



アルゴリズムとデータ構造 並びに同演習 ～第10回 木構造と2分探索木～

総合情報学専攻
メディア情報学専攻
橋本直己

naoki@cs.uec.ac.jp



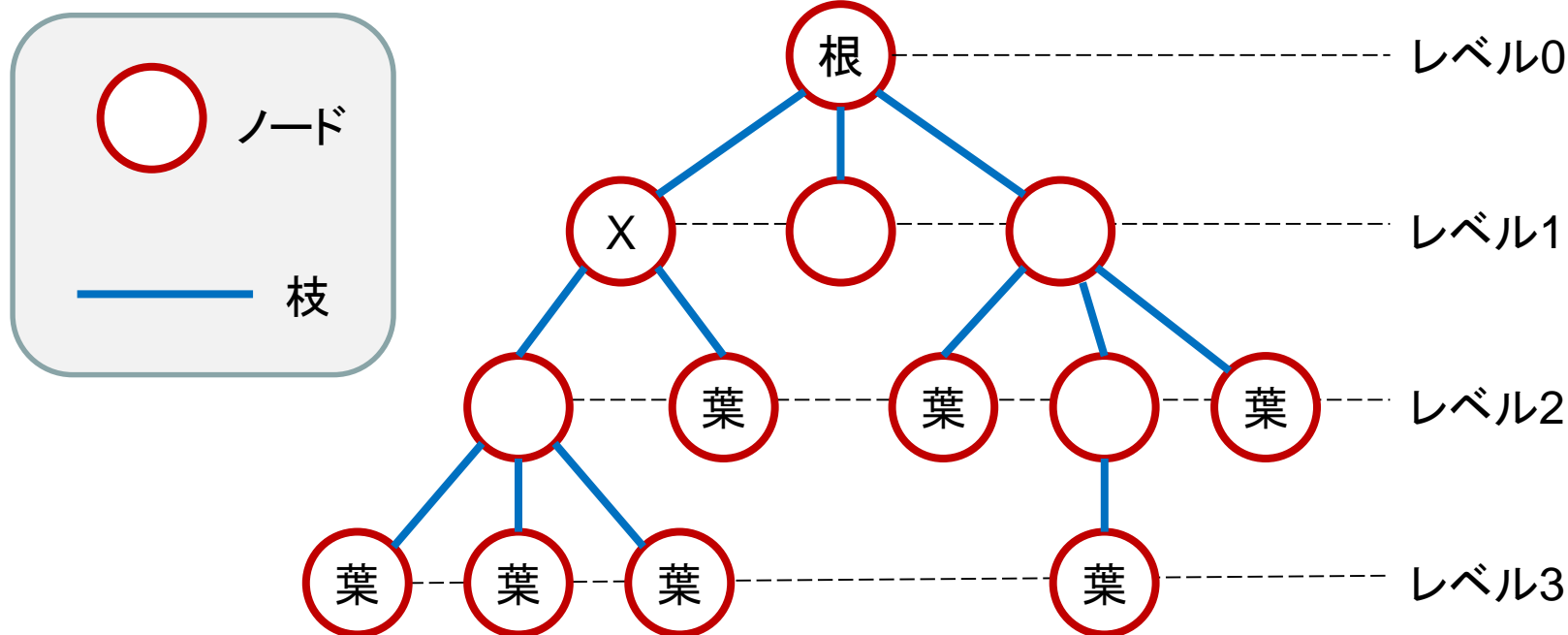


木構造を学ぼう！

- **木構造**(tree structure)は、階層的な関係を表現するデータ構造
 - 以前学習した「リスト構造」は、順序づけられたデータを表現していた.
- 既に木構造を学んでいるかもしれませんが、その場合には復習だと思って確認してください.

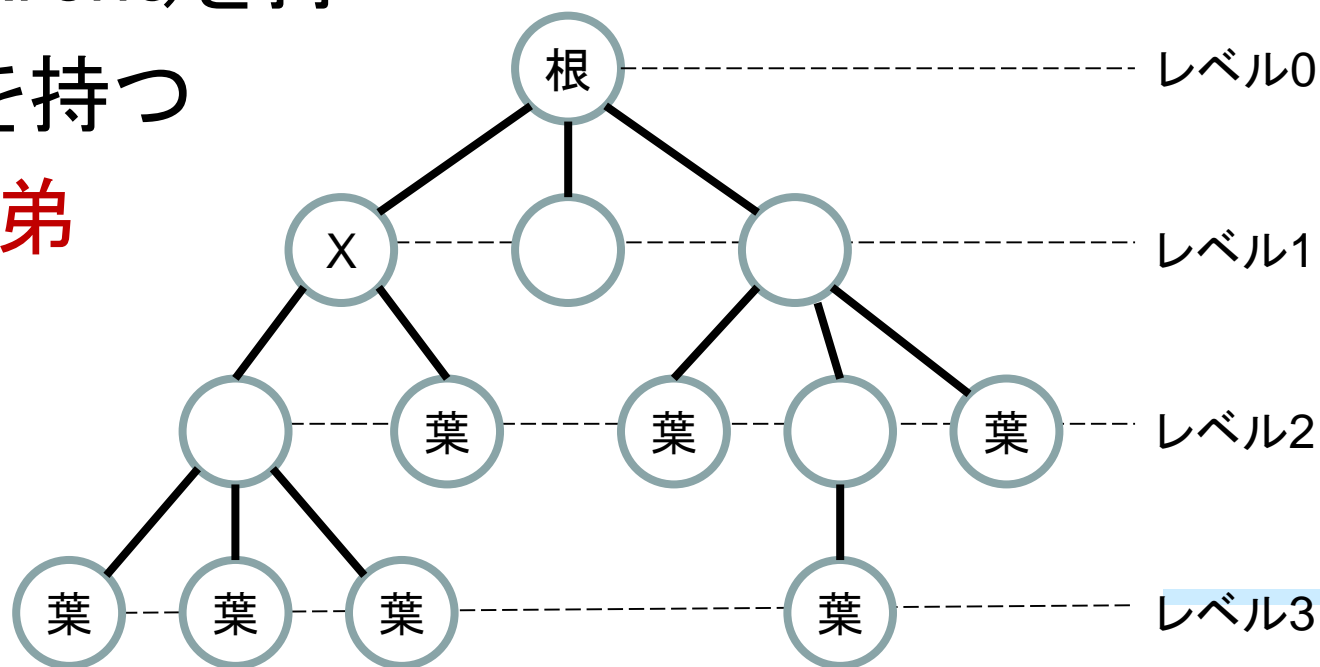
木に関する用語①

- で示している**ノード**(節/node)が、線で示している**枝**(edge)で結ばれている。
 - 上下逆さまにすると<木>のイメージ



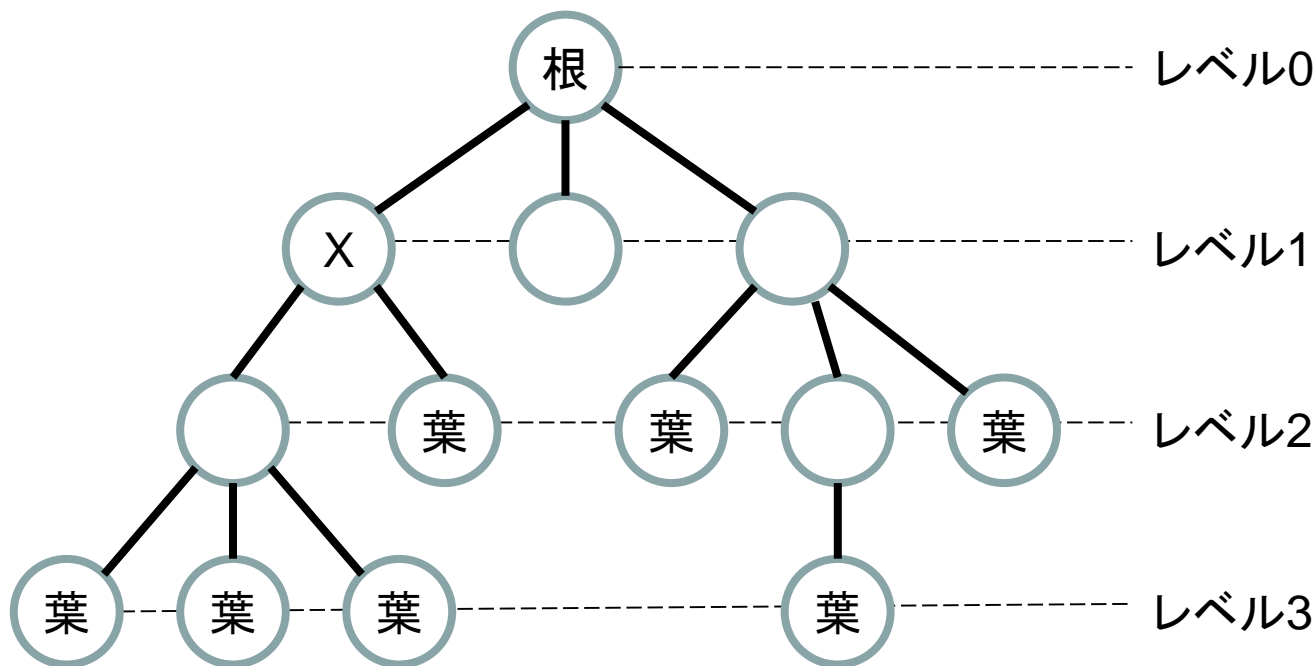
木に関する用語②

- 最も上にあるノードを**根**(root)と呼ぶ
 - 1つの木に対して、根は1つだけ存在
- 各ノードは複数の**子**(child)を持ち、1つの**親**(parent)を持つ
- 共通の親を持つノードは**兄弟**(sibling)



