

EXAMEN LSD

iAPÄ

Curs 1. Multimi

- o multime este o colectie de elemente
- ordinea elementelor nu conteaza
- un element nu apare de mai multe ori.

FUNCTII

Componentele functiei:

1. domeniul de def.
2. domeniul de val. (codomeniu)
3. asocierea propriu-zisa (legea)

Proprietati:

1) Injectivitate:

$$\begin{array}{l} \forall x_1, x_2 \in A \Rightarrow f(x_1) \neq f(x_2) \\ x_1 \neq x_2 \end{array}$$

SAU

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

Daca $f: A \rightarrow B$ inj: $\Rightarrow |A| \leq |B|$

2) Surjectivitate:

$$\forall y \in B \exists x \in A \text{ s.t. } f(x) = y$$

Daca $f: A \rightarrow B$ surj: $\Rightarrow |A| \geq |B|$

3) Bijectiv:

→ funcție injectivă + bijectivă

Dacă $f: A \rightarrow B$ bij. $\Rightarrow |A| = |B|$

FUNCTU IN PYTHON

→ indentarea este foarte importantă în scrierea codului

Functii anumite

lambda argument: expresie

exemplu:

(lambda x: x+3)(2) $\Rightarrow 5.$

Cursul 2

CARDINALUL UNEI MULTIMI

→ nr. de elemente al multimii

→ se notiază cu $|A|$.

TUPLURI

→ un n-tuplu e un sir de n elemente (x_1, x_2, \dots, x_n)

Proprietăți:

- elem. nu sunt neapărat distincte
- ordinea în tuplu contează

PRODUS CARTEZIAN

- a două multimi e multimea perechilor

$$A \times B = \{ (a, b) \mid a \in A, b \in B \}$$

- a m multimi e multimea m-tupelor

$$A_1 \times A_2 \times A_3 \times \dots \times A_m = \{ (x_1, x_2, \dots, x_n) \mid x_i \in A_i \}$$

- dacă mult. sunt finite atunci

$$|A_1 \times A_2 \times \dots \times A_m| = |A_1| \cdot |A_2| \cdot \dots \cdot |A_m|.$$

COMPOUNEREA FUNCȚIILOR

$$f: A \rightarrow B \quad \text{și} \quad g: B \rightarrow C$$

$$g \circ f: A \rightarrow C \quad (g \circ f)(x) = g(f(x)).$$

- putem compune doar cind codomeniul lui f

"domeniul lui g

Proprietăți:

1) Asociativitate

$$(f \circ g) \circ h = f \circ (g \circ h)$$

FUNCȚII INVERSABILE

→ definim fct. identitate $\text{id}_A: A \rightarrow A$
 $\text{id}_A(x) = x$.

→ o fct. $f: A \rightarrow B$ e inversabilă dacă și

fct. $f^{-1}: B \rightarrow A$ a.i.

$f^{-1} \circ f = \text{id}_A$

$f \circ f^{-1} = \text{id}_B$

! o fct. este inversabilă dacă e bijectivă

IMAGINE și PREIMAGINE

fi $f: A \rightarrow B$

* $S \subseteq A$ mult. elementelor $f(x)$ cu $x \in S$
 se numește imaginea lui S prin f
 notată $f(S)$

* $T \subseteq B$ mult. elementelor x cu $f(x) \in T$
 se numește preimaginea lui T prin f ,
 notată $f^{-1}(T)$

$$f^{-1}(f(S)) \supseteq S$$

PROBLEME DE NUMĂRARE

A, B - mult. finite $f: A \rightarrow B$

$|B|^{|\mathbb{A}|}$ - fct. de la A la B .

$A|B| = \frac{|B|!}{(|B|-|\mathbb{A}|)!}$ fct. injectivă

$|PA| = |A|!$ fct. bijectiv

TIPII DATE PYTHON

- lista
- tuplu
- multime (set)
- dictionar

LISTĂ

pare = [0, 2, 4, 6]

↑
pare[0] pare[1]

TUPLU

pare = (0, 2, 4, 6).

