# C++講習会 #4 反復処理と動的計画法(DP)

高校 2 年 赤澤侑

2020/09/24

## 1 反復処理

### 1.1 while 文

皆さんは今まで反復処理を書くときには次のようなソースコード 1 を書いていたことでしょう．

ソースコード 1: for ループ

```
for(int i = 0; i < N; ++i){
  //ここに処理をかく
}
```

実際，殆どの繰り返し処理はこの for ループを書くことができれば事足りてしまいます．然しながら，例えば，ある条件を満たす場合はずっとループし続けるといった処理をしたい場合に for ループを使用すると次のようになってしまい少しばかり冗長…というよりもわかりにくくなってしまいます．

ソースコード 2: for ループで”条件を満たす間ずっと繰り返し”

```
for(; (条件式); ){
  //ここに処理をかく
}
```

そこで次のような文を書きます．これは whlie 文と言って最初に条件式を評価して，それが true であれば文の中身を実行します．それではソースコード 3 を見てください．

ソースコード 3: while 文

```
while(条件式){
  //ここに処理をかく
}
```

うん，ずっとスッキリしましたね．

**課題 1** while 文をつかってみよ

また while 文の仲間としてソースコード 4 のような書き方があります．この文を do-while 文といいます．此れはまず最初に文の中身を実行して最後に条件式を評価します．そしてそれが条件を満たせばもう一度，文の中身を実行する……といった具合に処理が進みます．while 文との違いを明確にしておくと，それは最低でも文の中身を一回は実行するという点です．while 文では最初から条件を満たさない場合，処理を行いませんが，do-while 文を使うと最初から条件を満たさない場合にも処理を行います．競技プログラミングに於いてはソースコード 5 のように書くことで順列全列挙をする際に多く用いられる文でもあります．なお，do-while 文では最後にセミコロンを忘れずに入れるようにしましょう．

ソースコード 4: do-while 文

```
do{
  //処理
}while(条件式);
```

ソースコード 5: do-while 文による順列全列挙

```
int N; //配列の要素数
int a[N]; //適当に初期化する
do{
  //処理
}while(next_permutation(a, a+N));
```

**課題 2** do-while 文では最初に 1 回絶対に処理が行われることを確認せよ

**課題 3** 1〜4 を入れた要素数 4 の配列 a に対して順列全探索をしてみよ

**課題　Verify-while**

A. ある自然数 a と b が与えられる．a と b の最大公約数を求めよ．$(0 < a, b < 10000000)$

B. ある自然数 $N$ が与えられる．$N$ の各桁の和を求めよ．($N$ は 200000 桁以下)

C. 10 進数で表現されたある自然数 $N$ を二進数表記せよ．($N$ は 200000 以下)

D. 相異なる要素をもつ数列 a と整数 $K$ が与えられる．a を並べ替えて得られる $N$ 桁の数字の中で $K$ は小さい方から何番目か求めよ．($N \leq 8$, $a_i \neq a_j (1 \leq i,\ j \leq N,\ i \neq j)$, $K$ は $N$ 桁の自然数)

## 1.2　再帰関数

唐突ですが $n$ 番目のフィボナッチ数を求めたくなりました．此れを求めるソースコードを書いてみましょう．その前に，まずはフィボナッチ数の定義を確認します．$n$ 番目のフィボナッチ数を $fib\,(n)$ と表記すると，フィボナッチ数は再帰的に

$$fib(n) = \begin{cases} 1 & (n = 0\ or\ n = 1) \\ fib(n-1) + fib(n-2) & (otherwise) \end{cases} \tag{1}$$

と定義されます．此れをそのままソースコードにしてみたいです．ソースコード 6 を見てください．long long 型の fib という関数が宣言されているのがわかるでしょう．fib は引数に int 型の $N$ という数をとります．此れは求めたいフィボナッチ数が先頭から何番目かを意味します．そして (1) で示した定義がそのまま実装されています．実際にプログラムを実行してみると例えば 4 という入力に対しては 5，6 に対しては 13 と返ってくることから問題なく動作していることが判ります．

