# アルゴリズム論
# （第4回）


岩手県立大学
Iwate Prefectural University

佐々木研（情報システム構築学講座）
講師　山田敬三
k-yamada@iwate-pu.ac.jp

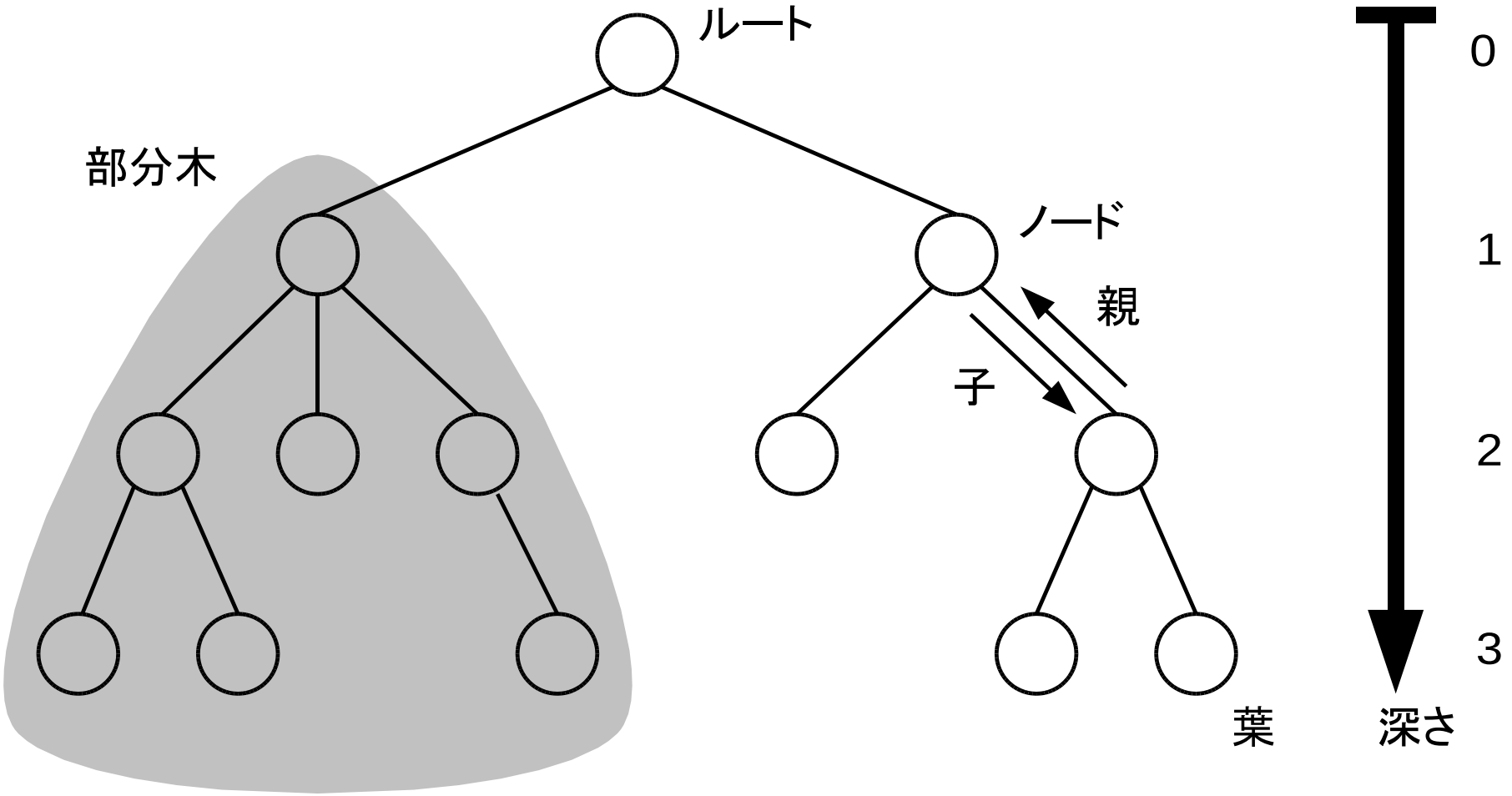
# 木構造

岩手県立大学
Iwate Prefectural University

# 木構造

- Tree Structure
- 階層構造を持つデータを扱う.
  - 章や節などの構造をもつ本
  - ディレクトリ，など
- 複数のノードからなるデータ構造

