Mulțimi

Capitolul

15

- **Tipul mulțime**
- **Operațiile posibile cu datele de tip mulțime**
- **Probleme propuse**
- **Soluțiile problemelor**

15.1. Tipul mulțime

Mulțimea este o structură statică de date cu ajutorul căreia se poate reprezenta o colecție finită de elemente. Caracterul finit ne permite indexarea elementelor sale: $M = \{m_1, m_2, ..., m_n\}$ ceea ce înseamnă că la nivelul reprezentării interne mulțimea poate fi asimilată cu un şir $(m_1, m_2, ..., m_n)$.

Pentru a putea preciza modul în care se vor efectua operațiile cu mulțimi, vom defini structura de *submulțime*. Aceasta se poate realiza cu ajutorul unui vector format din valorile *funcției caracteristice* asociate submulțimii:

$v_s = (s_1, s_2, ..., s_n)$, unde $s_i = \begin{cases} 1, & \text{dacă } m_i \in S \\ 0, & \text{dacă } m_i \notin S \end{cases}$

Fie $S_1, S_2 \in M$ şi v_{s_1}, v_{s_2} vectorii lor corespunzători. Atunci:

a) $S_1 \cup S_2$ va fi caracterizată de vectorul v_1 obținut din v_{s_1} şi v_{s_2}, efectuând operația logică $\vee$ (sau) element cu element;

b) $S_1 \cap S_2$ va fi caracterizată de vectorul v_2 obținut din v_{s_1} şi v_{s_2}, efectuând operația logică $\wedge$ (şi) element cu element.

Oricare alte operații cu submulțimile unei mulțimi pot fi imaginate ca operații logice asupra vectorilor ataşați.

Tipul mulțime este un *tip structurat, diferit de celelalte tipuri structurate; el corespunde noțiunii de mulțime din matematică.*

O valoare a unui tip mulțime este *o submulțime a valorilor unui tip ordinal, standard* sau *definit de utilizator*, numit ***tipul de bază al mulțimii***. (De exemplu, în limbajul Pascal tipul de bază al mulțimii are cel mult 256 de valori; valorile ordinale ale acestora trebuie să aparțină intervalului [0, 255].)

Un tip mulțime are atâtea valori câte submulțimi admite mulțimea valorilor tipului de bază şi mulțimea vidă, respectiv 2^n submulțimi, dacă tipul de bază are *n* valori.

15.2. Operațiile posibile cu datele de tip mulțime

1. Atribuirea
Unei variabile de tip mulțime i se atribuie valoarea unei expresii de acelaşi tip mulțime.

Exemple (Pascal)
```
A:=[];                                    { inițializare cu mulțimea vidă }
B:=[1,2,3,4,5];
C:=['A','E','I','O','U','a','e',i','o','u'];
```

2. Declararea tipului mulțime (Pascal)
type *nume_tip*=**set of** *tip_de_bază*;
var *identificator_variabilă*:*nume_tip*;
unde *tip_de_bază* este tipul de bază al mulțimii şi are cel mult 256 de valori.

Exemple
```
type mult1=set of Byte;              { numere întregi mai mici decât 255 }
     mult2=set of 0..9;                                  { mulțime de cifre }
     mult3=set of Char;                                  { mulțime de caractere }
     mult4=set of 'a'..'z';                              { mulțimea literelor mici }
```

3. Operații cu mulțimi
Operațiile cunoscute din teoria mulțimilor – se implementează în funcție de instrumentele limbajului de programare.

a. Operatori binari
- Reuniunea a două mulțimi: +
- Intersecția a două mulțimi: *
- Diferența a două mulțimi: -

b. Operatori relaționali
≤ (incluziune), ≥ (include pe), = (egalitate), <> (diferite).

Exemple
```
dacă A <= B atunci
  scrie 'A este inclus in B'
sfârşit dacă
```

```
dacă A >= B atunci
  scrie 'A il include pe B'
sfârşit dacă
dacă A = B atunci
  scrie 'A egal cu B'
sfârşit dacă
dacă A <> B atunci
  scrie 'A este diferita de B'
sfârşit dacă
```

c. Test de apartenență a unui element la o mulțime

Se foloseşte operatorul relațional **in**.

