情報理工学基礎 1 レポート課題

tax_free

2020 年 7 月 26 日

概要

授業で扱った内容 (教科書 [1] 二章まで) を Python を用いて実装した．また，それらの概念を目で見て理解できるように，集合，写像，合成写像の状態を表示する Python のプログラムを作成した．

目次

第 I 部　集合 3

1 集合と組 3
- 1.1 Python における集合の実装 . . . 3
- 1.2 部分集合の実装と集合の同一性 . . . 5
- 1.3 Python における組の実装 . . . 6
- 1.4 組の同一性 . . . 6
- 1.5 Python における列と記号列の実装と同一性 . . . 7

2 集合演算 7
- 2.1 共通部分集合 . . . 8
- 2.2 和集合 . . . 8
- 2.3 差集合 . . . 9
- 2.4 補集合 . . . 9
- 2.5 直積 . . . 9
- 2.6 直和 . . . 10
- 2.7 べき . . . 10

第 II 部　写像 11

3 写像 11
- 3.1 写像の定義と Python での実装 . . . 11
- 3.2 写像の同一性 . . . 11
- 3.3 写像の集合 . . . 12

4 写像の合成 13

5 様々な写像 13

5.1 単射，全射，全単射 13
5.2 恒等写像と逆写像 16
5.3 置換 17

6 写像と集合 18
6.1 同型 18

第 III 部　tkinter を用いて写像を可視化しよう 18

7 モチベーション 18

8 Visualizer の概要 18

9 生成されたもの 19

第 IV 部　アピール 21

実行環境

このレポートにあるソースコードは，表 1 の環境で実行した．

OS	MacOS Big Sur 11.6
CPU	Core i5-8259U
メモリ	8GB LPDDR3
Python version	3.8.9
tkinter version	8.5.9

表 1　実行環境

第 I 部
集合

この部では，Python で集合や組，それに関連する概念を実装する．

1　集合と組

1.1　Python における集合の実装

教科書 [1] の定義に従うと集合は，次で定義される．

> 要素の集まりを集合と呼ぶ．

また，集合は要素の重複を許さないから，集合の実装で要求されるのは「要素の重複を許さず，かつ要素が順序をもたない」データ型である．Python には，集合を表現するためのデータ型があり Set オブジェクトと呼ばれる (以下，単に Set と書く)．Set は，組み込み関数 `set()`を用いるか，要素を {} で囲むことで実装できる．前述の通り，数学における集合と同様に Python の Set は，重複する要素を持たず，集合の要素 (元) に順番を持たない [2]．Set の要素になることができるのは，ハッシュ化可能 (hashable) なオブジェクトのみである．ここで，ハッシュ化可能なオブジェクトとは，あるオブジェクトが存在している間はそのハッシュ値が変化せず，他のオブジェクトと比較できるオブジェクトのことである [3]．ハッシュ化可能なオブジェクトは，文字列型 (string) やタプル型 (tuple) 等で，逆にハッシュ化不可能なオブジェクトは，リスト型 (list) や辞書型 (dictionary) 等がある．Set の要素がハッシュ化可能なオブジェクトに制限されるのは，「同値なハッシュ化可能オブジェクトは必ず同じハッシュ値を持つ」という性質を集合の要素に重複がないかを調べるのに使っているからである．

Python で整数 1, 2, 3 を要素に持つ集合 A を実装するとソースコード 1 になる．

ソースコード 1　1, 2, 3 を要素に持つ集合 A

```
1    A = set([1, 2, 3])
```

また，空集合 $\emptyset$ は，ソースコード 2 で実装できる．ソースコード 3 では，集合型ではなく辞書型になる．

ソースコード 2　空集合 $\emptyset$ の正しい実装

```
emptyset = set()
```

ソースコード 3　空集合 $\emptyset$ の誤った実装

```
emptyset = {} #type(emptyset) = <class 'dict'>
```

集合に要素を追加するときは，addメソッドを用いる．逆に集合から要素を取り除くときは，discard，remove等のメソッドを用いる．discard，remove等の違いはドキュメント [5] を参照せよ．例えば，集合 A に整数 5 を加えた後に整数 1 を除いて $A = 2, 3, 5$ にする操作，ソースコード 4 実装できる．

ソースコード 4　集合に要素を追加/削除

```
A = set([1, 2, 3])
A.add(5) #A = {1, 2, 3, 5}
A.discard(1) #A = {2, 3, 5}
```

ところで，これまでに書いた「集合」とは有限集合のことであった．では，無限集合はどのように実装できるのだろうか．コンピュータ上では，メモリに変数を乗せてその値を保持している．その大きさは，一般に集合の要素が増えると大きくなるので，高々有限なメモリしかない現実のコンピュータで直接的に無限集合を実装することは不可能である．よって，以降，断りがなければ集合は有限集合である．

集合を実装できたので，次は集合族 F を実装する．ここで，集合族 $F = \{\{1\}, \{2\}\}$ をソースコード 5 のように実装すると，TypeError が返される．これは，Set がハッシュ化不可能なオブジェクトだからである．つまり，Set は要素にハッシュ化可能なオブジェクトを要求するが，Set 自身はハッシュ化可能ではない．

ソースコード 5　集合族の誤った実装

```
F = set([set([1]), set([2])]) #TypeError: unhashable type: 'set'
```

これを解決するために Python には，ハッシュ化可能な集合型 frozenset が用意されている．frozenset を用いれば集合族 F をソースコード 6 で実装できる．

ソースコード 6　集合族の正しい実装

```
F = set([frozenset([1]), frozenset([2])])
```

集合の要素数は，組み込み関数 len()で実装できる．例えば集合 A の要素数 a は，ソースコード 7 で実装できる．

ソースコード 7　有限集合の要素数

```
A = set([1, 2, 3])
a = len(A) #3
```

ある集合 (族) にある要素が「属する」,「属さない」は，所属検査演算 inによって実装できる．[*1]ある要素 x が集合 A に属するとき，ソースコード 8 の一行目は Trueを返し，そうではないときは Falseを返す．集合族について調べるときに所属検査演算 in の前に書くオブジェクトは，frozenset ではなく Set でもよい．

ソースコード 8　所属検査演算 in

```
x in A
```

これを用いて集合 A に整数 1, 5 が属するかを調べるにはソースコード 9 のように書けばよい．

[*1] 一般に Set オブジェクトに対する所属検査演算 in の計算量は $\mathcal{O}(1)$ で要素数に依存しない．

ソースコード 9　所属検査演算 in の例

```
A = set([1, 2, 3])
1 in A #True
5 in A #False
```

1.2　部分集合の実装と集合の同一性

教科書 [1] の定義に従うと部分集合は，次で定義される．

> 任意の集合 A と B に対し，$a \in A$ であるならば $a \in B$ であることが任意の $a \in A$ で対して成り立つとき，A は B の部分集合であると呼ばれ $A \subseteq B$ と表す．

