ESOF 322
Homework 5

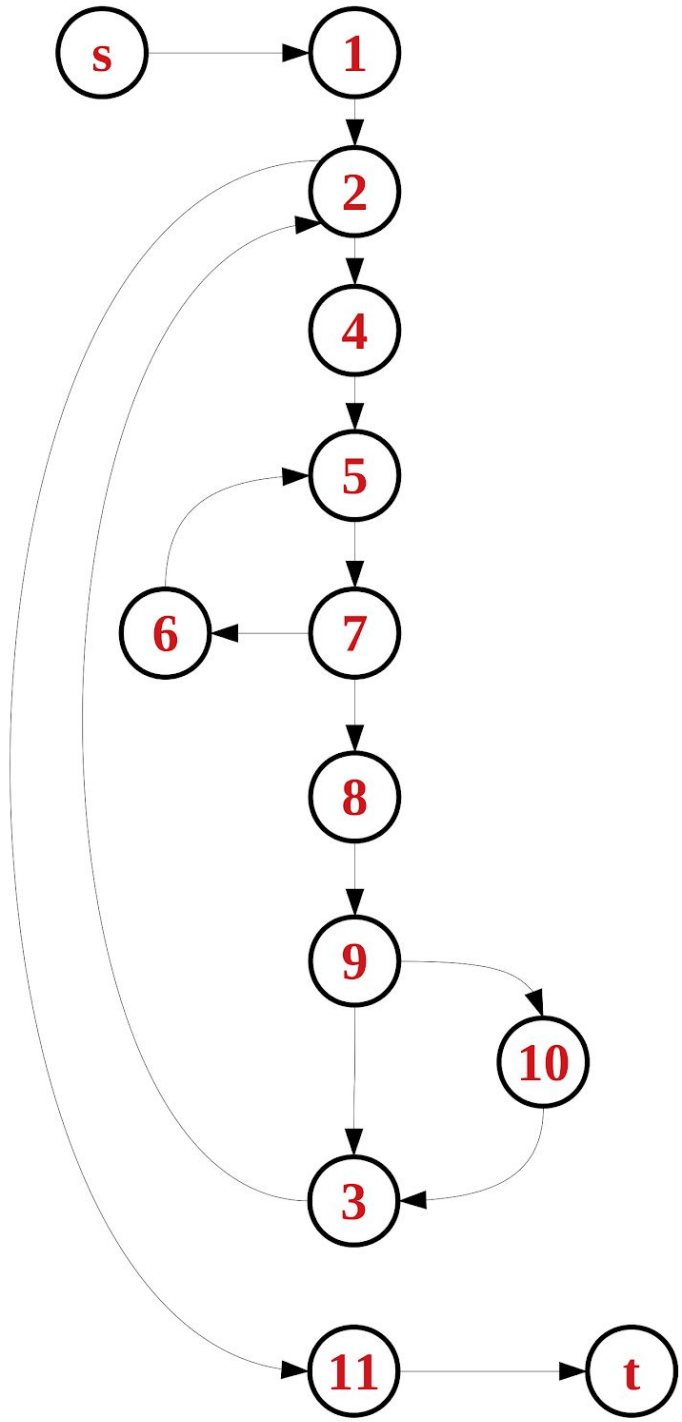
Thane Richard
Matteo Bjornsson

Question 1

a) Identify nodes on the source code and create a control flowgraph of the *sieve* algorithm.

```
1.  /* Find all primes from 2-upper_bound using Sieve of Eratosthanes */
2.
3.  #include
4.  typedef struct IntList {
5.            int value;
6.            struct IntList *next;
7.            } *INTLIST, INTCELL;
8.  INTLIST sieve ( int upper_bound ) {
1   9.
10          INTLIST prime_list = NULL;   /* list of primes found */
11          INTLIST cursor;              /* cursor into prime list */
12          int candidate;               /* a candidate prime number */
13          int is_prime;                /* flag: 1=prime, 0=not prime */
14
15           /* try all numbers up to upper_bound */
16          for (candidate=2;
17
18           2  candidate <= upper_bound;
19           3  candidate++) {
20
21        4 is_prime = 1; /* assume candidate is prime */
22          for(cursor = prime_list;
23
24            5 cursor;
25            6 cursor = cursor->next) {
26
27           7  if (candidate % cursor->value == 0) {
28
29                /* candidate divisible by prime */
30              8 /* in list, can't be prime */
31                is_prime = 0;
32                break;  /* "for cursor" loop */
33               }
34            }
35        9   if(is_prime) {
36
37              /* add candidate to front of list */
38              cursor = (INTLIST) malloc(sizeof(INTCELL));
39        10    cursor->value = candidate;
40              cursor->next  = prime_list;
41              prime_list = cursor;
42            }
43          }
44     11   return prime_list;
45        }
```

Flowgraph:



b) Provide a set of test cases that would give 100% Node Coverage.

The set of test cases below from 100% Edge Coverage would also provide 100% Node Coverage. Node Coverage is a subset of Edge Coverage, so any test case that provides Edge Coverage will also provide Node Coverage.

$$T = \{t_1 = \{1, 2, 4, 5, 7, 6, 5, 7, 8, 9, 10, 3, 2, 11\}\}$$

c) Provide a set of test cases that would give 100% Edge Coverage.

Providing 100% Edge Coverage for this program would need to provide a upper_bound that includes both prime and non-prime numbers before the upper_bound is reached. Therefore any test value greater than 5 would provide 100% Edge Coverage.

$$T = \{t_1 = \{1, 2, 4, 5, 7, 6, 5, 7, 8, 9, 10, 3, 2, 4, 5, 7, 8, 9, 3, 2, 11\}\}$$

d) Is 100% NC or EC possible in general?

