Divide et impera

I Merge sort

3. subprograme:
   - sort(p, q, a) - verif dacă a[p] > a[q] și swap
   - interclasare: vector nou în care interclasezi prima și a 2-a jumătate a unui vector a
   - divide(p, q, a) - dacă q - p <= 1 - sort
     altfel se reapelează pt fiecare jum de vector

II Mediana a 2 vect de lungimi diferite

→ dacă len(v1) <= 2,
- se ia un vect nou și dacă len(v2) % 2 == 0 se pun cele 2 el din mijloc, altfel se ia doar el din mijloc
- se interclasează (într-un vector nou) el din v1 și noul vector
- se returnează mediana: el din mij dacă sunt nr impar și ~~el~~ media el din mij dacă lungimea e pară

→ altfel:
- se ia mediana fiecărui vector: m(v1) = x, m(v2) = y
- dacă sunt egale, se returnează această valoare
- dacă x < y se renunță la a 2-a jumătate din v1, și la tot atâtea elemente din prima jumătate a lui v2.
  → se reactualizează lungimile: n = n//2; m = m//2.
- dacă x > y → se renunță la prima jum din v1 și la tot atâtea el din a 2-a jum a lui v2 și se reactualizează indicii
- se reapelează mediană pt noii vectori

III Arie maximă

→ coordonate în matrice

→ subprogr (stg, sus, dr, jos):
- global se declară maximul și lista cu coord. găuri
- se verifică, pt fiecare mask, dacă se află în aria respectivă (up < i < down and left < j < right)
  - ok = Fals
  - se reapelează pt fiecare bucată (se împarte în 4 matricea și se reapelează subprogr)
- dacă ok = True atunci:
  sup = (right - left) * (down - up)
  if sup > maxS:
  maxS = sup

→ se returnează maxS.

IV Al k-lea minim cu quick_select

→ pivotul e primul el din v
→ se construiesc 2 vectori: 1 cu el din stg pivotului ~~mai~~ și unul din dr
→ dacă len de stg e k-1 ⇒ piv este el căutat
→ len(stg) > k-1 quick_select de (stg, k)
→ len(stg) < k-1 quick_select de (dr, m-l)

a = [0] * n

a = [ ]

Programare dinamică

I Problema scărilor → generalizare a Fibo

m - nr maxim de scări pe care le urcă la un pas

n - nr de scări

m = 2

n = 0 - 1 mod

n = 1 - 1 mod

n = 2 - 1,1 ; 2 ⇒ 2 moduri

n = 3 - 3 moduri
1, 2
1, 1, 1
2, 1

n = 4 - 5 moduri
1, 1, 2
2, 2
1, 2, 1
1, 1, 1, 1
2, 1, 1

n = 5
scara 3 + 2 { 1, 2, 2 ; 1, 1, 1, 2 ; 2, 1, 2 }
scara 4, 1 { 1, 1, 2, 1 ; 2, 2, 1 ; 1, 2, 1, 1 ; 1, 1, 1, 1, 1 ; 2, 1, 1, 1 }

Pasul i se poate baza pe ultimii m pași
$(i-1, i-2, i-3 \dots i-m)$

Scara	0	1	2	3	4	5	6	7
Nr de moduri	1	1	2	3	5	8	13	21

n = 5 $f(0) = 1$
$f(1) = 1$
$f(2) = f(2-1) + f(2-2) = f(1) + f(0)$
$f(3) = f(3-1) + f$

$$f(n) = f(n-1) + f(n-2)$$

$$f(n) = \sum_{k=1}^{m} f(n-k)$$

II Nr min de sărituri pentru a ajunge la finalul vectorului

V: 2 1 3 2 3 4 5 1 2 9
0 1 2 3 4 5 6 7 8 9

Min_jump: 0 1 1 2 2 2 3 3 3 3 - nr min de sărituri necesare
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Jump_path: 0 0 2 2 2 4 4 5 5 - vectorul de reconstituire a drumului
0 1 2 3 4 5 6 7 8 9

i = indicele locului din vect

```
for i in range(0,n):
    for j in range(i):
```

Min_jump[i] = min(Min_jump[i], Min_j[j+1])

jump_path[i] = min(j_p[i], j)

III Cel mai lung subșir comun

len(str1) = n = m
len(str2) = m = n
M[n+1][m+1]
M[len(str2)][len(str1)]
M[n+1]*(m+1)

X	empty	b	z	d	n	c	v	e	f	g	h
empty	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0
b	0	1	1	1	1	1	1	1	1	1	1
c	0	1	1	1	1	2	2	2	2	2	2
v	0	1	1	1	1	2	3	3	3	3	3
d	0	1	1	2	2	2	3	3	3	3	3
e	0	1	1	2	2	2	3	4	4	4	4
f	0	1	1	2	2	2	3	4	5	5	5
g	0	1	1	2	2	2	3	4	5	6	6
h	0	1	1	2	2	2	3	4	5	6	7 = 7

bcvefgh

```
for i in range(len(str1)):
    for j in range(len(str2)):
        if str1[i] == str2[j]:
            M[i][j] = M[i-1][j-1]+1
        else: M[i][j] = max(M[i-1][j], M[i][j-1])
```

