アルゴリズムとデータ構造

http://www-ui.is.s.u-tokyo.ac.jp/~takeo

五十嵐 健夫

takeo@acm.org

目標

効率のよいプログラムを書くため
基本的な知識・技法を学ぶ

効率＝実行時間とメモリ使用量

コンピュータサイエンスの基礎。

具体的内容

基本的なアルゴリズムとデータ構造を学ぶ

ソート、グラフ、検索、など

計算量の解析について学ぶ

計算量の意味、その計算方法、など

進め方

教科書に沿う

データ構造とアルゴリズム　五十嵐健夫
数理工学社

基本的に教科書が理解できればＯＫ。

成績

毎回出席を取る。

期末テスト（1/１６ 予定）

ネット上に過去問あり

スケジュール（仮）

10/10 （休講）
10/17 アルゴリズムと計算量 基本的なデータ構造
擬似言語、実行時間, 列, スタック, 待ち行列,木
10/24 集合の表現　ヒープ、2分探索木、平衡木
10/31 集合の表現　ハッシュ、集合群
11/7 ソート バブルソート, 内挿ソート, 選択ソート, クイックソート
11/14 （休講）
11/21 ソート ヒープソート, ビンソート, 基数ソート, マージソート
11/28 有向グラフ　ダイクストラ, フロイド, DAG, 強成分
12/5 （休講）
12/12 無向グラフ　プリム, クラスカル, 関節点, 最大マッチング
12/19 文字列　KMPアルゴリズム, BMアルゴリズム
1/9 設計法　欲張り、α－β枝刈り、分岐制約探索法、
1/16 [テスト]

演習との関係

アルゴリズムとデータ構造演習
水曜日３限

Ｃ言語による講義の演習
＋
関数型言語の勉強

自己紹介

五十嵐　健夫　　情報科学科　助教授

専門：　ユーザインタフェース
　　インタラクティブＣＧ

電子ホワイトボード、電子カルテ
３次元モデリング、アニメーション
音声によるインタフェース　など

アルゴリズムとは

問題を解く手順

1) **問題を定式化（モデル化）する**
2) **解法をアルゴリズムとして記述する**
3) **アルゴリズムにしたがって問題を解く**

定義:

「明瞭な意味を持ち、有限時間内の有限な計算で実行できるような命令を有限個並べた形で記述される問題の解法」

アルゴリズムとは

段階的詳細化の例（ユークリッドの互除法）

文章で書いたアルゴリズム
擬似言語のプログラム
プログラミング言語による記述

プログラムの実行時間

2つの目標

わかりやすい。構造がシンプルである。
実行時間が早く、メモリを消費しない。

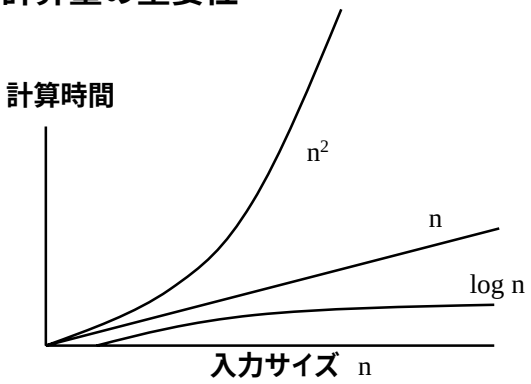
実行時間を決める要素

入力データの性質・大きさ
コンパイラの質
ハードウェアの性質
アルゴリズムの計算量

アルゴリズムによって計算時間が変わる例

線形探索と2分探索　n vs $\log_2 n$

計算量の重要性



計算量の重要性

n=100のときの時間を1[sec]として計算 ($O(2^n)$を除く)

計算量 \ n	10	50	100	500	1000	10000
O(1)	1	1	1	1	1	1
O(log n)	0.5	0.85	1	1.35	1.5	2
O(n)	0.1	0.5	1	5	10	100
O(n log n)	0.05	0.42	1	6.75	15	200
$O(n^2)$	0.01	025	1	25	100	2.78[h]
$O(2^n)$	0.001	34.8[y]	–	–	–	–

