データ構造とアルゴリズム
（第5回）

モバイルコンピューティング研究室
柴田史久

1

1

本日の講義内容

- 基本的なデータ構造（４）
  - 木構造とは
  - 順序木と無順序木
  - 二分木
  - 木のなぞり
  - 再帰
  - 木の実現

2

2

教科書 第6章（pp.157～142）

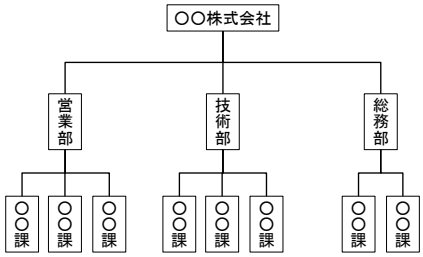
基本的なデータ構造(4)
木構造

3

3

木構造とは?

- 木（tree）は階層関係を表現するデータ構造
  - 階層関係：上下関係（親子関係）
  - Ex：会社の組織図



4

4

木構造の例

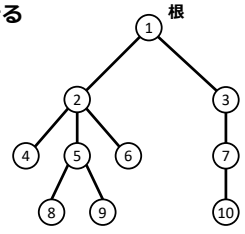
- 本の構成
  - 章（chapter），節（section），小節（subsection）
- コンピュータのフォルダ構造
  - 階層ディレクトリ
- 数式
- 計算の過程

5

5

木に関する用語

- 節（node）
  - 節点，ノード，頂点（vertex）
  - 節には名前（番号）をつけて識別
  - 節には値を持たせることができる
    - ラベル
- 根（root）
  - 一番上の節（右図の節1）
- エッジ（edge）
  - 枝，辺
  - 節同士を結ぶもの

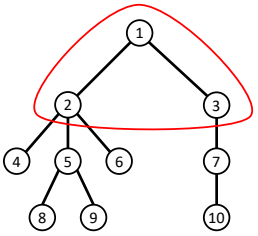


6

6

木構造(1)

- 階層関係に従って節を配置
  - 例：親（parent）：節1，子（child）：節2，節3
- 根には親がない
- 節はたかだか1つの親を持つ
- 節は任意の個数の子を持つ
- 子のない節は葉（leaf）
  - 端節（terminal node）
  - 外部節（external node）
- 葉でない節
  - 非終端節（non-terminal node）
  - 枝節（branch node）
  - 内部節（internal node）

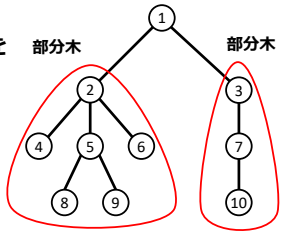


7

7

木構造(2)

- ある節の子を根とする木は部分木（subtree）
  - 節2を根とする部分木，節3を根とする部分木，etc.
- 同じ親を持つ節同士を兄弟（sibling）
- 節xの先祖（ancestor）
  - ある節xから根に向かって節をたどるときに出現する節
- 節xの子孫（descendant）
  - ある節xから子の関係をたどって到達できる節

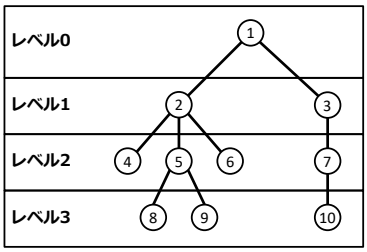


8

8

木構造(3)

- レベル（level，水準）
  - 木の根はレベル0
  - 根からの距離でレベルを定義（＝何回エッジをたどるか）



9

9

木の定義(教科書 p.162)

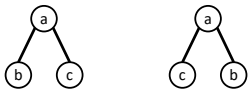
- 1つの節は，それ自体が木である．この木に含まれるただ1つの節が，この木の根である．
- k個の木T_1～T_kがありそれぞれの根を節n_1～n_kとする．節nを節n_1～n_kの親にすると，節nを根とする新しい木Tが得られる．
  このとき，木T_1～T_kは，木Tの部分木であるという．
  部分木の根n_1～n_kは，節nの子であるという．
- 節をまったくもたない木は空の木（null tree）

