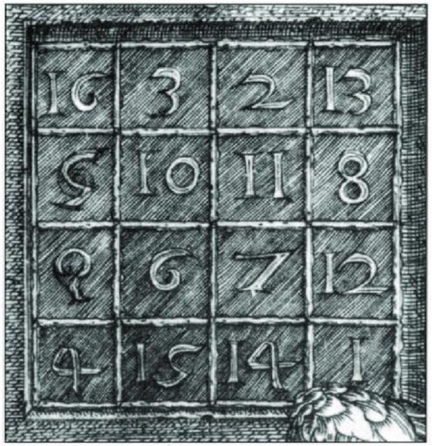
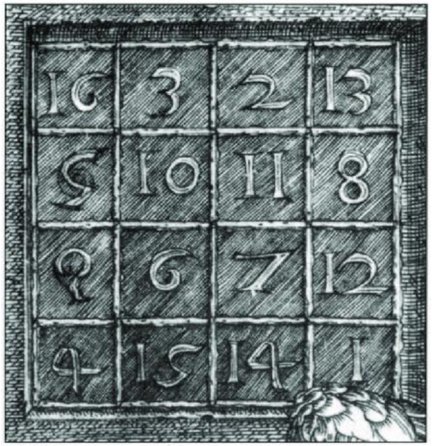
Logică și Structuri Discrete -LSD



Cursul 5 – Relații. Dicționare

dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro



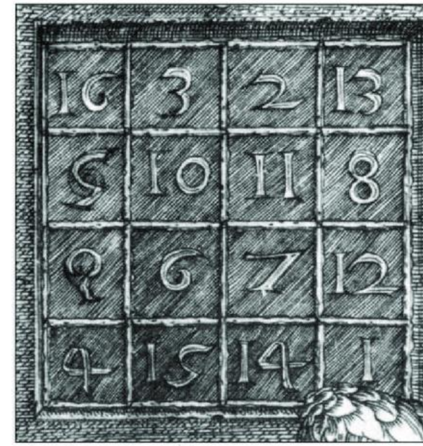
Ce am parcurs până acum?

Funcții

Funcții recursive

Liste

Mulțimi



Relații – aspecte teoretice

Relații binare

Compunerea relațiilor

Dicționare în PYTHON

Relații implementate cu Dicționare

Relații – în lumea rală și informatică

O relație (matematică) modelează *legătura* dintre două entități (posibil de *tip* diferit)

Exemple:

– Relații subiect-obiect: un om a citit o carte
– Relații umane: copil , părinte , prieten
– Relații cantitative : egal, mai mic

Relații – în lumea rală și informatică

Transpuse în informatică:

Rețele sociale : “prieten”, “follow”, “în cercuri”, etc.

O relație între elementele *aceleiași* mulțimi definește un *graf*

(elementele sunt noduri, relația e reprezentată prin muchii)

⇒ relațiile sunt o noțiune cheie în *teoria grafurilor*

Relații- mulțimi de perechi

O *relație binară* R între două mulțimi A și B e o *mulțime de perechi*: o *submulțime a produsului cartezian* $A \times B$: $R \subseteq A \times B$

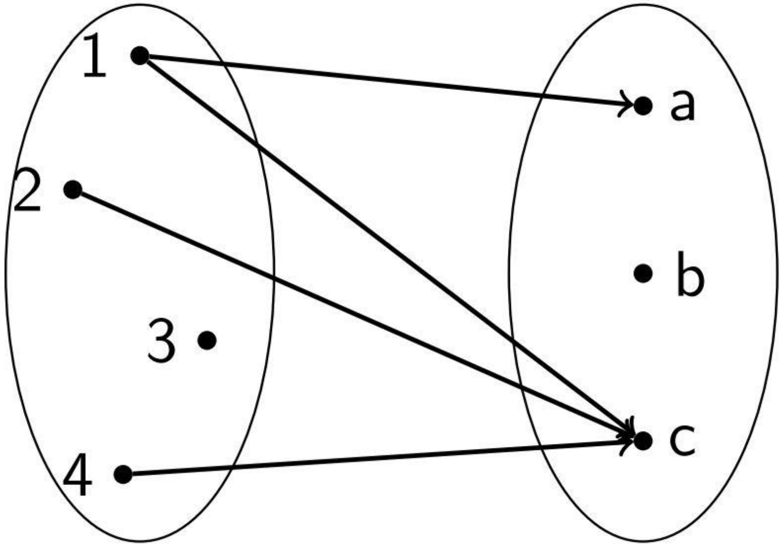
Notăm $(x, y) \in R$ sau xRy sau $R(x, y)$

