

DIMA OANA TEODORA IUA IAN 2020
INTRODUCERE

- **Variabile** sunt văzute ca **pointerii**

$a = 2 \quad a \rightarrow [2] \leftarrow b$

$a = 1 \quad a \rightarrow [3]$

$b = 2$

- **Afinge**: ptint($\&$)

→ **implicit** are **ends**

→ putem pune **separatori**

ptint (**obiecte**, sep = "**"**", end = "**";**")

→ **format output**

ex: $x = 5$

$y = 6$

ptint ("Valoarea lui x este **39**, iar a lui y este **39**".
, format (x, y))

⇒ Valoarea lui x este 5, iar a lui y este 6

ptint ("Ana are **30** și **21**". format ("**Ana**", "**pere**"))

⇒ Ana are metri și pere

ptint ("Ana are **219** și **30**". format ("**metri**", "**pere**"))

⇒ Ana are pere și metri

- **Citire**: $x = \text{input} ("menaj")$

→ **implicit** returnată **string**

$x = \text{int} (\text{input}()) \mid \text{float} (\text{input}())$

- **Operatori și operări**: (aritmatici)

- $a \% b$

- a / b → împărțire **REALĂ**

- $a // b$ → împărțire **INTREAGĂ**

- $a ** b$ → ridicare la putere

$a ** 0,5$ → radical \sqrt{a} (ord 2)

- **operatori de identitate**:

$x \in y()$

(x) $x \in \text{not } y$

- operatorii membership: \in im y
- \in not im y
- operatorii logici: and or

- Transfomarea din mai multe baze:

ex: $a = \boxed{101}1010$

prefixe

pt baza 2 $\Rightarrow 1010$ binar

$c = \boxed{001}310$

postfix

pt baza 8 $\Rightarrow 310$ octal

$d = \boxed{0\times12}c$

(baza 16)

print (d) $\Rightarrow 300$

• Există și: $x = (1 = \text{True}) \Rightarrow x = \text{True}$

$x = (\text{}) \Rightarrow x = (\text{1} = \text{False}) \Rightarrow x = \text{False}$

$x = \text{None}$ (nu este cheie)

$x = y = 2 = 3$

• Commentarii - linie # bloc ...

• Structuri pe același număr []

• Convenții:

* structură \rightarrow int în baza b:

$x = \text{int } (s, b) / \text{int } (s)$

* int \rightarrow structură

$s = \text{str } (x)$

• Există și clasa COMPLEX: $z = a + bi$

• Returnarea tipului clasa variabilelor
type (a)

• Returnarea adresa: id (a)

- Functia : range (n) $\rightarrow 0, 1, \dots, n-1$
range (a, n) $\rightarrow a, a+1, \dots, n-1$
range (a, n, pas) $\rightarrow a, a+pas, \dots, n-1$ (pas)
 - Iată lucruri \downarrow break (iese din înțeles curată)
continuă
pas \downarrow
continuă fără să execute ce este după el
 - Operatorul de UNPACKING:  / 

ex: print (range(4)) \Rightarrow range(4)

print(*range(4)) => 0 1 2 3
l = [1, 2]

print (d) , => [1,2]

print (*l) => N2

plant change (\times) \Rightarrow 1

• for cu else ex: $d = [0, 1, 5]$

White

for x in d : print(x) ; else :

Cifra

```
f = open ("input.txt", "r")
```

`print("X")
=> 0:1:5:X`

```
f = open ("input.txt", "r")
```

`new() → tot figielue + fclose() ↳ "xit" ("r+`

P = f.read() I.readline() → total lines

→ Atenție orice cîtărea se face pe hîmii:

(biex): input(text) for $x \in f$:

$a = \text{int}(\infty, 10)$

Afisrati:

`g = open("output.txt", "w")`

g. xi kute (cutting) → me pure implicit, and
↪ s + "me" (eventual)

CARACTERISTICI STRUCTURII DE DATE

ITERABIL - pot fi parcurse element cu element
- folosim for în ITERABIL:

MUTABIL - poate fi modificat

ex: `list.append(x)`

IMUTABIL - nu poate fi modificat ulterior

ex: la stringuri toate funcțiile returnează ceva nou (nou obiect) nu modifică obiectul apelat
ex: `SET` ≠ `FROZENSET`

INDEXABIL - permit `if s[i] == a[i]`

STRINGURI (clasa string)

ITERABILE + IMMUTABLE

- Declarație: `" "` `' '` `+ INDEXABILE`
- len() = lungimea nr
- NU EXISTĂ tipul char ⇒ caracterele sunt stringuri de lungime 1

• Operatorul `[]`:

- folosit pt a accesa un anumit caracter / slice
- nu poate fi folosit în atribuiri (adică)

• Operatorul `==`:

- verifică dacă 2 stringuri sunt egale

(nu modifică!) METODE SPECIFICE CLASEI STRING

- crează un nou obiect (alt id)

• Returnată string:

1) `s.upper()` `s.lower()`

2) `s.title()` - prima lită a fiecărui cuvânt este transformată în literă mare

3) `s.replace (old, new, [năz.] optional)`

4) `s.strip ("ab")` - elimină de la început și la sfârșit optional

• Returnarea bool:

- 1) $s.isupper()$ * $s.isalpha()$
- 2) $s.isnumeric()$ } \rightarrow daca conține numai cifre
- 3) $s.isdigit()$ (p)
- $s.isspace()$ (s)

• Returnarea int: (specific strînguri!)

- $s.find(x, [s, [t]])$
- indicele primei poziții apărută a lui x în s
 - dacă $x \notin s \Rightarrow -1$
 - s, t - interval de căutare

• Returnarea listă:

$V = s.split([rep, [m]])$

- face în funcție de nr patrator care este nr
- m = numărul maxim de tăieturi de la stânga la dreapta

• Lucrul cu adrese:

$X1 = "Ana"$

$V = "Ana"$

$X1 \in V \Rightarrow True$

$X1 == V \Rightarrow True$

Obs: $ord(x) \Rightarrow$ cod ASCII
 $chr(ascii) \Rightarrow$ string

* Operator $\text{def concatenare a 2 rezultati} \quad V + V$
 (adică) copiere (nu se leagă adresele) $V = V$

• Operări pe mijlocii de caractere:

1) Modificare caracter aflat pe poziția p :

$S = S[:p] + \text{char_nout} + S[p+1:]$

2) Stergerea de pe poziția p :

$S = S[:p] + S[p+1:]$

Stergerea de pe ultima poziție " $\backslash n$ " (fisiere)
 poz = len(S) ($\backslash n$ este pe poz-1)

$S = S[:poz-1] + S[poz:]$

3) Modificarea tuturor aparițiilor unui subșir $\Delta = \text{S. replace}(\text{old}, \text{new})$

LUCRUL CU FELII / SLICE-URI

- Când se crează un slice se face o copie pentru întregul slice etc
 $\text{id}(\text{L}[i:j]) \neq \text{id}(\text{L})$
- $\text{L}[i:j] \rightarrow$ elementele de la i la $j-1$
- $\text{L}[i:j] \rightarrow$ de la i până la cap
- $\text{L}[i:] \rightarrow$ elementele de pe poziția i ($i \in \mathbb{Z}$)

for x in $\text{L}[k:]$:

spǎrge x

CLASA LIST (liste = vectori monomogeni)

ITERABILE + MUTABILE + INDEXABILE

- Operatorul $[]$ - accesarea unui anumit element
- Declarație: $\text{l} = []$
Declarație în completare | initializare $\text{l} = [\text{o}]^*\text{m}$
- Functia $\text{len}(\text{v})$ = lungime lista
- Creare: $\text{l} = \text{list}(\dots)$ | $\text{l} = [\text{ }]$

METODE SPECIFICE CLASEI LIST

- Modifică lista:

1) $\text{l.append}(x) \rightarrow \text{b}(1)$

2) $\text{l.extend}(\text{iterabil}) \rightarrow \text{b}(0(\text{len(iterabil)})$
- concatenează pe dreapta lui toate elementele din iterabil

3) $\text{l.insert}(i, x) \rightarrow \text{b}(k)$

- inseră pe x pe poz i

4) $\text{l.remove}(x) \rightarrow \text{b}(n)$

- sterge primă apariție a lui x

- ValueError - dacă $\exists x$

5) $\text{l.pop}([i]) \rightarrow \text{b}(k)$

- returnează și sterge elem de pe poz i

- dacă $\exists i$ returnează ultimul elem

sterge ($\text{b}(1)$)

6) `l.clear()` - $O(1)$

7) `l.append('noilea')` - $O(nlog n)$

8) `l.reverse()` - $O(n)$
- inversarea ordinii elementelor din lista

- Returnarea de la:

1) `l.index(x)` - $O(n)$

- returnarea poziției primei apariții a elementului lui x în l.

