第 5 章

漸化式と動的計画法，数え上げ

2017-10-25 Wed

お知らせ (再掲)

poj.org や szkopul.edu.pl のアカウントは aoj とは別なので、適宜作成してください。なお，どのサイトも好意で運営されているので，迷惑をかけないこと (パスワードをなくさないこと).

概要

全ての可能性を調べ尽くすことが難しいような問題について，小さな問題を予め解いて解を表に覚えておくなどの整理を適切に行うことで簡単に解けるようになる場合もある．大小の問題の関係を表す式を立てて，それをプログラムにしてみよう．動的計画法は，問題が持つ部分構造最適性を利用して効率の良い計算を実現する方法である．

5.1 数を数える

例題 **Fibonacci Number** (AOJ)

N 番目の Fibonacci 数を求めよ。
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_A

簡単のため i 番目の Fibonacci 数を F_i と表記すると，以下のように定義される ($F_0 = 0$ と定義することも多いが本質ではないので問題文に合わせる):

$$F_i = \begin{cases} 1 & i = 0 \\ 1 & i = 1 \\ F_{i-1} + F_{i-2} & \text{(otherwise)} \end{cases} \tag{5.1}$$

上記の定義をそのまま再帰で定義すると以下のようになるだろう:

Ruby

```ruby
def fib(n) # 遅いバージョン
  # p ["fib",n] 関数呼び出しの際に引数を表示
```

```
3    if n==0
4      0
5    elsif n == 1
6      1
7    else
8      fib(n-2)+fib(n-1)
9    end
10 end
```

しかし，この方法は計算の重複が多く，効率的でない．改善方法の一つはメモ化により一度計算した答えを記憶しておくことである (プログラミング演習参照).

余談

フィボナッチ聖人の彫像 (Camposanto, Pisa)

今回主として扱う，より素直な方法は，小さいフィボナッチ数から順に計算することである．式 (5.1) から，F_i を求めるには，F_0 から F_{i-1} までの値のみが必要であり，それらがあれば加算 1 回で計算できるという関係がわかる．そこで F_0 から順に計算すれば，各 F_i は加算 1 回で求められる．

C++

```
1  int F[100]; // 必要なだけ
2  int main() {
3    F[0] = 1;
4    F[1] = 1;
5    for (int i=2; i<45; ++i) { // 必要なだけ
6      F[i] = F[i-2]+F[i-1];
7    }
8  }
```

今回の場合は，回答にあたって直接必要なものは N 番目のデータだけだが，一見回り道のようでも N 番目*まで*のデータを全て計算しておくアプローチが有効である．全体として必要な基本演算の回数 (以下，単に計算量と表記する) は $O(N)$ である．なお，繰り返し自乗法を使うことで $O(\log N)$ に改善することもできるので[*1]，興味のある人は 5.A 節を参照されたい．

この節の目標は，式 (5.1) のような漸化式から，対応するプログラムを書けるようになることである．

[*1] 簡単のために，この資料では整数演算のコストを定数と扱っている．しかし，多倍長整数を使う場合は，大きな数の演算は遅くなることに注意.

例題 **Kannondou** (PC 甲子園 2007)

階段を 1 足で 1,2,3 段上がることができる人が，n (< 30) 段登るときの登り方の種類を，全て試したとして何年かかるかを求めよ (詳細は原文参照).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0168

各段への登り方の数を配列で表現しよう．漸化式は，i 段に居るために A_i 通りの登り方があるとすると．登る前は $A_0 = 1$, 答えは A_n:

$$A_i = \begin{cases} 1 & i = 0 \\ A_{i-1} & i = 1 \\ A_{i-1} + A_{i-2} & i = 2 \\ A_{i-1} + A_{i-2} + A_{i-3} & \text{(otherwise)} \end{cases} \tag{5.2}$$

フィボナッチ数の計算と同様に，N 段に対する答えを求める計算量は $O(N)$ となる．

配列の範囲外アクセスに注意


C や C++ では配列の範囲外，たとえば `A[-1]`，を参照したり書き込んだりしてはいけない (すぐに実行時エラーになるかもしれないし，ならないかもしれないし，もっと困ったことをしでかすかもしれない)． 典型的には，`A[i-2]` にアクセスする文を書いたら，`i`≤ 1 では実行されないことをプログラマが保証する必要がある．

回答例 (入出力)

C++

```
1    while (cin >> N && N)
2        cout << ((A[N]+..)/10+...)/365 << endl;
```

整数除算で切り上げるには，割る前に (除数-1) を足せば良い (整数除算 `(x+9)/10` が，$x \in [0, 9]$ に対してどのような結果になるかを確認しよう).

