

CURS 03.

TESTARE WHITE-BOX

Verificarea și validarea sistemelor soft
[11 Martie 2024]

Lector dr. Camelia Chisăliță-Crețu
Universitatea Babeș-Bolyai

Conținut

- **Abordări ale testării**
- **Testare White-Box**
 - Definiție. Caracteristici
 - Tehnici de testare white-box
- **Testare bazată pe fluxul de control. Componente**
 - Definiție. Caracteristici. Avantaje și dezavantaje
 - • Graful fluxului de control. Exemple
 - Drumuri în CFG. Exemple
 - • Complexitatea ciclomatică. Exemple
- **Testare bazată pe acoperirea drumurilor**
 - Definiție. Algoritm. Exemplu
- **Testare bazată pe acoperirea codului sursă**
 - Definiție. Criterii de acoperire
 - Acoperirea instrucțiunilor, deciziilor, condițiilor, deciziilor și condițiilor, condițiilor multiple, buclelor
- **Testare White-box vs Testare Black-box**
 - Avantaje și dezavantaje ale Testării White-Box
 - Testare Black-box vs. Testare White-box
- **Bibliografie**

ABORDĂRI ALE TESTĂRII

Abordări ale testării. Clasificare

Tehnici de testare asociate

Abordări ale testării. Clasificare

- abordare a testării
 - modalitate de realizare a testării în care se aplică una sau mai multe tehnici de testare în cadrul unei strategii de testare stabilită anterior;
- clasificare
 - testare Black-box (**criteriul cutiei negre**, *engl. Black-box testing*);
 - testare White-box (**criteriul cutiei transparente**, *engl. White-box testing*);
 - testare Grey-box (**criteriul cutiei gri**, *engl. Grey-box testing*);
 - testare bazată pe experiență (*engl. Experienced-based testing*);
 - testare bazată pe scripturi (*engl. Scripted-based testing*);

Abordări ale testării. Tehnici de testare asociate

- Testare Black-Box – testare funcțională:
 - Partiționarea în clase de echivalență;
 - Analiza valorilor limită;
 - Tabele de decizie, Cazuri de utilizare, Scenarii de utilizare, etc.;
- Testare White-box – testare structurală:
 - Acoperirea fluxului de control (e.g., instrucțiuni, ramificații, decizii, condiții, bucle, drumuri);
 - Acoperirea fluxului de date;
- Testare Grey-box – testare mixtă:
 - folosirea simultană a avantajelor abordărilor black-box și white-box pentru proiectarea cazurilor de testare.

TESTARE WHITE-BOX

Definiție. Caracteristici

Tehnici de testare white-box

Quiz WBT

- Quiz WBT

- **CE?** Completați quiz-ul WBT cu întrebări referitoare la testarea white-box, disponibil la acest [link](#).
- **CUM?** Activitatea este facultativă. Fiecare întrebare din quiz are o singură variantă de răspuns corectă.
- **CINE?** Activitatea se adresează tuturor studenților înscriși la disciplina VVSS (IR, MIR, MIE).
- **CÂND?** Quiz-ul trebuie completat până cel târziu **Luni, 18 Martie 2024, orele 20:00**.
- **CÂT?** Sunt necesare max. 15 minute pentru completarea quiz-ului.
- **DE CE?** Studenții primesc **2 puncte** pentru activitatea **Quiz WBT**, inclusă în nota de la seminar. Punctele pentru această activitate se acordă indiferent de numărul de răspunsuri corecte indicate.

Testare White-Box. Definiție. Caracteristici

- **criteriul cutiei transparente** (*engl. white-box testing, logic driven testing*):
 - testare structurală;
 - datele de intrare se aleg pe baza instrucțiunilor care trebuie executate, programul este văzut ca o cutie transparentă;
 - avem acces la structura internă a programului (codul sursă);
 - permite identificarea situațiilor în care execuția programului nu acoperă diferite structuri ale acestuia.

Tehnici de testare white-box

- tehnici de proiectare a cazurilor de testare white-box bazate pe:
 1. **fluxul de control:**
 - **acoperirea drumurilor** [[NT2005](#)];
 - **acoperirea codului sursă:**
 - **instrucțiunilor, ramificațiilor, deciziilor, condițiilor, deciziilor și condițiilor, condițiilor multiple** [[Myers2004](#)], condițiilor/deciziilor modificate;
 - **buclelor** [[Beizer1990](#)];
 - **acoperirea predicatelor** (*engl.* **predicate complete coverage**);
 - **acoperirea prin mutații**;
 2. **fluxul de date** [[NT2005](#)].

TESTARE BAZATĂ PE FLUXUL DE CONTROL. COMPONENTE

Definiție. Caracteristici

Graful fluxului de control. Exemple

Drumuri în CFG. Exemple

Complexitatea ciclomatică. Exemple

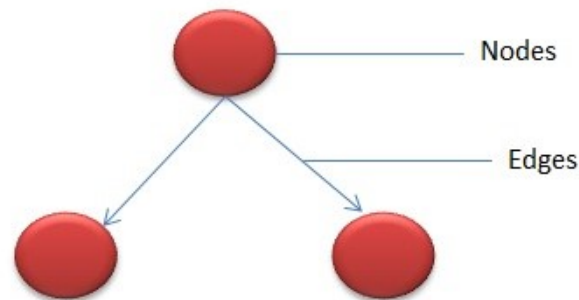
Testare bazată pe fluxul de control – Avantaje și dezavantaje

Testarea bazată pe fluxul de control

- **testarea bazată pe fluxul de control**
 - utilizează **structurile de control** pentru proiectarea cazurilor de testare;
 - **scop:** acoperirea prin cazuri de testare la un nivel satisfăcător a structurilor de control din programul testat;
- **componente:**
 - graful fluxului de control;
 - complexitatea ciclomatică.

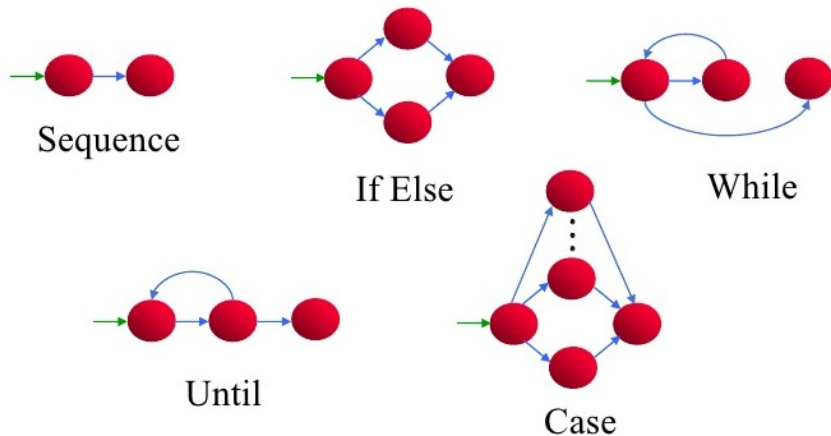
Graful fluxului de control. Definiție

- **graful fluxului de control** (*engl.* **Control Flow Graph, CFG**):
 - reprezentare grafică detaliată a unei unități de program;
 - permite vizualizarea tuturor drumurilor din unitatea de program;
- **graf orientat**:
 - **vârf** (*engl.* **node**):
 - indică structuri secvențiale și condițiile din structurile alternative sau repetitive;
 - **arc** (*engl.* **edge**):
 - indică sensul transmiterii controlului logic în cadrul programului.



CFG. Caracteristici

- permite reprezentarea grafică a structurilor de programare;
- tipuri de vârfuri:
 - **decizie:**
 - are o condiție prin care se permite ramificarea execuției prin cel puțin două căi;
 - e.g., instrucțiunile `if`, `while`, `repeat/until`, `case`;
 - **instrucțiune/calcul:**
 - conține o secvență de instrucțiuni;
 - **conector:**
 - nu conține o instrucțiune și reprezintă un punct al programului care unește mai multe ramificații;
 - **intrare, ieșire:**
 - există un singur vârf intrare și un singur vârf ieșire;
 - în vârful de intrare nu intră nici un arc;
 - din vârful de ieșire nu iese nici un arc.



CFG. Construire

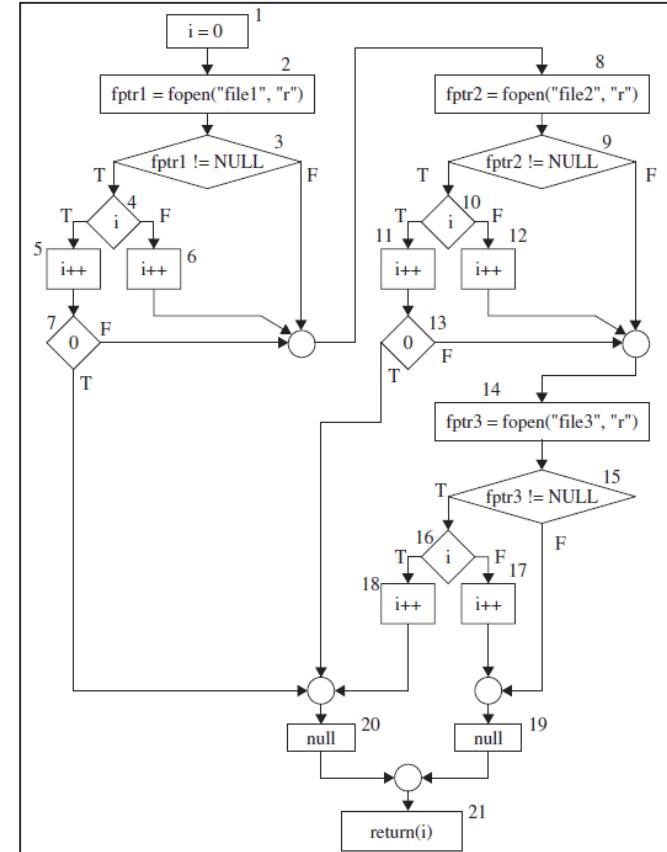
- pași de elaborare a unui CFG:
 1. se numerotează unic fiecare element de structură secvențială (calcul) și condițională (decizie);
 2. se începe pornind de la vârful de intrare, care are (de obicei) numărul 1;
 3. se adaugă celelalte vârfuri corespunzătoare structurilor numerotate și se unesc prin arce, evidențiind transmiterea controlului în cadrul programului;
 4. la final, toate ieșirile posibile din program se unesc în vârful de ieșire;
- condiții complexe care conțin atribuiri ==> CFG are o descriere complexă [[NT2005](#)];
 - e.g., `if (((fptrl = fopen(''file1'', ''r'')) != NULL) && (i++) && (0)){...}`.

