Unit DT

Decision Trees and Recursion

In many situations one needs to make a series of decisions. This leads naturally to a structure called a "decision tree." Decision trees provide a geometrical framework for organizing the decisions. The important aspect is the decisions that are made. Everything we do in this unit could be rewritten to avoid the use of trees; however, trees

- give us a powerful intuitive basis for viewing the problems of this chapter,
- provide a language for discussing the material,
- allow us to view the collection of all decisions in an organized manner.

We'll begin with elementary examples of decision trees. We then show how decision trees can be used to study recursive algorithms. Next we shall look at decision trees and "Bayesian methods" in probability theory. Finally we relate decision trees to induction and recursive equations.

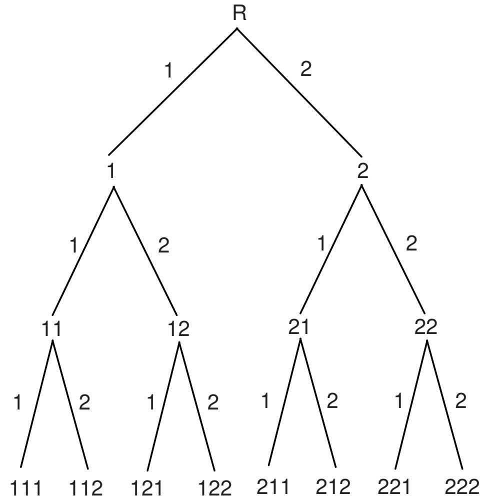
Section 1: Basic Concepts of Decision Trees

One area of application for decision trees is systematically listing a variety of functions. The simplest general class of functions to list is the entire set $\underline{n}^{\underline{k}}$. We can create a typical element in the list by choosing an element of $\underline{n}$ and writing it down, choosing another element (possibly the same as before) of $\underline{n}$ and writing it down next, and so on until we have made k decisions. This generates a function in one line form sequentially: First $f(1)$ is chosen, then $f(2)$ is chosen and so on. We can represent all possible decisions pictorially by writing down the decisions made so far and then some downward "edges" indicating the possible choices for the next decision.

We begin this section by discussing the picture of a decision tree, illustrating this with a variety of examples. Then we study how a tree is traversed, which is a way computers deal with the trees.

What is a Decision Tree?

Example 1 (Decision tree for $\underline{2}^{\underline{3}}$) Here is an example of a decision tree for the functions $\underline{2}^{\underline{3}}$. We've omitted the commas; for example, 121 stands for the function 1,2,1 in one-line form.



The set

$$V = \{\text{R}, 1, 2, 11, 12, 21, 22, 111, 112, 121, 122, 211, 212, 221, 222\}$$

is called the set of *vertices* of the decision tree. The vertex set for a decision tree can be any set, but must be specified in describing the tree. You can see from the picture of the decision tree that the places where the straight line segments (called *edges*) of the tree end is where the vertices appear in the picture. Each vertex should appear exactly once in the picture. The symbol R stands for the *root* of the decision tree. Various choices other than R can be used as the symbol for the root.

The *edges* of a decision tree such as this one are specified by giving the pair of vertices at the two ends of the edge, top vertex first, as follows: $(\text{R}, 1)$, $(21, 212)$, etc. The vertices at the ends of an edge are said to be "incident" on that edge. The complete set of edges of this decision tree is the set $E = \{$ (R,1), (R,2), (1,11), (1,12), (2,21),(2,22), (11,111), (11,112), (12,121), (12,122), (21,211), (21,212), (22,221), (22,222)$\}$. In addition to the edges, there are "labels," either a "1" or a "2," shown on the line segments representing the edges in the picture. This labeling of the edges can be thought of as a function from the set of edges, E, to the set $\{1, 2\}$.

If $e = (v, w)$ is an edge, the vertex w, is called a *child* of v, and v is the *parent* or w. The children of 22 are 221 and 222. The parent of 22 is 2.

The *degree* of a vertex v is the number of edges incident on that vertex. The *down degree* of v is the number of edges $e = (v, w)$ incident on that vertex (and below it); in other words, it is the number of children of v. The degree of 22 is 3 (counting edges $(2, 22), (22, 221), (22, 222)$). The down degree of 22 is 2 (counting edges $(22, 221), (22, 222)$). Vertices with down degree 0 are called *leaves*. All other vertices are called *internal* vertices.

For any vertex v in a decision tree there is a unique list of edges $(x_1, x_2), (x_2, x_3), \ldots,$ (x_k, x_{k+1}) from the root x_1 to $v = x_{k+1}$. This sequence is called the *path to a vertex* in the decision tree. The number of edges, k, is the length of this path and is called the *height* or *distance* of vertex v to from the root. The path to 22 is $(\mathrm{R}, 2), (2, 22)$. The height of 22 is 2. $\square$

The decision tree of the previous example illustrates the various ways of generating a function in $\underline{2}^{\underline{3}}$ sequentially. It's called a *decision tree for generating the functions in* $\underline{2}^{\underline{3}}$. Each edge in the decision tree is labeled with the choice of function value to which it corresponds. Note that the labeling does not completely describe the corresponding decision — we should have used something like "Choose 1 for the value of $f(2)$" instead of simply "1" on the line from 1 to 11.

In this terminology, a vertex v represents the partial function constructed so far, when the vertex v has been reached by starting at the root, following the edges to v, and making the decisions that label the edges on that unique path from the root to v. The edges leading out of a vertex are labeled with all possible decisions that can be made next, given the partial function at the vertex. We labeled the edges so that the labels on edges out of each vertex are in order, 1,2, when read left to right. The leaves are the finished functions. Notice that the leaves are in lexicographic order. In general, if we agree to label the edges from each vertex in order, then any set of functions generated sequentially by specifying $f(i)$ at the ith step will be in lex order.

To create a single function we start at the root and choose downward edges (i.e., make decisions) until we reach a leaf. This creates a *path* from the root to a leaf. We may describe a path in any of the following ways:

- the sequence of vertices $v_0, v_1, \ldots, v_m$ on the path from the root v_0 to the leaf v_m;
- the sequence of edges $e_1, e_2, \ldots, e_m$, where $e_i = (v_{i-1}, v_i)$, $i = 1, \ldots, m$;
- the sequence of decisions $D_1, D_1, \ldots, D_m$, where e_i is labeled with decision D_i.

We illustrate with three descriptions of the path from the root R to the leaf 212 in Example 1:

- the vertex sequence is R, 2, 21, 212;
- the edge sequence is $(R, 2), (2, 21), (21, 212)$;
- the decision sequence is 2, 1, 2.

Decision trees are a part of a more general subject in discrete mathematics called "graph theory," which is studied in another unit.

It is now time to look at some more challenging examples so that we can put decision trees to work for us. The next example involves counting words where the decisions are based on patterns of consonants and vowels.

Example 2 (Counting words) Using the 26 letters of the alphabet and considering the letters AEIOUY to be vowels how many five letter "words" (i.e. five long lists of letters) are there, subject to the following rules?

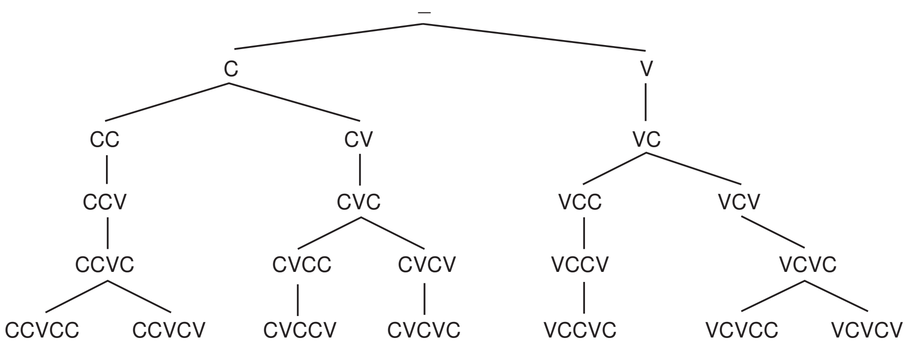
(a) No vowels are ever adjacent.

(b) There are never three consonants adjacent.

(c) Adjacent consonants are always different.

To start with, it would be useful to have a list of all the possible patterns of consonants and vowels; e.g., CCVCV (with C for consonant and V for vowel) is possible but CVVCV and CCCVC violate conditions (a) and (b) respectively and so are not possible. We'll use a decision tree to generate these patterns in lex order. Of course, a pattern CVCCV can be thought of as a function f where $f(1) = \text{C}$, $f(2) = \text{V}$, ..., $f(5) = \text{V}$.

We could simply try to list the patterns (functions) directly without using a decision tree. The decision tree approach is preferable because we are less likely to overlook something. The resulting tree can be pictured as follows:



At each vertex there are potentially two choices, but at some vertices only one is possible because of rules (a) and (b) above. Thus there are one or two decisions at each vertex. You should verify that this tree lists all possibilities systematically.

We have used the dash "–" as the symbol for the root. This stands for the empty word on the letters C and V. The set of labels for the vertices of this decision tree T is a set of words of length 0 through 5. The vertex set is determined by the rules (or "syntax") associated with the problem (rules (a), (b), and (c) above).

Using the rules of construction, (a), (b), and (c), we can now easily count the number of words associated with each leaf. The total number of words is the sum of these individual counts. For CCVCC we obtain $(20 \times 19)^2 \times 6$; for CCVCV, CVCCV, VCCVC, and VCVCC we obtain $(20 \times 19) \times 20 \times 6^2$; for CVCVC we obtain $20^3 \times 6^2$; for $VCVCV$ we obtain $20^2 \times 6^3$. $\square$

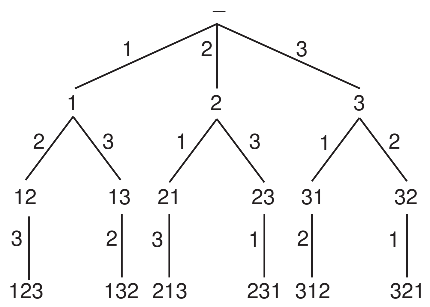
Definition 1 (Rank of an element of a list) *The rank of an element in a list is the number of elements that appear before it in the list. The rank of a leaf of a decision tree is the number of leaves that are to the left of it in the picture of the tree. The rank is denoted by the function RANK.*

In Example 1, RANK(212) = 5 and RANK(111) = 0.

Rank can be used to store data in a computer. For example, suppose we want to store information for each of the $10! \approx 3.6 \times 10^6$ permutations of $\underline{10}$. The naive way to do this is to have a $10 \times 10 \times \cdots \times 10$ array and store information about the permutation f in location $f(1), \ldots, f(10)$. This requires storage of size 10^{10}. If we store information about permutations in a one dimensional array with information about f stored at RANK(f), we only need 10! storage locations, which is much less. We'll discuss ranking permutations soon.

The inverse of the rank function is also useful. Suppose we want to generate objects at random from a set of n objects. Let RANK be a rank function for them. Generate a number k between 0 and $n-1$ inclusive at random. Then $\text{RANK}^{-1}(k)$ is a random object.

Example 3 (Permutations in lexicographic order) Recall that we can think of a permutation on $\underline{3}$ as a bijection $f : \underline{3} \to \underline{3}$. Its one-line form is $f(1), f(2), f(3)$. Here is an example of a decision trees for this situation (omitting commas):



Because we first chose $f(1)$, listing its values in increasing order, then did the same with $f(2)$ and finally with $f(3)$, the leaves are listed lexicographically, that is, in "alphabetical" order like a dictionary only with numbers instead of letters.

We could have abbreviated this decision tree a bit by shrinking the edges coming from vertices with only one decision and omitting labels on nonleaf vertices. As you can see, there is no "correct" way to label a decision tree. The intermediate labels are simply a tool to help you correctly list the desired objects (functions in this case) at the leaves. Sometimes one may even omit the function at the leaf and simply read it off the tree by looking at the labels on the edges or vertices associated with the decisions that lead from the root to the leaf. In this tree, the labels on an edge going down from vertex v tell us what values to add to the end of v to get a "partial permutation" that is one longer than v. $\square$

Permutations

We now look at two ways of ranking permutations. The method for generating random permutations in Example 23 of Unit Fn can provide another method for ranking permutations. If you'd like a challenge, you can think about how to do that. We won't discuss it.

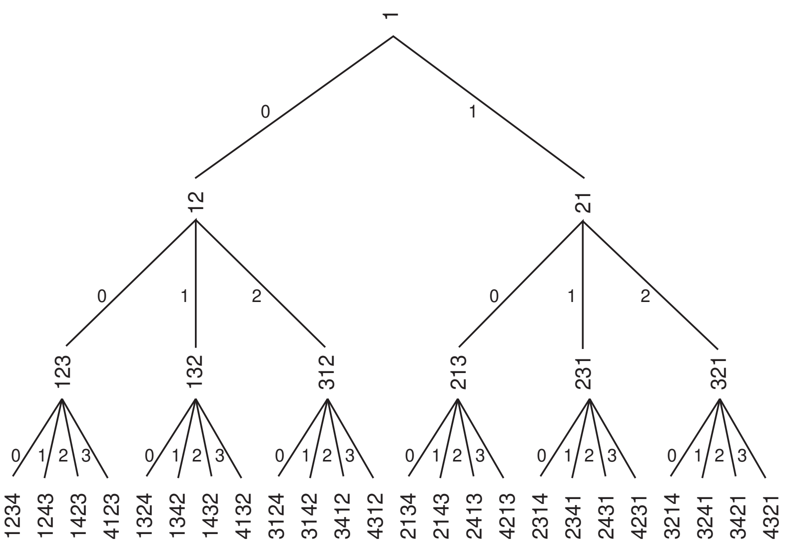
Example 4 (Permutations in direct insertion order) Another way to create a permutation is by *direct insertion.* (Often this is simply called "insertion.") Suppose that we have an ordered list of k items into which we want to insert a new item. It can be placed in any of $k+1$ places; namely, at the end of the list or immediately before the ith item where $1 \le i \le k$. By starting out with 1, choosing one of the two places to insert 2 in this list, choosing one of the three places to insert 3 in the new list and, finally, choosing one of the four places to insert 4 in the newest list, we will have produced a permutation of $\underline{4}$. To do this, we need to have some convention as to how the places for insertion are numbered when the list is written from left to right. The obvious choice is from left to right; however, right to left is often preferred. We'll use right to left. One reason for this choice is that the leftmost leaf is $12\ldots n$ as it is for lex order.

If there are $k+1$ possible positions to insert something, we number the positions $0, 1, \ldots, k$, starting with the rightmost position as number 0. We'll use the notation $(\,)_i$ to stand for position i so that we can keep track of the positions when we write a list into which something is to be inserted.

Here's the derivation of the permutation of $\underline{4}$ associated with the insertions 1, 1 and 2.

- Start with $(\,)_1 1 (\,)_0$.
- Choose the insertion of the symbol 2 into position 1 (designated by $(\,)_1$) to get 21. With positions of possible insertions indicated, this is $(\,)_2 2 (\,)_1 1 (\,)_0$.
- Now insert symbol 3 into position 1 (designated by $(\,)_1$) to get 231 or, with possible insertions indicated $(\,)_3 2 (\,)_2 3 (\,)_1 1 (\,)_0$.
- Finally, insert symbol 4 into position 2 to get 2431.

Here is the decision tree for permutations of $\underline{4}$ in direct insertion order. We've turned the vertices sideways so that rightmost becomes topmost in the insertion positions.



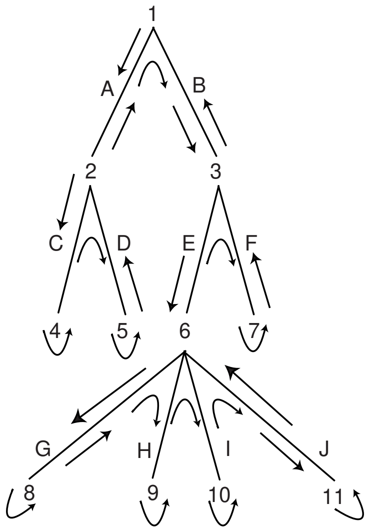
The labels on the vertices are, of course, the partial permutations, with the full permutations appearing on the leaves. The decision labels on the edges are the positions in which to insert the next number. Notice that the labels on the leaves are no longer in lex order because we constructed the permutations differently. Had we labeled the vertices with the positions used for insertion, the leaves would then be labeled in lex order. For example, 2413 becomes 1,0,2, which is gotten by reading edge labels on the path from the root to the leaf labeled 2413. Similarly 4213 becomes, 1,0,3.

Like the method of lex order generation, the method of direct insertion generation can be used for other things besides permutations. However, direct insertion cannot be applied as widely as lex order. Lex order generation works with anything that can be thought of as an (ordered) list, but direct insertion requires more structure. Note that the RANK(3412) = 10 and RANK(4321) = 23. What would RANK(35412) be for the permutations on $\underline{5}$ in direct insertion order? $\square$

Traversing Decision Trees

We conclude this section with an important class of search algorithms called *backtracking algorithms*. In many computer algorithms it is necessary either to systematically inspect all the vertices of a decision tree or to find the leaves of the tree. An algorithm that systematically inspects all the vertices (and so also finds the leaves) is called a *traversal of the tree*. How can we create such an algorithm? To understand how to do this, we first look at how a tree can be "traversed."

Example 5 (Traversals of a decision tree) Here is a sample decision tree T with edges labeled A through J and vertices labeled 1 through 11.



The arrows are not part of the decision tree, but will be helpful to us in describing certain ideas about linear orderings of vertices and edges that are commonly associated with decision trees. Imagine going around ("traversing") the decision tree following arrows. Start at the root, 1, go down edge A to vertex 2, etc. Here is the sequence of vertices as encountered in this process: 1, 2, 4, 2, 5, 2, 1, 3, 6, 8, 6, 9, 6, 10, 6, 11, 6, 3, 7, 3, 1. This sequence of vertices is called the *depth first vertex sequence*, DFV(T), of the decision tree T. The number of times each vertex appears in DFV(T) is one plus the down degree of that vertex. For edges, the corresponding sequence is A, C, C, D, D, A, B, E, G, G, H, H, I, I, J, J, E, F, F, B. This sequence is the *depth first edge sequence*, DFE(T), of the tree. Every edge appears exactly twice in DFE(T). If the vertices of the tree are read left to right, top to bottom, we obtain the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. This is called the *breadth first vertex sequence*, BFV(T). Similarly, the *breadth first edge sequence*, BFE(T), is A, B, C, D, E, F, G, H, I, J.

