Contents

Chapter 1

Classes

1.1 permute – 置換 (対称) 群

- **Classes**
  - **Permute**
  - **ExPermute**
  - **PermGroup**

1.1.1 Permute – 置換群の元

Initialize (Constructor)

Permute(`value`: ***list/tuple***, `key`: ***list/tuple***) → **Permute**

Permute(`val_key`: ***dict***) → **Permute**

Permute(`value`: ***list/tuple***, `key`: ***int***=**None**) → **Permute**

置換群の元を新しく作成.

インスタンスは“普通の”方法で作成される. すなわち, ある集合の (インデックス付けられた) 全ての元のリストである `key` と, 全ての置換された元のリストである `value` を入力.

普通は, 同じ長さのリスト (またはタプル) である `value` と `key` を入力. または上記の意味での “value” のリストである `values()`, “key” のリストである `keys()` を持つ辞書 `val_key` として入力することができる. また,`key` の入力には簡単な方法がある:

- もし key が $[1,\ 2, \ldots, N]$ なら,`key` を入力する必要がない.
- もし key が $[0,\ 1, \ldots, N-1]$ なら,`key` として 0 を入力.
- もし key が `value` を昇順として整列したリストと等しければ,1 を入力.
- もし key が `value` を降順として整列したリストと等しければ,−1 を入力.

Attributes

key :
`key` を表す.

data :
†`value` のインデックス付きの形式を表す.

Operations

operator	explanation
`A==B`	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
`A*B`	右乗算 (すなわち, 通常の写像の演算 $A \circ B$)
`A/B`	除算 (すなわち,$A \circ B^{-1}$)
`A**B`	べき乗
`A.inverse()`	逆元
`A[c]`	`key` の `c` に対応した `value` の元
`A(lst)`	A で `lst` を置換

Examples

```
>>> p1 = permute.Permute(['b','c','d','a','e'], ['a','b','c','d','e'])
>>> print p1
['a', 'b', 'c', 'd', 'e'] -> ['b', 'c', 'd', 'a', 'e']
>>> p2 = permute.Permute([2, 3, 0, 1, 4], 0)
>>> print p2
[0, 1, 2, 3, 4] -> [2, 3, 0, 1, 4]
>>> p3 = permute.Permute(['c','a','b','e','d'], 1)
>>> print p3
['a', 'b', 'c', 'd', 'e'] -> ['c', 'a', 'b', 'e', 'd']
>>> print p1 * p3
['a', 'b', 'c', 'd', 'e'] -> ['d', 'b', 'c', 'e', 'a']
>>> print p3 * p1
['a', 'b', 'c', 'd', 'e'] -> ['a', 'b', 'e', 'c', 'd']
>>> print p1 ** 4
['a', 'b', 'c', 'd', 'e'] -> ['a', 'b', 'c', 'd', 'e']
>>> p1['d']
'a'
>>> p2([0, 1, 2, 3, 4])
[2, 3, 0, 1, 4]
```

Methods

1.1.1.1 setKey – key を変換

setKey(`self`, `key`: *list/tuple*) → *Permute*

他の key を設定.

`key` は **key** と同じ長さのリストまたはタプルでなければならない.

1.1.1.2 getValue – "value" を得る

getValue(`self`) → *list*

`self` の (`data` でなく)`value` を返す.

1.1.1.3 getGroup – PermGroup を得る

getGroup(`self`) → *PermGroup*

`self` の所属する **PermGroup** を返す.

1.1.1.4 numbering – インデックスを与える

numbering(`self`) → *int*

置換群の `self` に数を定める. (遅いメソッド)

次に示す置換群の次元による帰納的な定義に従って定められる.
$(n-1)$ 次元上の $[\sigma_1,\ \sigma_2, ..., \sigma_{n-2},\ \sigma_{n-1}]$ の番号付けを k とすると, n 次元上の $[\sigma_1,\ \sigma_2, ..., \sigma_{n-2}, \sigma_{n-1}, n]$ の番号付けは k, また n 次元上の $[\sigma_1,\ \sigma_2, ..., \sigma_{n-2},\ n,\ \sigma_{n-1}]$ の番号付けは $k+(n-1)!$, などとなる. (Room of Points And Lines, part 2, section 15, paragraph 2 (Japanese))

1.1.1.5 order – 元の位数

order(`self`) → *int/long*

群の元としての位数を返す.

