

I482F: 実践的アルゴリズム特論

6 (予定を無視して)再帰解析

- 浅野先生直伝の由緒正しい計算幾何
- 上原得意の活発な研究分野グラフアルゴリズム
- 上原が最近開拓している計算折り紙のアルゴリズム
どれをやるか悩んでいます...

上原隆平

北陸先端科学技術大学院大学

uehara@jaist.ac.jp

再帰解析とは？

- 再帰アルゴリズムの実行時間など、**再帰式**で表現できるものを解析する技法
 - 再帰式の例: フィボナッチ数列
$$f(0)=1, f(1)=1,$$
$$i>1 \text{ のとき: } f(i)=f(i-1)+f(i-2)$$
- 再帰アルゴリズムの代表例: **分割統治法**
- いくつかの典型的なパターンにはまれば、難しくはない
- 特に**マスター定理**を知っていれば、かなり広範囲の再帰式が解ける.

オーダー記法

漸近的計算量: 入力のサイズ n が十分に大きくなったときに
計算量がどのような割合で増加するかを表したもの.
計算量の増加の割合を示すのが目的なので,
主要項だけで十分, また係数も重要でない.

ビッグオー記法 $O(f(n))$

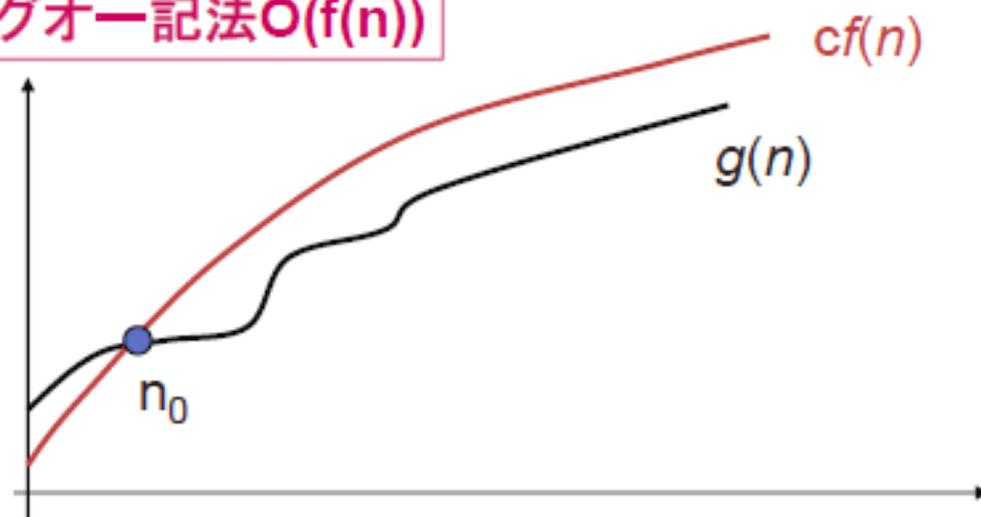
$$O(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0 (g(n) \leq cf(n))\}$$

$$O(f(n)) = \{g(n) : \text{すべての } n \geq n_0 \text{ に対して } g(n) \leq cf(n) \\ \text{であるような正の定数 } c \text{ と } n_0 \text{ が存在する}\}$$

$g(n) \in O(f(n))$ と書く代わりに, 便宜上,

$g(n) = O(f(n))$ と書くこともある.

ビッグオー記法 $O(f(n))$



計算量の上界を表すのに用いる.

