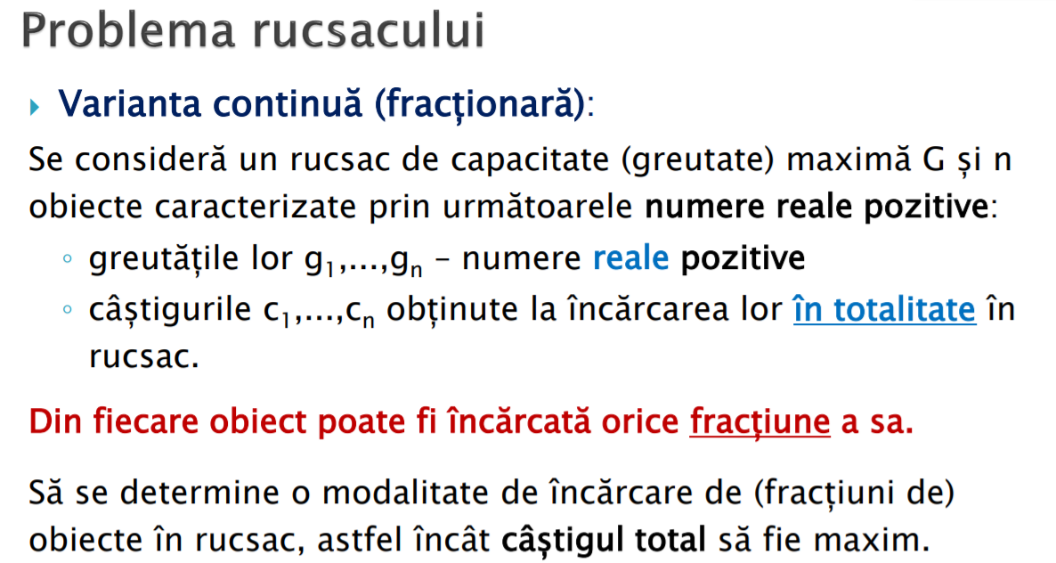
**TEHNICI DE PROGRAMARE**

**1)Greedy**

**a)PROBLEMA RUCSACULUI**

**def** citire\_obiecte(nume\_fisier):  
 fin = open(nume\_fisier)  
 G = float(fin.readline())  
  
 obiecte = []  
 i = 0  
 **for** linie **in** fin:  
 aux = linie.split()  
 obiecte.append((i, float(aux[0]), float(aux[1])))  
 i = i+1  
  
 fin.close()  
  
 **return** G, obiecte  
  
  
**def** cheie\_obiect(ob):  
 **return** ob[2] / ob[1]  
  
  
**def** rucsac\_fractionar(G, obiecte):  
  
 obiecte\_sortate=sorted(obiecte,key=cheie\_obiect, reverse=**True**)  
  
 G\_ramasa=G  
 x = [0 **for** i **in** range(len(obiecte))] *#vectorul cu fractiuni* **for** ob **in** obiecte\_sortate:  
 **if** ob[1] <= G\_ramasa:  
 G\_ramasa -= ob[1]  
 x[ob[0]] = 1 *#obiectul cu indice ob[0] - luat intreg* **else**:  
 x[ob[0]] = G\_ramasa / ob[1]  
 **break  
  
 return** x  
  
**def** afisare(x,obiecte):  
 castig\_total=0  
 **for** i **in** range(len(x)):  
 **if** x[i]>0:  
 print(**f"Obiectul {**i+1**} procent {**x[i]\*100**:.2f}%"**)  
 castig\_total += x[i]\*obiecte[i][2]  
 print(**f"castig total {**castig\_total**}"**)  
  
  
  
G,obiecte=citire\_obiecte(**"rucsac.in"**)  
x=rucsac\_fractionar(G,obiecte)  
print(x)  
afisare(x,obiecte)

**b)PARTITIONAREA INTERVALELOR**

**import** heapq  
**def** citire\_intervale(nume\_fisier):  
 ls = []  
 **with** open(nume\_fisier) **as** f:  
 **for** linie **in** f:  
 s, t = (int(x) **for** x **in** linie.strip(**"[]\n"**).split(**","**))  
 ls.append([s,t]) *#mergea si tuplu* **return** ls  
  
**def** partitionare\_greedy(ls):  
 ls=sorted(ls) *#mergea si ls.sort(), atunci modifica parametrul si se vedea in main modificarea* print(**"intervalele (activitatile) sortate dupa timpul de inceput"**)  
 print(ls)  
 sali = []  
 prima\_activ = ls[0]  
 sali.append([prima\_activ]) *# lista de liste, pe pozitia i =lista cu activitatile din sala i* h = [] *# heap de perechi (timp de terminare, numar sala), cu salile numerotate de la 0* heapq.heappush(h, (prima\_activ[1], 0))  
  
 **for** act\_curenta **in** ls[1:]: *# activitatea curenta* t\_final, nr\_sala = heapq.heappop(h)  
  
 **if** act\_curenta[0] > t\_final: *# se poatea adauga la nr\_sala* sali[nr\_sala].append(act\_curenta)  
 heapq.heappush(h, (act\_curenta[1], nr\_sala)) *# inserez sala nr\_sala cu noul timp de terminare* **else**: *# sala noua pentru activitatea curenta* nr\_sala\_noua = len(sali) *# sala noua are numarul n=len(sali), salile au numar de la 0* sali.append([act\_curenta]) *# sala noua cu activitatea x* heapq.heappush(h, (act\_curenta[1], nr\_sala\_noua)) *# pun sala noua cu timpul de terminar* heapq.heappush(h, (t\_final, nr\_sala)) *# !!pun sala veche inapoi in heap asa cum era, nu am putut adauga activitatea curenta la ea* **return** sali  
  