- dacă x nu \Rightarrow ValueError

2) `l.max()` \rightarrow în pe lângă `l.min()`

3) `l.copy()` - $O(n)$

- un nou obiect

+ o nouă copie

Obs: [Pentru că liste și în A este $O(n)$]

[Existe și în matrice: `list()`; $l = [\{ \}, \{ \}]$]

• LIST COMPREHENSIONS

$v = [2**x \text{ for } x \text{ in range}(10)]$

= =

$v = []$

for x in range(10):

{ $v.append(2**x)$ }

- Pentru că liste sunt MUTABILE avem:

$v = [10, 20]$

$x1 = v$: (acă) - legătură de adrese

$v.append(30) \Rightarrow \{ x1 = [10, 20, 30] \}$

$\{ v = [10, 20, 30] \}$

- Citire folosind list comprehension:

$A = (1, 2, 3)$ $S = \text{input}()$

$v = [int(x) \text{ for } x \text{ in } S \text{ if } x \in A]$

$\{ \text{print}(v) \} \quad \{ \text{True} \}$

alegorie

CLASA TUPLE (tuple)

[IMUTABIL + ITERABIL + INDEXABIL]

- Declarație | Creare : $d = (\dots)$

METODE SPECIFICE : - Returnarea listă :

- 1) $d.\text{count}(x) = O(n)$

- nr aparițiilor lui x în t

- 2) $d.\text{index}(x) = O(n)$

- căută prima apariție a lui x
în trimitere posicția ei

- dacă $\exists x \Rightarrow \text{ValueMethod}$

- Accesare tuplet : $t[i]$

CLASA DICT (dictioanar)

[MUTABIL + ITERABIL + INDEXABIL]

- Declarație | Creare : $d = \{\dots\}$

METODE SPECIFICE :

- Modificare :

- 1) $d.\text{clear}() = O(n)$

- 2) $d.\text{update}(\text{key}, \text{value}) = O(1)$

- $= d[\text{key}] = \text{value}$

- dacă nu se găsește key atunci
se înregistrează perechea (key, value)

- Returnarea listă :

- 1) $d.\text{keys}() = O(n)$

- 2) $d.\text{values}() = O(n)$

- 3) $d.\text{items}() = O(n)$

- listă formată din perechile

- 4) $d.\text{sorted}() =$ listă cu $(\text{key}, \text{value})$

$d = \{ \text{item 1}, \dots, \text{item n} \}$ IMUTABIL

Key₁: value₁ !? (nu liste)

Key₂: value₂ !?

- item 2 !? (A): $d = \{ 1 : [1, 2] \}$

prinț ($d[1, 0] \rightarrow 1$)

- Returnarea listă / set / etc.

- 1) $d.\text{get}(\text{key}) = O(1)$

= $d[\text{key}] = \text{valoarea de la key}$

CLASA SET (SET-URI și FROZENSET-URI)

- dicționar particularizat - are doar chei
- Declarație / Create - $S = \text{set}(\dots)$

METODE SPECIFICE

• Modifică :

1) $S.add(x)$

- adaugă pe x în $\text{ord}\uparrow$

2) $S.update(l)$

3) $S.remove(x)$

- dacă $x \notin S \Rightarrow \text{EROARE}$

4) $S.discard(x)$

- nu dă EROARE

5) $S.clear()$

6) $S.pop()$
- elimină și returnează un element

Random

$S.pop(x)$

- elimină și returnează x de pe pozitie

• Returnează bool:

1) $S.isdisjoint(S)$

- $S \cap S = \emptyset$

2) $S.issuperset(S)$

- $S \supseteq S$

• Returnează alt set:

1) $S.copy()$

• Operări matematice:

- se pot face doar pe set-uri care
conțin elemente întregi

1) Reuniune: $A \cup B$

num

$A.union(B)$

2) Intersecție: $A \cap B$

num

$A.intersection(B)$

3) Diferență: $A - B$

sau

A. difference (B)

4) Diferență simetrică: $A' B$

sau

A. symmetric-difference (B)

• Alte metode:

[len(s)
max(s)
min(s)
sum(s)
sorted(s)]

Obs: 1) Compararea a 2 seturi: if $a == b$;

2) FROZEN SET-URILE sunt set-uri IMUTABILE

3) Seturile sunt mult mai eficiente
decât liste

ex: $x \in \text{set}$ - O(1)

$x \in \text{list}$ - O(len(e))

FUNCȚII

Reguli:

1) La declarare parametrii se pun primii cei FĂRĂ
valoare implicită (se folosesc când nu primește altceva)
ex: def f(mumne, meny = "Hello"):

↑ int, strîng, tuple

2) Parametrii mutabili nu se modifică după apel

3) Declararea de tip global se face în funcție

4) La returnare se pot da mai multe valori

def f(list):

< ... >

return S, P | Atenție!
 $S \ni P = \text{mumne}(L)$ |
print(S, P) | t = mumne(L)
| t → tuple

5) Return - oprește funcția
YIELD [generator]
Returnarea pînă cînd nu mai
are ce

Obs: Se poate ofiza comentariul dintr-o funcție:

ex: def double (num):

"4" Functile care dublață

return $2^{\ast} x$

print (double. — dec —)

=> Function can't be duplicate

- Funcții ANONIME definite prin lambda:

Sintaxă: lambda arguments : expression
 "parametrii"

ex: double = lambda x: x * 2
print(double(5)) => 10

- combinatoră de funcție nested ()

```
L = sorted (l, key = lambda t: t[1])
```

* sortare după mai multe criterii:
 $\text{Key} = \lambda t : (t[1], t[0], \dots)$

- Funcția **filter** (functie, liniă):

ex: $\ell = [1, 5, 4, 6, 8, 11, 3, 12]$

```
L = list(filter(lambda x: (x%2==0), e))
```

```
print(L) # [4, 6, 8, 12]
```

→ returnată acela elemente din lăță pentru care funcția este adevărată

- Fumată map (fumăie, cintă):

→ futuricează elemente modificate

ex: $\mathcal{L} = [1, 3, 5]$

```
L = list(map(lambda x : 2*x, l))
```

$$\text{shift}(L) \Rightarrow [2, 6, 10]$$

MODULE

- Se pot numi:

import exemplu as e

use: e.add(parametri)

- Aparțin:

fișier: exemplu.py

import exemplu

use: exemplu.mumefunct(a,b,c)

sau

from my-module import mumefunct

use: mumefunct

COMPLETARE REGULI FUNCȚII

6) Numărul variabil de parametri:

ex: def mumef(*args):

if len(args) > 0:

for x in args:

<ptrl x>

apel: mumef(*L) / mumef(a,b,c...)

