Combinatorics Notes

Sera Gunn

2026-01-24

Table of contents

About

Notes for applied combinatorics, intended to supplement the book Applied Combinatorics by Keller and Trotter

Contains:

- An alternate proof of the Inclusion/Exclusion formula and a fully worked out proof of the Euler Totient formula
- Notes on the structure of permutations (cycles, inversions, signs)
- Notes on generating functions formalized through regular languages
- Notes on finite state automata and their connection to regular languages and generating functions

1 Preface

These notes were developed to accompany MTH 337: Applied Combinatorics during the Spring semester of 2024. They are meant to be used interspersed with the material from Keller and Trotter's book Applied Combinatorics. Both these notes and Keller and Trotter's book are intended to have a heavy algorithmic and applied focus.

A sample syllabus is (KT is Keller and Trotter's book)

1. KT-2: Strings, Sets, and Binomial Coefficients
2. KT-3: Induction
3. KT-4.1: Pidgeonhole
    - Can cover the rest of KT-4 if students haven't seen "big-O" in their other courses yet
4. KT-5: Graph Theory
5. KT-7 and Inclusion-Exclusion from these notes for supplementary proofs
6. Permutations from these notes
7. KT-12, 13, 14 on graph algorithms (Prim/Kruskal, Dijkstra, Ford-Fulkerson/Edmonds-Karp)
8. Generating Functions from these notes (instead of KT-8)
9. Regular Languages from these notes

Here is a rough overview of how these notes tie into the Keller/Trotter book.

Inclusion/Exclusion makes a very brief appearance Note 2 but is otherwise separate in these notes.

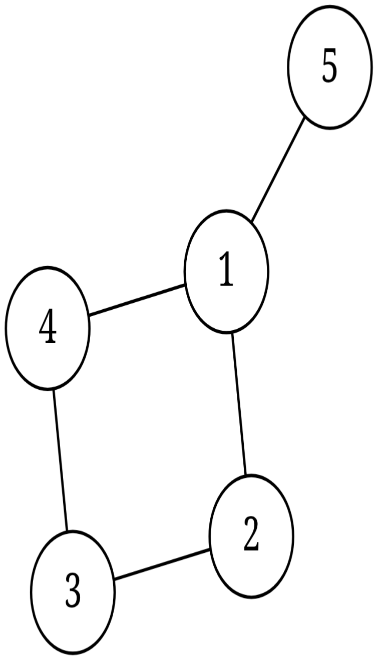
Part I

Trees

2 Trees

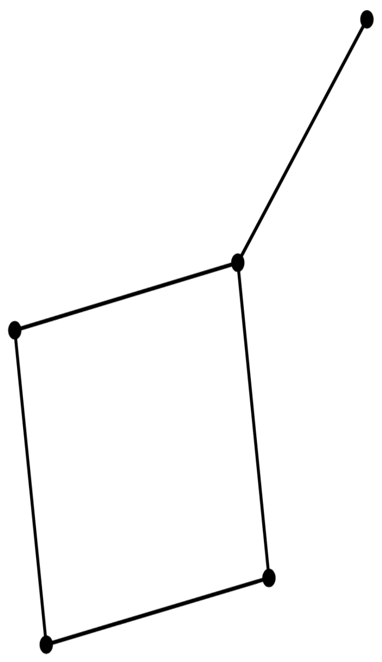
Definition 2.1. A **graph** consists of a set of vertices V and a set of edges E. Each edge is an unordered pair $\{v, w\}$ representing a connection between the vertices v and w.

Consider the graph where the set of vertices are $V = \{1, 2, 3, 4, 5\}$ and the edges are $\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}$ and $\{1, 5\}$. Graphs such of these are more easily depicted as pictures. We try as much as possible to talk about the picture rather than sets of numbers.



