

Curs

Instrucțiuni

① break, continue și else

```
for x in range(n+1):  
    if x % 2 == 0:  
        continue  
    print(x)
```

Afișare

1
3
5
7

```
for x in range(a, b+1):  
    for d in range(2, x//2 + 1):  
        if x % d == 0:  
            break  
    else:  
        print(x)  
        break  
else:  
    print('Nu există numere prime')
```

Afișare

Numerele prime
între (a, b)

Afizează cel
mai mic
număr prim

② pari

TIPURI DE DATE PREDEFINITE

- Siruri de caractere

O sir de caractere - o secvență de caractere indexată de la 0

constante: ' ', ", ", " ", null, null

```
s = 'test'  
s = "test"  
s = "test"\\"  
      "simple" => s = "test simple"
```

```
s = "nn" test  
      simple"nn"  print  test  
                  simple
```

Secvențe escape: \n - linie nouă
\\t - tab (4 spații)
\\ - \

Bazin de siruri (String pool)

```
s = 'test'  
t = 'test'
```

t → 'test' intervening

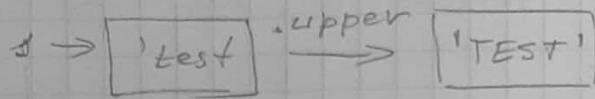
Sirurile in python sunt inmutabile

inmutabil - valoare
continutul nu mai poate fi modificat, referința poate fi modificată

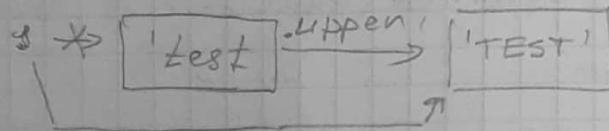
clasa
↑
șablon

Object
↑
Instanțiere
a unei clase

s = 'test'
s.upper()
print(s) → "test"



s = 'test'
s = s.upper()
print(s) → "TEST"



Accesare elemente din un sir

a) print(s[2]) → "o"

s = "programare"
s = P r o g r a m a r e
 ^ 0 1 2 3 4 5 6 7 8 9
 | -10 -9 -8 -7 -6 -5 -4 -3 -2 -1

b) slice

s[1:4] = "rog"
s[-9:-6] = "rog"
s[-9:4] = "rog"
s[:4] = "prog"
s[6:] = "mare"

inversare
s[2:-1] = "eramargorop"
s[3:2] = "w" - sir vid
 3 > 2

Operatori

+ → concatenare

"abc" + "de" = "abcde"
"abc" * 3 = "abcabcabc"

*

multiplicare

in
not in } → testarea apartenenței

"abc" in "bcabed" = True
"a" in "baabod" = True

Functii predefinite

a) len(str)

len('test') = 4

b) str(valoare)

str(123) = "123"

str(True) = "True"

type(x) + str(x) → print('x' + str(x))

c) min(sir)

min('Teste') = 'T'

max(sir)

max('Teste') = 'e'

a) ord (caracter)
chr (nr. nat.)

ord ('A') = 65
chr(65) = 'A'

Metode

1) s = "Ana are {} mere {}!" formatarea

s = s.format('5', "rosii") \Rightarrow "Ana are 5 mere rosii!"

sau

s = "Ana are {} mere {}!"

s = s.format("rosii", 5) \Rightarrow "Ana are 5 mere rosii!"

sau

s = "Ana are {nr} mere {culoare}!"

s = s.format(nr=5, culoare="rosii") \Rightarrow "Ana are 5 mere rosii!"

2) pt. transformarea la nivel de caracter

- lower()
- upper()
- swapcase()
- title()
- capitalize()
- doar primul caracter din sir
- A \rightarrow a a \rightarrow A
- A \rightarrow A a \rightarrow a
- A \rightarrow a a \rightarrow A
- "Ana are mere" \rightarrow "Ana Are Mere!"
- "ana are mere. și pene" \rightarrow
- \rightarrow "Ana are mere. și pene"

3) pt. clasificare

- islower() - placez toate caracterile sunt mici
- isupper() - sunt mari
- isdigit() - sunt numere

4) pt. căutare

- count (subsr, st, dr) $\begin{matrix} \text{optionali} \\ \text{inclusiv} \end{matrix}$ - numără de câte ori se exclusiv întâlnește în sir, subșirul
- s = "abodeeabc"
- s.count("abc") = 2
- s.count("abc", 3) = 1

- find (subsr, st, dr) - prima apariție de la stânga la dreapta

- rfind (subsr, st, dr) - prima apariție de la dreapta la stânga (ultima)

- s.find('aba') = 0
- s.rfind('aba') = 5
- s.find('za') = -1

5) .split (separator)

s = "Ana are mere, mere și prune."

t = s.split()

t = ["Ana", "are", "mere", "și", "prune"]

6) .join (listă)

separator

s = ", ".join(t) → "Ana, are, mere, mere, și, prune."

7) .strip () - Elimină spațiile de la început și sfârșitul sirului

Datetime

.strftime()

'%A' > get day name # Monday

'%a' > get day name # Mon

CURS

COLECTII DE DATE

- ① Lista (list) = o secvență de valori închise de la 0
- mutabilă

a) Crearea unei liste

1) l = []

2) l = [1, 2, "text", 3.14, 7, 8]

3) l = [[1, 2], [3, 4, 5]]

4) l = ["MURĂ Victor", 135, [9, 10, 10]]

5) l = [a ** 2 for a in range(10)]

6) l = [a for a in range(10) if a % 2 == 0]

7) l1 = [...]

l2 = [...]

l3 = [a for a in l2 if a in l1]

b) Accesarea elementelor unei liste

1) indicei poz / neg.

2) secvențe de indici

* b = a[1:5]

* a[2] = 7

* a = [1, 2, 3, 4, 5]

a[1:3] = [7, 8, 9] ⇒ a = [1, 7, 8, 9, 4, 5]

3) stergerea unui element
 del a[1:3]

c) Functii predefinite

- len (list)
- min (list)
- max (list)
- list (str)

d) Operatori

- (+) - concatenare a două liste
- (*) - repetarea aceluiși sevență de n ori
- (+=), • (in), • (not in), • (<, <=, >, >=) necesită același tip de date pentru elementele comparabile
- (>=), • (!=)

e) Metode specifice

- append (elem)

Ex. for i in range(n):
 x = int(input('x = '))
 l.append(x)
 sau
 l += [x]

- extend (lista)

Ex. l = [1, 2, 3] -
 l.append([4, 5])
 # [1, 2, 3, [4, 5]]

 l.extend([4, 5])
 # [1, 2, 3, 4, 5]

- error.
dacă nu există elem.
- insert (poz, element)
 - remove (element) - șterge prima apariție în listă
 - index (element) - poziția primului element gasit
 - count (val) - numără apariția elementului în listă
 - pop (poz) - furnizează valoarea de pe poziția respectivă și împăcare o șterge
returnând
 - pop () - returnând și elimină ultimul element din listă
 - clear () $\Leftrightarrow l = []$
 - reverse () - întoarce elementele în listă

f) Copierea unei liste : b = a.copy()

a = [1, 2, 3]
b = a
referințe
a[2] = 7 \Rightarrow b[2] = 7

$b = a \cdot \text{copy}()$ Shallow copy
 $a[2] = 7 \Rightarrow b[2] = 2$

$a = [1, 2, [3, 4]]$
 $b = a \cdot \text{copy}()$
 $a[2][0] = 7 \Rightarrow b[2][0] = 7$ # $b = [1, 2, [7, 4]]$

`import copy`
`b = copy.deepcopy(a)`

Matrice
 $a = [[0] * n] * m \leftarrow \text{linii}$
sau
coloane
 $a = [[0 \text{ for } a \text{ in range}(n)] \text{ for } l \text{ in range}(m)]$

