CURS 3:

Verificarea corectitudinii algoritmilor

Structura

- Analiza algoritmilor
- Noțiuni de bază
- Etapele verificării corectitudinii
- Reguli pentru verificarea corectitudinii
- Exemple

Analiza algoritmilor

Analiza algoritmilor se referă la două aspecte principale:

- Corectitudine:
  - Se analizează dacă algoritmul produce rezultatul dorit după efectuarea unui număr finit de operații

- Eficiența:
  - Se estimează volumul de resurse (spațiu de memorie și timp de execuție) necesare pentru execuția algoritmului

Verificarea corectitudinii

Exista două modalități principale de a verifica corectitudinea unui algoritm:

- Experimentală (prin testare): algoritmul este executat pentru un set de instanțe ale datelor de intrare
  - De cele mai multe ori este imposibil să se analizeze toate cazurile posibile
- Formală (prin demonstrare): se demonstrează că algoritmul produce rezultatul corect pentru orice instanță a datelor de intrare care satisface cerințele 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

Observație: în practică se folosește o variantă hibridă în care verificarea prin testare poate fi ghidată de rezultate parțiale obținute prin analiza formală a unor module de dimensiuni mici

Noțiuni de bază

- Precondiții și postcondiții
- Starea unui algoritm
- Aserțiuni
- Adnotare

Precondiții și postcondiții

- Precondiții = proprietăți satisfăcute de către datele de intrare
- Postcondiții= proprietăți satisfăcute de către datele de ieșire (rezultate)

Exemplu: *Să se determine valoarea minimă, m, dintr-o secvență (tablou) nevidă, x[1..n]*

Precondiții: n>=1 (secvența este nevidă)

Postcondiții: m=min{x[i]; 1<=i<=n} (sau m <=x[i] pentru orice i)

(variabila m conține cea mai mică valoare din x[1..n])

Precondiții și postcondiții

Verificarea corectitudinii parțiale = se demonstrează că dacă algoritmul se termină după un număr finit de prelucrări atunci conduce de la precondiții la postcondiții

Corectitudine totală = corectitudine parțială + finitudine

Etape intermediare în verificarea corectitudinii:

- analiza stării algoritmului
- și a efectului pe care îl are fiecare pas de prelucrare asupra stării algoritmului

Starea unui algoritm

- Stare algoritm= set de valori corespunzătoare variabilelor utilizate în cadrul algoritmului

- De-a lungul execuției algoritmului starea acestuia se modifică întrucât variabilele își schimbă valorile

- Algoritmul poate fi considerat corect dacă la sfârșitul execuției prelucrărilor starea lui implică postcondițiile (adică variabilele corespunzătoare datelor de ieșire conțin valorile corecte)

Starea unui algoritm

Exemplu: Rezolvarea ecuației a*x=b, a≠0

Date de intrare: a

Precondiții: a≠0

Date de ieșire: x

Postcondiții: x satisface a*x=b

Algoritm:

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

Apel:

```
  solve(a0,b0)
```

Stare algoritm

Valori curente pt. a și b

a=a0, b=b0, x nedefinit

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

a*x=b

Aserțiuni

- Aserțiune = afirmație (adevărată) privind starea algoritmului

- Aserțiunile sunt utilizate pentru a adnota algoritmii

- Adnotarea este utilă atât în
  - Verificarea corectitudinii algoritmilor

cât și ca
  - Instrument de documentare și depanare a programelor

- Obs: limbajele de programare permit specificarea unor aserțiuni și generarea unor excepții dacă aserțiunea nu este satisfăcută. In Python aserțiunile se specifică prin assert

Adnotare

Precondiții: a,b,c sunt numere reale distincte
Postcondiții: m=min(a,b,c)

```
min (real a,b,c)                //{a≠b, b ≠ c, c ≠ a}
real m
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
```

Adnotare

Precondiții: a,b,c sunt numere reale distincte
Postcondiții: m=min(a,b,c)

Altă variantă de determinare a minimului a trei valori

```
min (real a,b,c)                               //{a≠b, b ≠ c, c ≠ a}
   real m
   m ← a                                       // m=a
   IF m>b THEN m ← b ENDIF                     // m<=a, m<b
   IF m>c THEN m ← c  ENDIF                    // m<=a, m<b, m<c
RETURN m
```