  
**def** afisare\_programare(sali):  
 **for** i **in** range(len(sali)):  
 print(**f"sala {**i + 1**}: {**sali[i]**}"**)  
  
ls = citire\_intervale(**"intervale.in"**)  
sali=partitionare\_greedy(ls)  
afisare\_programare(sali)

**c)SPECTACOLE**

**def** citire\_intervale(nume\_fisier):  
 ls = []  
 i=0  
 **with** open(nume\_fisier) **as** f:  
 **for** linie **in** f:  
 i=i+1  
 s, t = (int(x) **for** x **in** linie.strip(**"[]\n"**).split(**","**))  
 ls.append((i,[s,t])) *#primul element- indicele activitatii, al doilea -intervalul de desfasurare (putea fi memorat si ca tuplu)* **return** ls  
  
**def** selectie\_activitati(ls):  
 **def** cheie\_activitate(a):  
 **return** a[1][1] *#a[0]=indicele activitatii, a[1]=intervalul de desfasurare, a[1][1]=timpul de terminare* ls.sort(key=cheie\_activitate)  
 print(**f"Activitati ordonate dupa terminare {**ls**}"**)  
 rez=[]  
 t\_ultima\_selectata=0  
 rez.append(ls[0])  
  
 **for** i **in** range(1,len(ls)):  
 act\_curenta=ls[i]  
 **if** act\_curenta[1][0]>t\_ultima\_selectata:  
 t\_ultima\_selectata=act\_curenta[1][1]  
 rez.append(act\_curenta)  
 **return** rez  
  
**def** afiseaza (ls):  
 **for**(i,interval) **in** ls:  
 print(**f"activitatea {**i**}: [{**interval[0]**},{**interval[1]**}]"**)  
  
  
ls = citire\_intervale(**"intervale.in"**)  
print(**f"Activitati initiale {**ls**}"**)  
afiseaza(selectie\_activitati(ls))

**d)INTERCLASARE OPTIMA**

**from** heapq **import** \*  
ls=[20,10,40,50,30,80,45]  
i=0  
h=[]  
**for** x **in** ls:  
 i=i+1  
 heappush(h,(x,str(i)))  
**for** j **in** range(len(ls)-1):  
 s1=heappop(h)  
 s2=heappop(h)  
 print(s1,s2)  
 i = i + 1  
 *#heappush(h,(s1[0]+s2[0],i))* heappush(h,(s1[0]+s2[0],**f"interclasare({**s1[1]**},{**s2[1]**})"**))  
print(h)

**e)PROFIT MAXIM**

**from** heapq **import** heappop,heappush  
ls=[]  
**with** open(**"profit\_maxim.in"**) **as** f:  
 i=0  
 n=int(f.readline())  
 **for** linie **in** f:  
 i=i+1  
 ls.append((i,[int(x) **for** x **in** linie.split()])) *#profit, deadline*print(ls)  
ls.sort(key=**lambda** a:a[1][1],reverse=**True**) *#sortat dupa deadline descrescator*print(ls)  
h=[]  
j=0  
planificare=[**None**]\*n  
**for** i **in** range(n,0,-1):  
 **while** j<len(ls) **and** ls[j][1][1]==i:  
 heappush(h,(-ls[j][1][0],ls[j][0])) *#profit, indice activitate* j+=1  
 print(i)  
 print(h)  
 **if** len(h)>0:  
 planificare[i-1]=heappop(h)[1]  
  
print(planificare)

**f)Cuburi**

Se dau n cuburi cu laturile diferite două câte două. Fiecare cub are o culoare, codificată cu un număr de la 1 la p (p dat). a) Să se construiască un turn de înălțime maximă de cuburi în care un cub nu poate fi aşezat pe un cub de aceeași culoare sau cu latură mai mică decât a sa – O(n logn). Pe prima linie a fişierului de intrare se dau n şi p, iar pe următoarele linii latura şi culoarea fiecărui cub. În fişierul de ieşire se vor afişa înălțimea turnului obținut şi indicele cuburilor folosite (de la bază la vârf)

f=open(**"date.in"**)  
g=open(**"date.out"**,**"w"**)  
s=f.readline()  
n,p=s.split()  
n=int(n)  
p=int(p)  
l=[]  
nr=0  
poz=[]  
**for** i **in** range(n):  
 s = f.readline()  
 a,b=s.split()  
 a=int(a) *#latura* b=int(b) *#culoarea* c=[a, b]  
 poz.append(i) *#pozitiile* l.append(c) *#lista cu [ [l1,c1] , [l2,c2] , .... , [ln,cn] ]*t=sorted(poz,key=**lambda** e: -l[e][0]) *#sortam descrescator dupa latura*print(t) *#afisam lista sortata*h=l[t[0]][0] *#primul cub il alegem*sol=[t[0]+1] *#tatal*us=t[0]  
**for** i **in** range(1,n):  
 k=l[us][1]  
 **if** k!=l[t[i]][1] **and** l[t[i]][0]<l[us][0]:  
 h=h+l[t[i]][0]  
 us=t[i]  
 sol.append(t[i]+1)  
print(h)  
print(sol)  
g.close()  
f.close()

**g)Acoperire**

Avem la dispozitie un interval inchis [A,B] si o multime de alte N intervale inchise [Ai,Bi], 1 ≤ i ≤ N. Scrieti un program care calculeaza si afiseaza numarul minim de intervale inchise din multimea data cu proprietatea ca prin reuniunea acestora se obtine un interval care include pe [A,B].

l=[]  
**with** open(**"acoperire.in"**) **as** f:  
 ls=f.readline()  
 a,b=ls.split()  
 a=int(a)  
 b=int(b)  
 ls=f.readline()  
 m=ls.split()  
 n=int(m[0])  
 ls=f.readline()  
  
