郵便切手の問題の構成的計算法

2023-12-31

片山　博文[1]

katayama.hirofumi.mz@gmail.com

Constructive Calculation Method for Stamp Folding Problem

Katayama Hirofumi

はじめに

数学において**郵便切手の問題**、または**切手折り畳み問題**（stamp folding problem）とは、n枚の横につながった切手について、一番左の切手が表向きになるようにすべての切り取り線を折り曲げる方法は何通りあるかを問う未解決問題である。切手がn枚のとき、関数fを用いてその方法が$f(n)$通りあるとすると、$f(1)=1$、$f(2)=1$、$f(3)=2$、$f(4)=4$、$f(5)=10$、$f(6)=24$、$f(7)=66$ となることが知られている。この$f(n)$の一般式、もしくは機械的な計算方法がないかを探るのが本書の目的である。

図解

図に表して考えてみよう。$n\leq 1$のときは自明。$n=2$のときは図 1A。$n=3$のときは図 1B のようになる。

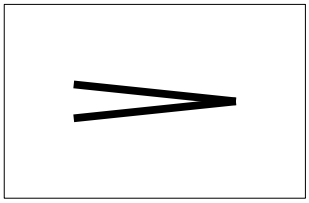

図 1A

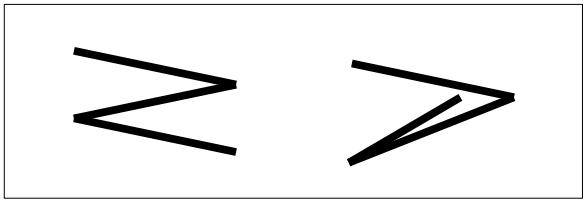

図 1B

$n=4$のときは図 1-C のようになる。

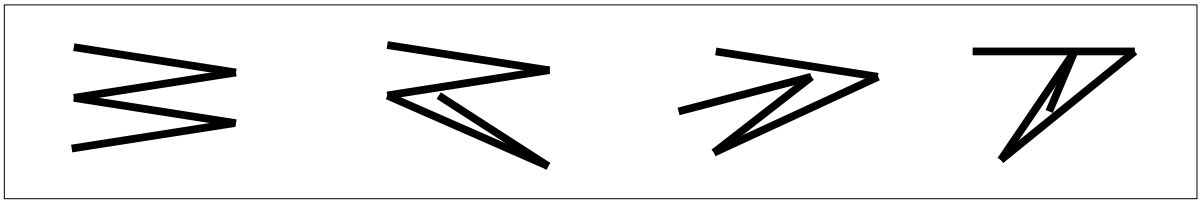

図 1C

1　Katayama Hirofumi is living in Miyazaki-city, Japan.

つながった線分を使って描くのは、制限があって描きにくく、分かりにくいものである。よって、もっと描きやすくて分かり易い表示方法を考えることにする。

折りたたんだ切手の中央を下から針で串刺しのように突き刺す。この針を**軸**を呼ぶことにする。位置関係がよくわかるように切手の突き刺さった場所に番号を付ける。すると、次のような図 2A、図 2B、図 2C のように描ける。これらはグラフ理論の有向グラフを意識したものだ。切手が有向グラフの頂点に相当し、切手のつながりが有向グラフの弧に相当する。