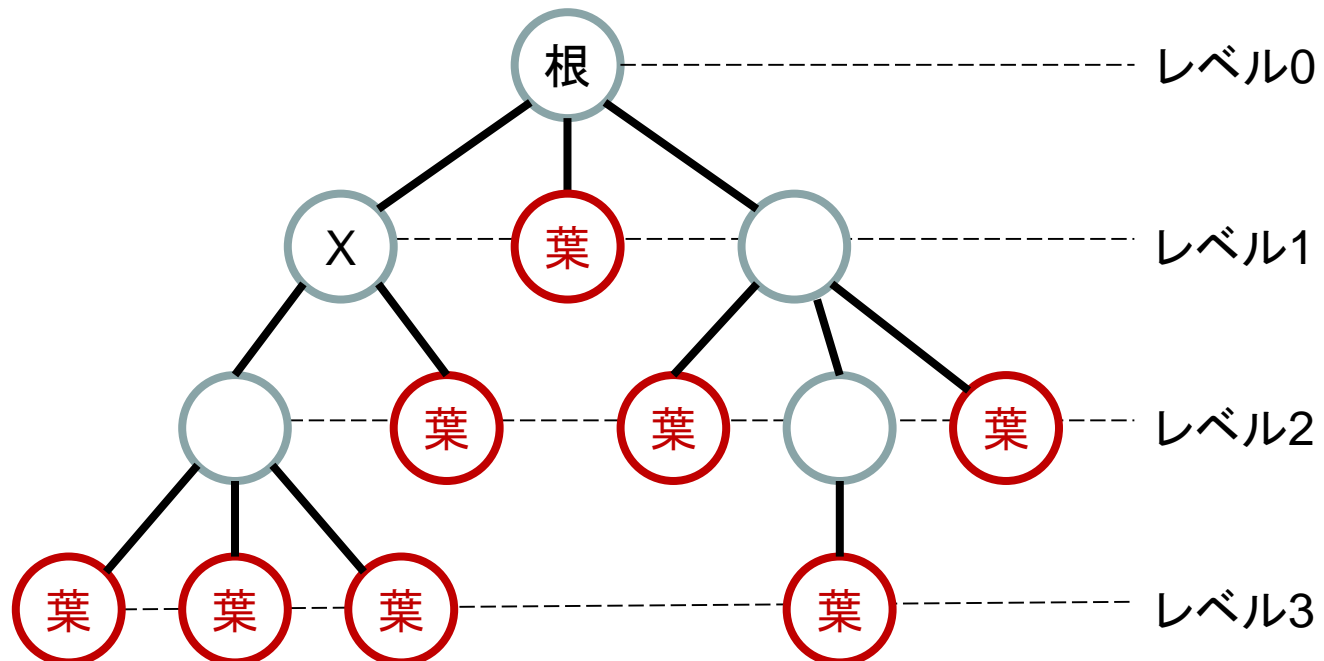
木に関する用語③

- 図の上方(根の方)を**上流**, 逆側を**下流**と呼ぶ.
 - あるノードより上流側にたどれるノードは**先祖**(ancestor)
 - 逆に下流側にたどれるノードは**子孫**(descendant)



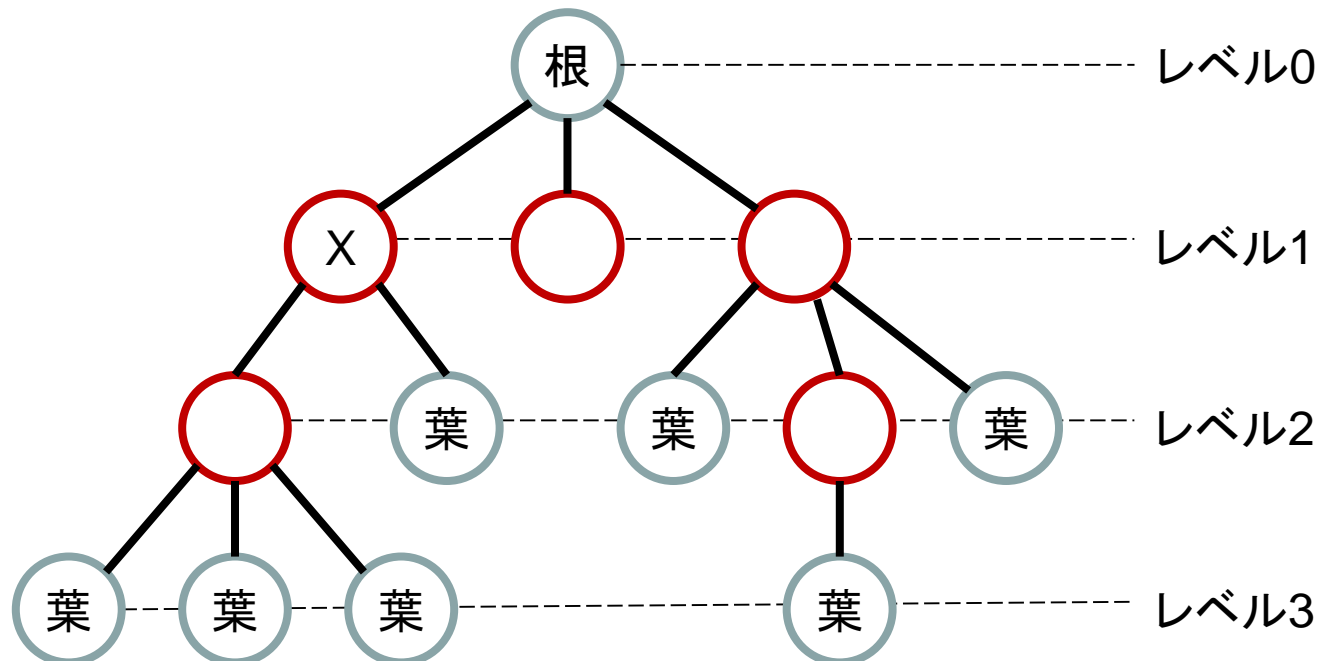
木に関する用語④

- 最下流のノードを**葉**(leaf), **終端節**(terminal node), **外部節**(external node)等と呼ぶ. 当然, **葉に子はない**.



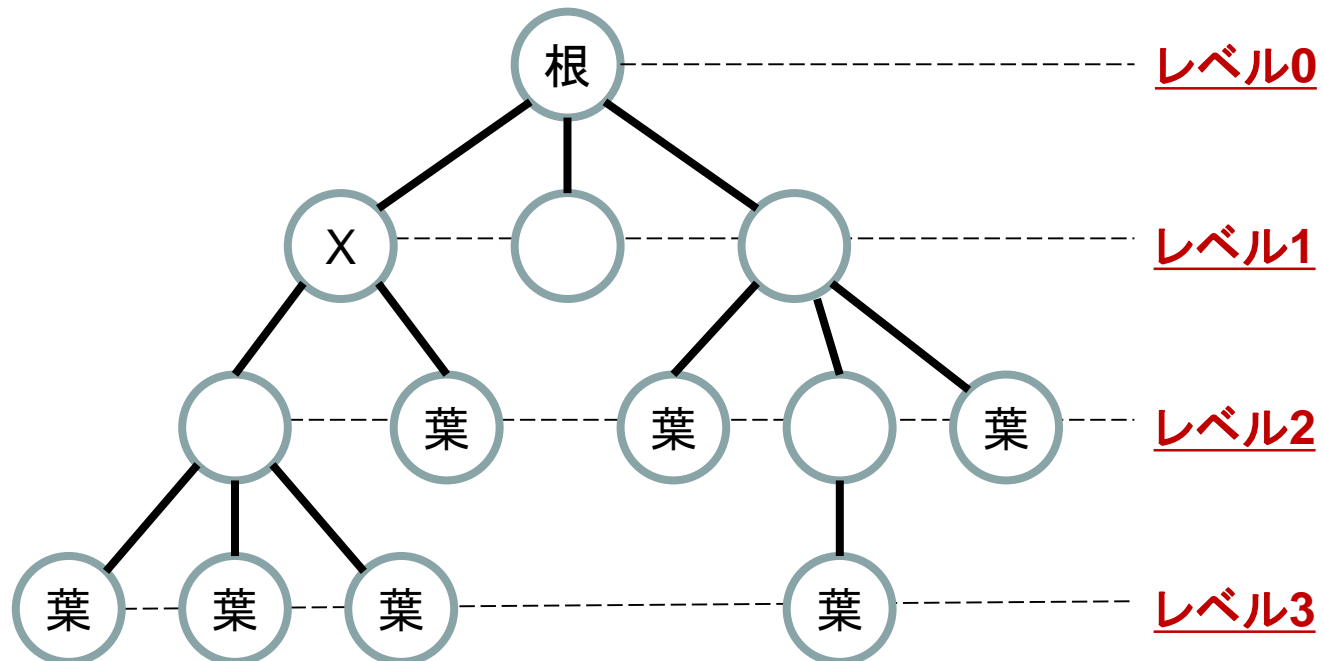
木に関する用語⑤

- 葉以外のノードを**非終端節**(non-terminal node), **内部節**(internal node)と呼ぶ.