↓

m=min(a,b,c)

Structura

- Analiza algoritmilor
- Noțiuni de baza
- Etapele verificării corectitudinii
- Reguli pentru verificarea corectitudinii

Etapele verificării corectitudinii

- Identificarea precondițiilor și a postcondițiilor

- Adnotarea algoritmului cu aserțiuni astfel încât

  - Precondițiile să fie satisfacute
  - Aserțiunea finală să implice postcondițiile

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

Câteva notații

Considerăm următoarele notații

P - precondiții
Q - postcondiții
A - algoritm

Tripletul (P,A,Q) reprezintă un algoritm corect dacă pentru datele de intrare ce satisfac precondițiile P, algoritmul:

- Conduce la postcondițiile Q
- Se oprește după un număr finit de prelucrări

Notație:

$$P \xrightarrow{A} Q$$

Reguli pentru verificarea corectitudinii

Pentru a demonstra că un algoritm este corect pot fi utile câteva reguli specifice principalelor tipuri de prelucrari:

- Prelucrări secvențiale
- Prelucrări de decizie (condiționale sau de ramificare)
- 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:

Dacă

$P \Rightarrow P_0$

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

$P_n \Rightarrow Q$

atunci

$P \xrightarrow{A} Q$

Cum interpretăm ?

Dacă

- Precondițiile implică aserțiunea inițială P_0
- Fiecare acțiune (A_i) implică aserțiunea următoare (P_i)
- Aserțiunea finală implică postcondițiile

atunci secvența de prelucrări este corectă

Regula prelucrării secvențiale

Exemplu: Fie x și y două variabile având valorile a și b. Să se interschimbe valorile celor două 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 și 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

Ce se poate spune despre următoarea variantă ?

$\{x=a, y=b\}$

$x \leftarrow y$

$\{x=b, y=b\}$

$y \leftarrow x$

$\{x=b, y=b\} \quad \not\Rightarrow Q$

Aceasta variantă nu satisface specificațiile problemei !

Regula prelucrării condiționale

Structura

A:

$\{P_0\}$

IF c

THEN

$\{c,P_0\}$

A_1

$\{P_1\}$

ELSE

$\{NOTc,P_0\}$

A_2

$\{P_2\}$

Regula:

Dacă

- c este bine definită
- c AND $P_0 \xrightarrow{A_1} P_1$
- $P_1 \Rightarrow Q$

și

- NOT c AND $P_0 \xrightarrow{A_2} P_2$
- $P_2 \Rightarrow Q$

atunci

$P \xrightarrow{A} Q$

Cum interpretam?

Condiția c (expresie logică) este considerată bine definită dacă poate fi evaluată

Ambele ramuri ale structurii conduc la postcondiții

Regula prelucrării conditionale

Exemplu: calcul minim a două valori

Precondiții: $a \neq b$

Postcondiții: $m=\min\{a,b\}$

```
    {a ≠ b}
IF a<b
     THEN
               {a<b, m nedefinită}
          m ← a
               {a<b, m=a}
     ELSE
               {b<a, m nedefinită}
          m ← b
              {b<a, m=b}
```

Dacă

$\{a<b, m=a\}$ implică $m=\min\{a,b\}$

și

$\{b<a, m=b\}$ implică $m=\min\{a,b\}$

Atunci algoritmul satisface specificațiile

Regula prelucrării repetitive

Verificarea corectitudinii structurii secvențiale și a celei condiționale este simplă …

verificarea corectitudinii prelucrărilor repetitive nu este la fel de simplă …

La nivel informal un ciclu este corect dacă are proprietățile:

- Dacă se termină conduce la satisfacerea postcondițiilor
- Se termină după un număr finit de pași

Dacă este satisfăcută doar prima proprietate ciclul este considerat parțial corect

Corectitudinea parțială poate fi demonstrată folosind inducția matematică sau așa numitele proprietăți invariante

Corectitudinea totală necesită și demonstrarea finitudinii

Pauză

Vă mai amintiți întrebarea de la sfârșitul cursului anterior ?