The sequences BFV(T) and BFE(T) are linear orderings of the vertices and edges of the tree T (i.e., each vertex or edge appears exactly once in the sequence). We also associate two linear orderings with DFV(T):

- PREV(T), called the *preorder sequence of vertices* of T, is the sequence of *first* occurrences of the vertices of T in DFV(T).
- POSV(T), called the *postorder sequence of vertices* of T, is the sequence of *last* occurrences of the vertices of T in DFV(T).

For the present tree

$$\text{PREV}(T) = 1,2,4,5,3,6,8,9,10,11,7 \quad \text{and} \quad \text{POSV}(T) = 4,5,2,8,9,10,11,6,7,3,1.$$

With a little practice, you can quickly construct PREV(T) and POSV(T) directly from the picture of the tree. For PREV(T), follow the arrows and list each vertex the first time you encounter it (and only then). For POSV(T), follow the arrows and list each vertex the last time you encounter it (and only then). Notice that the order in which the leaves of T appear, 4, 5, 8, 9, 10, 11, is the same in both PREV(T) and POSV(T). □

We now return to the problem of creating a traversal algorithm. One way to imagine doing this is to generate the depth first sequence of vertices and/or edges of the tree as done in the preceding example. We can describe our traversal more precisely by giving an algorithm. Here is one which traverses a tree whose leaves are associated with functions and lists the functions in the order of PREV(T).

Theorem 1 (Systematic Traversal Algorithm) *The following procedure systematically visits the vertices of a tree T in depth-first order,* DFV(T), *listing the leaves as they occur in the list* DFV(T).

1. **Start:** *Mark all edges as unused and position yourself at the root.*
2. **Leaf:** *If you are at a leaf, list the function.*
3. **Decide case:** *If there are no unused edges leading out from the vertex, go to Step 4; otherwise, go to Step 5.*
4. **Backtrack:** *If you are at the root, STOP; otherwise, return to the vertex just above this one and go to Step 3.*
5. **Decision:** *Select the leftmost unused edge out of this vertex, mark it used, follow it to a new vertex and go to Step 2.*

Step 4 is labeled **Backtrack**. What does this mean? If you follow the arrows in the tree pictured in Example 5, backtracking corresponds to going toward the root on an edge that has already been traversed in the opposite direction. In other words, backtracking refers to the process of moving along an edge *back* toward the root of the tree. Thinking in terms of the decision sequence, backtracking corresponds to undoing (i.e., backtracking on) a decision previously made. Notice that the algorithm only needs to keep track of the decisions from the root to the present vertex — when it backtracks, it can "forget" the decision it is backtracking from. You should take the time now to apply the algorithm to Example 1, noting which decisions you need to remember at each time.

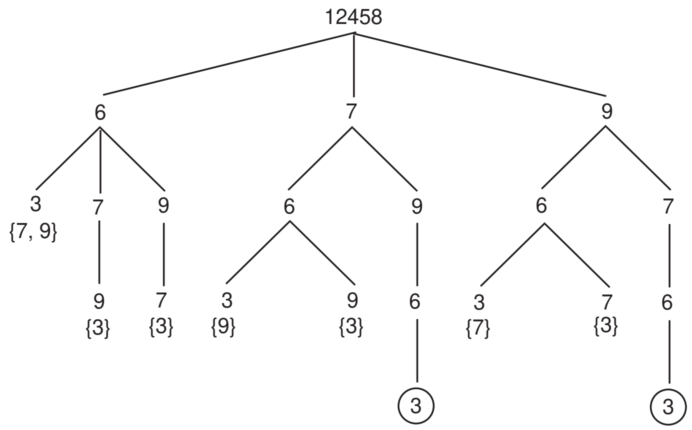
So far in our use of decision trees, it has always been clear what decisions are reasonable; i.e., are on a path to a solution. (In this case every leaf is a solution.) This is because we've looked only at simple problems such as listing *all* permutations of $\underline{n}$ or listing *all* functions in $\underline{n}^{\underline{k}}$. We now look at a situation where that is not the case.

Example 6 (Restricted permutations) Consider the following problem.

$$\text{List all permutations } f \text{ of } \underline{n} \text{ such that}$$
$$|f(i) - f(i+1)| \le 3 \quad \text{for} \quad 1 \le i < n\,.$$

It's not at all obvious what decisions are reasonable in this case. For instance, when $n = 9$, the partially specified one line function 124586 cannot be completed to a permutation.

There is a simple cure for this problem: We will allow ourselves to make decisions which lead to "dead ends," situations where we cannot continue on to a solution. With this expanded notion of a decision tree, there are often many possible decision trees that appear reasonable for doing something. Suppose that we're generating things in lex order and we've reached the vertex 12458. What do we do now? We'll simply continue to generate more of the permutation, making sure that $|f(i) - f(i+1)| \le 3$ is satisfied for that portion of the permutation we have generated so far. The resulting portion of the tree that starts with the partial permutation 12458 is represented by the following decision tree. Because the names the of the vertices are so long, we've omitted all of the name, except for the function value just added, for all but the root. Thus the rightmost circled vertex is 124589763.



Each vertex is labeled with an additional symbol of the permutation after the symbols 12458. The circled leaves represent solutions. Each solution is obtained by starting with 12458 and adding all symbols on vertices of the path from the root, 12458, to the circled leaf. Thus the two solutions are 124587963 and 124589763. The leaves that are not circled are places where there is no way to extend the partial permutation corresponding to that leaf such that $|f(i) - f(i+1)| \le 3$ is satisfied. Below each leaf that is not a solution, you can see the set of values available for completing the partial permutation at that leaf. It is clear in those cases that no completion satisfying $|f(i) - f(i+1)| \le 3$ is possible.

Had we been smarter, we might have come up with a simple test that would have told us that 124586 could not be extended to a solution. This would have lead to a different decision tree in which the vertex corresponding to the partial permutation 124586 would have been a leaf.

You should note here that the numbers 3, 6, 7, 9 are *labels* on the vertices of this tree. A vertex of this tree is the partial permutation gotten by concatenating (attaching) the labels on the path from the root to that vertex to the permutation 12458. Thus, 124586 is a vertex with label 6. The path from the root to that vertex is 12458, 124586 (corresponding to the edge $(12458, 124586)$). The vertices are not explicitly shown, but can be figured out from the rules just mentioned. The labels on the vertices correspond to the decisions to be made (how to extend the partial permutation created thus far).

Our tree traversal algorithm, Theorem 1, requires a slight modification to cover this type of extended decision tree concept where a leaf need not be a solution: Change Step 2 to

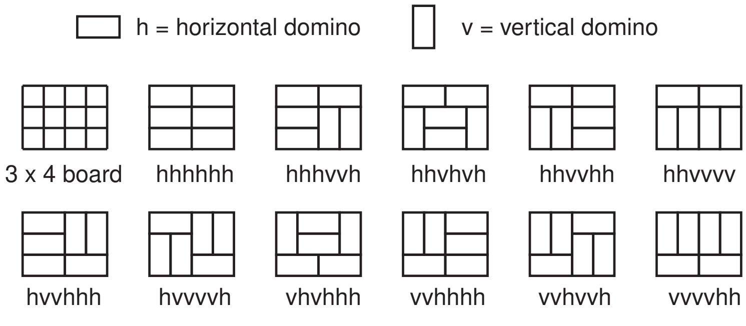
$2'$. **Leaf:** If you are at a leaf, take appropriate action.

For the tree rooted at 12458 in this example, seven of the leaves are not solutions and two are. For the two that are, the "appropriate action" is to print them out, jump and shout, or, in some way, proclaim success. For the leaves that are not solutions, the appropriate action is to note failure in some way (or just remain silent) and **Backtrack** (Step (4) of Theorem 1). This backtracking on leaves that are not solutions is where this type of algorithm gets its name: "backtracking algorithm." $\square$

How can there be more than one decision tree for generating solutions in a specified order? Suppose someone who was not very clever wanted to generate all permutations of $\underline{n}$ in lex order. He might program a computer to generate all functions in $\underline{n}^{\underline{n}}$ in lex order and to then discard those functions which are not permutations. This leads to a much bigger tree because n^n is much bigger than $n!$, even when n is as small as 3. A somewhat cleverer friend might suggest that he have the program check to see that $f(k) \neq f(k-1)$ for each $k > 1$. This won't slow down the program very much and will lead to only $n(n-1)^{n-1}$ functions. Thus the program should run faster. Someone else might suggest that the programmer check at each step to see that the function produced so far is an injection. If this is done, nothing but permutations will be produced, but the program may be much slower.

The lesson to be learned from the previous paragraph is that there is often a trade off between the size of the decision tree and the time that must be spent at each vertex determining what decisions to allow. Because of this, different people may develop different decision trees for the same problem. The differences between computer run times for different decision trees can be truly enormous. By carefully defining the criteria that allow one to decide that a vertex is a leaf, people have changed problems that were too long to run on a supercomputer into problems that could be easily run on a personal computer. We'll conclude this section with two examples of backtracking of the type just discussed.

Example 7 (Domino coverings) We are going to consider the problem of covering a m by n board (for example, $m = n = 8$ gives a chess board) with 1 by 2 rectangles (called "dominoes"). A domino can be placed either horizontally or vertically so that it covers two squares and does not overlap another domino. Here is a picture of the situation for $m = 3$, $n = 4$. (The sequences of h's and v's under eleven covered boards will be explained below.)



If the squares of the board are numbered systematically, left to right, top to bottom, from 1 to 12, we can describe any placement of dominoes by a sequence of 6 h's and v's: Each of the domino placements in the above picture has such a description just below it. Take as an example, hhvhvh (the third domino covering in the picture). We begin with no dominoes on the board. None of the squares, numbered 1 to 12 are covered. The list of "unoccupied squares" is as follows:

1	2	3	4
5	6	7	8
9	10	11	12

Thus, the smallest unoccupied square is 1. The first symbol in hhvhvh is the h. That means that we take a horizontal domino and cover the square 1 with it. That forces us to cover square 2 also. The list of unoccupied squares is as follows:

		3	4
5	6	7	8
9	10	11	12

Now the smallest unoccupied square is 3. The second symbol in hhvhhv is also an h. Cover square 3 with a horizontal domino, forcing us to cover square 4 also. The list of unoccupied squares is as follows:

5	6	7	8
9	10	11	12

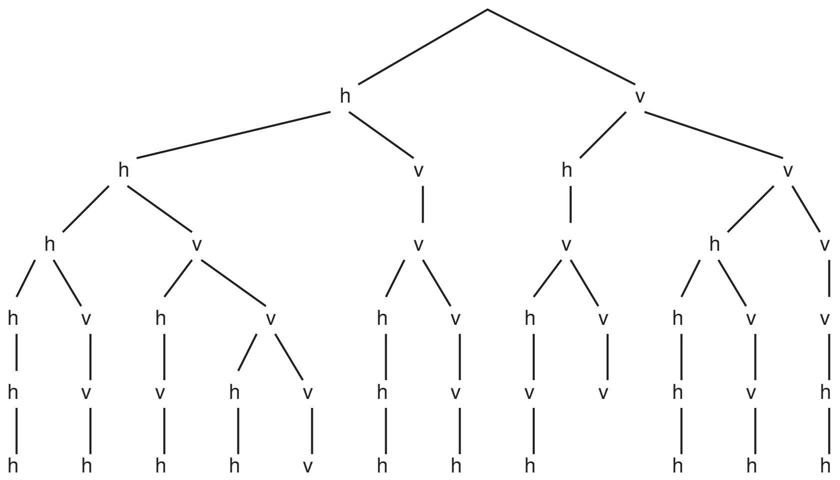
At this point, the first row of the board is covered with two horizontal dominoes (check the picture). Now the smallest unoccupied square is 5 (the first square in the second row). The third symbol in hhvhvh is v. Thus we cover square 5 with a vertical domino, forcing us to cover square 9 also. The list of unoccupied squares is as follows:

6	7	8
10	11	12

We leave it to you to continue this process to the bitter end and obtain the domino covering shown in the picture.

Here is the general description of the process. Place dominoes sequentially as follows. If the first unused element in the sequence is h, place a horizontal domino on the first unoccupied square and the square to its right. If the first unused element in the sequence is v, place a vertical domino on the first unoccupied square and the square just below it. Not all sequences correspond to legal placements of dominoes (try hhhhhv). For a 2×2 board, the only legal sequences are hh and vv For a 2×3 board, the legal sequences are hvh, vhh and vvv. For a 3×4 board, there are eleven legal sequences as shown in the picture at the start of this example.

To find these sequences in lex order we used a decision tree for generating sequences of h's and v's in lex order. Each decision is required to lead to a domino that lies entirely on the board and does not overlap another domino. Here is our decision tree:



Note that in this tree, the decision (label) that led to a vertex is placed at the vertex rather than on the edge. The actual vertices, not explicitly labeled, are the sequences of choices from the root to that vertex (e.g., the vertex hvv has label v). The leaf vhvvv associated with the path v, h, v, v, v does not correspond to a covering. It has been abandoned (i.e., declared a leaf but not a solution) because there is no way to place a domino on the lower left square of the board, which is the first free square. Draw a picture of the board to see what is happening. Our criterion for deciding if a vertex is a leaf is to check if that vertex corresponds to a solution or to a placement that does not permit another domino to be placed on the board. It is not hard to come up with a criterion that produces a smaller decision tree. For example, vhvv leaves the lower left corner of the board isolated. That means that vhvv cannot be extended to a solution, even though more dominoes can be placed on the board. But, checking this more restrictive criterion is more time consuming.

Exercises for Section 1

1.1. List the nonroot vertices of the decision tree in Example 2 in PREV, POSV and BFV orders.

1.2. Let RANK_L denote the rank in lex order and let RANK_I denote the rank in insertion order on permutations of $\underline{n}$. Answer the following questions and give reasons for your answers:

(a) For $n = 3$ and $n = 4$ which permutations σ have $\text{RANK}_L(\sigma) = \text{RANK}_I(\sigma)$?

(b) What is $\text{RANK}_L(2314)$? $\text{RANK}_L(45321)$?

(c) What is $\text{RANK}_I(2314)$? $\text{RANK}_I(45321)$?

(d) What permutation σ of $\underline{4}$ has $\text{RANK}_L(\sigma) = 15$?

(e) What permutation σ of $\underline{4}$ has $\text{RANK}_I(\sigma) = 15$?

(f) What permutation σ of $\underline{5}$ has $\text{RANK}_L(\sigma) = 15$?

1.3. Draw the decision tree to list all sequences of length six of A's and B's that satisfy the following conditions:

- There are no two adjacent A's.
- There are never three B's adjacent.
- If each leaf is thought of as a word, the leaves are in alphabetical order.

1.4. Draw a decision tree for $D(\underline{6}^{\underline{4}})$, the strictly decreasing functions from $\underline{4}$ to $\underline{6}$. You should choose a decision tree so that the leaves are in lex order when read from left to right

(a) What is the rank of 5431? of 6531?

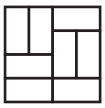
(b) What function has rank 0? rank 7?

(c) Your decision tree should contain the decision tree for $D(\underline{5}^{\underline{4}})$. Indicate it and use it to list those functions in lex order.

(d) Indicate how all of the parts of this exercise can be interpreted in terms of subsets of a set.

1.5. Modify Theorem 1 to list all vertices in PREV order. Do the same for POSV order.

1.6. The president of Hardy Hibachi Corporation decided to design a series of different grills for his square-topped hibachis. They were to be collectibles. He hoped his customers would want one of each different design (and spend big bucks to get them). Having studied combinatorics in college, his undergrad summer intern suggested that these grills be modeled after the patterns associated with domino arrangements on 4×4 boards. Their favorite grill was in the design which has the code vvhvvhhh. The student, looking at some old class notes, suggested seven other designs: vvhhvhvh, hhvvhvvh, vhvhhvvh, hvvhvhvh, hvvvvhhh, vhvhvvhh, hhhvvvh. These eight grills were fabricated out of sturdy steel rods, put in a box, and shipped to the boss. When he opened up the box, much to his disgust, he found that all of the grills were the same. What went wrong? How should the collection of different grills be designed? (This is called an *isomorph rejection* problem.)

The favorite grill: vvhvvhhh = 

Section 2: Recursive Algorithms

A *recursive algorithm* is an algorithm that refers to itself when it is executing. As with any recursive situation, when an algorithm refers to itself, it must be with "simpler" parameters so that it eventually reaches one of the "simplest" cases, which is then done without recursion. Let's look at a couple of examples before we try to formalize this idea.

Example 8 (A recursive algorithm for 0-1 sequences) Suppose you are interested in listing all sequences of length eight, consisting of four zeroes and four ones. Suppose that you have a friend who does this sort of thing, but will only make such lists if the length of the sequence is seven or less. "Nope," he says, "I can't do it — the sequence is too long." There is a way to trick your friend into doing it. First give him the problem of listing all sequences of length seven with three ones. He doesn't mind, and gives you the list 1110000, 1011000, 0101100, etc. that he has made. You thank him politely, sneak off, and put a "1" in front of every sequence in the list he has given you to obtain 11110000, 11011000, 10101100, etc. Now, you return to him with the problem of listing all strings of length seven with four ones. He returns with the list 1111000, 0110110, 0011101, etc. Now you thank him and sneak off and put a "0" in front of every sequence in the list he has given you to obtain 01111000, 00110110, 00011101, etc. Putting these two lists together, you have obtained the list you originally wanted.

How did your friend produce these lists that he gave you? Perhaps he had a friend that would only do lists of length 6 or less, and he tricked this friend in the same way you tricked him! Perhaps the "6 or less" friend had a "5 or less friend" that he tricked, etc. If you are sure that your friend gave you a correct list, it doesn't really matter how he got it. □

Next we consider an example from sorting theory. We imagine we are given a set of objects which have a linear order described on them (perhaps, but not necessarily, lexicographic order of some sort). As a concrete example, we could imagine that we are given a set of integers S, perhaps a large number of them. They are not in order as presented to us, be we want to list them in order, smallest to largest. That problem of putting the set S in order is called *sorting* S. On the other hand, if we are given two ordered lists, like $(25, 235, 2333, 4321)$ and $(21, 222, 2378, 3421, 5432)$, and want to put the combined list in order, in this case $(21, 25, 222, 235, 2333, 2378, 3421, 4321, 5432)$, this process is called *merging* the two lists. Our next example considers the relationship between sorting and merging.

