Advanced Counting Techniques

Assylbek Issakhov,
Ph.D., professor

Advanced Counting Techniques

- Many counting problems cannot be solved easily using the methods discussed in previous lecture about combinatorics. One such problem is: How many bit strings of length n *do not contain two consecutive zero?* To solve this problem, let a_n *be the number of such strings of length* n. *An argument can* be given that shows that the sequence $\{a_n\}$ *satisfies the recurrence relation* $a_{n+1} = a_n + a_{n-1}$ and the initial conditions $a_1 = 2$ *and* $a_2 = 3$.

Advanced Counting Techniques

- This recurrence relation and the initial conditions determine the sequence $\{a_n\}$. *Moreover, a_n explicit formula can be found for a_n from the equation* relating the terms of the sequence. As we will see, a similar technique can be used to solve many different types of counting problems.

Advanced Counting Techniques

- We will discuss two ways that recurrence relations play important roles in the study of algorithms. First, we will introduce an important algorithmic paradigm known as dynamic programming. Algorithms that follow this paradigm break down a problem into overlapping subproblems. The solution to the problem is then found from the solutions to the subproblems through the use of a recurrence relation. Second, we will study another important algorithmic paradigm, divide-and-conquer.

Advanced Counting Techniques

- Algorithms that follow this paradigm can be used to solve a problem by recursively breaking it into a fixed number of nonoverlapping subproblems until these problems can be solved directly. The complexity of such algorithms can be analyzed using a special type of recurrence relation.

Advanced Counting Techniques

- Many other kinds of counting problems cannot be solved using the previous techniques, such as:
- How many ways are there to assign seven jobs to three employees so that each employee is assigned at least one job?
- How many primes are there less than 1000?

Advanced Counting Techniques

- Both of these problems can be solved by counting the number of elements in the union of sets. We will develop a technique, called the principle of inclusion-exclusion, that counts the number of elements in a union of sets, and we will show how this principle can be used to solve counting problems.

Applications of Recurrence Relations

- A recursive definition of a sequence specifies one or more initial terms and a rule for determining subsequent terms from those that precede them. Also, a rule of the latter sort (whether or not it is part of a recursive definition) is called a **recurrence relation** and that a sequence is called a *solution of a recurrence relation if its terms satisfy the* recurrence relation.

Applications of Recurrence Relations

- For example, suppose that the number of bacteria in a colony doubles every hour. If a colony begins with five bacteria, how many will be present in n *hours?*
- *To solve this problem,* let a_n *be the number of bacteria at the end of* n *hours. Because the number of bacteria doubles* every hour, the relationship $a_n = 2a_{n-1}$ *holds whenever* n *is a positive integer.*

Applications of Recurrence Relations

- This recurrence relation, together with the initial condition $a_0 = 5$, *uniquely determines* a_n *for all nonnegative* integers n. *We can find a formula for* a_n *using the iterative approach,* namely that

$$a_n = 5 \cdot 2^n$$

- *for all nonnegative integers* n.

Modeling With Recurrence Relations: Rabbits and the Fibonacci Numbers

- Consider the problem, which was originally posed by Leonardo Pisano, also known as Fibonacci, in the thirteenth century in his book *Liber abaci.*
- *A* young pair of rabbits (one of each sex) is placed on an island. A pair of rabbits does not breed until they are 2 months old. After they are 2 months old, each pair of rabbits produces another pair each month, as shown in the next Figure.

Modeling With Recurrence Relations: Rabbits and the Fibonacci Numbers

Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
		1	0	1	1
		2	0	1	1
		3	1	1	2
		4	1	2	3
		5	2	3	5
	 	6	3	5	8

Modeling With Recurrence Relations: Rabbits and the Fibonacci Numbers

- Find a recurrence relation for the number of pairs of rabbits on the island after n *months, assuming that no rabbits ever die.*
- *Solution: Denote by* f_n *the number of pairs of rabbits after* n *months. We will show that* f_n, $n = 1, 2, 3, \ldots$, *are the terms of the Fibonacci sequence.*

Modeling With Recurrence Relations: Rabbits and the Fibonacci Numbers

- *Solution (cont.):* The rabbit population can be modeled using a recurrence relation. At the end of the first month, the number of pairs of rabbits on the island is $f_1 = 1$. *Because this pair does not* breed during the second month, $f_2 = 1$ *also.* To find the number of pairs after n *months, add* the number on the island the previous month, f_{n-1}, *and the number of newborn pairs, which* equals f_{n-2}, *because each newborn pair comes from a pair at least 2 months old.*

Modeling With Recurrence Relations: Rabbits and the Fibonacci Numbers

- *Solution (cont.):* Consequently, the sequence $\{f_n\}$ *satisfies the recurrence relation*

$$f_n = f_{n-1} + f_{n-2}$$

for $n \geq 3$ *together with the initial conditions* $f_1 = 1$ *and* $f_2 = 1$. *Because this recurrence relation* and the initial conditions uniquely determine this sequence, the number of pairs of rabbits on the island after n *months is given by the* n*th Fibonacci number.*

Modeling With Recurrence Relations: The Tower of Hanoi

- A popular puzzle of the late nineteenth century invented by the French mathematician Édouard Lucas, called the Tower of Hanoi, consists of three pegs mounted on a board together with disks of different sizes. Initially these disks are placed on the first peg in order of size, with the largest on the bottom (as shown in Figure 2). The rules of the puzzle allow disks to be moved one at a time from one peg to another as long as a disk is never placed on top of a smaller disk.