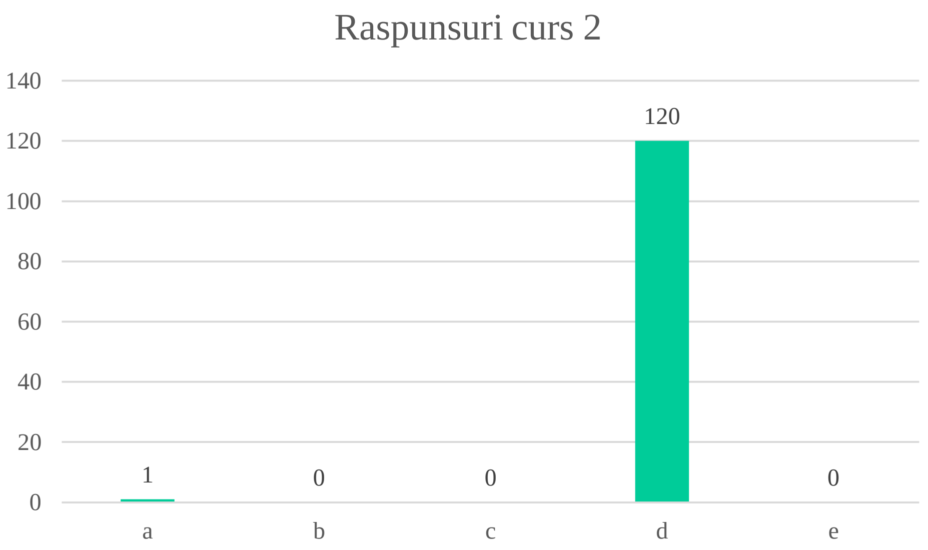
```
x = 4
y = 6
while y>0 do
  x = x+1
  y = y-1
endwhile
```

Ce valoare are x?

Variante de răspuns:

a) 5
b) 4
c) 6
d) 10
e) 2

Răspunsurile voastre:



Răspunsul corect: d

Pauză

Vă mai amintiți întrebarea de la sfârșitul cursului anterior ?

```
x ← 4
y ← 6
while y>0 do
   x ← x+1
   y ← y-1
endwhile
```

Ce valoare are x?

x	y	x+y
4	6	10
5	5	10
6	4	10
7	3	10
8	2	10
9	1	10
10	0	10

x+y=4+6=x0+y0 (x0,y0=valorile initiale)

La ieșirea din ciclu: y=0 => x=x0+y0

x+y=x0+y0 se numește proprietate invariantă

Proprietăți invariante

Considerăm următorul ciclu WHILE :

```
P =>{I}
WHILE c  DO
        {c,I}
      A
        {I}
ENDWHILE
{NOT c, I} => Q
```

Definiție:

O proprietate invariantă este o afirmație privind starea algoritmului (aserțiune) care satisface următoarele condiții:

1. Este adevarată la intrarea în ciclu
2. Rămâne adevarată prin execuția corpului ciclului
3. Când c devine falsă (la ieșirea din ciclu) proprietatea implică postcondițiile

Dacă poate fi identificată o proprietate invariantă pentru un ciclu atunci ciclul este considerat parțial corect

Proprietăți invariante

Precondiții:

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

Postcondiții:

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 când i=1 și m=x[1] proprietatea considerată invariantă este adevarată
- După execuția corpului ciclului proprietatea m=min{x[j]; j=1..i} rămâne adevarată
- La ieșirea din ciclu (când i=n) proprietatea invariantă devine m=min{x[j]; j=1..n} care este chiar postcondiția

Proprietăți invariante

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

Altă variantă 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-1}

De ce ? Pentru că …

- dacă i=2 și m=x[1] atunci invariantul este satisfăcut
- Cât timp i<=n execuția corpului ciclului asigură conservarea proprietății invariante
- La ieșirea din ciclu are loc i=n+1 ceea ce implică m=min{x[j]; j=1..n}, adică postcondiția

