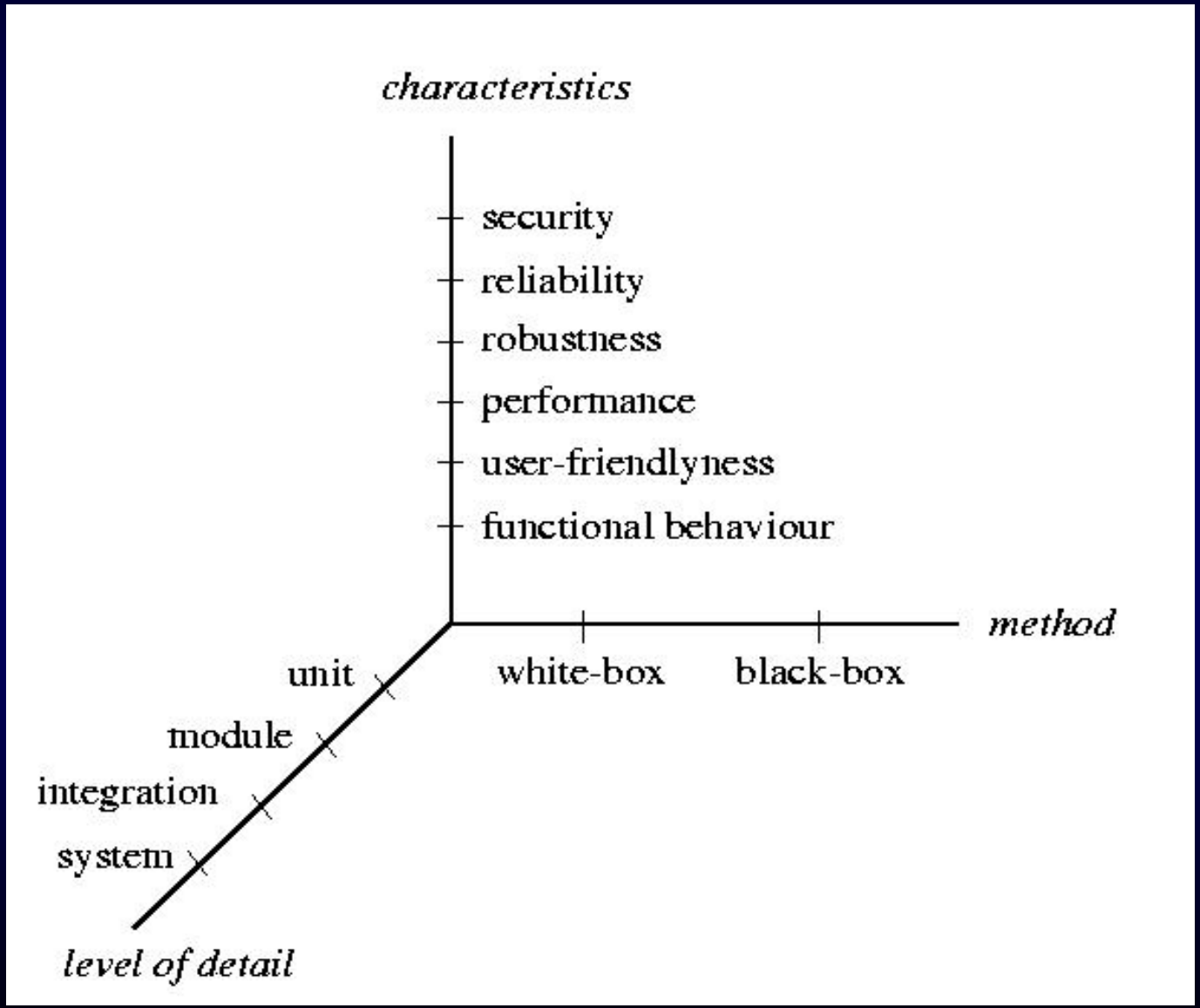
Test case design techniques I: Whitebox testing

CISS

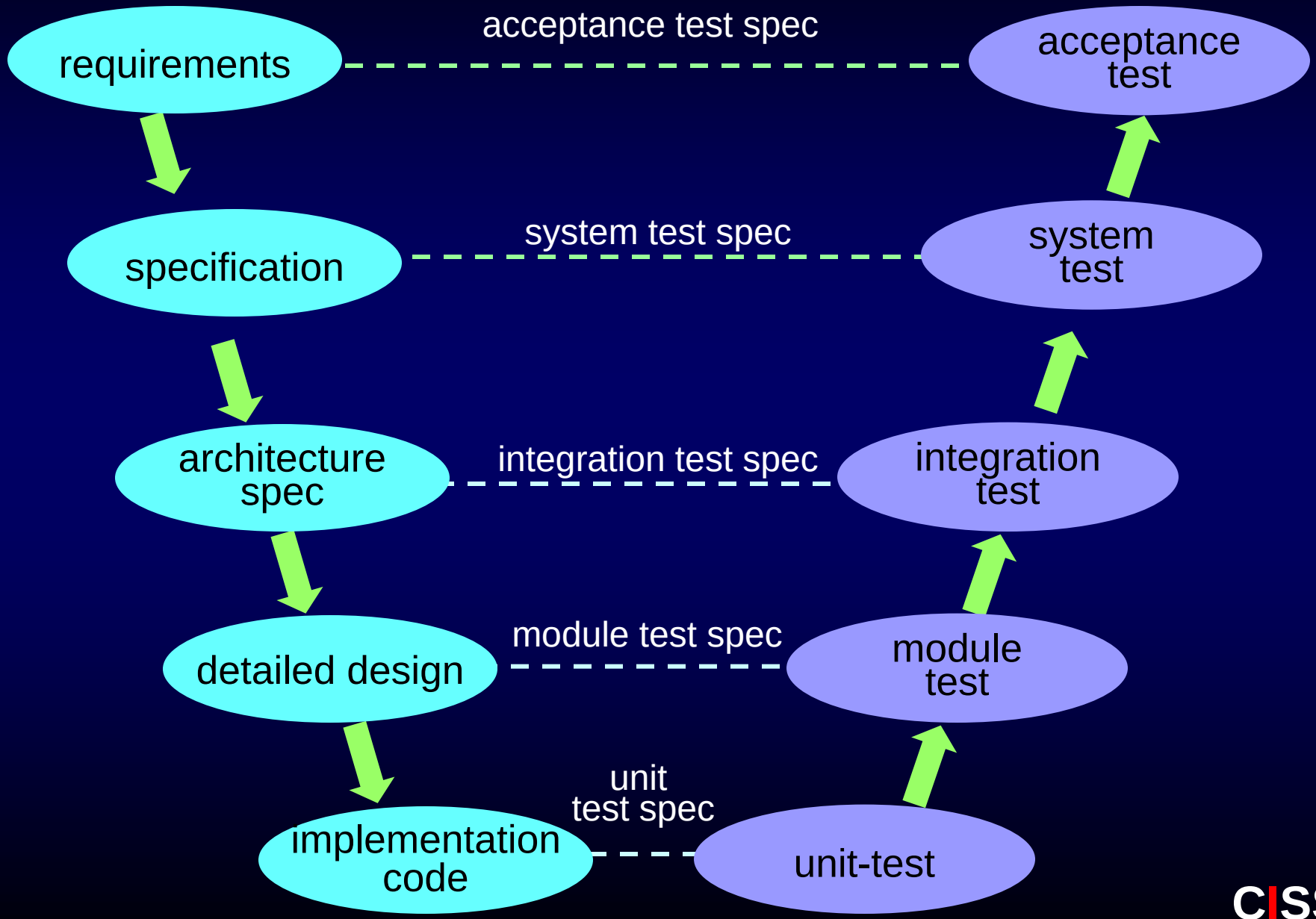
Overview

- What is a test case
- Sources for test case derivation
- Test case execution
- White box testing
  - Flowgraphs
  - Test criteria/coverage
    - Statement / branch / decision / condition / path coverage
  - Looptesting
    - Data flow testing
  - Def-use pairs
  - Efficiency of different criteria