Exemple

```
dacă element in A atunci
  scrie element,' se afla in multime'
sfârşit dacă
dacă not (element in A) atunci
  scrie element,' nu se afla in multime'
sfârşit dacă
```

d. Citirea unei variabile de tip mulțime

În limbajul Pascal nu este permisă citirea unei variabile de tip mulțime. În concluzie, se va proceda similar cu citirea unei variabile de tip tablou, citind componentele mulțimii una câte una:

```
Subalgoritm Citire_Mulţime(m):
  m ← ∅      { în Pascal mulțimea vidă se notează cu ajutorul constructorului: [] }
  cât timp mai există elemente execută:
    citeşte element
              { element trebuie să fie de acelaşi tip cu tipul de bază al mulțimii }
    m ← m + [element]
           { operația de  reuniune se efectuează între doi operanzi de tip mulțime }
  sfârşit dacă
sfârşit subalgoritm
```

e. Scrierea unei variabile de tip mulțime

În limbajul Pascal nu este permisă scrierea unei variabile de tip mulțime. Şi de data aceasta se va proceda similar cu scrierea unei variabile de tip tablou, scriind componentele mulțimii una câte una.

În subalgoritmul următor presupunem că mulțimea are tipul de bază `Byte`.

```
Subalgoritm Scriere_Mulțime(m):
  pentru element=0,255 execută:
                { generăm fiecare element care aparține tipului de bază al mulțimii }
    dacă element ∈ m atunci          { dacă element aparține mulțimii }
      scrie element                                  { atunci îl scriem }
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Observație

Nu orice structură de date de tip mulțime poate fi implementată cu un tip predefinit în limbajul de programare respectiv. De exemplu, chiar dacă în Pascal există posibilitatea folosirii tipului **set**, acesta nu este utilizabil în cazul unei mulțimi având cardinalitate mai mare decât 256, sau în cazul în care elementele unei mulțimi nu sunt de tip ordinal. În aceste cazuri vom implementa un tip mulțime propriu, apelând la tipul tablou (**array**). În cazul implementărilor proprii va trebui să creăm subalgoritmii care asigură operațiile specifice cum sunt reuniunea şi intersecția (vezi capitolul 6).

15.3. Probleme propuse

15.3.1. Cifre comune

Dintr-un fişier text se citesc numere întregi. Afişați cifrele comune tuturor numerelor citite şi cifrele care nu apar în componența nici unui număr din fişier.

Date de intrare

Fişierul de intrare **NUMERE.IN** conține un număr oarecare de linii pe care sunt scrise un număr oarecare de numere întregi, separate prin câte un spațiu.

Date de ieşire

În fişierul de ieşire **NUMERE.OUT** se vor scrie cifrele comune pe prima linie, în ordine crescătoare, două cifre fiind separate printr-un spațiu. Cifrele care nu apar în componența nici unui număr se vor scrie pe a doua linie, în ordine crescătoare, două cifre fiind separate printr-un spațiu. Dacă nu există cifre care să fie comune tuturor numerelor din fişierul de intrare, pe prima linie din fişierul de ieşire se va scrie mesajul `'Nu exista cifre comune.'` Dacă toate cele 10 cifre apar în scrierea numerelor din fişier, în fişierul de ieşire se va scrie mesajul `'Apar toate cifrele.'`

