For Tutors

MeBio　情報テキスト

フカシギの数え方

—コードの解説—

Edited by kamei

第 1 章

フカシギとは

§ 1 お姉さんのコンピュータ

メビオの理系講師の方から，「フカシギ」の動画が興味深いとの話を伺いました．実際にそれを見て，「この問題は追いかけてみる価値がある」と思い，それを実現するために Linux 環境を整え，C 言語を扱えるようにした上で，実際にプログラムを組み（これは windows 上)，フカシギを計算してみた結果がこれです．フカシギについてご存じない方は，ネットをご覧ください．

『フカシギの数え方』 おねえさんといっしょ！ みんなで数えてみよう！
http://www.youtube.com/watch?v=Q4gTV4r0zRs
「フカシギの数え方」 同じところを 2 度通らない道順の数
http://www.youtube.com/watch?v=ge8vy4tc_kQ&feature=player_embedded&list=UUdBvq7IgL4U6u3CzeZaeoFg
「フカシギの数え方」の問題を解いてみた
http://handasse.blogspot.com/2012/09/blog-post.html

膨大な場合の数の数え上げ問題は以前から私の興味のあるところでした．スリザーリンクの解法プラグラムを Java で作った経緯もあるので，この問題には非常にひかれたわけです．お姉さんの数え方に改良の余地があることは，誰が見ても明らかであり，それを実現してみようと思ったのがきっかけです．

調べている過程で，この種の問題の解法には BDD (binary dicision diagram) や ZDD (zero–suppressed binary dicision diagram) が非常に有効であることや，実際に Knuth 先生がこの経路問題の BDD, ZDD を作成されておられることを知りました．また，それを使ってこの問題を パソコンレベルで解いておられる方もおり，実際にその方の解法プラグラムをダウンロード，コンパイル，実行してみました．（そのために Linux 環境が必要だった.）

結果は非常に満足のいくものであり，ZDD はマスターしなければならないと思いましたが，この問題に関しては私の思いついたアルゴリズムの方がうまくいくのではないだろうかという考えもあり，実際に組んでみたわけです. .

§ 2 フカシギ数の値

2 × 2 : 12 通り
3 × 3 : 184 通り
4 × 4 : 8,512 通り
5 × 5 : 1,262,816 通り
6 × 6 : 575,780,564 通り
7 × 7 : 789,360,053,252 通り
8 × 8 : 3,266,598,486,981,642 通り
9 × 9 : 41,044,208,702,632,496,804 通り
10 × 10 : 1,568,758,030,464,750,013,214,100 通り
11 × 11 : 182,413,291,514,248,049,241,470,885,236 通り

12×12 : 64,528,039,343,270,018,963,357,185,158,482,118 通り

13×13 : 69,450,664,761,521,361,664,274,701,548,907,358,996,488 通り

14×14 : 227,449,714,676,812,739,631,826,459,327,989,863,387,613,323,440 通り

15×15 : 2,266,745,568,862,672,746,374,567,396,713,098,934,866,324,885,408,319,028 通り

16×16 : 68,745,445,609,149,931,587,631,563,132,489,232,824,587,945,968,099,457,285,419,306 通り

§ 3　実行結果の比較

	2.67GHz Intel Xeon E7–8837 CPU with 1TB memory (北大　湊研)		2.93GHz Intel Core i3 CPU with 10 GB memory (亀井　自宅)
size	time(sec)	Men (MB)	time(sec)
9×9	0.1	1	1
10×10	0.3	3	1
11×11	1.0	6	1
12×12	3.2	13	3
13×13	11.7	37	8
14×14	52.4	111	32
15×15	206.0	351	114
16×16	701.9	958	不可能
17×17	2326.0	3018	不可能
18×18	7607.1	7514	不可能
29×19	28279.2	27394	不可能
20×20	91944.1	74504	不可能
21×21	284117.0	216871	不可能

亀井の結果が 15×15 止まりなのはメモリーの制約のためです．（10 GB あれば 16×16 もできるはずですが，32 bit コンパイラの配列限界などがあります．）

§ 4　2012/11/13 の変更点

フカシギのコードを少し変更して，Jordin 閉曲線を数えるプログラムを作りました．ただ，こちらはネット上に答が載っていませんので，確認の工夫がいります，少し考えて，「$m \times n$ の総数と $n \times m$ の総数が一致していればよいだろう」ということに気付きました．そこでプログラムも正方形に固執せず，長方形に対応できるようにしたのですが，その変更をフカシギにも反映しました．

第 2 章

基本的なアイデア

§ 1　経路の閉曲線化と切断の定義

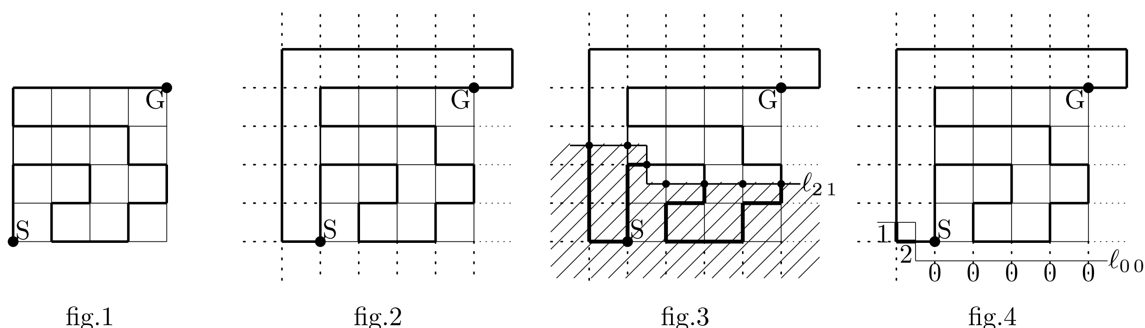


fig.1　fig.2　fig.3　fig.4

fig.1 のように，4×4 の格子の左下 S(0, 0) から右上 G(4, 4) を結ぶ経路があったとする．この経路に (4, 4), (5, 4), (5, 5), (−1, 5), (−1, 0), (0, 0) を結ぶ経路をつなげて Jordin 閉曲線にする（fig.2）．この Jordin 閉曲線を C とする．

