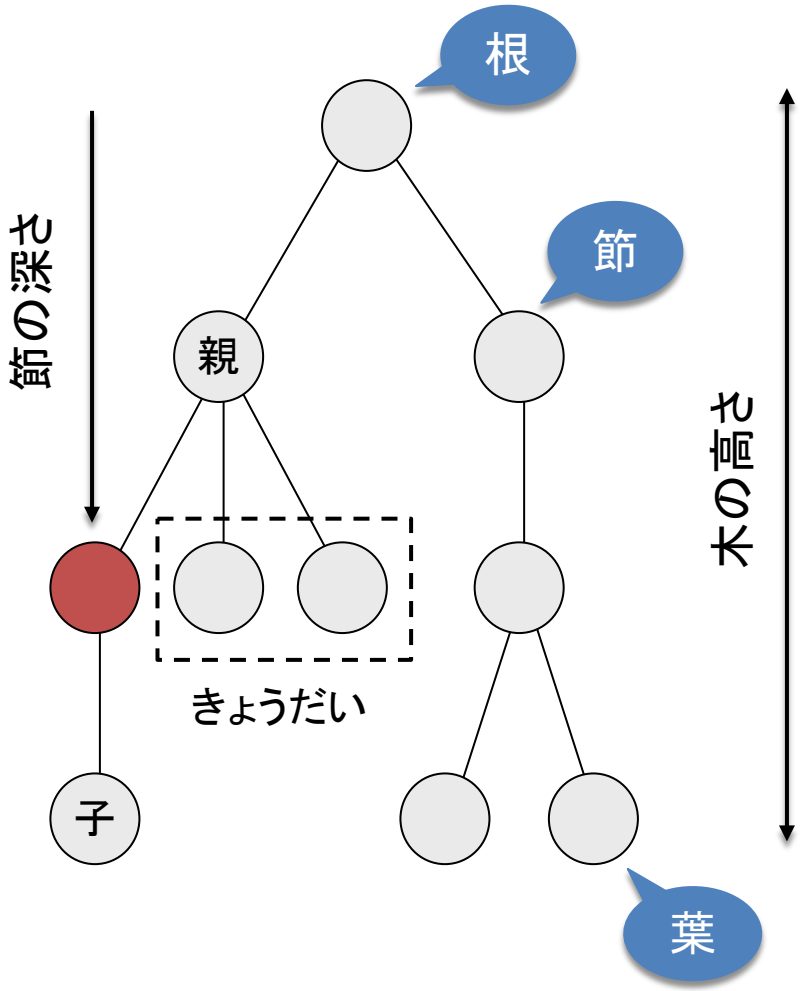
第12回のキーワード

アルゴリズム関係

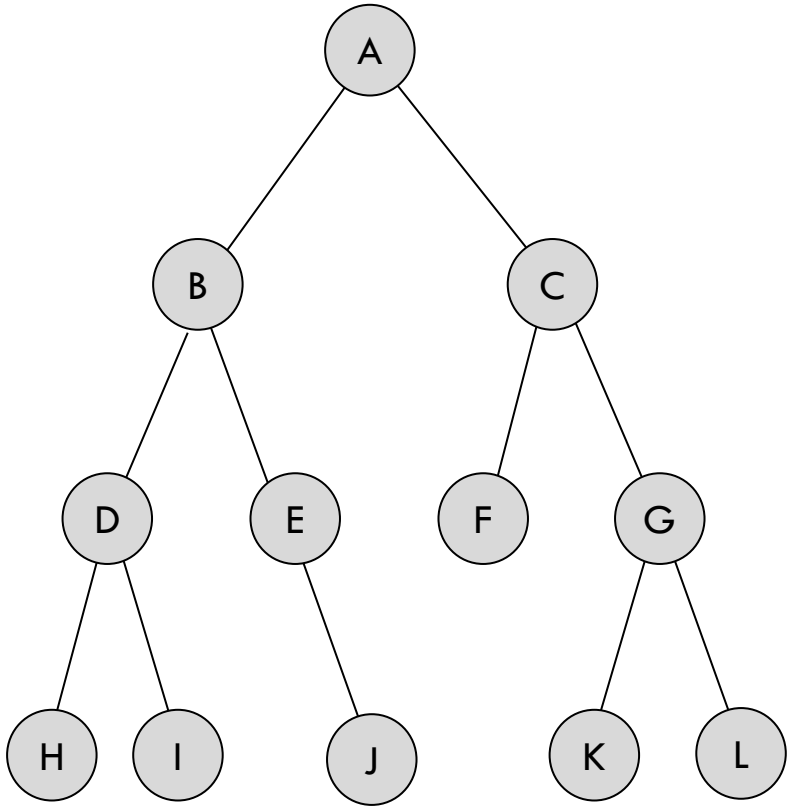
- 木構造（tree structure）
- 木，根，節，葉
  （tree, root, node, leaf）
- 部分木（subtree）
- 親，子，きょうだい
  （parent, children, sibling）
- 木の高さ（height）
- 節の深さ（depth）
- 2分木 / 2進木
  （binary tree）
- 多分木 / N分木
  （multi-branch / n-ary tree）
- 木の巡回（走査）
  （tree traversal）
- 深さ優先探索
  （depth first search）
- 行きがけ順 / 先行順
  （preorder）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