10

10

順序木と無順序木

- 順序木（ordered tree）
  - 兄弟間で順序付けをする木
- 無順序木（unordered tree）
  - 兄弟間で順序付けをしない木
- 順序木なら両者は別の木．無順序木なら同じ木．



11

11

二分木(binary tree)(1)

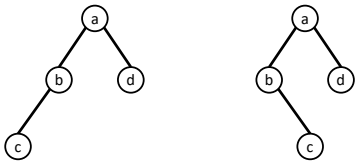
- 木構造の一種
- 二分木の定義
  - 空の木は二分木
  - 次のいずれかの条件を満たす節のみからなる木は二分木
    - 子をもたない
    - 左の子のみをもつ
    - 右の子のみをもつ
    - 左右2個の子をもつ

12

12

二分木(2)

- 二分木の節はたかだか2個の子しか持たない
- 左の子と右の子を区別
  - 左部分木：左の子を根とする部分木
  - 右部分木：右の子を根とする部分木
- 異なる二分木



13

13

二分木の実装

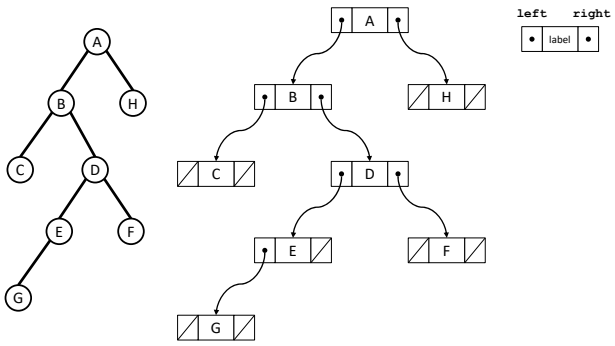
- 二分木の節をクラスで実現
- BinaryTreeNodeクラス
  - left：左部分木（左の子）
  - right：右部分木（右の子）
  - label：この節のラベル　←　適切なクラス／データ型に

```
class BinaryTreeNode{
  BinaryTreeNode left ;  // 左部分木(左の子)
  BinaryTreeNode right ; // 右部分木(右の子)
  Object         label ; // この節のラベル
}
```

14

14

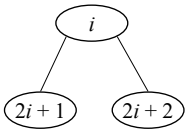
BinaryTreeNodeによる木の表現



15

二分木を配列で表現

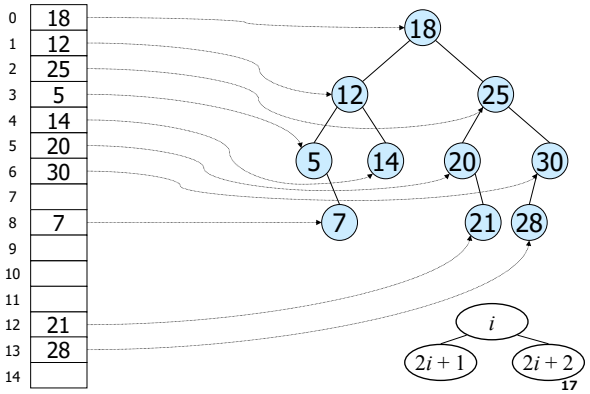
- 配列にデータのみを格納し，位置で親子関係を表現
  - 参照を使った接続はなくなる
  - 注：配列の添字が0から始まる場合の例
- ノード i の子ノード
  - 左の子：$2i+1$
  - 右の子：$2i+2$
- ノード j の親ノード
  - ノード $(j-1)/2$



16

16

配列で表現した木



17

17

木のなぞり

- なぞる（traverse，名詞形はtraversal）
  - 木の節を系統的に1つ残らず調べて，各節に1回だけ立ち寄る操作
  - 立ち寄った順に節を順序付け可能
  - 走査（scan）とも
- なぞり方は３通り
  - 行きがけ順（preorder）：前順走査
  - 通りがけ順（inorder）：間順走査
  - 帰りがけ順（postorder）：後順走査
- どの時点で節に立ち寄るかがポイント
- 再帰的に定義