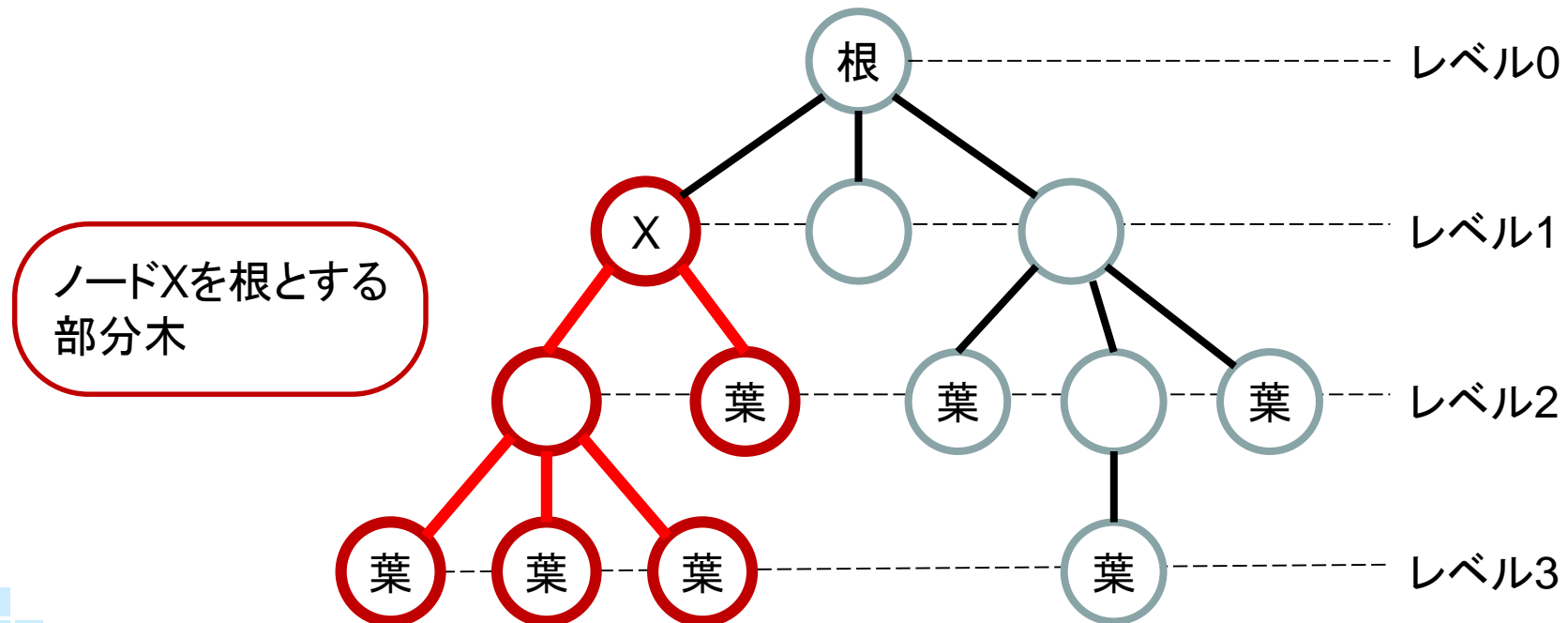
木に関する用語⑥

- 根からどれくらい離れているかを示すのが
レベル(level)
 - 最上流である根のレベルは0
 - 枝を1つ下流へとたどる毎にレベルは1増加



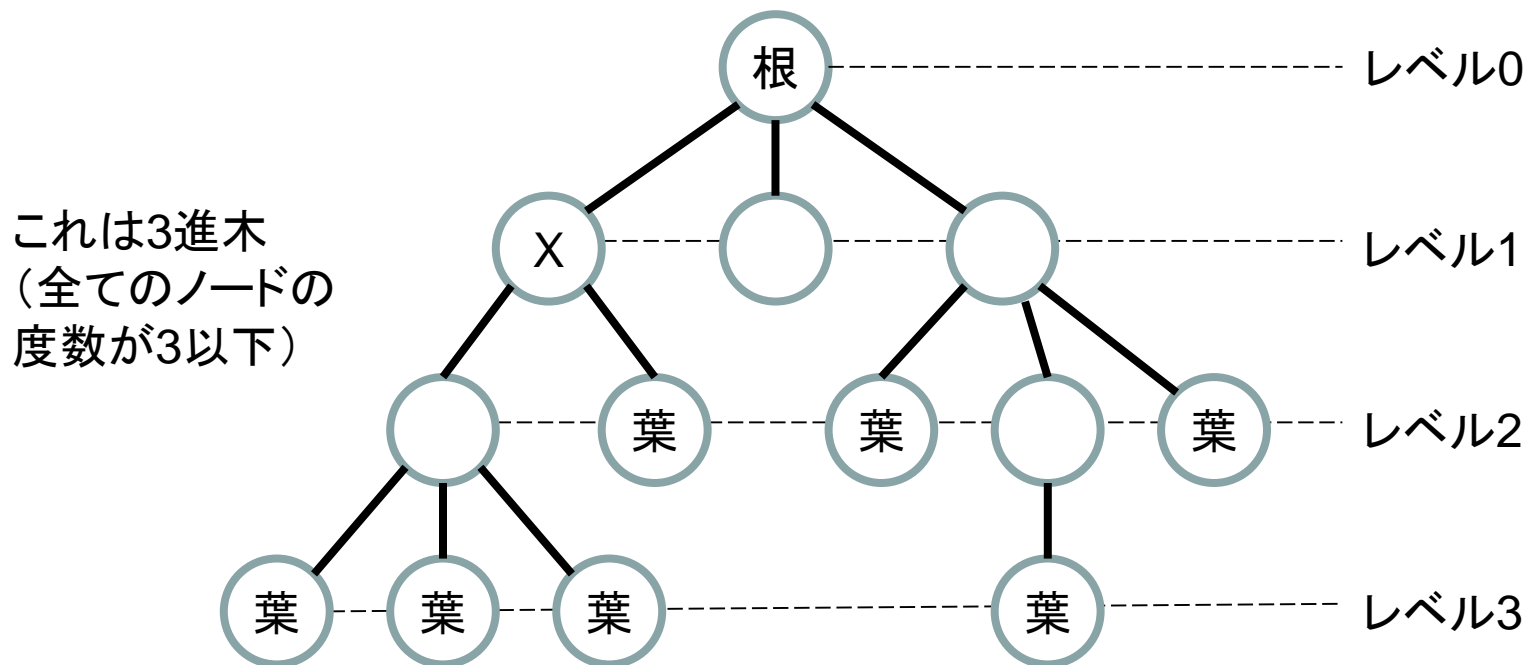
木に関する用語⑦

- あるノードに着目すると, そこから下流の部分も木構造となる
 - このような木の一部である木を**部分木**(subtree)



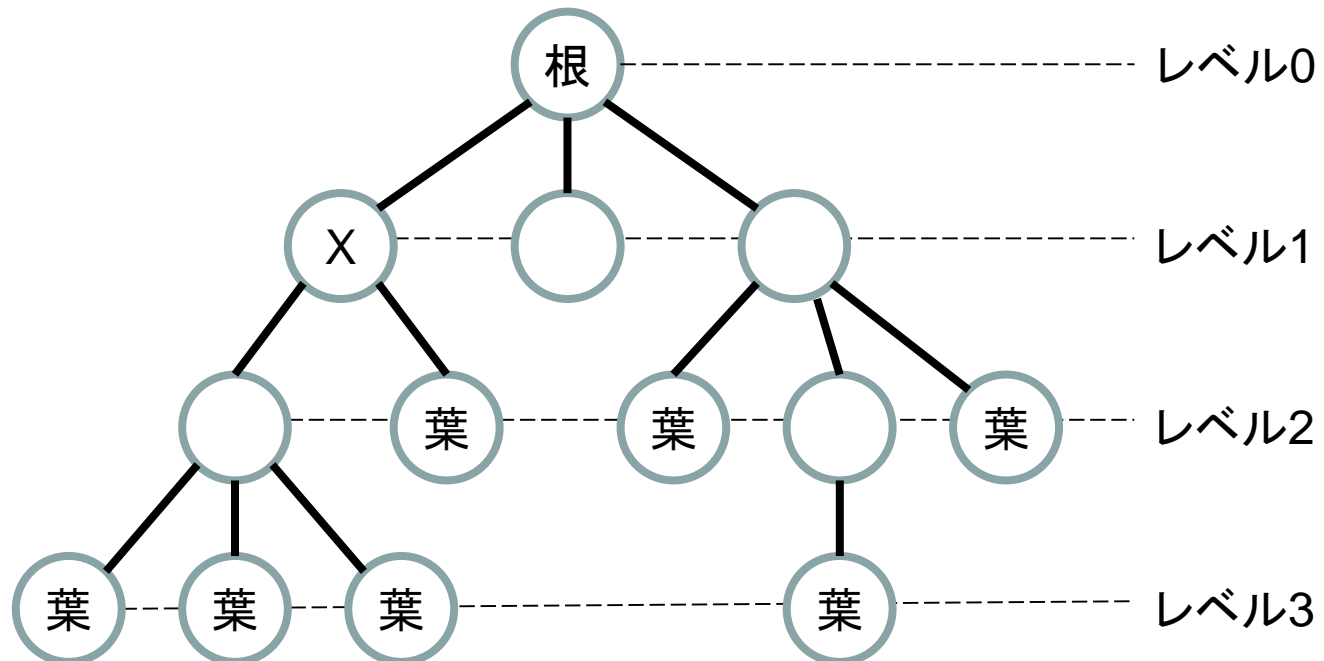
木に関する用語⑧

- 各ノードがもつ子の数を**度数**(degree)
 - 例えば, Xの**度数**は2
- 全てのノードの度数がn以下である木を**n進木**



木に関する用語⑨

- 根から最も遠い葉までの距離を木の**高さ**(height)
 - 以下の木の高さは3
 - すなわち, **葉のレベルの最大値**



木に関する用語⑩

- 根以外のノードや枝が全く存在しないものも木
 - 空木(null tree)