Modeling With Recurrence Relations: The Tower of Hanoi

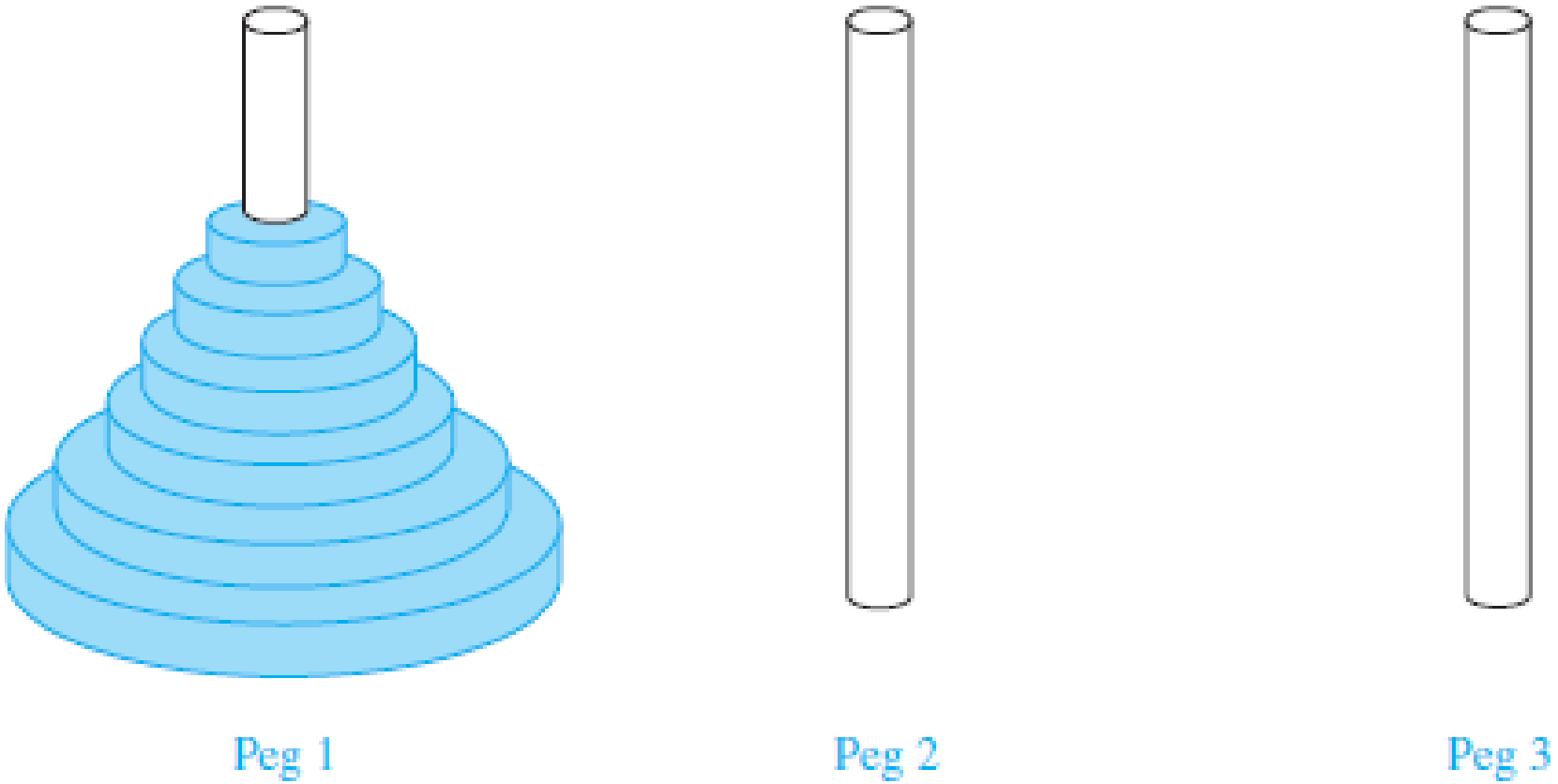


FIGURE 2 The Initial Position in the Tower of Hanoi.

Modeling With Recurrence Relations: The Tower of Hanoi

- The goal of the puzzle is to have all the disks on the second peg in order of size, with the largest on the bottom.
- Let H_n *denote the number of moves needed to solve the Tower of Hanoi problem with* n disks. Set up a recurrence relation for the sequence $\{H_n\}$.

Modeling With Recurrence Relations: The Tower of Hanoi

- *Solution: Begin with n disks on peg 1.We can transfer the top $n-1$ disks, following the rules* of the puzzle, to peg 3 using H_{n-1} *moves (see Figure 3 for an illustration of the pegs and disks* at this point). We keep the largest disk fixed during these moves. Then, we use one move to transfer the largest disk to the second peg. We can transfer the $n-1$ *disks on peg 3 to peg 2* using H_{n-1} *additional moves, placing them on top of the largest disk, which always stays fixed* on the bottom of peg 2.

