# Case Studies for BB and WB Testing

Week 11

# First Case:

Insertion Sort

# Code to test: Insertion Sort

```
void insertionSort()
{
        int i, j, v;
        int smallest;           // boolean variable
        for (i=2; i<=n; i++) {                      // condition C1
            v=a[i]; j=i; smallest=0;
            while (a[j-1] > v && !smallest) {       // condition C2
                a[j] =a[j-1]; j=j-1;
                if (j <= 1)                         // condition C3
                    smallest=1;
            }
                a[j]=v;
        }
}
```

# Test by: BB Testing

- Apply …
- ***category-partition***
- method !!!

# Steps of Category Partition

i. Specify input *categories*

ii. Divide categories into *equivalence classes*

iii. Determine test cases

iv. Generate test cases for the test frames into executable form (using a tool), combination into *test suites*.

v. Store the testware into a *test database.*

vi. Apply test!...

  i. *Test the unit by the test cases determined & generated in (iii)&(iv),*

  ii. *refine conflicting choices,*

  iii. *maintain test database (using a tool).*

# I. Specify Input Categories

- ***Input Categories***
  - A. *Size of array*
  - B. *Types of elements*
  - C. *Max. element value*
  - D. *Min. element value*
  - E. *Position of max. element*
  - F. *Position of min. element*

# II. Divide Categories into ECs

## A. *Size of array*

- a negatif value, 0, 1, 3, 101, 100000 ***(6 cases)***

## B. *Types of elements*

- integer, non-integer (float, char, invalid) ***(4 cases)***

## C. *Max. and Min. element value*

- large neg., -1, typical, 1, large pos. ***(5 cases each)***

## D. *Position of Max. and Min. element*

- *at start, in the middle, at end, out of bounds* ***(4 cases each)***

# Questions for further study…

Number of exhaustive combinatory test cases?

6*4*4*5*5*4*4 = 9600 cases (too many test cases)

What you should do in case the number of exhaustive combinatory test cases is infeasible!

Use ***optimizing*** (i.e., cover each NV class by a test case, and each other equivalence class by at least one test case) and ***extending*** (i.e., cover each parameter in a test case, in the set of pairs, triplets, etc) principles

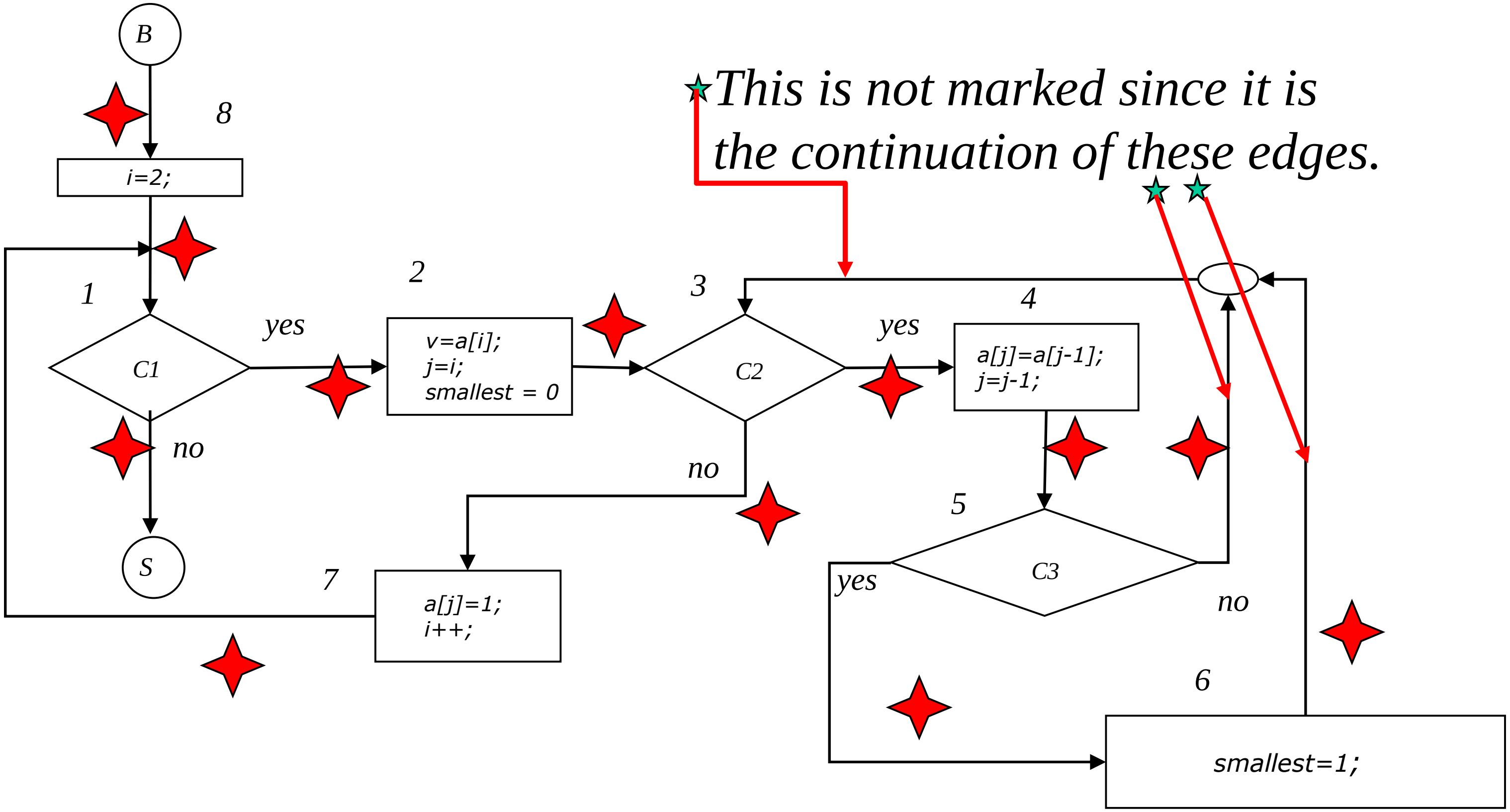
Two test case examples?

