C5 - Multimi

miercuri, 24 ianuarie 2024 14:13

Spre deosebire de liste:

Ordinea elementelor nu conteaza {1, 2, 3} = {2, 1, 3}

Un element nu apare de mai multe ori {1, 2, 3, 2}

A e o *submulțime* a lui B: $A \subseteq B$
dacă fiecare element al lui A e și un element al lui B.
A e o *submulțime proprie* a lui B: $A \subset B$
dacă $A \subseteq B$ și există (măcar) un element $x \in B$ astfel ca $x \notin A$.

$\in$ e o relație între un *element* și o mulțime.
$\subseteq$ s, i $\subset$ sunt relat, ii între *două mulțimi*.

Ca să demonstrăm $A \not\subseteq B$ e suficient să găsim un element $x \in A$ pentru care $x \notin B$.

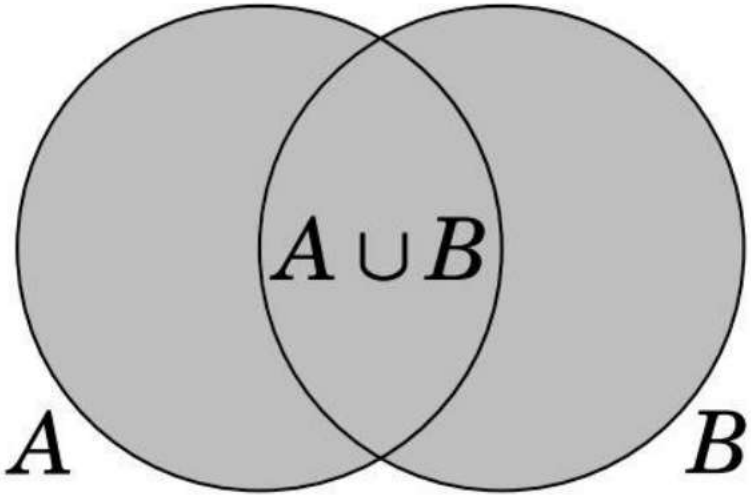
A pentru care $x \notin B$.

Dacă $A \subseteq B$ și $B \subseteq A$, atunci $A = B$ (mulțimile sunt egale)

Reuniunea a două mulțimi:

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

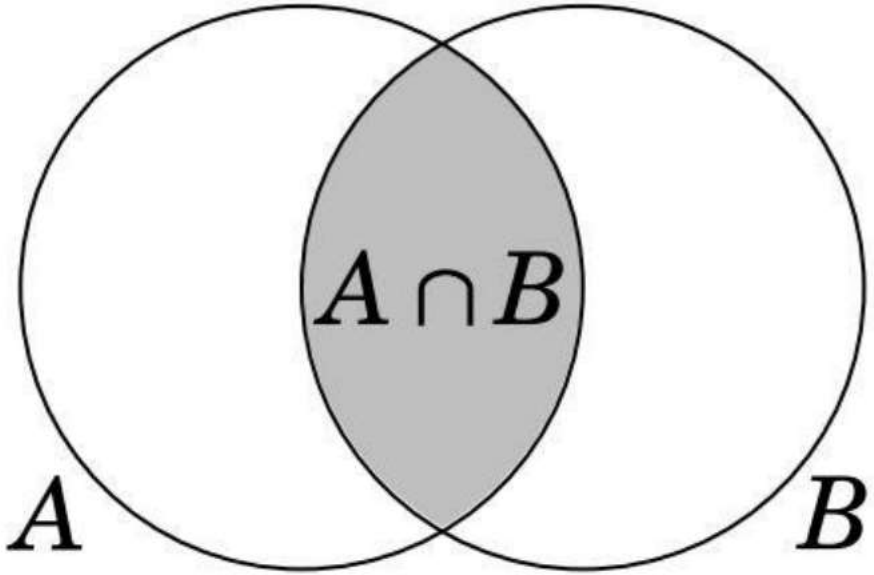
Reprezentare cu diagrame Venn:



https://en.wikipedia.org/wiki/Venn_diagram

Intersecția a două mulțimi:

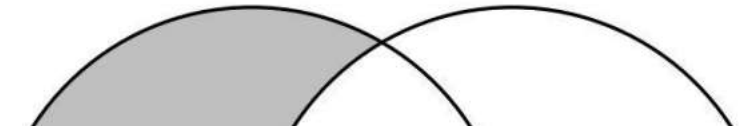
$$A \cap B = \{x \mid x \in A \text{ și } x \in B\}$$

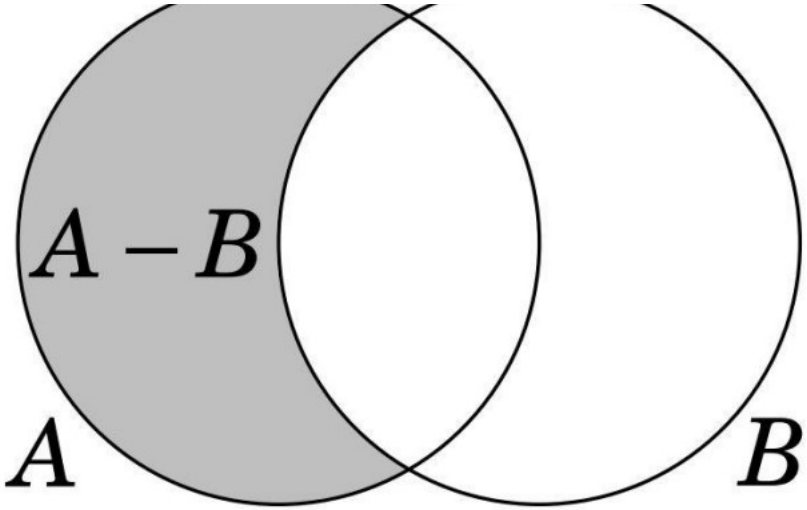


https://en.wikipedia.org/wiki/Venn_diagram

Diferența a două mulțimi:

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

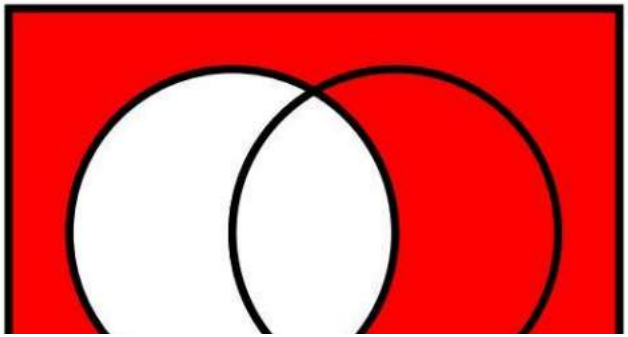


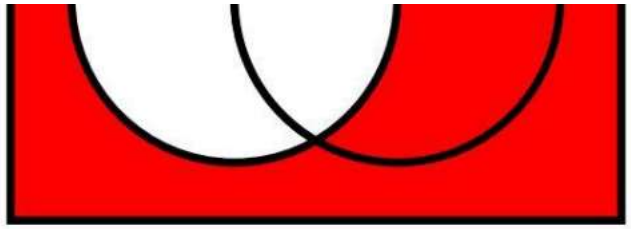


Uzual, discutăm într-un context: avem un univers U al tuturor elementelor la care ne-am putea referi.