根

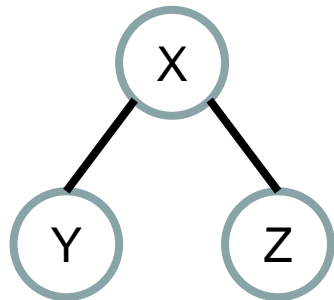


木構造を探索に活用しよう！

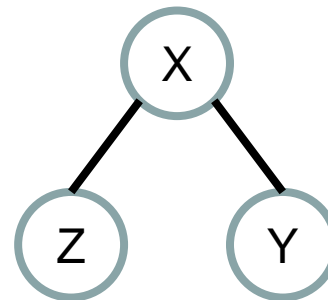
- これまで学んできた木構造を使って、効率的な探索アルゴリズムを取り扱います.
- 木は単なる構造なので, これに“**順序**”を導入します.

順序木と無順序木

- 兄弟関係にあるノードについて、順序関係を有する木が**順序木**(ordered tree), そうでない木を**無順序木**(unordered tree)
- 以下の2つは、順序木ならば別の木だが、無順序木としてみれば同じ木となる.



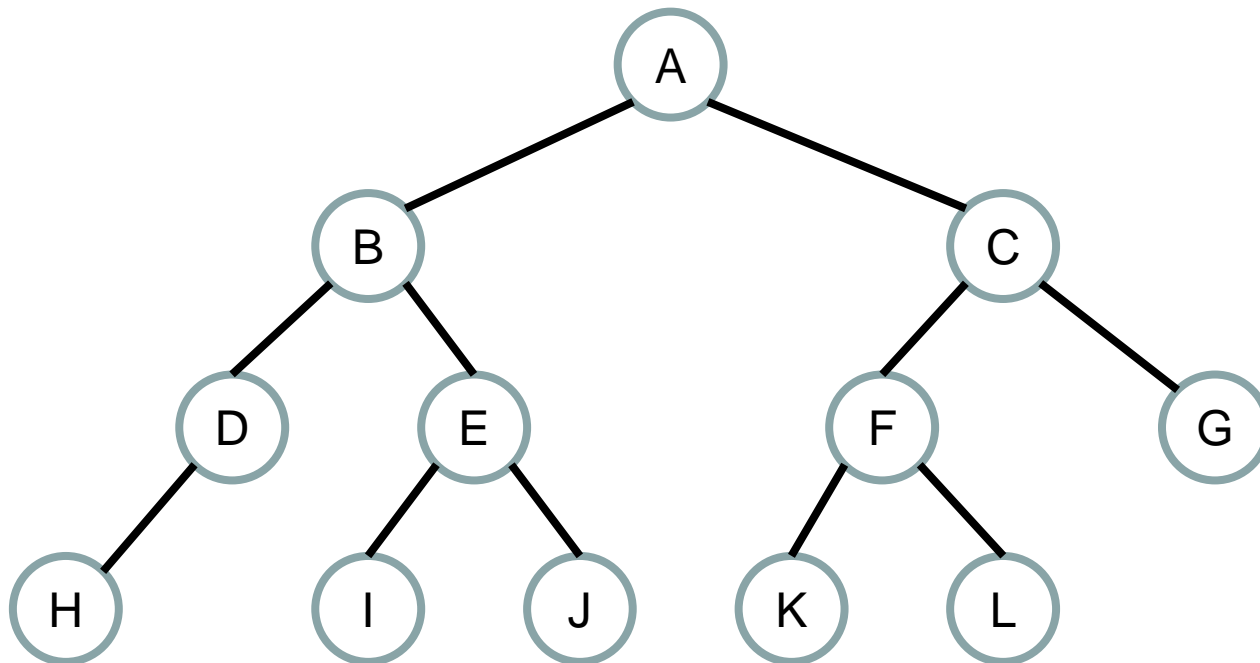
(a)



(b)

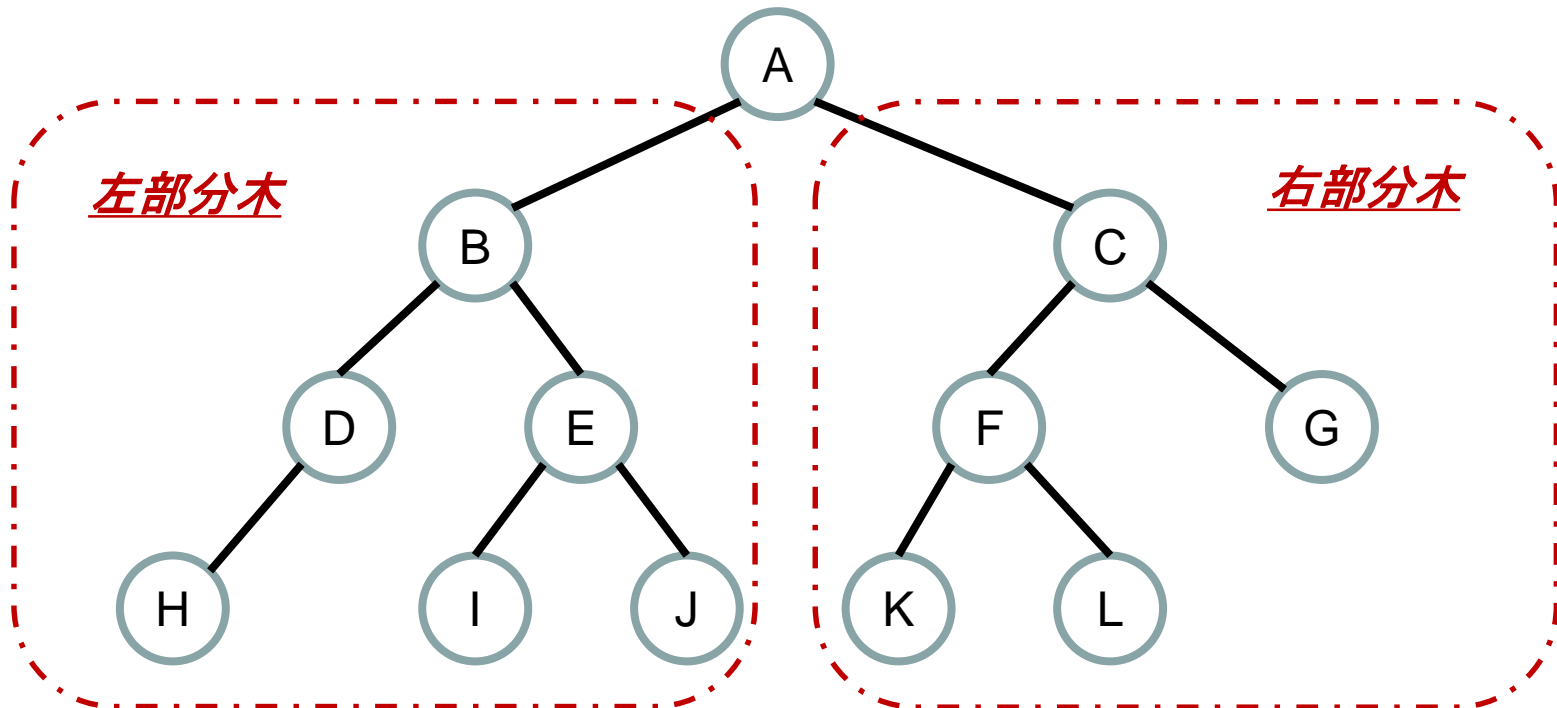
2分木①


- 木上のノードが、**左の子**(left child)と**右の子**(right child)の(最大)2つの子をもつ木を**2分木**(binary tree)と呼ぶ。
= 2進木かつ順序木(右の子と左の子を区別)



2分木②

- 根 (root) であるAの左の子がB, 右の子がC
 - 左の子を根とする部分木を**左部分木**(left subtree)
 - 右の子を根とする部分木を**右部分木**(right subtree)