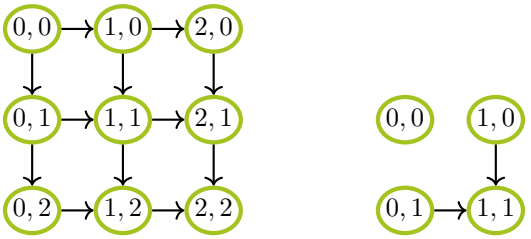
答えが合わない場合の参考


15 段の場合は 5768 通りあり，2 年かかる．

問題 **平安京ウォーキング** (UTPC2009)

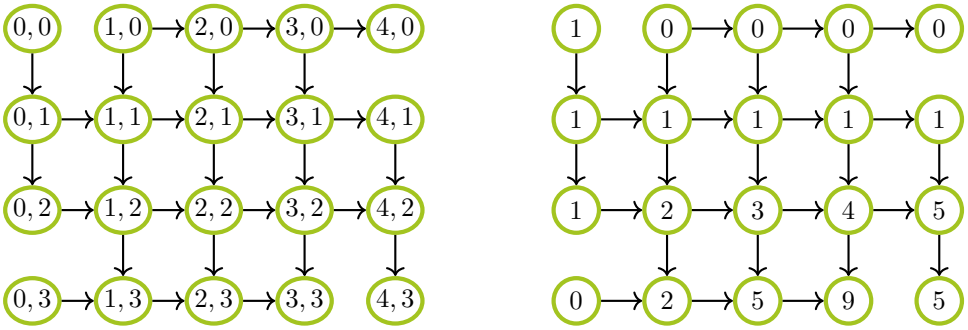
グリッド上の街をスタートからゴールまで，(ゴールに近づく方向にのみ歩く条件で) 到達する経路を数える．ただし，マタタビ (猫にとっての障害物) の落ちている道は通れない．
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2186
サンプル入力の初めの 2 つは次の状況を表す。

(問題 続き)



なお，マタタビがなければ組み合わせ (目的地までに歩く縦横の道路の合計から縦の道路をいつ使うか) の考え方から，直ちに計算可能である．

マタタビがある場合は，交差点毎に到達可能な経路の数を数えてゆくのが自然な解法で，サンプル入力 3 つめの状況は，下図左の接続状態を持ち，各交差点は下図右の到達経路数を持つ．



交差点の座標と通れる道 (右または下に移動可)　　各交差点に到達可能な経路の数

ある交差点に (x, y) に到達可能な数 $T_{x,y}$ は，そこに一歩で到達できる交差点 (通常は左と上) 全ての値が定まっていればそれらの和として計算可能である．

$$T_{x,y} = \begin{cases} 0 & (x,y) \text{ が範囲外} \\ 1 & (x,y) = (0,0) \\ 0 & \text{上にも左にもマタタビ} \\ T_{x-1,y} & \text{上のみにマタタビ} \\ T_{x,y-1} & \text{左のみにマタタビ} \\ T_{x-1,y} + T_{x,y-1} & \text{上にも左にもマタタビなし} \end{cases}$$

マタタビの入力が多少冗長な形式で与えられるので，以下のように前処理して，上と左からそれぞれ移動可能かどうかを示す配列に格納しておくと使い勝手が良い．

- x 座標が同じ – (x1, max(y1,y2)) には上から移動不可
- y 座標が同じ – ((max(x1,x2), y1) には左から移動不可

たとえば，上と左からの移動できるかどうかを `Vert[x][y]` と `Horiz[x][y]` で管理する．ある点 (x,y) に上から移動可能であるなら `Vert[x][y]==true`，そうでなければ `Vert[x][y]==false` とする．同様に，左から移動可能かで `Horiz[x][y]` の各要素を設定する．あらかじめ各要素を `true` に初期化してお