Restricții şi precizări

- numerele din fişierul de intrare au cel mult 9 cifre fiecare.

Exemplu

NUMERE.IN	NUMERE.OUT
124 1230894 441122 2341 112244 555421 54321	1 2 4 6 7

15.3.2. Agenda

Vasile şi-a cumpărat o agendă nouă şi şi-a notat în ea *n* numere de telefon ale cunoştinţelor sale într-o ordine dorită de el. Chiar dacă are agendă, Vasile ar vrea să „înveţe" toate numerele de telefon notate. Deoarece el a observat că cel mai uşor se pot păstra în minte acele numere de telefon care au un număr minim de cifre distincte, îşi propune să le înveţe mai întâi pe acestea. Apoi ar urma numerele de telefon din cele rămase care acum au număr minim de cifre distincte şi aşa mai departe.

Scrieţi un program care stabileşte care sunt numerele de telefon din cele existente în agenda lui Vasile, pe care le va „învăţa" în prima zi, apoi care sunt cele pe care le va reţine în a doua etc.

Date de intrare

Din fişierul de intrare **AGENDA.IN** se citesc numerele de telefon de pe linii distincte.

Date de ieşire

În fişierul de ieşire **AGENDA.OUT** se vor scrie pe aceeaşi linie, separate prin spaţiu, numerele de telefon având acelaşi număr de cifre distincte, linia încheindu-se cu numărul de ordine al zilei în care se memorează acele numere.

Restricţii şi precizări

- un număr de telefon are cel mult 9 cifre;
- $1 \le n \le 100$

Exemplu

AGENDA.IN	AGENDA.OUT
177456 234567 432167 433321 534211 645321 111233 222222	222222 1 111233 2 433321 3 177456 534211 4 234567 432167 645321 5

15.3.3. Indivizi

Într-un grup de *n* indivizi se stabilesc relații de prietenie directă şi indirectă prin intermediul altor indivizi. Dându-se pentru fiecare individ prietenii lui direcți, afişați pentru un anumit individ toți prietenii lui indirecți.

Date de intrare

De pe prima linie a fişierului de intrare **`INDIVIZI.IN`** se citeşte numărul de indivizi *n*. De pe următoarele *n* linii se citesc prietenii direcți ai fiecărui individ; indivizii din grup sunt simbolizați prin numere de la 1 la *n*. În cadrul fiecărei linii aceste numere sunt separate prin câte un spațiu. Pe ultima linie este trecut numărul de ordine al individului căruia trebuie să i se găsească prietenii indirecți.

Date de ieşire

Fişierul de ieşire **`INDIVIZI.OUT`** va conține o singură linie în care se vor scrie numerele de ordine ale prietenilor indirecți ai individului dat. Aceste numere se vor separa printr-un spațiu.

Dacă un individ nu are prieteni indirecți, atunci în fişier se va scrie mesajul `'Nu'`.

Restricții şi precizări

- relația de prietenie nu este comutativă, adică dacă individul *i* este prieten cu individul *j*, atunci nu înseamnă că individul *j* este prieten cu individul *i*;
- dacă linia *i* + 1 este vidă în fişierul de intrare, acest lucru înseamnă că individul având numărul de ordine *i* nu are prieteni direcți.

Exemplu

`INDIVIZI.IN`

