

Deciding on Sub-Problems for Dynamic Programming

I have used the technique of dynamic programming multiple times however today a friend asked me how I go about defining my sub-problems, I realized I had no way of providing an objective formal answer. How do you formally define a sub-problem for a problem that you would solve using dynamic programming?

algorithms

dynamic-programming

edited Mar 25 '12 at 16:36



Kaveh

17.3k 3 41 96

asked Mar 22 '12 at 3:30



jozefg

965 1 8 18

3 Answers

The principle of dynamic programming is to think top-down (i.e recursively) but solve bottom up. So a good strategy for designing a DP is to formulate the problem recursively and generate sub-problems that way.

answered Mar 22 '12 at 4:36



Suresh

4,171 2 19 30

13 I claim that's the *only* strategy. – [JeffE](#) Mar 22 '12 at 9:35

2 @JeffE Yes, I read and used your notes when teaching my algorithm classes and it was effective. Quote: "Dynamic is not about filling in tables. It's about smart recursion!" – [Dai](#) Mar 22 '12 at 14:06

2 My understanding of how to teach DPs is STRONGLY influenced by @JeffE's notes :) – [Suresh](#) Mar 22 '12 at 16:47

3 It's also possible to automatically turn a top-down recursive procedure into a dynamic programming algorithm. When you are about to return, store the answer in a hash table. On the start of each call, check if the answer is already in the hash table, and if so, return it immediately. Many algorithms become easy with this idea. For example with such a table, recursive algorithms on tries automatically work on DAWGs. By storing a sentinel value in the table at the start of a call, the same algorithms can work even on DFAs. Algorithms on BDDs become very easy to specify recursively. – [Jules](#) Mar 23 '12 at 19:21

1 Last but not least, this can have huge performance advantages. For example the traditional bottom up subset sum algorithm can compute tons of unneeded table entries. With this method only the necessary table entries will be computed. – [Jules](#) Mar 23 '12 at 19:26

As @Suresh pointed out, once you know that your problem can be solved by DP (i.e. it exhibits optimal substructure and overlapping subproblems), you may think of a divide and conquer recursive solution.

Of course, divide and conquer will be highly inefficient because every subproblem encountered in the associated recursion tree will be solved again even if it has already been found and solved. This is where DP differs: each time you encounter a subproblem, you solve it and store its solution in a table. Later, when you encounter again that subproblem, you access in $\mathcal{O}(1)$ time its solution instead of solving it again. Since the number of overlapping subproblems is typically bounded by a polynomial, and the time required to solve one subproblem is polynomial as well (otherwise DP can not provide a cost-efficient solution), you achieve in general a polynomial solution.

Therefore, thinking about a divide and conquer solution will provide you with the insight about what a subproblem may be for your particular problem.

edited Mar 22 '12 at 11:07



Raphael ♦

56k 22 137 303

answered Mar 22 '12 at 9:16



Massimo Cafaro

3,331 10 17


1 "optimal substructure" (whatever that means) is probably not a sufficient condition for DP-solvability. "Overlapping subproblems" is certainly not a necessary one. – [Raphael](#) ♦ Mar 22 '12 at 11:08

1 Optimal substructure and overlapping subproblems are both exhibited by problems that can be efficiently solved by DP. Of course optimal substructure alone is not enough for DP solvability. However, if you do not have overlapping subproblems, then you can solve the problem by ordinary divide and conquer with the same cost: indeed, the advantage of DP over divide and conquer is that each subproblem is solved exactly once when encountered in the recursion tree. – [Massimo Cafaro](#) Mar 22 '12 at 12:16

1 It's not my formulation: you will find it on "Introduction to algorithms" by Cormen, Leiserson, Rivest and Stein and on many other textbooks on algorithms. – [Massimo Cafaro](#) Mar 22 '12 at 15:00

1 Jup, and most are at least partially wrong. I am happy to elaborate if you post a suitable question. – [Raphael](#) ♦ Mar 22 '12 at 15:35

1 I am not sure I understand correctly your last comment. To show that this kind of characterization is wrong (it

can't be partially wrong: either is correct or it is wrong), you can simply show as a counterexample, a problem which does not exhibit both optimal substructure and overlapping subproblems, yet is amenable to a polynomial DP solution. But note that, in this context, that means a solution which is provably better than ordinary divide and conquer. – [Massimo Cafaro](#) Mar 22 '12 at 16:23 

|

My experience is finding out a way to "cut down redundant enumerating with help of storing useful value already enumerated". By the way, Dynamic Programming is really popular in ICPC(International Collegiate Programming Contest. Anyone can have his own feeling about DP after practice several ICPC problems.

answered Mar 22 '12 at 9:32



[Peng Zhang](#)

133  5