Lecture outline: setting-up recurrences

In this short set of notes, we explore how, given a counting question, we can turn it into a recurrence relation.

As mentioned in the previous notes, there is no technique, although there are a few principles that apply in a wide range of situations, especially for strings and trees.

In these notes, we will mostly see examples, as practice is the only real way to learn to set-up recurrences. In particular we will see some examples that are not related to counting combinatorial objects but algorithms analysis.

Introductory example

We get back to our initial example: how many strings over the alphabet $\{A, C, G, T\}$ of size n are-there in which an A is always followed by a C.

We did answer this question with a recurrence

$$\begin{cases} c_0 &= 1 \\ c_1 &= 3 \\ c_n &= 3c_{n-1} + 1c_{n-2} \quad n \geq 2 \end{cases}$$

which was based on the following recursive **decomposition** of the considered family of strings: For $n \geq 2$, a string of $\mathcal{C}_n$ is

- either a prefix C, G, T (three choices) followed by a string of $\mathcal{C}_{n-1}$
- or the prefix AC (one choice) followed by a string of $\mathcal{C}_{n-2}$

This simple example contains all the elements of what is needed to set-up a recurrence. We were able to decompose $\mathcal{C}_n$ into the **union** of disjoint sets of strings, themselves defined in terms of the **cartesian product** of sets of strings of smaller sizes, and this decomposition is both **complete** and **unambiguous**.

Two combinatorial operators: union and cartesian product

The kinds of decomposition we will work with involve two ways to combine sets of combinatorial objects to create larger sets: union and cartesian product.

Union. If a set of combinatorial objects of size n $\mathcal{A}_n$ is the union of two disjoint subsets (of objects of size n obviously) $\mathcal{B}_n$ and $\mathcal{C}_n$, then

$$a_n = b_n + c_n$$

The statement above is trivial, and is nothing else than a repeat, in the context of counting, of what is called the Rule of Sum in your textbook.

Remark. The key point here is that $\mathcal{B}_n$ and $\mathcal{C}_n$ actually partition $\mathcal{A}_n$: every element of $\mathcal{A}_n$ is member of exactly one of these two subsets. If an element of $\mathcal{A}_n$ was neither in $\mathcal{B}_n$ nor $\mathcal{C}_n$, or in both, the combinatorial decomposition would not be complete and unambiguous, and could **not** be used to set-up a recurrence.

Cartesian Product. Let $\mathcal{A}_k$ and $\mathcal{B}_\ell$ be two families of combinatorial objects. The cartesian product $\mathcal{A}_k \times \mathcal{B}_\ell$ is the set

$$\{(a, b),\ a \in \mathcal{A}_k,\ b \in \mathcal{B}_\ell\}$$

In this case,

$$|\mathcal{A}_k \times \mathcal{B}_\ell| = |\mathcal{A}_k||\mathcal{B}_\ell|$$

This was used in our example as follows:

- in our decomposition we have two terms that are summed, as we partition our strings into two disjoint subsets
- term $3a_{n-1}$ (the cardinality of the subsets of strings that start by C, G or T), that can be interpreted as the cardinality of

  $$\{C, G, T\} \times \mathcal{A}_{n-1}.$$

  Similarly, a_{n-2} is similar to $1a_{n-2}$, the cardinality of

  $$\{AC\} \times \mathcal{A}_{n-2}.$$

Examples

Rooted, ordered, binary trees.

Let t_n denote the number of rooted ordered binary trees of size $n \geq 1$. We want to show that $t_1, t_2, \ldots$ satisfies the recurrence

$$\begin{cases} t_1 &= 1 \\ t_n &= \sum_{k=1}^{n-1} t_k t_{n-1-k} \quad n \geq 2 \end{cases}$$

Regions of a line arrangement.

