Using the code to measure test adequacy
(and derive test cases)

Structural Testing

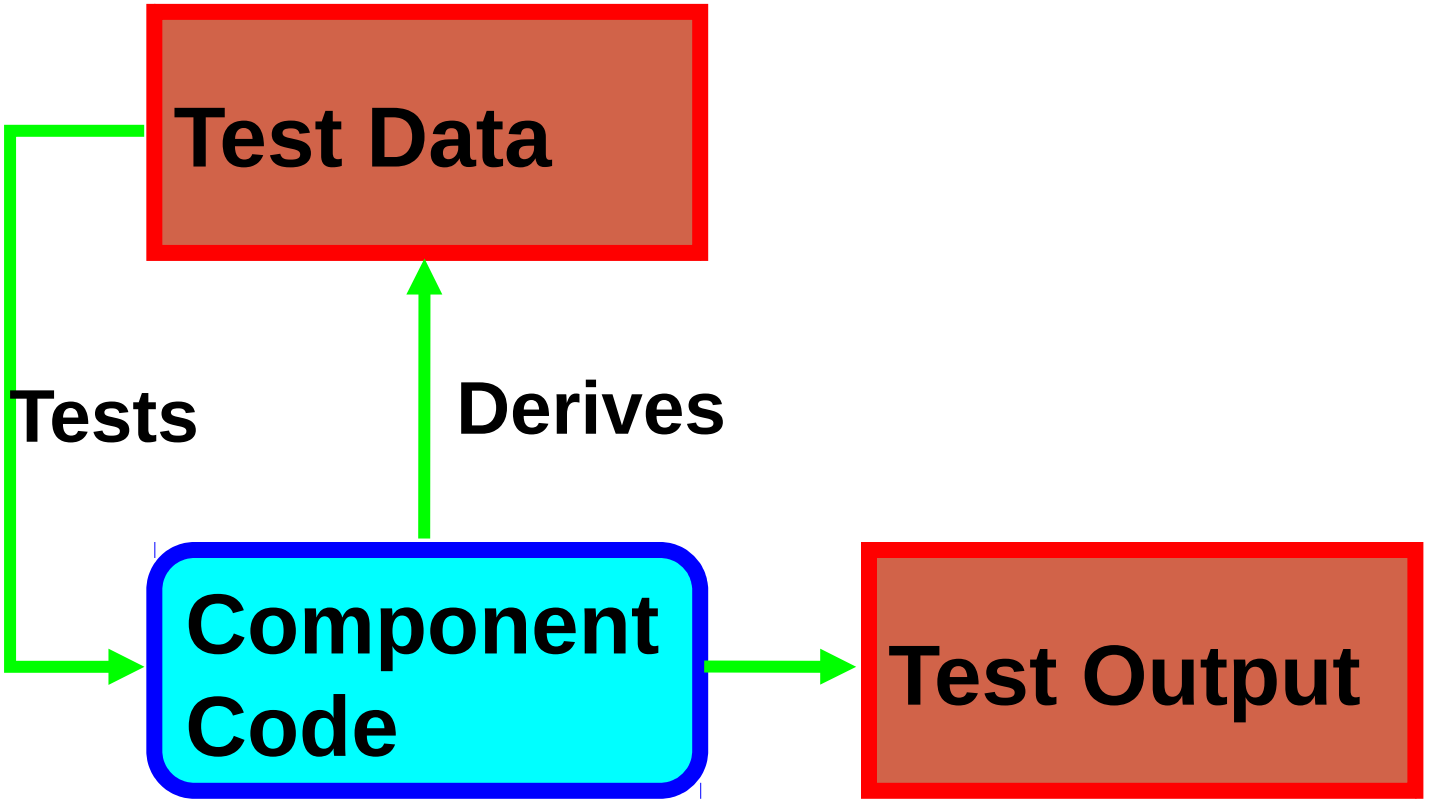
Objectives

- To describe a second approach to testing which is geared to find program defects
- To explain the use of program structure analysis in testing
  - Statement Coverage
  - Branch coverage
  - Path coverage
- Discuss the concept of program complexity

Structural Testing

- Sometime called white-box testing
- Derivation of test cases according to program structure
  - Knowledge of the program is used to identify test cases
- Objective is to exercise a certain percentage of statements, branches, or condition (not all path combinations)
  - Why??