Proprietăți invariante

Problema: n un număr natural nenul. Să se calculeze suma cifrelor sale

Exemplu: pentru n=5482 se obține 2+8+4+5=19

P: $n>=1, n=c_k c_{k-1} \ldots c_1 c_0$

Q: $s=c_k+c_{k-1}+\ldots+c_1+c_0$

```
s ← 0
WHILE n != 0 DO
  d ← n MOD 10  //extragerea ultimei cifre
  s ← s+d            //adăugarea cifrei extrase
  n ← n DIV 10    // eliminarea cifrei din n
ENDWHILE
```

Analiza stării algoritmului:

Inainte de intrarea în ciclu (p=0):

$\{d=?, s = 0, n=c_k c_{k-1} \ldots c_1 c_0\}$

După prima execuție a corpului ciclului (p=1):

$\{d=c_0, s=c_0, n=c_k c_{k-1} \ldots c_1\}$

După a doua execuție a corpului ciclului (p=2):

$\{d=c_1, s=c_0+c_1 , n=c_k c_{k-1} \ldots c_2\}$

...

Care este proprietatea invariantă?

Proprietăți invariante

Problema: n un număr natural nenul. Să se calculeze suma cifrelor sale

P: $n>=1, n=c_kc_{k-1}...c_1c_0$

Q: $s=c_k+c_{k-1}+...+c_1+c_0$

```
s ← 0
WHILE n != 0 DO
  d ← n MOD 10
  s ← s+d
  n ← n DIV 10
ENDWHILE
```

Obs: p este o variabilă (implicită) care contorizează numărul de execuții ale ciclului (contorul ciclului)

Proprietate invariantă:

$\{s=c_0+c_1+...+ c_{p-1}, n=c_kc_{k-1}...c_p\}$

Analiza stării algoritmului:

(p=0):$\{d=?, s = 0, n=c_kc_{k-1}...c_1c_0\}$

(p=1):$\{d=c_0, s=c_0, n=c_kc_{k-1}...c_1\}$

(p=2):$\{d=c_1, s=c_0+c_1 , n=c_kc_{k-1}...c_2\}$

...

Proprietăți invariante

Problema: n un număr natural nenul. Să se calculeze suma cifrelor sale

P: $n>=1, n=c_kc_{k-1}...c_1c_0$

Q: $s=c_k+c_{k-1}+...+c_1+c_0$

```
s ← 0
WHILE n != 0 DO
  d ← n MOD 10
  s ← s+d
  n ← n DIV 10
ENDWHILE
```

Proprietate invariantă:

I: $\{s=c_0+c_1+...+ c_{p-1}, n=c_kc_{k-1}...c_p\}$

Verificare:

P→I: $p=0, s=0, n=c_kc_{k-1}...c_1c_0 \rightarrow$ I

I rămâne adevărat după execuția ciclului (etapa p+1):

I (etapa p)$\rightarrow \{s=c_0+c_1+\ldots+ c_{p-1}, n=c_kc_{k-1}...c_p\}$

$\rightarrow \{s=c_0+c_1+\ldots+ c_{p-1}+ c_p , n=c_kc_{k-1}...c_{p+1}\}$

$\rightarrow$ I (etapa p+1)

I→Q (la ieșirea din ciclu)

$n=0 \rightarrow p=k+1 \rightarrow s=c_0+c_1+ ...+c_{p-1} =c_0+c_1+ ...+c_k$

Analiza stării algoritmului:

(p=0):$\{d=?, s = 0, n=c_kc_{k-1}...c_1c_0\}$

(p=1):$\{d=c_0, s=c_0, n=c_kc_{k-1}...c_1\}$

(p=2):$\{d=c_1, s=c_0+c_1 , n=c_kc_{k-1}...c_2\}$

...

Proprietăți invariante

Problema: Fie x[1..n] un tablou care conține valoarea x0. Să se determine cea mai mică valoare a lui i pentru care x[i]=x0 (indicele primei pozitii pe care se află valoarea x0)

P: n>=1 și există1<= k <= n astfel încât x[k]=x0

Q: x[i]=x0 și x[j] ≠ x0 pentru j=1..i-1

```
i ← 1
WHILE x[i] != x0 DO
  i ← i+1
ENDWHILE
```

