Number of bit strings
of length n with no
two consecutive 0s:

End with a 1:	Any bit string of length $n-1$ with no two consecutive 0s	1	a_{n-1}
End with a 0:	Any bit string of length $n-2$ with no two consecutive 0s	1 0	a_{n-2}

Total: $a_n = a_{n-1} + a_{n-2}$

FIGURE 4 **Counting bit strings of length n with no two consecutive 0s.**

Remark: Many people have studied variations of the original Tower of Hanoi puzzle discussed in Example 2. Some variations use more pegs, some allow disks to be of the same size, and some restrict the types of allowable disk moves. One of the oldest and most interesting variations is the **Reve's puzzle**,[*] proposed in 1907 by Henry Dudeney in his book *The Canterbury Puzzles*. The Reve's puzzle involves pilgrims challenged by the Reve to move a stack of cheese wheels of varying sizes from the first of four stools to another stool without ever placing a cheese wheel on one of smaller diameter. The Reve's puzzle, expressed in terms of pegs and disks, follows the same rules as the Tower of Hanoi puzzle, except that four pegs are used. Similarly, we can generalize the Tower of Hanoi puzzle where there are p pegs, where p is an integer greater than three. You may find it surprising that no one has been able to establish the minimum number of moves required to solve the generalization of this puzzle for p pegs. (Note that there have been some published claims that this problem has been solved, but these are not accepted by experts.) However, in 2014 Thierry Bousch showed that the minimum number of moves required when there are four pegs equals the number of moves used by an algorithm invented by Frame and Stewart in 1939. (See Exercises 38–45 and [St94] and [Bo14] for more information.)

Example 3 illustrates how recurrence relations can be used to count bit strings of a specified length that have a certain property.

EXAMPLE 3 Find a recurrence relation and give initial conditions for the number of bit strings of length n that do not have two consecutive 0s. How many such bit strings are there of length five?

Solution: Let a_n denote the number of bit strings of length n that do not have two consecutive 0s. We assume that $n \geq 3$, so that the bit string has at least three bits. Strings of this sort of length n can be divided into those that end in 1 and those that end in 0. The bit strings of length n ending with 1 that do not have two consecutive 0s are precisely the bit strings of length $n-1$ with no two consecutive 0s with a 1 added at the end. Consequently, there are a_{n-1} such bit strings.

Bit strings of length n ending with a 0 that do not have two consecutive 0s must have 1 as their $(n-1)$st bit; otherwise they would end with a pair of 0s. Hence, the bit strings of length n ending with a 0 that have no two consecutive 0s are precisely the bit strings of length $n-2$ with no two consecutive 0s with 10 added at the end. Consequently, there are a_{n-2} such bit strings.

We conclude, as illustrated in Figure 4, that

$$a_n = a_{n-1} + a_{n-2}$$

for $n \geq 3$.

_____

[*]*Reve*, more commonly spelled *reeve*, is an archaic word for *governor*.

The initial conditions are $a_1 = 2$, because both bit strings of length one, 0 and 1 do not have consecutive 0s, and $a_2 = 3$, because the valid bit strings of length two are 01, 10, and 11. To obtain a_5, we use the recurrence relation three times to find that

$$a_3 = a_2 + a_1 = 3 + 2 = 5,$$
$$a_4 = a_3 + a_2 = 5 + 3 = 8,$$
$$a_5 = a_4 + a_3 = 8 + 5 = 13.$$

◀

Remark: Note that $\{a_n\}$ satisfies the same recurrence relation as the Fibonacci sequence. Because $a_1 = f_3$ and $a_2 = f_4$ it follows that $a_n = f_{n+2}$.

Example 4 shows how a recurrence relation can be used to model the number of codewords that are allowable using certain validity checks.

EXAMPLE 4   **Codeword Enumeration** A computer system considers a string of decimal digits a valid codeword if it contains an even number of 0 digits. For instance, 1230407869 is valid, whereas 120987045608 is not valid. Let a_n be the number of valid n-digit codewords. Find a recurrence relation for a_n.

Solution: Note that $a_1 = 9$ because there are 10 one-digit strings, and only one, namely, the string 0, is not valid. A recurrence relation can be derived for this sequence by considering how a valid n-digit string can be obtained from strings of $n - 1$ digits. There are two ways to form a valid string with n digits from a string with one fewer digit.

First, a valid string of n digits can be obtained by appending a valid string of $n - 1$ digits with a digit other than 0. This appending can be done in nine ways. Hence, a valid string with n digits can be formed in this manner in $9a_{n-1}$ ways.

