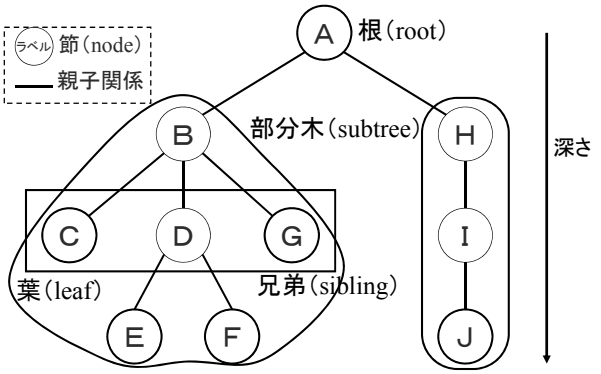
データ構造（2）

木構造

木構造とは

- 階層関係を表現するためのデータ構造
  - 要素間に上下関係がある
  - 身近な例：会社組織や家系、本の構成
  - コンピュータ関連：OSのファイル管理、数式、プログラムなど
- さまざまなアルゴリズムの実現に利用される

木（tree）を構成する要素



木の定義

- 以下の二つの規則により再帰的に定義
  - 一つの節は、それ自身が木である。この木に含まれるただ一つの節が、この木の根である
  - k個の木T1～Tkがあり、それぞれの根を節n1～nkとする。新しく節nを用意し節n1～nkの親とすると、節nを根とする新しい木Tが得られる。この時、木T1～Tkは、木Tの部分木であるという。部分木の根n1～nkは、節nの子であるという
- 一つも節をもたないものを空の木と呼ぶ

順序木と無順序木

- 順序木：兄弟の間に順番づけを行ったもの

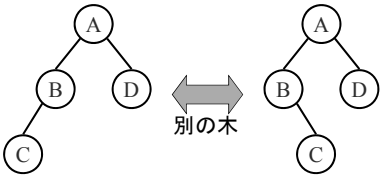
- 木をデータ構造として実現する時には、節をなんらかの順番に並べることになる
  ＝順序木

二分木（binary tree）

- 二分木の定義
  - 空の木は二分木である
  - 次のいずれかを満たす節のみからなる木は二分木である
    - 子をもたない
    - 左の子のみをもつ
    - 右の子のみをもつ
    - 左右二つの子をもつ

二分木の特徴

- 二分木を構成する節はたかだか二つの子しかもたない
- 節が一つの子をもっているとき、左の子か右の子かを区別する

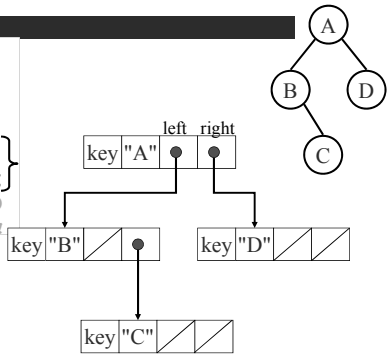


二分木を表現する構造体の例

```
struct Node {
  int key;
  char *data;
  struct Node *left;
  struct Node *right;
};
```

ラベル

左右の子へのポインタ



木のなぞり(traverse)

- 木の節を系統的に一つ残らず調べ、かつ、各節には一回しか立ち寄らないようにすること
- 立ち寄った順に節を順序づけることができる
- 3通りの方法
  - 行きがけ順(preorder)、通りがけ順(inorder)、帰りがけ順(postorder)

二分木をなぞる手順

- 空の木なら何もしない
- 空でなければ、

(行きがけ順)

1. 根に立ち寄る、2. 左部分木をなぞる、3. 右部分木をなぞる

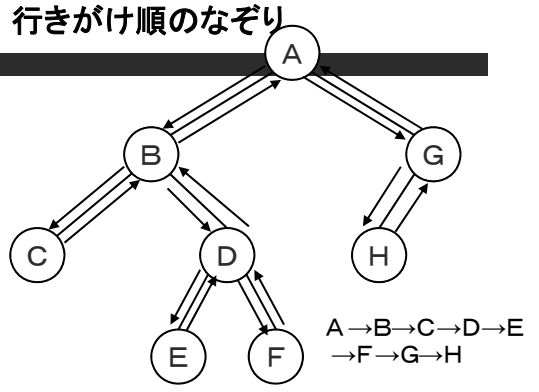
(通りがけ順)

