08.10.2024

Structuri de date și algoritmi
(curs 3 - 52)

Capitolul III - Baze matematice

COMPUTER
- SCIENCE - MATH
- ENGINEERING - PHISICS

Mulțimi

- are elemente unice
- elementele apartin de obicei unui tip de date
- nu exista conceptul de duplicare
- orice valoare poate sa apartina sau nu unei multimi

O **secvență** = o colecție de elemente într-o anumită ordine, care se pot repeta.

Atenție: secvența **poate conține elemente duplicate**.

Secvența < 1, 2, 3 > este diferită de secvența < 1, 3, 2 > → ordinea contează

< un fel de listă >

O **relație** R definită pe o mulțime S, este un set de perechi ordonate, formate din elemente ale lui S

Exemplu

S= {a,b,c}

R1 = {<a,c>,<b,c>,<c,b>} este o relație

R2 = {<a,a>,<a,c>,<b,b>,<b,c>,<c,c>} este o altă relație

Notația xRy, ne arată ca elementele <x , y> sunt în relația R

Ex: 2 <= 3, <2,3> sunt in relația mai mic sau egal (sau 2 mai mic sau egal cu 3)

O **relație** poate fi:
- Reflexivă – dacă aRa pentru oricare a ∈S
- Simetrică – dacă aRb atunci și bRa, pentru oricare a ∈S
- Antisimetrică – dacă aRb și bRa, atunci a=b, pentru oricare a,b ∈S
- Tranzitivă – dacă aRb și bRc, atunci aRc, pentru oricare a,b,c ∈S

O relație este una de **echivalență** dacă este este **reflexivă, simetrică și tranzitivă**

Exemplu
- Pentru întregi = este o relație de echivalență
  1. a=a
  2. Daca a=b atunci b=a
  3. Daca a=b si b=c atunci a=c

Logaritmi

Logaritm in baza b din y este puterea la care trebuie ridicat b ca să obținem valoarea y

$\log_b y = x$

Dacă $\log_b y = x$ atunci $b^x = y$ și $b^{\log_b y} = y$

Proprietăți:

$log_A B = \frac{log_C B}{log_C A}, \quad A, B, C > 0, A \neq 1$

$\log (AB) = \log A + \log B, \quad A,B > 0$

$\log (A/B) = \log A - \log B, \quad A,B > 0$

$\log (A^B) = B \log A, \quad A,B > 0$

$\log 1 = 0$

Ex: Care este numărul minim de biți necesari pentru a reprezenta n valori distincte

Răspuns $\lceil log_2 n \rceil$ = ceiling $(log_2 n)$

Pentru 1000 de valori, avem nevoie de cel puțin 10 biți , $\lceil log_2 1000 \rceil$=10, 2^{10}=1024

ceil ↗ 2,4 → 3

floor ↘ 2,5 → 2

Sume și recurențe

Sume și recurențe:

Exemplu de utilizare: Când analizăm timpul de rulare pentru program care conține bucle, trebuie să adunăm timpii de rulare pentru fiecare iterație.

Notație:

$$\sum_{i=1}^{n} f(i)$$

Suma valorilor funcției f, pe un interval de valori întregi (i = $\overline{1, n}$)

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6} = \frac{n(2n+1)(n+1)}{6}$$

$$\sum_{i=1}^{\log n} n = n \log n$$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \; pentru \; 0 < a < 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1} pentru \; a \neq 1$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{\log n} 2^i = 2^{\log n +} \; - 1 = 2n - 1$$

$$\sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n}$$

$$\sum_{i=1}^{\log n} \frac{1}{2^i} = 2 - \frac{n+2}{2^n}$$

În matematică se spune că un **șir** a_n este definit printr-o **relație de recurență** dacă fiecare termen al acestuia poate fi scris ca o funcție de termeni anteriori

Exemplu funcția factorial:

$$\begin{cases} n! = (n-1)! \cdot n, \text{ pentru } n>1 \\ 1!=0!=1 \end{cases}$$

Funcția Fibonacci

$$\begin{cases} Fib(n)=Fib(n-1)+Fib(n-2), \text{ pentru } n>2 \\ Fib(1)=Fib(0)=1 \end{cases}$$

Exemplu de utilizare: Pentru calculul timpului de rulare a unei funcții recursive

Ex. funcția factorial

Pentru cazurile de baza 0 și 1, durata funcției este constantă, în rest timpul poate fi modelat prin ecuația:

$T(n) = T(n-1) + c, pentru\ n > 1, T(0) = T(1) = c$, unde c = constantă, și T(n) costul apelului pentru valoarea n

Ca în cazul sumei, dorim să reducem ecuația la o formă compactă

$T(0) = c$

$T(1) = c$

$T(2) = T(1) + c = c + c = 2c$

. . . .