Second, a valid string of n digits can be obtained by appending a 0 to a string of length $n - 1$ that is not valid. (This produces a string with an even number of 0 digits because the invalid string of length $n - 1$ has an odd number of 0 digits.) The number of ways that this can be done equals the number of invalid $(n - 1)$-digit strings. Because there are 10^{n-1} strings of length $n - 1$, and a_{n-1} are valid, there are $10^{n-1} - a_{n-1}$ valid n-digit strings obtained by appending an invalid string of length $n - 1$ with a 0.

Because all valid strings of length n are produced in one of these two ways, it follows that there are

$$a_n = 9a_{n-1} + (10^{n-1} - a_{n-1})$$
$$= 8a_{n-1} + 10^{n-1}$$

$$a_n = 9a_{n-1} + 10^{n-1} - a_{n-1}$$

valid strings of length n.

◀

Example 5 establishes a recurrence relation that appears in many different contexts.

EXAMPLE 5   Find a recurrence relation for C_n, the number of ways to parenthesize the product of $n + 1$ numbers, $x_0 \cdot x_1 \cdot x_2 \cdot \cdots \cdot x_n$, to specify the order of multiplication. For example, $C_3 = 5$ because there are five ways to parenthesize $x_0 \cdot x_1 \cdot x_2 \cdot x_3$ to determine the order of multiplication:

$$((x_0 \cdot x_1) \cdot x_2) \cdot x_3 \qquad (x_0 \cdot (x_1 \cdot x_2)) \cdot x_3 \qquad (x_0 \cdot x_1) \cdot (x_2 \cdot x_3)$$
$$x_0 \cdot ((x_1 \cdot x_2) \cdot x_3) \qquad x_0 \cdot (x_1 \cdot (x_2 \cdot x_3)).$$

Solution: To develop a recurrence relation for C_n, we note that however we insert parentheses in the product $x_0 \cdot x_1 \cdot x_2 \cdot \cdots \cdot x_n$, one "·" operator remains outside all parentheses, namely, the operator for the final multiplication to be performed. [For example, in $(x_0 \cdot (x_1 \cdot x_2)) \cdot x_3$, it is the final "·", while in $(x_0 \cdot x_1) \cdot (x_2 \cdot x_3)$ it is the second "·".] This final operator appears between two of the $n+1$ numbers, say, x_k and x_{k+1}. There are $C_k C_{n-k-1}$ ways to insert parentheses to determine the order of the $n+1$ numbers to be multiplied when the final operator appears between x_k and x_{k+1}, because there are C_k ways to insert parentheses in the product $x_0 \cdot x_1 \cdot \cdots \cdot x_k$ to determine the order in which these $k+1$ numbers are to be multiplied and C_{n-k-1} ways to insert parentheses in the product $x_{k+1} \cdot x_{k+2} \cdot \cdots \cdot x_n$ to determine the order in which these $n-k$ numbers are to be multiplied. Because this final operator can appear between any two of the $n+1$ numbers, it follows that

$$C_n = C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-2} C_1 + C_{n-1} C_0$$

$$= \sum_{k=0}^{n-1} C_k C_{n-k-1}.$$

Note that the initial conditions are $C_0 = 1$ and $C_1 = 1$. ◀

The recurrence relation in Example 5 can be solved using the method of generating functions, which will be discussed in Section 8.4. It can be shown that $C_n = C(2n, n)/(n+1)$ (see Exercise 43 in Section 8.4) and that $C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$ (see [GrKnPa94]). The sequence $\{C_n\}$ is the

Links ❯

sequence of **Catalan numbers**, named after Eugène Charles Catalan. This sequence appears as the solution of many different counting problems besides the one considered here (see the chapter on Catalan numbers in [MiRo91] or [RoTe03] for details).

8.1.3 Algorithms and Recurrence Relations

Recurrence relations play an important role in many aspects of the study of algorithms and their complexity. In Section 8.3, we will show how recurrence relations can be used to analyze the complexity of divide-and-conquer algorithms, such as the merge sort algorithm introduced in Section 5.4. As we will see in Section 8.3, divide-and-conquer algorithms recursively divide a problem into a fixed number of nonoverlapping subproblems until they become simple enough to solve directly. We conclude this section by introducing another algorithmic paradigm known as **dynamic programming**, which can be used to solve many optimization problems efficiently.