② Tupluri / tuple = liste imutabile \Rightarrow dimensiune fixă

a) Crearea unei tupluri

- $t = ()$
- $t = (1, 2, 3)$ sau $t = 1, 2, 3$
- $t = (1,)$

b) Accesarea elementelor

- prin indecsă poz/ neg
- ~~index~~ > eroare
- ~~for~~ > eroare

c) Operatori

- $(+)$
- $(*)$
- $(+=)$
- $(in, not in)$
- $(<, >, <=, >=)$
- $(==, !=)$

Ex.
 $a = (1, 2, 3)$
 $a = a + "test"$
$a == (1, 2, 3, "t", "e", "s", "t")$

$a = (1, 2, 3)$
 $a = a + ("test",)$
$a == (1, 2, 3, "test")$

d) Functii predefinite

- $\text{len}(\text{tuple})$
- $\min_{\max}(\text{tuple})$
- $\text{tuple}(\text{val})$

e) Metode specifice

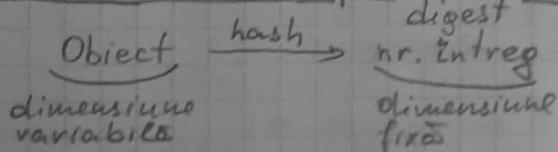
- count(element)
- index(element)

f) Împachotare, despachetare

$$t = 1, 2, 3$$

$$a, b, c = t$$

Tabele de dispersie (hash)

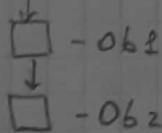


$Ob_1 \neq Ob_2 \Rightarrow \text{hash}(Ob_1) \neq \text{hash}(Ob_2) \rightarrow \text{coleziune}$

h	0	1	2	3	4	5
	■	□	□	□	□	□

$$x = \text{hash}(\text{obj}) \quad x=1$$

h	0	1	2	3	4	5
	■	□	□	□	□	□



Hashable

imutabil

↳ se poate asocia o valoare fixă sau constantă pe toată durata sa de viață
 $\text{---hash}()$
 se poate compara cu alte obiecte. $\text{---eq}()$

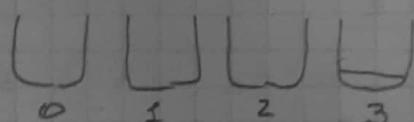
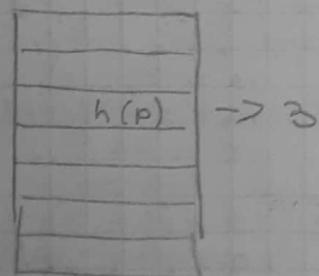
— Seminariu

$$h: A \rightarrow \text{uint}$$

$$f: [0, 2^{32}-1] \rightarrow [0, 16] \mod 17$$

$$f(x) = x \mod 17$$

$$\begin{aligned} A == B &\Rightarrow f(A) == f(B) \\ f(A) == f(B) &\nRightarrow A == B \end{aligned}$$



Less Pass.

Exemplu funcție HASH

def h(s):

$$p = 0$$

for 'c' in s:

return $\rho =$

$$A = \{1, 2, 3, 4\}$$

$$B = [1, 1, 1, 2, 1, 3, 2, 1]$$

1

TEMA: COLECTII DE DATE

③ Mulfini (set) (non-hashable)

Multime = o colectie de val. fară duplicate

```

graph TD
    Multime["Multime = o colectie de val. fară duplicate"] --> MUTABILA[MUTABILĂ]
    Multime --> IMUTABILA[IMUTABILE]
    MUTABILA --> P1["nu păstrează ordinea de inserare"]
    MUTABILA --> P2["nu sunt indexate"]
    IMUTABILA --> P3["nu păstrează ordinea de inserare"]
  
```

- Crearea

a) $S = \{1, 2, 3, 1, 2, 4\} \Rightarrow S = \{1, 2, 3, 4\}$

b) ~~s = f()~~, s = set()

$$c) s = \text{set}([1, 2, 1, 2, 3, 1]) \Rightarrow s = \{1, 2, 3\}$$

d) $s = \text{set}(\text{"teste"}) \Rightarrow s = \{ \text{"t"}, \text{"s"}, \text{"e"} \}$

e) $s = \text{set}([\text{"teste"}]) \Rightarrow s = \{\text{"teste"}\}$

`s = set(("teste",))` => `s = {"teste"}`

$$f) s = \{x \% 2 \text{ for } x \text{ in range}(10)\} \Rightarrow s = \{0, 1\}$$

- Accesarea

for x in `multime`:

三

- Operators

`in`, not `ih`, `=`, `!=`, `<`, `<=`, `>`, `>=`

in, not in, \geq , \leq , \neq , \geq , \leq , \geq , \leq , \geq

intersecție diferență simetrică

reunione & differenti

inclusive

$\langle \cdot, \cdot \rangle = \langle \cdot, \cdot \rangle$

olifeneus tā simetria ē
A. c)

- Funcții predefinite

`min(...), max(...), len(...), set(...)`

- Metode specifice

- a) . add (elem)
 - b) . update (multifime (sir, lista, tuple))
 - c) . remove (elem) → EROARE dacă elementul nu se găsește în multifime
 - d) . discard (elem) → NU DA EROARE dacă nu este elem.
 - e) . clear()
 - f) . union (colecție) * sir, multifime, tuple, lista
 - g) . intersection (colecție)
 - h) . semantic_difference (colecție)

④ Frozenset = multime immutabilo (hashible)

$s = \text{frozenset}([1, 2, 3, 1, 3])$

⑤ Dictionare (dict)

Dictionar = o colecție de perechi cheie : valoare
 ↑
 MUTABIL
 Indeterminate
 prin cheie, ci
 nu prin poziție

- Creare

- a) $d = \{ \}$

b) $d = \{ "a": 1, "b": 2, (1,2,3): "c" \}$

c) $d["a"] = 1$
 $d["b"] = 2$

d) $d[(1,2,3)] = "c"$

e) $d = dict([("a", 1), ("b", 2), ((1,2,3), "c")])$
 $d = \{ str(chr(65+x)): x \text{ for } x \text{ in range}(5) \} \Rightarrow$
 $\Rightarrow \{ "A": 0, "B": 1, "C": 2, "D": 3, "E": 4 \}$

- Accesarea unui element

d. [cheie] = valoare
d. get(cheie)

- Operators

in, not in, = , ! x
chei

$$\{ "A": 1, "B": 2 \} == \{ "B": 2, "A": 1 \}$$

- Functii predefinite

min(...), max(...), len(...), dict(...)
chei

- Metode specifice

a). `Keys() → [chei]`

. `values() → [valori]`

. `items() → [(cheie, valoare)]`

for K, V in d.items():
 print(K, V)

b). `pop(cheie)` - ERDARE în caz de inexistentă a cheii
. `pop(cheie, None)` - returnează valoarea și sterge itemul
. `pop(cheia, None)` - None în caz de inexistentă a cheii

c). `update(dict)`

TEMA: FIŞIERE TEXT

Fișier text = o secvență de caractere organizată pe linii și stocată pe un suport de memorie externă

organizarea pe linii

↑
CR + LF ↑
'\r' ' \n' ' \n'
Windows Unix

- Deschiderea unui fișier text

a) `f = open('cale fișier')` → implicit v-a fi deschis pentru citire

b) `f = open('cale fișier', 'mod de deschidere')`

↑ ↑ ↑
'r' - read 'x' - exclusive
'w' - write 'a' - append

- Citirea din fișier text

a). `read()` → tot fișierul într-un singur obiect de caractere.

b). `readline()` → linia curentă într-un singur obiect de caractere

c). `readlines()` → toate linile într-o listă de sururi

Exemplu

test.txt
Ana are
mere
și pere

s = f.read()
↳ s = "Ana are mere\nmeră și pere"
s = f.readline()
↳ s = "Ana are
mere
și pere"
s = f.readlines()
↳ s = ["Ana are\nmeră", "mere\nși pere"]

numere.txt

```
f = open("numere.txt")
```

```
f.readline()
```

```
v = [int(x) for x in f.readline().split()]
```

```
[7, -100, 5, 15, 20]
```