→ le consideră ca fiind variabile dif.

7) Dacă păstrăm numele parametrilor în

apel, putem să le amestecăm erilind

def mumef(a=2, b=7):

apel: mumef(b=6, a=3) ↑ a=3
b=6

8) Folosirea generatorului yield:

ex: def mumef():

<ptrl x>

yield x

apel: for x in mumef():
<ptrl x>

EXCEPTII

try :

i = <bloc 1> # bloc de evaluat

în care ar putea apărea o eroare

except FileNotFound as e

ValueError as e

ZeroDivisionError as e

Exception as e

print(e)

* Blocuri optionale :

else :

(^{nu}) <bloc 2> # se execută dacă n-am
avut eroare

(^{nu}) finally :

<bloc 3> # se execută mereu

(^{nu}) try : <bloc de evaluat>

except FileNotFoundError :

print("fișierul nu există")

except ZeroDivisionError :

print("numărul este 0")

except Exception :

print("nu m-a dat numere naturale")

Algoritm căutare binară - $O(\log_2 n)$

⇒ Vector: sortat

def caut (v, val, l=0, f=None):

if f == None:

f = len(v)

Sau am
ajuns la
2 nr con-
secutive

{ if f-s == 1:

if v[s] == val:

return s

else:

raise IndexError

else:

mid = (l+f)/2

if v[mid] == val:

return mid

elif v[mid] > val:

return caut (v, val, l, mid)

elif v[mid] < val:

return caut (v, val, mid+1, f)

L = [2, 5, 9, 12, 13, 34, 38, 55]

val = 13

print (caut (L, val))

Error: "string index out of range"

→ Repunem condiția cu len(a) prima la

Inceputul fiecărui structură care ar putea să depășească

Generare parola de forma: Abdf2799 (random)

M: from string import digits, ascii_lowercase,

ascii_uppercase

from random import choices

lista

A = choices (ascii_uppercase)

de string-uri

a = choices (ascii_lowercase, k=3)

cif = choices (digits, k=4)

parola = """.join (A+a+cif)

• Metoda: JOIN \Rightarrow returnează string

Sintaxă: string.join (listă | tuple | etc
de stringuri)

Ex: $t = ("John", "Ana", "Vero")$
 $x = "H".join(t)$
print(x) \Rightarrow "John#Ana#Vero"

• Introducere peredi de date de la tastatură
prin intermediul lui -1:

```
z = input()
while z != "-1":
    print(z)
    z = input()
```

• Funcția **SORTED()** nu poate aplica și set-urile:
list = sorted(set)

• **CREARE MATRIX** (elementele de la 0 la m^m)

```
M = int(input())
```

```
matrix = []
z = 1
```

```
for i in range(M):
```

```
    for j in range(M):
```

```
        a = []
        for k in range(M):
```

```
            a.append(z)
```

```
            z += 1
```

```
        matrix.append(a)
```

```
    matrix.append(a)
```

Afișare:

```
for i in range(M):
```

```
    for j in range(M):
```

```
        print(matrix[i][j], end=" ")
```

```
    print()
```

• CREAȚE MATRICE DE ADIACENȚĂ:

- cu n dat

- lista de muchii date

\Rightarrow matrice adiacență pt graf neorientat

$f = \text{open}("imtitare.txt", "r")$

citire m $m = \text{int}(f.readline())$

declarare m $f. a = [None] * (m+1)$

for i in range(m+1):

initializare m $a[i+1] = [0] * (m+1)$

for line in f:

l = line.split()

i = int(l[0])

j = int(l[1])

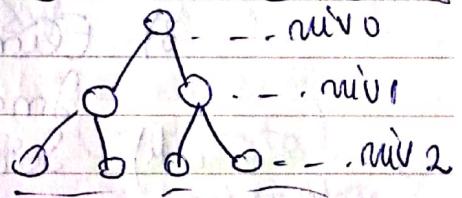
completare matrice $a[i][j] = a[j][i] = 1$

for i in range(m):

afișare / prelucrare print(a[i+1][1:m+1])

HEAP-URI

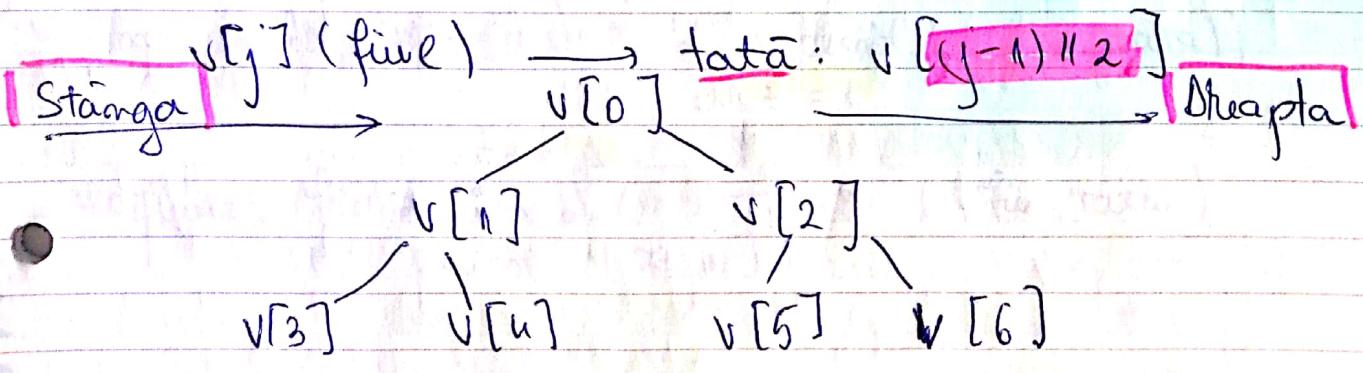
HEAP = arbore binar complet



Vector cu m elemente \Rightarrow Heap cu m noduri

$v[0]$ - radacina

$v[i]$ (tatal) \rightarrow fiul stâng: $v[2i+1]$
 \rightarrow fiul drept: $v[2i+2]$



- Max-heap = arbore binar complet
 = valoarea memorată în grice
 mod al rău \geq valorile memorate
 în modurile fizice
 $\text{val}(\text{tatāl}) \geq \text{val}(\text{f.s})$
 $\geq \text{val}(\text{f.d})$
- Min-heap = arbore binar complet
 = val memorată în grice mod
 de rău \leq valorile din modurile fizice
 $\text{val}(\text{tatāl}) \leq \text{val}(\text{f.s})$
 $\leq \text{val}(\text{f.d})$
- Îmărtîmea heap-ului = $\log_2(n)$
- Efectuare operații de bază eficient:
 $O(\log n)$
 - 1) Căutare minim im $O(1)$
 - 2) Creare structură de heap dintr-un vector oarecare $O(n)$
 - 3) Eliminare element $O(\log n)$
 Însetare element
 - 4) Sortare $O(n \log n)$

Grafuluri în python:

- 1) Matrice de adiacență (clasică)
- 2) Lintă de arce = liniță de tupluri
- 3) Lintă de adiacență = dicționar

