プログラミングコンテスト攻略のための

アルゴリズムとデータ構造

第8章 木

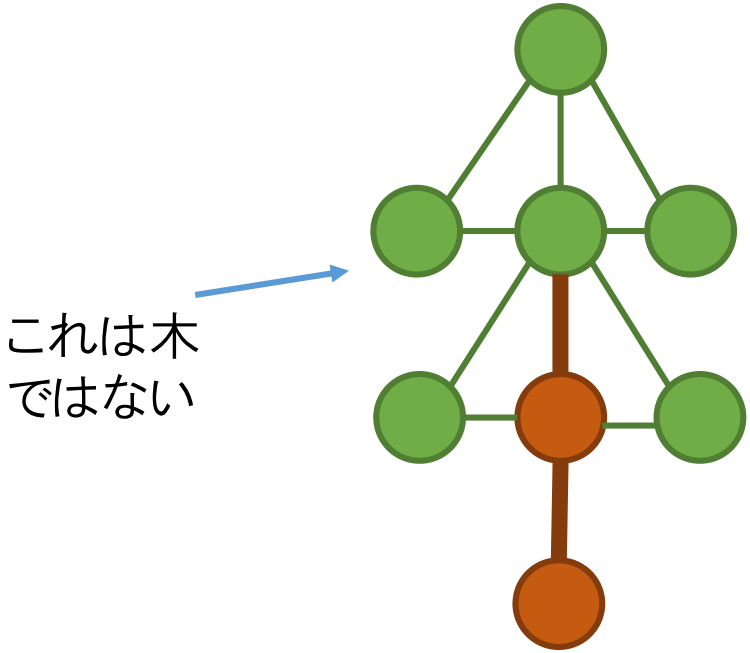
2017/06/16

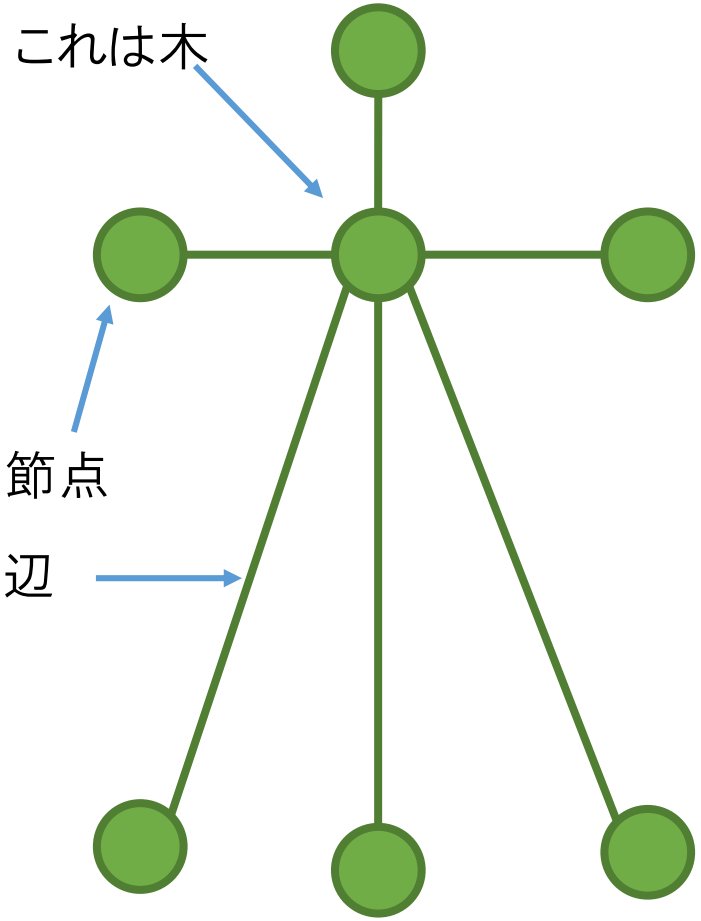
秋山研究室 修士2年 後藤公太

木とは