 **while**(ls!=**''**):  
 ls1=[]  
 a1,b1=ls.split()  
 ls1.append(int(a1))  
 ls1.append(int(b1))  
 l.append(ls1)  
  
  
 ls=f.readline()  
 f.close()  
l=sorted(l,key=**lambda** t:(t[1],t[0]),reverse=**True**)  
i=0  
a1=l[i][0]  
b1=l[i][1]  
**if**(b1<b):  
 print(-1)  
**else**:  
 cap1=a1  
 cap2=b1  
 nr=1  
 i=1  
 a1=l[i][0]  
 b1=l[i][1]  
  
 **while**(i<len(l)-1 **and** a<cap1):  
  
 **if**(cap1>a1 **and** cap1<=b1<=cap2):  
 nr=nr+1  
 cap1=a1  
 cap2=b1  
 i=i+1  
 a1=l[i][0]  
 b1=l[i][1]  
 **else**:  
 i=i+1  
 a1=l[i][0]  
 b1=l[i][1]  
print(nr)  
print(l)

**h)Memorarea textelor pe banda cu frecventa de acces**

f=open(**"date.in"**)  
n=f.readline()  
n=int(n)  
l=[]  
poz=[]  
**for** i **in** range(n):  
 s=f.readline()  
 x,y=s.split()  
 x=int(x)  
 y=int(y) *#frecventa* ll=[x,y]  
 l.append(ll)   
 poz.append(i) *#pozitia textului*poz.sort(key=**lambda** e:-l[e][1]/l[e][0]) *#descrescator dupa frecventa/text***for** i **in** range(len(poz)):  
 print(poz[i]+1)  
f.close()

**2)Divide et Impera**

**a)Cautare binara**

**def** citire\_vector\_ordonat(nume\_fisier):  
 f=open(nume\_fisier)  
 ls=[int(x) **for** x **in** f.read().split()]  
 **return** ls  
  
  
**def** cautare\_binara(x,ls,p,u):  
 **if** p > u:  
 **return** (**False**,u)  
 **else**:  
 mij = (p + u) // 2  
 **if** x == ls[mij]:  
 **return** (**True**,mij)  
 **elif** x < ls[mij]:  
 **return** cautare\_binara(x,ls, p, mij-1)  
 **else**:  
 **return** cautare\_binara(x,ls, mij+1, u)  
  
**def** cautare\_binara\_nerecursiv(x,ls):  
 p=0  
 u=len(ls)-1  
 **while** p<=u:  
 mij = (p + u) // 2  
 **if** x == ls[mij]:  
 **return** (**True**, mij)  
 **elif** x < ls[mij]:  
 u = mij - 1  
 **else**:  
 p = mij + 1  
 **return** (**False**,u)  
  
  
  
**def** cautare(x,ls):  
 n = len(ls)  
 **return** cautare\_binara(x,ls,0,n-1)  
  
  
**def** cautare(x,ls):  
 n=len(ls)  
 **return** cautare\_binara(x,ls,0,n-1)  
  
ls=citire\_vector\_ordonat(**"vector.in"**)  
x=int(input(**"Dati valoarea cautata "**))  
rez=cautare(x,ls)  
**if** rez[0]:  
 print(**f"{**x**} gasit pe pozitia {**rez[1]**}"**)  
**else**:  
 print(**f"{**x**} nu este in vector {**rez[1]**}"**)  
  
rez=cautare\_binara\_nerecursiv(x,ls)  
**if** rez[0]:  
 print(**f"{**x**} gasit pe pozitia {**rez[1]**}"**)  
**else**:  
 print(**f"{**x**} nu este in vector {**rez[1]**}"**)

**b)Inversiuni-prin interclasare**

**def** citire\_vector(nume\_fisier):  
 f=open(nume\_fisier)  
 ls=[int(x) **for** x **in** f.read().split()]  
 **return** ls  
  
**def** nr\_inversiuni(v, p, u):  
 **if** p==u:  
 **return** 0  
 **else**:  
 m = (p+u)//2  
 n1 = nr\_inversiuni(v, p, m)  
 n2 = nr\_inversiuni(v, m+1, u)  
 **return** n1 + n2 + interclaseaza(v, p, m, u)  
  
  
  
**def** interclaseaza(a, p, m, u):  
 b = [**None**]\*(u-p+1)  
 nr = 0  
 i = p; j = m + 1; k = 0  
 **while** (i<=m) **and** (j <= u):  
 **if** a[i] <= a[j]:  
 b[k] = a[i]; i += 1  
 **else**:  
 b[k] = a[j]; j += 1; nr += (m-i+1)  
 k+=1  
  
 **while** i<=m:  
 b[k] = a[i]; k += 1; i += 1  
  
 **while** j<=u:  
 b[k] = a[j]; k += 1; j += 1  
  
 **for** i **in** range(p,u+1):  
 a[i] = b[i-p]  
  
 **return** nr  
  
  
**def** sortare(v):  
 n=len(v)  
 **return** nr\_inversiuni(v,0,n-1)  
  
**def** nr\_inversiuni\_n2(v):  
 nr=0  
 **for** i **in** range(len(v)):  
 **for** j **in** range(i+1,len(v)):  
 **if** v[i]>v[j]:  
 nr+=1  
 **return** nr  
  
v=citire\_vector(**"interclasare.in"**)  
print(v)  
print(nr\_inversiuni\_n2(v))  
print(sortare(v))  
print(v)

**c)Al k-lea minim dintr-un sir de numere**

**from** random **import** randint  
  
**def** citire\_vector(nume\_fisier):  
 f = open(nume\_fisier)  
 ls = [int(x) **for** x **in** f.read().split()]  
 k = int(input(**"k="**))  
 **return** k, ls  
  
  
**def** poz\_rand(v, p, u):  
 r = randint(p, u)  
 v[r], v[p] = v[p], v[r]  
 **return** poz(v, p, u)  
  
  
**def** poz(v, p, u):  
 i = p;  
 j = u  
 depli = 0;  
 deplj = -1  
 **while** i < j:  
 **if** v[i] > v[j]:  
 v[i], v[j] = v[j], v[i]  
 *# depli, deplj = -deplj, -depli* aux = depli;  
 depli = -deplj;  
 deplj = -aux;  
 i += depli;  
 j += deplj  
  