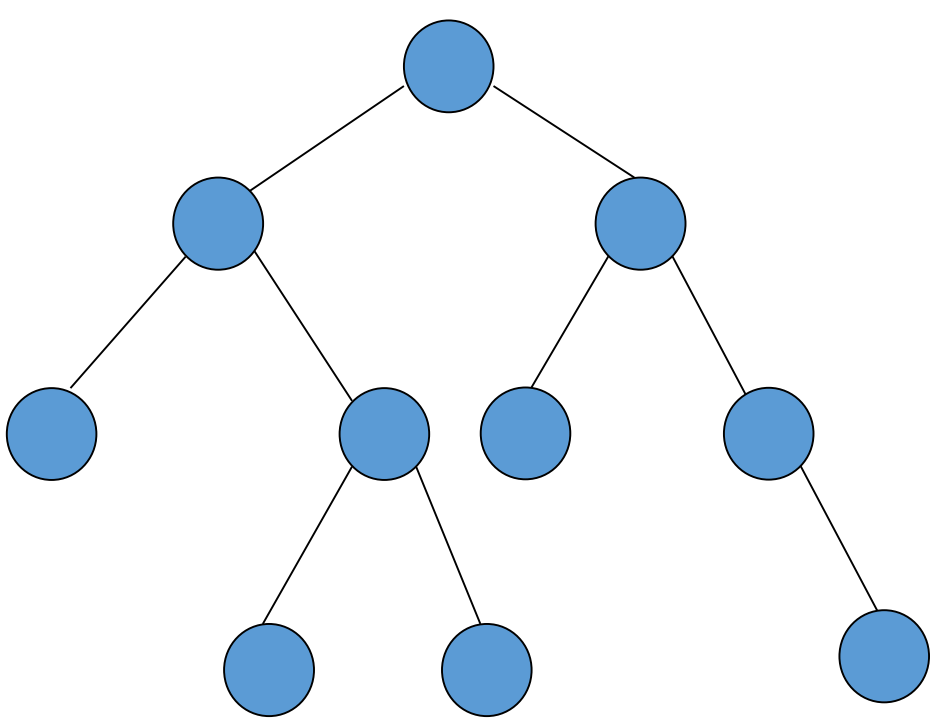
# 木構造



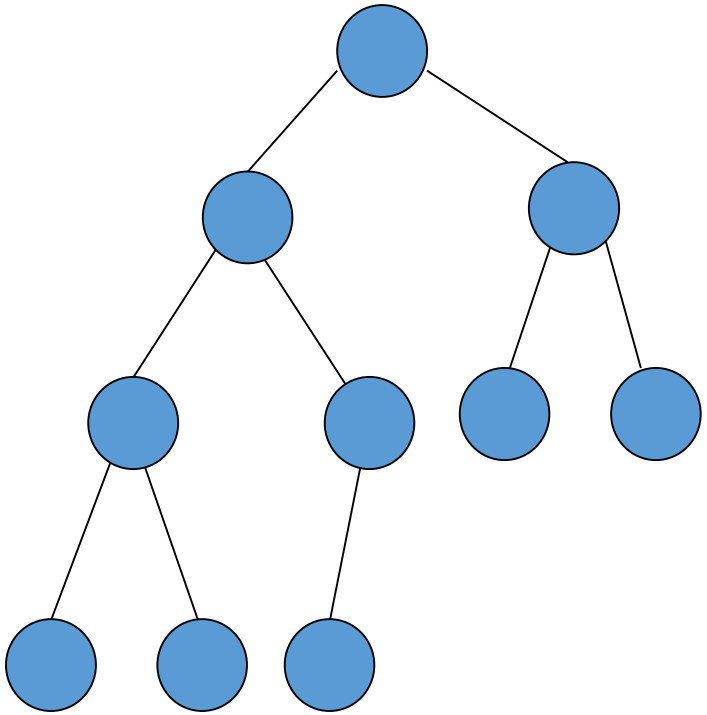
# 木構造

- 二分木（Binary Tree）
  - 各ノードで枝分かれ
  - 子が2つ以下の木構造
- 完全二分木（Complete Binary Tree）
  - 詰められた二分木
    - ルートから，上から下へ
    - 同レベルでは左から右へ