când x e *în relație* cu y

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

$B = \{a, b, c\}$

$R = \{(1, a), (1, c), (2, c), (4, c)\}$



Relații- mulțimi de perechi

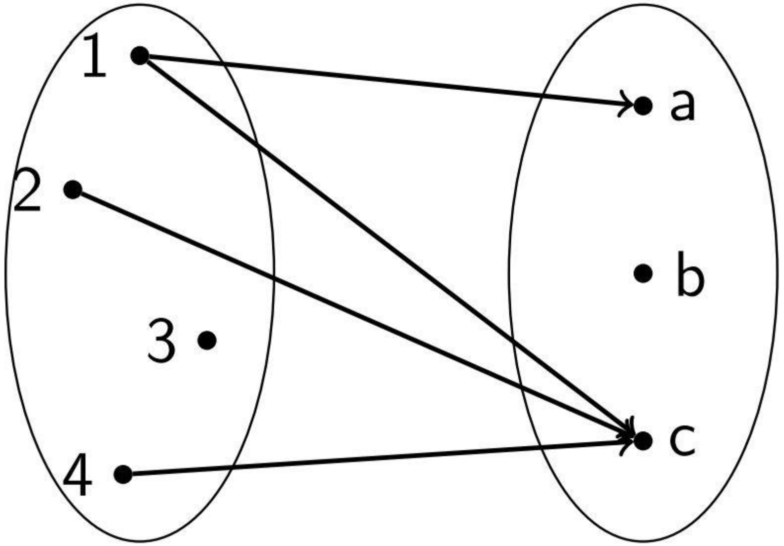
O relație e o noțiune mai generală decât *o funcție*:

- o funcție asociază fiecărui x ∈ A un singur y ∈ B

Într-o relație putem avea

(vezi figura):

- 1: are asociate mai multe elemente: a, c
- 2: are asociat un singur element: c
- 3: nu are asociat niciun element din B



Relații –generalități

În general, o relație *nu* e o noțiune simetrică: produsul cartezian/perechea sunt noțiuni ordonate,

$$(x, y) \neq (y, x)$$

Există, desigur, relații simetrice (în lumea reală și în matematică)

Generalizat, putem avea o *relație n-ară* care e o mulțime de n-tupluri (din produsul cartezian a n mulțimi).

Exemplu: $R \subseteq$ Z x Z x Z

- $R(x, y, m)$ dacă m e un multiplu comun pentru x și y :

$R(2, 9, 18)$, $R(6, 9, 18)$, $R(2, 9, 36)$, etc.

Reprezentarea unei relații

Putem reprezenta o relație:

1. Explicit, prin *mulțimea perechilor* (dacă e finită)

$$R \subseteq \{1, 2, 3, 4\} \times \{a, b, c\}$$

$$R = \{(1, a), (1, c), (2, c), (4, c)\}$$

2. Printr-o *regulă* care leagă elementele:

$$R = \{(x, x^2 + 1) \mid x \in \mathbb{Z}\}$$

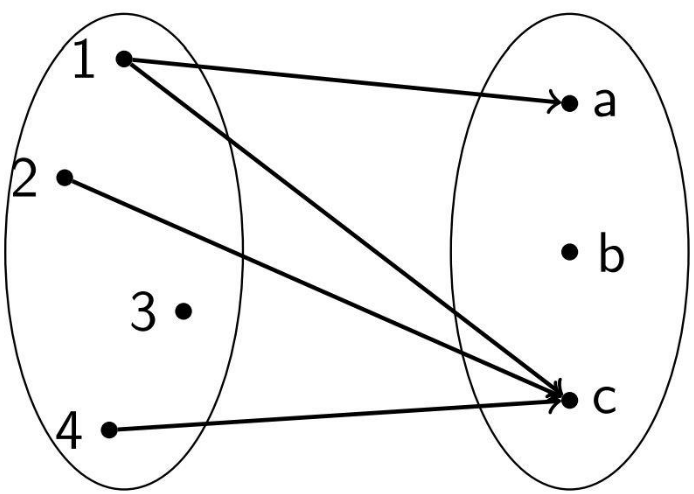
Reprezentarea unei relații

3. Ca *matrice booleană/binară*, dacă *A*, *B* finite,
   - linii indexate după *A*, și coloanele după *B*

m_{xy} *= 1 dacă (x, y) ∈ R,*

m_{xy} *= 0 dacă (x, y) ∉ R*

În practică putem folosi acest tip de reprezentare dacă *A* și *B* nu sunt foarte mari



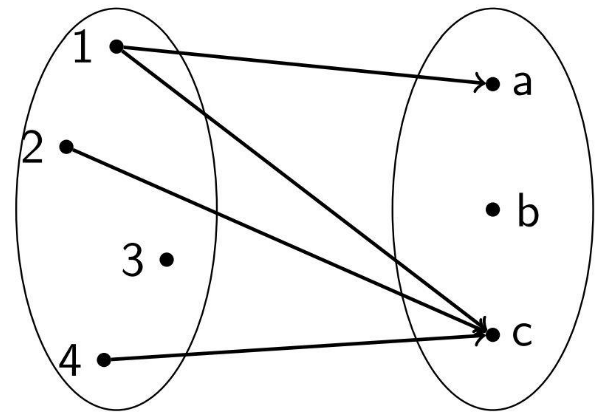
	a	b	c
1	1	0	1
2	0	0	1
3	0	0	0
4	0	0	1

Relația văzută ca funcție

O *relație* $R \subseteq A \times B$ poate fi văzută ca o *funcție* f_R de la A la mulțimea părților lui B

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Asociază fiecărui x mulțimea elementelor lui B cu care x e în relație (posibil vidă): $f_R(1) = \{a, c\}$, $f_R(3) = \emptyset$



Un vector de biți/booleni poate reprezenta o mulțime:

a	b	c
1	0	1

reprezintă {a, c} (prin funcția caracteristică)

Numărul de relații între două mulțimi

Între A și B (finite) există $2^{|A|\cdot|B|}$ relații $R \subseteq A \times B$

Rezultă direct din definiție: *o relație e o submulțime* $R \subseteq A \times B$. Deci, $R \in \mathrm{P}(A \times B)$.

Dar $|\mathrm{P}(A \times B)| = 2^{|A \times B|} = 2^{|A|\cdot|B|}$.

Sau, folosind *reprezentarea ca matrice*, care are $|A|\cdot|B|$ elemente. fiecare: ales independent în 2 feluri: 0 sau 1, deci $2^{|A|\cdot|B|}$ variante.

Funcții parțiale

O funcție parțială $f : A \to B$ e un *caz particular de relație*:

- asociază câte *un singur* element din B (ca funcția)
- dar *nu neapărat fiecărui* element din A (cum e obligată funcția)

Funcțiile parțiale sunt utile:

- când domeniul *exact* al funcției *nu* e cunoscut

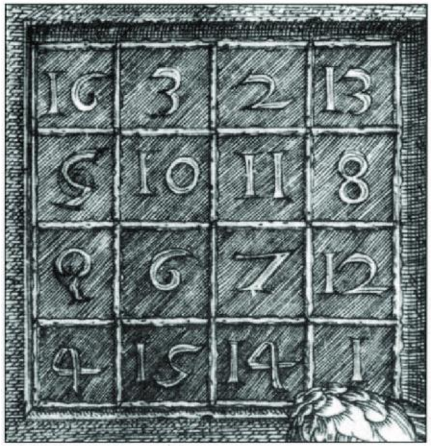
(funcții care nu sunt neaparat calculabile în orice punct).