18

18

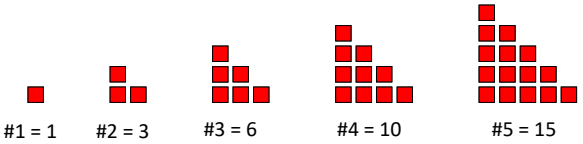
再帰(recursive)

- 作業単位である**自分自身を自分自身で呼び出して**，再び同じ処理を繰り返し行うテクニック

19

19

例題:三角数(1)



- #6のときの値はいくら？
- #20なら？

20

20

例題:三角数(2)

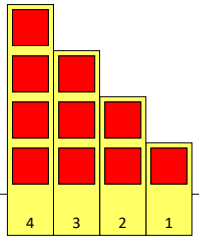
- n番目の項をループを使って求める

```
public static int triangle(int n) {

  int total = 0 ;

  while (n > 0) {
    total = total + n ;
    --n ;
  }

  return total ;
}
```



21

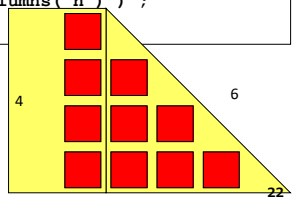
21

例題:三角数(3)

- n番目の項を再帰を使って求める
- n番目の項は以下の２つの和
  - 最初の（最も背が高い）桁の数 = n
  - そのほかのすべての桁の数

```
public static int triangle(int n) {
  return ( n + sumRemainingColumns( n ) ) ;
}
```

sumRemainingColumns は
どうやって書けばいい？



22

22

例題:三角数(4)

```
public static int triangle(int n) {
  return ( n + sumAllColumns( n-1 ) ) ;
}
```

```
public static int triangle(int n) {
  return ( n + triangle( n-1 ) ) ;
}
```

- 他人まかせ
  - このままでは無限に責任転嫁してしまう
  - どこかで誰かが自分の答えに責任を持つ必要がある

23

23

例題:三角数(5)　再帰の完成

- 他人まかせの終点
  - １番目の人は自分の答えを知っている　（＝１）

```
public static int triangle(int n) {
  if (n == 1) {
    return 1 ;
  } else {
    return ( n + triangle( n-1 ) ) ;
  }
}
```

24

24

再帰の動作

- 再帰の動作を確認

```
public static int triangle(int n) {
  System.out.println("Entering: n=" + n) ;
  if (n == 1) {
    System.out.println("Returning [1]") ;
    return 1 ;
  } else {
    int temp = n + triangle( n-1 ) ;
    System.out.println("Returning " + temp) ;
    return temp ;
  }
}
```

25

25

再帰アルゴリズムのパターン

- 処理手順が自身を用いて定義されているもの

```
recursive (n) {
    if (自明なケース) {
        自明なケースの処理 ;      // 終了条件
    } else {
        recursive (m) ;          // m < n
        (処理) ;
    }
}
```

  - 自身の引数より小さな引数で自身を呼び出す
    - サブゴール
  - 自明なケースの処理が存在
  - 表面的にループが出現しない

26

26

再帰の特徴

- 自分自身を呼び出す
- 自分自身を呼び出すのは，自分に与えられた問題より一回り小さな問題を解くためである
- 最後には，自身で解を求められるような，極小の問題に到達する．そこでは再帰呼び出しをせずに解を出す．

27

27

再帰のメリット

- 再帰を利用するメリットは何か
  - 再帰を利用するプログラムは，繰り返しやスタックなどを使うことで再帰を利用せずにプログラムすることも可能
  - 再帰を用いる利点は解法を非常にスマートに表現できる点
- 再帰は，数学的帰納法のプログラミング版
  - 何かをそれ自身によって定義する
  - $\mathrm{tri}(n) = 1 \quad (\text{if } n = 1)$
  - $\mathrm{tri}(n) = n + \mathrm{tri}(n-1) \quad (\text{if } n > 1)$