 **return** i  
  
  
**def** sel\_k\_min(v, k, p, u):  
 m = poz\_rand(v, p, u)  
  
 **if** m == k - 1:  
 **return** v[m]  
 **if** m < k - 1:  
 **return** sel\_k\_min(v, k, m + 1, u)  
 **return** sel\_k\_min(v, k, p, m - 1)  
  
  
**def** sel\_k\_min\_di(v, k):  
 **return** sel\_k\_min(v, k, 0, len(v) - 1)  
  
  
k, v = citire\_vector(**"interclasare.in"**)  
print(k, v)  
print(sel\_k\_min\_di(v, k))

**d)Munte**

*1. Se dă un vector a=(a1,…an) de tip munte (există un indice i astfel încât a1<a2<…<ai > ai+1>…>an;  
ai se numește vârful muntelui). Propuneți un algoritm O(log n) care determină vârful muntelui  
(în calculul complexității algoritmului nu se consideră și citirea vectorului)  
"""***def** varf(a, p, u):  
 **if** u-p<2:  
 **return** max(a[p],a[u])  
  
 mij=(p+u)//2  
 **if** a[mij-1]<a[mij] **and** a[mij]>a[mij+1]:  
 **return** a[mij]  
 **if** a[mij-1]<a[mij] **and** a[mij]<a[mij+1]:  
 **return** varf(a,mij+1,u)  
 **return** varf(a,p,mij-1)  
  
**def** get\_varf(v):  
 **return** varf(v,0,len(v)-1)  
  
ls=[6,4,2]  
print(get\_varf(ls))

**e)Quicksort**

**from** random **import** randint  
  
**def** citire\_vector(nume\_fisier):  
 f = open(nume\_fisier)  
 ls = [int(x) **for** x **in** f.read().split()]  
  
 **return** ls  
  
  
**def** poz\_rand(v, p, u):  
 r = randint(p, u)  
 v[r], v[p] = v[p], v[r]  
 **return** poz(v, p, u)  
  
  
**def** poz(v, p, u):  
 i = p  
 j = u  
 depli = 0  
 deplj = -1  
 **while** i < j:  
 **if** v[i] > v[j]:  
 v[i], v[j] = v[j], v[i]  
 depli, deplj = -deplj, -depli  
 *#aux = depli; depli = -deplj; deplj = -aux* i += depli  
 j += deplj  
  
 **return** i  
  
  
**def** quick\_sort\_di(v, p, u):  
  
 **if** p >= u:  
 **return** m = poz\_rand(v, p, u)  
 *#m = poz(v, p, u)* quick\_sort\_di(v, p, m - 1)  
 quick\_sort\_di(v, m + 1, u)  
  
  
  
**def** quick\_sort(v):  
 **return** quick\_sort\_di(v, 0, len(v) - 1)  
  
  
v = citire\_vector(**"interclasare.in"**)  
quick\_sort(v)  
print(v)

**f)Sortare prin interclasare**

**def** citire\_vector(nume\_fisier):  
 f=open(nume\_fisier)  
 ls=[int(x) **for** x **in** f.read().split()]  
 **return** ls  
  
**def** sort\_interclasare(v, p, u):  
 **if** p==u:  
 **pass  
 else**:  
 m = (p+u)//2  
 sort\_interclasare(v, p, m)  
 sort\_interclasare(v, m+1, u)  
 interclaseaza(v, p, m, u)  
  
  
  
**def** interclaseaza(a, p, m, u):  
 b = [**None**]\*(u-p+1)  
 i = p; j = m + 1; k = 0  
 **while** (i<=m) **and** (j <= u):  
 **if** a[i] <= a[j]:  
 b[k] = a[i]; i += 1  
 **else**:  
 b[k] = a[j]; j+= 1  
 k+=1  
  
 **while** i<=m:  
 b[k] = a[i]; k += 1; i += 1  
  
 **while** j<=u:  
 b[k] = a[j]; k += 1; j += 1  
  
 **for** i **in** range(p,u+1):  
 a[i] = b[i-p]  
  
  
  
  
**def** sortare(v):  
 n=len(v)  
 sort\_interclasare(v,0,n-1)  
  
v=citire\_vector(**"interclasare.in"**)  
sortare(v)  
print(v)

**g)Suma elementelor unei matrice patratice unde n este putere a lui 2**

**def** citire\_matrice(nume\_fisier):  
 f=open(nume\_fisier)  
 m=[]  
 **for** linie **in** f:  
 m.append([int(x) **for** x **in** linie.split()])  
 f.close()  
 **return** m  
  
**def** suma(m,x,y,n):  
 print(x,y)  
 **if** n==1:  
 **return** m[x][y] *#!!da eroare cu n/2, e de tip float* s1 = suma(m, x, y, n//2)  
 s2 = suma(m, x+n//2, y, n//2)  
 s3 = suma(m, x, y+n//2, n//2)  
 s4 = suma(m, x+n//2, y+n//2, n//2)  
 **return** s1+s2+s3+s4  
  
**def** suma\_matrice(m):  
 **return** suma(m,0,0,len(m))  
m=citire\_matrice(**"matrice.in"**)  
s=0  
**for** linie **in** m:  
 s=s+sum(linie)  
print(s)  
print(suma\_matrice(m))