この閉曲線を fig.3 のように, 折れ線 $\ell_{m,n}$ で切断する．$\ell_{m,n}$ は $(-1.5,\ m{+}0.5)$, $(n{-}0.5,\ m{+}0.5)$, $(n{-}0.5,\ m{-}0.5)$, $(4.5,\ m-0.5)$ を結んで出来る折れ線である．$\ell_{m,n}$ の下側の領域を L, 上側の部分を U とする．$C \cap L$ は何本かの折れ線分の集合になっている．見やすくするために $C \cap L$ を太くしておこう．

$l_{m,n}$ 上の点のうち，x 座標もしくは y 座標が整数である点（ただし $-1.5 < x < 4.5$, $-1 < y < 5$）は 7 個存在するが，これらを左（上）から順に P_0, P_1, $\cdots$, P_6 とする．これらに 0, 1, 2 のいずれかの数値を，次の様に対応させる．

$$\begin{cases} \mathrm{P_k} \text{ が } C \text{ 上にない場合} & \cdots \quad 0 \\ \mathrm{P_k} \in C \cap L \text{ で，線分の左側の端点になっている場合} & \cdots \quad 1\text{（気分的には開き括弧）} \\ \mathrm{P_k} \in C \cap L \text{ で，線分の右側の端点になっている場合} & \cdots \quad 2\text{（気分的には閉じ括弧）} \end{cases}$$

fig.3 の場合，対応する数値は 1020102 である．これを C の $\ell_{2,1}$ による切断 (section) と呼ぼう．fig.4 の場合，$\ell_{0,0}$ による切断 (section) は 1200000 である．（経路 C がどのようなものであれ，C の $\ell_{0,0}$ での切断は 1200000 になる.）

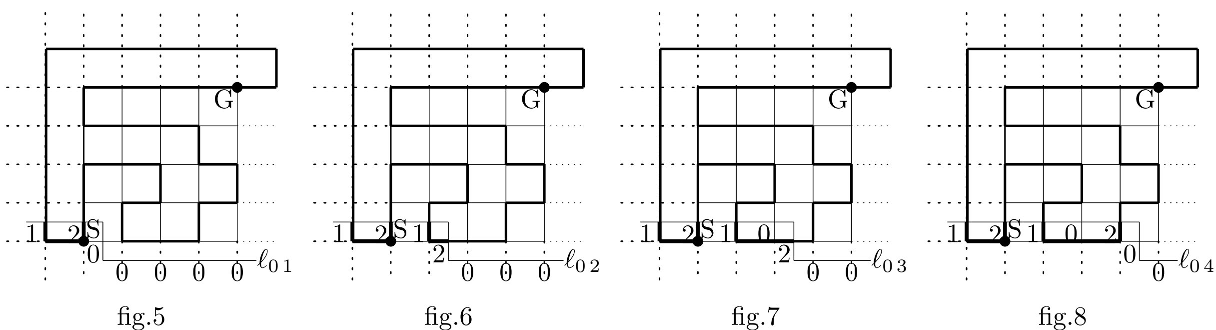


fig.5　fig.6　fig.7　fig.8

切り口の折れ線を図のように $\ell_{0,1}$, $\ell_{0,2}$, $\ell_{0,3}$, $\ell_{0,4}$ と変えていくと，対応する切断は 120000, 1212000, 1210200, 1210200 と変わっていく．

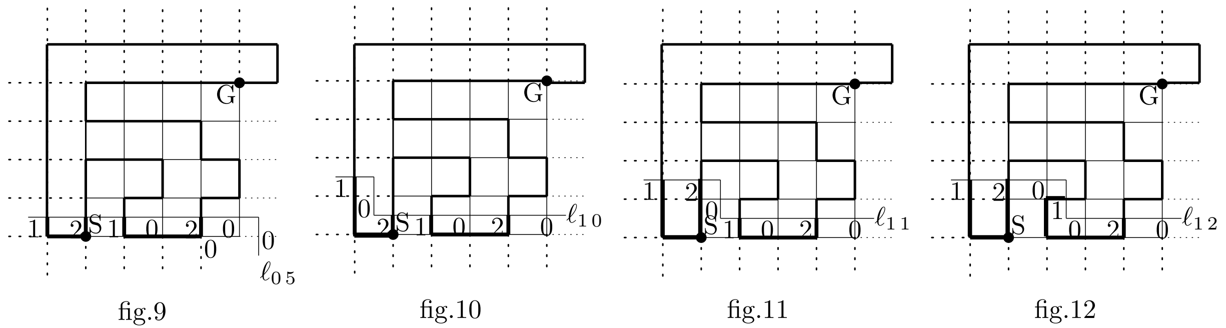
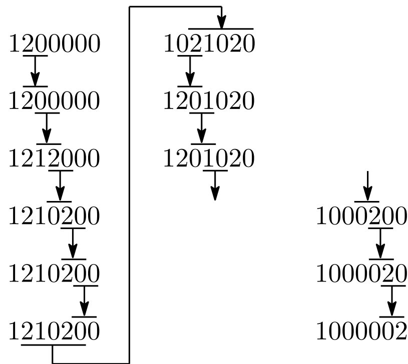


fig.9　fig.10　fig.11　fig.12



さらに続けると，$\ell_{0,5}$ による切断は 121020 である．この場合，最後の数は 0 でなければならない．次の $\ell_{1,0}$ は $\ell_{0,0}$ を上に 1 平行移動させた折れ線である．$\ell_{1,0}$ による切断 102102 は，$\ell_{0,5}$ の切断の最後の 0 を取り除き，最初の 1 の後に 0 を挿入して残りを右にずらしたものになっている．以後 $\ell_{1,1}$ による切断 1201020 (fig 3), $\ell_{1,2}$ による切断 1201020 (fig 4), $\cdots$ と続く．この数値の変化に注意して欲しい．