Links ❯

An algorithm follows the dynamic programming paradigm when it recursively breaks down a problem into simpler overlapping subproblems, and computes the solution using the solutions of the subproblems. Generally, recurrence relations are used to find the overall solution from the solutions of the subproblems. Dynamic programming has been used to solve important problems in such diverse areas as economics, computer vision, speech recognition, artificial intelligence, computer graphics, and bioinformatics. In this section we will illustrate the use of dynamic programming by constructing an algorithm for solving a scheduling problem. Before doing so, we will relate the amusing origin of the name *dynamic programming*, which was introduced by the mathematician Richard Bellman in the 1950s. Bellman was working at the RAND Corporation on projects for the U.S. military, and at that time, the U.S. Secretary of Defense was hostile to mathematical research. Bellman decided that to ensure funding, he needed a name not containing the word mathematics for his method for solving scheduling and planning problems. He decided to use the adjective *dynamic* because, as he said "it's impossible to use the word dynamic in a pejorative sense" and he thought that dynamic programming was "something not even a Congressman could object to."

AN EXAMPLE OF DYNAMIC PROGRAMMING   The problem we use to illustrate dynamic programming is related to the problem studied in Example 7 in Section 3.1. In that problem our goal was to schedule as many talks as possible in a single lecture hall. These talks have preset start and end times; once a talk starts, it continues until it ends; no two talks can proceed at the same time; and a talk can begin at the same time another one ends. We developed a greedy algorithm that always produces an optimal schedule, as we proved in Example 12 in Section 5.1. Now suppose that our goal is not to schedule the most talks possible, but rather to have the largest possible combined attendance of the scheduled talks.

We formalize this problem by supposing that we have n talks, where talk j begins at time t_j, ends at time e_j, and will be attended by w_j students. We want a schedule that maximizes the total number of student attendees. That is, we wish to schedule a subset of talks to maximize the sum of w_j over all scheduled talks. (Note that when a student attends more than one talk, this student is counted according to the number of talks attended.) We denote by $T(j)$ the maximum number of total attendees for an optimal schedule from the first j talks, so $T(n)$ is the maximal number of total attendees for an optimal schedule for all n talks.

We first sort the talks in order of increasing end time. After doing this, we renumber the talks so that $e_1 \le e_2 \le \cdots \le e_n$. We say that two talks are **compatible** if they can be part of the same schedule, that is, if the times they are scheduled do not overlap (other than the possibility one ends and the other starts at the same time). We define $p(j)$ to be largest integer i, $i < j$, for which $e_i \le s_j$, if such an integer exists, and $p(j) = 0$ otherwise. That is, talk $p(j)$ is the talk ending latest among talks compatible with talk j that end before talk j ends, if such a talk exists, and $p(j) = 0$ if there are no such talks.

EXAMPLE 6   Consider seven talks with these start times and end times, as illustrated in Figure 5.

Talk 1: start 8 A.M., end 10 A.M.   Talk 5: start 8:30 A.M., end 2 P.M.
Talk 2: start 9 A.M., end 11 A.M.   Talk 6: start 11 A.M., end 2 P.M.
Talk 3: start 10:30 A.M., end 12 noon   Talk 7: start 1 P.M., end 2 P.M.
Talk 4: start 9:30 A.M., end 1 P.M.

Find $p(j)$ for $j = 1, 2, \ldots, 7$.

Solution: We have $p(1) = 0$ and $p(2) = 0$, because no talks end before either of the first two talks begin. We have $p(3) = 1$ because talk 3 and talk 1 are compatible, but talk 3 and talk 2 are not compatible; $p(4) = 0$ because talk 4 is not compatible with any of talks 1, 2, and 3; $p(5) = 0$

EUGÈNE CHARLES CATALAN (1814–1894)   Eugène Catalan was born in Bruges, then part of France. His father became a successful architect in Paris while Eugène was a boy. Catalan attended a Parisian school for design hoping to follow in his father's footsteps. At 15, he won the job of teaching geometry to his design school classmates. After graduating, Catalan attended a school for the fine arts, but because of his mathematical aptitude his instructors recommended that he enter the École Polytechnique. He became a student there, but after his first year, he was expelled because of his politics. However, he was readmitted, and in 1835, he graduated and won a position at the Collège de Châlons sur Marne.

In 1838, Catalan returned to Paris where he founded a preparatory school with two other mathematicians, Sturm and Liouville. After teaching there for a short time, he was appointed to a position at the École Polytechnique. He received his doctorate from the École Polytechnique in 1841, but his political activity in favor of the French Republic hurt his career prospects. In 1846 Catalan held a position at the Collège de Charlemagne; he was appointed to the Lycée Saint Louis in 1849. However, when Catalan would not take a required oath of allegiance to the new Emperor Louis-Napoleon Bonaparte, he lost his job. For 13 years he held no permanent position. Finally, in 1865 he was appointed to a chair of mathematics at the University of Liège, Belgium, a position he held until his 1884 retirement.