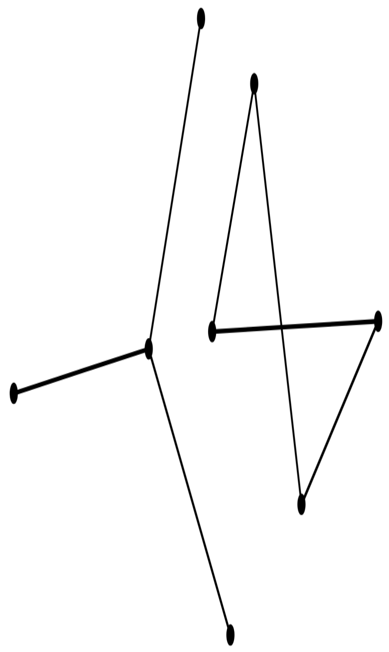
A graph consisting of 4 vertices in a cycle and a fifth vertex branching off

Per the definition, a graph is **labelled** (usually the vertices are $1, \dots, n$). Graph theory has a lot of applications and theory has a lot of applications and results in which these labels play no role and so we also have simplified pictures where we do not write down the vertex number (however they still exist in the background).



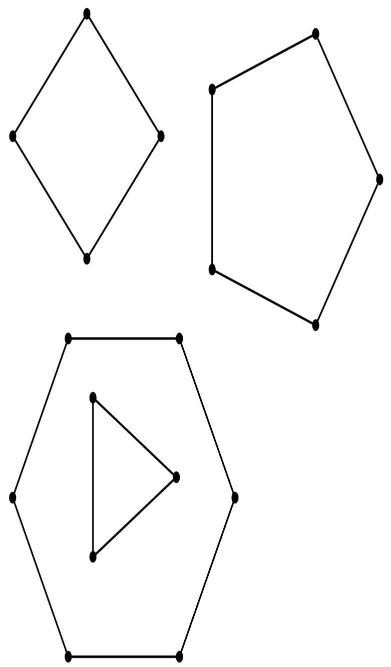
The same graph but without labels

Definition 2.2. A **tree** is a special type of graph which satisfies two additional properties: 1. the graph must be **connected**. We will give a mathematical definition for this later, but for now, this is a picture of what a **disconnected** graph is:



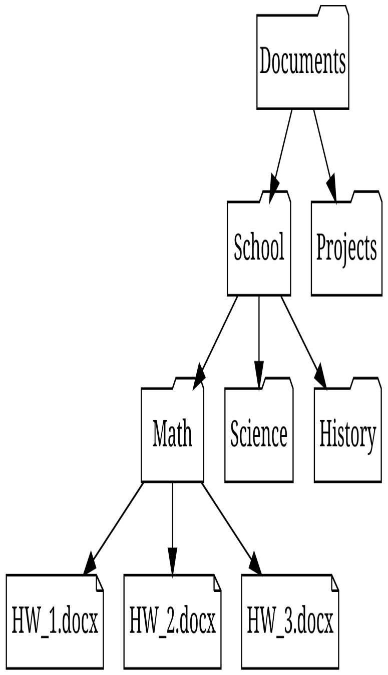
A disconnected graph consisting of two components

2. the graph cannot have any cycles. Again, we will just use pictures right now:

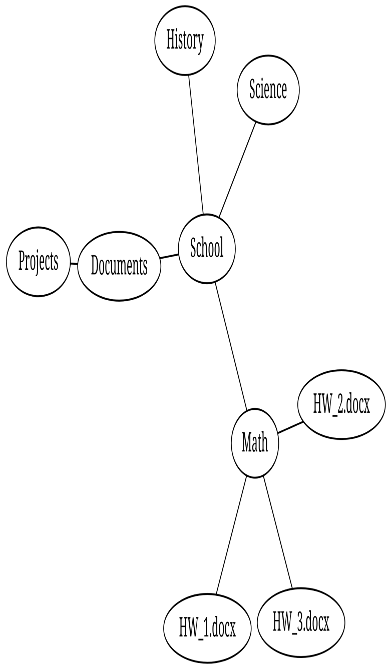


Various Cycles

Trees are common structures in a lot of computer science applications. For instance, filetrees are examples of trees.