```
6
2 3
3 4
5
3 5
6
1 2
3
```

`INDIVIZI.OUT`

```
1 2 4 6
```

15.3.4. Întâlnire

Într-un liceu se organizează o întâlnire cu foştii absolvenți. Din păcate persoana care are sarcina să trimită invitațiile are la dispoziție doar informații de genul: persoana *x* a fost coleg de an cu persoana *y*, persoana *z* a fost coleg de an cu persoana *x* etc.

Stabiliți lista invitaților pe baza informațiilor de tipul celor descrise, în aşa fel încât colegii de an să fie grupați.

Date de intrare
Pe prima linie a fişierului de intrare **COLEGI.IN** se află numărul *n* al perechilor de numere naturale care identifică două persoane care au fost colegi de an. Pe următoarele *n* linii se află perechi de numere naturale, despărțite printr-un spațiu, reprezentând doi colegi de an.

Date de ieşire
Fişierul de ieşire **COLEGI.OUT** va conține atâtea linii câte grupuri s-au format din numerele de ordine ale persoanelor din fişierul de intrare. Pe fiecare din următoarele linii se vor scrie numerele de ordine aparținând unui astfel de grup, despărțite prin câte un spațiu.

Restricții şi precizări
- persoanele sunt identificate prin numere de ordine între 0 şi 255;
- $1 \le n \le 1000$;
- dintr-un grup nu fac parte mai mult de 256 de parsoane.

Exemplu

COLEGI.IN
```
6
2 3
3 4
5 1
6 1
```

COLEGI.OUT
```
2 3 4
1 5 6
```

15.4. Soluțiile problemelor propuse

15.4.1. Cifre comune

În rezolvarea acestei probleme vom folosi trei mulțimi:
– mulțimea *comune* va reține cifrele comune tuturor numerelor;
– mulțimea *temp* va păstra cifrele numărului citit la pasul curent;
– mulțimea *nusunt* va păstra cifrele care nu apar în nici unul dintre numerele date.

Exemplu
Fie conținutul fişierului de intrare:
```
1234 23456
43267
12309 2345111
```
Mulțimea *comune* şi mulțimea *nusunt* se inițializează cu mulțimea cifrelor sistemului zecimal:
comune = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

nusunt = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

Numerele se citesc pe rând din fişierul de intrare. Prelucrarea unui număr înseamnă descompunerea lui în cifre şi depunerea acestora în mulțimea *temp*. Deci, în cazul numărului *n* = 1234, mulțimea *temp* va fi {1, 2, 3, 4}.

Mulțimea *comune* se transformă în intersecția dintre ea însăşi şi mulțimea *temp*, adică va conține după această primă citire doar cifrele primului număr din fişier:
comune = *comune* ∩ *temp*, deci *comune* = {1, 2, 3, 4}.

Din mulțimea *nusunt* care conținea inițial toate cifrele, se scad cifrele găsite în primul număr:
nusunt = *nusunt* – *temp*, deci *nusunt* = {0, 5, 6, 7, 8, 9}.

Procedeul continuă cu citirea următoarelor numere. Ilustrăm în tabelul de mai jos conținutul celor trei mulțimi pentru fiecare număr citit în cazul exemplului.

	n = 1234	*n* = 23456	*n* = 43267	*n* = 12309	*n* = 2345111	*Rezultate*
comune	{1,2,3,4}	{2,3,4}	{2,3,4}	{2,3}	{2,3}	2 3
nusunt	{0,5,6,7,8,9}	{0,7,8,9}	{0,8,9}	{9}	{9}	9
temp	{1,2,3,4}	{2,3,4,5,6}	{2,3,4,6,7}	{0,1,2,3,9}	{1,2,3,4,5}	

```
Subalgoritm CifreComune:
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte n
    temp ← ∅
    cât timp n > 0 execută:
      temp ← temp ∪ rest[n/10]
      n ← [n/10]
    sfârşit cât timp
    comune ← comune ∩ temp
    nusunt ← nusunt \ temp
  sfârşit cât timp
                          { după ce s-au citit toate numerele din fişier, se verifică }
                                  { conținutul mulțimilor comune şi nusunt }
                      { dacă mulțimea comune nu este vidă ⇒ există cifre comune }
  dacă comune ≠ ∅ atunci
    pentru i=0,9 execută:              { se scriu elementele mulțimii în fişier }
      dacă i ∈ comune atunci
        scrie i
      sfârşit dacă
    sfârşit pentru
  altfel                        { altfel se scrie mesajul cerut în fişierul de ieşire }
    scrie 'Nu exista cifre comune.'
  sfârşit dacă
```