Catalan made many contributions to number theory and to the related subject of continued fractions. He defined what are now known as the Catalan numbers when he solved the problem of dissecting a polygon into triangles using non-intersecting diagonals. Catalan is also well known for formulating what was known as the *Catalan conjecture*. This asserted that 8 and 9 are the only consecutive powers of integers, a conjecture not solved until 2003. Catalan wrote many textbooks, including several that became quite popular and appeared in as many as 12 editions. Perhaps this textbook will have a 12th edition someday!
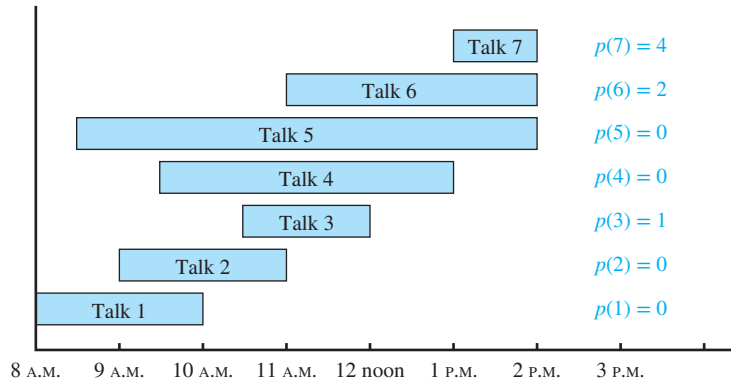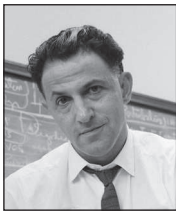
FIGURE 5   **A schedule of lectures with the values of $p(n)$ shown.**

because talk 5 is not compatible with any of talks 1, 2, 3, and 4; and $p(6) = 2$ because talk 6 and talk 2 are compatible, but talk 6 is not compatible with any of talks 3, 4, and 5. Finally, $p(7) = 4$, because talk 7 and talk 4 are compatible, but talk 7 is not compatible with either of talks 5 or 6.   ◀

To develop a dynamic programming algorithm for this problem, we first develop a key recurrence relation. To do this, first note that if $j \leq n$, there are two possibilities for an optimal schedule of the first j talks (recall that we are assuming that the n talks are ordered by increasing end time): *(i)* talk j belongs to the optimal schedule or *(ii)* it does not.

Case (i): We know that talks $p(j) + 1, \ldots, j - 1$ do not belong to this schedule, for none of these other talks are compatible with talk j. Furthermore, the other talks in this optimal schedule must comprise an optimal schedule for talks $1, 2, \ldots, p(j)$. For if there were a better schedule for talks

RICHARD BELLMAN (1920–1984)   Richard Bellman, born in Brooklyn, where his father was a grocer, spent many hours in the museums and libraries of New York as a child. After graduating high school, he studied mathematics at Brooklyn College and graduated in 1941. He began postgraduate work at Johns Hopkins University, but because of the war, left to teach electronics at the University of Wisconsin. He was able to continue his mathematics studies at Wisconsin, and in 1943 he received his masters degree there. Later, Bellman entered Princeton University, teaching in a special U.S. Army program. In late 1944, he was drafted into the army. He was assigned to the Manhattan Project at Los Alamos where he worked in theoretical physics. After the war, he returned to Princeton and received his Ph.D. in 1946.

After briefly teaching at Princeton, he moved to Stanford University, where he attained tenure. At Stanford he pursued his fascination with number theory. However, Bellman decided to focus on mathematical questions arising from real-world problems. In 1952, he joined the RAND Corporation, working on multistage decision processes, operations research problems, and applications to the social sciences and medicine. He worked on many military projects while at RAND. In 1965 he left RAND to become professor of mathematics, electrical and biomedical engineering and medicine at the University of Southern California.

In the 1950s Bellman pioneered the use of dynamic programming, a technique invented earlier, in a wide range of settings. He is also known for his work on stochastic control processes, in which he introduced what is now called the Bellman equation. He coined the term *curse of dimensionality* to describe problems caused by the exponential increase in volume associated with adding extra dimensions to a space. He wrote an amazing number of books and research papers with many coauthors, including many on industrial production and economic systems. His work led to the application of computing techniques in a wide variety of areas ranging from the design of guidance systems for space vehicles, to network optimization, and even to pest control.

Tragically, in 1973 Bellman was diagnosed with a brain tumor. Although it was removed successfully, complications left him severely disabled. Fortunately, he managed to continue his research and writing during his remaining ten years of life. Bellman received many prizes and awards, including the first Norbert Wiener Prize in Applied Mathematics and the IEEE Gold Medal of Honor. He was elected to the National Academy of Sciences. He was held in high regard for his achievements, courage, and admirable qualities. Bellman was the father of two children.

