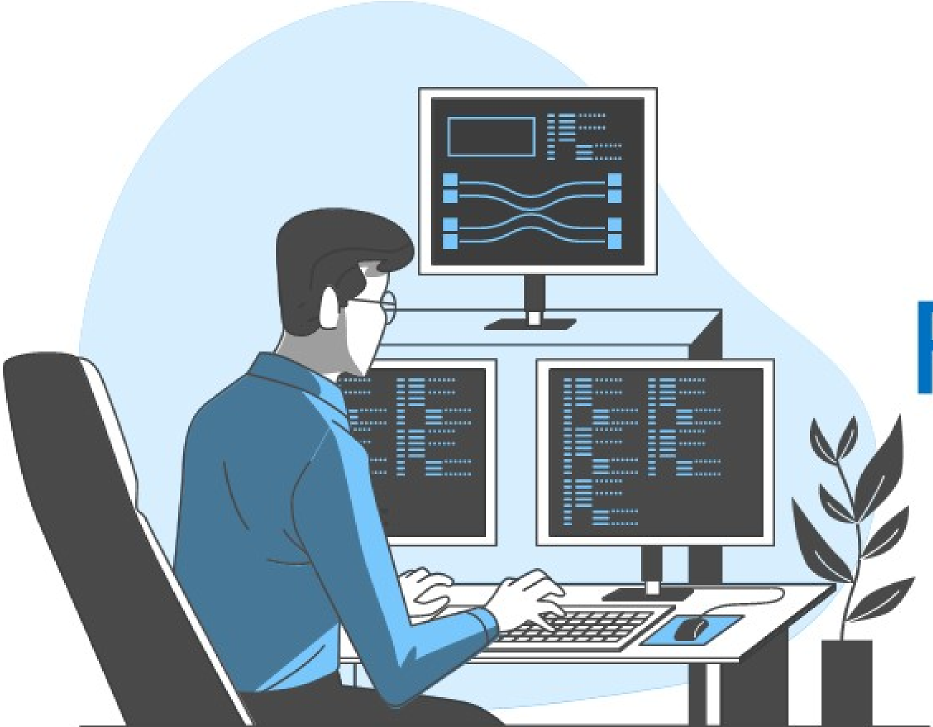


Algoritmi Fundamentali

Lector dr.
Dorin IORDACHE

Cursul nr. 2

Verificarea corectitudinii algoritmilor

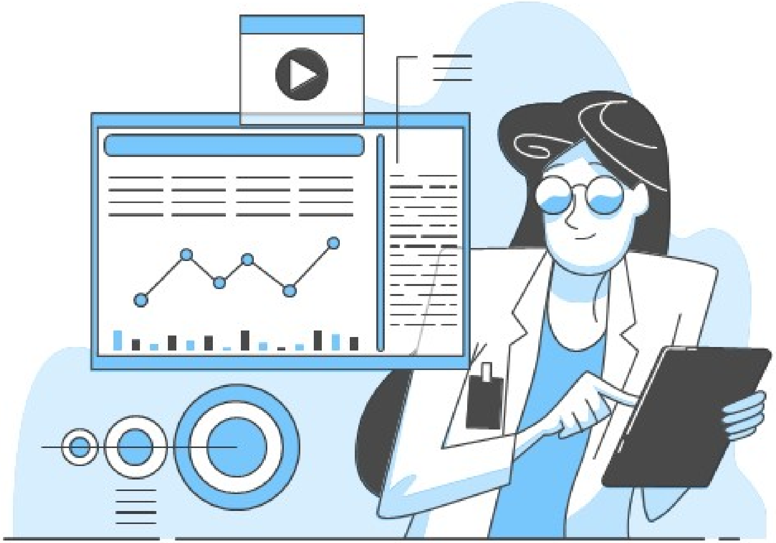
Agenda

01 Analiza algoritmilor

02 Verificarea algoritmilor

03 Reguli

04 Exemple



01

Analiza algoritmilor

ALGORITM

Analiza algoritmilor se refera la doua aspecte principale:

Corectitudine:
Se analizeaza daca algoritmul produce rezultatul dorit dupa efectuarea unui numar finit de operatii

Eficienta
Se estimeaza volumul de resurse (spatiu memorie si timp de executie) necesare pentru executia algoritmului

ALGORITM

Corectitudine

- **Parțială** — Verificare empirică
  - Intrare validă și încheie execuția
- **Totală** — Verificare prin inducție
  - Intrare validă, încheie execuția și rezultatul este corect întotdeauna

ALGORITM – corectitudine

Exista doua modalitati principale de a verifica corectitudinea unui algoritm:

Experimentală (prin testare): algoritmul este executat pentru un set de instante ale datelor de intrare

Formală (prin demonstrare): se demonstreaza ca algoritmul produce rezultatul corect pentru orice instanta a datelor de intrare care satisface cerintele problemei

Avantaje și Dezavantaje

	Experimentală	Formală
Avantaje	• simplă • relativ ușor de aplicat	• garantează corectitudinea
Dezavantaje	• nu garantează corectitudinea …	• destul de dificilă • nu poate fi aplicată algoritmilor complecși

02

Verificarea algoritmilor

Pre- și Post- condiții

Preconditii = proprietati satisfacute de catre datele de intrare
Postconditii = proprietati satisfacute de catre datele de iesire (rezultate)

Exemplu: *Sa se determine valoarea minima, m, dintr-o secventa (tablou) nevida, x[1..n]*

Preconditii: $n>=1$ (secventa este nevida)

Postconditii: $m=\min\{x[i]; 1<=i<=n\}$
(variabila m contine cea mai mica valoare din x[1..n])

Pre- și Post- condiții

Verificarea corectitudinii partiale = se demonstreaza ca daca algoritmul se termina dupa un numar finit de prelucrari atunci conduce de la preconditii la postconditii

Corectitudine totala = corectitudine partiala + finitudine

Etape intermediare in verificarea corectitudinii:

- analiza starii algoritmului , si
- a efectului pe care il are fiecare pas de prelucrare asupra starii acestuia

Verificare empirica

```cpp
#include <iostream>
using namespace std;

int main() {
  int numere[4] = {13, 4, 24, 7};
  int maxNum = -1;
for (int i = 0; i < sizeof(numere)/ sizeof(int); i++) {
  if (numere[i] > maxNum) {
    maxNum = numere[i];
  }
}
cout << maxNum;

}
```

Output

```
24
```

Corect

Verificare empirica

```cpp
#include <iostream>
using namespace std;

int main() {
  int numere[4] = {-13, -4, -24, -7};
  int maxNum = -1;
for (int i = 0; i < sizeof(numere)/ sizeof(int); i++) {
  if (numere[i] > maxNum) {
    maxNum = numere[i];
  }
}
cout << maxNum;

}
```

Output

```
-1
```

Incorect

Ce este greșit ???

Verificare prin inductie

```cpp
#include <iostream>
using namespace std;

int main() {
  int numere[4] = {-13, -4, -24, -7};
  int maxNum = numere[0]; // prima ipoteza a inductiei
for (int i = 0; i < sizeof(numere)/ sizeof(int); i++) {
  if (numere[i] > maxNum) {
    maxNum = numere[i];
  }
}
cout << maxNum;

}
```

Output

```
-4
```

Corect total

Starea unui algoritm

Stare algoritm= set de valori corespunzatoare variabilelor utilizate in cadrul algoritmului

De-a lungul executiei algoritmului starea acestuia se modifica intrucat variabilele isi schimba valorile

Algoritmul poate fi considerat corect daca la sfarsitul executiei prelucrarilor starea lui implica postconditiile (adica variabilele corespunzatoare datelor de iesire contin valorile corecte)