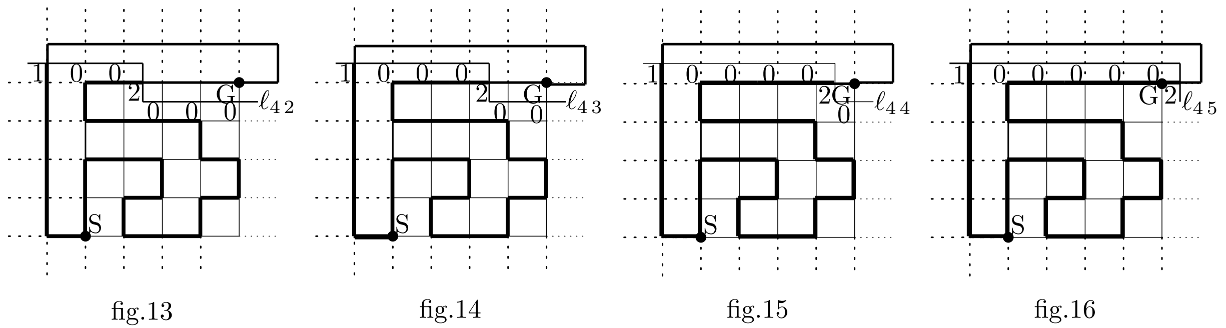


fig.13　fig.14　fig.15　fig.16

最後の $\ell_{4,5}$ による切断は 1000002 でなければならない．

§ 2　切断の延長

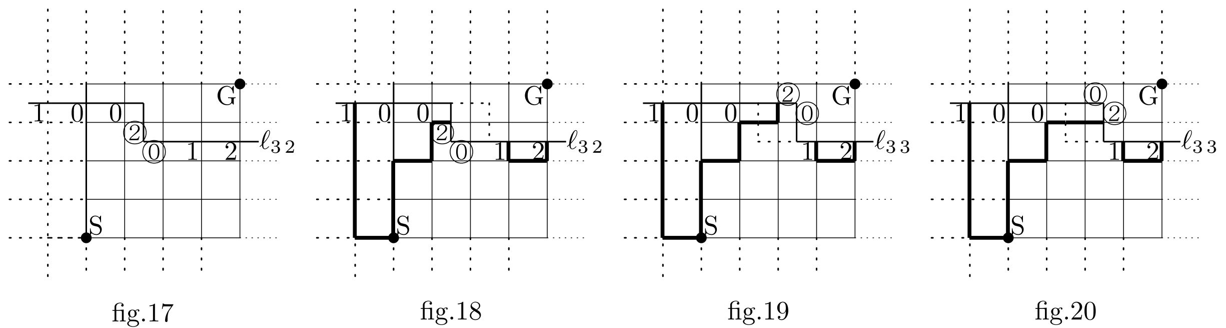


fig.17　fig.18　fig.19　fig.20

$\ell_{3,2}$ による切断が 100 ②⓪ 12 だったとしよう (fig.17)．これは例えば fig.18 のような経路（の部分集合）に対応している．この部分集合を含む経路の $\ell_{3,3}$ による切断はどうなるであろうか．(わかりやすさのため，変更の可能性のある箇所を〇で囲っている．)

(1.5, 3) が②になっているということは (1, 3) – (1.5, 3) が経路になっているということである．これは (1.5, 3) – (2, 3) まで延長されなければならない．一方 (2, 2.5) が⓪になっているということは (2, 2) – (2, 2.5) が経路になっていないということである．もちろん (2, 2.5) – (2, 3) も経路にはならない．

従って (2, 3) に左側からやってきた経路は上へ折れ曲がるか，右へ直進するかしないといけないことになる．つ

まり $\ell_{3,3}$ による切断は， 100 ②⓪ 12(fig.19) と 100 ⓪② 12(fig.20) の 2 つである．これらを $\ell_{3,2}$ の切断 1002012 の（1 ステップの）延長と呼ぼう．

折れ線 $\ell_{m,n}$ $(0 \leqq n \leqq 4)$ による切断 s を 折れ線 $\ell_{m,n+1}$ に延長した場合（もしくは折れ線 $\ell_{m,5}$ による切断 s を折れ線 $\ell_{m+1,0}$ に延長した場合），起こりうる切断の集合を $B(n,\ s)$（または $B(5,\ s)$）とする．（この集合が m にはよらないということに注意.）今の場合は $B(2,\ 1002012) = \{1002012,\ 1000212\}$ ということになる．

$n < 5$ として $\ell_{m,n}$ による切断を 1 ステップ延長する場合，変化する可能性がある点は 2 つある．格子点に集まる経路の数は 0 か 2 だから，延長の結果は高々 2 通りである．つまり $B(n,\ s)$ は要素数が高々 2 の A の部分集合である．

例えば fig.17〜fig.20 の様に，2 点の一方が 0 で他方が x $(x = 1,\ 2)$ の場合，延長後の 2 点は $0x$ か $x0$ になる．