White-box Testing

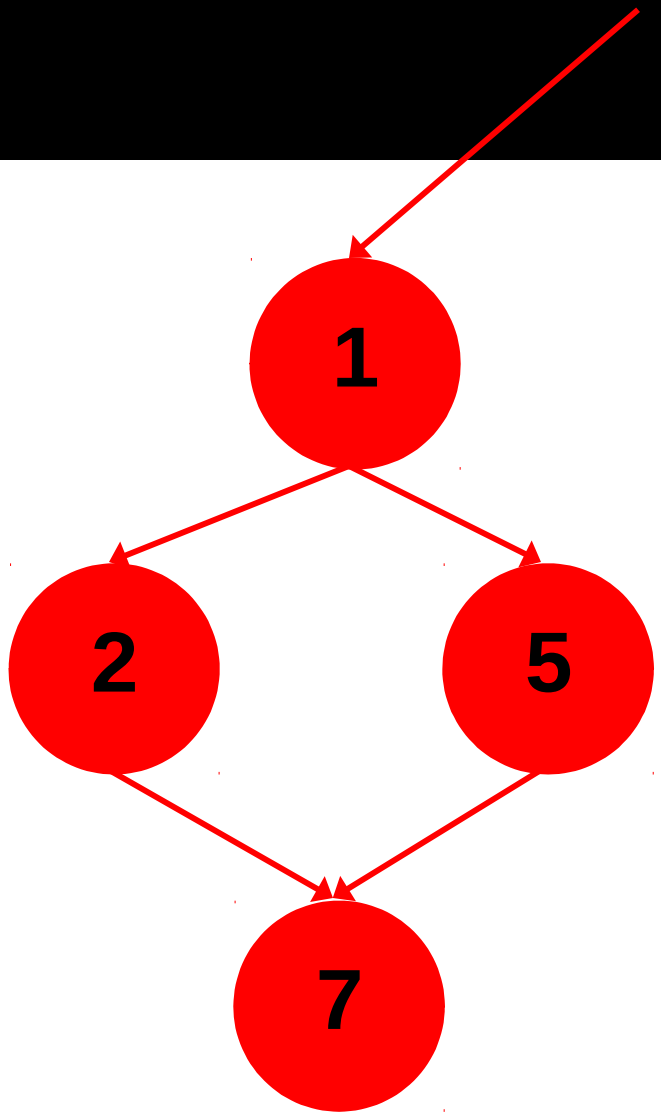


Program Flow Graphs

- Describes the program control flow
- Used as a basis for test data selection
- Used as a basis for computing the cyclomatic complexity
- Complexity = Number of edges - Number of nodes +1
  - Number of decision points + 1
  - N way branch counts as N-1 decision points

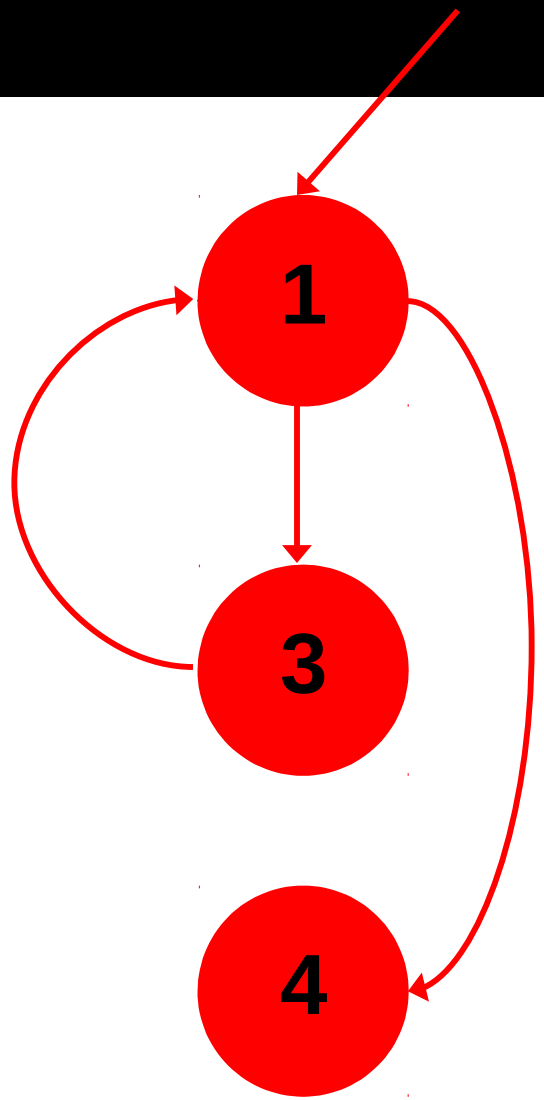
If-then-else

```
1 if (1==x) {
2 y=45;
3 }
4  else {
5 y=23456;
6 }
7 /* continue */
```