CFG. Complexitatea construirii

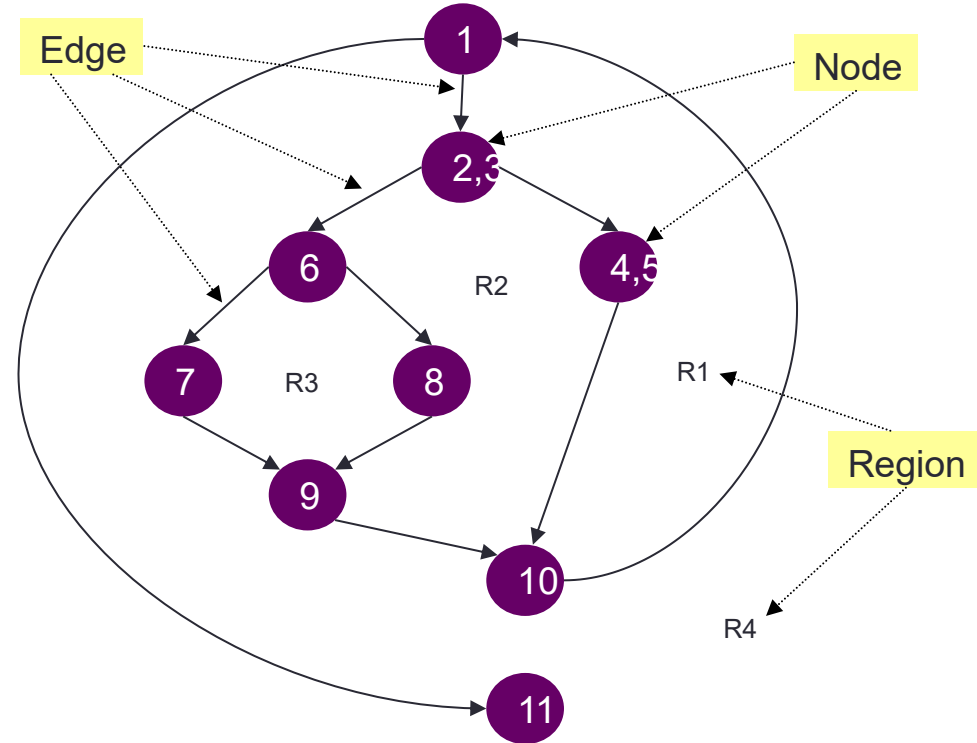
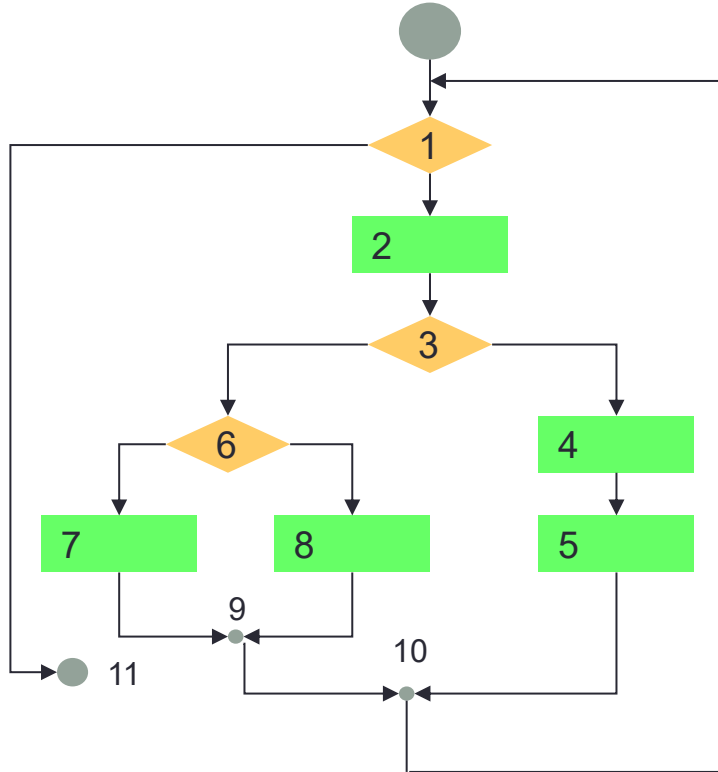
- pentru metoda `openfiles()`, avem CFG alăturat;

```
FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */

int openfiles(){
    /*
       This function tries to open files "file1", "file2", and
       "file3" for read access, and returns the number of files
       successfully opened. The file pointers of the opened files
       are put in the global variables.
    */
    int i = 0;
    if(
        ((( fptr1 = fopen("file1", "r")) != NULL) && (i++)
          && (0)) ||
        ((( fptr2 = fopen("file2", "r")) != NULL) && (i++)
          && (0)) ||
        ((( fptr3 = fopen("file3", "r")) != NULL) && (i++))
    );
    return(i);
}
```

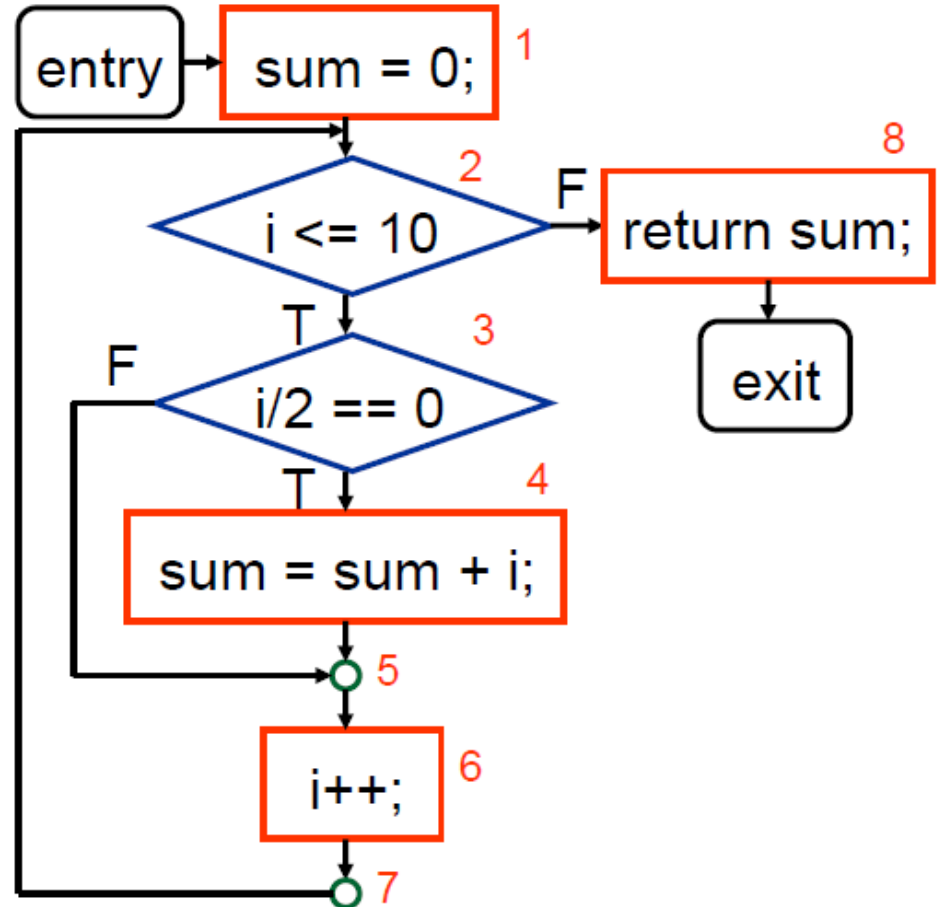


CFG. Exemple de notații

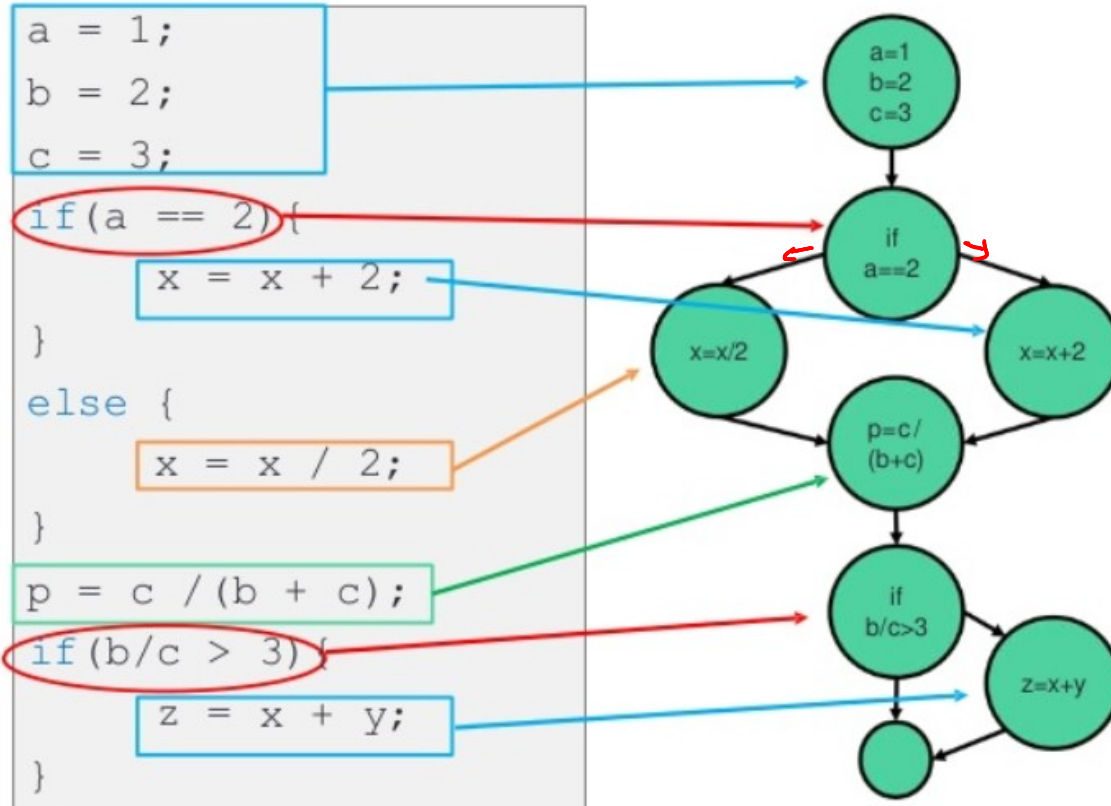


CFG. Example (1)

*	int evenSum(int i) {
1	int sum = 0;
2	while (i <= 10) {
3	if (i/2 == 0) {
4	sum = sum + i;
5	}
6	i++;
7	}
8	return sum;
*	}



CFG. Example (2)

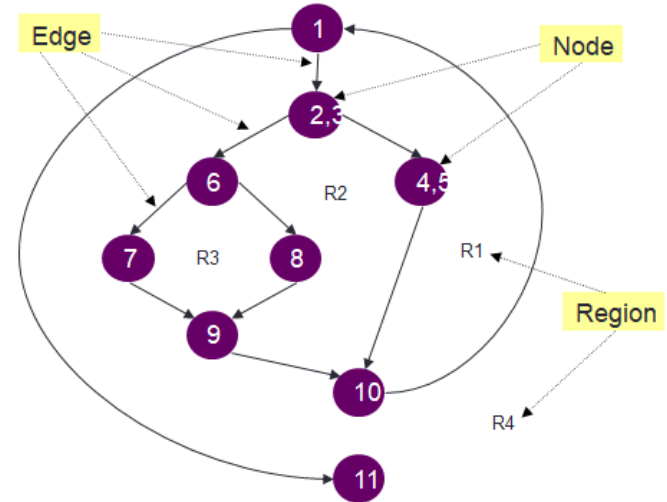
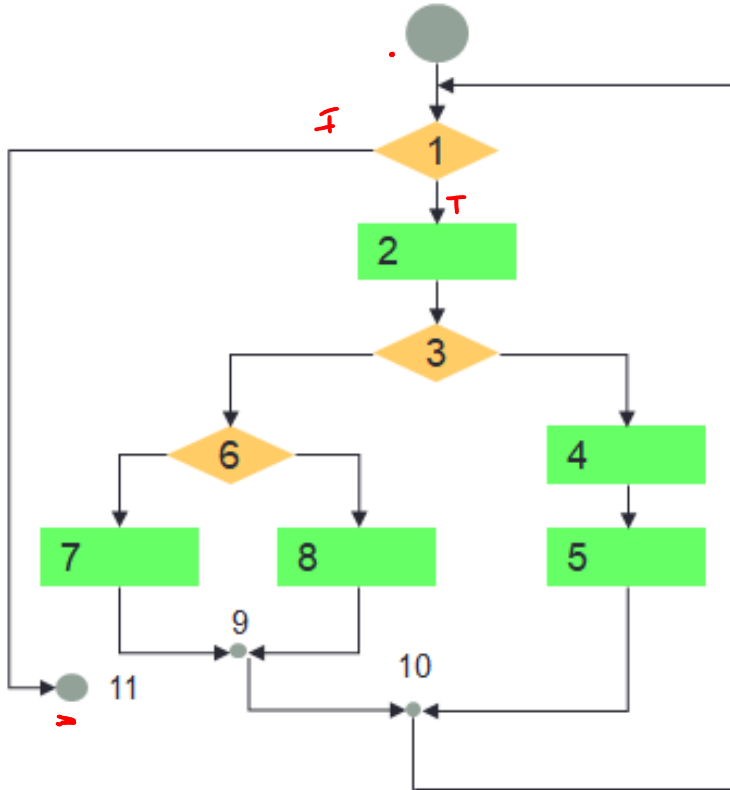


CFG. Drumuri în CFG

- **drum:**
 - execuția unei secvențe de instrucțiuni **de la punctul de intrare până la punctul de ieșire** al CFG asociat unei unități de program;
- **drum independent** (*engl. independent path*):
 - orice drum în CFG care introduce **cel puțin o instrucțiune nouă sau o condiție nouă**, care este executată cel puțin o dată;
- mulțimea drumurilor independente formează **mulțimea drumurilor de bază** (*engl. basis path set*) a unui CFG;
 - indică **numărul minim de cazuri de testare** care trebuie executate pentru ca fiecare instrucțiune să fie executată cel puțin o dată.

