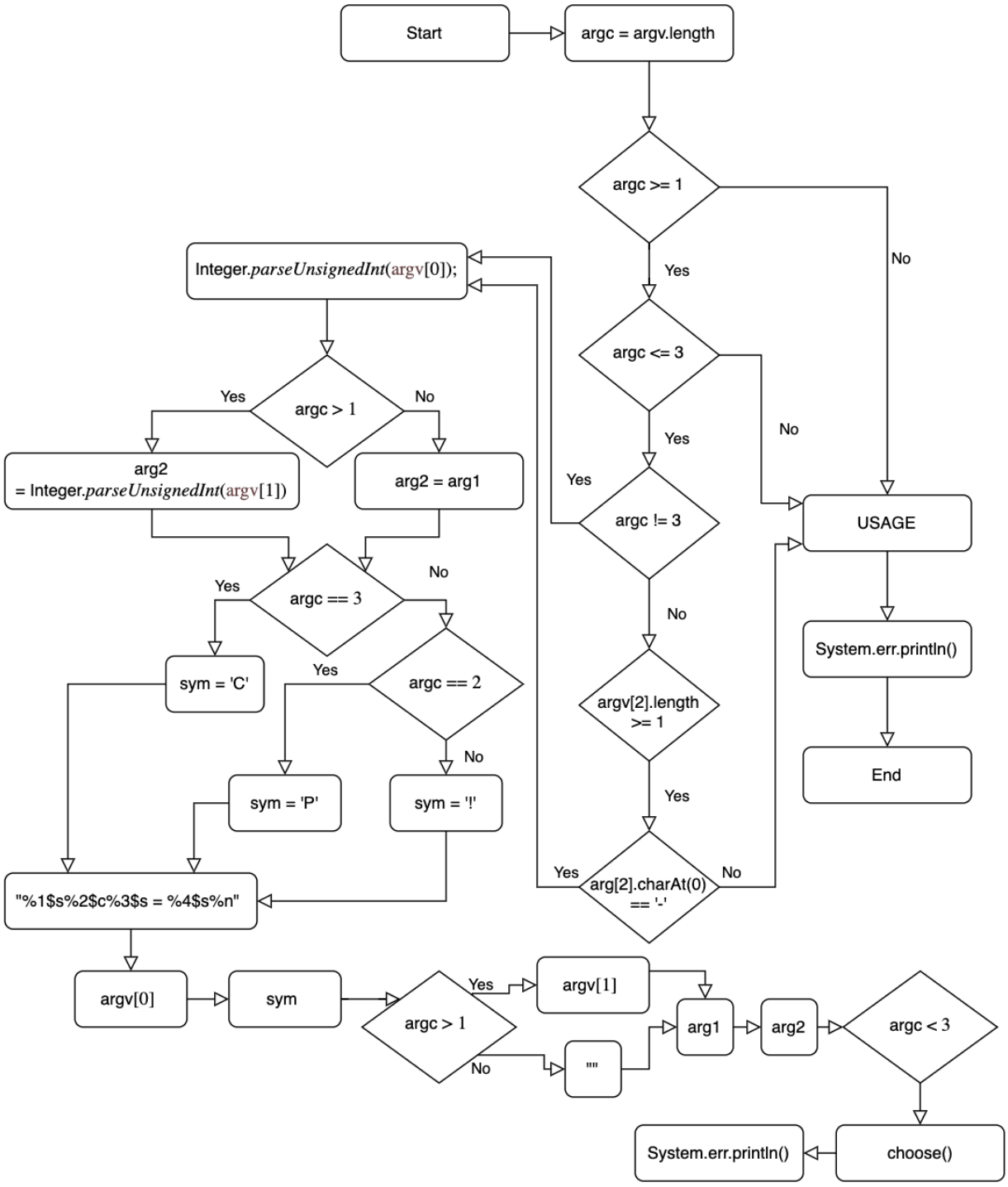
Moti Begna
SENG 5811
3/17/2023

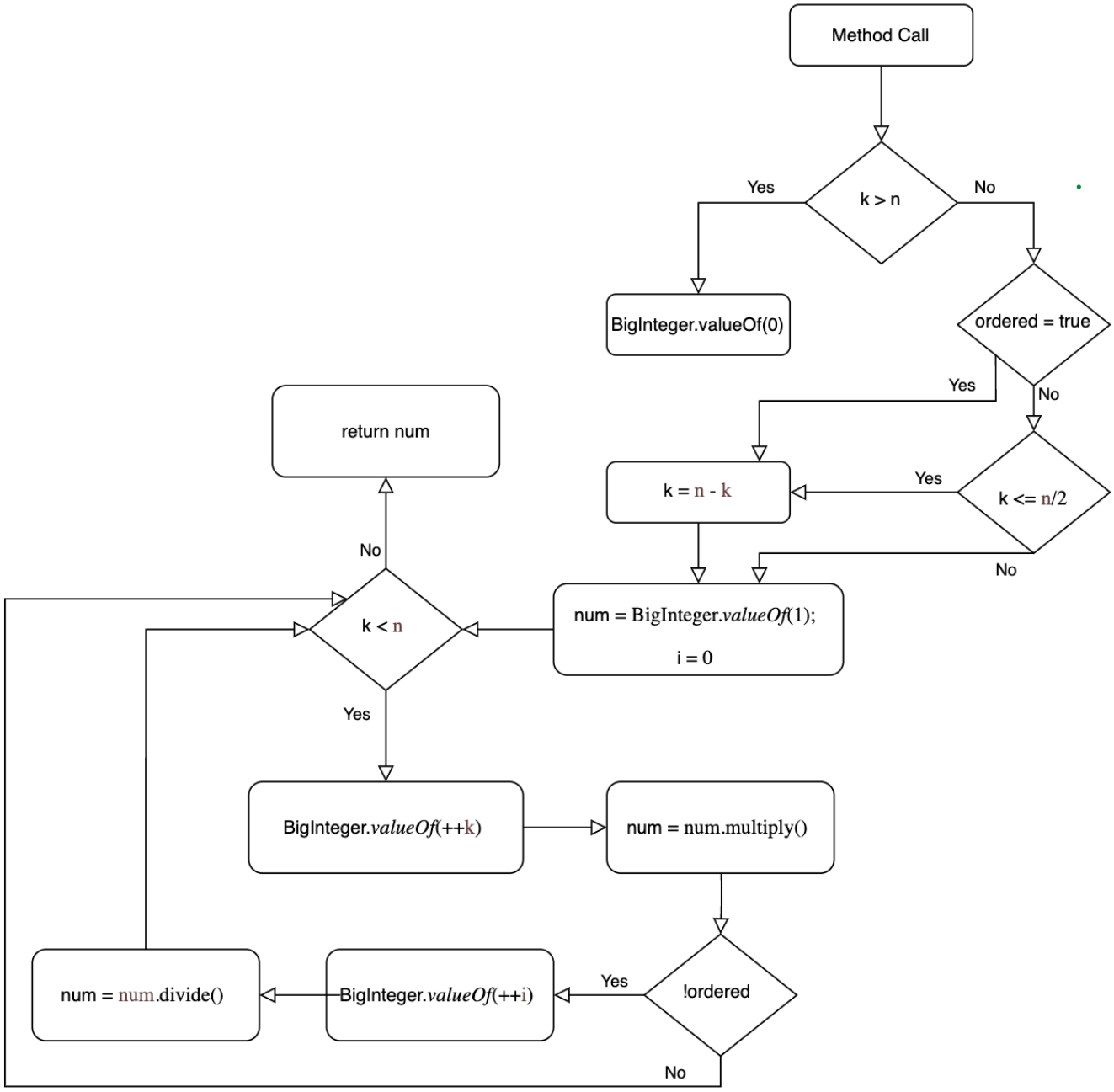
Homework 4

- **Problem 1**
  - **a.** Original: $(((a + b) < c) \land \neg p) \lor (r > s)$
    Mutants:
    1. $(((a + b) > c) \land \neg p) \lor (r > s)$ Incorrect Relational Operator
    2. $(\neg ((a + b) < c) \land \neg p) \lor (r > s)$ Incorrect Negation Operator
    3. $(((a + b) < c) \land \neg p) \land (r > s)$ Incorrect Boolean Operator
    4. $(((a + b) < c) \land \neg p) \lor (r < s)$ Incorrect Boolean Operator
  - **b.** Test 1 kills Mutation 1 (Expected: True, Actual False)
    Test 1 kills Mutation 2 (Expected: True, Actual False)
    Test 1 kills Mutation 3 (Expected: True, Actual False)
    Test 2 kills Mutation 1 (Expected: False, Actual True)
    Test 2 kills Mutation 2 (Expected: False, Actual True)
    Test 3 kills Mutation 4 (Expected: False, Actual True)
    Test 4 kills Mutation 3 (Expected: True, Actual False)
    Test 4 kills Mutation 4 (Expected: True, Actual False)
  - **c.** In order to prove that all test cases in a test suite are necessary to kill all possible mutations, you would need to first list all of the mutants (both singular and a combination of faults where 2 or more variations are introduced) and then show that for each test, there exists at least one mutation that is only killed by that test. For example, if Test 1 kills Mutation 1, Test 2 kills Mutation 2, and both Test 1 and Test 2 kill Mutation 3, even though either test can be used to kill Mutation 3, those tests are still needed to kill the other 2 mutations.
  - **d.** Because all the Mutations listed above are killed by at least 1 test case, all the tests in my test suite would not be considered necessary for killing these mutants. For example, Mutations 1, 2, and 3 are killed by Test 1 and Mutation 4 is killed by Test 3 thus Tests 2 and 4 would not be necessary.
  - **e.** In order to prove that my test suite is sufficient to kill all possible mutants derived from the original statement, I would have to first list all of the mutants (both singular and a combination of faults where 2 or more variations are introduced) and then show that for each mutant *at least* 1 test is able to create a different outcome than the original statement would have.
  - **f.** From my definition above, since *at least* 1 test from my test suite is able to kill each mutant, then we are able to say that the test suite is sufficient for killing mutants.