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

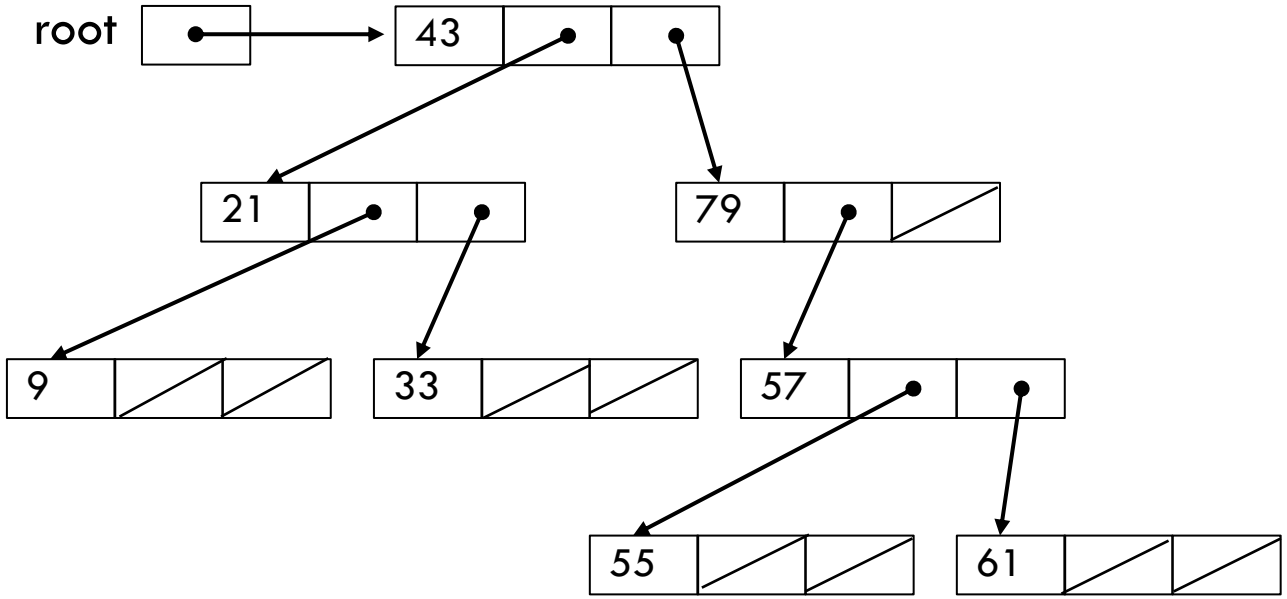


2分木の実装

- ノードの構造と定義

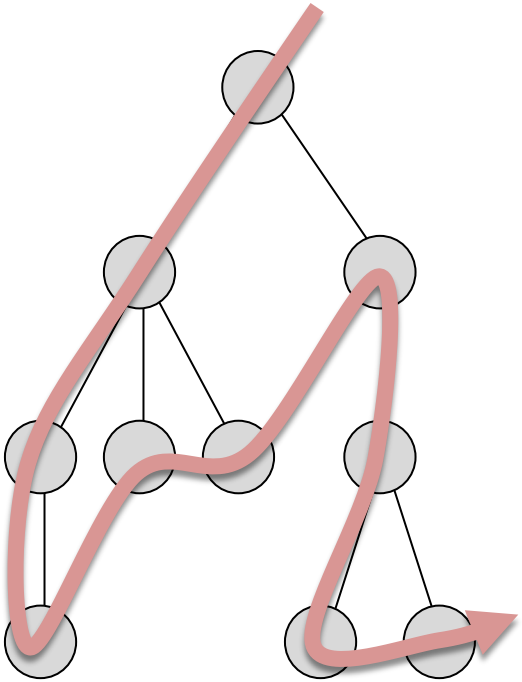

```
class Node {
    String data;
    Node left;
    Node right;
}
```

