EE 367 Spring 2018 Final Exam

Last updated May 1, 2018

Instructions:

You are allowed to use *your own*

- textbooks
- homeworks
- homework solutions
- lecture notes
- lab assignments and solutions from this course or from previous courses, e.g., your discrete math course.

You may ask the instructor questions verbally or by email. Once you get a copy of this exam, you are ***not*** allowed to use anything else including

- discussions with others,
- the Internet,
- other books or references,
- course materials from others,
- or materials from other courses.

By taking this exam, you agree to abide by these instructions.

Note that there is a bonus problem 7 that is an extra 2 points. It is focused on Boolean algebra and has been included to collect data for accreditation.

Submission Instructions:

Problems 1, 2, 3, and 4a: Deadline May 10 Thursday 1:00-4:00PM Holmes 436 (Sasaki's office)

- Submit a hard copy to me in my office Holmes 436 or slide it under my door during the hours.

Problems 4b, 5, and 6: Deadline May 10 Thursday 11:00 PM laulima

- Submit the program files change.c, palinfinal.c, and bfs.c in laulima
- Also submit a softcopy (pdf) of Problems 1, 2, 3, and 4a in laulima to ensure I get a copy.

Problem 1 [20 pts]. Consider the following binary search tree which has the following node structure and search function. Assume that the tree is not empty, and that a search will always be successful.

```
struct node {
	int key;
	struct node * left;  /* Left child */
	struct node * right /* Right child */
};

struct node * search(struct node * head, int key)
{
struct node *current = head;

while (current->key != key) {
	if (key < current->key) {
		current = current->left;
	}
	else {
		current = current->right;
	}
}
return current;
}
```

Prove that the search function is correct, and in particular, answer the following questions (you can find the definitions in CLRS textbook, pages 18-19):

What is the loop invariant of the while-loop?

What is the initialization?

What is the maintenance?

What is the termination?

Problem 2. Consider following RANDOM-SEARCH algorithm that searches for an integer value x in an array A[] that consists of n elements. The algorithm picks a random index k into A. If A[k] = x then we terminate; otherwise, we continue the search by picking random indices into A until we find an index j such that A[j] = x or until we have checked every element of A. Note that we pick from the whole set of indices each time, so that we may examine a given element more than once. The following is the pseudo code for RANDOM-SEARCH. It use a random number generator (or a pseudo random number generator) Random(u, v) that returns a random integer in the range [u, v], where each integer is equally likely.

RANDOM-SEARCH

```
int visited[n]          Comment:  visited[k] indicates if element A[k] has been searched
int numberVisited = 0   Comment:  keeps track of the number of nodes visited so far

for i = 1, 2, ... n
    visited[n] = 0  Comment:  initialize visited

while numberVisited < n  Comment:  keep searching if there are some elements that haven't been visited
    k = Random(1,n)
    if visited[k] = 0 then
        visited[k] = 1
        numberVisited = numberVisited + 1
    if A[k] = x then
        return k;
```

(a) [5 pt] Suppose there is exactly one index k such that A[k] = x. What is the expected number of indices into A that we must pick before we find x and RANDOM-SEARCH terminates?

(b) [5 pt] Suppose there are no indices k such that A[k] = x. We are interested in computing the expected number of indices into A that we must pick before we have checked all elements of A and RANDOM-SEARCH terminates.

Problem 3. This problem comes from CLRS Textbook problem 16.2-3 (page 427). Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is in the same as the order when sorted by decreasing value. Consider the following algorithm to solve the problem:

Greedy-Knapsack Algorithm:

```
Sort items according to increasing weight, i.e., lightest first
Pick items to put in the knapsack according to the order until the knapsack can hold no more.
```

(a) What is the time complexity of this algorithm? [5 pts] (Don't forget to explain details such as how the algorithm will sort.)

(b) Prove that the algorithm is correct [5 pts]. (Note that this is a greedy algorithm)

Problem 4. Given an infinite number of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent), let *f*(*n*) = the number of ways to represent *n* cents. For example, *f*(10) = 4 because there are 4 ways to represent 10 cents: { 1 dime }, { 2 nickels }, { 1 nickel + 5 pennies} , { 10 pennies }

(a) [10 pts] Write a recursive function *f*(*n*) for all n > 0. Of course, the base case is f(1) = 1.