アルゴリズムの選択が重要（秒ー時間ー年）。
ハードウェア・コンパイラ・チューニングなどは小手先。

オーダー記法

「問題のサイズnに対して、予測される計算量の上界値を、<u>定数部分を省略</u>して表現する方法」

正確には、
「アルゴリズムの実行時間がO(f(n))である」とは
「正の定数c, n_0が存在して、n_0以上のnに対しては
$T(n) <= c\ f(n)$となる。」
ただし T(n)は大きさnの入力のプログラムの実行時間

（Ωは下界）

オーダー記法

例： 線形探索 $T(n) = an + b$ …. O(n)
2分探索 $T(n) = a \log n + b$ … O(log n)

オーダー記法

$O(1)<O(\log n)<O(n^a)<O(n \log n)<O(n^b)<O(\alpha^n)<O(n!)$

$0<a\leq 1,\ 1<b,\ \alpha>0$

プログラムの実行時間

和と積の法則

$T_1(n) + T_2(n)$ … $O(\max(f_1(n), f_2(n))$
$T_1(n) \times T_2(n)$ … $O(f_1(n) \times f_2(n))$

解析は難しいことも多い。

いくつかの規則

一連の文は和の公式 ＝ 最も遅い部分に依存
if 文は、長い方に依存
ループは、ループの回数と最長の内部実行時間の積
再帰手続き＝再帰方程式を解く

アルゴリズムの選択の注意点

使用回数	多い場合にはオーダーに注意
入力サイズ	大きい場合にはオーダーに注意
保守	保守が必要なら読みやすさ優先
メモリ	外部記憶が使えるか
安定性、精度	数値アルゴリズムで重要

よいプログラミングの習慣

計画的に設計する。 段階的詳細化。

オーダーを意識する。

カプセル化・モジュール化する。

既存プログラムを活用する。

汎用性のある・応用の利くコードを書く。

まとめ

講義の進め方
アルゴリズムとは
モデル化と段階的詳細化
計算量の話 オーダー記法

アルゴリズムとデータ構造

第2回　基本的なデータ構造

http://www-ui.is.s.u-tokyo.ac.jp/~takeo/course/

五十嵐 健夫

takeo@acm.org

前回の内容

イントロダクション
モデル化の例
計算量（OとΩ）

今回の内容

リスト
スタック
待ち行列
木

抽象データ型としての「列」

$a_1, a_2, a_3, \ldots, a_{1n}$　線形順序

insert, indexOf, get, remove, next, prev, clear, first, print, end

列の実現

配列による実現

○ ランダムアクセス　× 挿入と削除

ポインタによる実現（通常のリスト）

× ランダムアクセス　○ 挿入と削除

配列

○ ランダムアクセス　× 挿入と削除

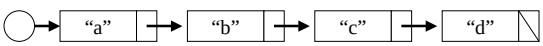
String[] labels = {“a”,”b”,”c” ,”d”}

“a”
“b”
“c”
”d”

リスト

× ランダムアクセス　○ 挿入と削除

```
LinkedList labels = new LinkedList();
labels.add(“a”);
labels.add(“b”);
labels.add(“c”);
labels.add(“d”);
```



スタック

clear, pop, push, empty

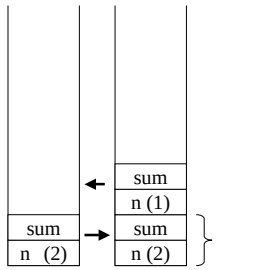
例 (再帰呼び出し)

実装　(ポインタと配列)

スタックと再帰呼び出し

$$\sum_{i=1}^{n} i^2$$

```
int foo(int n){
   if (n == 1) return 1
   int sum = foo(n-1)+n*n;
   return sum;
}
```



活動レコード

待ち行列 (Queue)

clear, front, enqueue,
dequeue, empty

例 (イベントキュー、データ転送)

実装　(ポインタと循環配列)

木 (Tree)

Insert, delete, member, etc.

階層構造を表す (例：住所)

実装　(ポインタ)

まとめ

配列
リスト
スタック
待ち行列
木