情報科学概論 I
アルゴリズムと計算量

亀山 幸義
http://logic.cs.tsukuba.ac.jp/~kam

亀山担当分の話題

アルゴリズムと計算量

- Fibonacci数列の計算を例にとり、アルゴリズムと計算量とは何か、具体的に学ぶ。
- 良いアルゴリズムの設計例として、整列(ソーティング)のアルゴリズムを学ぶ。

Fibonacci数(1)

$$\mathrm{Fib}(n) = \begin{cases} 1 & \text{if } n = 1, 2 \\ \mathrm{Fib}(n-2) + \mathrm{Fib}(n-1) & \text{if } n > 2 \end{cases}$$

Fib(1) = Fib(2) = 1
Fib(3) = Fib(1) + Fib(2) = 1 + 1 = 2
Fib(4) = Fib(2) + Fib(3) = 1 + 2 = 3

Fibonacci数(2)

$$\mathrm{Fib}(n) = \begin{cases} 1 & \text{if } n = 1, 2 \\ \mathrm{Fib}(n-2) + \mathrm{Fib}(n-1) & \text{if } n > 2 \end{cases}$$

Fib(5) = 5, Fib(6) = 8, Fib(7) = 13,
Fib(10) = 55, Fib(20) = 6765,
Fib(30) = 832040

Fibonacci数を求める(1)

$$\mathrm{Fib}(n) = \begin{cases} 1 & \ldots\ n = 1, 2 \\ \mathrm{Fib}(n-2) + \mathrm{Fib}(n-1) & \ldots\ n > 2 \end{cases}$$

```
class Fib1 {
   public static int fib (int n) {
     if (n < 3)  return 1;
     else return (fib(n-2) + fib(n-1));
  }
  public static void main (String args[]) {
     int n = Integer.valueOf(args[0].intValue());
     int r = fib(n);
    System.out.println("fib of " + args[0] + " is " + r);
  }
}
```

Javaプログラム

Fibonacci数を求める(2)

$$\mathrm{Fib}(n) = \begin{cases} 1 & \ldots\ n = 1, 2 \\ \mathrm{Fib}(n-2) + \mathrm{Fib}(n-1) & \ldots\ n > 2 \end{cases}$$

```
class Fib1 {
   public static int fib (int n) {
     if (n < 3)  return 1;
     else return (fib(n-2) + fib(n-1));
  }
  public static void main (String args[]) {
     int n = Integer.valueOf(args[0].intValue());
     int r = fib(n);
    System.out.println("fib of " + args[0] + " is " + r);
  }
}
```

n	Fib(n)
10	55
20	6765
30	832040
40	止まらない

Fibonacci数を求める（素朴法）

Fib (n)

$$= \begin{cases} 1 & \cdots\ n = 1, 2 \\ Fib(n-2) + Fib(n-1) & \cdots\ n > 2 \end{cases}$$

関数Fib（入力 n : int)
　n =1, 2　ならば、1 を返す。
　n>2 ならば、
　　r1 = Fib(n-2)
　　r2 = Fib(n-1)
　　r1+r2を返す。

疑似コード(疑似プログラム）による記述

Fibonacci数を求める（メモ化１）

Fib (n)

$$= \begin{cases} 1 & \cdots\ n = 1, 2 \\ Fib(n-2) + Fib(n-1) & \cdots\ n > 2 \end{cases}$$

関数Fib（入力 n : int)
　n =1, 2　ならば、
　　Fib(n)=1をメモして1 を返す。
　n>2 ならば、
　　Fib(n-2)の値からメモから取り出しr1とする
　　Fib(n-1)の値からメモから取り出しr1とする
　　Fib(n)=r1+r2をメモしてr1+r2を返す。

Fibonacci数を求める（メモ化２）

Fib (n)

$$= \begin{cases} 1 & \cdots\ n = 1, 2 \\ Fib(n-2) + Fib(n-1) & \cdots\ n > 2 \end{cases}$$

関数Fib（入力 n : int)
　n =1, 2　ならば、
　　Fib(n)=1をメモして1 を返す。
　n>2 ならば、
　　Fib(n-2)の値からメモから取り出しr1とする
　　Fib(n-1)の値からメモから取り出しr1とする
　　Fib(n)=r1+r2をメモしてr1+r2を返す。

