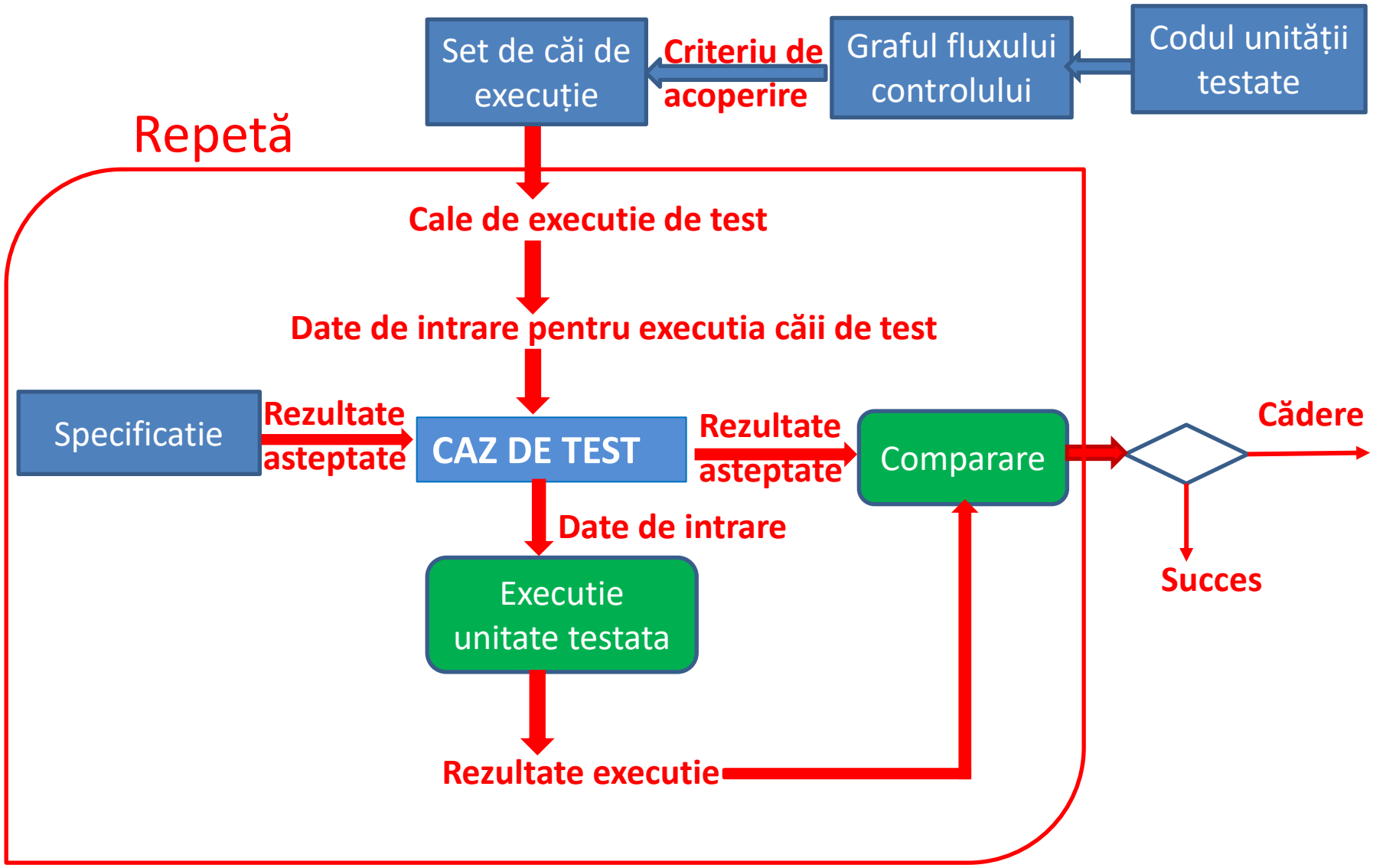
Alegerea cazurilor de test în testarea structurala

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Ingineria programelor* – UPB, Automatică şi Calculatoare
2020-2021

Testare structurala: testare white box



Testarea structurală
- white box testing -

- Focalizată pe implementarea unității testate – datele de test se aleg pe baza codului

Unitatea testată este tratata ca o “cutie transparenta”(white box).

- Se foloseşte în testarea unitară, în testarea de integrare şi în testarea programelor mici.
- Vectorii de test se aleg a.î. prin executiile de test sa fie traversate caile executabile ale unitatii testate: testarea structurala permite evidenţierea următoarelor tipuri de defecte:

1. Căi absente in graful fluxului controlului - ca urmare a ignorării unor condiţii (de exemplu: test de împărţire la zero);
2. Căi definite incorect sau incomplet (datorita unor conditii de ramificare gresite)
3. Acţiuni incorecte sau absente; de exemplu: calculul unei valori cu o metodă incorectă, neatribuirea unei valori unei anumite variabile, apelul unei proceduri cu o listă de argumente incorectă, etc.

Dintre acestea, cel mai simplu de depistat sunt greselile de tip 3. Pentru descoperirea lor este suficient să se asigure execuţia tuturor instrucţiunilor componentei testate.