# 二分木と完全木分木

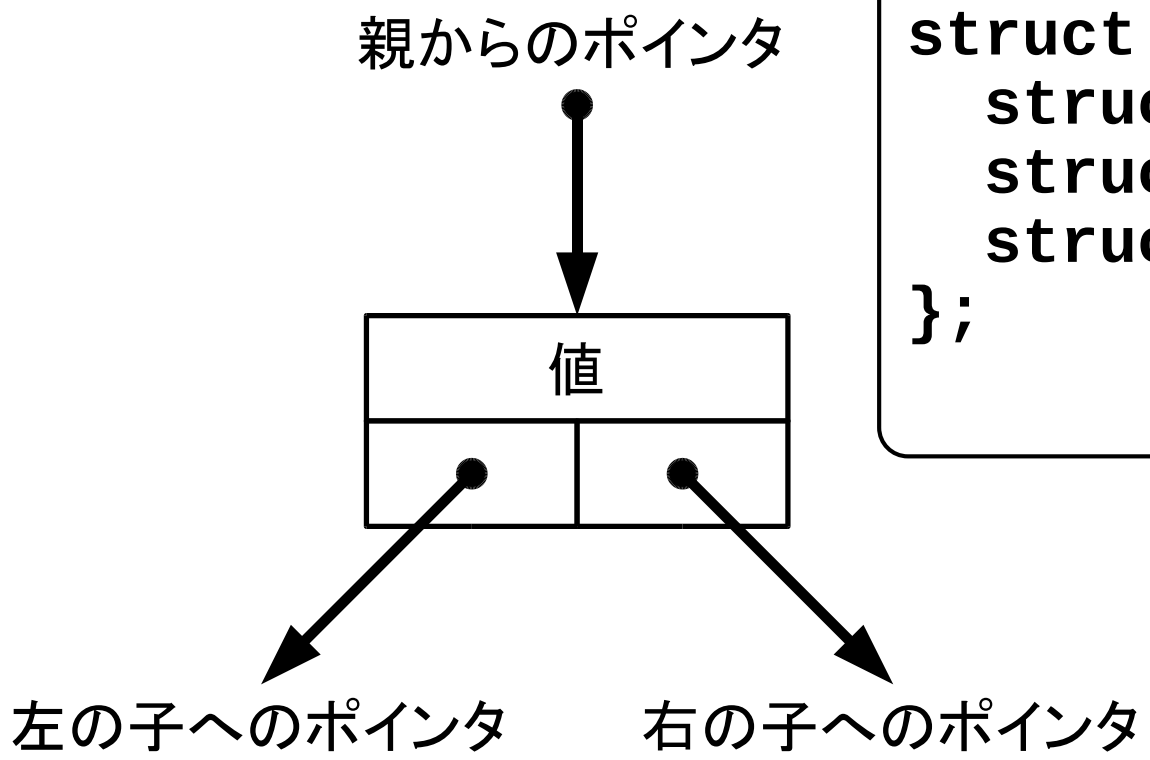


(a)二分木



(b)完全二分木

# 二分木の実装

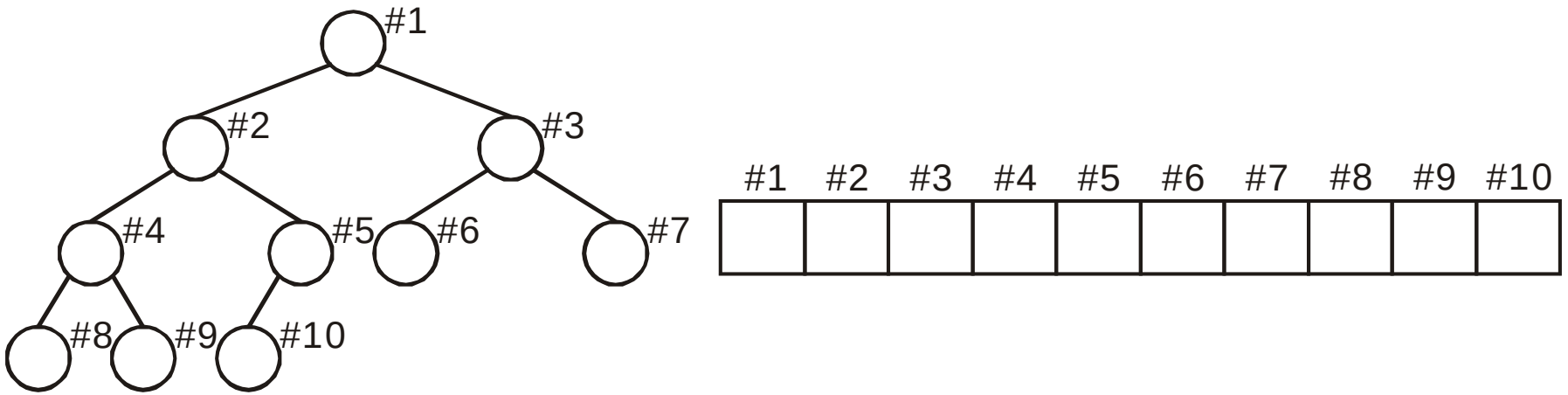



```
struct person {
  char *name;
  int year;
};

struct node {
  struct person *value;
  struct node *child_l;
  struct node *child_r;
};
```

# 完全二分木の場合

- 配列と簡単な計算式で実現可能
- あるノード **n** に対して
  - 親ノードは **n/2**
  - 子ノードは **2n** および **2n+1**