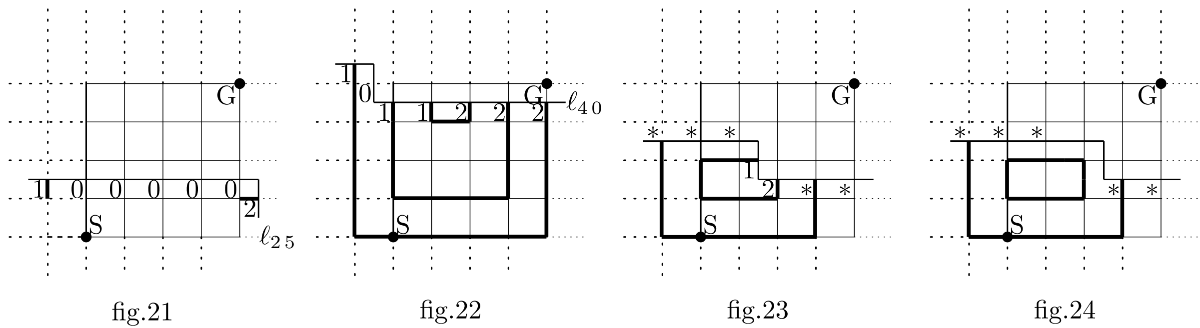


fig.21　fig.22　fig.23　fig.24

注意　実際の経路の $\ell_{2,5}$ による切断が 1000002 になったりはしない．(右端が正方形からはみ出している．fig.21)

また $\ell_{4,0}$ の切断が 1011222 になることもない．（U 部分が狭すぎて，その経路をうまく閉じることが出来ない．fig.22）しかし経路を構成する途中の状態としては，これらも切断の可能性としては除去しないことにする．実際これらは手を進めたとき，ゴールにたどり着けない袋小路だから，数え上げの総数には影響しない．

変化する 2 点が 12 である場合，延長は存在しない．これは連結成分がひとつでなくなるからである．集合の言葉で言うと例えば $B(2,\ ***21**) = \phi$ ということになる．(fig.23, fig24)

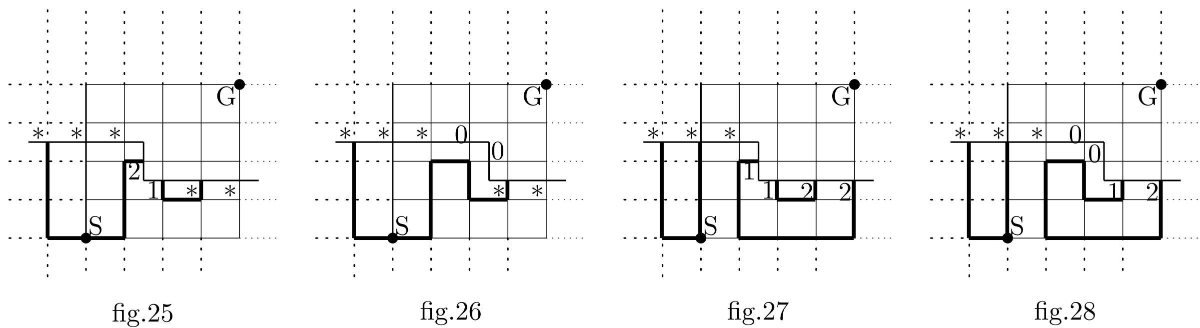


fig.25　fig.26　fig.27　fig.28

変化する 2 点が 21 である場合，延長後は 00 になる (fig.25, fig.26)．この場合，2 本の折れ線がつながって 1 本になる．例えば $B(2,\ ***21**) = \{***00**\}$ である．

変化する 2 点が 11 である場合も延長後は 00 になるが，この場合右側の 1 とつながっている 2 が 1 に変わる．(fig.27, fig.28)．そのため各切断の端点が与えられた場合，対になる端点がどこであるかを知っておく必要がある．

変化する 2 点が 22 である場合も 11 とほぼ同様である．

変化する 2 点が 00 である場合，延長後は 00 か 12 である．12 になるのは線分を 1 本増やす場合に対応する．

注意　切断が与えられたとき，その切断の端点に対して対になる端点は，実際の経路の描き方によらず一意的に定まる．

§ 3　アルゴリズム

以上の考察により，フカシギ数を求める手順はほぼおわかりいただけたと思うが，次の様なものである．

あり得るすべての切断の集合を A としよう．4×4 の正方形の場合, $A = \{1000002,\ 1000020,\ 1000122,\ 1000200,\ 1000212, \cdots,$ $\#(A) = 76$ である．この総数はカタラン数と二項係数の積和で計算できる．（後述する．）

A の要素を辞書式に数え上げたい．まず最初の数は 1 である．次は 0, 1, 2 のいずれでもよいが，1022... のように左端からの 1 の総数を 2 の総数が超えてはいけない．そこで，その時点までに決定しているデータ `char *s`，その時点の桁位置 `unsigned int k`，1 の総数 − 2 の総数 を表す変数 `int br_bal` (bracket balance) を引数に持たせて次の桁を決定させる再帰的関数 `make_all_section(char *s,unsigned int k, int br_bal)` を作ってやるとよい．最後の桁まで到達したときに `br_bal =0` になっていればデータが見つかったことになり，登録カウンターを一つ進めて構造体の切断状態 `section[].end_state[]` に登録する．ついでに端点の対応も調べて `section[].mate_end[]` に登録しておく．そのまま return して，再帰手続きを進める．

すべての切断を探し終わったら，切断から登録番号を検索する関数 `unsigned long search_index(char *s)` をつくる．切断は辞書式に登録されているから簡単で，絞り込みの中点との大小比較で，上または下を中点と置き換え続ければよい．

次に，すべての切断 s に対し $n = 0.\ 1,\ \cdots,\ 5$ の場合の $B(n,\ s)$ を調べ，その登録番号を構造体の `section[].nextsec[n][j]` $(j = 0,\ 1)$ に登録する．なければ 0 を登録する．

折れ線 $\ell_{m,n}$ と切断 s に対し，その切断を折れ線の左下部分 L で矛盾無く実現できる線分群の総数を $f(s,\ \ell_{m,n})$ と書こう．（U 部分でうまく閉じるかどうかは考えない．）　4×4 の正方形の場合，$f(1200000,\ \ell_{0,0}) = 1$ で，その他の s に対しては $f(1200000, \ell_{0,0}) = 0$ である．折れ線を $\ell_{0,0}$ から初めてまず右に一つずつ，右端に達したら一段持ち上げてまた左から右へ一つずつ動かしていくときに，$f(s,\ \ell_{m,n})$ がどのように変化していくかを調べる．最後の $\ell_{4,5}$ まで動かし切ったとき，$f(1000002,\ \ell_{4,5})$ が求めるフカシギ数ということになる．

