■ 連載

Haskell プログラミング

Rubic キューブと置換の乗算

和田 英一（IIJ 技術研究所）
wada@u-tokyo.ac.jp

Rubic キューブ

Rubic キューブ（Rubik's cube とか magic cube（魔方体（魔–方体であって魔法–体ではない））ともいう）はハンガリーの Ern Rubik が 1974 年に発明したといわれる．Doug Hofstadter は物理的に不可能と思ったと書いている [1] が，私も最初にその説明を聞いたときはまさかと疑った．しかしその後実物を見てなるほどそうかと納得し，かつ驚いた（日本の特許だという話もある）．

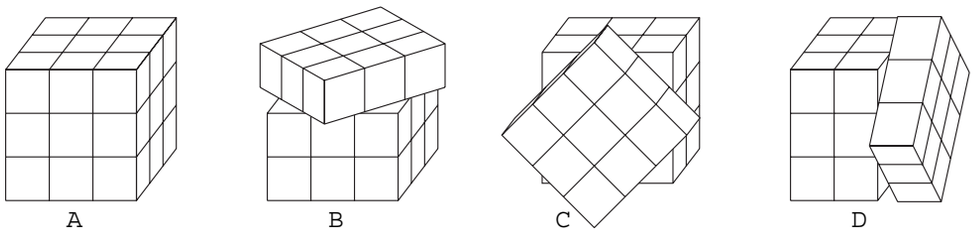

図-1

Rubic キューブは**図 -1** の A のように，3×3×3=27 個の立方体が集まっているように見え，1 辺が 5.7 センチの大きさで各面が異なる色に塗られている立方体である．どの方向からも 3×3×1=9 個の板状のものが 3 段重なっていると考えられ，上下端の板状のものは B, C, D のように中央を軸として回転できる．したがって表面の色はばらばらの位置に移動する．それを元のように戻すのを競うゲームである．

Rubic キューブで思い出すのは 1980 年夏，Stanford 大学での第 1 回 Lisp コンファレンスだ．昼休み，まわりを見回していると，Multics Emacs を話したばかりの Bernie Greenberg が来た．「やぁ」といっていると今度は Doug Hofstadter が近づいて来る．「ねぇ Doug, こちらは Bernie」というより早く Doug は「Eiiti は Bernie を知っていたのか」と大笑いになった．彼らがどういう知合いかは未詳．

Bernie は私が MIT の ProjectMAC にいたとき，博士課程の学生で，1974 年に Multics のページ管理で D 論を書いた．論文を出し終わると私の部屋へ来て，空き机の上で仰向けに眠りこけた．指導教官の Jerry Saltzer は Multics のページ管理を完全に理解しているのは彼のほかにはいないと断言する．

Doug と会ったのは 1979 年，東京での IJCAI（人工知能国際会議）の時だ．その直前の Scientific American で M. Gardner による Gödel, Escher, Bach の書評を読んでいたので，記事にあった GEB を 3 次元でくり抜いた表紙の図案を覚えていた．そのT シャツを着ていたのが Scott Kim で，この絵知っていると言うと Kim は隣を指し，これが著者（Doug Hofstadter）だといった．それを機に Doug と知り合いになった．

さて Stanford のキャンパス．Bernie は Rubic キューブを取り出すと Doug に捏ねろという．Doug が散々捻って返すと，Bernie は 10 秒ほどで元に戻してしまった．後で Metamagic Game に出た Doug の Rubic キューブの話[1] をよく読むと，Doug に Rubic キューブの存在を教えたのは Bernie であった．

Rubic キューブは当時 2,000 円ほどで売っていたが，最近は見かけない．日本にも凝っていた人がいた．最近でも Web をサーチすると，関連するページがいろいろ見つかる．

島内本

立教大学の島内剛一先生（1989 年 12 月 19 日ご他界）も凝っていた 1 人で，自己の完成法を S 流と称し，「ルービック・キューブ免許皆伝」を著された[2]．私は島内先生と親しかったので，著書を頂戴した．これはどう見ても数学者の著作でしかない．天の巻から少し引用する．

「1. ルービック･キューブ（魔方体）の全体（全方体）は, 26 個の小方体（小体）によって 6 つの面が構成されている．この 26 個は，つねに 1 面を外に向けている 1 面体 6 個，2 面を外に向けている 2 面体 12 個，3 面を外に向けている 3 面体 8 個に分類される．」