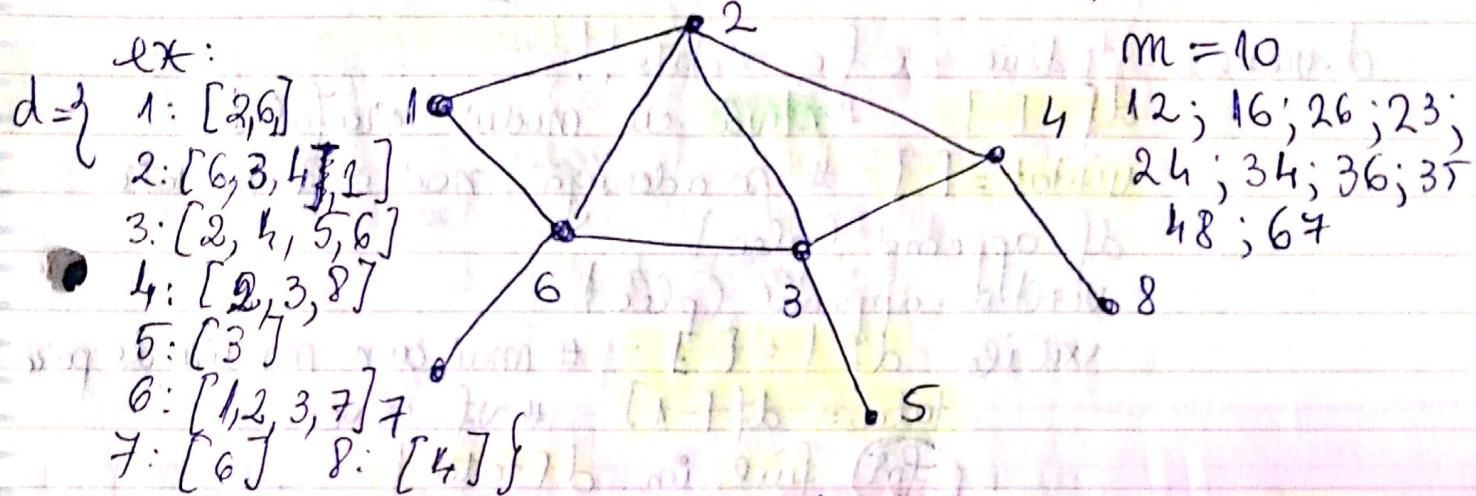
Parcurgerea grafurii:

Parcurgerea BF: (breadth first)

→ se lucrează pe lista de adiacență

→ DICTIONAR

→ am nevoie de nodul de plecare = plec



BF: Toti vecinii vizitati in ordine crescătoare:

(2) 1 3 5 6 7 8 →

plec = 2 \Rightarrow BF: 2 1 3 5 6 7 8

bf = [] # coada de tupluri (mod, tata)
sunt vizitate

bf.append((plec, -1))

nt = dn = 0

vizibile nt <= dh :

nodul curent

tata = bf[nt][0]

for fiu in d[tata]:

#daca nu a fost deja vizitat # nu vizitat if fiu mod in [mod for (mod, parinte) in bf]:

bf.append((fiu, tata))

dh += 1

nt += 1

afisare for x in bf: (fiu, tata)

print(x[0])

Parcurgerea DF (depth first)

→ se lucraza pe lista de adiacenta

→ DICTIONAR

→ am nevoie de modul de plecare
pe exemplu anterior:

DF: "Albinuta" primul vecin vizitat

plec = 2 → DF: 1 2 1 6 3 4 8 5 7
↓↓↓↓↓-----

d.values - trebuie să fie nereitate!!

df = [] # stiva cu noduri grafului

vizitat = [] # se adauga nodurile vizitate

df.append(plec)

vizitat.append(plec)

while df != []: # mai am noduri de parc

tata = df[-1] # vf stivei

for fiu in d[tata]:

if fiu nu este in vizitat:

df.append(fiu)

vizitat.append(fiu)

break

optiune căutarea

else

df.pop(-1)

în caz dim vf

stivei dacă a fost deja vizitat

afinare

for x in vizitat:

print(x)

Drumul de lungime minima

(tata, x)) într-o modură (plec, ajung)

→ folosesc lista bf rezultată în urma parc. BF:
bf = [(fiu1, tata1), (fiu2, tata2), ...]

→ ideea: merg din tata în tata

→ plec de la modul final (ajung)

→ trebuie să intr-o linte să le pot afisa corect

plec → ajung

tata = ajung # plec de la nodul de ajuns

afis = [ajung] # lista pt afisare

while tata != None:

for a in bf:

if a[0] == tata:

tata = a[1]

break

afis.append(tata)

afis.reverse()

if len(afis) == 0:

print("Nu există")

else

print(*afis)

④ Funcția anonimă LAMBDA - exemplu -

1) Se dă un vector l de cuvinte

Se cere să se sorteze după lungime și

pt cuvintele de ac lungime după ordinea alfabetica

$l = \text{sortid}(l, \text{key} = \lambda x : (\text{len}(x), x))$

2) Crescător după lungimea cuvintelor, iar pt
cuvintele de aceeasi lungime să se păstreze ordi-
nea din lista originală

$l = \text{sortid}(l, \text{key} = \lambda x : (\text{len}(x), x))$

reverse = True

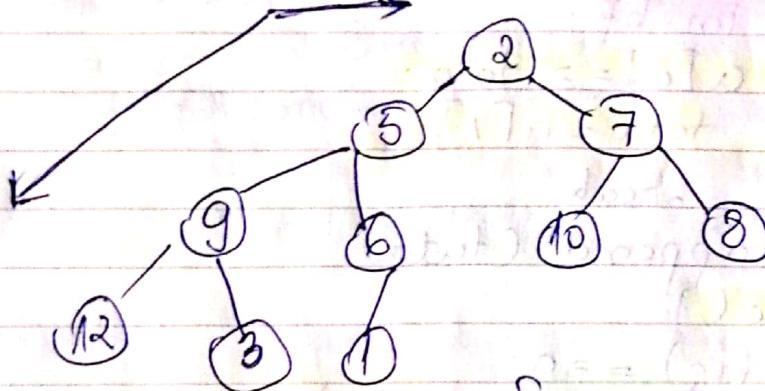
Bună ieri și în continuare

lina

(and with him) yesterday, off (ex

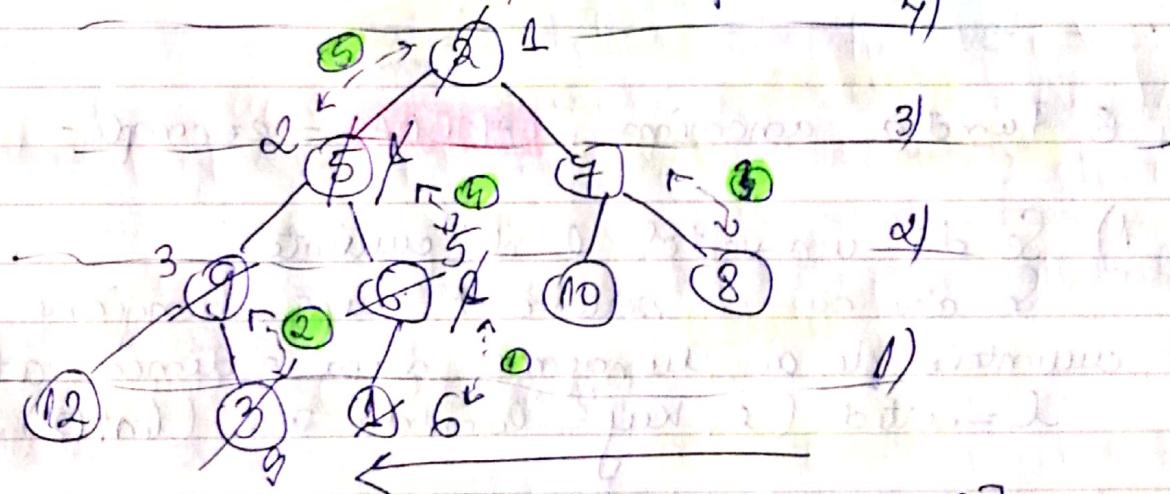
* Create heap dintr-o listă dată:

$$l = [2, 5, 7, 9, 6, 10, 8, 12, 3, 1] \quad n=10$$



Transformarea se face:

- pătrind de pe ultimul nivel
- de la dreapta la stânga
- împărțire pe mini-heapuri



$$\Rightarrow H = [1, 2, 7, 3, 5, 10, 8, 12, 9, 6]$$

Functie: import heapq as HQ
HQ.heapify(L) \rightarrow O(m)

Alte funcții:

1) HQ.heappop(L) \rightarrow O(1)

- elimină în returnată cel mai mic element

- următorul cel mai mic il ia locul

2) HQ.heappush(L, key=None)

\Rightarrow lista cu cele mai mici n elemente din L

3) HQ. heappush (L, element)

- adaugă elementul său în heap-ul

(Obs.: Metoda ne imizează | constituie polimorfism dinspre stânga; altfel ar trămașe cîștige goale în vector.)