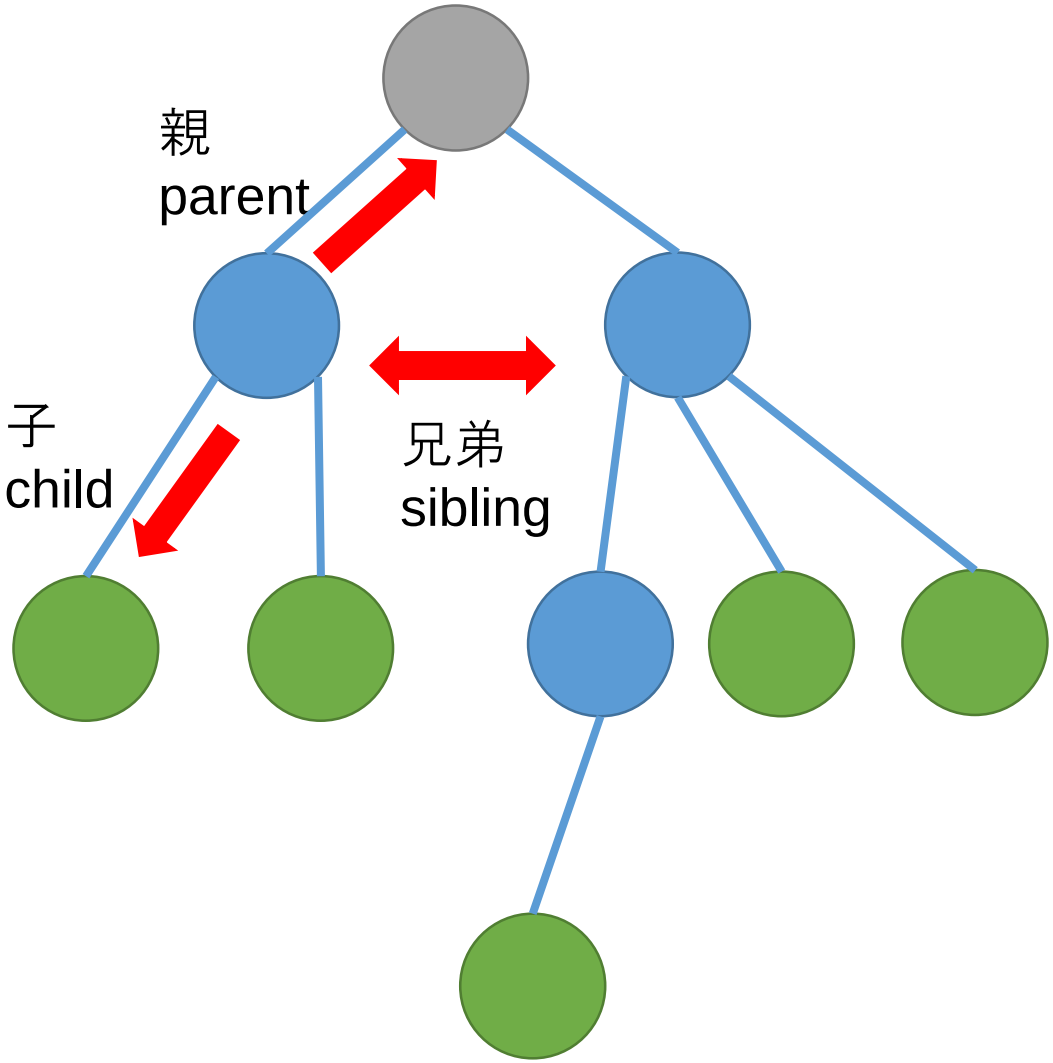
木とは、**節点（node）**とそれを結ぶ
辺（edge）で表されるデータ構造
木は閉路を持たない

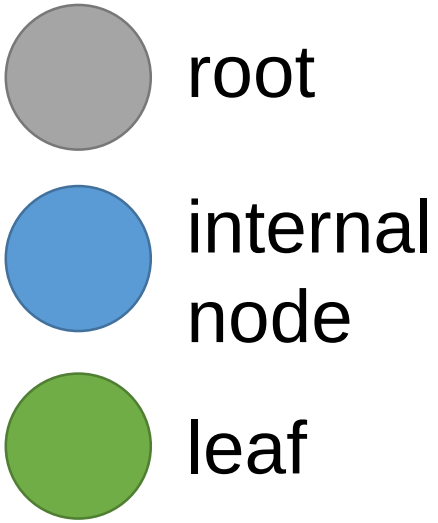
根（root）と呼ばれる
特別な節点を持つ木を
根付き木（rooted tree）という