2分木を使ったデータ表現

- 2進木かつ順序木である2分木は、ある一定の順序を持ったデータを格納することができます。
- まずは、木構造で表現されたデータの読み取り方を学びましょう！

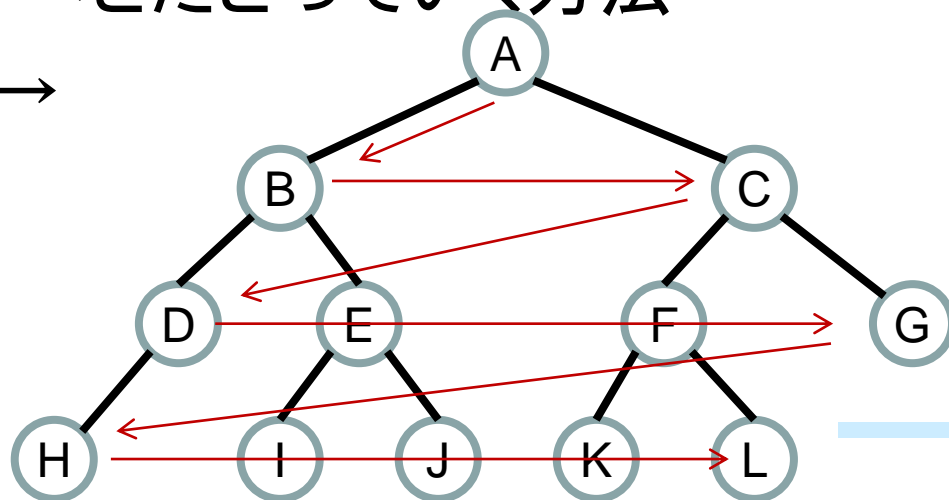
木の探索～横型探索～

- 2分木に限らず，順序木上のノードをなぞっていく方法として，大きく2つの方法がある。

- 横型探索**（幅優先順探索）

- レベルの低い点から始めて，左側から右側へ，それが終わると次のレベルへとたどっていく方法

- $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow$
 $G \rightarrow H \rightarrow I \rightarrow J \rightarrow K \rightarrow L$

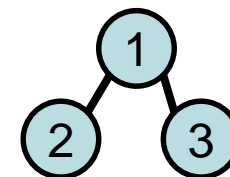


木の探索～縦型探索～①

- 縦型探索（深さ優先順探索）
 - まず葉に到達するまで枝を下るのを優先
 - 葉に到達して行き止まった場合、1つ親に戻り、次のノードへとたどっていく
- 三つの方法がある
 1. 行きがけ順／前順 (preorder)
 2. 通りがけ順／間順 (inorder)
 3. 帰りがけ順／後順 (postorder)

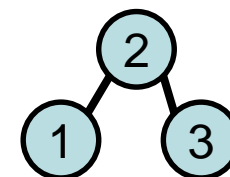
木の探索～縦型探索～②

1. 行きがけ順／前順 (preorder)



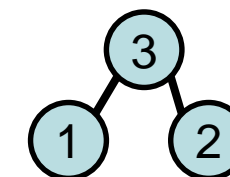
- 親ノードに立ち寄る→左部分木をなぞる→右部分木をなぞる

2. 通りがけ順／間順 (inorder)



- 左部分木をなぞる→親ノードに立ち寄る→右部分木をなぞる

3. 帰りがけ順／後順 (postorder)



- 左部分木をなぞる→右部分木をなぞる→親ノードに立ち寄る

○ 基本ルール: 左部分木→右部分木

「いつ親ノードに立ち寄るか？」で区別される

木の探索～縦型探索～③

1. 行きがけ順／前順 (preorder)

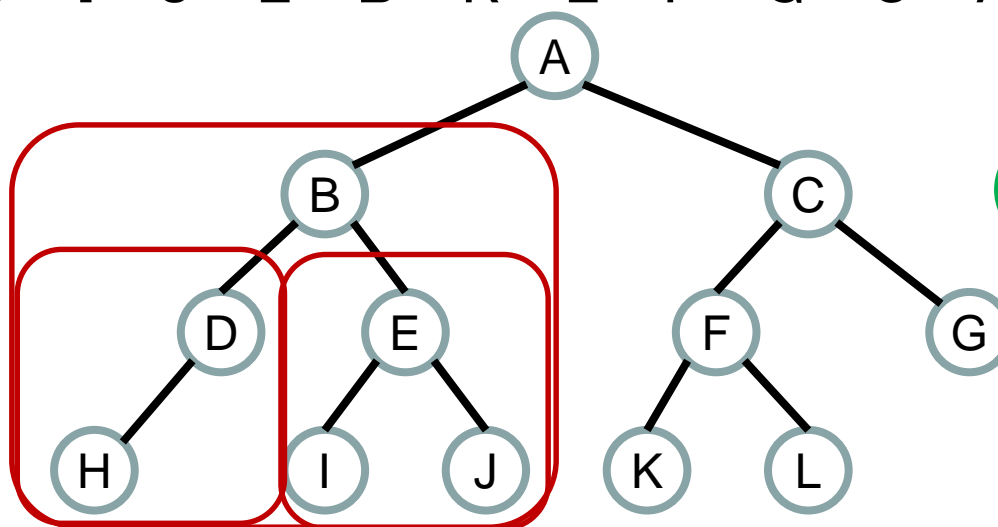
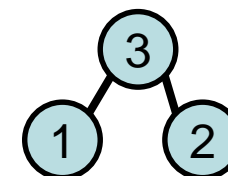
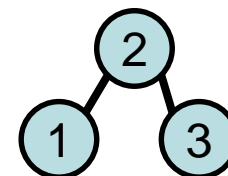
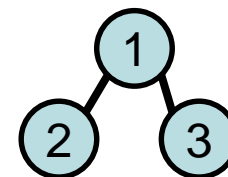
— $A \rightarrow B \rightarrow D \rightarrow H \rightarrow E \rightarrow I \rightarrow J \rightarrow C \rightarrow F \rightarrow K \rightarrow L \rightarrow G$

2. 通りがけ順／間順 (inorder)

— $H \rightarrow D \rightarrow B \rightarrow I \rightarrow E \rightarrow J \rightarrow A \rightarrow K \rightarrow F \rightarrow L \rightarrow C \rightarrow G$

3. 帰りがけ順／後順 (postorder)


— $H \rightarrow D \rightarrow I \rightarrow J \rightarrow E \rightarrow B \rightarrow K \rightarrow L \rightarrow F \rightarrow G \rightarrow C \rightarrow A$



木が複雑な
場合には再帰
的に探索

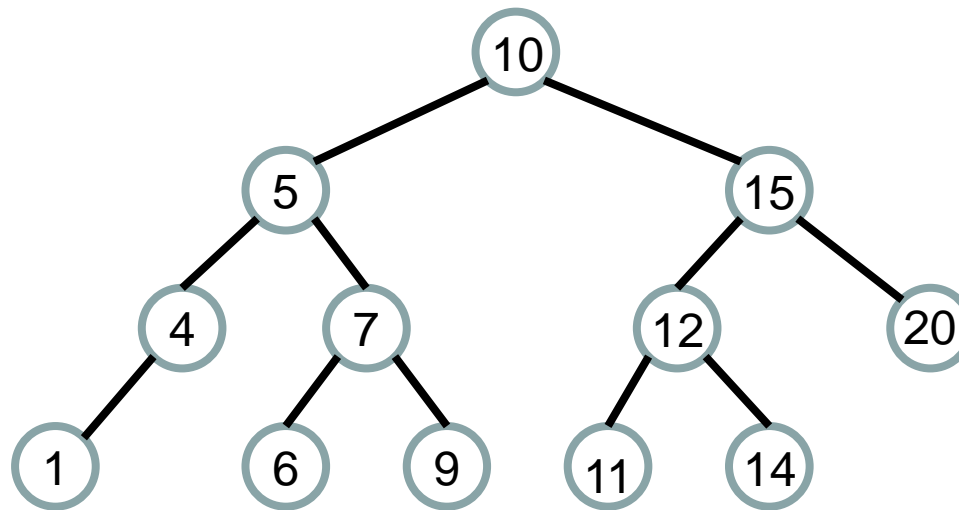


木構造を使った探索

- これまで学んできた木構造を使って、実際にデータの探索アルゴリズムを取り扱います.
 - 基本は2分木を使います.
 - 縦型探索手法が重要になります.
- 

2分探索木


- 全てのノードに対して、①左部分木のノードの値は、そのノード値より小さく、②右部分木のノードの値は、そのノード値より大きいという条件を満たす2分木を2分探索木(binary search tree)と呼ぶ。



この木を通りがけ順の縦型探索でなぞると、 $1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \dots$ となり、キー値の昇順でノードが得られる



2分木探索の実現(1)

- code10-1.cを参照
 - Data型
 - 会員番号と氏名をセットした構造体
 - 氏名をキーとする
 - BinNode型
 - 2分探索木上のノードを表すための型
 - leftは左の子ノードへのポインタ(rightも同様)
- 



