アルゴリズムとデータ構造

第12回　木構造と2分木

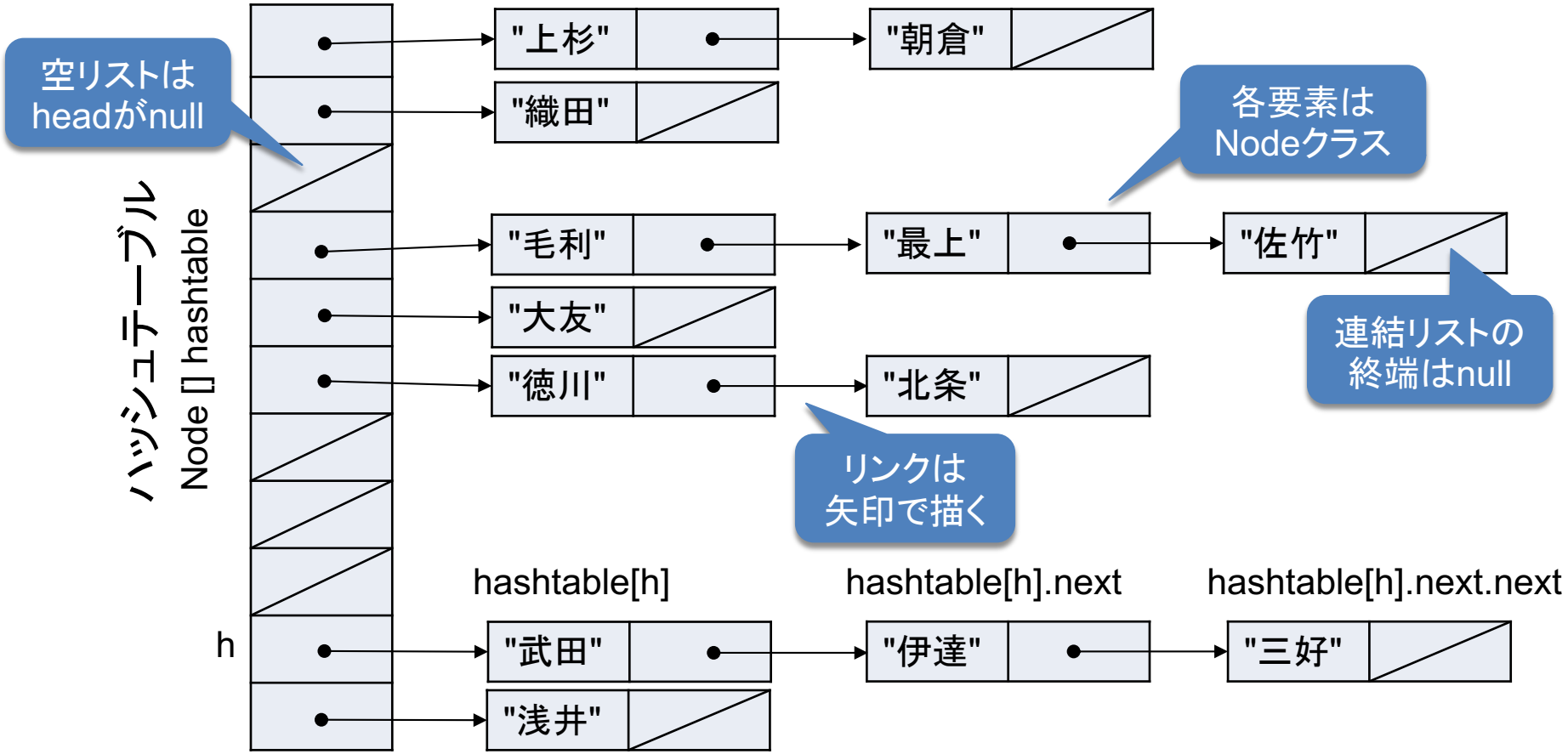
第12回のキーワード

アルゴリズム関係

- 木構造（tree structure）
- 木，根，節，葉
  （tree, root, node, leaf）
- 部分木（subtree）
- 親，子，きょうだい
  （parent, children, sibling）
- 木の高さ（height）
- 節の深さ（depth）
- 2分木 / 2進木
  （binary tree）
- 多分木 / N分木
  （multi-branch/n-ary tree）
- 木の巡回（走査）
  （tree traversal）
- 深さ優先探索
  （depth first search）
- 行きがけ順 / 先行順
  （preorder）
- 通りがけ順 / 中間順
  （inorder）
- 帰りがけ順 / 後行順
  （postorder）