# 二分探索木

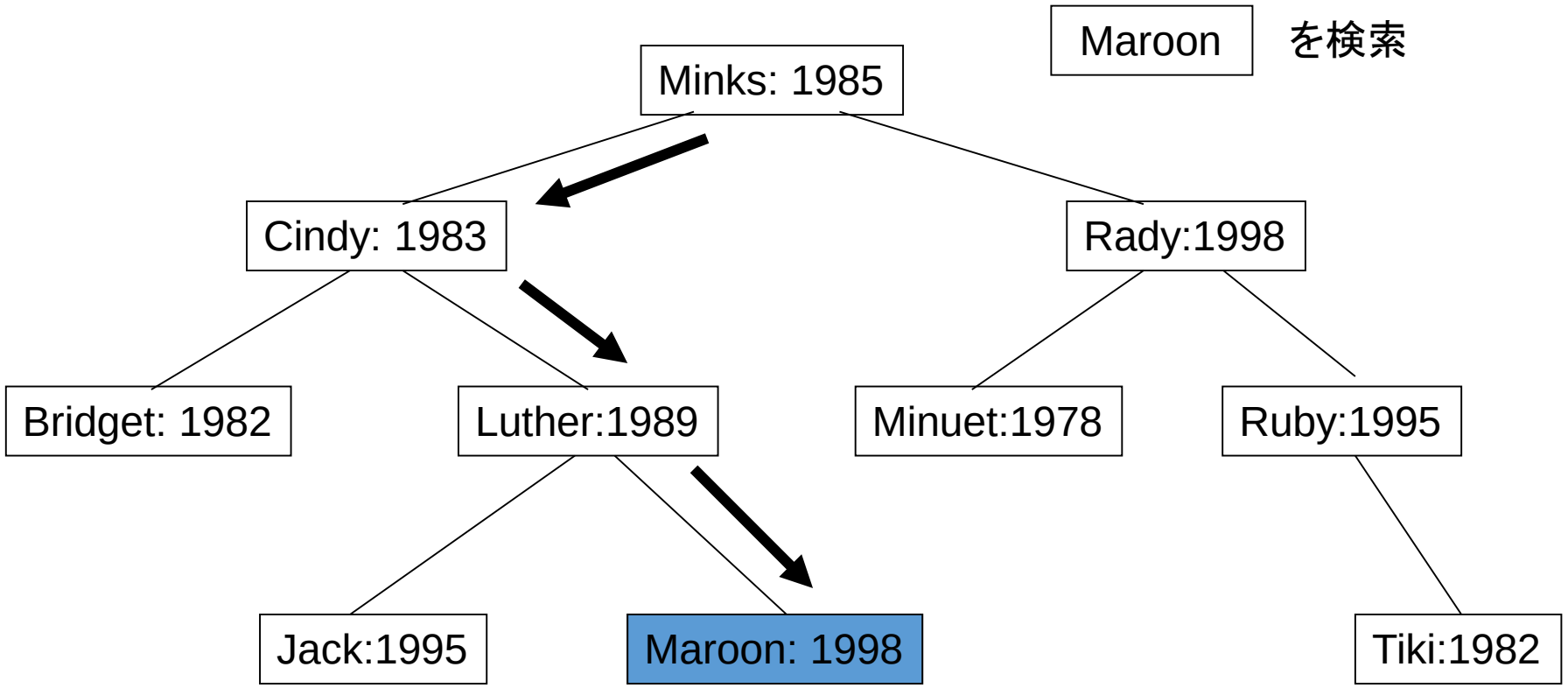

岩手県立大学
Iwate Prefectural University

# 二分探索木

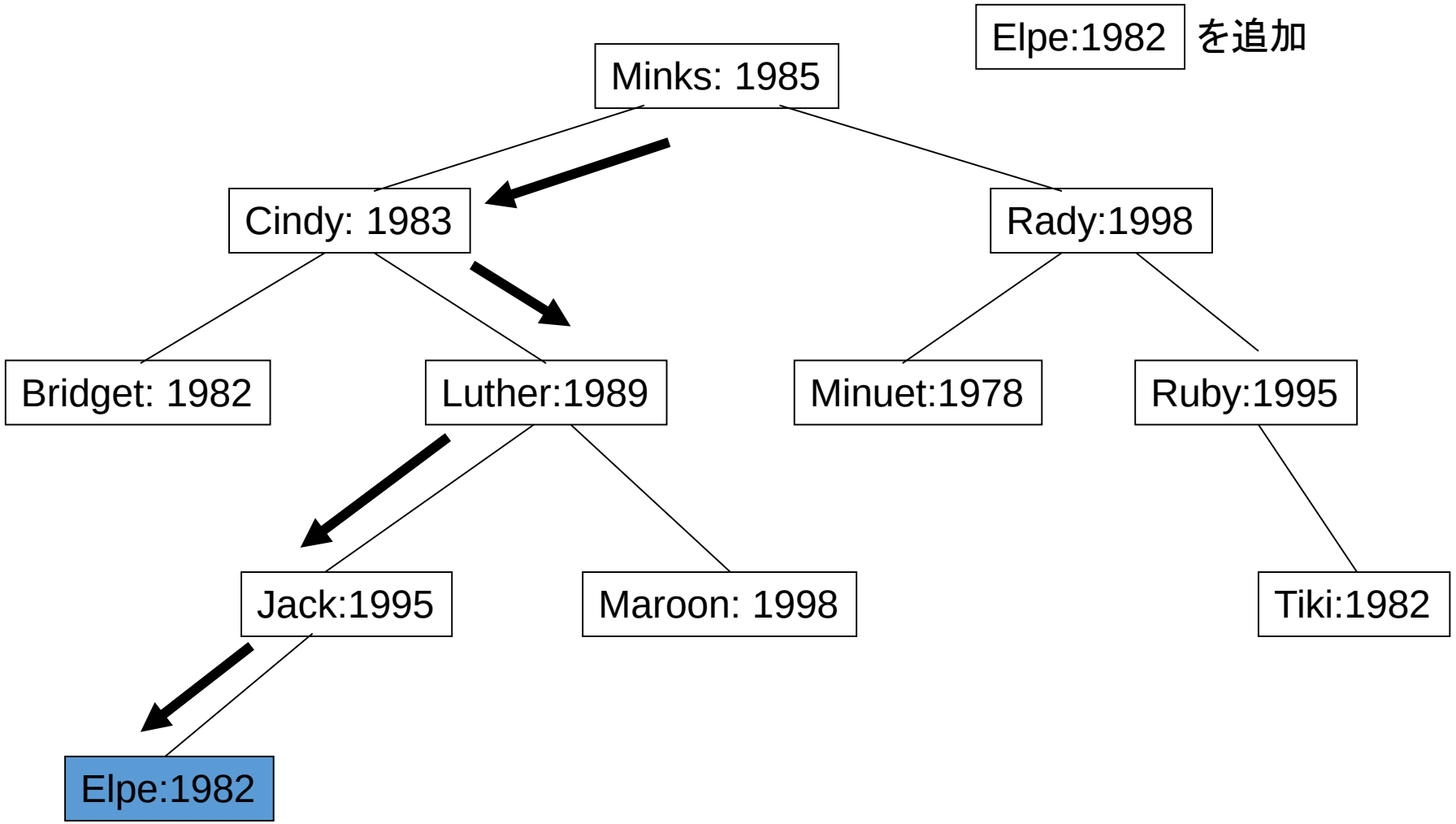
- Binary Search Tree
  - ある検索キーを与えると，それに対応する値を検索可能

- あるノード **x** に対して
  - 左部分木の各ノードの値は **x** より小さい.
  - 右部分木の各ノードの値は **x** より大きい.

# ノードの探索


Maroon
を検索
Minks: 1985
Cindy: 1983
Rady:1998
Bridget: 1982
Luther:1989
Minuet:1978
Ruby:1995
Jack:1995
Maroon: 1998
Tiki:1982

# ノードの追加


Elpe:1982 を追加
Minks: 1985
Cindy: 1983
Rady:1998
Bridget: 1982
Luther:1989
Minuet:1978
Ruby:1995
Jack:1995
Maroon: 1998
Tiki:1982
Elpe:1982

# ノードを探索する関数

```
struct node *search_node(struct
 node *pointer, char *key);
```

- 見つかったノードへのポインタを返す
- ノードへのポインタと探索用のキーを引数

# ノードを探索する関数

1. 引数として与えられたポインタが**NULL**ならば、見つからなかったとして**NULL**を返す

```
if(pointerの示す先がNULL) {
  NULLを返す;
}
```