Example 9 (Sorting by recursive merging) Sorting by recursive merging, called *merge sorting*, can be described as follows.

- The lists containing just one item are the simplest and they are already sorted.
- Given a list of $n > 1$ items, choose k with $1 \le k < n$, sort the first k items, sort the last $n - k$ items and merge the two sorted lists.

This algorithm builds up a way to sort an n-list out of procedures for sorting shorter lists. Note that we have not specified how the first k or last $n-k$ items are to be sorted, we simply assume that it has been done. Of course, an obvious way to do this is to simply apply our merge sorting algorithm to each of these sublists.

Let's implement the algorithm using people rather than a computer. Imagine training a large number of obedient people to carry out two tasks: (a) splitting a list for other people to sort and (b) merging two lists. We give one person the unsorted list and tell him to sort it using the algorithm and return the result to us. What happens?

- Anyone who has a list with only one item returns it unchanged to the person he received it from.
- Anyone with a list having more than one item splits it and gives each piece to a person who has not received a list, telling each person to sort it and return the result. When the results have been returned, this person merges the two lists and returns the result to whoever gave him the list.

If there are enough obedient people around, we'll eventually get our answer back.

Notice that *no one needs to pay any attention to what anyone else is doing* to a list. This is makes a *local description* possible; that is, we tell each person what to do and they do not need to concern themselves with what other people are doing. This can also be seen in the pseudocode for merge sorting a list L:

```
Sort(L)
    If length is 1, return L
    Else
        Split L into two lists L1 and L2
        S1 = Sort(L1)
        S2 = Sort(L2)
        S = Merge(L1, L2)
        Return S
    End if
End
```

The procedure is not concerned with what goes on when it calls itself recursively. This is very much like proof by induction. (We discuss proof by induction in the last section of this unit.) To see that, let's prove that the algorithm sorts correctly. We assume that splitting and merging have been shown to be correct — that's a separate problem. We induct on the length n of the list. The base case, $n = 1$ is handled correctly by the program since it returns the list unchanged. Now for induction. Splitting L results in shorter lists and so, by the induction hypothesis, S1 and S2 are sorted. Since merging is done correctly, S is also sorted.

This algorithm is another case of divide and conquer since it splits the sorting problem into two smaller sorting problems whose answers are combined (merged) to obtain the solution to the original sorting problem. $\square$

Let's summarize some of the above observations with two definitions.

Definition 2 (Recursive approach) *A recursive approach to a problem consists of two parts:*

1. *The problem is reduced to one or more problems of the same kind which are simpler in some sense.*
2. *There is a set of simplest problems to which all others are reduced after one or more steps. Solutions to these simplest problems are given.*

The preceding definition focuses on tearing down (reduction to simpler cases). Sometimes it may be easier or better to think in terms of building up (construction of bigger cases):

Definition 3 (Recursive solution) *We have a recursive solution to the problem (proof, algorithm, data structure, etc.) if the following two conditions hold.*

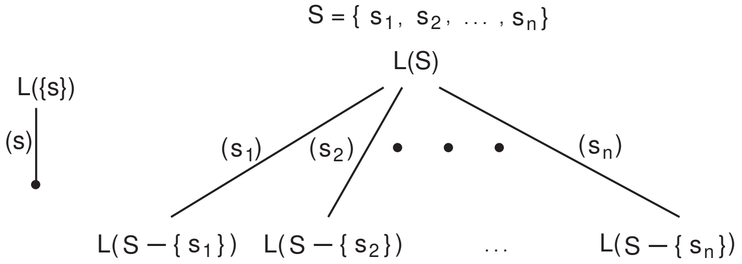
1. *The set of simplest problems can be dealt with (proved, calculated, sorted, etc.).*
2. *The solution to any other problem can be built from solutions to simpler problems, and this process eventually leads back to the original problem.*

The recursion $C(n, k) = C(n-1, k-1) + C(n-1, k)$ for computing binomial coefficients can be viewed as a recursive algorithm. Such algorithms for computing can be turned into algorithms for constructing the things we are counting. To do this, it helps to have a more systematic way to think about recursive algorithms. In the next example we introduce a tree to represent the local description of a recursive algorithm.

Example 10 (Permutations in lex order) The following figure represents the *local description* of a decision tree for listing the permutations of an ordered set

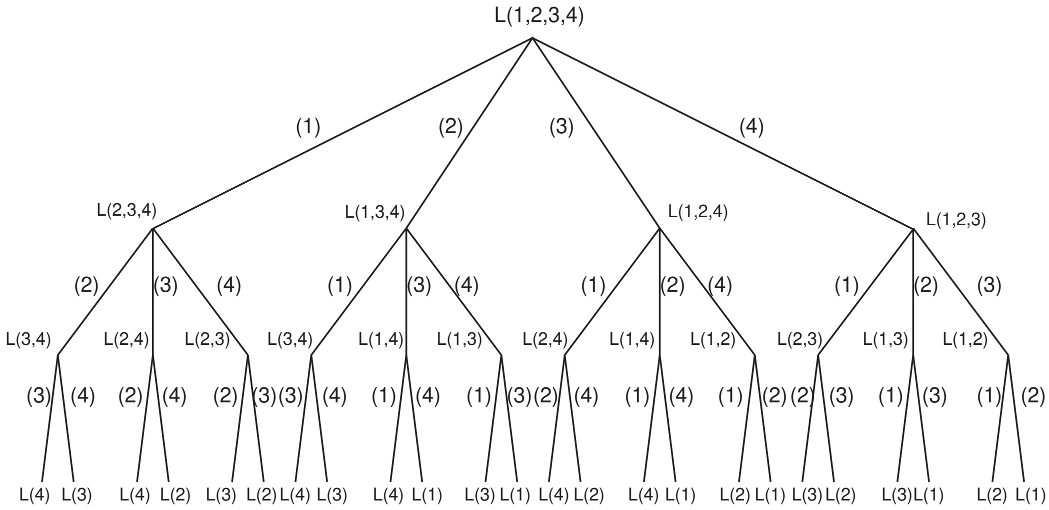
$$\mathrm{S} = \{\mathrm{s}_1, \mathrm{s}_2, \ldots, \mathrm{s}_\mathrm{n}\} \quad \text{with} \quad \mathrm{s}_1 < \mathrm{s}_2 < \cdots < \mathrm{s}_\mathrm{n}.$$

The permutations in the figure are listed in one-line form. The vertices of this decision tree are of the form L(X) where X is some set. The simplest case, shown below, is where the tree has one edge. The labels on the edges are of the form (t), where t is an element of the set X associated with the uppermost vertex L(X) incident on that edge.



The leaves of the recursive tree tell us to construct permutations of the set S with the already chosen element removed from the set. (This is because permutations are injections.)

One way to think of the local description is to regard it as a rule for recursively constructing an entire decision tree, once the set S is specified. Here this construction has been carried out for $S = \{1, 2, 3, 4\}$.

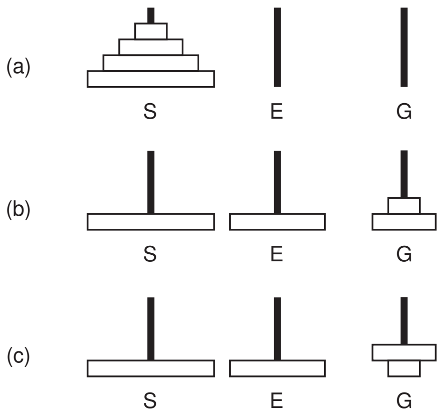


To obtain a permutation of $\underline{4}$, read the labels (t) on the edges from the root to a particular leaf. For example the if this is done for the preorder first leaf, one obtains (1)(2)(3)L(4). L(4) is a "simplest case" and has the label (4), giving the permutation 1234 in one line notation. Repeating this process for the leaves from left to right gives the list of permutations of $\underline{4}$ in lex order. For example, the tenth leaf gives the permutation 2341.

We'll use induction to prove that this is the correct tree. When $n = 1$, it is clear. Suppose it is true for all S with cardinality less than n. The permutations of S in lex order are those beginning with s_1 followed by those beginning with s_2 and so on. If s_k is removed from those permutations of S beginning with s_k, what remains is the set of permutations of $S - \{s_k\}$ listed in lex order. By the induction hypothesis, these are given by $L(S - \{s_k\})$. Note that the validity of our proof does not depend on how they are given by $L(S - \{s_k\})$. $\square$

No discussion of recursion would be complete without the entertaining example of the Towers of Hanoi puzzle. We shall explore additional aspects of this problem in the exercises. Our approach will be the same as the previous example. We shall give a local description of the recursion. Having done so, we construct the trees for some examples and try to gain insight into the sequence of moves associated with the general Towers of Hanoi problem.

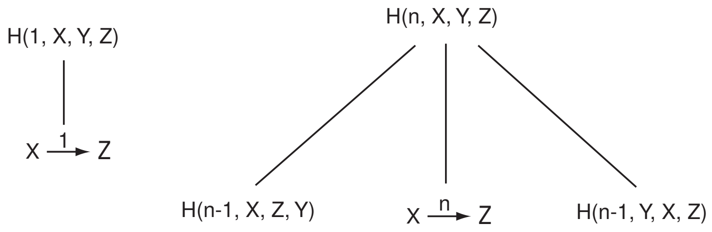
Example 11 (Towers of Hanoi) The *Towers of Hanoi* puzzle consists of n different sized washers (i.e., discs with holes in their centers) and three poles. Initially the washers are stacked on one pole as shown below.



The object is to switch all of the washers from the pole S to G using pole E as a place for temporary placement of discs. A legal move consists of taking the top washer from a pole and placing on top of the pile on another pole, provided it is not placed on a smaller washer. Configuration (a), above, is the starting configuration, (b) is an intermediate stage, and (c) is illegal.

We want an algorithm H(n, S, E, G) that takes washers numbered $1, 2, \ldots, n$ that are stacked on the pole called S and moves them to the pole called G. The pole called E is also available. A call of this procedure to move 7 washers might be H(7,"start","extra","goal").

Here is a recursive description of how to solve the Towers of Hanoi. To move the largest washer, we must move the other $n-1$ to the spare peg. After moving the largest, we can then move the other $n-1$ on top of it. Let the washers be numbered 1 to n from smallest to largest. When we are moving any of the washers 1 through k, we can ignore the presence of all larger washers beneath them. Thus, moving washers 1 through $n-1$ from one peg to another when washer n is present uses the same moves as moving them when washer n is not present. Since the problem of moving washers 1 through $n-1$ is simpler, we practically have a recursive description of a solution. All that's missing is the observation that the simplest case, $n = 1$, is trivial. The following diagram gives the local description of a decision tree that represents this recursive algorithm.



The "simplest case," n equal 1 is shown on the left. The case for general n is designated by H(n, X, Y, Z). You can think of the symbol H(n, X, Y, Z) as designating a vertex of a decision tree. The local description tells you how to construct the rest of the decision tree, down to and including the simplest cases. There is a simple rule for deciding how to rearrange X, Y and Z:

- for the left child: X is fixed, Y and Z are switched;
- for the right child: Z is fixed, X and Y are switched.

All leaves of the decision tree are designated by symbols of the form "$U \xrightarrow{k} V$." This symbol has the interpretation "move washer number k from pole U to pole V." These leaves in preorder (left to right order in the tree) give the sequence of moves needed to solve the Towers of Hanoi puzzle. The local description tells us that, in order to list the leaves of H(n, S, E, G), we

- list the leaves of H(n-1, S, G, E), moving the top n-1 washers from S to E using G
- move the largest washer from S to G
- list the leaves of H(n-1, E, S, G), moving the top n-1 washers from E to G using S

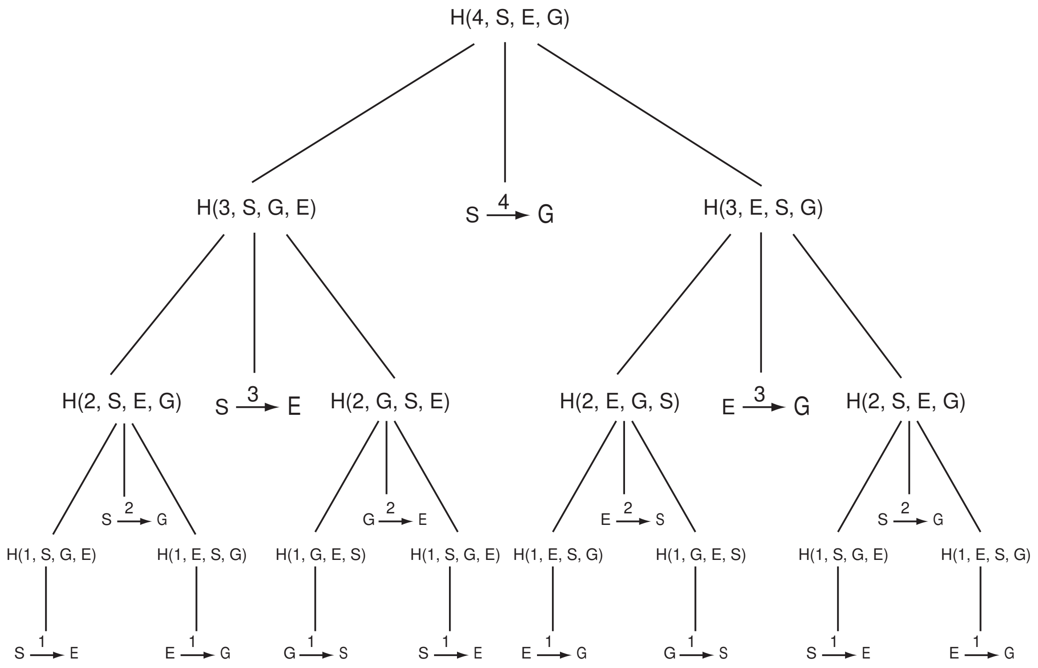
For example, the leaves of the tree with root H(2,S,E,G) are, in order,

$$\mathrm{S} \xrightarrow{1} \mathrm{E}, \quad \mathrm{S} \xrightarrow{2} \mathrm{G}, \quad \mathrm{E} \xrightarrow{1} \mathrm{G}.$$

The leaves of H(3,S,E,G) are gotten by concatenating (piecing together) the leaves of the subtree rooted at H(2,S,G,E) with $\mathrm{S} \xrightarrow{3} \mathrm{G}$ and the leaves of the subtree rooted at H(2,E,S,G). This gives

$$\mathrm{S} \xrightarrow{1} \mathrm{G},\ \mathrm{S} \xrightarrow{2} \mathrm{E},\ \mathrm{G} \xrightarrow{1} \mathrm{E},\ \mathrm{S} \xrightarrow{3} \mathrm{G},\ \mathrm{E} \xrightarrow{1} \mathrm{S},\ \mathrm{E} \xrightarrow{2} \mathrm{G},\ \mathrm{S} \xrightarrow{1} \mathrm{G}. \qquad \square$$

Example 12 (The Towers of Hanoi decision tree for $n = 4$) Starting with the local description of the general decision tree for the Towers of Hanoi and applying the rules of construction specified by it, we obtain the decision tree for the Towers of Hanoi puzzle with $n = 4$. For example, we start with n = 4, X = S, Y = E and Z = G at the root of the tree. To match the H(n,X,Y,Z) pattern when we expand the rightmost son of the root (namely H(3,E,S,G)), we have n = 3, X = E, Y = S and Z = G.



There are fifteen leaves. You should apply sequentially the moves specified by these leaves to the starting configuration ((a) of Example 11) to see that the rules are followed and the washers are all transferred to G.

There are some observations we can make from this example. There are $2^4 - 1 = 15$ leaves or, equivalently, "moves" in transferring all washers from S to G. If h_n is the number of moves required for n washers, then the local description of the decision tree implies that, in general, $h_n = 2h_{n-1} + 1$. Computing some numbers for the h_n gives 1, 3, 7, 15, 31, etc. It appears that $h_n = 2^n - 1$. This fact can be proved easily by induction.

Note that the washer number 1, the smallest washer moves every other time. It moves in a consistent pattern. It starts on S, then E, then G, then S, then E, then G, etc. For H(3, S, E, G), the pattern is S, G, E, S, G, E, etc. In fact, for n odd, the pattern is always S, G, E, S, G, E, etc. For n even, the pattern is always S, E, G, S, E, G, etc. This means that if someone shows you a configuration of washers on discs for the H(n, S, E, G) and says to you, "It's the smallest washer's turn to move," then you should be able to make the move. If they tell you it is not the smallest washer's turn, then you should also be able to make the move. Why? Only one move *not* involving the smallest washer is legal! $\square$

Example 13 (The Towers of Hanoi, recursion, and stacks) One way to generate the moves of H(n, S, E, G) is to use the local description to generate the depth first vertex sequence (Example 5).

- The depth first vertex list for $n = 4$ would start as follows:

  $$\mathrm{H(4, S, E, G)},\ \ \mathrm{H(3, S, G, E)},\ \ \mathrm{H(2, S, G, E)},\ \ \mathrm{H(1, S, G, E)},\ \ \mathrm{S} \xrightarrow{1} \mathrm{E}$$

  At this point we have gotten to the first leaf. It should be printed out.