Starea unui algoritm

Exemplu: Rezolvarea ecuatiei ax=b, a<>0

Date de intrare: a
Preconditii: a<>0

Data de iesire: x
Postconditii: x satisface ax=b

Algoritm:

```
Solutie (real a,b)
    real x
    x←b/a
  return x
```

Stare algoritm

Valori curente pt. a și b

a=a0, b=b0, x nedefinit
a=a0, b=b0, x=b0/a0

ax=b

Aserțiuni și adnotări

Aserțiune = afirmație (valoare de adevăr) privind starea algoritmului

Aserțiunile sunt utilizate pentru a adnota algoritmii

Adnotarea este utilă atât în

- Verificarea corectitudinii

cât și ca

- Instrument de documentare a programelor

Aserțiuni și adnotări

Preconditii: a,b,c sunt numere reale distincte
Postconditii: m=min(a,b,c)

```
min (real a,b,c)       //{a<>b, b<>c, c<>a}

IF a<b THEN            //{a<b}
   IF a<c THEN
          m ← a        //{a<b, a<c, m=a}   ⟹  m=min(a,b,c)
   ELSE
          m ← c        //{a<b, c<a, m=c}   ⟹  m=min(a,b,c)
   ENDIF
ELSE                   //{b<a}
   IF b<c THEN
          m ← b        //{b<a, b<c, m=b}   ⟹  m=min(a,b,c)
   ELSE
          m ← c        //{b<a, c<b, m=c}   ⟹  m=min(a,b,c)
   ENDIF
ENDIF
RETURN m
```

Aserțiuni și adnotări

Preconditii: a,b,c sunt numere reale distincte
Postconditii: m=min(a,b,c)

Alta varianta de determinare a minimului a trei valori

```
min (real a,b,c)                 //{a<>b, b<>c, c<>a}

m ← a                            // m=a
IF m>b THEN                      // m<=a, m<=b
         m ← b
ENDIF    IF m>c THEN             // m<=a, m<=b, m<=c
         m ← c
ENDIF
RETURN m
```

m=min(a,b,c)

Etapele verificării corectitudinii

Identificarea precondițiilor și a postcondițiilor

Adnotarea algoritmului cu aserțiuni astfel încât:

- Precondițiile să fie satisfăcute
- Aserțiunea finală să implice postcondițiile

Se demonstrează că fiecare pas de prelucrare asigură modificarea stării algoritmului astfel încât aserțiunea următoare să fie adevărată

Notații

Considerăm următoarele notații:

P - precondiții
Q - postcondiții
A - algoritm

Tripletul (P,A,Q) reprezinta un algoritm corect daca pentru datele de intrare ce satisfac preconditiile P, algoritmul:

- Conduce la postconditiile Q
- Se opreste dupa un numar finit de prelucrări

Notatie: $P \xrightarrow{A} Q$

Verificarea algorimilor

- exemplu - empiric

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

$$\text{Case } n = 1 : \sum_{i=1}^{1} i = \frac{1(1+1)}{2} \Rightarrow 1 = 1$$

$$\text{Case } n = 5 : \sum_{i=1}^{5} i = \frac{5(5+1)}{2} = 1+2+3+4+5 = 15 \Rightarrow 15 \leq 15$$

$$\text{Case } n = 30 : \sum_{i=1}^{30} i = \frac{30(30+1)}{2} \Rightarrow 465 = 465 \text{ Check my math on your own!}$$

```cpp
#include <iostream>
using namespace std;

int main() {
  int n=5;
  int sum = 0;
  for (int i = 0; i <=n; i++)
        sum += i;
  cout << sum;

}
```

Output

15

```cpp
#include <iostream>
using namespace std;

int main() {
  int n=30;
  int sum = 0;
  for (int i = 0; i <=n; i++)
        sum += i;
  cout << sum;

}
```