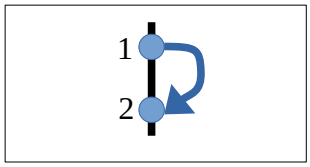

図 2A

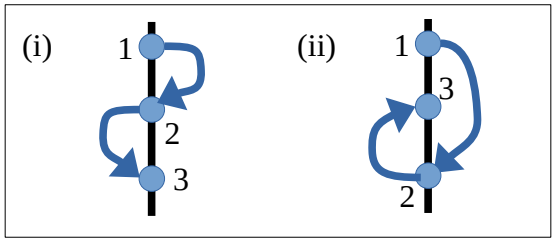

図 2B

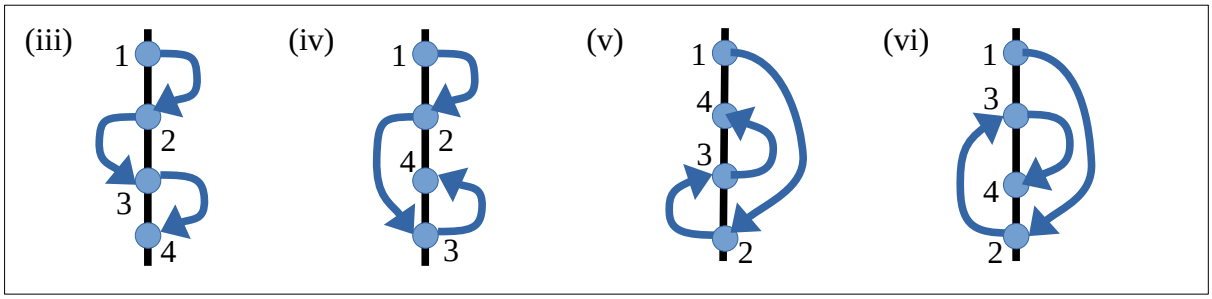

図 2C

非常に分かりやすくなった。こうして見ると、図 2B の(i)から図 2C の(iii)と(iv)が構成できるように見える。(iii)と(iv)は、切手 4 を切手 3 の下に生やすか、切手 2 と 3 の間に生やすかの違いのように見える。また、図 2B の(ii)から図 2C の(v)と(vi)が構成できるように見える。(v)と(vi)は、切手 4 を切手 1 と切手 3 の間に生やすか、切手 3 と切手 2 の間に生やすかの違いのように見える。このような図 2A～図 2C のような表示を**片山のグラフ**と呼ぶことにする。

片山のグラフの制約条件について考えよう。折りたたんだ後の切手 1 は一番上でなければならない。切手 1 から弧が出る場合、それは軸の右側へ出ていかないといけない。切手 1 から出て来る弧は、軸の右側に出て、切手 2 を終点とする。切手 2 から出て来る弧は、軸の左側に出て、切手 3 を終点とする。切手 3 から出て来る弧は、軸の右側に出て、切手 4 を終点とする。このように軸の

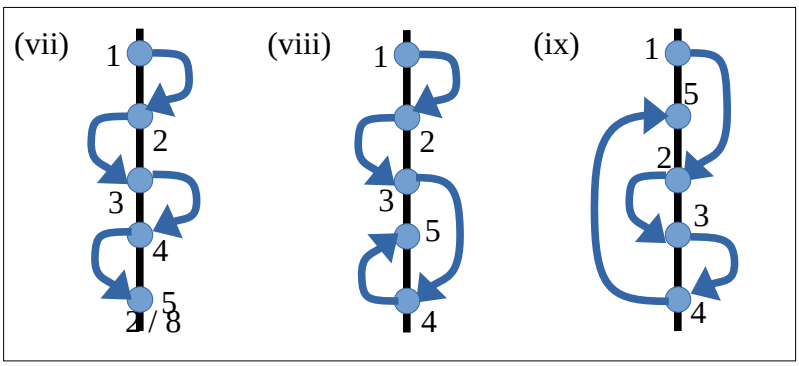

図 3A

「右側、左側、右側、左側」のように、交互に繰り返す。つまり、奇数性の切手を始点とする弧は、軸の右側に出る。偶数性の切手を始点とする弧は、軸の左側に出る。図 2C の(iii)から構成される片山のグラフは図 3A の(vii)、(viii)、(ix)のようになる。図 2C の(iv)から構成される片山のグラフは図 3B の(x)と(xi)のようになる。図 2C の(v)から構成される片山のグラフは図 3C の(xii)、(xiii)、(xiv)のようになる。

