大阪電気通信大学 情報通信工学部 光システム工学科 2年次配当科目

# コンピュータアルゴリズム

# いろいろなデータ構造

第3講: 平成20年10月17日 (金) 4限 E252教室

中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

## 第 1 講の復習

- アルゴリズムの定義
  - 入力と出力
  - 正当性, 決定性, 有限性, 停止性
- ユークリッドの互除法
- フローチャートの描き方
- 擬似言語の書き方



## 第 2 講の復習

- アルゴリズムの評価は時間計算量で行う
  - 領域計算量もある
- 計算量の評価にはオーダー記法を使う
  - 並んでいる計算量は足し算
  - 繰り返しに含まれる計算量は掛け算
  - 係数は省略する
- 多項式オーダーと指数オーダー
  - 指数オーダーのアルゴリズムは使い物にならない
- 再帰プログラム



## 今日の講義の内容

- データ構造(抽象データ型)の紹介
  - 配列
  - リスト
  - キュー(待ち行列)
  - スタック
  - 木



## データ構造とは

- データを計算機内部でどのように表現するか
  - データのメモリ上の表現方法
  - Data Structure
- 良いアルゴリズムを作成するには, 問題に適したデータ構造が必要
- 抽象データ型
  - データ構造をその表現と扱う手続きの集合で一まとまりで表現したもの



## 列(Sequence)

- 同じ種類のデータが 1 列に並んだもの
- 中身のデータと並び方の両方に意味がある
  - {1, 2, 3} と {1, 3, 2} は違う列
- 列は抽象的な概念 = 抽象データ型
- プログラムでは何らかのデータ構造で表現
  - 配列, リスト
- 列に対する操作を限定したデータ構造も存在
  - キュー, スタック

## 配列(Array)

- 列を表現するデータ構造の 1 つ
- 同じ型のデータを決まった個数だけ並べた構造
- C 言語での配列の作成
  - `int a[100] ;` 整数型, 100個の配列
- 番号(添字)を指定して要素を参照
- 参照操作の計算量は
- データの挿入・削除は最悪
  - 要素を移動する操作が必要なため



## 配列(Array)の操作

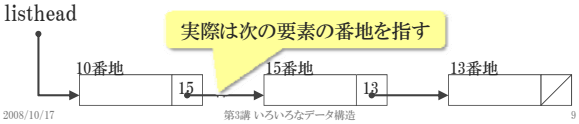
| 添字 | データ |
|---|---|
| 1 | 3 |
| 2 | 8 |
| 3 | 2 |
| 4 | 7 |
| 5 | 15 |
| 6 | 12 |
| 7 | 6 |
| 8 | 9 |
| 9 | 23 |
| 10 | 11 |

- 参照
  - 直接参照可能
    - `a[3]`, `a[7]`
  - $O(1)$
- 挿入, 削除
  - `a[4]` と `a[5]` の間にデータを挿入するには, `a[5]` ～ `a[10]` をそれぞれ `a[6]` ～ `a[11]` にずらす必要がある
  - 削除も同様
  - $O(n)$



## リスト(list)

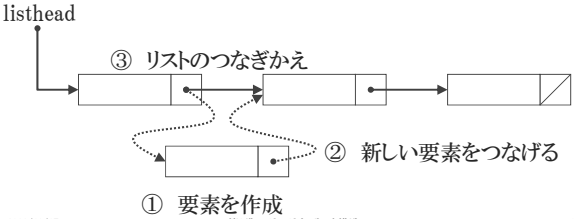
- リスト: list, linked list, linear list (線形リスト)
- データを入れた箱をポインタでつないだもの
- 2 個の箱からなり, 1 つは要素のデータ, もう 1 つは次の要素を指すポインタをいれる
- ランダムアクセスの機能がない
- $k$ 番目の要素を参照する計算量は $O(n)$
  - 先頭から辿らないといけない






## リスト(list)の操作

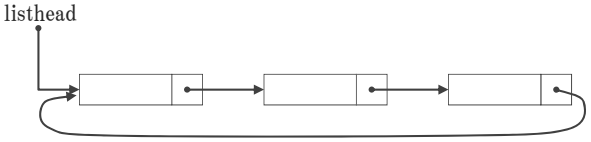
- リストはデータの挿入・削除が容易
  - つなぎかえるだけで良い
- データの挿入・削除の計算量は $O(1)$






## そのほかのリスト (1)

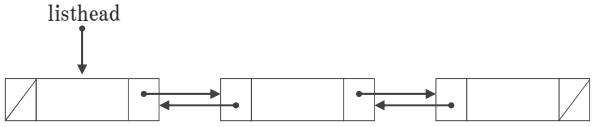
- 環状リスト:
  Circular list, 循環リスト, 巡回リスト
- 最後の要素が先頭の要素を指す
- 列の端に意味がない場合に有効






## そのほかのリスト (2)

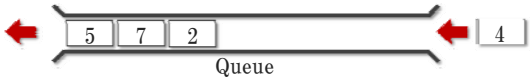
- 双方向リスト: doubly-linked list
- 各要素に 2 つのポインタを用意し, 前後の要素を指すようにしたもの