2. 値が探索対象だったら、そのノードへのポインタを返す

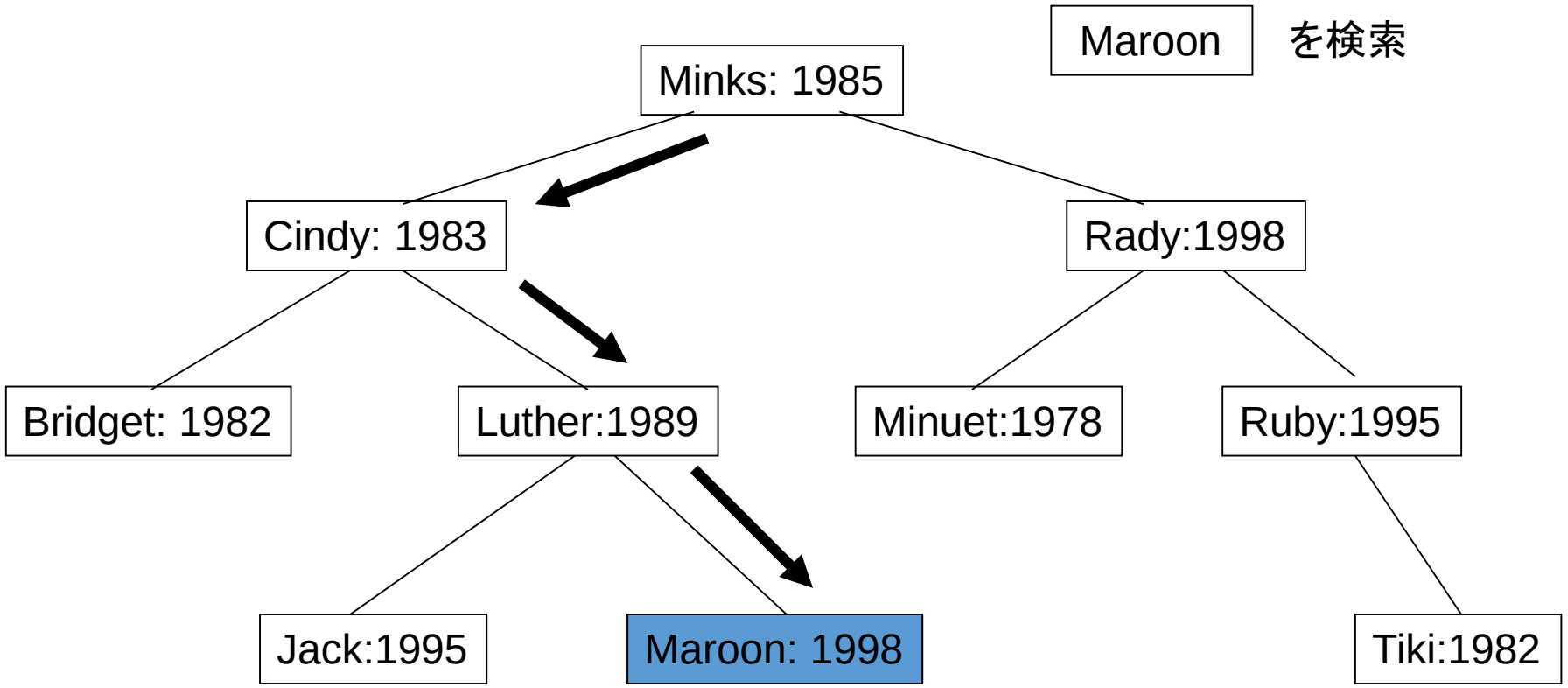
```
if(pointerの示す先の値がキー) {
  pointerの値を返す;
}
```

# ノードを探索する関数

3. ポインタの示す先の値が探索対象より大きければ(探索対象のほうが小さければ)左子部分木を再帰的に探索、小さければ右子部分木を再帰的に探索

```
if(pointerの示す値が対象より大きい) {
   左の子部分木を探した結果を返す;
} else {
   右の子部分木を探した結果を返す;
}
```

# ノードの探索


Maroon
を検索
Minks: 1985
Cindy: 1983
Rady:1998
Bridget: 1982
Luther:1989
Minuet:1978
Ruby:1995
Jack:1995
Maroon: 1998
Tiki:1982

