# 再帰呼び出し

山本昌志*

2005 年 12 月 19 日

## 1　本日の学習内容

### 1.1　前回の復習

リストに引き続き，前回の授業ではスタックとキューと呼ばれるデータ構造を学習した．

**スタック**　最後に入れられたものが最初に取り出されるデータ構造である．このことから，LIFO(last in first out, 後入れ先出し) と呼ばれる．そのイメージは，図 1 の通りである．

**キュー**　スタックではデータの挿入と取り出しが列の一方からのみであったの対して，キューは列の両端から行う．一方がデータの追加で一方がデータの取り出しとして使われ，最初に入れたデータが一番最初に取り出される．取り出されるデータは格納されている最古のデータで，最初に入れられたものが最初に取り出されることから，FIFO(first in first out, 先入れ先出し) と呼ばれる．そのイメージは，図 2 の通りである．

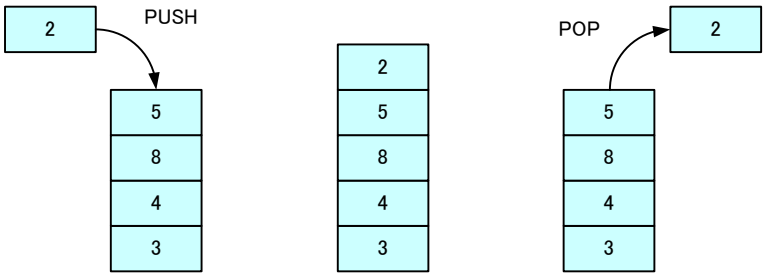


図 1: スタック

*独立行政法人　秋田工業高等専門学校　電気情報工学科

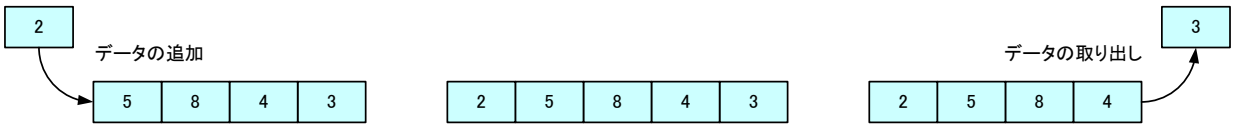


図 2: キュー

### 1.2 本日の学習内容

本日はデータ構造ではなく，再帰呼び出し (recursive call) と言われるアルゴリズムの学習を行う．本日のゴールは，以下の通り．

- 再帰呼び出しと普通の関数の違いが分かる．
- 再帰呼び出しの関数の作り方が理解できる．少なくとも，再帰呼び出しを使った階乗のプログラムが書けること．

## 2 再帰呼び出しとは

関数の中で，自分自身の関数を呼び出すことを再帰呼び出し (recursive call) と言う．このアルゴリズムを使うと，ある種の問題に対して，手続きが非常に簡素に表現できることがある．要するにプログラムが簡単に書けると言うことである．

これは，実際にプログラムを示した方がよく分かるので，以降に簡単な例を示す．

### 2.1 階乗の計算

数学で，演算子！で表される階乗という演算にしばしばお目にかかる．たとえば，5 の階乗は

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ &= 120 \end{aligned} \tag{1}$$

である．一般の整数 $n$ では，

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1 \tag{2}$$

となる．また，$0! = 1$ となっている．これらのことから，漸化式は

$$n! = n \times (n-1)! \tag{3}$$

$$0! = 1 \tag{4}$$

と表すことができる．

ここで，整数を入力して，その階乗を計算するプログラムを作ってみよう．通常は，式 (2) の通りにプログラムを書くだろう．この式の通りにプログラムを書くと，リスト 1 のようになる．

同じ計算を，式 (3) と (4) に着目してプログラムを作ってみよう．すると，リスト 2 のようになる．このプログラムを見ると，漸化式をそのままプログラムしたことが分かるだろう．

リスト 2 のプログラムは，リスト 1 とは異なり，関数内で自分自身を呼びだしている．このように，自分自身を呼び出すことを再帰呼び出し (recursive call) と言う．C 言語ではこのように自分自身を呼び出すことができるのである．