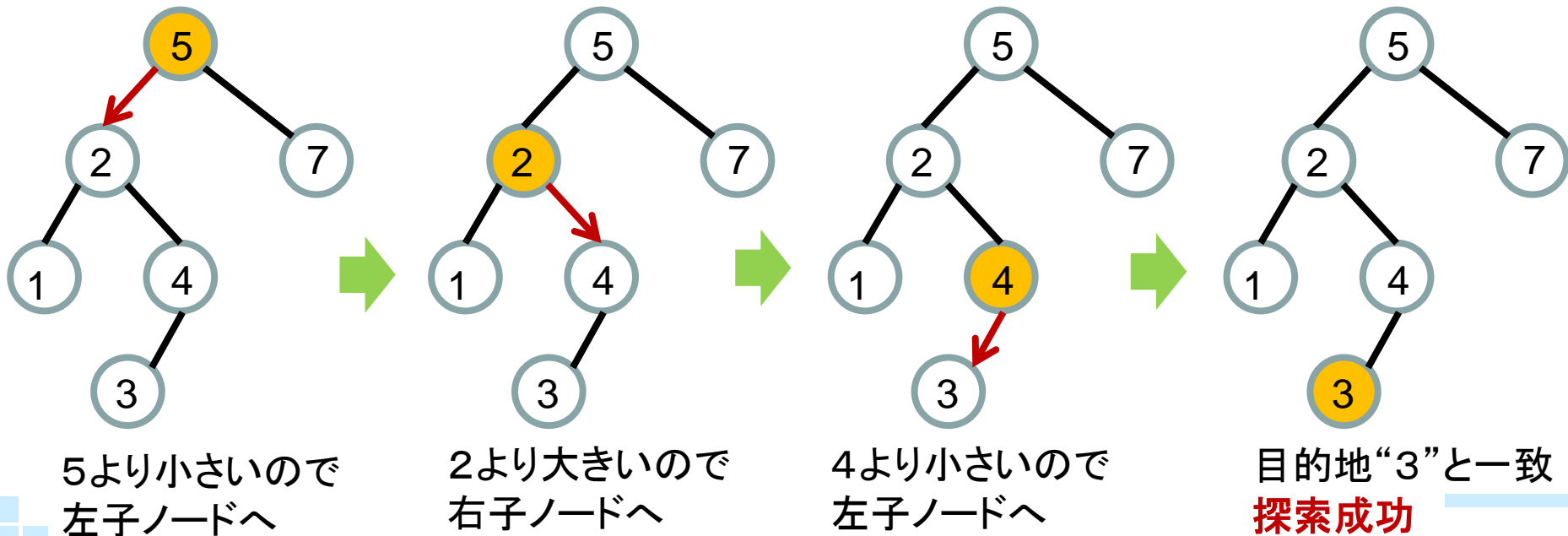
2分木探索の実現(2)

- 関数NameCmp
 - キー値の大小比較を行う
 - $x > y$ ならば1, $x = y$ ならば0, $x < y$ ならば-1
- 関数AllocNode
 - BinNode型のオブジェクトを動的に生成する
 - ノードを追加するときに使用
- 関数SetBinNode
 - BinNode型のオブジェクトに, キー値, 左の子ノード(ポインタ), 右の子ノード(ポインタ)を追加

ノードの探索

- まず根に着目し，目的とする値の方が小さければ左の子ノード，大きければ右の子ノードをたどっていく。

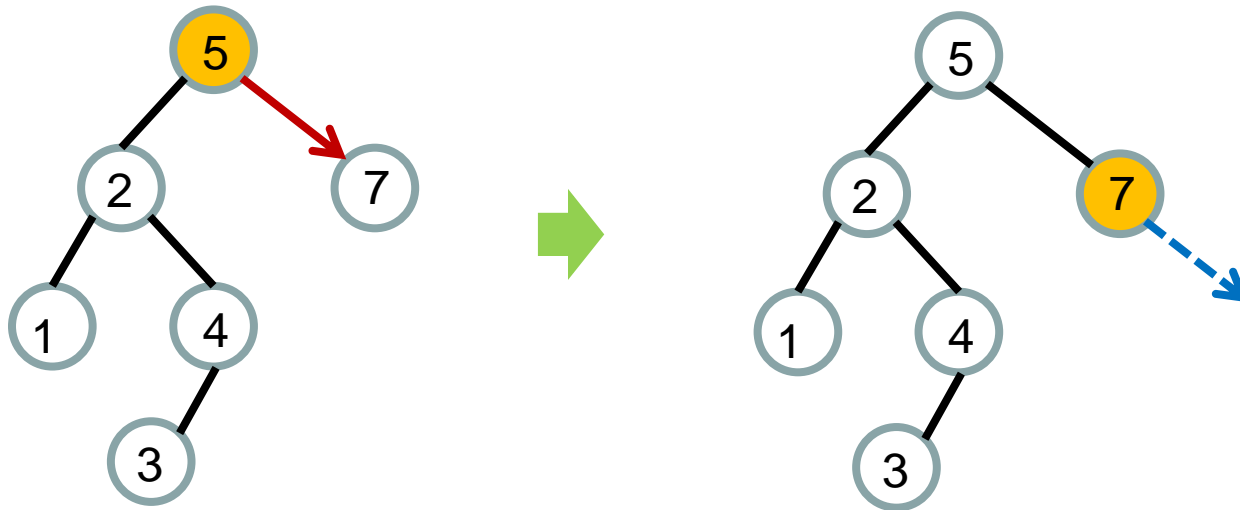
例) 3を探索する



ノードの探索(失敗例)

- 葉に到達して、それ以上たどれない場合は、**探索失敗**となる。

例) 8を探索する



根である5に着目し、目的とする8は5よりも大きいので右の子ノードをたどる

右の子ノードに着目し、値が一致せず、子ノードも存在しないので、**探索失敗**

ノードの探索を行う関数

再帰的に定義
されている

[code10-2.c](#)を
参照

<NameCmp関数の仕様>

「 $X > p \rightarrow data$ 」ならば
(キー値のほうが小さければ)

→ $cond < 0$

「 $X < p \rightarrow data$ 」ならば
(キー値のほうが大きければ)

→ $cond > 0$

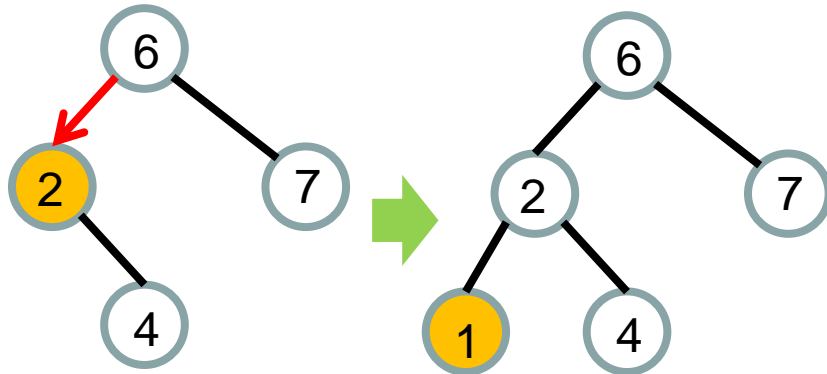
```
/*--- 氏名による探索 ---*/
BinNode *SearchNode(BinNode *p, Data x)
{
    int cond;

    if(p == NULL)
        return (NULL); /* 探索失敗 */
    else if ((cond = NameCmp(x, p->data)) == 0)
        return (p); /* 探索成功 */
    else if (cond < 0)
        return SearchNode(p->left, x); /* 左部分木からの探索 */
    else
        return SearchNode(p->right, x); /* 右部分木からの探索 */
}
```

ノードの挿入

- まず、そのキー値をもつノードが存在するかどうかを探索する
- 探索が**失敗したときのみ挿入**を行う
 - SearchNode関数に似た構成になる

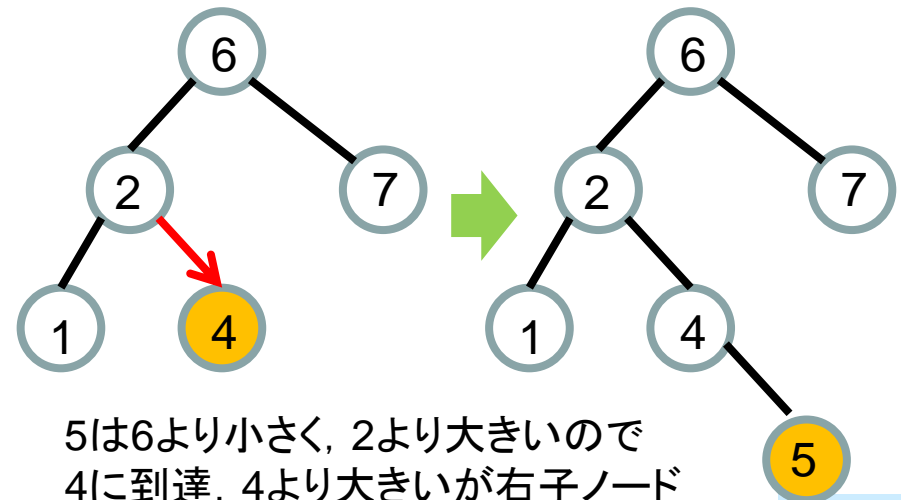
例) 1を挿入



1は6より小さいので左へ.

1は2より小さいが、左子ノードはNULL.なので追加

例) 5を挿入



5は6より小さく、2より大きいので4に到達. 4より大きい右子ノードがNULLないので、右側に追加.