28

28

3通りのなぞり方

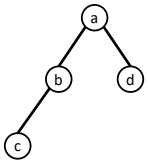
- 行きがけ順　　赤字部分が再帰
  1. 節に立ち寄る
  2. 左部分木を（行きがけ順で）なぞる
  3. 右部分木を（行きがけ順で）なぞる
- 通りがけ順
  1. 左部分木を（通りがけ順で）なぞる
  2. 節に立ち寄る
  3. 右部分木を（通りがけ順で）なぞる
- 帰りがけ順
  1. 左部分木を（帰りがけ順で）なぞる
  2. 右部分木を（帰りがけ順で）なぞる
  3. 節に立ち寄る

29

29

行きがけ順のなぞり(1)

1. 節 a に立ち寄る
2. （節 b を根とする）左部分木を行きがけ順でなぞる
3. （節 d を根とする）右部分木を行きがけ順でなぞる

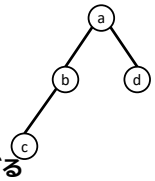


30

30

行きがけ順のなぞり(2)

2. （節 b を根とする）左部分木を
   行きがけ順でなぞる
   1. 節 b に立ち寄る
   2. （節 c を根とする）左部分木を
      行きがけ順でなぞる
   3. 右部分木（空の木）を行きがけ順でなぞる

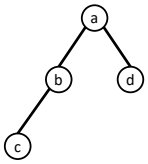


31

31

行きがけ順のなぞり(3)

2. （節 b を根とする）左部分木を
   行きがけ順でなぞる
   2. （節 c を根とする）左部分木を
      行きがけ順でなぞる
      1. 節 c に立ち寄る
      2. 左部分木（空の木）を行きがけ順でなぞる
      3. 右部分木（空の木）を行きがけ順でなぞる

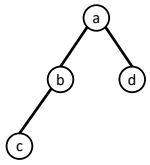


32

32

行きがけ順のなぞり(4)

3. （節 d を根とする）右部分木を
   行きがけ順でなぞる
   1. 節 d に立ち寄る
   2. 左部分木（空の木）を
      行きがけ順でなぞる
   3. 右部分木（空の木）を
      行きがけ順でなぞる

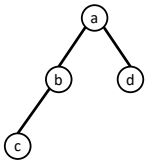


33

33

行きがけ順のなぞり(5)

1. 節 a に立ち寄る
2. （左部分木）
   1. 節 b に立ち寄る
   2. （左部分木）
      1. 節 c に立ち寄る
3. （右部分木）
   1. 節 d に立ち寄る

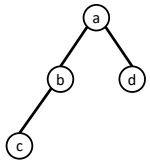


34

34

通りがけ順のなぞり

1. （左部分木）
   1. （左部分木）
      1. （左部分木：空の木）
      2. 節 c に立ち寄る
      3. （右部分木：空の木）
   2. 節 b に立ち寄る
   3. （右部分木：空の木）
2. 節 a に立ち寄る
3. （右部分木）
   1. （左部分木：空の木）
   2. 節 d に立ち寄る
   3. （右部分木：空の木）

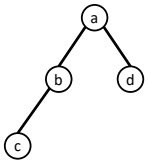


35

35

帰りがけ順のなぞり

1. （左部分木）
   1. （左部分木）
      1. （左部分木：空の木）
      2. （右部分木：空の木）
      3. 節 c に立ち寄る
   2. （右部分木：空の木）
   3. 節 b に立ち寄る
2. （右部分木）
   1. （左部分木：空の木）
   2. （右部分木：空の木）
   3. 節 d に立ち寄る
3. 節 a に立ち寄る



36

36

実装:行きがけ順のなぞり

- traversePreorderメソッド

```
static void traversePreorder(BinaryTreeNode p)
{
  if (p == null) {
    return ;
  }
  System.out.println("節" + p.label + "に立ち寄りました");
  traversePreorder(p.left);
  traversePreorder(p.right);
}
```

終了条件

再帰呼び出し

```
class BinaryTreeNode{
  BinaryTreeNode left ;  // 左部分木(左の子)
  BinaryTreeNode right ; // 右部分木(右の子)
  String         label ; // この節のラベル
}
```