- când domeniul de definiție al funcției e foarte mare sau nelimitat, dar reprezentăm funcția explicit *doar* pentru valorile de interes

Exemplu: populația unei localități

- posibil să nu știm populația pentru toate localitățile

- dacă argumentul e un șir, nu orice șir e nume de localitate



Relații – aspecte teoretice

Relații binare

Compunerea relațiilor

Dicționare în PYTHON

Relații implementate cu Dicționare

Relații binare pe o mulțime

Următoarele proprietăți sunt definite pentru relații binare pe *o* (aceeași) mulțime X : $R \subseteq X \times X$

- *reflexivă*: pentru orice $x \in X$ avem $(x, x) \in R$
- *ireflexivă*: pentru orice $x \in X$ avem $(x, x) \notin R$
- *simetrică*: pentru orice $x, y \in X$, dacă $(x, y) \in R$ atunci și $(y, x) \in R$
- *antisimetrică*: pentru orice $x, y \in X$, dacă $(x, y) \in R$ și $(y, x) \in R$, atunci $x = y$
- *tranzitivă*: pentru orice $x, y, z \in X$, dacă $(x, y) \in R$ și $(y, z) \in R$, atunci $(x, z) \in R$

Proprietăți ale relațiilor binare

Ce proprietăți au următoarele relații?

Proprietatea Relația	reflexivă	simetrică	antisimetrică	tranzitivă
$x \equiv y \pmod{n}$				
$x \mid y$				
$x \leq y$				

Proprietăți ale relațiilor binare

Ce proprietăți au următoarele relații?

Proprietatea Relația	reflexivă	simetrică	antisimetrică	tranzitivă
$x \equiv y \pmod{n}$	DA	DA	NU	DA
$x \mid y$	DA	NU	DA	DA
$x \leq y$	DA	NU	DA	DA

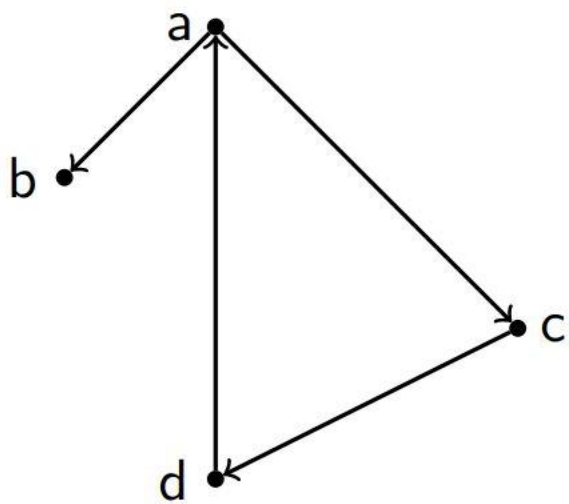
Relații binare și grafuri

O *relație binară* pe o mulțime X poate fi reprezentată ca un *graf* cu X ca mulțime de noduri:

graf *orientat*:

relație *oarecare*

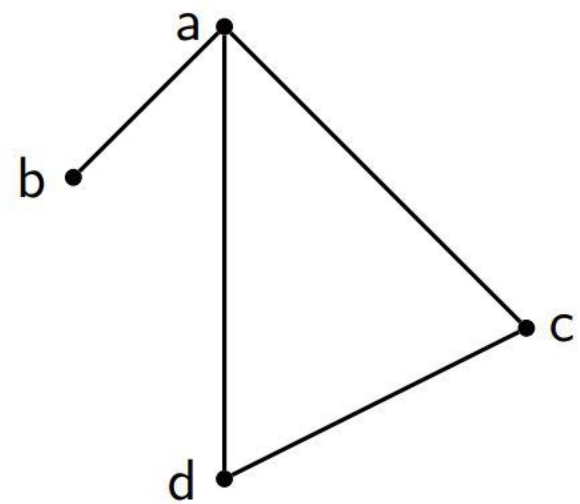
$R = \{(a,b), (a,c), (c,d), (d,a)\}$



graf *neorientat*:

relație *simetrică*

$R = \{(a, b), (a, c), (a, d), (b, a), (c, a), (c, d), (d, a), (d, c)\}$



Relații de echivalență

O relație e *de echivalență* dacă e *reflexivă*, *simetrică* și *tranzitivă*

Relația de egalitate e (evident) o relație de echivalență.

