Lab 9

Due No Due Date **Points** 1

You are suggested to use the teaching servers burrow.soic.indiana.edu or hulk.soic.indiana.edu or tank.soic.indiana.edu for practicing C programs.

Lab 9: Dynamic programming

Rod Cutting

You are given a piece of rod of length n and because different rod with different length sells at different price, you want to decide how to cut the rod into pieces to make the maximal profit.

Price for different length of rod is given like the following(suppose the rod is of length 8):

```
length   | 1   2   3   4   5   6   7   8
--------------------------------------------
price    | 3  4.5  8   9  10   17  17  20
```

The optimal way to cut the above rod of length 8 is to cut it into 8 pieces of length 1 and the maximal profit is 24.

How to solve this problem with dynamic programming?

First of all, what is subproblem structure in this problem?

Naturally we can try to look for a subproblem structure in term of the rod length. So for a rod of length n, can we combine the optimal solution for rods with other lengths to get the optimal solution for rod of length n? The answer is yes. We can first consider where to put the **right most** cut on the rod and then we can remove the right most part and consider recursively where to put the next right most cut. So the problem now is where to put the right most cut. Without any further information for us to decide where to put the right most cut, we can try every possible cut including no cut at all. So the recursive formula is:

OPT(n) = max(price[n] , price[t] + OPT(n-t)) where 1<= t <=n-1
OPT(1) = price[1], // you cannot cut further

This recursive formula shows OPT(n) only depends on smaller subproblems and the computation of these subproblems overlap a lot, so we can use a memoization array of size n to avoid repeated computation.

Pseudo code:

```
Mem[1..n]  // for memoization
OPT(price_list, n):
        Mem[1] = price_list[1]
        for ( i from 2 to n) {
             m = price_list[i]
             for ( j from 1 to i-1){
                    m = max (m , price[j]+Mem[i-j])
             }
             Mem[i] = m
        }
        return Mem[n]
```

Of course, this just returns the maximal profit. We also want the actually length of every piece. **So you also need to recover the length of every piece from the memoization array.**

Question: What is the running time and space complexity?

Some more condition

Suppose every time we cut the rod we have to pay of price of c for it. As a result, if we cut too many times, we would lose a lot of profit.

Modify your program so that every time you make a cut you pay an aditional cost of c.

In case of cost being 1, the above example will yield {1,1,6} with maximal profit 21.