- 幅優先探索
  （breadth first search）
- 2分探索木
  （binary search tree）

ハッシュテーブル：チェーン法

- 各バケットを連結リストにして複数の値を格納する
  - バケットには，リストの先頭要素への参照（head）が入る

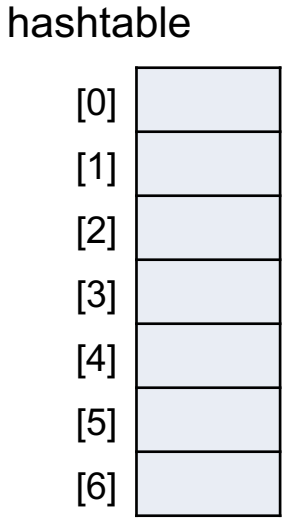


確認問題

- ハッシュテーブル
  - 以下のNodeクラスとその配列を用いて，チェーン法のハッシュテーブルを定義した

    class Node { int data; Node next; }

    Node [] hashtable = new Node[7];



  - ハッシュ関数は下記のメソッドを用いる

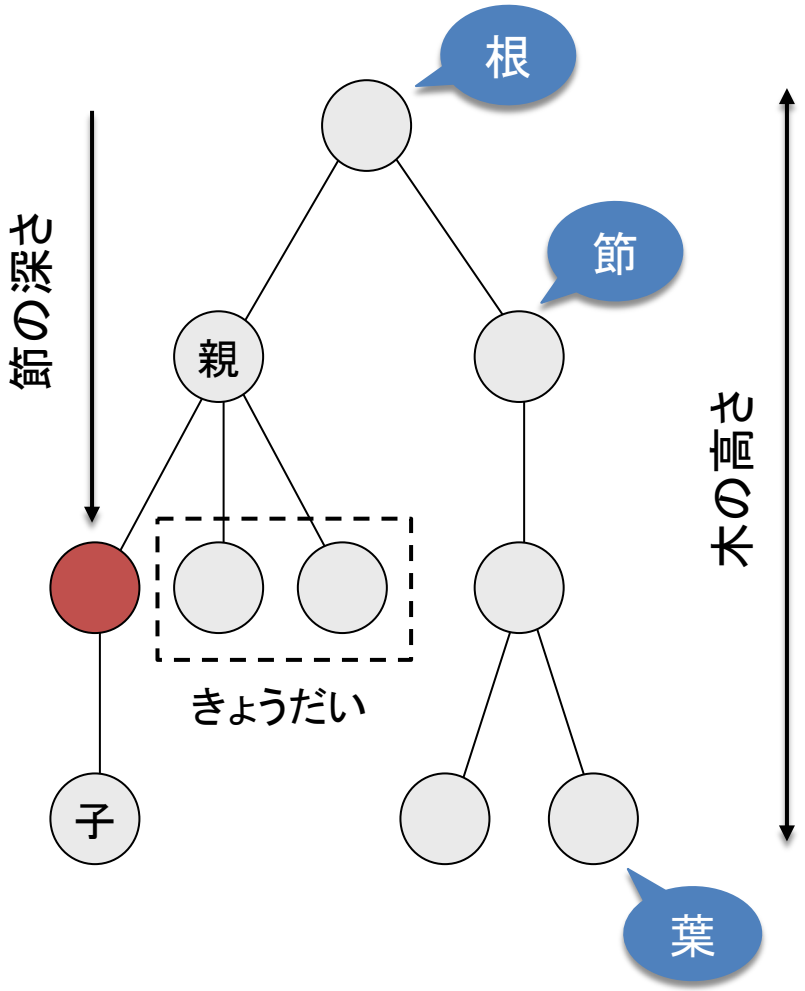
    public int hash(int data) { return (data * 3) % 7; }