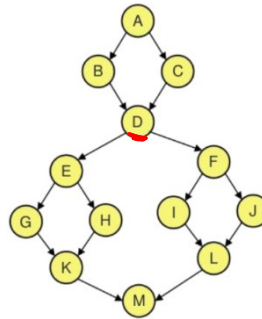
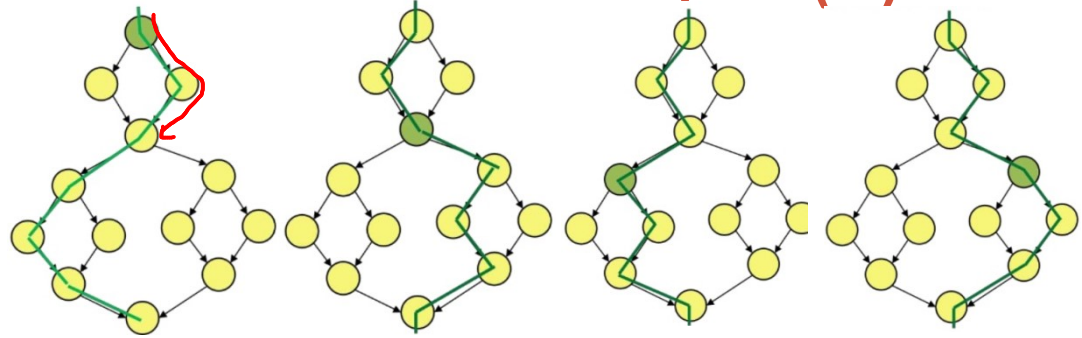
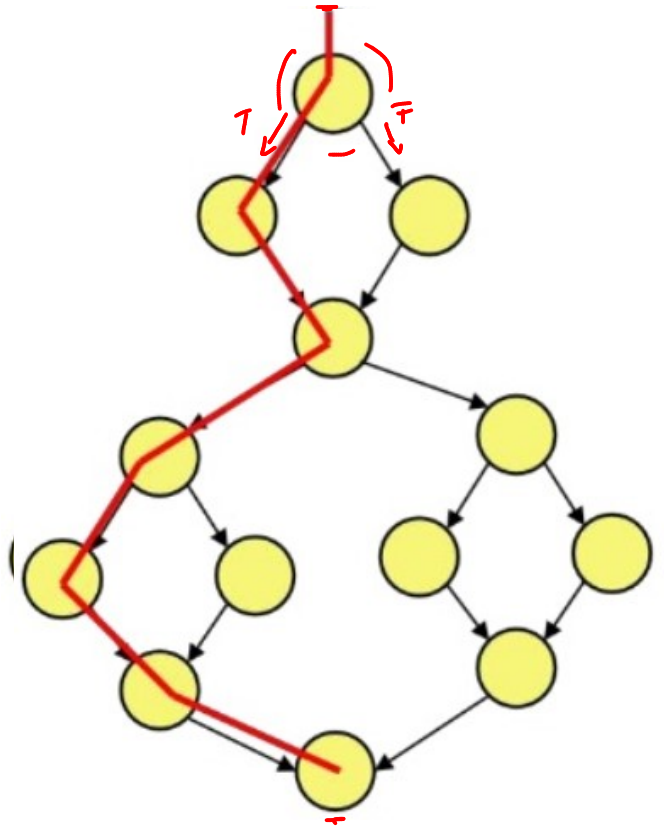
S . . . E

Drumuri independente în CFG. Exemple (1)



- drumuri independente:
 - **drum 1:** 1(F)-11.
 - **drum 2:** 1(T)-2-3(T)-4-5-10-1(F)-11.
 - **drum 3:** 1(T)-2-3(F)-6(T)-8-9-10-1(F)-11.
 - **drum 4:** 1(T)-2-3(F)-6(F)-7-9-10-1(F)-11.

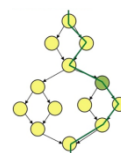
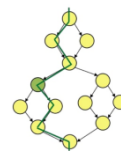
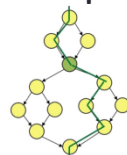
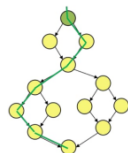
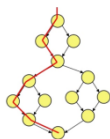
Drumuri independente în CFG. Exemple (2)



- drumuri independente:
 - **drum 1:** A-B-D-E-G-K-M;
 - **drum 2:** A-C-D-E-G-K-M;
 - **drum 3:** A-B-D-F-I-L-M;
 - **drum 4:** A-B-D-E-H-K-M;
 - **drum 5:** A-C-D-F-J-L-M.

CFG. Algoritm de construire a drumurilor independente

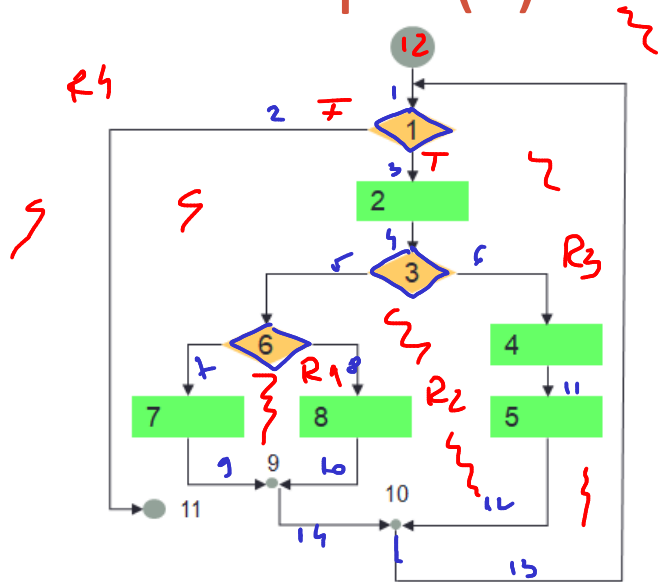
- **Algoritmul lui McCabe [McCabe1984, McCabe Baseline Method]**
 1. se alege un drum normal (numit **drum inițial, D1**); se recomandă ca acesta să aibă cât mai multe decizii este posibil;
 2. pentru generarea următorului drum (**D2**), se modifică rezultatul evaluării primei decizii de pe D1 și păstrând numărul de decizii ale drumului D1;
 3. pentru generarea următorului drum (**D3**), se modifică rezultatul evaluării celei de a doua decizii de pe D1;
 4. se repetă pasul 3 până când toate deciziile de pe D1 au fost modificate/inversate;
 5. se reiau pașii de la 1..4 considerând ca drum inițial pe D2, modificând/inversând deciziile, până când toate se obțin toate drumurile independente din mulțimea de bază a CFG.



Complexitatea ciclomatică. Definiție

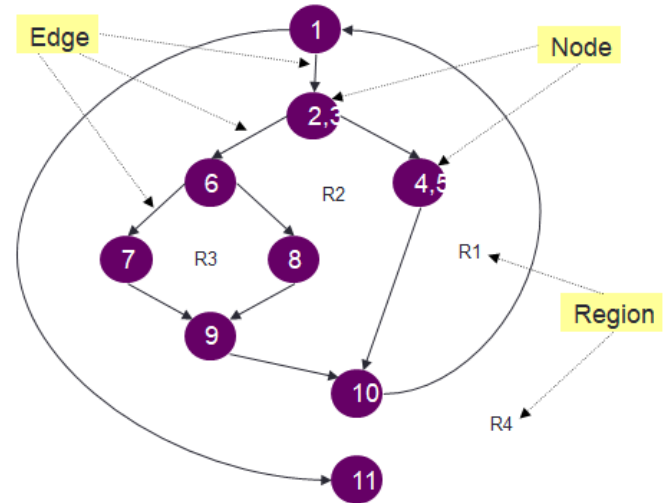
- **complexitatea ciclomatică** (*engl. McCabe's cyclomatic complexity, CC*):
 - metrică software aplicată pentru măsurarea cantitativă a complexității logice a unui program;
 - *permite determinarea numărului de drumuri independente din mulțimea de bază a unui CFG;*
- modalități de calcul a CC la nivelul CFG:
 - • **$CC = \text{numărul de regiuni din CFG}$** ;
 - • **$CC = E - N + 2$** , unde E - #arce, N - #vârfuri ;
 - • **$CC = P + 1$** , unde P - #vârfuri condiție.
- **regiune:**
 - zonă a CFG marginită parțial sau în totalitate de arce și vârfuri;

CC. Exemple (1)



• CC pentru CFG:

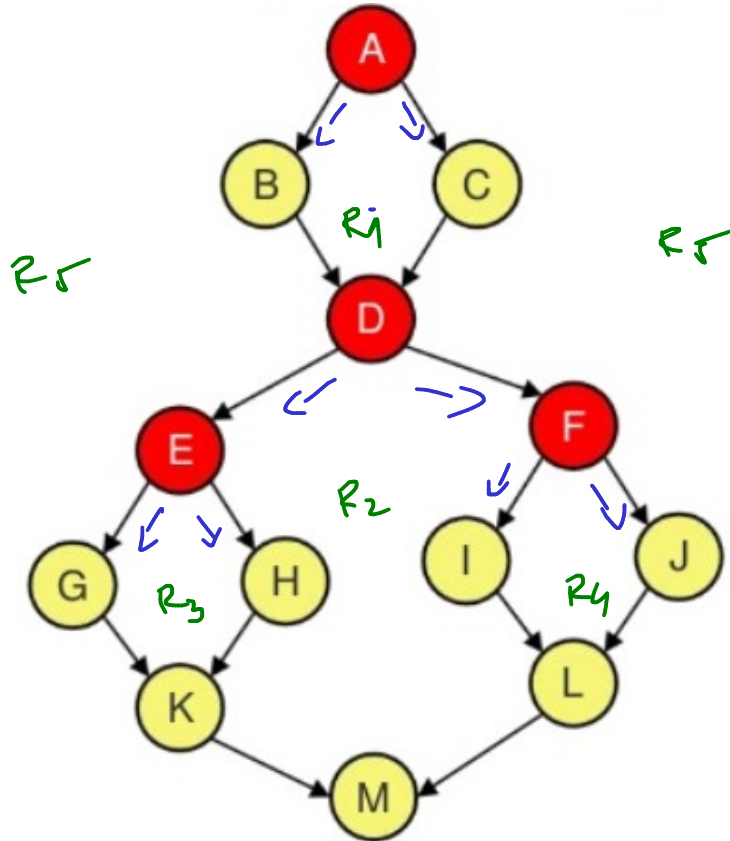
- CC = numărul de regiuni = 4 regiuni = 4. ✓
- CC = E - N + 2 = 14 arce - 12 vârfuri + 2 = 4. ✓
- CC = P + 1 = 3 vârfuri condiție + 1 = 4. ✓



• drumuri independente:

- **drum 1:** 1(F)-11.
- **drum 2:** 1(T)-2-3(T)-4-5-10-1(F)-11.
- **drum 3:** 1(T)-2-3(F)-6(T)-8-9-10-1(F)-11.
- **drum 4:** 1(T)-2-3(F)-6(F)-7-9-10-1(F)-11.

CC. Exemple (2)



- drumuri independente: (5)
 - drum 1: A-B-D-E-G-K-M;
 - drum 2: A-C-D-E-G-K-M;
 - drum 3: A-B-D-F-I-L-M;.
 - drum 4: A-B-D-E-H-K-M;
 - drum 5: A-C-D-F-J-L-M.
- CC pentru CFG: (5)
 - $CC = \text{numărul de regiuni} = 5 \text{ regiuni} = 5;$
 - $CC = E - N + 2 = 16 \text{ arce} - 13 \text{ vârfuri} + 2 = 5;$
 - $CC = P + 1 = 4 \text{ vârfuri condiție} + 1 = 5.$

Utilizarea CFG în testare. Avantaje și dezavantaje

Avantaje

- Testarea de bază aplicată în **testarea unitară**, care sunt dezvoltate la momentul curent;
- • Se aplică pentru modulele pentru care **prin inspectare nu pot fi suficient verificate**;
- Limbajele orientate-obiect reduc numărul de bug-uri la nivelul fluxului de control.