変化はもちろん切断の延長 $B(n, s)$ を参照する漸化式で与えられる．この漸化式は $1 \leqq n \leqq 4$ の場合，$f(s, \ell_{m,n}) = \sum f(t, \ell_{m,n-1})$ で与えられる．ただし和は $s \in B(n,\ t)$ となる t を渡る．また $s = B(5,\ t)$ のとき $f(s, \ell_{m,0}) = f(t, \ell_{m-1,4})$ である．（これらは参照型の漸化式ではなく，押し出し型の漸化式になっている．）

この数値の記録，計算のためには多倍長の数値を格納する領域が $\#(A)$ の 2 倍必要となる．本プログラムでは構造体内部に `unshined long long section[].path_num[k][i]` $(k = 0,\ 1, i = 0,\ 1,\ 2,\ 3,\ 4)$ を用意した．`unshined long long` 型は 64 bit のデータなので $2^{64} \fallingdotseq 1.8 \times 10^{19}$ までの数を扱えるが，これを 10^5 までの桁あふれに耐えられる 10^{14} 用のメモリーとして用い，$i = 0$ が桁あふれを起こしたら $i = 1$ に繰り上げる（以下同様）ことにして 10^{70} までの数値を記憶する．桁あふれ処理は折れ線の段が増加するときのみ行う．その際に計算の必要な桁のデータ `fcp` (flow check position) を更新する．`fcp` より大きい上位の桁の計算は行わない．

また，計算前のデータと計算後のデータの場所を固定すると，全 section の計算が終了するたびに全データのコピーをする必要が生じる．section 数が膨大なのでこの時間は実行時間に大きく影響する．そこでコピーの時間を節約するために，計算前のデータと計算後のデータの場所を flip flop 式に交代するようにした．これは `parity` というフラグを立て，全数計算が終わる毎にその 0, 1 の値を入れ替えていく（計算方向は `parity` で判断する．）ものである．

