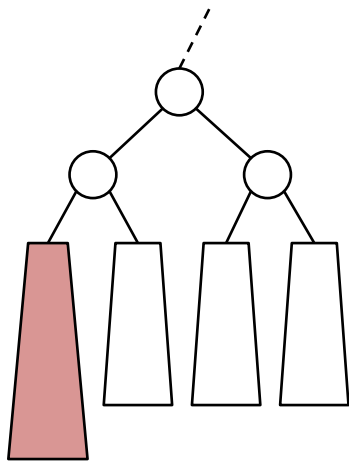
- 平衡木: バランスを維持する2分探索木
  - ▣ 回転操作: 2分探索木の条件を保ったまま変形する操作
  - ▣ 下図をよく観察して, ノードの入れ替わりを理解すること!



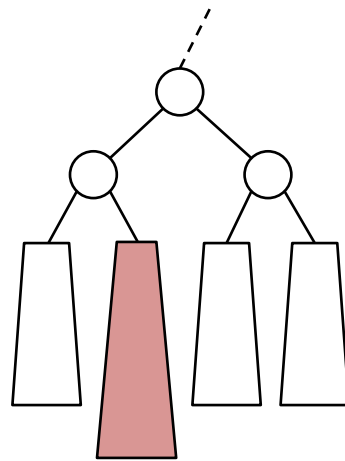
# AVL木: バランスが崩れた状態

6

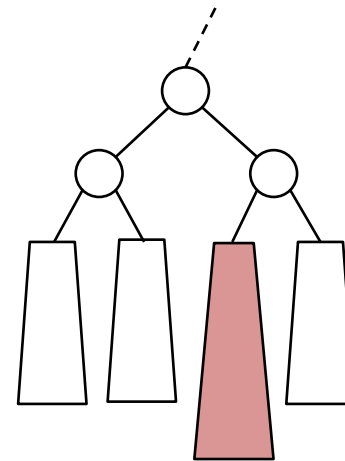
- 4パターンに整理して対処
  - ▣ まず, 左右のバランスが崩れているノードを検出する
  - ▣ そのノードの孫ノードの下の状態(高さ)によって分類する



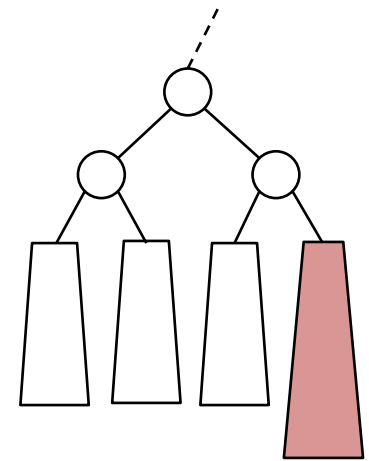
パターンA



パターンB



パターンC

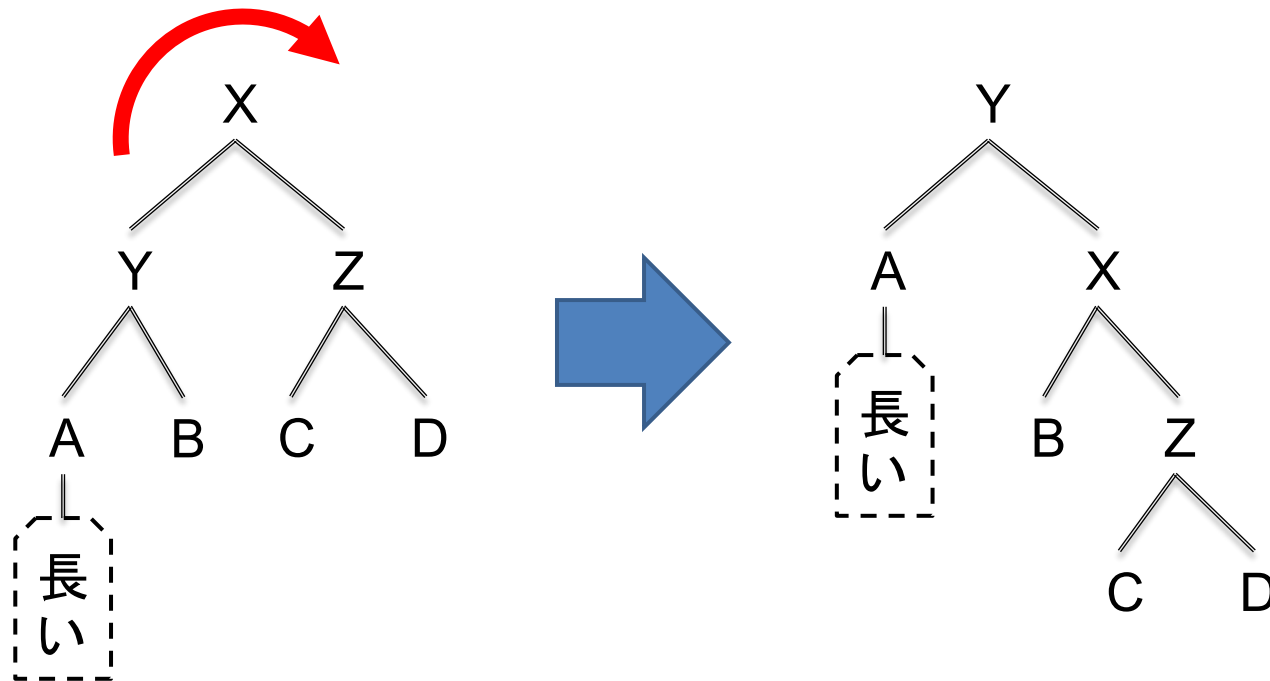


パターンD

# AVL木: 1重回転

7

- パターンA, パターンD
  - ▣ 1回の回転操作によって, バランスが維持できる
  - ▣ パターンDはAと対照なので, 回転を逆向きにすればよい



# AVL木: 2重回転

8

## □ パターンB, パターンC

- バランスを維持するには, 2回の回転操作が必要になる

