exc2-Copy1

May 7, 2021

1 Laboratorio de Combinatoria

1.0.1 Generación de combinaciones

Combinaciones Recursivo sin detectar final

[1]:
```
def comb_rec(m,n,pre,desde,l):
    if n==0:
        l.append(pre)
    else:
        for a in range(desde,m+1):
            prenuevo=pre+[a]
            comb_rec(m,n-1,prenuevo,a+1,l)
```

[2]:
```
l=[]
comb_rec(5,3,[],1,l)
print(l)
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

Función preámbulo para generar la llamada externa para calcular las combinaciones

[3]:
```
def comb_rec_ext(m,n):
    l=[]
    comb_rec(m,n,[],1,l)
```

[4]:
```
comb_rec_ext(5,3)
print(l)
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

Versión modificada que no genera `prenuevo` al extender `pre`, sino que se extiende `pre` y después de la llamada recursiva se recupera el valor de `pre` antes de extenderlo, simplemente aplicandole `pop` a `pre`

[5]:
```
def comb_rec2(m,n,pre,desde,l):
    if n==0:
```

```
        l.append(pre)   # Cambiar a l.append(pre[:]) si se hace el cambio de␣
↪abajo
    else:
        for a in range(desde,m+1):
            pre=pre+[a]  # Equivalentemente podemos hacer pre.append(a) si␣
↪arriba se
                         # hace el cambio indicado
            comb_rec(m,n-1,pre,a+1,l)
            pre.pop()
```

[6]:
```
l=[]
comb_rec2(5,3,[],1,l)
print(l)
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

Versión cambiada que además detecta el valor máximo de cada elemento de `pre`

[7]:
```
def comb_rec3(m,n,pre,desde,l):
    if n==0:
        l.append(pre[:])
    else:
        for a in range(desde,m-n+2): # Observad que el i-esimo elemento de pre␣
↪se genera
                             # con n igual al valor inicial de n menos (i-1)
                             # su valor máximo es entonces m-n+1
                             # p.e. para m=5 y n=3 el segundo elemento de pre, ␣
↪pre[1]
                             # se genera con n=2 y por tanto su valor máximo es␣
↪5-2+1=4
            pre.append(a)
            comb_rec(m,n-1,pre,a+1,l)
            pre.pop()
```

[8]:
```
l=[]
comb_rec3(5,3,[],1,l)
print(l)
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

Combinaciones Iterativo sin detectar final

[9]:
```
def comb_iter(m,n):
    l=[]
    pre=[1]
    while pre!=[]:
```

```
        if len(pre)==n:
            l.append(pre[:])    # copiamos pre para que luego no se pueda␣
↪modificar el
                                # valor añadido a la lista
            if pre[n-1] < m:
                pre[n-1]+=1
            else:
                pre.pop()
                if  pre!=[]:
                    pre[-1]+=1
        else:
            if pre[-1]==m:
                pre.pop()
                if pre!=[]:
                    pre[-1]+=1
            else:
                pre.append(pre[-1]+1)
    return(l)
```

[10]:
```
print(comb_iter(5,3))
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

Combinaciones Iterativo detectando el final

[11]:
```
def comb_iter2(m,n):
    l=[]
    pre=[1]
    while pre!=[]:
        if len(pre)==n:
            l.append(pre[:])    # copiamos pre para que luego no se pueda␣
↪modificar el
                                # valor añadido a la lista
            if pre[n-1] < m:
                pre[n-1]+=1
            else:
                pre.pop()
                if  pre!=[]:
                    pre[-1]+=1
        else:
            if pre[-1]==m+len(pre)-n+1:  # dejo que el i-esimo elemento de pre␣
↪rebase en una
                            # unidad su valor máximo, pero en cuanto sucede lo␣
↪detecto
                            # y elimino ese elemento p.e. para m=5 y n=3 el␣
↪segundo elemento
```

```
                            #  de pre,  pre[1] dejamos que valga 5+2-3+1=5 y␣
↪entonces lo quito
                pre.pop()
                if pre!=[]:
                    pre[-1]+=1
            else:
                pre.append(pre[-1]+1)
    return(l)
```

[12]:
```
print(comb_iter2(5,3))
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4],
[2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