第 3 章

ソースコード

```
// fukashigi_kamei.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//

#include "stdafx.h"     // これは Microsoft Visual C++ のとき
//#include <stdio.h>    // これは Linux のとき
#include <string.h>         //  文字列を比較するための strcmp を使うため必要
#include <time.h>           //  タイマーを使用するためのライブラリ

#define MAXSECTION 4200000  //  NOH = 16 でおよそ 418 万，NOH = 17 で 1167 万必要だが
                            //  420 万がマシンの限界..
//#define MAXSECTION 11662300   //これは無理

unsigned int noh=12;// number of lines: 対称の矩形の横の線の数 = 一辺の長さ + 1. main() で
                    // 入力される
unsigned int nov=12;// number of lines: 対称の矩形の縦の線の数 = 一辺の長さ + 1. main() で
                    // 入力される
#define NOH 16      // noh の最大値. 構造体はこの数で生成される. (noh<NOH の時は使わないだけ)
#define FCP 5       // fcp の最大値+1. 計算可能なのは FCP x 14 桁.


typedef struct section_{
    char end_state[NOH+3];      // 三つ多いのは最後に NUL 文字をいれて文字列扱いするため
    char mate_end[NOH+2];
    unsigned long nextsec[NOH+2][2];    //  推移できる次の切断の番号 (最大 2 つ) を登録する
    unsigned long long path_num[2][5];  //  path の計算場所. 14 桁 x 5 =70 桁
} structsection;

unsigned long seccnt = 0;   // section counter: 全 section を作成，検索，操作するときのカウンター
unsigned long last_section; // section の総数が入る.
unsigned int fcp=0;         // flow check position: path_num[] のうち現在使っている最大場所.
                            // 最初は 0 だが, 14 桁を越える毎に 1 ずつ増えていく.
unsigned int fcp_of=0;      // fcp overflow: fcp が FCP になったときに設定される.
unsigned int fcp_of2=0;     // fcp overflow2: 警告を一度だけにするための変数.
unsigned parity=0;          // 計算前データ領域と計算後データ領域を交互に使うためのフラグ
structsection section[MAXSECTION];  // section[0] は初期登録のとき以外使わない
```

```
//////////////////////////////////////////////////////////////////////////////////////////
unsigned int mate(char *s,unsigned int k){  // s[k]=1（開き括弧）のとき，mate[k] は対応する閉じ
                                            // 括弧の番号．閉じ括弧も同様.
                                            // 括弧でないときは m[k] = k（自分自身）
    unsigned int i;
    int br_bal=0;   // brackets balance: 開き括弧を +1, 閉じ括弧を -1 として足していったもの
    if(s[k]=='0') return k;         // 括弧ではないので同じ値を返す
    if(s[k]=='1'){                  // 開き括弧の場合，対応する閉じ括弧は右にあるので，
        for(i=k;i<=noh+2;i++){      // bracket balance が初めて 0 になる所をみつける.
            if(s[i]=='1')br_bal++;
            if(s[i]=='2')br_bal--;
            if(br_bal==0)return i;
        }
    }
    if(s[k]=='2'){                  // 閉じ括弧の場合対応する開き括弧は左にある.
        for(i=k;i>=0;i--){
            if(s[i]=='1')br_bal++;
            if(s[i]=='2')br_bal--;
            if(br_bal==0)return i;
        }
    }
    return 255;
}

//////////////////////////////////////////////////////////////////////////////////////////
void make_all_section(char *s,unsigned int k,int br_bal){

//  0<k<=noh+2 で呼び出される．k番目をつくる．作れたら再帰的に k+1 番目に回す.
//  作れなければ戻る.
//  s は s[0]s[1]...s[noh+1] のポインター，k はこれから探す場所
//  s[0],...,s[k-1] までは確定している．s[k]='0','1','2' を調べる.
//  s[j]='1' は開き括弧「(」, この場合 a_j=+1,
//  s[j]='2' は閉じ括弧「)」, この場合  a_j=-1,
//  s[j]='0' は括弧なし，この場合 a_j=0 を表す.
//  br_bal =Σ{j=0;j<=k-1;j++}('2'+a_j) としてある．これが負になったらだめ.
//  また最後には 0 でないとだめ.

    int next_br_bal;
    unsigned int m;
    if(k==noh+2){
        if(br_bal==0){  // 一つの状態が見つかった！
            seccnt++;   // 登録カウンターを一つ増やして登録する.
            for(m=0;m<=noh+1;m++){
                section[seccnt].end_state[m]=s[m];
                section[seccnt].mate_end[m]=mate(s,m);
            }
```

```
            section[seccnt].end_state[noh+2]=0; //最後に NUL 文字をいれておく.
        }
        return;
    }                                       // k==noh+2 の場合の登録終了. 以下は k<=noh+2 の場合
    s[k]='0';                               // k 番目を '0' としてみる. br_bal はそのまま.
    next_br_bal=br_bal;                     // br_bal はそのまま.
    make_all_section(s,k+1,next_br_bal);    // k+1 を呼び出す.

    s[k]='1';                               // k 番目を '1' としてみる
    next_br_bal=br_bal+1;                   // br_bal はそのまま.
    make_all_section(s,k+1,next_br_bal);    // k+1 を呼び出す.

    s[k]='2';                               // k 番目を '2' としてみる
    next_br_bal=br_bal-1;                   // br_bal は 1 減らす.
    if (next_br_bal>=0){                    // もしも br_bal<0 になれば不適
        make_all_section(s,k+1,next_br_bal);    // k+1 を呼び出す.
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
unsigned long search_index(char *s){        // 括弧の配置, 例えば (((.).())(..)) は
    unsigned long low=1;                    // 文字列             11102012210022 のように表現さ
れている
    unsigned long high=last_section+1;      // この文字列が何番目の section であるかを検索する.
    unsigned long middle;                   // 整列されているので探すのはたやすい.
    int result;

    while(1){
        middle=(low+high)/2;
        result=strcmp(s,section[middle].end_state);
        if(result==0) return middle;
        if(result<0) high=middle;
        if(result>0) low=middle;
    }
}


////////////////////////////////////////////////////////////////////////////////////////////////////
void make_next_section1(unsigned long bangou,unsigned int k){
                                            //  s[k]s[k+1] の状態変化が「なしなし」の場合
    char *s=section[bangou].end_state;          //  s[k] の線のあるなしと s'[k] の線のあるなしが
    char *t=section[bangou].mate_end;           //  一致し, s[k+1] と s'[k+1] も一致する場合
    char u[NOH+3];
    unsigned int i;
    if(k<noh+1){    //  通常の推移 (右端ではない場合))
        if(s[k]!='0'&&s[k+1]!='0'){             // s[k] にも s[k+1] にも線がある場合
            section[bangou].nextsec[k][0]=0;    // s'[k] にも s'[k+1] にも線があるはずがない
```

```
            return;
        }
        if(s[k]=='0'||s[k+1]=='0'){              // s[k] と s[k+1] の少なくとも一方に線がない場合
                section[bangou].nextsec[k][0]=bangou;    // 状態変化はないので，bangou を返す
            return;
        }
    }
    if(k=noh+1){                                 // 右端での推移.
        if(s[k]!='0'){                           // 右端が 0 でなければ不能.
            section[bangou].nextsec[k][0]=0;
            return;
        }
        if(s[k]=='0'){                           //  0 ならそれを除去して s[2] 以降を右にスライド
            u[0]='1';