MULTIMI DEFINITE INDUCTIV

elem. de bază

$A = \{3, 5, 7, 9, \dots\}$

• putem defini: $A = \{x \mid x = 2k + 3, k \in \mathbb{N}\}$

inductia
(reguli de
construcție)

Cursul 3

SIRUL LUI FIBONACCI

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

termenul n din sir are forma:

$$f(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$$

RECURSIVITATE

* Elementele unei definiții recursive:

1. Calcul de bază
 2. Relație de recurență
 3. Demonstrația de oprire a recurenței
- o fct. e recursivă dacă apare în propria sa definiție

FUNCTII RECURSIVE IN PYTHON

```
def functie_recursiva():
    ...
    functie_recursiva()
...
functie_recursiva()
```

LIMITARE IN PYTHON

→ fiecare apel de funcție care rămâne în execuție folosește spațiu de memorie pe stivă

→ PYTHON limitează nr. de apeluri ale același expresii la 1000: maximum recursiune de pti

exceeded error

se poate modifica folosind
setrecursionlimit() din
modul sys

TAIL RECURSION

factorical

classic

```
def factorial(x):
    if(x == 1):
        return 1
    else:
        return x * factorial(x-1)
```

tail recursion

```
def fact(m, a=1):
    if(m == 1):
        return a
    else:
        return fact(m-1, m*a)
```

Avantajele recurentării în programare:

- cod mai scurt, elegant, curat
- probleme impărțite în subprobleme

mai simple

- generația de sisteme mai simple

dezavantajele recurentării în programare:

- mai greu de urmărit
- apelurile recursive repetate folosesc multă memorie
- erori mai greu de corectat

Curs 4

* Tipurile de date primitive sunt înmutabile.

! Tipuri predefinite pt. colectii de date:

- liste
- tupluri
- multime (set)
- dictionar

LISTE []

- reprezentă colectiv de elemente
 - secvență finită, ordonată
 - ordinea elementilor contează $[1; 3; 2] \neq [3; 1; 2]$
 - acces doar la primul element
 - două liste sunt egale dacă au același elem. în același ordine
- Listă ↗ cap listă
- ↗ coadă - este o listă, nu ultimul element
- poate contine ca element sau tip de date, inclusiv alte liste

lista = [1, 2, 3, 4] ← creare listă sau lista = list(1, 2, 3, 4)

len(lista) = 4

lista.append(5) → adaugă elementul 5 la finalul listei

lista.insert(1, 5) → adaugă elementul 5 pe poz. 1.

 /
 pozitie
 \element

lista2 = [101, 102, 103]

lista.extend(lista2) → concatenare

lista.remove(2) → elimină elementul 2

 ↳ elimină doar prima apariție

lista.pop(2) → elimină elementul de pe pozitia 2.

del lista[0] → sterge elementul de pe pozitia 0.

lista.clear() → o sterge pe totată

lista.sort() → sortază lista

 ↳ putem da ca argument reverse = True

lista.reverse() → o întoarcе invers

lista1 = lista.copy() → copiază lista

lista1 + lista2 → concatenare

numere = [1, 2, 3, 4] → scrie că la patrat

lista = map (patrat, numere) → parcurgeaza
functie lista elem. după o
functie

import functiile → pt. fct. reduce

functiile. reduce (lambda acc, el: functie, lista, tip)

lista2 = filter (functie, lista)
filter păstrează doar elem. care susțin
functie

head, tail = lista[0], lista[1:]