任意の集合 A と B が互いに部分集合であるかの判定には，組み込み演算子<=，>=[6]，または，issubsetメソッド [5] を用いる．部分集合であるならば，Trueを返す．例えば，10 以下の非負整数の集合 N，10 以下の偶数の非負整数の集合 E，10 以下の奇数の非負整数の集合 O の包含関係は，

$$E \subseteq N,\ O \subseteq N, O \nsubseteq E,\ E \nsubseteq O$$

となる．この包含関係を調べるには，ソースコード 10 のように書けばよい．

ソースコード 10　部分集合の確認

```
N = {i for i in range(0, 11)}
E = {i for i in range(0, 11, 2)}
O = {i for i in range(1, 11, 2)}

print(E <= N) #True
print(N >= O) #True
print(O <= E) #False
print(O.issubset(E)) #False
```

また，集合の同一性とは，教科書 [1] の定義によると

> 任意の集合 A と B は，$A \subseteq B$ であり，かつ $B \subseteq A$ であるとき，等しいと呼ばれ $A = B$ と表す．

とあるから，$A >= B$ と $A <= B$ がどちらも True であればよいことになる．例えば例題 1.8 を実装するとソースコード 11 になる．ただし，p and qは，p かつ q が真なら Trueを，偽なら Falseを返す．

ソースコード 11　例題 1.8

```
B = {0, 1}
N_2 = {0, 1}

print(B <= N_2 and B >= N_2) #True
```

これで，集合の同一性を確認することができるが，$A >= B$ と $A <= B$ を書くのは面倒であるから，$A >= B$ と $A <= B$ をまとめて $A == B$ と書く．集合 A と B が同一であるとき True を返す．これを用いればソースコード 11 四行目は print(B == N_2)と書き換えられる．

また，教科書 [1] の定義に従うと真部分集合は，次で定義される．

> 任意の集合 A と B に対し，$A \subseteq B$ であり，かつ $A \neq B$ であるとき，A は B の真部分集合であると呼ばれ $A \subset B$ と表す．

部分集合の包含関係の判定と同様に組み込み演算子<，>，または issubsetメソッドを用いればよい．

1.3 Python における組の実装

教科書 [1] の定義に従うと組 (tuple) は，次で定義される．

> 組は順序がつけられている成分の集まりである．

また，組は集合と異なり要素の重複を許すから，組の実装に要求されるのは「要素が順序を持ち，要素が重複してもよい」型である．Python では，この要求を満たす型としてリスト型 (list) とタプル型 (tuple)[5] がある．これらの大きな違いは，リスト型はミュータブルで，タプル型はイミュータブルであるという点である．前述の通り，Set はハッシュ化可能 (イミュータブルなオブジェクトのほとんど) なオブジェクトである必要であるから，組を要素にする集合を考えるときを考えると，組はタプル型で実装する．

タプルオブジェクトは，組み込み関数 tuple()を用いるか，要素を ,で区切ることで実装できる．例えば，整数 1, 2, 3 を要素に持つ組 P は，ソースコード 12 で実装できる．

ソースコード 12　整数 1, 2, 3 を要素に持つ組の実装

```
1    P = 1, 2, 3
2    P = tuple(1, 2, 3)
```

Set と異なり空の組 P_{empty} は，ソースコード 13 一行目，または二行目で実装できる．これらは等価な表現であることをソースコード 17 で確認する．

ソースコード 13　空の組

```
1    P_empty = ()
2    P_empty = tuple()
```

組の大きさ，つまりタプルオブジェクトの大きさは，Set と同様に組み込み関数 len()で実装できる．組 P の要素数 p は，ソースコード 14 で実装できる．

ソースコード 14　組の大きさ

```
1    P = 1, 2, 3 # = tuple(1, 2, 3)
2    p = len(P) #p = 3
```

また，組の i 番目の要素は，tuple[i]で実装できる．ただし，i は整数で $0 \leq i \leq$ タプルオブジェクトの大きさ で，0-indexed であることに注意する．例えば，P の 2 番目の要素を p_2 は，ソースコード 15 で実装できる．

ソースコード 15　組の i 番目の要素

```
1    P = 1, 2, 3
2    p_2 = P[1] #p_2 = 2
```

1.4 組の同一性

教科書 [1] の定義に従うと組の同一性は，次で定義される．

> 任意の組 A と B は，大きさが等しく，かつ各成分の値がすべて等しいとき，等しい (equal) と呼ばれ $A = B$ と表す．

$len(A)$ と $len(B)$ を確認して，for文で要素を確認することで組の同一性を確認することもできるが，タプル型がハッシュ化可能なオブジェクトであることを思いだすと，オブジェクトのハッシュ値を調べれば十分であることが分かる．つまり，任意の組 A と B が等しいとき，そのタプル A とタプル B のハッシュ値も等しい．よって，id(A)== id(B)が Trueならば $A = B$，Falseなら $A = B$ ではない．これはさらに A == Bと省略でき，任意の組 A と B が等しいとき Trueを返す．例えば，組 $Q = (1, 2, 3)$ を新たに定義して組 P との同一性は，ソースコード 16 で確認できる．

ソースコード 16　組 P と Q の同一性

```
1    P = 1, 2, 3
2    Q = 1, 2, 3
3    print(P == Q) #True
```

また，異なる表記をした空の組の同一性を調べるプログラムはソースコード 17 である．これが Trueを返すことから，ソースコード 13 の一行目と二行目は等価であることが分かる．

ソースコード 17　空のタプルオブジェクトの同一性

```
1    print(() == tuple()) #True
```

1.5　Python における列と記号列の実装と同一性

列 (sequence) は，for文を用いてタプルオブジェクトから要素を順番に取り出せば実装できる．例えば，10 以下の非負整数の組 S の列は，ソースコード 18 で実装できる．

ソースコード 18　列 (sequence)

```
1    S = (i for i in range(0, 11))
2    for i in S:
3        print(i)
```

組 S の列の長さは 11 である．

a から z までの小文字のアルファベット (十進 ASCII コードが 97 から 122 まで) からなる有限集合 Σ を考える．このとき有限集合 Σ は文字集合であると言えるから，有限集合 Σ 上の列は記号列 (string) であると呼ばれる．例えば，apple は Σ 上の記号列だが，Apple は $A \notin \Sigma$ であるから Σ 上の記号列ではない．記号列の実装は，文字集合の定義によるが，一般に Python では文字列 (string)，str オブジェクトで実装される [5]．Python の str オブジェクトは，組み込み関数 str()，またはシングルクォート’，ダブルクォート”，三重引用符”’ で囲むことで実装できる．Python の str オブジェクトは，イミュータブルな配列である．また，タプルオブジェクトと同様に順番を持つ．例えば，appleを文字列 apple，Appleを文字列 Apple とすると，これらはソースコード 19 で実装され，この 2 つのオブジェクトは異なる．