Modeling With Recurrence Relations: The Tower of Hanoi

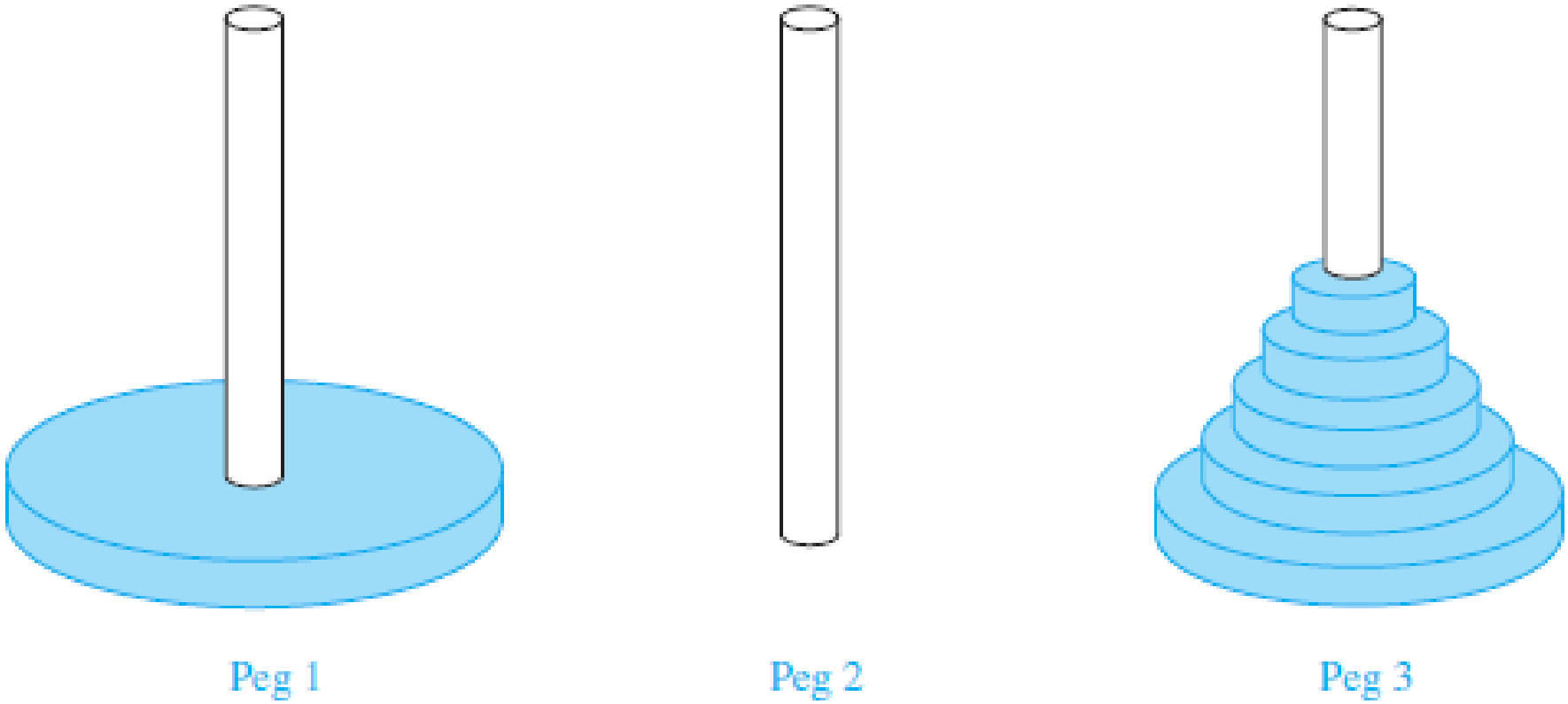


FIGURE 3 An Intermediate Position in the Tower of Hanoi.

Modeling With Recurrence Relations: The Tower of Hanoi

- Moreover, it is easy to see that the puzzle cannot be solved using fewer steps. This shows that

$$H_n = 2H_{n-1} + 1.$$

- The initial condition is $H_1 = 1$, *because one disk can be transferred from peg 1 to peg 2,* according to the rules of the puzzle, in one move.

Modeling With Recurrence Relations: The Tower of Hanoi

- We can use an iterative approach to solve this recurrence relation. Note that
- $H_n = 2H_{n-1} + 1 = 2(2H_{n-2} + 1)$
  $= 2^2 H_{n-2} + 1 = 2^2(2H_{n-3} + 1) + 2 + 1$
  $= 2^3 H_{n-3} + 2^2 + 2 + 1 = \ \ldots$
  $= 2^{n-1} H_1 + 2^{n-2} + 2^{n-3} + \ \ldots + 2 + 1$
  $= 2^{n-1} + 2^{n-2} + \ \ldots + 2 + 1 = 2^n - 1$

Modeling With Recurrence Relations: The Tower of Hanoi

- A myth created to accompany the puzzle tells of a tower in Hanoi where monks are transferring 64 gold disks from one peg to another, according to the rules of the puzzle. The myth says that the world will end when they finish the puzzle. How long after the monks started will the world end if the monks take one second to move a disk?

Modeling With Recurrence Relations: The Tower of Hanoi

- From the explicit formula, the monks require

$$2^{64} - 1 = 18{,}446{,}744{,}073{,}709{,}551{,}615$$

moves to transfer the disks. Making one move per second, it will take them more than 500 billion years to complete the transfer, so the world should survive a while longer than it already has.