ソースコード 6: フィボナッチ数を求める

```
#include <iostream>
using namespace std;

long long fib(int n){
        if(n == 0 || n == 1)return 1;
        else return fib(n-1) + fib(n-2);
}

int main(){
        int N;
        cin >> N;
```

```
        cout << fib(N) << endl;
        return 0;
}
```

このようにある関数の中で自分自身を呼び出すことを再帰呼び出しといい，そういった関数を再帰関数といいます．例えば次のようなものも再帰関数で表現することができます．

- 階乗の計算…… $n!$ を $fact(n)$ とすると

$$fact(n) = \begin{cases} 1 & (n = 1) \\ n \times fact(n-1) & (otherwise) \end{cases} \tag{2}$$

- 繰り返し 2 乗法……[1] $a^x$ を $O(\log x)$ 程度で求めるアルゴリズム．これを $pow(a, x)$ とすると

$$pow(a, x) = \begin{cases} 1 & (x = 0) \\ (pow(a, \lfloor x/2 \rfloor))^2 \times a & (x\ is\ odd) \\ (pow(a, x/2))^2 & (x\ is\ even) \end{cases} \tag{3}$$

慣れないうちは少し変な感じがしてしまうかもしれません．また，再帰呼び出しの何が良いのかわからない方も多いでしょう．実装の難易を度外視して言ってしまえばある処理を再帰で書くこととループを使って書くことはほとんど違いがありません．然しながら次節で説明する動的計画法など複雑な状態遷移をする場合や数学的な定義をそのままプログラムに落とし込む場合など再帰関数を用いたほうがわかりやすい，あるいは，書きやすいコードになる場合も多いです．重要なものなので是非身につけましょう．

また，再帰関数は意図せず無限ループを生みやすいです．これを回避するために終了条件をしっかり意識して書くことが肝要です．

**課題 4** フィボナッチ数を求めるプログラムを実装してみよ
**課題 5** 階乗の計算を実装してみよ
**課題 6** 繰り返し 2 乗法を実装してみよ

## 課題 Verify-再帰関数

A. ある自然数 $N$ が与えられる．1 から $N$ までの和，即ち $\sum_{i=1}^{N} i$ を求めよ．$(0 < N < 100000)$

B. @$\sum$ さんはアルファベット a,c,f,g,i,m,p,s,t,u,y が好きである．彼は身の回りのものにこれらのアルファベットだけを使って 5 文字からなる名前を付ける．彼は名前が重複しないように辞書順で最小の名前から順に使用することにした (aaaaa → aaaac → aaaaf といった具合だ)．このとき atsum は何番目の名前になるか？

C. $N$ 個の整数からなる数列 a が与えられる．数列 a に含まれる要素をいくつか使ってその和を K にすることはできるか？$(1 \le N \le 20,\ |a_i| < 2 \times 10^8\ (1 \le i \le N),\ |K| \le 2 \times 10^8)$

[1]繰り返し 2 乗法はコンセプトをそのままプログラムに落とすとこのようになりますが実はあまり高速に動作しません．そこで bit 演算 (今後扱います) と組み合わせて以下のような実装をすると高速化します．

```
long long modpow(long long x, long long n) {
   long long ret = 1;
   while (n > 0) {
     if (n & 1) ret = ret * x % MOD;
     x = x * x % MOD;
     n >>= 1;
   }
   return ret;
}
```

# 2 動的計画法

## 2.1 再帰関数の計算量

再度フィボナッチ数を考えてみましょう．試みに先程書いた $N$ 番目のフィボナッチ数を求めるプログラムに 50 と入力してみてください．どうでしょうか？おそらく 50 と入力して以降何も表示されないでしょう．実は再帰関数というのはナイーヴに実装するととんでもない計算量[2]になってしまうことがあります．どうしてこうなるのかというと，何度も同じ計算をしているからです．図 1 を見てください．