今回の例の場合，再帰呼び出しを使うメリットはほとんど無い．実行速度も繰り返し文を使うリスト 1 の方が速いだろうし，メモリーの消費も少ない．しかし，次のハノイの塔のプログラムになると，再帰呼び出しを使うと，簡潔なプログラムが書ける．ここの例題の階乗の計算では，再帰呼び出しのアルゴリズムの大体のイメージがつかめれば良い．

リスト 1: 繰り返し文を使った階乗を計算するプログラム

```
1  #include <stdio.h>
2
3  /*=============================================================*/
4  /*  階乗を計算する関数                                         */
5  /*=============================================================*/
6  int kaijyo(int n){
7    int i, value;
8
9    value=1;
10
11   for(i=n; i>=1; i--){
12     value*=i;
13   }
14
15   return value;
16
17 }
18
19 /*=============================================================*/
20 /*  メイン関数                                                 */
21 /*=============================================================*/
22 int main(){
23   int n;
24
25   printf("階乗を計算します．値を入れてください\n");
26   scanf("%d",&n);
27
28   printf("%d!=%d\n",n,kaijyo(n));
29
30   return 0;
31 }
```

リスト 2: 再帰呼び出しを使った階乗を計算するプログラム

```
1  #include <stdio.h>
2
3  /*=============================================================*/
4  /*  階乗を計算する関数                                         */
5  /*=============================================================*/
6  int kaijyo(int n){
7
```

```
 8     if(n==0){
 9       return 1;
10     }else{
11       return n*kaijyo(n-1);
12     }
13
14  }
15
16  /*=============================================================*/
17  /*   メイン関数                                                */
18  /*=============================================================*/
19  int main(){
20    int n;
21
22    printf("階乗を計算します．値を入れてください\n");
23    scanf("%d",&n);
24
25    printf("%d!=%d\n",n,kaijyo(n));
26
27    return 0;
28  }
```

## 2.2 ハノイの塔

### 2.2.1 パズルの内容

1883 年，フランスの数学者エドゥアール・リュカが考案したパズル「ハノイの塔」の製品版には，以下の話が書かれていたということである [1].

> 世界の中心の地・ベレナスに，天高くそびえる 3 本のダイヤモンドの柱．そのうち 1 本には，64 枚の黄金の円盤が刺さっている．向かって左側の柱だ．一番下のものがもっとも大きく，上に行くにしたがって円盤は小さくなっていく．そうして積み重ねられた 64 枚の円盤．これを 3 つのルールに従って移動していく．
>
> 1. 円盤は一度に 1 枚ずつしか移動できない．
> 2. 柱のないところに円盤を置いてはならない．
> 3. 小さい円盤の上に，大きな円盤を重ねてはならない．
>
> そうして右の柱にすべての円盤を移したとき，世界は崩壊し，その終焉を迎えるだろう．