- 双方向かつ環状リストも可能

## キュー(待ち行列, queue)

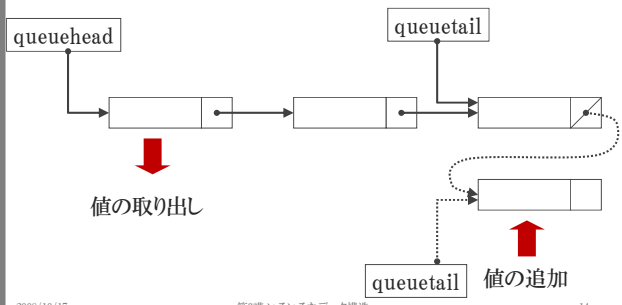
- データの追加を列の一方の端から行い，取り出しをもう一方の端から行う列
- 先入れ先出し：FIFO (First-In-First-Out)
- 必要な操作は 4 つ
  - ClearQueue：待ち行列を初期化
  - EnterQueue：待ち行列に値を追加
  - RemoveQueue：待ち行列から値を取り出す
  - EmptyQueue：待ち行列が空かどうかを判定






## キュー(queue)の実現

- リストによる実現





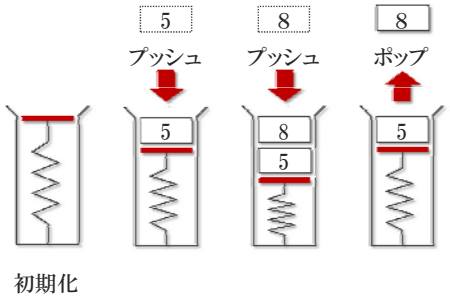

## スタック(stack)

- スタック：Stack, Push-down Stack
- データの追加・取出しを一方の端だけで行う列
- 後入れ先出し：LIFO (Last-In-First-Out)
- 必要な操作は 3 つ
  - ClearStack：スタックを初期化
  - Push：スタックに値を追加．押込み，プッシュ，プッシュダウン(Push Down)
  - Pop：スタックから値を取り出す．跳ね上げ，ポップ，ポップアップ(Pop Up)



## スタック(stack)の動作

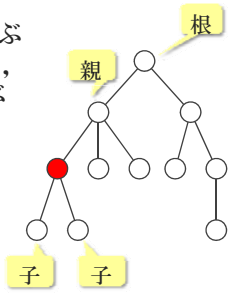
- 具体例：書類の山






## 木(Tree)

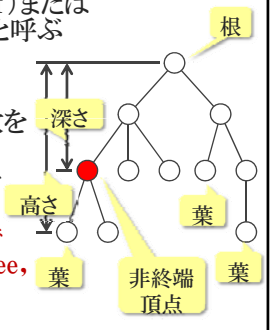
- 頂点(Vertex, Node, 節点)と枝(Branch Edge, Arc, 辺)から構成される
- 一番上の頂点を根(Root)と呼ぶ
- 枝の上側の頂点を親(Parent)，下側の頂点を子(Child)と呼ぶ
  - ある頂点から見て親，親の親などをまとめて祖先(Ancestor)と呼ぶ
  - ある頂点から見て子，子の子などをまとめて子孫(Descendant)と呼ぶ





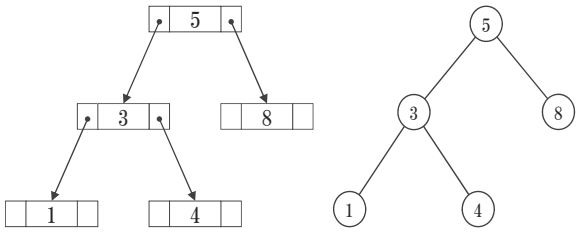

## 木(Tree)

- 子を持たない頂点を葉(Leaf)または終端頂点(Terminal Node)と呼ぶ
- 子を持つ頂点を非終端頂点(Nonterminal Node)と呼ぶ
- 根からある頂点までの枝の数を深さ(Depth)と呼ぶ
- 根から最も遠い頂点の深さを木の高さ(Height)と呼ぶ
- 各頂点の子の数が高々 2 である木を 2 分木(Binary Tree, 2 進木)と呼ぶ



## 木(Tree)の実現

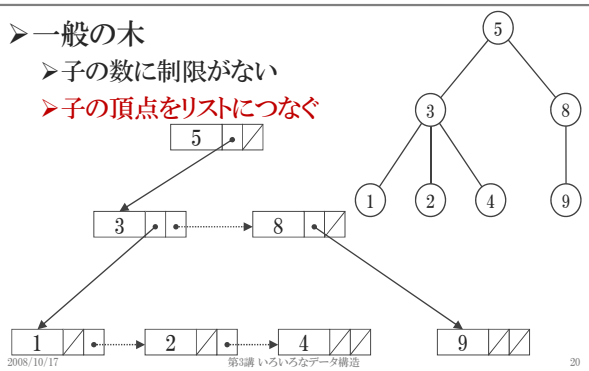
- 2 分木の場合
  - 2 つの子を指すポインタとデータをいれる箱で実現






## 木(Tree)の実現

- 一般の木
  - 子の数に制限がない
  - 子の頂点をリストにつなぐ






## 第 3 講のまとめ

- アルゴリズムとデータ構造
  - アルゴリズムに適したデータ構造の選択が必要
- 基本的なデータ構造(抽象データ型)
  - 配列
  - リスト
  - キュー(待ち行列)
  - スタック
  - 木