④ Coada de prioritate:

→ pt min-heap cu prioritate mă mică

→ pt max-heap cu prioritate mă mare

COADA (DEQUE)

from collections import deque

a = [1, 2, 3, 4]

l = deque(a)

print(l) \Rightarrow deque([1, 2, 3, 4])

l.popleft() \Rightarrow deque([2, 3, 4])

l.pop() \Rightarrow deque([2, 3])

l.pop(0) \Rightarrow 2

MINIMIZAREA TIMPULUI MEDIU DE ASTEPTARE

⊗ LAB 5. P 1 min
1: 7 min

2: 7 + 15 min

3: 7 + 15 + 3 min

$l =$ lista de tuple cu nr său de ordine
în lista inițială și timpul individual de rezolvare

$l = \text{sorted}(l, \text{Key} = \lambda \text{lambda } \lambda: \lambda[1])$

def afis_t_s(tis):
 $s = 0$

for i in range(len(tis)):

 ord = tis[i][0]

 retv = tis[i][1]

 if i == 0:

 ast = tis[0][1]

 else:

 ast = ast + retv

 st = ast

 print(f"({ord}, {retv}, {ast}, sep="")

→ Sortare crescătoare după timpul de rezolvare

DIVIDE ET IMPERA

Merge Sort - Sortare prin interclasare

subprogram interclasare & returneaza

arrayul de la $a(p, m)$ si $a(m+1, q)$

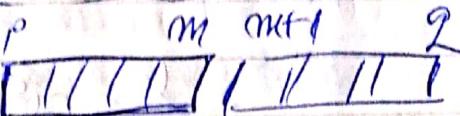
def inter (p, m, q):

+ global a

i = p

j = m + 1

b = []



i = p \rightarrow m
j = m + 1 \rightarrow q

while $i \leq m$ and $j \leq q$:

if $a[i] < a[j]$:

b.append (a[i])
i += 1

else:

b.append (a[j])
j += 1

while $i \leq m$:

b.append (a[i])
i += 1

while $j \leq q$:

b.append (a[j])
j += 1

\Rightarrow # transfer in a cu-am sortat

a[p: q+1] = b.copy ()

subprogramul de mergesort - divide

def ms (p, q):

if $p \leq q$:

$$m = (p+q)/2$$

ms (p, m)

ms (m+1, q)

inter (p, m, q)

I'm maine!

$a = [1, 5, 2, 3, 9, 3] \rightarrow 6 \text{ Elemente}$
 $mS(0, 5) \rightarrow mS(0, m-1)$
print(a)

Calculare Ordin de complexitate:

$T(n)$ = nr. operați elementare executate de algoritm pt a ordona n elemente

Se imparte en 2 módulos $\Rightarrow T(n/2)$

2 jumătăți formatează vectorul $\Rightarrow 2^* \tau(u/2)$

Plus celle d'opérations de la interclasare

$$T(\mu) = \mu + \frac{1}{2} T(\mu/2)$$

$$T(m) = m + m + 2^2 * T(m(2^2))$$

$$T(n) = M + n + n + 2^3 * T(n/2^3)$$

$$T(n) = n + T(n/2^k)$$

Kontinuierliche Teilung

In final: $m/2^K = 1 \Rightarrow m=2^K \Rightarrow K = \log_2 m \Rightarrow$
 $\Rightarrow O(m \log_2 m)$

def commode(a,b):

$$a, b = \min(a, b), \max(a, b)$$

While $a < b$:

$$b = a$$

$$a, b = \min(a, b), \max(a/b)$$

rotundata

def divide(p, q)

global

problema elementara

if $g-p \leq 1$:

return cmmdc ($\ell(p)$, $\ell(g)$)

else:

mid = $(p+g)/2$

divide (p , mid)

divide ($p+mid+1$, g)

return cmmdc (divide (p , mid),

divide ($mid+1$, g))

Apel: divide ($0, m-1$)

TURNURILE DIN HANOI

- 3 bete, se deschid

- se mută folosind betele așezate unele
mari se pun în mijlocul celor mai mici

def mută (x, i, y):

if $x > 1$: # mai am de mutat

mută ($x-1, i, 6-i-j$)

print (" $i, j, sep = " ")$

mută ($x-1, 6-i-j, j$)

else:

printez ultima mișcare print (" $i, j, sep = " ")$

Apel: ($x, 1, 2$)

BACTRACKING

1) Descompunete număr:

Varianta 1 $m = 3 \Rightarrow$

$$\begin{array}{r} 111 \\ 12 \\ 21 \\ \hline 3 \end{array}$$

def bkt(k)

global sol, m, s

if $s = m$:

print(*sol[1:k])

else:

sol[k] = 0

while $sol[k] + 1 < m$:

sol[k] += 1

s += sol[k]

bkt(k+1)

s -= sol[k]

(m-1) + m = int(input())

sol = [0]*m

s = 0

bkt(1)

Varianta 2

$m = 3$

111

12

3

def bkt(k):

global m, sol, s:

if $s = m$:

print(*sol[1:k])

else:

for i in range(sol[k-1], m-k+3):

$\text{sol}[k] = i$
 $i = 1$
 $\text{bkt}(k+1)$
 $i = 2$
 $m = \text{int}(\text{input}())$
 $\text{sol} = [0] * (m+1)$
 $\text{sol}[0] = 1$
 $s = 0$
 $\text{bkt}(1)$

Varianta 3

$m = 3 \Rightarrow [3]$ def bkt(k) :
 global sol, m, s.
 if $s == m$:
 print(*sol[1:k])
 else :
 for i in range
 ($\text{sol}[k-1]+1, m-k+3$):
 $\text{sol}[k] = i$
 $s + = i$
 $\text{bkt}(k+1)$
 $s - = i$

$m = \text{int}(\text{input}())$
 $\text{sol} = [0] * (m+1)$
 $s = 0$
 $\text{bkt}(1)$

Pări problemă:

- # intersectarea $v, x_1 \Rightarrow c$ dacă $\text{len}(v) \leq 2$
- # calcul mediana lui c
- # calcul m_1, m_2 medianele lui v, x_1
- # în funcție de m_1, m_2 , $m_1 < m_2 \Rightarrow$ eliminare jumătatea de veci

DIVIDE ET IMPERA

1) Problema MEDIANEL' a 2 vectori v și x
 → folosim unirea

def med(v, x):

if len(v) <= 2:

Primară algoritm unirea v cu x

a = 0

b = 0

x = []

while a < len(v) and b < len(x):

if v[a] < x[b]:

x.append(v[a])

a += 1

else:

x.append(x[b])

b += 1

while a < len(v):

x.append(v[a])

a += 1

while b < len(x):

x.append(x[b])

b += 1

k = len(x) // 2

returnez med a vectorial

if len(x) % 2 == 0:

return (x[k-1] + x[k]) / 2

else:

return x[k]

$$a = [1, 2, 3, 4]$$

⇒ len(a) = 4 - paște

$$\text{med} = \frac{a[1] + a[2]}{2} =$$

$$0, 1, 2$$

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

⇒ len(a) = 3 - paște

$$\text{med} = v[3/2] = 2$$

i = len(v) // 2

j = len(x) // 2

if len(v) % 2 == 0:

$$m1 = (v[i-1] + v[i]) / 2$$

else

$$m1 = v[i]$$

m_1, m_2

↳ medianele

vectorilor v, w
curenti

if $\text{len}(w) \% 2 == 0$:

$m_2 = (w[j-1] + w[j]) / 2$

else:

$m_2 = w[j]$

if $m_1 < m_2$:

$v = v[0:i]$

$w = w[i : \text{len}(w) - i]$

tau I sum a lui v

tau II sum a lui w

divide paritate

lungime v

else:

$w = w[i :]$

if $\text{len}(v) \% 2 != 0$:

$v = v[0:i+1]$

else:

$v = v[0:i]$

return $\text{med}(v, w)$

Apc: print($\text{med}(v, w)$)

2) Dintamă minimă dintre 2 pt în plan

from math import sqrt

Soluția care calculează distanța între 2 pt

def dist(p1, p2):

return sqrt((p1[0] - p2[0]) ** 2 +
+ (p1[1] - p2[1]) ** 2)

Implementare de divide et impera

def closestPair(x, y):

$YS = []$ zone

$YD = []$

if len(x) == 2:

return dist(*[x for x in x])

if len(x) == 3:

return min(dist(x[0],
x[1]), dist(x[1], x[2]),
dist(x[2], x[0]))

problema
elementară

X
P
R
O
B
L
E
M
A

V
U
C
R
A
T
U
R
A

ii) plăcătul \rightarrow

$c = x[\text{len}(x) // 2]$

Impact de la taille et du pivotage

$$XS = [x \text{ for } x \text{ in } X : \text{len}(x) // 2 + 1]$$

$$XD = [x \text{ for } x \text{ in } X : \text{len}(x) // 2 + 1 :]$$

Import si aici
pe zone

parsing mult de cele noi pentru
să se înțeleagă:

else:

Y.S. append (x)

1

11. append(x)

#dint. minima d = min (d₁, d₂)

3. Verific dacă cerință o distanță

mai mică în vecinătatea pivotului
 $S = \{x \text{ for } x \in S : \text{if } (x[0] > e[0] + d)\}$

$$\text{SD} = \sqrt{\frac{SS}{N}}$$

5. sort (key = lambda x: x[1])

for x in S :

for $y \in S$:

if $\text{dist}(x, y) = 0$ and

$$\text{dint}(x, y) \leftarrow d;$$
$$d = \text{dint}(x, y)$$

return d

$$(1,3), (3,2), (1,5), (5,5)]$$

(P, Key = lambda x : x[0])

(P. Kuy = lambda $\alpha \in \mathbb{R}[1]$)

print (closestPair (X, Y))

COMPLEXITATI DIVERSE

[LISTE] : $\text{len}(l) \rightarrow O(1)$ $l[i] \rightarrow O(1)$
 $l.pop() \rightarrow O(1)$
 ~~$l.clear() \rightarrow O(1)$~~
 $l[i] = l[j] \rightarrow O(1)$

$l == L \rightarrow O(m)$
 $x \in L \rightarrow O(m)$
 $\min(l), \max(l) \rightarrow O(m)$
 $l.pop(i) \rightarrow O(m)$
 $slice \rightarrow O(\text{len}(slice))$
construire lista $\rightarrow O(\text{len}(\text{list}))$
copy = list(a) $\rightarrow O(m)$

[SETURI] : $\text{len}(s) \rightarrow O(1)$; $s[i] \rightarrow O(1)$
 $s.add(x) \rightarrow O(1)$
 $x \in s \rightarrow O(1)$
+ remove, discard, pop, ~~clear~~ $\rightarrow O(1)$

$s_1 == s_2 \rightarrow O(m)$
set = set(a) $\rightarrow O(m)$
 $s.copy() \rightarrow O(m)$

[DICTIONARE] : $d[k] \rightarrow O(1)$
 $d[k] = v \rightarrow O(1)$
 $\text{len}(d) \rightarrow O(1)$
 $d.get(k) \rightarrow O(1)$
 $d.pop, \text{clear}, \text{keys} \rightarrow O(1)$

Obs: $\text{for } i \text{ in range}(0, 10): \rightarrow O(1)$

$\text{for } i \text{ in range}(\text{len}(alist)): \rightarrow O(m)$
if $alist[i] \in alist[i+1:] \rightarrow O(m)$
return False $\rightarrow O(1)$

DIVIDE ET IMPERA

1) Vectorul munte → Determinarea valoarei

def munte (a, p, q)

if $p = q$:

return a[q]

else :

mid = $(p+q)/2$

max1 = munte (a, p, mid)

max2 = munte (a, mid+1, q)

if $\text{max1} > \text{max2}$:

return max1

else :

return max2

Apel : munte (a, 0, m-1)

a) Făieturi părăse ale găuri \Rightarrow aria cea mai mare

def faiet (xs, ys, l, h)

global gauri, am, xm, ym

gauri = false

for $x \in \text{im gauri}$:

if $xs < x[0] < xs + l$ and
 $ys < x[1] < ys + h$:

faiet ($xs, ys, x[0]-xs, x[1]-ys$)

faiet ($x[0], ys, xs + l - x[0],$
 $x[1] - ys$)

faiet ($x[0], x[1], xs + l -$
 $x[0], ys + h - x[1]$)

faiet ($xs, x[1], x[0] - xs,$
 $ys + h - x[1]$)

gauri = True

if $gant == \text{True}$:
 $a = l * h$
if $a > am$:
 $am = a$
 $xm = xs$
 $ym = ys$

SABLON BACKTRACKING

```
def bkt(k):  
    global x  
    for v in range(prime_k, ultimo_k + 1):  
        x[k] = v  
        if soluce_parcial(x[:k+1]):  
            if soluce(x[:k+1]):  
                prenudrare(x[:k+1])  
            else:  
                bkt(k+1)
```

⊕ Problema cuburilor

- Se dă m cuburi
 - lățime latitudine
 - cub
 - culoare

- Nu există 2 culori de același dimensiune
- Se cărează cel mai înalt turn format din m cuburi cu proprietatea:
 - * pleacă un cub cu lățirea l_x și se pună un cub cu lățirea l_y dacă $l_y < l_x$
 - * pleacă un cub de culoare c_x și se pună un cub de culoare c_y dacă $c_x \neq c_y$
- Formează lăția L de tupleturi (lungime, culoare)
- (+) Sortează lăția L crescător după lungime

Sort(L)

Sol $\leftarrow [L_0]$

pentru $i \in \{1, 2, \dots, m-1\}$:

culoarea
cubului actual \neq
+ culoarea ultimului cub pus

dacă $L_{i+1} \neq Sol_{i+1}$:
Sol.append(L_i)

$O(m \log m)$

⊕ Problema bancnotelor

- Se dă o sumă Sum pe care urmărești să o platim cu un nr. cât mai mic de bancnote din multimea B cu proprietatea:

$$2) \quad B_i < B_{i+1}; \quad i = 0, m-2$$

$$3) \quad B_i \mid B_{i+1}; \quad i = 0, m-2$$

- Sortează crescător multimea B (dacă multimea este deja sortată)
- Formează lăția Sol de tupleturi (bancnote, nr. b)

Sol $\leftarrow []$

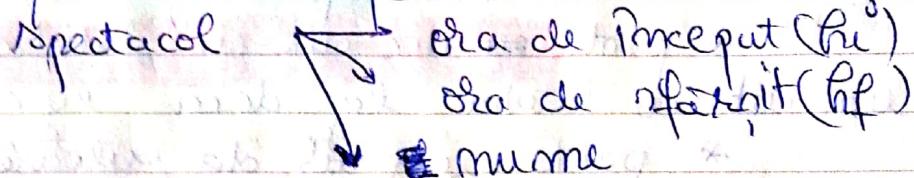
Sort(B): $i \in \{0, 1, \dots, m-1\}$

cât timp $Sum > 0$:

dacă $B_i \leq \text{Sum}$:
 Sol.append($(B_i, B_i // \text{Sum})$)
 $\text{Sum} \leftarrow \text{Sum} \% B_i$
 $i \leftarrow i + 1$

⊗ Problema Spectacolelor

- Aflăm o mulțime de m spectacole



- Vrem să maximizăm programările nr de spectacole într-o lună sănătoasă.