「操作手順などを説明記述するために，6 面の方向を，上 (T), 下 (B), 東 (E), 北 (N), 西 (W), 南 (S) と表し，手順の始まる前の小方体の各面を表すにもこの記号を使うことにする．」

「ルービック・キューブの単位操作は，1 面を共有する 9 個の小方体を，中央の 1 面体の回りに直角に回転することである．時計回りの回転は，面の名前を小文字にしたもの (t, e, n, w, s, b) を使い，反時計回りの回転は，その右肩に "$^-$" を付けたもの ($t^-, e^-, n^-, w^-, s^-, b^-$) を使って書き表すことにする．」

「操作を続けて行うのは，操作を表す記号を左から右へ続けて書き，同じことを 2 回，3 回と続けるのは，右肩に "^{2}", "^{3}" などを付ける．」

「式による表現は，それぞれの 2 面体，3 面体を縦に 2 文字，または 3 文字の列で表し，（位置や向きが変わった小方体について，）どこにあったものがどこに移ったかを，ある列とすぐ右の列とを見比べれば分かるように，書き表したものである．」

「たとえば，単位操作 e は

$$e = \begin{bmatrix} \mathrm{STNBS} \\ \mathrm{EEEEE} \end{bmatrix} \begin{bmatrix} \mathrm{TNBST} \\ \mathrm{EEEEE} \\ \mathrm{STNBS} \end{bmatrix}$$

と書ける．」

「むすびと呼ばれる手順 $enwst^2s^-w^-n^-e^-t^2$ についていえば，式を使って

$$enwst^{\,2}\, s^-w^-n^-e^-t^{\,2} = \begin{bmatrix} \mathrm{SSNS} \\ \mathrm{ETTE} \end{bmatrix}$$

と書ける．」

上の手順を**図 -2** に示す．e では最初の E の図が F になる．むすびの移動は G の通り．

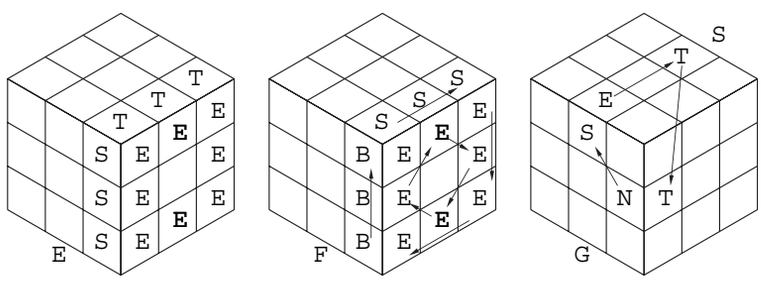

図-2

実はこの手順の並びによる小方体の移動関係を島内先生がどう確認したか，知りたいのである．長い間の疑問であったが，このたび Haskell で置換の積を計算するプログラムを書き，基本的にはいろいろ分かってきた．

Rubic キューブの話題にはときどき出会った[3),4)]．いろいろな問題として考えられるが，ここでは置換の積の計算のそれとして捉える．

置換の積

置換の話は Knuth の The Art of Computer Programming，第 1 巻の 1.3.3 Applications to Permutations[5)] にあるから読んだ人も多いであろう．applications とはこの直前に例の MIX の話があり，MIX の置換への応用ということである．しかし MIX のプログラムは読む気がしないので，Haskell で考えることにしよう（本連載はいわば The Art of Functional Programming である）．

置換の復習にはその回転がプロムナードにも登場した[6)] サイコロを使う．サイコロは島内流に方向が決めてあるとする．上 (T) を北 (N) へ倒し，次に新しい上 (T) を西 (W) に倒すと各面はどこへ移るか．

上 (T) は 1 回目に北 (N) へ移り，2 回目は不変．南 (S) は 1 回目に上 (T) へ移り，2 回目に西 (W) へ移る．西 (W) は 1 回目は不変，2 回目は下 (B) へ移る．北 (N) は 1 回目に下 (B) へ移り，2 回目に東 (E) へ移る．東 (E) は 1 回目は不変, 2 回目は上 (T) へ移る．下 (B) は 1 回目に南 (S) へ移り, 2 回目は不変．したがってこの 2 操作で生じる置換は，Knuth 流に出発点を上段，到着点を下段に書くと

$$\begin{pmatrix} \mathrm{T\ S\ W\ N\ E\ B} \\ \mathrm{N\ W\ B\ E\ T\ S} \end{pmatrix}$$