Dezavantaje

- Dacă este aplicată de tester și nu de programator, este necesar ca testerul să aibă abilități de programare pentru a înțelege codul sursă și modul de execuție al acestuia;
- • **Tehnică de testare consumatoare de timp**, deoarece mai întâi se elaborează CFG, CC, drumuri independente și ulterior se proiectează cazurile de testare;
- • Consideră că:
 - Specificațiile sunt corecte;
 - Datele sunt definite și accesate corespunzător;
- • Nu există alte bug-uri pe lângă cele determinate de fluxul de control.

TESTARE BAZATĂ PE ACOPERIREA DRUMURILOR

Definiție

Algoritm

Exemplu

Testare bazată pe acoperirea drumurilor. Definiție

- **acoperirea tuturor drumurilor** (*engl.* all path coverage, **apc**):
 - testarea tuturor drumurilor programului;
- **avantaje și dezavantaje:**
 - permite identificarea tuturor defectelor, dar **nu și de pe drumurile care lipsesc**;
- • **dificil** de realizat în practică pentru **programe cu structuri repetitive** ==> se alege un număr redus de drumuri;

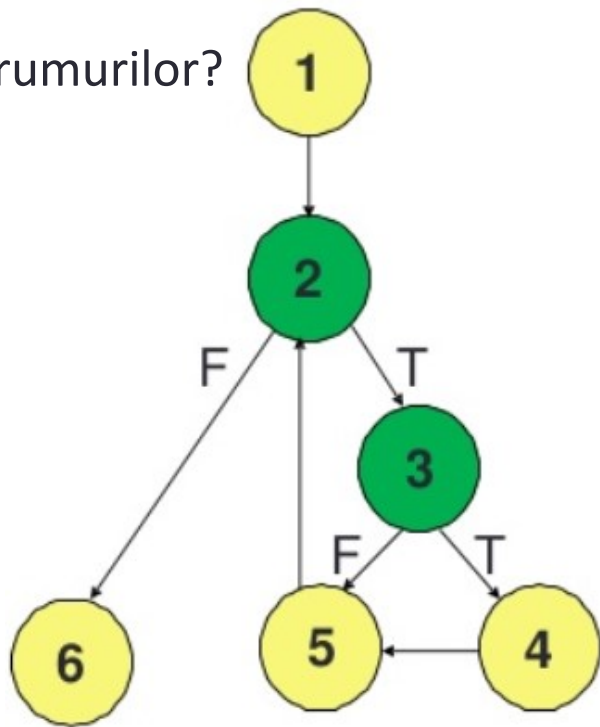
Testare bazată pe acoperirea drumurilor. Algoritm

- Algoritmul de proiectare a cazurilor de testare ^{apc} bazat pe drumuri este:
 1. Se elaborează CFG;
 2. Se calculează CC pe baza CFG;
 3. Se determină mulțimea de bază a CFG (cu drumuri independente, liniare);
 4. Se proiectează câte un caz de testare pentru fiecare drum independent identificat.
- ordinea de selectare a drumurilor:
 - drumuri scurte;
 - drumuri de lungime crescândă;
 - drumuri lungi, complexe, alese arbitrar.

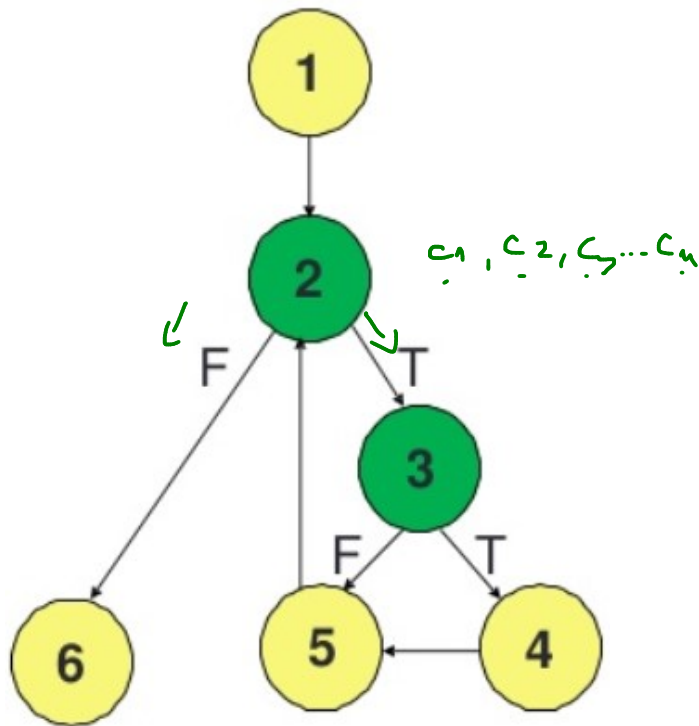
Testare bazată pe acoperirea drumurilor. Exemplu

Câte cazuri de testare sunt necesare pentru acoperirea drumurilor?

```
const int SIZE = 10;
int i;
int array[SIZE] = {52, 88, 90, 21, 62, 10, 16, 39, 45, 80};
int min = array[0];
while(i < 10) {
    if(array[i] < min) {
        min = array[i];
    }
    i = i + 1;
}
cout << min;
```



Testare bazată pe acoperirea drumurilor. Exemplu (cont)



- CC pentru CFG:
 - $CC = \text{numărul de regiuni} = 3 \text{ regiuni} = 3;$
 - $CC = E - N + 2 = 7 \text{ arce} - 6 \text{ vârfuri} + 2 = 3;$
 - $CC = P + 1 = 2 \text{ vârfuri condiție} + 1 = 3.$
- drumuri independente:
 - **drum 1:** 1-2(F)-6.
 - **drum 2:** 1-2(T)-3(F)-5-2(F)-6.
 - **drum 3:** 1-2(T)-3(T)-4-5-2(F)-6.

TESTARE BAZATĂ PE ACOPERIREA CODULUI SURSĂ

Definiție. Criterii de acoperire

Acoperirea instrucțiunilor. Definiție. Exemplu

Acoperirea deciziilor. Definiție. Exemplu

Acoperirea condițiilor. Definiție. Exemplu

Acoperirea deciziilor și condițiilor. Definiție. Exemplu

Acoperirea condițiilor multiple. Definiție. Exemplu

Acoperirea buclelor. Definiție. Exemplu

Testare bazată pe acoperirea codului sursă. Definiție

- **acoperirea codului sursă:**

- testarea tuturor structurilor de control folosind un număr minim de teste, astfel încât să fie satisfăcute criteriile:

0 → • **acoperirea instrucțiunilor** (*engl. statement/line/node coverage*); *sc*

1 → • **acoperirea ramificațiilor:**

→ • **acoperirea deciziilor** (arcelor) (*engl. decision/branch/edge coverage*); *dc*

→ • **acoperirea condițiilor** (*engl. condition coverage*); *cc*

→ • **acoperirea deciziilor și condițiilor** (*engl. decision-condition coverage*); *dcc*

→ • **acoperirea condițiilor multiple** (*engl. multiple condition coverage*); *mcc*

• acoperirea structurilor repetitive:

2 → • **acoperirea buclelor** (*engl. loop coverage*). *lc*



Acoperirea instrucțiunilor. Definiție

- **acoperirea instrucțiunilor** (*engl. statement/line/node coverage*, sc):
 - proiectarea cazurilor de testare astfel încât toate instrucțiunile sunt executate cel puțin o dată, adică fiecare vârf al CFG este vizitat;
 - **cel mai slab criteriu de acoperire în testare;**
 - o mulțime de teste care nu realizează acoperire 100% a vârfurilor nu este considerată acceptabilă.

SC. Exemplu

Fiecare instrucțiune trebuie să fie executată cel puțin o dată.

// returnează valoarea absolută a lui y

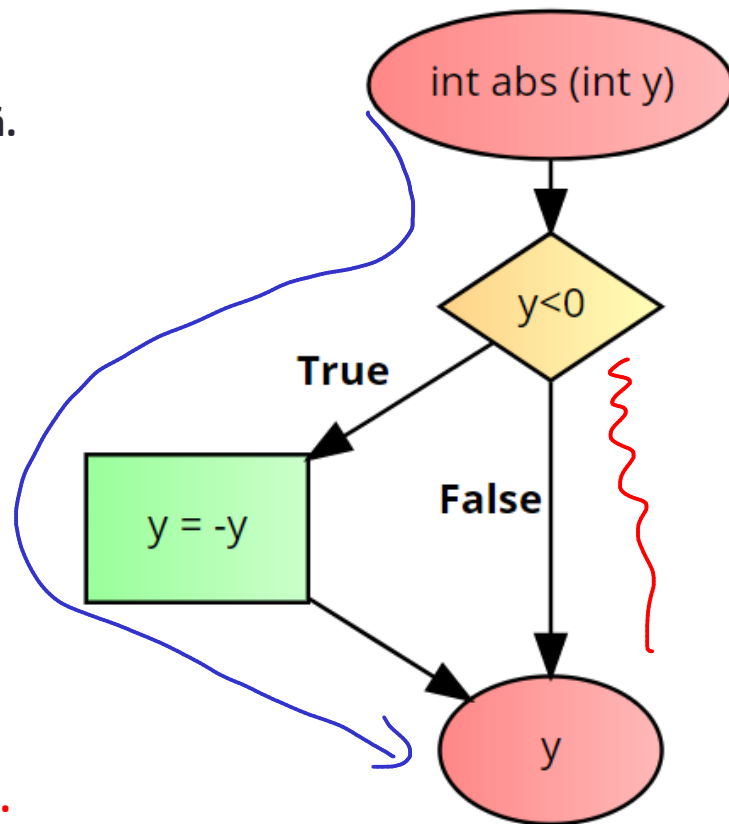
```
int abs (int y) {  
    . if (y<0)  
        . y = -y;  
    . return y;  
}
```

sc \Rightarrow TC1

Care este numărul minim de cazuri de testare necesar?

TC	Input	Expected result	Actual result
1	<u>-2</u>	2	2

Criteriul de acoperire a instrucțiunilor este îndeplinit 100%.



SC. Observații

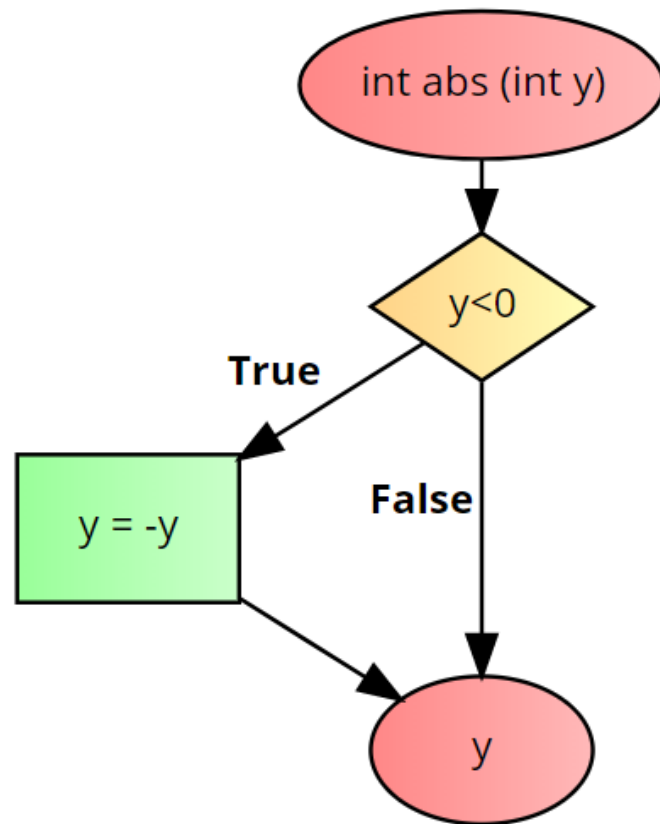
- cel mai slab criteriu de acoperire deoarece:
 - nu acoperă ramificația `else` pentru instrucțiunile `if` care nu descriu explicit această ramificație; nu evidențiază **implicit** prezența posibilelor bug-uri de pe aceste ramificații;
- SC se recomandă doar atunci când nu există alte criterii de acoperire care se pot aplica.