(101, integer, -5820, 7101, 2, 47);     (3, float, 0, 100, 2, 3);

# Test by: WB Testing

1. *Draw* *control flow graph* of the unit …
2. *Compute* *cyclomatic complexity* (CC)
3. *Determine* *independent paths* (IPs)
4. *Decide* on the *coverage* (statement, branch, condition, multi-condition or path)
5. *Prepare* the *test cases* regarding *coverage*

# Control Flow Graph



This is not marked since it is the continuation of these edges.

*Red stars symbolize the connections encountered for while computing the cyclomatic complexity. B and S nodes are not numbered but are included in the calculation of CC. (i.e., #nodes=8+2=10)*

# Cyclomatic Complexity

$$CC = \underbrace{12}_{\textit{\# of edges}} - \underbrace{10}_{\textit{\# of nodes}} + 2 = 4$$

# ***Independent Paths***

*8 1*

*8 1 2 3 7 1*

*8 1 2 3 4 5 3 7 1*

*8 1 2 3 4 5 6 3 7 1*

# Second Case:

Topological Sort

# Code to test: Topological Sort

```
Void Toposort ()
{
  Queue Q; int ctr=0; Vertex v,w;
  Q=createQueue(NumVertex);                                  (1)
  for each vertex v                                          (2)
          if (indegree[v] == 0) enqueue(v,Q);                (3)(4)
  while (!IsEmpty(Q)) {                                      (5)
          v=dequeue(Q); topnum[v]=++ctr;                     (6)
          for each w adjacent to v                           (7)
            if (--indegree[w] == 0) enqueue(w,Q);            (8)(9)
  }
  if (ctr != NumVertex) report error (‘graph cyclic!’)   (10)(11)
  free queue;                                                (12)

}
```

# I. Specify Input Categories

- ***Input Categories***

A. *Number of nodes in the graph*

B. *Number of edges*

C. *Types of connectedness*

D. *Type of graph (graph topologies)*

A.

# II. Divide Categories into ECs

## A. *Number of nodes, **n**, in the graph*

- 0, 1, 2, 3, 4, 5, 20, 50, 100, 1000, 10000 ***(10 cases)***

## B. *Number of edges (n: #nodes)*

- 0, 1, 3, n, $n^2/4$, $n^2/2$, $3n^2/4$, $n^2$-1, $n^2$ ***(10 cases)***

## C. *Types of connectedness*

- Not connected, sparsely connected,
- densely connected, fully connected ***(4 cases)***

## D. *Type of graph (graph topologies)*

- bus, ring, star, mesh and hybrid topologies ***(5 cases)***

# Questions for further study…

Number of exhaustive combinatory test cases?

10*10*4*5 = 2000 cases (too many test cases)

What you should do in case the number of exhaustive combinatory test cases is infeasible!

Use ***optimizing*** (i.e., cover each NV class by a test case, and each other equivalence class by at least one test case) and ***extending*** (i.e., cover each parameter in a test case, in the set of pairs, triplets, etc) principles