となる．これは上下の対応だけが問題で，左右の順はどうでもよいから，

$$\begin{pmatrix} \mathrm{T\ E\ S\ B\ W\ N} \\ \mathrm{N\ T\ W\ S\ B\ E} \end{pmatrix}$$

と書いてもよい（Knuth だとほかに 718 通りの書き方があると書く）．

こういう上下対応の表現のほか，T は N へ，その N は E へ，その E は T へ戻る．また B は S へ，その S は W へ，その W は B へ戻ると分かるので，

$$(\mathrm{T\ N\ E})(\mathrm{B\ S\ W})$$

と表現することもある．これはかっこ内は左から順に右へ移る先が書いてあり，最後からは最初へ戻るのでサイクル表現という．島内流はこれに近い．

置換を続けて行うと，結果の置換はサイクル表現の積で表される．

先ほどの北へ倒す置換のサイクル表現は (T N B S) であり，西へ倒す置換は (T W B E) である．(T N B S) (T W B E) の積はどうなるか．まず T は左の置換で N になる．そこで T→N．右の置換には N がないので，結局 T→N．N は左の置換で B へ，その B は右の置換で E になるので N→E．B は左の置換だけで B→S．S は T を経て S→W．左の置換にない E と W は右の置換でそれぞれ E→T，W→B．全部書くと T→N，N→E，E→T; B→S，S→W，W→B．これで上のサイクル表現が得られる．

サイコロ置換のプログラム

では上の置換の積を計算するプログラムを書いてみよう．Lisp だと T や E は記号アトムで表現すればよいが，型にうるさい Haskell ではそうはいかない．数値や文字列以外は自分で型を宣言するところから始める．

計算の対象 (object) というつもりで，その Obj 型を決める．

```
data Obj = T | S | E | B | N | W deriving (Eq, Enum, Show)
```

これは Obj 型の要素を宣言する．つまり T, S, E,... などが Obj 型である．最後の deriving (Eq, Enum, Show) はこの型の Eq, Enum と Show のクラスの instance 宣言を自動的にするもので，これによりこの型は等不等，数列

表記，出力ができるようになる．やってみると

```
Main> T == T
True
Main> [T .. W]
[T,S,E,B,N,W]
Main> show T
"T"
```

というわけだ．まず全体のプログラムを示す[☆1]．

```
import List                  -- Listのモジュールを読み込む
data Obj = T | S | E | B | N | W deriving (Eq, Enum, Show)
type Perm = [(Obj, Obj)]     -- 型に名前をつける
type Cycle = [Obj]
type CyclePerm = [Cycle]

allObj :: [Obj]
allObj = [T .. W]            -- Objの全体（リスト）

go :: Obj -> Cycle -> Obj    -- goesToの下請け 1個のサイクルでの移動
go o c = case elemIndices o c of
         []  -> o
         [i] -> cycle c !! succ i   -- cycle c は c++c++...,succ i は i + 1

goesTo :: Obj -> CyclePerm -> Obj
goesTo = foldl go            -- Obj のリストに左から go を作用させる

assoc :: Eq k => k -> [(k, v)] -> [(k, v)]      -- Lispのassocのようなもの
assoc c as = [(k, v) | (k, v) <- as, c == k]

makeCycle0 :: Perm -> CyclePerm -> CyclePerm    -- サイクル表現にする
makeCycle0 [] qs = qs
makeCycle0 ((x,y):ss) qs
  | x == y   = makeCycle0 ss qs                 -- 単一サイクル除去
  |otherwise = makeCycle1 ss ([x,y]:qs)

makeCycle1 :: Perm -> CyclePerm -> CyclePerm
makeCycle1 ss (cs:css)
  | c == head cs  = makeCycle0 ss' (cs:css)
  | otherwise     = makeCycle1 ss' ((cs ++ [c]):css)
        where c = snd d
              d = head (assoc (last cs) ss)
              ss' = delete d ss

prodPerm :: [CyclePerm] -> CyclePerm       -- 各種の手順による置換の計算
prodPerm ops = makeCycle0 (zip allObj allObj') []
           where allObj' = map (flip goesTo (concat ops)) allObj

t,s,e,b,n,w :: CyclePerm    -- 回転の定義
t = [[S,W,N,E]]
s = [[E,B,W,T]]
e = [[B,S,T,N]]
b = [[N,W,S,E]]
n = [[W,B,E,T]]
w = [[T,S,B,N]]
```