き，マタタビがあった場合は false に書き換える．(移動*不可能*であることを表す場合は true と false の対応が逆になる．混乱しなければどちらでも良い．)

入力例

Ruby

```ruby
dataset = gets.to_i
dataset.times {
  gx, gy = gets.split("␣").map(&:to_i)
  p [gx,gy]
  matatabi = gets.to_i
  # 問題文では変数 p を使っているが，表示コマンドの p と名前を分けた
  (0..matatabi-1).each{|i|
    x1,y1, x2,y2 = gets.split("␣").map(&:to_i)
    p [x1,y1, x2,y2]
  }
}
```

プログラム作成手順のお勧め

各交差点に到達可能な経路の数を，前ページ図右のように全て表示し，手計算と一致するかどうかを確認しよう．

答えが合わない場合のヒント

またたびの縦横，上端や左端の扱い，$x_1 \leq x_2$ とは限らないこと (y_1, y_2 も) などに注意．

番兵法 (sentinel): 見通しの良いプログラムのヒント

“Kannondou” の例題同様に，x==0 で T[x-1][y] にアクセスしないようにすることと，y==0 の時 T[x][y-1] にアクセスしないようにする必要がある．if 文で書くこともできるが，上端と左端に仮想的なマタタビがあると考えて Vert[x][0] と Horiz[0][y] を設定すると，簡潔に書くことができる．

5.2　さまざまな動的計画法

■最長共通部分列と編集距離　文字列などデータ列から，一部の要素を抜き出して並べた列 (あるいは一部の要素を消して詰めた列) を部分列という．二つのデータ列に対してどちらの部分列にもなっている列を共通部分列という．共通部分列の中で長さが最大の列を最長共通部分列と言う．最長共通部分列は複数存在する場合がある．

問題	**Dynamic Programming - Longest Common Subsequence**	(AOJ)

最長共通部分列の長さを求めよ

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_10_C

文字列の長さを N とすると，部分列の候補は 2^N 通りあるので，全て試すと効率が悪い．以下のように工夫して $O(N^2)$ で計算すると良い．

考え方: 文字列 X の先頭 i 文字部分を切り出した部分列を X_i と表す (この式では, 文字列の先頭を 1 文字目としていることに注 多くのプログラミング言語とは 1 ずれている)．X_0 は空文字列とする．文字列 X と文字列 Y の最長共通部分列は，その部分問題である X_i と Y_j の最長共通部分列の長さ $L_{i,j}$ を利用して求めることが出来る．

$$L_{i,j} = \begin{cases} 0 & i \leq 0 \text{ または } j \leq 0 \\ 1 + L_{i-1,j-1} & \text{X の i 文字目と Y の j 文字目が同じ文字} \\ \max(L_{i,j-1}, L_{i-1,j}) & \text{otherwise} \end{cases}$$

この計算は二次元配列を利用して，効率よく行うことが出来る．

問題	**Combinatorial - Edit Distance (Levenshtein Distance)**	(AOJ)

二つの文字列の編集距離を求めよ．

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_E&lang=jp

■最適経路を求める　最適値を動的計画法で求めるだけではなく，最適値を実現する具体的なプランを構成したい場合もある．多くの場合には，ほぼ同じ手間で求めることができる．

問題	**Spiderman**	(Tehran 2003 Preliminary)

指定の高さ H[i] だけ登るか降りるかを繰り返すトレーニングメニューを消化して地面に戻る．メニュー中で必要になる最大の高さを最小化する．

http://poj.org/problem?id=2397

この問題では，合計値ではなく最小値が必要とされる．

i 番目の上下移動を H_i(i は 0 から)，i 番目のビルで高さ h で居るために必要な最小コスト (=経路中の最大高さ) を $T_i[h]$ とする．それらの値を $T_0[0] = 0$(スタート時は地上に居るので), 他を ∞ で初期化した後，隣のビルとの関係から T_i と T_{i+1} を順次計算する．

$$T_{i+1}[h] = \min \begin{cases} \max(T_i[h - H_i], h) & \cdots i \text{ 番目のビルから登った場合 } (h \geq H_i) \\ T_i[h + H_i] & \cdots \text{同降りた場合} \end{cases}$$