Proprietăți invariante

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

P: n>=1 și există1<= k <= n astfel încât x[k]=x0

Q: x[i]=x0 și 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
```

Proprietatea invariantă:

x[j] ≠ x0 for j=1..i-1

De ce ? Pentru că …

- dacă i=1 atunci domeniul de valori pentru j (j=1..0) este vid deci orice afirmatie referitoare la j din acest domeniu este adevarată
- Presupunem că x[i]≠x0 și invariantul e adevărat. Atunci x[j] ≠ x0 pt j=1..i
- După i ← i+1 se obține că x[j] ≠ x0 pt j=1..i-1
- La final, când x[i]=x0 rezultă postcondiția

Proprietăți invariante

Proprietățile invariante nu sunt utile doar pentru verificarea corectitudinii ci și pentru proiectarea ciclurilor

La modul ideal la proiectarea unui ciclu ar trebui ca:

– prima dată să se identifice proprietatea invariantă
– după aceea să se construiască ciclul

Exemplu: să se calculeze suma primelor n valori naturale

Precondiție: n>=1 Postcondiție: S=1+2+…+n

Ce proprietate ar trebui să satisfacă S dupa execuția pentru a i-a oară a corpului ciclului?

Invariant: S=1+2+…+i

Ideea pentru proiectarea ciclului:

– Prima dată se pregătește termenul de adăugat
– Apoi se adună termenul la sumă

Proprietăți invariante

Algoritm:
 	
i ← 1
S ← 1
WHILE i<n DO
    i ← i+1
    S ← S+i
ENDWHILE
	Algoritm:
	
	S ← 0
	i ← 1
	WHILE i<=n DO
	    S ← S+i
	    i ← i+1
	ENDWHILE

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 prelucrări repetitive de tip

```
WHILE c DO
    prelucrare
ENDWHILE
```

este suficient să se identifice o funcție de terminare

Definiție:

O funcție $F:N \rightarrow N$ (care depinde de contorul ciclului) este o funcție de terminare dacă satisface următoarele proprietăți:

i. F este strict descrescătoare

ii. Dacă condiția de continuare c este adevarată atunci $F(p)>0$

iii. Dacă $F(p)=0$ atunci c este falsă

Funcții de terminare

Observație:

- F depinde de contorul (implicit) al ciclului (notat în continuare cu p). La prima execuție a corpului ciclului p este 1, la a doua execuție a corpului ciclului este 2 s.a.m.d …)
- F fiind strict descrescătoare și luând valori naturale, va ajunge la 0 iar atunci când devine 0 condiția de continuare (condiția c) devine falsă, astfel că ciclul se termină.

Funcții de terminare

Exemplu: S=1+…+n

Prima variantă:

i_p reprezintă valoarea variabilei i la a p-a execuţie a corpului ciclului

$i \leftarrow 1$
$S \leftarrow 1$
WHILE i<n DO
    i:=i+1
        $\{i_p=i_{p-1}+1\}$
    $S \leftarrow S+i$
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 variantă:

$S \leftarrow 0$
$i \leftarrow 1$
WHILE i<=n DO
    $S \leftarrow S+i$;
    $i \leftarrow i+1$
        $\{i_p=i_{p-1}+1\}$
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 conține valoarea x0 pe cel puțin o poziție; să se determine cel mai mic indice k pentru care x[k]=x0.

Fie k prima apariție a lui x0 în x[1..n]

```
i ← 1
WHILE x[i] != x0 DO
      i ← i+1
        {i_p=i_(p-1)+1}
ENDWHILE
```

Funcţie de terminare:

$F(p)=k-i_p$

Verificare proprietăţi:

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

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

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

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, a0, b0 sunt numere naturale

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

Funcție 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

Funcție de terminare: F(p)=min$\{a_p,b_p\}$
$(b_0>a_1>b_1>a_2>b_2>\ldots$ => 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: $cmmdc(a,b)=cmmdc(a_0,b_0)$

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

Dacă $a_{p-1} > b_{p-1}$

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

altfel

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

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

Problema succesorului

Reminder Curs 2: determinarea succesorului în ordine crescătoare a unui număr constituit din 10 cifre distincte (în ipoteza că un astfel de succesor există)