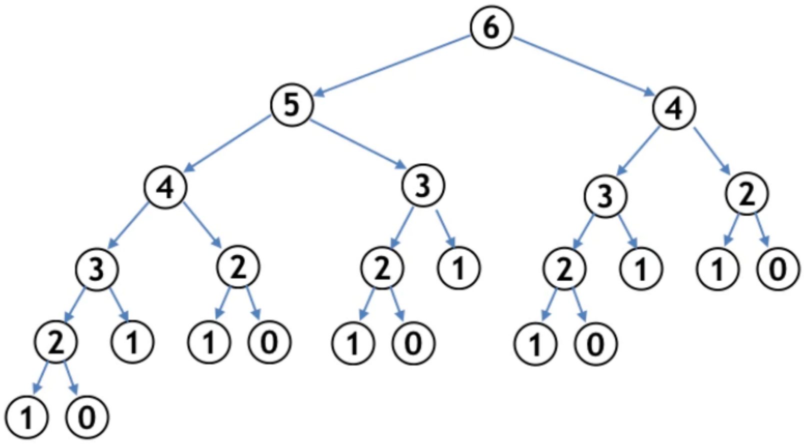


図 1: 再帰呼び出し (drken『再帰関数を学ぶと、どんな世界が広がるか』より)

これは fib(6) を求めるときにどのような呼び出しが行われているかを表したものです．こうしてみてみるとすでに呼び出したことのある数に対して，以降の呼び出しは全く同じ形をしていることが判ります．即ち，すでに計算したものを何度も何度も呼び出しているのです．このように再帰関数は実装方法によっては指数時間になってしまうことがあります．

**課題 7** 再帰関数による階乗計算の計算量を求めよ

**課題 8** 繰り返し 2 乗法の計算量を求めよ（尤もすでにどこかで書きましたが……）

## 2.2 メモ化再帰

これを回避するためにメモ化再帰という手法があります．メモ化と言っても一度計算した値を保存しておいて利用するというものです．ソースコード 7 を確認してください

ソースコード 7: メモ化再帰

```
#include <iostream>
using namespace std;

long long memo[100000] = {};

long long fib(int n){
        if(memo[n] != 0)return memo[n];
        if(n == 0 || n == 1)return 1;
```

[2] フィボナッチ数を求める再帰関数の計算量は今回の実装で $O((\frac{1+\sqrt{5}}{2})^n)$ です．これは $O(2^N)$ と同じ指数時間アルゴリズムです．どうしてこの計算量になるか氣になる方は http://www.aoni.waseda.jp/ichiji/2012/java-01/java-14-3.html を参照すると良いでしょう．

```
        else return memo[n] = fib(n-1) + fib(n-2);
}

int main(){
        int N;
        cin >> N;
        cout << fib(N) << endl;
        return 0;
}
```

long long 型の memo という配列は $k$ 番目のフィボナッチ数を格納しています．再帰関数の方はすでに調べた値 (memo の値が 0 以外であるもの) については計算せずにその値を返すように変更しました．こうすることによって調べないといけないものは高々$N$ 通りしかなくなるので $O(N)$ になります．

このように部分的な問題の答えを求めておきそれを利用することで大きな問題に対する答えを与えるような手法を動的計画法 (DP; Dynamic Programing) と言います．メモ化再帰は DP を実現する手法の一つです．

**課題 8** メモ化再帰を用いてフィボナッチ数を計算するプログラムを書いてみよ

**課題 9** 課題 8 で書いたプログラムに 50 という値を入れると答えを返すことを確認せよ

**課題　Verify-メモ化再帰**

A. トリボナッチ数とは次の漸化式によって定義される数である ($n$ 番目のトリボナッチ数を $trib(n)$ とする)．$N$ 番目のトリボナッチ数を $10^9+7$ で割った余りを求めるプログラムを記述せよ．$(0 \leq N \leq 1000000)$

$$trib(n) = \begin{cases} 0 & (n = 0 \; or \; n = 1) \\ 1 & (n = 2) \\ trib(n-1) + trib(n-2) + trib(n-3) & (otherwise) \end{cases} \tag{4}$$