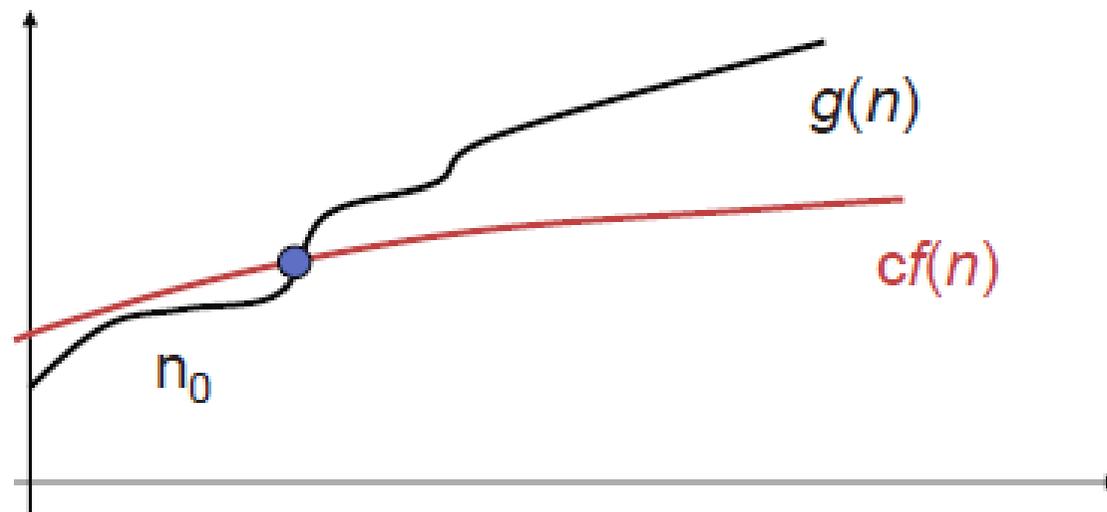
たとえば, 挿入法と呼ばれるデータ整列(ソート)法は, 逆順に並べられたデータが入力されると, データ数 n に対して n^2 に比例する時間がかかってしまう.

したがって, 挿入法の計算時間は $O(n^2)$ と言える.

挿入法は $O(n)$ 時間でソートを完了することもある.

すでにソート済みのデータが入力された場合がそうである.

ビッグオメガ記法 $\Omega(f(n))$



$$\Omega(f(n)) = \{g(n) \mid \exists c > 0, \exists n_0, \forall n \geq n_0 (cf(n) \leq g(n))\}$$

$\Omega(f(n)) = \{g(n) : \text{すべての } n \geq n_0 \text{ に対して } cf(n) \leq g(n)$
であるような正の定数 c と n_0 が存在する}

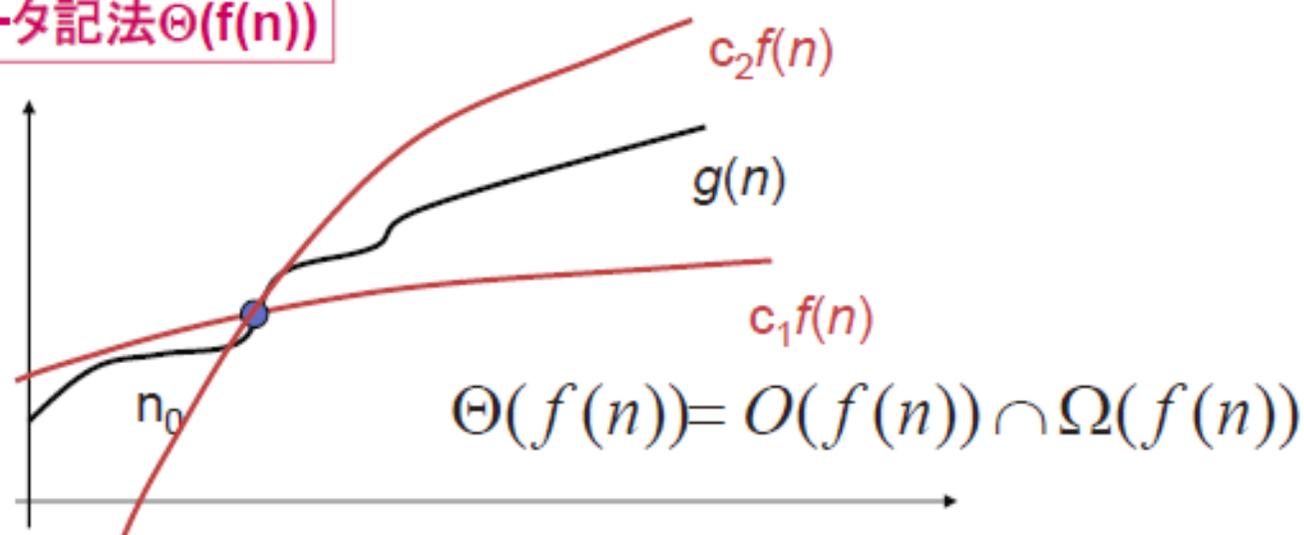
ビッグオメガ記法 $\Omega(f(n))$

計算量の下界を表すのに用いる.

n 個のデータをデータ間の比較を用いてソートするには、
どんなアルゴリズムでも必ず $n \log n$ に比例する時間以上が
かかってしまう最悪の場合が存在する.