ノードの挿入を行う関数

- 再帰的に定義される

- [code10-3.c](#)を参照

```
/*--- ノードの挿入 ---*/
```

```
BinNode *InsertNode(BinNode *p, Data x)
```

```
{
```

```
    int cond;
```

```
    if( p == NULL ){
```

```
        p = AllocNode();
```

```
        SetBinNode(p, x, NULL, NULL);
```

```
    } else if ((cond = NameCmp(x, p->data)) == 0)
```

```
        printf("【エラー】 %sは既に登録されています。¥n", x.name);
```

```
    else if ( cond < 0 ) // p->dataがxよりも大きい場合
```

```
        p->left = InsertNode(p->left, x);
```

```
    else
```

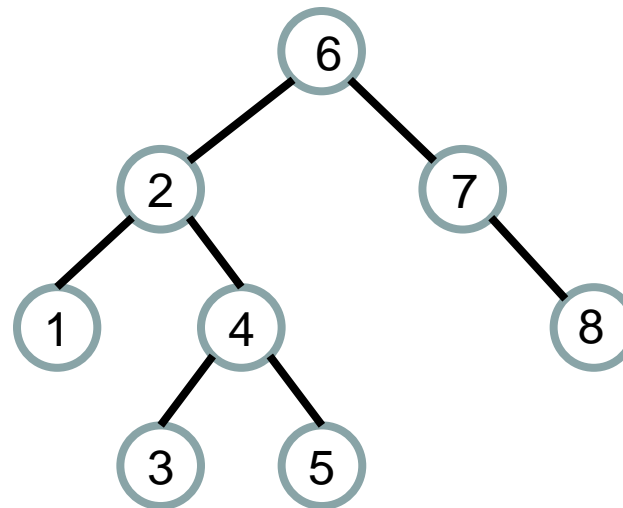
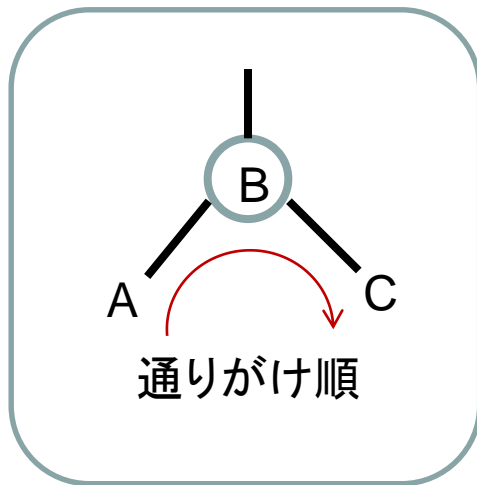
```
        p->right = InsertNode(p->right, x);
```

```
    return (p);
```

```
}
```

全ノードの昇順表示

- 2分探索木上のノードは、通りがけ順の縦型探索でなぞるとキー値を昇順で得られる



通りがけ順でたぐると, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$ と昇順になる

全ノードを昇順表示する関数

- 再帰的に定義される
- [code10-4.c](#)を参照

```
/*--- データの番号と氏名を表示 ---*/  
void PrintData (Data x)  
{  
    printf("番号:%d 氏名:%s¥n", x.no, x.name);  
}  
  
void PrintTree(BinNode *p)  
{  
    if(p != NULL) {  
        PrintTree(p->left);    % 再帰呼び出し  
        PrintData(p->data);  
        PrintTree(p->right); % 再帰呼び出し  
    }  
}
```


全ノードを削除する関数

- 再帰的に定義される
- [code10-5.c](#)を参照
- 関数PrintTreeに似ているが、再帰呼出を行う順番が異なる点に注意
 - 右と左の子を削除してから親を消す
 - = 帰りがけ順

```
/*--- 全ノードを開放 ---*/  
void FreeTree (BinNode *p)  
{  
    if (p != NULL){  
        FreeTree(p->left);  
        FreeTree(p->right);  
        free(p);  
    }  
}
```

演習10-1: 2分探索木の操作

- code10-1～5.cを統合し, 2分探索木を対話的に操作するプログラムを完成させよ.
- またプログラムを実行し, 動作を理解せよ.

(解答例: prog10-1.c) 

prog10-1.cの実行例

> ./prog10-1

(1)挿入 (2)探索 (3)表示 (0)終了:1
挿入するデータを入力してください。
番号:10
氏名:azuma

(1)挿入 (2)探索 (3)表示 (0)終了:1
挿入するデータを入力してください。
番号:30
氏名:sato

(1)挿入 (2)探索 (3)表示 (0)終了:1
挿入するデータを入力してください。
番号:20
氏名:shibata

(1)挿入 (2)探索 (3)表示 (0)終了:3
【一覧表】
番号:10 氏名:azuma
番号:30 氏名:sato
番号:20 氏名:shibata

(1)挿入 (2)探索 (3)表示 (0)終了:2
探索するデータを入力してください。
氏名:shibata
番号:20 氏名:shibata

(1)挿入 (2)探索 (3)表示 (0)終了:2
探索するデータを入力してください。
氏名:azuma
番号:10 氏名:azuma


(1)挿入 (2)探索 (3)表示 (0)終了:2
探索するデータを入力してください。
氏名:akiyama

(1)挿入 (2)探索 (3)表示 (0)終了:1
挿入するデータを入力してください。
番号:azuma
氏名:【エラー】azumaは既に登録されています。

(1)挿入 (2)探索 (3)表示 (0)終了:




演習10-2

- prog10-1.cを使って, 以下の操作を行って動作を確認せよ.
 - a) 自分の知人の名前を5人分挿入せよ
 - b) 5人分の名前を検索せよ
 - c) 未登録な名前を検索を行い, 失敗することを確認せよ
- 

演習10-3

- prog10-1.cにおいて, 関数PrintTreeは, キー値である氏名を昇順に表示する. 逆順に表示する以下の関数を作成し, 動作を確認せよ (prog10-1.c内に組み込め).

作成する関数:

- void PrintTreeR (BinNode *p); 

演習10-3: 実行例

(1)挿入 (2)探索 (3)表示 (4)逆表示 (0)終了:3

【一覧表】

番号:10 氏名:apple

番号:20 氏名:binary

番号:30 氏名:code

番号:40 氏名:dentsu

(1)挿入 (2)探索 (3)表示 (4)逆表示 (0)終了:4

【逆一覧表】

番号:40 氏名:dentsu

番号:30 氏名:code

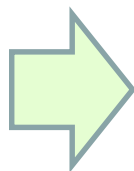
番号:20 氏名:binary

番号:10 氏名:apple

演習10-4

- prog10-1.cにおいて, 関数 FreeTreeを以下のように実現するとどうなるのか説明せよ.

```
void FreeTree( BinNode *P)
{
    if (p != NULL) {
        FreeTree( p->left );
        FreeTree( p->right );
        free(p);
    }
}
```



```
void FreeTree( BinNode *P)
{
    if (p != NULL) {
        FreeTree( p->left );
        free(p);
        FreeTree( p->right );
    }
}
```

演習10-5

- prog10-1.cにおいて, データを入力した時に, 現在のレベルとそれまでの最大レベルを表示せよ.



演習10-5: 実行例

> ./prog10-5

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:10

氏名:becky

[現在レベル = 0, 最大レベル = 0]

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:20

氏名:candy

[現在レベル = 1, 最大レベル = 1]

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:30

氏名:anny

[現在レベル = 1, 最大レベル = 1]

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

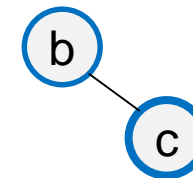
番号:40

氏名:elly

[現在レベル = 2, 最大レベル = 2]

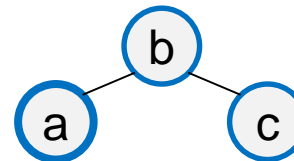


Level 0



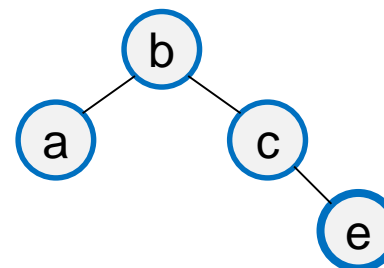
Level 0

Level 1



Level 0

Level 1



Level 0

Level 1

Level 2

演習10-6

- prog10-1.cでは、キー値を氏名としたが、この代わりに、キー値を番号として2分木を構築し、番号によって探索を行うプログラムを作成せよ.



演習10-6: 実行例

> ./prog10-6

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:30

氏名:andy

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:20

氏名:becky

(1)挿入 (2)探索 (3)表示 (0)終了:1

挿入するデータを入力してください。

番号:10

氏名:candy

(1)挿入 (2)探索 (3)表示 (0)終了:3

【一覧表】

番号:10 氏名:candy

番号:20 氏名:becky

番号:30 氏名:andy

(1)挿入 (2)探索 (3)表示 (0)終了:2

探索するデータを入力してください。

番号:20

番号:20 氏名:becky

(1)挿入 (2)探索 (3)表示 (0)終了:



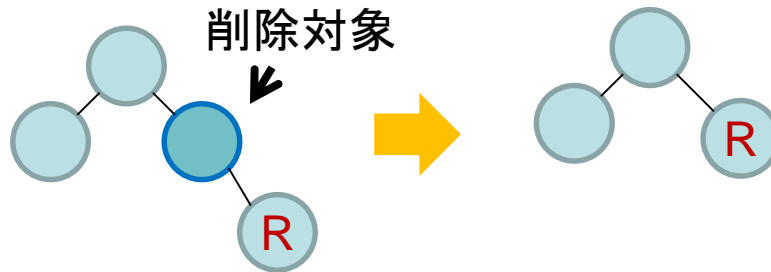
演習10-7

- prog10-1.cにおいて, ノードを削除する機能を追加せよ.

