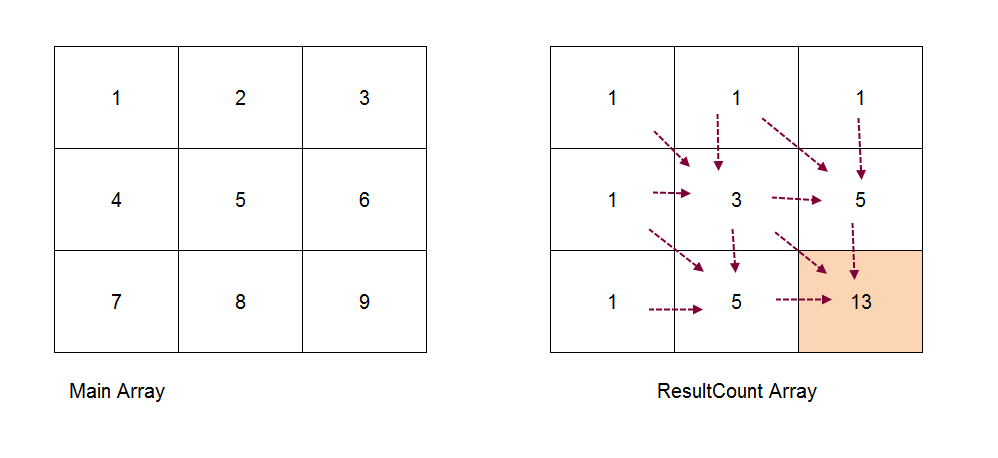
**Dynamic Programming**

1. Read the background info in the **"00 Info"** folder.
2. You will begin by (again!) implementing the Fibonacci sequence. However, this time you start in Excel.
   1. Open the **"Fibonacci tables"** file in the corresponding folder. Noting the ***recurrence relation*** of the sequence (f(i) = f(i-2) + f(i-1)), manually fill in the table on the first sheet to find the ith Fibonacci number, given the values of the previous numbers. You are dynamically programming! Sort of.
   2. Convert the hand-filled algorithm to Excel functions, and use Excel's fill-right ability to populate the array on the second sheet.
3. Peruse the **Bottles** problem *with extreme prejudice.*
   1. You should hopefully recognize this as a recursive backtracking problem, which you *could* now memoize for a decently fast solution for arbitrarily long arrays. Instead, think about how you could build the *next* solution from the base cases, then how you would build the *next* solution after that, given the result of the previous solution.
   2. Repeat what you did for Fibonacci in Bottles' Excel file. Complete both sheets, noting the recurrence relation of the solutions.
   3. Convert the Excel algorithm to Java code to solve the problem in the folder, given the input file.
4. Solve the **Count Paths** problem. **IMPORTANT: Before writing any code, solve the problem by hand. Begin with a 1x1 matrix** (shouldn't take long!)**, then move to a 2x2 matrix, then to a 3x3 matrix** (working it out by hand should *always* be your first step with DP problems). If it helps, you could create another Excel file for this. Observe that each solution can be used to build the next solution:



Convert your hand-written algorithm to Java code. You could also use this method to solve [Project Euler problem 15](https://projecteuler.net/problem=15). Nice!

1. In the previous problems, you were given the base cases and the recurrence relation (essentially the solution algorithm). For the next problem (**Rod Cutting**), you must figure this out on your own (with some help, of course).
   1. Just like with recursion, your first step should be to establish your base cases – problems so trivially simple, you can solve them right away.
   2. Next, you must establish the recurrence relation. With recursion, after you establish base cases your next step is to break the problem up into smaller and smaller pieces.

With DP, your job is to build the *next* solution given the already solved *previous* solutions. DP is bottom up, while recursion is top down! All problems that are suitable for DP exhibit this ***optimal sub-structure*** and ***overlapping sub-problems*** property.

* 1. Tabulate solutions to sub-problems in the data structure of your choice (which will almost always be an array).

1. Complete the remainder of the problems in order. Good luck! ***Always* start on paper first; writing code should never be the first step with DP** (except for the most trivial problems). DP is like recursion: it is equal parts art and science. It can take a lot of practice to get good at!