// returnează valoarea absolută a lui y

```
int abs (int y) {  
    if (y<0)  
        y = -y;  
    return y;  
}
```

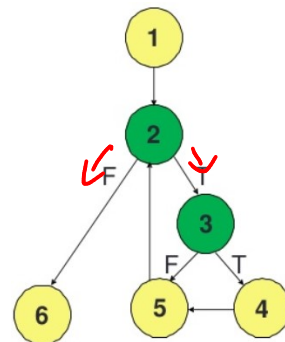
! rep. ! alt. } ⇒ SC ✓

Este necesară acoperirea deciziilor. (de)



Acoperirea deciziilor. Definiție

- **ramificație** (*engl.* **branch/edge**):
 - arc care pornește dintr-un vârf;
 - din fiecare vârf pornește cel mult un arc, mai puțin din vârful de ieșire al CFG;
 - din vârfurile de decizie pornesc două arce, etichetate cu true și false;
- **acoperirea deciziilor** (*engl.* **branch/edge/decision coverage**, **dc**):
 - acoperirea unui arc **a** = drum care parcurge arcul **a**;
 - proiectarea cazurilor de testare se face astfel încât *fiecare arc de decizie* să fie parcurs cel puțin o dată;
- regulă de selectare:
 - fiecare decizie selectată, evaluată la `true` sau `false`, trebuie să se găsească pe cel puțin un drum.



DC. Exemplu

Pentru fiecare decizie, fiecare ramificație (true, false) trebuie să fie executată cel puțin o dată.

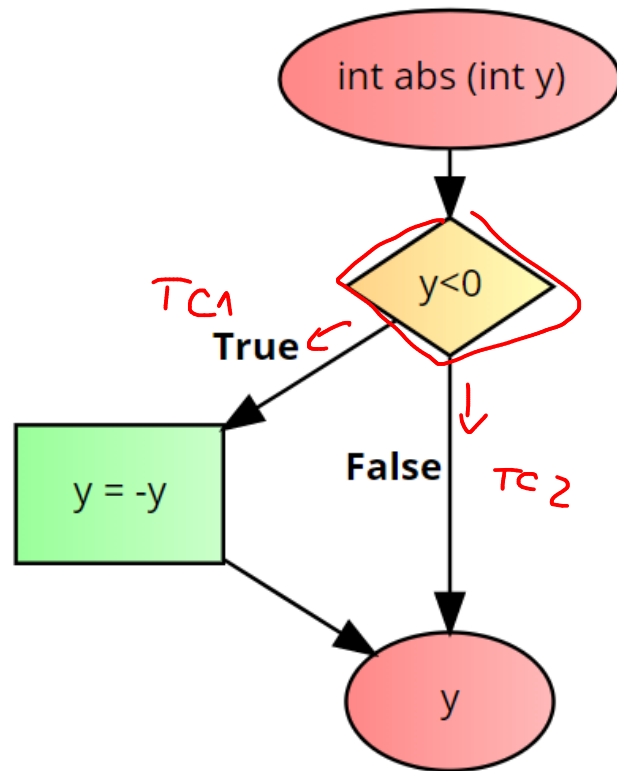
// returnează valoarea absolută a lui y

```
int abs (int y) {  
    if (y<0)  
        y = -y;  
    return y;  
}
```

Care este numărul minim de cazuri de testare necesar ?

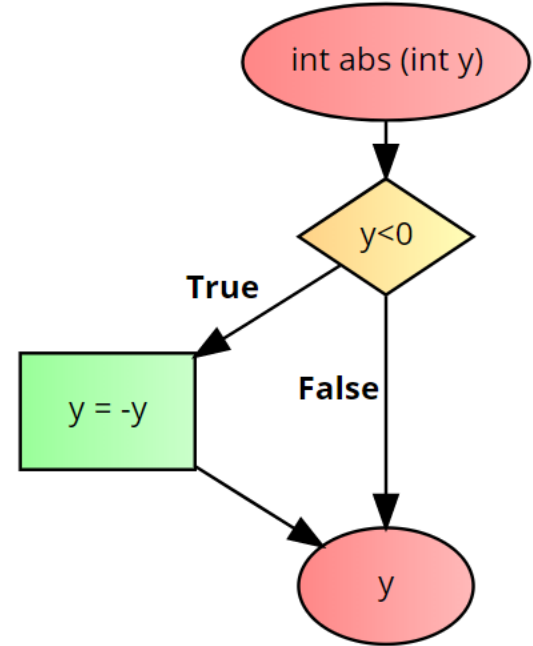
TC	Input	Decision (y<0)	Expected result	Actual result
1	-2	true	2	2
2	3	false	3	3

Criteriul de acoperire a deciziilor este îndeplinit 100%. Ambele ramificații ale deciziei au fost explorate.



DC vs. SC

- dc \implies sc;
 - instrucțiunile se află pe arce; dacă se parcurge fiecare arc atunci se execută și instrucțiunile asociate;

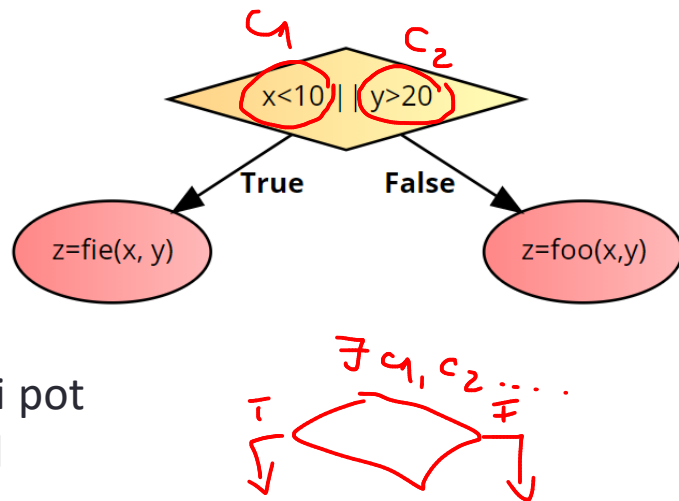


DC. Observații

- e.g.,

```
y = fou(x);  
if (x<10 || y>20)  
    {z=fie(x, y);}  
else {z=foo(x,y);}
```

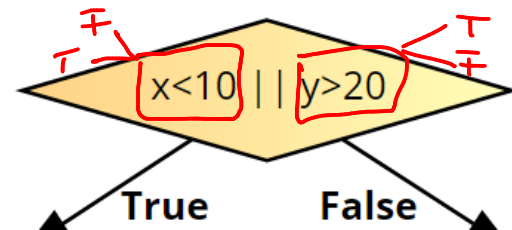
 - pentru x=1 și y=2 nu mai este relevantă evaluarea condiției y>20;
- în deciziile formate din mai multe condiții, unele condiții pot să rămână neacoperite fiind **irelevante** pentru rezultatul final al deciziei;
- dacă condiția este scrisă greșit, e.g., y<20 în loc de y>20, cazuri de testare ca x=1, y=2 nu evidențiază defectul.



Este necesară acoperirea condițiilor. (c_c)

Acoperirea condițiilor. Definiție

- **condiție:**
 - expresie logică dintr-un vârf de decizie;
 - o decizie este formată din una mai mai multe condiții;
- **acoperirea condițiilor** (*engl.* **condition coverage**, **cc**):
 - proiectarea cazurilor de testare se realizează astfel încât fiecare condiție din fiecare decizie ia fiecare dintre valorile posibile, cel puțin o dată;
- **regulă de selectare:**
 - pentru fiecare decizie care conține mai multe condiții, fiecare condiție selectată va fi evaluată la `true` sau `false` și se va găsi pe cel puțin un drum.



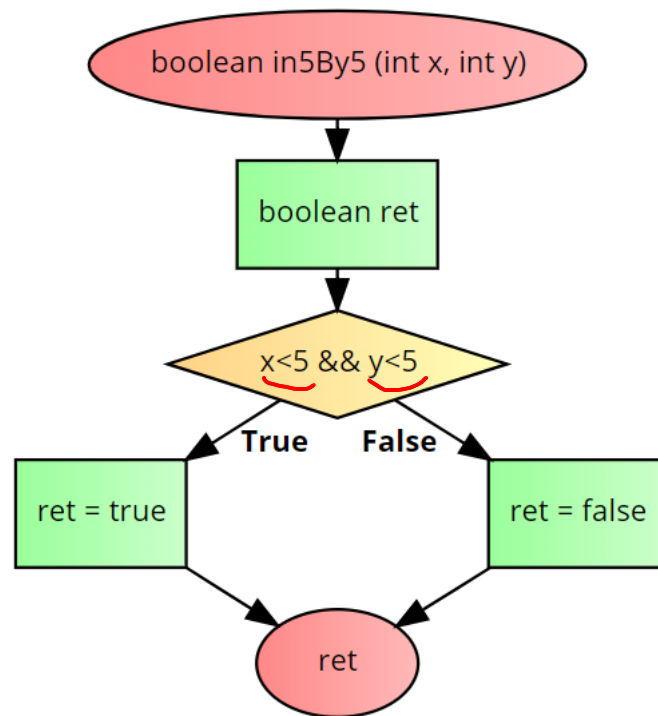
CC. Exemplu

Fiecare condiție din fiecare decizie trebuie să fie executată cel puțin o dată cu fiecare din valorile posibile (e.g., true, false)

// returnează true dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

Care este numărul minim de cazuri de testare necesar ?



CC. Exemplu (cont.)

// returnează true dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

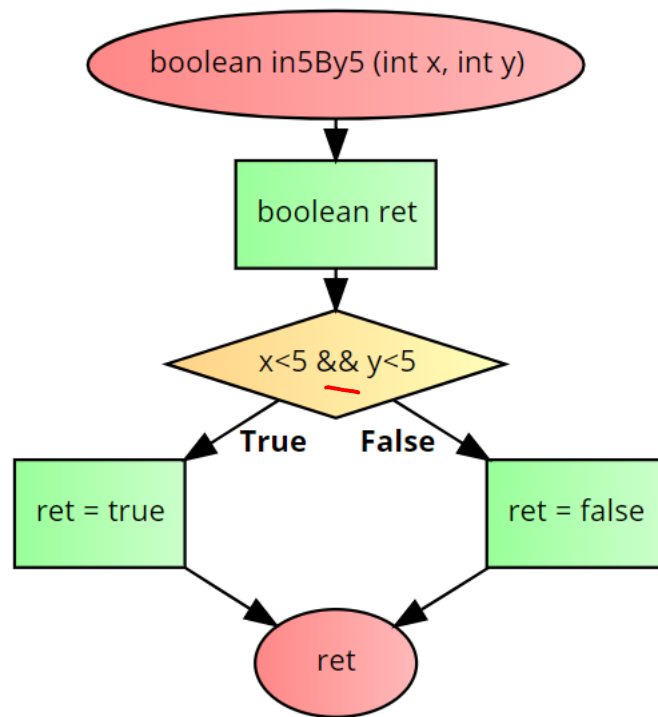
Care este numărul minim de cazuri de testare necesar ?

TC	x	y	Decision (x<5) && (y<5)	Expected result	Actual result
1	2	9	T && F = F	false	false
2	9	2	F && T = F	false	false

✓ ✓ T de

Criteriul de acoperire a condițiilor este îndeplinit 100%.

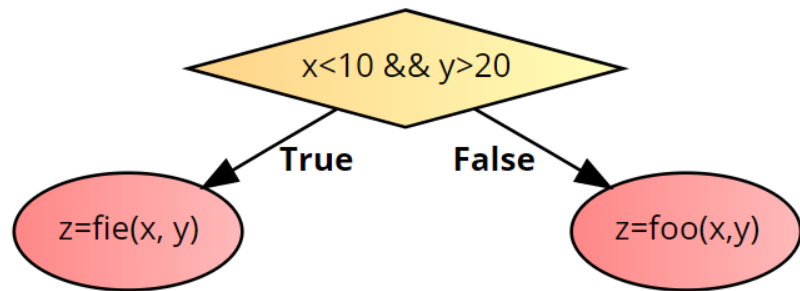
Toate rezultatele posibile ale evaluării condițiilor din decizie au fost explorate.