1, 2, …, $p(j)$, by adding talk j, we will have a schedule better than the overall optimal schedule. Consequently, in case *(i)*, we have $T(j) = w_j + T(p(j))$.

Case (ii): When talk j does not belong to an optimal schedule, it follows that an optimal schedule from talks 1, 2, …, j is the same as an optimal schedule from talks 1, 2, …, $j - 1$. Consequently, in case *(ii)*, we have $T(j) = T(j - 1)$. Combining cases *(i)* and *(ii)* leads us to the recurrence relation

$$T(j) = \max(w_j + T(p(j)), T(j - 1)).$$

Now that we have developed this recurrence relation, we can construct an efficient algorithm, Algorithm 1, for computing the maximum total number of attendees. We ensure that the algorithm is efficient by storing the value of each $T(j)$ after we compute it. This allows us to compute $T(j)$ only once. If we did not do this, the algorithm would have exponential worst-case complexity. The process of storing the values as each is computed is known as **memoization** and is an important technique for making recursive algorithms efficient.

ALGORITHM 1 Dynamic Programming Algorithm for Scheduling Talks.

procedure *Maximum Attendees* $(s_1, s_2, …, s_n$: start times of talks;
  $e_1, e_2, …, e_n$: end times of talks; $w_1, w_2, …, w_n$: number of attendees to talks)
  sort talks by end time and relabel so that $e_1 \leq e_2 \leq \cdots \leq e_n$
for $j := 1$ **to** n
    if no job i with $i < j$ is compatible with job j
        $p(j) = 0$
    else $p(j) := \max\{i - i < j$ and job i is compatible with job $j\}$
    $T(0) := 0$
for $j := 1$ **to** n
    $T(j) := \max(w_j + T(p(j)), T(j - 1))$
return $T(n)\{T(n)$ is the maximum number of attendees$\}$

In Algorithm 1 we determine the maximum number of attendees that can be achieved by a schedule of talks, but we do not find a schedule that achieves this maximum. To find talks we need to schedule, we use the fact that talk j belongs to an optimal solution for the first j talks if and only if $w_j + T(p(j)) \geq T(j - 1)$. We leave it as Exercise 53 to construct an algorithm based on this observation that determines which talks should be scheduled to achieve the maximum total number of attendees.

Algorithm 1 is a good example of dynamic programming as the maximum total attendance is found using the optimal solutions of the overlapping subproblems, each of which determines the maximum total attendance of the first j talks for some j with $1 \leq j \leq n - 1$. See Exercises 56 and 57 and Supplementary Exercises 14 and 17 for other examples of dynamic programming.

Exercises

1. Use mathematical induction to verify the formula derived in Example 2 for the number of moves required to complete the Tower of Hanoi puzzle.

2. a) Find a recurrence relation for the number of permutations of a set with n elements.
  b) Use this recurrence relation to find the number of permutations of a set with n elements using iteration.

3. A vending machine dispensing books of stamps accepts only one-dollar coins, $1 bills, and $5 bills.
  a) Find a recurrence relation for the number of ways to deposit n dollars in the vending machine, where the order in which the coins and bills are deposited matters.

$a_n = 2a_{n-1} + a_{n-5}$

b) What are the initial conditions?

c) How many ways are there to deposit $10 for a book of stamps?

4. A country uses as currency coins with values of 1 peso, 2 pesos, 5 pesos, and 10 pesos and bills with values of 5 pesos, 10 pesos, 20 pesos, 50 pesos, and 100 pesos. Find a recurrence relation for the number of ways to pay a bill of n pesos if the order in which the coins and bills are paid matters.

5. How many ways are there to pay a bill of 17 pesos using the currency described in Exercise 4, where the order in which coins and bills are paid matters?

*6. a) Find a recurrence relation for the number of strictly increasing sequences of positive integers that have 1 as their first term and n as their last term, where n is a positive integer. That is, sequences $a_1, a_2, \ldots, a_k$, where $a_1 = 1$, $a_k = n$, and $a_j < a_{j+1}$ for $j = 1, 2, \ldots, k-1$.

b) What are the initial conditions?

c) How many sequences of the type described in (a) are there when n is an integer with $n \geq 2$?

7. a) Find a recurrence relation for the number of bit strings of length n that contain a pair of consecutive 0s.

b) What are the initial conditions?

c) How many bit strings of length seven contain two consecutive 0s?

8. a) Find a recurrence relation for the number of bit strings of length n that contain three consecutive 0s.

b) What are the initial conditions?

c) How many bit strings of length seven contain three consecutive 0s?