Modeling With Recurrence Relations: Another Example

- 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?

Modeling With Recurrence Relations: Another Example

- *Solution: Let a_n denote the number of bit strings of length n that do not have two consecutive 0s.* To obtain a recurrence relation for $\{a_n\}$, *note that by the sum rule, the number of bit strings of* length n *that do not have two consecutive 0s equals the number of such bit strings ending with* a 0 plus the number of such bit strings ending with a 1.We will assume that $n \geq 3$, *so that the* bit string has at least three bits.

Modeling With Recurrence Relations: Another Example

- *Solution (cont.):* 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.*

Modeling With Recurrence Relations: Another Example

- *Solution (cont.):* It follows that 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 $n \geq 3$.

Modeling With Recurrence Relations: Another Example

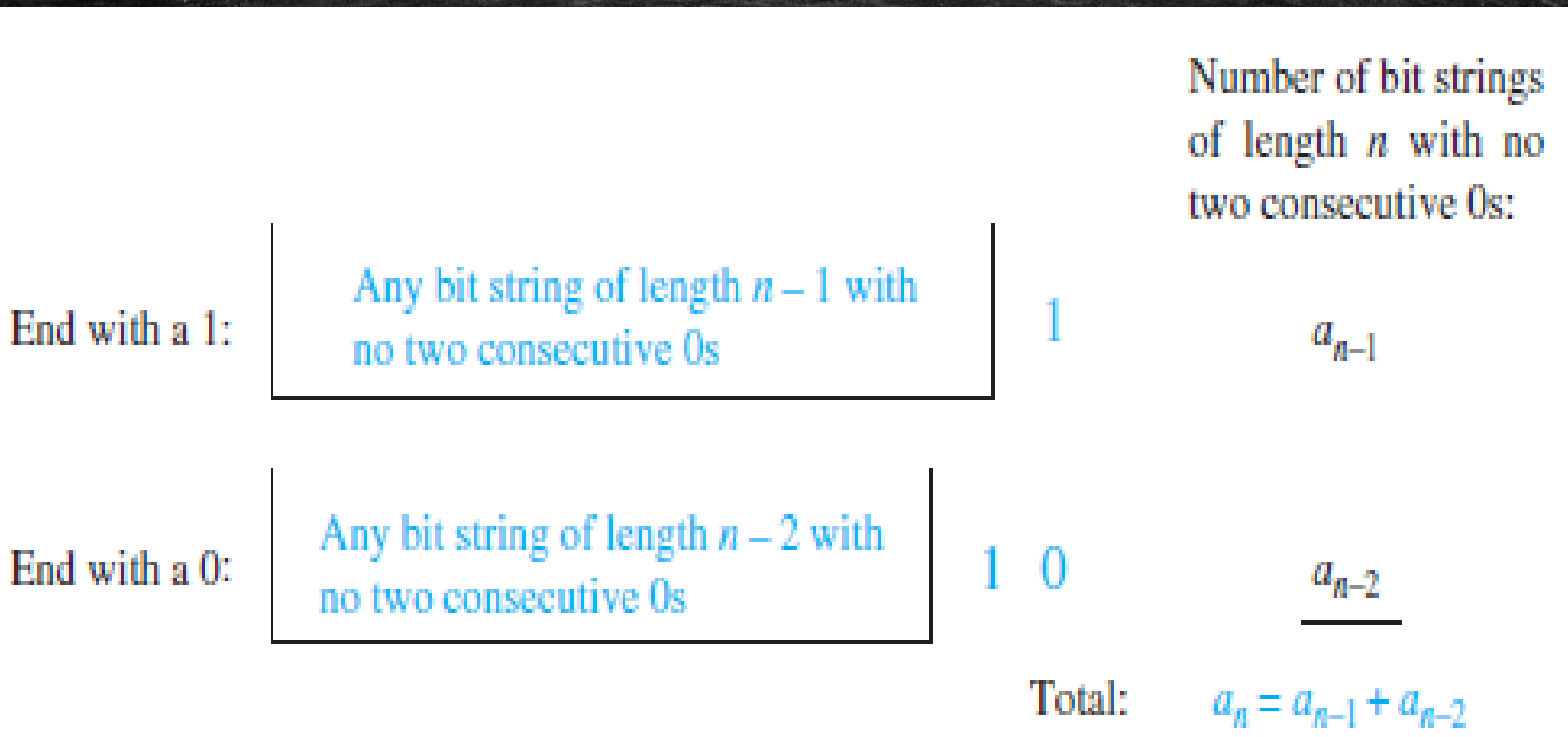


FIGURE 4 Counting Bit Strings of Length n with No Two Consecutive 0s.

Modeling With Recurrence Relations: Another Example

- 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*

$$a_3 = a_2 + a_1 = 3 + 2 = 5,$$

$$a_4 = a_3 + a_2 = 5 + 3 = 8,$$

$$a_5 = a_4 + a_3 = 8 + 5 = 13.$$

Modeling With Recurrence Relations: 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.