たとえば S が [[S,W,B],[T,N,E]] の置換によりどこに移る (goto) かを計算する goesTo だが

[☆1] このプログラムは http://www.sampou.org/haskell/ipsj/ から取ることができる．

```
Main> goesTo S [[S,W,B],[T,N,E]]
W
```

のように使う．goesTo は置換の列を順に見ていく．その単位の作業を go が担当する．go は出現位置の添字のリストを返す elemIndices を使う．assoc は Lisp のそれと類似しているが，リストを返す．

```
Main> elemIndices 1 [3,1,4,1,5]
[1,3]
Main> assoc E [(T,B),(S,N),(E,W)]
[(E,W)]
```

go ができればあとはこれを置換のリストに左から操作すればよく，goesTo = foldl go とできる．

prodPerm は置換 (permutation) の積 (product) を作る．入力は CyclePerm 型，すなわち [[S,W,B],[T,N,E]] のようなものの並び，いいかえれば置換の並びである．それをとりあえず concat ops でサイクルの並びにし，map (flip goesTo (concat ops)) allObj によりすべての Obj (allObj) を goesTo での移り先のリストにする (flip goesTo (concat ops) は \x -> goesTo x (concat ops) と同じ)．これを allObj' としておく．例でいえば concat した結果の [[S,W,B],[T,N,E]] により，allObj(= [T, S, E, B, N, W]) は [N, W, T, S, E, B] へ移ることが分かる．allObj と allObj' を zip すると [(T,N), (S,W), (E,T), (B,S), (N,E),(W,B)] ができる．

makeCycle0, makeCycle1 はサイクル表現を作る．makeCycle0 の第 1 引数は zip の結果．第 2 引数 qs は結果の場所で最初は空リストである．makeCycle0 の最初の定義は第 1 引数を順に処理した結果，それが空リストになったら，結果を返す．そうでなければ，zip で作った対のリストの先頭を (x,y) とし，まず x == y なら単一サイクルだから，これは棄てる．そうでなければ，結果のリストに [x, y] を前付けし，残りの対のリストを持って，makeCycle1 を呼ぶ．

makeCycle1 は対のリストが ss, 結果のリストが (cs:css) となっている．現在構築中のサイクルが cs であり，すでに構築済みのサイクルリストが css である．まず cs の最後の Obj を先頭に持つ対を assoc で探し，それを d とする．d の対の 2 番目 (snd d) が今度の Obj の移り先なので，これを c とする．一方，対のリスト ss から d を取り除いた (delete した) ものを ss' とする．c が cs の head（Lisp の car のこと）と同じなら，サイクルは完成なので，makeCycle0 へ戻り，次のサイクル構築にかかる．そうでなければ，cs の最後に c を連結し，makeCycle1 を継続する．

奥へ倒し，左へ倒す [e,n] の置換を計算してみる．また [e,e,e,e] もやってみる．

```
Main> prodPerm[e,n]
[[S,W,B],[T,N,E]]
Main> prodPerm[e,e,e,e]
[]
```

Knuth の本の 1.3.3 節には計算機向きとして Algorithm B が出ている．これは全 Obj について goesTo を並列に 1 パスで処理する．そのプログラムも Haskell で書いてあるが省略する．

Rubic キューブの場合

島内本では Obj は SE のような 2 面体 12 個と TES のような 3 面体 8 個である．しかし SE は見方によっては ES と見えるかもしれない．また 3 面体は，3 個の文字を時計回りに見るか (TES)，反時計回りに見るか (TSE) の選択がある．しかもどの文字から開始するかで 3 通りの表現がある．回り方は時計回りとし，3 文字いずれからも始める Obj を用意することにした．したがって Obj は 12×2 + 8×3 で 48 個になった．単一サイクルをはずすから，いらないものは出てこないということで，これで始めることにした．サイコロと同様に data などを宣言する．

```
-- 2面体と3面体の型
data Obj = TE | TS | TW | TN | ET | ES | EB | EN | ST | SE | SB | SW
         | BE | BS | BW | BN | WT | WS | WB | WN | NT | NE | NB | NW
         | TES | EST | STE | TSW | SWT | WTS | TWN | WNT | NTW | TNE
         | NET | ETN | BSE | SEB | EBS | BEN | ENB | NBE | BNW | NWB
         | WBN | BWS | WSB | SBW deriving (Eq, Enum, Show)

allObj :: [Obj]          -- すべての2面体と3面体のリスト
allObj = [TE .. SBW]
```