1.0.2 Generación de variaciones

Variaciones recursivo

[13]:
```
def var_rec(m,n,pre,usados,l):  # usados marca si cada valor i está en pre vía␣
↪usados[i-1]
    if n==0:
        l.append(pre[:])
    else:
        for a in range(1,m+1):
            if not usados[a-1]:
                prenuevo=pre+[a]  # equivalentemente, uso pre en vez de␣
↪prenuevo y añado abajo
                usados[a-1]=True
                var_rec(m,n-1,prenuevo,usados,l)
                usados[a-1]=False  # aquí añado pre.pop()
```

[14]:
```
def var_llama(m,n):
    usados=[False]*m
    l=[]
    pre=[]
    var_rec(m,n,pre,usados,l)
    return(l)
```

[15]:
```
print(var_llama(5,3))
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 2], [1, 3, 4], [1, 3, 5], [1, 4, 2],
[1, 4, 3], [1, 4, 5], [1, 5, 2], [1, 5, 3], [1, 5, 4], [2, 1, 3], [2, 1, 4], [2,
1, 5], [2, 3, 1], [2, 3, 4], [2, 3, 5], [2, 4, 1], [2, 4, 3], [2, 4, 5], [2, 5,
1], [2, 5, 3], [2, 5, 4], [3, 1, 2], [3, 1, 4], [3, 1, 5], [3, 2, 1], [3, 2, 4],
[3, 2, 5], [3, 4, 1], [3, 4, 2], [3, 4, 5], [3, 5, 1], [3, 5, 2], [3, 5, 4], [4,
1, 2], [4, 1, 3], [4, 1, 5], [4, 2, 1], [4, 2, 3], [4, 2, 5], [4, 3, 1], [4, 3,
2], [4, 3, 5], [4, 5, 1], [4, 5, 2], [4, 5, 3], [5, 1, 2], [5, 1, 3], [5, 1, 4],
```

```
[5, 2, 1], [5, 2, 3], [5, 2, 4], [5, 3, 1], [5, 3, 2], [5, 3, 4], [5, 4, 1], [5,
4, 2], [5, 4, 3]]
```

Variaciones iterativo

[16]:
```python
def menor_no_usado(m,usados,j):  # menor elemento mayor que j no usado en usados
                                 # devuelve m+1 si no hay ninguno
    j+=1
    while j <= m and usados[j-1]:
        j+=1
    return j
```

[17]:
```python
def var_iter(m,n):
    usados=[False]*m
    l=[]
    pre=[1]
    usados[0]=True
    while pre!=[]:
        if len(pre)==n:
            l.append(pre[:])
            enc=False
            while not enc and pre!=[]:
                usados[pre[-1]-1]=False
                sig=menor_no_usado(m,usados,pre[-1])
                if sig <=m:
                    enc=True
                    pre[-1]=sig
                    usados[sig-1]=True
                else:
                    pre.pop()
        else:
            sig=menor_no_usado(m,usados,0)
            pre.append(sig)
            usados[sig-1]=True
    return(l)
```

[18]:
```python
print(var_iter(5,3))
```

```
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 2], [1, 3, 4], [1, 3, 5], [1, 4, 2],
[1, 4, 3], [1, 4, 5], [1, 5, 2], [1, 5, 3], [1, 5, 4], [2, 1, 3], [2, 1, 4], [2,
1, 5], [2, 3, 1], [2, 3, 4], [2, 3, 5], [2, 4, 1], [2, 4, 3], [2, 4, 5], [2, 5,
1], [2, 5, 3], [2, 5, 4], [3, 1, 2], [3, 1, 4], [3, 1, 5], [3, 2, 1], [3, 2, 4],
[3, 2, 5], [3, 4, 1], [3, 4, 2], [3, 4, 5], [3, 5, 1], [3, 5, 2], [3, 5, 4], [4,
1, 2], [4, 1, 3], [4, 1, 5], [4, 2, 1], [4, 2, 3], [4, 2, 5], [4, 3, 1], [4, 3,
2], [4, 3, 5], [4, 5, 1], [4, 5, 2], [4, 5, 3], [5, 1, 2], [5, 1, 3], [5, 1, 4],
[5, 2, 1], [5, 2, 3], [5, 2, 4], [5, 3, 1], [5, 3, 2], [5, 3, 4], [5, 4, 1], [5,
4, 2], [5, 4, 3]]
```