**Black Box Testing:**

Write test inputs for black-box test input partitioning and boundary value analysis for a given specification.

Input Partitioning:

For each input, based on the specification, divide the input space into valid and invalid partitions:

* If input in a range, then need at least one valid in the range and two invalid outside the low end and high end of range
* If input represents an allowed number of things, then one in the range, and one outside such as zero or more than the max
* If input is a set of values and each is handled differently, then one valid for each one and one invalid
* If input is a *must-be* condition, then the valid is easy and have one invalid.

Key: must define invalid partitions as well as valid for each input.

Boundary Value Analysis:

Boundaries on the edges of equivalence classes often reveal defects:

* Select tests so that each edge of each partition, or boundary, is subject to test
* Rather than only focus on input, create tests that cover output equivalence classes

No easy formula or recipe to follow.

Class Standards:

* The specification should make clear the requires and ensures for correct output
* The ensures clauses must be checked in any test
* Invalid input that violates any requires clause should be tested and checked that it throws a Java Runtime exception: use Assertions.assertThrows.
* Ensures only includes something about exceptions if there is a throws-clause with a non-runtime exception; otherwise any exception is implicit, runtime, and checked as part of the invalid input from the equivalence partitioning.
* Pick a naming convention for the tests and be consistent with it!

**White Box Testing:**

**Condition**: a leaf-level Boolean expression containing no Boolean operators (it cannot be broken down into simpler Boolean expressions)---a proposition.

Branch Coverage

Write test inputs for white-box test branch coverage for a given Java method.

**Decision** or **Branch**: a Boolean expression composed of conditions and zero or more Boolean operators. A decision without a Boolean operator is a condition. If a condition appears more than once in a decision, each occurrence is a distinct condition.

Statement Coverage

Write test inputs for white-box statement coverage for a given Java method.

**Statement coverage**: each statement is executed at least once by some test (e.g., each node in the control flow graph is seen). Minimum test standard for white-box.

MC/DC?

**Control Flow Graph:**

Draw a control flow graph for a given Java method. Please do add an entry and exit node to the graph.

**Dominance tree:**

Compute the dominance tree from the control flow graph in the previous problem.

**Mutation Analysis:**

Add additional test inputs to, or modify the method in, the previous problem to cover a given set of mutations.

**Specifications in Dafny with loop invariant and decreases:**

Write a formal specification for a program suitable for Dafny to use in a proof of total correctness. The specification needs to include a loop invariant and decreases clause for termination

**Weakest Precondition Calculus**

Basic Rules:

If statements:

Prove that a given Dafny program with if-statements satisfies its specification using the weakest-precondition calculus

While loops:

Total Correctness:

Prove that a given Dafny program with a while-loop satisfies its specification using the weakest-precondition calculus.