ソースコード 19　記号列 (string)

```
1    apple = 'apple'
2    Apple = "Apple"
3
4    print(apple == Apple) #False
```

2　集合演算

このセクションでは色々な集合の演算を実装する．

2.1 共通部分集合

教科書 [1] の定義に従うと共通部分集合，または積集合は，次で定義される．

任意の集合 A と B に対し，A と B の共通部分集合 $A \cap B$ を $A \cap B = \{c | c \in A, c \in B\}$ とする．

for文を用いて A と B の全要素について調べて共通部分集合を返すこともできる．しかし，この方法では複数の集合の共通部分集合を求めるのが非常に面倒になる．そこで，演算子&や intersectionメソッドを用いる [5]．集合 N, E, O,$\emptyset$ の共通部分集合は，ソースコード 20 で実装できる．ここで，演算 $\cap$ の交換則と結合則は既知とした．

ソースコード 20　集合 N, E, O,$\emptyset$ の共通部分集合

```
N = {i for i in range(0, 11)}
E = {i for i in range(0, 11, 2)}
O = {i for i in range(1, 11, 2)}

print(N & N) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(N & set()) #set()
print(N & E) #{0, 2, 4, 6, 8, 10}
print(N.intersection(O)) #{1, 3, 5, 7, 9}
print(E & O) #set()
print(N & E & O) #set()
```

2.2 和集合

教科書 [1] の定義に従うと和集合は，次で定義される．

任意の集合 A と B に対し，A と B の和集合 $A \cup B$ を $A \cup B = \{c | c \in A$ または $c \in B\}$ とする．

for文を用いて A と B の全要素について調べて和集合を返すこともできる．しかし，この方法では複数の集合の和集合を求めるのが非常に面倒になる．そこで，演算子|や unionメソッドを用いる [5]．集合 N, E, O,$\emptyset$ の和集合は，ソースコード 21 で実装できる．ここで，演算 $\cup$ の交換則と結合則は既知とした．

ソースコード 21　集合 N, E, O,$\emptyset$ の和集合

```
N = {i for i in range(0, 11)}
E = {i for i in range(0, 11, 2)}
O = {i for i in range(1, 11, 2)}

print(N | N) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(N | set()) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(N | E) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(N.union(O)) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(E | O) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(N | E | O) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

2.3 差集合

任意の集合 A と B に対し，A と B の差集合 $A \setminus B$ を $A \setminus B = \{c | c \in A, c \notin B\}$ とする．

for文を用いて A と B の全要素について調べて差集合を返すこともできる．しかし，この方法では複数の集合の差集合を求めるのが非常に面倒になる．そこで，演算子-や differenceメソッドを用いる [5]．集合 N, E, O,$\emptyset$ の差集合は，ソースコード 22 で実装できる．演算 $\setminus$ は，$\cap$ や $\cup$ と違い交換則と結合則が成り立たないことに注意する．

ソースコード 22　集合 N, E, O,$\emptyset$ の差集合

```
1    N = {i for i in range(0, 11)}
2    E = {i for i in range(0, 11, 2)}
3    O = {i for i in range(1, 11, 2)}
4
5    print(N - N) #set()
6    print(N - set()) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
7    print(N - E) #{1, 3, 5, 7, 9}
8    print(N.difference(O)) #{0, 2, 4, 6, 8, 10}
9    print(N - E == E - N) #False
10   print(N - E - O) #set()
```

2.4 補集合

教科書 [1] の定義に従うと補集合は，次で定義される．補集合は全体集合と呼ばれる集合の部分集合に対して定義される．

任意の全体集合 S と S の任意の部分集合 A に対し，A の補集合 $\bar{A}$ を $\bar{A} = \{c | c \in S, c \notin A\}$ とする．

補集合そのものを返す関数は，組み込み関数にはない．しかし，補集合は全体集合から対象の集合を引いた集合なので，差集合と同様に実装できる．全体集合を N としたときに，E, O,$\emptyset$ の補集合を E_bar，O_bar，empty_bar とすると，これらはソースコード 23 で実装できる．

ソースコード 23　集合 N に対する E, O,$\emptyset$ の補集合

```
1    N = {i for i in range(0, 11)}
2    E = {i for i in range(0, 11, 2)}
3    O = {i for i in range(1, 11, 2)}
4
5    E_bar = N - E
6    O_bar = N - O
7    empty_bar = N - set()
8    print(E_bar) #{1, 3, 5, 7, 9}
9    print(O_bar) #{0, 2, 4, 6, 8, 10}
10   print(empty_bar) #{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

2.5 直積

教科書 [1] の定義に従うと直積集合は，次で定義される．

任意の集合 A と B に対し，A と B の直積集合 $A \times B$ を $A \times B = \{(a,b)|a \in A, b \in B\}$ とする．

直積集合は対象の集合が 2 つなので `for`文で書いても極端に面倒になることはない．`for`文を用いた集合 A と B の直積集合 `AxB`は，ソースコード 24 で実装できる．

ソースコード 24　for 文を用いた A と B の直積集合

```
1     def direct_product(A, B):
2         AxB = set()
3         A = list(A)
4         B = list(B)
5         for i in range(len(A)):
6             for j in range(len(B)):
7              AxB.add((A[i], B[j]))
8
9         return AxB
```

しかし，これは面倒であるから標準ライブラリの itertools[7] を用いて実装する．itertools を用いれば，集合 A と B の直積集合 AxB は，ソースコード 25 で実装できる．ただし，itertools.product は，Set ではなく itertools.product 型を返す．

ソースコード 25　itertools を用いた A と B の直積集合

```
1     def direct_product(A, B): #A,B = Set object
2         return {i for i in itertools.product(A, B)}
```

2.6　直和

教科書 [1] の定義に従うと直和集合は，次で定義される[*2]．

任意の集合 A と B に対し，A と B の直和集合 $A + B$ を $A + B = \{(a,0)|a \in A\} \cup \{(b,1)|b \in B\}$ と定義する．

Python には直和を返す関数が用意されていないから `for`文を用いて実装する．`for`文を用いた集合 A と B の直和集合 `ApB`は，ソースコード 26 で実装できる．

ソースコード 26　for 文を用いた A と B の直和集合

```
1     def direct_sum(A, B): #A,B = Set object
2         return {(list(A)[i], 0) for i in range(len(A))} | {(list(B)[i], 1) for i in range(
              len(B))}
```

一般に A と B の直和集合 $A + B$ と $B + A$ は異なる．これは，各集合から生成される組の添字が異なるからである．

2.7　べき

教科書 [1] の定義に従うとべき集合は，次で定義される．

任意の集合 A に対し，A のべき集合 2^A を $2^A = \{B|B \subseteq A\}$ とする．

[*2] 直和集合の定義だけ「と定義する」で終わってるのはなんでだろう

Python にはべき集合を返す関数が用意されていないから `for`文と itertools を用いて実装する．`for`文と itertools を用いた集合 A のべき集合 2_A は，ソースコード 27 で実装できる [7][8].