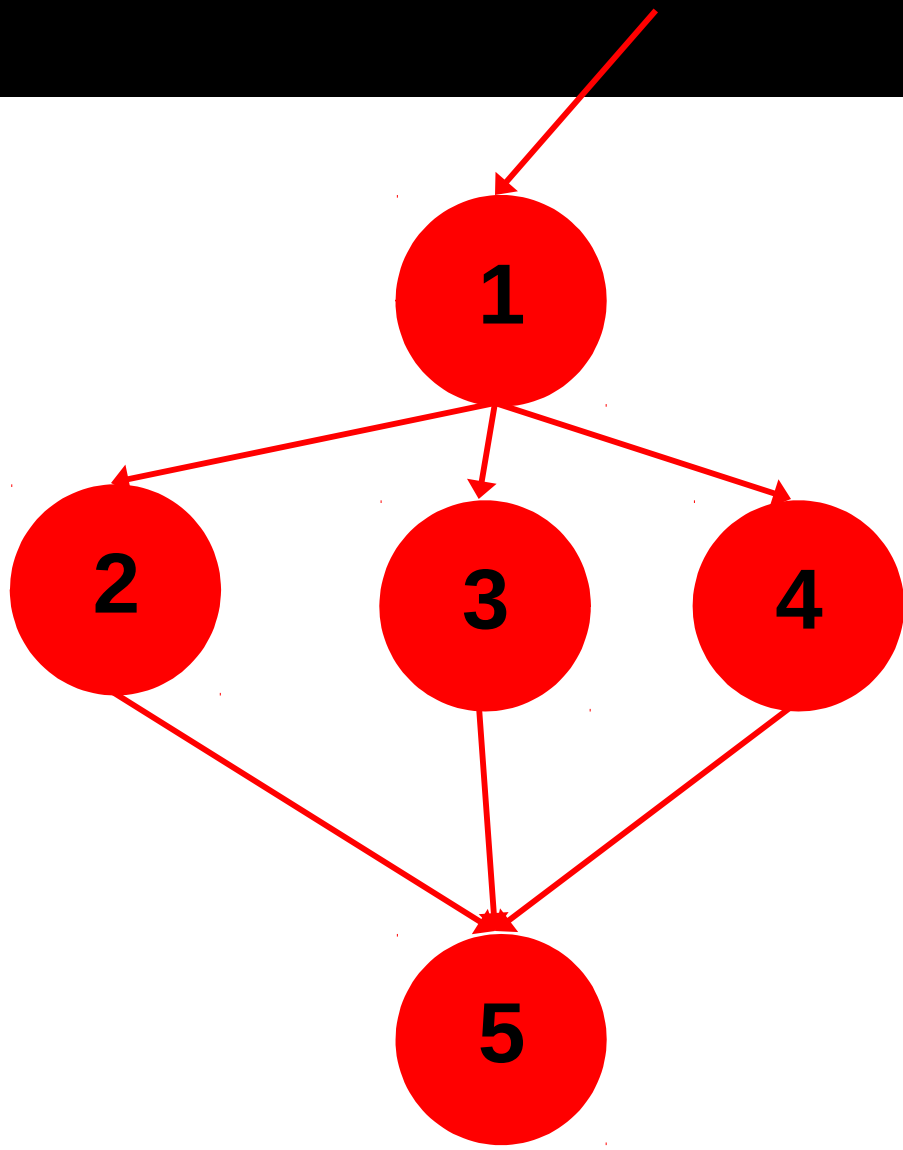
Loop

```
1 while (1<x) {
2 x--;
3 }
4 /* continue */
```