Modeling With Recurrence Relations: Codeword Enumeration

- *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.*

Modeling With Recurrence Relations: Codeword Enumeration

- *Solution (cont.):* 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.*

Modeling With Recurrence Relations: Codeword Enumeration

- *Solution (cont.): (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.*

Modeling With Recurrence Relations: Codeword Enumeration

- *Solution (cont.):* Because all valid strings of length n *are produced in one of these two ways, it follows that* there are

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

$$= 8a_{n-1} + 10^{n-1}$$

valid strings of length n.

Modeling With Recurrence Relations: Another Example

- Find a recurrence relation for C_n, *the number of ways to parenthesize the product of* $n + 1$ *numbers,* $x_0 \cdot x_1 \cdot x_2 \cdot \ldots \cdot x_n$, *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:*

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

Modeling With Recurrence Relations: Another Example

- *Solution:* To develop a recurrence relation for C_n, we note that however we insert parentheses in the product $x_0 \cdot x_1 \cdot x_2 \cdot \ldots \cdot x_n$, one "$\cdot$" operator remains outside all parentheses, namely, the operator for the final multiplication to be performed. [For example, in $\big(x_0 \cdot (x_1 \cdot x_2)\big) \cdot x_3$, it is the final "$\cdot$", while in $(x_0 \cdot x_1) \cdot (x_2 \cdot x_3)$ it is the second "$\cdot$"]. This final operator appears between two of the $n + 1$ numbers, say, x_k and x_{k+1}.

Modeling With Recurrence Relations: Another Example

- *Solution (cont.):* 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 $x_0 \cdot x_1 \cdot \ldots \cdot x_k$ 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 $x_{k+1} \cdot x_{k+2} \cdot \ldots \cdot x_n$ to determine the order in which these $n - k$ numbers are to be multiplied.

Modeling With Recurrence Relations: Another Example

- *Solution (cont.):* Because this final operator can appear between any two of the $n + 1$ numbers, it follows that

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

$$+ C_{n-2} C_1 + C_{n-1} C_0 = \sum_{k=0}^{n-1} C_k C_{n-k-1}$$

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

Algorithms and Recurrence Relations

- We also introduce another algorithmic paradigm known as **dynamic programming**, which can be used to solve many optimization problems efficiently.
- An algorithm follows the dynamic programming paradigm when it recursively breaks down 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.

Algorithms and Recurrence Relations

- 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. Here 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.

Algorithms and Recurrence Relations

- 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

- There is a problem where a goal 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 could develop a greedy algorithm that always produces an optimal schedule. But 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.

An example of dynamic programming

- 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.)

An example of dynamic programming

- 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 $e_1 \leq e_2 \leq \cdots \leq e_n$.

An example of dynamic programming

- 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 $e_i \leq s_j$, 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.

An example of dynamic programming

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

$$Talk\ 1\text{: } start\ 8\ a.m.\text{, } end\ 10\ a.m.$$
$$Talk\ 2\text{: } start\ 9\ a.m.\text{, } end\ 11\ a.m.$$
$$Talk\ 3\text{: } start\ 10\text{:}30\ a.m.\text{, } end\ 12\ noon$$
$$Talk\ 4\text{: } start\ 9\text{:}30\ a.m.\text{, } end\ 1\ p.m.$$
$$Talk\ 5\text{: } start\ 8\text{:}30\ a.m.\text{, } end\ 2\ p.m.$$
$$Talk\ 6\text{: } start\ 11\ a.m.\text{, } end\ 2\ p.m.$$
$$Talk\ 7\text{: } start\ 1\ p.m.\text{, } end\ 2\ p.m.$$