このメソッドは一般の群のそれよりも早い.

1.1.1.6 ToTranspose – 互換の積として表す

ToTranspose(`self`) → *ExPermute*

`self` を互換の積で表す.

互換 (すなわち二次元巡回) の積とした **ExPermute** の元を返す. これは再帰プログラムであり,**ToCyclic** よりも多くの時間がかかるだろう.

1.1.1.7 ToCyclic – ExPermute の元に対応する

ToCyclic(`self`) → *ExPermute*

巡回表現の積として `self` を表す.

ExPermute の元を返す. † このメソッドは `self` を互いに素な巡回置換に分解する. よってそれぞれの巡回は可換.

1.1.1.8 sgn – 置換記号

sgn(`self`) → *int*

置換群の元の置換符号を返す.

もし `self` が偶置換, すなわち,`self` を偶数個の互換の積として書くことができる場合,1 を返す. さもなければ, すなわち奇置換の場合,−1 を返す.

1.1.1.9 types – 巡回置換の形式

types(`self`) → *list*

それぞれの巡回置換の元の長さによって定義された巡回置換の形式を返す.

1.1.1.10 ToMatrix – 置換行列

ToMatrix(`self`) → Matrix

置換行列を返す.

行と列は key に対応する. もし self G が $G[a] = b$ を満たせば, 行列の $(a,\ b)$ 成分は 1. さもなくば, その元は 0.

Examples

```
>>> p = Permute([2,3,1,5,4])
>>> p.numbering()
28
>>> p.order()
6
>>> p.ToTranspose()
[(4,5)(1,3)(1,2)](5)
>>> p.sgn()
-1
>>> p.ToCyclic()
[(1,2,3)(4,5)](5)
>>> p.types()
'(2,3)type'
>>> print p.ToMatrix()
0 1 0 0 0
0 0 1 0 0
1 0 0 0 0
0 0 0 0 1
0 0 0 1 0
```

1.1.2 ExPermute – 巡回表現としての置換群の元

Initialize (Constructor)

ExPermute(`dim`: *int*, `value`: *list*, `key`: *list*=None) → ExPermute

新しい置換群の元を作成.

インスタンスは“巡回の” 方法で作成される. すなわち, 各タプルが巡回表現を表すタプルのリストである `value` を入力. 例えば, $(\sigma_1,\ \sigma_2,\ \sigma_3, \ldots, \sigma_k)$ は 1 対 1 写像, $\sigma_1 \mapsto \sigma_2,\ \sigma_2 \mapsto \sigma_3, \ldots, \sigma_k \mapsto \sigma_1$.

`dim` は自然数でなければならない, すなわち,int,long または **Integer** のインスタンス. `key` は `dim` と同じ長さのリストであるべきである. 元が `value` としての `key` に入っているタプルのリストを入力. `key` が $[1,\ 2, \ldots, N]$ という形式なら `key` を省略することができることに注意. また,`key` が $[0,\ 1, \ldots, N-1]$ という形式なら `key` として 0 を入力することができる.

Attributes

dim :
`dim` を表す.

key :
`key` を表す.

data :
† インデックスの付いた `value` の形式を表す.

Operations

operator	explanation
`A==B`	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
`A*B`	右乗算 (すなわち, 普通の写像 $A \circ B$)
`A/B`	除算 (すなわち,$A \circ B^{-1}$)
`A**B`	べき乗
`A.inverse()`	逆元
`A[c]`	`key` の `c` に対応する `value` の元
`A(lst)`	`lst` を A に置換する
`str(A)`	単純な表記.**simplify** を用いる.
`repr(A)`	表記

Examples

```
>>> p1 = permute.ExPermute(5, [('a', 'b')], ['a','b','c','d','e'])
>>> print p1
[('a', 'b')] <['a', 'b', 'c', 'd', 'e']>
>>> p2 = permute.ExPermute(5, [(0, 2), (3, 4, 1)], 0)
>>> print p2
[(0, 2), (1, 3, 4)] <[0, 1, 2, 3, 4]>
>>> p3 = permute.ExPermute(5,[('b','c')],['a','b','c','d','e'])
>>> print p1 * p3
[('a', 'b'), ('b', 'c')] <['a', 'b', 'c', 'd', 'e']>
>>> print p3 * p1
[('b', 'c'), ('a', 'b')] <['a', 'b', 'c', 'd', 'e']>
>>> p1['c']
'c'
>>> p2([0, 1, 2, 3, 4])
[2, 4, 0, 1, 3]
```

Methods

1.1.2.1 setKey – key を変換

setKey(`self`, `key`: *list*) → ***ExPermute***

他の key を設定.

