PERMUTARI

def bkt(k):  
 global s, n, cnt  
 for v in range(1,n+1):  
 s[k] = v  
 if s[k] not in s[1:k]:  
 if k==n:  
 cnt += 1  
 print(cnt, ".",\*s[1:])  
 else:  
 bkt(k+1)  
  
n=4  
s=[0] \* (n+1)  
cnt=0  
bkt(1)

ARANJAMENTE

def bkt(k):  
 global s, n, cnt, m  
 for v in range(1,n+1):  
 s[k] = v  
 if s[k] not in s[1:k]:  
 if k==m:  
 cnt += 1  
 print(cnt, ".", \*s[1:])  
 else:  
 bkt(k+1)  
  
n=5  
m=3  
s=[0] \* (m+1)  
cnt=0  
bkt(1)

COMBINARI:

def bkt(k):  
 global p, s, n, cnt, m  
 for v in range(s[k-1]+1, n+1):  
 s[k] = v  
 if k==m:  
 cnt += 1  
 print(str(cnt).rjust(3) + ".", \*s[1:])  
 else:  
 bkt(k+1)  
  
n=5  
m=3  
s=[0] \* (m+1)  
cnt=0  
bkt(1)

SUBMULTIMILE DE 3 LITERE ALE CUV „EXAMEN”(trb salvate intr o lista pt ca sunt 2 de e si se repeta)

def bkt(k):  
 global p, s, n, cnt, m  
 for v in range(s[k-1]+1, n+1):  
 s[k] = v  
 if k==m:  
 cnt += 1  
 for i in range (1, k+1):  
 print(sir[s[i]-1], end="")  
 print()  
 else:  
 bkt(k+1)  
  
sir = "examen"  
n=len(sir)  
m=3  
s=[0] \* (m+1)  
cnt=0  
bkt(1)

SCRIEM N CA SUME

def bkt(k):  
 global s,n  
 #s[1]=1, s[2]=1, ..., s[k-1]=1, s[k] <= n - (k-1) = n-k+1  
 for v in range(s[k-1], n-k+2):  
 s[k] = v  
 scrt=sum(s[1:k+1])  
 if scrt <= n:  
 if scrt == n and k >= 2:  
 print(\*s[1:k+1], sep="+")  
 else:  
 bkt(k+1)  
  
n=int(input("n = "))  
s=[0] \* (n+1)  
s[0] = 1  
bkt(1)

Cele n regine care sa nu se atace

(permutari)

(1,2,3,4,5,6,7,8) de ex (nr din tuplu sunt coloanele, iar indecsii sunt liniile)

Se consideră n spectacole pentru care se cunosc intervalele de desfășurare. Să se găsească toate planificările cu număr maxim de spectacole care se pot efectua într-o singură sală astfel încât, în cadrul fiecărei planificări, spectacolele să nu se suprapună.

(NU MERGE DA NUJ DC)

def cmp\_spectacole(sp):  
 return sp[2]  
  
def numarMaximSpectacole(lsp):  
 lsp.sort(key = cmp\_spectacole)  
 ult = "00:00"  
 cnt = 0  
 for sp in lsp:  
 if sp[1] >= ult:  
 cnt += 1  
 ult = sp[2]  
  
 return cnt  
  
def bkt(k):  
 global sol, nms, fout, lsp  
 for v in range(len(lsp)):  
 sol[k] = v  
 if v not in sol[:k] and (k == 0 or lsp[sol[k]][1] >= lsp[sol[k-1]][2]):  
 if k == nms-1:  
 for p in sol:  
 fout.write(lsp[p][1] + "-" + lsp[p][2] + " " + lsp[p][0] + " ")  
  
fin = open("spectacole.txt")  
  
lsp = []  
for linie in fin:  
 aux = linie.split()  
 tsp = aux[0].split("-")  
 lsp.append((" ".join(aux[1:]), tsp[0], tsp[1]))  
  
fin.close()  
  
fout = open("spectacole.txt","w")  
nms = numarMaximSpectacole(lsp)  
  
sol = [0] \* nms  
bkt(0)

Să se afișeze toate numerele naturale având cifre distincte și suma cifrelor egală cu o valoare 𝑐 dată. De exemplu, pentru 𝑐 = 3, trebuie să fie afișate numerele: 102, 12, 120, 201, 21, 210, 3 și 30 (nu neapărat în această ordine).

def bkt(k):  
 global c  
 for i in range(1 if k==1 else 0, 10):  
 s[k] = i  
 sumcif = sum(s[1:k+1])  
 if sumcif <= c and s[k] not in s[1:k]:  
 if sumcif == c:  
 print(\*s[1:k+1], sep='')  
 if 0 not in s[1:k+1]:  
 print(\*s[1:k+1], 0, sep='')  
 else:  
 bkt(k+1)  
  
  
c = 4  
s = [0] \* 10  
bkt(1)

Se dau n perechi (x, y) cu proprietatea că x < y. Se cere lungimea maximă k a unui lanț de perechi de forma (x1, y1),…, (xi, yi), (xi+1, yi+1), …, (xk, yk) astfel încât yi < xi+1.

Exemplu: Pentru n = 6 și lp = [(12, 15), (5, 7), (20, 30), (6, 8), (9, 11), (13, 18)], lungimea maximă k a unui lanț cu proprietatea cerută este egală cu 4, iar un posibil astfel de lanț este (5, 7), (9, 11), (12, 15), (20, 30). Atenție, soluția nu este unică!

lp = [(12, 15), (5, 7), (20, 30), (6, 8), (9, 11), (13, 18)]  
n = 6  
  
lp.sort(key = lambda element: element[1])  
  
Lmax = [1 for i in range(n)]  
  
# pred[i] = pozitia elementului predecesor  
pred = [-1 for i in range(n)]  
  
for i in range(1,n):  
 for j in range(i):  
 if lp[i][0] > lp[j][1] and Lmax[j] +1 > Lmax[i]:  
 Lmax[i] = Lmax[j] + 1  
 pred[i] = j  
  
maxi = Lmax.index(max(Lmax))  
  
print(Lmax,maxi)  
poz = maxi  
sol = []  
while poz != -1:  
 sol.append(lp[poz])  
 poz = pred[poz]  
  
print(sol[::-1])

Partiționarea unei mulțimi în două submulțimi cu sume cât mai apropiate (problemă NP completă)

(PROGR DINAMICA)