**3)Programare Dinamica**

**a)Monede**

Având la dispoziție un număr nelimitat de monede de valori date {v1,v2,...,vn}, să se determine o modalitate de a plăti o sumă de bani S dată folosind un număr minim de astfel de monede (dacă suma se poate plăti). Exemplu: pentru monede de valori {7, 5, 1} şi S = 11 se vor da 3 monede: două monede de valoare 5, o monedă de valoare 1.O(nS)

v=[1,3,4,5]  
S=7  
n=len(v)  
nr=[S+1 **for** i **in** range(S+1)] *#infinit- S+1*moneda =[0 **for** i **in** range(S+1)]  
nr[0] = 0  
moneda[0] = -1  
**for** s **in** range(1,S+1):  
 nr[s] = S+1 *#infinit* moneda[s] = -1  
 **for** i **in** range(n):  
 **if** (v[i]<= s) **and** nr[s-v[i]] + 1<nr[s]:  
 nr[s] = nr[s-v[i]] + 1  
 moneda[s] = v[i]  
print(**f"Numarul minim de monede: {**nr[S]**}"**)  
**if** nr[S] < S+1:  
 s = S  
 **while** s>0:  
 print(moneda[s],end=**" "**)  
 s = s- moneda[s]  
**else**:  
 print(**"Nu se poate plati suma S"**)

**b)Rucsac**

**def** afis(i, gr):  
 **if** i == 0 **or** gr == 0:  
 **return  
 if** (g[i] <= gr) **and** (s[i][gr] == c[i] + s[i - 1][gr - g[i]]):  
 afis(i-1, gr-g[i])  
 print(i,end=**" "**)  
 **else**:  
 afis(i - 1, gr)  
  
f=open(**"rucsac.in"**)  
g=[0]  
g.extend([int(x) **for** x **in** f.readline().split()])  
c=[0]  
c.extend([int(x) **for** x **in** f.readline().split()])  
G=int(f.readline())  
  
n=len(g)-1  
  
s=[[0 **for** i **in** range(G+1)] **for** j **in** range(n+1)]  
  
  
**for** gr **in** range(G+1):  
 s[0][gr]= 0  
**for** i **in** range(n+1):  
 s[i][0]=0  
  
**for** i **in** range(1,n+1):  
 **for** gr **in** range(1,G+1):  
 **if** g[i] <= gr:  
 **if** c[i]+s[i-1][gr-g[i]] > s[i-1][gr]:  
 s[i][gr]=c[i]+s[i-1][gr-g[i]]  
 **else**:  
 s[i][gr]=s[i-1][gr]  
 **else**:  
 s[i][gr]=s[i-1][gr]  
print(**f"Castigul total {**s[n][G]**}"**)  
  
  
print(**"obiectele incarcate determinate folosind relatia de recurenta - varianta iterativa"**)  
gr=G  
i=n  
**while** gr > 0 **and** i > 0:  
 **if** (g[i] <= G) **and** (s[i][gr] == c[i] + s[i - 1][gr - g[i]]):  
 print(i, end=**" "**)  
 gr = gr - g[i]  
 i-=1  
  
print()  
print(**"obiectele incarcate determinate folosind relatia de recurenta - varianta recursiva"**)  
afis(n, G)  
print(s)

**c) Subsir comun de la 1**

**def** afis(i, j):  
 **if** i == 0 **or** j == 0:  
 **return  
 if** a[i] == b[j]:  
 afis(i - 1, j - 1)  
 print(a[i], end=**" "**)  
 **else**:  
 **if** s[i - 1][j] > s[i][j - 1]:  
 afis(i - 1, j)  
 **else**:  
 afis(i, j - 1)  
f=open(**"subsircomun.in"**)  
a=[0]  
a.extend([int(x) **for** x **in** f.readline().split()])  
b=[0]  
b.extend([int(x) **for** x **in** f.readline().split()])  
  
m=len(a)-1  
n=len(b)-1  
s=[[0 **for** j **in** range(n+1) ] **for** i **in** range(m+1)]  
**for** j **in** range(n+1):  
 s[0][j]=0  
**for** i **in** range(m+1):  
 s[i][0]=0  
**for** i **in** range(1,m+1):  
 **for** j **in** range(1, n + 1):  
 **if** a[i]==b[j]:  
 s[i][j]=s[i-1][j-1]+1  
 **else**:  
 s[i][j]=max(s[i-1][j],s[i][j-1])  
print(s[m][n])  
  
afis(m,n)  
  
*#print()  
#print(s)*

**d)Subsir comun**

**def** afis(i, j):  
 **if** i == 0 **or** j == 0:  
 **return  
 if** a[i - 1] == b[j - 1]:  
 afis(i - 1, j - 1)  
 print(a[i - 1], end=**" "**)  
 **else**:  
 **if** s[i - 1][j] > s[i][j - 1]:  
 afis(i - 1, j)  
 **else**:  
 afis(i, j - 1)  
f=open(**"subsircomun.in"**)  
a=[int(x) **for** x **in** f.readline().split()]  
b=[int(x) **for** x **in** f.readline().split()]  
  
m=len(a)  
n=len(b)  
s=[[0 **for** j **in** range(n+1) ] **for** i **in** range(m+1)]  
**for** j **in** range(n+1):  
 s[0][j]=0  
**for** i **in** range(m+1):  
 s[i][0]=0  
**for** i **in** range(1,m+1):  
 **for** j **in** range(1, n + 1):  
 **if** a[i-1]==b[j-1]:  
 s[i][j]=s[i-1][j-1]+1  
 **else**:  
 s[i][j]=max(s[i-1][j],s[i][j-1])  
print(s[m][n])  
  
afis(m,n)  
  
print()  
print(s)