日夜，インドの坊主たちが塔から塔へ円盤を移動させているのである．これが，世界の終わりの時を刻んでいるというから驚きである．

64 枚の円盤を移動させるのは大変なので，3 枚の円盤を移動させてみよう．それは，図 3 のようになる．

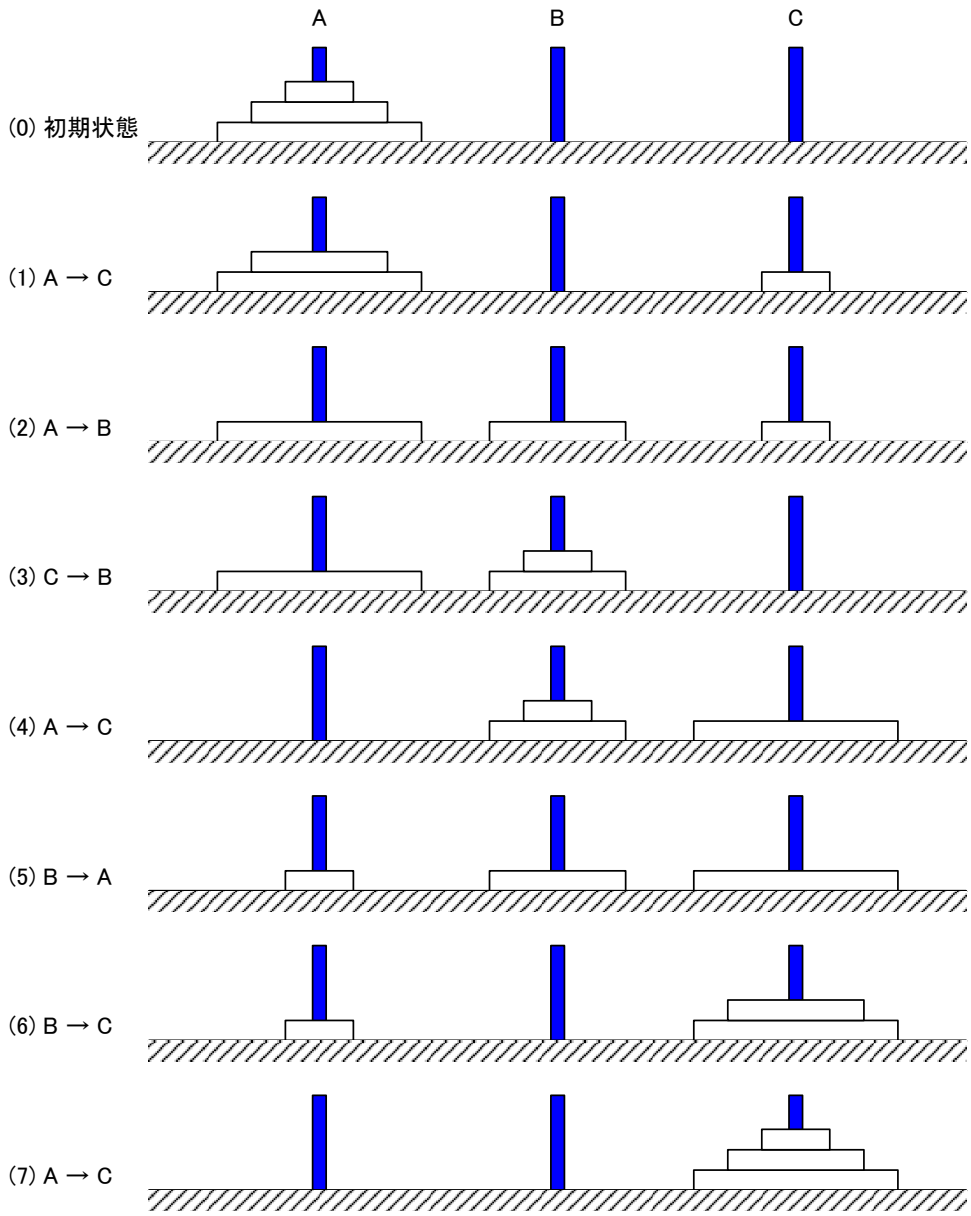


図 3: 円盤が 3 枚の場合のハノイの塔

### 2.2.2 パズルの解法

このパズルを C 言語のプログラムで解こうというのである．これまでの講義をほとんど完璧に分かってきた者でも，どのように考えて良いか分からないだろう．しかし，このプログラムはびっくりするほど簡単に書けるのである．リスト 3 を見よ．このパズルを解いているところは，関数 `move()` の数行である．びっ

くりして驚いて，そしてこのプログラムを理解してほしい．

このパズルを解くということは，円盤の移動の手順を示すことである．図 3 の左端に書かれている円盤の移動元と移動先を示せれば良い．この手順をすべて示して，左端にある円盤を右端に移動させるのである．

これを一つずつ考えていたのではとてもプログラムはできない．次にようにすれば，このパズルが完成することが分かるだろう．塔を便宜上，移動元 (from)，作業用 (work), 移動先 (to) に分ける．

1. 移動させるべき円盤が一つの場合，移動元 (from) から移動先 (to) の塔へ直接移動させる．
2. 移動させるべき円盤が $n$ 個 ($\geq 2$) の場合
   (a) 移送先 (to) を作業用に使い，$n-1$ 枚の円盤を移動元 (from) から，作業用 (work) へ移動させる．
   (b) 移動元 (from) の一番下にあった円盤を，移動先 (to) へ移動させる．
   (c) 移動元 (form) を作業用に使い，$n-1$ 枚の円盤を作業用 (work) から，移動先 (to) へ移動させる．