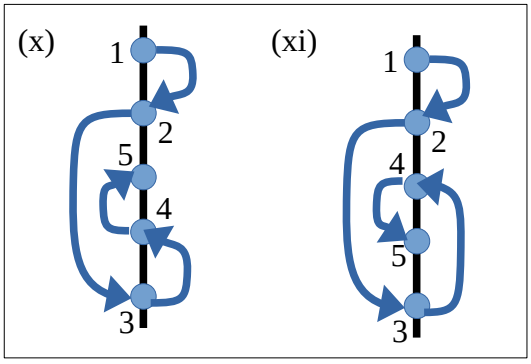

図 3B

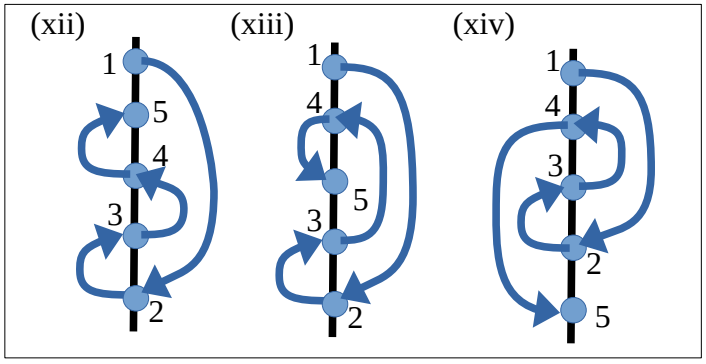

図 3C

弧の終点は切手 1 より上であってはならない。また、弧は、他の弧とクロスしてはならない。つまり、軸の左側の弧は別の左側の弧とクロスしない。軸の右側の弧は別の右側の弧とクロスしない。

片山のグラフを形式的に記号的に表現することを考える。

最初を 1 番目とするとき、順序対xのi番目の要素を$x[i]$と表すことにする。これを **indexing 関数**と呼ぶことにする。また、順序対xの長さを集合の濃度と同様に$|x|$と表すことにする。例えば、$(1,2,3,4)[3]=3$であり、$|(1,2,3,4)|=4$である。

図 3B の(x)は、$V=(1,2,5,4,3)$、$E_L=\{(2,3),(4,5)\}$、$E_R=\{(1,2),(3,4)\}$と表せる。Vは集合ではなく順序対とした。なぜなら郵便切手の問題では、切手の順序関係が重要であるからである。**軸の左側の弧**の集合E_Lと、**軸の右側の弧**の集合E_Rは、弧の始点の偶奇性とVから容易に求められる。すなわち、iが$1\leq i<|V|$を満たす自然数のとき、iが偶数ならば$(i,i+1)\in E_L$であり、iが奇数ならば$(i,i+1)\in E_R$である。最初は 1 でなければならないので、$V[1]=1$である。また、$x[i]$の逆写像が存在するとき、$x[i]$の逆写像を$x^{-1}[i]$と表すことにする。これを**逆 indexing 関数**と呼ぶことにする。例えば$(2,3,1)^{-1}[3]=2$であり、$(2,3,1)^{-1}[1]=3$である。n枚の切手$1,2,\cdots,n$からなるVには逆 indexing 関数$V^{-1}[i]$が存在する。

弧がクロスしないという条件をどう考えればいいか。有向グラフの弧には始点と終点があり、この問題の場合は始点と終点は異なるものである。弧xの始点は$x[1]$、終点は$x[2]$と表せる。ここで、軸から見て同じ側の異なる 2 つの弧x_1,x_2について

$$m_1=min(V^{-1}[x_1[1]],V^{-1}[x_1[2]]),\quad m_2=min(V^{-1}[x_2[1]],V^{-1}[x_2[2]]),$$

$$m_3=max(V^{-1}[x_1[1]],V^{-1}[x_1[2]]),\quad m_4=max(V^{-1}[x_2[1]],V^{-1}[x_2[2]])$$

と表す。$m_1<m_2<m_3<m_4$を満たすとき、**弧x_1,x_2はクロスしている**ということができる。

図2Cの(v)から図3Cの(xii)、(xiii)、(xiv)のみが構成されることを示そう。(v)は、$V=(1,4,3,2)$と表せる。(xii)、(xiii)、(xiv)は、それぞれ$V=(1,5,4,3,2)$、$V=(1,4,5,3,2)$、$V=(1,4,3,2,5)$と表せる。(v)の$(1,4,3,2)$のどこかに切手5を生やせば、(xii)、(xiii)、(xiv)を構成できるはずだ。最初は1でなければならないので、$(5,1,4,3,2)$はあり得ない。$(1,4,3,2)$に切手5を生やすとすれば、考えうる候補は$(1,5,4,3,2)$、$(1,4,5,3,2)$、$(1,4,3,5,2)$、$(1,4,3,2,5)$のいずれかとなる。候補$(1,5,4,3,2)$について、$E_L=\{(2,3),(4,5)\}$、$E_R=\{(1,2),(3,4)\}$となる。このE_L、E_Rについてクロスしている弧はない。よって候補$(1,5,4,3,2)$は構成可能。その次の候補$(1,4,5,3,2)$もクロスしている弧はない。よって候補$(1,4,5,3,2)$は構成可能。候補$(1,4,3,5,2)$について、$E_L=\{(2,3),(4,5)\}$、$E_R=\{(1,2),(3,4)\}$となるが、弧$(2,3)$と弧$(4,5)$がクロスしている。なぜなら$x_1=(4,5)$かつ$x_2=(2,3)$とするとき、