- The next vertex in the depth first vertex sequence is H(1, S, G, E) again. We represent this by removing $\mathrm{S} \xrightarrow{1} \mathrm{E}$ to get

  $$\mathrm{H(4, S, E, G)},\ \ \mathrm{H(3, S, G, E)},\ \ \mathrm{H(2, S, G, E)},\ \ \mathrm{H(1, S, G, E)}.$$

- Next we remove H(1, S, G E) to get

  $$\mathrm{H(4, S, E, G)},\ \ \mathrm{H(3, S, G, E)},\ \ \mathrm{H(2, S, G, E)}.$$

- The next vertex in depth first order is $\mathrm{S} \xrightarrow{2} \mathrm{G}$. We add this to our list to get

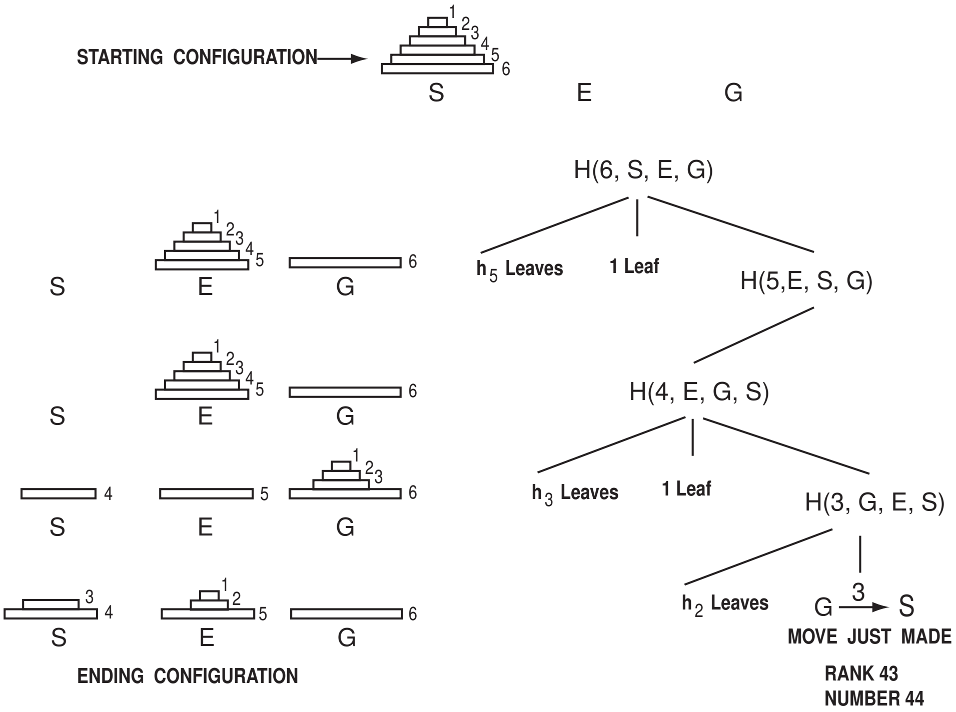
  $$\mathrm{H(4, S, E, G), H(3, S, G, E), H(2, S, G, E), S} \xrightarrow{2} \mathrm{G}.$$

Continuing in this manner we generate, for each vertex in the decision tree, the path from the root to that vertex. The vertices occur in depth first order. Computer scientists refer to the path from the root to a vertex v as the *stack* of v. Adding a vertex to the stack is called *pushing* the vertex on the stack. Removing a vertex is *popping* the vertex from the stack. Stack operations of this sort reflect how most computers carry out recursion. This "one dimensional" view of recursion is computer friendly, but the geometric picture provided by the local tree is more people friendly. $\square$

Example 14 (The Towers of Hanoi configuration analysis) In the figure below, we show the starting configuration for H(6, S, E, G) and a path,

$$\text{H}(6,\ \text{S},\ \text{E},\ \text{G}),\ \ \text{H}(5,\ \text{E},\ \text{S},\ \text{G}),\ \ \text{H}(4,\ \text{E},\ \text{G},\ \text{S}),\ \ \text{H}(3,\ \text{G},\ \text{E},\ \text{S}),\ \ \text{G} \xrightarrow{3} \text{S}.$$

This path goes from the root H(6, S, E, G) to the leaf G $\xrightarrow{3}$ S. Given this path, we want to construct the configuration of washers corresponding to that path, assuming that the move G $\xrightarrow{3}$ S has just been carried out. This is also shown in the figure and we now explain how we obtained it.



The first part of the path shows what happens when we use the local tree for H(6,S,E,G). Since we are going to H(5,E,S,G), the first edge of the path "slopes down to the right." At this point, the left edge, which led to H(5,S,G,E) moved washers 1 through 5 to pole E using $h_5 = 2^5 - 1 = 31$ moves and move S $\xrightarrow{6}$ G has moved washer 6. This is all shown in the figure and it has taken $31 + 1 = 32$ moves.

Next, one replaces H(5,E,S,G) with the local tree (being careful with the S, E, G labels!). This time the path "slopes to the left". Continuing in this manner we complete the entire path and have the configuration that is reached.

We can compute the rank of this configuration by noticing how many moves were made to reach it. Each move, except our final one G $\xrightarrow{3}$ S, is a leaf to the left of the leaf corresponding to the move G $\xrightarrow{3}$ S at the end of the path. We can see from the figure that there were

$$(h_5 + 1) + 0 + (h_3 + 1) + h_2 = (31 + 1) + (7 + 1) + 3 = 43$$

such moves and so the rank of this configuration is 43.

You should study this example carefully. It represents a very basic way of studying recursions. In particular

(a) You should be able to do the same analysis for a different path.

(b) You should be able to start with an ending configuration and reconstruct the path.

(c) You should be able to start with a configuration and, by attempting to reconstruct the path (and failing), be able to show that the configuration is illegal (can never arise).

We've already discussed (a). When you know how to do (b), you should be able to do (c). How can we reconstruct the path from the ending configuration? Look at the final configuration in the previous figure and note where the largest washer is located. Since it is on G, it must have been moved from S to G. This can only happen on the middle edge leading out from the root. Hence we must take the rightmost branch and are on the path that starts H(5,E,S,G). We are now dealing with a configuration where washers 1–5 start out stacked on E, washer 6 is on G and washer 6 will never move again. This takes care of washer 6 and we ignore it from now on. We are now faced with a new problem: There are 5 washers starting on E. In the final configuration shown in the figure, washer 5 is still on E, so it has not moved in this new problem. Therefore, we must take leftmost branch from H(5,E,S,G). This is H(4,E,G,S). Again, we have a new problem with 4 washers starting out on E. Since washer 4 must end up on S, we take the rightmost branch out of the vertex H(4,E,G,S). We continue in this manner until we reach H(1,...). If washer 1 must move, that is the last move. Otherwise, the last move is the leaf to the left of the last right-pointing branch that we took. In our particular case, from H(4,E,G,S) we go right to H(3,G,E,S), right to H(2,E,G,S), left to H(1,E,S,G). Since washer 1 is on E, it has not yet moved in doing H(1,E,S,G). (Of course, it may have moved several times earlier.) The last right branch was H(2,E,G,S) from H(3,G,E,S) so that the last move was washer 3 from G to S.

How could the previous process ever fail? Not all configurations arise. For example, if washer 5 were on S, we would have decided to move it in H(5,E,S,G) since it is not on E. But the only time H(5,E,S,G) moves washer 5 is from E to G so it cannot end up on S. $\square$

We conclude this section with another "canonical" example of recursive algorithm and decision trees. We want to look at all subsets of $\underline{n}$. It will be more convenient to work with the representation of subsets by functions with domain $\underline{n}$ and range $\{0,1\}$. For a subset S of $\underline{n}$, define

$$\chi_S(i) \;=\; \begin{cases} 1 & \text{if } i \in S,\\ 0 & \text{if } i \notin S.\end{cases}$$

This function is called the *characteristic function* of S. We have a characteristic function for every subset S of $\underline{n}$. In one line notation, these functions become n-strings of zeroes and ones: The string $a_1 \ldots a_n$ corresponds to the subset T where $i \in T$ if and only if $a_i = 1$. Thus the all zeroes string corresponds to the empty set and the all ones string to $\underline{n}$. This correspondence is called the *characteristic function* interpretation of subsets of $\underline{n}$.

Our goal is to make a list of all subsets of $\underline{n}$ such that subsets adjacent to each other in the list are "close" to each other. Before we can begin to look for such a *Gray code*, we must say what it means for two subsets (or, equivalently, two strings) to be close. Two strings will be considered close if they differ in exactly one position. In set terms, this means one of the sets can be obtained from the other by removing or adding a single element. With this notion of closeness, a Gray code for all subsets when $n = 1$ is 0, 1. A Gray code for all subsets when $n = 2$ is 00, 01, 11, 10.

How can we produce a Gray code for all subsets for arbitrary n? There is a simple recursive procedure. The following construction of the Gray code for $n = 3$ illustrates it.

0 00	1 10
0 01	1 11
0 11	1 01
0 10	1 00

You should read down the first column and then down the second. Notice that the sequences in the first column begin with 0 and those in the second with 1. The rest of the first column is simply the Gray code for $n = 2$ while the second column is the Gray code for $n = 2$, read from the last sequence to the first.

We now prove that this two column procedure for building a Gray code for subsets of an n-set from the Gray code for subsets of an $(n-1)$-set always works. Our proof will be by induction. For $n = 1$, we have already exhibited a Gray code. Suppose that $n > 1$ and that we have a Gray code for $n - 1$. (This is the induction assumption.)

- Between the bottom of the first column and the top of the second, the only change is in the first position since the remaining $n - 1$ positions are the last element of our Gray code for $n - 1$.
- Within a column, there is never any change in the first position and there is only a single change from line to line in the remaining positions because they are a Gray code by the induction assumption.

This completes the proof.

As an extra benefit, we note that the last element of our Gray code differs in only one position from the first element (Prove it!), so we can cycle around from the last element to the first by a single change.

Example 15 (Decision tree for the subset Gray code) Here is another notation for describing our subset Gray code. Let $\overrightarrow{\mathrm{GRAY}}(1) = 0, 1$, and let $\overleftarrow{\mathrm{GRAY}}(1) = 1, 0$. As the arrows indicate, $\overrightarrow{\mathrm{GRAY}}(1)$ is the Gray code for $n = 1$ listed from first to last element, while $\overleftarrow{\mathrm{GRAY}}(1)$ is this Gray code in reverse order. In general, if $\overrightarrow{\mathrm{GRAY}}(n)$ is the Gray code for n-bit words, then $\overleftarrow{\mathrm{GRAY}}(n)$ is defined to be that list in reverse order.

We define

$$\overrightarrow{\mathrm{GRAY}}(2) = 0\overrightarrow{\mathrm{GRAY}}(1), 1\overleftarrow{\mathrm{GRAY}}(1).$$

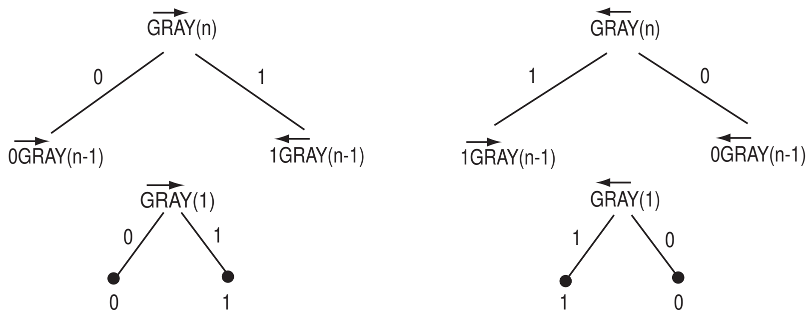
The meaning of $0\overrightarrow{\mathrm{GRAY}}(1)$ is that 0 is put at the front of every string in $\overrightarrow{\mathrm{GRAY}}(1)$. Juxtaposing the two lists (or "concatenation") means just listing the second list after the first. Thus, $0\overrightarrow{\mathrm{GRAY}}(1) = 00, 01$, and $1\overleftarrow{\mathrm{GRAY}}(1) = 11, 10$. Hence,

$$\overrightarrow{\mathrm{GRAY}}(2) = 0\overrightarrow{\mathrm{GRAY}}(1), 1\overleftarrow{\mathrm{GRAY}}(1) = 00, 01, 10, 11\,.$$

If we read $\overrightarrow{\text{GRAY}}(2)$ in reverse order, we obtain $\overleftarrow{\text{GRAY}}(2)$. You should verify the following equality.

$$\overleftarrow{\text{GRAY}}(2) = 1\overrightarrow{\text{GRAY}}(1), 0\overleftarrow{\text{GRAY}}(1).$$

What we did for $n = 2$ works in general: The following diagram gives the local description of a decision tree for constructing subset Gray codes:

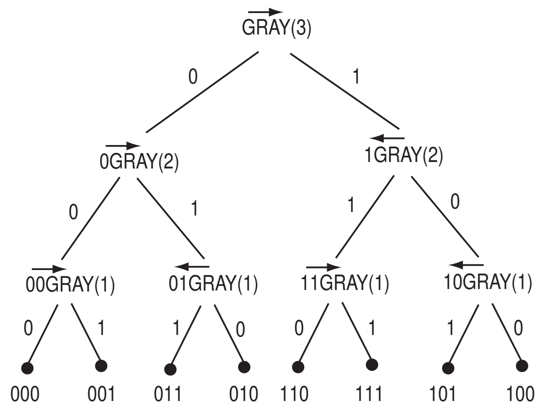


The left side of the figure is a *definition* for $\overrightarrow{\text{GRAY}}(n)$. We must verify two things:

- This definition gives us a Gray code.
- Given the left figure and the fact that $\overleftarrow{\text{GRAY}}(n)$ is the reversal of $\overrightarrow{\text{GRAY}}(n)$, the right figure is correct.

The first part was already done because the figure simply describes the construction we gave before we started this example. The second part is easy when we understand what the tree means. Reading the $\overrightarrow{\text{GRAY}}(n)$ tree from the right, we start with the reversal of $1\overleftarrow{\text{GRAY}}(n-1)$. Since $\overleftarrow{\text{GRAY}}$ and $\overrightarrow{\text{GRAY}}$ are defined to be reversals of each other, we get $1\overrightarrow{\text{GRAY}}(n-1)$. Similarly, reversing $0\overrightarrow{\text{GRAY}}(n-1)$ gives $0\overleftarrow{\text{GRAY}}(n-1)$.

If we apply the local description to the case $n = 3$, we obtain the following decision tree:



In the above decision tree for $\overrightarrow{\text{GRAY}}(3)$, the elements of the Gray code for $n = 3$ are obtained by listing the labels on the edges for each path that ends in a leaf. These paths are listed in preorder of their corresponding leaves (left to right in the picture). This gives 000, 001, 011, 010, 110, 111, 101, 100. You should practice doing the configuration analysis for this recursion, analogous to Example 14. In particular, given a sequence, 10011101 say, construct its path in the decision tree. What is the RANK of 10011101 in $\overrightarrow{\text{GRAY}}(8)$? What is the element in $\overrightarrow{\text{GRAY}}(8)$ just before 10011101; just after 10011101?

Note in the above decision tree for $\overrightarrow{\text{GRAY}}(3)$ that every time an edge with label 1 is encountered (after the first such edge), that edge changes direction from the edge just prior to it in the path. By "changing direction," we mean that if an edge is sloping downward to the right (downward to the left) and the previous edge in the path sloped downward to the left (downward to the right), then a change of direction has occurred. Conversely, every time an edge with label 0 is encountered (after the first such edge), that edge does not change direction from the edge just prior to it in the path. This is a general rule that can be proved by induction. □

Exercises for Section 2

2.1. Suppose the permutations on $\underline{8}$ are listed in lexicographic order.

(a) What is the RANK in the list of all such permutations of 87612345?

(b) What permutation has RANK 20,160?

2.2. Consider the Towers of Hanoi puzzle, H(8, S, E, G). Suppose that pole S has washers 6, 5, 2, 1; pole E has no washers; pole G has washers 8, 7, 4, 3. Call this the *basic configuration*.

(a) What is the path in the decision tree that corresponds to the basic configuration?

(b) What was the move that produced the basic configuration and what was the configuration from which that move was made?

(c) What was the move just prior to the one that produced the basic configuration and what was the configuration from which that move was made?

(d) What will be the move just after the one that produced the basic configuration?

(e) What is the RANK, in the list of all moves of H(8, S, E, G), of the move that produced the basic configuration?

2.3. Consider $\overrightarrow{\text{GRAY}}(9)$.

(a) What is the element just before 110010000? just after 110010000?

(b) What is the first element of the second half of the list?

(c) What is the RANK of 111111111?

(d) What is the element of RANK 372?

***2.4.** Consider the Towers of Hanoi puzzle with four poles and n washers. The rules are the same, except that there are two "extra" poles E and F. The problem is to transfer all of the n washers from S to G using the extra poles E and F as temporary storage. Let h'_n denote the optimal number of moves needed to solve the three pole problem. Let f_n denote the optimal number of moves needed to solve the four pole problem with n washers.

(a) Recall that $h_n = 2^n - 1$ is the number of moves in the recursive algorithm H(n, S, E, G). Prove by induction that $h'_n = h_n$.

(b) Compute f_n for $n = 1, 2, 3$, describing, in the process, optimal sequences of moves.

Let's adopt a specific strategy for doing four poles and n washers. Choose integers $p \geq 0$ and $q > 0$ so that $p + q = n$. We now describe strategy G(p, q, S, E, F, G). To execute G(p, q, S, E, F, G), proceed as follows:

(i) If $p = 0$, then $q = n$. Use H(n, S, E, G) to move washers $1, \ldots, n$ to G.

(ii) If $p > 0$, choose integers $i \geq 0$ and $j > 0$ such that $i + j = p$. Use G(i, j, S, E, G, F) to move washers $1, 2, \ldots, p$ to pole F (the washers are numbered in order of size). Next, use H(q, S, E, G) to move washers q, . . . , n to G. Finally, use G(i, j, F, S, E, G) to move $1, 2, \ldots, p$ to pole G, completing the transfer. For all possible choices of i and j, choose the one that minimizes the number of moves.

Finally, to move the n washers, choose that G(p, q, S, E, F, G) with $n = p + q$ which has the minimum number of moves. Call this number s_n.

(c) What are the simplest cases in this recursive algorithm? How can you compute the values of i and j to minimize the number of moves? Use your method to solve the problem for $n \leq 6$.

(d) What is the recursion for s_n?

(e) Prove that $f_n \leq 2\min(f_{n-q} + h_q)$, where the minimum is over $q > 0$ and $f_0 = 0$. recursion.

Section 3: Decision Trees and Conditional Probability

We conclude our discussion of decision trees by giving examples of the use of decision trees in elementary probability theory. In particular, we focus on what are called *conditional probabilities* and *Bayesian methods* in probability.

Definition 4 (Conditional probability) *Let U be a sample space with probability function P. If $A \subseteq U$ and $B \subseteq U$ are events (subsets) of U then the conditional probability of B given A, denoted by $P(B|A)$, is*

$$P(B|A) = \begin{cases} P(A \cap B)/P(A), & \text{if } P(A) \neq 0, \\ \text{undefined}, & \text{if } P(A) = 0. \end{cases}$$

How should we interpret $P(B|A)$? If an experiment is performed n times and b of those times B occurs, then b/n is nearly $P(B)$. Furthermore, as n increases, the ratio b/n almost surely approaches $P(B)$ as a limit.* Now suppose an experiment is performed n times but we are only interested in those times when A occurs. Furthermore, suppose we would like to know the chances that B occurs, given that A has occurred. Let the count for A be a and that for $A \cap B$ be c. Since we are interested only in the cases when A occurs, only a of the experiments matter. In these a experiments, B occurred c times. Hence the probability that B occurs given that A has occurred is approximately $c/a = (c/n)/(a/n)$, which is approximately $P(A \cap B)/P(A)$, which is the definition of $P(B|A)$. As n increases, the approximations almost surely approach $P(B|A)$. Hence

> $P(B|A)$ should be thought of as the probability that B occurred, given that we know A occurred.