以上はサイコロと同じだが, 6 個の Obj が 8 倍に増えた．次は回転．Haskell ではプライム付きの名前が使えるから, 逆回転はプライム付きで表現することにする.

```
e,e',s,s',w,w',n,n',t,t',b,b' :: CyclePerm  -- 各面を時計回りに90度回転
e = [[SE,TE,NE,BE],[ES,ET,EN,EB],
     [TES,NET,BEN,SEB],[STE,TNE,NBE,BSE],[EST,ETN,ENB,EBS]]
s = [[WS,TS,ES,BS],[SW,ST,SE,SB],
     [TSW,EST,BSE,WSB],[WTS,TES,EBS,BWS],[SWT,STE,SEB,SBW]]
w = [[NW,TW,SW,BW],[WN,WT,WS,WB],
     [TWN,SWT,BWS,NWB],[NTW,TSW,SBW,BNW],[WNT,WTS,WSB,WBN]]
n = [[EN,TN,WN,BN],[NE,NT,NW,NB],
     [TNE,WNT,BNW,ENB],[ETN,TWN,WBN,BEN],[NET,NTW,NWB,NBE]]
t = [[ST,WT,NT,ET],[TS,TW,TN,TE],
     [WTS,NTW,ETN,STE],[SWT,WNT,NET,EST],[TSW,TWN,TNE,TES]]
b = [[SB,EB,NB,WB],[BS,BE,BN,BW],
     [EBS,NBE,WBN,SBW],[SEB,ENB,NWB,WSB],[BSE,BEN,BNW,BWS]]
e' = prodPerm [e,e,e]  -- e'などはeなどの反時計回り回転，時計回り3回で定義
s' = prodPerm [s,s,s]
w' = prodPerm [w,w,w]
n' = prodPerm [n,n,n]
t' = prodPerm [t,t,t]
b' = prodPerm [b,b,b]
```

以下は島内本から手順の割りには結果が単純なものを選んで実行したものである．手順名と手順の番号，島内本での式を書き，その後このプログラムでの実行結果がある．4 つの手順を**図 -3** の H, I, J, K に示す．似たような移動なのに手順名がまったく関係ないのが気に入らぬがそれはとにかく，隣辺向き変えとねじりの式は他のと違う．つまり最左の組が再び現れる前に終わっている．島内本の表現がそうなっている．最左の組がそのままでなくても，巡回シフトした組が現れると分かったことにしたらしい.

• 単純 3 角形 9a（図 -3 H）

$$s^2(t[e^-w]s^2[ew^-]t)s^2 = \begin{bmatrix} \mathrm{T} & \mathrm{T} & \mathrm{T} & \mathrm{T} \\ \mathrm{S} & \mathrm{W} & \mathrm{E} & \mathrm{S} \end{bmatrix}$$

```
Main> prodPerm[s,s,t,e',w,s,s,e,w',t,s,s]
[[ET,ST,WT],[TE,TS,TW]]
```

• 隣辺向き変え 24a（図 -3 I）

$$(t^-n^-b^-s^-)e^-(sbnt)e \bullet (sbnt)e(t^-n^-b^-s^-)e^- = \begin{bmatrix} \mathrm{T} & \mathrm{S} \\ \mathrm{S} & \mathrm{T} \end{bmatrix}\begin{bmatrix} T & \mathrm{E} \\ \mathrm{E} & T \end{bmatrix}$$

```
Main> prodPerm[t',n',b',s',e',s,b,n,t,e,s,b,n,t,e,t',n',b',s',e']
[[TS,ST],[TE,ET]]
```