Generally…. Yes, if all paths are feasible, all nodes reachable. This is not always the case. For example, the code

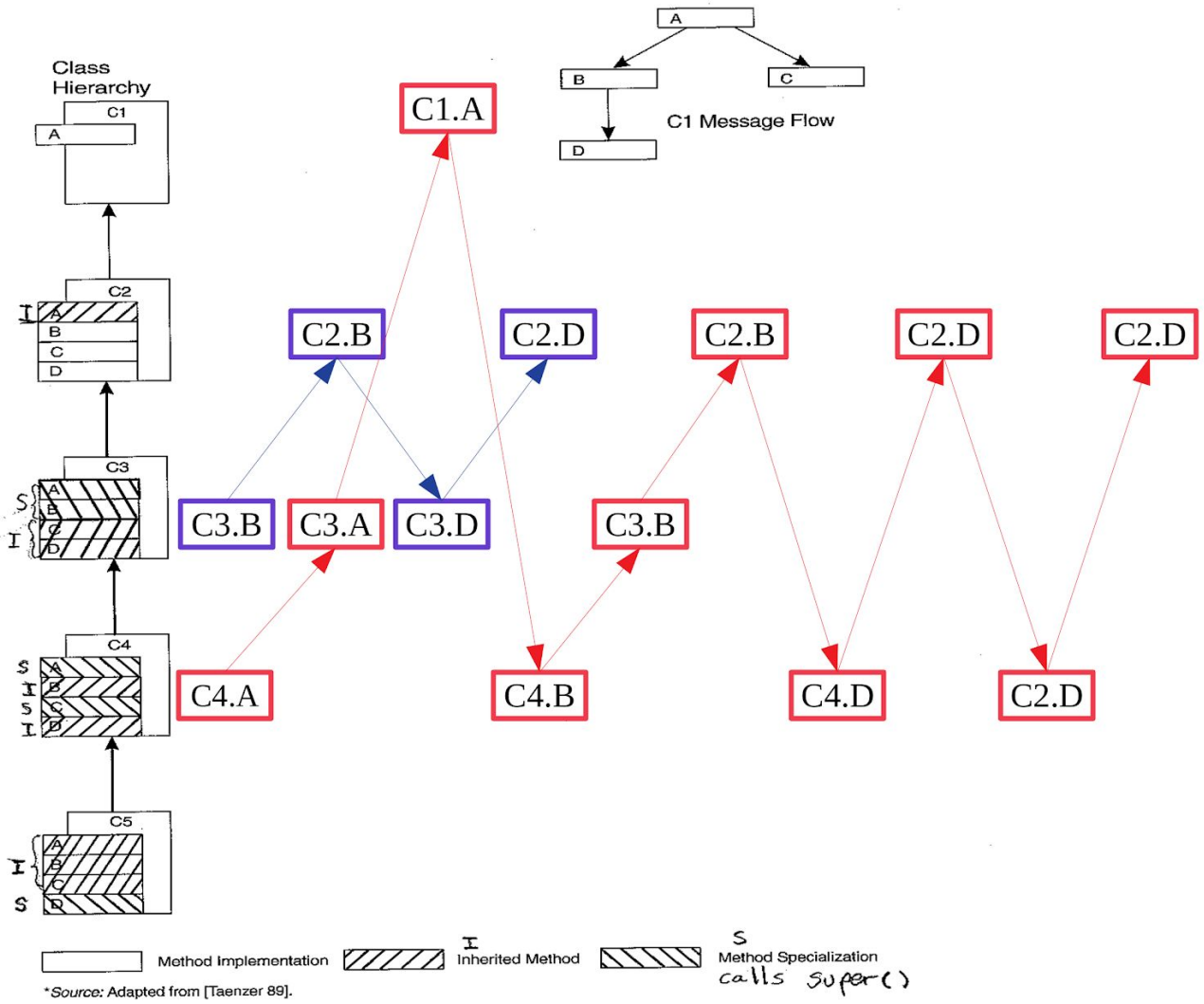
```
if (false) {
     unreachableMethod();
}
```

has a statement that is a node and would be a part of the control flowgraph, but is unreachable. No test data would be able to reach this code. If you are defining coverage by the test data set, then no, 100% EC is not always possible. If you define coverage by possible paths in the flowgraph, then yes, 100% edge and node coverage is generally possible.

As the size of software increases, the practical barrier to writing test cases that provide 100% Node Coverage and 100% Edge Coverage becomes increasingly difficult and potentially impossible. For example, creating a test case that provides 100% Edge Coverage for Google's entire codebase may be functionally possible, but practically impossible given the time and resources it would take.

Question 2

a) Draw the runtime trace of C3.B (blue) and C4.A (red)



b) Describe what happens when we call C1.D

This would return an error since C1 is the most generalized form of the C-types and method D is only defined in more specific classes that inherit from C1. It is not defined in C1. In order to call that method, the diagram would need to show an association between C1 and a subclass that contains (or inherits) D.

Question 3 Give test cases that will kill the following mutations:

a) Line 6: if (i < 1)

Test case i = 1 will kill this mutant.

b) Line 6: if (i == 1)

Test case i = 0 (or any i < 1) will kill this mutant.

c) Line 12: fib2 = fib;

Test case i = 3 (or any i > 2) will kill this mutant.

Summary of original program outputs and mutant outputs, illustrating the requisite test sets:

input	0	1	2	3	4
Fib	1	1	2	3	5
M1	1	0	2	3	5
M2	0	1	2	3	5
M3	1	1	2	4	8