            u[1]='0';
            for(i=2;i<=noh+1;i++){
                u[i]=s[i-1];
            }
            u[noh+2]=0;
            section[bangou].nextsec[k][0]=search_index(u);
            return;
        }
    }
}

//////////////////////////////////////////////////////////////////////////////////////////////
void make_next_section2(unsigned long bangou,unsigned int k){
                                        // s[k]s[k+1] の状態変化が「ありあり」の場合
    char *s=section[bangou].end_state;
    char *t=section[bangou].mate_end;
    char u[NOH+3];
    unsigned int i;
    for(i=0;i<=noh+2;i++){
        u[i]=s[i];
    }
    if(k<noh+1){                            // 通常の推移（右端ではない場合)）
        if(s[k]=='0'&&s[k+1]=='0'){         // 両方 0 の場合
            u[k]='1';                       // 新たな線分が登場する
            u[k+1]='2';
            section[bangou].nextsec[k][1]=search_index(u);
            return;                         // 一方が 0 の場合
        } else if(s[k]=='0'||s[k+1]=='0'){  // 端点の位置交換
            u[k]=s[k+1];
            u[k+1]=s[k];
            section[bangou].nextsec[k][1]=search_index(u);
            return;
        }
```

```
        if(s[k]=='1'&&s[k+1]=='2'){          // 左が '1', 右が "2" の場合
            section[bangou].nextsec[k][1]=0;    // loop ができてしまうのでだめ
            return;
        }
        if(s[k]=='2'&&s[k+1]=='1'){          // 左が '2', 右が "1" の場合
            u[k]='0';                        // 両方を '0' に変えるだけ
            u[k+1]='0';
            section[bangou].nextsec[k][1]=search_index(u);
            return;
        }
        if(s[k]=='1'&&s[k+1]=='1'){          // 両方 '1' の場合
            u[k]='0';                        // 両方を '0' に書き換えて
            u[k+1]='0';
            u[t[k+1]]='1';                   // s[k+1] に対応する閉じ括弧を開き括弧に変更する
            section[bangou].nextsec[k][1]=search_index(u);
            return;
        }
        if(s[k]=='2'&&s[k+1]=='2'){          // 両方 '2' の場合
            u[k]='0';                        // 両方を '0' に書き換えて
            u[k+1]='0';
            u[t[k]]='2';                     // s[k] に対応する開き括弧を閉じ括弧に変更する
            section[bangou].nextsec[k][1]=search_index(u);
            return;
        }
    }
    if(k=noh+1){    //  右端での推移は「なしなし」で済ませている.
        section[bangou].nextsec[k][1]=0;
        return;
    }
}


//////////////////////////////////////////////////////////////////////////////////////////
unsigned long get_start_index(){        // 初期状態 "1200...0" の番号を探す.
    unsigned int i;
    char u[NOH+3];
    u[0]='1';
    u[1]='2';
    for(i=2;i<=noh+1;i++){
        u[i]='0';
    }
    u[noh+2]=0;
    return search_index(u);
}

//////////////////////////////////////////////////////////////////////////////////////////
void path_num0_clear_all(){     // 構造体 section の計算結果保存用変数 path_num[0][i] (0<=i<=4) を
```

```
    int i;                     // クリアする
    for(seccnt=0;seccnt<=last_section;seccnt++){
        for(i=0;i<=FCP-1;i++){
            section[seccnt].path_num[0][i]=0;
        }
    }
}

void path_num1_clear_all(){     // 構造体 section の計算結果保存用変数 path_num[1][i] (0<=i<=4) を

    int i;                     // クリアする
    for(seccnt=0;seccnt<=last_section;seccnt++){
        for(i=0;i<=FCP-1;i++){
            section[seccnt].path_num[1][i]=0;
        }
    }
}

void path_num_clear(){
    unsigned int i;
    if(parity==0){  // 構造体 section の計算結果保存用変数 path_num[0][i] (0<=i<=fcp) を
                    // クリアする
        for(seccnt=0;seccnt<=last_section;seccnt++){
            for(i=0;i<=fcp;i++){
                section[seccnt].path_num[0][i]=0;
            }
        }
    }else if(parity==1){    // 構造体 section の計算結果保存用変数 path_num[1][i] (0<=i<=fcp) を
                            // クリアする
        for(seccnt=0;seccnt<=last_section;seccnt++){
            for(i=0;i<=fcp;i++){
                section[seccnt].path_num[1][i]=0;
            }
        }
    }
}

void path_num_copy(){
    unsigned int i;
    if(parity==0){          // 構造体 section の変数 path_num[1][i] (0<=i<=fcp) を
        for(seccnt=0;seccnt<=last_section;seccnt++){    // path_num[0][i] にコピー
            for(i=0;i<=fcp;i++){
                section[seccnt].path_num[0][i]=section[seccnt].path_num[1][i];
            }
        }
    }else if(parity==1){    // 構造体 section の変数 path_num[0][i] (0<=i<=fcp) を
```

```
        for(seccnt=0;seccnt<=last_section;seccnt++){    // path_num[1][i] にコピー
            for(i=0;i<=fcp;i++){
                section[seccnt].path_num[1][i]=section[seccnt].path_num[0][i];
            }
        }
    }
}

///////////////////////////////////////////////////////////////////////////////////////////////
void path_num_sum_onestep(int i){   // section を一つ移動させたときの path 可能数を計算する.
        unsigned long nextsec0;
        unsigned long nextsec1;
        unsigned int j;
    if(parity==0){                  // path_num[0] から path_num[1] をつくる
        for(seccnt=0;seccnt<=last_section;seccnt++){
            nextsec0=section[seccnt].nextsec[i][0];
            nextsec1=section[seccnt].nextsec[i][1];
            for(j=0;j<=fcp;j++){                        // とりあえず桁あふれを気にせず足す.
                if(nextsec0!=0)section[nextsec0].path_num[1][j]+=section[seccnt].path_num[0][j];
                if(nextsec1!=0)section[nextsec1].path_num[1][j]+=section[seccnt].path_num[0][j];
            }
        }
    }else if(parity==1){            // path_num[1] から path_num[0] をつくる
        for(seccnt=0;seccnt<=last_section;seccnt++){
            nextsec0=section[seccnt].nextsec[i][0];
            nextsec1=section[seccnt].nextsec[i][1];
            for(j=0;j<=fcp;j++){                        // とりあえず桁あふれを気にせず足す.
                if(nextsec0!=0)section[nextsec0].path_num[0][j]+=section[seccnt].path_num[1][j];
                if(nextsec1!=0)section[nextsec1].path_num[0][j]+=section[seccnt].path_num[1][j];
            }
        }
    }
}

///////////////////////////////////////////////////////////////////////////////////////////////
void path_num_overflow(){   // 桁あふれの処理をする