IV Monede / timbre / partiția întregilor

n	ways	P(n)
1	1	1
2	2 1+1	2
3	3 1+2 1+1+1	3

Suma →

↓ monedă	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	1	1	1	1	1
2	1	1	2	2	3	1+2
3	1	1	2	3	4	3+2
4	1	1	2	3	5	5+1
5	1	1	2	3	5	6+1 =7

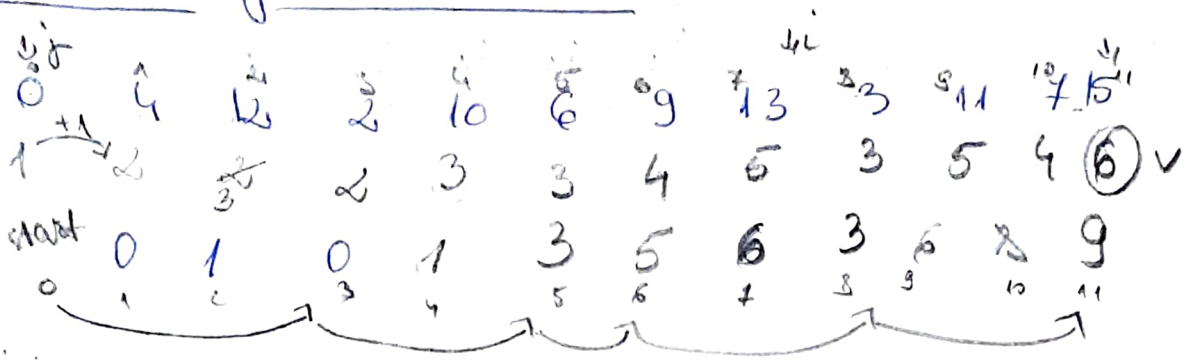
1. Exclude moneda
2. Include moneda
3. Adună ① și ②

	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	0+1	0+1	0+1	0+1	0+1
2	1	1	1+1	1+1	1+2	1+2
3	1	1	2	3	4	5
4	1	1	2	3	5	6
5	1	1	2	3	5	7

if i > j - copiază val de deasupra
else: V[i][j] = V[i-1][j] +
+ V[i][j-1]

Pentru monede / bancnote normale:
dată de forma [x, y] unde
x este indicele (și matrice) și y este
valoarea efectivă.

V Cel mai lung subșir crescător



INITIAL: L=[1]*(len(v)+1)

```
for i in range(n+1):
    for j in range(i):
        if V[i] > V[j]:
            L[i] = max(L[j]+1, L[i])
            # if L[j]+1 > L[i]:
            #     S[i] = j
            S[i] = j
```

i: 0 3 5 6 9 11

L=[[[a,b],c],[[a,b],c]]
L[0][0][0]

VI Programare spectacole

Nume	a	b	c	d	e	f
timp	(1,4)	(2,6)	(4,7)	(6,8)	(5,9)	7,10

P:

3	5	2	6	4	8
3	5	5	11	7	13

I Se sortează după final

II
```
for i in range(1, n):
    for j in range(i):
        if V[i][0][0] >= V[j][0][1]
            {→ se pot executa împreună
               → se incrementează profitul ( P[i] = P[i]+P[j] )}
        P[i] = max(V[i][1] + V[j][1], P[i])
```

VII Rucsac 0/1

	greutate	profit
0)	1	1
1)	3	4
2)	4	5
3)	5	7
4)	2	4

I Sortare după greutate

II If item_g <= sum $d[i-1][j]$ valoarea

valul = max(val_vechi, val_v + nou)

else: copiază de deasupra

$n = 9$ sume parțiale →

i	p	g	0	1	2	3	4	5	6	7	8	9	
0	1	1	0	1	1	1	1	1	1	1	1	1	i
1	4	2	0	1	4	5	5	5	5	5	5	5	i
2	4	3	0	1	4	5	5	8	9	9	9	9	i
3	5	4	0	1	4	5	5	8	9	10	10	13	i
4	7	5	0	1	4	5	5	8	9	11	12	13	i
			j	j	j	j	j	j	j	j	j	j	

În celule se trece profitul

$M[i][j] = max(M[i-1][j], P[i] + M[i-G[i]])$

4 + $M[i][j - G[i]]$

1 3 - 2

VIII Problema monedelor

V: | 0 | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 2 | 2 |
0 1 2 3 4 5 6 7 8 9 10 11 12 13

M = [1, 5, 7]

Q: ~~1~~, ~~5~~, ~~7~~, 2, 6, ~~8~~, 10, 12, 3

IX Sortarea topologică

	0	1	2	3	4	5	6	7	8
Vizit:	-1	0	0	0	0	0	0	0	0
Ard:									

I În Q se pun toate nodurile de gr interm 0 și marcăm ca vizitat
II Începi cu nodul 1 și îi pui vecinii în coadă
III Pt toți din coadă mergi și pui vecinii (dacă nu sunt deja)

V_m = [( ), ( ), ( )]
Viz = [0, 0, 0, 0]
Gr = [0, 0, 0 ... 0]
Q = [ ]