**e)Submultime de suma M**

*#subpb: (s,i) - se poate obtine suma s din primele i elemente  
#v[s][i] = true ddc v[s-ai][i-1] este true #s=1,..,M  
#stim v[0][0]=True, v[s][0]=False (cu submultimea vida putem obtine suma 0)*f=open(**"multime.in"**)  
n=int(f.readline())  
a=[0]  
a.extend([int(x) **for** x **in** f.readline().split()])  
M=int(f.readline())  
v=[[**False for** j **in** range(M+1)] **for** i **in** range(n+1)]  
elem=[[0 **for** j **in** range(M+1)] **for** i **in** range(n+1)]  
v[0][0]=**True  
for** i **in** range(1, n + 1):  
 **for** s **in** range(0, M + 1):  
 *#daca s nu s-a putut inca obtine vedem daca putem obtine suna s folosind a[i]  
 #s se poate obtine folosind a[i] <=> s-a[i] s-a putut deja obtine (pana la pasul i-1)* **if** v[i-1][s]==**False and** s >= a[i] **and** v[i-1][s-a[i]]==**True**:  
 v[i][s] = **True** elem[i][s] = a[i]  
 **else**:  
 v[i][s] = v[i-1][s]  
 elem[i][s] = elem[i - 1][s]  
print(v[n][M])  
**if** v[n][M]:  
 i = n  
 s = M  
 **while** s>0:  
 print(elem[i][s],end=**" "**)  
 s=s-elem[i][s]  
 i=i-1  
  
f.close()  
print()  
print(**"varianta cu un vector"**)  
*#VARIANTA  
#suficient vector pentru memoizare, nu matrice*f=open(**"multime.in"**)  
n=int(f.readline())  
a=[int(x) **for** x **in** f.readline().split()]  
M=int(f.readline())  
v=[**None**]\*(M+1)  
v[0]=-1  
**for** i **in** range(n):  
 *#print(v)  
 #print(a[i])* **for** s **in** range(M,a[i]-1,-1):  
 **if** v[s]==**None and** v[s-a[i]] != **None**:  
 v[s]=i *#indice ultimului element  
#print(a)  
#print(v)***if** v[M]!=**None**:  
 s=M  
 **while** s>0:  
 print(a[v[s]],end=**" "**)  
 s=s-a[v[s]]

**f)Subsir crescator de lungime maxima**

**def** rezolva(a):  
 n = len(a)  
 lung = [0]\*n  
 succ = [n]\*n  
 lung[n-1] = 1  
 succ[n-1] = n  
 poz = n-1  
 l\_max = 1  
 **for** i **in** range(n-2,-1,-1):  
 succ[i] = n  
 max\_lungj=0  
 **for** j **in** range(i+1, n):  
 **if** (a[i]<a[j]) **and** (lung[j]>max\_lungj):  
 max\_lungj = lung[j]  
 succ[i] = j  
 lung[i] = 1 + max\_lungj  
 **if** lung[i]> l\_max:  
 poz = i  
 l\_max = lung[i]  
  
  
 print(**f"Lungimea maxima {**l\_max**}"**)  
  
 print(**"O solutie:"**)  
 **for** i **in** range(l\_max):  
 print(**f"{**a[poz]**} "**,end=**""**)  
 poz = succ[poz]  
 **return** lung  
  
  
**def** calculNrSubsiruri(lung):  
 n=len(lung)  
 lmax=max(lung)  
 nr=[0]\*n  
 nr[n-1]=1  
 **for** i **in** range(n-2,-1,-1):  
 nr[i]=0  
 **for** j **in** range(i+1,n):  
 **if**(a[i]<a[j]) **and** (1+lung[j]==lung[i]):  
 nr[i]=nr[i] + nr[j]  
  
  
 **if** nr[i]==0:  
 nr[i] = 1  
  
   
  
 s=0  
 **for** i **in** range(n):  
 **if** lung[i]==lmax:  
 s=s+nr[i]  
 **return** s  
   
   
  
**with** open(**"subsir.in"**) **as** f:  
 a = [int(x) **for** x **in** f.readline().split()]  
 lung=rezolva(a)  
 print()  
 print(**f"sunt {**calculNrSubsiruri(lung)**} subsiruri de lungime maxima"**)

**g)Traseu S-E**

**def** afis\_matrice(m):  
 **for** linie **in** m:  
 **for** x **in** linie:  
 print(**f"{**x**:4}"**, end=**""**)  
 print()  
  
  
f = open(**"matrice\_S\_E.in"**)  
m,n= (int(x) **for** x **in** f.readline().split())  
a = []  
**for** linie **in** f:  
 a.append([int(x) **for** x **in** linie.split()])  
f.close()  
  
print(a)  
  
*#VARIANTA 1 s[i][j] = suma maxima pentru un traseu care se termina pe pozitia (i,j) (incepand din (1,1), de fapt (0,0))*s=[[0 **for** j **in** range(n)] **for** i **in** range(m)]  
  
*#stim direct*s[0][0]=a[0][0]  
  
*#calculam prima linie din s***for** j **in** range(1,n):  
 s[0][j]=a[0][j]+s[0][j-1]  
  
*#calculam prima coloana din s***for** i **in** range(1,m):  
 s[i][0]=a[i][0]+s[i-1][0]  
  
**for** i **in** range(1,m):  
 **for** j **in** range(1,n):  
 **if** s[i-1][j]>s[i][j-1]:  
 s[i][j]=a[i][j]+s[i-1][j]  
 **else**:  
 s[i][j]=a[i][j]+s[i][j-1]  
afis\_matrice(a)  
print()  
afis\_matrice(s)  
print(**"suma maxima "**,s[m-1][n-1])  
  
*#reconstituire traseu din recurenta - de la sfarsit (m-1,n-1) la inceput:***def** traseu(i,j):  
 **if** i>=0:  
  
 **if** j>0 **and** s[i][j]==a[i][j]+s[i][j-1]:  
 traseu(i,j-1)  
 **else**:  
 traseu(i-1, j)  
 print(**f"{**i + 1**} {**j + 1**}"**)  
  
traseu(m-1,n-1)  
  
  
*#VARIANTA 1 s[i][j] = suma maxima pentru un traseu care incepe pe pozitia (i,j)  
# (si terminand pe pozitia (m,n))*s=[[0 **for** j **in** range(n)] **for** i **in** range(m)]  
  
*#stim direct*s[m-1][n-1]=a[m-1][n-1]  
  
*#calculam ultima linie din s***for** j **in** range(n-2,-1,-1):  
 s[m-1][j]=a[m-1][j]+s[m-1][j+1]  
  