Relația de congruență modulo un număr (mod n):

$$a \equiv b \pmod{n} \text{ dacă } n \mid a - b \text{ (divide diferența)}$$

Clasa de echivalență a lui x e mulțimea elementelor aflate în relație cu x

$\{y \mid (y, x) \in R\}$ notată $\hat{x}$ sau $[x]$

Relații de ordine strictă

O relație $\prec$ e o *ordine strică* dacă e *ireflexivă* și *tranzitivă*

– nu există x cu $x \prec x$

– dacă $x \prec y$ și $y \prec z$ atunci $x \prec z$

Exemple:

- relațiile $<$ și $>$ între numere
- relația “*descendent*” între persoane

Relații de ordine totală

O relație $\leq$ e o *ordine totală* dacă e

- *reflexivă*,
- *antisimetrică* (dacă $x \leq y$ și $y \leq x$ atunci $x = y$),
- *tranzitivă*, și în plus oricare două elemente sunt *comparabile*,

adică pentru orice x , y avem $x \leq y$ sau $y \leq x$

Exemple: relațiile $\leq$ și $\geq$ între numere (întregi, reale, etc.)

Relații de ordine parțială

În practică apar adesea relații de ordine care nu sunt totale:
- clasament pe grupe, dar nu și între grupe diferite
- Știm ordinea sosirii mesajelor, dar nu și ordinea trimiterii lor
- în expresia $f(x) + g(x)$, f și g se apelează *înainte* de adunare, dar nu știm dacă se evaluează întâi f sau g

O relație e o *ordine parțială* (non-strictă), dacă e *reflexivă*, *antisimetrică* și *tranzitivă*

Exemple:

- Relația de divizibilitate între întregi
- Relația de incluziune $\subseteq$ pe mulțimea părților

Relații de ordine parțială

Orice ordine totală e și o ordine parțială

(dar nu și reciproc).

Orice ordine parțială induce o ordine strictă, și reciproc:

Definim: $a \prec b$ dacă $a \leq b$ și $a \neq b$

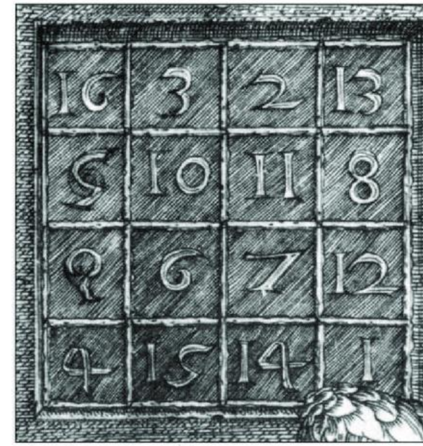
Invers, definim $a \leq b$ dacă $a \prec b$ sau $a = b$

Proprietăți ale relațiilor binare

Prop. / Relația	reflexivă	simetrică	antisim.	tranzitivă	
$x \equiv y \pmod{n}$	DA	DA	NU	DA	
x | y	DA	NU	DA	DA	
x ≤ y	DA	NU	DA	DA	

Proprietăți ale relațiilor binare

Prop. / Relația	reflexivă	simetrică	antisim.	tranzitivă	
$x \equiv y$ (mod n)	DA	DA	NU	DA	Relație de echivalență
x | y	DA	NU	DA	DA	Relații de ordine parțială
x ≤ y	DA	NU	DA	DA	



Relații – aspecte teoretice

Relații binare

Compunerea relațiilor

Dicționare în PYTHON

Relații implementate cu Dicționare

Inversa unei relații

Inversa unei relații $R \subseteq A \times B$ e relația

$$R^{-1} \subseteq B \times A,$$

cu $(y, x) \in R^{-1}$ dacă și numai dacă $(x, y) \in R$

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

Compunerea de relații

Fie două relații $R_1 \subseteq A \times B$ și $R_2 \subseteq B \times C$.