Cuțit 5

MULTIMI

Operări de bază cu multimi

Reuniune $A \cup B = \{x \mid x \in A \text{ sau } x \in B\}$

Intersecția $A \cap B = \{x \mid x \in A \text{ și } x \in B\}$

Diferența $A \setminus B = \{x \mid x \in A \text{ și } x \notin B\}$

Complementul $A^c = \bar{A} = \{x \in U \mid x \notin A\} = U \setminus A$

Diferență simetrică $A \Delta B = (A \setminus B) \cup (B \setminus A)$

Identitate

$$A \cup \emptyset = A$$

$$A \cap U = A$$

U - univers
∅ mult. vidică

Complement

$$A \cup A^c = U$$

$$A \cap A^c = \emptyset$$

Idempotenta

$$A \cup A = A$$

$$A \cap A = A$$

Absorbție

$$A \cup (A \cap B) = A$$

$$A \cap (A \cup B) = A$$

Dublu complement

$$(A^c)^c = A$$

Complementele elementelor

identitate

$$\emptyset^c = U$$

$$U^c = \emptyset$$

Limită universală

$$A \cup U = U$$

$$A \cap \emptyset = \emptyset$$

Legile lui de Morgan

$$(A \cup B)^c = A^c \cap B^c$$

$$(A \cap B)^c = A^c \cup B^c$$

Codiciul reunirii, intersecției, diferenței

legea reunirii

$$|A \cup B| = |A| + |B| - |A \cap B|$$

legea diferenței

$$|A \setminus B| = |A| - |A \cap B|$$

▷ Multimea sub multimile

ZISI multimile

Multimile in PYTHON set()

- sunt colecții neordonate
- neindexate
- nu permit duplicate printre elemente
- pot contine doar elem.
- nemodificabile

multime = {1, 2, 3, 4, 5} sau multime = set(1, 2, ..., 5)

len(multime) = 6

- poate contine tipuri diferențe de date
- — — — tuple, dan NU și liste sau dictionare

multime. add(z) → adaugă elem.

multime. update(multime2) → adaugă elem. din mult. 2.

multime. remove(z) sau multime. discard(z)

multime. clear() → stergă tot

del multime

uniune = m1 | m2

intersectie = m1 & m2

diferenta = m1 - m2

diferenta - aritmetica = m1 ^ m2

12

Cursul 6

RELATII

* Între A și B există $Z^{(A) \times (B)}$ "relații"

Reprezentare:

1. mult. perechilor

2. regulă

3. matrice booleană / binară

Relație binară pe o multime: $X : R \subseteq X \times X$

→ reflexivă $\forall x \in R$ avem $(x, x) \in R$.

→ irreflexivă $\forall x \in R$ avem $(x, x) \notin R$.

→ simetrică $\forall x, y \in X$ dacă $(x, y) \in R$ atunci $(y, x) \in R$.

→ antisimetrică $\forall x, y \in X$ dacă $(x, y) \in R$ și $(y, x) \in R$.

$(y, x) \in R$, atunci $x = y$

→ transitivă $\forall x, y, z \in X$, dacă $(x, y) \in R$ și $(y, z) \in R$ atunci $(x, z) \in R$.

Relație de echivalență = reflexivă + simetrică + transitivă

* Clasa de echivalență a lui x = mult. elem. aflată în relație cu x . Not. \hat{X} sau $[x]$.

Relații de ordine stricte

→ o relație \prec e o ordine strică dacă e
irreflexivă și transitivă

$$\left\{ \begin{array}{l} \nexists x \text{ cu } x \prec x \\ \text{dacă } x \prec y \text{ și } y \prec z \text{ atunci } x \prec z \end{array} \right.$$

$$\left\{ \begin{array}{l} \nexists x \text{ cu } x \prec x \\ \text{dacă } x \prec y \text{ și } y \prec z \text{ atunci } x \prec z \end{array} \right.$$

Relație de ordine totală = irreflexivă + antisimetrică + transitivă + f două elem. sunt comparabile

Relație de ordine parțială = irreflexivă + antisimetrică + transitivă

• Orică ordine totală e și ordine parțială

Inversa unei relații

$$R \subseteq A \times B - \text{relație} \quad (x, y) \in R$$

$$R^{-1} \subseteq B \times A \quad (y, x) \in R^{-1}$$

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}$$

Compozierea de relații

fie două relații $R_1 \subseteq A \times B$

$$R_2 \subseteq B \times C$$

Compozierea: $R_2 \circ R_1 \subseteq A \times C$ e relația

$$R_2 \circ R_1 = \{(x, z) \mid \exists y \in B \mid (x, y) \in R_1 \text{ și } (y, z) \in R_2\}$$

$$(y, z) \in R_2\}$$

Componerua de relații

Se poate scrie că $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$

Pt. o relație de echivalență R , $R = R^{-1}$

R e transitivă $\Leftrightarrow R \circ R \subseteq R$

Pt. o relație binară $R \subseteq A \times A$, se mărește
 $R^2 = R \circ R$

DICTIONARE { } { pochi cheie : valoare }

- colecție: -ordonată
- poate fi schimbată după creare
- nu permite duplicatele

dictionar = {

"nume": "Ana", "an": 1,

"facultate": "Automatică și Calculatoare"

