基本情報技術概論 I（第5回）

アルゴリズム と データ構造

埼玉大学 理工学研究科
堀山 貴史

1

前回の復習：アルゴリズム と データ構造

- 基本的なデータ構造
  - 配列 (array)
  - リスト (list)
- その他のデータ構造
  - スタック (stack)
  - 待ち行列 (queue, キュー)
  - 木 (tree)、2分木 (binary tree)
    - 2分探索木、ヒープ

} この後、すぐ

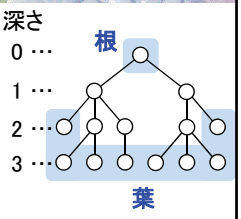
- アルゴリズム
  - 線形探索　先頭から順番に調べる　　計算量 O(n)
  - 2分探索　探索範囲の中央を調べる　　O(log n)

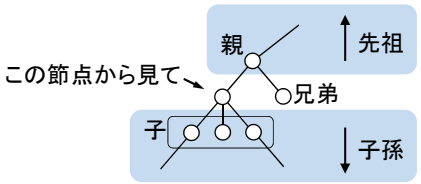
2

データ構造：木 (tree)

- **節点** と **辺**（枝 ともいう）からなり
  以下の 1、2 を満たすもの

1. すべての節点が連結である
2. サイクルを持たない



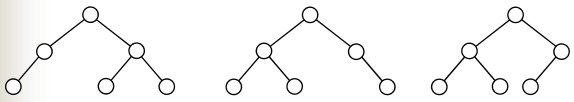


3

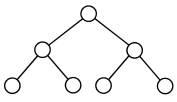
データ構造：

- 各節点の子は、高々2つ
- **右の子** と **左の子** は、区別する

例）下の3つは、どれも異なる2分木



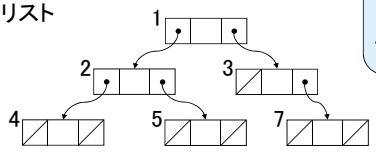
- 完全2分木

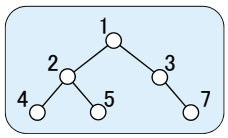


4

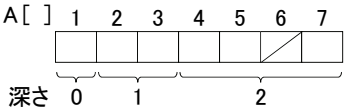
2分木の実現法

- リスト





- 配列



節点 *i* の
- 左の子
- 右の子
- 親

5

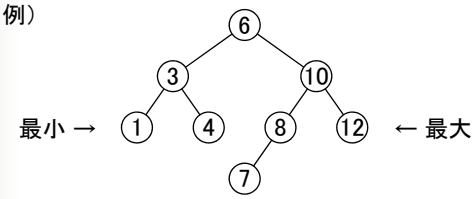
2分探索木

6

データ構造：

- 2分木の各節点に、データを格納
- 節点 v のデータは
  - (1) v のどの左の子孫よりも大きい
  - (2) v の 〃 右 〃 小さい

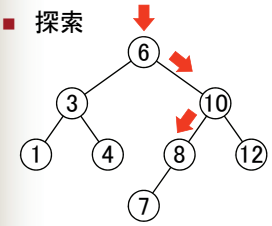
例）



7

2分探索木の操作

- 探索

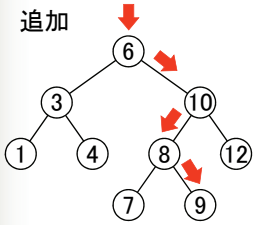


- 根から出発
- 探索キー = 節点のキー … 探索成功
- 探索キー < 節点のキー … 左の部分木へ
- 探索キー > 節点のキー … 右の部分木へ

8

2分探索木の操作

- 追加

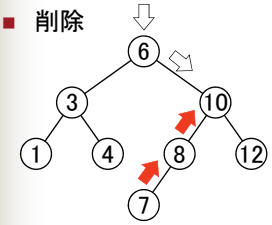


- 探索と同様にして、木をたどる
- 木から飛び出したところに、新しい節点を加える

9

2分探索木の操作

- 削除



- 削除するキーを探索して削除
- 2分探索木の条件に合うように移動（子孫も同様）
  左部分木の最大値を移動
  or 右部分木の最小値を移動

10

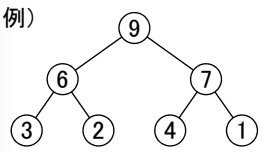
ヒープ

11

データ構造：

- 完全2分木の各節点に、データを格納
- 親と子の大小関係をそろえる（ヒープ ソートに利用）
  - 親 > 子 なら、根が最大
  - 親 < 子 なら、根が最小

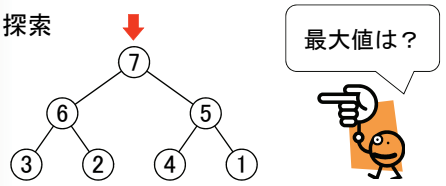
例）



12

ヒープの操作

- 探索

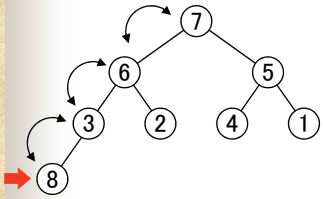


ヒープ ソート
- 値を順に**追加**
- 最大値を順に取り出す（最大値を**探索** & **削除**）

13

ヒープの操作

- 追加

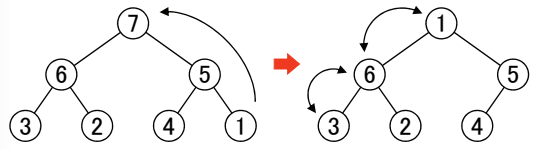


- ヒープの**最後**（**深さ最大**、**左詰め**）に新しい節点を追加
- **ヒープの維持**

14

ヒープの操作

- 削除



- ヒープの**最後**を削除する節点に追加移動
- **ヒープの維持**

15

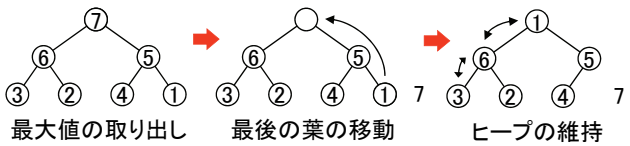
データ構造：木（tree）まとめ

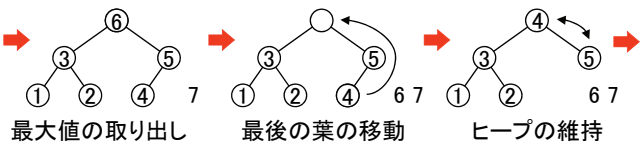
- 2分木
  - **2分探索木**　左の子孫 < 親 < 右の子孫
  - **ヒープ**　親 > 子　（または　親 < 子）
  - バランス木　部分木の節点数や深さにバランスを
    - **完全バランス木**
      左右の部分木の節点数の差が1以内
    - **AVL木**
      左右の部分木の深さの差が1以内
- 多分木
  - **B木**　（各節点が最大 m 個の子をもつバランス木）

16

アルゴリズム：ヒープソート（heap sort）

- ヒープの最大値の取り出し、ヒープの維持を繰り返す

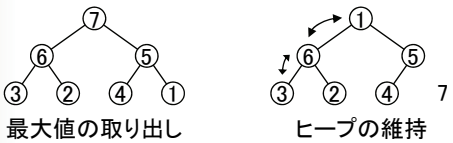




17

アルゴリズム：ヒープソート（heap sort）

- ヒープの最大値の取り出し、ヒープの維持を繰り返す



計算量 O(1)　　計算量 O(log n)

n 回　繰り返し　…トータルの計算量
O(n log n)

18