Types of Testing

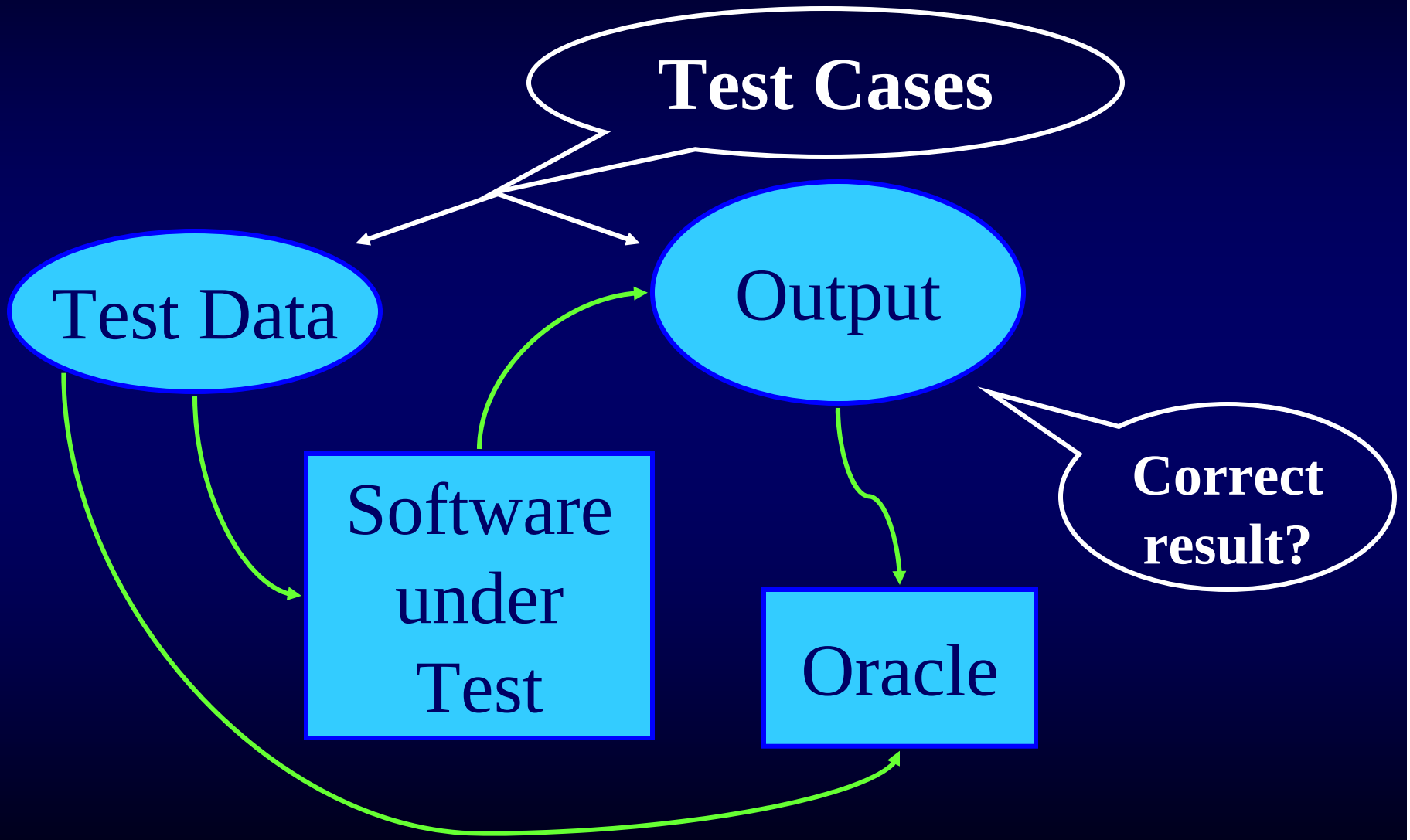


V - Model


requirements
acceptance test spec
acceptance test
specification
system test spec
system test
architecture spec
integration test spec
integration test
detailed design
module test spec
module test
implementation code
unit test spec
unit-test

CISS

What is a Test?



Development of Test Cases

Complete testing is impossible

⇓

Testing cannot guarantee the absence of faults

⇓

How to select subset of test cases from all possible test cases with a high chance of detecting most faults ?

⇓

Test Case Design Strategies

Sources for test case design

- The requirements to the program (its specification)
  - An informal description
  - A set of scenarios (use cases)
  - A set of sequence diagrams
  - A state machine
- The program itself
- A set of selection criteria
- Heuristics
- Experience

Test case execution

- Single stepping via a debugger
  - Very clumsy for large programs
  - Hard to rerun
- Manual via a set of function calls
  - Hard to check when the number of test cases grows
- Fully automatic without programmers assistance
  - Not possible so far
  - Offline/online
- Fully automatic with programmers assistance
  - Started with Junit
  - State of the art
  - Growing interest

White-Box Testing

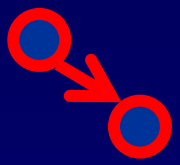
- Testing based on program code
- Extent to which (source) code is executed, i.e. *Covered*
- Different kinds of *coverage* :
  - statement coverage
  - path coverage
  - (multiple-) condition coverage
  - decision / branch coverage
  - loop coverage
  - definition-use coverage
  - .....

White box testing: flow graphs

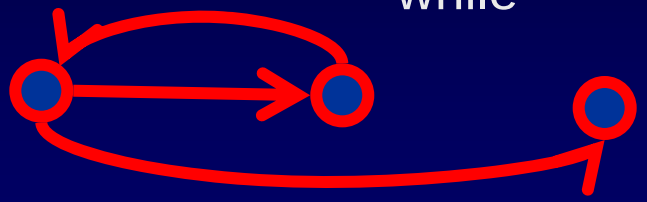
- Syntactic abstraction of source code
- Ressembles classical flow charts
- Forms the basis for white box test case generation principles
- Purpose of white box test case generation: Coverage of the flow graph in accordance with one or more test criteria

Flow graph construction

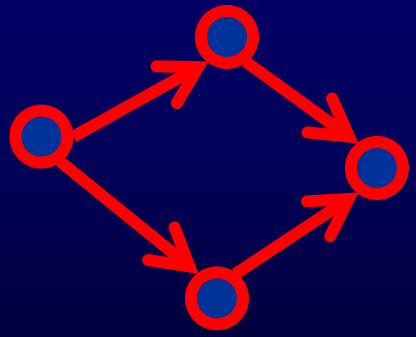
sequence



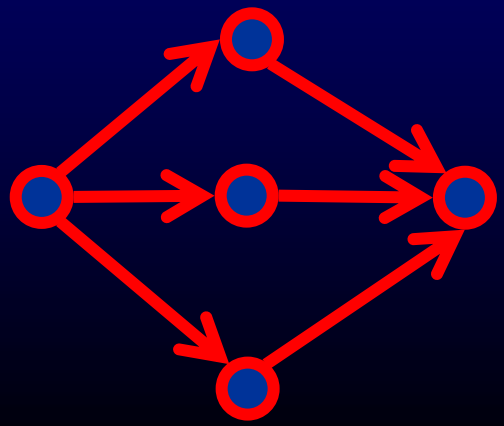
while



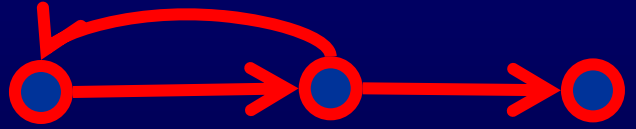
if



case



until



White-Box : Statement Testing

- Execute every statement of a program
- Relatively weak criterion
- Weakest white-box criterion