*#calculam ultima coloana din s***for** i **in** range(m-2,-1,-1):  
 s[i][n-1]=a[i][n-1]+s[i+1][n-1]  
  
**for** i **in** range(m-2,-1,-1):  
 **for** j **in** range(n-2,-1,-1):  
 **if** s[i+1][j]>s[i][j+1]:  
 s[i][j]=a[i][j]+s[i+1][j]  
 **else**:  
 s[i][j]=a[i][j]+s[i][j+1]  
afis\_matrice(a)  
print()  
afis\_matrice(s)  
print(**"suma maxima "**,s[0][0])  
  
*#reconstituire traseu din recurenta - de la (0,0)***def** traseu(i,j):  
 **if** i<=m-1 **and** j<=n-1:  
 print(**f"{**i + 1**} {**j + 1**}"**)  
 **if** i<m-1 **and** s[i][j]==a[i][j]+s[i+1][j]: *#!!iesire din matrice* traseu(i+1,j)  
 **else**:  
 traseu(i, j+1)  
  
  
traseu(0,0)

**4)BACKTRACKING**

**a)Anagrame**

**def** back(k):  
 **if** k==n+1:  
 rez.add(**""**.join([s[x[i]-1] **for** i **in** range(1,k)]))  
 **else**:  
 **for** i **in** range(1, n+1):  
 x[k] = i  
 **if** x[k] **not in** x[:k]: *#mai bine x.index(x[k],0,k)* back(k+1)  
  
  
s=input(**"cuvant "**)  
n=len(s)  
*# o soluție s va avea n elemente*x = [0]\*(n+1)  
print(**f"Anagrame:"**)  
rez=set()  
back(1)  
print(rez)

**b)Aranjamente**

**def** back(k):  
 **if** k==m+1:  
 print(\*x[1:],sep=**", "**)  
 **else**:  
 **for** i **in** range(1, n+1):  
 x[k] = i  
 *#print(x[1:k+1])* **if** x[k] **not in** x[:k]: *#mai bine x.index(x[k],0,k)* back(k+1)  
  
  
n = int(input(**"n = "**))  
m = int(input(**"n = "**))  
x = [0]\*(m+1)  
print(**f"Aranjamente de lungime {**m**} ale multimii {{1,...,{**n**}}}:"**)  
back(1)

**c)Combinari**

**def** back(k):  
 **if** k==m+1:  
 print(\*x[1:],sep=**", "**)  
 **else**:  
 **for** i **in** range(1, n+1):  
 x[k] = i  
 **if** x[k] > x[k-1]:  
 back(k+1)  
  
**def** back1(k):  
 **if** k==m+1:  
 print(\*x[1:],sep=**", "**)  
 **else**:  
 **for** i **in** range(x[k-1]+1, n+1):  
 x[k] = i  
 back1(k+1)  
  
n = int(input(**"n = "**))  
m = int(input(**"m = "**))  
*# o soluție s va avea m elemente*x = [0]\*(m+1)  
print(**f"Toate combinarile (submultimi) de {**m**} ale multimii {{1,...,{**n**}}}"**)  
back(1)  
  
print(**f"Toate combinarile (submultimi) de {**m**} ale multimii {{1,...,{**n**}}}-metoda 2"**)  
back1(1)

**d)Descompuneri distincte**

**def** back1(k):  
 **global** s  
 **if** sum(x[1:k])==n:  
 print(\*x[1:k],sep=**"+"**)*#!!!ramane completat cu valori pana la k* **else**:  
 **for** i **in** range(x[k-1], n+1):  
 x[k] = i  
 **if** sum(x[1:k+1])<=n:  
 back1(k+1)  
  
  
**def** back(k):  
 **global** s  
 **if** s==n:  
 print(\*x[1:k],sep=**"+"**)*#!!!ramane completat cu valori pana la k* **else**:  
 **for** i **in** range(x[k-1], n+1):  
 x[k] = i  
 s = s + x[k]  
 **if** s<=n:  
 back(k+1)  
 s = s - x[k]  
  
  
n = int(input(**"n = "**))  
x = [0]\*(n+1)  
print(**f"Descompuneri distincte:"**)  
x[0]=1  
s=0  
back(1)  
back1(1)

**e)Monede**

**def** back1(k):  
 **global** s  
 **if** k==n+1:  
 **if** sum([x[j]\*v[j] **for** j **in** range(1,k)])==S:  
 **for** i **in** range(1,k):  
 print(**f"{**x[i]**}\*{**v[i]**} {'+' if** i<k-1 **else ' '}"**,end=**" "**)  
 print()  
 **else**:  
 **for** i **in** range(0, S//v[k]+1):  
 x[k] = i  
 **if** sum([x[j]\*v[j] **for** j **in** range(1,k+1)])<=S:  
 back1(k+1)  
  
  
**def** back(k):  
 **global** s  
 **if** k==n+1:  
 **if** s==S:  
 **for** i **in** range(1, k):  
 print(**f"{**x[i]**}\*{**v[i]**} {'+' if** i < k - 1 **else ' '}"**, end=**" "**)  
 print()  
 **else**:  
 **for** i **in** range(0, S//v[k]+1):  
 x[k] = i  
 s = s + x[k]\*v[k]  
 **if** s<=S:  
 back(k+1)  
 s = s - x[k]\*v[k]  
  
v=[0]  
v.extend([int(x) **for** x **in** input(**"valori monede: "**).split()])  
n=len(v)-1  
S=int(input(**"S="**))  
x = [0]\*(n+1)  
s=0  
print(**f"Plati suma {**S**}:"**)  
back(1)  
  
print(**f"Plati suma {**S**}:"**)  
back1(1)