  - このとき，下記の整数を全て登録するとハッシュテーブルの構造はどうなるか図示せよ

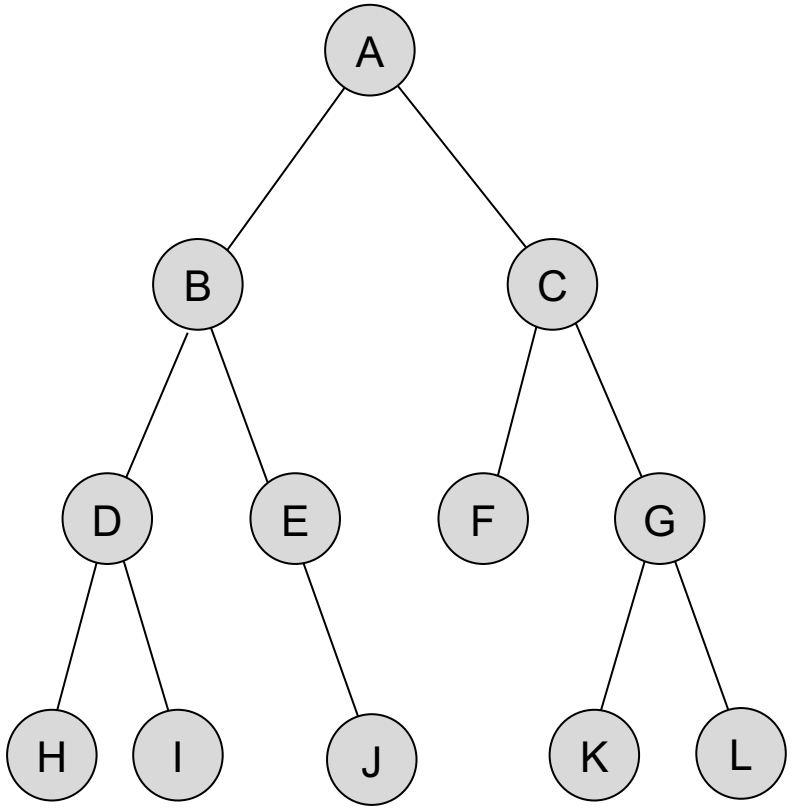
    8, 4, 3, 7, 1, 9, 2, 10

木構造(tree)

- 木構造(多分木)