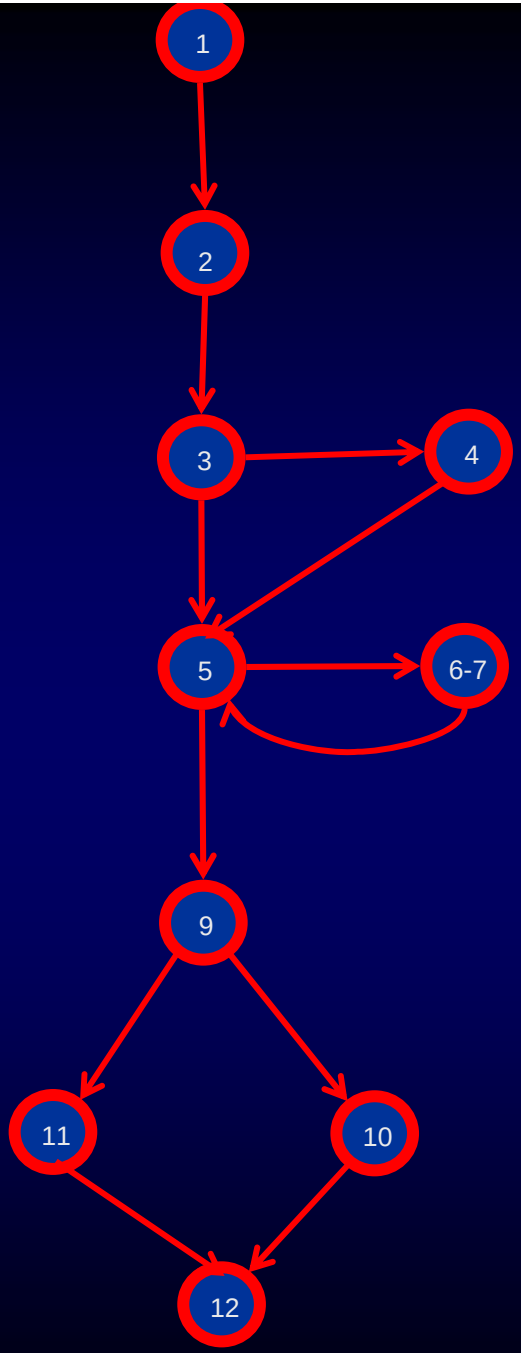
Example : Statement Testing

(*result* = 0+1+…+|value|, if this <= *maxint*, error otherwise)

```
1       PROGRAM  maxsum ( maxint, value : INT )
2               INT    result := 0 ;   i := 0 ;
3               IF   value < 0
4               THEN   value  :=  - value ;
5               WHILE   ( i < value )  AND  ( result <= maxint )
6               DO       i  :=  i + 1 ;
7                        result  :=  result + i ;
8               OD;
9               IF   result <= maxint
10              THEN   OUTPUT ( result )
11              ELSE   OUTPUT ( “too large” )
12      END.
```

CISS

```
1    PROGRAM  maxsum ( maxint, value : INT )
2                    INT    result := 0 ;   i := 0 ;
3                    IF   value < 0
4                    THEN   value  :=  - value ;
5                    WHILE   ( i < value )  AND  ( result <= maxint )
6                    DO           i  :=  i + 1 ;
7                                 result  :=  result + i ;
8                    OD;
9                    IF   result <= maxint
10                   THEN   OUTPUT ( result )
11                   ELSE   OUTPUT ( “too large” )
12           END.
```



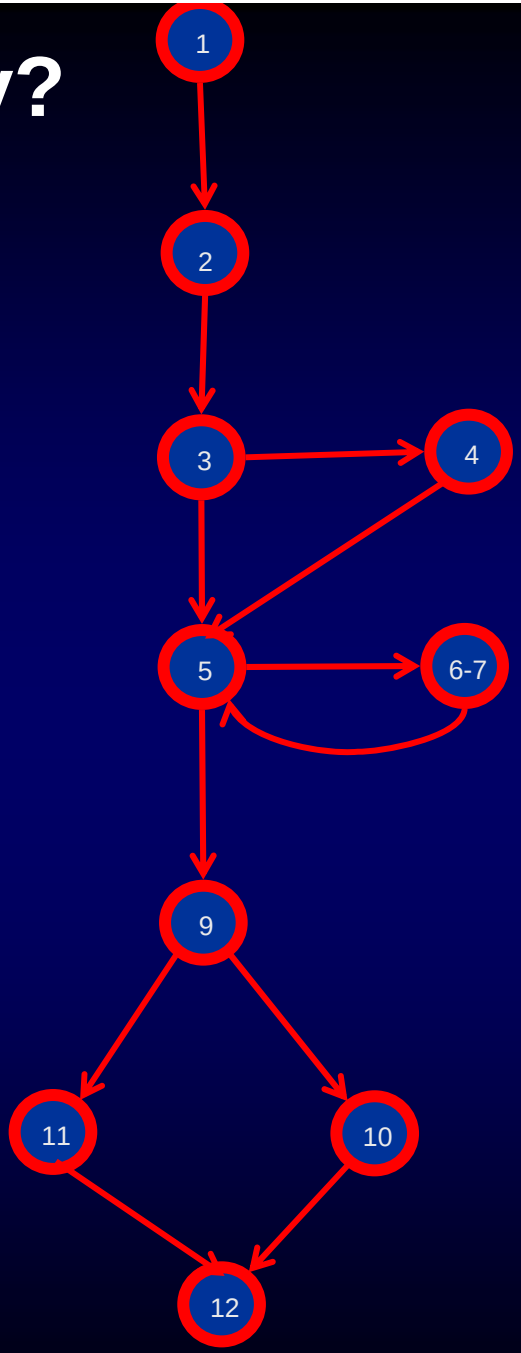
CISS

Flow graph: Cyclomatic complexity

- #edges - #nodes + 2
- Defines the maximal number of test cases needed to provide statement coverage
- Mostly applicable for Unit testing
- Strategy for statement coverage:
  1. Derive flow graph
  2. Find cyclomatic complexity #c
  3. Determine at most #c independent paths through the program (add one new edge for each test case)
  4. Prepare test cases covering the edges for each path (possibly fewer than #c cases)