ソースコード 27　for 文と itertools を用いた A のべき集合

```
1    def power_set(A): #A = Set object
2    return {frozenset(i) for i in itertools.chain.from_iterable(itertools.combinations(list(
         A), r) for r in range(len(list(A)) + 1))}
```

第 II 部
写像

この部では，Python で写像を実装する．

3　写像

3.1　写像の定義と Python での実装

教科書 [1] の定義に従うと写像は，次で定義される．

> 集合 A から集合 B への写像 f が A の各要素に B のある要素を一意に対応させ
>
> $$f : A \to B$$
>
> と表される．写像 f によって $a \in A$ が $b \in B$ に対応するとき
>
> $$f : a \mapsto b$$
>
> と表す．

このとき，集合 A を定義域 (domain)，集合 B を値域 (codomain) と呼ぶ．また，$f : a \to b$ は，$b = f(a)$ とも表され，これを f による要素 a の像 (image) と呼ぶ．

写像 f は，定義域 A と値域 B を Set とすると，ソースコード 28 で実装できる．

ソースコード 28　写像

```
1    def f(A):
2        ...
3    B = set(map(f, A)) | B
```

`set(map(f, A))`は $\boldsymbol{range}(f)$ を表し，一般に $\boldsymbol{range}(f) \subseteq \boldsymbol{codom}(f)$ であるが，形式的に `B = set(map(f, A))| B`と書いた．写像 f が部分関数であるときは，未定義な定義域の要素が引数で `(None)`を返して，実質的に f を全域関数として考える．

3.2　写像の同一性

教科書 [1] の定義に従うと写像の同一性は，次で定義される．

> 任意の集合 f と g は定義域と値域がそれぞれ等しく，かつ任意の要素の f による像と g による像が等しいとき等しい (equal) と呼ばれ $f = g$ と表す．

写像 f と g の定義域を A_f，A_g，値域を B_f，B_g とする．このとき，値域はソースコード 28 と書けるから，写像 f と g が同一であるかを確認するには，ソースコード 29 のように実装すればよい．写像 f と g が同一であるとき True を返す．

ソースコード 29　写像の同一性

```
    def image(A_f, B_f):
        if A_f != B_f:
            return False
        check = False
        for x in A_f:
            if f(x) != g(x):
                check = True
        if check:
            return False
        else:
            return True
    print(A_f == B_f and (set(map(lambda x: f, A_f)) | B_f) == (set(map(lambda x: g, B_f))
         | B_g) and image(A_f, B_f))
```

3.3　写像の集合

教科書 [1] の定義に従うと写像の集合は，次で定義される．

> 任意の集合 A と B に対し，
> $$B^A = \{f | f : A \to B\}$$
> とする．

これは，写像 f_i を defで定義し，関数 f_iの集合 F を考えれば写像の集合を定義できる (厳密には関数型 (class 'function') の集合)．つまり，ソースコード 30 のように実装すればよい．

ソースコード 30　写像の集合

```
    def f_1(x):
        ...
    def f_2(x):
        ...
    ...
    def f_n(x):
        ...

    B_A = set([f_1, f_2, ..., f_n])
```

例 2.7 は，ソースコード 31 で実装できる．

ソースコード 31　写像の集合 (例 2.7)

```
    def f_0(i):
        return 2
    def f_1(i):
        return i + 1
```

```
    def f_2(i):
        return 4 - i
    def f_3(i):
        return 3

    B_A = set([f_0, f_1, f_2, f_3])
```

4 写像の合成

教科書 [1] の定義に従うと写像の集合は，次で定義される．

> 任意の写像 $f : A \to B$ と任意の写像 $g : B \to C$ に対し，$a \in A$ を $g(f(a)) \in C$ に対応させる写像を f と g の合成写像とし，$g \circ f$ と表す．$g \circ f : A \to C$ であり，$g \circ f : a \to g(f(a))$ である．

f と g の定義域を A_f，B_f，$g \circ f : a \to g(f(a))$ の値域を C とすれば，写像 f_i を `def`で定義し，`lambda`を用いれば実装できる．つまり，ソースコード 32 のように実装すればよい [10]．

ソースコード 32　写像の合成

```
    def f(x):
        ...
    def g(x):
        ...
    C = set(map(lambda x: g(x), set(map(lambda x: f(x), A_f)) | B_f)) | C
```

例 2.8 は，ソースコード 33 で実装できる．

ソースコード 33　写像の合成 (例 2.8)

```
    def f(n):
        return 2 * n + 1

    def g(n):
        return 2 * n

    g_circ_f = lambda n: g(f(n))
    C = set(map(lambda n: f_1(f_0(n)), N))
```

5 様々な写像

このセクションでは，写像の性質について Python で実装する．

5.1 単射，全射，全単射

教科書 [1] の定義に従うと単射は，次で定義される．

> 任意の写像 f は，任意の a，$a' \in \boldsymbol{dom}(f)$ に対し，$a \neq a'$ であるならば $f(a) \neq f(a')$ であるとき，単射と呼ばれる．

単射の確認は，写像 f が全域関数とすれば (全域関数でないなら定義されない要素の返り値を適当に定義して全域関数とする)，定義域の大きさと値域の大きさを比較すれば実装できる．つまり，ソースコード 34 のようにすればよい．

ソースコード 34　単射の確認

```
    def f(A):
        ...
    B = set(map(f, A)) | B
    print(len(A) == len(set(map(f, A))))
```

これが True なら写像 f は単射である．

例えば，例 2.7 の f_1 は $A = 1, 2$ とすれば単射であるが，f_0 は単射でない．しかし，$A = 1$ とすれば 2 つとも単射になる．これは，ソースコード 35 で確認できる．

ソースコード 35　単射の確認 (例 2.7)

```
    def f_0(i):
        return 2
    def f_1(i):
        return i + 1

    A = {1,2}
    print(len(A) == len(set(map(f_0, A)))) #False
    print(len(A) == len(set(map(f_1, A)))) #True

    A = {1}
    print(len(A) == len(set(map(f_0, A)))) #True
    print(len(A) == len(set(map(f_1, A)))) #True
```