$T(n) = T(n-1) + c = (T(n-2) + c) + c = \ldots = T(1) + (n-1)c =$

$= n \cdot c$

Tehnici pentru demonstrații matematice

Rezolvarea unei probleme are două părți: **investigarea** și **demonstrația**

Prin **investigare** căutăm o soluție și o dată gasită aceasta trebuie demonstrată ca fiind soluția corectă pentru toate cazurile vizate

Pentru **demonstrarea** unei soluții matematice avem o serie de metode standard

Cele mai folosite tehnici:
- Deducția, demostrarea **directă**
- Demonstrarea prin **contradicție**
- **Inducția matematică**

Demonstrarea directă
- Prin deducție logică, folosind logica matematică
- Ex: pentru a demonstra că două propoziții matematice P și Q sunt echivalente, se poate demonstra că P implică Q și Q implică P

Demonstrarea prin **contradicție**, demonstrația indirectă
- Demonstrația prin contradicție este o formă de demonstrație care stabilește adevărul sau validitatea unei propoziții
- Demonstrarea prin contradicție, pleacă de la ipoteza că teorema este **falsă**, apoi folosind logica arată că asumarea propoziției ca fiind falsă duce la o **contradicție**
- Pentru a demonstra că o soluție nu este corectă ajunge să aducem un contraexemplu
- Ca urmare, niciun numar de exemple pozitive nu pot demonstra o teoremă

Exemplu
- Propoziția: Nu există o valoare maximă pentru numerele naturale
- Demonstrație:

  Pasul 1 – Negarea propoziției și asumarea ei ca ipoteză. Există o valoare maximă pentru numerele naturale (o notăm cu M)

  Pasul 2 – Demonstrăm că ipoteza duce la o contradicție.

  N = M+1, N este tot un numar natural pentru că este suma a două numere naturale

  N>M => Ipoteza este falsă

Inducția matematică

Este o modalitate de demonstrație utilizată în matematică pentru a stabili dacă o anumită propoziție este valabilă pentru un număr nelimitat de cazuri, contorul cazurilor parcurgând toate **numerele naturale**

Poate fi folosită pentru o gamă largă de teoreme

Este folosită în recursivitate

Conține doi pași principali: **cazul inițial** și **pasul de inducție**

Fie Trm deorema de demonstrat pentru un parametru pozitiv n. Prin inducție matematică se va demonstra că Trm este adevărată pentru orice valoare n, pentru n>c (unde c este o constantă), dacă următoarele condiții sunt adevărate:

1. Cazul inițial: Trm(c) – este adevărată
2. Pasul de inducție: Dacă Trm(n-1) este adevărată, atunci și Trm(n) este adevărată

Pentru pasul 2, avem varianta de inducție puternică (strong induction) prin care demonstrăm că dacă Trm(k) este adevărată pentru oricare k, c<=k<=n, atunci Trm(n) este adevărată.

Recursivitatea este o metodă de rezolvare a problemelor, în care găsirea soluției se bazează pe împărțirea problemei în instanțe mai simple

În oricare din variantele inducției, avem o asemănare puternică între demonstrarea prin inducție și **recursivitate**:

- Ambele au cazuri simple/ de bază
- Ambele se bazează pe instanțe simplificate ale aceleași probleme

Exemplu

Suma lui Gauss

S(n) = n(n+1)/2, pentru n>=0

Pasul 1: Caz de baza, n=1, S(1)=1(1+1)/2=1 (Adevarat)

Pasul 2: S(n-1) -> S(n)

S(n-1) = (n-1)(n-1+1)/2 = (n-1)n/2

Dar S(n) = S(n-1) + n

S(n) = (n-1)n/2 +n = $(n^2 - n + 2n)$/2=n(n+1)/2

Exemple

Demonstrația T(n) = n·c, pentru timpul de execuție al funcției factorial recursive, pentru n >=1

n!= $\begin{cases} 1, n = 1 \\ n * (n-1)!, n > 1 \end{cases}$

Pasul 1: T(1) = c, caz de bază

Pasul 2: T(n-1) -> T(n),(daca T(n-1) =(n-1)c => T(n)=nc)

T(1)=c, T(2)=c+c=2c, ...T(k)=kc =>

T(n-1) = (n-1)c

T(n) = T(n-1)+c, (din definitie)

T(n) = (n-1)c + c = n·c

```
long factorial(int n)
{
    if (n == 1)
        return 1;  //conditia de oprire
    else
    return(n * factorial(n - 1));
}
```

În mod asemănător cu inducția matematică și recursivitatea conține două cazuri (doi pași):

- Cazul de bază
  - rezolvă problema pentru cel mai mic/ cel mai simplu set de date
- Cazul recursiv
  - definește ipoteza (pentru inducție matematică) - presupunem rezolvă problema prin apelul funcției pe un set de date restrâns (simplificat)
  - bazat pe ipoteză, combină apelurile pe seturile restrânse de date, pentru a rezolva problema pentru setul de date de intrare dat

Ex1: Să se scrie o funcție recursivă care afișează în ordine crescătoare numerele naturale din intervalul [start, end], unde inceput si sfarsit sunt parametri de intrare pentru functia dată:

void printAsc(int start, int end);

printAsc(1,4) => 1 2 3 4

Determinarea dimensiunii setului de date

end – start + 1

Pasul 1: determinarea cazului de baza

start == end

Pasul 2:

- Definirea ipotezei: Dacă apelăm funcția pe un set interval restrâns de date, funcția va afișa numerele din acel interval în ordine crescătoare
- Bazat pe ipoteză, se implementează soluța pentru setul de date de intrare

```
//Varianta 1
void printAsc(int start, int end)
{
   if (start == end)
         printf("%d ", start);  //cazul de baza
   else
   {
         printAsc(start, end - 1);   //ipoteza/apelul recursiv
         printf("%d ", end);
   }
}
```

```
//Varianta 2
void printAsc(int start, int end)
{
   if (start == end)
         printf("%d ", end);  //cazul de baza
   else
   {
         printf("%d ", start);
         printAsc(start+1, end);   //ipoteza/apelul recursiv
   }
}
```

```
//Varianta 3
void printAsc(int start, int end)
{
   if (start == end)
         printf("%d ", start);  //cazul de baza
   else
   {
               printAsc(start, ((start+end)/2));
//ipoteza/apelul recursiv
               printAsc(((start+end)/2 +1), end);
   }
}
```