Complementul unei mulțimi (față de universul U):

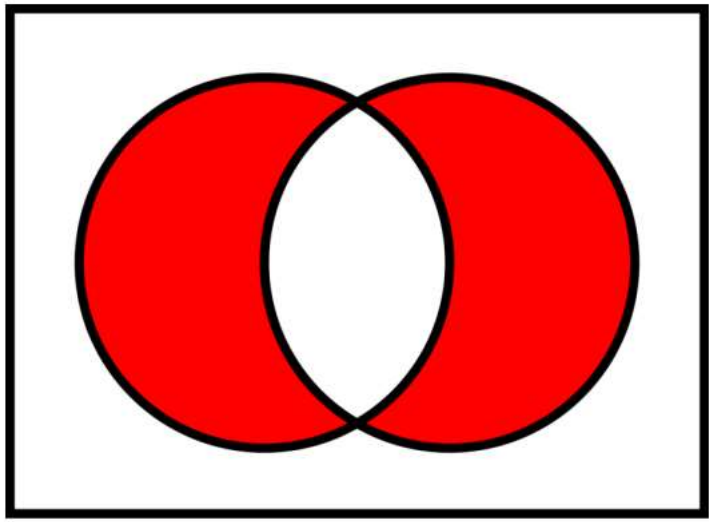
$A^C = \{x \in U \mid x \notin A\} = U \setminus A$ *(notat si $\bar{A}$)*





Diferența simetrică a două mulțimi:

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



Dacă fixăm universul U al elementelor, putem reprezenta orice mulțime $S \subseteq U$ prin *funcția caracteristică*

$$f_S : U \to \mathrm{B}: f(x) = \begin{cases} True \text{ dacă } x \in S \\ False \text{ altfel (dacă } x \notin S) \end{cases}$$

Operațiile unei algebre Boolene (aici $\cup$ și $\cap$) satisfac proprietățiile:

Comutativitate: $A \cup B = B \cup A$ $\quad\quad$ $A \cap B = B \cap A$

Asociativitate:
$(A \cup B) \cup C = A \cup (B \cup C)$ și $(A \cap B) \cap C = A \cap (B \cap C)$

Distributivitate: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ și
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

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

Identitate: există două valori (aici $\emptyset$ și universul U) astfel ca:

$$A \cup \emptyset = A \qquad A \cap U = A$$

Complement: orice A are un complement A^c (sau $\bar{A}$) astfel ca:

$$A \cup A^c = U \qquad A \cap A^c = \emptyset$$

Idempotență: $A \cup A = A$ $A \cap A = A$

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

Dublu complement: $(A^c)^c = A$

Dublu complement: $(A^c)^c = A$

Complementele elementelor identitate: $\emptyset^c = U$
$U^c = \emptyset$

Limită universală: $A \cup U = U \quad A \cap \emptyset = \emptyset$

Legile lui de Morgan:
$(A \cup B)^c = A^c \cap B^c$ $\qquad$ $(A \cap B)^c = A^c \cup B^c$
Vom revedea aceste legi la *logica propozițională*.

Axioma extensionalității:
Două mulțimi sunt egale dacă și numai dacă au aceleași elemente

Axioma extensionalității:

Două mulțimi sunt egale dacă și numai dacă au aceleași elemente

(dacă fiecare element al lui A e și un element al lui B, și reciproc)

$$\forall A, \forall B\ (A = B \Leftrightarrow \forall c\ (c \in A \Leftrightarrow c \in B))$$

Axioma mulțimii vide (existență):

Există o mulțime care nu are niciun element

$$\exists E \forall X\ \neg(X \in E\ )$$

Axioma regularității (a fundației)

Orice mulțime nevidă are un element $x \in A$ disjunct de ea:

$x \cap A = \emptyset$

$$\forall X\ (X \neq \emptyset) \Rightarrow \exists Y\ (Y \in X\ \wedge\ \neg\exists Z\ (Z \in X\ \wedge\ Z \in Y\ ))$$

Rezultă că nu există un șir infinit $A_0, A_1, \ldots A_n \ldots$ astfel încât

$$A_0 \ni A_1\ \ni\ \ldots \ni\ A_n\ \ni\ \ldots$$

($\{A_0, A_1, \ldots\}$ ar fi o astfel de multime)

$$A_0 \ni A_1 \ni \ldots \ni A_n \ni \ldots$$

($\{A_0, A_1, \ldots\}$ ar fi o astfel de mulțime)

