Week3-testing

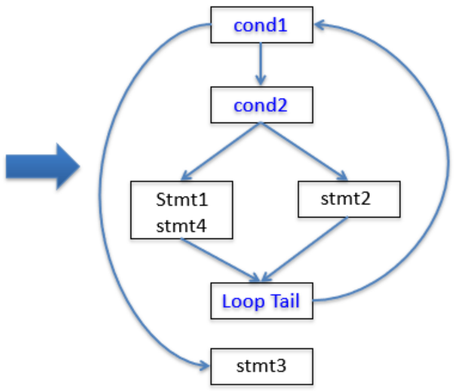
Black Box Testing

- invariant: a condition that is always true
- low-level specification: test a certain calculation function
- high-level specification: complex test (error url; UI operation)
- Black box test cosideration: run the simplest test first (exp: choose the simplest string when testing user case function) ...
- Draw state diagram: draw the most normal test first
- Test Design Policy / How to design a test
    - Every use case should be covered (picture)
    - more likely many tests for a single use case(use case is complex)
- equivalence class partitioning: 所有预期将表现得“类似“的数据 (exp: all valid email addresses for a email address validity function)
    - exp: sorting-random array; sorted array; exactly in the reverse oder; with duplicates(quite important: <= or <); with negatives
    - reverse a list: 回文式(p开头的 派林壮), empty list, a list with only one element
    - boundayr analysis/corner case: the case that lies on the boundary between two expected cases (exp: su@@sutd.edu.sg)
- *BlackBox/Fuctional Test Design
    - Check use case Diagrams. For every use case, there must be at least one test.
    - Evey Test should relate to some use case
    - For evey use case, find input space for the respective tests and perform equivalence class partitioning.
    - For each equivalence class, find middle and boundary value
- BlackBox testing: only useful to find whether the software could perform a certain function. (Will discuss more in week 10)

White Box Testing

- Control Flow Graph - for advanced testing
- Method Coverage: methods, tests
    - For every method, there is at least one test - Method Coverage: 100%
    - Method Coverage: tested methods/ all methods
    - The difference with the use case is that all the use cases must be covered. So black box test is more strict
    - method == function (exp: in the example given, any test with taht function will give a 100% method coveage)

```
while (cond1)  {
      if (cond2)
        //stmt1
        //stmt4
      else
        //stmt2
}
//stmt3
```



Every Branch

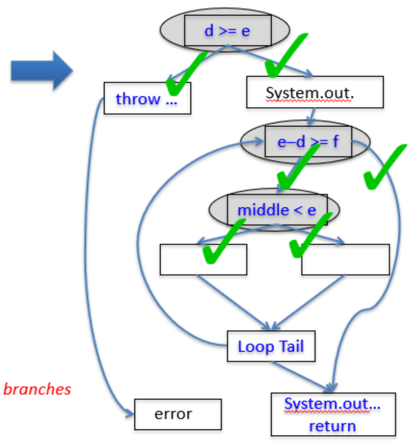
`BiSectionExample.root()`

A test where d >= e
A test where d < e

A test where e-d >= f
A test where e-d < f

A test where middle < e
A test where middle >= e

Note: A single test may cover multiple branches

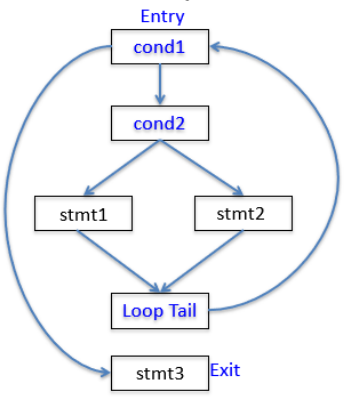


Every Condition

- For each condition, there must be one test case which satisfies it and one which dissatisfies it.
- Question: how many test cases we need?
  - if (A && B)
    - {A = true, B = false}, {A = false, B = true}
  - if ((j>=0) && salary[j] > 10000)
    - ?

Every Path

- A path is defined as a sequence of **executed** nodes in the control flow graph between the entry node of the graph and the exit node



cond1->cond2->stmt1->loop tail->cond1->stmt3 is a path

cond1->cond2->stmt1->loop tail->cond1->cond2->stmt2->loop tail->cond1->stmt3 is also a path

cond1->cond2->stmt1->stmt2->loop tail->cond1->stmt3 is **not** a path

How many paths in total?
(assuming the loop is executed exactly 100 times)