$$m_1=min(V^{-1}[4],V^{-1}[5])、m_2=min(V^{-1}[2],V^{-1}[3])、$$

$$m_3=max(V^{-1}[4],V^{-1}[5])、m_4=max(V^{-1}[2],V^{-1}[3])$$

となるが、$V^{-1}[4]=2$かつ$V^{-1}[5]=4$かつ$V^{-1}[2]=5$かつ$V^{-1}[3]=3$であるから、

$$m_1=min(2,4)=2、m_2=min(5,3)=3、m_3=max(2,4)=4、m_4=max(5,3)=5$$

すなわち、$m_1<m_2<m_3<m_4$となってクロスする条件を満たすからである。

このように郵便切手の問題の組み合わせは、「切手が1から始まる」という条件と「同じ側の弧がクロスしない」と言う条件が加わった、順列の部分集合のように見える。それは実際そうである。本書では数学的に厳密な証明は与えない。プログラム言語のC++言語で郵便切手の問題の構成的な計算方法を以下に表す。

プログラムリスト1

```
// postage_stamp.cpp --- A C++ program for the postage stamp problem of
math
#include <cstdio>
#include <vector>
#include <algorithm>
#include <cassert>

typedef std::vector<size_t> stamps_t;
typedef std::pair<size_t, size_t> edge_t;

bool is_crossing(const stamps_t& stamps)
{
    assert(stamps.size() > 0);

    stamps_t inverse;
    inverse.resize(stamps.size());
    for (size_t i = 0; i < stamps.size(); ++i)
    {
```

```
        inverse[stamps[i]] = i;
    }

    std::vector<edge_t> left_side, right_side;
    for (size_t i = 0; i < stamps.size() - 1; ++i)
    {
        if (i % 2 == 1)
            left_side.push_back(edge_t(i, i + 1));
        else
            right_side.push_back(edge_t(i, i + 1));
    }

    for (size_t i = 0; i < left_side.size(); ++i)
    {
        for (size_t j = 0; j < left_side.size(); ++j)
        {
            if (i == j)
                continue;

            edge_t x1 = left_side[i], x2 = left_side[j];
            size_t m1 = std::min(inverse[x1.first], inverse[x1.second]);
            size_t m2 = std::min(inverse[x2.first], inverse[x2.second]);
            size_t m3 = std::max(inverse[x1.first], inverse[x1.second]);
            size_t m4 = std::max(inverse[x2.first], inverse[x2.second]);
            if (m1 < m2 && m2 < m3 && m3 < m4)
                return true;
        }
    }

    for (size_t i = 0; i < right_side.size(); ++i)
    {
        for (size_t j = 0; j < right_side.size(); ++j)
        {
            if (i == j)
                continue;

            edge_t x1 = right_side[i], x2 = right_side[j];
            size_t m1 = std::min(inverse[x1.first], inverse[x1.second]);
            size_t m2 = std::min(inverse[x2.first], inverse[x2.second]);
            size_t m3 = std::max(inverse[x1.first], inverse[x1.second]);
            size_t m4 = std::max(inverse[x2.first], inverse[x2.second]);
            if (m1 < m2 && m2 < m3 && m3 < m4)
                return true;
        }
    }

    return false;
}

size_t postal_stamp(size_t n)
{
    assert(n > 0);

    stamps_t stamps;
    for (size_t i = 0; i < n; ++i)
    {
        stamps.push_back(i);
    }
```

```
    size_t count = 0;
    do
    {
        if (stamps[0] != 0)
            continue;
        if (is_crossing(stamps))
            continue;
        ++count;
    } while (std::next_permutation(stamps.begin(), stamps.end()));
    return count;
}

int main(void)
{
    for (size_t i = 1; i <= 10; ++i)
    {
        size_t number = postal_stamp(i);
        std::printf("postal_stamp(%d) = %d\n", (int)i, (int)number);
    }
    return 0;
}
```