Rezultă că *nicio mulțime nu se poate avea ca element*, $X \notin X$,

altfel $X \ni X \ni X \ldots$ ar fi un astfel de șir

Intuitiv: orice mulțime e formată din elemente (posibil mulțimi) mai simple, care la rândul lor conțin elemente mai simple, până ajungem la *elemente fundamentale*

⇒ *elimină* paradoxul lui Russell

27

Pentru mulțimi *finite*:

Legea reuniunii:

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

Legea diferenței:

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

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

Mulțimea submulțimilor (engl. *power set*) unei mulțimi S, notată $\mathrm{P}(S)$ (uneori 2^S):

$$\mathrm{P}(S) = \{X \mid X \subseteq S\}$$

Exemplu, pentru $S = \{a, b, c\}$, avem:

$\mathrm{P}(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Dacă S e finită, atunci $|\mathrm{P}(S)| = 2^{|S|}$

30

Tupluri și produs cartezian

Un *n-tuplu* e un șir de n elemente $(x_1, x_2, \ldots, x_n)$

Proprietăți:

- elementele *nu sunt neapărat distincte*
- *ordinea elementelor* în tuplu contează

Cazuri particulare: *pereche* (a, b), *triplet* (x, y, z)

Mulțimile sunt colecții care rețin mai multe elemente într-o

Mulțimile sunt colecții care rețin mai multe elemente într-o singură variabilă.

Sunt una dintre *colecțiile de bază* în PYTHON (pe lângă Liste, Tupluri și Dicționare)

Mulțimile sunt colecții:

- *neordonate*
- *neindexate*
- *nu permit duplicate* printre elemente
- pot conține doar elemente *nemodificabile* (eng. *immutable*)

Pentru *a crea* o mulțime vom enumera elementele mulțimii între *acolade* { } sau vom folosi constructorul *set()*

Pentru *a crea* o mulțime vom enumera elementele mulțimii între *acolade* { } sau vom folosi constructorul *set()*

```
multime = {1, 2, 3, 4, 5}
multime2 = set ((1, 2, 3, 4, 5))
multime3 = set ([1, 2, 3, 4, 5])
```

Pentru *a crea o mulțime vidă* vom putea folosi doar funcția *set()*, dacă definim o mulțime vidă cu două acolade, PYTHON o va interpreta ca fiind dicționar.

```
X = set()
print(type(X))
```

```
print(type(X))
# <class 'set'>

Y = {}
print(type(Y))
#<class 'dict'>
```

Ordinea elementelor nu contează. La afișare elementele pot apărea în *ordine diferită*.

```
multime = {1, 11, 4, 5, 3, 2}
print(multime)

# {1, 2, 3, 4, 5, 11}
# {1, 2, 3, 4, 11, 5}
```

 # {1, 2, 3, 4, 11, 5}
 # {1, 3, 4, 2, 11, 5}

Elementele duplicate vor fi reținute *o singură dată*

```
multime = {1, 11, 4, 5, 3, 2, 1, 1, 2}
print(multime)
```

 # {1, 2, 3, 4, 5, 11}

Pentru a afla numărul de elemente dintr-o mulțime putem folosi funcția *len()*

Pentru a afla numărul de elemente dintr-o mulțime putem folosi funcția *len()*

```
multime = {1, 11, 4, 5, 3, 2}
print(len(multime))

# 6
```

Elementele unei mulțimi pot avea *diferite tipuri de date*:

```
zile = {"luni", "marti", "miercuri"}
numere = {1, 2, 3}
valori = {True, False}
```

numere = {1, 2, 3}

valori = {True, False}

O mulțime poate conține diferite tipuri de date *în același timp*:

multime = {"luni", 1, True}

40

Elementele unei mutimi sunt nemodificabile (eng. immutable

Un tuplu poate fi element al unei mulțimi:

x = {3, 4, (1, 2, 3)}

O listă (sau un dicționar) nu poate fi element:

A={3, [4, 5,6]}

A={3, [4, 5,6]}

Va genera eroare

TypeError: unhashable type: 'list'

Accesul la elementele mulțimii *nu se poate face prin index*.