1. 左部分木をなぞる、2. 根に立ち寄る、3. 右部分木をなぞる

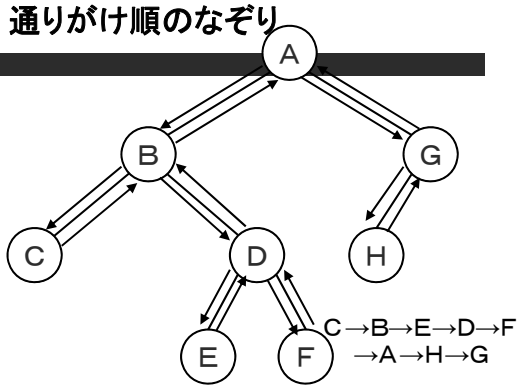
(帰りがけ順)

1. 左部分木をなぞる、2. 右部分木をなぞる、3. 根に立ち寄る

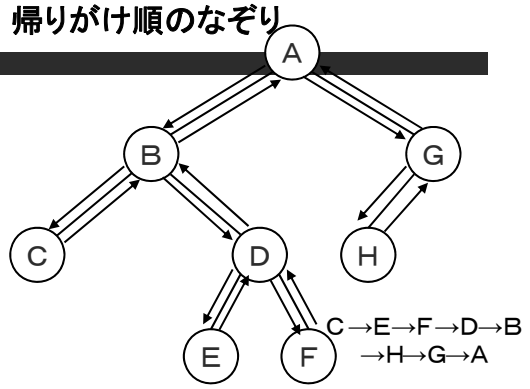
行きがけ順のなぞり



通りがけ順のなぞり

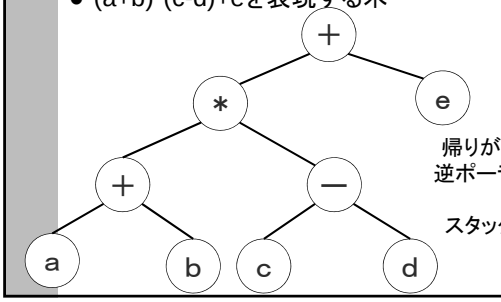


帰りがけ順のなぞり



数式の木

- (a+b)*(c-d)+eを表現する木



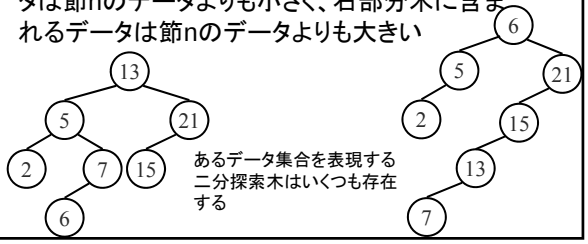
帰りがけ順でなぞると
逆ポーランド記法となる
↓
スタックを使った演算
が可能

木構造を使った探索

- 根から葉へ向かってたどる経路が探索プロセスに相当する
- 探索経路の長さは木の枝ぶりに依存する
- 探索に適した木構造
  - 二分探索木、AVL木、B木、など

二分探索木（binary search tree）

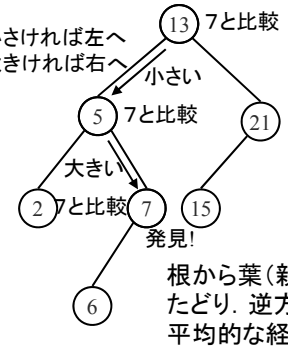
- 節に大小比較可能なデータをもたせた二分木
- 任意の節nについて、左部分木に含まれるデータは節nのデータよりも小さく、右部分木に含まれるデータは節nのデータよりも大きい



あるデータ集合を表現する
二分探索木はいくつも存在
する

二分探索木の探索

根からスタート
・探索キーが小さければ左へ
・探索キーが大きければ右へ



7を探索

13→5→7

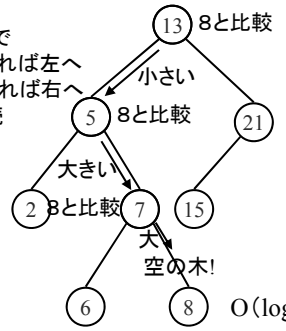
根から葉（親から子）へ向かうたどり．逆方向の戻りは無い．
平均的な経路長はO(log n)