このプログラムをC++コンパイラでビルドし、実行すると次のような結果が得られる。

```
katahiromz@SUPERHACKER MINGW32
/c/Users/katahiromz/Documents/work/Math/postage_stamp
$ g++ postage_stamp.cpp -o postage_stamp

katahiromz@SUPERHACKER MINGW32
/c/Users/katahiromz/Documents/work/Math/postage_stamp
$ ./postage_stamp.exe
postal_stamp(1) = 1
postal_stamp(2) = 1
postal_stamp(3) = 2
postal_stamp(4) = 4
postal_stamp(5) = 10
postal_stamp(6) = 24
postal_stamp(7) = 66
postal_stamp(8) = 174
postal_stamp(9) = 504
postal_stamp(10) = 1406
```

この結果は、前述の$f(n)$の値と一致していて、Wikipediaの Map folding に書かれている内容に一致する。C++は indexing がゼロから始まるので注意する。偶奇性も index がゼロから始まることに影響する。`std::next_permutation` は次の順列を計算する標準関数である。プログラム中の `inverse` は、逆 indexing 関数を与える。

このプログラムはさらに高速化できる。高速化したものを以下に示す。

プログラムリスト 2

```
// postage_stamp.cpp --- A C++ program for the stamp folding problem
#include <cstdio>
#include <vector>
#include <algorithm>
#include <cassert>

typedef std::vector<size_t> stamps_t;
typedef std::pair<size_t, size_t> edge_t;

bool is_crossing(const stamps_t& stamps)
{
    assert(stamps.size() > 0);

    const size_t n1 = stamps.size() - 1;

    for (size_t i = 1; i < n1; i += 2)
    {
        for (size_t j = i + 2; j < n1; j += 2)
        {
            edge_t x1 = { i, i + 1 }, x2 = { j, j + 1 };
            size_t m1 = std::min(stamps[x1.first], stamps[x1.second]);
            size_t m2 = std::min(stamps[x2.first], stamps[x2.second]);
            size_t m3 = std::max(stamps[x1.first], stamps[x1.second]);
            size_t m4 = std::max(stamps[x2.first], stamps[x2.second]);
            if (m1 < m2 && m2 < m3 && m3 < m4)
                return true;
            if (m2 < m1 && m1 < m4 && m4 < m3)
                return true;
        }
    }

    for (size_t i = 0; i < n1; i += 2)
    {
        for (size_t j = i + 2; j < n1; j += 2)
        {
            edge_t x1 = { i, i + 1 }, x2 = { j, j + 1 };
            size_t m1 = std::min(stamps[x1.first], stamps[x1.second]);
            size_t m2 = std::min(stamps[x2.first], stamps[x2.second]);
            size_t m3 = std::max(stamps[x1.first], stamps[x1.second]);
            size_t m4 = std::max(stamps[x2.first], stamps[x2.second]);
            if (m1 < m2 && m2 < m3 && m3 < m4)
                return true;
            if (m2 < m1 && m1 < m4 && m4 < m3)
                return true;
        }
    }

    return false;
}

size_t postal_stamp(size_t n)
{
    assert(n > 0);
```

```
    stamps_t stamps;
    for (size_t i = 0; i < n; ++i)
    {
        stamps.push_back(i);
    }

    size_t count = 0;
    do
    {
        if (stamps[0] != 0)
            continue;
        if (is_crossing(stamps))
            continue;
        ++count;
    } while (std::next_permutation(stamps.begin(), stamps.end()));
    return count;
}

int main(void)
{
    for (size_t i = 1; i <= 10; ++i)
    {
        size_t number = postal_stamp(i);
        std::printf("postal_stamp(%d) = %d\n", (int)i, (int)number);
    }
    return 0;
}
```

このプログラムも同じ結果を出力する。

結論

おそらく郵便切手の問題の構成的な計算方法をC++のプログラムで示すことができた。私はプロの数学者ではないので、厳密な証明は与えることができないが、示されたアルゴリズムを改良することは可能だろう。

参考にした資料

- 『数学100の問題』数学セミナー
- 『郵便切手の問題の解決に迫る』数学班、北村奈菜子
- https://en.wikipedia.org/wiki/Map_folding