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

An example of dynamic programming

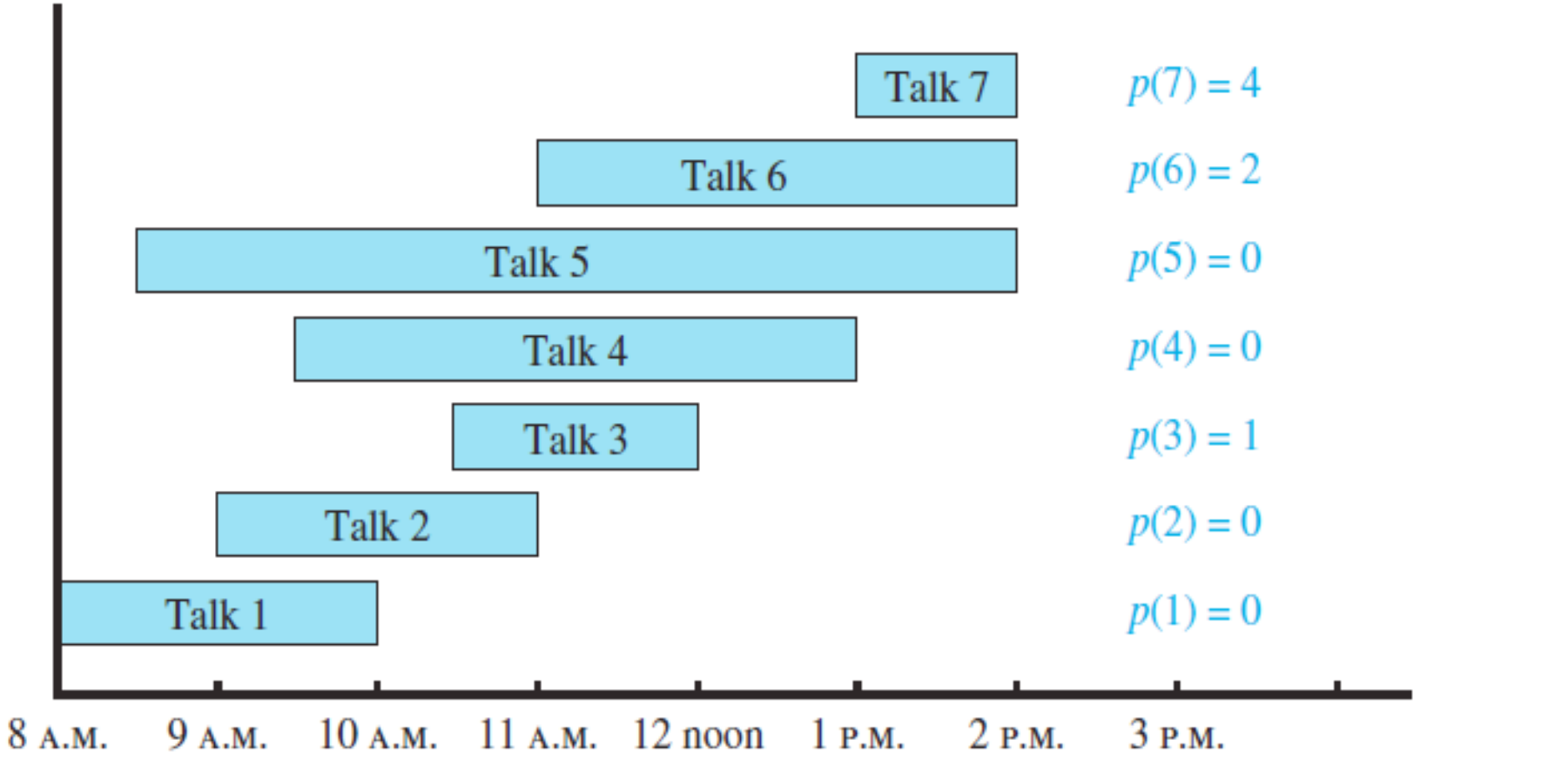


FIGURE 5 **A Schedule of Lectures with the Values of $p(n)$ Shown.**

An example of dynamic programming

- *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; $p(5) = 0$ 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.

An example of dynamic programming

- 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.

An example of dynamic programming

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

An example of dynamic programming

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

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

An example of dynamic programming

- 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.

An example of dynamic programming

ALGORITHM 1 Dynamic Programming Algorithm for Scheduling Talks.

procedure *Maximum Attendees* ($s_1, s_2, \ldots, s_n$: start times of talks;
$e_1, e_2, \ldots, e_n$: end times of talks; $w_1, w_2, \ldots, w_n$: 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 $p(j) := \max\{i \mid i < j$ 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}