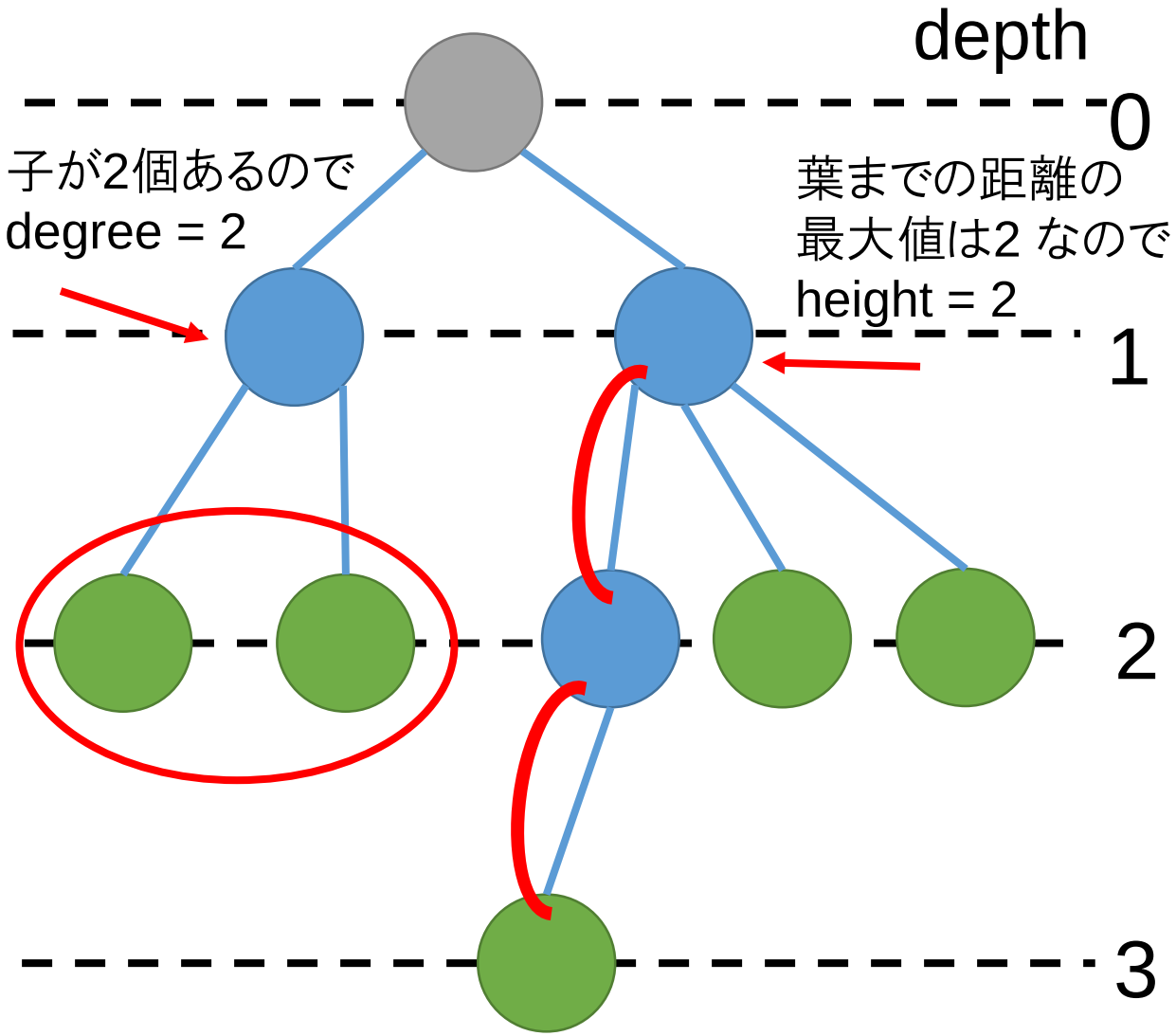


木とは





木とは



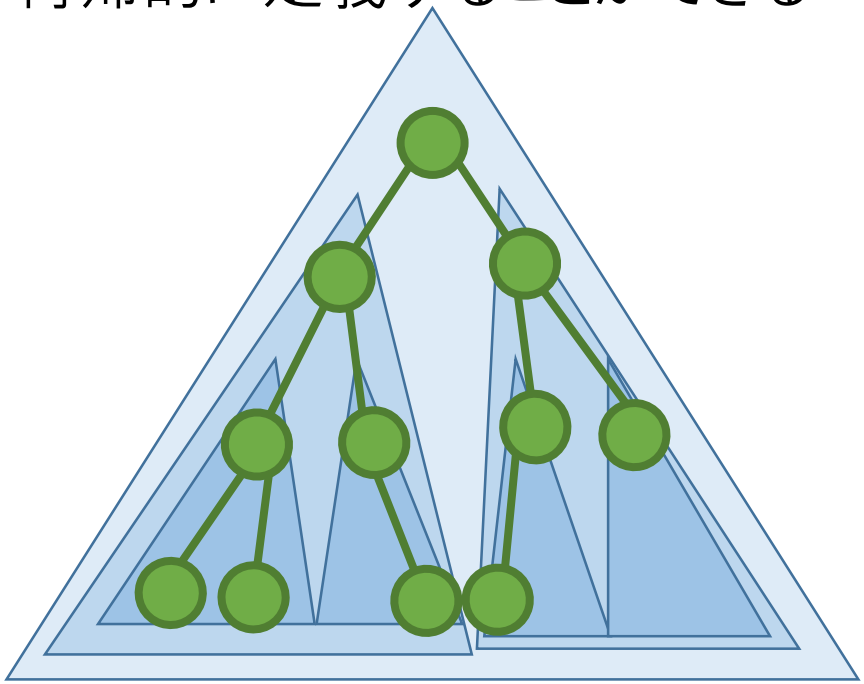
- depth
  根からの距離
- degree
  子の数
- height
  自分の子孫の
  葉までの距離の
  最大値

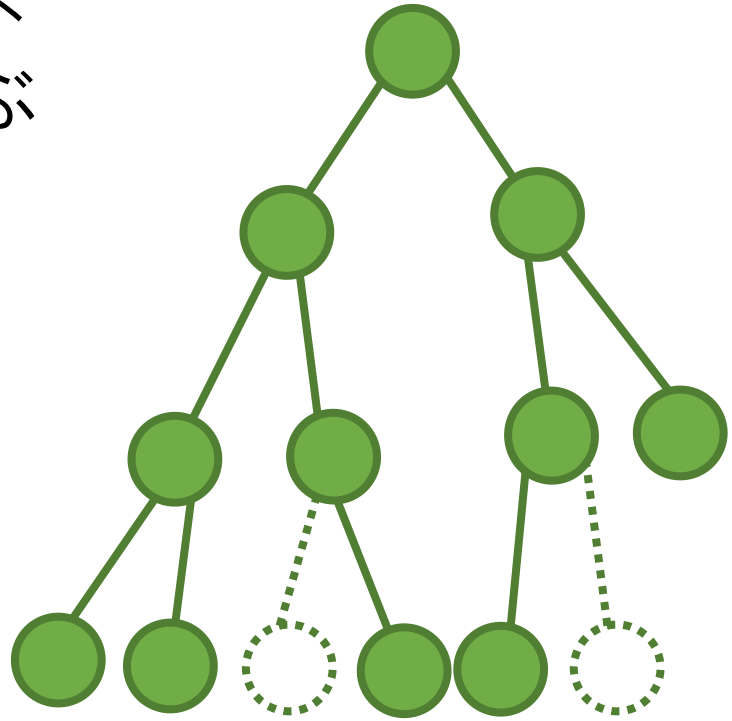
二分木

- 1つの根を持つ
- 全ての節点で子の数が2以下

である木を**根付き二分木**と呼ぶ

再帰的に定義することができる





子の左右は
区別されることに注意！

問題:二分木の表現

ALDS1_7_B: Binary Tree

問題: 与えられた根付き二分木 Tの各節点u について、
以下の情報を出力するプログラムを作成して下さい。

- uの節点番号
- uの親
- uの兄弟
- uの子の数
- uの深さ
- uの高さ
- 節点の種類
  (根 or 内部節点 or 葉)