CC. Observații

- e.g.,

```
if (x<10 && y>20)
    {z=fie(x, y);}
else {z=foo(x,y);}
```
- pentru $x=11$ și $y=21$, avem $\text{false} \ \&\& \ \text{true} = \mathbf{false}$;
- pentru $x=1$ și $y=1$, avem $\text{true} \ \&\& \ \text{false} = \mathbf{false}$;

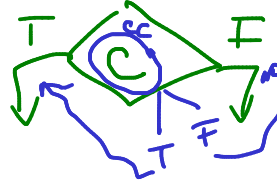


- fiecare *condiție* selectată este acoperită prin evaluarea la `true` și `false`, dar *decizia* nu este acoperită, doar ramificația `false` este explorată;

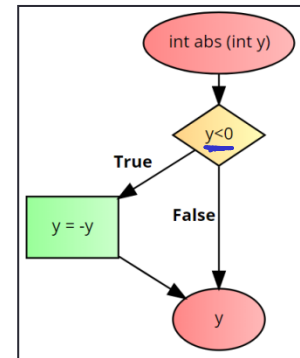
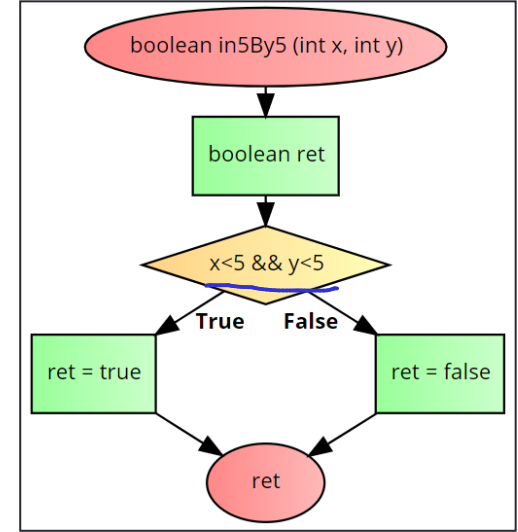
Este necesară acoperirea deciziilor și condițiilor.

(dec)

CC vs. DC



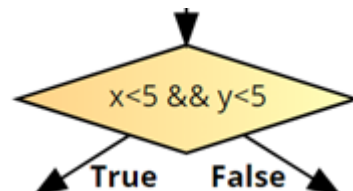
- în general, cc \implies dc;
 - prin acoperirea condițiilor se poate acoperi și decizia;
- caz particular:
 - cc = dc atunci când decizia conține doar o condiție;
 - e.g., decizia $y < 0$ este evaluată la true sau false, similar cu evaluarea condiției $y < 0$, care este evaluată la true sau false \implies acoperirea condiției este similară cu acoperirea deciziei;



Acoperirea deciziilor și condițiilor. Definiție

- **acoperirea deciziilor și condițiilor** (*engl.* decision and condition coverage, **dcc**):

- proiectarea cazurilor de testare astfel încât:
 - fiecare condiție din fiecare decizie ia toate valorile posibile, cel puțin o dată;
 - fiecare decizie ia toate valorile posibile cel puțin o dată;



- **regulă de selectare:**

- pentru fiecare decizie care conține mai multe condiții, fiecare condiție selectată va fi evaluată la `true` sau `false` și împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puțin un drum.

DCC. Exemplu

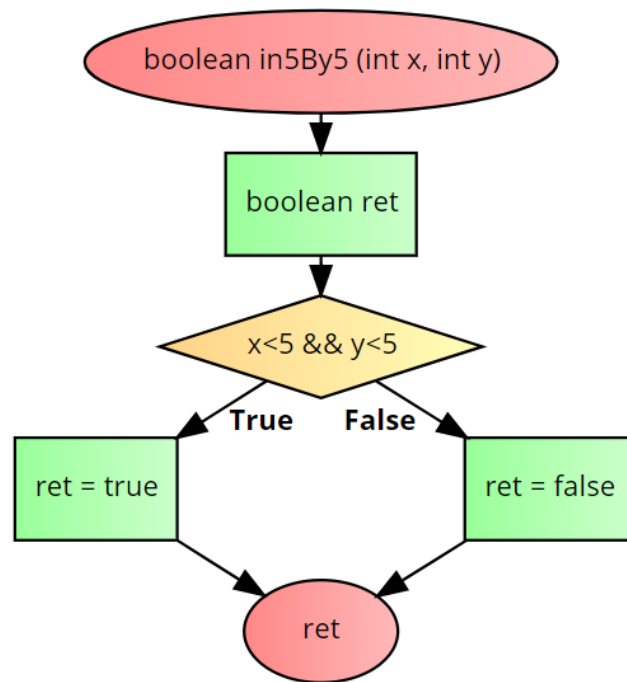
Fiecare *condiție* din fiecare decizie trebuie să fie executată cel puțin o dată cu fiecare din valorile posibile, e.g., `true`, `false`.

Fiecare *decizie* trebuie să fie executată cel puțin o dată cu fiecare din valorile posibile, e.g., `true`, `false`.

// returnează `true` dacă (x,y) este în cadranul $(5,5)$.

```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

Care este numărul minim de cazuri de testare necesar ?



DCC. Exemplu (cont.)

// returnează true dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

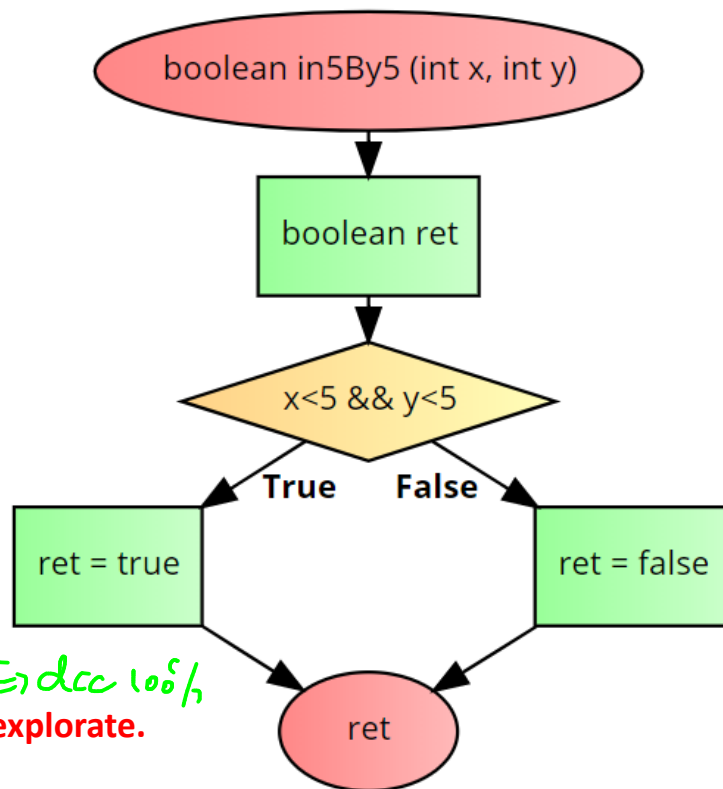
Care este numărul minim de cazuri de testare necesar ?

TC	x	y	Decision (x<5) && (y<5)	Expected result	Actual result
1	2	3	T && T = T	true	true
2	9	7	F && F = F	false	false

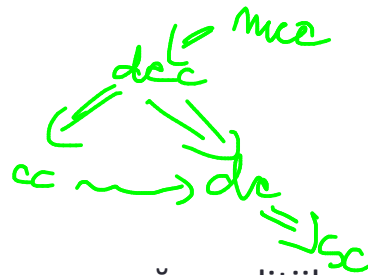
Criteriul de acoperire a deciziilor și condițiilor este îndeplinit 100%. *⇒ dcc 100%*

Toate rezultatele posibile ale evaluării condițiilor din decizie au fost explorate.

Ambele ramificații ale deciziei au fost explorate.

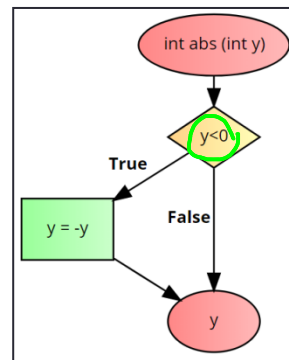
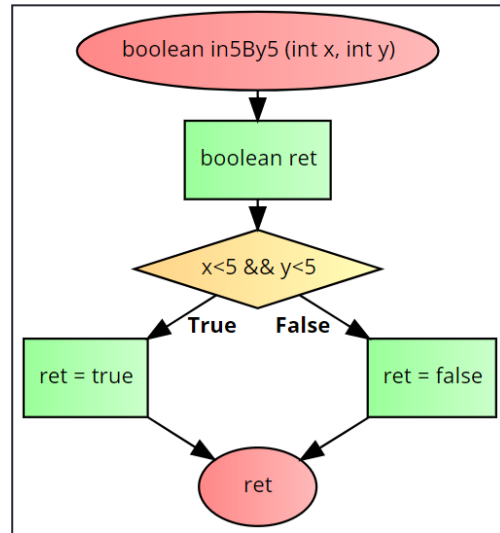


DCC vs. CC, DCC vs. DC



- dcc ==> cc;
 - prin acoperirea deciziilor și condițiilor se acoperă condițiile;
- dcc ==> dc;
 - prin acoperirea deciziilor și condițiilor se acoperă deciziile;
- caz particular:
 - dcc = dc și dcc = cc atunci când decizia conține doar o condiție;
 - e.g., decizia `y < 0` este evaluată la `true` sau `false`, similar cu evaluarea condiției `y < 0`, care este evaluată la `true` sau `false` ==> acoperirea deciziei și condiției (dcc) este similară cu acoperirea deciziei (dc), care este similară cu acoperirea condiției (cc);

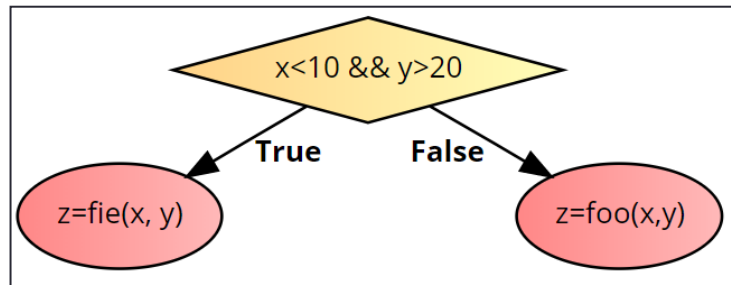
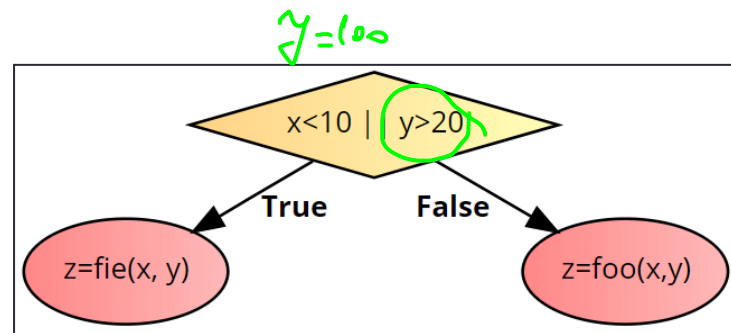
dcc se aplică doar atunci când decizia este formată din mai multe condiții.



DCC. Observații

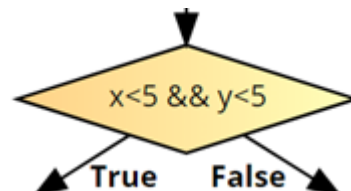
- condițiile logice care folosesc operatorii `&&` și `||` nu pot fi acoperite prin **dcc**, deoarece compilatorul realizează diverse optimizări (scurt-circuitare la evaluare);

Este necesară acoperirea condițiilor multiple.



Acoperirea condițiilor multiple. Definiție

- **acoperirea condițiilor multiple** (*engl. multiple condition coverage*, mcc):
 - proiectarea cazurilor de testare se realizează astfel încât:
 - toate combinațiile posibile ale valorilor de ieșire ale unei condiții, în fiecare decizie, să fie parcurse cel puțin o dată;



- regulă de selectare:
 - fiecare decizie care conține mai multe condiții, va combina fiecare condiție selectată care este evaluată la `true` sau `false` cu celelalte condiții în toate variantele posibile și împreună cu decizia evaluată la `true` sau `false` se vor găsi pe cel puțin un drum.

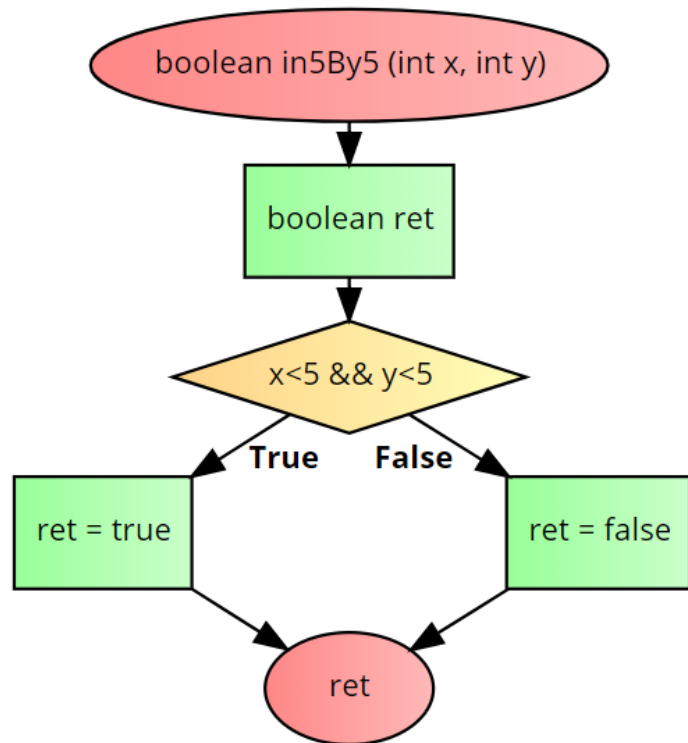
MCC. Exemplu

Fiecare *condiție* din fiecare decizie trebuie să fie executată în toate combinațiile posibile cu toate celelalte condiții din cadrul aceleași decizii.

// returnează true dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {  
    boolean ret;  
    if (x<5 && y<5)  
        ret = true;  
    else ret = false;  
    return ret;  
}
```