よって、ソーティング問題の下界は $\Omega(n \log n)$ であると言える.

シータ記法 $\Theta(f(n))$



$$\Theta(f(n)) = \{g(n) \mid \exists c_1, c_2 > 0, \exists n_0, \forall n \geq n_0 \\ (c_1 f(n) \leq g(n) \leq c_2 f(n))\}$$

$\Theta(f(n)) = \{g(n) : \text{すべての } n \geq n_0 \text{ に対して} \\ c_1 f(n) \leq g(n) \leq c_2 f(n) \text{ であるような正の} \\ \text{定数 } c_1, c_2 \text{ と } n_0 \text{ が存在する } \}$

分割統治法によるマージソートのアルゴリズム

入力は配列 $a[0], \dots, a[n-1]$

ソートすべき区間を $[low, high]$ で表現する. 最初の区間は $[0, n-1]$

区間 $[low, high]$ の配列要素のソート

関数 $Mergesort(low, high)$

$low = high$ なら, そのまま終了.

$mid = (low + high)/2$ として中央のインデックスを求める.

区間 $[low, high]$ を $[low, mid]$ と $[mid+1, high]$ に分割.

区間 $[low, mid]$ の配列要素を再帰的にソート.

$Mergesort(low, mid)$

区間 $[mid+1, high]$ の配列要素を再帰的にソート.

$Mergesort(mid+1, high)$

上記の2つのソート結果を一つのソート列に統合する.

マージソート法の解析

n 個のデータをマージソートするのに要する時間を $T(n)$ とする.

分割ステップ

サイズ n の問題をサイズ $n/2$ の2つの問題に分割して,
それぞれを再帰的に解く

その時間は $2T(n/2)$ と表現できる.

統合ステップ

2つの部分問題の解(この場合は, 2つのソート列)を統合
して最終的な答(この場合は, 2つのソート列をマージした
列)を得る.

このための時間は $O(n)$



両方の列の先頭を比較して小さい方を取るとい
う操作を最大 $n/2$ 回繰り返す.

マージソート法の解析

n個のデータをマージソートするのに要する時間を $T(n)$ とする.

分割ステップ

サイズnの問題をサイズn/2の2つの問題に分割して,
それぞれを再帰的に解く

その時間は $2T(n/2)$ と表現できる.

統合ステップ

2つの部分問題の解(この場合は, 2つのソート列)を統合
して最終的な答(この場合は, 2つのソート列をマージした
列)を得る.

このための時間は $O(n)$

よって, 全体の計算時間は

$$T(n) = 2T(n/2) + O(n)$$

と表現できる.

これを, $T(1) = 0, T(2)=2$ のような境界条件の下に解く.

再帰式 $T(n) = 2T(n/2) + O(n)$, $T(1) = 0$, $T(2)=2$ の解法

まず, $O(n)$ のところを1次式 $an + b$ で置き換えると

$$T(n) = 2T(n/2) + an + b$$

を得る. ここで, $n = 2^k$ と置くと

$$T(2^k) = 2T(2^{k-1}) + a2^k + b$$

となる. ここで, さらに $T(2^k) = t_k$ と置くと,

$$t_k = 2t_{k-1} + a2^k + b$$

を得る. これを繰り返し用いると

$$t_k = 2(2t_{k-2} + a2^{k-1} + b) + a2^k + b$$

$$= 2^2t_{k-2} + a2^k + 2b + a2^k + b$$

$$= 2^2t_{k-2} + (1+1)a2^k + (2+1)b$$

$$= 2^2(2t_{k-3} + a2^{k-2} + b) + (1+1)a2^k + (2+1)b$$

$$= 2^3t_{k-3} + (1+1+1)a2^k + (2^2+2+1)b$$

$$= \dots = 2^{k-1}t_1 + (1+1+\dots+1)a2^k + (2^{k-2}+\dots+2+1)b$$

$$= 2^k + ak2^k + 2^{k-1}b$$

$$= n + an \log n + b/2 \log n = O(n \log n) \quad (n = 2^k, k = \log n)$$

マージソート覚書き

- 配列を二つ使うところが弱点
- 実装が簡単なので, バグも出にくく, 安定して高速に動作するので, ファンが多い
- 「安定ソート」でもある. (同じ値を持つデータが反転しない)