```
  dacă nusunt ≠ ∅ atunci      { dacă există elemente în mulțimea nusunt, ⇒ }
                              { există cifre care nu au apărut în componența nici unui număr }
    pentru i=0,9 execută:     { acestea se scriu pe cea de-a doua linie în fișier }
      dacă i ∈ nusunt atunci
        scrie i
      sfârșit dacă
    sfârșit pentru
  altfel                                  { dacă mulțimea nusunt este vidă }
    scrie 'Apar toate cifrele.'   { se scrie în fișierul de ieșire mesajul cerut }
  sfârșit_dacă;
sfârșit subalgoritm
```

În programul Pascal mulțimile folosite au ca tip de bază subdomeniul `0..9` deoarece vor conține cifre. Numerele pe care le citim din fișier sunt de tip `Longint` și le vom reține fiecare număr citit în variabila `n`. De asemenea, amintim că în Pascal o mulțime A se creează cu ajutorul constructorului ($A = [a_1, a_2, ..., a_k]$, unde a_i, $i = 1, 2, ..., k$, reprezintă elementele mulțimii).

15.4.2. Agenda

Vom începe discuția privind rezolvarea problemei cu un exemplu:

În tabloul *nr* avem cele *n* numere de telefon citite din fișier.

nr = (177456, 234567, 432167, 433321, 534211, 645321, 01110233, 222222).

Creăm un tablou de mulțimi *cif* în care cif_i va conține cifrele distincte din numărul de telefon nr_i:

cif = ({1, 4, 5, 6, 7}, {2, 3, 4, 5, 6, 7}, {1, 2, 3, 4, 6, 7}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2, 3, 4, 5, 6}, {0, 1, 2, 3}, {2})

În tabloul *câte* păstrăm numărul de componente (cardinalitatea) fiecărei mulțimi cif_i. În exemplul considerat *câte* = (5, 6, 6, 4, 5, 6, 4, 1).

În funcție de conținutul tabloului *câte* vom alege în timpul afișării numărul de telefon nr_i având la pasul respectiv număr minim de cifre. Această cerință o realizăm generând cardinalitățile posibile ale mulțimilor. Acestea sunt numere cuprinse în mulțimea {1, 2, ..., 10}, presupunând că într-un număr de telefon putem avea toate cifrele *egale* (cazul 1) sau toate cifrele *distincte* (10; de fapt 9, conform enunțului).

Înainte de a descrie pașii algoritmului de rezolvare, vom observa că un număr de telefon poate să înceapă și cu cifra 0. Această observație este importantă, deoarece dacă numerele de telefon le-am citi ca pe niște numere, acea cifră 0 de la început s-ar pierde (025 = 25). În concluzie alegem ca modalitate de reprezentare pentru numerele de telefon șirul de caractere (*stringul*).

În program vom lucra cu tipul `str9`, reprezentând șiruri de caractere având cel mult 9 caractere (corespunzător restricției din enunț). Numerele de telefon le vom păstra într-un tablou având elemente de tip `str9`, deoarece vom avea nevoie de întreg

tabloul în momentul în care vom dori să ordonăm numerele de telefon în funcție de numărul cifrelor distincte în componența lor.

Pentru a obține cifrele distincte dintr-un număr de telefon, din cifrele fiecărui număr în parte vom crea câte o mulțime. Astfel vom genera un tablou de mulțimi.

```
Subalgoritm Generare_tablou_mulțimi(cif):
      { descompunem fiecare număr de telefon nr[i] în mulțimea de cifre cif[i] }
  pentru i=1,n execută:
    cif[i] ← ∅
  sfârșit pentru
  pentru i=1,n execută:
    pentru j=1, lungimea lui nr[i] execută: { de la prima cifră până la ultima }
                                  { al j-lea caracter din al i-lea număr de telefon }
      cif[i] ← cif[i] ∪ {nr[i][j]}
    sfârșit pentru
  sfârșit pentru
sfârșit subalgoritm
```

Stabilim cardinalitatea mulțimilor de cifre din fiecare mulțime `cif[i]` în `câte[i]`.

```
Subalgoritm Generare_tablou_dimensiuni(câte):
  pentru i=1,n execută:
    p ← 0
    pentru j='0','9' execută:
      dacă j ∈ cif[i] atunci
        p ← p + 1
      sfârșit dacă
    sfârșit pentru
    câte[i] ← p
  sfârșit pentru
sfârșit subalgoritm
```

Ordonăm numerele de telefon şi mulțimile de cifre crescător după cardinalitate.

```
Subalgoritm Ordonare(câte,nr,cif):
  pentru i=1,n-1 execută:
    pentru j=i+1,n execută:
      dacă câte[i] > câte[j] atunci
        aux1 ← nr[i]; nr[i] ← nr[j]; nr[j] ← aux1;
        aux2 ← cif[i]; cif[i] ← cif[j]; cif[j] ← aux2;
        aux3 ← câte[i]; câte[i] ← câte[j]; câte[j] ← aux3
      sfârșit dacă
    sfârșit pentru
  sfârșit pentru
sfârșit subalgoritm
```

În scopul afişării numerelor de telefon în ordine crescătoare după numărul cifrelor distincte din componenţa lor, vom căuta, pe rând, numere de telefon având 1, 2, ..., 9 cifre distincte.