- Formez lista L de tupluri $(hi, hf, nume)$

- Sortez L crescător după hf

Sort(L)

Sol $\leftarrow [L_0]$

pentru i de la 1 la $m-1$:

dacă $Sol_{-1,1} \leq L_{i,0}$:

Sol.append(L_i)

ora de sfârșit
a ultimului spectacol programat
trebuie să fie \leq

\leq cu ora de început a următorului spectacol pe care vrem să-l programăm

⊗ Problema minimizării întârzierii maxime

- Se dau n activități

activitate

$d_0 = 0$ fi (deadline)

s_i = moment de început activității i

f_i = moment de sfârșit activității i

t_i = durată activității i

$h_i = \max\{s_0, f_i - t_i\}$

H = întârzierea maximă = $\max(h_i)$

- Formez lista L de tripleturi (durată, deadline)
 - Sort(L)
 - $H \leftarrow 0$
 - $i \leftarrow 0$

pentru i de la 0 la n-1 :

$$f_i \leftarrow s_i + L_i$$

$$h_i \leftarrow \max(0, f_i - t_i)$$

dacă $h_i > H$:

$$H \leftarrow h_i$$

$$s_i \leftarrow f_i$$

④ Problema cuielor :

- Ni se dau m secunduri
- secundura \rightarrow capăt stâng
- Sa se găsească multimea minimă de cui
- astfel încât să se fixeze cele m secunduri
- Formez lista L de tripleturi : (capst, cap^{dr})
- Sortez lista L crescător după capătul din stânga

Sort(L)

 $\Rightarrow S \leftarrow []$

$cui \leftarrow -1$ # pozitia celui mai din dreapta

pentru i de la 0 la n-1 :

dacă $L_{i,0} \leq cui \leq L_{i,1}$:

continuă

dacă $L_{i,0} > cui$:

Sol. append ($L_{i,1}$)

$cui = L_{i,1}$

dacă $L_{i,1} < cui$:

Sol. pop()

Sol. append ($L_{i,1}$)

$cui = L_{i,1}$

$M \leftarrow len(S)$

(*) Problema frunzelui

- Se dau m obiecte și o greutate maximă G

obiect \rightarrow valoare
greutate

- Se cere alegerea obiectelor în limite greutății G astfel încât să maximizăm profitul total

- Formez lista L de dupluri: $\{(\text{val}, \text{gr}), (\text{val}^2, \text{gr}^2)\}$

- Sortez L după denumătorul după raportul val/gr

Sort (L)

Sol $\leftarrow []$; $i \leftarrow 0$; $V \leftarrow 0$

căt timp $L_{i+1} \leq G$:

Sol.append ($(L_{i,0}, L_{i,1})$)

$V \leftarrow V + L_{i,0}$

$G \leftarrow G - L_{i,1}$

$i \leftarrow i + 1$

dacă $G \neq 0$:

$p \leftarrow G / L_{i,1}$

$V \leftarrow V + p * L_{i,0}$

Sol.append ($(p * L_{i,0}, p * L_{i,1})$)

TEOREMA MASTER (rec)

$$T(n) = a \cdot T(n/b) + f(n) \rightarrow \text{timp necesar rezolvării problemelor}$$

$a \geq 1$

$b > 1$

$$f(n) = O(m^k \log^p m)$$

3 cazuri:

$$1) \text{ Dacă } \log_b a > k \Rightarrow \Theta(m^{\log_b a})$$

$$2) \text{ Dacă } \log_b a = k$$

$$3.1 \text{ dacă } p \geq -1 \Rightarrow \Theta(m^k \log^{p+1} m)$$

$$3.2 \text{ dacă } p = -1 \Rightarrow \Theta(m^k \log \log m)$$

$$3.3 \text{ dacă } p < -1 \Rightarrow \Theta(m^k)$$

$$3) \text{ Dacă } \log_b a < k$$

$$1. \text{ dacă } p \geq 0 \Rightarrow \Theta(m^k \log^p m)$$

$$2. \text{ dacă } p < 0 \Rightarrow \Theta(m^k)$$

$$T(n) = aT(n/b) + f(n); f(n) \in O(n^k)$$

$$1) \text{ Dacă } a > b^k \text{ atunci } T(n) \in O(m^{\log_b a})$$

$$2) \text{ Dacă } a = b^k \text{ atunci } T(n) \in O(m^k \log m)$$

$$3) \text{ Dacă } a < b^k \text{ atunci } T(n) \in O(m^k)$$

PROGRAMARE DINAMICĂ

(*) Cel mai lung subșir ↑

0	1	2	3	4	5	6	7	8	9	10
v = [5 9 2 4 7 6 7 15 10 11 9]										

→ cel mai lung subșir ↑: 2, 4, 6, 7, 10, 11
 $M = \text{len}(v)$

Iau: T - vector auxiliar de lungimi

$T[i]$ = lungimea celui mai lung subșir ↑ care se termină cu $v[i]$

$$T[i] = 1 + \max(0, \{ T[j] | 0 \leq j < i \text{ și } v[i] > v[j] \})$$

0	1	2	3	4	5	6	7	8	9	10
$T = [1 2 1 2 3 3 3 4 5 5 6 5]$										

lungime max

Pred - vector auxiliar de precedenți

0	1	2	3	4	5	6	7	8	9	10
$\text{Pred} = [-1 0 -1 2 3 3 5 6 6 8 6]$										

$6(m^2)$

Răsolvare: $T = [1] * n$

$\text{Pred} = [-1] * n$

for i in range ($1, n$):

for j in range (i):

if $v[i] > v[j]$ and $T[i] < T[j] + 1$:

$T[i] = T[j] + 1$

$\text{Pred}[i] = j$

$m = \max(T)$

$poz = T. \text{index}(m)$

$afis = []$

while $m > 0$:

$afis.append(v[poz])$

$poz = \text{pred}[poz]$

$m -= 1$

$afis.reverse()$

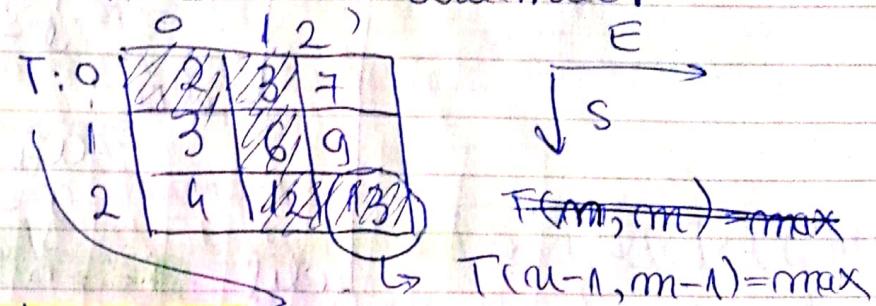
Print (afis)

⊗ Drumuri în matrice (de cost maxim)

Am un robotel care poate să meargă doar în direcțile S și E. Tocmai costul maxim al costă ceva. Văd costul maxim și drumul.