Case

```
1 switch (test) {
2 case 1 : ...
3 case 2 : ...
4   case 3 : ...
5 }
6 /* continue */
```

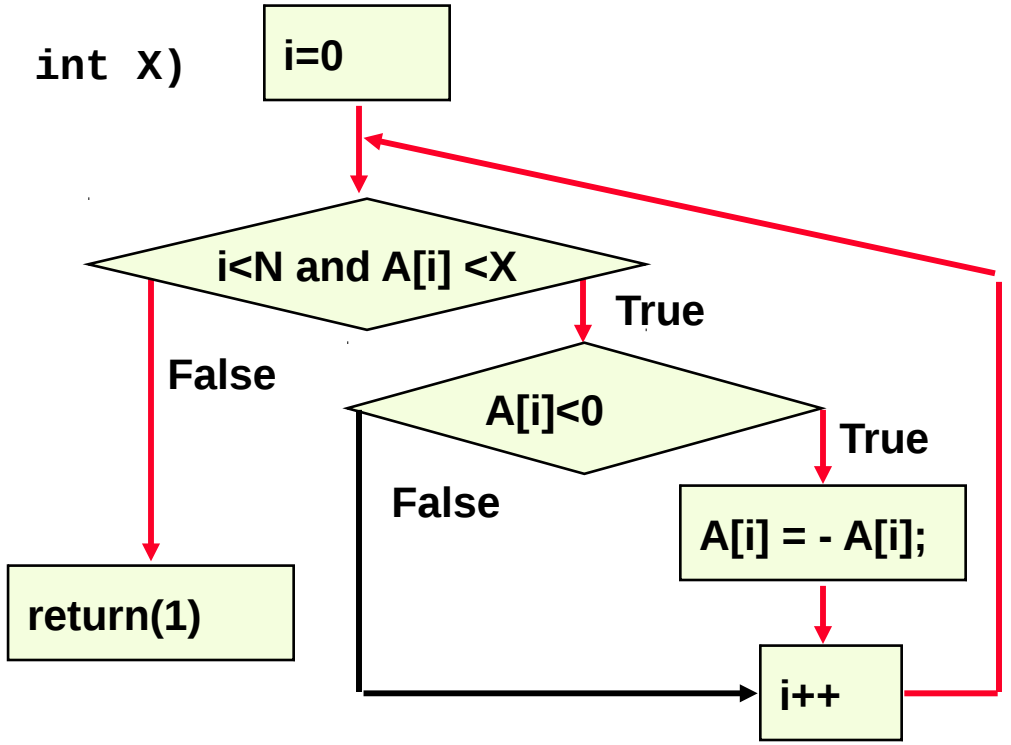


Structural Coverage Testing

- (In)adequacy criteria
  - If significant parts of program structure are not tested, testing is surely inadequate
- Control flow coverage criteria
  - Statement (node, basic block) coverage
  - Branch (edge) coverage
  - Condition coverage
  - Path coverage
  - Data flow (syntactic dependency) coverage
- Attempted compromise between the impossible and the inadequate