- Scrierea în fișier text

a). `write ("sir de caractere")`

b). `writelines (colecție de siruri)`

```
for i in range(10):  
    f.write(str(i) + '\n')
```

```
l = ["Ana", "are", "mere"]  
f.writelines(l) => Ana are mere
```

```
f.writelines('\n'.join(l)) => Ana  
are  
mere
```

- Închiderea unui fișier text

```
f.close()
```

with open('...') as f:

```
...  
...  
...  
...
```

Maia
Nu mai trebuie
=> f.close()

CURS

TEMA: FUNCȚII

① Parametrii unei funcții

```
def nume (parametri):  
    corpul funcției
```

a) parametri simpli

```
def suma(x,y):  
    return x+y
```

b) parametri cu valori implicite:

```
def suma(x=0, y=0):  
    return x+y
```

→ t = suma() #0
→ t = suma(2) #2
→ t = suma(7,3) #10
→ t = suma(y=3) #3

c) nr. variabile de parametri (varargs)

$$t = \text{suma}(1, 2)$$
$$t = \text{suma}(1, 2, 5, -3)$$

```
def suma(*args):  
    s = 0  
    for x in args:  
        s += x  
    return s
```

② Instr. return

```
def suma_prod(x, y):  
    return x + y, x * y
```

$$a, b = \text{suma_prod}(5, 3)$$

③ Transmiterea parametrilor de intrare

Call-by-object / Call-by-sharing
reference

```
def f(lst):  
    lst.append(100)
```

$$\begin{aligned} t &= [1, 2, 3] \\ f(t) \\ \text{print}(t) &\Rightarrow t = [1, 2, 3, 100] \end{aligned}$$

```
def f(lst):  
    lst = lst + [100]
```

$$\begin{aligned} t &= [1, 2, 3] \\ f(t) \\ \text{print}(t) &\Rightarrow t = [1, 2, 3] \end{aligned}$$

Parametri → immutabili \Rightarrow nu că modificarea configurației său nu se reflectă în exterior

mutabil \Rightarrow se reflectă în exterior

④ Transmiterea funcțiilor ca parametri (call-back)

a) def suma(a, b, t)?

```
s = 0  
for i in range(a, b+1):  
    s += t(i)  
return s
```

b) Sortare \rightarrow metoda sort (listă)
 funcția sorted
 reverse = False

$L = [215, 30, 47, 9, 11, 6]$

```
def suma_cifrelor(x):
    s = 0
    while x != 0:
        s += x % 10
        x //= 10
    return s
```

$l.sort(key=suma_cifrelor)$

$L = [11, 30, 6, 215, 9, 47]$

$L = ['Ana', 'mene', 'ane']$

$L.sort(key=len) \Rightarrow L = ['Ana', 'ane', 'mene']$

$L = [('Pop Ion', 131), ('Mihai Ane', 132)]$

```
def cmpName(t)
    return t[0]
```

⑤ Functii împărțite (nested)

`def f(x):`

```
    def g():
        print(x)
```

⑥ Variabile globale și locale

`def f():
 print(x)`

$x=100$
 $f() \Rightarrow 100$

`def f():
 x=200
 print(x)`

$x=100$
 $f() \Rightarrow 200$
 $print(x) \Rightarrow 100$

`def f():
 print(x)` $\xrightarrow{x=200}$ global x

Envrion

⑦ Generatoare

```
def parn(n):
    for i in range(n+1):
        if i%2 == 0:
            yield i
```

$pg = \text{parn}(100)$
 $x = next(pg)$

```

while x <= 10
    print(x)
    x = next(pg)

```

SEMINAR

$S_1 \leftarrow S_2$
 nu schimbă
 ID, friend,
 mult mai
 rapidă

$S_1 = S_1 | S_2$

CURS

TEMA: Complexitatea algoritmilor

$O(n^2)$ numărul aproximativ de operații elementare = n^2
 • T
 dimensiunea
 datelor de intrare = n

Operații elementare
 $\begin{cases} \xrightarrow{\quad} \text{op. de atribuire + op. aritmetică} \\ \xrightarrow{\quad} \text{op. de decizie + op. de salt} \\ \xrightarrow{\quad} \text{op. de citire / scriere} \end{cases}$

Obs Niciodată complexitatea unui algoritm NU poate fi
 mai mică decât complexitatea citirii datelor de intrare
 și/sau afisarea datelor de ieșire!

$$\begin{aligned}
 O(2n^2) &\approx O(n^2) \\
 O(n^2+7) &\approx O(n^2) \\
 O(2^n+n^2) &\approx O(2^n)
 \end{aligned}$$

- Clase de complexitate (în ordine crescătoare)

(1) Algoritmi cu complexitate constantă $\rightarrow O(1)$

Ex: Suma a două numere

(2) Algoritmi cu complexitate logaritmică $\rightarrow O(\log_2 n)$

Ex: Op. de căutare binară

(3) Algoritmi cu complexitate liniară $\rightarrow O(n)$

Ex: Min, Max a n valori

(4) Algoritmi cu complexitate $n \log_2 n$

Ex: Quick sort, Merge sort, Heap sort

⑤ Algoritmi cu complexitate pătratică $\rightarrow O(n^2)$
Ex: Bubble sort

⑥ Algoritmi cu complexitate polinomială $\rightarrow O(n^k)$, $k \geq 3$
Ex: Roy - Floyd

⑦ Algoritmi cu complexitate exponentială $\rightarrow O(a^n)$; $a \geq 2$
Ex: Generarea tuturor submulțimilor unei mulțimi cu n elemente $\rightarrow O(2^n)$

MINIMĂ

Ex: Un sir format din n nr. naturale distincte.
Se dă def. toate permutările cu sumă = s

$$v = (5, 1, 7, 3, 8, 9, 16, 10, 4) \\ s = 10$$

- a) considerăm numerele nesortate
 $O(n^2) \rightarrow$ căutare directă
 $O(n \cdot \max) \rightarrow$ căutare cu vector de frecvență
 $O(n) \rightarrow$ căutare val. ^{frecvență} compl cu dicționar

b) soritate strict crescător

- $O(n^2)$ $j \text{ max } v[i] + v[j] \leq s$
- $v = (1, 3, 4, 5, 7, 8, 9, 10, 16)$
 ↑ $i \text{ max } v[i] \leq s/2$
- căut $v[i] + v[j] = s \rightarrow O(n^2)$
+ căut $s - v[i]$ prin căutare binară $\rightarrow O(n^2)$

$O(n \log_2 n) \rightarrow$ căut $s - v[i]$ cu căutare binară

$O(n) \rightarrow$ căut $s - v[i]$ - folosind vectori de marcare / dicționare

ATENȚIE LA MEMORIE
 $v = (1, 3, 4, 5, 7, 8, 9, 10, 16) \quad \text{two pointers}$
 $s = 10 \quad O(n)$

Ex 2

$i = 0$

$p = 1$

while $i <= n$:

$j = 1$

while $j <= p$:

 print(j)

$j += 1$

$i += 1$

$p = p * 2$

$O(2^n)$

Ex. 3:

```

p = 1
while p < a:
    p = p * 2
while p <= b:
    print(p)
    p = p * 2

```

$$\begin{aligned}
2^k &\in [a, b] \\
2^k &\leq b \\
\downarrow \\
k &\leq \log_2 b \\
O(\log_2 b) &
\end{aligned}$$

Ex. 4

$$\begin{aligned}
i &= 0 \\
j &= \text{len}(v) - 1
\end{aligned}$$

while $i < j$:

 while $i < \text{len}(v)$ and $v[i] < 0$

$$i = i + 1$$

 while $j >= 0$ and $v[j] >= 0$ $O(n)$

$$j = j - 1$$

 if $i < j$:

$$v[i], v[j] = v[j], v[i]$$

□ CURS

TEMA: Tehnica de programare Greedy

Ement, general: Fie $A \neq \emptyset$. Să se determine o submulțime $S \subseteq A$ a. z. val. unei funcții obiectiv f să fie minimă / maximă pe S