これを，階乗を計算したときの漸化式のように記述すると，

- 移動する円盤が一つの場合，移動先の塔へ移動させて処理は終了．
- $n$ 枚の円盤の移動は，

$$n_{\mathrm{from}\to\mathrm{to}} = (n-1)_{\mathrm{form}\to\mathrm{work}} + 1_{\mathrm{form}\to\mathrm{to}} + (n-1)_{\mathrm{work}\to\mathrm{to}} \tag{5}$$

  と漸化式を書くことができる．

となる．

このアルゴリズムをプログラムで記述したものが，リスト 3 である．このアルゴリズムとリスト 3，および図 3 を考えよ．

リスト 3: ハノイ塔を解くプログラム

```
1  #include <stdio.h>
2
3  /*=============================================================*/
4  /*  ハノイの塔の移動を示す関数                                 */
5  /*=============================================================*/
6  void move(int n, char from, char work, char to){
7
8    if(n==1){
9      printf("%c -> %c\n", from, to);
10   }else{
11     move(n-1, from, to, work);
12     printf("%c -> %c\n", from, to);
13     move(n-1,work, from, to);
14   }
15 }
16
17 /*=============================================================*/
18 /*  メイン関数                                                 */
19 /*=============================================================*/
20 int main(){
21   int n;
22
23   printf("ハノイの塔の円盤の枚数を入力してください\n");
```

```
24    scanf("%d",&n);
25
26    move(n, 'A', 'B', 'C');
27
28    return 0;
29  }
```

## 2.3 クイックソート

実を言うと諸君は，既に再帰呼び出しを使っているのである．リスト 4(教科書 p.8-9 List 1-3) クイックソートの関数は，再帰呼び出しのアルゴリズムとなっている．この関数の動作は単純で，

1. 並べ替えるべき数列の先頭と末尾が，同じか末尾の方が前ならば (`if(bottom>=top)`) ならば，何もしない．並び替えの必要なし．
2. `bottom<top` ならば，その範囲を基準値より大きいものと小さいもの分ける．そして，基準値を大きい組と小さい組の間に入れる．
3. 再帰呼び出しにより，小さい組で同じ処理をする (26 行目).
4. 再帰呼び出しにより，大きい組で同じ処理をする (27 行目).

となっている．

リスト 4: クイックソートの関数

```
1   void QuickSort(int bottom,int top,int *data)
2   {
3       int lower,upper,div,temp;
4       if(bottom>=top)
5           return;
6       /* 先頭の値を「適当な値」とする */
7       div=data[bottom];
8       for(lower=bottom,upper=top;lower<upper;)
9       {
10          while(lower<=upper&&data[lower]<=div)
11              lower++;
12          while(lower<=upper&&data[upper]>div)
13              upper--;
14          if(lower<upper)
15          {
16              temp=data[lower];
17              data[lower]=data[upper];
18              data[upper]=temp;
19          }
20      }
21      /* 最初に選択した値を中央に移動する */
22      temp=data[bottom];
23      data[bottom]=data[upper];
24      data[upper]=temp;
25
26      QuickSort(bottom,upper-1,data);   /* 数列の前半について，再帰呼び出し */
27      QuickSort(upper+1,top,data);      /* 数列の後半について，再帰呼び出し */
28  }
```

# 3　再帰呼び出しを使うためには

## 3.1　再帰呼び出しの関数の条件

これまでの例で，再帰呼び出しは繰り返し文と似ていることが分かっただろう．繰り返し文では，無限ループにならないように，必ず終了条件が必要であった．同じように，再帰呼び出しでも，呼び出しの終了条件が絶対に必要である．そうしないと，無限に関数を呼び出すことになり，いずれはプログラムがクラッシュするであろう．

もうひとつ重要なことは，再帰呼び出しが出きるようなアルゴリズムを考えなくてはならない．一つの考え方は，漸化式を書いてみることである．数学の漸化式のような考え方が出きれば，それを忠実にプログラムすれば，再帰呼び出しができるであろう．