9. a) Find a recurrence relation for the number of bit strings of length n that do not contain three consecutive 0s.

b) What are the initial conditions?

c) How many bit strings of length seven do not contain three consecutive 0s?

*10. a) Find a recurrence relation for the number of bit strings of length n that contain the string 01.

b) What are the initial conditions?

c) How many bit strings of length seven contain the string 01?

11. a) Find a recurrence relation for the number of ways to climb n stairs if the person climbing the stairs can take one stair or two stairs at a time.

b) What are the initial conditions?

c) In how many ways can this person climb a flight of eight stairs?

12. a) Find a recurrence relation for the number of ways to climb n stairs if the person climbing the stairs can take one, two, or three stairs at a time.

b) What are the initial conditions?

c) In how many ways can this person climb a flight of eight stairs?

A string that contains only 0s, 1s, and 2s is called a **ternary string**.

13. a) Find a recurrence relation for the number of ternary strings of length n that do not contain two consecutive 0s.

b) What are the initial conditions?

c) How many ternary strings of length six do not contain two consecutive 0s?

14. a) Find a recurrence relation for the number of ternary strings of length n that contain two consecutive 0s.

b) What are the initial conditions?

c) How many ternary strings of length six contain two consecutive 0s?

*15. a) Find a recurrence relation for the number of ternary strings of length n that do not contain two consecutive 0s or two consecutive 1s.

b) What are the initial conditions?

c) How many ternary strings of length six do not contain two consecutive 0s or two consecutive 1s?

*16. a) Find a recurrence relation for the number of ternary strings of length n that contain either two consecutive 0s or two consecutive 1s.

b) What are the initial conditions?

c) How many ternary strings of length six contain two consecutive 0s or two consecutive 1s?

*17. a) Find a recurrence relation for the number of ternary strings of length n that do not contain consecutive symbols that are the same.

b) What are the initial conditions?

c) How many ternary strings of length six do not contain consecutive symbols that are the same?

**18. a) Find a recurrence relation for the number of ternary strings of length n that contain two consecutive symbols that are the same.

b) What are the initial conditions?

c) How many ternary strings of length six contain consecutive symbols that are the same?

19. Messages are transmitted over a communications channel using two signals. The transmittal of one signal requires 1 microsecond, and the transmittal of the other signal requires 2 microseconds.

a) Find a recurrence relation for the number of different messages consisting of sequences of these two signals, where each signal in the message is immediately followed by the next signal, that can be sent in n microseconds.

b) What are the initial conditions?

c) How many different messages can be sent in 10 microseconds using these two signals?

20. A bus driver pays all tolls, using only nickels and dimes, by throwing one coin at a time into the mechanical toll collector.

a) Find a recurrence relation for the number of different ways the bus driver can pay a toll of n cents (where the order in which the coins are used matters).

b) In how many different ways can the driver pay a toll of 45 cents?

21. a) Find the recurrence relation satisfied by R_n, where R_n is the number of regions that a plane is divided into by n lines, if no two of the lines are parallel and no three of the lines go through the same point.

b) Find R_n using iteration.

*22. a) Find the recurrence relation satisfied by R_n, where R_n is the number of regions into which the surface of a sphere is divided by n great circles (which are the intersections of the sphere and planes passing through the center of the sphere), if no three of the great circles go through the same point.

   b) Find R_n using iteration.

*23. a) Find the recurrence relation satisfied by S_n, where S_n is the number of regions into which three-dimensional space is divided by n planes if every three of the planes meet in one point, but no four of the planes go through the same point.

   b) Find S_n using iteration.

24. Find a recurrence relation for the number of bit sequences of length n with an even number of 0s.

25. How many bit sequences of length seven contain an even number of 0s?

26. a) Find a recurrence relation for the number of ways to completely cover a $2 \times n$ checkerboard with 1×2 dominoes. [*Hint:* Consider separately the coverings where the position in the top right corner of the checkerboard is covered by a domino positioned horizontally and where it is covered by a domino positioned vertically.]

   b) What are the initial conditions for the recurrence relation in part (a)?

   c) How many ways are there to completely cover a 2×17 checkerboard with 1×2 dominoes?

27. a) Find a recurrence relation for the number of ways to lay out a walkway with slate tiles if the tiles are red, green, or gray, so that no two red tiles are adjacent and tiles of the same color are considered indistinguishable.

   b) What are the initial conditions for the recurrence relation in part (a)?

   c) How many ways are there to lay out a path of seven tiles as described in part (a)?

28. Show that the Fibonacci numbers satisfy the recurrence relation $f_n = 5f_{n-4} + 3f_{n-5}$ for $n = 5, 6, 7, \ldots,$ together with the initial conditions $f_0 = 0, f_1 = 1, f_2 = 1, f_3 = 2,$ and $f_4 = 3$. Use this recurrence relation to show that f_{5n} is divisible by 5, for $n = 1, 2, 3, \ldots$.