- 2分木(2進木)
  - 子の個数が2つ以下
  - 子の左右を区別する

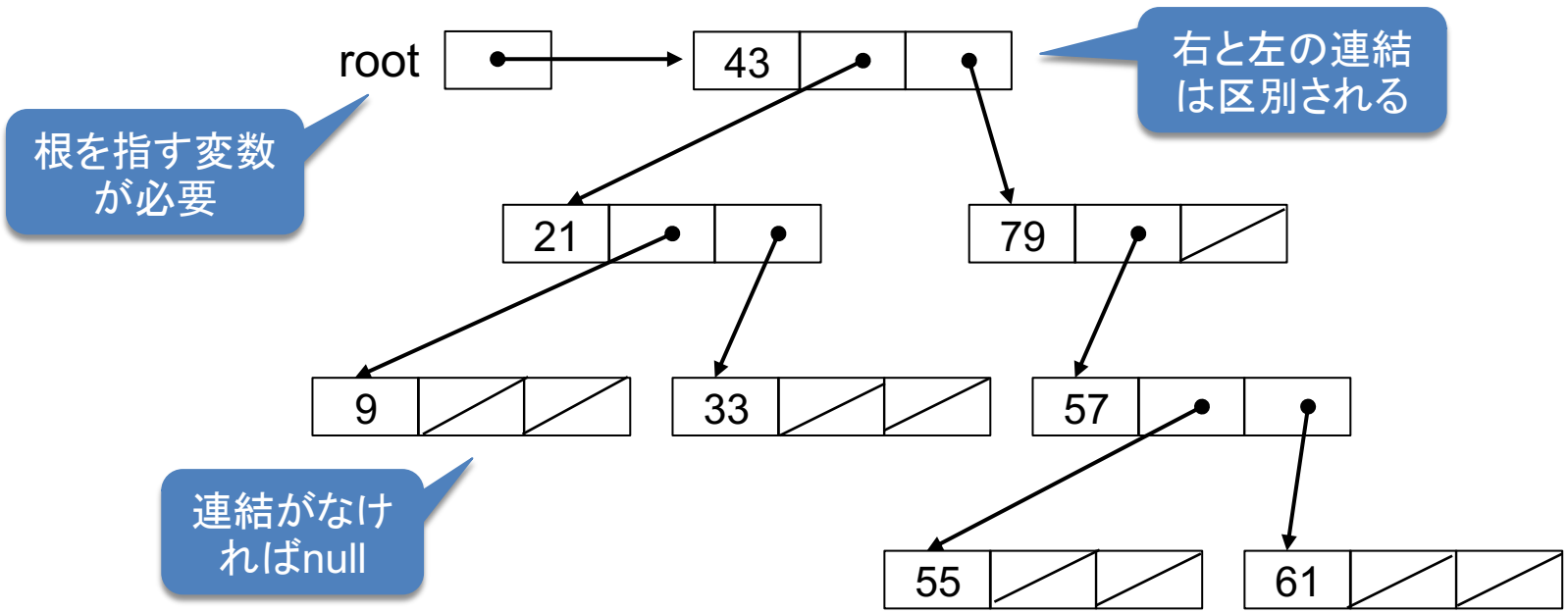


2分木の実装

- ノードの構造と定義


```
class Node {
    int data;
    Node left;
    Node right;
}
```

- 2分木の実装例



確認問題

- 2分木の構築
  - 前ページのノードを表すクラスに右のコンストラクタを追加したと想定して，前ページの図の2分木を構築する処理を示せ

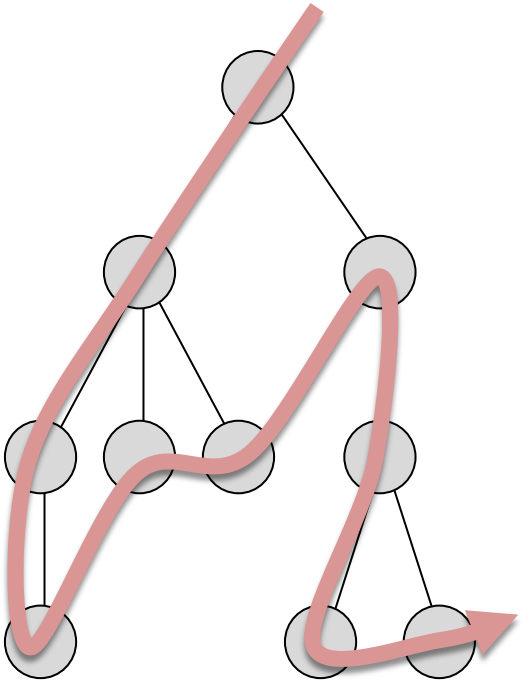
```
Node(int data) {
    this.data = data;
    left = right = null;
}
```

```
Node n1 = new Node(43);
Node n2 = new Node(21);
Node n3 = new Node(79);
Node n4 = new Node(9);
Node n5 = new Node(33);
Node n6 = new Node(57);
Node n7 = new Node(55);
Node n8 = new Node(61);
```

