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

良いアルゴリズムとは

第2講: 平成20年10月10日 (金) 4限 E252教室

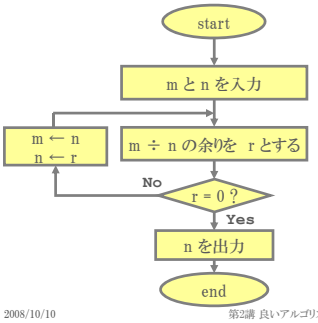
中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

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 1 講の補足

- フローチャート と 擬似言語




```
int gcd(int m, int n)
{
    do{
        r ← m ÷ n の余り;
        m ← n;
        n ← r;
    }while(r <> 0);
    return m;
}
```

擬似言語とは
この部分のこと

今日の講義の内容

- 良いアルゴリズムの評価基準
  - 時間計算量
  - 領域計算量
- 多項式時間アルゴリズムと指数時間アルゴリズム
- オーダ記法
- 再帰アルゴリズム

計算量

- アルゴリズムの計算量
  アルゴリズムを実行するのに必要となる計算の量
  - 時間計算量
    - アルゴリズム実行に必要な時間の尺度
  - 領域計算量
    - アルゴリズム実行に必要な領域(メモリ)の尺度
    計算量が小さい＝アルゴリズムは効率的
- 時間計算量と領域計算量はトレードオフの関係
  - 本講義では時間計算量で評価していく

平均計算量と最大計算量

一般にアルゴリズムの計算量は入力に依存する

- アルゴリズムごとに「得意な入力」と「苦手な入力」がある

- 最大(時間, 空間)計算量
  - 最も不得意な入力が与えられたときの計算量
- 平均(時間, 空間)計算量
  - 全ての入力に対する計算量の平均

時間計算量の評価例 1

```
#define             MAX    5
int    perm[MAX]={2, 5, 3, 4, 1};

search(key)/* 配列 perm の中から値 key の位置を探す */
int    key;
{
       int    i = 0;

       while (i < MAX) {
              if (perm[i] == key)
                     return(i);
              i++;
       }
       return(-1);
}
```

配列：データを一列に並べたもの，先頭から番号を使って参照できる

例　perm[i]:
配列 perm の i 番目の要素

key の値	1	2	3	4	5
ステップ数	15	3	9	12	6

時間計算量の評価例 2

- 最大ステップ数

$$15 = 3 \times \mathrm{MAX}$$

- 平均ステップ数

$$9 = \left(\sum_{i=1}^{\mathrm{MAX}} 3i \right) / \mathrm{MAX} = \frac{3}{2}(\mathrm{MAX} + 1)$$

key の値	1	2	3	4	5
ステップ数	15	3	9	12	6

計算量評価のコストパフォーマンス

- プログラムのステップ数を厳密に評価することは，一般にはかなり手間がかかる
- ステップ数を厳密に評価しても，現実世界の時間単位への対応付けは難しい

- もっと大雑把で良いから簡単に使える尺度が欲しい！ ⇒ アルゴリズムのオーダー

アルゴリズムのオーダー

- アルゴリズムの時間計算量が $f(n)$ のオーダーである：$O(f(n))$ である
  - 入力データの大きさ n に対し，アルゴリズムの実行時間が関数 $f(n)$ に比例して増加する

係数は考えない

さきほどの例の場合：
最大ステップ数　$15 = 3 \times \mathrm{MAX}$
平均ステップ数　$9 = \left(\sum_{i=1}^{\mathrm{MAX}} 3i \right) / \mathrm{MAX} = \frac{3}{2}(\mathrm{MAX} + 1)$

配列サイズ＝入力データサイズと考えると．．．
最大時間計算量，平均時間計算量とも $O(n)$ である

オーダーの見積もり

- 計算量のオーダー表現：
  - きわめて大雑把な評価尺度
  - 大雑把な見積もりで導出することができる

1. アルゴリズムを小さな操作単位に分割
2. 各操作単位のオーダーを評価
3. 操作単位のオーダーを合成して，全体のオーダーを得る

アルゴリズムの分割

```
search(key) /* 配列 perm の中から値 key の位置を探す */
int    key;
{
       int    i = 0;

       while (i < MAX) {
              if (perm[i] == key)
                     return(i);
              i++;
       }
       return(-1);
}
```