Statement Coverage

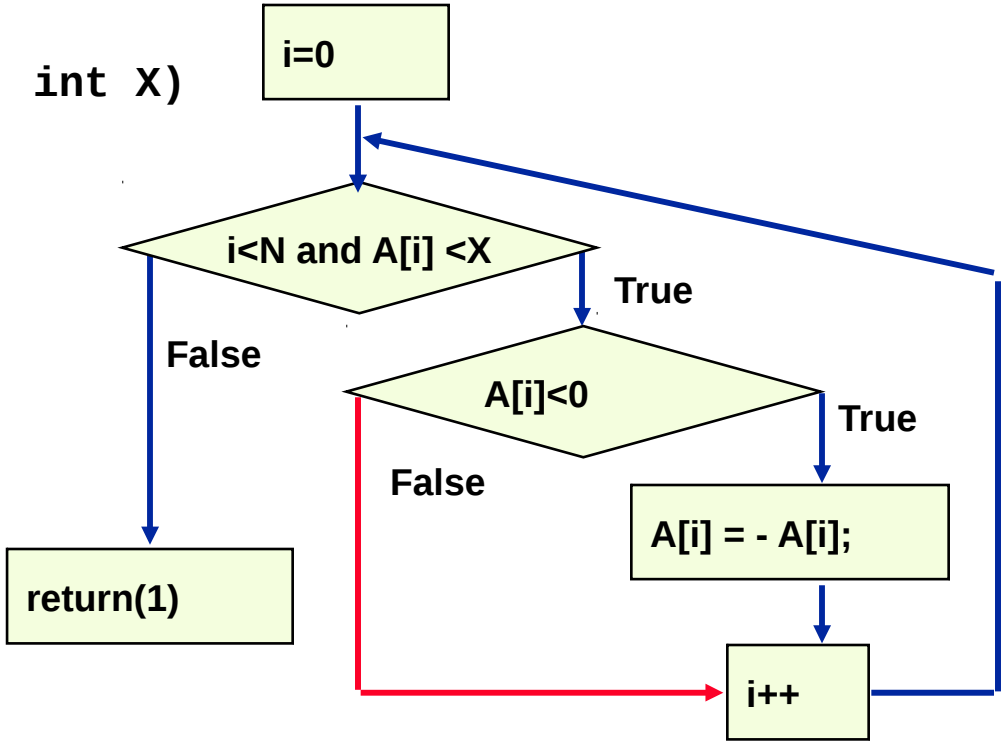
```
int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
     if (A[i]<0)
        A[i] = - A[i];
     i++;
  }
  return(1);
}
```



One test datum (N=1, A[0]=-7, X=9) is enough to guarantee statement coverage of function select
Faults in handling positive values of A[i] would not be revealed

Branch Coverage

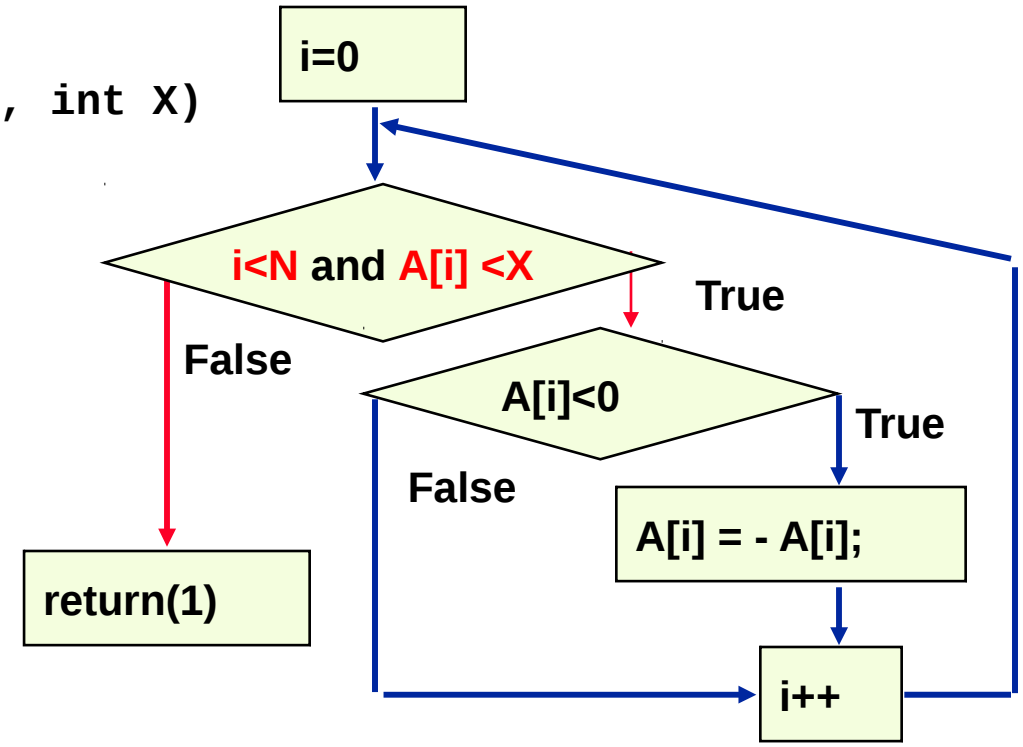
```
int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
     if (A[i]<0)
        A[i] = - A[i];
     i++;
  }
  return(1);
}
```



We must add a test datum (N=1, A[0]=7, X=9) to cover branch False of the if statement. Faults in handling positive values of A[i] would be revealed. Faults in exiting the loop with condition A[i] <X would not be revealed

Condition Coverage

```
int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
     if (A[i]<0)
        A[i] = - A[i];
     i++;
  }
  return(1);
}
```



Both conditions (i<N), (A[i]<X) must be false and true for different tests. In this case, we must add tests that cause the while loop to exit for a value greater than X. Faults that arise after several iterations of the loop would not be revealed.