B. $N$ 個の正整数からなる数列 a が与えられる．数列 a に含まれる要素をいくつか使ってその和を K にすることはできるか？$(1 \leq N \leq 100,\ 0 < a_i \leq 10^3\ (1 \leq i \leq N),\ 0 < K \leq 10^4)$

## 2.3　ループで回す DP テーブル

ここまでの過程に疑問を持った方はいないでしょうか？というのもフィボナッチ数が漸化式 (1) によって与えられるなら純粋な for ループで記述できるのではないかと考えられるからです．実際にプログラムを書いてみましょう．ソースコード 8 にそのようなプログラムの例を示しておきます．

ソースコード 8: for ループで回す

```
#include <iostream>
using namespace std;

int main(){
        int N;
        cin >> N;
        long long fib[100000];
        fib[0] = fib[1] = 1;
        for(int i = 2; i <= N; ++i){
                fib[i] = fib[i-1] + fib[i-2];
        }
        cout << fib[N] << endl;
        return 0;
}
```

**課題 10** ソースコード 8 を書き，実際の動作を確認してみよ

この方法を用いると計算量は $O(N)$ になります[3]．ここで fib という配列を使いましたが，このような部分的な解を記録している配列を DP テーブルといい，ループを用いた DP ではこの DP テーブルを更新していくことで解に至ります．

**課題 Verify-DP**

A. 忍者の@$\sum$ くんはビル 1 からビル $N$ までジャンプして移動したいです．ただし彼はジャンプ力がないので "いま立っているビルの高さ+$K$" 以下の高さのビルにしか飛び移ることしか出来ません．ビルは番号が小さい方から大きい方にしか飛び移れないとすると@$\sum$ くんが目的を達成できるジャンプのやり方は何通りあるでしょうか．答えは非常に大きくなりうるので $10^9+7$ で割った余りを求めてください．

- $1 \leq N \leq 1000$
- $0 \leq K \leq 100$
- $0 \leq h_i \leq 1000$ $(1 \leq i \leq N)$

B. 運動選手の@$\sum$ くんは階段を登るトレーニングをしています．階段は $N$ 段あり，うち $M$ 段は壊れてしまって踏むことができません．@$\sum$ くんは $K$ 段まで飛ばして飛ぶことができる時，彼が $N$ 段登りきるやり方は何通りありますか？$N$,$K$,$M$ と壊れてしまった階段が何段目か ($s_i$ とします) が与えられます．答えは非常に大きくなりうるので $10^9+7$ で割った余りを求めてください．

- $1 \leq N \leq 1000$
- $0 \leq M \leq N$
- $0 \leq K \leq 6$
- $1 \leq s_i < N$ $(1 \leq i \leq M)$

## 2.4 ナップサック問題とその亜種について

DP を用いて解くことのできる有名な問題にナップサック問題というものがあります．問題の概要はこうです．

**【ナップサック問題】**

商品が $N$ 種類 (各商品は一つずつ) あります．$i$ $(1 \leq i \leq N)$ 番目の商品は重さが $w_i$ で価値が $v_i$ です．あなたは重さが $W$ を超えないように商品を買うことができます．買う商品の価値の合計の最大値はいくらですか？

この問題は少し難しいので後々解説しますが, 余力があれば考えてみてください．ヒントを与えると DP テーブルは二次元になり, dp[i][j] := i 番目までの商品を買って重さが j の時の価値の最大値として考えます．

[3] なお，今回は扱いませんが $N$ 番目のフィボナッチ数は $O(logN)$ で求めることができます．詳しくは調べてほしいですが (僕も詳しいことはわからないです)，フィボナッチ数は行列のべき乗として表現することができるので先程あげた繰り返し 2 乗法のお気持ちで上記の計算量を達成できそう，ということが判っていただければ良いと思います．