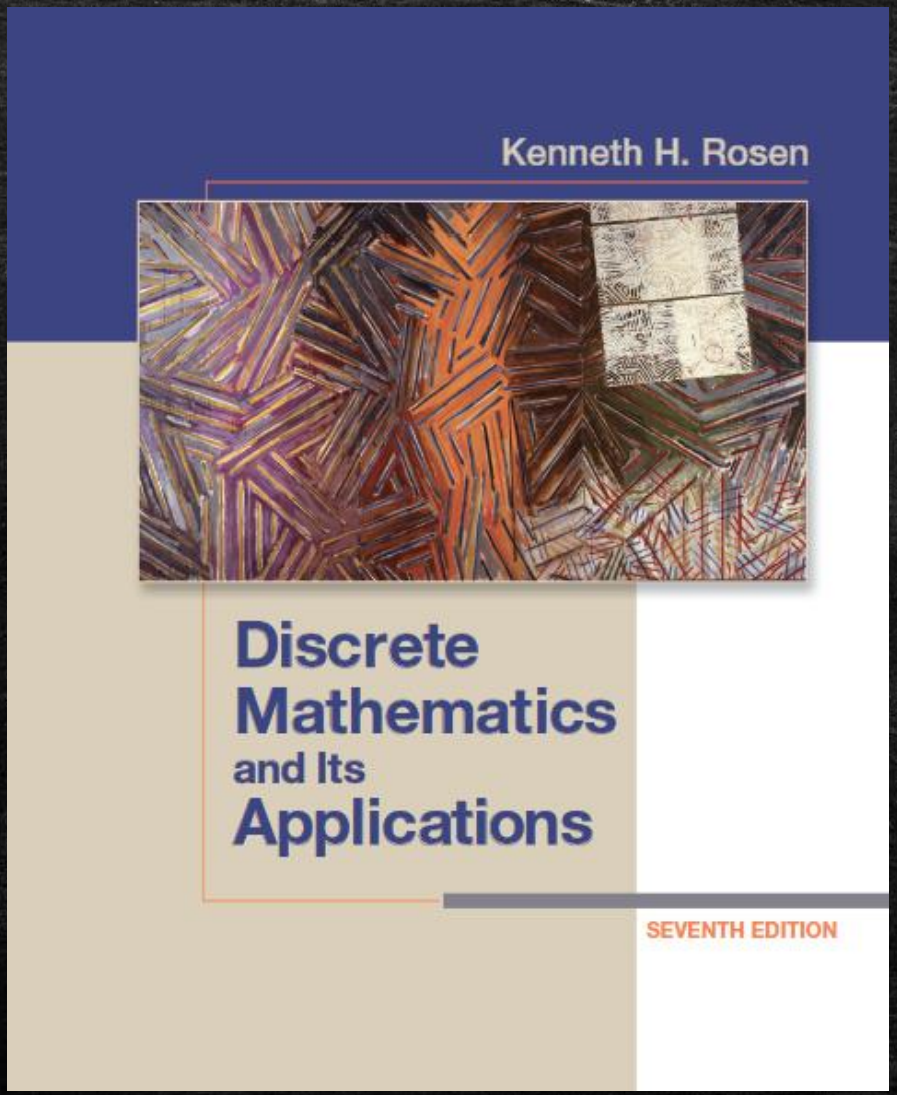
An example of dynamic programming

- 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 $w_j + T\big(p(j)\big) \geq T(j-1)$. We leave it as Exercise to construct an algorithm based on this observation that determines which talks should be scheduled to achieve the maximum total number of attendees.

An example of dynamic programming

- 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 homework Exercises for other examples of dynamic programming.

HOMEWORK: Exercises 2, 4, 8, 12, 14, 24, 26, 28 on pp. 510-512



Solving Linear Recurrence Relations

- A wide variety of recurrence relations occur in models. Some of these recurrence relations can be solved using iteration or some other ad hoc technique. However, one important class of recurrence relations can be explicitly solved in a systematic way. These are recurrence relations that express the terms of a sequence as linear combinations of previous terms.

Solving Linear Recurrence Relations

- **DEFINITION 1.** A *linear homogeneous recurrence relation of degree k with constant coefficients* is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ \ldots \ + c_k a_{n-k},$$

where $c_1, c_2, \ldots, c_k$ are real numbers, and $c_k \neq 0$.

Solving Linear Recurrence Relations

- The recurrence relation in the definition is **linear** because the right-hand side is a sum of previous terms of the sequence each multiplied by a function of n. The recurrence relation is **homogeneous** because no terms occur that are not multiples of the a_j's. The coefficients of the terms of the sequence are all **constants**, rather than functions that depend on *n*. The **degree** is k because a_n is expressed in terms of the previous k terms of the sequence.

Solving Linear Recurrence Relations

- A consequence of the second principle of mathematical induction is that a sequence satisfying the recurrence relation in the definition is uniquely determined by this recurrence relation and the k initial conditions

$$a_0 = C_0, a_1 = C_1, \ldots, a_{k-1} = C_{k-1}.$$

Solving Linear Recurrence Relations: Example

- The recurrence relation $P_n = 11P_{n-1}$ is a linear homogeneous recurrence relation of degree one. The recurrence relation $f_n = f_{n-1} + f_{n-2}$ is a linear homogeneous recurrence relation of degree two. The recurrence relation $a_n = a_{n-5}$ is a linear homogeneous recurrence relation of degree five.

Solving Linear Recurrence Relations: Example

- The recurrence relation $a_n = a_{n-1} + a_{n-2}^2$ is not linear. The recurrence relation $H_n = 2H_{n-1} + 1$ is not homogeneous. The recurrence relation $B_n = nB_{n-1}$ does not have constant coefficients.
- Linear homogeneous recurrence relations are studied for two reasons. First, they often occur in modeling of problems. Second, they can be systematically solved.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- The basic approach for solving linear homogeneous recurrence relations is to look for solutions of the form $a_n = r^n$, where r is a constant. Note that $a_n = r^n$ is a solution of the recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ \ldots \ + c_k a_{n-k}$$

- if and only if

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \ \ldots \ + c_k r^{n-k}.$$

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- When both sides of this equation are divided by r^{n-k} and the right-hand side is subtracted from the left, we obtain the equation

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \ \ldots - c_{k-1} r - c_k = 0.$$

- Consequently, the sequence $\{a_n\}$ with $a_n = r^n$ is a solution if and only if r is a solution of this last equation.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- We call this the **characteristic equation** of the recurrence relation. The solutions of this equation are called the **characteristic roots** of the recurrence relation. As we will see, these characteristic roots can be used to give an explicit formula for all the solutions of the recurrence relation.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- We will first develop results that deal with linear homogeneous recurrence relations with constant coefficients of degree two. Then corresponding general results when the degree may be greater than two will be stated. Because the proofs needed to establish the results in the general case are more complicated, they will not be given here.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- We now turn our attention to linear homogeneous recurrence relations of degree two. First, consider the case when there are two distinct characteristic roots.
- The case when there are two equal characteristic roots is left as exercise.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- **THEOREM 1.** Let c_1 and c_2 be real numbers. Suppose that

$$r^2 - c_1 r - c_2 = 0$$

- has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$$

- for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof:*** We must do two things to prove the theorem. First, it must be shown that if r_1 and r_2 are the roots of the characteristic equation, and α_1 and α_2 are constants, then the sequence $\{a_n\}$ with $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ is a solution of the recurrence relation. Second, it must be shown that if the sequence $\{a_n\}$ is a solution, then $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for some constants α_1 and α_2.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** Now we will show that if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, then the sequence $\{a_n\}$ is a solution of the recurrence relation. Because r_1 and r_2 are roots of $r^2 - c_1 r - c_2 = 0$, it follows that

$$r_1^2 = c_1 r_1 + c_2, r_2^2 = c_1 r_2 + c_2.$$

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** From these equations, we see that

$$c_1 a_{n-1} + c_2 a_{n-2}$$
$$= c_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + c_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2})$$
$$= \alpha_1 r_1^{n-2}(c_1 r_1 + c_2) + \alpha_2 r_2^{n-2}(c_1 r_2 + c_2)$$
$$= \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 = \alpha_1 r_1^n + \alpha_2 r_2^n = a_n$$