Fib(1) = 1
Fib(2) = 1
Fib(3) = 1 + 1 = 2
Fib(4) = 1 + 2 = 3
Fib(5) = 2 + 3 = 5

n	Fib(n)
10	55
20	6765
30	832040
40	102334155
50	-298632863

Javaのint型として実装した場合

Fibonacci数を求める（メモ化3）

Fib (n)

$$= \begin{cases} 1 & \cdots\ n = 1, 2 \\ Fib(n-2) + Fib(n-1) & \cdots\ n > 2 \end{cases}$$

Int型がオーバーフロー
⇒下3けたのみを計算
実際には、「最近2つの値」のみメモ化すればよい

n	Fib(n)
10	055
20	765
30	040
40	155
50	025
…	…
8000	125

メモ化法による改善

- 計算速度は向上
  - 素朴法では　Fib(30)　まで
  - メモ化法では　Fib(8000)　でも計算可能
    - ただし、int　のオーバーフローを無視
- メモリは多く使う
  - メモ化のための記憶領域が必要

⇒感覚的な比較ではなく、きちんと比較をしたい。

- 「速度が10msecである」などはあまり意味がない。（その時に使用した計算機の速度に依存してしまい、客観的、普遍的な比較にならない。）

アルゴリズム

- アルゴリズム
  - 「問題」に対する「解き方」
  - 与えられた基本命令をどう組み合わせて、解を求めるかを記述したもの。
- アルゴリズムとプログラムの違い
  - プログラムは、特定のプログラム言語で記述。その言語の処理系を用いて実行可能。
  - アルゴリズムは、より抽象的・一般的な疑似プログラムとして記述する。（特定のプログラム言語や、特定の実装方式に関する記述を取り除いた記述とする。）そのままでは実行可能でないことが多い。

アルゴリズムの計算量

- 計算量
  - 「アルゴリズム」をはかる尺度
  - 「計算の複雑さ」とも言う。computational complexity
- 2つの計算量
  - 時間計算量　time complexity
    - 計算のスピード、実行速度
  - 空間計算量　space complexity
    - メモリ（アルゴリズムを実行する過程で、瞬間的に使用する最大メモリ量）
  - どちらも小さい（少ない）方が良い。

13

具体的な時間計算量

- Fib(n)に対する素朴法
  - 足し算の回数　　Fib(n)
  - 関数呼び出しの回数　　＞Fib(n)
  - その他の計算
- Fib(n)に対するメモ化法
  - 足し算の回数　　n-2
  - 関数呼び出しの回数　　n-1
  - その他の計算
- たしかに、後者の方が速い

14

Fib(n)に対する、より高速な方法（1）

準備：　x から x^n　を(掛け算だけで)計算する

- 例：x^13を計算したいとき、
  - x　-> x^2 -> (x^2)^2 = x^4 -> (x^4)^2 = x^8 …
  - x * x^4 * x^8 = x^13
  - 掛け算5回で x^13が求まった。
- nの2進数表現の桁数を r　とするとき、2r-1回以下の掛け算で、x^n　が求まる。

⇒　2 r - 1　< 2 log_2 n　- 1

15

Fib(n)に対する、より高速な方法（2）

メモ化法：

x = Fib(n-1)　⇒ x' = y = Fib(n)

y = Fib(n)　　y' = x+y = Fib(n-1)+Fib(n)

= Fib(n+1)

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} Fib(n) \\ Fib(n+1) \end{bmatrix}$$

16

Fib(n)に対する、より高速な方法（3）

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

A　->　A^2　-> A^4 -> …. -> A^1024

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} Fib(n) \\ Fib(n+1) \end{bmatrix}$$

17

Fib(n)に対する、より高速な方法（4）

2*2行列Aのｎ－１乗を計算すれば、1回の足し算でFib(n)が求まる。

- ２つの２＊２行列の積
  - 8回の掛け算
  - 4回の足し算
- n乗の計算は2log_2 n-1回以下の乗算
- よって、Fib(n)は、
  - 16log_2(n-1) -8回以下の掛け算
  - 8log_2(n-1)-3回以下の足し算

18

問題

- 「より高速な方法」は本当に高速か？

	正数の乗算	整数の加算	合計
素朴法	0	Fib(n)	Fib(n)
メモ化法	0	n-2	n-2
より高速な方法	< 16 log n	< 8 log n	<24 log n