## 3.2　再帰呼び出しを使う方がよいか?

結論から先に言うと，再帰呼び出しは出来るだけ使わない方が良い．繰り返し文で可能ならば，再帰呼び出しをわざわざ使う必要はない．なぜならば，再帰呼び出しは関数を何回もコールするため，速度は遅くなり，さらにメモリーも多く使う．しかし，プログラムの記述が簡素になる場合は，再帰呼び出しを使うべきである．ハノイの塔やクイックソートを再帰呼び出しを使わないで，プログラムを書くとなると，かなり手間がかかるだろう．とても難しく思えたハノイの塔を解くプログラムが，びっくりするほど簡単に書けるのは再帰呼び出しの威力である．

# 4　教科書の例

## 4.1　1 の数を数えるプログラム

教科書 [2] の最初の例は，整数を入力して，その中に含まれている 1 の数を表示するプログラムである．たとえば，入力した整数が 4513101 ならば， 3 と表示するのである．

これを実現するためには，整数の中にふくまれている 1 の数を数える必要がある．教科書では以下の手順で，それを行っている．

1. 整数の最下位の桁 (一の位) の値を取り出している．ある整数を 10 で割った余りが最下位の桁の値となる．ある整数を `value` として，剰余演算子`%`をつかって，`value%10` の演算により最下位の桁を取り出している．たとえば，`value` の値が 4513101 の場合，`value%10` の演算の結果は 1 となる．

2. 最下位の桁を取り出したので，`value` の他の桁を右に 1 桁ずつシフトさせる． 4513101 を 451310 とするのである．これは，除算演算子を使って，`value/10` により実現できる．コンピューターでは整数同士のわり算は整数になり，あまりは切り捨てられるからである．

3. `value` の値がゼロになるまでこれを繰り返し，最下位の桁を調べ，1 の数をカウントする．

### 4.1.1 繰り返し文を使う場合

繰り返し文である for を使って，1 の数を数えるプログラムが教科書の List 5-1(p.139-140) である．それを，リスト 5 に載せる．これは，9-11 行目にわたっての繰り返し文の実行順序が分かれば，プログラムの動作は分かるであろう．

リスト 5: 繰り返し文を使った 1 の数を数えるプログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int num_of_one(int value)
5  {
6      int ret;
7      /* valueを次々に10で割って桁をずらしながら，
8          最下位の桁が1であるかどうかを調べていく */
9      for(ret=0;value >0;value /=10)
10         if(value%10==1)
11             ++ret;
12     return ret;
13 }
14
15 int main(void)
16 {
17     int i;
18
19     scanf("%d",&i);
20     printf("%d中に1は%d個含まれています\n",i,num_of_one(i));
21     return EXIT_SUCCESS;
22 }
```

### 4.1.2 再帰呼び出しを使う場合

次に，同じ処理をするプログラムを再帰呼び出しを使った例が，教科書の List 5-2(p.140)，プリントのリスト 6, 7 である．最下位の桁を調べると，value の桁を一つずらして，同じことが続くので，再帰呼び出しを行っている．これも，階乗を計算したときの漸化式のようなものを書くと，以下のようになる．

- 探索すべき整数 value の値がゼロであれば，処理は終了．
- 漸化式は，以下のように書ける．

$$(\texttt{value}\text{の}1\text{の数}) = (\texttt{value}\text{の最下位の桁の}1\text{の数}) + (\texttt{value}\text{の桁を右に一桁ずらした値の}1\text{の数}) \tag{6}$$

この漸化式もどき通りに C 言語の文法を用いて記述すれば，再帰呼び出しのプログラムができあがるのである．

このプログラムで使われている主なテクニックは，以下の通りである．

- 1 行目 unsigned long value
  これは，unsigned long int value と同じで，変数 value の型を符号無し倍々精度整数で定義 (予約) している．こうすると，value は 32 ビットで，0〜4294967295 の整数値を使うことができる．

リスト 6: 再帰呼び出しを使った 1 の数を数える関数