実行時間が入力サイズに依存しないステップ（基本ステップ）

ループ回数が入力サイズに依存するループ構造

オーダーの評価 (1)

- ルール 1：基本ステップのオーダーは $O(1)$
- 基本ステップ
  - 実行時間が入力サイズに依存しないステップ
    - 変数への代入
    - 数値の演算
    - ポインタ操作 etc.
- 一般に，以下は基本ステップでないことに注意
  - (入力サイズに依存した)配列のコピー
  - 関数呼び出し

オーダーの評価 (2)

- **ルール 2：$O(f(n))$ の操作と $O(g(n))$ の操作を連続して行う場合，操作全体のオーダーは $O(\max(f(n), g(n)))$**

ただし，関数の大小比較は増加率により行う

$1 < \log n < n < n \log n < n^2 < n^3 < \ldots 2^n$

オーダーの評価 (3)

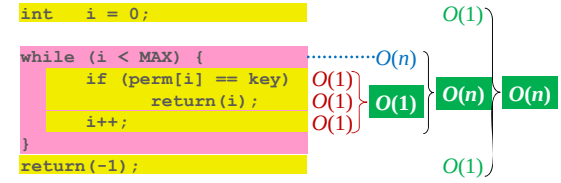
- **ルール 3：$O(f(n))$ 回だけまわるループの内部で $O(g(n))$ の操作を実行する場合，全体のオーダーは $O(f(n) \times g(n))$**

- 係数は無視してよい
- 最高次の項以外は無視してよい

アルゴリズムの分割

```
search(key) /* 配列 perm の中から値 key の位置を探す */
int   key;
{
        int   i = 0;

        while (i < MAX) {
                if (perm[i] == key)
                        return(i);
                i++;
        }
        return(-1);
}
```



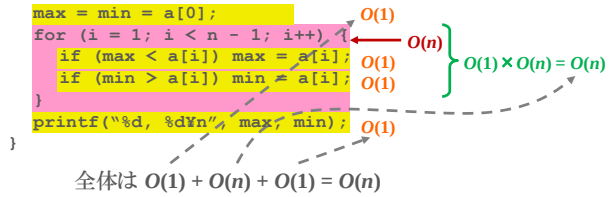
ループの回数：平均時，最悪時とも $O(n)$
⇒ 平均時間計算量，最大時間計算量とも $O(n)$

練習問題 1

- 以下の手続きのオーダーを求めよ

```
void maxmin(int a[], int n)
{
   int  i, max, min;

   max = min = a[0];
   for (i = 1; i < n - 1; i++) {
      if (max < a[i]) max = a[i];
      if (min > a[i]) min = a[i];
   }
   printf("%d, %d¥n", max, min);
}
```



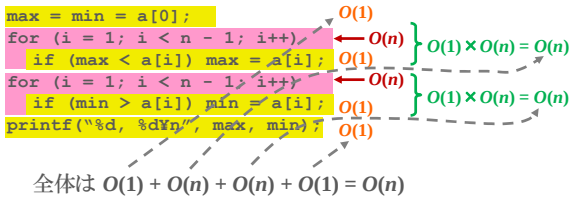
全体は $O(1) + O(n) + O(1) = O(n)$

練習問題 2

- 以下の手続きのオーダーを求めよ

```
void maxmin2(int a[], int n)
{
   int  i, max, min;

   max = min = a[0];
   for (i = 1; i < n - 1; i++)
      if (max < a[i]) max = a[i];
   for (i = 1; i < n - 1; i++)
      if (min > a[i]) min = a[i];
   printf("%d, %d¥n", max, min);
}
```



全体は $O(1) + O(n) + O(n) + O(1) = O(n)$

練習問題 3

- 以下の手続きのオーダーを求めよ

```
void bubble(int a[], int n)
{
  int i, j, t;

  for (i = 0; i < n - 1; i++)
    for (j = n - 1; j > i; j--)
      if (a[j - 1] > a[j]) {
        t = a[j]; a[j] = a[j - 1]; a[j - 1] = t;
      }
}
```