Compunerea $R_2 \circ R_1 \subseteq A \times C$ e relația

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

La fel ca la funcții, scriem $R_2 \circ R_1$ și vedem că pentru $x \in A$ găsim întâi $y \in B$ și apoi $z \in C$.

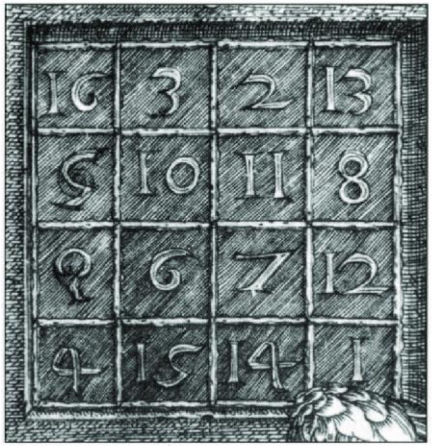
Compunerea de relații

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

Pentru o *relație de echivalență* R, $R = R^{-1}$

R e *tranzitivă* dacă și numai dacă $R \circ R \subseteq R$

Pentru o *relație binară* $R \subseteq A \times A$, se notează
$R^2 = R \circ R$, etc.



Relații – aspecte teoretice

Relații binare

Compunerea relațiilor

Dicționare în PYTHON

Relații implementate cu Dicționare

Dicționare în PYTHON

- *Dicționarul* este o colecție:
  - *ordonată* (începând cu versiunea Python 3.7),
  - care *poate fi schimbată după creare* (en.: changeable) și
  - *nu permite duplicatele*.

- Dicționarele sunt folosite pentru a stoca datele în *perechi cheie:valoare*.

Dicționare în PYTHON

Dicționarele se scriu între două acolade {} și au ca elemente *perechi* de *cheie:valoare* separate prin virgulă.

```
dictionar = {
  "nume": "Alin", "an": 1,
  "facultate": "Automatica si Calculatoare"
}
print(dictionar)

# {'nume': 'Alin', 'an': 1, 'facultate': 'Automatica si Calculatoare'}
```

Dicționare în PYTHON

Valorile din perechea cheie: valoare pot fi orice tip de date și se pot repeta.

Cheile din perechea cheie: valoare pot fi doar date care nu se pot modifica ulterior creeri lor (en.: immutable) și nu ne pot repeta.

```
dictionar = {}
dictionar2 = {1: "unu", 2: "doi"}
dictionar3 = {
        "nume": "Ana",
        "copii": ["Andrei", "Maria"]
}
```

Dicționare în PYTHON

Putem crea dicționare și cu ajutorul constructorului dict()

```
dictionar = dict()
dictionar2 = dict({1: "unu", 2: "doi"})
dictionar3 = dict(((10, "zece"), (100, "o suta")))

# {}
# {1: 'unu', 2: 'doi'}
# {10: 'zece', 100: 'o suta'}
```

Accesarea elementelor din dicționar

Dacă la liste folosim indecși pentru a accesa elementele, la dicționare vom folosi cheile. Pentru a accesa un element folosim *paranteze drepte []* sau metoda get().

```
dictionar ={
    "nume": "Alin", "an": 1,
    "facultate": "Automatica si Calculatoare"
}
print(dictionar["an"])                         # 1
print(dictionar.get("nume"))                   # Alin
```

Accesarea elementelor din dicționar

Pentru a accesa elementele putem folosi metodele: keys(), values() și items() astfel:

```
dictionar ={"nume": "Alin", "an": 1, "facultate": "AC"}
print(dictionar.keys())
print(dictionar.values())
print(dictionar.items())

# dict_keys(['nume', 'an', 'facultate'])
# dict_values(['Alin', 1, 'AC'])
# dict_items([('nume', 'Alin'), ('an', 1), ('facultate', 'AC')])
```