Graful Fluxului Controlului

Graful fluxului controlului - GFC (Control Flow Graph), pe scurt **Graful controlului** unitatii testate este principalul instrument în testarea structurală.

- Se determina pe baza codului unitatii testate.
- Are trei **tipuri de noduri** :
  - **Noduri de prelucrare**, reprezentand "**blocuri de instrucţiuni indivizibile maximale"** : porţiuni liniare maximale de instrucţiuni care se execută întotdeauna în aceeaşi secvenţă
  - **Noduri de decizie**, care corespund instructiunilor de decizie (if, conditia de repetare sau iesire dintr-un ciclu).
  - **Noduri de jonctiune** a mai multor ramificatii (noduri ne-etichetate)
- **Arcele** reprezintă transferul controlului între noduri.
- **Graful controlului are un nod unic de intrare şi un nod unic de ieşire**. Nodul de intrare are gradul de intrare zero, iar cel de ieşire are gradul de ieşire zero.
- Daca în unitatea testata exista mai multe puncte de iesire, acestea se reunesc prin arce intr-un nod de jonctiune din care se ajunge direct in nodul de iesire.

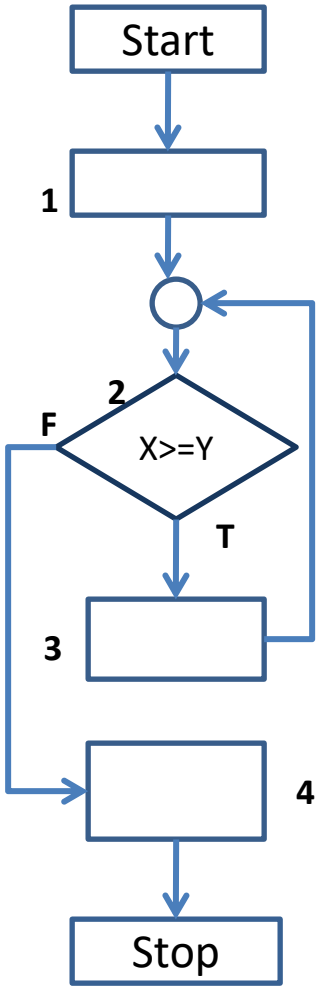
Graful fluxului controlului unei unitati program

EXEMPLU

```
f1=fopen(….);
f2=fopen(….);
fscanf(f1, ”%f”, x);
fscanf(f1, “%f”, y);
z=0;
while(x>=y)
{ x-=y;
  z++;
}
fprintf(f2, “%d”, z);
fclose(f1);
fclose(f2);
```

Codul este decupat în următoarele blocuri:

```
1. f1=fopen(….);
   f2=fopen(….);
   fscanf(f1, ”%f, x);
   fscanf(f1, “%f, y);
   z=0
2. while(x>=y)
3. x-=y;
   z++;
4. fprintf(f2, “%d”, z);
   fclose(f1);
   fclose(f2);
```



Graful controlului

Căi în graful fluxului controlului

- **O cale în graful controlului** este o secventa de noduri de prelucrare si de decizie, care începe cu nodul de intrare şi se termină cu nodul de ieşire.

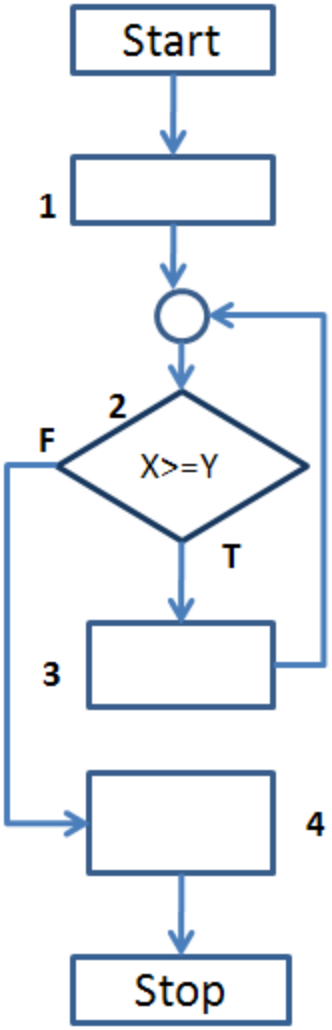
  Exemple:

  (1, 2(F), 4); (1, 2(T), 3, 2(F), 4); (1, 2(T), 3, 2(T), 3, 2(F), 4); etc.

- **Cale executabilă**: cale executată cel putin pentru un vector de intrare (Vector de intrare: asignare de valori variabilelor de intrare).