Ex. Se consideră $A \subseteq \mathbb{Z}$. Să se determine $S \subseteq A$ pt. care suma elementelor este max. pe S

$$f(S) = \sum_{x \in S} x$$

① Forma generală a unui algoritm de tip Greedy :

- 1) Stabilirea unui criteriu de selecție pt. ca un element $x \in A$ să fie adăugat în S și dem. optimizării criteriului.
- 2) (optional) Sortarea convenabilă a elementelor mulțimii A
- 3) Se parcurge A element cu element și se adaugă în S elem. care verifică criteriul stabilit în 1)

② Complexități: $O(n)$, $O(n \log_2 n)$, $O(n^2)$

specifice

cu sortare cu criteriu

$|A|n$

complex de selecție

• Cazuri - exemplu: Plata unei sume folosind un număr minim de monede

S v_1, v_2, \dots, v_n	a) $\sigma = (8, 7, 2)$ $S = 23E$ $\quad \quad \quad \parallel$ $2 \cdot 8 + 7E$ $\quad \quad \quad \parallel$ 3 monede $1 \times 7E$	$\sigma = (2, 7, 1)$ $S = 14E$ $\quad \quad \quad \parallel$ $1 \times 8E + 6E$ $\quad \quad \quad \parallel$ $6 \times 1E$ $+ \text{monedă}$ $(\text{optim: } 2 \times 7E)$
-------------------------------	---	---

c) $\sigma = (8, 7, 5)$

$S = 14E$

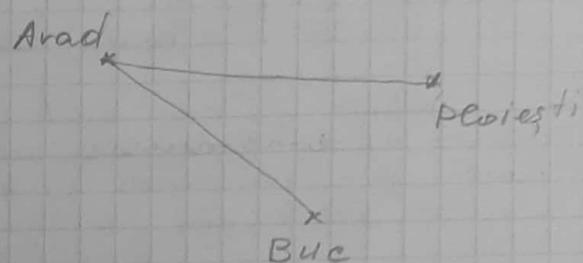
"

$1 \times 8E + 6E$

"

$1 \times 5E + 1E$

$\frac{\text{rest}}{\text{neplătitabil}}$



Pb. I Minimizarea timpului median de servire (agălpăre)

$n=8$ perechi

$\zeta_i = (7, 5, 2, 5, 3, 4, 6, 2)$

\checkmark

$\zeta_S = (7, 12, 14, 19, 22, 26, 32, 34)$

$t_m = 166/8 = 20,75$

$\zeta_{\zeta'} = (2, 2, 3, 4, 5, 5, 6, 7)$

$\zeta_S' = (2, 4, 7, 11, 16, 21, 27, 34)$

$t_{m'} = 122/8 = 61/4 = 15,25$

$t_m > t_{m'}$

Sol. pb: $G \in S_n$ a. i. $\zeta_{G(i)} \leq \zeta_G(j) \leq \dots \leq \zeta_{G(n)} \leq \dots \leq \zeta_{G(j)} \leq \dots \leq \zeta_{G(i)}$

Th: Permutările δ do mai sus asigură minimizarea timpului median de servire

Denum. Presupunem că ζ nu minimizează timpul median de servire $\Rightarrow \exists G' \in S_n$ care min. TMS $\Rightarrow \exists i < j$ în ζ' a. i. $\zeta \neq \zeta'$

$\zeta_{\zeta'(i)} > \zeta_{\zeta'(j)}$

• Fie ζ'' permutarea ζ' în care interzchimbăm $\zeta'(i)$ cu $\zeta'(j)$

$TMS(\zeta') = \underline{\underline{i \zeta_{\zeta'(1)} + (n-1) \zeta_{\zeta'(2)} + \dots + (n-i+s) \zeta_{\zeta'(i)} + \dots + (n-j+s) \zeta_{\zeta'(j)}} + \dots + \underline{\underline{n \zeta_{\zeta'(n)}}}$

$$TMS(\sigma'') = \frac{n t_{\sigma''(1)} + (n-1) t_{\sigma''(2)} + \dots + (n-i+1) t_{\sigma''(i)} + \dots + (n-j+1) t_{\sigma''(j)} + \dots + t_{\sigma''(n)}}{n}$$

$$TMS(\sigma') - TMS(\sigma'') = \frac{(n-i+1)(t_{\sigma'(i)} - t_{\sigma''(i)})}{n} - \frac{(n-j+1)(t_{\sigma'(j)} - t_{\sigma''(j)})}{n} \\ = \frac{(n-i+1-n+j-1)(t_{\sigma'(i)} - t_{\sigma''(j)})}{n} = \frac{(j-i)(t_{\sigma'(i)} - t_{\sigma''(j)})}{n} > 0$$

$\Rightarrow TMS(\sigma') > TMS(\sigma'')$ - contradicție

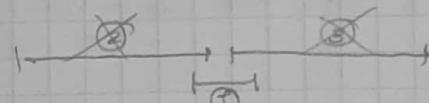
Pb. 2: Programarea spectacolelor (job scheduling)

n spectacole $\rightarrow [t_{i_k}, t_{f_k}]$

i său de spectacole

Să se programare un număr maxim de spectacole care nu se suprapun

a) după durată 1 - cale
b) după ora initială 1 - cale



c) după ora de sfârșit \uparrow OK

Alg.:

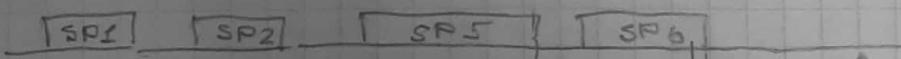
$O(n \log_2 n)$

1) se sorteară spectacole crescător după ora de sfârșit

2) Se planifică primul spectacol

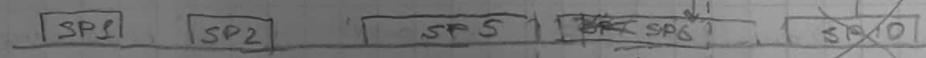
3) Se parcurg spectacolele rămase și se programă spectacolele care nu se suprapun cu ultimul spect. programat

Sol. alg.
Greedy



m

Sol optimă



n

Exchange
(SWAP)

argument

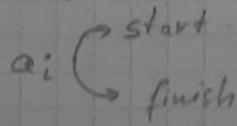
se termină
cel puțin în
același timp cu SP6

$m > n$ (absurd) $\Rightarrow m = n$
 $m < n$ (absurd)



SEMINAR

- Fie $A = \{a_1, a_2, \dots, a_n\}$ - o mulțime de activități.
 Să se programă toate aceste activități într-un număr minim de săli



$$f: A \rightarrow DCN$$

Definim adincuirea ca funcții

(A) Sortez aleașă ora de start: $O(n \log n)$

$$S = \emptyset; nrs = 0;$$

pt fiecare x din A

je s-sala care îi eliberează cel mai devreme dacă și poate fi programată în x astunci

$$S = S \cup f(x, S)$$

actualizare ora de final a lui s

altfel

$$S = S \cup \{v, nrs\}$$

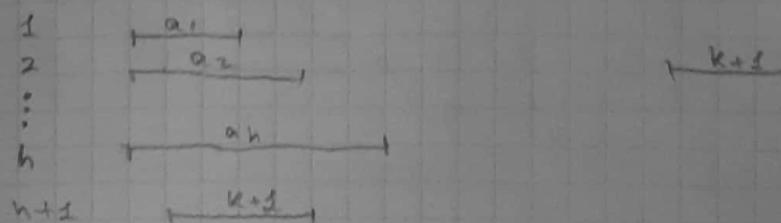
$$A = \{I(2, 7); II(3, 5); III(4, 9); IV(6, 8); V(7, 10), VI(10, 15), VII(11, 13)\}, \dots, n$$

$$nrs = 0$$

	ora de final	S	$\frac{9}{3} \frac{13}{2} \frac{15}{1}$
1: I(2, 7)	(10, 15)		
2: II(3, 5)	(6, 8)	S	
3: III(4, 9)			

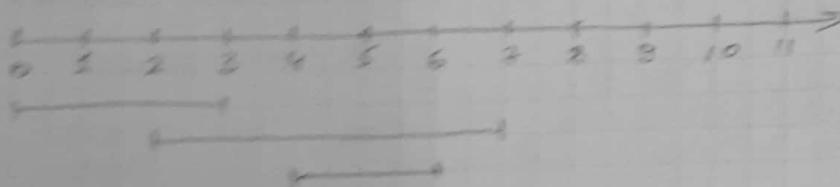
$$P_K \rightarrow P_{K+1}$$

Presupunem că algoritmul a programat optim primele K activități



- Problema căutării

Să dă o mulțime de căutare reprezentată ca un interval (pecare să numere) către care se adaugă un număr număr de căte.