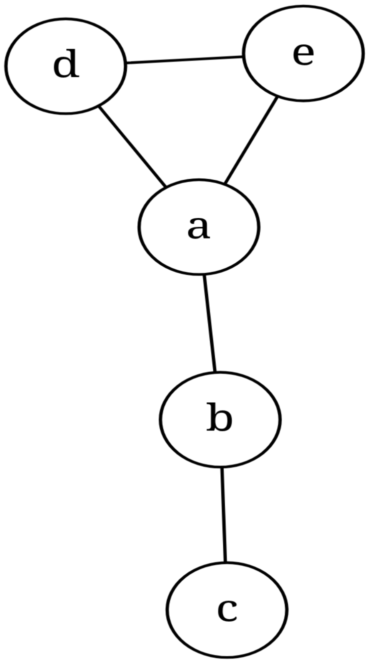
Many applications have a hierarchical picture to tree but this visual hierarchy is not part of our base definition (although we can always add a hierarchy on top of the definition). Here is that same tree but without the hierarchy as part of the picture



2.1 Properties of Graphs and Trees

The first property is called the *handshaking lemma* or *degree sum formula.*

Definition 2.3. The **degree** of a vertex, deg(v) is the number of other vertices w for which there is an edge between v and w. For instance, the vertex a below has a degree of 3 because it is connected to 3 other vertices: b, d, e.



The vertex a has a degree of 3

2.1.1 Double Counting

An important idea in combinatorics is *double counting* where we count the same thing in two different ways. For instance, in the above graph we can count the edges just by looking at the picture (5 edges). Or... we can go vertex by vertex and count the edges at that vertex: a has 3, b has 2, c has 1, d and e both have 2. This gives 10 total. If we think carefully about what is going on here, this second method of vertex-by-vertex counts every edge twice. For instance the edge between a and b contributes both to the count for a and for b. This idea is called the *handshaking lemma* or *degree-sum formula.*

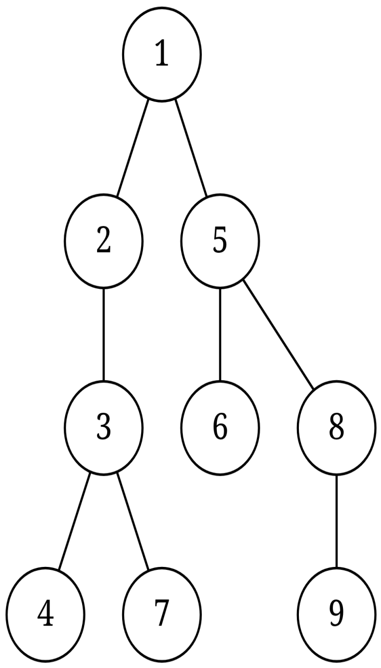
Theorem 2.1. *If we go vertex by vertex and add up the number of edges at that vertex (i.e. the* degree*) every edge is counted exactly twice.*

$$\sum \deg(v) = 2|E|.$$

2.2 Leaf Vertices

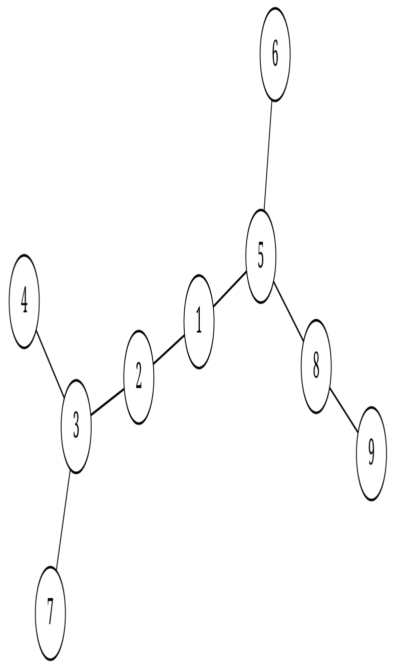
In developing a developing an understanding of all the properties that trees have, one foundational fact is that every tree with at least two vertices has a **leaf vertex** where a leaf vertex is a vertex with degree 1.

