# First Line Indent

In mathematics and computer science, dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems which are only slightly smaller[1] and optimal substructure (described below). When applicable, the method takes far less time than naive methods.

# Line Spacing and First line Indent

The key idea behind dynamic programming is quite simple. In general, to solve a given problem, we need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution. Often, many of these subproblems are really the same. The dynamic programming approach seeks to solve each subproblem only once, thus reducing the number of computations. This is especially useful when the number of repeating subproblems is exponentially large.

# Right indent and left aligned

Top-down dynamic programming simply means storing the results of certain calculations, which are later used again since the completed calculation is a sub-problem of a larger calculation. Bottom-up dynamic programming involves formulating a complex calculation as a recursive series of simpler calculations.

## Center aligned and first line indent

The term dynamic programming was originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another. By 1953, he refined this to the modern meaning, referring specifically to nesting smaller decision problems inside larger decisions,[2] and the field was thereafter recognized by the IEEE as a systems analysis and engineering topic. Bellman's contribution is remembered in the name of the Bellman equation, a central result of dynamic programming which restates an optimization problem in recursive form.

# Justified

In mathematics and computer science, dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems which are only slightly smaller[1] and optimal substructure (described below). When applicable, the method takes far less time than naive methods.

# Right Aligned

In mathematics and computer science, dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems which are only slightly smaller[1] and optimal substructure (described below). When applicable, the method takes far less time than naive methods.

If subproblems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the subproblems.[5] In the optimization literature this relationship is called the Bellman equation.

[edit]Dynamic programming in mathematical optimization

In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of value functions V1, V2, ..., Vn, with an argument y representing the state of the system at times i from 1 to n. The definition of Vn(y) is the value obtained in state y at the last time n. The values Vi at earlier times i = n -1, n - 2, ..., 2, 1 can be found by working backwards, using a recursive relationship called the Bellman equation. For i = 2, ..., n, Vi-1 at any state y is calculated from Vi by maximizing a simple function (usually the sum) of the gain from decision i - 1 and the function Vi at the new state of the system if this decision is made. Since Vi has already been calculated for the needed states, the above operation yields Vi-1 for those states. Finally, V1 at the initial state of the system is the value of the optimal solution. The optimal values of the decision variables can be recovered, one by one, by tracking back the calculations already performed.

Dynamic programming in computer programming

There are two key attributes that a problem must have in order for dynamic programming to be applicable: optimal substructure and overlapping subproblems. However, when the overlapping problems are much smaller than the original problem, the strategy is called "divide and conquer" rather than "dynamic programming". This is why mergesort, quicksort, and finding all matches of a regular expression are not classified as dynamic programming problems.

Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Consequently, the first step towards devising a dynamic programming solution is to check whether the problem exhibits such optimal substructure. Such optimal substructures are usually described by means of recursion. For example, given a graph G=(V,E), the shortest path p from a vertex u to a vertex v exhibits optimal substructure: take any intermediate vertex w on this shortest path p. If p is truly the shortest path, then the path p1 from u to w and p2 from w to v are indeed the shortest paths between the corresponding vertices (by the simple cut-and-paste argument described in CLRS). Hence, one can easily formulate the solution for finding shortest paths in a recursive manner, which is what the Bellman-Ford algorithm or the Floyd-Warshall algorithm does.

Overlapping subproblems means that the space of subproblems must be small, that is, any recursive algorithm solving the problem should solve the same subproblems over and over, rather than generating new subproblems. For example, consider the recursive formulation for generating the Fibonacci series: Fi = Fi-1 + Fi-2, with base case F1 = F2 = 1. Then F43 = F42 + F41, and F42 = F41 + F40. Now F41 is being solved in the recursive subtrees of both F43 as well as F42. Even though the total number of subproblems is actually small (only 43 of them), we end up solving the same problems over and over if we adopt a naive recursive solution such as this. Dynamic programming takes account of this fact and solves each subproblem only once. Note that the subproblems must be only slightly smaller (typically taken to mean a constant additive factor[citation needed]) than the larger problem; when they are a multiplicative factor smaller the problem is no longer classified as dynamic programming.