37

37

実装:通りがけ順のなぞり

- traverseInorderメソッド

```
static void traverseInorder(BinaryTreeNode p)
{
  if (p == null) {
    return ;
  }
  traverseInorder(p.left);
  System.out.println("節" + p.label + "に立ち寄りました");
  traverseInorder(p.right);
}
```

終了条件

再帰呼び出し

```
class BinaryTreeNode{
  BinaryTreeNode left ;  // 左部分木(左の子)
  BinaryTreeNode right ; // 右部分木(右の子)
  String         label ; // この節のラベル
}
```

38

38

実装:帰りがけ順のなぞり

- traversePostorderメソッド

```
static void traversePostorder(BinaryTreeNode p)
{
  if (p == null) {
    return ;
  }
  traversePostorder(p.left);
  traversePostorder(p.right);
  System.out.println("節" + p.label + "に立ち寄りました");
}
```

終了条件

再帰呼び出し

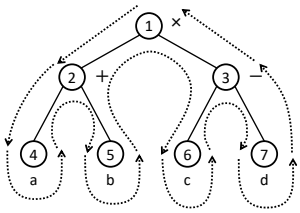
```
class BinaryTreeNode{
  BinaryTreeNode left ;  // 左部分木(左の子)
  BinaryTreeNode right ; // 右部分木(右の子)
  String         label ; // この節のラベル
}
```

39

39

数式の木

- 数式:$(a + b) \times (c - d)$



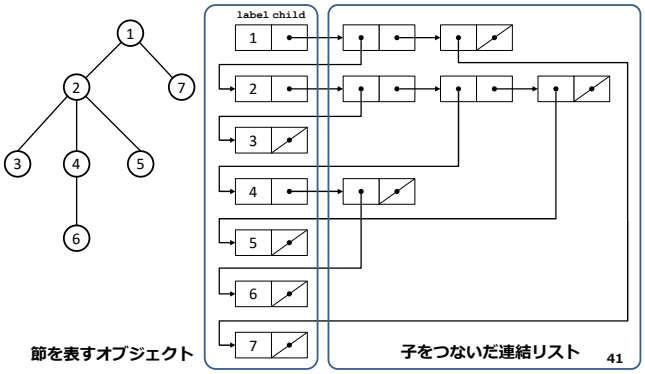
- 通りがけ順
  - 中置記法(通常のもの)
  - 演算子に優先順位
  - 括弧が必要
- 行きがけ順
  - 前置記法
- 帰りがけ順
  - 後置記法
  - 逆ポーランド記法(reverse polish notation; RPN)

40

40

(一般の)木の実現(1)

- 任意個の子を管理する連結リストを導入

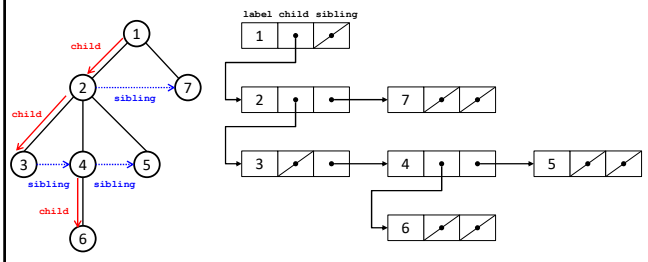


41

41

(一般の)木の実現(2)

- 節に子と兄弟へのリンクを持たせる
  - 二分木による表現



42

42

木の実現法の欠点

- ある節の親を知る手段がない
- ある節の兄（左隣の兄弟）を知る手段がない

43

43

まとめ

- 木構造とは
- 順序木と無順序木
- 二分木
- 木のなぞり
- 再帰
- 木の実現

44

44

参考文献

- 定本 Javaプログラマのための
  アルゴリズムとデータ構造（近藤嘉雪）
- 新・明解 Javaで学ぶ
  アルゴリズムとデータ構造（柴田望洋）
- 岩波講座ソフトウェア科学 3
  アルゴリズムとデータ構造（石畑清）
- Javaで学ぶアルゴリズムとデータ構造
  Robert Lafore（著）・岩谷 宏（翻訳）
- Java アルゴリズム+データ構造完全制覇
  オングス (著)・杉山 貴章・後藤 大地 (監修)

45

45