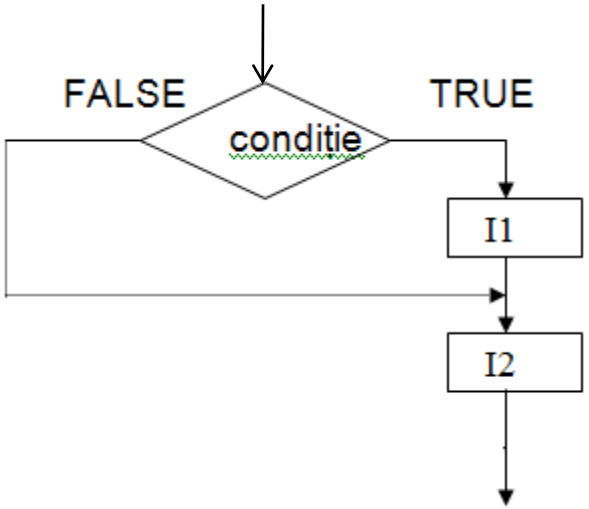
- **Scopul testarii structurale** : ideal→ parcurgerea tuturor cailor executabile prin executiile de test → in general, imposibil din cauza prezenţei ciclurilor.

- **Alegerea căilor de testat**: pe baza unor **criterii de acoperire a grafului controlului, care asigură descoperirea tipurilor de defecte mentionate.**



Acoperirea tuturor nodurilor grafului controlului

Acoperirea tuturor nodurilor= executia tuturor instructiunilor



➢Conform criteriului este suficient sa se testeze calea care contine nodul de decizie si cele 2 noduri de prelucrare.

➢ Criteriul nu impune testarea caii care contine ramificatia FALSE a conditiei.

→Programatorul nu va verifica rezultatul programului in acest caz.

Se recomanda alegerea mai multor cai de test, chiar daca toate nodurile grafului pot fi incluse intr-o singura cale:

- Cea mai scurta cale
- Alte câteva cai, de lungime crescândă

➢ **Este un criteriu de testare minimal. Nu este satisfăcător, în cele mai multe cazuri.**

Acoperirea tuturor ramificatiilor nodurilor de decizie ale grafului controlului (1)

- Criteriul impune ca prin executiile de test sa se traverseze toate ramificatiile nodurilor de decizie.

- Conform acestui criteriu se va putea verifica dacă transferul controlului este corect pentru valoarea FALSE a condiţiei, în exemplul anterior.

➢ Este posibil ca o singura cale executabila sa acopere toate arcele. Exemplu:

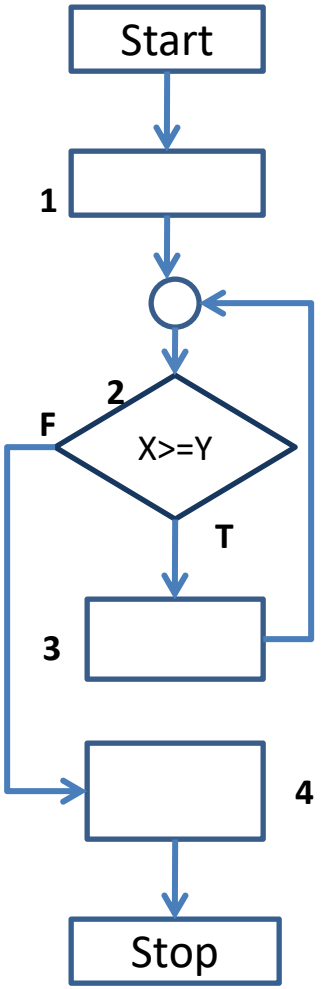
(1, 2(T), 3, 2(F), 4) → corespunde unei singure iteratii a ciclului

➢ In prezenta ciclurilor:

Cate cai se vor alege?

Cat de lungi?

Exemple: (1, 2(T), 3, 2(F), 4); (1, 2(T), 3, 2(T), 3, 2(F), 4), etc.



Acoperirea tuturor ramificatiilor nodurilor de decizie ale grafului controlului (2)

Testarea ciclurilor: recomandari empirice

- **Se alege o cale care nu traverseaza ciclul (zero iteratii)**. Testul poate evidentia greseli de initializare a ciclului, sau in calcule efectuate la iesirea din ciclu care se bazeaza pe executia ciclului.
- **Se alege o cale care contine o singura iteratie a ciclului.** Testul poate evidentia greseli de initializare a ciclului.
- **Se alege o cale care contine 2 iteratii ale ciclului**. Testul poate evidentia greseli care impiedica repetarea ciclului.