`key` は **dim** と同じ長さのリストでなければならない.

1.1.2.2 getValue – "value" を得る

getValue(`self`) → *list*

`self` の (`data` でなく)`value` を返す.

1.1.2.3 getGroup – PermGroup を得る

getGroup(`self`) → ***PermGroup***

`self` が所属する **PermGroup** を返す.

1.1.2.4 order – 元の位数

order(`self`) → *int/long*

群の元としての位数を返す.

このメソッドは一般の群のそれよりも早い.

1.1.2.5 ToNormal – 普通の表現

ToNormal(`self`) → ***Permute***

`self` を **Permute** のインスタンスとして表す.

1.1.2.6 simplify – 単純な値を使用

simplify(`self`) → ***ExPermute***

より単純な巡回表現を返す.

† このメソッドは **ToNormal** と **ToCyclic** を使用.

1.1.2.7 sgn – 置換符号

sgn(`self`) → ***int***

置換群の元の置換符号を返す.

もし `self` が偶置換なら, すなわち,`self` が偶数個の互換の積として書くことができる場合,1 を返す. さもなくば, すなわち奇置換なら,−1 を返す.

Examples

```
>>> p = permute.ExPermute(5, [(1, 2, 3), (4, 5)])
>>> p.order()
6
>>> print p.ToNormal()
[1, 2, 3, 4, 5] -> [2, 3, 1, 5, 4]
>>> p * p
[(1, 2, 3), (4, 5), (1, 2, 3), (4, 5)] <[1, 2, 3, 4, 5]>
>>> (p * p).simplify()
[(1, 3, 2)] <[1, 2, 3, 4, 5]>
```

1.1.3 PermGroup – 置換群

Initialize (Constructor)

PermGroup(key: *int/long*) → PermGroup
PermGroup(key: *list/tuple*) → PermGroup

新しい置換群を作成.

普通は,key としてリストを入力. もしある整数 N を入力したら,key は $[1, 2, \ldots, N]$ として設定される.

Attributes

key :
key を表す.

Operations

operator	explanation
A==B	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
card(A)	grouporder と同じ
str(A)	単純な表記
repr(A)	表記

Examples

```
>>> p1 = permute.PermGroup(['a','b','c','d','e'])
>>> print p1
['a','b','c','d','e']
>>> card(p1)
120L
```

Methods

1.1.3.1 createElement – シードから元を作成

createElement(`self`, `seed`: *list/tuple/dict*) → ***Permute***
createElement(`self`, `seed`: *list*) → ***ExPermute***

`self` の新しい元を作成.

`seed` は **Permute** または **ExPermute** の “value” の形式でなければならない

1.1.3.2 identity – 単位元

identity(`self`) → ***Permute***

普通の表現で `self` の単位元を返す.

巡回表現の場合,**identity_c** を使用.

1.1.3.3 identity_c – 巡回表現の単位元

identity_c(`self`) → ***ExPermute***

巡回表現として置換群の単位元を返す.

普通の表現の場合,**identity** を使用.

1.1.3.4 grouporder – 群の位数

grouporder(`self`) → *int/long*

群としての `self` の位数を計算.

1.1.3.5 randElement – 無作為に元を選ぶ

randElement(`self`) → ***Permute***

普通の表現として無作為に新しい `self` の元を作成.

Examples

```
>>> p = permute.PermGroup(5)
>>> print p.createElement([3, 4, 5, 1, 2])
[1, 2, 3, 4, 5] -> [3, 4, 5, 1, 2]
>>> print p.createElement([(1, 2), (3, 4)])
[(1, 2), (3, 4)] <[1, 2, 3, 4, 5]>
>>> print p.identity()
[1, 2, 3, 4, 5] -> [1, 2, 3, 4, 5]
>>> print p.identity_c()
[] <[1, 2, 3, 4, 5]>
>>> p.grouporder()
120L
>>> print p.randElement()
[1, 2, 3, 4, 5] -> [3, 4, 5, 2, 1]
```

Bibliography