primary address: <https://en.wikipedia.org/wiki/Dynamic_programming>

In mathematics, management science, economics, computer science, and bioinformatics, dynamic programming (also known as dynamic optimization) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions - ideally, using a memory-based data structure. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. (Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) The technique of storing solutions to subproblems instead of recomputing them is called "memoization".

Dynamic programming algorithms are often used for optimization. A dynamic programming algorithm will examine the previously solved subproblems and will combine their solutions to give the best solution for the given problem. In comparison, a greedy algorithm treats the solution as some sequence of steps and picks the locally optimal choice at each step. Using a greedy algorithm does not guarantee an optimal solution, because picking locally optimal choices may result in a bad global solution, but it is often faster to calculate. Fortunately, some greedy algorithms (such as Kruskal's or Prim's for minimum spanning trees) are proven to lead to the optimal solution.

For example, in the coin change problem of finding the minimum number of coins of given denominations needed to make a given amount, a dynamic programming algorithm would find an optimal solution for each amount by first finding an optimal solution for each smaller amount and then using these solutions to construct an optimal solution for the larger amount. In contrast, a greedy algorithm might treat the solution as a sequence of coins, starting from the given amount and at each step subtracting the largest possible coin denomination that is less than the current remaining amount. If the coin denominations are 1,4,5,15,20 and the given amount is 23, this greedy algorithm gives a non-optimal solution of 20+1+1+1, while the optimal solution is 15+4+4.

In addition to finding optimal solutions to some problem, dynamic programming can also be used for counting the number of solutions, for example counting the number of ways a certain amount of change can be made from a given collection of coins, or counting the number of optimal solutions to the coin change problem described above.

Sometimes, applying memoization to the naive recursive algorithm (namely the one obtained by a direct translation of the problem into recursive form) already results in a dynamic programming algorithm with asymptotically optimal time complexity, but for optimization problems in general the optimal algorithm might require more sophisticated algorithms. Some of these may be recursive (and hence can be memoized) but parametrized differently from the naive algorithm. For other problems the optimal algorithm may not even be a memoized recursive algorithm in any reasonably natural sense.