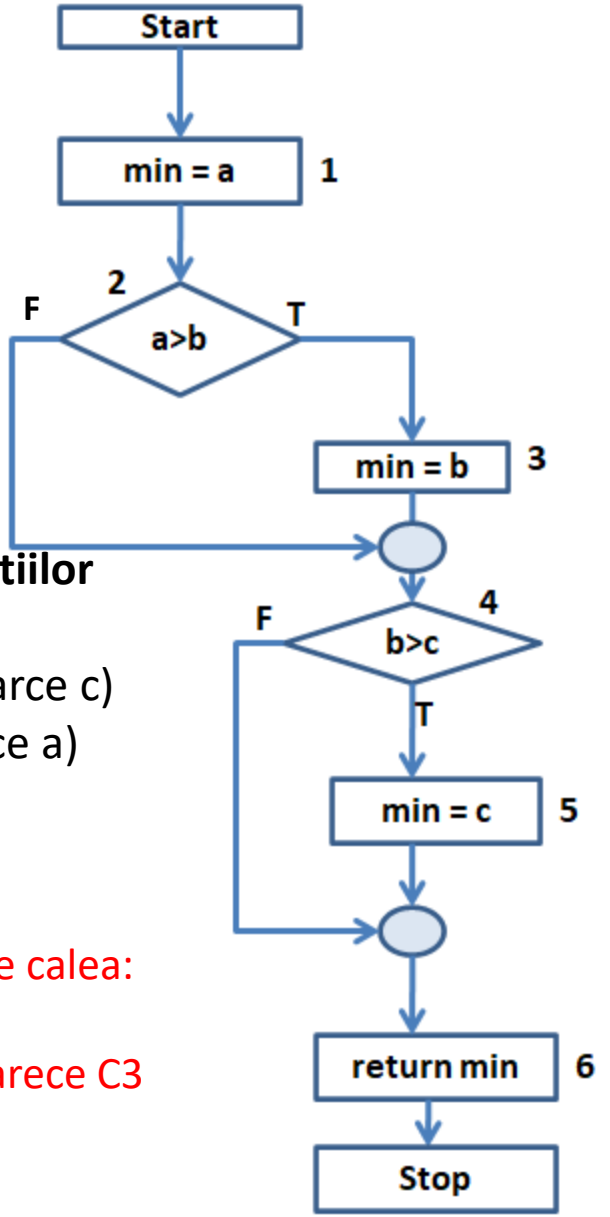
CONCLUZIE: în alegerea setului de cai de testat pe baza criteriului acoperirii tuturor ramificatiilor, se va tine cont de aceste recomandari.

Acoperirea tuturor ramificatiilor nodurilor de decizie ale grafului controlului (3)

Fie o functie care trebuie sa intoarca cea mai mica valoare dintre trei valori primite ca parametri.

Consideram urmatoarea implementare a functiei:

```
int minim (int a, int b, int c)
{ int min = a;
  if(a>b) min = b;
  if(b>c) min = c;
  return min;
}
```



Caile de test alese pe baza criteriului acoperirii tuturor ramificatiilor pot fi:

C1: (1, 2(T), 3, 4(T), 5, 6) → se testeaza cazurile: a>b si b>c (intoarce c)

C2: (1, 2(F), 4(F), 6) → se testeaza cazurile: a<=b si b<=c (intoarce a)

Cele 2 executii de test pot fi:

a = 50, b=40, c = 30 ; a=10, b=30, c = 40

Fie vectorul de intrare: a=15, b=30, c=20. Programul se va executa pe calea: C3: 1, 2(F), 4(T), 5, 6

Valoarea intoarsa este 20: greseala din cod nu va fi descoperita deoarece C3 nu a fost selectata pentru executiile de test.

Conditia din nodul 4 este gresita; trebuie sa fie: min>c

Acoperirea tuturor ramificatiilor nodurilor de decizie ale grafului controlului (4)

Exemple de cazuri de test pentru functia **minim**

Nr.	a	b	c	Relatia	Calea	Valoarea intoarsa	Valoare corecta
1.	1	1	1	a=b=c	1,2F,4F,6	1	1
2.	1	5	5	a<b si b=c	1,2F,4F,6	1	1
3.	5	5	3	a= b si b>c	1,2F,4T,5,6	3	3
4.	10	5	5	a>b si b= c	1,2T,3,4F,6	5	5
5.	10	5	20	a>b si b<c	1,2T,3,4F,6	5	5
6.	20	10	1	a>b si b>c	1,2T,3,4T,5,6	1	1
7.	**1**	**20**	**10**	**a<b si b>c si a<c**	**1,2F,4T,5,6**	**10**	**1**
8.	17	20	15	a<b si b>c si a>c	1,2F,4T,5,6	15	15

Cazurile de test 1, 2 si 6 corespund cailor C1 si C2 alese **pe baza criteriului acoperirii tuturor ramificatiilor.** Cf criteriului sunt suficiente 2 cazuri de test, care asigura executia pe caile C1, respectiv C2. Pentru orice caz de test care acopera una dintre cele 2 cai programul furnizeaza raspuns corect!

In cazurile (a<=b si b>c si a<c) programul furnizeaza raspuns gresit; exemplu: cazul de test 7.