```
Subalgoritm Afişare(nr,câte):
  ziua ← 0
  pentru i=1,9 execută:
    găsit ← fals
    pentru j=1,n execută:                    { studiem fiecare număr de telefon }
      dacă i=câte[j] atunci
                          { dacă am găsit un număr de telefon având i cifre distincte }
        găsit ← adevărat
        scrie nr[j]                                          { îl scriem în fişier }
      sfârşit dacă
    sfârşit pentru
                  { dacă am găsit cel puţin un număr de telefon având cu 1 mai multe }
                      { cifre decât cele propuse spre învăţare pentru ziua precedentă }
                          { creştem numărul de ordine al zilei şi îl scriem în fişier }
    dacă găsit atunci
      ziua ← ziua + 1
      scrie ziua
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

15.4.3. Indivizi

Vom începe analiza problemei cu un exemplu. În acest exemplu ne propunem afişarea prietenilor indirecţi pentru individul având numărul de ordine 3.

indivizii	1	2	**3**	4	5	6
prieteni direcţi	{2, 3}	{3, 4}	{5}	{3, 5}	{**6**}	{1, 2}
prieteni indirecţi	{2, 3}	{3, 4}	{**5, 6**}	{3, 5}	{6}	{1, 2}
	{2, 3}	{3, 4}	{**5, 6**}	{3, 5}	{6}	{**1**, **2**}
	{2, 3}	{3, 4}	{**1, 2, 5, 6**}	{3, 5}	{6}	{1, 2}
	{**2**, **3**}	{3, 4}	{**1, 2, 3, 5, 6**}	{3, 5}	{6}	{1, 2}
	{2, 3}	{**3**, **4**}	{**1, 2, 3, 4, 5, 6**}	{3, 5}	{6}	{1, 2}
se scad prietenii direcţi	{2, 3}	{3, 4}	{**1, 2, 3, 4, 6**}	{3, 5}	{6}	{1, 2}
se scade persoana *p*	{2, 3}	{3, 4}	{**1, 2, 4, 6**}	{3, 5}	{6}	{1, 2}

Persoana 3 îl are ca prieten direct pe al 5-lea. Adăugăm la mulţimea prietenilor direcţi (care de acum încolo îi va conţine şi pe cei indirecţi) mulţimea prietenilor direcţi ai celui de-al 5-lea. Mulţimea devine: {5, 6}.

Al 5-lea îl are ca prieten direct pe al 6-lea. Adăugăm la mulțimea corespunzătoare lui 3 mulțimea corespunzătoare celui de-al 6-lea. Mulțimea acum devine: {1, 2, 5, 6}.

În mulțimea rezultată apare individul 1, deci se vor adăuga şi prietenii acestuia la lista de prieteni indirecți ai lui 3. Mulțimea este: {1, 2, 3, 5, 6}.

În mulțimea rezultată apare individul 2, deci se vor adăuga şi prietenii acestuia la lista de prieteni indirecți ai lui 3. Mulțimea este: {1, 2, 3, 4, 5, 6} .

Din această mulțime se scade mulțimea prietenilor direcți ai lui 3 şi se obține mulțimea: {1, 2, 3, 4, 6}, apoi se scade şi persoana 3.

Una dintre facilitățile limbajului Pascal este faptul că ne pune la dispoziție tipul **`set`** care, poate fi utilizat (pe lângă multe altele) ca tip de bază într-un tablou (**`array`**).

Problema dată este rezolvabilă prin utilizarea unui tablou de mulțimi. Tabloul are n elemente, fiecărui individ i, (i = 1, 2, …, n) îi corespunde o poziție în tablou. Valoarea $prieteni_i$ este mulțimea formată din prietenii direcți ai lui i.