また，単射の保存 (定理 2.3) と単射の性質 (定理 2.4) を，式 5 のときに確認するには，ソースコード 36 のように実装すればよい．

$$\boldsymbol{dom}(f) = \{1, 2\} \tag{1}$$
$$\boldsymbol{dom}(g) = \{2, 3, 4\} \tag{2}$$
$$\boldsymbol{codom}(g \circ f) = \{0, 1\} \tag{3}$$
$$f : x \mapsto x + 1 \tag{4}$$
$$g : x \mapsto x \text{ を } 2 \text{ で割った余り} \tag{5}$$

ソースコード 36　式 5 と定理 2.3，2.4

```
    A = {1, 2}
    B = {2, 3, 4}

    def f(x):
        return x + 1
    def g(x):
        return x % 2

        print(len(A) == len(set(map(f, A)))) #True
        print(len(B) == len(set(map(g, B)))) #False
        print(len(A) == len(set(map(g, set(map(f, A)))))) #True
```

実行結果から写像 $f : A \to B$ は単射であるが，写像 $g : B \to C$ は単射ではない．また，合成写像 $g \circ f : A \to C$ が単射なので，写像 f は単射である．

教科書 [1] の定義に従うと全射は，次で定義される．

> 任意の写像 f は，任意の $b \in \boldsymbol{codom}(f)$ に対し，$b = f(a)$ を満たす $a \in \boldsymbol{dom}(f)$ が存在するとき，全射と呼ばれる．

写像 f が全射であること $\iff \boldsymbol{range}(f) = \boldsymbol{codom}(f)$ なので，写像 f が全射であるかの確認は，f の像の大きさと値域の大きさを比較すれば実装できる．つまり，ソースコード 37 のようにすればよい．

ソースコード 37　全射の確認

```
    def f(A):
        ...
    B = set(map(f, A)) | B
    print(B == set(map(f, A)))
```

これが True なら写像 f は全射である．

また，全射の保存 (定理 2.9) と全射の性質 (定理 2.10) を，式 5 のときに確認するには，ソースコード 38 のように実装すればよい．ただし，$g \circ f$ の値域を $C = \{0, 1\}$ とした．

ソースコード 38　式 5 と定理 2.9，2.10

```
    A = {1, 2}
    B = {2, 3, 4}
    C = {0, 1}

    def f(x):
        return x + 1
    def g(x):
        return x % 2

    print(B == set(map(f, A))) #False
    print(C == set(map(g, B))) #True

    C = set(map(lambda x: g(x), set(map(lambda x: f(x), A)) | B)) | C
    print(C == set(map(lambda x: g(x), set(map(lambda x: f(x), A)) | B)))
```

実行結果から写像 $f : A \to B$ は全射でないが，写像 $g : B \to C$ は全射である．また，合成写像 $g \circ f : A \to C$ が全射なので，写像 g は全射である．

最後に，全単射を実装する．教科書 [1] の定義に従うと全単射は，次で定義される．

> 任意の写像 f は，単射であり，かつ全射であるとき，全単射と呼ばれる．

単射の確認 34 と全射の確認 37 の式を単純に andで繋げればよいので，ソースコード 39 のように実装できる．

ソースコード 39　全単射の確認

```
    def f(A):
        ...
    B = set(map(f, A)) | B
```

```
4     print(len(A) == len(set(map(f, A))) and B == set(map(f, A)))
```

これが True なら写像 f は全単射である．

系 2.1 と系 2.2 はソースコード 36 と 38 を組合せれば確認できる．

5.2 恒等写像と逆写像

教科書 [1] の定義に従うと恒等写像は，次で定義される．

> 任意の集合 A に対し，A 上の写像
> $$i_A : a \mapsto a$$
> は恒等写像と呼ばれる．

恒等写像は，ソースコード 40 で実装できる．

ソースコード 40　恒等写像

```
1     def identify_mapping(x):
2         return x
```

教科書 [1] の定義に従うと逆写像は，次で定義される．

> 任意の集合 A から任意の集合 B への全単射
> $$f : a \mapsto f(a)$$
> に対して，B から A への写像
> $$f^{-1} : f(a) \mapsto a$$
> を f の逆写像と呼ぶ．

写像 f の逆写像 f^{-1} を実装するには，f が全単射であることが保証されているので，(ハッシュ化可能な型に変換した)$f(a)$ を key に，value に引数 a を持つ辞書 [5] を実装すればよく，ソースコード 41 のように実装する．[*3][*4]

ソースコード 41　逆写像

```
1     def f(x):
2         ...
3     def f_inverse(f, s, x):
4         d = dict()
5         for i in range(len(A)):
6             a = A.pop()
7             d[f(a)] = a #f(a) is required hashable
8         return d[x]
```

教科書の例 2.14 を実装するとソースコード 42 になる．

[*3] a と $f(a)$ のタプルを要素に持つリスト型ではなく辞書型で実装したのは，一般に長さ n リスト型に対する所属検査演算 in の計算量は $\mathcal{O}(n)$ で要素数に依存するが，辞書型は key から value の参照が $\mathcal{O}(1)$ で要素数に依存しないからである．

[*4] この実装方法は，***graph***(f) の実装に近い．

ソースコード 42　逆写像例 2.14

```
    def f(z):
        return z + 1

    f_inverse = dict()
    for i in range(len(A)):
        a = A.pop()
        f_inverse[f(a)] = a

    print(f_inverse[2]) #1
```

5.3　置換

教科書 [1] の定義に従うと置換は，次で定義される．

> 任意の有限集合上の全単射は置換と呼ばれる．

つまり置換は任意の有限集合の要素を「並び換える」ような写像である．この「並び換える」は，集合が要素を持つということではなく，定義域と値域を並べて書くと並び換えたように見えると意味である．任意の置換は，互換を繰り返して表現できるから互換を実装する．単純に 2 つの要素を交換する実装は，ソースコード 43 となる．

ソースコード 43　互換

```
    def transposition(s, x, y):
        res = set()
        for i in s:
            if i == x:
                res.add(y)
            elif i == y:
                res.add(x)
            else:
                res.add(i)
        return res
```

しかし，Set は順序を持たないから互換したことが分かりにくい．そこで，Set ではなくリスト型を用いて順序があるように実装すればソースコード 44 になる．

ソースコード 44　リスト型を用いた互換

```
    def transposition_list(l, x, y):
        res = [] #ここが set()じゃない
        for i in l:
            if i == x:
                res.append(y)
            elif i == y:
                res.append(x)
            else:
                res.append(i)
        return res
```

ソースコード 43 とソースコード 44 の違いを確認する．$S = \{1, 2, 3, 4\}$ とすれば，これらの違いはソースコード 45 となる．

ソースコード 45　ソースコード 43 とソースコード 44 の違い

```
1    S = {i for i in range(1, 5)}
2    print(transposition(S, 1, 4)) #{1, 2, 3, 4}
3    print(transposition_list(S, 1, 4)) #[4, 2, 3, 1]
```

