lezione22

December 11, 2023

1 Fondamenti di Programmazione

Andrea Sterbini

lezione 22 - 11 dicembre 2023

2 RECAP:

- rappresentazione di **espressioni algebriche**
  - calcolo
  - semplificazione
  - parsing

3 Combinazioni e permutazioni

- **combinazioni** di **k** elementi scelti tra **N** valori (CON ripetizioni)
- **permutazioni** di **k** elementi scelti tra **N** valori (SENZA ripetizioni)

3.1 Combinazioni

Per ottenere una delle possibili combinazioni - per k volte - scelgo uno dei N valori e lo aggiungo al risultato

Per ottenerle tutte - inizio con le soluzioni per N=1, ciascuna contenente uno degli N valori - per k-1 volte - aggiungo ciascuno degli N valori a tutte le soluzioni già calcolate

[34]:
```
## Versione iterativa
def combinazioni_iterativa(L : list, k : int) -> list:
    risultato = [ [X] for X in L ]
    for i in range(1,k):  # per k-1 volte
        nuovo_risultato = [ [Y]+precedente  for precedente in risultato
                                            for Y in L ]
        risultato = nuovo_risultato
    return risultato
```

[23]:
```
# esempio
print(combinazioni_iterativa([1, 2, 3], 3))
```

```
[[1, 1, 1], [2, 1, 1], [3, 1, 1], [1, 2, 1], [2, 2, 1], [3, 2, 1], [1, 3, 1],
[2, 3, 1], [3, 3, 1], [1, 1, 2], [2, 1, 2], [3, 1, 2], [1, 2, 2], [2, 2, 2], [3,
2, 2], [1, 3, 2], [2, 3, 2], [3, 3, 2], [1, 1, 3], [2, 1, 3], [3, 1, 3], [1, 2,
3], [2, 2, 3], [3, 2, 3], [1, 3, 3], [2, 3, 3], [3, 3, 3]]
```

Per realizzarlo ricorsivo - usiamo k come indice del problema e lo riduciamo di 1 mano a mano (**riduzione**) - il **caso base** è k==1 - in questo caso ciascuna combinazione contiene uno solo degli N valori - altrimenti basta aggiungere (**composizione**) a tutte le sottosoluzioni ciascuno degli N valori

convergenza: da k>0 a forza di togliere 1 si arriva a 1

[24]:
```
## versione ricorsiva
def combinazioni_ricorsiva(L:list, k:int) -> list:
    if k == 1:
        return [ [X] for X in L ]
    else:
        return [ [Y]+sottosoluzione for sottosoluzione in␣
 ↪combinazioni_ricorsiva(L,k-1)
                                    for Y in L ]
```

[22]:
```
# esempio
print(combinazioni_ricorsiva([1, 2, 3], 3))
```

```
[[1, 1, 1], [2, 1, 1], [3, 1, 1], [1, 2, 1], [2, 2, 1], [3, 2, 1], [1, 3, 1],
[2, 3, 1], [3, 3, 1], [1, 1, 2], [2, 1, 2], [3, 1, 2], [1, 2, 2], [2, 2, 2], [3,
2, 2], [1, 3, 2], [2, 3, 2], [3, 3, 2], [1, 1, 3], [2, 1, 3], [3, 1, 3], [1, 2,
3], [2, 2, 3], [3, 2, 3], [1, 3, 3], [2, 3, 3], [3, 3, 3]]
```

3.2 Permutazioni (k elementi SENZA ripetizioni)

- idea 1: costruisco tutte le combinazioni e TOLGO quelle con ripetizioni
  - inefficiente (meglio NON GENERARE cose che dovremo filtrare)
  - NON FUNZIONA se nella lista degli elementi da permutare ci sono doppioni
- idea 2:
  - come la generazione delle combinazioni ma stando attenti a non riusare lo stesso elemento

3.3 versione ricorsiva

- come prima, riduciamo su k
- ma ogni volta che scegliamo un elemento da aggiungere ad una sottosoluzione NON lo passiamo alla chiamata ricorsiva

[39]:
```
def permutazioni_ricorsiva(L:list, k:int) -> list:
    assert k <= len(L), f"k={k} è maggiore della lunghezza di L ({len(L)})"
    if k==1:
        return [ [X] for X in L]
    else:
        risultato = []
        for i in range(len(L)):
```

```
            resto = L.copy()
            Y = resto.pop(i)
            #Y = L[i]                  # prendo l'elemento Y
            #resto = L[:i]+L[i+1:]     # tolgo Y da L
            for sottosoluzione in permutazioni_ricorsiva(resto, k-1):  # NOTA:␣
 ↪la lista NON contiene l'elemento Y
                risultato.append([Y]+sottosoluzione)
        return risultato
```

[42]:
```
# Esempio
print(permutazioni_ricorsiva([1,2,3,4],2))
```

```
[[1, 2], [1, 3], [1, 4], [2, 1], [2, 3], [2, 4], [3, 1], [3, 2], [3, 4], [4, 1],
[4, 2], [4, 3]]
```

[ ]:

[ ]:

4 Rotazione ricorsiva di immagini

- vediamo che succede se divido una immagine in 4 parti e la ruoto

```
| 1 |
```

resta com'è

```
0 | 1
--|--
2 | 3
```

diventa ==>

```
1 | 3
--|--
0 | 2
```

```
 0 1 | 2 3
 4 5 | 6 7
 ----|----
 8 9 | A B
 C D | E F
```

diventa ==>

```
 3 7 | B F
 2 6 | A E
 ----|----
 1 5 | 9 D
 0 4 | 8 C
```

4.1 Suddivisione e rotazione ricorsiva NxN

- se N==1: la matrice resta così (**caso base**)
- se N>1:
  - divido in 4 sottomatrici (**riduzione**)
  - le ruoto di 90° (**chiamata ricorsiva**)
  - le scambio
  - le fondo in una matrice più grande (**composizione**)
- a forza di dividere per 2 arrivo sempre a 1 (**convergenza**)

[43]:
```
# A B
# C D

def dividiP2(matrice):
    "divido la matrice nei suoi 4 quadranti"
    # NOTA: la matrice deve avere dimensione potenza di 2
    L = len(matrice)
    assert L>1 and L%2==0 # FIXME: dovrei controllare 2^n==L
    metà = L//2
    fascia_superiore = matrice[:metà]
    fascia_inferiore = matrice[metà:]
    A = [ riga[:metà] for riga in fascia_superiore ]
    B = [ riga[metà:] for riga in fascia_superiore ]
    C = [ riga[:metà] for riga in fascia_inferiore ]
    D = [ riga[metà:] for riga in fascia_inferiore ]
    return A, B, C, D
```

[44]:
```
# vediamo se funziona
M = [[1,2,3,4],
     [5,6,7,8],
     [9,10,11,12],
     [13,14,15,16]]
N = [[1,2],[3,4]]
print(*dividiP2(M),sep='\n\n')
```

```
[[1, 2], [5, 6]]

[[3, 4], [7, 8]]

[[9, 10], [13, 14]]

[[11, 12], [15, 16]]
```

[3]:
```
# A B
# C D
def fondiP2(A, B, C, D):
    "fondo 4 matrici in una sola"
    AB = [ rigaA+rigaB for rigaA,rigaB in zip(A,B)  ]
```

```
    CD = [ rigaC+rigaD for rigaC,rigaD in zip(C,D)  ]
    return AB + CD
```

[45]:
```
# test
A = [[1,2],[5,6]]
B = [[3,4],[7,8]]
C = [[9,10],[13,14]]
D = [[11,12],[15,16]]
print(*fondiP2(A,B,C,D), sep='\n')
```

```
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 16]
```

[46]:
```
# A B
# C D

# B D
# A C

def ruotaP2(M):
    "ruoto una matrice che ha lato 2^L"
    # se N==1 la ritorno tal quale
    if len(M) == 1:
        return M
    # la divido in 4
    A, B, C, D = dividiP2(M)
    # le ruoto tutte e 4
    Aruotata = ruotaP2(A)
    Bruotata = ruotaP2(B)
    Cruotata = ruotaP2(C)
    Druotata = ruotaP2(D)
    # le scambio e le fondo
    return fondiP2(Bruotata, Druotata, Aruotata, Cruotata)
```

[47]:
```
M = [['1', '2', '3', '4'],
     ['5', '6', '7', '8'],
     ['9', 'A', 'B', 'C'],
     ['D', 'E', 'F', 'G']]
print(*ruotaP2(M), sep='\n')
```

```
['4', '8', 'C', 'G']
['3', '7', 'B', 'F']
['2', '6', 'A', 'E']
['1', '5', '9', 'D']
```

[51]:
```
def ruota(M):
    "generico ruota applicabile a matrici != da potenza di 2"
    # allargo la matrice alla prossima potenza di 2 nelle due direzioni
    W = len(M[0])
    H = len(M)
    i = 0
    while 2**i < max(W,H):
        i +=1
    larghezza = altezza = 2**i
    # FIXME: lavoriamo su una copia di M invece che su M
    copia = [ riga.copy() for riga in M ]
    for riga in copia:
        riga += [(0,0,0)] * (larghezza-W)
    copia += [ [(0,0,0)] * larghezza for i in range(altezza-H) ]
    # ruoto la matrice
    ruotata = ruotaP2(copia)
    # elimino le righe/colonne aggiunte
    ruotata = ruotata[larghezza-W:]
    return [ riga[:H] for riga in ruotata  ]
```

[52]:
```
#  3 7 B F
#  2 6 A E
#  1 5 9 D
#  0 4 8 C
M = [ ['0', '1', '2', ],
      ['4', '5', '6', ],
      ['8', '9', 'A', ]]

print(*ruota(M), sep='\n')
```

```
['2', '6', 'A']
['1', '5', '9']
['0', '4', '8']
```

[53]:
```
import images

img = images.load('scarabocchio.png')

ruotata = ruota(img)

images.save(ruotata, 'scarabocchio-90°.png')
images.visd(img), images.visd(ruotata)
```

[53]: (None, None)

[55]:
```
# %%%

trecime = images.load('3cime.png')
ruotata = ruota(trecime)
images.save(ruotata, '3cime-ruotata.png')

images.visd(trecime), images.visd(ruotata)
```

[55]: (None, None)

5 Esercizi d'esame