Output

465

Verificarea algorimilor

- exemplu - inductie

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

$$1+2+3+...(n-1)+n = \frac{n(n+1)}{2}$$

$$1+2+3+...+((n+1)-1)+(n+1) = \frac{(n+1)[(n+1)+1]}{2}$$

$$1+2+3+...+n+(n+1) = \frac{(n+1)(n+2)}{2}$$

$$(1+2+3+...+n)+(n+1) = \frac{(n+1)(n+2)}{2}$$

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

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

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

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

```cpp
#include <iostream>
using namespace std;

int main() {
  int n=30;
  int sum = (n*(n+1)/2);

  cout << sum;

}
```

Output

```
465
```

03
Reguli verificare algoritmi

Reguli pentru verificarea corectitudinii unui algoritm

Pentru a demonstra ca un algoritm este corect pot fi utile cateva reguli specifice principalelelor tipuri de prelucrari:

- Prelucrări secvențiale
- Prelucrări condiționale
- Prelucrări repetitive

Regula prelucrării secvențiale

Structura

A:

$\{P_0\}$

A_1

$\{P_1\}$

…

$\{P_{i-1}\}$

A_i

$\{P_i\}$

…

$\{P_{n-1}\}$

A_n

$\{P_n\}$

Regula:

Daca

P » P_0

$P_{i-1} \rightarrow P_i$, $i=1..n$

P_n » Q

atunci

$$P \xrightarrow{A} Q$$

Cum interpretăm ?

Dacă

- Preconditiile implica asertiunea initiala,
- Fiecare actiune implica asertiunea urmatoare
- Asertiunea finala implica postconditiile

Atunci secventa de prelucrari este corecta

Regula prelucrării secvențiale

Problema: Fie x si y doua variabile avand valorile a si b.
Sa se interschimbe valorile celor doua variabile.

P: {x=a, y=b}

Q: {x=b, y=a}

Varianta 1:

{x=a, y=b, aux nedefinit}

aux ← x

{x=a, y=b, aux=a}

x ← y

{x=b, y=b, aux=a}

y ← aux

{x=b, y=a, aux=a} » Q

Varianta 2 (a si b sunt numere):

{x=a, y=b}

x ← x+y

{x=a+b, y=b}

y ← x-y

{x=a+b, y=a}

x ← x-y

{x=b, y=a} » Q

Regula prelucrării secvențiale

Problema: Fie x si y doua variabile avand valorile a si b.
Sa se interschimbe valorile celor doua variabile.

P: {x=a, y=b}

Q: {x=b, y=a}

Ce se poate spune
despre această variantă ?

Varianta 3:

{x=a, y=b}

$x \leftarrow y$

{x=b, y=b}

$y \leftarrow x$

{x=b, y=b} ? Q

Regula prelucrării condiționale

Structura

A:

$\{P_0\}$

IF c

THEN

$\{c, P_0\}$

A_1

$\{P_1\}$

ELSE

$\{\text{NOT } c, P_0\}$

A_2

$\{P_2\}$

Regula:

Daca

- c este bine definita
- c AND $P_0 \xrightarrow{A_1} P_1$
- P_1 » Q
- NOT c AND $P_0 \xrightarrow{A_2} P_2$
- P_2 » Q

Atunci

$$P \xrightarrow{A} Q$$

…

Cum interpretăm ?

Conditia c (expresie logica) este bine definita daca poate fi evaluata

Ambele ramuri ale structurii conduc la postconditii

Regula prelucrării condiționale

Problema: calculeaza minimul a doua valori
Preconditii: a<>b
Postconditii: m=min{a,b}

{a<>b}
IF a<b
    THEN
        {a<b, m nedefinita}
        m   a
        {a<b, m=a}
    ELSE
        {b<a, m nedefinita}
        m   b
        {b<a, m=b}

Dacă

{a<b, m=a} implica m=min{a,b}

și

{b<a, m=b} implica m=min{a,b}

Atunci algoritmul satisface specificațiile

Regula prelucrării repetitive

Verificarea corectitudinii structurii secventiale si a celei conditionale este simpla …