- This shows that the sequence $\{a_n\}$ with $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ is a solution of the recurrence relation.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** To show that every solution $\{a_n\}$ of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ has $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \ldots$, for some constants α_1 and α_2, suppose that $\{a_n\}$ is a solution of the recurrence relation, and the initial conditions $a_0 = C_0$ and $a_1 = C_1$ hold. It will be shown that there are constants α_1 and α_2 such that the sequence $\{a_n\}$ with $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ satisfies these same initial conditions.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** This requires that

$$a_0 = C_0 = \alpha_1 + \alpha_2,$$

$$a_1 = C_1 = \alpha_1 r_1 + \alpha_2 r_2.$$

- We can solve these two equations for α_1 and α_2. From the first equation it follows that $\alpha_2 = C_0 - \alpha_1$. Inserting this expression into the second equation gives

$$C_1 = \alpha_1 r_1 + (C_0 - \alpha_1) r_2.$$

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** Hence, $C_1 = \alpha_1(r_1 - r_2) + C_0 r_2$. This shows that

$$\alpha_1 = C_1 - C_0 r_2 r_1 - r_2 \text{ and } \alpha_2 = C_0 - \alpha_1 =$$

$$= C_0 - C_1 - C_0 r_2 r_1 - r_2 = C_0 r_1 - C_1 r_1 - r_2,$$

- where these expressions for α_1 and α_2 depend on the fact that $r_1 = r_2$. (When $r_1 = r_2$, this theorem is not true.) Hence, with these values for α_1 and α_2, the sequence $\{a_n\}$ with $\alpha_1 r_1^n + \alpha_2 r_2^n$ satisfies the two initial conditions.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** We know that $\{a_n\}$ and $\{\alpha_1 r_1^n + \alpha_2 r_2^n\}$ are both solutions of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ and both satisfy the initial conditions when $n = 0$ and $n = 1$. Because there is a unique solution of a linear homogeneous recurrence relation of degree two with two initial conditions, it follows that the two solutions are the same, that is, $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for all nonnegative integers n.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

- ***Proof (cont.):*** We have completed the proof by showing that a solution of the linear homogeneous recurrence relation with constant coefficients of degree two must be of the form $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, where α_1 and α_2 are constants.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients: Example

- What is the solution of the recurrence relation

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

with $a_0 = 2$ and $a_1 = 7$?

- *Solution:* Theorem 1 can be used to solve this problem. The characteristic equation of the recurrence relation is $r^2 - r - 2 = 0$. Its roots are $r = 2$ and $r = -1$. Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = \alpha_1 2^n + \alpha_2 (-1)^n$, for some constants α_1 and α_2.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients: Example

- *Solution (cont.):* From the initial conditions, it follows that

$$a_0 = 2 = \alpha_1 + \alpha_2,$$

$$a_1 = 7 = \alpha_1 \cdot 2 + \alpha_2 \cdot (-1).$$

- Solving these two equations shows that $\alpha_1 = 3$ and $\alpha_2 = -1$. Hence, the solution to the recurrence relation and initial conditions is the sequence $\{a_n\}$ with

$$a_n = 3 \cdot 2^n - (-1)^n.$$

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients: Example

- Find an explicit formula for the Fibonacci numbers.
- *Solution:* Recall that the sequence of Fibonacci numbers satisfies the recurrence relation $f_n = f_{n-1} + f_{n-2}$ and also satisfies the initial conditions $f_0 = 0$ and $f_1 = 1$. The roots of the characteristic equation $r^2 - r - 1 = 0$ are

$$r_1 = \frac{1+\sqrt{5}}{2} \text{ and } r_2 = \frac{1-\sqrt{5}}{2}.$$

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients: Example

- *Solution (cont.):* Therefore, from Theorem 1 it follows that the Fibonacci numbers are given by

$$f_n = \alpha_1 \left(\frac{(1+\sqrt{5})}{2} \right)^n + \alpha_2 \left(\frac{1-\sqrt{5}}{2} \right)^n,$$

for some constants α_1 and α_2. The initial conditions $f_0 = 0$ and $f_1 = 1$ can be used to find these constants.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients: Example

- *Solution (cont.):* We have $f_0 = \alpha_1 + \alpha_2 = 0$, $f_1 = \alpha_1\left(\frac{1+\sqrt{5}}{2}\right) + \alpha_2\left(\frac{1-\sqrt{5}}{2}\right) = 1.$
- The solution to these simultaneous equations for α_1 and α_2 is $\alpha_1 = \frac{1}{\sqrt{5}}$, $\alpha_2 = -\frac{1}{\sqrt{5}}$. Consequently, the Fibonacci numbers are given by

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{(1+\sqrt{5})}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

HOMEWORK: Exercises 2, 4, 6 on p. 524