In a filesystem, any file is a leaf where its unique edge is the edge connecting it to the parent folder. Additionally, any empty folder (that is not the topmost folder) is also a leaf.



In this tree, vertices 4, 6, 7, 9 are leaves

I want to take a moment here also to point out that the tree is the data of which vertices are connected and in particular lengths and directions of edges are not part of the data. So we could draw the tree with the vertices placed elsewhere and it will still be the same tree with the same leaves.



The same tree with a different drawing

2.2.1 Every tree has at least one leaf

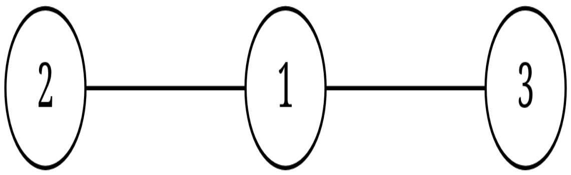
It is a fact that every tree with 2 or more vertices has at least one (in fact at least two) leaf vertex. We will discuss the why of this (the proof) in a couple levels of formality.

Note

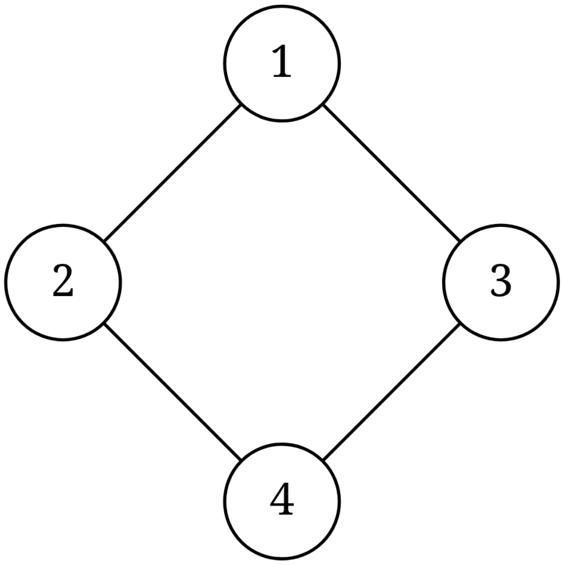
The reason we say here "2 or more vertices" is because if we have only a single vertex that vertex would have no edges (degree 0) and would not be a leaf because a leaf needs to have exactly one edge.

2.2.1.1 Informal proof

Lets pick any vertex, maybe vertex 1. We have two possibilities 1. this vertex is a leaf (and we have what we were looking for) 2. this vertex has at least two neighbours and in the picture we will call those 2 and 3



Now repeat this reasoning: either 2 or 3 are leaves or they too have a new neighbour other than 1.

> **Important point**
>
> All of these new neighbours are distinct (e.g. the new neighbour for 2 is not the same as the new neighbour for 3). If lets say the new neighbour for 2 and 3 were the same well then that would create a cycle (not allowed)
>
> 
>

So as long as we never hit a leaf vertex we can always find more vertices. But there are only a finite amount of vertices so we cannot continue this indefinitely. At some point we have to find a leaf vertex (and in fact, one on each side).

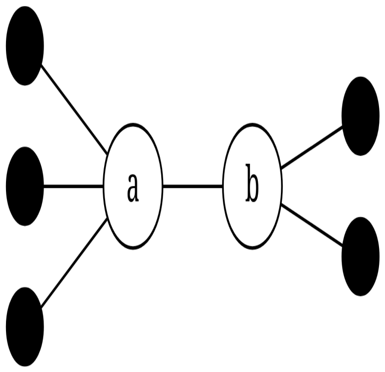
2.2.1.2 Induction

This idea of "just keep going and eventually we have to stop because there was only a finite number" has a name. It is known as the **principle of mathematical induction**. Suppose we show two things: 1. the minimal sized case(s) have a property 2. any big case can be made into a smaller case then we are done because any case can be made smaller and smaller until we eventually find what we are looking for and the process stops because everything is finite.

