Combinatorial Classes

Contents

1 The theme of this course: answering fundamental questions about combinatorial classes

This class is centered around the fundamental understanding of combinatorial classes. First we learn about the structures, and their properties, and then we understand the different places they appear in computer science, pure mathematics, biology and physics. If you study objects that appear in any of these discliplines, understanding the mathematics of structures will help you analyze these objects in a *smarter* way. The specific outline for our treatment of the topic is as follows:

1. Describe and become acquainted with some fundamental discrete structures, including variations of...
   (a) binary strings
   (b) permutations
   (c) set partitions
   (d) trees
   (e) random walks
   (f) set systems
   (g) graphs;
2. Determine the number of objects of a given size using generating functions and recurrence relations;
3. Describe bijections between classes that are equinumerous;
4. Generate random elements of each class using:
   (a) Recursive generation
   (b) Generating trees
   (c) Boltzmann generation;
5. Generate *all* elements of a given class
   (a) in lexicographic (dictionary) order
   (b) in a Minimal Change Order (eg. Gray codes, de Bruijn order)
6. Describe algorithms to search for precise elements:
   (a) Backtracking algorithms
   (b) Heuristic search.

2 Combinatorial Classes

Our work is centered around the notion of a combinatorial class. So what does this mean, exactly? Imagine a set of objects defined by a set of rules, coupled with some notion of size. For example, the set of all binary words constructed from the symbols $\{0, 1\}$, where the size is the length of the word defines a combinatorial class. Another class is the set of all ways to place n balls into 5 boxes. Here size is the the number of balls. Yet another class is the set of ways to walk along a lattice taking only unit steps $(1, 0)$ and $(0, 1)$ and arriving at the point of the form (n, n). Here, the size is given by n.

Here is a sample of the kinds of questions and the form of the answers that we will address in this course:

1. What are number of binary words of length 23, n? $(2^{23}, 2^n)$
2. Give an algorithm to generate a random binary string of length 23, such that every string is equally likely. *Flip a fair coin 23 times.*
3. Determine an algorithm to generate all binary words of length n, one at a time. *Build a binary odometer.*
4. How many binary strings of length 23 are there such that there is no substring with 5 consecutive ones? *!!!*

Obviously, as the classes become more complicated, the answers become more difficult to find. Our preference is for *universal, algorithmic* approaches. Let us become more formal now.

2.1 A formal definition

Definition. A **combinatorial class** $\mathcal{A}$, is a finite or denumerable set on which a **size function** is defined, satisfying the following conditions:

(i) the size of an element is a non-negative integer,

(ii) the number of elements of any given size is finite.

Some notation:

- The size of $\alpha \in \mathcal{A}$ is denoted $|\alpha|$.
- $\mathcal{A}_n$ denotes the subset of elements that have size exactly n.
- $A_n = \text{cardinality}(\mathcal{A}_n)$.

So given this, a central question is the computation of A_n, when possible.

Definition. The **counting sequence** of a combinatorial class is the sequence of integers $(A_n)_{n\geq 0}$ where $A_n = \text{cardinality}(\mathcal{A}_n)$ is the number of objects in class $\mathcal{A}$ that have size exactly n.

Let us start with a few very basic examples to introduce some underlying combinatorial objects and also some notation and ideas.

Example (Binary words)**.** Let $\mathcal{W}$ be the set of **binary words** — sequences of elements taken from the binary alphabet $\mathcal{A} = \{0, 1\}$:

$$\mathcal{W} = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, \dots\}$$

where ϵ is the empty word. The most natural definition of "size" for $\omega \in \mathcal{W}$ is the number of letters in ω.

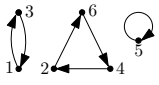
Since each letter of ω may be chosen independently of all the others and there are exactly 2 choices, the counting sequence is simply

$$W_n = (\mathcal{W}_n) = 2^n.$$

Exercise. Suppose the size of a binary word is defined as the number of zeroes in the string. Is this a combinatorial class?

The next example is fundamental across many sciences, and has many different ways of presenting itself.

Example (Permutations)**.** A **permutation** of size n is by definition a bijective mapping of the integer interval $I_n = [1..n]$ to itself. When we get around to playing with them we will frequently write them as an array $\begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma_1 & \sigma_2 & \cdots & \sigma_n \end{pmatrix}$, or equivalently by the sequence $\sigma_1\sigma_2\cdots\sigma_n$. We may represent a permutation by drawing a directed graph whose vertices are the integers $[n]$, and a directed edge from each vertex i to σ_i. A fourth way to represent a permutation is the **cyclic form**, which describes the cycles of this graph in the format $(1, \sigma_1, \sigma_{\sigma_1}, \dots)(i, \sigma_i, \sigma_{\sigma_i}, \dots)\cdots(\dots)$. For example, the permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 1 & 2 & 5 & 4 \end{pmatrix}$ can be represented in any of the following three ways.

$3, 6, 1, 2, 5, 4$  $(13)(264)(5)$.

The set of permutations is

$$\mathcal{P} = \{1, 12, 21, 123, 132, 213, 231, 312, 321, 1234, \dots\}$$

As we have likely seen in high school, we have n-choices for σ_1, $(n-1)$ choices for σ_2 and so on. Hence the number P_n of permutations of size n is

$$P_n = n! = n \cdot (n-1) \cdots 2 \cdot 1$$

Example (Graphs)**.** A **(finite simple) labeled graph** $\mathcal{G} = (V, E)$ consists of a finite set V of vertices and a finite set E of edges such that each edge is a two element subset of vertices. The size of a graph is often the number of vertices, but can also be the number of edges (if the graphs are simple and connected), or some other parameter as long as there are a finite number of graphs for any given size. The word **graph** usually means **unlabeled** graph. Here two labeled graph are considered to be equivalent if one labeled graph can be obtained from the other by changing the labels of its vertices. Every equivalence class of graphs is an unlabeled graph. Sometimes we allow graphs to have **multiple edges** and **loops**. A **directed graph** is a graph each edge has one end which is called its **head** and the other end is called its **tail**.

Example (Latin Squares)**.** A **Latin square** of size, or order n is an $n \times n$ array A whose entries are chosen from an n-element set $\mathcal{Y}$ such that each symbol in $\mathcal{Y}$ occurs in exactly one cell in each row and column.

$$\mathcal{Y} = \{1, 2, 3, 4\} \quad A = \begin{array}{cccc} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{array}$$

These have an application in error correcting codes, and Sudoku.

2.2 Questions we might ask about a class

Now that we have established our notation, let us rephrase our main questions.

Representation How do you represent the elements of $\mathcal{C}_n$? (for input to a computer program, say)

Enumeration Given combinatorial class $\mathcal{C}$, determine a method to determine C_n. *This becomes difficult when there is more than one representation for the same object!*

Random generation Give an algorithm to construct an element in $\mathcal{C}_n$ with uniform probability. That is, each element has probability $1/C_n$ of being generated.

Exhaustive generation Give an algorithm to construct every element in $\mathcal{C}_n$. *Observe, that if you can do this, you can probably count all of the objects. Can you think of a way to combine the previous two problems to determine a solution to this problem? Is it efficient?*

Search Determine an element (if it exists) of a particular form from $\mathcal{C}$. Equivalently, can you determine n for which $C_n = 0$? *Is this property easy to determine given your representation?*

In all of these cases we are also interested in the standard algorithmic questions: what is the performance of our algorithm.