Another way you can think of this is that we are changing to a new sample space A. To define a probability function P_A on this sample space, we rescale P so that $\sum_{a \in A} P_A(a) = 1$. Since $\sum_{a \in A} P(a) = P(A)$, we must set $P_A(a) = P(a)/P(A)$. Then, the probability that B occurs is the sum of $P_A(a)$ over all $a \in B$ that are in our new sample space A. Thus

$$P_A(B) = \sum_{a \in A \cap B} P_A(a) = \sum_{a \in A \cap B} P(a)/P(A) = P(A \cap B)/P(A),$$

which is our definition of $P(B|A)$.

The following theorem contains some simple but important properties of conditional probability.

Theorem 2 (Properties of conditional probability) *Let (U, P) be a probability space. All events in the following statements are subsets of U and the conditional probabilities are assumed to be defined. (Recall that $P(C|D)$ is undefined when $P(D) = 0$.)*

(a) $P(B|U) = P(B)$ *and* $P(B|A) = P(A \cap B \mid A)$.

(b) A *and* B *are independent events if and only if* $P(B|A) = P(B)$.

(c) (Bayes' Theorem) $P(A|B) = P(B|A)P(A)/P(B)$.

(d) $P(A_1 \cap \cdots \cap A_n) = P(A_1)\, P(A_2 \mid A_1)\, P(A_3 \mid A_1 \cap A_2) \cdots P(A_n \mid A_1 \cap \cdots \cap A_{n-1})$.

You can think of (b) as a justification for the terminology "independent" since $P(B|A) = P(B)$ says that the probability of B having occurred is unchanged even if we know that A occurred; in other words, A does not influence B's chances. We will encounter other forms of Bayes' Theorem. All of them involve reversing the order in conditional probabilities. (Here, $A|B$ and $B|A$.)

Proof: All the proofs are simple applications of the definition of conditional probability, so we prove just (b) and (d) and leave (a) and (c) as exercises.

* For example, we might toss fair coin 100 times and obtain 55 heads, so $a/n; = 55/100$ is nearly $1/2 = P(\text{head})$. With 10,000 tosses, we might obtain 4,930 heads and 4,930/10,000 is even closer to 1/2 than 55/100. (This is the sort of accuracy one might realistically expect.)

We prove (b). Suppose A and B are independent. By the definition of independence, this means that $P(A \cap B) = P(A)P(B)$. Dividing both sides by $P(A)$ and using the definition of conditional probability, we obtain $P(B|A) = P(B)$. For the converse, suppose $P(B|A) = P(B)$. Using the definition of conditional probability and multiplying by $P(A)$, we obtain $P(A \cap B) = P(A)P(B)$, which is the definition of independence.

We prove (d) simply by using the definition of conditional probability and doing a lot of cancellation of adjacent numerators and denominators:

$$\begin{aligned} & P(A_1)\, P(A_2 \mid A_1)\, P(A_3 \mid A_1 \cap A_2) \cdots P(A_n \mid A_1 \cap \cdots \cap A_{n-1}) \\ & \quad = P(A_1)\, \frac{P(A_2 \cap A_1)}{P(A_1)}\, \frac{P(A_1 \cap A_2 \cap A_3)}{P(A_1 \cap A_2)} \cdots \frac{P(A_1 \cap \cdots \cap A_n)}{P(A_1 \cap \cdots \cap A_{n-1})} \\ & \quad = P(A_1 \cap \cdots \cap A_n). \end{aligned}$$

This completes the proof.

An alternative proof of (d) can be given by induction on n. For $n = 1$, (d) becomes $P(A_1) = P(A_1)$, which is obviously true. For $n > 1$ we have

$$\begin{aligned} & P(A_1)\, P(A_2 \mid A_1) \cdots P(A_{n-1} \mid A_1 \cap \cdots \cap A_{n-2})\, P(A_n \mid A_1 \cap \cdots \cap A_{n-1}) \\ & \quad = \Big(P(A_1)\, P(A_2 \mid A_1) \cdots P(A_{n-1} \mid A_1 \cap \cdots \cap A_{n-2})\Big) P(A_n \mid A_1 \cap \cdots \cap A_{n-1}) \\ & \quad = P(A_1 \cap \cdots \cap A_{n-1}) P(A_n \mid A_1 \cap \cdots \cap A_{n-1}) && \text{by induction} \\ & \quad = P(A_1 \cap \cdots \cap A_n) && \text{Definition 4.} \end{aligned}$$

This completes the proof. □

Example 16 (Diagnosis and Bayes' Theorem) Suppose we are developing a test to see if a person has a disease, say the dreaded wurfles. It's known that 1 person in about 500 has the wurfles. To measure the effectiveness of the test, we tried it on a lot of people. Of the 87 people with wurfles, the test always detected it, so we decide it is 100% effective at detection. We also tried the test on a large number of people who do not have wurfles and found that the test incorrectly told us that they have wurfles 3% of the time. (These are called "false positives.")

If the test is released for general use, what is the probability that a person who tests positive actually has wurfles?

Let's represent our information mathematically. Our probability space will be the general population with the uniform distribution. The event W will correspond to having wurfles and the event T will correspond to the test being positive. Our information can be written

$$P(W) = 1/500 = 0.002 \qquad P(T|W) = 1 \qquad P(T|W^c) = 0.03,$$

and we are asked to find $P(W|T)$. Bayes' formula (Theorem 2(c)) tells us

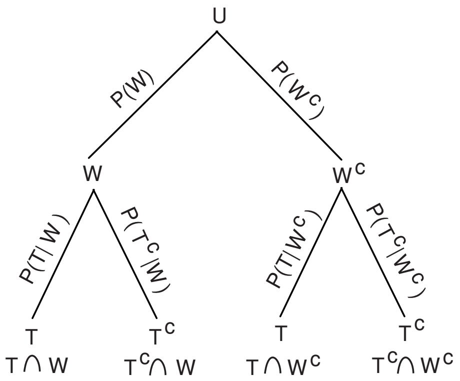
$$P(W|T) = \frac{P(T|W)\, P(W)}{P(T)}.$$

Everything on the right is known except $P(T)$. How can we compute it? The idea is to partition T using W and then convert to known conditional probabilities:

$$\begin{aligned} P(T) &= P(T \cap W) + P(T \cap W^c) && \text{partition } T \\ &= P(T|W)P(W) + P(T|W^c)P(W^c) && \text{convert to conditional} \\ &= 1 \times 0.002 + 0.03 \times (1 - 0.002) \approx 0.032, \end{aligned}$$

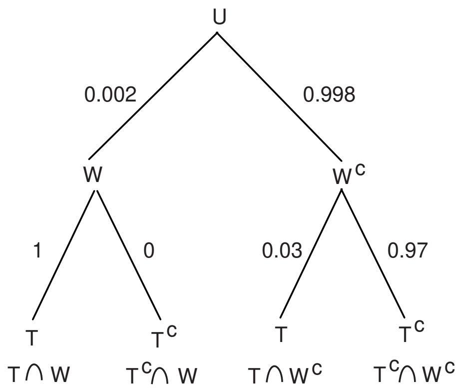
where we have rounded off. Thus $P(W|T) \approx 1 \times 0.002/0.032 \approx 6\%$. In other words, even if the test is positive, you only have a 6% chance of having wurfles. This shows how misleading a rather accurate test can be when it is used to detect a rare condition. $\square$

Example 17 (Decision trees and conditional probability) We can picture the previous example using a decision tree. We start out with the sample space U at the root. Since we have information about how the test behaves when wurfles are present and when they are absent, the first decision partitions U into W (has wurfles) and W^c (does not have wurfles). Each of these is then partitioned according to the test result, T (test positive) and T^c (test negative). Each edge has the form (A, B) and is labeled with the conditional probability $P(B|A)$. The labels $P(W)$ and $P(W^c)$ are equal to $P(W|U)$ and $P(W^c|U)$ respectively (by Theorem 2(a)). Here is the decision tree for our wurfles test.



If you follow the path (U, W, T), your choices were first W (has wurfles) then T (tests positive). In terms of sets, these choices correspond to the event (i.e., set) $T \cap W$ of all people who both test positive and have wurfles. Accordingly, the leaf that is at the end of this path is labeled with the event $T \cap W$. Similar "event" labels are placed at the other leaves.

Using the definition of conditional probability, you should be able to see that the probability of the event label at a vertex is simply the product of the probabilities on the edges along the path from the root to the vertex. For example, to compute $P(T \cap W^c)$ we multiply $P(W^c)$ and $P(T|W^c) = P(T \cap W^c)/P(W^c)$. Numerically this is $0.998 \times 0.03 \approx 0.03$. To compute $P(T \cap W)$ we multiply $P(W)$ and $P(T|W)$. Numerically this is $0.002 \times 1.0 = 0.002$. Here is the tree with the various numerical values of the probabilities shown.

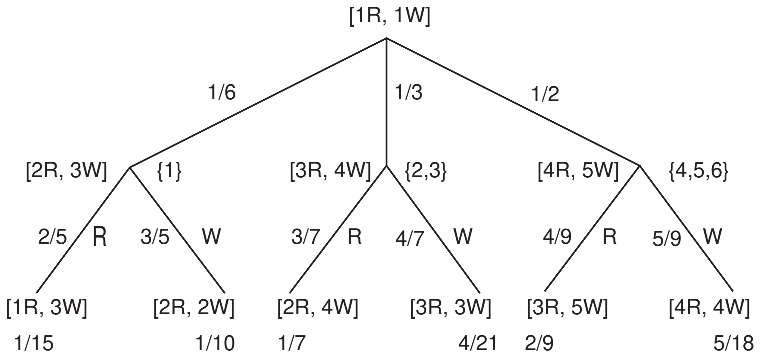


Using the above tree and the computational rules described in the previous paragraph, we can compute $P(W|T) = P(W \cap T)/P(T) = P(T \cap W)/P(T)$ as follows.

1. Compute $P(T)$ by adding up the probabilities of the event labels of *all leaves* that are associated with the decision T. (These are the event labels $T \cap W$ and $T \cap W^c$.) Thus, $P(T) = P(T \cap W) + P(T \cap W^c)$. Using the actual probabilities on the edges of the decision tree we get $P(T) = 0.002 \times 1.0 + 0.998 \times 0.03 \approx 0.032$.
2. Compute $P(W|T)$ using $P(W|T) = P(T \cap W)/P(T)$. Using the computation in step (1), we get $P(W|T) = P(T \cap W)/P(T) = (0.002 \times 1.0)/0.032 \approx 0.06$ These are the same calculations we did in the previous example, so why go to the extra trouble of drawing the tree? The tree gives us a systematic method for recording data and carrying out the calculations. □

In the previous example, each vertex was specifically labeled with the event, such as $W \cap T^c$, associated with it. In the next example, we simply keep track of the information we need to compute our answer.

Example 18 (Another decision tree with probabilities) We are given an urn with one red ball and one white ball. A fair die is thrown. If the number is 1, then 1 red ball and 2 white balls are added to the urn. If the number is 2 or 3, then 2 red balls and 3 white balls are added to the urn. If the number is 4, 5, or 6, then 3 red balls and 4 white balls are added to the urn. A ball is then selected uniformly at random from the urn. We represent the situation in the following decision tree.



The root is represented by the initial composition of the urn. The children of the root [1R, 1W] are [2R, 3W], [3R, 4W], and [4R, 5W]. Beside each of these children of the root is the outcome set of the roll of the die that produces that urn composition: $\{1\}$, $\{2,3\}$, $\{4,5,6\}$. The probabilities on the edges incident on the root are the probabilities of the outcome sets of the die. The probabilities on the edges incident on the leaves are the conditional probabilities as discussed in Example 17. Thus, 3/7 is the conditional probability that the final outcome is R, given that the outcome of the die was in the set $\{2,3\}$.

Here is a typical sort of question asked about this type of probabilistic decision tree: "Given that the ball drawn was red, what is the probability that the outcome of the die was in the set $\{2,3\}$." We could write this mathematically as $P(\{2,3\} \mid \mathrm{R})$, where $\{2,3\}$ represents the result of rolling the die and R represents the result of the draw. Note in this process that the basic data given are conditional probabilities of the form $P(\text{drawing is R} \mid \text{die in S})$. We are computing conditional probabilities of the form $P(\text{die roll in S} \mid \text{drawing is R})$. This is exactly the same situation as in Example 17. Thus our question is answered by carrying out the two steps in Example 17:

1. Add up the probabilities of *all leaves* resulting from the drawing of a red ball to obtain $P(\mathrm{R}) = 1/15 + 1/7 + 2/9 = 136/315$. (The probabilities of the leaves were computed by multiplying along the paths from the root. The results for all leaves are shown in the picture of the decision tree.)
2. Compute the conditional probability $P(\{2,3\} \mid \mathrm{R})$ by dividing $P(\{2,3\} \cap \mathrm{R}) = 1/7$ by $P(\mathrm{R})$. Divide this by the answer from part (1). In this case, we get $(1/7)/(136/315) = 0.331$.

If you wish, you can think of this problem in terms of a new sample space. The elements of the sample space are the leaves. Step 1 (multiplying probabilities along paths) computes the probability function for this sample space. Since an event is a subset of the sample space, an event is a set of leaves and its probability is the sum of the probabilities of the leaves it contains. Can we interpret the nonleaf vertices? Yes. Each such vertex represents an event that consists of the set of leaves below it. Many people prefer this alternative way of thinking about the decision tree. □

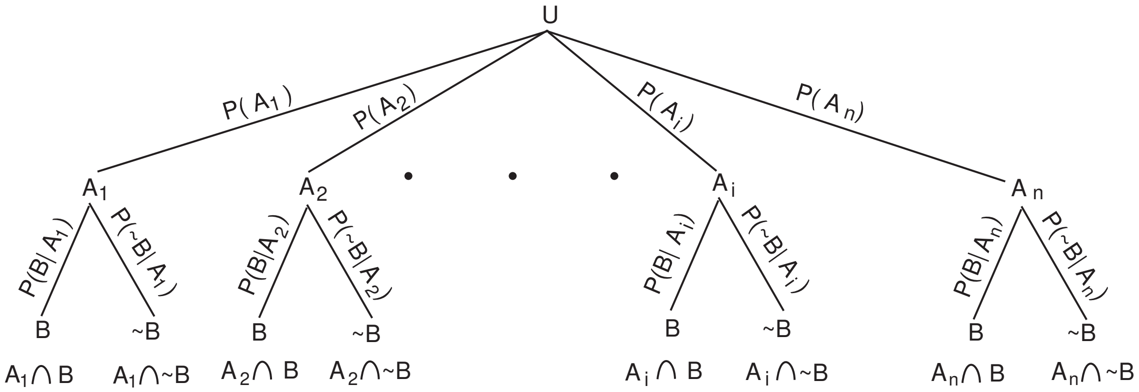
The procedure we used to compute conditional probabilities in Steps 1 to 3 of two previous examples can be stated as a formula, which is another form of Bayes' Theorem:

Theorem 3 (Bayes' Theorem) *Let (U,P) be a probability space, let $\{A_i : i = 1,2,\ldots,n\}$ be a partition of U, and let $B \subset U$. Then*

$$P(A_i|B) = \frac{P(A_i)P(B|A_i)}{\sum_{t=1}^{n} P(A_t)P(B|A_t)}.$$

Most students find decision trees much easier to work with than trying to apply the formal statement of Bayes' theorem. Our proof will closely follow the terminology of Example 17.

Proof: We can draw a decision tree like the ones in the previous examples, but now there are n edges of the decision tree coming down from the root and 2 edges coming down from each child of the root. Here is a decision tree for this generalization:



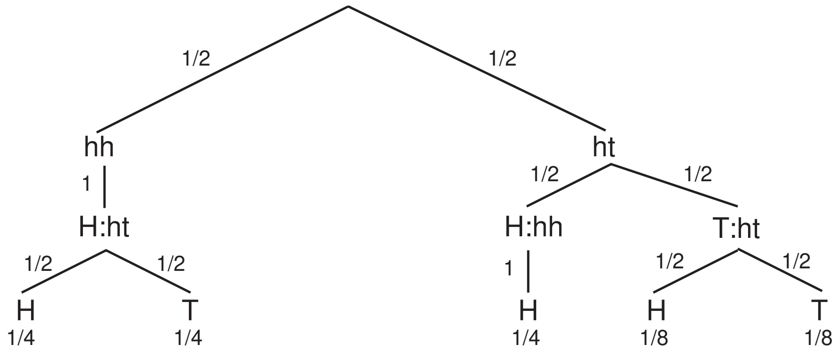
We follow the two step process of Example 17. In doing this, we need to compute, for $1 \le t \le n$, the products of the probabilities along the path passing through the vertex A_t and leading to the leaves labeled by the events $A_t \cap B = B \cap A_t$.

1. Add up the probabilities of *all leaves* contained in B, i.e., add up $P(A_t)P(B|A_t)$ over $1 \le t \le n$ to obtain $P(B)$.
2. Compute $P(A_i|B) = P(A_i \cap B)/P(B)$. Since $P(A_i \cap B) = P(A_i)P(B|A_i)$, this quotient is the formula in the theorem.

This process gives the formula in the theorem. □

All of our probabilistic decision trees discussed thus far have had height two. However, probabilistic decision trees can have leaves at any distance from the root and different leaves may be at different distances. The two step procedure in Example 17 contains no assumptions about the height of leaves and, in fact, will work for all trees. The next example illustrates this.

Example 19 (Tossing coins) Suppose you have two coins. One has heads on both sides and the other is a normal coin. You select a coin randomly and toss it. If the result is heads, you switch coins; otherwise you keep the coin you just tossed. Now toss the coin you're holding. What is the probability that the result of the toss is heads? Here is the decision tree.



The labels hh and ht indicate which coin you're holding — two headed or normal. The labels H and T indicate the result of the toss. A label like H:ht means the toss was H and so I am now holding the ht coin. The conditional probabilities are on the edges and the leaf probabilities were computed by multiplying the probabilities along the paths, as required by Step 1. Adding up, we find that the probability of heads is $1/4 + 1/4 + 1/8 = 5/8$.

Given that the final toss is heads, what is the probability that you're holding the double-headed coin? The leaf where you're holding the double headed coin and tossed a head is the middle leaf, which has probability 1/4, so the answer is $(1/4)/(5/8) = 2/5$.