19

オーダ記法

「定数倍」などを無視し、nが非常に大きくなったときの関数の振る舞いだけを表す表記方法

正確な定義は「データ構造とアルゴリズム」の授業を参照すること。

- 素朴法　O(α^n)　α=(√5 + 1)/2
- メモ化法　O(n)
- より高速　O(log n)

21

オーダの違いは、とんでもない差を生じる

100n と 2^n の差

- n =10　1000　1024
- n = 20　2000　ほぼ10^6
- n = 30　3000　ほぼ10^9

計算機が2倍のスピードになっても、指数関数的アルゴリズムでは、計算できるnが1増えるだけ。

⇒アルゴリズムの良し悪しの差は、ハードウェアの改良だけでは、とても埋められない。

⇒良いアルゴリズムを考えるのは、今も将来も極めて重要。

22

前半のまとめ

- アルゴリズムとは何か。
- 計算量とは何か。
- アルゴリズムの良し悪しは、入力データが非常に大きくなったときの計算時間に非常に大きく響く。

参考：

O(n^k)となるアルゴリズムが存在する問題たちの集合を「クラスP」と呼ぶ。

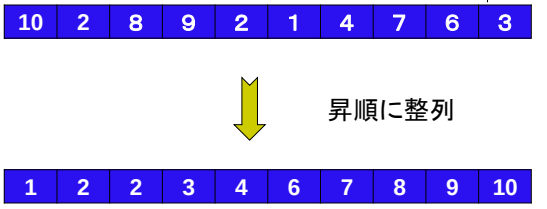
- P は多項式を意味する　a x^n + b x^(n-1) + …

23

後半：　整列アルゴリズム

24

整列（ソート）



25

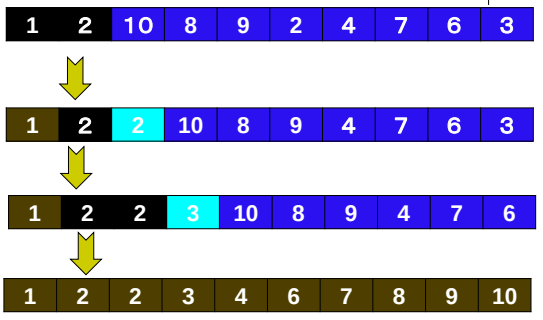
素朴な整列アルゴリズム(1)

26

素朴な整列アルゴリズム(2)

27

素朴な整列アルゴリズム(3)



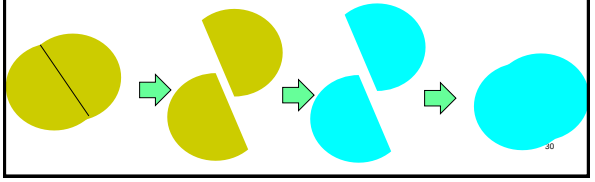
素朴な整列アルゴリズム(4)

- 時間計算量 (データの個数をnとする)
  - 2つのデータの比較
    - (n-1) + (n-2) + …+ 1 = n(n-1)/2
  - データ列における2つのデータの交換
    - n+(n-1) + … + 1 = n(n+1)/2
  - その他(ループの制御等)は無視
  - 時間計算量は O(n^2)
- 用途によってはこれでも十分利用価値があるがnが非常に大きなときは、遅すぎる。
  - 例:筑波大学のすべての科目を科目番号順に整列

29

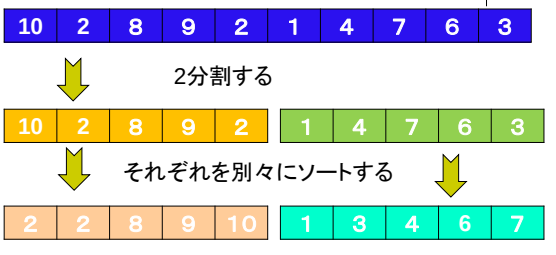
マージソート(併合ソート)のアルゴリズム(1)

- 基本アイディア:
  Divide and Conquer (分割統治法)
  - 多数のデータ数に対して何らかの解と求める場合、データを2つ(以上)に分割して、それぞれの解を求め、その結果を用いて全体に対する解を求める方法。