Care este numărul minim de cazuri de testare necesar ?



MCC. Exemplu (cont.)

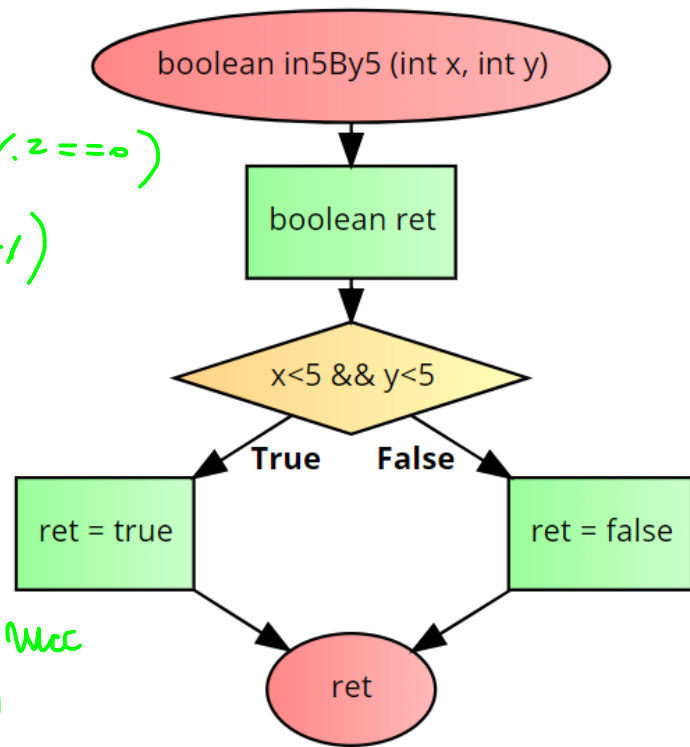
// returnează true dacă (x,y) este în cadranul (5,5).

```
boolean in5By5 (int x, int y) {
    boolean ret;
    if (x<5 && y<5)
        ret = true;
    else ret = false;
    return ret;
}
```

if (x < 100 && x % 2 == 0)
(x == 0 || x == 1)
x = 0
T || F
F || T

Care este numărul minim de cazuri de testare necesar ?

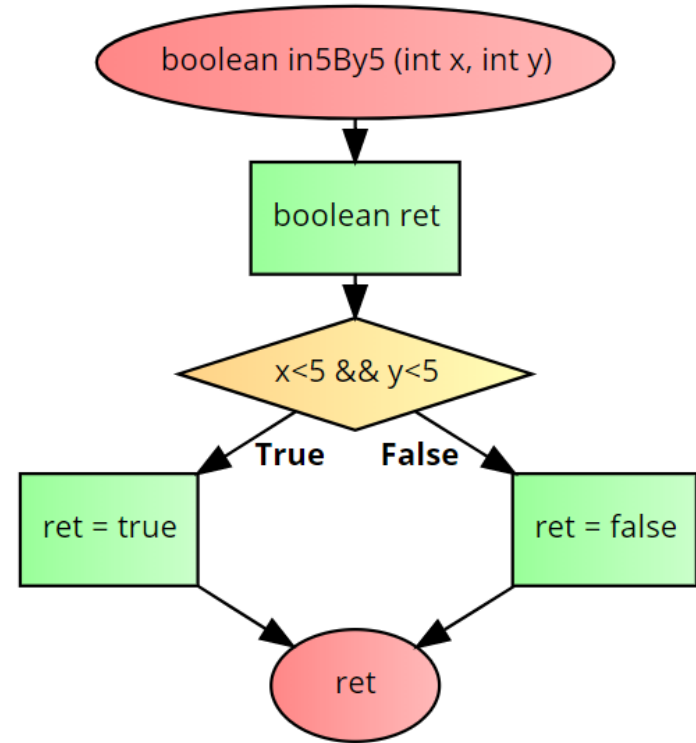
TC	x	y	Decision (x<5) && (y<5)	Expected result	Actual result
1	2	3	T && T = T	true	true
2	9	7	F && F = F	false	false
3	2	7	T && F = F	false	false
4	9	3	F && T = F	false	false



Criteriu de acoperire a condițiilor multiple îndeplinit 100%. Toate combinațiile posibile ale condițiilor au fost explorate.

MCC vs. DCC

- mcc ==> dcc;
 - prin acoperirea multiplă a condițiilor se acoperă deciziile și condițiile;



Testare bazată pe acoperirea codului sursă. Reguli de acoperire minimală

$\exists c$

- dacă programul are o singură condiție în fiecare vârf de decizie, atunci se aplică

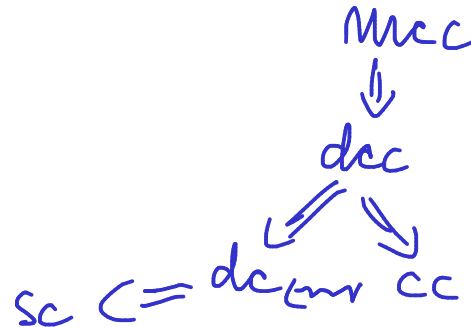
- **acoperirea deciziilor (dc)**, în acest caz **dc = cc**;

$$mcc = dc = \underbrace{dc} = \underbrace{cc}$$

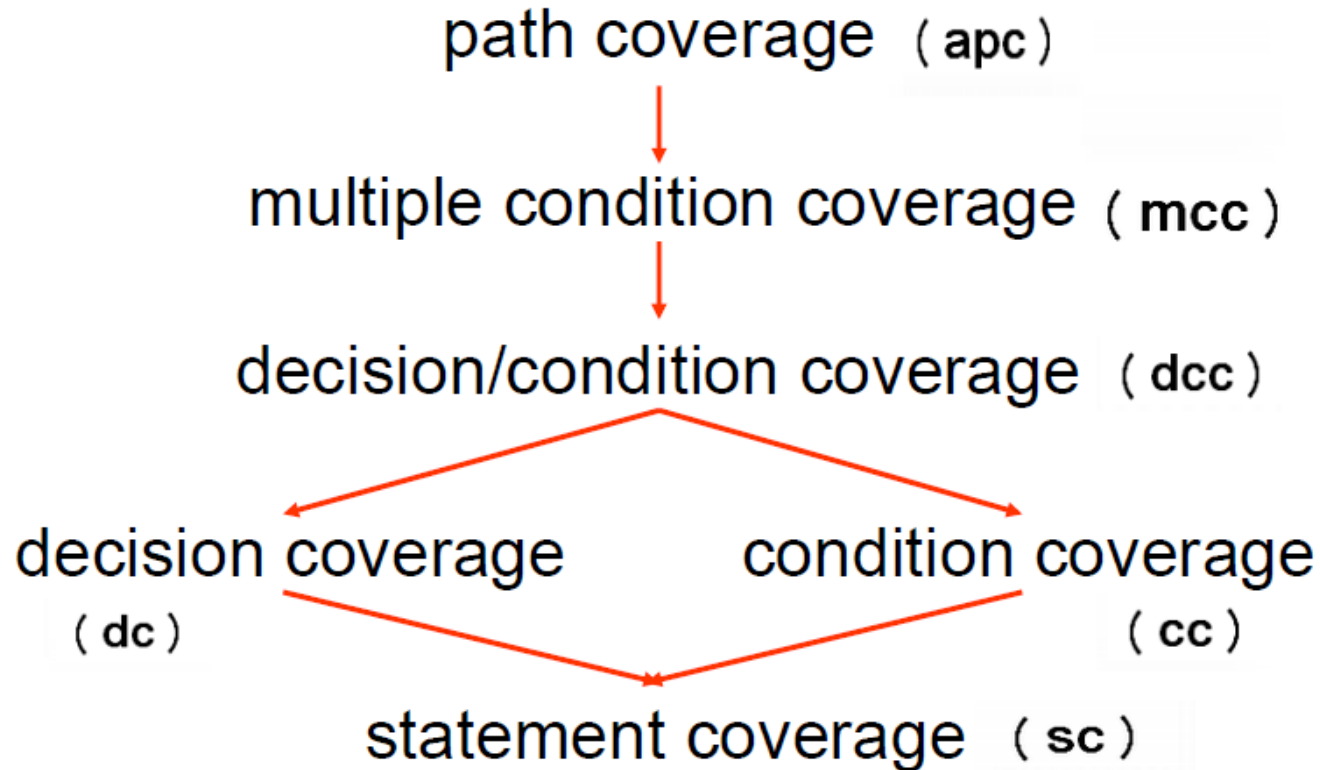
$\exists c_1, c_2 \dots c_n$

- dacă programul are condiții multiple în vârfuri de decizie, atunci se aplică

- **acoperirea condițiilor multiple (mcc)**.