二分探索木の探索手順

以下を探索失敗するまで繰り返す{
  木が空であれば探索失敗とする
  根のデータxと探索キーkを比較
  k==xならばその根を探索結果として探索操作終了
  k<xならば左部分木に対象を移す
  k>xならば右部分木に対象を移す
}
探索失敗

二分探索木への挿入

根からスタート,
空の木が見つかるまで
・挿入データが小さければ左へ
・挿入データが大きければ右へ
新しい葉を作って接続



O(log n)の操作

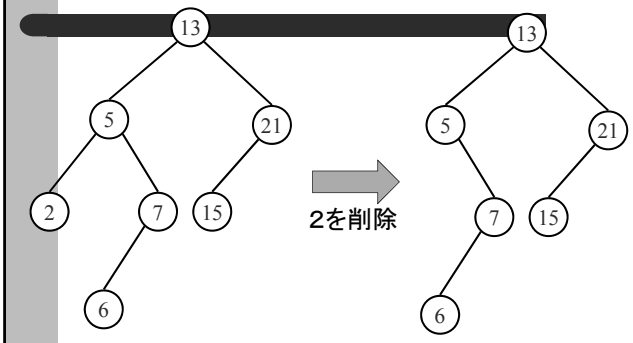
二分探索木への挿入手順

挿入すべき場所が見つかるまで繰り返す{
木が空であれば、そこが挿入すべき場所
根のデータxと挿入データkを比較
k==xならば既に登録済みとして挿入操作を中断
k<xならば左部分木の根に対象を移す
k>xならば右部分木の根に対象を移す
}
新しい葉を作って接続する

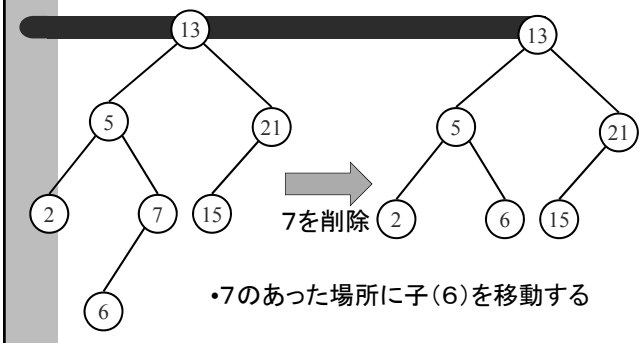
二分探索木からの削除

- 削除する節を探索する
- 節の削除処理
  - 子を持たない節(葉)の場合
    - その節を削除する
  - 子を一つ持つ節の場合
    - その節を削除し、節があった場所に子を移動する
  - 子を二つ持つ節の場合
    - その節を削除し、節があった場所に右部分木の最小要素を取り出して置く

子をもたない節の削除

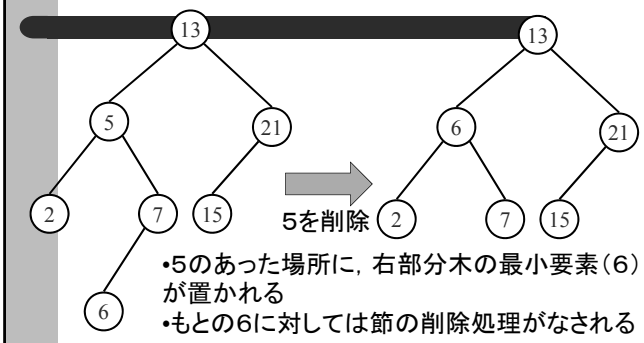


子を一つもつ節の削除



•7のあった場所に子(6)を移動する

子を二つもつ節の削除



•5のあった場所に，右部分木の最小要素(6)が置かれる
•もとの6に対しては節の削除処理がなされる

二分探索木からの削除手順

削除対象の探索に失敗するまで繰り返す{
　木が空であれば探索失敗
　根のデータxと探索キーkを比較
　k==xならば{
　　節の削除処理（場合分けが必要）をして操作完了
　}
　k<xならば左部分木の根に対象を移す
　k>xならば右部分木の根に対象を移す
}

次回予定

- 木構造を利用したプログラミング演習を行う
  - データの挿入，削除，並べかえ

http://www.sys.wakayama-u.ac.jp/~manda/alg/