Given that the final toss is heads, what is the probability that the coin you picked up at the start was not double headed? This is a bit different than what we've done before because there are two leaves associated with this event. Since the formula for conditional probability is

$$\frac{P((\text{chose ht}) \cap (\text{second toss was H}))}{P(\text{second toss was H})},$$

we simply add up the probability of those two leaves to get the numerator and so our answer is $(1/4 + 1/8)/(5/8) = 3/5$.

In the last paragraph we introduced a generalization of our two step procedure: If an event corresponds to more than one leaf, we add up the probability of those leaves. $\square$

---

*Generating Objects at Random

To test complicated algorithms, we may want to run the algorithm on a lot of random problems. Even if we know the algorithm works, we may want to do this to study the speed of the algorithm. Computer languages include routines for generating random numbers. What can we do if we want something more complicated?

In Section 4 of Unit Fn, we gave an algorithm for generating random permutations. Here we show how to generate random objects using a decision tree.

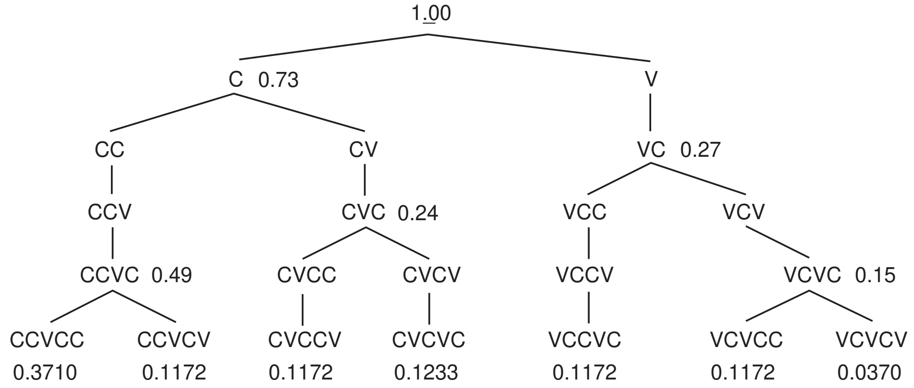
Let (U, P) be a probability space. Suppose we want to choose elements of U at random according to the probability function P. In other words, $u \in U$ will have a probability $P(u)$ of being chosen each time we choose an element. This is easy to do if we have a decision tree whose leaves correspond to the elements of U. The process is best understood by looking at an example

Example 20 (Generating random words) In Example 2 we looked at the problem of counting certain types of "words." Go back and review that example before continuing.

• • •

We want to generate those words at random.

We'll use a two step approach. First, we'll select a CV-pattern corresponding to one of the leaves in the tree from Example 2. (We've reproduced the figure below. For the present, ignore the numbers in the figure.) Second, we'll generate a word at random that fits the pattern.



Generating a random word to fit the pattern is simple. To illustrate, suppose the pattern is CCVCV. Since there are 20 choices for the first C, use the computer software to generate a random number between 1 and 21 to decide what consonant to choose for C. The second C has 19 choices since adjacent consonants must be different and so on. Here's the result of some random choices

position & type	number of choices	random number	letter chosen	comments
1 C	20	5	G	5th among consonants (BCDFG...)
2 C	19	11	P	11th among consonants except G
3 V	6	2	E	2nd among vowels (AEIOUY)
4 C	20	11	N	11th among consonants (BCDFG...)
5 V	6	3	I	3rd among vowels (AEIOUY)

How should we choose a pattern? We discovered in Example 2 that some patterns fit more words than other patterns fit. Each pattern should be chosen in proportion to the number of words it fits so that each word will have an equal chance. Using the counts in Example 2, we computed the probabilities of the leaves in the preceding figure. Thus

$$P(\text{leaf}) = \frac{\text{number of words with leaf pattern}}{\text{total number of words}}$$

You should compute those values yourself. We have constructed a probability space where U is the set of leaves in the tree and P has the values shown at the leaves. Each vertex in the tree corresponds to an event, namely the set of leaves that are below it in the tree. Thus we can compute the probability of each vertex in the tree by adding up the probabilities of the leaves below that vertex. Many of those probabilities are shown in the previous figure.

How do we generate a leaf at random using the probabilities we've computed? We start at the root of the tree and choose a path randomly as follows. If we are at a vertex v that has edges (v,w), (v,x) and (v,y), we simply choose among w, x and y by using the conditional probabilities $P(w|v) = P(w)/P(v)$, $P(x|v) = P(x)/P(v)$ and $P(y|v) = P(y)/P(v)$. In other words, choose w with probability $P(w|v)$ and so on. (This can be done using random number generators on computers.)

Someone might say:

> All that work with the tree is not necessary since we can use the following "direct" method: Using the leaf probabilities, generate a random pattern.

That approach is not always feasible. For example, suppose we wanted to generate a random strictly decreasing function from $\underline{200}$ to $\underline{100}$. We learned in Section 3 of Unit Fn that there are $\binom{200}{100}$ of these functions. This number is about 3×10^{58}. Many random number generators cannot reliably generate random integers between 1 and a number this large. Thus we need a different method. One way is to use a decision tree that lists the functions. It's a much bigger tree than we've looked at, but we don't need to construct the tree. All we need to know is how to compute the conditional probabilities so that each leaf will have probability $1/\binom{200}{100}$. It turns out that this can be done rather easily. In summary, the tree method can be used when the "direct" method is not practical. $\square$

---

*The First Moment Method and the SAT Problem

We now review briefly the concept of conjunctive normal form. Suppose $p, q, r, \ldots$ are Boolean variables (that is, variables that can be 0 or 1). The operations $\sim$, $\vee$, and $\wedge$ stand for "negation", "or", and "and", respectively. A *disjunctive clause* is a list of Boolean variables and negations of Boolean variables joined by $\vee$. Here are four examples of disjunctive clauses:

$$q \vee r, \qquad p \vee (\sim q) \vee r, \qquad (\sim p) \vee (\sim q) \vee (\sim r), \qquad (\sim r) \vee q.$$

Conjunctive normal form is a statement form consisting of disjunctive clauses joined by $\wedge$; for example

$$(q \vee r) \wedge (p \vee (\sim q) \vee (\sim r)) \wedge ((\sim p) \vee (\sim q) \vee r) \wedge ((\sim r) \vee q).$$

(*Disjunctive normal form* is the same as conjunctive normal form except that $\wedge$ and $\vee$ are switched.) The *satisfiability problem* is the following. Given a statement in conjunctive normal form, is there some choice of values for the Boolean variables that make the statement equal to 1? One may also want to know what choice of variables does this. The satisfiability problem is also called the *SAT problem.* The SAT problem is known to be hard in general. (The technical term is "NP-complete".)

Given a statement in conjunctive normal form, how might we try to solve the satisfiability problem? One way is with the following backtracking algorithm for a statement involving $p_1, p_2, \ldots, p_n$

Step 1. Set $k = 1$.

Step 2. Set $p_k = 0$.

Step 3. (Test) Check to see if any of the clauses that contain only $p_1, \ldots, p_k$ are 0. If so, go to Step 4_0; if not, go to Step 4_1.

Step 4_0. (Failure) If $p_k = 0$, set $p_k = 1$ and go to Step 3. If $p_k = 1$, replace k with $k - 1$ and go to Step 4_0.

Step 4_1. (Partial success) If $k = n$, stop because the current values of the variables make the statement 1. If $k < n$, replace k with $k+1$ and go to Step 2.

You should use the algorithm on the conjunctive normal form statement given earlier.

The following example shows that we can sometimes guarantee that the algorithm will succeed if there are not too many clauses. However, it does not give us values of the variables that will make the statement 1. In the example after that, we will see how to use the idea from the next example to find those values without backtracking.

Example 21 (SAT with just a few clauses) Suppose we have a conjunctive normal form statement $S = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, where the C_i are clauses in the Boolean variables $p_1, \ldots, p_n$. Make the set $\times^n\{0,1\}$, the possible values for $p_1, \ldots, p_n$, into a probability space by letting each n-tuple have probability $1/2^n$.

Let X_i be a random variable whose value is 1 if C_i has the value 0, and let $X_i = 0$ if C_i has the value 1. (Be careful: note the reversal — X_i is the opposite of C_i.) The number of clauses which are 0 is $X = X_1 + \cdots + X_k$. If we can show that $P(X = 0) = 0$, we will have shown that there is some choice of $p_1, \ldots, p_n$ for which all clauses are 1 and so S will be 1 as well. How can we do this? Here is one tool:

Theorem 4 (First Moment Method) *Suppose that X is an integer-valued random variable and $E(X) < m + 1$, then $P(X \le m)$ is greater than 0.*

This is easy to prove:

$$m + 1 > E(X) = \sum_k kP(X = k) \ge \sum_{k \ge m+1} (m+1)P(X = k) = (m+1)P(X \ge m+1).$$

Thus $P(X \ge m+1) < 1$ and so $P(X \le m) = 1 - P(X \ge m+1) > 0$.

To apply this, we need to compute $E(X) = E(X_1) + \cdots + E(X_k)$. Let v_i be the number of variables and their negations appearing in C_i. We claim that $E(X_i) = 2^{-v_i}$. Why is this? Note that $E(X_i)$ equals the probability that C_i has the value 0. To make C_i have the value 0, each variable in C_i must be chosen correctly: 0 if it appears without being negated and 1 if it appears negated. The variables not appearing in C_i can have any values whatsoever.

We have shown that $E(X) = 2^{-v_1} + \cdots + 2^{-v_k}$. By the First Moment Method, we are done if this is less than 1. We have proved:

Theorem 5 (SAT for few clauses) *Suppose we have a conjunctive normal form statement $S = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, where the C_i are clauses in the Boolean variables $p_1, \ldots, p_n$. Let v_i be the number of variables (and negations of variables) that appear in C_i. If $2^{-v_1} + \cdots + 2^{-v_k} < 1$, then there is a choice of values for $p_1, \ldots, p_n$ which gives S the value 1.*

Let's apply the theorem to

$$S = (q \vee r) \wedge (p \vee (\sim q) \vee (\sim r)) \wedge ((\sim p) \vee (\sim q) \vee r) \wedge ((\sim r) \vee q).$$

We have $v_1 = 2$, $v_2 = 3$, $v_3 = 3$, and $v_4 = 2$. Thus

$$E(X) = 2^{-2} + 2^{-3} + 2^{-3} + 2^{-2} = 3/4 < 1.$$

Thus there is a choice of variables that give S the value 1. If you carried out the backtracking algorithm as you were asked to earlier, you found such an assignment. Of course, you may find the assignment rather easily without backtracking. However, the theorem tells us a lot more: It doesn't look at the structure of the clauses, so you could change p to $\sim p$ and so on in any of the clauses you wish and the statement would still be satisfiable. $\square$

Example 22 (Satisfiability without backtracking) Suppose the situation in the preceding example holds; that is, $E(X) < 1$. We want to find values for $p_1, \ldots, p_n$ that satisfy S (give it the value 1). We have

$$\begin{aligned} E(X) &= P(p_n = 0)\, E(X \mid p_n = 0) \ + \ P(p_n = 1)\, E(X \mid p_n = 1) \\ &= \tfrac{1}{2} E(X \mid p_n = 0) \ + \ \tfrac{1}{2} E(X \mid p_n = 1). \end{aligned}$$

Since $E(X) < 1$ at least one of $E(X \mid p_n = 0)$ and $E(X \mid p_n = 1)$ must be less than 1. Suppose that $E(X \mid p_n = 0) < 1$. Set $p_n = 0$ and simplify S to get a new statement S' in $p_1, \ldots, p_{n-1}$. To get this new statement S' from S when $p_n = 0$:

- any clause not containing p_n or $\sim p_n$ is unchanged;
- any clause containing $\sim p_n$ will have the value 1 regardless of the remaining variables and so is dropped;
- any clause containing p_n depends on the remaining variables for its value and so is kept, with p_n removed.

When $p_n = 1$, the last two cases are reversed to produce S'. This method will be illustrated soon.

Let X' be for S' what X is for S. You should show that

$$E(X') = E(X \mid p_n = 0) < 1.$$

We can now repeat the above procedure for S', which will give us a value for p_{n-1}. Continuing in this way, we find values for $p_n, p_{n-1}, \ldots, p_1$.

Let's apply this to

$$S = (q \vee r) \wedge (p \vee (\sim q) \vee (\sim r)) \wedge ((\sim p) \vee (\sim q) \vee r) \wedge ((\sim r) \vee q).$$

When $p = 0$, this reduces to

$$(q \vee r) \wedge ((\sim q) \vee (\sim r)) \wedge ((\sim r) \vee q),$$

and so $E(X \mid p = 0) = 2^{-2} + 2^{-2} + 2^{-2} < 1$. Thus we can take the previous statement to be S'. Suppose we try $q = 0$. Then

$$(q \vee r) \wedge ((\sim q) \vee (\sim r)) \wedge ((\sim r) \vee q)$$

reduces to $r \wedge (\sim r)$ because the middle clause disappears. The expectation is $2^{-1} + 2^{-1} = 1$, so this is a bad choice. (Of course this is obviously a bad choice, but we're applying the method blindly like a computer program would.) Thus we must choose $q = 1$. The statement reduces to $\sim r$, and we choose $r = 0$. $\square$

Exercises for Section 3

3.1. A box contains 3 white and 4 green balls.

(a) Two balls are sampled *with* replacement, what is the probability that the second is white if the first is green? If the first is white?

(b) Two balls are sampled *without* replacement, what is the probability that the second is white if the first is green? If the first is white?

3.2. Two dice are rolled and the total is six.

(a) What is the probability that at least one die is three?

(b) What is the probability that at least one die is four?

(c) What is the probability that at least one die is odd?

3.3. In a certain college, 10 percent of the students are physical science majors, 40 percent are engineering majors, 20 percent are biology majors and 30 percent are humanities majors. Of the physical science majors, 10 percent have read Hamlet, of the engineering majors, 50 percent have read Hamlet, of the biology majors, 30 percent have read Hamlet, and of the humanities majors, 20 percent have read Hamlet.

(a) Given that a student selected at random has read Hamlet, what is the probability that that student is a humanities major?

(b) Given that a student selected at random has not read Hamlet, what is the probability that that student is an engineering or physical science major?

3.4. We are given an urn that has one red ball and one white ball. A fair die is thrown. If the number is a 1 or 2, one red ball is added to the urn. Otherwise three red balls are added to the urn. A ball is then drawn at random from the urn.

(a) Given that a red ball was drawn, what is the probability that a 1 or 2 appeared when the die was thrown?

(b) Given that the final composition of the urn contained more than one red ball, what is the probability that a 1 or 2 appeared when the die was thrown?

3.5. A man starts with one dollar in a pot. A "play" consists of flipping a fair coin and, if heads occurs, a dollar is added to the pot, if tails occurs, a dollar is removed from the pot. The game ends if the man has zero dollars or if he has played four times. Let X denote the random variable which, for each outcome of the game, specifies the maximum amount of money that was ever in the pot, from (and including) the start of the game to (and including) that final outcome. What is the expected value $E(X)$?

3.6. The probability of team A winning any game is $1/3$, of B winning $2/3$ (no ties in game play). Team A plays team B in a tournament. If either team wins two games in a row, that team is declared the winner. At most four games are played in the tournament and, if no team has won the tournament at the end of four games, the tournament is declared a draw. What is the expected number of games in the tournament?

3.7. The platoon commander knows:

- If the air strike is successful, there is a 60% probability that the ground forces *will not* encounter enemy fire.
- If the air strike is not successful, there is a 80% probability that the ground forces *will* encounter enemy fire.
- There is a 70% probability that the air strike will be successful.

Answer the following questions.

(a) What is the **probability** that the ground forces *will not* encounter enemy fire?

(b) The ground forces did not encounter enemy fire. What is the **probability** that the air strike was successful?

Section 4: Inductive Proofs and Recursive Equations

Proof by induction, familiar from prior courses and used occasionally in earlier sections, is central to the study of recursive equations. We'll begin by reviewing proof by induction. Then we'll look at recursions (another name for recursive equations). The two subjects are related since induction proofs use smaller cases to prove larger cases and recursions use previous values in a sequence to compute later values. A "solution" to a recursion is a formula that tells us how to compute any term in the sequence without first computing the previous terms. We will find that it is usually easy to verify a solution to a recursion if someone gives it to us; however, it can be quite difficult to find the solution on our own — in fact there may not even be a simple solution even when the recursion looks simple.

Induction

Suppose $\mathcal{A}(n)$ is an assertion that depends on n. We use *induction* to prove that $\mathcal{A}(n)$ is true when we show that

- it's true for the smallest value of n and
- if it's true for everything less than n, then it's true for n.

Closely related to proof by induction is the notion of a recursion. A *recursion* describes how to calculate a value from previously calculated values. For example, $n!$ can be calculated by using $n! = 1$ if $n = 0$, $n! = n(n-1)!$ if $n > 0$.

Notice the similarity between the two ideas: There is something to get us started and then each new thing depends on similar previous things. Because of this similarity, recursions often appear in inductively proved theorems. We'll study inductive proofs and recursive equations in this section.

Inductive proofs and recursive equations are special cases of the general concept of a recursive approach to a problem. Thinking recursively is often fairly easy when one has mastered it. Unfortunately, people are sometimes defeated before reaching this level. In Section 3 we look at some concepts related to recursive thinking and recursive algorithms.

We recall the theorem on induction and some related definitions:

Theorem 6 (Induction) *Let $\mathcal{A}(m)$ be an assertion, the nature of which is dependent on the integer m. Suppose that $n_0 \le n_1$. If we have proved the two statements*

(a) "$\mathcal{A}(n)$ is true for $n_0 \le n \le n_1$" and

(b) "If $n > n_1$ and $\mathcal{A}(k)$ is true for all k such that $n_0 \le k < n$, then $\mathcal{A}(n)$ is true."

Then $\mathcal{A}(m)$ is true for all $m \ge n_0$.

Let's look at a common special case: $n_0 = n_1$ and, in proving (b) we use only $\mathcal{A}(n-1)$. Then the theorem becomes

> *Let $\mathcal{A}(m)$ be an assertion, the nature of which is dependent on the integer m. If we have proved the two statements*
>
> *(a) "$\mathcal{A}(n_0)$ is true" and*
>
> *(b) "If $n > n_0$ and $\mathcal{A}(n-1)$ is true, then $\mathcal{A}(n)$ is true."*
>
> *Then $\mathcal{A}(m)$ is true for all $m \ge n_0$.*

Some people use terms like "weak induction", "simple induction" and "strong induction" to distinguish the various types of induction.