• 巡回 28a（図 -3 J）

$$e^2s^2e^-n^-es^2e^-ne^- = \begin{bmatrix} \mathrm{T} & \mathrm{T} & \mathrm{T} & \mathrm{T} \\ \mathrm{E} & \mathrm{S} & \mathrm{N} & \mathrm{E} \\ \mathrm{S} & \mathrm{W} & \mathrm{E} & \mathrm{S} \end{bmatrix}$$

```
Main> prodPerm[e,e,s,s,e',n',e,s,s,e',n,e']
[[STE,WTS,ETN],[EST,SWT,NET],[TES,TSW,TNE]]
```

• ねじり 33a（図 -3 K）

$$e(t^2 es^{-}b^2 se^{-})^2 e^{-} = \begin{bmatrix} \mathrm{T} & \mathrm{E} \\ \mathrm{E} & \mathrm{S} \\ \mathrm{S} & \mathrm{T} \end{bmatrix} \begin{bmatrix} T & \mathrm{W} \\ \mathrm{S} & \mathrm{T} \\ \mathrm{W} & \mathrm{S} \end{bmatrix}$$

```
Main> prodPerm[e,r,r,e'] where r = prodPerm[t,t,e,s',b,b,s,e']
[[TSW,WTS,SWT],[TES,EST,STE]]
```

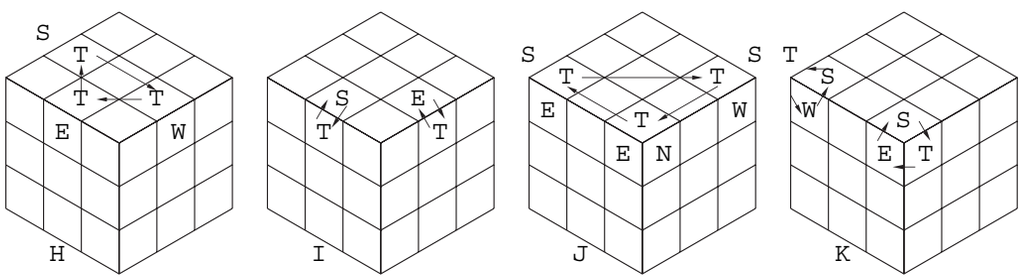

図-3

6 面完成術

図 -3 を眺めると

• H:（単純 3 角形）3 個の 2 面体の移動

• I:（隣辺向き変え）2 個の 2 面体の回転

• J:（巡回）3 個の 3 面体の移動

• K:（ねじり）2 個の 3 面体の回転

になっている．捻り方は複雑だが，他の小方体には影響をまったく与えない．つまりこれだけ知っていれば，2 面体同士を入れ替え，向きを揃え，次に 3 面体同士を入れ替え，向きを揃え，全面が完成する．Rubic キューブをしっかり保持し，手順を間違えないようにして揃えていくと，おぅ！完成するではないか．途中で間違えると再度ばらばらになり，新規まき直しになるが．

図 -3 の 4 つの手順は最低これだけ知っていれば，効率は悪かろうと 6 面が完成するというので選んだ．もちろん Bernie Greenberg はこんなやり方ではなく，効率のよい手順をたくさん知っていたはずだ．

それにしても島内先生はどのようにしてこれだけ書いたのか．一緒に議論した東大の米田信夫先生（1996 年 4 月 21 日ご逝去）は 1 年後輩の島内さんから「精神力が足りない」とよく叱られていた．してみると島内さんは精神力だけで置換の積を求めていたのかも知れぬ．それが数学者なのかも知れぬ．数学者は安易にプログラムを書かぬものらしい．

参考文献

1）Hofstadter, D.: キューブ術, キューブ芸術, キュービズム, (竹内郁雄, 斉藤康己, 片桐恭弘訳: メタマジック・ゲーム, 白楊社, 1990).
2）島内剛一: ルービック・キューブ免許皆伝, 日本評論社 (1981).
3）小谷善行: ルービック・キューブをエレガントに表す, ナノピコ教室解答, bit, Vol.13, No.13 (Nov. 1981).
4）古川康一: PROLOG による問題解決, 情報処理学会第 23 回全国大会 5G-2.
5）Knuth, D. E.: The Art of Computer Programming, Volume 1, Third edition, Addison-Wesley (1997).
6）寺田　実: サイコロパズル, 情報処理, Vol.46, No.1, pp.75-79 (Jan. 2005).

（平成 17 年 5 月 12 日受付）