Inițial tabloul este format din n mulțimi vide. Acestea se completează cu numerele de ordine ai prietenilor direcți.

```
Subalgoritm Prietenii(prieteni):
  pentru i=1,n execută:
    prieteni[i] ← ∅      { încă nu am descoperit prietenii nici unei persoane }
  sfârşit pentru
  pentru i=1,n execută:                          { pentru fiecare persoană i }
    cât timp nu urmează marcajul de sfârşit de linie execută:
      citeşte p                { persoana i îl are printre prietenii direcți pe p }
      prieteni[i] ← prieteni[i] ∪ {p}
                              { adăugăm p la mulțimea prietenilor direcți ai lui i }
    sfârşit cât timp
  sfârşit pentru
  citeşte p                              { numărul de ordine al persoanei investigate }
                    { în temp păstrăm mulțimea persoanelor prietene direct cu p }
  temp ← prieteni[p]
  pentru i=1,n execută:                                { luăm pe rând persoanele }
    dacă i ∈ prieteni[p] atunci { dacă persoana i este prieten direct cu p, }
      prieteni[p] ← prieteni[p] ∪ prieteni[i]
       { adăugăm mulțimea prietenilor direcți ai acestuia mulțimii prietenilor lui p }
    sfârşit dacă
    pentru j=1,n execută:              { luăm încă o dată persoanele pe rând }
                          { dacă j este prieten direct sau indirect cu p, adăugăm }
      dacă j ∈ prieteni[p] atunci
                     { la mulțimea prietenilor direcți şi indirecți ai lui p se adaugă }
                                        { mulțimea prietenilor direcți ai acestuia }
```

```
        prieteni[p] ← prieteni[p] ∪ prieteni[j]
      sfârşit dacă
    sfârşit pentru
  sfârşit pentru
                                  { în final din mulțimea prietenilor direcți şi }
  { indirecți extragem mulțimea prietenilor direcți (i-am păstrat în temp) }
  prieteni[p] ← prieteni[p] - temp
  prieteni[p] ← prieteni[p] - p                  { îl extragem şi pe p }
                              { dacă mulțimea prietenilor indirecți nu este vidă }
  dacă prieteni[p] ≠ ∅ atunci
    pentru i=1,n execută:              { afişăm mulțimea prietenilor indirecți }
      dacă i ∈ prieteni[p] atunci
        scrie i
      sfârşit dacă
    sfârşit pentru
  altfel scrie 'Nu'
sfârşit subalgoritm
```

15.4.4. Întâlnire

Observăm că dacă *a* a fost coleg cu *b*, atunci şi *b* a fost coleg cu *a* (comutativitate). De asemenea, dacă *a* a fost coleg cu *b* şi *b* a fost coleg cu *c*, atunci *a* a fost coleg şi cu *c* (asociativitate). Amintim şi proprietatea că în orice situație *a* este coleg cu *a*.

```
Subalgoritm Citire(n,gr):
  citeşte n
  pentru i=1,n execută:
    citeşte a,b                                  { citim perechile de numere }
    gr[i] ← {a,b}              { formăm câte un grup din fiecare pereche citită }
  sfârşit pentru
sfârşit subalgoritm

Subalgoritm Formare_Grupuri(n,gr):
  pentru i=1,n-1 execută:
    pentru j=i+1,n execută:
                                  { dacă intersecția a două grupuri nu este vidă }
      dacă gr[i] ∩ gr[j] ≠ ∅ atunci
        gr[j] ← gr[i] ∪ gr[j]          { unim grupurile în al doilea grup }
        gr[i] ← ∅                        { şi pe primul grup îl facem vid }
      sfârşit dacă
    sfârşit pentru
  sfârşit pentru
sfârşit subalgoritm
```

```
Subalgoritm Formare_Listă(n,gr):
                                                  { elimină grupurile vide }
  nr ← 0
  pentru i=1,n execută:
    dacă gr[i] ≠ ∅ atunci
      nr ← nr + 1
      gr[nr] ← gr[i]
    sfârşit dacă
  sfârşit pentru
  n ← nr                                  { lungimea nouă, fără grupuri vide }
sfârşit subalgoritm

Subalgoritm Afişare(n,gr):
  scrie n
  am_scris ← fals
  pentru i=1,n execută:
    pentru j=0,255 execută:
      dacă j ∈ gr[i] atunci
        scrie j
        am_scris ← adevărat
      sfârşit dacă
    sfârşit pentru
    dacă am_scris atunci
      trecem la linie nouă   { dacă am afişat un grup, trecem la linie nouă în fişier }
    sfârşit dacă
    am_scris ← fals
  sfârşit pentru
sfârşit subalgoritm
```