*29. Let $S(m, n)$ denote the number of onto functions from a set with m elements to a set with n elements. Show that $S(m, n)$ satisfies the recurrence relation

$$S(m, n) = n^m - \sum_{k=1}^{n-1} C(n, k)S(m, k)$$

whenever $m \geq n$ and $n > 1$, with the initial condition $S(m, 1) = 1$.

30. a) Write out all the ways the product $x_0 \cdot x_1 \cdot x_2 \cdot x_3 \cdot x_4$ can be parenthesized to determine the order of multiplication.

   b) Use the recurrence relation developed in Example 5 to calculate C_4, the number of ways to parenthesize the product of five numbers so as to determine the order of multiplication. Verify that you listed the correct number of ways in part (a).

   c) Check your result in part (b) by finding C_4, using the closed formula for C_n mentioned in the solution of Example 5.

31. a) Use the recurrence relation developed in Example 5 to determine C_5, the number of ways to parenthesize the product of six numbers so as to determine the order of multiplication.

   b) Check your result with the closed formula for C_5 mentioned in the solution of Example 5.

*32. In the Tower of Hanoi puzzle, suppose our goal is to transfer all n disks from peg 1 to peg 3, but we cannot move a disk directly between pegs 1 and 3. Each move of a disk must be a move involving peg 2. As usual, we cannot place a disk on top of a smaller disk.

   a) Find a recurrence relation for the number of moves required to solve the puzzle for n disks with this added restriction.

   b) Solve this recurrence relation to find a formula for the number of moves required to solve the puzzle for n disks.

   c) How many different arrangements are there of the n disks on three pegs so that no disk is on top of a smaller disk?

   d) Show that every allowable arrangement of the n disks occurs in the solution of this variation of the puzzle.

Exercises 33–37 deal with a variation of the **Josephus problem** described by Graham, Knuth, and Patashnik in [GrKnPa94]. This problem is based on an account by the historian Flavius Josephus, who was part of a band of 41 Jewish rebels trapped in a cave by the Romans during the Jewish-Roman war of the first century. The rebels preferred suicide to capture; they decided to form a circle and to repeatedly count off around the circle, killing every third rebel left alive. However, Josephus and another rebel did not want to be killed this way; they determined the positions where they should stand to be the last two rebels remaining alive. The variation we consider begins with n people, numbered 1 to n, standing around a circle. In each stage, every second person still left alive is eliminated until only one survives. We denote the number of the survivor by $J(n)$.

33. Determine the value of $J(n)$ for each integer n with $1 \leq n \leq 16$.

34. Use the values you found in Exercise 33 to conjecture a formula for $J(n)$. [*Hint:* Write $n = 2^m + k$, where m is a nonnegative integer and k is a nonnegative integer less than 2^m.]

35. Show that $J(n)$ satisfies the recurrence relation $J(2n) = 2J(n) - 1$ and $J(2n + 1) = 2J(n) + 1$, for $n \geq 1$, and $J(1) = 1$.

36. Use mathematical induction to prove the formula you conjectured in Exercise 34, making use of the recurrence relation from Exercise 35.

37. Determine $J(100)$, $J(1000)$, and $J(10,000)$ from your formula for $J(n)$.

Exercises 38–45 involve the Reve's puzzle, the variation of the Tower of Hanoi puzzle with four pegs and n disks. Before presenting these exercises, we describe the Frame–Stewart algorithm for moving the disks from peg 1 to peg 4 so that no disk is ever on top of a smaller one. This algorithm, given the number of disks n as input, depends on a choice of an integer k with $1 \le k \le n$. When there is only one disk, move it from peg 1 to peg 4 and stop. For $n > 1$, the algorithm proceeds recursively, using these three steps. Recursively move the stack of the $n - k$ smallest disks from peg 1 to peg 2, using all four pegs. Next move the stack of the k largest disks from peg 1 to peg 4, using the three-peg algorithm from the Tower of Hanoi puzzle without using the peg holding the $n - k$ smallest disks. Finally, recursively move the smallest $n - k$ disks to peg 4, using all four pegs. Frame and Stewart showed that to produce the fewest moves using their algorithm, k should be chosen to be the smallest integer such that n does not exceed $t_k = k(k + 1)/2$, the kth triangular number, that is, $t_{k-1} < n \le t_k$. The long-standing conjecture, known as **Frame's conjecture**, that this algorithm uses the fewest number of moves required to solve the puzzle, was proved by Thierry Bousch in 2014.