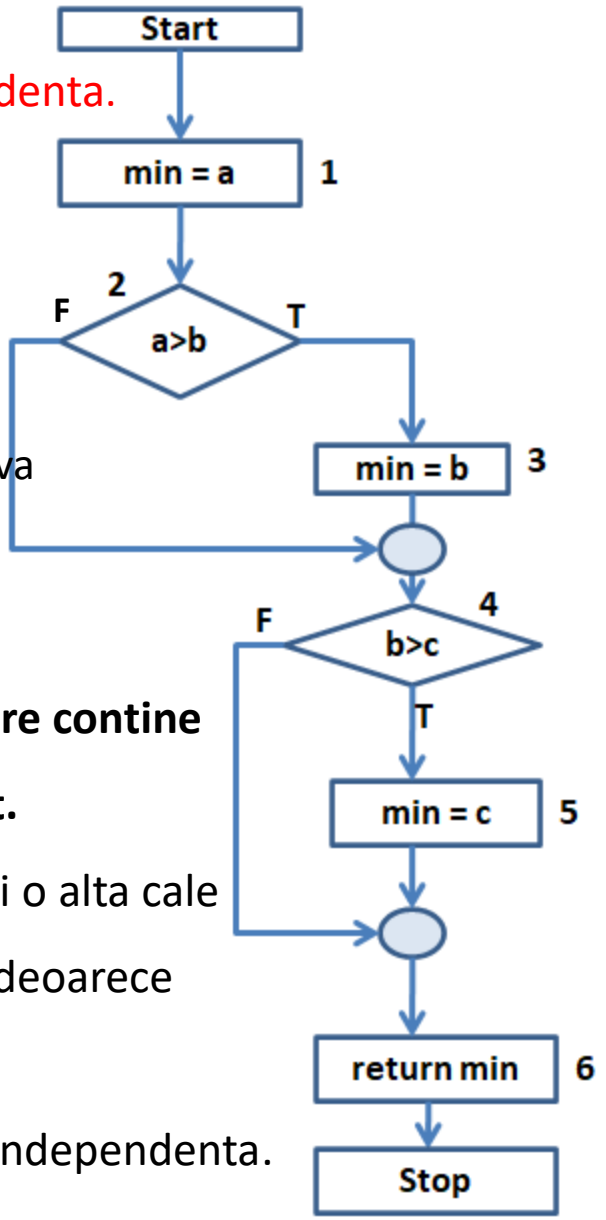
Acoperirea setului de căi de bază ale grafului controlului (1)

Criteriul acoperirii setului de cai de baza: fiecare ramificatie a unui nod de decizie trebuie testata pe o cale liniar independenta.

Pentru graful din figură se aleg căile:

1) TT - C1: (1, 2(T), 3, 4(T), 5, 6)
2) FT – C2: (1, 2(F), 4 (T), 5, 6) → acopera cazul a<b si b>c
3) FF – C3: (1, 2(F), 4(F), 6)

La executia caii C2 folosind cazul de test 7 din tabelul anterior se va descoperi ca programul produce un rezultat incorect!



➢ **Setul cailor de baza este un set de căi liniar independente**

- **Se construieste iterativ, adaugand de fiecare data o cale care contine cel putin un arc care nu este inclus in nici o alta cale din set.**
- Daca o cale introduce un nod nou (care nu este inclus in nici o alta cale liniar independenta) atunci calea este liniar independenta, deoarece introduce automat si un arc nou.
- O cale care este subcale a unei alte cai nu este o cale liniar independenta.

Acoperirea setului de căi de bază ale grafului controlului (2)

Complexitatea ciclomatica a unui program (McCabe, 1982):

v = e - n + 2

e este numarul de arce(edges) ale grafului controlului

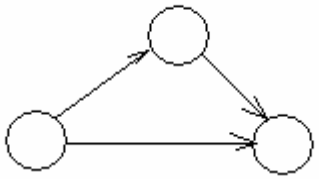
n este numarul de noduri(nodes)

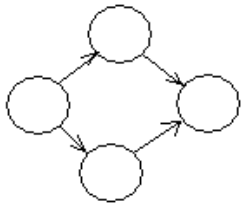
Complexitatea ciclomatică indică numarul de căi liniar independente prin codul sursă =

numarul de cazuri de test pe baza criteriului acoperirii setului de căi de baza.

Secventa  e=0, n=1, v=1

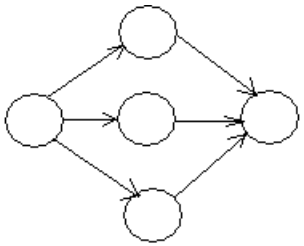
➢Un singur caz de test este suficient pentru testarea unei secvente

if ... then..  e=3, n=3, v=2

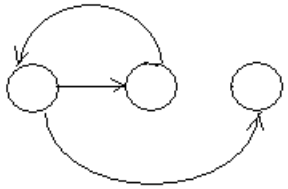
if..then..else..endif  e=4, n=4, v=2

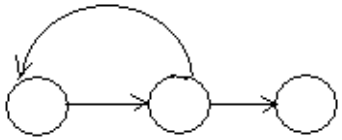
Fiecare nod de decizie creşte complexitatea ciclomatica cu 1: 1(secventa) +1

Acoperirea setului de căi de bază ale grafului controlului (3)

case  e=6, n=5, v=3

3 căi liniar independente: 3 cazuri de test

while..do  e=3, n=3, v=2

repeat ..until  e=3, n=3, v=2

Un nod de decizie introduce 2 căi liniar independente (2 cazuri de test)