Verificarea corectitudinii prelucrarilor repetitive nu este la fel de simpla …

La nivel informal un ciclu este corect daca are proprietatile:

- Daca se termina conduce la satisfacerea postconditiilor
- Se termina dupa un numar finit de pasi

Daca este satisfacuta doar prima proprietate ciclul este considerat corect

Corectitudinea partiala poate fi demonstrata folosind inductia matematica sau asa numitele proprietati invariante

Corectitudinea totala necesita si demonstrarea finitudinii

Proprietăți invariante

Consideram urmatorul ciclu WHILE :

P » {I}
WHILE c DO
{c,I}
A
{I}
ENDWHILE
{NOT c, I} » Q

Definitie:

O proprietate invarianta este o afirmatie privind starea algoritmului (asertiune) care satisface:

1. Este adevarata la intrarea in ciclu
2. Ramane adevarata prin executia corpului ciclului
3. Cand c devine falsa proprietatea implica postconditiile

Daca poate fi identificata o proprietate invarianta pentru un ciclu atunci ciclul este partial corect

Proprietăți invariante

Preconditii: x[1..n] tablou nevid(n>=1)

Postconditii: m=min{x[i]|1<=i<=n}

```
m ← x[1]
FOR i:=2,n DO
  IF x[i]<m THEN
        m ← x[i]
  ENDIF
ENDFOR
```

```
i ← 1
m ← x[i]
WHILE i<n DO
   i ← i+1
   IF x[i]<m THEN
      m ← x[i]
   ENDIF
ENDWHILE
```

Proprietăți invariante

P: $n>=1$

Q: $m=\min\{x[i];\ i=1..n\}$

```
i ← 1
m ← x[i]
     {m=min{x[j]; j=1..i}}
WHILE i<n DO {i<n}
   i ← i+1
     {m=min{x[j]; j=1..i-1}}
   IF x[i]<m THEN
          m ← x[i]
    {m=min{x[j]; j=1..i}}
   ENDIF
ENDWHILE
```

Invariant:

$m=\min\{x[j];\ j=1..i\}$

De ce ? Pentru ca …

- Atunci cand $i=1$ si $m=x[1]$ proprietatea considerata invarianta este adevarata
- Dupa executia corpului ciclului proprietatea $m=\min\{x[j];\ j=1..i\}$ ramane adevarata
- La iesirea din ciclu (cand $i=n$) proprietatea invarianta devine $m=\min\{x[j];\ j=1..n\}$ care este chiar postconditia

Proprietăți invariante

P: $n>=1$ Q: $m=min\{x[i]; i=1..n\}$

Alta varianta de determinare a minimului

```
m ← x[1]
i ← 2
     {m=min{x[j]; j=1..i-1}}
WHILE i<=n DO {i<=n}
  IF x[i]<m THEN
       m ← x[i]
    {m=min{x[j]; j=1..i}}
   ENDIF
   i ← i+1
     {m=min{x[j]; j=1..i-1}}
ENDWHILE
```

Invariant:

m=min{x[j]; j=1..i}

De ce ? Pentru ca …

- daca i=2 si m=x[1] atunci invariantul este satisfacut
- Cat timp i<=n executia corpului ciclului asigura conservarea proprietatii invariante
- La iesirea din ciclu are loc i=n+1 ceea ce implica m=min{x[j]; j=1..n}, adica postconditia

Proprietăți invariante

Problema: Fie x[1..n] un tablou care contine valoarea x0. Sa se determine cea mai mica valoare a lui i pentru care x[i]=x0

P: n>=1 si exista1<= k <= n astfel incat x[k]=x0

Q: x[i]=x0 si x[j]<>x0 pentru j=1..i-1

```
i ← 1
WHILE x[i]<>x0 DO
   i ← i+1
ENDWHILE
PRINT i
```

?