}

dictionar ["an"] → 1.

dictionar.keys() → chei

dictionar.values() → valori

dictionar.items() → ambele ca tuple

dictionar.update({ "nume": "Maricela" })

↪ adăugăm elem sau modificăm.

pop() - sterge elementul indicat ca parametru

popitem() - sterge un element aleator

clear() - sterge toate elementele

def - \rightarrow este elem. individual sau intreg dictionarul

\rightarrow un dictionar poate avea ca elem. alt dictionar

Cursul 4

GRAFURI

Un graf G e o pereche ordonată

$$G = (V, E)$$

mult. muchiilor
mult. nodurilor

V - finită, nevidă

\rightarrow Ordin al unui graf = nr. de noduri al grafului

\rightarrow Un nod v este incident cu o multime de muchii și dacă muchia e atinge nodul v .

\rightarrow Două noduri se numesc adiacente dacă o muchie care le unește.

\rightarrow Grad nod = nr. de muchii incidente

$$\sum d = 2 \cdot m$$

\rightarrow Mult. muchiilor unui graf formează o relație $E \subseteq V \times V$ pe mult. nodurilor.

Drumuri și cicluri

* Lungimea unei drume = nr. de muchii parcursă

Un graf orientat este slab conex dacă are
un drum mecanic de la
o vîrstă nod la o vîrstă nod.
Este conex dacă are
un drum orientat de la
o vîrstă nod la o vîrstă nod.

Un drum euclidian = drum care conține
stări muchiile unei graf
exact o singură dată.

Un ciclu euclidian = ciclu care conține stări
muchiile unei graf exact
o dată.

Graful în Python : dictionar

cheie = nodul din graf

valoare = multimea nodurilor adiacente

Parcursuri în prioritate

```
def gresd(tree):
    if (tree != None):
        return [tree["value"] + gresd(tree["left"])
+ gresd(tree["right"])]
    else:
        return []
```

Passengers in inordine

```
def srd(tree):
```

```
    if tree != None:
```

```
        return srd(tree["left"]) + [tree["value"]]  
        + srd(tree["right"])
```

```
    else:
```

```
        return []
```

Passenger in postordine

```
def sdn(tree):
```

```
    if tree != None:
```

```
        return sdn(tree["left"]) + sdn(tree  
        ["right"]) + [tree["value"]]
```

```
    else:
```

```
        return []
```

Afisare noduri graf

```
def afisare_moduri(graf):
```

```
    return list(graf.keys())
```

Cursul 8

ARBORI

→ graf neorientat conex și fără cicluri.

Pădure

→ graf neorientat conex cu căsuțe componente conexe sunt arbori.

! Graful $G = (V, E)$ neorientat și fără cicluri.
 $|V| = n$

Numărătoarele sunt adevărate:

→ G este un arbore

→ pt. fiecare nod are distinție divizibil cu un singur drum către el.

→ G nu poate conține cicluri dacă adăugăm o muchie în plus vom avea cicluri.