M:	0	1	2
0	2	1	4
1	1	3	2
m linii	2	1	6

m coloane



$T[i][j] = \text{suma maximă ce se poate obține}\newline \text{ajungând în } M[i][j]$

$$T[i][j] = M[i][j] + \max \begin{cases} T[i-1][j] & ; i > 0 \\ T[j-1][i] & ; j > 0 \end{cases}$$

→ Ca să afișez drumul plec de la $(m-1, m-1)$

$O(m \times m)$

citire M

declarare $\leftarrow T = [\ [0 \text{ for } j \text{ in range}(m)] \text{ for } i \text{ in range}(n)]$

$$T[0][0] = M[0][0]$$

for i in range(m):

 for j in range(m):

metig doar pe

 coloana linie

 if $i > 0 \text{ and } j = 0$:

$$T[i][j] = M[i][j] + T[i-1][j]$$

metig doar

 pe coloana

 elif $j > 0 \text{ and } i = 0$:

$$T[i][j] = M[i][j] + T[i][j-1]$$

 elif $i > 0 \text{ and } j > 0$:

 aplic formula

$$T[i][j] = M[i][j] +$$

$$+ \max(T[i-1][j], T[i][j-1])$$

print ($T[m-1][m-1]$)

$i = m-1$
 $j = m-1$

```

afis = [(i,j)]
while i>0 or j>0:
    if i>0 and j>0:
        if a[i-1][j] >= a[i][j-1]:
            afis.append((i-1,j))
            i -= 1
        else:
            afis.append(((i,j-1)))
            j -= 1
    elif i==0:
        afis.append((i,j-1))
        j -= 1
    elif j==0:
        afis.append((i-1,j))
        i -= 1
    afis.reverse()
print(afis)

```

(*) Problema monedelor

coins = [1, 5, 7] → nu contează ordinea

sum = 13

a = [0] * (sum + 1)

for i in range (1, sum + 1):

if i in coins:

a[i] = 1

else:

v = []

for x in coins:

if i > x:

v.append(a[i-x])

a[i] = min(v) + 1

print(a[-1])

$a[i]$ = număr de monede cu care poate fi obținută suma i

* Cale mai lungă subșir de dominouri.

0 1 2 3 4 5

$V = [(2,5), (4,9), (5,6), (9,4), (4,3), (1,2)]$
 $(3,2), (8,8)] \rightarrow$ indexate de la 0

$m = \text{len}(V)$

$l = [1]^* m ; lr = [1]^* m$

$p = [-1]^* m ; pr = [-1]^* m$

for i in range $(1, m)$:

for j in range (i) :

if $V[j][1] = V[i][0]$ and $l[i] < l[j+1]$:

$l[i] = l[j]+1$

$p[i] = j$

if $V[j][0] = V[i][0]$ and $pr[i] <$
 $< lr[j+1]$:

$lr[i] = lr[j]+1$

$pr[i] = j$

$m = \max(l)$

$poz = l \cdot \text{indexe}(m)$

$afls = []$

while $m > 0$:

$afls.append(V[poz])$

$poz = p[poz]$

$m -= 1$

$afls.reverse()$

print(afls)

~~l~~ vector auxiliar pt lungimea lărg domino

$l[i] \leftarrow$ lungimea celui mai lung

subșir de dominouri care se

termină cu $V[i]$

$p \leftarrow$ vector auxiliar de predecesori

$l[i] = \max\{0, \{l[j]\} | 0 \leq j < i \text{ and } V[i] = V[j][1]\}$

$= V[j][1]\}$

→ Se căută cu lungimea celui mai mare
subșir ↑

(*) Cel mai lung subșir comun:
dintre x_1 și x_2

$$x_1 = \boxed{0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9} \quad \rightarrow m = 9$$

$$x_2 = \boxed{0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9} \quad \rightarrow m = 9$$

$$x_2 = \boxed{5 \ 9 \ 2 \ 7 \ 10 \ 14 \ 3 \ 16 \ 19} \quad \rightarrow m = 9 \quad (\text{indexat } 0)$$

$$\Rightarrow 5, 2, 7, 3, 10 - \text{lung} = 5$$

$T = 0$	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1
2	0	1	1	2	2	2	2	2	2
3	0	1	1	2	3	3	3	3	3
4	0	1	2	2	3	3	3	3	3
5	0	1	2	2	3	3	3	3	3
6	0	1	2	2	3	3	3	4	4
7	0	1	2	2	3	3	3	4	4
8	0	1	2	2	3	4	4	4	5
9	0	1	2	2	3	4	4	5	5

$T[m][m]$

lungime
maximă

$T[i][j] = \text{lungimea celui mai lung subșir comun dintre prefixul de lungime } i \text{ și prefixul de lungime } j$

$$T[i][j] = \begin{cases} T[i-1][j-1] + 1, & \forall [i] = x_1[j] \\ \max(T[i-1][j], T[i][j-1]) & \text{altfel} \end{cases}$$

\rightarrow Ce să afleam subșirul plec din (m, m)

dacă vecinii $\boxed{V_1}$ și $\boxed{V_2}$ sunt $V_1 = V_2 \Rightarrow$

$$\boxed{V_1} \rightarrow (m, m) \quad \boxed{V_2}$$

\Rightarrow merge pe diagonală, altfel merge înainte

$$m = g$$

$$M = 9 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$$

$$V = [5, 2, 4, 9, 4, 13, 12, 10, 4]$$

$$X = [5, 9, 2, 7, 10, 14, 3, 10, 19]$$

initialize matrix T

$T = [T_0]^* (m+1) \text{ for } i \in \text{range}(m+1)$

completeare T

for i in range (1, m+1):

for j in range (n, m+1) :

if $v[i-1] == x[i - 1]$:

pt cā vēd dōunt

de la O!

else

$$T[i][j] = \cancel{F[i][j]}^{\text{max}}$$

$$(T(i-1)G_j, T(G_i)G_{i-1})$$

cel mai lung sub sir $\rightarrow \text{TRMTCM}$

print ($T[m^3]m^3$)

afisare mit:

def parabola(m, m):

~~if $m = -6$ or $m = -8$:~~

~~Handwritten~~

if $m > 0$ and $m \neq 0$:

if $T[m][m] = \max$

if $T[m][m] = \max$ (

$$T[m-1][m] = T[m][m-1]$$

if $f(m)(m) = T(m-1, m)$
 for which $(m-1, m)$

parcuhg, (m-1, m)

pairwise $(m, m-1)$

else.

pe diagonală

parcurs (m, m)

parcuhg $(m-1, m)$,
return $\lfloor \sqrt{m} \rfloor$, end = ''

Următoarele sunt rezolvări la 2 vectori nortati (DEF)

Dacă $m = m'$

Romunt la:

Prima permătate a vechi cu med. mai mică

și la a doua permătate a vechi cu med. mai mare

Dacă $m \neq m'$

$\rightarrow m' \rightarrow m$

mediana (v_1, v_2) :

dacă $M \leq 2$:

L'intervalez v_1 cu centrul extincii și a lui $v_2 > \dots$ + **hătutinez mediana**

Centrul extincii

\rightarrow par: a, m, m, b

k impar: a, m, b

par: $a \frac{[k+2-n]}{2} + a \frac{[k+2]}{2}$

impar: $a \frac{[k+2]}{2}$

$x =$ mediana lui v_1

$y =$ mediana lui v_2

dacă $x = y$:

return x

dacă $x < y$:

< Romunt la I - - - - ->

dacă $x > y$

< Romunt la II - - - - ->

mediana (v_1, v_2)