Basic Condition Coverage

- Make each condition both True and False

Test Case	Cond 1	Cond 2
1	True	False
2	False	True

Compound Condition Coverage

- Evaluate every combination of the conditions

Test Case	Cond 1	Cond 2
1	True	True
2	True	False
3	False	True
4	False	False

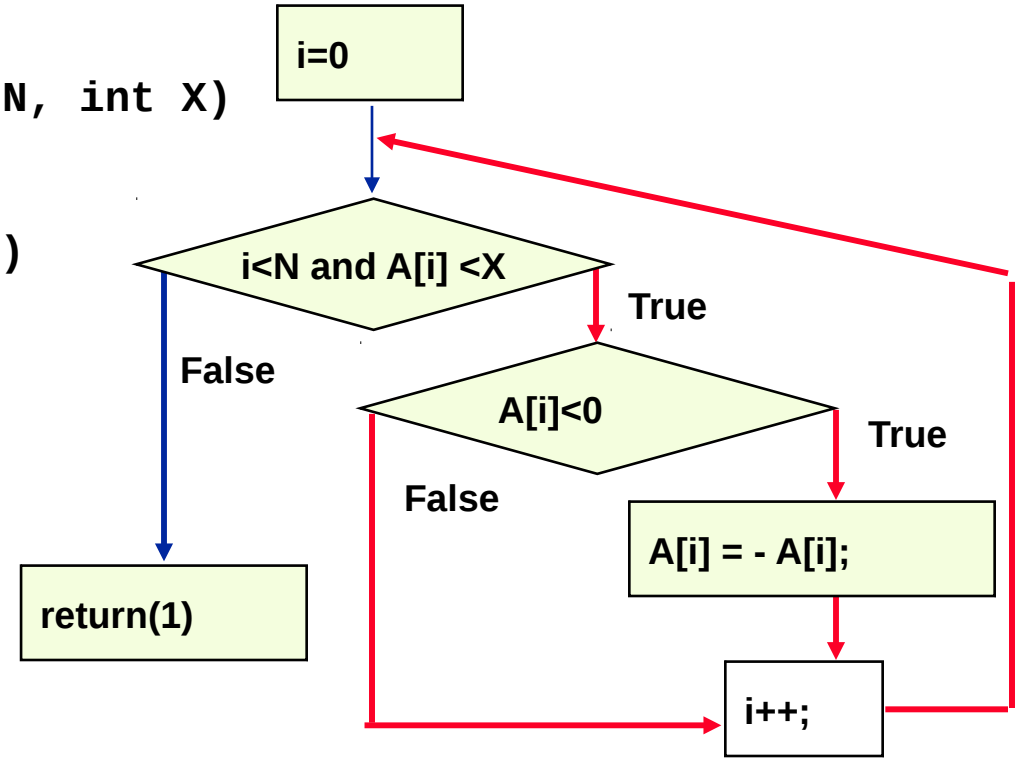
Compound Coverage (cont.)

- May lead to a lot of test cases

Test Case	Cond 1	Cond 2	Cond 3	Cond 4
1	True	False	False	False
2	True	True	False	False
3	True	True	True	False
4	True	True	True	True
5	False	True	False	False
6	False	True	True	False
7	False	True	True	True
8	False	False	True	False
9	False	False	True	True
10	False	False	False	True
11	True	False	True	False
12	True	False	True	True
13	True	False	False	True
14	False	True	True	False
15	False	True	False	False
16	False	False	True	False

Path Coverage

```
int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
     if (A[i]<0)
        A[i] = - A[i];
     i++;
  }
  return(1);
}
```