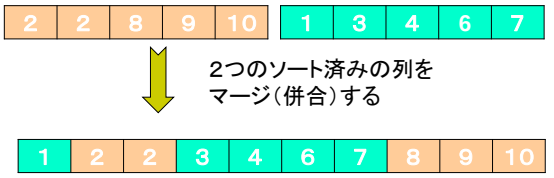
30

マージソート(併合ソート)のアルゴリズム(2)



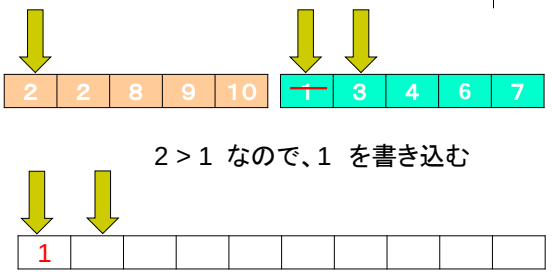
31

マージソート(併合ソート)のアルゴリズム(3)



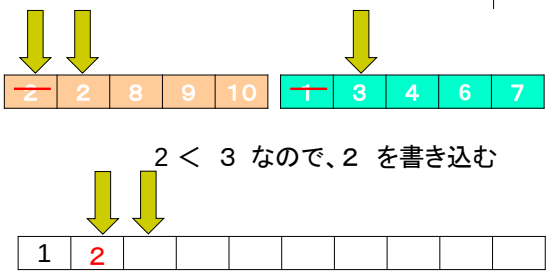
32

マージ操作(1)



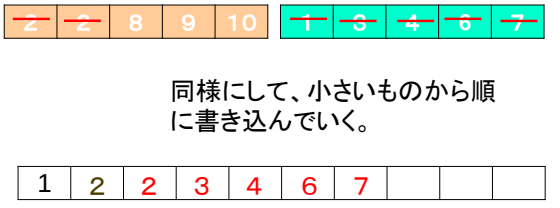
33

マージ操作(2)



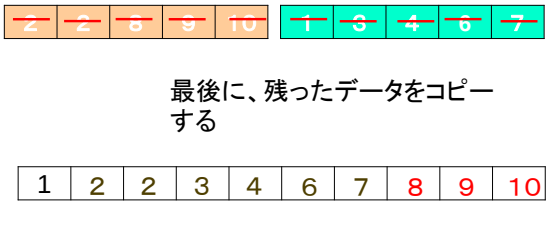
34

マージ操作(3)



35

マージ操作(4)



36

再帰 (recursion)

- マージソート(を含む分割統治法)は、再帰を使う。
- 再帰的な関数定義
  - Fib(n) = if n < 3 then 1 else Fib(n-2)+Fib(n-1)
  - 関数の定義の中で、定義されつつある関数自身を1回以上呼ぶ関数
  - 再帰を上手に使うことにより、アルゴリズムやプログラムを簡潔明瞭に表現することができる。
  - cf. 計算論(計算可能性関数の理論)

37

マージソートの計算量(1)

時間計算量(データの個数をnとし、n=2^mと仮定)

- 2つのデータの比較のみをカウントする。
- N個のデータのマージソートにおける「2つのデータの比較」回数を T(n) とすると、次の式が成立する。

T(n) <= T(n/2) + T(n/2) + マージ操作における計算量

~~2n-1~~ n-1

授業時点のスライドの誤り修正

よって

T(n) <= 2 T(n/2) + ~~2~~n – 1

38

マージソートの計算量(2)

時間計算量(データの個数をnとし、n=2^mと仮定)

- T(n) <= 2 T(n/2) + ~~2~~n – 1
- T(1) = 0

- これらより (n=2^mとおくと)
  - T(2^m) <= ~~2~~ m 2^m + 1
  - T(n) <= ~~2~~ n log n + 1
- よって T(n) は O(n log n) である。

マージソートの時間計算量はO(n log n)

39

関数の増大度の差

N	N log N	N^2	2^N
10	33	10^2	10^3
100	664	10^4	10^30
1000	9966	10^6	10^301
10000	132877	10^8	10^3010

40

整列アルゴリズムのあれこれ

古来、多くの整列アルゴリズムが考案されてきた。

- 素朴な方法(インサーションソートなど) O(n^2)
- マージソート O(n log n)
- その他、クイックソート、ヒープソートなど。
- データに対して「比較」演算のみが許される場合は、O(n log n)が最善であることが証明されている。