38. Show that the Reve's puzzle with three disks can be solved using five, and no fewer, moves.

39. Show that the Reve's puzzle with four disks can be solved using nine, and no fewer, moves.

40. Describe the moves made by the Frame–Stewart algorithm, with k chosen so that the fewest moves are required, for
   a) 5 disks.   **b)** 6 disks.   **c)** 7 disks.   **d)** 8 disks.

*41. Show that if $R(n)$ is the number of moves used by the Frame–Stewart algorithm to solve the Reve's puzzle with n disks, where k is chosen to be the smallest integer with $n \le k(k + 1)/2$, then $R(n)$ satisfies the recurrence relation $R(n) = 2R(n - k) + 2^k - 1$, with $R(0) = 0$ and $R(1) = 1$.

*42. Show that if k is as chosen in Exercise 41, then $R(n) - R(n - 1) = 2^{k-1}$.

*43. Show that if k is as chosen in Exercise 41, then $R(n) = \sum_{i=1}^{k} i2^{i-1} - (t_k - n)2^{k-1}$.

*44. Use Exercise 43 to give an upper bound on the number of moves required to solve the Reve's puzzle for all integers n with $1 \le n \le 25$.

*45. Show that $R(n)$ is $O(\sqrt{n}2^{\sqrt{2n}})$.

Let $\{a_n\}$ be a sequence of real numbers. The **backward differences** of this sequence are defined recursively as shown next. The **first difference** ∇a_n is

$$\nabla a_n = a_n - a_{n-1}.$$

The **$(k + 1)$st difference** $\nabla^{k+1}a_n$ is obtained from $\nabla^k a_n$ by

$$\nabla^{k+1}a_n = \nabla^k a_n - \nabla^k a_{n-1}.$$

46. Find ∇a_n for the sequence $\{a_n\}$, where
   a) $a_n = 4$.   **b)** $a_n = 2n$.
   c) $a_n = n^2$.   **d)** $a_n = 2^n$.

47. Find $\nabla^2 a_n$ for the sequences in Exercise 46.

48. Show that $a_{n-1} = a_n - \nabla a_n$.

49. Show that $a_{n-2} = a_n - 2\nabla a_n + \nabla^2 a_n$.

*50. Prove that a_{n-k} can be expressed in terms of a_n, ∇a_n, $\nabla^2 a_n$, ..., $\nabla^k a_n$.

51. Express the recurrence relation $a_n = a_{n-1} + a_{n-2}$ in terms of a_n, ∇a_n, and $\nabla^2 a_n$.

52. Show that any recurrence relation for the sequence $\{a_n\}$ can be written in terms of a_n, ∇a_n, $\nabla^2 a_n$, ... The resulting equation involving the sequences and its differences is called a **difference equation**.

*53. Construct the algorithm described in the text after Algorithm 1 for determining which talks should be scheduled to maximize the total number of attendees and not just the maximum total number of attendees determined by Algorithm 1.

54. Use Algorithm 1 to determine the maximum number of total attendees in the talks in Example 6 if w_i, the number of attendees of talk i, $i = 1, 2, \ldots, 7$, is
   a) 20, 10, 50, 30, 15, 25, 40.
   b) 100, 5, 10, 20, 25, 40, 30.
   c) 2, 3, 8, 5, 4, 7, 10.
   d) 10, 8, 7, 25, 20, 30, 5.

55. For each part of Exercise 54, use your algorithm from Exercise 53 to find the optimal schedule for talks so that the total number of attendees is maximized.

56. In this exercise we will develop a dynamic programming algorithm for finding the maximum sum of consecutive terms of a sequence of real numbers. That is, given a sequence of real numbers $a_1, a_2, \ldots, a_n$, the algorithm computes the maximum sum $\sum_{i=j}^{k} a_i$ where $1 \le j \le k \le n$.
   a) Show that if all terms of the sequence are nonnegative, this problem is solved by taking the sum of all terms. Then, give an example where the maximum sum of consecutive terms is not the sum of all terms.
   b) Let $M(k)$ be the maximum of the sums of consecutive terms of the sequence ending at a_k. That is, $M(k) = \max_{1 \le j \le k} \sum_{i=j}^{k} a_i$. Explain why the recurrence relation $M(k) = \max(M(k - 1) + a_k, a_k)$ holds for $k = 2, \ldots, n$.
   c) Use part (b) to develop a dynamic programming algorithm for solving this problem.
   d) Show each step your algorithm from part (c) uses to find the maximum sum of consecutive terms of the sequence 2, −3, 4, 1, −2, 3.
   e) Show that the worst-case complexity in terms of the number of additions and comparisons of your algorithm from part (c) is linear.