```
Succesor(int x[1..n])
int  i, k
i ← Identifica(x[1..n])
if i==1
then write “nu exista succesor !"
else
   k ← Minim(x[i-1..n])
   x[i-1]↔x[k]
   x[i..n] ← Inversare(x[i..n])
   write x[1..n]
endif
```

Subalgoritmi:

Identifica (x[1..n])

P: n>1, exista i a.i. x[i-1]<x[i]

Q: x[i-1]<x[i] și x[j-1]>x[j], j=i+1..n

Minim (x[i-1..n])

P: x[i-1]<x[i] si x[j-1]>x[j], j=i+1..n

Q: x[k]>x[i-1], k în {i,...,n} și x[k]<=x[j], j în {i,...,n} cu x[j]>x[i-1] (cel mai mic element din x[i..n] care e mai mare decat x[i-1])

Inversare(x[i..n])

P: x[j]=x0[j], j=i..n

Q: x[j]=x0[n+i-j], j=i..n

Problema succesorului

Identifică cel mai din dreapta element, x[i], care este mai mare decât vecinul său din stânga (x[i-1])

```
Identifica(int x[1..n])
int i
i ← n
WHILE (i>1) and (x[i-1]>x[i])
DO
   i ← i-1
ENDWHILE
RETURN i
```

P: n>1, există i a.i.x[i-1]<x[i]

Q: x[i-1]<x[i] and x[j-1]>x[j], j=i+1..n

Invariant:

x[j-1]>x[j], j=i+1..n

Funcție de terminare:

$F(p)= (i_p-1)\, H(x[i_p]-x[i_p-1])$

$H(u)=0$ dacă $u>0$

$\quad\;\; 1$ dacă $u<0$

Problema succesorului

Determină indicele celei mai mici valori din subtabloul x[i..n] care este mai mare decât x[i-1]

```
Minim(int x[i..n])
int j
k ← i
j ← i+1
WHILE j<=n do
  IFx[j]<x[k] and x[j]>x[i-1]
     THEN  k ← j
  ENDIF
  j ← j+1
RETURN k
```

P: x[i-1]<x[i]

Q: x[k]<=x[j], j=i..n, x[j]>x[i-1], x[k]>x[i-1]

Invariant:

x[k]<=x[r], r=i..j-1 cu x[r]>x[i-1]

x[k]>x[i-1]

Funcție de terminare:

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

Problema succesorului

Inversează ordinea elementelor din subtabloul x[left..right]

```
inversare (int x[left..right])
   int i,j
   i ← left
   j ← right
   WHILE i<j DO
        x[i]↔x[j]
         i ← i+1
         j ← j-1
   ENDWHILE
   RETURN x[left..right]
```

P: x[k]=x0[k], k=left..right

Q: x[k]=x0[left+right-k], k=left..right

Invariant:

x[k]=x0[left+right-k], k=left..i-1

x[k]=x0[k], k=i..j

x[k]=x0[left+right-k], k=j+1..right

Funcție de terminare:

$F(p)=H(j_p-i_p)$

$$H(u)=\begin{cases} u & u>0 \\ 0 & u<=0 \end{cases}$$

Sumar

Verificarea corectitudinii algoritmilor presupune:

- Să se demonstreze că prin execuția instrucțiunilor se ajunge de la precondiții la postcondiții (corectitudine parțială)
- Să se demonstreze că algoritmul se termină după un număr finit de pași

Invariantul unui ciclu este o proprietate (referitoare la starea algoritmului) care satisface următoarele condiții:

- Este adevărată înainte de intrarea în ciclu
- Rămâne adevărată prin execuția corpului ciclului
- La sfârșitul ciclului implică postcondițiile

Următorul curs…

- Estimarea costului unui algoritm
  - Cel mai favorabil caz
  - Cel mai defavorabil caz
  - Cazul mediu

Întrebare de final

Care dintre următoarele proprietăți poate fi folosită ca invariant pentru a demonstra că algoritmul alg returnează valoarea factorialului:

a) f=1*2*…*(i-1)
b) i<n
c) f=1*2*…*i
d) f=1*2*…*n
e) f=f*i

```
alg (int n)
    int f,i
    i ←1
    f ←1
    while (i<n) do
        i ← i+1
        f ←f*i
    endwhile
    return f
```