Adăugarea de elemente în dicționar

Dicționarele pot fi modificate după ce au fost create: putem adăuga elemente noi sau putem modifica valoarea de la o anumită cheie existentă.

dictionar ={"nume": "Alin", "an": 1, "facultate": “AC"}

dictionar["nume"] = "Marius"
dictionar["varsta"] = 20

print(dictionar)
 # {'nume': 'Marius', 'an': 1, 'facultate': AC', 'varsta': 20}

Adăugarea de elemente în dicționar

Putem adauga elemente noi sau modifica elemente existente folosind și metoda *update()*

dictionar ={"nume": "Alin", "an": 1, "facultate": "AC"}

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

dictionar.update({"nume de familie": "Popescu", "nota": 10})

print(dictionar)

#{'nume': 'Marian', 'an': 1, 'facultate': 'AC', 'nume de familie': 'Popescu', 'nota': 10}

Ștergerea de elemente din dicționar

Pentru a șterge elemente din dicționar putem folosi metodele:

pop() - șterge elementul indicat ca parametru,

popitem()- șterge un element aleator din dicționar

clear() - șterge toate elementele din dicționar

```
dictionar = {"nume": "Alin", "varsta": 20, "an": 1, "facultate": "AC"}
dictionar.pop("facultate")
print(dictionar)          # {'nume': 'Alin', 'varsta': 20, 'an': 1}
dictionar.popitem()
print(dictionar)          # {'nume': 'Alin', 'varsta': 20}
dictionar.clear()
print(dictionar)          # {}
```

Ștergerea de elemente din dicționar

Putem șterge elemente individuale sau întreg dicționarul cu del

dictionar = {"nume": "Alin", "varsta": 20, "an": 1, "facultate": "AC"}

del dictionar['nume']

print(dictionar)   # {'varsta': 20, 'an': 1, 'facultate': 'AC'}

del dictionar

print(dictionar)   # NameError: name 'dictionar' is not defined.

Verificarea existenței unui element

Pentru a verifica dacă o cheie există în dicționar vom folosi *in.* Nu putem căuta după valoare ci doar după cheie.

```
dublu = {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}

x = 2
if(x in dublu):
   print("cheia cautata este in dictionar")
else:
   print("cheia cautata nu este in dictionar")
```

Verificarea existenței unui element

Pentru a parcurge toate elementele din dicționar putem folosi for in

```
dublu = {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
for x in dublu:
   print(dublu[x])
```

Va afișa:

2

4

6

8

10

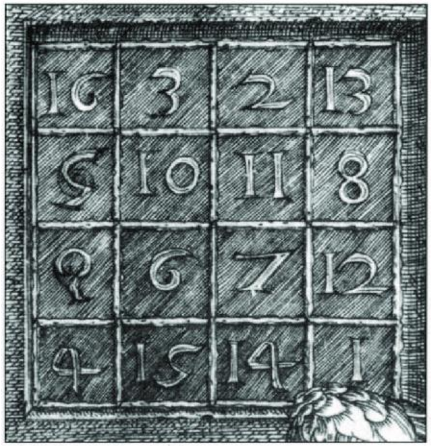
Verificarea existenței unui element

Putem avea un dicționar ca și element al altui dicționar (dicționar imbricat – en.: nested dictionary)

```
dictionar = {
    "dictionar1": {1: 1, 2: 4, 3: 9},
    "dictionar2": {1: "unu", 2: "doi"}
}

print(dictionar["dictionar1"][3])
print(dictionar["dictionar2"][2])

# 9
# doi
```



Relații – aspecte teoretice

Relații binare

Compunerea relațiilor

Dicționare în PYTHON

Relații implementate cu Dicționare

Relații cu ajutorul dicționarelor

Am văzut că o *relație* $R \subseteq A \times B$ poate fi văzută ca o *funcție* f_R de la A la mult, imea părților lui B

$$f_R(x) = \{y \in B \mid (x, y) \in R\}$$

Asociază fiecărui x mulțimea elementelor lui B cu care x e în relație (posibil vidă): $f_R(1) = \{a, c\}$, $f_R(3) = \emptyset$

