PermPy_3

May 28, 2025

1 Permutazioni (con un tocco di python) - parte III

1.1 Factoradic e permutazioni ordinate

Dopo aver risolto il problema di generare la permutazione corrispondente ad una posizione qualsiasi dell'insieme ordinato, affrontiamo il problea inverso:
data una permutazione su n elementi determiniamone la posizione corrispondente nell'insieme ordinato di tutte le prmutazioni.

Definiamo, quindi, un algoritmo che ci permetta di passare da una permutazione casuale qualsiasi al suo factoradic corrispondente (e quindi alla sua posizione) e viceversa.
Applicando queste *regole* saremo in grado di:

1. Data una *posizione* generare la permutazione corrispondente nell'insieme ordinato delle permutazioni
2. Data una permutazione determinarne la sua posizione nell'insieme ordinato delle permutazioni

In entrambi i casi avremo bisogno della permutazione base (la prima) che è facilmente generabile a partire dal numero di elementi che vogliamo compongano l'insieme:

Per esempio se n = 7 (permutazioni su un insieme di 7 elementi), la permutazione base si definisce con p0 = list(range(n))

1.2 Esempio (data una permutazione ammissibile, determinare la sua posizione)

Dati in **input**: permutazione p (ammissibile)
Una permutazione su n elementi è *ammissibile* se contiene solo gli elementi da 0 a n (escluso) ciascuno uno sola volta
Risultato in **output**: **posizione** k

1.2.1 Algoritmo in pseudo-codice

1. Genera le permutazione base di n elementi: p0 = list(range(n))
2. Inizializza una lista vuota f = []
3. Se p è ammissibile
    1. Per ciascun elemento x di p
        1. Determina la posizione dell'elemento x in p0 (p0.index(x))
        2. Aggiungila a f
        3. Elimina l'elemento x da p0
4. Calcola il valore decimale (k) del factoradic f
5. Restituisci k

1.2.2 Applicazione dell'algoritmo su dati specifici

Eseguiamo le regole sopra definite sulla seguente permutazione:
p = [1,3,0,2]
calcoleremo, quindi la posizione della permutazione (utilizzando il factoradic associato)

1. p0 = [0, 1, 2, 3]
2. f = []
3. p è ammissibile, quindi:
    1. x = 1 (primo elemento di p)
        1. La posizione di x = 1 nella p0 è **1**
        2. f diventa [1]
        3. p0 diventa [0, 2, 3]
    2. x = 3 (secondo elemento di p)
        1. La posizione di x = 3 nella p0 è **2**
        2. f diventa [1, 2]
        3. p0 diventa [0, 2]
    3. x = 0 (terzo elemento di p)
        1. La posizione di x = 0 nella p0 è **0**
        2. f diventa [1, 2, 0]
        3. p0 diventa [2]
    4. x = 2 (ultimo elemento di p)
        1. La posizione di x = 2 nella p0 è **0**
        2. f diventa [1, 2, 0, 0]
        3. p0 diventa []
4. Calcola in k il valore di f (k = 1*3! + 2*2! + 0*1 + 0 = 1*6 + 2*2 = 10)
5. Restituisci k

[8]:
```python
# definizione della funzione python che restituisce il valore di un factoradic
from math import factorial
def val(f):
    k = 0
    n = len(f) - 1
    for x in f:
        k = k + x*factorial(n)
        n = n-1
    return k
```

[9]:
```python
# definizione della funzione python che restituisce il factoradic di una␣
 ↪permutazione p
def getFact(p):
    p0 = list(range(len(p)))
    f = []
    for x in p:
        e = p0.index(x)
        f.append(e)
        p0.pop(e)
    return f
```

[10]:
```
# definizione della funzione python che traduce l'algoritmo sopra descritto
def getPos(p):
    f = getFact(p)
    return val(f)
```

1.2.3 Verifica dell'algoritmo in python

[11]:
```
getPos([6,3,5,2,4,0,1])
```

[11]: 4792

[13]:
```
getPos([1,5,2,3,6,0,4])
```

[13]: 1234

[ ]:
```

```