恒等写像は，自分自身との交換 `def transposition(s, 1, 1)`とすればよい．

6　写像と集合

6.1　同型

教科書 [1] の定義に従うと同型は，次で定義される．

> 任意の集合 A と B に対し，A から B への全単射が存在するとき，A と B を同型であると呼び $A \cong B$ と表す．

これは，ソースコード 39 で確認できる．しかし，無限集合を実装できないので自明な結果しか返さない (有限集合は要素数が同じならすべて同型)．

第 III 部
tkinter を用いて写像を可視化しよう

7　モチベーション

教科書の 1，2 章に登場する概念を Python で実装したが，それらを理解するのに教科書の図がたいへん役立った．そこで，これらの概念を可視化できるプログラムを作れば，より理解が深まるのではないかと考えた．具体的に，単一の集合，写像，合成写像を表示するプログラム (Visualizer) を作った．

8　Visualizer の概要

集合を写像を表示するには，楕円や矢印を描く必要がある．そこで，今回は Python の tkinter ライブラリを用いて実装した．tkinter のバージョンは，表 1 を参照せよ．

今回作ったプログラムは，Visualizer クラスで，メソッドとして ensemble[*5]，mapping，chain_mapping を持つ．これらのメソッドがそれぞれ集合，写像，合成写像の描画を実行する．具体的な実装は，ソースコード 51 を参照せよ．簡単に説明すると，Visualizer のインスタンスが作成される [9](`Visualizer()`を実行する) と，8 行目の `self.canvas = tk.Canvas(self.root, bg = 'white')`によって空のキャンバス (図を描画するオブジェクト) が生成される．これに各メソッドを用いて描画するのである．写像を結ぶ矢印は，マウスポインタが上にあると色が変わって強調する設定にした．

[*5] ensemble はフランス語で集合の意

9　生成されたもの

まず，集合 $\mathbb{N}_8$ を描画する．$\mathbb{N}_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$ であるから図 1 になる．

ソースコード 46　$\mathbb{N}_8$

```
1    N_8 = {i for i in range(8)}
2
3    v = Visualizer()
4    v.ensemble(N_8, set_name = 'N_8')
```

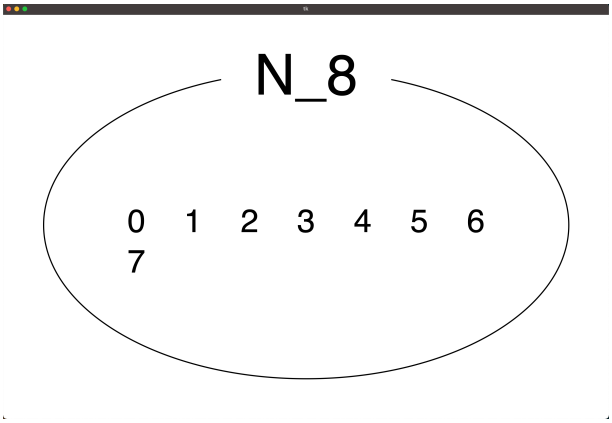

図 1　ソースコード 46 の実行結果

次に，集合 $A = \{1, 2, 3, 4\}$ から集合 $B = \{0, 1, 4, 8, 9, 16\}$ への写像 $f : n \mapsto n^2$ を描画する．写像は，引数を key に，その返り値を value に持つ dict で与える必要がある．また，$x \in \boldsymbol{codom}(f) \wedge x \notin \boldsymbol{range}(f)$ となる要素は，'empty'を key にして与える．これを実装するとソースコード 47 で，図 2 になる．

ソースコード 47　$f : A \to B$

```
1    f = {1:(1, ), 2:(4, ), 3:(9, ), 4:(16, ), 'empty':(0, 1, 4, 8, 16)}
2
3    v = Visualizer()
4    v.mapping(f)
```

最後に，集合 $A = \{1, 2, 3, 4\}$ から集合 B への写像 $f : n \mapsto n\%2$[*6]，集合 $B = \{0, 1, 5\}$ から集合 C への写像 $g : n \mapsto n\%4$ の合成写像 $g \circ f$ を描画する．これは，ソースコード 48 で，図 3 になる．

ソースコード 48　$g \circ f : A \to C$

```
1    f = {1:(1, ), 2:(0, ), 3:(1, ), 4:(0, )}
2    g = {0:(0, ), 1:(1, ), 5:(1, )}
3
4    v = Visualizer()
5    v.chain_mapping(f, g, dom_name = 'A',dom_name_f = 'B',codom_name = 'C')
```

ソースコード 51 は，本レポートではしっかり扱っていないが，関係も実装できる．例えば，関係 f は，ソースコード 49 で，図 4 になる．

[*6] `n % 2`は n を 2 で割った余りを表す

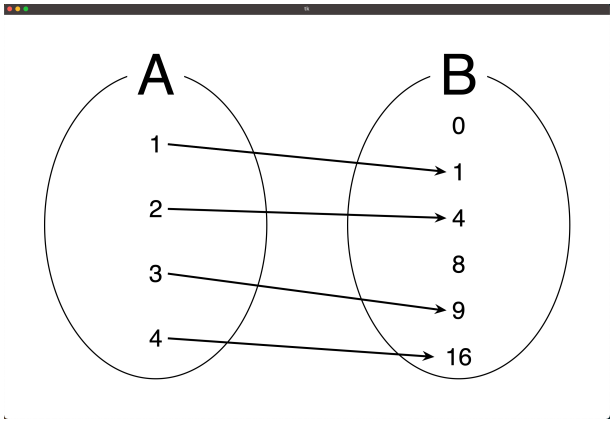

図 2　ソースコード 47 の実行結果

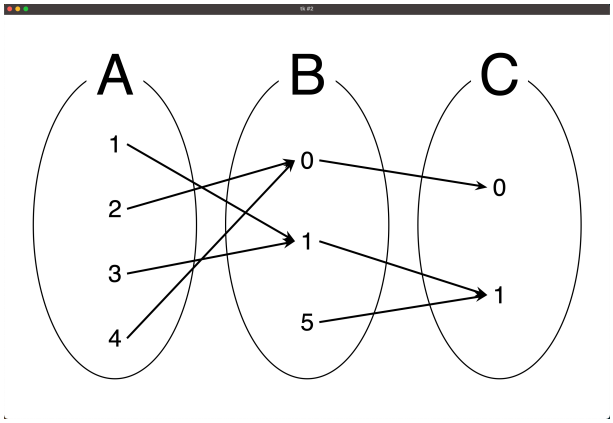

図 3　ソースコード 48 の実行結果

ソースコード 49　関係の実装

```
1    f = {1:(1, 3), 2:(4, 2), 3:(9, 6), 4:(16, )}
2
3    v = Visualizer()
4    v.mapping(f)
```

