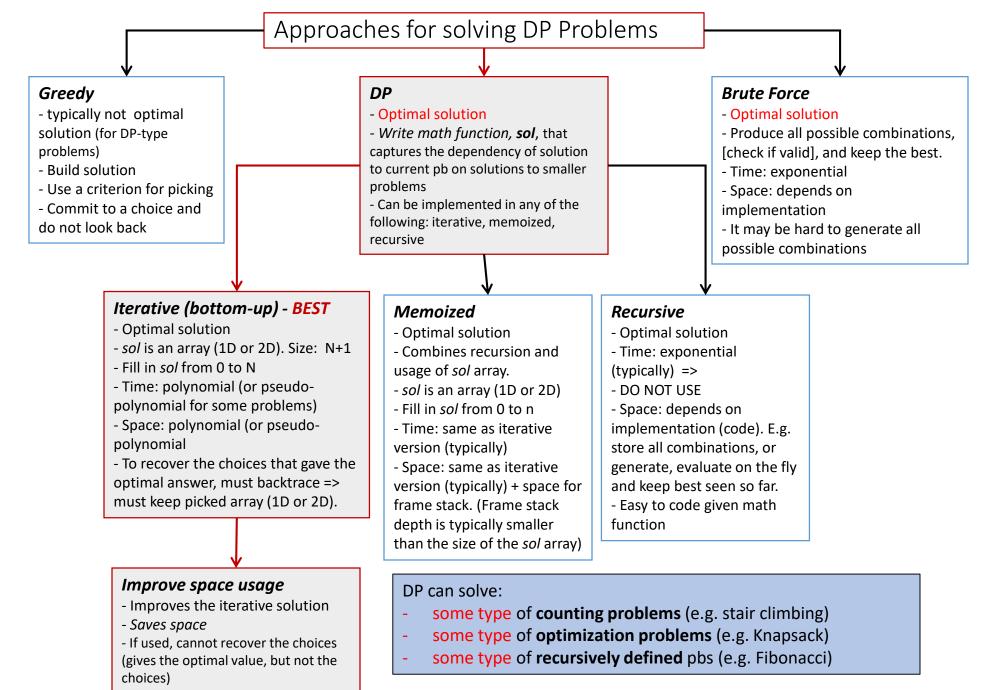
Dynamic Programming General

CSE 3318 – Algorithms and Data Structures University of Texas at Arlington

Alexandra Stefan



SOME DP solutions have *pseudo* polynomial time

Dynamic Programming (DP) - CLRS

- Dynamic programming (DP) applies when a problem has both of these properties:
 - **1. Optimal substructure**: "optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently".
 - **2. Overlapping subproblems**: "a recursive algorithm revisits the same problem repeatedly".
- Dynamic programming is typically used to:
 - Solve optimization problems that have the above properties.
 - Solve counting problems –e.g. Stair Climbing or Matrix Traversal.
 - Speed up existing recursive implementations of problems that have overlapping subproblems (property 2) – e.g. Fibonacci.
- Compare dynamic programming with divide and conquer.

Iterative or Bottom-Up Dynamic Programming

- Main type of solution for DP problems
- We can define the problems size and solve problems from size 0 going up to the size we need.
- Iterative because it uses a loop
- Bottom-up because you solve problems from the bottom (the smallest problem size) up to the original problem size.

Bottom-Up vs. Top Down

- There are two versions of dynamic programming.
 - Bottom-up.
 - Top-down (or memoization).

- Bottom-up:
 - Iterative, solves problems in sequence, from smaller to bigger.
- Top-down:
 - Recursive, start from the larger problem, solve smaller problems as needed.
 - For any problem that we solve, <u>store the solution</u>, so we never have to compute the same solution twice.
 - This approach is also called **memoization**.

Top-Down Dynamic Programming (Memoization)

- Maintain an array/table where solutions to problems can be saved.
- To solve a problem P:
 - See if the solution has already been stored in the array.
 - If yes, return the solution.
 - Else:
 - Issue recursive calls to solve whatever smaller problems we need to solve.
 - Using those solutions obtain the solution to problem P.
 - Store the solution in the solutions array.
 - Return the solution.

Steps for iterative (bottom up) solution

- 1. Identify trivial problems
 - 1. typically where the size is 0
- 2. Look at the last step/choice in an optimal solution:
 - 1. Assuming an optimal solution, what is the last action in completing it?
 - 2. Are there more than one options for that last action?
 - 3. If you consider each action, what is the smaller problem that you would combine with that last action?
 - 1. Assume that you have the optimal answer to that smaller problem.
 - 4. Generate all these solutions
 - 5. Compute the value (gain or cost) for each of these solutions.
 - 6. Keep the optimal one (max or min based on problem)
- 3. Make a 1D or 2D array and start feeling in answers from smallest to largest problems.

Other types of solutions:

- 1. Brute force solution
- Recursive solution (most likely exponential and inefficient)
- 3. Memoized solution