Proprietăți invariante

Problema: Fie x[1..n] un tablou care contine valoarea x0. Sa se determine cea mai mica valoare a lui i pentru care x[i]=x0

P: n>=1 si exista1<= k <= n astfel incat x[k]=x0

Q: x[i]=x0 si x[j]<>x0 pentru j=1..i-1

```
i ← 1
    {x[j]<>x0  for j=1..0}
WHILE x[i]<>x0  DO
    {x[i]<>x0,  x[j]<>x0  for j=1..i-1}
   i ← i+1
    {x[j]<>x0  for j=1..i-1}
ENDWHILE
PRINT  i
```

Proprietatea invarianta:

x[j]<>x0 for j=1..i-1

De ce ? Pentru ca …

- daca i=1 atunci domeniul de valori pentru j (j=1..0) este vid deci orice afirmatie referitoare la j din acest domeniu este adevarata
- Presupunem ca x[i]<>x0 si invariantul e adevarat. Atunci x[j]<>x0 for j=1..i
- Dupa i:=i+1 se obtine ca x[j]<>x0 pt j=1..i-1
- La final, cand x[i]=x0 rezulta postconditia

Proprietăți invariante

Proprietatile invariante nu sunt utile doar pentru verificarea corectitudinii ci si pentru proiectarea ciclurilor

La modul ideal la proiectarea unui ciclu ar trebui:
Prima data se identifica proprietatea invarianta
Dupa aceea se construieste cicul

Problema: sa se calculeze suma primelor n valori naturale
Preconditie: n>=1 Postconditie: S=1+2+…+n

Ce proprietate ar trebui sa satisfaca S dupa executia pentru a i-a oara a corpului ciclului?
Invariant: S=1+2+…+i

Ideea pentru proiectarea ciclului:
- Prima data se pregateste termenul de adaugat
- Apoi se aduna termenul la suma

Proprietăți invariante

Algoritm:

```
i ← 1
S ← 1
   {S=1+2+…+i}
WHILE i<n DO
         {S=1+2+…+i}
    i ← i+1
         {S=1+2+…+i-1}
    S ← S+i
         {S=1+2+…+i}
ENDWHILE
-----------
{i=n, S=1+2+…+i} » S=1+…+n
```

Algoritm:

```
S ← 0
i ← 1
   {S=1+2+…+i-1}
WHILE i<=n DO
         {S=1+2+…+i-1}
    S ← S+i
         {S=1+2+…+i}
    i ← i+1
         {S=1+2+…+i-1}
ENDWHILE
-----------
{i=n+1, S=1+2+…+i-1} » S=1+…+n
```

Funcții de terminare

Pentru a demonstra finitudinea unei prelucrari repetitive este suficient sa se identifice o functie de terminare

Definitie:

O functie F:N→N (care depinde de contorul ciclului) este o functie de terminare daca satisface urmatoarele proprietati:

1. F este strict descrescatoare
2. dacă c este adevarata atunci F(p)>0 si
   dacă F(p)=0 atunci c este falsa

Observatie:

F depinde de contorul (implicit) al ciclului (notat in continuare cu p). La prima executie a corpului ciclului p este 1, la a doua executie a corpului ciclului este 2 s.a.m.d …)

F fiind strict descrescatoare va ajunge la 0 iar atunci cand devine 0 conditia de contnuare (conditia c) devine falsa, astfel ca ciclul se termina.

Funcții de terminare

Exemplu: $S=1+\ldots+n$

Prima varianta:

```
i ← 1
S ← 1
WHILE i<n DO
    i ← i+1
        {i_p=i_{p-1}+1}
    S ← S+i
ENDWHILE
```

$F(p)=n-i_p$

$F(p)=n-i_{p-1}-1=F(p-1)-1<F(p-1)$

$i<n \Rightarrow F(p)>0$

$F(p)=0 \Rightarrow i_p=n$

A doua varianta:

```
S ← 0
i ← 1
WHILE i<=n DO
    S ← S+i;
    i ← i+1
        {i_p=i_{p-1}+1}
ENDWHILE
```