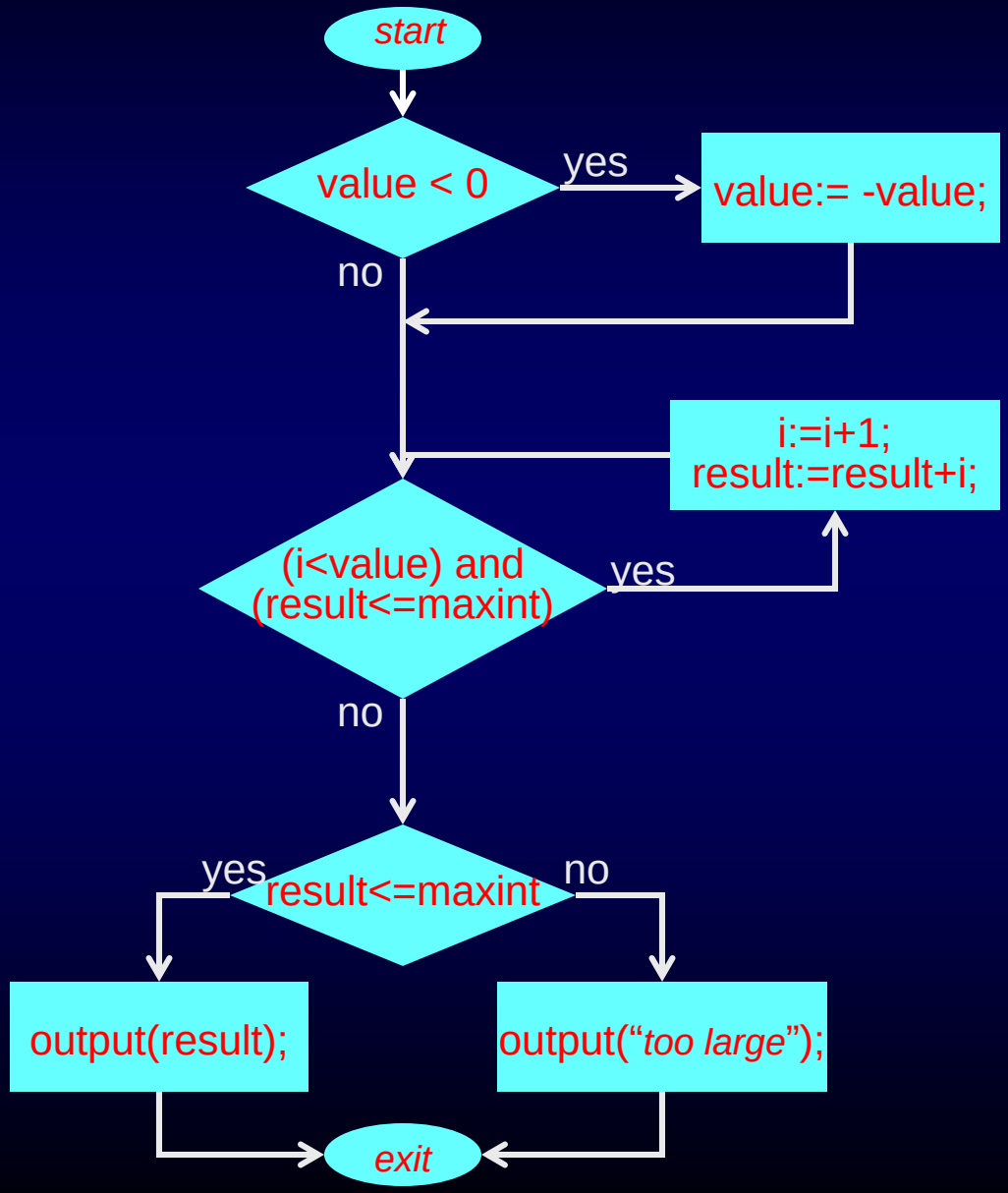
Cyclomatic complexity?

```
1    PROGRAM  maxsum ( maxint, value : INT )
2                    INT    result := 0 ;   i := 0 ;
3                    IF   value < 0
4                    THEN   value  :=  - value ;
5                    WHILE   ( i < value )  AND  ( result <= maxint )
6                    DO           i  :=  i + 1 ;
7                                 result  :=  result + i ;
8                    OD;
9                    IF   result <= maxint
10                   THEN   OUTPUT ( result )
11                   ELSE   OUTPUT ( “too large” )
12           END.
```



CISS

Example : Statement Testing



Tests for complete statement coverage:

maxint	*value*
10	-1
0	-1

CISS

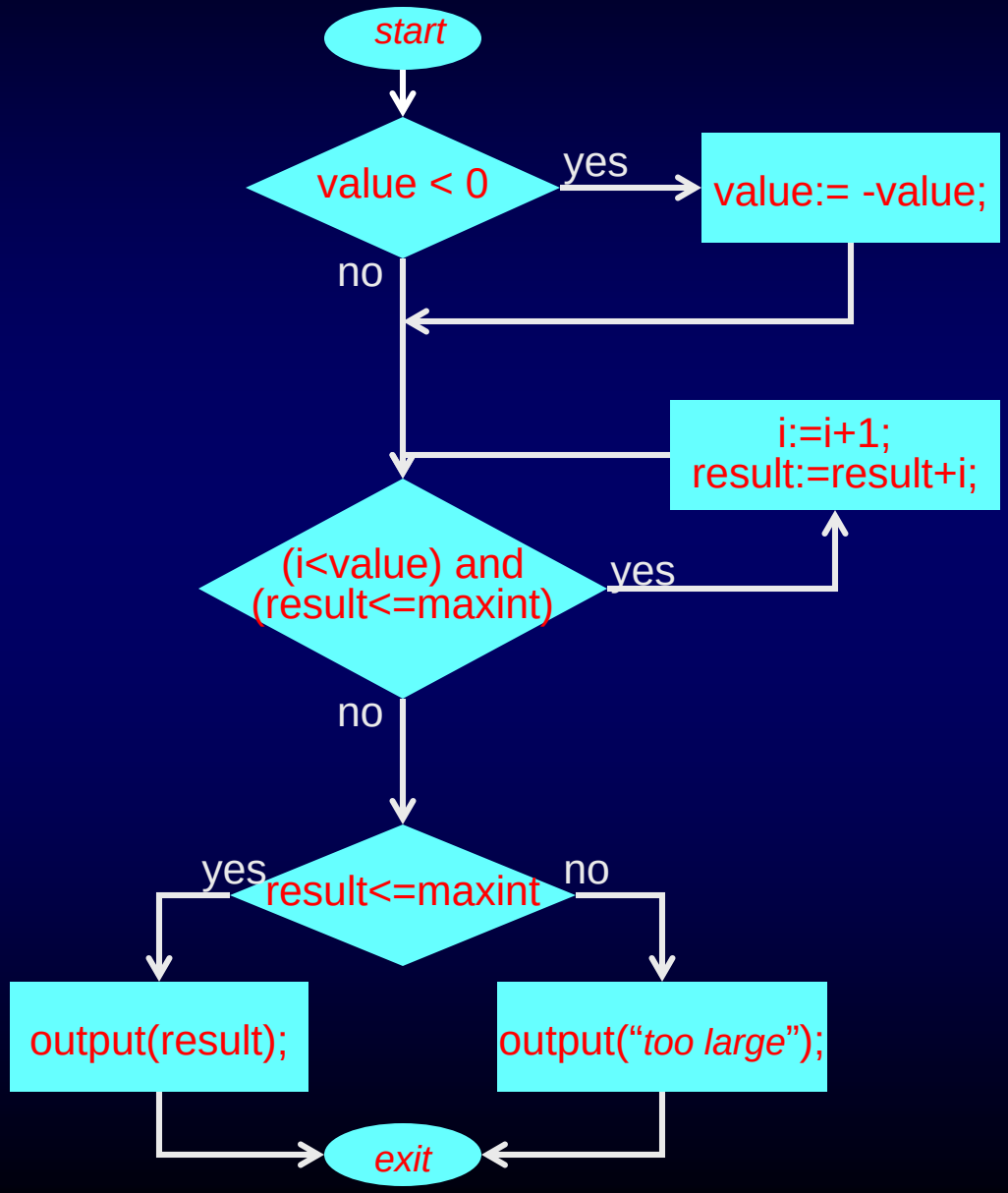
White-Box : Path Testing

- Execute every possible *path* of a program,

  i.e., every possible sequence of statements
- Strongest white-box criterion
- Usually impossible: infinitely many paths (in case of loops)
- So: not a realistic option
- But note : enormous reduction w.r.t. all possible test cases

  (each sequence of statements executed for only one value)

Example : Path Testing



Path:

start

i:=i+1;

result:=result+i;

i:=i+1;

result:=result+i;

….

….