**f)Permutari cu un punct fix**

**def** back1(k):  
 **global** s  
 **if** k==n+1:  
 **if** sum([x[j]\*v[j] **for** j **in** range(1,k)])==S:  
 **for** i **in** range(1,k):  
 print(**f"{**x[i]**}\*{**v[i]**} {'+' if** i<k-1 **else ' '}"**,end=**" "**)  
 print()  
 **else**:  
 **for** i **in** range(0, S//v[k]+1):  
 x[k] = i  
 **if** sum([x[j]\*v[j] **for** j **in** range(1,k+1)])<=S:  
 back1(k+1)  
  
  
**def** back(k):  
 **global** s  
 **if** k==n+1:  
 **if** s==S:  
 **for** i **in** range(1, k):  
 print(**f"{**x[i]**}\*{**v[i]**} {'+' if** i < k - 1 **else ' '}"**, end=**" "**)  
 print()  
 **else**:  
 **for** i **in** range(0, S//v[k]+1):  
 x[k] = i  
 s = s + x[k]\*v[k]  
 **if** s<=S:  
 back(k+1)  
 s = s - x[k]\*v[k]  
  
v=[0]  
v.extend([int(x) **for** x **in** input(**"valori monede: "**).split()])  
n=len(v)-1  
S=int(input(**"S="**))  
x = [0]\*(n+1)  
s=0  
print(**f"Plati suma {**S**}:"**)  
back(1)  
  
print(**f"Plati suma {**S**}:"**)  
back1(1)

**g)Permutari**

**def** back(k):  
 **if** k==n+1:  
 print(\*x[1:],sep=**", "**)  
 **else**:  
 **for** i **in** range(1, n+1):  
 x[k] = i  
 *#print(x[1:k+1])* **if** x[k] **not in** x[:k]: *#mai bine x.index(x[k],0,k)* back(k+1)  
  
  
n = int(input(**"n = "**))  
*# o soluție s va avea n elemente*x = [0]\*(n+1)  
print(**f"Permutările de lungime {**n**}:"**)  
back(1)

**h)Produs cartezian**

**def** back(k):  
 **if** k==n+1:  
 **for** i **in** range(1,n+1):  
 print(m[i-1][x[i]],end=**" "**)  
  
 print()  
 **else**:  
 **for** i **in** range(0,len(m[k-1])):  
 x[k] = i  
 back(k+1)  
  
  
f=open(**"multimi.in"**)  
m=[[int(x) **for** x **in** linie.split()] **for** linie **in** f]  
print(m)  
n=len(m) *#numarul de multimi*x = [0]\*(n+1)  
print(**"Elementele podusului cartezian:"**)  
back(1)

**i)Submultimi**

**def** back(k):  
 **if** k==n+1:  
 print(**"submultime:"**,end=**" "**)  
 **for** i **in** range(1,n+1):  
 **if** x[i]==1:  
 print(v[i-1],end=**" "**)  
  
 print()  
 **else**:  
 **for** i **in** range(0,2):  
 x[k] = i  
 back(k+1)  
  
  
s=input(**"multime:"**)  
v=[int(x) **for** x **in** s.split()]  
n=len(v)  
x = [0]\*(n+1)  
print(**f"Submultimile multimii {{ {','**.join(s.split())**} }}:"**)  
back(1)

**j)Submultimi de suma M**

**def** suma(x,n):  
 s=0  
 **for** i **in** range(1, n + 1):  
 **if** x[i] == 1:  
 s=s+v[i - 1]  
 **return** s  
  
  
**def** back(k):  
 **if** k==n+1:  
 **if** suma(x,n)==M:  
 **for** i **in** range(1,n+1):  
 **if** x[i]==1:  
 print(v[i-1],end=**" "**)  
  
 print()  
 **else**:  
 **for** i **in** range(0,2):  
 x[k] = i  
 **if** suma(x,k)<=M:  
 back(k+1)  
  
**def** back2(k):  
 **global** s  
 **if** k==n+1:  
 **if** s==M:  
 **for** i **in** range(1,n+1):  
 **if** x[i]==1:  
 print(v[i-1],end=**" "**)  
  
 print()  
 **else**:  
 **for** i **in** range(0,2):  
 x[k] = i  
 **if** x[k] == 1:  
 s = s + v[k - 1]  
 **if** s <= M:  
 back2(k + 1)  
 **if** x[k] == 1:  
 s = s - v[k - 1]  
  
s=input(**"multime:"**)  
v=[int(x) **for** x **in** s.split()]  
M=int(input(**"M="**))  
n=len(v)  
x = [0]\*(n+1)  
print(**f"Submultimile de suma {**M**}:"**)  
back(1)  
  
print(**f"Submultimile de suma {**M**} - metoda 2:"**)  
s=0  
back2(1)

**k)Toate subsirurile**

**def** rezolva(a):  
 n = len(a)  
  
 lung[n-1] = 1  
 l\_max = 1  
 **for** i **in** range(n-2,-1,-1):  
 max\_lungj=0  
 **for** j **in** range(i+1, n):  
 **if** (a[i]<a[j]) **and** (lung[j]>max\_lungj):  
 max\_lungj = lung[j]  
 lung[i] = 1 + max\_lungj  
 **if** lung[i]> l\_max:  
 l\_max = lung[i]  
 **return** l\_max  
  
**def** afisToateSolutiile():  
  
 **for** i **in** range(0,n):  
 **if** lung[i] == l\_max:  
 x[1] = i  
 back(2)  
**def** back(k):  
 **if** k == l\_max + 1:  
 **for** i **in** range(1,k):  
 print(a[x[i]], end=**" "**)  
 print()  
  
 **else**:  
 **for** j **in** range(x[k-1]+1,n):  
 x[k]=j  
 **if** (a[x[k-1]] < a[x[k]]) **and** lung[x[k-1]] == 1+lung[x[k]]:  
 back(k+1)  
f=open(**"subsir.in"**)  
a = [int(x) **for** x **in** f.readline().split()]  
n=len(a)  
lung = [0]\*n  
x= [0]\*(n + 1)  
l\_max=rezolva(a)  
afisToateSolutiile()