(b) [10 pts] The attached program "change.c" will compute and display *f*(*n*) given an input *n*. The following will run the program:

```
./a.out  <n>
```

Its function "change(*n*)" computes and returns the value *f*(*n*). Currently, the function doesn't work and only returns the value 0.

Rewrite "change( )" so that it works using top-down dynamic programming with memoization. Upload your version of change.c in laulima.

Problem 5 [20 pts]. Attached is the program palinfinal.c which is a solution to Problem 5(a) in Take Home Exam 2. The function palinsub( ) uses a bottom-up, dynamic programming algorithm with memoization. The memoization uses a 2-dimensional array, so the space complexity is $\Theta(n^2)$, where n is the length of the input string.

Rewrite palinsub( ) so that it has space complexity $\mathbf{O}(\boldsymbol{n})$. Upload your version of palinfinal.c in laulima.

(Hint: Notice that computing memo[k][k+diff] for all k is to compute along a diagonal of the matrix memo[][]. This diagonal one-dimensional. The recursion to compute the diagonal uses memo[i][i+diff-1], memo[i+1][i+diff], and memo[i+1][i+diff-1], i.e., it uses two one-dimensional, diagonals.)

Note: Also attached are two files string1.dat and string2.dat which are example input files.

Problem 6 [20 pts]. Attached to this document is the program bfs.c which computes shortest paths in an undirected graph G = (V,E), where a path's length is the number of links. To run the program, do the following:

```
./a.out   <input file>
```

The input file has the following format:

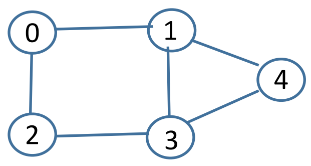
```
N
M
X0  Y0
X1  Y1
...
```

where N is the number of nodes, which are assumed to be 0, 1, ..., N-1; M is the number of undirected links; (X0, Y0) is the first link; (X1, Y1) is the second link; and so on.

The program bfs.c will display the shortest paths from the source, which is assumed to be 0, to all the other nodes. Thus, if N = 10, the program will display 9 shortest paths from source node 0 to the other nodes 1, 2, ..., 9.

For example, consider the following input file for the graph below:

```
5
6
0 1
0 2
1 3
2 3
3 4
1 4
```



Then function bf will compute the shortest paths and display them, e.g.,

```
Shortest paths:
0 -> 1
0 -> 2
0 -> 2 -> 3
0 -> 1 -> 4
```

(Note that there may be more than one shortest path. For example, there are two shortest paths from 0 to 3: 0->2->3 and 0->1->3. The program can display any of the shortest paths.)

There is a function bfs(adjlist, n) in bfa.c which computes the shortest paths using the breadth first search algorithm. The function also displays the shortest paths. The input parameter 'adjlist' is the adjacency list of the graph and its link weights, and n is the number of nodes in the graph. The implementation of the breadth first search algorithm should have O(|V| + |E|) time complexity.

Currently bfs( ) doesn't work, so you must make the changes so bfs.c works properly. Upload your version of 'bfs.c' in laulima.

Note: Also attached is bfs.dat which is an example input file.

Problem 7 BONUS:

(a) [1 pt] Consider the following function (it assumes its input parameter values and return values are 0 or 1):

```
int check(int allie, int billie, int cassie)
{
if (((allie==0) && (cassie==0)) || ((allie==1) && (cassie==0)) || ((allie==1)&&(billie==1))
        return 1;
else
        return 0;
}
```

Rewrite the function by simplifying the condition in the if-statement (use boolean algebra or a K-map).

(b) [1 pt] Suppose we wanted to implement the function 'check' as a digital circuit composed of AND, OR, and NOT gates:

- There are three inputs to the circuit: 'allie', 'billie', and 'cassie'.
- The output of the circuit is the return value of the function.

Draw the digital circuit which uses the minimum sum of product expression.