….

i:=i+1;

result:=result+i;

output(result);

exit

CISS

White-Box : Branch Testing

- Branch testing == decision testing
- Execute every branch of a program :

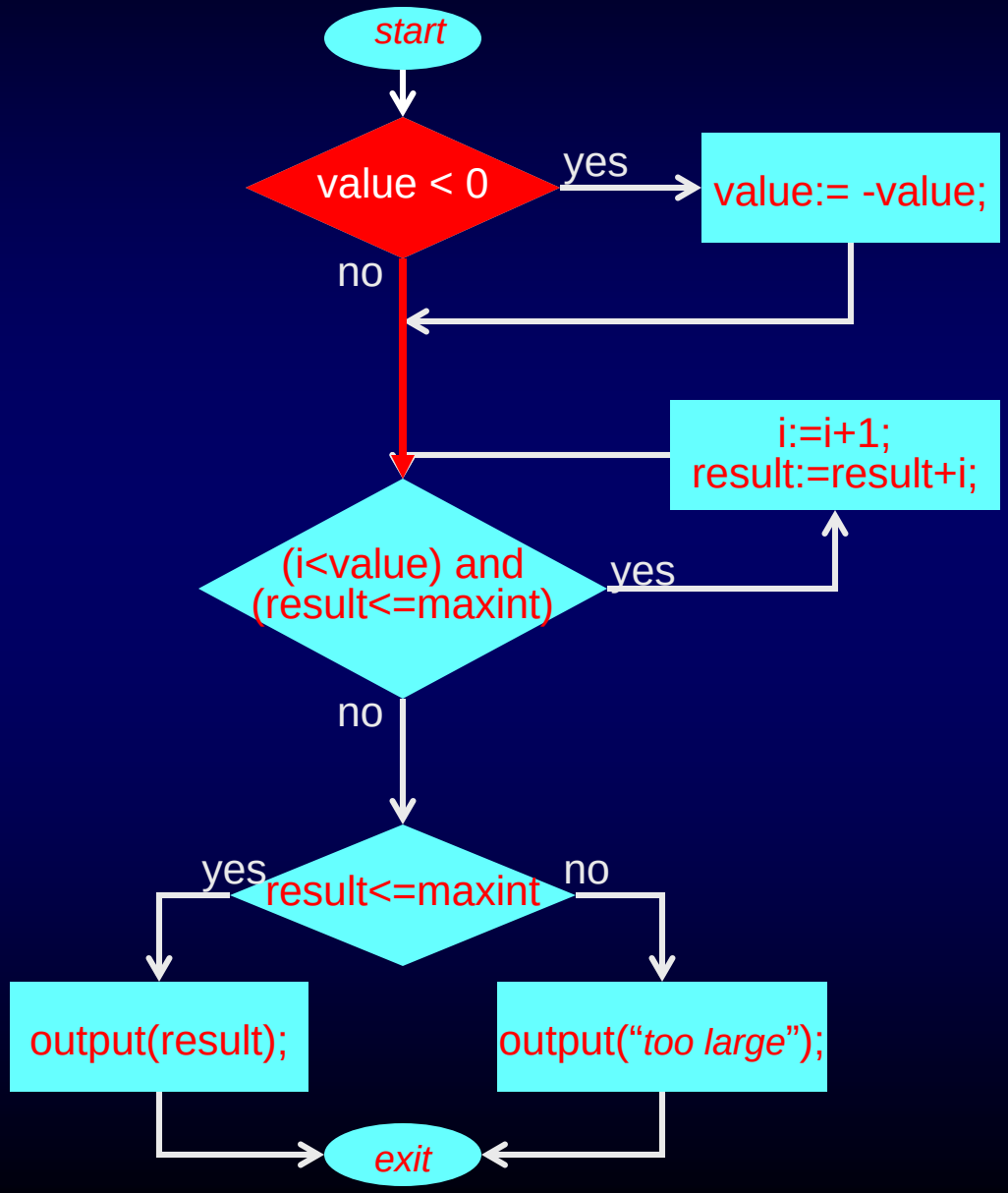
  each possible outcome of each decision occurs at least once
- Example:
  - IF b THEN s1 ELSE s2
  - IF b THEN s1; s2
  - CASE x OF

    1 : ....

    2 : ....

    3 : ....

Example : Branch Testing



Tests for complete statement coverage:

maxint	*value*
10	-1
0	-1

is not sufficient for branch coverage;

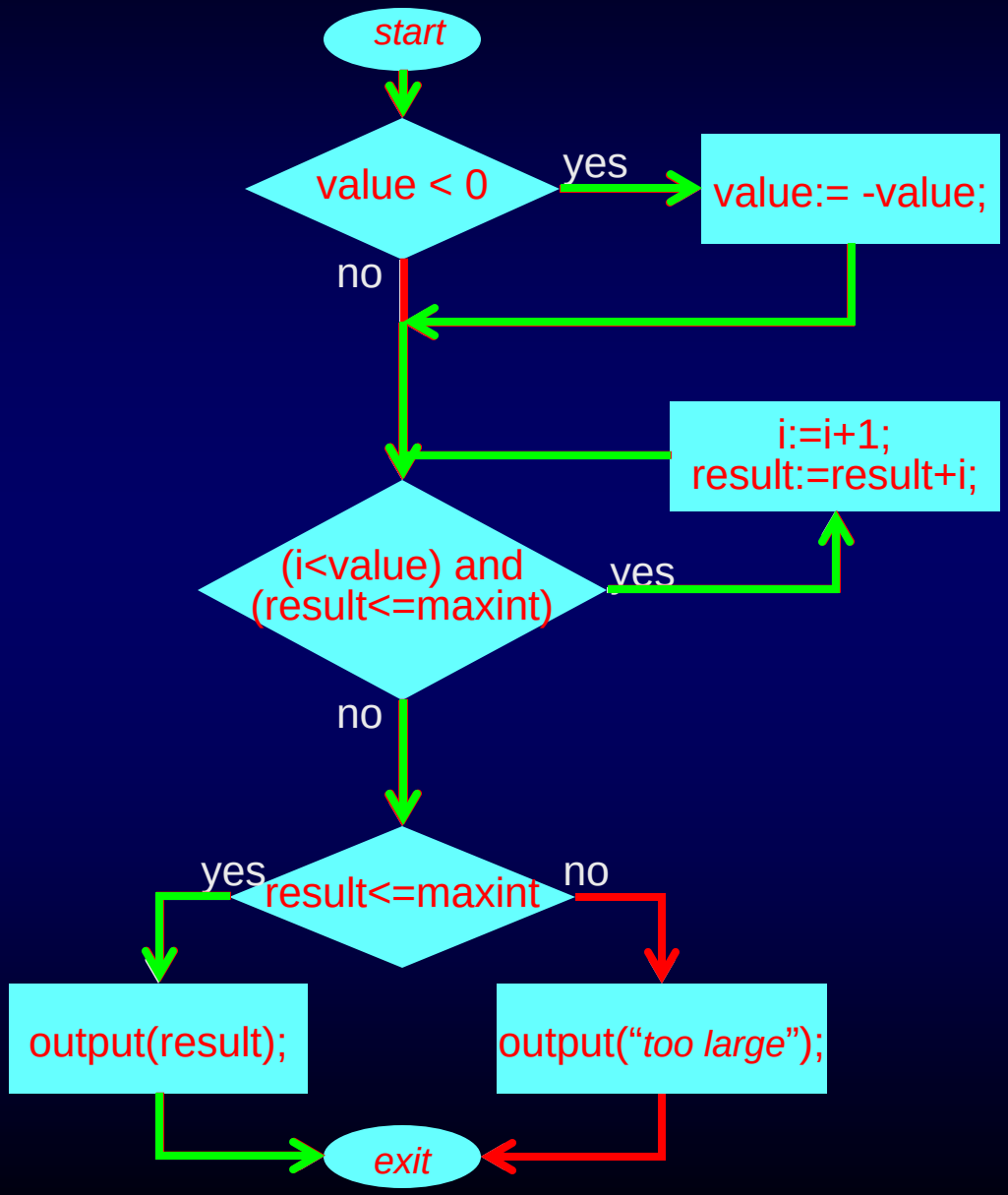
Take:

maxint	*value*
10	3
0	-1

for complete branch coverage

CISS

Example : Branch Testing



maxint	*value*
-1	-1
-1	-1
10	3
10	3

But:

No green path !