→ G conex, iar $|E| = n - 1$

→ un arbore are grădăcina

→ oricănd pe lângă grădăcina are un unic
precintă

→ un nod poate avea mai mulți copii

→ nodurile fără copii se numesc noduri

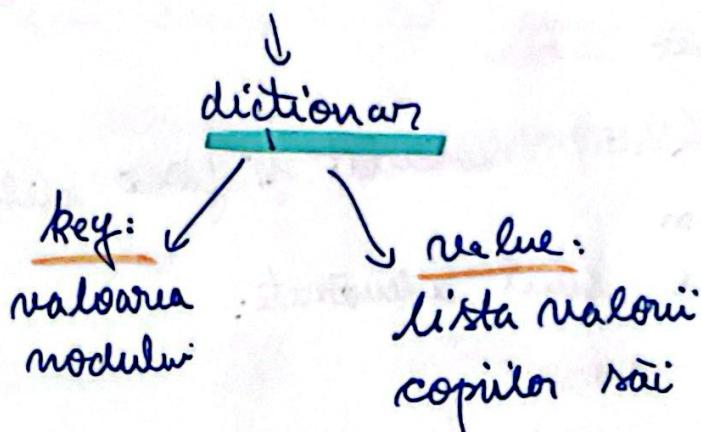
frunză

Există n^{m-2} arbori reorientați cu m noduri

Un arbore binar de înălțime m are cel mult

$2^{m+1}-1$ noduri

Reprezentarea unui arbore cătreare



TUPLURI ÎN PYTHON

→ colecție de date ordonate și nu se mai poate schimba după creare

Elementele unui tuple:

- sunt ordonate
- nu se mai pot schimba după creare
- permit duplicate

$$a = (1, 6, 8)$$

$$\text{len}(a) = 3.$$

* NU se pot adăuga și
nu se pot șterge
elemente

Cursul 3

Logica - noțiuni generale

$\text{NU} \rightarrow \text{SAU} \rightarrow \text{SI}$
 not } or } and \neg python

Sintaxă - o mulțime de reguli care definește construcțiile unei limbiș.

Sintaxă formală - precizarea modul

exact de scriere.

Sintaxă abstractă - intenționată structura formulei din subformule

Implicatie: contrapozitivă, inversă, reciproca

Fiind dateă o implicatie $A \rightarrow B$, definim:

reciproca: $B \rightarrow A$

inversă: $\neg A \rightarrow \neg B$

contrapozitivă: $\neg B \rightarrow \neg A$



SEMANTICĂ
„Entități”

$$A \rightarrow B \Leftrightarrow \neg B \rightarrow \neg A$$

$$B \rightarrow A \Leftrightarrow \neg A \rightarrow \neg B$$

O formulă poate fi:

tautologie: adevărată în toate interpretările

realizabilă: — " — în cel puțin o interpretare;

contradicție: nu e adevărată în nicio interpretare

contingentă: adevărată în unele interpretări;
falsă în altele

O formulă cu n propoziții are 2^n interpretări
(se calculă în tabel)

Cursul 1.1

→ logica are o infinitate de interpretări

→ logica e consistentă
completă

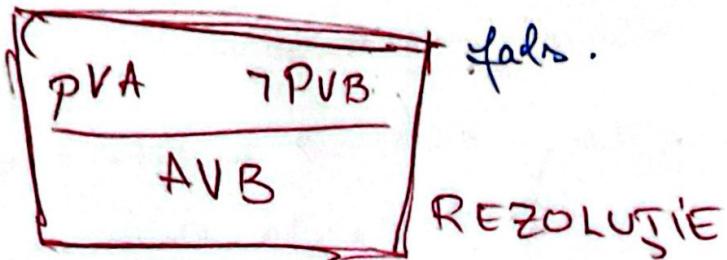
→ orice teoremă e validă

→ orice formulă validă poate fi demonstrată

→ logica ne permite să facem demonstrații

dim axiomă
ipoteze

→ un predicat = o afirmație relativă la una sau mai multe variabile, pe care rezultă aderentă sau fals.



Curs 13

alfabet \rightarrow simboluri
limbaj \rightarrow cuvinte

Un automat e dat de

simboluri de
înțețare
stări
stare initială
stare acceptoare

automat finit \rightarrow tuplu cu 5 elem.

AVANTAJE NFA

- se scrie mai ușor decât DFA
- e util când specificăm un sistem

NFA \rightarrow DFA

DFA $\not\rightarrow$ NFA

automatele finite pot fi minimezate
cu memorie finitoare

Cursul 14

masina turning = automat cu stari finite +

memorie neliimitata

→ componente

automat cu stari finite

o banda cu un nr.

infinit de celule

un cap de citire/scriere

al simbolurilor de pe
banda

→  se opreste la terminarea sirului de intrare

masina turning = tuplu cu * elem.

GRAMATICA ⇒ set de reguli care defineste

in care cuv. sunt combinate intr-o limbă nat./
limbaj de programare

- scopul lor de a definii structura

intactică a limbajelor

de a permite maximilor să
proceseze limbajul respectiv.