- Dynamic programming is used to solve optimization problems
- In contrast to the divide-and-conquer problem, dynamic programming applies when the subproblems overlap.
- A dynamic-programming algorithm solves each sub-subproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each sub-subproblem.
- The basic idea behind dynamic programming is eliminating overlapping recursive calls by using more memory to keep track of previous values.
- We typically apply dynamic programming to **optimization problems**, such problems can have many possible solutions.
- (P359) When developing a dynamic-programming algorithm, we follow four steps:
  1. Characterize the structure of an optimal solution
  2. Recursively define the value of an optimal solution
  3. Compute the value of an optimal solution, typically in a bottom-up fashion
  4. Construct an optimal solution from computed information

15.1 Rod cutting

- P360 We can cut up a rod of length n in $\mathbf{2^{n-1}}$ different ways, since we have an independent option of cutting/not cutting at distance i from the left end, for i=1,2, …, n-1.
- (P362) The overall optimal solution incorporates optimal solutions to the two related subproblems, maximizing revenue from each of those two pieces.
- (P362) We say that the rod-cutting problem exhibits optimal substructure: optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently.
- P362 More generally, we can frame the values r_n for n ≥ 1 in terms of optimal revenues from shorter rods:

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}+, \dots, r_{n-1} + r_1)$$

- Read circled paragraph in P362
- Read circled paragraph in P365
- P367 When we think about a dynamic-programming problem, we should understand the set of subproblems involved and how subproblems depend on one another.
- P367 subproblem graph: the vertex labels give the sizes of the corresponding problems. A directed edge (x,y) indicates that we need a solution to subproblem y when solving subproblem x.
- Subproblem is a directed graph.
- P368 We can think of the subproblem graph as a “reduced” version of the recursion tree for the top-down recursive method, in which we coalesce all nodes for the same subproblem into a single vertex and direct all edges from parent to child.
- Bottom-up method: no subproblem is considered until all of the subproblems it depends upon have been solved.

- P368 We can view the top-down method with memoization for dynamic programming as a depth-first search of the subproblem graph.
- The size of the subproblem graph G = (V,E) can help us determine the running time of the dynamic programming algorithm. The running time of dynamic programming is linear in the number of vertices and edges.
- Reconstructing a solution: we can extend the dynamic-programming approach to record not only the optimal value computed for each subproblem, but also a choice that led to the optimal value.
- See P369 PRINT-CUT-ROD-SOLUTION

15.2 Matrix-chain multiplication

- P371 We can multiply two matrices A and B only if they are **compatible**: the number of columns of A must equal the number of rows of B.
- Goal: to minimize the number of required scalar multiplication
- We express costs of matrix-chain multiplication in terms of the number of scalar multiplications.
- P371 circled paragraph: definition of the **matrix-chain multiplication problem**
- P374 read circled paragraph
- P375 Hallmark of dynamic programing:
  1. optimal substructure
  2. overlapping subproblems

- P375 read circled paragraph and MATRIX-CHAIN-ORDER
- P377 PRINT-OPTIMAL-PARENS

15.3 Elements of dynamic programming

- P379 A problem exhibits **optimal substructure** if an optimal solution to the problem contains within it optimal solutions to subproblems.
- Consequently, we must take care to ensure that the range of subproblems we consider includes those used in an optimal solution.
- P380
  Optimal substructure varies across problem domains in two ways:
  1. how many subproblems an optimal solution to the original problem uses
  2. how many choices we have in determining which subproblem(s) to use in an optimal solution

- In the rod-cutting problem, an optimal solution for cutting up a rod of size n uses just one subproblem (of size n - i) , but we must consider n choices for i in order to determine which one yields an optimal solution.

- P380 Informally, the running time of a dynamic-programming algorithm depends on the product of two factors: the number of subproblems overall and how many choices we look at for each subproblem.
- **Running time of rod-cutting: O(n^2) -> θ(n) subproblems and at most n choices to examine for each**
  Running time of matrix-chain multiplication: O(n^3) -> θ(n^2) subproblems, we had at most n-1 choices in each

- Dynamic programming vs. greedy algorithm
  Common: optimal substructure
  Difference: instead of first finding optimal solutions to subproblems and then making an informed choice, greedy algorithms first make a greedy choice - the choice that looks best at the time - and then solve a resulting subproblem, without bothering to solve all possible related smaller subproblems.

- P382 shortest simple path: optimal substructure
  Longest simple path: no optimal substructure

- P383 Why is the substructure of a longest simple path so different from that of a shortest path? Although a solution to a problem for both longest and shortest paths uses two subproblems, the subproblems in finding the longest simple path are not independent, whereas for shortest paths they are. Independent means that the solution to one subproblem doesn't affect the solution to another subproblem of the same problem
- P383 Looked at another way, using resources in solving one subproblem (those resources being vertices) renders them unavailable for the other subproblem.
- P386 Whenever a recursion tree for the natural recursive solution to a problem contains the same subproblem repeatedly, and the total number of distinct subproblems is small, dynamic programming can improve efficiency, sometimes dramatically.
- P387 As a practical matter, we often store which choice we made in each subproblem in a table so that we do not have to reconstruct this information from the costs that we stored.

Memoization

- P387 Memoization: As in the bottom-up approach, we maintain a table with subproblem solutions, but the control structure for filling in the table is more like the recursive algorithm.
- Each table entry initially contains a special value to indicate that the entry has yet to be filled in. When the subproblem is first encountered as the recursive algorithm unfolds, its solution is computed and then stored in the table.
- P389 In summary, we can solve the matrix-chain multiplication problem by either a top-down, memorized dynamic-programming algorithm or a bottom-up dynamic-programming algorithm in O(n^3) time.
- P389 read circled paragraph

15.4 Longest common subsequence

- P391 A subsequence of a given sequence is just the given sequence with zero or more elements left out.
- P391 read circled paragraph for longest-common-subsequence problem definition
- P392 theorem 15.1 (optimal substructure of an LCS) and circled paragraph
- The way that Theorem 15.1 characterizes longest common subsequences tells us that an LCS of two sequences contains within it an LCS of prefixes of the two sequences.
- P393 read circled paragraph
- Finding an LCS : rules out subproblems based on conditions in the problem
  Rod-cutting & matrix-chain multiplication: do not rule out subproblems due to conditions in the problem

- LCS has θ(mn) distinct subproblems (m is the length of sequence Y, n is the length of sequence X), each subproblem (i.e. each table entry) takes O(1), so the running time of the algorithm LCS-LENGTH is θ(mn).
- P394 LCS-LENGTH
- P395 PRINT-LCS
- Overlapping subproblems: A recursive solution contains a “small” number of distinct subproblems repeated many times.
- Top-down approach of LCS-LENGTH algorithm with memoization