入力: 入力の最初の行に、節点の個数nが与えられます。続くn行に各接点の情報が以下の形式で1行に与えられます。節点はそれぞれ0～n-1の番号が割り当てられています。
n≦25

id left right

id:節点の番号 *left*:左の子の番号 *right*:右の子の番号
個がない場合には-1が与えられます。

問題:二分木の表現

出力： 次の形式で節点の情報を出力して下さい。

node␣*id*:␣parent␣=␣*p*,␣sibling␣=␣*s*,␣degree␣=␣*deg*,␣depth␣=␣*dep*,␣height␣=␣*h*,␣*type*

- *id*
  その節点のid
- *p*
  親の番号
  親が存在しなければ -1
- *s*
  兄弟の番号
  兄弟が存在しなければ-1

- *deg, dep, h*
  子の数, 深さ, 高さ
- *type*

root	根
internal node	内部節点
leaf	葉

問題:二分木の表現

入力例

```
9
0 1 4
1 2 3
2 -1 -1
3 -1 -1
4 5 8
5 6 7
6 -1 -1
7 -1 -1
8 -1 -1
```

出力例

```
node 0: parent = -1, sibling = -1, degree = 2, depth = 0, height = 3, root
node 1: parent = 0, sibling = 4, degree = 2, depth = 1, height = 1, internal node
node 2: parent = 1, sibling = 3, degree = 0, depth = 2, height = 0, leaf
node 3: parent = 1, sibling = 2, degree = 0, depth = 2, height = 0, leaf
node 4: parent = 0, sibling = 1, degree = 2, depth = 1, height = 2, internal node
node 5: parent = 4, sibling = 8, degree = 2, depth = 2, height = 1, internal node
node 6: parent = 5, sibling = 7, degree = 0, depth = 3, height = 0, leaf
node 7: parent = 5, sibling = 6, degree = 0, depth = 3, height = 0, leaf
node 8: parent = 4, sibling = 5, degree = 0, depth = 2, height = 0, leaf
```

- Depth, Heightは先に計算して保存すると楽
- node 0がrootとは限らないことに注意

解説：二分木の表現

データ構造

```
typedef struct Node{
        int parent, left, right;
}Node;  //ノードは親、子のノード番号を持つ
```

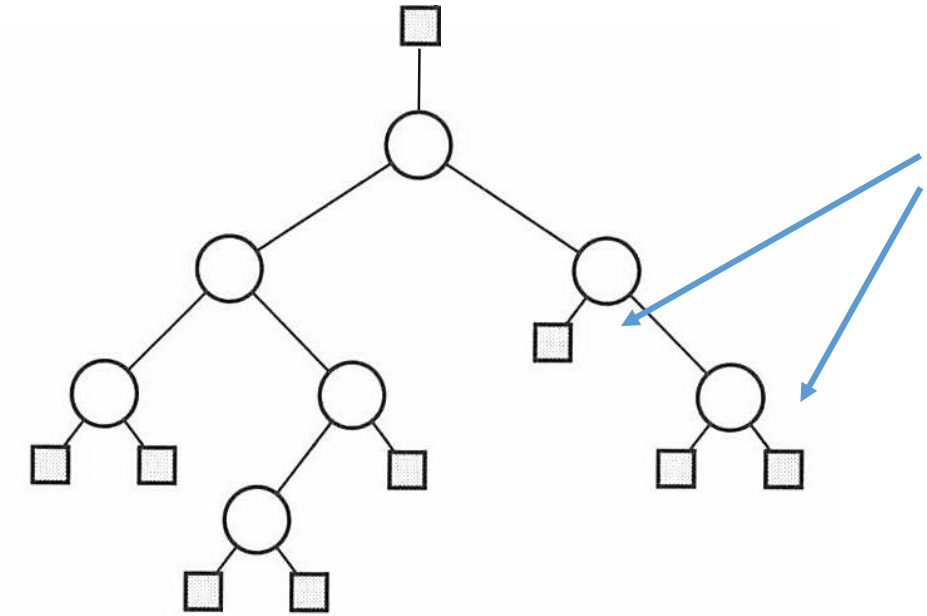


図8.6: 二分木の番兵

“番兵”として
使わない値（-1）を入れておく

子の番号が-1ならば
子を持たないということ

親の番号が-1ならば
その節点が根(root)