```
1  int num_of_one(unsigned long value)
2  {
3      int ret;
4      /* valueが0桁（もうこれ以上解析する桁がない） */
5      if(value==0)
6          return 0;
7      if(value%10==1)       /* いちばん下の位が1 */
8          ret=1;
9      else
10         ret=0;
11
12     /* 10で割って桁を1つずらし，再びnum_of_one()で調べる */
13     return ret+num_of_one(value /10);
14 }
```

リスト 7はリスト 5と同じプログラムではあるが，条件演算子[1]「?」を使ってコンパクトに記述している．このプログラムで使われている主なテクニックは，以下の通りである．

- 5 行目 `((value%10)==1) ?1:0`

  条件演算子?は，

  条件式 ? 式 1 : 式 2

  と書き，条件式が真ならば式 1 を，偽ならば式 2 を値として返す．

リスト 7: 再帰呼び出しを使った 1 の数を数える関数

```
1  int num_of_one(unsigned long value)
2  {
3      if(value==0)
4          return 0;
5      return ( ((value%10)==1) ?1:0) +num_of_one(value/10);
6  }
```

## 4.2 最大公約数を求める

与えられた複数の整数の最大公約数を求めるプログラムである．

### 4.2.1 繰り返し文

教科書の List 5-3(p.143) のプログラム，プリントのリスト 8 は繰り返し文を用いて，最大公約数を求めている．この方法は単純で，最大公約数を求める 2 つの整数のあたいのいずれかを選んで，それで割った余りがゼロになるか調べている．そして，割るべき整数を一つずつ減らして，最初に割り切れた整数を最大公約数としている．

[1]教科書では 3 項演算子と言っている．

リスト 8: 2 つの整数の最大公約数を求める関数

```
1  int gcd(int a,int b)
2  {
3      int i;
4      for(i=a;i>0;i--)
5          if(a%i==0 && b%i==0)
6          break;
7      return i;
8  }
```

### 4.2.2 再帰呼び出し

教科書の List 5-4(p.145) のプログラム，プリントのリスト 9 は再帰呼び出しを用いて，2 個以上の整数の最大公約数を求めている．リスト 9 の関数 `multi_gcd(int n)` は，配列 `N[]` に格納されている整数 `N[0]`〜`N[n]` の最大公約数を求める関数である．このプログラムの内容はそれほど難しくないので，各自，動作を考えること．

このプログラムも漸化式のようなものを考えると，次のようになる．

- 公約数を求める整数の組が 2 つの場合は，先の `gcd()` 関数を呼び出して，2 つの整数の最大公約数を求める．
- 漸化式は以下のように書ける．

$$\text{複数最大公約数 (N[0]〜N[n])} = 2\text{ 整数最大公約数 [N[n], 複数最大公約数 (N[0]〜N[n-1])]} \quad (7)$$

リスト 9: 再帰呼び出しを使い複数の整数の最大公約数を求める関数

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 6
5  int N[MAX]={98,140,84,28,42,126};
6
7  int gcd(int a,int b)
8  {
9      int i;
10     for(i=a;i>0;i--)
11         if(a%i==0&&b%i==0)
12             break;
13     return i;
14 }
15
16 int multi_gcd(int n)
17 {
18     /* n==1(数が2つしかない)の場合は，普通にgcdを呼ぶだけ */
19     if(n==1)
20         return gcd(N[0],N[1]);
21
22     /* n>1の場合は，N[n]と，N[0]..N[n-1]のgcd を呼び出す */
23     return gcd(N[n],multi_gcd(n-1));
24 }
25
```

```
26 int main(void)
27 {
28     printf("配列Nの最大公約数は%dです\n",multi_gcd(MAX-1));
29     return EXIT_SUCCESS;
30 }
```

