Homework 4 Exercises

Exercise 15.1-2

Show, by means of counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the density of the rod of length i to be $\frac{p_i}{i}$, that is, its value per inch. The greedy strategy for a rod of length n cuts off a piece of length i, where $1 \le i \le n$, having the maximum density. It then continue by applying the greedy strategy to the remaning pieve of length n-i.

For this problem, I create a table of rod values.

length i	*value p_i*	*density $\frac{p_i}{i}$*
1	1	1
2	1	0.5
3	12	4
4	1	0.25
5	15	3

Starting with a rod of length 5, it's trivial to see the optimal solution is zero cuts, yielding a single rod of length 5, total value of 15. However, if we use the greedy strategy:

1. Starting with a rod of length 5, we see that a rod of length 3 has the highest density. We make our first cut giving a rod of length 3, and a remainder of length 2.

2. Using the remaining rod of length 2, the highest density remaining is length 1. We cut our length 2 rod into two sections of length 1, and the greedy algorithm is completed.

Now, for the greedy approach, we have a total value of $p_3 + p_1 + p_1 = 12 + 1 + 1 = 14$. However, if we made no cuts at all, we would have had a single rod of length 5, giving $p_5 = 15$. Thus, by counterexample, the greedy strategy did not give an optimal solution.

Exercise 15.1-5

The Fibonacci numbers are defined by recurrence (3.22). Give an O(n) time dynamic-programming algorithm to compute the n-th Fibonacci number. Draw the subproblem graph. How many vertices and edges are in the graph?

To solve this with dynamic programming, we solve all of the subproblems in a bottom-up approach, starting at 0. However, the sequence is pre-defined for zero and one, so we can hard-code those values. NOTE: These will not show up as vertices in the graph. Below is my c++ implementation of a dymanic fibonacci sequence calculator for integer n:

```
int dynamic_fibonacci(int n) {
    int fib [n+1];
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i <= n; i++) {
        fib[i] = fib[i-1] + fib[i-2];
    }

    return fib[n];
}
```

The subproblem graph is fairly well behaved; every vertex in our graph depends on two previous subproblems. Two subproblems, 0 and 1, have no dependencies on previous results, and are thus hardcoded. Because the fibonacci sequence starts at 0 and not 1, we have a subproblem graph with $\#V = n + 1$ vertices. Each vertex has 2 subproblems, except for vertex 0 and 1. Thus, we have $\#E = (n - 1) * 2$ total edges. Two subproblem graphs are shown, one for n=4, and one for generic n.

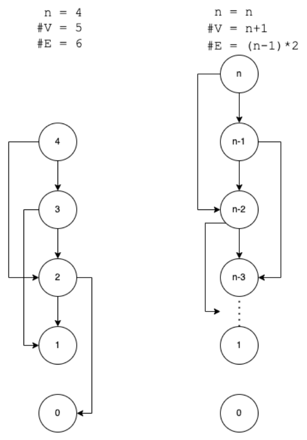

Figure 1: Subproblem graph of dynamic fibnocci algorithm for n=4 (left) and generic n (right).

Exercise 15.4-1

Determine an LCS of <1,0,0,1,0,1,0,1> and <0,1,0,1,1,0,1,1,0>.

We will work through this step by step. First, noting the final characters do not match, we have two subproblems, one LCS with one character removed from either string. If the final characters match, then we only have one subproblem; we subtract that chracter from both strings, and run our algorithm on the remainder two strings. This is repeated until we get to the beginning of either string. Running this in python, we get the following six character LCS:

$< 0, 1, 0, 1, 0, 1 >$