- ② Selecție secundară - după capătul din stânga O(log n)
cui x -> e cel mai din dreapta eui
x <= x^l
x > x^r

x -> dreapta x -> stânga x -> eui
x -> dreapta x -> eui
x -> stânga x -> eui

x -> stânga > eui
-> sprijin (x, dr)
eui > x, dr

alt fel:

x -> dreapta > eui
x -> sprijin (x, dr)
x -> stânga (x, dr)
eui > x, dr

- Problema

Pie C -> mulțime de cuburi

Cuf(x₁, x₂, ..., x_n) = prop cā $\exists R_i^j, K_j^i$
cu $e_i = e_j$

$$\begin{matrix} \hookrightarrow e_i \\ \forall i \end{matrix}$$

Fie G turnul construit de algoritm și O - turnul obținut maxim pe O $\neq G$

CURS

TEMA: Metoda Greedy

- ① Programearea a n spectacole folosind un număr minim de săli.

08:20 - 09:50	Sp. 1
10:00 - 11:20	Sp. 2
09:30 - 12:10	Sp. 3
12:10 - 13:10	Sp. 4
11:15 - 13:15	Sp. 5
11:30 - 14:00	Sp. 6
15:00 - 15:30	Sp. 7

Optim - după ora de început crescător

Sp. 1, Sp. 3, Sp. 2, Sp. 5, Sp. 6, Sp. 4, Sp. 7
implementare optimă: $O(n \log_2 n)$

Coadă cu priorități
 \uparrow (Priority Queue)

$O(\log_2 n)$ $\begin{cases} \text{extragere min} \\ \text{inserare} \\ \text{ștergere} \end{cases}$

Cheia = ora de sfârșit al ultimului spectacol programat.

Bună corectitudinii:

- Obs.
- 1) Pentru un sir de intervale definim adâncimea sa (h) ca fiind nr. max. de spectacole care se suprapun într-un moment oarecare de timp.
 - 2) Orică progr. corectă utilizează cel puțin h săli
 - 3) Alg. Greedy folosește pt. un sir de intervale cu adâncimea h de mult h săli



Alg. Greedy este optim

② Problema rucsacului

n obiecte \rightarrow căstig c_i (integral)
 \rightarrow greutatea g_i

rucsac $\rightarrow G$ Kg

Trebuie să gasim modalitatea de încărcare a rucsacului astfel încât căstigul să fie maxim

Variante \rightarrow Varianta discrète: obiectele nu pot fi tăiate
 continuu (fracționar) în care obiectele pot fi tăiate

$$\text{Căstigul unitar} = c_i/g_i$$

Alg. Greedy:

- Observații
- 1) Suntură crescător după căstigul unitar
 - 2) Pt. fiecare obiect verificăm dacă poate fi înălțat complet sau nu în rucsac:
 - a) dacă $b^k \Rightarrow$ încărcăm și actualizăm capacitatea liberă din rucsac
 - b) dacă $N \Rightarrow$ "tăiem" ob. a. z. rucsacul să fie înălțat la max și ne sprijin

Dem. corectitudinei:

Pp. că obiectele sunt sortate crescător după căstigul unitar $\Rightarrow \frac{c_1}{g_1} \geq \frac{c_2}{g_2} \geq \frac{c_3}{g_3} \dots \geq \frac{c_n}{g_n}$
 și $g_1 + g_2 + \dots + g_n > G \Rightarrow X \neq (1, 1, \dots, 1)$

Sol. Greedy: $X = (x_1, x_2, \dots, x_n)$ $x_j \in [0, 1]$

$$\sum x_i g_i = G$$

Sol. optimă: $Y = (y_1, y_2, \dots, y_n)$ $\sum y_i g_i = G$

$$\text{și } \sum y_i \cdot c_i = \max$$

Presupunem că $Y \neq X \Rightarrow \exists$ un indice k minim a. z. $x_k \neq y_k$

- Ob:
- 1) $k \leq j$ (altfel $\Rightarrow \sum y_i g_i > G$ pt că $y_k \neq 0$)
 - 2) $y_k < x_k$
 - a) $k < j \Rightarrow x_k = 1 \quad \left\{ \begin{array}{l} y_k = 0 \\ y_k + x_{j+1} \end{array} \right.$

$$b) K=j \text{ dacă } x_K = x_j \Rightarrow \sum y_i \cdot g_i > G$$

Rie $Z = (z_1, z_2, \dots, z_{K-1}, z_K, z_{K+1}, \dots, z_n)$ o soluție astfel încât $\sum z_i \cdot g_i > G \Rightarrow \sum z_i \cdot g_i = \sum y_i \cdot g_i$

✓ CURS

TEMĂ: Metoda Devide et Impera

C1: Problema inițială se poate descompune, în mod repetat, în două subprobleme cu același tip și dimensiuni aproximativ egale (DIVIDE)

C2: Soluția unei probleme se poate obține soluțiile subproblemelor în care au fost descompuse (IMPERA)

Ex. : Suma elementelor dintr-o listă

$$\begin{aligned} [5, 1, 3, 7, 2] &= 18 \\ [5, 1, 3] &= 9 \quad [7, 2] = 9 \\ [5, 1] &= 6 \quad [3] \quad [7] \quad [2] \\ [5] & \quad [1] \quad 3 \quad 7 \quad 2 \\ 5 & \quad 1 \end{aligned}$$

```
def suma(ls):
    if len(ls) == 1:
        prob. olv O(1) → return ls[0]
    rezolv. mij = len(ls) // 2
    divide- sst = suma(ls[:mij])
    sdn = suma(ls[mij:])
    impera - return sst + sdn
```

- Complexitatea:

- a) prin ghicire complexitate + inducție matematică
- b) metoda substituțiilor repetate
- c) teorema master
- d) teorema Akra - Bazzi

$$\text{Ex: } T(n) = \begin{cases} O(1), \text{ d.c. } n=1 \\ 2 \cdot T\left(\frac{n}{2}\right) + 2, \text{ d.c. } n \geq 2 \end{cases}$$