Putem verifica dacă un element este într-o mulțime cu *in*:

```
if (x in {1, 2, 3})
        print("elementul face parte din multime")
else
        print("elementul nu face parte din multime")
```

Putem parcurge element cu element cu *for in*:

Putem parcurge element cu element cu *for in*:

```
multime = {1, 11, 4, 5, 3, 2}
for x in multime:
        print(x)
```

Afișează:

1
2
3
4
5
11

Odată ce o mulțime este creată, elementele sale nu se pot modifica, în schimb *se pot adăuga* sau *șterge* elemente din mulțime.

Putem adăuga elemente noi în mulțime cu metoda *add()*

Putem adăuga elemente noi în mulțime cu metoda *add()*

```
multime = {1, 11, 4, 5, 3, 2}
multime.add(29)
print(multime)
# {1, 2, 3, 4, 5, 11, 29}
```

44

Adăugarea elementelor noi

Putem adăuga *toate elementele unei alte mulțimi* în mulțimea curentă cu metoda *update()*

```
zile = {"luni","marti","miercuri"}
zile_weekend = {"sambata","duminica"}
zile.update(zile_weekend)
```

```
zile.update(zile_weekend)
print(zile)

#{'duminica', 'marti', 'miercuri', 'luni', 'sambata'}
```

Ștergerea elementelor

Putem șterge elementedintr-o mulțime utilizând metodele remove() sau discard()

```
zile = {"luni","marti","miercuri"}
zile.remove("marti")
zile.discard("miercuri")
print(zile)

# {'luni'}
```

46

46

Ștergerea elementelor

Metoda *remove()* va genera eroare dacă se apelează cu un element inexistent, în timp ce metoda *discard()* nu va genera eroare.

```
zile = {"luni","marti","miercuri"}
zile.remove("joi")        # va genera eroare
zile.discard("joi")       #nu va genera eroare
print(zile)
```

47

Stergerea elementelor

Ștergerea elementelor

Pentru a șterge toate elementele unei mulțimi folosim metoda *clear()*

```
zile = {"luni","marti","miercuri"}
zile.clear()
print(zile)

# set()
```

48

Ștergerea elementelor

Pentru a șterge complet mulțimea putem folosi del

```
zile = {"luni","marti","miercuri"}
```

```
zile = {"luni","marti","miercuri"}
del zile
print(zile)

# print(zile)
#NameError: name 'zile' is not defined
```

49

Operații cu mulțimi

Putem calcula reuniunea a două mulțimi cu metoda *union()*. Metoda union() va crea *o nouă mulțime* care va conține elementele ambelor mulțimi.

```
multime1 = {"a", "b", "c"}
multime2 = {1, 2, 3}
multime3 = multime1.union(multime2)
print(multime3)

# {'a', 1, 'b', 2, 3, 'c'}
```

```
# {'a', 1, 'b', 2, 3, 'c'}
```

51

Operații cu mulțimi

Putem calcula intersecția a două mulțimi cu metoda *intersection()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.intersection(multime2)
print(multime3)
```

```
# {2, 3}
```

52

Operații cu mulțimi

Putem calcula diferența a două mulțimi cu metoda *difference()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.difference(multime2)
print(multime3)

# {4, 5}
```

53

Operații cu mulțimi

Putem calcula diferența simetrica a două mulțimi cu metoda *symmetric_difference()*.

```
multime1 = {2, 3, 4, 5}
multime2 = {1, 2, 3}
multime3 = multime1.symmetric_difference(multime2)
print(multime3)

# {1, 4, 5}
```

54

Operații cu mulțimi

Metodele *union(), intersection(), difference()* pot fi folosite și cu mai multe argumente:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
C = {3, 4, 5, 6}
D = {4, 5, 6, 7}
E = A.union(B, C, D)          # E = {1, 2, 3, 4, 5, 6, 7}
F = A.intersection(B, C)      # F = {3, 4}
G = A.difference(C, D)        # G = {1, 2}
```