    unsigned int j;
    if(parity==0){
        for(seccnt=0;seccnt<=last_section;seccnt++){
            if (section[seccnt].path_num[0][fcp] > 90000000000000)fcp++;
            if (fcp==FCP){fcp=4;fcp_of++;}
            for(j=0;j<fcp;j++){                     // fcp 未満の桁あふれの処理をする.
                section[seccnt].path_num[0][j+1]+=(section[seccnt].path_num[0][j] / 100000000000000);
                section[seccnt].path_num[0][j]=(section[seccnt].path_num[0][j] % 100000000000000);
            }
        }
    }else if(parity==1){
```

```
        for(seccnt=0;seccnt<=last_section;seccnt++){
            if (section[seccnt].path_num[1][fcp] > 90000000000000)fcp++;
            if (fcp==FCP){fcp=4;fcp_of++;}
            for(j=0;j<fcp;j++){                     // fcp 未満の桁あふれの処理をする.
                section[seccnt].path_num[1][j+1]+=(section[seccnt].path_num[1][j] / 100000000000000);
                section[seccnt].path_num[1][j]=(section[seccnt].path_num[1][j] % 100000000000000);
            }
        }
    }
}


////////////////////////////////////////////////////////////////////////////////////////////////////
int _tmain(int argc, _TCHAR* argv[]){   //  これは Microsoft Visual C++ のとき
//int main(){                           // これは Linux のとき

////////////////////////////////////////////////////////////////////////////////////////////////////

    unsigned int m;
    unsigned int i;
    unsigned int j;
    int k;

    unsigned long start_index;  // 最初は 120...000
    unsigned long goal_index=1; // 最後は 100...002 になる

    int flag=0;                 // 総数を表示するときの最初のメモリを見つけるためのカウンター
    unsigned long long result;  // 得られた総数の 1 メモリー分のデータを格納する

    printf("長方形の横の長さ (1〜15) を入力してください. ");
    scanf("%d",&noh)    ;
    printf("\n");
    noh++;

    if(noh>16||noh<2){
    printf("不正な入力です. \n");
    return 0;
    }

    printf("長方形の縦の長さ (1〜99) を入力してください. ");
    scanf("%d",&nov)    ;
    printf("\n");
    nov++;

    time_t timer;

    char time_st[32];
```

```
    time(&timer);       // 現在時刻の取得

//////  以下は Visual C++ の場合 ///////////////////////////////////////////
    ctime_s(time_st, 26, &timer);
    printf("現在時刻: %s", time_st);       // 現在時刻を文字列に変換して表示 (by Ishikawa)
//////  以下は Linux の場合 ///////////////////////////////////////////
//  printf("現在時刻: %s\n", ctime(&timer));    // 現在時刻を文字列に変換して表示 */
//////  以上 ///////////////////////////////////////////

    printf("%1u x %1u の格子のフカシギ数を数えます. \n",noh-1,nov-1);
    printf("まず section (経路をある閉曲線で切断してできる線分群の同値類) をすべて調べ上げます.
\n\n");

//  次の make_all_section ですべての状態を作り上げて, section[] に登録していく.
//  最初は section[1],section[0] は以後使わない.

    section[0].end_state[0]='1';
    make_all_section(section[0].end_state,1,1);
    last_section=seccnt;

    start_index=get_start_index();

//  すべて作って登録した. 総数は last_section だった. 最後は section[last_section]

    printf("総数は %1lu で, 初期状態は %1lu 番目です. \n",last_section,start_index);
    time(&timer);                               // 現在時刻の取得
//////  以下は Visual C++ の場合 ///////////////////////////////////////////
    ctime_s(time_st, 26, &timer);
    printf("現在時刻: %s", time_st);       // 現在時刻を文字列に変換して表示 (by Ishikawa)
//////  以下は Linux の場合 ///////////////////////////////////////////
//  printf("現在時刻: %s\n", ctime(&timer));    // 現在時刻を文字列に変換して表示 */
//////  以上 ///////////////////////////////////////////

    printf("次に切断面を移動させたときの同値類の対応を調べます. \n\n");

    for(seccnt=1;seccnt<=last_section;seccnt++){
        for(m=1;m<=noh+1;m++){
            make_next_section1(seccnt,m);
            make_next_section2(seccnt,m);
        }
    }

    printf("対応の調査が完了しました. \n");

    time(&timer);                               // 現在時刻の取得
//////  以下は Visual C++ の場合 ///////////////////////////////////////////
    ctime_s(time_st, 26, &timer);
```

```
    printf("現在時刻: %s", time_st);        // 現在時刻を文字列に変換して表示 (by Ishikawa)
//////  以下は Linux の場合  ////////////////////////////////////////////
//  printf("現在時刻: %s\n", ctime(&timer));    // 現在時刻を文字列に変換して表示 */
//////  以上  ////////////////////////////////////////////

    printf("切断面をゴールに近づけながら，各 section の数の変化を調べていきます．\n\n");

    path_num0_clear_all();
    path_num1_clear_all();
    section[start_index].path_num[0][0]=1;

    for(i=1;i<=nov;i++){

        if(fcp_of>=1 && fcp_of2==0){
            printf("最高桁を超えました．以下の値は正しい値の下 76 桁です．\n");
            fcp_of++;
            fcp_of2=1;
        }

        printf("%2u x %2u",noh-1,i-1);
        for(j=1;j<=noh+1;j++){
            path_num_sum_onestep(j);
            path_num_clear();
            parity=1-parity;
        }
        path_num_overflow();
        printf(" ");
        flag=0;
        for(k=4;k>=0;k--){
            result=section[goal_index].path_num[parity][k];
            if(flag==1){
                printf("%014llu",result);
            } else  if(flag==0&&result!=0){
                printf("%1llu",result);
                flag=1;
            }
        }
        printf("\n");
    }

    printf("\n 終了しました.\n\n");
    printf("総数は ");
    flag=0;
    for(k=4;k>=0;k--){
        result=section[goal_index].path_num[parity][k];
        if(flag==1){
            printf("%014llu",result);
```

```
        } else  if(flag==0&&result!=0){
            printf("%1llu",result);
            flag=1;
        }
    }
    printf(" です. \n\n");

    time(&timer);                              // 現在時刻の取得
//////  以下は Visual C++ の場合  //////////////////////////////////////////////
    ctime_s(time_st, 26, &timer);
    printf("現在時刻: %s", time_st);         // 現在時刻を文字列に変換して表示 (by Ishikawa)
//////  以下は Linux の場合  ////////////////////////////////////////////
//  printf("現在時刻: %s\n", ctime(&timer));    // 現在時刻を文字列に変換して表示 */
//////  以上  //////////////////////////////////////////


    printf("終了します. 何か入力してください. ");
    scanf("%d",&noh)    ;
    printf("\n");

    return 0;
}
```

第 4 章

ソースの解説

このあたりはソースを解読してください.

§ 1 括弧の付け方とカタラン数

§ 2 対応する括弧の見つけ方

§ 3 切断の延長の見つけ方

§ 4 切断から登録番号を調べる方法

§ 5 70 桁の自然数を扱う方法

§ 6 計算を速くする工夫