Definition 5 (Induction hypothesis) *The statement "$\mathcal{A}(k)$ is true for all k such that $n_0 \le k < n$" is called the induction assumption or induction hypothesis and proving that this implies $\mathcal{A}(n)$ is called the inductive step. $\mathcal{A}(n_0), \ldots, \mathcal{A}(n_1)$ are called the base cases or simplest cases.*

Proof: We now prove the theorem. Suppose that $\mathcal{A}(n)$ is false for some $n \ge n_0$. Let m be the least such n. We cannot have $m \le n_1$ because (a) says that $\mathcal{A}(n)$ is true for $n_0 \le n \le n_1$. Thus $m > n_1$.

Since m is as small as possible, $\mathcal{A}(k)$ is true for $n_0 \le k < m$. By (b), the inductive step, $\mathcal{A}(m)$ is also true. This contradicts our assumption that $\mathcal{A}(n)$ is false for some $n \ge n_0$. Hence the assumption is false; in other words, $\mathcal{A}(n)$ is never false for $n \ge n_0$. This completes the proof. $\square$

Example 23 (Every integer is a product of primes) A positive integer $n > 1$ is called a *prime* if its only divisors are 1 and n. The first few primes are 2, 3, 5, 7, 11, 13, 17, 19, 23. If a number is not a prime, such as 12, it can be written as a product of primes (*prime factorization*: $12 = 2 \times 2 \times 3$). We adopt the terminology that a single prime p is a product of one prime, itself. We shall prove $\mathcal{A}(n)$ that "every integer $n \ge 2$ is a product of primes." Our proof will be by induction. We start with $n_0 = n_1 = 2$, which is a prime and hence a product of primes. The induction hypothesis is the following:

"Suppose that for some $n > 2$, $\mathcal{A}(k)$ is true for all k such that $2 \le k < n$."

Assume the induction hypothesis and consider n. If n is a prime, then it is a product of primes (itself). Otherwise, $n = st$ where $1 < s < n$ and $1 < t < n$. By the induction hypothesis, s and t are each a product of primes, hence $n = st$ is a product of primes. $\square$

In the example just given, we needed the induction hypothesis "for all k such that $2 \le k < n$." In the next example we have the more common situation where we only need to assume "for $k = n - 1$." We can still make the stronger assumption and the proof is valid, but the stronger assumption is not used; in fact, we are using the simpler form of induction described after the theorem.

Example 24 (Sum of first n integers) We would like a formula for the sum of the first n integers. Let us write $S(n) = 1 + 2 + \ldots + n$ for the value of the sum. By a little calculation,

$$S(1) = 1, \quad S(2) = 3, \quad S(3) = 6, \quad S(4) = 10, \quad S(5) = 15, \quad S(6) = 21.$$

What is the general pattern? It turns out that $S(n) = \frac{n(n+1)}{2}$ is correct for $1 \le n \le 6$. Is it true in general? This is a perfect candidate for an induction proof with

$$n_0 = n_1 = 1 \qquad \text{and} \qquad \mathcal{A}(n): \quad \text{“}S(n) = \tfrac{n(n+1)}{2}.\text{”}$$

Let's prove it. We have shown that A(1) is true. In this case we need only the restricted induction hypothesis; that is, we will prove the formula for $S(n)$ by assuming the formula for $k = n - 1$. Thus, we assume only $\mathcal{A}(n-1)$ is true. Here it is (the inductive step):

$$\begin{aligned}
S(n) &= 1 + 2 + \cdots + n && \text{by the definition of } S(n)\\
&= \Big(1 + 2 + \cdots + (n-1)\Big) + n &&\\
&= S(n-1) + n && \text{by definition of } S(n-1),\\
&= \frac{(n-1)\big((n-1)+1\big)}{2} + n && \text{by } \mathcal{A}(n-1),\\
&= \frac{n(n+1)}{2} && \text{by algebra.}
\end{aligned}$$

This completes the proof. We call your attention to the fact that, in the third line we proved $S(n) = S(n-1) + n$. $\square$

Recursive Equations

The equation $S(n) = S(n-1) + n$ (for $n > 1$) that arose in the inductive proof in the preceding example is called a *recurrence relation*, *recursion*, or *recursive equation*. A recursion is *not complete* unless there is information on how to get started. In this case the information was $S(1) = 1$. This information is called the *initial condition* or, if there is more than one, *initial conditions*. Many examples of such recurrence relations occur in computer science and mathematics. We discussed recurrence relations in Section 3 of Unit CL (Basic Counting and Listing) for binomial coefficients $C(n, k)$ and Stirling numbers $S(n, k)$.

In the preceding example, we found that $S(n) = n(n + 1)/2$. This is a *solution* to the recursion because it tells us how to compute $S(n)$ without having to compute $S(k)$ for any other values of k. If we had used the recursion $S(n) = S(n-1) + n$, we would have had to compute $S(n-1)$, which requires $S(n-2)$, and so on all the way back to $S(1)$.

A recursion tells us how to compute values in a sequence a_n from earlier values $a_{n-1}, a_{n-2}, \ldots$ and n. We can denote this symbolically by writing $a_n = G(n, a_{n-1}, a_{n-2}, \ldots)$. For example, in the case of the sum of the first n integers, which we called $S(n)$, we would have

$$a_n = S(n) \quad \text{and} \quad G = a_{n-1} + n \quad \text{since} \quad S(n) = S(n-1) + n.$$

Induction proofs deduce the truth of $\mathcal{A}(n)$ from earlier statements. Thus it's natural to use induction to prove that a formula for the solution to a recursion is correct. That's what we did in the previous example. There's a way to avoid giving an inductive proof each time we have such a problem: It turns out that the induction proofs for solutions to recursions all have the same form. A general pattern often means there's a general theorem. If we can find and prove the theorem, then we could use it to avoid giving an inductive proof in each special case. That's what the following theorem is about. (The a_n and $f(n)$ of the theorem are generalizations of S_n and $\frac{n(n+1)}{2}$ from the previous example.)

Theorem 7 (Verifying the solution of a recursion) *Suppose we have initial conditions that give a_n for $n_0 \le n \le n_1$ and a recursion that allows us to compute a_n when $n > n_1$. To verify that $a_n = f(n)$, it suffices to do two things:*

Step 1. Verify that f satisfies the initial conditions.

Step 2. Verify that f satisfies the recursion.

Proof: The goal of this theorem is to take care of the inductive part of proving that a formula is the solution to a recursion. Thus we will have to prove it by induction. We must verify (a) and (b) in Theorem 6. Let $\mathcal{A}(n)$ be the assertion "$a_n = f(n)$." By Step 1, $\mathcal{A}(n)$ is true for $n_0 \le n \le n_1$, which proves (a). Suppose the recursion is $a_n = G(n, a_{n-1}, a_{n-2}, \ldots)$ for some formula G. We have

$$\begin{aligned} f(n) &= G\Big(n, f(n-1), f(n-2), \ldots\Big) && \text{by Step 2,} \\ &= G(n, a_{n-1}, a_{n-2}, \ldots) && \text{by } \mathcal{A}(k) \text{ for } k < n, \\ &= a_n && \text{by the recursion for } a_n. \end{aligned}$$

This proves (b) and so completes the proof. □

Example 25 (Proving a formula for the solution of a recursion) Let $S(n)$ be the sum of the first n integers. The initial condition $S(1) = 1$ and the recursion $S(n) = n + S(n-1)$ allow us to compute $S(n)$ for all $n \ge 1$. It is claimed that $f(n) = \frac{n(n+1)}{2}$ equals $S(n)$.

The initial condition is for $n = 1$. Thus $n_0 = n_1 = 1$. Since $f(1) = 1$, f satisfies the initial condition. (This is Step 1.) For $n > 1$ we have

$$n + f(n-1) = n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2} = f(n)$$

and so f satisfies the recursion. (This is Step 2.)

We now consider a different problem. Suppose we are given that

$$a_0 = 2, \quad a_1 = 7, \quad \text{and} \quad a_n = 3a_{n-1} - 2a_{n-2} \text{ when } n > 1$$

and we are asked to prove that $a_n = 5 \times 2^n - 3$ for $n \ge 0$.

Let's verify that the formula is correct for $n = 0$ and $n = 1$ (the initial conditions — Step 1 in our theorem):

$$n = 0: \quad a_0 \ = \ 2 \ = \ 5^0 - 3 \qquad n = 1: \quad a_1 \ = \ 7 \ = \ 5^1 - 3.$$

Now for Step 2, the recursion. Let $f(x) = 5 \times 2^x - 3$ and assume that $n > 1$. We have

$$\begin{aligned} 3f(n-1) - 2f(n-2) &= 3(5 \times 2^{n-1} - 3) - 2(5 \times 2^{n-2} - 3) \\ &= (3 \times 5 \times 2 - 2 \times 5)2^{n-2} - 3 \\ &= 5 \times 2^n - 3 = f(n). \end{aligned}$$

This completes the proof.

As a final example, suppose $b_0 = b_1 = 1$ and $b_{n+1} = n(b_n + b_{n-1})$ for $n \geq 1$. We want to prove that $b_n = n!$. Since our theorem stated the recursion for a_n, let's rewrite our recursion to avoid confusion. Let $n+1 = k$ in the recursion to get $b_k = (k-1)(b_{k-1}+b_{k-2})$. The initial conditions are $b_0 = 1 = 0!$ and $b_1 = 1 = 1!$, so we've done Step 1. Now for Step 2:

$$\text{Is} \quad k! = (k-1)((k-1)! + (k-2)!) \quad \text{true?}$$

Yes because $(k-1)! = (k-1) \times (k-2)!$ and so $(k-1)!+(k-2)! = ((k-1)+1)(k-2)! = k \times (k-2)!$. We could have done this without changing the subscripts in the recursion: Just check that $(n+1)! = n(n! + (n-1)!)$. We'll let you do that. $\square$

So far we have a method for checking the solution to a recursion, which we just used in the previous example. How can we find a solution in the first place? If we're lucky, someone will tell us. If we're unlucky, we need a clever guess or some tools. Let's look at how we might guess.

Example 26 (Guessing solutions to recurrence relations)

(1) Let $r_k = -r_{k-1}/k$ for $k \geq 1$, with $r_0 = 1$. Writing out the first few terms gives $1, -1, 1/2, -1/6, 1/24, \ldots$. Guessing, it looks like $r_k = (-1)^k/k!$ is a solution.

(2) Let $t_k = 2t_{k-1} + 1$ for $k > 0$, $t_0 = 0$. Writing out some terms gives $0, 1, 3, 7, 15, \ldots$. It looks like $t_k = 2^k - 1$, for $k \geq 0$.

(3) What is the solution to $a_0 = 0$, $a_1 = 1$ and $a_n = 4a_{n-1} - 4a_{n-2}$ for $n \geq 2$? Let's compute some values

$$\begin{array}{rcccccccc} n: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ a_n: & 0 & 1 & 4 & 12 & 32 & 80 & 192 & 448 \end{array}$$

These numbers factor nicely: $4 = 2^2$, $12 = 2^2 \times 3$, $32 = 2^5$, $80 = 2^4 \times 5$, $192 = 2^6 \times 3$, $448 = 2^6 \times 7$. Can we see a pattern here? We can pull out a factor of 2^{n-1} from a_n:

$$\begin{array}{rcccccccc} n: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ a_n: & 0 & 1 & 4 & 12 & 32 & 80 & 192 & 448 \\ a_n/2^{n-1}: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

Now the pattern is clear: $a_n = n2^{n-1}$. That was a lot of work, but we're not done yet — this is just a guess. We have to prove it. You can use the theorem to do that. We'll do it a different way in a little while.

(4) Let $b_n = b_1 b_{n-1} + b_2 b_{n-2} + \cdots + b_{n-1} b_1$ for $n \geq 2$, with $b_1 = 1$. Here are the first few terms: $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, \ldots$. Each term is around 3 or 4 times the preceding one. Let's compute the ratio exactly

$$\begin{array}{cccccccccc} n & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ b_n/b_{n-1} & 1 & 2 & 5/2 & 14/5 & 3 & 22/7 & 13/4 & 10/3 & 17/5 \end{array}$$

These ratios have surprisingly small numerators and denominators. Can we find a pattern? The large primes 13 and 17 in the numerators for $n = 8$ and 10 suggest that maybe we should look for $2n-3$ in the numerator.[1] Let's adjust our ratios accordingly:

n	2	3	4	5	6	7	8	9	10
$b_n/(2n-3)b_{n-1}$	1	2/3	1/2	2/5	1/3	2/7	1/4	2/9	1/5

Aha! These numbers are just $2/n$. Our table leads us to guess $b_n = 2(2n-3)b_{n-1}/n$, a much simpler recursion than the one we started with.

This recursion is so simple we can "unroll" it:

$$b_n = \frac{2(2n-3)}{n}\,b_{n-1} = \frac{2(2n-3)}{n}\,\frac{2(2n-5)}{n-1}\,b_{n-2} = \cdots = \frac{2^{n-1}(2n-3)(2n-5)\cdots 1}{n!}.$$

This is a fairly simple formula. Of course, it is still only a conjecture and it is not easy to prove that it is the solution to the original recursion because the computations in Theorem 7 using this formula and the recursion $b_n = b_1 b_{n-1} + b_2 b_{n-2} + \cdots + b_{n-1} b_1$ would be very messy.

(5) Let $d_n = (n-1)d_{n-1} + (n-1)d_{n-2}$ for $n \geq 2$, with $d_0 = 1$ and $d_1 = 0$. In the previous example, we looked at a recursion that was almost like this: The only difference was that $d_1 = 1$. In that case we were told that the answer was $n!$, so maybe these numbers look like $n!$. If this were like $n!$, we'd expect nd_{n-1} to equal d_n. Here are the first few values of d_n together with nd_{n-1}:

n	0	1	2	3	4	5	6
d_n	1	0	1	2	9	44	265
nd_{n-1}	–	1	0	3	8	45	264

We're close! The values of d_n and nd_{n-1} only differ by 1. Thus we are led to guess that $d_n = nd_{n-1} + (-1)^n$. This is not a solution—it's another recursion. Nevertheless, we might prefer it because it's a bit simpler than the one we started with. $\square$

As you can see from the previous example, guessing solutions to recursions can be difficult. Now we'll look at a couple of theorems that tell us the solutions without any guessing.

Theorem 8 (Solutions to Some Recursions) *Let $a_0, a_1, \ldots, a_n, \ldots$ be a sequence of numbers. Suppose there are constants b and c such that b is not 0 or 1 and $a_n = ba_{n-1} + c$ for $n \geq 1$. Then*

$$a_n = Ab^n + K \quad \text{where} \quad K = \frac{c}{1-b} \quad \text{and} \quad A = a_0 - K = a_0 - \frac{c}{1-b}.$$

[1] "Why look at large primes?" you ask. Because they are less likely to have come from a larger number that has lost a factor due to reduction of the fraction.

This gives us the solution to the recursion $t_k = 2t_{k-1}+1$ (with $t_0 = 0$) of the previous example: $K = \frac{1}{1-2} = -1$ and $A = 0-(-1) = 1$. That gives the solution $t_k = 2^k - 1$, no guessing needed!

Proof: (of Theorem 8) We'll use Theorem 7. The initial condition is simple:

$$Ab^0 + K \;=\; A + K \;=\; (a_0 - K) + K \;=\; a_0.$$

That's Step 1: For Step 2 we want to show that $a_n \;=\; Ab^n + K$ satisfies the recursion. We have

$$\begin{aligned} b(Ab^{n-1}+K)+c \;&=\; Ab^n + bK + c \;=\; Ab^n + \frac{bc}{1-b} + c \\ &=\; Ab^n + \frac{bc+(1-b)c}{1-b} \;=\; Ab^n + \frac{c}{1-b} \;=\; Ab^n + K. \end{aligned}$$

We're done. $\square$

Example 27 (I forgot the formulas for A and K!) If you remember the b^n, you can still solve the recursion even if the initial condition is not at a_0. Let's do the example

$$a_1 = 3 \qquad \text{and} \qquad a_n \;=\; 4a_{n-1} - 7 \quad \text{for} \quad n > 1.$$

The solution will be $a_n = A4^n + K$ for some A and K. If we know a_n for two values of n, then we can solve for A and K. We're given $a_1 = 3$ and we compute $a_2 \;=\; 4\times 3 - 7 \;=\; 5$. Thus

$$a_1 \text{ gives us } \quad 3 \;=\; A4^1 + K \qquad \text{and} \qquad a_2 \text{ gives us } \quad 5 \;=\; A4^2 + K.$$

Subtracting the first equation from the second: $12A = 2$ so $A = 1/6$. From a_1, $3 = 4/6 + K$ and so $K = 7/3$. $\square$

Now let's look at recursions where a_n depends on a_{n-1} and a_{n-2} in a simple way.

Theorem 9 (Solutions to Some Recursions) *Let $a_0, a_1, \ldots, a_n, \ldots$ be a sequence of numbers. Suppose there are constants b and c such that $a_n = ba_{n-1} + ca_{n-2}$ for $n \ge 2$. Let r_1 and r_2 be the roots of the polynomial $x^2 - bx - c$.*

- *If $r_1 \ne r_2$, then $a_n = K_1r_1^n + K_2r_2^n$ for $n \ge 0$, where K_1 and K_2 are solutions to the equations*

$$K_1 + K_2 = a_0 \qquad \text{and} \qquad r_1K_1 + r_2K_2 = a_1.$$

- *If $r_1 = r_2$, then $a_n = K_1r_1^n + K_2nr_1^n$ for $n \ge 0$, where K_1 and K_2 are solutions to the equations*

$$K_1 = a_0 \qquad \text{and} \qquad r_1K_1 + r_2K_2 = r_1K_1 + r_1K_2 = a_1.$$

The equation $x^2 - bx - c = 0$ is called the characteristic equation of the recursion.