# ノードを追加する関数

```
void add_node(struct node
 **pointer, struct node *new_node)
```

- ノードの追加位置の候補を示すポインタへのポインタと、追加するノードへのポインタを引数

1. 追加位置の候補を示すポインタがNULLなら追加

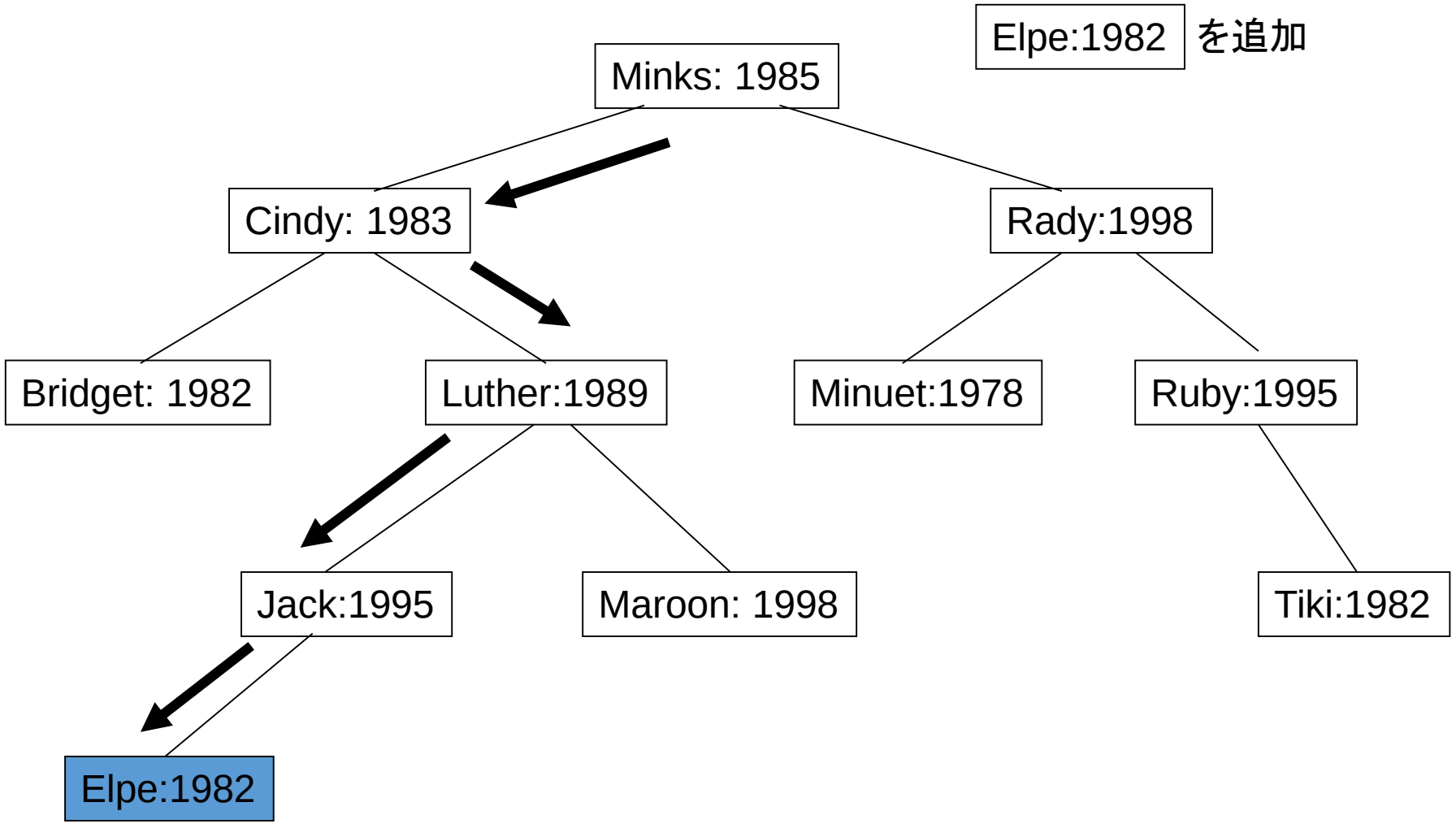
```
if(*pointerの示す先がNULL) {
  *pointerに新規ノードへのポインタを代入;
}
```

# ノードを追加する関数

2. 空いてなければ、そのノードの値と新規ノードを比較し、新規ノードの値が小さければ左の子、大きければ右の子を調べる

```
if(*pointerの示す値より新規ノードが小さい) {
   左の子部分木に対し追加位置を調べる;
} else {
   右の子部分木に対し追加位置を調べる;
}
```

# ノードの追加


Elpe:1982 を追加
Minks: 1985
Cindy: 1983
Rady:1998
Bridget: 1982
Luther:1989
Minuet:1978
Ruby:1995
Jack:1995
Maroon: 1998
Tiki:1982
Elpe:1982

# トラバーサル

- 二分探索木のすべてのノードを訪問
  - 二分木のノードの表示に使用
  - 再帰的な処理
- 訪れたノードを表示する処理をどこに置くかで3種類

# 行きがけ順

- 先順 : Preorder Traversal

  1. ノードの表示
  2. 左の木を走査する再帰呼出
  3. 右の木を走査する再帰呼出

# 通りがけ順

- 中順：Inorder Traversal

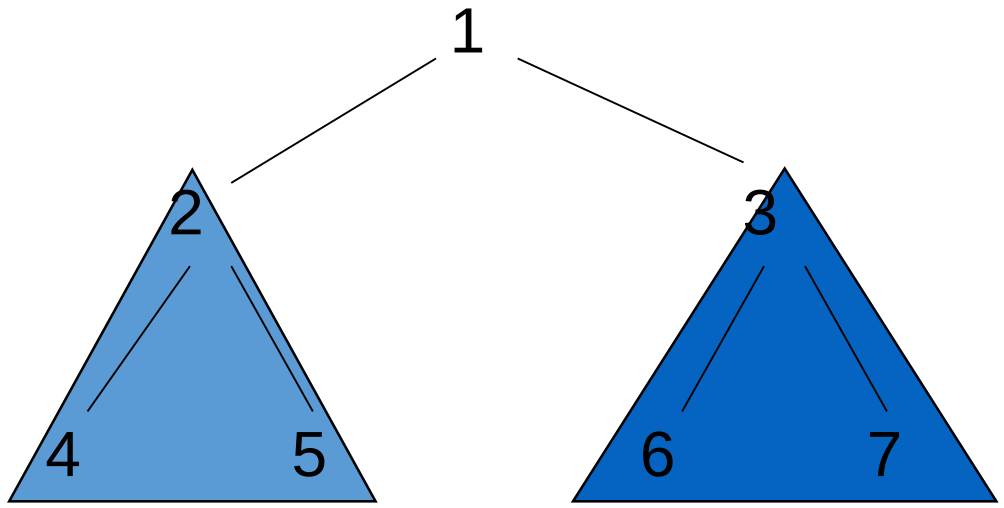
  1. 左の木を走査する再帰呼出
  2. ノードの表示
  3. 右の木を走査する再帰呼出