- データが「0から100までの整数値」のように有限個の範囲であれば、O(n)のソートが可能
  - バケツソート

41

後半のまとめ

- 整列問題を取り上げ
  - 素朴なアルゴリズム
  - 分割統治法に基づく効率の良いアルゴリズム

  について考えた。
- ポイント
  - アルゴリズムを設計するとはどういうことか。
  - そのアルゴリズムの性能(計算量)を見積もるのは、どうすればよいか。
  - O(n^2)とO(n log n)は大差がある。

42

全体のまとめ

- 「問題」をモデル化し、それに対する良いアルゴリズムを考えるのは、コンピュータサイエンスにおける最重要問題の1つ。
- 「良い」アルゴリズム
  - 重要な尺度は、時間計算量や空間計算量: データのサイズを表すnが非常に大きいときの計算量を(おおざっぱに)示す。
  - 現実問題の解決にあたって、(理想的なアルゴリズムがいつでもそのまま使えるわけではないが)大きなヒントとなる。

43

レポート (締切 7月25日)

以下の内容(1つ以上)の解答を、レポートボックス(支援室で設置)に提出せよ。(名前、学籍番号明記)

1. Fib(n)を O(log n)で計算するアルゴリズムで、Fib(1024)の下3ケタを求めなさい。
2. マージソートの時間計算量について、
   T(n) <= 2 T(n/2) + n – 1 から
   T(n)がO(n log n)であることを、導きなさい。
3. 以下のデータに対するマージソートの過程を図示しなさい。

   10, 2, 8, 9, 2, 1, 4, 7, 6, 3
4. 本日の授業内容を、A4サイズで1ページ程度のサイズで、まとめなさい。

44

レポート問題1. 解答

1. Fib(n)を O(log n)で計算するアルゴリズムで、Fib(1024)の下3ケタを求めなさい。

解。スライド17ページの行列Aの1024乗を計算

A → A^2 → A^4 → … →

A^1024=[282 243 / 243 525]

ただし、行列の要素は下3ケタのみ計算

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{1024} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} Fib(1025) \\ Fib(1026) \end{bmatrix}$$

これより **Fib(1024)の下3けた=243**

45

レポート問題2. 解答

2. T(n) <= 2 T(n/2) + n – 1 から
   T(n)がO(n log n)であることを、導きなさい。
   (n=2^m と仮定してよい。)

解。 T(2^m) <= 2 T(2^(m-1)) + 2^m -1
<= 4 T(2^(m-2)) + 2・2^m – (1+2)
<= 8 T(2^(m-3)) + 3・2^m – (1+2+4)
…
<= 2^m ・T(1) + m・2^m – (2^m-1)
= 2^m ・(m+T(1)-1) + 1

よって十分大きなmに対して、
T(2^m) <= 2^m・2m であるので、T(n)<= 2 n log_2 n
よって、T(n) は O(n log n)である。

46

レポート 問題3の解答

3. 以下のデータに対するマージソートの過程を図示しなさい。

   10, 2, 8, 9, 2, 1, 4, 7, 6, 3

解。 (奇数個の列を半分するとき、前半が後半より1長くすると仮定。逆パターンで解答しても構わない。)以下では、整列済みの列を(。。。)で表す。

[10, 2, 8, 9, 2, 1, 4, 7, 6, 3]
→ [[10, 2 ,8, 9, 2], [1, 4, 7, 6, 3]]
→ [[[10, 2, 8], [9, 2]], [[1,4,7], [6,3]]]
→ [[[[10,2],[8]], [[9], [2]]], [[[1,4],[7]], [[6],[3]]]]
→ [[[[[10],[2]],(8)], [(9), (2)]], [[[[1],[4]],(7)], [(6),(3)]]]

レポート 問題3の解答(続き)

→ [[[[(10),(2)],(8)], (2,9)], [[[(1),(4)],(7)], (3,6)]]
→ [[[(2,10),(8)], (2,9)], [[(1,4),(7)], (3,6)]]
→ [[(2,8,10),(2,9)], [(1,4,7),(3,6)]]
→ [(2,2,8,9,10), (1,3,4,6,7)]
→ (1,2,2,3,4,6,7,8,9,10)

解答は、上記の趣旨がかかれていればよく、この通りでなくてよい。

48