# Optimizing/Extending principles

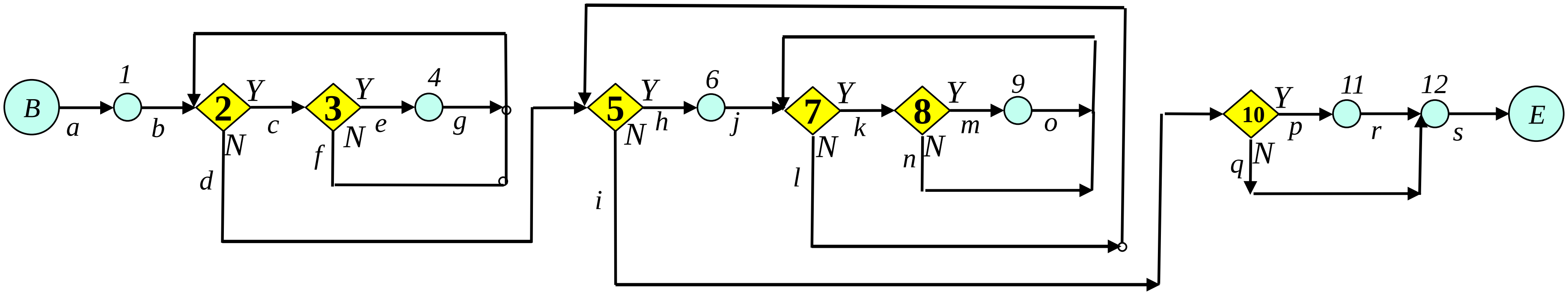
- *Optimizing principle*:
  - one test case for each *NV* equivalence class
  - each equivalence class covered by *at least one* test case

- *Extending principle*:
  - Add combinations over the *number* of parameters
    - Use name of existing file
    - a test case with all parameters missing (0 present)
    - a test case for each individual parameter (1 present)
    - each parameter included in the set of pairs (2 present)
    - each parameter included in the set of triplets (3 present)
    - all parameters given (4 present)

# Test by: WB Testing

- 1\. *Draw control flow graph* of the unit …
- 2\. *Compute cyclomatic complexity* (CC)
- 3\. *Determine independent paths* (IPs)
- 4\. *Decide* on the *coverage* (statement,
- branch, condition, multi-condition or
- path)
- 5\. *Prepare* the *test cases* regarding *coverage*

# Control Flow Graph



1) N=?

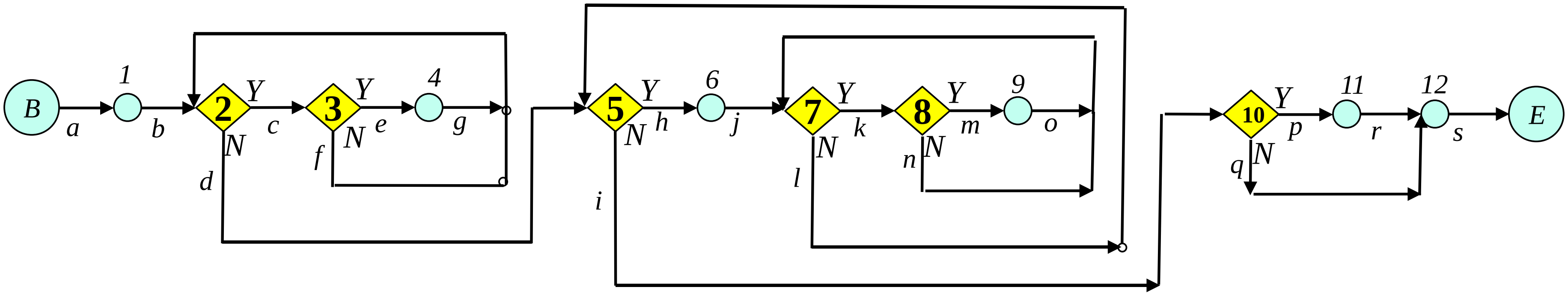
2) E=?

3) CC=?

4) Independent Paths (IPs)?

5) Sample dependent paths and their dependence on what IPs?

# Control Flow Graph



N=12+2=14; (12 nodes as marked and 2 from begin (B) and end (E) nodes)

E=|{a,…,s}|=19; CC=E-N+2=19-14+2=7;

Independent Paths:

1) 1 2 5 10 12

2) 1 2 5 10 11 12

3) 1 2 5 6 7 5 10 12

4) 1 2 5 6 7 8 7 5 10 12

5) 1 2 5 6 7 8 9 7 5 10 12

6) 1 2 3 2 5 10 12

7) 1 2 3 4 2 5 10 12

| Some dependent paths | dependent on... |
|---|---|
| 1) 1 2 5 6 7 5 10 11 12 | [2,3] |
| 2) 1 2 5 6 7 8 7 5 10 11 12 | [2,4] |
| 3) 1 2 5 6 7 8 9 7 5 10 11 12 | [2,5] |
| 4) 1 2 3 4 2 5 6 7 8 9 7 5 10 11 12 | [2,5,7] |

Ex: the order you traverse the IPs to obtain path 4 is:

Start: ***(7)*** (after N5) → (at N6) ***(5)*** (after N10) → (at N11) ***(5)*** End.

# Exercises!!!

1. ***Prepare test cases observing*** the *optimizing/extending principles*!!! *(BB)*
2. ***Draw the control flow graph and find CC!!! (WB)***