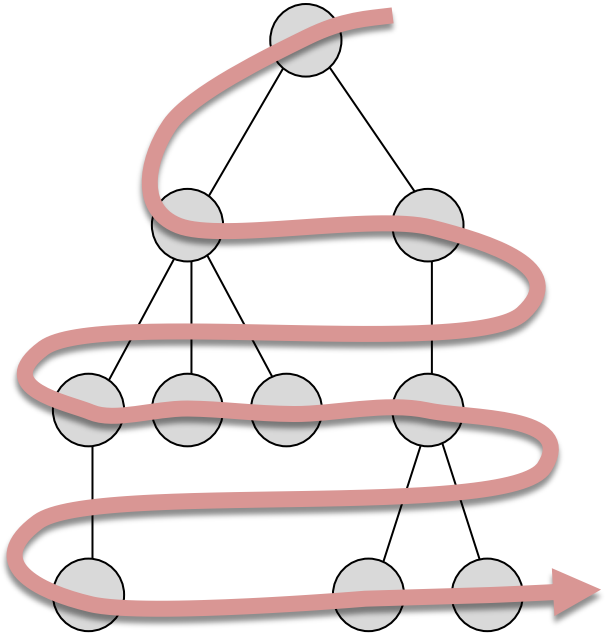
```
root = n1;
```

木の巡回

- 深さ優先探索
  - きょうだいより先に子をたどる
  - 再帰（内部でスタックを使用）を用いると，実現が簡単



- 幅優先探索
  - 子より先にきょうだいをたどる
  - キューの操作が必要なので，深さ優先よりも実現が難しい

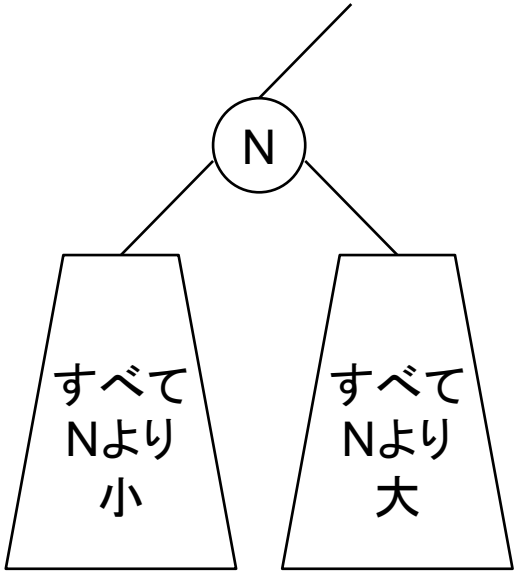


深さ優先探索

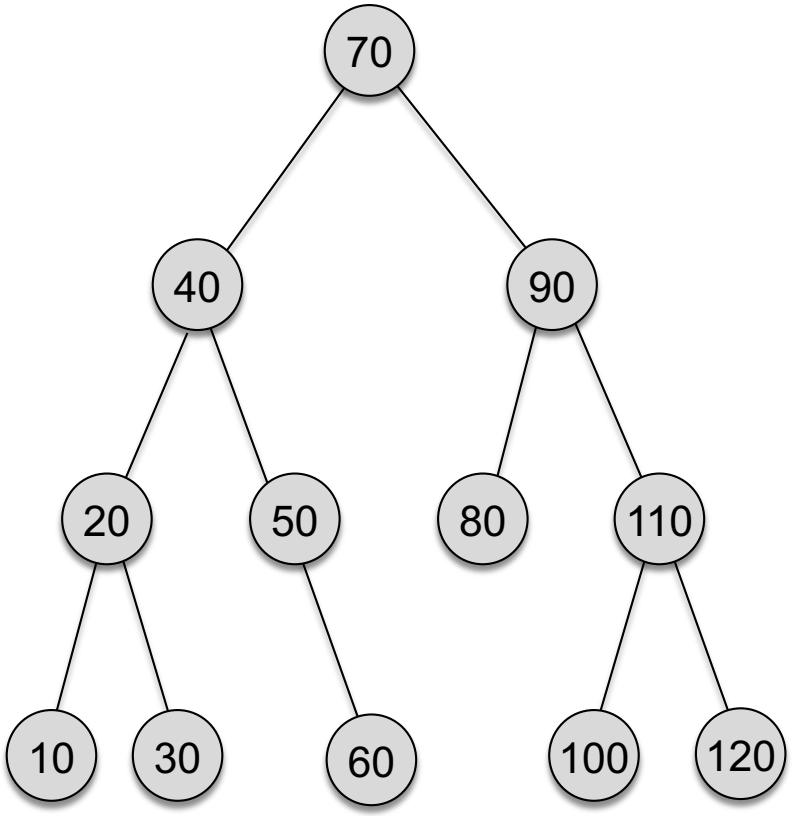
- 行きがけ順（先行順）
  - たどり着いたノードを最初に処理
  - ノードを処理 → 左の部分木を処理 → 右の部分木を処理
- 通りがけ順（中間順）
  - たどりついたノードを中間で処理
  - 左の部分木を処理 → ノードを処理 → 右の部分木を処理
  - 2分探索木で，要素が整列されて表示される順序
- 帰りがけ順（後行順）
  - たどり着いたノードを最後に処理
  - 左の部分木を処理 → 右の部分木を処理 → ノードを処理