$O(n) \times O(n) = O(n^2)$

$O(n)$

$O(1) \times O(n) = O(n)$

$O(n)$

$O(1)$

全体は $O(n^2)$

オーダー評価:特殊ケース 1

- 条件分岐部の評価には要注意

```
if (x % 2 == 0)
  O(f(n)) の処理
else
  O(g(n)) の処理
```

→ 計算量は $O(\max(f(n), g(n)))$

```
if (x % 2 == 3)
  O(f(n)) の処理
else
  O(g(n)) の処理
```

→ 計算量は $O(g(n))$

表現上の構造にとらわれず，実質的な振舞いの把握が必要

オーダー記法に用いる関数

- n, $n\log n$, n^2, n^3 ：n の多項式
  - 多項式時間アルゴリズム
  - Polynomial Time Algorithm
  - 現実的
- 2^n, $n!$, n^n ：n の指数関数
  - 指数時間アルゴリズム
  - Exponential Time Algorithm
  - 非現実的

多項式オーダーと指数オーダー

- 計算速度向上の効果

表 1.3 計算速度向上の効果

	1秒で解ける最大の問題サイズ				
計算速度	$O(2^n)$	$O(n)$	$O(n^2)$	$O(n^5)$	$O(n^{10})$
1	100	100	100	100	100
2	101	200	141	115	107
10	103	1000	316	158	126
100	107	10000	1000	251	158
1000	110	100000	3162	398	200
10000	113	1000000	10000	631	251
100000	117	10000000	31623	1000	316
1000000	120	100000000	100000	1585	398

再帰アルゴリズム

- 処理手順が自身を用いて定義されているもの

```
recursive (n) {
    if (自明なケース) {
        自明なケースの処理 ; /* 終了条件 */
    } else {
        recursive (m) ;  /* m < n */
        (処理) ;
    }
}
```

  - 自身の引数より小さな引数で自身を呼び出す
  - 自明なケースの処理が存在
  - 表面的にループが出現しない

再帰プログラムの例: 階乗の計算

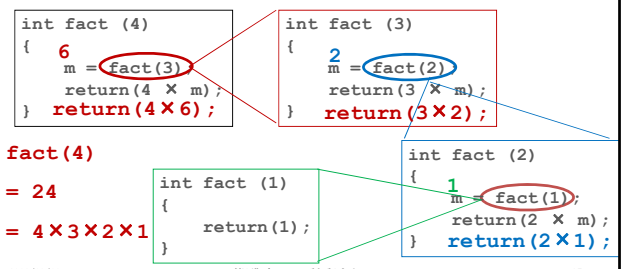
- 階乗
  - 例: $6! = 5 \times 4 \times 3 \times 2 \times 1$
- ヒント
  - $6! = 6 \times 5!$, $5! = 5 \times 4!$, $\cdots$, $2! = 2 \times 1!$, $1! = 1$
- プログラム

```
int fact (int n)
{
    int m;
    if(n = 1)
        return(1);
    else{
        m = fact(n-1);
        return(n × m);
    }
}
```

ちょっとフローチャートでは書けない

再帰プログラムの概念

- ちょっと分かりにくいので以下の図のように考えるとよい



ユークリッドの互除法を再帰で書く

- ヒント
  - r = 0 でないなら，m, n の最大公約数の代わりに n, r の最大公約数を求める

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

r=0 なら n が 最大公約数

r=0 でないなら n と r の 最大公約数を求める

オーダー評価：特殊ケース 2

再帰プログラムのオーダー評価は，少し面倒

```
int recursive(int n)
{
  if (n <= 1)
    return(1);
  else
    return(recursive(n - 1) + recursive(n - 1));
}
```

入力が n のときの，この再帰プログラムの計算量を T_n とする

この場合のステップ数は，漸化式 $T_n = 2T_{n-1}$ で与えられる
⇒ 計算量は $O(2^n)$
（互除法は $T_n = T_{n-1}$ なので $O(n)$）

第 2 講のまとめ

- 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