Previous Chapter Return to Table of Contents Next Chapter

PART IV: Advanced Design and Analysis Techniques

Introduction

This part covers three important techniques for the design and analysis of efficient algorithms: dynamic programming (Chapter 16), greedy algorithms (Chapter 17), and amortized analysis (Chapter 18). Earlier parts have presented other widely applicable techniques, such as divide-and-conquer, randomization, and the solution of recurrences. The new techniques are somewhat more sophisticated, but they are essential for effectively attacking many computational problems. The themes introduced in this part will recur later in the book.

Dynamic programming typically applies to optimization problems in which a set of choices must be made in order to arrive at an optimal solution. As choices are made, subproblems of the same form often arise. Dynamic programming is effective when a given subproblem may arise from more than one partial set of choices; the key technique is to store, or "memoize," the solution to each such subproblem in case it should reappear. Chapter 16 shows how this simple idea can easily transform exponential-time algorithms into polynomial-time algorithms.

Like dynamic-programming algorithms, greedy algorithms typically apply to optimization problems in which a set of choices must be made in order to arrive at an optimal solution. The idea of a greedy algorithm is to make each choice in a locally optimal manner. A simple example is coin-changing: to minimize the number of U.S. coins needed to make change for a given amount, it suffices to select repeatedly the largest-denomination coin that is not larger than the amount still owed. There are many such problems for which a greedy approach provides an optimal solution much more quickly than would a dynamic-programming approach. It is not always easy to tell whether a greedy approach will be effective, however. Chapter 17 reviews matroid theory, which can often be helpful in making such a determination.

Amortized analysis is a tool for analyzing algorithms that perform a sequence of similar operations. Instead of bounding the cost of the sequence of operations by bounding the actual cost of each operation separately, an amortized analysis can be used to provide a bound on the actual cost of the entire sequence. One reason this idea can be effective is that it may be impossible in a sequence of operations for all of the individual operations to run in their known worst-case time bounds. While some operations are expensive, many others might be cheap. Amortized analysis is not just an analysis tool, however; it is also a way of thinking about the design of algorithms, since the design of an algorithm and the analysis of its running time are often closely intertwined. Chapter 18 introduces three equivalent ways to perform an amortized analysis of an algorithm.

Go to Chapter 16   Back to Table of Contents