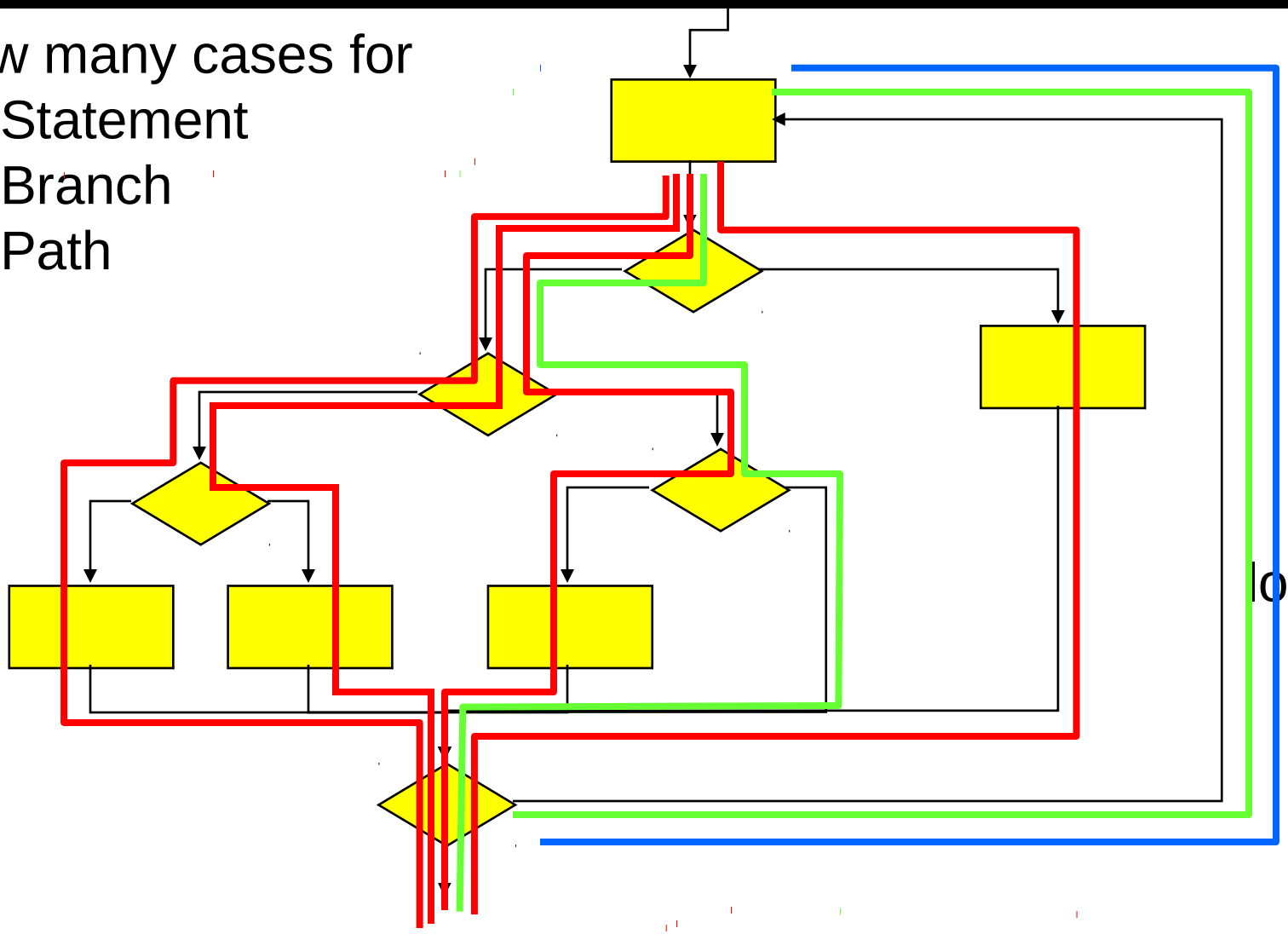


The loop must be iterated given number of times.
PROBLEM: uncontrolled growth of test sets. We need to select a significant subset of test cases.