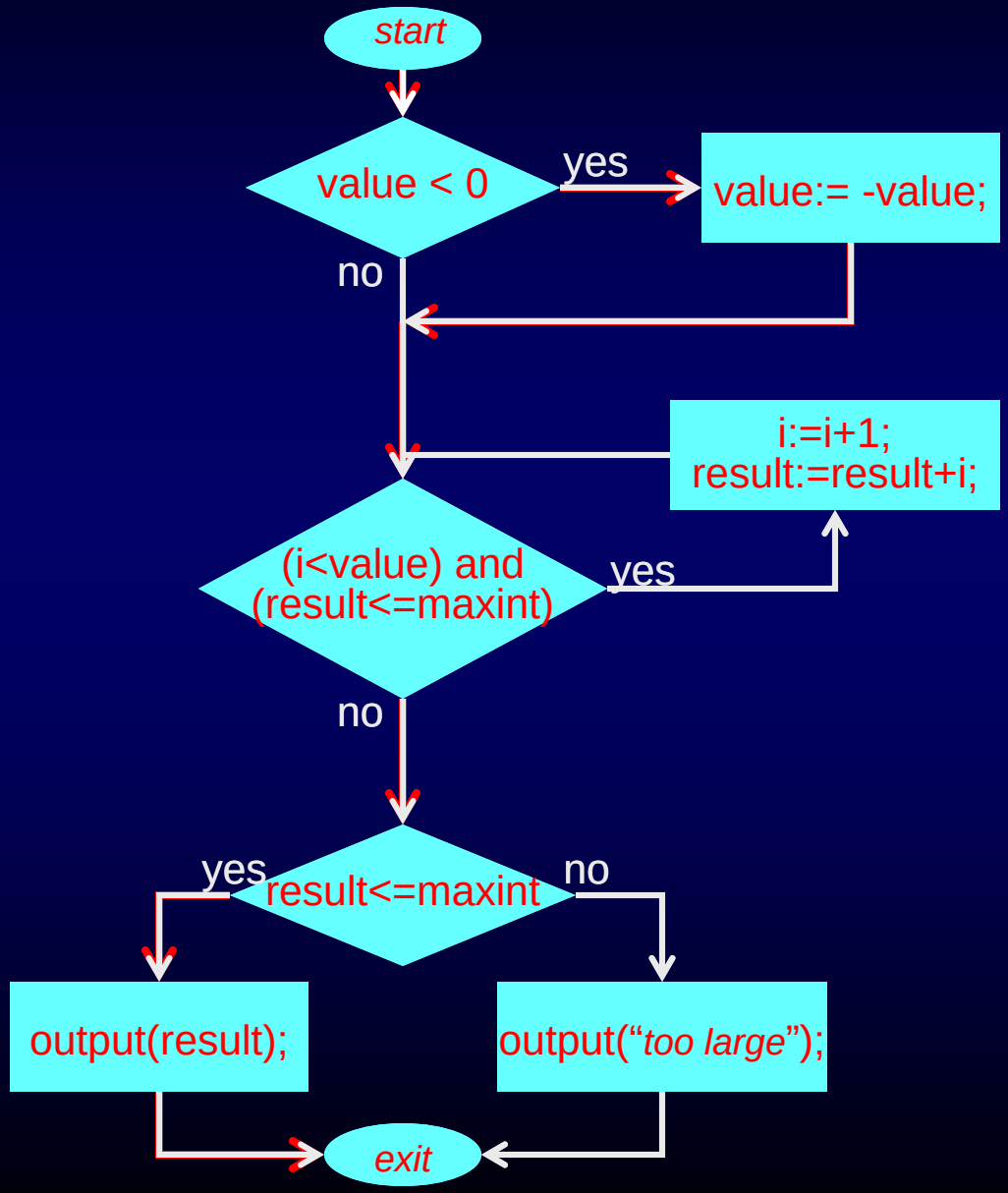
Needed :

Combination of decisions

10 -3

CISS

Example : Branch Testing



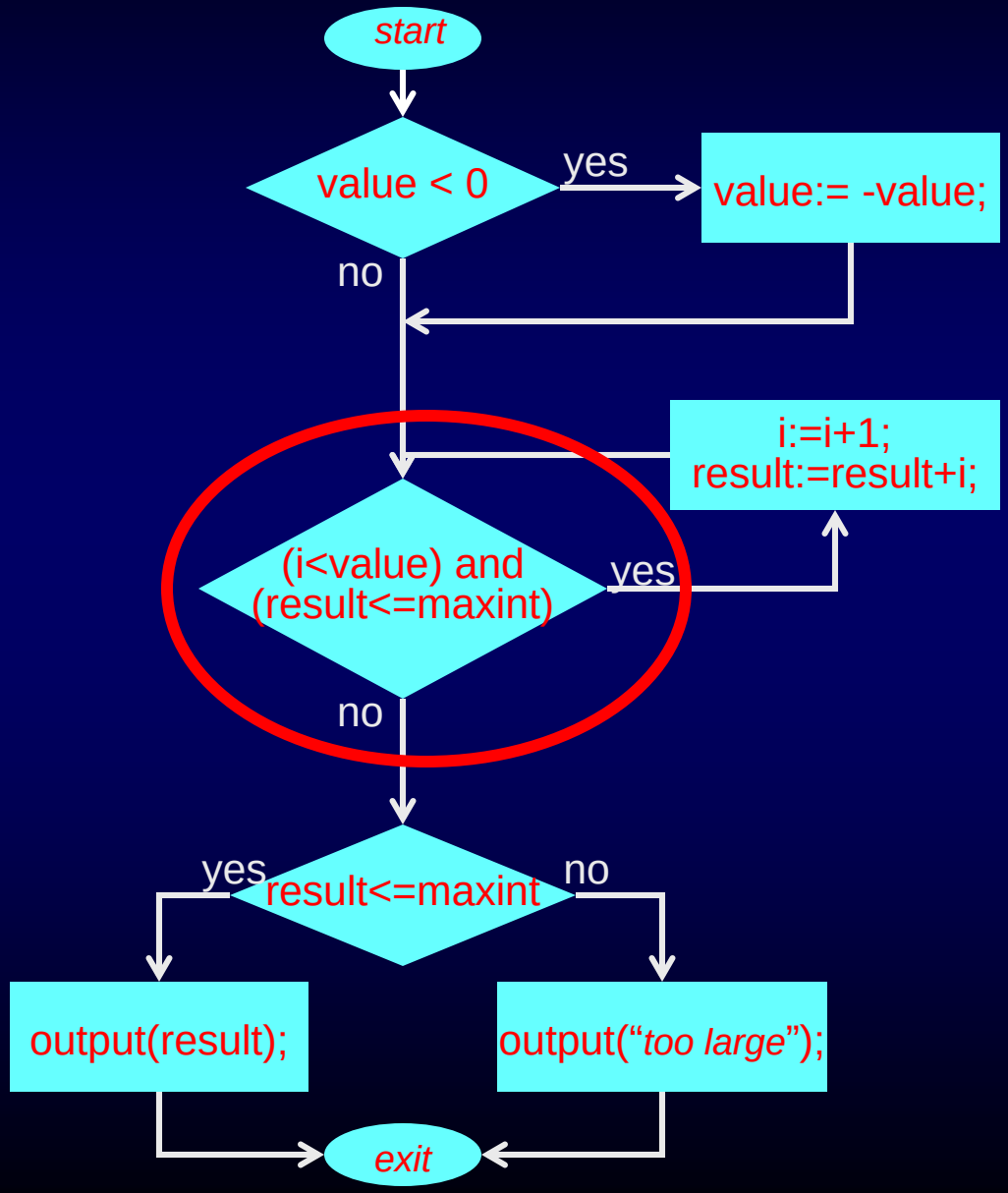
Sometimes there are infeasible paths (infeasible combinations of conditions)