In terms of trees, the minimal case is a tree consisting of a pair of vertices and a single edge. Each vertex here is a leaf edge.

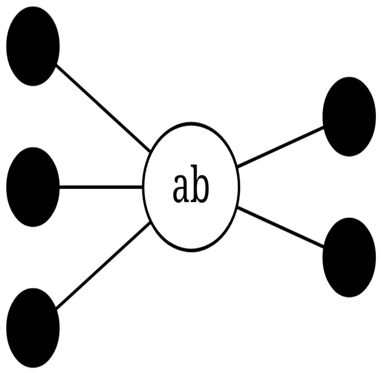
Minimal tree with 2 vertices

Now suppose we have a larger, more complex tree. Focus on a single edge somewhere in the tree, let us call its endpoints a and b.



Larger tree focusing on the edge ab

If neither a nor b are leaves then we smush the two vertices together merging the entire edge into single vertex which we will label "ab." This process is known as **edge contraction**.



Contracting the edge.

By contracting the edge we create a smaller graph with one fewer vertex and one fewer edge.

This process may be carried out again and again. Choose a new edge, either we find a leaf vertex or we contract. Because we cannot contract indefinitely we say now "by induction" there must be a leaf vertex.

2.2.1.3 A little bit more formal

Going the next step, induction is often thought about as "going up" rather than "going down." We worded the last argument as creating smaller and smaller graphs but another way to say that is

1. Every tree with two vertices (there is only one such tree) has a leaf vertex
2. Every tree with three vertices may be smushed (contracted) to a tree with two vertices thus every tree with three vertices has a leaf vertex.
3. Every tree with four vertices may be contracted to a tree with three vertices so every tree with four vertices has a leaf vertex
4. (Induction is where we observe in formal terms that this pattern extends to all whole numbers)
5. Every tree with $n+1$ vertices has a leaf vertex because (in the step before) every tree with n vertices has a leaf vertex (because every tree with $n-1$ vertices has a leaf vertex because every tree with $n-2$ vertices has a leaf vertex...)

Induction is revisited in Keller and Trotter's book, chapter 3.

2.2.2 Counting Leaves

Actually, before we go any further we need another fact

Theorem 2.2. *For any tree, there is always exactly one more vertex than edge. I.e.* $|V| = |E| + 1$.

Proof. Have a look at what we did in the previous section for induction.

1. A tree with 1 vertex has 0 edges
2. A tree with 2 vertices has 1 edge
3. A tree with 3 vertices has 2 edges
4. A tree with $n+1$ vertices may be smushed into a tree with n vertices. This contraction process removes 1 vertex and 1 edge so the property of "$|V| = |E| + 1$" is preserved.

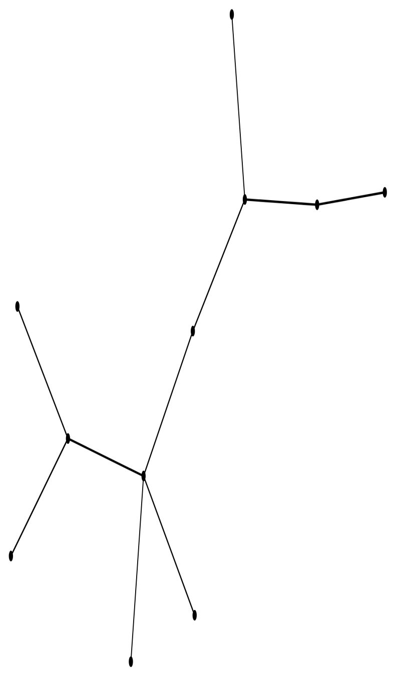
So by induction all trees have $|V| = |E| + 1$. □

Where the last few sections were about induction, this one will show some ideas for enumeration (i.e. counting stuff). Going all the way back up to the Handshake Lemma Theorem 2.1 let us break up that formula a bit in a way that lets us count leaf vertices.

To that end, let n_d be the number of vertices of degree d. So n_1 is the number of vertices of degree 1 (leaf vertices).