ユークリッドの互除法

2つの正整数 m, n が与えられたとき、それらの最大公約数を求めよ。

人類最古のアルゴリズム！

ユークリッドの互除法

基本的な性質

(1) $\text{GCD}(m, 0) = m$ 任意の整数 m と0の最大公約数は m とする

(2) $\text{GCD}(m, n) = \text{GCD}(n, m \bmod n)$

この2つの性質を知っていればプログラムが書ける。

```
int GCD(int m, int n){
    while(n > 0){
        int r = m % n; m = n; n = r;
    }
    return m;
}
```

```
int GCD(int m, int n){
    while(n > 0){
        int r = m % n; m = n; n = r;
    }
    return m;
}
```

例題:

GCD(954, 288)

=GCD(288, 954 % 288) = GCD(288, 90)

= GCD(90, 288 % 90) = GCD(90, 18)

= GCD(18, 90 % 18) = GCD(18, 0)

= 18.

954と288の最大公約数は18.

```
int GCD(int m, int n){
  while(n > 0){
    int r = m % n; m = n; n = r;
  }
  return m;
}
```

再帰的に表現することも可能.

$GCD(m, 0) = m$

$GCD(m, n) = GCD(n, m \bmod n)$

このままでプログラムになる.

```
int GCD(int m, int n){
  if(n==0) return m;
  else return GCD(n, m % n);
}
```

```
int GCD(int m, int n){
  if(n==0) return m;
  else return GCD(n, m % n);
}
```

実行時間を $T(m, n)$ とする.

$T(m, 0) = c$ (定数)

$n > m$ のときに $GCD(m, n)$ を呼び出すと, $GCD(n, m \% n)$ を呼び出すことになるが, $n = qm + r$ とすると, $q \geq 1$ なので, $n > m + r$. また, $r < m$ だから, $n + m > m + 2r$, すなわち, $r < n/2$ を得る. よって, $GCD(m, n)$ が $GCD(n, m \% n)$ を呼び出すとき, $m \% n$ の値(=r)は $n/2$ 以下になっている.

さらにもう一度GCD関数を呼び出すと, もう一方の値も半分以下になるので,

$T(m, n) = T(m/2, n/2) + 2c$

という漸化式を得る.

これを m について解くと, $T(m, n) = O(\log m)$ となる.

```
int GCD(int m, int n){
    if(n==0) return m;
    else return GCD(n, m % n);
}
```

実行時間を $T(m,n)$ とする. 高々2回再帰呼び出しを行えば, 次の $T(m',n')$ において $m' \leq m/2$ が成立することを示す:

1. $m < n$ のとき: 1回の再帰呼び出しで $GCD(n,m)$ を呼び出し, それ以降はずっと $m > n$ が成立するので, 最初以外は考えなくてよい.
2. $m > n$ で $m/2 \geq n$ のとき: 1回の再帰呼び出しで $GCD(n, m \bmod n)$ となるが, この時点で $n \leq m/2$ であるので, 主張は成立する.
3. $m > n$ で $m/2 < n$ のとき: 1回の再帰呼び出しで $GCD(n, m-n)$ となり, 次の再帰呼び出しで $GCD(m-n, (m-n) \bmod n)$ となるが, $m/2 < n$ より, $m-n < m/2$ となり, この時点で $m' \leq m/2$ であるので, 主張が成立する.

以上より, 一般の場合(最初の呼び出しを除いて)2回再帰呼び出しを行えば

$$T(m,n) = T(m/2, n/2) + c$$

としてよく, 実行時間は $O(\log m)$ となる.

再帰呼び出しの危険性

n個のデータが入った配列 $a[0], \dots, a[n-1]$ における最大値を求める関数を再帰的に記述し、その計算時間を漸化式で解析せよ。

配列の low 番目から $high$ 番目までの値の最大値を求める関数を $Max(low, high)$ とすると、

$Max(low, high) = a[low]$ $low = high$ のとき、

$Max(low, high) = \max(a[low], Max(low+1, high))$ それ以外するとき



配列のlow番目からhigh番目までの値の最大値を求める関数を
Max(low, high)とすると,
Max(low, high) = a[low] low = highのとき,
Max(low, high) = max(a[low], Max(low+1, high)) それ以外するとき