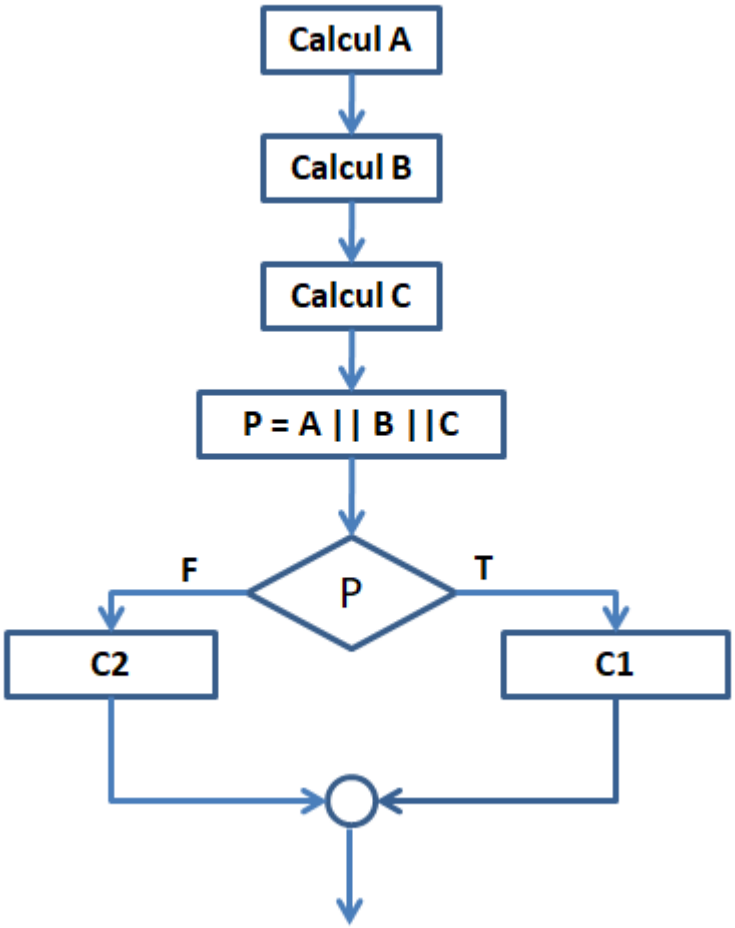
Pentru functia ***minim*** din exemplul anterior:

e = 7, n= 6, v = 7-6+2 = 3

- setul cailor de baza este compus din 3 cai liniar independente
- minim 3 executii de test

Acoperirea tuturor predicatelor

Criteriile bazate pe acoperirea tuturor ramificațiilor sunt insuficiente atunci cand nodurile de decizie conțin predicate compuse: de ex. (A && B && C), (A|| B || C). De aceea, în astfel de cazuri, setul de căi determinat pe baza acoperirii tuturor ramificațiilor/setului de căi de bază trebuie să fie completat cu teste pentru acoperirea predicatelor.



Caz	A	B	C	P
1	T	F	F	T
2	F	F	F	F

2 cazuri de test selectate pe baza criteriilor de acoperire a tuturor ramificatiilor. Nu sunt testate cazurile in care B=T sau C=T.

Caz	A	B	C	P
1	T	T	T	T
2	T	F	T	T
3	T	T	F	T
4	T	F	F	T
5	F	T	T	T
6	F	F	T	T
7	F	T	F	T
8	F	F	F	F

8 cazuri de test trebuie sa fie selectate pe baza criteriului acoperirii predicatelor:

pot fi descoperite erori in calculul predicatelor B si C.

Alegerea cazurilor de test (1)

- **După stabilirea setului căilor de test, C = {c}, trebuie alese cazurile de test: datele de intrare folosite în teste (vectorii de test) si rezultatele asteptate la fiecare test.**

1. Se determină predicatul fiecărei căi, $R_c(I)$, $c \in C$, unde

$I = (i_1, i_2, \ldots, i_n)$ este vectorul variabilelor de intrare;

2. Pentru fiecare $R_c(I)$, $c \in C$, se determină un set de atribuiri pentru variabilele de intrare, X_c care satisface predicatul căii:

$R_c(X_c) = true$ **X_c** → vectorul de intrare care asigură parcurgerea căii

Atunci setul vectorilor de intrare folosit în teste este:

$VT = \{ X_c \mid c \in C, X_c \in D_I, R_c(X_c) = true \}$,

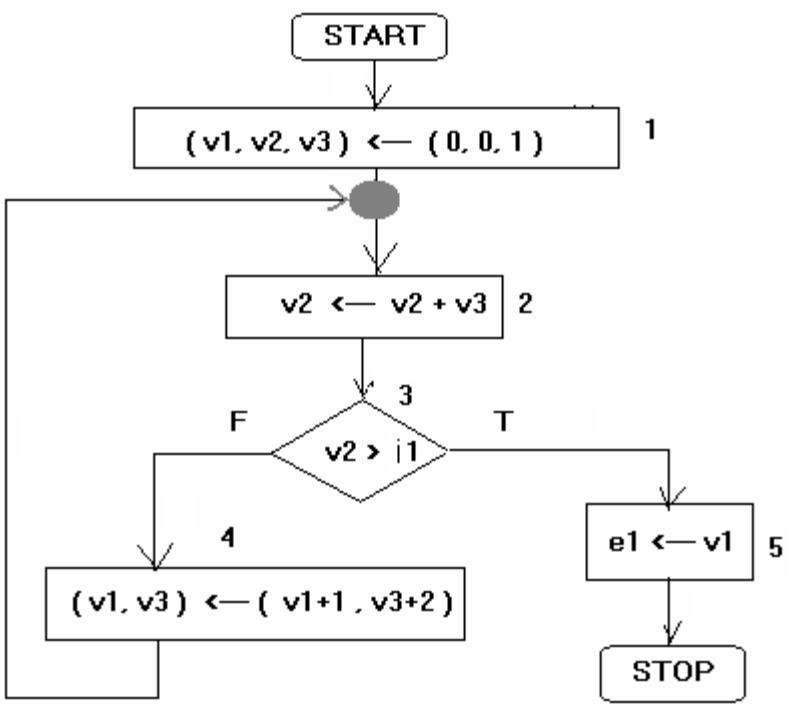
unde $D_I = X\ D_{ik}$ (produs cartezian), iar D_{ik} este domeniul de valori al variabilei de intrare i_k.