2分探索木

- 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

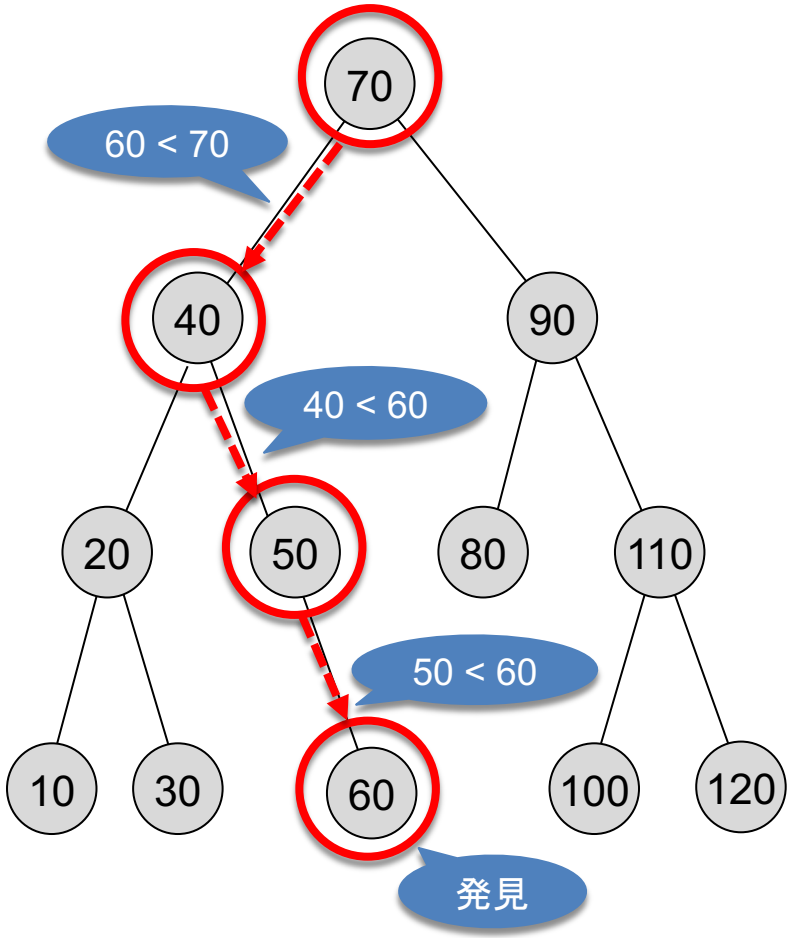


- 例



要素の探索と追加

- 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（次回）
  - 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



確認問題

- 2分探索木の処理
  - 前ページの2分探索木から「35」を探索する場合，どのような順番でノードと比較を行い，結果はどうなるか示せ
  - 前ページの2分探索木に「45」を追加する場合，その処理が行われた後の木を図示せよ

- 2分探索木の条件
  - 右図の2分木は，2分探索木の条件を満たすか調べよ
  - この2分木は，どの2つの要素を交換すれば2分探索木の条件を満たすようになるか示せ