上の定義に基づいてプログラムを書いてみよう

```
int Max(int low, int high){  
    if(low == high) return a[low];  
    if(a[low] > Max(low+1, high)) return a[low];  
    else return Max(low+1, high);  
}
```

```
int main(void){  
    配列aへの値の入力;  
    Max(0, n-1)が最大値;  
}
```

```

#include <stdio.h>

int n = 50;
int a[] = {1993, 2834, 2774, 18834, 1321, 3245, 3321, 784, 983, 102,
          293, 1324, 763, 6654, 890, 8762, 653, 7866, 9645, 19323,
          1832, 3224, 4327, 8766, 543, 243, 2441, 5342, 653, 877,
          7835, 4523, 765, 462, 8799, 8745, 674, 293, 7765, 864,
          8732, 3254, 6546, 4327, 4566, 48255, 321, 33, 987, 765};

int MAX(int i)
{
    if(i == 0) return a[0];
    else if( a[i] > MAX(i-1) ) return a[i];
    else return MAX(i-1);
}

int main(void)
{
    int i, m;

    do{
        scanf("%d", &m);
        if(m <= 0 || m > 50) break;
        for(i=0; i<m; i++) printf("%d ", a[i]);
        printf("¥n¥nMAX = %d¥n", MAX(m-1));
    }while(1);
    return 1;
}

```

ナゾの現象:

- n=20あたりだと問題なく動作するが, , ,
- n=30くらいで普通のPCはだんだん遅くなり, n=40あたりだと返ってこなくなる...

```
int Max(int low, int high){
    if(low == high) return a[low];
    if(a[low] > Max(low+1, high)) return a[low];
    else return Max(low+1, high);
}
```

このプログラムはなぜ遅いのか？
実行時間を解析してみよう。

呼び出しは

$\text{Max}(0, n-1) \Rightarrow \max(a[0], \text{Max}(1, n-1)) \Rightarrow \max(a[1], \text{Max}(2, n-2))$

と進んで行く。つまり、

$\text{Max}(0, n-1)$

1. $\text{if}(a[0] > \text{Max}(1, n-1)) \text{ return } a[0];$

2. $\text{else return Max}(1, n-1);$

1行目で $\text{Max}(1, n-1)$ を呼び出しているのだから、時間は $T(n-1)$

2行目でも1行目の条件が成り立たなければ再び $\text{Max}(1, n-1)$ を実行。

よって、**最悪の場合は、毎回 $\text{Max}()$ を2回呼び出す。**

漸化式は、 $T(n) = 2T(n-1) + c$. これを解くと、 $T(n) = O(2^n)$.

ちょっと書き換えるだけで、線形時間になります。どうやればよいでしょう？

マスター

ここでは証明はしませんが、非常に丁寧な証明が「アルゴリズムイントロダクション」に載っています。

マスター定理(Master Theorem)

$a \geq 1, b \geq 1$ を定数とし、 $f(n)$ を関数とする。非負整数上の関数 $T(n)$ を再帰式

$$T(n) = aT(n/b) + f(n)$$

によって定義する。このとき、 $T(n)$ は漸近的に次のような限界をもつ。

1. ある定数 $\varepsilon > 0$ に対して $f(n) = O(n^{\log_b a - \varepsilon})$ ならば、

$$T(n) = \Theta(n^{\log_b a}).$$

2. $f(n) = \Theta(n^{\log_b a} \log^k n)$ ならば、 $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.

3. ある定数 $\varepsilon > 0$ に対して $f(n) = \Omega(n^{\log_b a + \varepsilon})$ であり、しかもある定数 $c < 1$ と十分大きなすべての n に対して $af(n/b) \leq cf(n)$ ならば、 $T(n) = \Theta(f(n))$.

先に見た漸化式 $T(n) = 2T(n/2) + O(n)$ は、

$a=b=2, f(n) = O(n)$ の場合に相当するので、2の場合が適用できて、 $T(n) = \Theta(n \log n)$ を得る。

マスター定理を用いて次の漸化式を解け.

1. $T(n) = 2T(n/2) + n$

2. $T(n) = 2T(n/2) + n \log n$

3. $T(n) = 3T(n/2) + n$

4. $T(n) = 4T(n/2) + n$

5. $T(n) = 4T(n/2) + n^2$

6. $T(n) = 4T(n/2) + n^3.$

7. $T(n) = T(n/2) + 1$