```
G = A.difference(C, D)          # G = {1, 2}
```

Metoda *symmetric_difference()* are doar un singur argument.

55

Operații cu mulțimi

Putem verifica dacă o mulțime este submulțime sau supramulțime pentru o altă mulțime cu metodele issubset() și issuperset()

```
A = {1, 2, 3, 4}
B = {2, 3}

print(B.issubset(A))   # True
print(A.issubset(B))   # False

print(B.issuperset({1, 2, 3, 4}))   # False
print(A.issuperset({1, 2, 3, 4}))   # True

print(A.issubset(A))   # True
```

56

Operații cu mulțimi

Operații cu mulțimi

În PYTHON putem folosi și *operatori pentru lucrul cu mulțimi*:

Reuniune	*operatorul |*	*C = A | B*
Intersecție	*operatorul &*	*D = A & B*
Diferența	*operatorul –*	*E = A - B*
Diferența simetrică	*operatorul ^*	*F = A ^ B*
Submulțime	*operatorii < și <=*	*A < B*
Supramulțme	*operatorii > și >=*	*B >= A*

57

Operații cu mulțimi

Exemplu:

A = {1, 2, 3, 4}

B = {3, 4, "a", "b", "c"}

```
B = {3, 4, "a", "b", "c"}
C = A | B          #C = {1, 2, 3, 4, 'a', 'b', 'c'}
D = A & B          #D = {3, 4}
E = A – B          #E = {1, 2}
F = A ^ B          #F = {1, 2, 'a', 'b', 'c'}
```

58

Operații cu mulțimi

Dacă vrem ca rezultatul unei operații să se regăsească în *mulțimea curentă* și să nu genereze *o nouă mulțime* cu rezultatul operației vom folosi metodele: *update()*, *intersection_update()*, *difference_update()*, *symmetric_difference_update()* astfel:

```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
A.update(B)                    # A = {1, 2, 3, 4, 5}
A.intersection_update(B)       # A = {2, 3, 4, 5}
A.difference_update(C)         # A = {2}
```

A.difference_update(C) # A = {2}

Atenție, fiecare operație de mai sus va modifica mulțimea A, iar următoarea metodă va folosi *noua componență* a lui A

59

Operații cu mulțimi

Dacă vrem ca o mulțime să aibă ca *element o altă mulțime* și scriem:

A = {{1,2,3}, {7}}

Vom primi eroarea *TypeError: unhashable type: 'set'*

Nu ne lasă să scriem această sintaxă pentru că elementele mulțimii A trebuie să fie obiecte *neschimbabile* (eng. *immutable*).

Pentru a putea face astfel de operații, PYTHON ne pune la dispoziție o colecție mulțime care nu permite modificări odată ce a fost creată: *frozenset*

60

Operații cu mulțimi

Operații cu mulțimi

Mulțimile de tip *frozenset* pot folosi toate metodele și operatorii pentru tipul de date *set cu excepția* celor care modifică structura mulțimii.

Exemplu:

```
A = frozenset({'a', 'b', 'c'})
print(A)                          # frozenset({'a', 'b', 'c'})
print(len(A))                     # 3
print(A & {'a', 'b','z'})         # frozenset({'a', 'b'})
A = {frozenset({1,2,3}), frozenset({7})}

print(A.add('d'))       # acest apel va genera eroarea:
AttributeError: 'frozenset' object has no attribute 'add'
```

61

Operații cu mulțimi

Funcțiile *map(), filter() și reduce()* discutate pentru lucrul cu liste se pot folosi similar și pentru mulțimi. Cele 3 funcții au ca parametru un *iterabil*.

Elementele unei mulțimi le putem parcurge (în stilul programării funcționale) cu ajutorul funcției *reduce()*:

```
import functools
```

programării funcționale) cu ajutorul funcției reduce().

```
import functools
M = {1, 2, 3, 4}
functools.reduce(lambda acc, elem: print(elem), M, 0)
```

Ultimul argument al funției reduce (0, în exemplul de mai sus) indică de la ce *valoare inițială* să se parcurgă funcția.

62