Path Testing

How many cases for

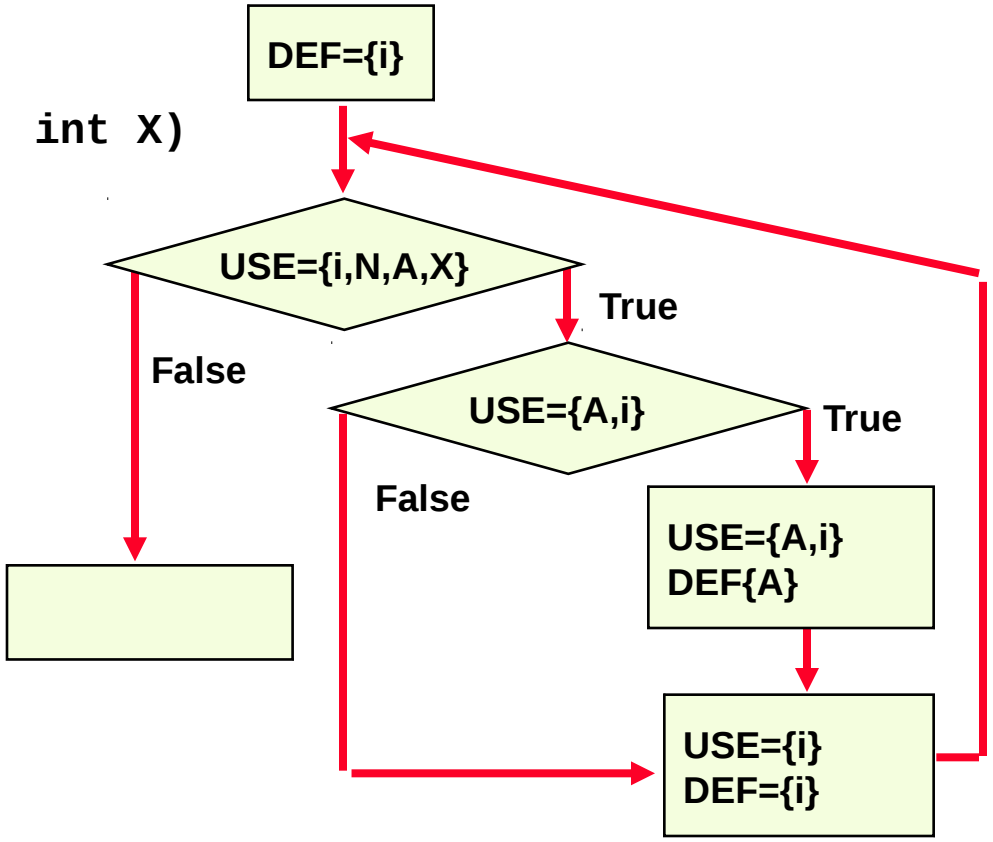
- Statement
- Branch
- Path



loop <= 20

Data Flow Coverage

```
int select(int A[], int N, int X)
{
  int i=0;
  while (i<N and A[i] <X)
  {
     if (A[i]<0)
        A[i] = - A[i];
     i++;
  }
  return(1);
}
```



Exercise Def-Use paths: selects paths based on effects on the variables, rather than number of iteration of loops

The Budget Coverage Criterion

- Industry's answer to "when is testing done"
  - When the money is used up
  - When the deadline is reached
- This is sometimes a rational approach!
  - Implication 1:
  - Adequacy criteria answer the wrong question. Selection is more important.
  - Implication 2:
  - Practical comparison of approaches must consider the cost of test case selection

Challenges in Structural Coverage

- Interprocedural and gross-level coverage
  - E.g., interprocedural data flow, call-graph coverage
- Regression testing
- Late binding (OO programming languages)
  - Coverage of actual and apparent polymorphism
- Fundamental challenge: infeasible behaviors
  - Underlies problems in inter-procedural and polymorphic coverage, as well as obstacles to adoption of more sophisticated coverage criteria and dependence analysis

The Infeasibility Problem

- Syntactically indicated behaviors (paths, data flows, etc.) are often impossible
  - Infeasible control flow, data flow, and data states
- Adequacy criteria are typically impossible to satisfy
- Unsatisfactory approaches:
  - Manual justification for omitting each impossible test case (esp. for more demanding criteria)
  - Adequacy “scores” based on coverage
  - example: 95% statement coverage, 80% def-use coverage

We Have Learned

- Test Coverage Measures
  - Statement, branch, and path coverage
  - Condition coverage (basic, compound)
  - Data flow coverage
- Test coverage measures ensure that statements have been executed to some level
  - However, it is not possible to exercise all combinations