# 5 練習問題

[**問 1**] 次に示す数列 (フィボナッチ数列) の $F_7$ の値は，いくつか?．値のみならず，計算過程も示せ．

$$F_k = F_{k-1} + F_{k-2} \qquad F_0 = 0 \qquad F_1 = 1$$

[**問 2**] $n$ の階乗[2]を再帰呼び出しで計算するための関数 $F(n)$ を以下に示すが，$\square$ の内容を選択肢から選べ．

$$F(0) = 1 \qquad F(n) = \square$$

選択肢

$F(n) \times F(n-1)$　　$F(n-1) \times F(n-2)$

$n \times F(n-1)$　　$(n-1) \times F(n)$

[**問 3**] 問 1 のフィボナッチ数列を計算するための，再帰呼び出しを使った関数を作成せよ．

[**問 4**] 問 2 の $n$ の階乗を計算するための，再帰呼び出しを使った関数を作成せよ．

[**問 5**] これは応用なので，分からない者は解答する必要はないが，おもしろい問題なので興味のある者は調べよ．

- フィボナッチ数列は，どのような場面で登場するか?
- フィボナッチ数列の $k$ が非常に大きい場合の計算方法を調べよ．(ヒント：固有値と固有ベクトルを使う)

## 5.1 レポート提出要領

提出方法は，次の通りとする．

[2] $n$ は非負の整数で，$n$ の階乗を $n!$ と書く．$n! = n(n-1)(n-2)\cdots 2 \times 1$ である．

期限　　1 月 16 日 (月) AM 10:40
用紙　　A4
提出場所　山本研究室の入口のポスト
表紙　　表紙を 1 枚つけて，以下の項目を分かりやすく記述すること．
　　授業科目名「情報工学」
　　課題名「課題　再帰呼び出し」
　　2E　学籍番号　氏名
　　提出日
内容　　2 ページ以降に問いに対する答えを分かりやすく記述すること．

# 6　付録

これは重要な話ではあるが，これを講義で述べると寝てしまう (思考が停止する) 者が多数でそうである．興味のある者は，ここの付録をしっかり学習せよ．大学への編入試験問題では，この程度の問題は出題される．

## 6.1　ハノイの塔の円盤の移動回数

図 3 に示したように，円盤が 3 枚の場合，円盤の移動回数は 7 回であった．仮に，インドの坊主が 1 秒間に 1 枚の円盤を移動させているとすると，わずか 7 秒で世界の終わりを迎えることになる．幸いなことに，円盤は 64 枚あり，もう少し世界の寿命は長そうである．それでは，世界の寿命はどの程度であろうか?.

$n$ 枚の時の円盤の移動回数 $f_n$ は、

$$f_n = f_{n-1} + 1 + f_{n-1} \tag{8}$$

となる。リスト 3 の関数 `move()` を 1 回呼び出すと、必ず円盤の移動が一回発生する。加えて、$n-1$ 枚の呼び出しを 2 回行っているからである。この式を変形すると、

$$f_n + 1 = 2(f_{n-1} + 1) \tag{9}$$

となる。これは等比数列なので、一般項は簡単に求められる。$f_1 = 1$ なので、

$$f_n + 1 = 2 \times 2^{n-1} \tag{10}$$

となる。したがって、円盤の移動回数 $f_n$ は、

$$f_n = 2^n - 1 \tag{11}$$

と表すことができる。

$n = 64$ のときの，移動回数は 18446744073709551615 回である．世界の終焉は，

$$18446744073709551615 \text{ 秒} = 2.13504 \times 10^{14} \text{日} = 5.89492 \times 10^{11} \text{年}$$

となる．5 千 8 百億年後と言うことである．現在の宇宙の年齢は，およそ 150 億年と言われていることを考えると，これは長すぎるように感じる．

エドゥアール・リュカも少し間違えたようで，円盤の数を 60 枚程度にしておけば，世界の寿命は 365 臆年となり，いい線をいっているように気がする[3]．どちらにしても，世界の終わりなんか分からないから，どうでも良いか．いつか，物理学が進歩して世界の終わりが予想できたならば，インドの坊主の仕事と比べてみたいものである．

[3]エドゥアール・リュカの生きた時代では現在の宇宙の寿命など分からなかったので，仕方なかった．64 というのは，2 の 6 乗で区切りはよいが ····

## 参考文献

[1] ハノイの塔. http://www.geocities.jp/chokyumei/tankoubon/hanoi.html.

[2] 春日伸弥紀平拓男. プログラミングの宝箱　アルゴリズムとデータ構造. ソフトバンクパブリッシング(株), 2004 年.