White box testing

derives test cases from program code

Structural Coverage Testing

(In)adequacy criteria
- If significant parts of program structure are not tested, testing is inadequate

Control flow coverage criteria
- Statement coverage
- Edge coverage
- Condition coverage
- Path coverage

Statement-coverage criterion

Select a test set T such that every elementary statement in P is executed at least once by some d in T
- an input datum executes many statements → try to minimize the number of test cases still preserving the desired coverage

Example

```
read (x); read (y);
if x > 0 then
        write ("1");
else
        write ("2");
end if;
if y > 0 then
        write ("3");
else
        write ("4");
end if;
```

{<x = 2, y = 3>, <x = - 13, y = 51>, <x = 97, y = 17>, <x = - 1, y = - 1>} covers all statements

{<x = - 13, y = 51>, <x = 2, y = - 3>} is minimal

Weakness of the criterion

```
if x < 0 then
        x := -x;
end if;
z := x;
```

{<x=-3} covers all statements

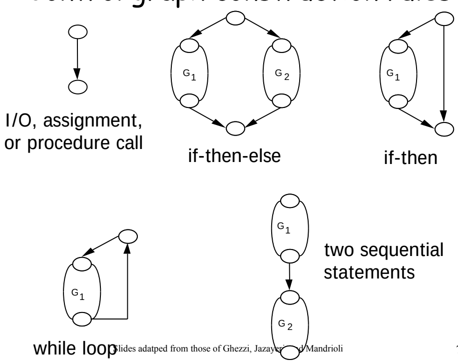
it does not exercise the case when x is positive and the then branch is not entered

Edge-coverage criterion

Select a test set T such that every edge (branch) of the control flow is exercised at least once by some d in T

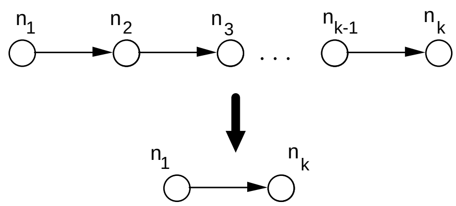
this requires formalizing the concept of the control graph, and how to construct it
- edges represent statements
- nodes at the ends of an edge represent entry into the statement and exit

Control graph construction rules



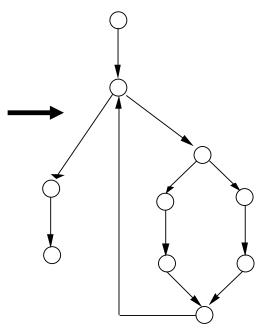
Simplification

a sequence of edges can be collapsed into just one edge



Exemple: Euclid's algorithm

```
begin
        read (x); read (y);
        while x ≠ y loop
                if x > y then
                        x := x - y;
                else
                        y := y - x;
                end if;
        end loop;
        gcd : = x;
end;
```



Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
        if table (counter) = desired_element then
                found := true;
        end if;
        counter := counter + 1;
end loop;
if found then
        write ("the desired element is in the table");
else
        write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

do not discover error (< instead of ≤)

Condition-coverage criterion

Select a test set T such that every edge of P’s control flow is traversed and all possible values of the constituents of compound conditions are exercised at least once

- it is finer than edge coverage

Weakness

```
if x ≠ 0 then
        y := 5;
else
        z := z - x;
end if;
if z > 1 then
        z := z / x;
else
        z := 0;
end if;
```

{<x = 0, z = 1>, <x = 1, z = 3>} causes the execution of all edges, but fails to expose the risk of a division by zero

Path-coverage criterion

Select a test set T which traverses all paths from the initial to the final node of P's control flow

- it is finer than previous kinds of coverage
- however, number of paths may be too large, or even infinite (see while loops)
  - additional constraints must be provided

The infeasibility problem

Syntactically indicated behaviors (statements, edges, etc.) are often impossible

- unreachable code, infeasible edges, paths, etc.

Adequacy criteria may be impossible to satisfy

- manual justification for omitting each impossible test case
- adequacy "scores" based on coverage
  - example: 95% statement coverage

Further problem

What if the code omits the implementation of some part of the specification?

White box test cases derived from the code will ignore that part of the specification!