CISS

White-Box : Condition Testing

- Design test cases such that each possible outcome of each condition in each decision occurs at least once
- Example:
  - decision (i < value) AND (result <= maxint)
    consists of two conditions : (i < value) AND (result <= maxint)
    test cases should be designed such that each gets value true and false at least once

Example : Condition Testing



(i = result = 0) :

maxint value	i<value	result<=maxint
-1	1	true	false
1	0	false	true

gives condition coverage
for all conditions

But it does not preserve
decision coverage

⇓

always take care that
condition coverage
preserves decision coverage :
decision / condition coverage

CISS

White-Box : Multiple Condition Testing

- Design test cases for each combination of conditions
- Example:
  - (i < value) (result <= maxint)

    false false

    false true

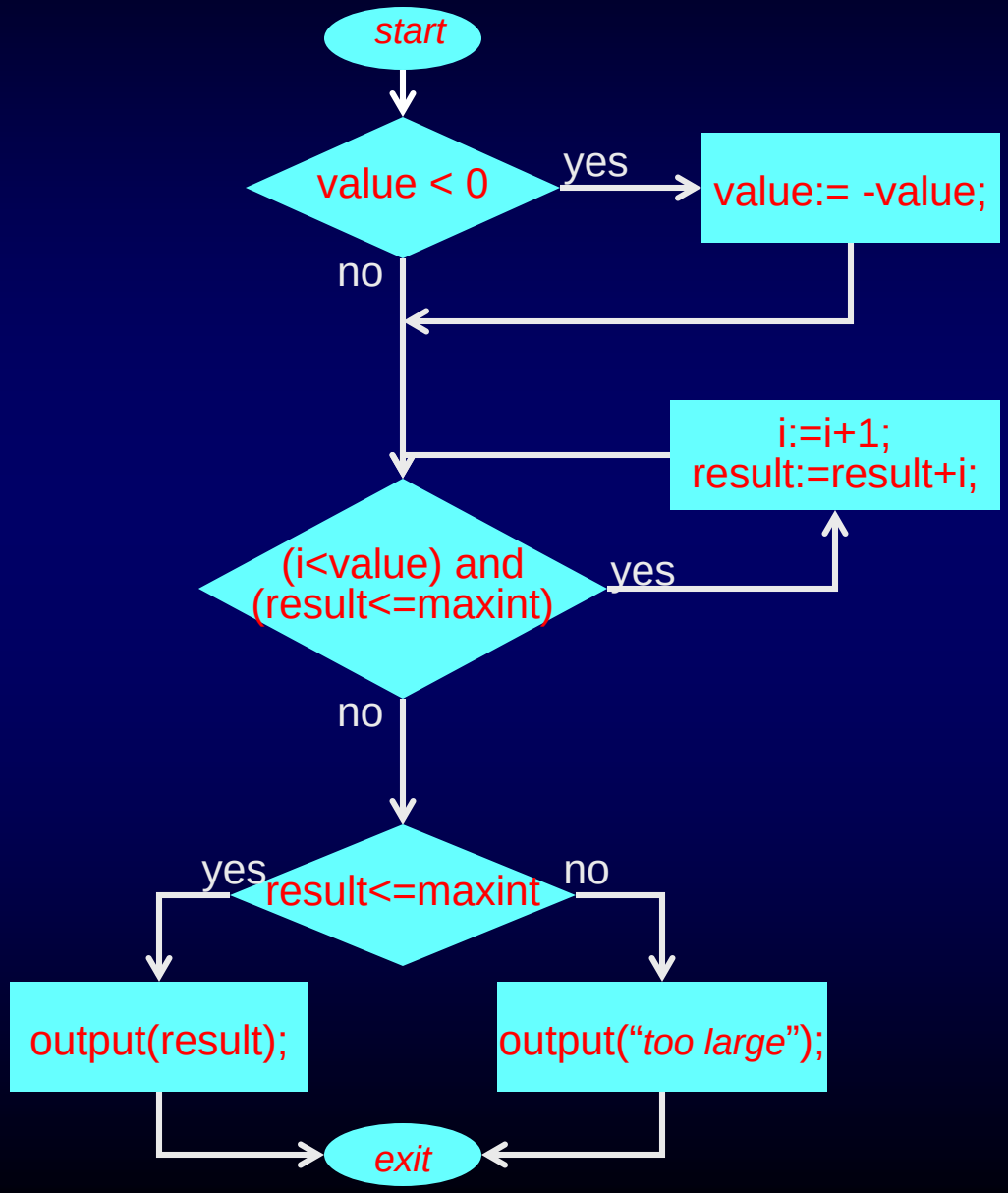
    true false

    true true
- Implies decision-, condition-, decision/condition coverage
- But : exponential blow-up
- Again : some combinations may be infeasible

White-box: loop testing

- Statement and branch coverage are not sufficient
- Single loop strategy:
  - Zero iterations
  - One iteration
  - Two iterations
  - Typical number of iterations
  - n-1, n, and n+1 iterations (n maximum number of allowable iterations)
- Nested loop strategy:
  - Single loop strategy often intractable
  - Select minimum values for outer loop(s)
  - Treat inner loop as a single loop
  - Work ‘outwards’ and choose typical values for inner loops
- Concatenated loops:
  - Treat as single, if independent
  - Treat as nested, if dependent

Example : Loop testing



Tests for complete loop coverage:

maxint	*value*
15	0
15	1
15	2
15	3
6	4
15	5

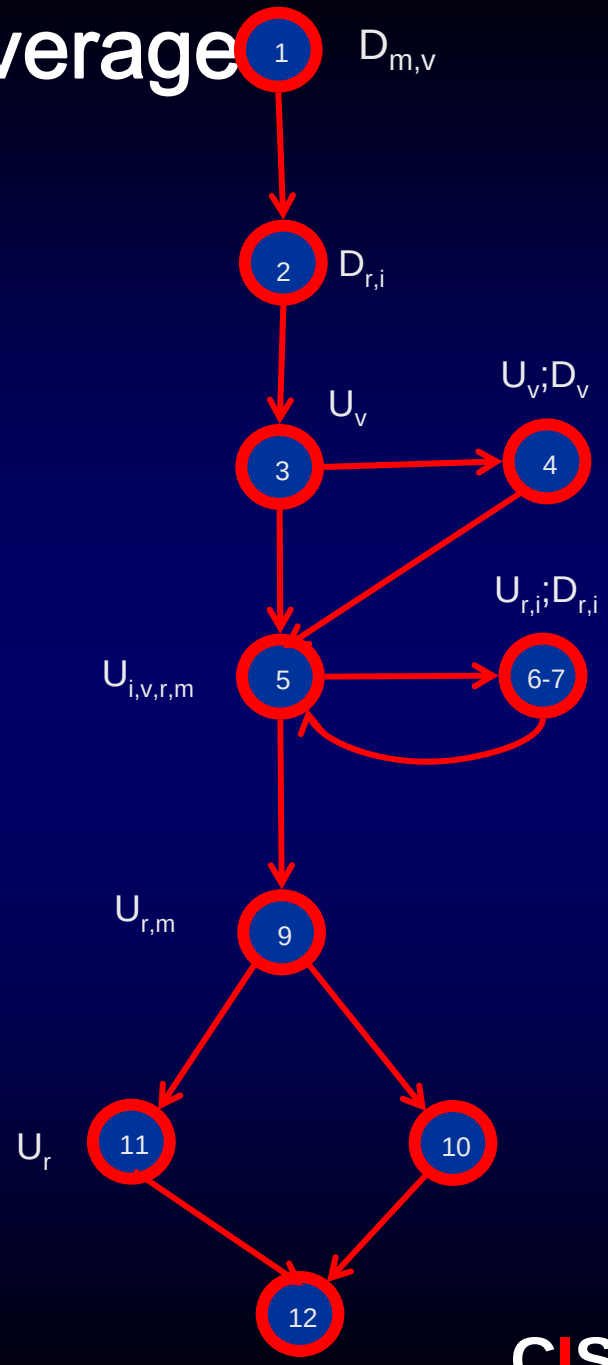
CISS

White-box testing: Data Flow criteria

- Basic idea: For each variable definition (assignment), find a path (and a corresponding test case), to its use(s). A pair (definition,use) is often called a DU pair.
- Three dominant strategies:
  - All-defs (AD) strategy: follow at least one path from each definition to some use of it
  - All-uses (AU) strategy: follow at least one path for each DU pair
  - All-du-uses strategy (ADUP): follow all paths between a DU pair
- Complements the testing power of decision coverage

Example: All-uses coverage

```
1    PROGRAM  maxsum ( maxint, value : INT )
2                      INT    result := 0 ;   i := 0 ;
3                      IF   value < 0
4                      THEN   value  :=  - value ;
5                      WHILE   ( i < value )  AND  ( result <= maxint )
6                      DO           i  :=  i + 1 ;
7                                   result  :=  result + i ;
8                      OD;
9                      IF   result <= maxint
10                     THEN   OUTPUT ( result )
11                     ELSE   OUTPUT ( “too large” )
12           END.
```



Def-use pairs:

1-3,1-5,1-9,1-4

2-5,2-9,2-6

4-5

6-5,6-9,6-11

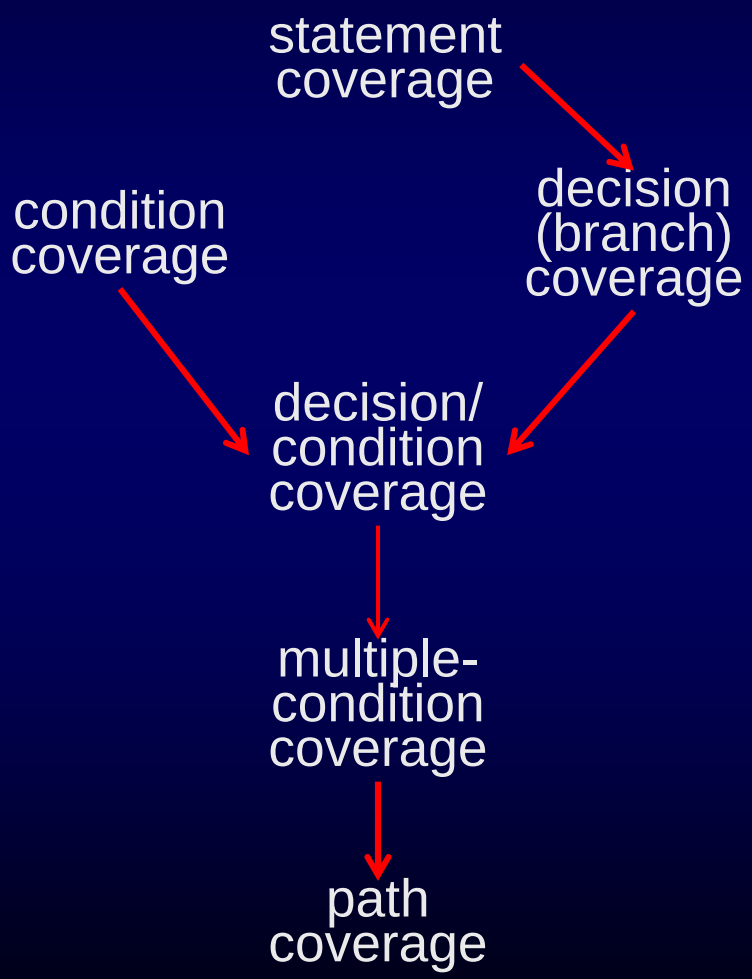
6-5-6

Tests for complete all-uses coverage:

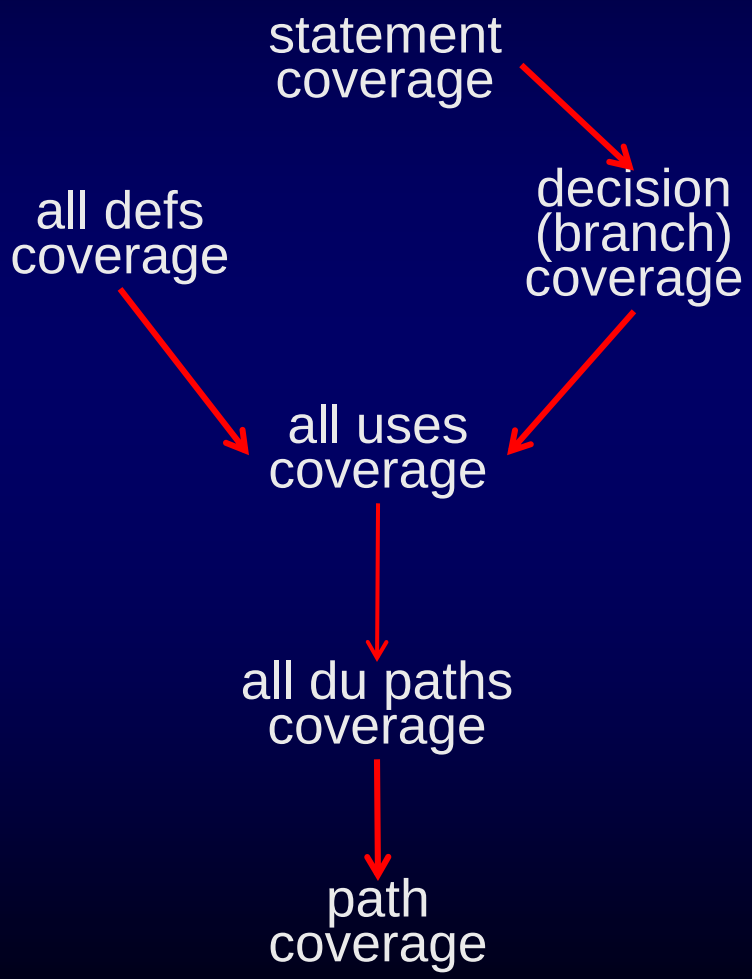
maxint	*value*
0	**0**
0	**-1**
10	**1**
10	**2**

CISS

White-Box : Overview



White-Box : Overview



Additional techniques: mutation and random testing

- Mutation testing:
  - Intended for evaluating the test cases
  - Create at set of slightly modified mutants of the original program containing errors
  - Run the test cases against the mutants
  - Criteria
    - All mutants must fail (strong)
    - All mutants will eventually fail (weak)
- Random testing:
  - Basic idea: run the program with arbitrary inputs
  - Inherent problems: How to define the oracle for arbitrary inputs and how to decide to stop?
  - Advantage: The program structure can be ignored

Efficiency of white-box techniques: two studies

Strategy	#test cases	%bugs found
Random	35	93.7
Branch	3.8	91.6
All-uses	11.3	96.3
Random	100	79.5
Branch	34	85.5
All-uses	84	90.0