3. Pentru fiecare $X_c \in VT$, se determina valorile corecte ale variabilelor de iesire.

Alegerea cazurilor de test (2)

Fie urmatorul graf al controlului, pentru care s-au ales caile de test:

c1 = (1, 2, 3(T), 5)
c2 = (1, 2, 3(F), 4, 2, 3(T) , 5)



- Astfel, cele 2 ramificatii ale nodului de decizie sunt testate pe 2 căi liniar independente.

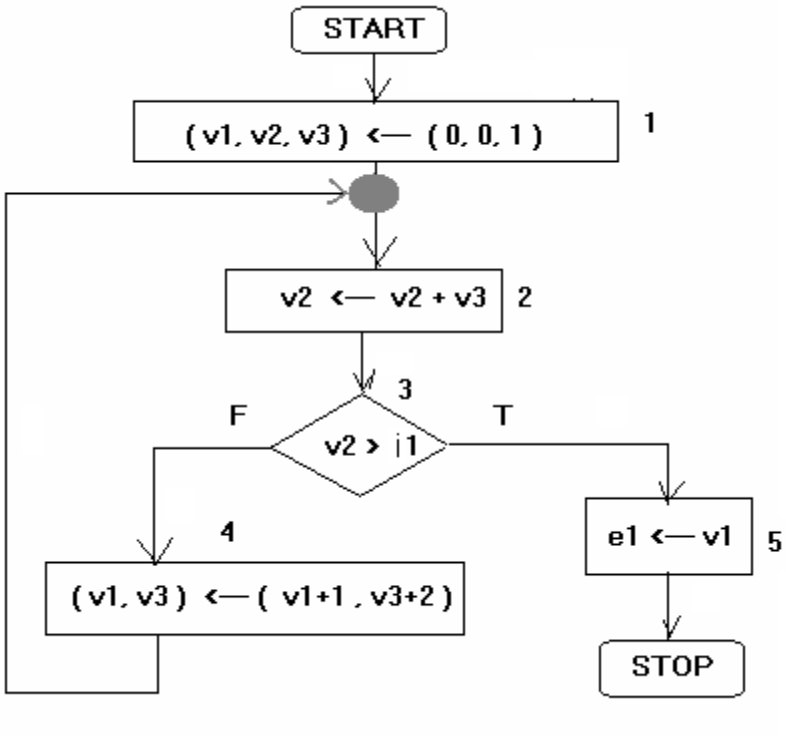
- **Pentru determinarea predicatului unei căi**, se parcurge calea de la nodul de iesire pana la nodul de intrare , concatenând conditiile de ramificare pe masura ce sunt intalnite.
- La intalnirea unei atribuiri, y←E, daca y face parte din expresia curenta a predicatului, se substituie y cu E.

Determinarea predicatului unei căi (1)

Exemplu:

c2 = (1, 2, 3(F), 4, 2, 3(T) , 5)

i1 este variabila de intrare, e1 este variabila de iesire



3T: Rc2 = v2> i1

2: Rc2 = (v2+v3)>i1

4: Rc2 = (v2 + v3 +2) > i1

3F: Rc2 = (v2 + v3 +2) > i1 Λ NOT(v2 > i1)

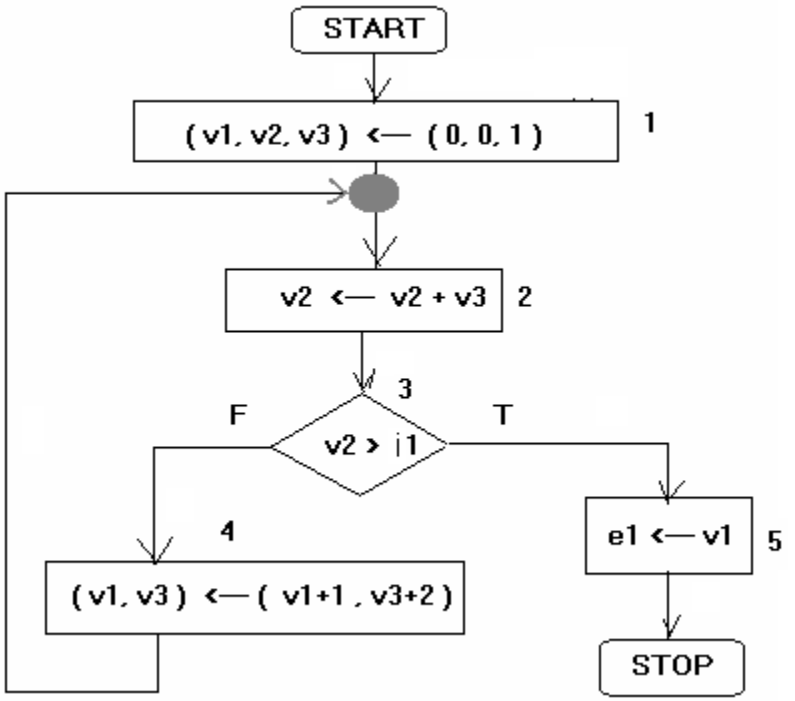
2: Rc2 = (v2 + v3 + v3 + 2) >i1 Λ NOT(v2+v3 >i1)