$F(p)=n+1-i_p$

$F(p)=n+1-i_{p-1}-1=F(p-1)-1<F(p-1)$

$i<=n \Rightarrow F(p)>0$

$F(p)=0 \Rightarrow i_p=n+1$

Funcții de terminare

Exemplu: Fie x[1..n] un tablou care contine valoarea x0 pe cel putin o pozitie; sa se determine cel mai mic indice k pentru care x[k]=x0.

```
i ← 1
WHILE x[i]<>x0 DO
      i ← i+1
        {i_p=i_{p-1}+1}
ENDWHILE
PRINT i
```

$\{i_p=i_{p-1}+1\}$

Fie k prima aparitie a lui x0 in x[1..n]

$F(p)=k-i_p$

$F(p)=k-i_{p-1}-1=F(p-1)-1<F(p-1)$

$x[i]<>x0 \Rightarrow i_p<k \Rightarrow F(p)>0$

$F(p)=0 \Rightarrow i_p=k \Rightarrow x[i]=x0$

Exemplu

Analiza corectitudinii algoritmului lui Euclid (varianta 1)

```
cmmdc(a,b)
d ← a
i ← b
r ← d MOD i
WHILE r<>0 DO
  d ← i
  i ← r
  r ← d MOD i
ENDWHILE
RETURN i
```

P: a=a0, b=b0

Q: i=cmmdc(a0,b0)

Invariant: cmmdc(d,i)=cmmdc(a0,b0)

1. d=a=a0, i=b=b0 => cmmdc(d,i)=cmmdc(a0,b0)
2. $cmmdc(d_p,i_p)=cmmdc(i_p,d_p \text{ MOD } i_p)= cmmdc(d_{p+1},i_{p+1})$
3. r=0 => i divide pe d => cmmdc(d,i)=i

Functie de terminare: $F(p)=r_p$

Exemplu

Analiza corectitudinii algoritmului lui Euclid (varianta 2)

```
cmmdc(a,b)
WHILE a<>0 AND b<>0 DO
   a←a MOD b
   IF a<>0 THEN
      b ← b MOD a
   ENDIF
ENDWHILE
IF a<>0 THEN RETURN a
         ELSE RETURN b
ENDIF
```

Invariant: cmmdc(a,b)=cmmdc(a0,b0)

1. p=0=> a=a0,b=b0=>
   cmmdc(a,b)=cmmdc(a0,b0)
2. $cmmdc(a0,b0)=cmmdc(a_{p-1},b_{p-1})$=>
   $cmmdc(a_{p-1},b_{p-1})= cmmdc(b_{p-1},a_p)=cmmdc(a_p,b_p)$
3. $a_p=0$ => $cmmdc(a,b)=b_p$
   $b_p=0$ => $cmmdc(a,b)=a_p$

Functie de terminare: $F(p)=min\{a_p,b_p\}$

(b0>a1>b1>a2>b2>…. => F(p) descresc.)

Exemplu

Analiza corectitudinii algoritmului lui Euclid (varianta 3)

```
cmmdc(a,b)
WHILE a<>b
  IF a>b THEN
      a ← a-b
  ELSE
      b ← b-a
  ENDIF
ENDWHILE
RETURN a
```

Invariant: $cmmdc(a,b)=cmmdc(a_p,b_p)$

1. $p=0 \Rightarrow a_p=a, b_p=b \Rightarrow cmmdc(a,b)=cmmdc(a_p,b_p)$
2. $cmmdc(a,b)=cmmdc(a_{p-1},b_{p-1}) \Rightarrow$