解説:二分木の表現

データ構造

```
typedef struct Node{
        int parent, left, right;
}Node;  //ノードは親、子のノード番号を持つ
```

```
Node tree[MAX_N];
init(tree) // tree[i].parent=-1に初期化する
for(i=0;i<n;i++){
    scanf("%d %d %d",&node,&left,&right);
    tree[node].left=left;
    tree[node].right=right;
    if(left!=-1){    tree[left].parent=node;  }
    if(right!=-1){  tree[right].parent=node;  }
    //子がいれば、その子からはparentとなる
}
```

解説：二分木の表現

再帰によるDepthとHeightの計算

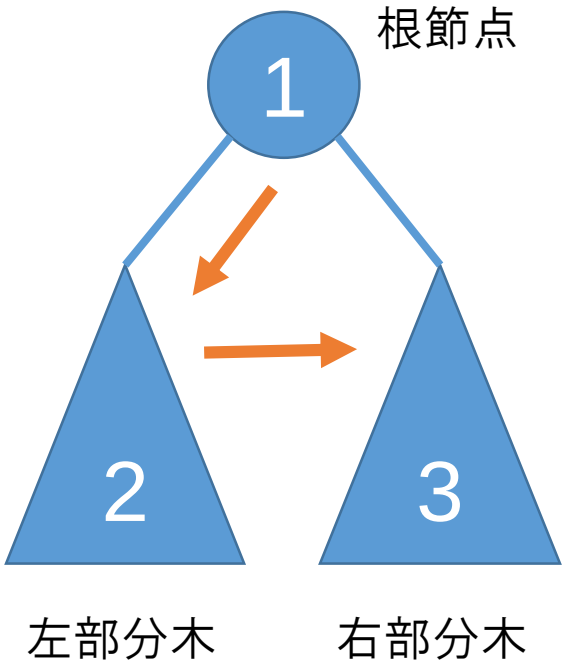
```
void setDepth(Node tree[MAX_N], int depth_list[MAX_N], int node, int depth){
    if(node==-1){  return;  }
    depth_list[node]=depth;
    setDepth(tree,depth_list,tree[node].left,depth+1);  //rootからの距離なので子は+1
    setDepth(tree,depth_list,tree[node].right,depth+1);
}
```

```
int setHeight(Node tree[MAX_N], int height_list[MAX_N], int node){
    int h_left,h_right;
    if(node==-1){ return -1; } //子が無いノードは高さ0 (-1+1)
    h_left=setHeight(tree,height_list,tree[node].left)+1;      //左側の子のHeight+1
    h_right=setHeight(tree,height_list,tree[node].right)+1;  //右側の子のHeight+1
    if(h_left>h_right){ height_list[node]=h_left;          //左と右で高い方が
    }else{                    height_list[node]=h_right;   }   //そのノードのHeight
    return height_list[node];
}
```