dimensiunile olațelor de intrare

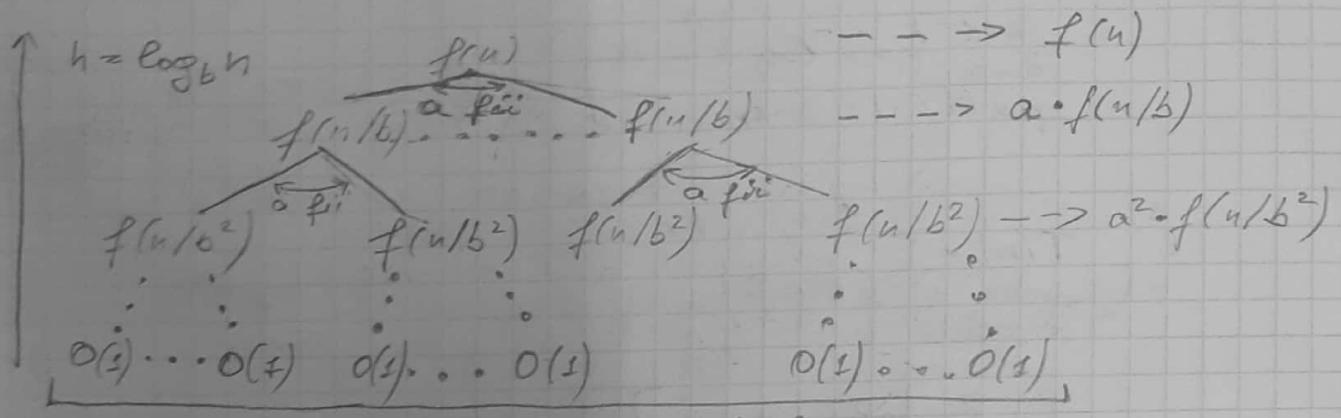
$$(b) Pp. n=2^k$$

$$\begin{aligned} T(n) &= T(2^k) = 2 \cdot T(2^{k-1}) + 2 = 2 \cdot [2 \cdot T(2^{k-2}) + 2] + 2 = \\ &= 2^2 \cdot T(2^{k-2}) + 2^2 + 2 = 2^2 [2T(2^{k-3}) + 2] + 2^2 + 2 = \\ &= \dots = 2^k \cdot T(2^0) + 2^k + \dots + 2 = 2^k + 2(2^{k-1} + \dots + 1) = \\ &= 2^k + 2(2^k - 1) = n + 2(n-1) = 3n - 2 \Rightarrow \boxed{T(n) = O(n)} \end{aligned}$$

© Teoreme master

$$T(n) = \begin{cases} O(1); \text{ pt. } n \leq d \\ a \cdot T\left(\frac{n}{b}\right) + f(n), \text{ pt. } n > d \end{cases}$$

numărul de subproblemeelor \uparrow complexitatea divizării și reconstituiri soluției
 dimensione dateelor ale intravei ale unei probleme



Nr. de frunze: $a^h = a^{\log_b n} = n^{\log_b a}$

$$T(n) = \underbrace{\sum_{i=0}^{\log_b n - 1} a^i \cdot f\left(\frac{n}{b^i}\right)}_{\text{Templu necesar divizarea problemei și reconstituirea soluției}} + \underbrace{n^{\log_b a} \cdot O(1)}_{\text{Templu necesar rezolvării problemelor direct}}$$

Templu necesar divizarea problemei și reconstituirea soluției | Templu necesar rezolvării problemelor direct

Teorema master: Fie relația obișnuită:

$$T(n) = \begin{cases} O(1), \text{ pt. } n \leq d \\ a \cdot T\left(\frac{n}{b}\right) + f(n), \text{ pt. } n > d \end{cases}$$

unde $a \geq 1$, $b > 1$, $f(n) \geq 0$, $f(n) = K \cdot n^p$

Pp. $f(n) \in O(n^p)$

a) Dacă $p < \log_b a \Rightarrow T(n) = O(n^{\log_b a})$ - domino frunze

b) Dacă $p = \log_b a \Rightarrow T(n) = O(n^{\log_b a} \cdot \log_2 n)$

c) Dacă $p > \log_b a \rightarrow T(n) = O(f(n))$

$\exists K < 1$ a.d.: $a \cdot f\left(\frac{n}{b}\right) \leq K \cdot f(n)$, pt. $\forall n \leq b$

Ex:

1) Sume elementelor dintr-o listă

$$\left. \begin{array}{l} a=2 \\ b=2 \\ p=0 \end{array} \right\} \Rightarrow \log_6 a = 1 \quad \left. \begin{array}{l} \\ \\ p=0 \end{array} \right\} \Rightarrow p < \log_6 a \Rightarrow T(n) = O(n^2) = O(n)$$

2) Căutare binară

$$T(n) = \begin{cases} O(1), & n=0 \\ T\left(\frac{n}{2}\right) + 1, & n \geq 1 \end{cases}$$

$$\left. \begin{array}{l} a=1 \\ b=2 \\ p=0 \end{array} \right\} \Rightarrow \log_6 a = 0 \quad \left. \begin{array}{l} \\ \\ p=0 \end{array} \right\} \Rightarrow \log_6 a = p \Rightarrow T(n) = O(n^0 \cdot \log_2 n) = O(\log_2 n)$$

$$3) T(n) = \begin{cases} O(1), & n \leq 1 \\ 2T\left(\frac{n}{2}\right) + n^2, & n \geq 2 \\ f(n) \end{cases}$$

$$\left. \begin{array}{l} a=2 \\ b=2 \\ p=2 \end{array} \right\} \Rightarrow \log_6 a = 1 \quad \left. \begin{array}{l} \\ \\ p=2 \end{array} \right\} \Rightarrow p > \log_6 a$$

$$\left. \begin{array}{l} \exists K \in \mathbb{R} \text{ a.i. } a \cdot f\left(\frac{n}{2}\right) \leq K \cdot f(n), \forall n \geq 6 \\ \exists K \in \mathbb{R} \text{ a.i. } 2 \cdot \left(\frac{n}{2}\right)^2 \leq K \cdot n^2, \forall n \geq 2 \\ \frac{n^2}{2} \leq K \cdot n^2 \Rightarrow n^2 \leq 2K \cdot n^2 \Rightarrow K \geq \frac{1}{2} \end{array} \right\} \Rightarrow$$

$$\Rightarrow T(n) = O(f(n)) = O(n^2)$$

$$\frac{f(n)}{n^{\log_6 a}} = \text{polinom}$$

Ex. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log_2 n}$

$$f(n) = \frac{n}{\log_2 n} \cdot \frac{1}{n} = \frac{1}{\log_2 n}$$

② Sortarea prin interclasare (MERGE SORT)

John von Neumann - 1945

```
def interclasare (lst, ldr):
    i = j = 0
    rez = []
    while i < len(lst) and j < len(ldr):
        if lst[i] <= ldr[j]:
            rez.append(lst[i])
            i += 1
        else:
```

```
            rez.append(ldr[j])
            j += 1
    if i < len(lst):
        rez.extend(lst[i:])
    if j < len(ldr):
        rez.extend(ldr[j:])
```

lst = [1, 7, 10, 15]
ldr = [2, 3, 9]
rez = [1, 2, 3, 7, 8, 10, 15]

```

↓ psd;
    rez.append(ldr[i])
    j += 1
    rez.extend(lst[i:j])
    rez.extend(ldr[j:])
return rez

```

def mergesort(ls):

- if** len(ls) <= 1:
- return** ls

 $O(s) \leftarrow m_{ij} = \text{len}(ls) // 2$
 $T(n/2) \leftarrow ls_l = \text{mergesort}(ls[:m_{ij}])$
 $T(n/2) \leftarrow ls_r = \text{mergesort}(ls[m_{ij}:])$
 $O(n) \leftarrow \text{return intercalare}(ls_l, ls_r)$

Complexitate:

$$T(n) = \begin{cases} O(s), & \text{nece} \\ 2T(n/2) + n, & \text{d.c. } n \geq 2 \end{cases}$$

$$\begin{matrix} a = 2 \\ b = 2 \\ p = 1 \end{matrix} \Rightarrow \log_b a = \epsilon \Rightarrow \log_b a = p \Rightarrow T(n) = O(n^{\epsilon} \cdot \log_2 n) = O(n \log_2 n)$$

② CURS

TEMA: METODA DIVIDE ET IMPERA

① Selectia celui de-al K-lea minim (quickselect)

$$A = [10, 7, 25, 4, 3, 4, 9, 12] \\ K = 5 \Rightarrow x = 9$$

Aleg pivot = 7 \Rightarrow
 aleator $L = [4, 3, 4,$
 $E = [7, 7]$
 $G = [10, 25, 9, 12]$

import random

def quickselect(A, K, f-pivot = random.choice):

$$pivot = f-pivot(A)$$

$$L = [x \text{ for } x \in A \text{ if } x < \text{pivot}]$$

$$E = [x \text{ for } x \in A \text{ if } x = \text{pivot}]$$

$$G = [x \text{ for } x \in A \text{ if } x > \text{pivot}]$$

$O(n)$
 obicei

if $K < \text{len}(L)$:

return quickselect(L, K, f-pivot)

elif $K < \text{len}(L) + \text{len}(E)$:

return E[0]

else:

return quickselect(G, K - len(L) - len(E), f-pivot)

pivot bun \Rightarrow imparte lista A în două subliste
 $\left\langle \leq \right\rangle$ cu $\approx n/2$ elemente

$$T(n) = \begin{cases} O(n), & n=1 \\ T\left(\frac{n}{2}\right) + n, & n \geq 2 \end{cases}$$

