A sidebar indicates the "answer" for each step — it is content that is part of following the algorithm development process, separate from commentary about the process or the solution itself.

Winning the Series

In sporting events where the outcome depends on the results of several games (such as World Series in baseball or the Stanley Cup in hockey), there are always statistics about how likely it is for a team that is down some number of games to win it all. The championship series problem is defined as follows: two teams A and B are playing a series of games in which the first team to win n games is the winner. Assume that team A has a probability of p of winning any given game (meaning that team B has a probability of 1-p of winning the game). Suppose that some number of games have already been played, where team A has won i of them and team B has won j. (Assume that ties are not allowed.) Determine the probability that A will go on to win the series.

Math refresher: the probability that two independent events $XandY$ will both occur is the probability of X occurring multiplied by the probablity of Y occurring. That's why, for example, the probability of flipping heads twice in a row is 1/4 - 1/2 for the first head times 1/2 for the second head. Furthermore, the probability that either X or Y will occur is the probablity of X occurring plus the probability of Y occurring.

Specifications. State complete specifications for the problem. What is the problem? What do you start with (input) and what is the end result (output)? What are the legal input instances and the required output for each? For optimization problems, distinguish between legal solutions and optimal ones.

Task: Determine the probability of A winning the series given that A has won i games so far and B has won j.

Input: number of games won by A and B, probability p of A winning a game

Output: probability of A winning the series

This isn't an optimization problem, so the optimization goal isn't relevant. What constitutes a legal solution also isn't really relevant.

Examples. Examples should clarify the task and/or help you figure out an algorithm.

If an example is given here, show how the probability is computed rather than just giving a number.

$p = 0.6$, $i = 2$, $j = 1$, $n = 4$ — A needs to win two games before B wins three. Ways for A to accomplish this:

```
A A        0.36
A B A      0.144
A B B A    0.0576
B A A      0.144
B A B A    0.0576
B B A A    0.0576
```

The probability of A winning series is the sum of these values, 0.8208.

Size. What is the size of the problem, and what constitutes a smaller problem? What is the simplest/smallest instance of the problem?

The size of the problem is n, the number of games that a team must win to win the series.

The smallest problem is $n = 1$.

Approaches. Identify the particular flavor, if applicable, as well as the two patterns (process input and produce output) and what they look like for this problem.

This is something of a labeling problem — each game has a winner, and we are considering the possible winners.

Process input: for each game, either A or B wins

Produce output doesn't really make sense for labeling problems.

Generalize / define subproblems. Friends get smaller versions of the original problem. Define the task, input, and output for the subproblems.

Partial solution. Identify what constitutes a solution-so-far. This will be the same kind of thing as a legal solution for the whole problem, but less complete (fewer choices have been made).

Which team has won each of the games so far, or, more directly, how many games A has won and lost so far.

Alternatives. Identify the choice being made and what the (legal) alternatives are for that choice. (Alternatives that result in an illegal solution aren't considered farther.)

The choice is who wins the next game, A or B.

Subproblem. The "rest of the problem". Define the generalized problem, its input, and its output. The input can include the partial solution so far, in which case the output would be a complete solution, or the input can only denote the subproblem, in which case the output is only the solution for the subproblem.

The original problem was the determine the probability of A winning the series given that A has won i games so far and B has won j — that's already generalized.

Define some notation:

`prob(`a`,`b`)` — the probability of A winning the series given that A has won a games so far and B has won b

Main case. This takes the form of: for each legal alternative for the current decision, a friend solves the remainder of the problem given the solution so far plus that choice. We return one of those solutions, the best of those solutions, or all of the those solutions depending on the structural variation.

```
prob($a$,$b$) = $p$*prob($a+1$,$b$) + (1-$p$)*prob($a$,$b+1$)
```

Base case(s). A base case occurs when there are no more choices to make.

A complete solution has been found when one team wins the series.

```
prob($a$,$n$) = 0     // B has won
prob($n$,$b$) = 1     // A has won
```

Top level. The top level puts the context around the recursion.

Initial subproblem. `prob(`i`,`j`)` — the original task was to determine the probability of A winning the series given that A has won i games so far and B has won j

Setup. Nothing needed.

Wrapup. Nothing — the solution to the initial subproblem is the desired answer. (We want the probability, not a sequence of game outcomes.)

Special cases. Make sure the algorithm works for all legal inputs — revise the previous steps to add handling for special cases as needed.

None.

Termination. Show that the recursion ends. For making progress, explain why each subproblem is smaller. For reaching the end, show that a base case is always reached.

Making progress. Either a or b increases by 1 in each step, so $a + b$ always increases by 1.

Reaching the end. The base case is when a or b reaches n, which is guaranteed to happen by the time that $a + b = 2n$. Since $a + b$ increases by 1 in each step, it will eventually reach $2n$.

Correctness. Explain why the base case answer is correct. Explain why the main case covers all of the legal next possibilities, the right parameters are passed to the subproblems, and answer is correct assuming correct subproblem answers. Explain why the parameter values for the initial subproblem are correct.

Establish the base case(s). When one team has won the series, that team is either A or B — A has either won, or has no chance because B won.

Show the main case. There are two possibilities for the outcome of the next game: A wins or B wins. The probability of A winning the series given a win in the next game is `prob(`$a + 1$`,`b`)` and the probability of A winning the series given a loss in the next game is `prob(`a`,`$b + 1$`)`. Thus the probability of A winning the series from the current point is `prob(`a`,`b`) = `p`*prob(`$a + 1$`,`b`) + (1-`p`)*prob(`a`,`$b + 1$`)`.

Final answer. The original problem asked for the probability of A winning the series given A has won i games and B has won j.

Implementation. Identify data structures and, as necessary, specific implementations of those data structures to efficiently support the algorithm. Also fill in any algorithmic steps that haven't been specified.

Memoization. Identify how to parameterize subproblem state for efficient lookup, typically in an array.

a and b are integers $0..n$, so `prob(`a`,`b`)` can be stored in `prob[`a`][`b`]`.

Order of computation. Loop(s) can be used to fill in the array; identify whether the loop(s) go from small large index values or vice versa.

The base cases are stored in `prob[`n`][`b`]` and `prob[`a`][`n`]`. Fill in the rest of the slots in the order `(n-1)..0` to ensure that subproblem solutions are computed before they are needed — higher a and b are done first because each problem depends on subproblems with a higher a or a higher b.

Dynamic programming. Combine the memoization and order of computation with the base and main cases and the top level.

`prob(`i`,`j`,`p`)` — the probability of A winning the series given that A has won i games so far and B has won j

Input: number of games won by A and B, probability p of A winning a game

Output: probability of A winning the series

```
// initialize the base cases
for a = 0 to n-1
  prob[a][n] = 0
for b = 0 to n-1
  prob[n][b] = 1

// fill in the rest of the array
for a = n-1 downto 0 do
  for b = n-1 downto 0 do
    prob[a][b] = $p$ prob[$a+1$][$b$] + (1-$p$) prob[$a$][$b+1$]

// solution
return prob[i][j]
```

Time and space. Assess the running time and space requirements of the algorithm given the implementation identified.

The running time for a dynamic programming algorithm is the number of slots in the array times the work done per slot to compute the subproblem solution.

Both a and b are $0..n$ so `prob` has n^2 slots. Computing `prob[a][b]` is $O(1)$ — it only involves some arithmetic. Thus the running time is $O(n^2)$.

Space is $O(n^2)$ for the array.