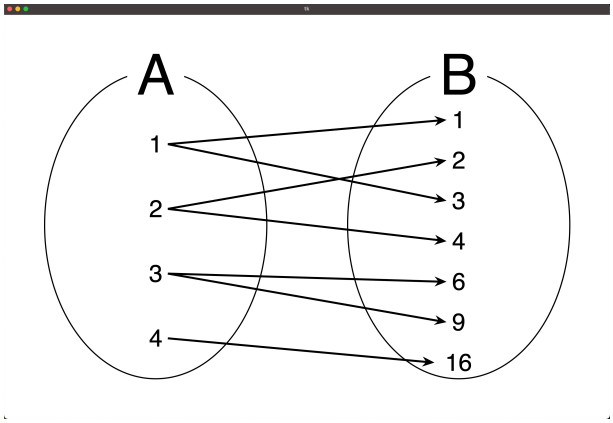

図 4　ソースコード 49 の実行結果

関係の合成も同様である．例えば，ソースコード 50 は，図 5 になる．

ソースコード 50　関係の合成

```
1    f = {1:(1, 3), 2:(4, 2), 3:(9, 6), 4:(16, )}
2    g = {1:(5, 6), 2:(7, ), 3:(1, ), 8:(0, 3), 'empty':(0, 8)}
3
4    v = Visualizer()
5    v.chain_mapping(f, g)
```

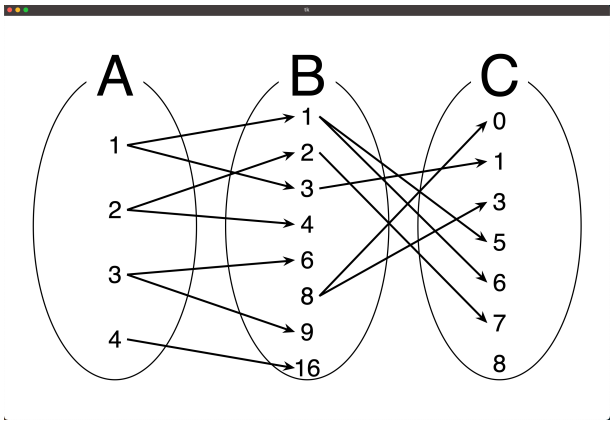

図 5　ソースコード 50 の実行結果

第 IV 部
アピール

slack に関する情報を下に示した．

- ユーザー名：tax_free
- 秘密の記号列[*7]：d41d8cd98f00b204e9800998ecf8427e

授業貢献度の自己評価とその根拠．

- 自己評価：SS
- 自己評価の根拠
  - 複数回チャンネルへ質問を投稿した．
  - 質問への返信と質問でない返信を合せて 25 回した．
  - スタンプや返信を通して授業を盛り上げた．

ソースコード 51　作成した Visualizer

```
1    import tkinter as tk
2
3    class Visualizer:
```

