Homework 2

To: All students in CSCI 5802 (Spring 2018)
CC: Teaching Assistant
From: Instructor
Date: 2/8/2018
Due: 2/19/2018, Monday, 11:55 PM, on Moodle.
Re: Homework Assignment 2 – Combinatorial Testing, Finite Models and Data-flow

---

There are 5 problems worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team, in PDF format, on Moodle.

1. 24 Points

Note: Briefly justify your answers to the questions in these exercises.

(a) Exercise 11.1 in the textbook.
(b) Exercise 11.3 in the textbook.

2. 25 Points

Exercise 11.5 in the textbook (application of catalog-based testing to the Airport-Connection-Check example):

1. Show the pre-conditions, post-conditions, definitions, variables and operations that you identified.
2. Show the initial set of test cases from the pre-conditions, post-conditions and definitions.
3. Show additional test cases derived using the catalog in Table 11.7.
4. Propose an extension to the catalog (a new catalog entry) and state your rationale for the extension (why do you think it is useful).
5. Show a test case in your test suite that corresponds to your proposed addition to the catalog.

You may find it useful to follow the example discussed in the textbook in section 11.4 for `cgi_decode` for answering this question.

3. 12 Points

Note: Briefly justify your answers to the questions in these exercises.

(c) Exercise 5.5 in textbook.
(d) Exercise 5.6 in textbook.

4. 15 Points

Consider the C program listing provided at the end of this document, which is supposedly an implementation of a weighted-round-robin (WRR) resource allocation algorithm. The program takes the number of clients (n) and their individual weights, $\{w_i \mid 1 \leq i \leq n\}$ as input and determines a weighted-round-robin schedule for allocating some shared resource to the clients. The output is a sequence of (repeated) client numbers (from the range 1….n) to be allocated in order, the shared resource for some unit time.

a. Show the control-flow graphs for the three procedures in the program.
b. Show the call graph for the program. Include the library functions in your call graph.
c. For the routine `next_elem` identify the LCSAJs.

5. 24 Points

In a directed graph with a designated exit node, we say that a node *m* post-dominates another node *n*, if *m* appears on every path from *n* to the exit node. Let us write *m pd n* to mean that *m* post-dominates *n*, and *pd(n)* to mean the set of all post-dominators of *n*, i.e., $\{m \mid m \; pd \; n\}$.

Answer the following providing justification for each:

a. Does *b pd b* hold true for all *b*?

b. Can both *a pd b* and *b pd a* hold true for two different nodes *a* and *b*?

c. If both *c pd b* and *b pd a* hold true, what can you say about the relation between *c* and *a*?

d. If both *c pd a* and *b pd a* hold true, what can you say about the relation between *c* and *b*?

e. How would you use the answer to your previous question to characterize the immediate post-dominator of a node other than the exit node?

f. Suppose the set of post-dominators *pd(n)* is known for every successor *n* of a node *m*, can we then derive *pd(m)*?

g. Cast the computation of *pd(m)* in terms of flow equations. What are the *gen* and *kill* sets for *m*? How would you classify the flow analysis: forward or backward, any-path or all-path?

h. Provide an iterative work-list algorithm (see Chapter 6 for examples) for computing the post-dominance relation based on the flow equations.

Source code for Problem 4:

```
#include <stdio.h>

#define SWAP(a,b) { int _tmp = a; a = b; b = _tmp; }

int cur_count = 0, alloc_count = 0, total_count = 0, num_elems = 0;

int *counts, *psums, *elems;

void initialize(int n, FILE *fptr) {
      int i = 0;

      num_elems = n;
      counts = (int *) calloc(n, sizeof(int));
      psums = (int *) calloc(n, sizeof(int));
      elems = (int *) calloc(n, sizeof(int));

      for (i = 0; i < n; ++i) {
            elems[i] = i + 1;
            fscanf(fptr, "%d", &counts[i]);
            psums[i] = total_count += counts[i];
      }
}

int next_elem() {
      int top = num_elems - 1;
      int ret = elems[top];

      if (alloc_count >= total_count)
            return 0;

      ++alloc_count;
      ++cur_count;
      --psums[top];
      --counts[top];

      while (top > 0 && counts[top] <= (cur_count * psums[top - 1])) {
            psums[top - 1] += (counts[top] - counts[top - 1]);
            SWAP(counts[top], counts[top - 1]);
            SWAP(elems[top], elems[top - 1]);
            --top;
      }

      if (top != num_elems - 1)
            cur_count = 0;

      return ret;
}
```

```
int main() {
      int next;

      printf("number of elements: ");
      scanf("%d", &num_elems);
      printf("\nweights:\n");
      initialize(num_elems, stdin);
      printf("\n");
      while ((next = next_elem()) > 0)
            printf("%d ", next);
      printf("\n");
}
```