※ 難しい問題なので, 以降の解説をよく読んでください。

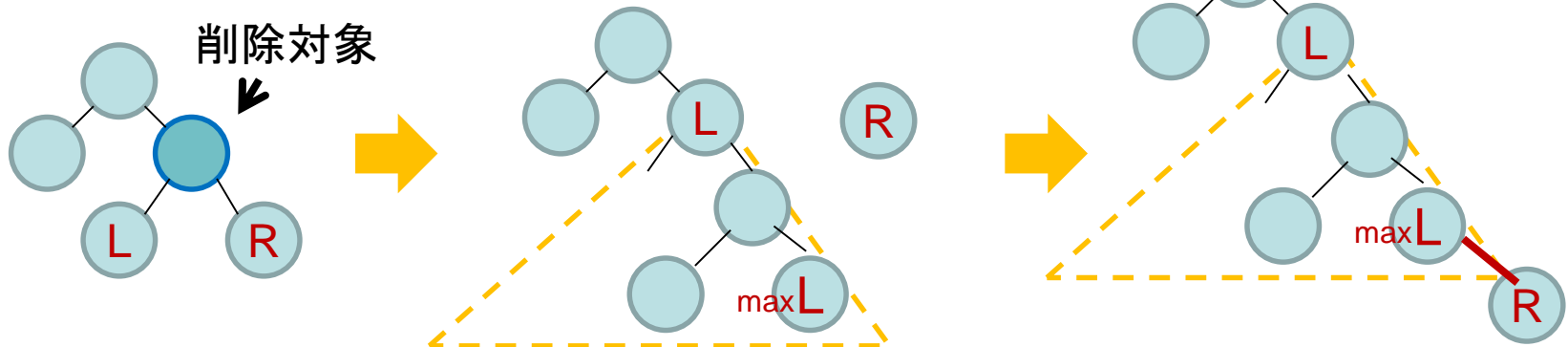
演習10-7: 考え方①

- 左部分木がNullな場合:



削除対象の代わりに、
対象の右部分木を接続すればOK

- 左部分木が存在する場合:



削除対象をその左部分木に置き換える
(ここまでは「左部分木がNullの場合」と同じ)

追加した左部分木内の最大ノード L_{\max}
(最も右側のノード)の右側にRノードを追加

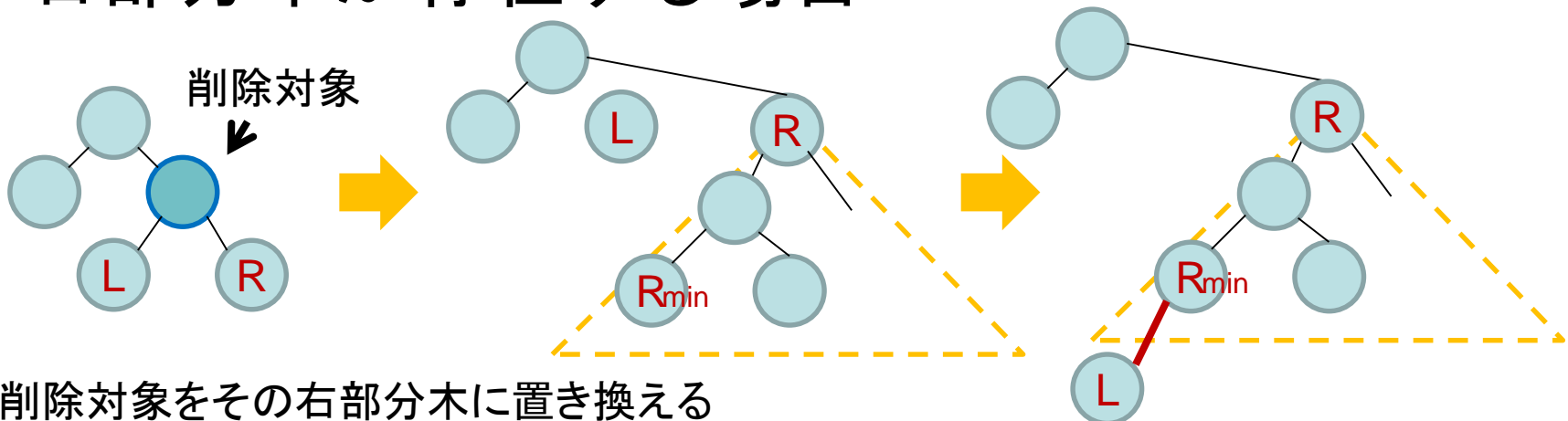
演習10-7: 考え方②

- 右部分木がNullな場合:



削除対象の代わりに,
対象の左部分木を接続すればOK

- 右部分木が存在する場合:

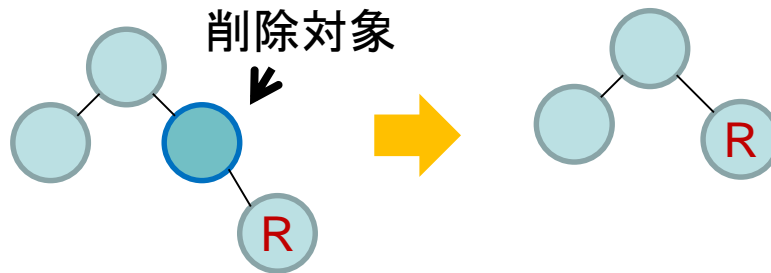


削除対象をその右部分木に置き換える

追加した右部分木内の最小ノード R_{min}
(最も左側のノード)の左側にLノードを追加

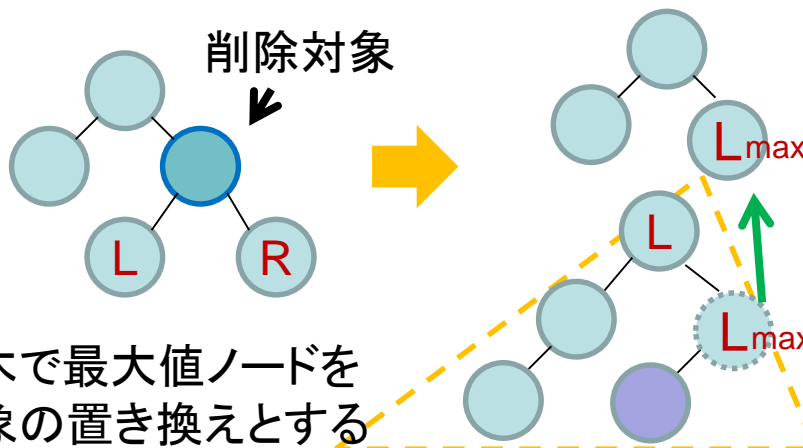
演習10-7: 考え方③

- 左部分木がNullな場合:

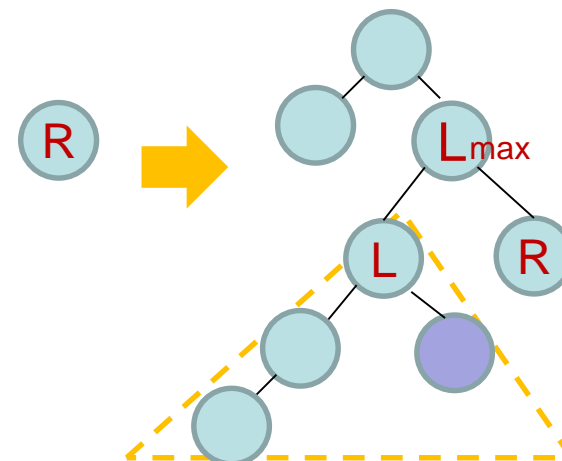


削除対象の代わりに、
対象の右部分木を接続すればOK

- 左部分木が存在する場合:



左部分木で最大値ノードを
削除対象の置き換えとする



L_{max}の右側に
右部分木Rを
繋げる

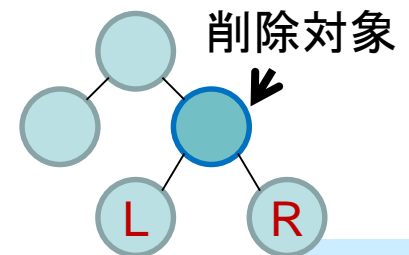



その他の方法

- 考え方①～③は一例
- 他にもあります
 - 例えば、①と②の混合タイプ等
- ①～③を通して典型的な考え方を学んでください


演習10-7: ヒント①

- 「考え方①」を元にしたプログラムについて解説します.
- まず, 削除対象を「探索」します.
 - 探索失敗の場合, 削除対象が見つからなかったということで削除も失敗
 - 探索成功の場合, そのノードの右部分木と左部分木をどう処理するかを考える





演習10-7: ヒント②

- 削除対象の左部分木がNullならば, 削除対象をその右部分木で置き換えて削除完了
 - そうでなければ,
 - 左部分木で削除対象を置き換える
 - 削除対象の左部分木の中で最大のノードを探す
※部分木内で最も右側に位置するノードを探す
→ノードの右部分木をたどり, 右部分木がNullになったらそこが最大ノード
 - 最大ノードの右側に削除対象の右部分木を繋げる
- 


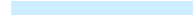


今回の演習内容

- 講義の復習：
 - 演習10-1～10-7（計7問）
- 提出課題：
 - 課題10-1～10-2（計2問）



最後に(1)

- 橋本が担当する回は今日が最後です.
 - しっかり復習してください. 講義で取り扱った演習や課題は, 暗記する位によく読んで, 理解してください.
- 
- 

最後に(2)

- 試験は、橋本を含めて3人の教員が、各担当分について作成します。
- 演習課題、提出課題など、講義で扱ったプログラムをよく復習してください。
- プログラム本体だけでなく、**アルゴリズムやその考え方**などもよく復習しておいてください。