[*7] 秘密の記号列の計算に hash 関数を使ったが，hash 関数に関するレポートでもよかったかも

```
4         def __init__(self):
5             self.root = tk.Tk()
6             self.root.geometry("1500x1000")
7
8             self.canvas = tk.Canvas(self.root, bg = 'white')
9             self.canvas.pack(fill = tk.BOTH, expand = True)
10
11        def ensemble(self, s, set_name = 'S'):
12            self.canvas.create_oval(100, 140, 1400, 900, width = 3)
13            self.canvas.create_rectangle(750 - 70 * len(set_name), 80, 750 + 70 * len(
                  set_name), 220, fill = 'white', outline = 'white')
14            self.canvas.create_text(750, 150, text = set_name, font = ('Helvetica',140))
15
16            step = 140
17            start_position_x = 750 - step / 2 * (min(7, len(s)) - 1)
18            start_position_y = 510 - (len(s) // 7 - 1) * 60
19            size = 80
20            cnt = 0
21            hei = 0
22            for i in s:
23                if cnt % 7 == 0:
24                    hei += 1
25                    cnt = 0
26                self.canvas.create_text(start_position_x + step * cnt, start_position_y +
                      100 * (hei - 1), text = i, font = ("Helvetica", size))
27                cnt += 1
28
29            self.root.mainloop()
30
31
32        def mapping(self, f, dom_name = 'A', codom_name = 'B'):
33            self.canvas.create_oval(100, 140, 650, 900, width = 3)
34            self.canvas.create_oval(850, 140, 1400, 900, width = 3)
35            self.canvas.create_rectangle(375 - 70 * len(dom_name), 80, 375 + 70 * len(
                  dom_name), 220, fill = 'white', outline = 'white')
36            self.canvas.create_rectangle(1125 - 70 * len(dom_name), 80, 1125 + 70 * len(
                  dom_name), 220, fill = 'white', outline = 'white')
37            self.canvas.create_text(375, 150, text = dom_name, font = ('Helvetica',140))
38            self.canvas.create_text(1125, 150, text = codom_name, font = ('Helvetica',140))
39
40            if 'empty' in f:
41                empty = f['empty']
42                s = set(f['empty'])
43                f.pop('empty')
44            else:
45                s = set()
46
47            for i in f.values():
```

```
48                for j in i:
49                    s.add(j)
50            dom_size = 800 / (len(list(f.keys())) + 1)
51            codom_size = 800 / (len(s) + 1)
52
53            dom_position = dict()
54            codom_positon = dict()
55
56            cnt = 1
57            for i in f.keys():
58                dom_position[i] = 160 + dom_size * cnt
59                cnt += 1
60            cnt = 1
61            for i in s:
62                codom_positon[i] = 160 + codom_size * cnt
63                cnt += 1
64
65            size = 60
66            check = set()
67            for i in dom_position.keys():
68                self.canvas.create_text(375, dom_position[i], text = i, font = ("Helvetica"
                      , size))
69                for j in f[i]:
70                    self.canvas.create_line(375 + len(str(i)) * size / 2, dom_position[i],
                          1125 - len(str(j)) * size / 2, codom_positon[j],
71                                      arrow = tk.LAST, arrowshape = (20, 30, 10), width = 5,
                                           activefill = "Red")
72                    if j in check:
73                        continue
74                    self.canvas.create_text(1125, codom_positon[j], text = j, font = ("
                          Helvetica",size))
75                    check.add(j)
76
77            for i in s - check:
78                self.canvas.create_text(1125, codom_positon[i], text = i, font = ("
                      Helvetica",size))
79            self.root.mainloop()
80
81
82        def chain_mapping(self, f, g, dom_name = 'A', dom_name_f = 'B', codom_name = 'C'):
83            s = 404
84            m = 72
85            self.canvas.create_oval(m, 140, s + m, 900, width = 3)
86            self.canvas.create_oval(s + 2 * m, 140, 2 * (s + m), 900, width = 3)
87            self.canvas.create_oval(2 * s + 3 * m, 140, 3 * (s + m), 900, width = 3)
88            self.canvas.create_rectangle(s / 2 + m - 70 * len(dom_name), 80, s / 2 + m +
                  70 * len(dom_name), 220, fill = 'white', outline = 'white')
89            self.canvas.create_rectangle(3 / 2 * s + 2 * m - 70 * len(dom_name), 80, 3 / 2
```

```
            * s + 2 * m + 70 * len(dom_name), 220, fill = 'white', outline = 'white')
90          self.canvas.create_rectangle(5 / 2 * s + 3 * m - 70 * len(dom_name), 80, 5 / 2
                * s + 3 * m + 70 * len(dom_name), 220, fill = 'white', outline = 'white')
91          self.canvas.create_text(s / 2 + m, 150, text = dom_name, font = ('Helvetica'
                ,140))
92          self.canvas.create_text(3 / 2 * s + 2 * m, 150, text = dom_name_f, font = ('
                Helvetica',140))
93          self.canvas.create_text(5 / 2 * s + 3 * m, 150, text = codom_name, font = ('
                Helvetica',140))
94
95          if 'empty' in f:
96              codom_f = set(f['empty'])
97              f.pop('empty')
98          else:
99              codom_f = set()
100         if 'empty' in g:
101             codom_g = set(g['empty'])
102             g.pop('empty')
103         else:
104             codom_g = set()
105
106         for i in f.values():
107             for j in i:
108                 codom_f.add(j)
109
110         for i in g.keys():
111             codom_f.add(i)
112
113         for i in g.values():
114             for j in i:
115                 codom_g.add(j)
116
117         dom_size = 800 / (len(list(f.keys())) + 1)
118         codom_size_f = 800 / (len(codom_f) + 1)
119         codom_size_g = 800 / (len(codom_g) + 1)
120
121         dom_position = dict()
122         codom_positon_f = dict()
123         codom_positon_g = dict()
124
125         cnt = 1
126         for i in f.keys():
127             dom_position[i] = 160 + dom_size * cnt
128             cnt += 1
129         cnt = 1
130         for i in codom_f:
131             codom_positon_f[i] = 160 + codom_size_f * cnt
132             cnt += 1
```

```
133             cnt = 1
134             for i in codom_g:
135                 codom_positon_g[i] = 160 + codom_size_g * cnt
136                 cnt += 1
137
138             size = 60
139             check_f = set()
140             for i in dom_position.keys():
141                 self.canvas.create_text(s / 2 + m, dom_position[i], text = i, font = ("
                        Helvetica", size))
142                 for j in f[i]:
143                     self.canvas.create_line(s / 2 + m + len(str(i)) * size / 2,
                            dom_position[i], 3 / 2 * s + 2 * m - len(str(i)) * size / 2,
                            codom_positon_f[j],
144                                     arrow = tk.LAST, arrowshape = (20, 30, 10), width = 5,
                                            activefill = "Red")
145                     if j in check_f:
146                         continue
147                     self.canvas.create_text(3 / 2 * s + 2 * m, codom_positon_f[j], text = j
                            , font = ("Helvetica",size))
148                     check_f.add(j)
149
150             for i in codom_f - check_f:
151                 self.canvas.create_text(3 / 2 * s + 2 * m, codom_positon_f[i], text = i,
                        font = ("Helvetica",size))
152
153             check_g = set()
154             for i in codom_positon_f.keys():
155                 if i not in check_f:
156                     self.canvas.create_text(3 / 2 * s + 2 * m, codom_positon_f[i], text = i
                            , font = ("Helvetica", size))
157                 if i not in g:
158                     continue
159                 for j in g[i]:
160                     self.canvas.create_line(3 / 2 * s + 2 * m + len(str(i)) * size / 2,
                            codom_positon_f[i], 5 / 2 * s + 3 * m - len(str(i)) * size / 2,
                            codom_positon_g[j],
161                                     arrow = tk.LAST, arrowshape = (20, 30, 10), width = 5,
                                            activefill = "Red")
162                     if j in check_g:
163                         continue
164                     self.canvas.create_text(5 / 2 * s + 3 * m, codom_positon_g[j], text = j
                            , font = ("Helvetica",size))
165                     check_g.add(j)
166
167             for i in codom_g - check_g:
168                 self.canvas.create_text(5 / 2 * s + 3 * m, codom_positon_g[i], text = i,
                        font = ("Helvetica",size))
```

```
169
170             self.root.mainloop()
```

参考文献

[1] 佐藤泰介，橋篤司，伊東利哉，上野修一 (2014). ”著情報基礎数学”. オーム社

[2] Python Software Foundation. ”5. データ構造”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/tutorial/datastructures.html

[3] Python Software Foundation. ”用 語 集/hashable”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/glossary.html#:~:text=(%E3%83%8F%E3%83%83%E3%82%B7%E3%83%A5%E5%8F%AF%E8%83%BD)%20%E3%83%8F%E3%83%83%E3%82%B7%E3%83%A5%E5%8F%AF%E8%83%BD%20%E3%81%AA,%E3%82%92%E6%8C%81%E3%81%A4%E5%BF%85%E8%A6%81%E3%81%8C%E3%81%82%E3%82%8A%E3%81%BE%E3%81%99%E3%80%82

[4] Python Software Foundation. ”3. データモデル”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/reference/datamodel.html

[5] Python Software Foundation. ”組み込み型/set”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/library/stdtypes.html#set

[6] Python Software Foundation. ”6. 式 (expression)”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/reference/expressions.html

[7] Python Software Foundation. ”itertools — 効率的なループ実行のためのイテレータ生成関数”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/library/itertools.html

[8] @hrsma2i(株式会社 ZOZO). ”python で、冪集合 powerset の iterator を作る”. Qiita. (2020–7-25). https://qiita.com/hrsma2i/items/d98f9b1fceeb23d67eef

[9] Python Software Foundation. ”9. クラス”. Python documentation. (2020–7-25). https://docs.python.org/ja/3/tutorial/classes.html

[10] @kekeho. ”Python で 2 つ以上の関数合成”. Qiita. (2020–7-25). https://qiita.com/kekeho/items/ab12fc1d5ad730d3aac9

[11] Akira. ”Python/tkinter】線や円などの図形の描画”. イメージングソリューション. (2020–7-25). https://imagingsolution.net/program/python/tkinter/canvas_drawing_lines_circles_shapes/

[12] pytry3g. ”【tkinter】Canvas を使ってみる。”. HatenaBlog. (2020–7-25). https://www.pytry3g.com/entry/2018/02/09/124143