Th master : $\left. \begin{array}{l} c=1 \\ b=2 \end{array} \right\} \Rightarrow \log_2 a \geq 0 \quad \left. \begin{array}{l} c=1 \end{array} \right\} \Rightarrow c > \log_2 a$

$$\exists K \leq 1 \text{ s.t. } a \cdot f\left(\frac{n}{b}\right) \leq K \cdot f(n) \Rightarrow \frac{n}{2} \leq K \cdot n \Rightarrow \\ \Rightarrow n \leq 2Kn \Rightarrow K \geq \frac{1}{2}$$

$$O(f(n)) = O(n)$$

CLRS

Algoritmul BPPRT (1973) - mediana medianelor
 Blum - Floyd - Pratt - Rivest - Tarjan

$$A = [\underbrace{L}_{\text{pivot}} \quad \underbrace{G}_{\text{pivot}}] \rightarrow n \text{ elem.} \\ \min \cdot \frac{3n}{10} \quad \max \cdot \frac{7n}{10}$$

$$L = [5, 10, 15, 20, 51, \dots, 8, 10, 13, 28, 40]$$

2	7	8	5	19
3	10	10	10	21
10	12	13	15	40
14	28	28	20	50
15	35	40	51	61
L_1	L_2	L_3	L_4	L_5

$$\text{Nr. elem.} < \text{pivot} ; \\ 3 \left(\frac{1}{2} \left[\frac{n}{5} \right] \right) \approx \frac{3n}{10}$$

$$\text{Nr. elem.} > \text{pivot} ; \\ \approx \frac{3n}{10}$$

$$T(n) = \underbrace{T\left(\frac{n}{5}\right)}_{\substack{\text{quickselect} \\ \text{dăt, mediane} \\ \text{medianelor}}} + \underbrace{T\left(\frac{3n}{10}\right)}_{\substack{\text{selectie}}} + O(n) \Rightarrow \\ \text{sabluță}$$

$$\Rightarrow T(n) \leq 10 \cdot c \cdot n \\ \text{cu} \\ O(n)$$

oare pivot - mediana(A):

```
if len(A) <= 5:  
    return sorted(A)[len(A)//2]  
subliste = [sorted(A[i:i+5]) for i in range(0,  
len(A), 5)]  
mediane = [sl[len(sl)//2] for sl in subliste]  
return pivot_mediana(mediane)
```

Apel: r=quickselect(A, K, pivot_mediana)

② Problema nucacului (van, continuă) în $O(n)$

Obiecte = $\left[\frac{63}{21}, \frac{10}{10}, \frac{30}{10}, \frac{100}{25}, \frac{19}{38} \right] \cup \frac{14}{7}, \frac{50}{5}, \frac{40}{10}, \frac{20}{5} \right]$
 \Rightarrow cîştigătoare
 \Rightarrow gresită

$G = 50$

$\left[\frac{19}{38}, \frac{10}{10}, \boxed{\frac{63}{21}}, \frac{30}{10}, \frac{100}{25} \right] \quad \parallel \quad \Rightarrow \text{pivot} = \frac{20}{5}$

$L = \left[\frac{63}{10}, \frac{10}{10}, \frac{30}{10}, \frac{19}{38}, \frac{14}{7} \right]$

$E = \left[\frac{100}{25}, \frac{40}{10}, \frac{20}{5} \right]$

$H = \left[\frac{50}{5} \right]$
(higher)

Caz 1: suma gr. obiectelor din $H \geq G \Rightarrow$ se apelăm pe H și G

Caz 2: suma gr. obiectelor din $H \leq G \Rightarrow$
 $\begin{cases} \text{căutare în soluție} \\ \text{toate obiectele din } H \\ \text{combinări la maxim} \\ \text{cu obiecte din } E \end{cases}$

doar mai avem loc
în neregă repetații
pt L și G nemax

TEMA: METODA BACKTRACKING

D. H. Lehmer ≈ 1950

Biștă $A_1, A_2, \dots, A_n \neq \emptyset$. Să se genereze toate tipurile

$X = (x_1, x_2, \dots, x_n)$ care verifică o proprietate P .

$$\begin{matrix} \cap & \cap & \cap \\ A_1 & A_2 & A_3 \end{matrix}$$

Ex: Generarea permutărilor

$A = \{1, 2, \dots, n\} \Rightarrow n!$ permutări

$$n=5$$

$$\begin{array}{r} 3125 = 5^5 \\ 120 = 5! \\ \hline 3005 \end{array} \quad \left[\begin{array}{l} (1, 1, 1, 1, 1) \\ (1, 1, 1, 1, 2) \\ \vdots \\ (5, 5, 5, 5, 5) \end{array} \right] \quad \begin{array}{l} \text{Părțea brută} \\ \text{Căutare exhaustivă} \end{array}$$

|
lupluri inutile

$$\begin{array}{c} (1, 2, 3, 4, 5) \\ \swarrow \searrow \quad \swarrow \searrow \\ 2, 1, 3, 4, 5 \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 1, 2, 3, 5, 4 \end{array} \quad X = \left(\begin{array}{c} \xrightarrow{\quad} \\ \downarrow \quad \downarrow \quad \cdots \quad \downarrow \end{array} \right)$$

$$|A_1 \times A_2 \times \dots \times A_n| = |A_1| \cdot |A_2| \cdot \dots \cdot |A_n| \geq 2^n$$

Algoritmici:

a) $X = (x_1, x_2, \dots, x_k)$
soluție parțială

$X = \text{soluție } (k=n)$

$\Rightarrow X \neq \text{soluție} \Rightarrow$ adaug $x_{k+1} \Rightarrow$
 $\Rightarrow X' = (x_1, x_2, \dots, x_k, x_{k+1})$

b) $X = (x_1, x_2, \dots, x_{k-1}, x_k)$
nu este soluție
parțială

\exists o val. necorespunzătoare
pt. $x_k \Rightarrow x_k$ se
inzolvă și se
succesionează
lui x_k

\nexists o val. necorespunzătoare
pt. $x_k \Rightarrow$ eliminăm $x_k \Rightarrow$
 $\Rightarrow X' = (x_1, x_2, \dots, x_{k-1})$

Backtracking template → pozitia componentei curente

def bkt(k):

global X

for v in range(prin_k, ultim_n+1):

X[k] = v

if soluție-partială (X[:k+1]):

if soluție (X[:k+1]):

prelucrare (X[:k+1])

else

bkt(k+1)

② Generarea permutărilor

def bkt(k):

global X, n

for v in range(1, n+1):

X[k] = v

if v not in X[:k]:

if k == n-1:

print (X[:k+1])

else

bkt(k+1)

n = int(input("n = "))

X = [0]*n

bkt(0)

def bkt():

global X, n

for v in range(1, n+1):

X.append(v)

if v not in X:

if len(X) == n-1:

print(X)

else

X.pop()

bkt()

n = int(input("n = "))

X = []

bkt()

③ Generarea aranjamentelor

$$(1, 2, 3) \neq (2, 1, 3)$$

$$A_n^P = \frac{n!}{(n-P)!}$$

Inlocuirea K == n-1 cu K == p-1 în gen. perm.