# 帰りがけ順

- 後順 : Postorder Traversal

  1. 左の木を走査する再帰呼出
  2. 右の木を走査する再帰呼出
  3. ノードの表示

# トラバースの結果




Pre-order: 1 2 4 5 3 6 7

In-order: 4 2 5 1 6 3 7

Post-order: 4 5 2 6 7 3 1

# 完全にバランスした二分木

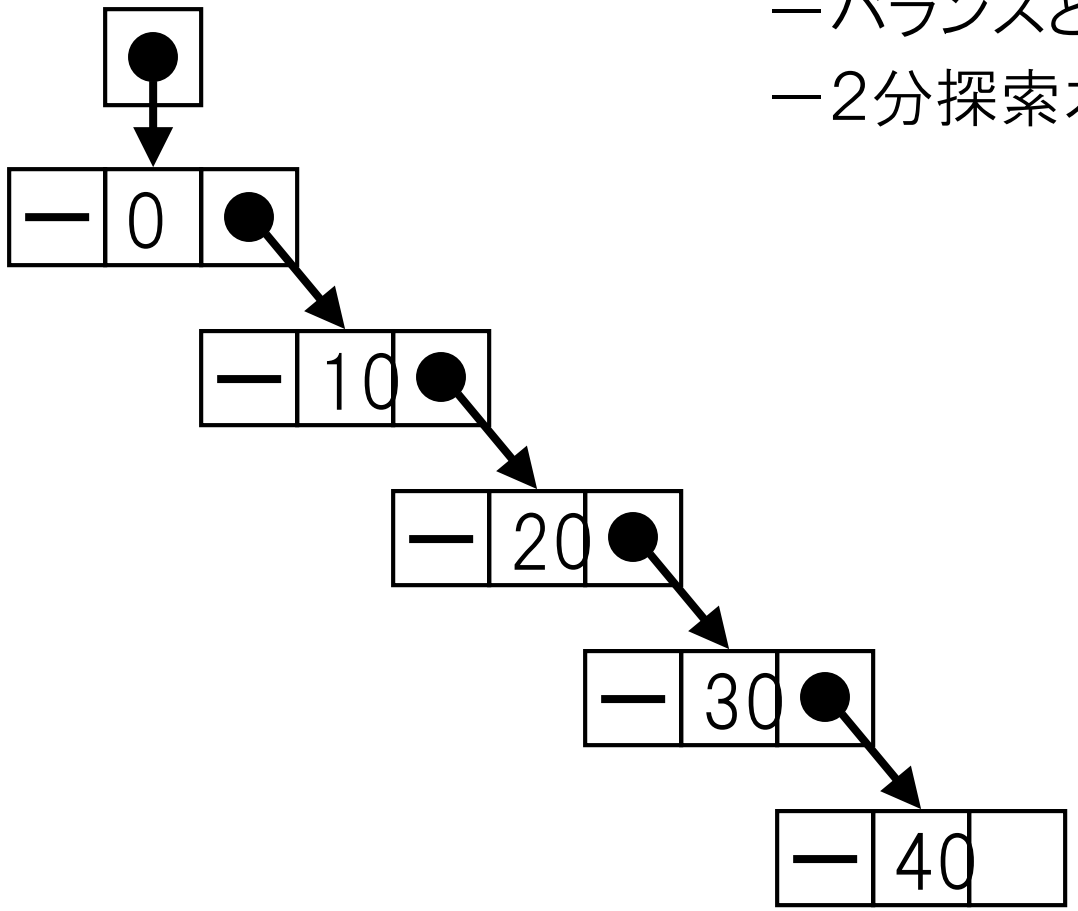

岩手県立大学
Iwate Prefectural University

# バランスのよい2分木

✓2分木はバランスしていることが望ましい.

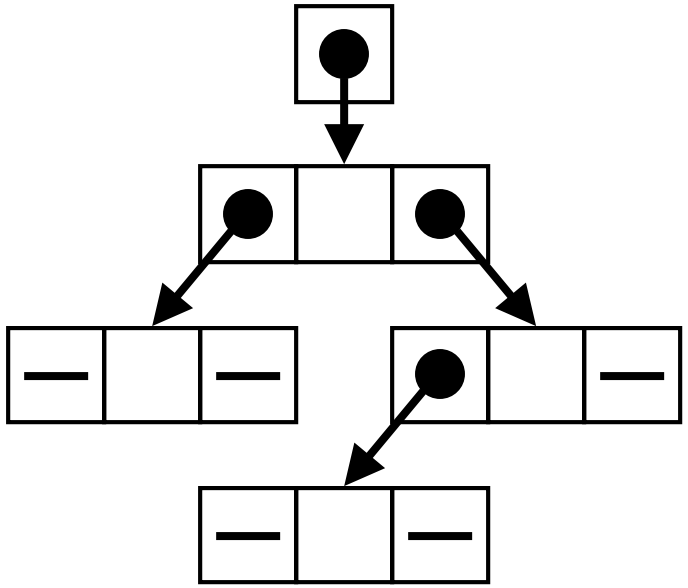
ーバランスとは

ー2分探索木と連結リストとの比較

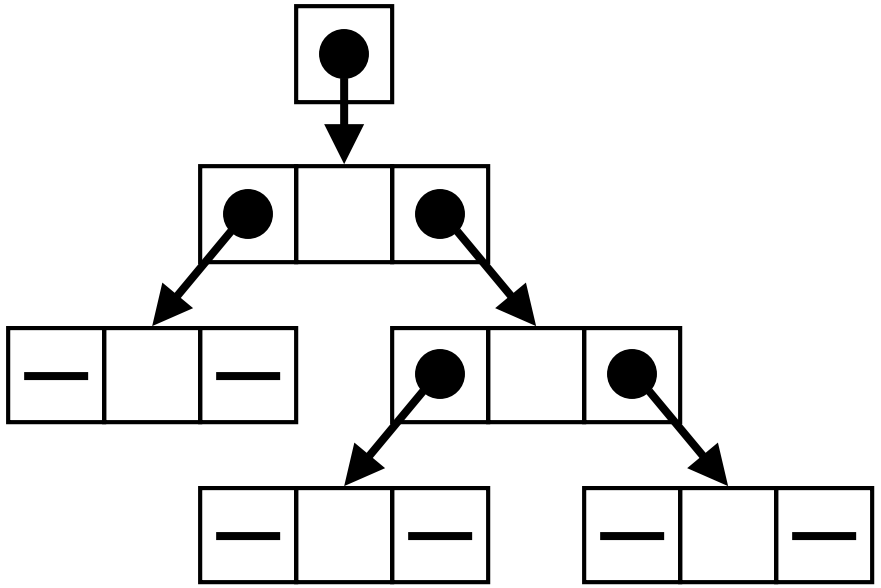


# 完全にバランスした木

✓全てのノードにおいて，その左部分木と右部分木で
それぞれのノードの数がたかだか1つしか違わない木.