1: Rc2 = 4> i1 Λ NOT(1 >i1)

Rc2 = 1 <= i1 < 4

- Calea c2 va fi executata pentru orice valoare a lui i1 care satisface acest predicat: 1, 2, 3.

Determinarea predicatului unei căi (3)



c3 = (1, 2, 3(F), 4, 2, 3(F), 4, 2, 3(T), 5)

3T: Rc3 = v2> i1

2: Rc3 = (v2+v3)>i1

4: Rc3 = (v2 + v3 +2) > i1

3F: Rc3 = (v2 + v3 +2) > i1 Λ NOT(v2 > i1)

2: Rc3 = (v2 + v3 + v3 + 2) >i1 Λ NOT(v2+v3 >i1)

4: Rc3 = (v2 + v3 + 2+ v3+2 + 2) >i1

Λ NOT(v2+v3+ 2>i1) → Rc3 = (v2+2v3+4) > i1 Λ NOT(v2+v3+2 >i1)

3F: Rc3 = (v2+2v3+4) > i1 Λ NOT(v2+v3+2 >i1) Λ NOT(v2>i1)

2: Rc3 = (v2+v3 + 2v3 + 4) >i1 Λ NOT(v2+v3 + v3+2 >i1) Λ NOT(v2+v3>i1)

1: Rc3 = 7>i1 Λ NOT(4 >i1) Λ NOT(1>i1) → **Rc3 = 4 <= i1 < 7**

Stabilirea rezultatului unei executii de test

Stabilirea rezultatului corect al unei executii de test presupune cunoasterea specificatiei programului.

- **Programul din exemplu calculeaza numarul maxim de termeni ai sumei 1+3+5+... ..+ (2j+1), ∀ j >= 0, care pot fi adunati a.î. suma lor să fie <= i1.**

Rc2 = 1 <= i1 < 4

- Executand programul pentru orice 1 <= i1 < 4, rezultatul corect este e1 = 1.

Rc3 = 4 <= i1 < 7

- Executand programul pentru orice 4 <= i1 < 7, rezultatul corect este e1 = 2.

➢ **Criteriile de acoperire a grafului controlului pot conduce la alegerea unor căi netraversabile.**

Cale netraversabila = **cale neexecutabilă**: cale care nu va fi executata niciodata (nici o combinatie de valori ale variabilelor de intrare nu satisface predicatul căii)!

Avantajele/dezavantajele testării structurale

Avantaje

Pot fi generate automat:

- Graful controlului, pornind de la codul sursa
- Predicatele de cale, pe baza grafului controlului
- Vectorii de test, alesi aleator din domeniul datelor de intrare a.î. sa fie satisfacute predicatele de cale.

Este asistată de instrumente.

Dezavantaje

- Cazurile de test depind de codul programului, trebuie recalculate la orice modificare → nu pot fi reutilizate dupa o modificare a codului
- Cazurile de test acopera functionalitatile implementate, nu le descopera pe cele absente!

CONCLUZIE:

- Testele structurale trebuie sa fie completate cu cele functionale.
- Testele structurale pot evidentia unele erori care nu sunt descoperite prin testele functionale, care folosesc esantioane ale datelor de intrare → se începe cu testarea functionala si se continua cu testarea structurala.

Exercitii

1) Fie o functie implementata in C, care calculeaza $z= x^y$, pentru x, y, intregi pozitivi.

- Functia este declarata astfel:
  int putere(int x, int y)
- Sa se determine un set de cazuri de test pentru functie, prin:
  - Metoda testarii functionale
  - Metoda testarii structurale

Pentru testarea structurala, se va considera urmatorul cod:

```
int putere( int x, int y)
{  int z;
   z=1;
   while(y!= 0)
   { if(y mod %2 ==1) z = z *x;
     y = y/2; x = x*x;
   }
   return z
   }
```

Exercitii

2) Fie o functie implementata in C, care calculeaza numarul de termeni pozitivi dintr-un tablou de n numere intregi, X, si suma lor.

- Functia este declarata astfel:

  void f (int *X, int n, int &s, int &k)

  k – numarul de termeni pozitivi

  s – suma termenilor pozitivi

- Sa se determine un set de cazuri de test pentru functie, prin:
  - Metoda testarii functionale
  - Metoda testarii structurale

Lecturi suplimentare

1. https://sttp.site/chapters/testing-techniques/structural-testing.html
2. http://www.cs.toronto.edu/~chechik/courses18/csc410/Ch12-13-StructuralAndDataflowTesting.pdf
3. https://www.testbytes.net/blog/structural-testing-in-software-testing/