Assume you draw n lines in the plane, with no two lines being parallel and no three (or more) lines intersecting at the same point. The number r_n of regions of the plane defined by these lines satisfies the recurrence

$$\left\{\begin{array}{lll} r_0 & = & 1 \\ r_n & = & r_{n-1} + n \quad n \geq 1 \end{array}\right.$$

Integer Compositions. What is the number of integer compositions of size n?

sequence of positive integers summing to n

n	int comp
1	(1)
2	$(2), (1,1)$
3	$(3), (2,1), (1,2), (1,1,1)$

let $c_n = \#$ int comp of n

$c_1 = 1$

$c_2 = 2$

$c_3 = 4$

int comp of n 1^{st} term is $1, 2, 3 \ldots n$

either

$(1, \underbrace{\quad\quad}_{\text{comp of } n-1})$

$(2, \underbrace{\quad\quad}_{\text{comp of } n-2})$

$\vdots$

$(n-1, \text{comp of } 1)$

(n)

So $c_n = c_{n-1} + c_{n-2} + \ldots + c_1 + 1$

closed form guess $c_n = 2^{n-1}$ for $n \geq 1$

pf sketch : induction   base $n=1$ $c_1 = 1 = 2^1$ ✓

ind step $c_n = c_{n-1} + \ldots + c_1 + 1$

$= 2^{n-1} + 2^{n-2} + \ldots + 2 + 1 + 1$

$= 2^n$

Strings with an even number of 1s. What is the number of strings of size n over the alphabet $\{0, 1, 2, 3\}$ that contain an even number of 1.?

Algorithm analysis. We are given an array of 2^n numbers, $A[1], \ldots, A[2^n]$ and we want to find the minimum m and maximum M in this array. We perform as follows:

- Find (recursively) the maximum and minimum (M_1, m_1) in $A_1 = A[1] \ldots A[2^{n-1}]$
- Find (recursively) the maximum and minimum (M_2, m_2) in $A_2 = A[1] \ldots A[2^{n-1}]$
- $M = \max(M_1, M_2)$ and $m = \min(m_1, m_2)$.

Our counting question is: what is the largest number of comparisons between pairs of elements of A that this algorithm could require.

lets find a recurrence! call the # of strings (satisfying the conditions) of length n a_n. Note $a_0 = 1$ (the empty string)
$a_1 = 3$ ((0), (2), (3))

For a string of length n either 1st letter is 0,2,3 and this is followed by a string of length n-1 (satisfying the conditions)

– there are $3a_{n-1}$ ways to make such strings or first letter is a 1. In this case the string looks like $1 \underbrace{0,2,3}_{k} 1 \underbrace{\qquad}_{n-2-k}$

the number of ways of making such a string is $3^k a_{n-2-k}$ but we have to consider all possibilities for k. Taking this into account, the number of strings starting with a 1 is given by

$$\sum_{k=0}^{n-2} 3^k a_{n-2-k}$$

Combining all of this gives the recurrence for $n \geq 1$ of

$$a_n = 3a_{n-1} + \sum_{k=0}^{n-2} 3^k a_{n-2-k}$$

note: $0 = a_{-1} = a_{-2} = \ldots$

Relevant examples of the textbook. Below is a list of examples of recurrences given in the textbook where you can see how the recurrences were obtained. They are all interesting in some way.

- 10.2: compound interest rate;
- 10.5: complexity of the bubble sort algorithm;
- 10.11: counting the number of subsets of $\{1, 2, \ldots, n\}$ that do not contain consecutive integers;
- 10.12: counting the number of tiling of an $n \times n$ square lattice with rectangles;
- 10.13: complexity of the Euclid algorithm for the greatest common divisor;
- 10.14: counting the number of parenthesis-free arithmetic expressions with n symbols;
- 10.15: counting the number of palindromes (strings that are invariant by mirroring, seen in a homework);
- 10.16: counting the number of binary strings with no two consecutive zeros;
- 10.19: counting the number of tiling of an $n \times n$ square lattice but with non-rectangle tiles;
- 10.28: counting the number of moves in the Towers of Hanoi game;
- 10.32: computing the area of the von Koch fractal;
- 10.35: complexity of computing Fibonacci numbers;
- 10.36: counting handshakes (a simplified model of pairwise communication).