- **Problem 2**
    - **a. Control Flow for main()**



Control Flow for choose()



b. Tests needed to exercise every node in both graphs at least once: 4
Tests needed to exercise every node in both graphs at least once: 7

In order for choose() to execute, main() must be passed arguments that will allow it to be called. Thus, in the first part we need to figure out the minimum number of test cases required for choose(), and we'll know that we need AT LEAST that many for the overall program. In the second part, we just need to figure out the minimum number needed to hit the other edges/nodes in main() that weren't already called in the first part. Thus, the overall all number of tests needed will be the two values from both parts added.

c. Minimal tests needed for maximal statement coverage
Test Condition: 0 inputs
Test Input:
Test Output: USAGE message

Test Condition: 1 input
Test Input: 4!
Test Output: 4! = 24

Test Condition: 2 inputs
Test Input: 8 5
Test Output: 8P5 = 6720

Test Condition: 3 inputs
Test Input: 10 2 -
Test Output: 10C2 = 45

The number of tests needed in order to achieve maximal statement coverage is equal to the number of tests needed to hit every node at least once. We were able to prove that with the above test case.

d. Minimal tests needed for maximal condition coverage
Test Condition: 0 inputs
Test Input:
Test Output: USAGE message

Test Condition: 1 input
Test Input: 4!
Test Output: 4! = 24

Test Condition: 2 inputs
Test Input: 8 5
Test Output: 8P5 = 6720

Test Condition: 3 inputs
Test Input: 10 2 -
Test Output: 10C2 = 45

Test Condition: 3 inputs
Test Input: 8 5 -
Test Output: 8C5 = 56

Test Condition: 4 inputs
Test Input: 8 5 - -
Test Output: USAGE MESSAGE

Test Condition: 3 inputs, 3rd input != '-'
Test Input: 8 5 !
Test Output: USAGE MESSAGE

One interesting thing to note is that on line 20, if (argc != 3) is false, then (argv[2].length >=1) can never be false since we know that there is 3 arguments so there wouldn't ever be a branch for it. This leaves us with only needing 7 tests that achieve maximal conditional coverage, which is equal to the number of tests needed to hit every edge at least once.

e. MC/DC for Line 20

(argc >= 1 && argc <= 3 && (argc != 3 || (argv[2].length() >= 1 && argv[2].charAt(0) == '-')))

- Let argc >= 1 be the atomic condition **a**
- Let argc <= 3 be the atomic condition **b**
- Let argc == 3 be the atomic condition **c**
- Let argv[2].length() >= 1be the atomic condition **d**
- Let argv[2].charAt(0) == '-' be the atomic condition **e**
- Thus, the compound condition with its atomic conditions is:

a && b && (¬c || (d && e))

Atomic Condition	**Test Specification**					
	Spec 1	**Spec 2**	**Spec 3**	**Spec 4**	**Spec 5**	**Spec 6**
a	T	F	-	T	T	T
b	T	-	F	T	T	T
c	F	-	-	T	T	T
d	~	-	-	F	T	-
e	~	-	-	-	T	F
Result	T	F	F	F	T	F

Result for "a" covered by: Spec 1, Spec 2

Result for “b” covered by: Spec 1, Spec 3
Result for “c” covered by: Spec 1, Spec 4
Result for “d” covered by: Spec 4, Spec 5
Result for “e” covered by: Spec 5, Spec 6

Spec: 1
Test Input: 8 5
Test Output: 8P5 = 6720

Spec: 2
Test Input:
Test Output: USAGE MESSAGE

Spec: 3
Test Input: 8 5 - -
Test Output: USAGE MESSAGE

Spec: 4
As mentioned before, if argc != 3 is false (i.e. argc == 3), then (argv[2].length >=1) can never be false since we know that there is 3 arguments. A test for this would not be possible.

Spec: 5
Test Input: 8 5 -
Test Output: 8C5 = 56

Spec: 5
Test Input: 8 5 !
Test Output: USAGE MESSAGE