Before proving the theorem, we give some examples. In all cases, the roots of x^2-bx-c can be found either by factoring it or by using the quadratic formula

$$r_1,\, r_2 = \frac{b \pm \sqrt{b^2+4c}}{2}.$$

Example 28 (Applying Theorem 9) Let's redo the recursion

$$a_0 = 2, \quad a_1 = 7, \quad \text{and} \quad a_n = 3a_{n-1} - 2a_{n-2} \text{ when } n > 1$$

from Example 25. We have $b = 3$ and $c = -2$. The characteristic equation is $x^2 - 3x + 2 = 0$. The roots of $x^2 - 3x + 2$ are $r_1 = 2$ and $r_2 = 1$, which you can get by using the quadratic formula or by factoring $x^2 - 3x + 2$ into $(x - 2)(x - 1)$. Since $r_1 \neq r_2$, we are in the first case in the theorem. Thus we have to solve

$$K_1 + K_2 = 2 \qquad \text{and} \qquad 2K_1 + K_2 = 7.$$

The solution is $K_1 = 5$ and $K_2 = -3$ and so $a_n = 5 \times 2^n - 3 \times 1^n = 5 \times 2^n - 3$.

• As another example, we'll solve the recursion $a_0 = 0$, $a_1 = 1$, and $a_n = 4a_{n-1} - 4a_{n-2}$ for $n \geq 2$. Applying the theorem, $r_1 = r_2 = 2$ and so $a_n = K_1 2^n + K_2 n 2^n$ where $K_1 = 0$ and $2K_1 + 2K_2 = 1$. Thus $K_1 = 0$, $K_2 = 1/2$, and $a_n = (1/2)n2^n = n2^{n-1}$.

• As a final example, consider the recursion

$$F_0 = F_1 = 1 \quad \text{and} \quad F_k = F_{k-1} + F_{k-2} \quad \text{when } k \geq 2.$$

This is called the *Fibonacci recursion*. We want to find an explicit formula for F_k.

The characteristic equation is $x^2 - x - 1 = 0$. By the quadratic formula, its roots are $r_1 = \frac{1+\sqrt{5}}{2}$ and $r_2 = \frac{1-\sqrt{5}}{2}$. Thus, we need to solve the equations

$$K_1 + K_2 \;=\; 1 \qquad \text{and} \qquad r_1 K_1 + r_2 K_2 \;=\; 1.$$

High school math gives

$$\begin{aligned} K_1 &= \frac{1 - r_2}{r_1 - r_2} = \frac{1 + \sqrt{5}}{2\sqrt{5}} = \frac{r_1}{\sqrt{5}} \\ K_2 &= \frac{1 - r_1}{r_2 - r_1} = \frac{1 - \sqrt{5}}{-2\sqrt{5}} = -\frac{r_2}{\sqrt{5}}. \end{aligned}$$

Thus

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

It would be difficult to guess this solution from a few values of F_n! $\square$

Example 29 (A shifted index) Let's solve the recursion

$$a_1 = 0, \qquad a_2 = 1 \qquad \text{and} \qquad a_n \;=\; 5a_{n-1} + 6a_{n-2} \quad \text{for} \quad n \geq 3.$$

This doesn't quite fit the theorem since it starts with a_1 instead of a_0. What can we do? The same thing we did in Example 27: Use values of a_n to get two equations in the two unknowns K_1 and K_2.

Let's do this. The characteristic equation $x^2 - 5x - 6 \;=\; 0$ gives us $r_1 = 6$ and $r_2 = -1$ and so $a_n \;=\; K_1 6^n + K_2(-1)^n$. Using a_1 and a_2:

$$a_1 \text{ gives us } \; 0 \;=\; 6K_1 - K_2 \qquad \text{and} \qquad a_2 \text{ gives us } \; 1 \;=\; 6^2K_1 + K_2.$$

Adding the two equations: $1 = 42K_1$. Thus $K_1 = 1/42$. a_1 gives us $0 \;=\; 6/42 - K_2$ and so $K_2 = 1/7$. Thus $a_n \;=\; (1/42)6^n + (1/7)(-1)^n$. □

We conclude this section with a proof of Theorem 9.

Proof: (of Theorem 9) We apply Theorem 7 with $n_0 = 0$ and $n_1 = 1$.

We first assume that $r_1 \neq r_2$ and we set $f(n) = K_1r_1^n + K_2r_2^n$ where K_1 and K_2 are as given by the theorem. Step 1 is simple because the equations for K_1 and K_2 are simply the equations $f(0) = a_0$ and $f(1) = a_1$. Here's Step 2

$$\begin{aligned}
bf(n-1) + cf(n-2) &= b(K_1r_1^{n-1} + K_2r_2^{n-1}) + c(K_1r_1^{n-2} + K_2r_2^{n-2})\\
&= K_1r_1^{n-2}(br_1 + c) + K_2r_2^{n-2}(br_2 + c)\\
&= K_1r_1^{n-2}r_1^2 + K_2r_2^{n-2}r_2^2\\
&= f(n).
\end{aligned}$$

Wait! Something must be wrong — the theorem says $r_1 \neq r_2$ and we never use that fact! What happened? We *assumed* that the equations could be solved for K_1 and K_2. How do we know that they have a solution? One way is to actually solve them using high school algebra. We find that

$$K_1 = \frac{a_0r_2 - a_1}{r_2 - r_1} \quad \text{and} \quad K_2 = \frac{a_0r_1 - a_1}{r_1 - r_2}\,.$$

Now we can see where $r_1 \neq r_2$ is needed: The denominators in these formulas must be nonzero.

We now consider the case $r_1 = r_2$. This is similar to $r_1 \neq r_2$. We sketch the ideas and leave it to you to fill in the details of the proof. Here it's clear that we can solve the equations for K_1 and K_2. Step 1 in Theorem 7 is checked as it was for the $r_1 \neq r_2$ case. Step 2 requires algebra similar to that needed for $r_1 \neq r_2$. The only difference is that we end up needing to show that

$$K_2r_2^{n-2}((n-1)br_2 + (n-2)c) = K_2nr_2^n.$$

You should be able to see that this is the same as showing $-br_2 - 2c = 0$. This follows from the fact that the only way we can have $r_1 = r_2$ is to have $\sqrt{b^2 + 4c} = 0$. In this case $r_2 = b/2$. □

Exercises for Section 4

4.1. Compute a_0, a_1, a_3 and a_4 for the following recursions. (Recall that $\lfloor x \rfloor$ is the greatest integer not exceeding x. For example $\lfloor 5.4 \rfloor = 5$ and $\lfloor -5.4 \rfloor = -6$.)

(a) $a_0 = 1$, $a_n = 3a_{n-1} - 2$ for $n \geq 1$.

(b) $a_0 = 0$, $a_n = \lfloor n/2 \rfloor + a_{n-1}$ for $n > 0$.

(c) $a_0 = 1$, $a_n = n + a_{\lfloor n/2 \rfloor}$ for $n > 0$.

(d) $a_0 = 0$, $a_1 = 1$, $a_n = 1 + \min(a_1 a_{n-1}, \ldots, a_k a_{n-k}, \ldots, a_{n-1} a_1)$ for $n > 1$.

4.2. We computed the first few values of some sequences that were defined by recursions. A table of values is given below. Guess simple formulas for each sequence.

n :	0	1	2	3	4	5	...
a_n :	0	0	1	1	2	2	...
b_n :	1	−1	2	−2	3	−3	...
c_n :	1	2	5	10	17	26	...
d_n :	1	1	2	6	24	120	...

4.3. What are the characteristic equations for the recursions $a_n = 6a_{n-1} - 5a_{n-2}$, $a_n = a_{n-1} + 2a_{n-2}$ and $a_n = 5(a_{n-1} + a_{n-2})$?
What are the roots of these equations?

4.4. Solve the recursion $a_0 = 0$, $a_1 = 3$ and $a_n = 6a_{n-1} - 9a_{n-2}$ for $n > 2$.

4.5. Solve the recursion $a_2 = 1$, $a_3 = 3$ and $a_n = 3a_{n-1} + 2a_{n-2}$ for $n > 3$.

4.6. Solve the recursion $a_k = 2a_{k-1} - a_{k-2}$, $k \geq 2$, $a_0 = 2$, $a_1 = 1$.

4.7. Suppose $A \neq 1$. Let $G(n) = 1 + A + A^2 + \ldots + A^{n-1}$ for $n \geq 1$.

(a) Using induction, prove that $G(n) = (1 - A^n)/(1 - A)$ for $n \geq 1$. (This is the formula for the sum of a *geometric series*.)

(b) Obtain a simple recursion for $G(n)$ from $G(n) = 1 + A + A^2 + \ldots + A^{n-1}$, including initial conditions.

(c) Use the recursion in (b) and Theorem 7 to prove that $G(n) = (1 - A^n)/(1 - A)$ for $n \geq 1$.

(d) By setting $A = y/x$ and doing some algebra, prove that

$$\frac{x^{k+1} - y^{k+1}}{x - y} = x^k y^0 + x^{k-1} y^1 + \cdots + x^0 y^{k-1} \quad \text{when } x \neq y.$$

4.8. In each of the following, find an explicit formula for a_k that satisfies the given recursion. Prove your formula.

(a) $a_k = a_{k-1}/(1 + a_{k-1})$ for $k \geq 1$, $a_0 = A > 0$.

(b) $a_k = Aa_{k-1} + B$, $k \geq 1$, $a_0 = C$.

4.9. Consider $a_k = a_{k-1} + Bk(k-1)$, $k \geq 1$, $a_0 = A$. Prove that $a_k = A + Bk(k^2-1)/3$, $k \geq 0$, is the solution to this recursion.

4.10. Consider $a_k = A2^k - a_{k-1}$, $k \geq 1$, $a_0 = C$.

(a) Prove that

$$a_k = A(2^k(-1)^0 + 2^{k-1}(-1)^1 + \cdots + 2^1(-1)^{k-1}) + (-1)^k C, \quad k \geq 1,$$

is the solution to this recursion.

(b) Write the formula for a_k more compactly using Exercise 4.7.

4.11. A gambler has $t \geq 0$ dollars to start with. He bets one dollar each time a fair coin is tossed. If he wins Q, $Q \geq t$, dollars, he quits, a happy man. If he loses all his money he quits, a sad man. What is the probability q_t that he wins Q dollars instead of losing all his money and quits a happy man? What is the probability p_t that he loses all his money and quits a sad man (i.e., ruined)? This problem is called the *Gambler's Ruin* problem.

Multiple Choice Questions for Review

1. In each case, two permutations on $\underline{6}$ are listed. In which case is the first permutation less than the second in direct insertion order?

(a) $2,3,1,4,5,6$ $\quad$ $1,3,2,4,5,6$

(b) $2,3,1,4,5,6$ $\quad$ $2,1,3,4,5,6$

(c) $2,3,1,4,5,6$ $\quad$ $4,5,6,1,3,2$

(d) $6,1,2,3,4,5$ $\quad$ $2,1,3,4,5,6$

(e) $6,2,3,1,4,5$ $\quad$ $2,3,1,4,5,6$

2. What is the rank, in direct insertion order, of the permutation $5,4,6,3,2,1$?

(a) 3 (b) 4 (c) 715 (d) 716 (e) 717

3. What is the rank, in lex order, of the permutation $6,1,2,3,4,5$?

(a) 20 (b) 30 (c) 480 (d) 600 (e) 619

4. Consider the list of all sequences of length six of A's and B's that satisfy the following conditions:

(i) There are no two adjacent A's.

(ii) There are never three B's adjacent.

What is the next sequence after ABBABB in lex order?

(a) ABABAB

(b) ABBABA

(c) BABABA

(d) BABBAB

(e) BBABBA

5. Which of the following 4×4 domino covers represent two distinct hibachi grills?

(a) hhhhhvvh and hvvhhhhh

(b) hvvhvvhh and vvhhvvhh

(c) vhvvvhvh and hhvhvhvv

(d) vvhhvvhh and hhhhvvvv

(e) vvvvvvvv and hhhhhhhh

6. Given that $a_0 = 1$, $a_n = n + (-1)^n a_{n-1}$ for $n \geq 2$ What is the value of a_4?

(a) 1 (b) 4 (c) 5 (d) 8 (e) 11

7. Given that $a_k = a_{k-1}/(1 + a_{k-1})$ for $k \geq 1$, $a_0 = 1$. Which of the following gives an explicit formula for a_k?

(a) $1/3^k$, $k = 0, 1, 2, 3, \ldots$

(b) $1/2^k$, $k = 0, 1, 2, 3, \ldots$

(c) $1/(3^{k+1} - 2)$, $k = 0, 1, 2, 3, \ldots$

(d) $1/(k + 1)$, $k = 0, 1, 2, 3, \ldots$

(e) $2/(k + 2)$, $k = 0, 1, 2, 3, \ldots$

8. Consider the recurrence relation $a_k = -8a_{k-1} - 15a_{k-2}$ with initial conditions $a_0 = 0$ and $a_1 = 2$. Which of the following is an explicit solution to this recurrence relation?

(a) $a_k = (-3)^k - (-5)^k$

(b) $a_k = k(-3)^k - k(-5)^k$

(c) $a_k = k(-3)^k - (-5)^k$

(d) $a_k = (-5)^k - (-3)^k$

(e) $a_k = k(-5)^k - k(-3)^k$

9. Consider the recurrence relation $a_k = 6a_{k-1} - 9a_{k-2}$ with initial conditions $a_0 = 0$ and $a_1 = 2$. Which of the following is an explicit solution to this recurrence relation, provided the constants A and B are chosen correctly?

(a) $a_n = A3^n + B3^n$

(b) $a_n = A3^n + B(-3)^n$

(c) $a_n = A3^n + nB3^n$

(d) $a_n = A(-3)^n + nB(-3)^n$

(e) $a_n = nA3^n + nB3^n$

10. In the Towers of Hanoi puzzle H(8, S, E, G), the configuration is

Pole S: 6, 5; Pole E: 1; Pole G: 8,7,4,3,2.

What move was just made to create this configuration?

(a) washer 1 from S to E

(b) washer 1 from G to E

(c) washer 2 from S to G

(d) washer 2 from E to G

(e) washer 5 from G to S

11. In the Towers of Hanoi puzzle H(8, S, E, G), the configuration is

Pole S: 6, 5; Pole E: empty; Pole G: 8, 7, 4 ,3 ,2, 1 .

What are the next two moves?

(a) washer 1 from G to E followed by washer 2 from G to S

(b) washer 1 from G to S followed by washer 2 from G to E

(c) washer 5 from S to E followed by washer 1 from G to E

(d) washer 5 from S to E followed by washer 1 from G to S

(e) washer 5 from S to E followed by washer 2 from G to S

12. In the Towers of Hanoi puzzle H(8, S, E, G), the configuration is

Pole S: 6, 5, 2; Pole E: 1; Pole G: 8, 7, 4 ,3.

The next move is washer 2 from S to G. What is the RANK of this move in the list of all moves for H(8, S, E, G)?

(a) 205 (b) 206 (c) 214 (d) 215 (e) 216

13. In the subset Gray code for $n = 6$, what is the next element after 111000?

(a) 000111

(b) 101000

(c) 111001

(d) 111100

(e) 101100

14. In the subset Gray code for $n = 6$, what is the element just before 110000?

(a) 010000

(b) 100000

(c) 110001

(d) 110100

(e) 111000

15. In the subset Gray code for $n = 6$, what is the RANK of 110000?

(a) 8 (b) 16 (c) 32 (d) 48 (e) 63

16. In the subset Gray code for $n = 6$, what is the element of RANK 52?

(a) 101011

(b) 101110

(c) 101101

(d) 110000

(e) 111000

17. The probability of team A winning any game is $1/3$. Team A plays team B in a tournament. If either team wins two games in a row, that team is declared the winner. At most three games are played in the tournament and, if no team has won the tournament at the end of three games, the tournament is declared a draw. What is the expected number of games in the tournament?

(a) 3 (b) $19/9$ (c) $22/9$ (d) $25/9$ (e) $61/27$

18. The probability of team A winning any game is $1/2$. Team A plays team B in a tournament. If either team wins two games in a row, that team is declared the winner. At most four games are played in the tournament and, if no team has won the tournament at the end of four games, the tournament is declared a draw. What is the expected number of games in the tournament?

(a) 4 (b) 11/4 (c) 13/4 (d) 19/4 (e) 21/8

19. A man starts with one dollar in a pot. A "play" consists of flipping a fair coin and,

- if heads occurs, doubling the amount in the pot,
- if tails occurs, losing one dollar from the pot. The game ends if the man has zero dollars or if he has played three times. Let Y denote the random variable which, for each outcome of the game, specifies the amount of money in the pot. What is the value of $\mathrm{Var}(Y)$?

(a) 9/8 (b) 10/8 (c) 12/8 (d) 14/8 (e) 447/64

20. We are given an urn that has one red ball and one white ball. A fair die is thrown. If the number is a 1 or 2, one red ball is added to the urn. Otherwise two red balls are added to the urn. A ball is then drawn at random from the urn. Given that a red ball was drawn, what is the probability that a 1 or 2 appeared when the die was thrown?

(a) 4/13 (b) 5/13 (c) 6/13 (d) 7/13 (e) 8/13

21. In a certain college,

- 10 percent of the students are science majors.
- 10 percent are engineering majors.
- 80 percent are humanities majors.
- Of the science majors, 20 percent have read Newsweek.
- Of the engineering majors, 10 percent have read Newsweek.
- Of the humanities majors, 20 percent have read Newsweek.

Given that a student selected at random has read Newsweek, what is the probability that that student is an engineering major?

(a) 1/19 (b) 2/19 (c) 5/19 (d) 9/19 (e) 10/19

22. The probability of team A winning any game is 1/3. Team A plays team B in a tournament. If either team wins two games in a row, that team is declared the winner. At most *four* games are played and, if no team has won the tournament at the end of four games, a draw is declared. Given that the tournament lasts more than two games, what is the probability that A is the winner?

(a) 1/9 (b) 2/9 (c) 4/9 (d) 5/9 (e) 6/9

23. Ten percent of the students are science majors (S), 20 percent are engineering majors (E), and 70 percent are humanities majors (H). Of S,10 percent have read 2 or more articles in Newsweek, 20 percent 1 article, 70 percent 0 articles. For E, the corresponding percents are 5, 15, 80. For H they are 20, 30, 50. Given that a student has read 0 articles in Newsweek, what is the probability that the student is S or E (i.e., not H)?

(a) 21/58 (b) 23/58 (c) 12/29 (d) 13/29 (e) 1/2

Decision Trees and Recursion

Answers: **1** (d), **2** (e), **3** (d), **4** (c), **5** (b), **6** (c), **7** (d), **8** (a), **9** (c), **10** (c), **11** (d), **12** (a), **13** (b), **14** (a), **15** (c), **16** (b), **17** (c), **18** (b), **19** (e), **20** (a), **21** (a), **22** (b), **23** (b).