CSCI 270 Lecture 5: Dynamic Programming

```
**Fibonacci**(int n)
1: If $n \leq 2$ Then Return 1
2: Return **Fibonacci**(n-1)+**Fibonacci**(n-2)
```

Calculating Fibonacci Numbers

- What is the recurrence relation for this algorithm?

- Can you use the solve-by-tree method to solve this recurrence?

- What is my algorithm doing which is rather stupid?

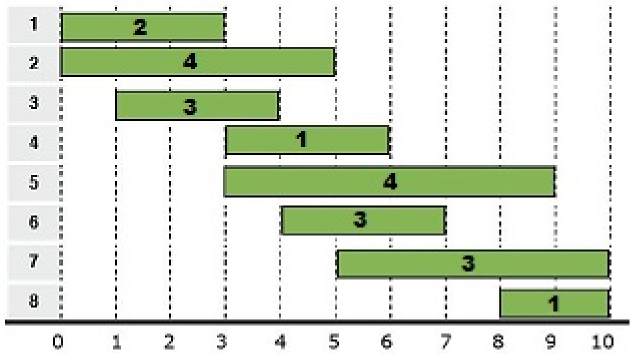
- How could I fix this problem?

```
**DynamicProgrammingFibonacci**(int n)
1: $A[1] = 1$
2: $A[2] = 1$
3: For $i = 3$ to $n$
4:     $A[i] = A[i-1] + A[i-2]$
5: Return $A[n]$
```

Tabulation is the process of writing down intermediate results in an array to refer to later.

Dynamic Programming is the process of transforming a recursive function which replicates work into an iterative one which solves each subproblem once, writing down the answer for future reference.

In the **Weighted Interval Scheduling** problem, we have n jobs. Job j has start time s_j, finish time f_j and value v_j. Two jobs are incompatible if their times overlap. Find the max-valued subset of mutually compatible jobs.



What is the optimal solution for the above instance?

Finding the optimal solution in general is a daunting task. It is always a good idea to split the problem up into smaller, bite-size pieces.

Question 1: Do I include interval 1 in my solution or not?

Question 2: Do I include interval 2 in my solution or not?, etc.

The top level of our recursion will tackle the first piece. Each successive function call will tackle a later piece. We *do not yet know* if we should include interval i. Therefore, we will try including it, and try not including it, and determine which of the two produce a better solution overall.

- If I **do not** include interval 1, what is the *next* question which should be asked?
- If I **do** include interval 1, what is the *next* question which should be asked?
- If x is the best value that can be obtained on intervals 4 through 8, then what is the best value that can be obtained if I include interval 1?

We will define $S[z]$ to be the first interval i such that $s_i \geq f_z$. We are assuming that the intervals are sorted by s_i, and that we fill this array out prior to running our recursive algorithm. We will define $S[z] = n + 1$ when there are no intervals that satisfy this condition.

```
WIS(integer $z$)
1: If $z > n$ Then Return 0
2: $x =$ WIS($z + 1$)
3: $y = v_z +$ WIS($S[z]$)
4: Return max($x, y$)
```

After this point we want to unroll this into a iterative solution which does not repeat work.

```
WIS(integer $z$)
1: $W[n+1] = 0$
2: For $z = n$ to 1
3:     $W[z] = \max(W[z+1], v_z + W[S[z]])$
4: Return $W[1]$
```

1. This is our base case from the recursive function.
2. Here we are identifying the order in which we fill the array. We need to identify the order such that we have the information we need when we need it. If we tried to fill this array in from 1 to n, we would have failed.
3. We're calculating $W[z]$ in the **exact** same way we calculated WIS(z), except we have the values $W[z+1]$ and $W[S[z]]$ written down (tabulated) for future use, instead of recursively calculating them on the spot.
4. After we have found our array, we need to identify where the answer is stored in the array, and return that value.

Here is the array for our example:

7	7	7	4	4	4	3	1	0

You can write your solutions in the above manner if you wish. I'll go through the exact same process, but in a less formal, more "pseudo-code" type description, which you are encouraged to use. This is generally easier to read and grade. You must identify 5 things in such a description:

1. State what parameters your function accepts, and what value it returns. We are not mind-readers, so please explain what you are doing.
   Let WIS[z] *contain the best value which is attainable on intervals z through n.*
2. Give the recursive function to calculate your intended function output. This is usually the hardest step.
   WIS[z] $= \max(\text{WIS}[z+1], v_z + \text{WIS}[S[z]])$
3. Give the base case(s) for your recursive function.
   WIS[$n+1$] $= 0$
4. State what order you will fill the array in for your iterative procedure.
   Fill the array in reverse order from WIS[$n+1$] *to* WIS[1]
5. State where the answer is stored in your final array.
   The answer is stored at WIS[1]

Dynamic Programming Design Process

1. Reduce the problem to a series of ordered decisions (What is the nth, (n-1)st, (n-2)nd, etc, fibonacci number? Do we include interval 1, 2, 3, etc?). The simpler the decisions are, the better.
2. Make sure your subproblems can be described concisely: this description is what we pass down to the next level of recursion (We only need to remember which fibonacci number we're currently calculating. We only need to remember which interval we're currently considering).
3. Design a recursive procedure to make these decisions in an ordered way.
4. Instead of using recursion, show how to solve this problem by iteratively filling out a table.

Why do you suppose this programming technique is called **Dynamic** Programming?

Extra Problems:

- Chapter 6: exercises 1, 4, 6, 19, 20, 26, 27
- Challenge problems: Chapter 6: exercise 24