完全にバランス



高さがバランス（AVL木）

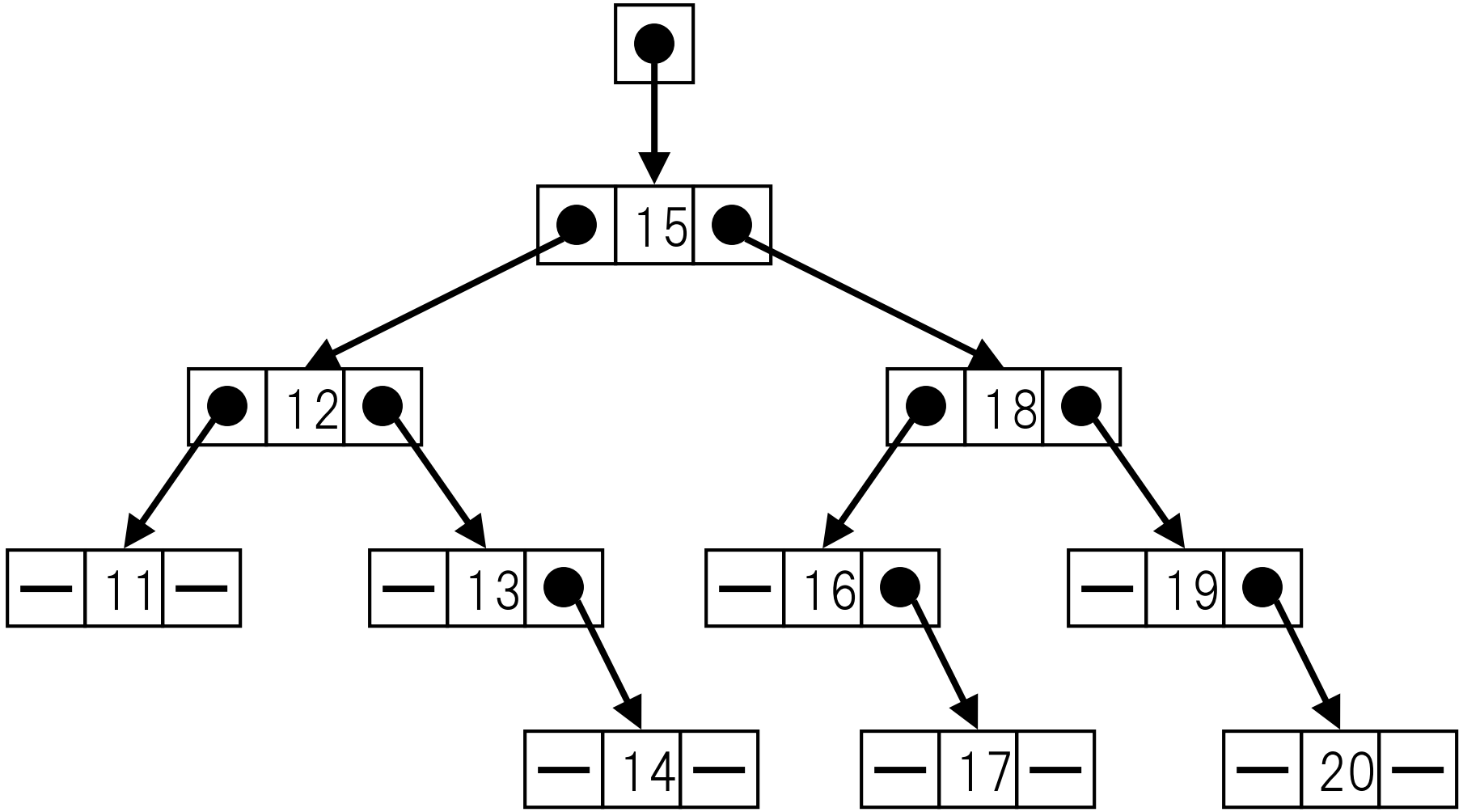
# 変換

✓2分木から「完全にバランスした2分木」への変換
✓読み込むデータが昇順で与えられている.
✓データの数があらかじめわかっている.

```
 1: node* pbtree(int n) {
 2:     int nleft, nright, nleftplusright = n - 1;
 3:     node* p;
 4:     if (n == 0) { return NULL; }
 5:     nleft = nleftplusright / 2;
 6:     nright = nleftplusright - nleft;
 7:     p = (node*)malloc(sizeof(node));
 8:     p->left = pbtree(nleft);
 9:     scanf("%d", &p->num);
10:     p->right = pbtree(nright);
11:     return p;
12: }
```

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;
```

# 2分探索木(データ数:10)



# 情報システムへの応用

✓データ：個人名と数値（電話番号，登録番号など）

1. ファイルから全てのデータを読み込み，
   完全にバランスした2分探索木を生成する．（.L）
2. 指定した名前のデータを探索する．（<name> ?）
3. 新しいデータを追加する．（<name> <value>）
4. 指定したデータを削除する．（<name> /）
5. 指定したデータの数値を変更する．（<name> <value>）
6. 全てのデータをファイルにセーブする．（.S）
7. 全てのデータをアルファベット順に表示する．（.P）

P.175～ ソースファイル ： infsys2.c

# 問題

- 下記の8個のデータに対して，関数 pbtree を実行したときの，完全にバランスした2分木を答えなさい.
  - 1, 2, 3, 4, 5, 6, 7, 8

# 2分探索木(データ数:8)