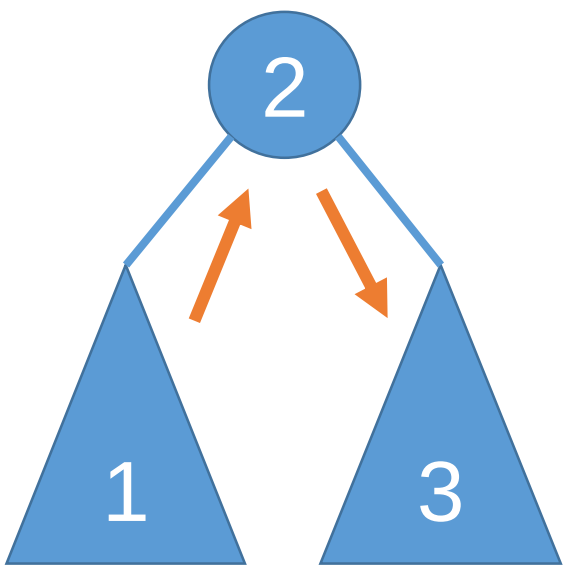
問題:木の巡回

巡回方法

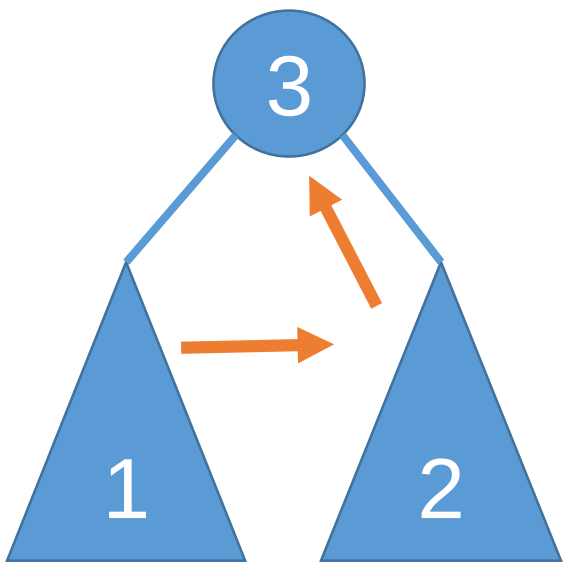
Preorder
（先行順巡回）



Inorder
（中間順巡回）



Postorder
（後行順巡回）



問題:木の巡回

ALDS1_7_C: Tree Walk

問題: 次に指定する方法で、与えられた二分木の全ての節点を訪問するプログラムを作成して下さい。

- Preorder Tree Walk
  根節点、左部分木、右部分木の順で出力する
- Inorder Tree Walk
  左部分木、根節点、右部分木の順で出力する
- Postorder Tree Walk
  左部分木、右部分木、根節点、の順で出力する

問題:木の巡回

入力: 入力の最初の行に、節点の個数nが与えられます。続くn行に各接点の情報が以下の形式で1行に与えられます。節点はそれぞれ0～n-1の番号が割り当てられています。
n≦25

id left right

id:節点の番号 *left*:左の子の番号 *right*:右の子の番号
個がない場合には-1が与えられます。

出力: 1行目に”Preorder”と出力し、2行目に節点番号を順番に出力
3行目に”Inorder”と出力し、4行目に節点番号を順番に出力
5行目に”Postorder”と出力し、6行目に節点番号を順番に出力
節点番号の前に空白を1文字入れる

問題:木の巡回

入力例

```
9
0 1 4
1 2 3
2 -1 -1
3 -1 -1
4 5 8
5 6 7
6 -1 -1
7 -1 -1
8 -1 -1
```

出力例

```
Preorder
 0 1 2 3 4 5 6 7 8
Inorder
 2 1 3 0 6 5 7 4 8
Postorder
 2 3 1 6 7 5 8 4 0
```

空白あり!

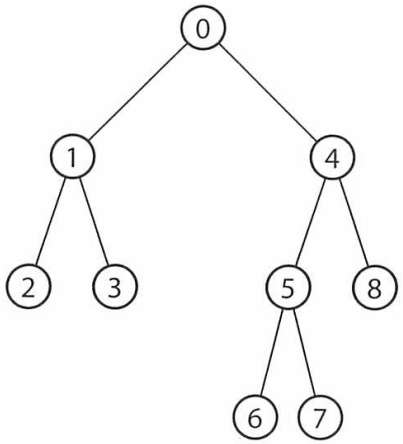


図8.7: 入力例

解説：木の巡回

巡回方法の定義通りに再帰で書く
データ構造は同じ

Preorder
（先行順巡回）

```
void preorder
(Node tree[MAX_N],
  int node){

if(node==-1){return;}
 printf(" %d",node);
 preorder(tree,
              tree[node].left);
 preorder(tree,
              tree[node].right);
}
```

Inorder
（中間順巡回）

```
void inorder
(Node tree[MAX_N],
  int node){

if(node==-1){return;}
 inorder(tree,
              tree[node].left);
 printf(" %d",node);
 inorder(tree,
              tree[node].right);
}
```

Postorder
（後行順巡回）

```
void postorder
(Node tree[MAX_N],
  int node){

if(node==-1){return;}
 postorder(tree,
              tree[node].left);
 postorder(tree,
              tree[node].right);
 printf(" %d",node);
}
```