SC vs DC. vs. CC vs. DCC vs. MCC vs. APC

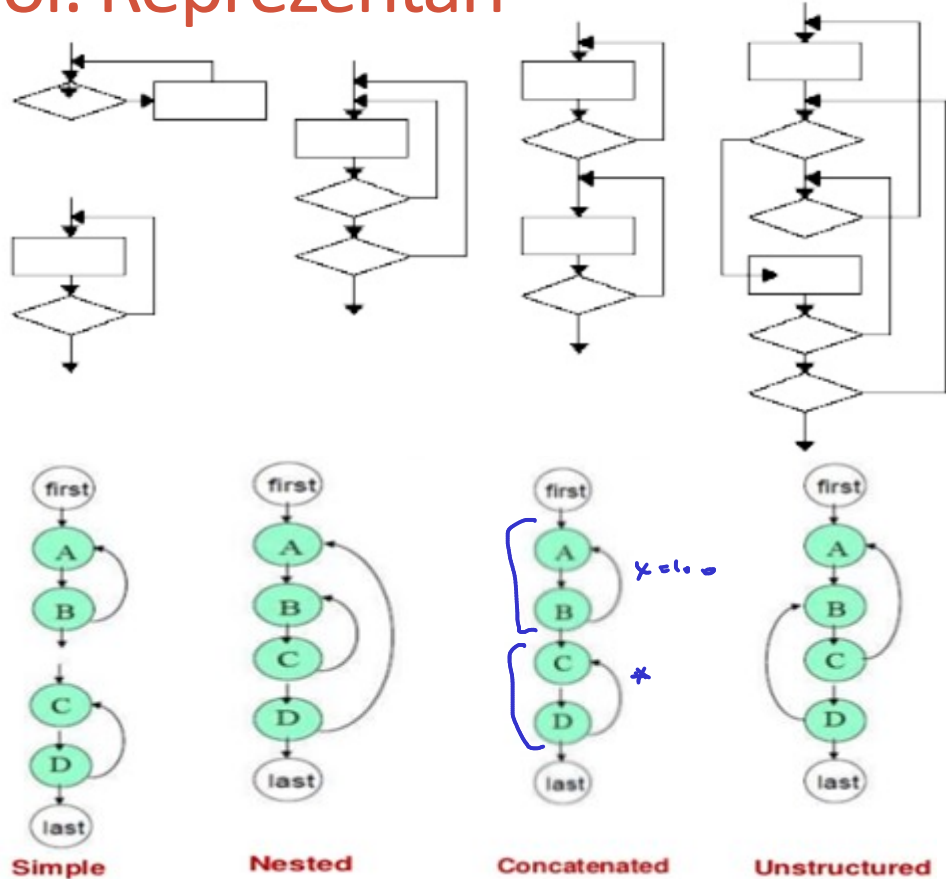


Acoperirea buclelor. Definiție

- **acoperirea buclelor** (*engl. loop coverage*, **lc**);
 - proiectarea cazurilor de testare astfel încât structurile repetitive să fie iterate de un număr *variabil* de ori;

LC. Clasificarea buclelor. Reprezentări

- tipuri de bucle:
 - • simple;
 - • imbricate;
 - • concatenate;
 - • nestructurate.



LC. Buclă simplă

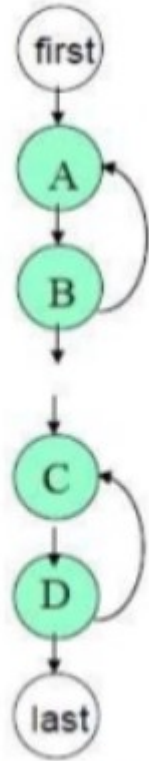
7 TCS

ECF, BVA

- **buclă simplă** (n – numărul maxim de parcurgeri ale buclei):

- omiterea buclei (0 parcurgeri); $L = \{ \}$
- 1 parcurgere a buclei; $L = \{ 1 \}$
- 2 parcurgeri ale buclei (evidențiază defecte de inițializare); $L = [2, 7]$
- m parcurgeri ale buclei, unde $m < n$; $m = 100$
- n-1 parcurgeri ale buclei;
- n parcurgeri ale buclei;
- n + 1 parcurgeri ale buclei.

```
while (.. c1 .. c2 - ... cn)
for (i = 0 -> i < n-1)
    continue
```



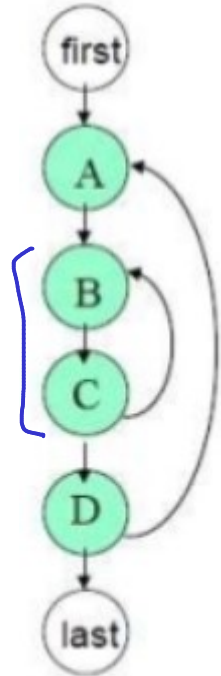
Simple

LC. Bucle imbricate

- bucle imbricate:

1. se pornește de la bucla cea mai interioară; toate celelalte bucle sunt setate pe valori minime;
2. se testează bucla cea mai interioară ca și buclă simplă, păstrând buclele exterioare la valoarea minimă a parametrului de iterație;
3. se progresează spre exterior, testându-se următoarea buclă și păstrând buclele exterioare la valorile minime;
4. se continuă până când toate buclele sunt testate.

b. ext b. int
1) 0 0
1 1
2 2
... ...
n-1 n-1
n n
n+1 n+1



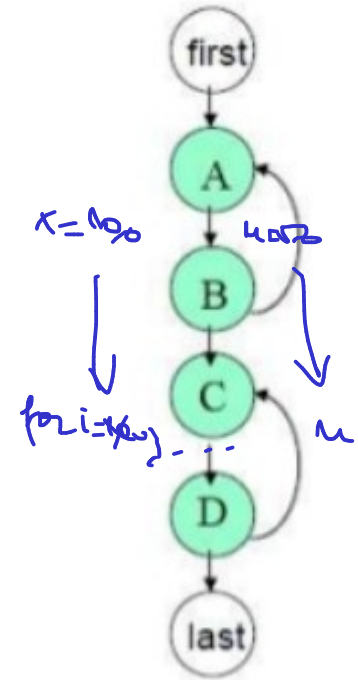
for(---)
{
 for(---)
 { ... }

}

→ apel de
 nucleu
Nested

LC. Bucle concatenate

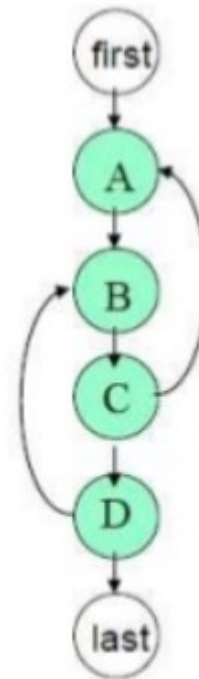
- **bucle concatenate:**
 - dacă buclele sunt independente unele de altele:
 - se aplică testarea buclelor simple;
 - dacă buclele sunt dependente (e.g., variabila de indexare a primei bucle este valoarea inițială a celei de a doua):
 - se aplică testarea buclelor imbricate.



Concatenated

LC. Buclă nestructurate

- **bucle nestructurate:**
 - în general, indică folosirea instrucțiunii goto;
 - se recomanda restructurarea acestui tip de buclă pentru a reflecta elementele programării structurate.



Unstructured

TESTARE WHITE-BOX VS. TESTARE BLACK-BOX

Testare White-Box. Avantaje si dezavantaje

Testare White-Box vs. Testare Black-Box

Testare White-Box

Avantaje

- cazurile de testare sunt proiectate pe baza
→ **structurii interne a codului sursă**, i.e., în funcție de structurile de programare folosite;
- identifică disfuncționalități în execuția anumitor secvențe de cod, e.g., unele structuri de programare nu sunt acoperite;
 - permite acoperirea cu teste a codului scris;

Dezavantaje

- **nu poate testa cerințe care nu sunt implementate**, nu poate identifica bug-urile din codul sursă care lipsește;
- • **proiectarea cazurilor de testare poate începe doar după implementare**;
- testerul trebuie să cunoască limbajul de programare în care a fost elaborat codul sursă;
- **ineficientă pentru module de mari dimensiuni.**

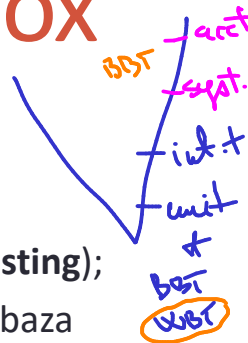
Testarea Black-Box vs. Testare White-Box

Testare Black-Box

- Testare funcțională, testare comportamentală (*engl. behavioral testing*);
- cazurile de testare sunt proiectate pe baza specificațiilor, nu este necesar să avem acces la codul sursă;
- • **suprinde ambiguitățile sau inconsistențele din specificații;**
- nu se există informații despre implementare;
- activitatea testerului este independentă de cea a programatorului; testerul poate proiecta cazurile de testare înainte de finalizarea codului sursă;
- eficientă și pentru module de mari dimensiuni.

Testare White-Box

- Testare structurală (*engl. structural testing*);
- cazurile de testare sunt proiectate pe baza structurii interne a codului sursă, i.e., în funcție de structurile de programare folosite;
- **nu poate testa cerințe care nu sunt implementate;**
- proiectarea cazurilor de testare poate începe doar după implementare;
- ineficientă pentru module de mari dimensiuni – construirea CFG și calculul CC sunt activități costisitoare.



Testarea Black-Box vs. Testarea White-Box

Întrebări:

- Ce cazuri de testare trebuie actualizate după modificarea specificațiilor și a codului sursă asociat?

spec₁ BBT₁ ✓ WBT₁ ✓ CFG₁

spec₂ BBT₂ WBT₂ CFG₂

- Ce cazuri de testare trebuie actualizate după modificarea codului sursă, ~~fără~~ modificarea specificațiilor?

spec₁ BBT₁ WBT₃ CFG₃

PENTRU EXAMEN...

Pentru examen...

- **testare white-box:**
 - definiție, caracteristici, avantaje și dezavantaje;
 - CFG (definiție și construire), drumuri independente (definiție), CC (definiție, 3 moduri de calcul);
 - construirea CFG, determinarea drumurilor independente și calculul CC (3 moduri) pentru metode concrete;
 - criteriile de acoperire **apc, sc, dc, cc, dcc, mcc** și **lc** (definiție, compararea a două criterii, relațiile existente între criterii);
 - testare black-box vs. testare white-box.

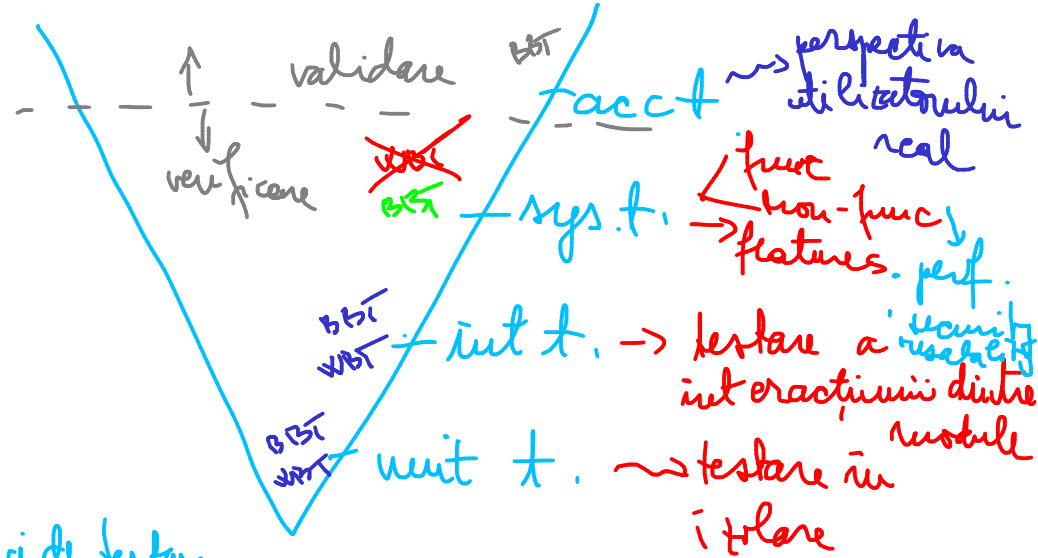
Cursul următor...

- Niveluri de testare
 - Definiție. Clasificare
- Testare unitară
- Testare de integrare
- Testare funcțională
- Testare de sistem
 - Testare funcțională
 - Testare non-funcțională
- Testare de acceptare
- Nivel de testare vs. Tip de testare

(4)

m

tehnici de testare
m



Referințe bibliografice

- **[Myers2004]** Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004.
- **[NT2005]** K. Naik and P. Tripathy. *Software Testing and Quality Assurance*, Wiley Publishing, 2005.
- **[Patton2005]** R. Patton, *Software Testing*, Sams Publishing, 2005.
- **[Collard2003]** J. F. Collard, I. Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- **[Beizer1990]** Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold., New York, 1990.