ゴール地点を M として，ゴールでは地上に居るので，$T_M[0]$ が最小値を与える．

地面に，めりこむことはないので，非負の高さのみを考える．また登り過ぎるとゴール地点に降りられなくなるので適当な高さまで考えれば良い．

さらに，この問題では最小値だけではなく最小値を与える**経路**が要求される．どちらかの方法で求められる:

1. 最小値 $T_M[0]$ を求めた後，ゴールから順にスタートに戻る．隣のビルから登ったか降りたかは，T_i と T_{i+1} の関係から分かる．(両方同じ値ならどちらでも良い).
2. $T_{i+1}[h]$ を更新する際に，登ったか降りたかを $U_{i+1}[h]$ に記録しておく．$T_M[0]$ を求めた後に，$U_M[0]$ から順に $U_{M-1}[H_0]$ までたどる

問題 **行けるかな?⋆** (UTPC 2008)

サイコロが巨大なすごろくでピッタリとまれる場所を求める (目の合計が 2^{31} まで).
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2107

- 繰り返し自乗法 (5.A 節) を用いる．
- n ターン後に行ける場所には，n 乗した遷移行列と初期位置を表すベクトルの積が対応する．
- ヒント: U ターン禁止に対応するには (以下白い文字で記載)

問題 **Army Training⋆** (Algorithmic Engagements 2010)

平面上の 1000 個までの点が与えられる．それらを適当につないで単純多角形について聞かれるので，領域に含まれる点の個数を数えよ．(質問数がたくさんあるので，数え方を工夫する)
https://szkopul.edu.pl/problemset/problem/nXF1qOIv5S88utFPI2V_0gt3/site/

ある点が 2 頂点を通る直線の左右にあるか，など基本的な幾何的な計算手法が必要なので，初見の場合は少し後の授業を終えてから取り組むと良い．

5.3 今週の課題

問題を解いて，AOJ で Accepted を得たことを確認して，各問題指定の考察課題もしくは指定がない問題については感想 (学んだことや課題にかけた時間など) とともに，ITC-LMS に提出する．

コメントの記述は，`#if 0` と`#endif` で囲むことが簡便である．

C++

```
1 // minmaxsum.cc
2 #if 0
3   所要時間は..であった．方針は....
4   ...
5 #endif
```

```
6  int main() {
7  }
```

扱う問題は以下のどちらかで合格 (言語指定なし):

- 「平安京ウォーキング」を含む問題 2 題以上を解いて提出する．
  それぞれに，感想 (学んだことや課題にかけた時間など) を添えること
- 「行けるかな?」または「Army Training」を含む問題 1 題以上を解いて提出する．
  回答方針や感想 (学んだことや課題にかけた時間など) を添えること

提出先: ITC-LMS
提出の注意:

- 問題毎にテキストファイルを一つ作り，提出する (zip は避ける; PDF, Microsoft Word, rtf などは避けること)．グラフを作成した場合は，グラフのみ別ファイルで提出する．
- 提出物は履修者全体に公開される． この授業では，他者のソースコードを読んで勉強することを推奨するが，課題を解く上で参考にした場合は，何をどのように参加したかを明記すること．
- 締切: 授業終了時，次回授業の前日，学期終了時のそれぞれで最善のものを提出する

5.A　参考: 繰り返し自乗法 (repeated squares)

適切な手法を用いると，計算時間を短縮できることがある．繰り返し自乗法を適用できる場合には，N ステップ後の結果を $\log N$ 回の計算で求めることができる．
応用先:

- すごろくでサイコロで $N(0 \leq N \leq 2^{31})$ が出た時に，行ける場所を全て求める
- 規則に従って繁殖する生物コロニーの N ターン後の状態を求める
- 規則に従った塗り分けが何通りあるかを求める

例: $3^8 = 6561$ の計算

- $3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$　➔　7 回の乗算
- `int a = 3*3, b = a*a, c = b*b;`　➔　3 回の乗算

関連:

- オーバーフローに注意: int で表せる範囲は，この環境では約 20 億くらい
- 剰余の扱い: (a*b)%M = ((a%M)*(b%M))%M

例題	**Elementary Number Theory - Power**	(AOJ)
2 つの整数 m, n について、m^n を 1 000 000 007 で割った余りを求めよ http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=NTL_1_B&		

(例題 続き)

```
lang=jp
```

5.A.1　正方行列の表現と演算

メモ:以下に 2x2 の行列のサンプルコードを掲載する．配列や vector, valarray 等のデータ構造を用いるとNxN の行列を自作することもできる．研究や仕事で必要な場合は，専用のライブラリを使う方が無難．

C++
```
struct Matrix2x2 {
  int a, b, c, d; // a,b が上の行，c,d が下の行とする
};
```

表示も作っておこう

C++
```
void show(Matrix2x2 A) {
    cout << "[␣" << endl
         << A.a << '␣' << A.b << endl
         << A.c << '␣' << A.d << endl
         << "]" << endl;
}
```

■行列同士の乗算　続いて乗算を定義する．以下の mult は行列 A, B の積を計算する．

C++
```
// returns C = A*B
Matrix2x2 mult(Matrix2x2 A, Matrix2x2 B) {
    Matrix2x2 C = {0}; // 0 で初期化
    C.a = A.a * B.a + A.b * B.c;
    C.b = A.a * B.b + A.b * B.d;
    C.c = A.c * B.a + A.d * B.c;
    C.d = A.c * B.b + A.d * B.d;
    return C;
}
// (注) 冪乗計算はすぐにオーバーフローするので注意： ここに細工をすることも多い
```

使用例:

C++
```
  Matrix2x2 A = {0,1, 2,3}, B = {0, 1, 2, 0};
  show(A);
  show(B);
  Matrix2x2 C = mult(A, B);
  show(C);
```

■冪乗の計算 (繰り返し自乗法)　次のコードは行列 A の $p(>0)$ 乗を計算し, O に書きこむ.

C++

```
1  // O = A^p
2  Matrix2x2 expt(Matrix2x2 A, int p) {
3      if (p == 1) {
4          return A;
5      } else if (p % 2) {
6          Matrix2x2 T = expt(A, p-1);
7          return mult(A, T);
8      } else {
9          Matrix2x2 T = expt(A, p/2);
10         return mult(T, T);
11     }
12 }
```

使用例:

C++

```
1   Matrix2x2 A = {0,1, 2,3};
2   Matrix2x2 C = expt(A, 3); // A の 3 乗
3   show(C);
```

5.A.2　練習問題

例題　**Fibonacci**　(Stanford Local 2006)

フィボナッチ数列の，n 項目の値を 10^4 で割った余りを計算しなさい.
$0 \leq n \leq 10^{16}$
http://poj.org/problem?id=3070

■行列表現

$$\begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ とすると，結合則から以下を得る:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

この方針であれば，n が 10^{16} まで大きくなっても現実的に計算できる．ただし，n が `int` で表せる範囲を超えるので，`expt()` の引数などでは `long long` を用いること．また，普通に計算すると要素の値も `int` で表せる範囲を越えるので，

- (a*b)%M = ((a%M)*(b%M))%M
- (a+b)%M = ((a%M)+(b%M))%M

などの性質を用いて，小さな範囲に保つ．

動作確認には，小さな n に対して，方針 3 の手法などと比較して答えが一致することを確認すると良い．10^4 の剰余は，$F_{10} = 55, F_{30} = 2040$ などとなる．

問題	**One-Dimensional Cellular Automaton**	(アジア地区予選 2012)

与えられた規則で遷移するオブジェクトの T 時間後の状態を求めよ．(意訳) 行列を T 乗する $(0 \leq T \leq 10^9)$

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=1327

NxN の行列の実装例:

C++

```
#include <valarray>
const int N=50;
struct Matrix {
    valarray<int> a;
    Matrix() : a(N*N) { a=0; }
};
Matrix multiply(const Matrix& A, const Matrix& B) {
    Matrix C;
    for (int i=0; i<N; ++i)
        for (int j=0; j<N; ++j)
            C.a[i*N+j] = (A.a[slice(i*N,N,1)]*B.a[slice(j,N,N)]).sum()%M;
    return C;
}
```