Daca $a_{p-1} > b_{p-1}$

$cmmdc(a_{p-1},b_{p-1})=cmmdc(a_{p-1}-b_{p-1},b_{p-1})=cmmdc(a_p,b_p)$

altfel

$cmmdc(a_{p-1},b_{p-1})=cmmdc(a_{p-1},b_{p-1}-a_{p-1})=cmmdc(a_p,b_p)$

3. $a_p=b_p \Rightarrow cmmdc(a,b)=cmmdc(a_p,b_p)=a_p$

Factorial n

Empiric

$n! = 1 \cdot . . . . \cdot (n - 1) \cdot (n - 2) \cdot \cdot n$

```cpp
#include <iostream>
using namespace std;
int factorial(int n)
{
  int out =1;
  for(int i=1;i<=n;i++)
      out *= i;
  return out;
}
int main() {
  int n=4;
  cout << factorial(n);

}
```

Output

24

$n = 4 \Rightarrow n! = 24$

Inductie

$(n+1)! = (n)! \cdot (n+1)$

```cpp
#include <iostream>
using namespace std;
int factorial(int n)
{
  int out =1;
  for(int i=1;i<=n;i++)
      out *= i;
  return out;
}
int main() {
  int n=4;
  cout << factorial(3)*n;
}
```

Output

24

$n = 4 \Rightarrow n! = 3! \cdot 4 = 6 \cdot 4 = 24$

Iterativ vs recursiv

Factorial n

$$n! = 1 \cdot \ldots \cdot (n-1) \cdot (n-2) \cdot \cdot n$$

```
int factorial(n){
  int out = 1;
  for (int i=1; i<= n; i++){
    out *= i;
  }
  return out;
}
```

```
#include <iostream>
using namespace std;
int factorial(int n)
{
  int out =1;
  for( int i=1;i<=n;i++)
      out*=i;
  return out;
}
int main() {
  int n=4;

  cout << factorial(n);

}
```

Output

24

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$$

```
int factorial(n){
  if (n=1):
    return 1;
  return n * factorial(n);
}
```

```
#include <iostream>
using namespace std;
int factorial(int n)
{
  int out =1;
  if(n==1)
    return 1;
  return n*factorial(n-1);
}
int main() {
  int n=4;
  cout << factorial(n);

}
```

Output

24

Verificare corectitudine algoritm de sortare

```
INSERTIE (A, n)
 for j ← 2 to n do
        cheie ← A[j]
        i ← j – 1
        while i > 0 && cheie < A[i] do
                A[i + 1] ← A[i]
                i ← i – 1
        endwhile
        A[i + 1] ← cheie
end Insertie
```

Initializare

Invariantul buclei înainte de inițializare.
Pentru j = 2, subvectorul A[1…j – 1] conține un singur element, adică A[1]. Demonstrează că invariantul este adevărat înainte de prima iterație.

Prelucrare

La fiecare iterație, algoritmul determină poziția corectă a cheii pentru a introduce elementul A[j] prin mutarea elementelor A[j – 1], A[j – 2],... După buclă, elementul A[j] va fi introdus în poziția corectă. După buclă, A[1…j] conține elemente la fel ca în A[1…j] înainte de buclă, dar în ordine sortată. Astfel, invariantul este adevărat.

Terminare

Bucla se termină când j = n. Fiecare iterație inserează A[j] în locația corectă, astfel încât după n iterații, toate elementele vor fi în poziția corectă. După buclă, A[1…n] conține elemente la fel ca în A[1…n] de dinainte, dar acum vor fi în ordine sortată.

Sumar

Verificarea corectitudinii algoritmilor presupune:

- Sa se demonstreze ca prin executia instructiunilor se ajunge de la preconditii la postconditii (corectitudine partiala)
- Sa se demonstreze ca algoritmul se termina dupa un numar finit de pasi

Invariantul unui ciclu este o proprietate (referitoare la starea algoritmului) care satisface urmatoarele conditii:

- Este adevarata inainte de intrarea in ciclu
- Ramane adevarata prin executia corpului ciclului
- La sfarsitul ciclului implica postconditiile

Mulțumesc

pentru atenție!

dorin.iordache@365.univ-ovidius.ro