Dicționarul va fi atunci de la *A* la sub*mulțimi* de elemente din *B*

Relații cu ajutorul dicționarelor

relatie = {

1: {"a", "c"},

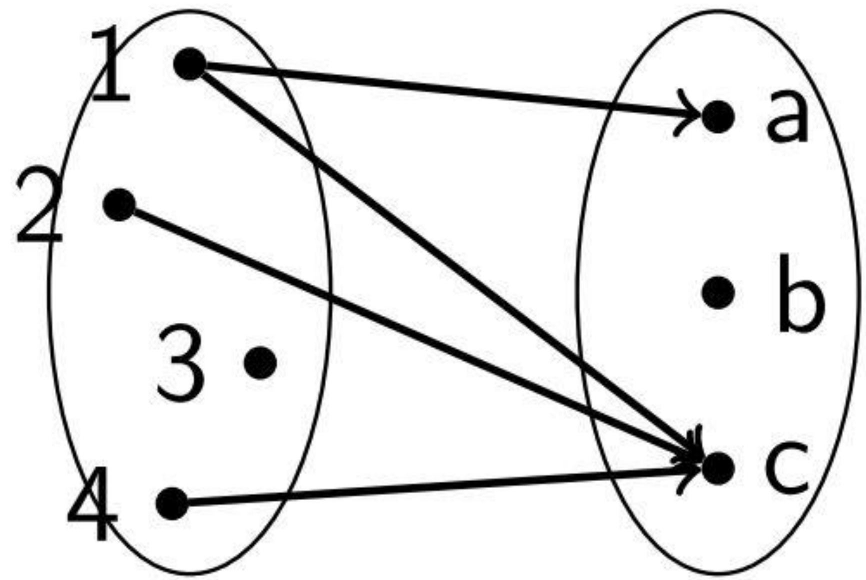
2: {"c"},

3: set()

4: {"c"}

}

#{1: {'a', 'c'}, 2: {'c'}, 3: set(), 4: {'c'}}



Exercitii cu dictionare

Scrieți o funcție care ia o listă de asociere cu perechi de tip (șir, întreg) și creează un dicționar în care fiecare șir e asociat cu *suma* tuturor valorilor cu care e asociat în listă.

Exemplu:

Input: [("Ana",7), ("Alin",3), ("Ana",9)]

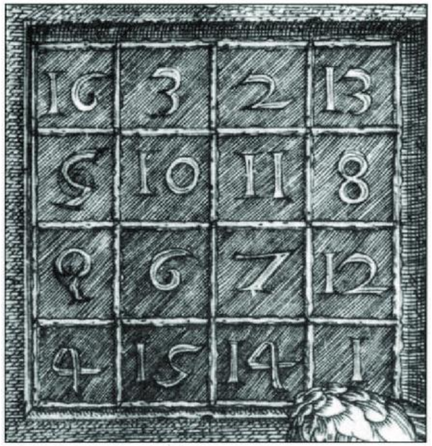
Output: {'Ana': 16, 'Alin': 3}

Exercitii cu dictionare

```
def transform(lista, dictionar = {}):
    if (lista == []):
        return dictionar
    if(lista[0][0] in dictionar):
        dictionar[lista[0][0]] = lista[0][1] + dictionar[lista[0][0]]
    else:
        dictionar[lista[0][0]] = lista[0][1]
    return transform(lista[1:],dictionar)

l = [("Ana",7), ("Alin",3), ("Ana",9)]

print(transform(l))
```



Vă mulțumesc!

Bibliografie

- Conținutul cursului se bazează preponderent pe materialele de anii trecuți de la cursul de LSD, predat de conf. dr. ing.Marius Minea și ș.l. dr. ing. Casandra Holotescu (http://staff.cs.upt.ro/~marius/curs/lsd/index.html)