④ Generarea combinațiilor → def bkt(k):

$$(1, 2, 3) = (2, 1, 3)$$

$$C_n^P = \frac{n!}{P!(n-P)!} = \frac{A_n^P}{P!}$$

prin
eliminarea
celor două
instrucțiuni

↓
generarea tuturor
submultimilor

global X, n, P
for v in range(1, if K == p else
X[K-1]+1, n+1):

K == p-1:

print (X[:k])

else:

bkt(k+1)

④ Pb celor n regine

tablă
de săh

	1	2	3	4
1	R			
2		R		
3	R			
4		R		

$n \times n \rightarrow n$ regine ($n = 4$)

$X[K] =$ coloana pe care să fie afisată
regina de pe linia K

1 2 3 4 \rightarrow linie

$x = [2, 4, 1, 3] \rightarrow$ coloane

1) $X[K] \in \{1, 2, \dots, n\}$

2) pt. soluția parfaite $\begin{cases} a) & (pe linie) \\ b) X[K] \neq X[i], \forall 0 \leq i \leq K \text{ (pe coloane)} \\ c) |X[K] - X[i]| \neq |K - i| \forall 0 \leq i \leq K \text{ (pe coloane)} \end{cases}$

$$m_{R_K R_i} = \pm 1$$

$$m_{R_K R_i} = \frac{X[K] - X[i]}{K - i}$$

3) $K = n - 1$

⑤ Descompunerea unui număr natural ca sumă de numere naturale nenule

$$\begin{aligned} n &= 4 \\ 1 + 1 + 1 + 1 \\ 1 + 1 + 2 \\ 1 + 2 + 1 \\ 2 + 1 + 1 \\ 1 + 2 \\ 3 + 1 \end{aligned}$$

$X[K] =$ un termen al sumei

$\Rightarrow X[K] \in \{1, 2, \dots, n - K + 1\}$

2) $X[0] + \dots + X[K] \leq n \rightarrow$ sol. parfaite

3) $X[0] + \dots + X[K] = n \rightarrow$ soluție

def $bkt(K)$:

global X, n
for i in rang ($2, n - K + 2$):

$X[K] = i$

sunt = suma ($X[i : K+1]$)

if sunt $\leq n$:

if sunt == n:

print ($X[i : K+1]$)

else:

$bkt(K+1)$

⑥ se N

Toate numerele naturale cu cifre distincte care
au suma cifrelor = 5

$$s = 3$$

$$102$$

$$12$$

$$102$$

$$3$$

$$30$$

a) Sol. are max 10 componente

b) $0 \leq s \leq 45$

$X[K] = 0$ altă $\Rightarrow X[K] \in \{1, 2, \dots, 9\}; K \geq 1$
 $\{1, 2, \dots, 9\}; K = 0$

- 2) a) $x[i] \neq x[j], \forall 0 \leq i < j$
 b) $x[0] + \dots + x[K] \leq s$
 c) $x[0] + \dots + x[K] = s$

⑦ Pb. colorării hărților (cu max. 4 culori)

8) CLIRS

TEMA: Metoda programării dinamice (PD)

Richard Bellman (≈ 1950)

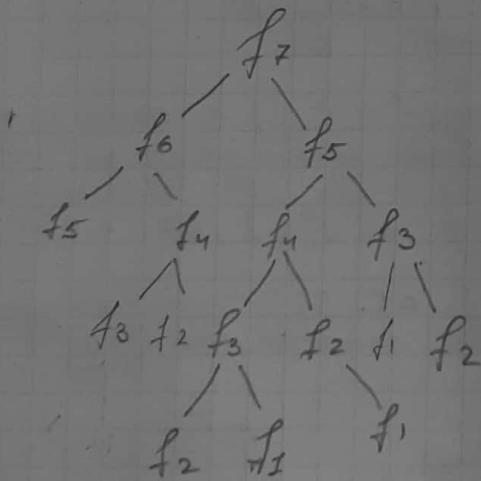
Se aplică în probleme de optimizare.

Cand.:

C1: Problema se descompune în două sau mai multe probleme de același tip, iar soluția să se obțină combinând soluțiile subproblemelor

(optimal global se obține combinând optime locale)

C2: Subproblemele se suprapun!

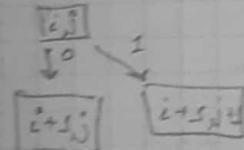
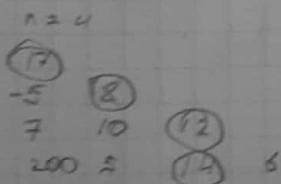


MEMOIZARE

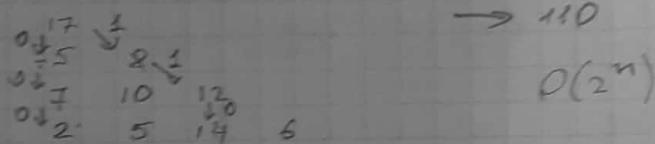
= memorarea soluțiilor subproblemelor într-o structură de date corespunzătoare pentru a nu a trebui recalculată de fiecare dată

OD → Găsirea unei relații de recurență prin căre optimul global se obține prin optimale locale
 Implementarea relației de recurență într-un mod recursiv folosind memoizarea.

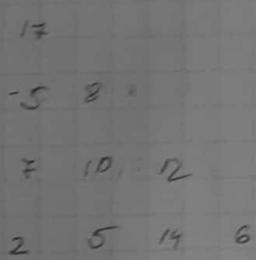
Ex Sume maxime dintr-un triunghi de numere întregi



Greedy: Aleg maximul dintre $[i+1, j]$ și $[i+1, j+1]$ ⇒ rezultatul este incoracționat.



Backtracking: Generez toate drumurile corecte și le refin pe cel cu suma maximă ⇒ este corectă, dar ineficientă



def smax(a, i, j) = suma max. pe un drum care începe cu $a[i][j]$

$$\text{smax} = \begin{cases} a[i][j], & \text{dă } i=n-1 \\ a[i][j] + \max\{ \text{smax}(a, i+1, j), \text{smax}(a, i+1, j+1) \}, & \text{dă } i < n-1 \end{cases}$$

Dezvoltăți imperativ ⇒

Backtracking $O(2^n)$

rezultat corectă, dar ineficientă

Memoizare!

def smax(a, i, j) $\Leftrightarrow \text{smax}[i][j]$

$$\text{smax}[i][j] = \begin{cases} a[i][j], & \text{d.c. } i = n-1 \\ a[i][j] + \max\{\text{smax}[i+1][j], \\ \text{smax}[i+1][j-1]\}, & \text{d.c. } i < n-1 \end{cases}$$

	17		
a	-5	8	
	7	10	12
	2	5	14 6

51							
smax	19	34					
	12	24	26				
	<u>2</u>	<u>5</u>	14	6			

Varianta II

	17		
a	-5	8	
	7	10	12
	2	5	14 6

17			
smax	12	25	
	19	35	37
	22	40	51 43

Complexitate: $O(n^2)$

Sol: maximul obținut pe ultima linie

Variantele PD $\begin{cases} \text{Inainte} & - \text{var. 1} \\ \text{Inapoi} & - \text{var. 2} \\ \text{mixta} & \end{cases}$

Plata unei sume folosind un număr minim de monede

$$\begin{matrix} S = 14 \\ d = [5, 3, 2] \end{matrix}$$

nrmin[i] - numărul minim de monede pt. a plăti sumă i

$$\begin{matrix} \text{nrmin}[14] = 1 + \text{nrmin}[14-5] \\ \uparrow \\ 6 \end{matrix}$$

$$\min[i] = \begin{cases} 0, & \text{da. } i=0 \\ 1 + \min_{j \in \{0, 1, \dots, m\} - \{i\}} \left[\text{min formula } [i-j, w_j] \mid i \geq w[j] \right], & \text{sonst} \end{cases}$$

\min	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	1	2	1	2	2	2	3	2	3	3	3	3	4	
1	2	3	2	5	3	5	5	5	5	5	5	5	5	5	

$$\min[2] = 2 + \min[2-2] = 1$$

$O(n \cdot s)$

$$\min[3] = 1 + \min[3-1]$$