- 2分木の実装例

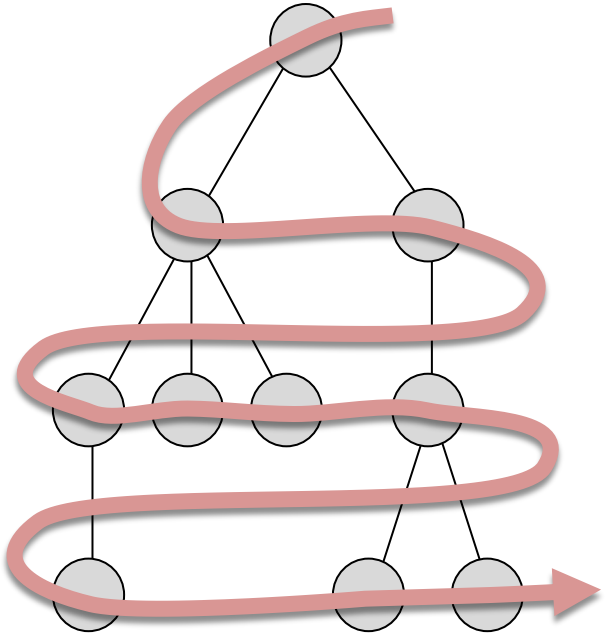


木の巡回

- 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

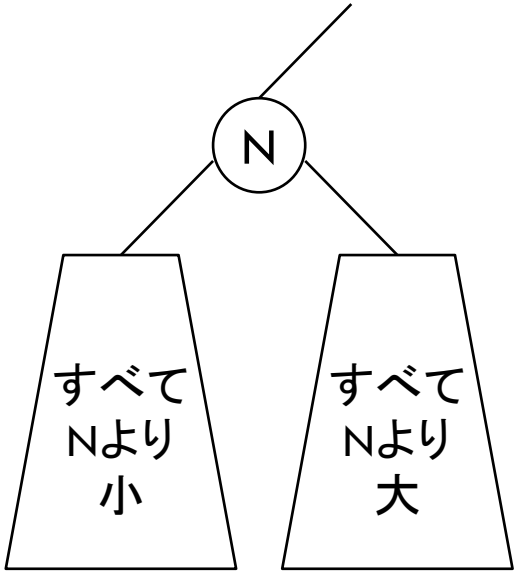


- 幅優先探索
  - 子より先にきょうだいをたどる
  - キューの操作が必要なので，深さ優先よりも実現が難しい

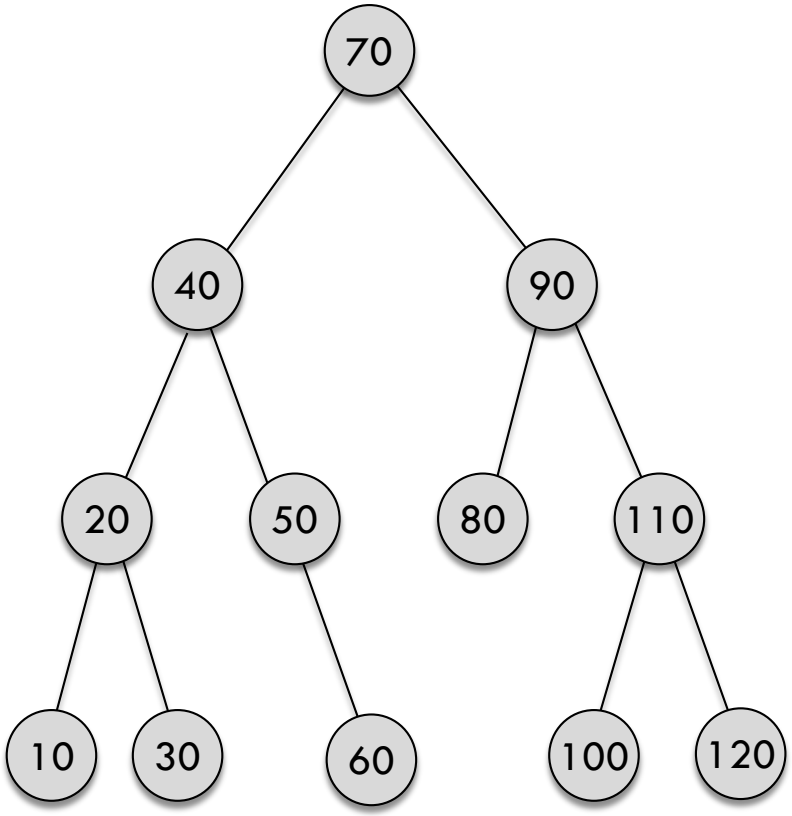


2分探索木

- 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
  - 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