Tip

A common trick in combinatorics is if we have a function like deg and we want to count where $\deg = 1$ then we also count where $\deg = 2, \deg = 3, \ldots$ and then later isolate $\deg = 1$.



In this tree, we have $n_1 = 6, n_2 = 2, n_3 = 2, n_4 = 1$. The degree sum is

$$\begin{aligned}\sum \deg(v) &= 1+1+1+1+1+1+2+2+3+3+4 \\ &= 1\cdot 6 + 2\cdot 2 + 3\cdot 2 + 4\cdot 1 \\ &= 1n_1 + 2n_2 + 3n_3 + 4n_4 = 2|E| = 20\end{aligned}$$

So separating the degree sum into $n_1, n_2, \ldots$ we get

$$\sum_v \deg(v) = \sum_d dn_d = 2|E|.$$

Spelled out:

$$n_1 + 2n_2 + 3n_3 + 4n_4 + \cdots = 2|E|.$$

Now let us look at a similar sum:

$$n_1 + n_2 + n_3 + n_4 + \cdots$$

So that is the number of vertices of degree 1, degree 2, degree 3, ... well that is every vertex so this sum is $|V|$. Or, as we learned in Theorem 2.2, $|V| = |E| + 1$.

So we have two identities:

$$\begin{aligned} n_1 + 2n_2 + 3n_3 + 4n_4 + \cdots &= 2|E| \\ n_1 + \ \ n_2 + \ \ n_3 + \ \ n_4 + \cdots &= |E| + 1. \end{aligned}$$

Subtract twice the second from the first and we have

$$-n_1 + \qquad + n_3 + 2n_4 + \cdots = -2.$$

Almost done! Lastly move the $-n_1$ to the right and the -2 to the left and we get our main result for this section.

Theorem 2.3. *In any tree (with at least two vertices), the number of leaves is*

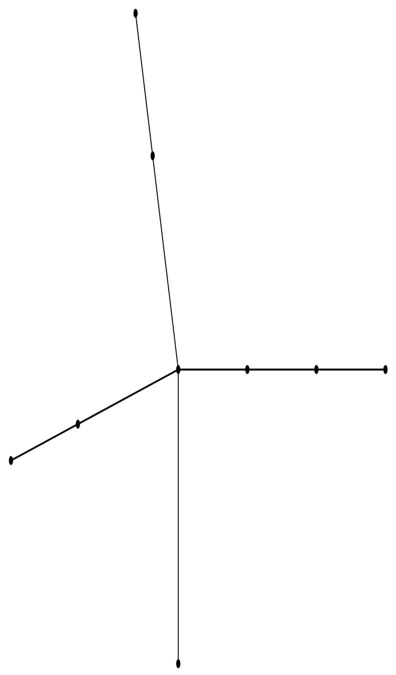
$$n_1 = 2 + n_3 + 2n_4 + 3n_5 + 4n_6 + \cdots$$

This tells us a few interesting things all at once!

Corollary 2.1. *Every tree with at least two vertices has at least two leaves:* $n_1 \geq 2$.

Corollary 2.2. *Moreover, for every vertex of degree 3 there is an additional leaf. For every vertex of degree 4 there are 2 additional leaves and so on.*

We can look at this last result in another way: take a vertex (let's say of degree 4) and follow the 4 edges until they lead to leaves. So this degree 4 vertex gives 4 leaves at least (more if there are other branches to follow).



a degree 4 vertex leading to four leaves

Part II

Inclusion-Exclusion

3 Inclusion/Exclusion

3.1 For two sets

Example 3.1. Suppose there are 100 CS students an a school. Of these, 50 know Java and 60 know Python (everyone knows at least one of these two languages). If we add $50 + 60$ we get the 100 total students plus 10 students who know both.

$$50 + 60 = 100 + 10.$$

Given two sets A and B, we can compute $|A \cup B|$ by first adding everything in A and adding everything in B. This counts all of $A \cup B$ except the items in $A \cap B$ were counted twice. Thus

$$|A| + |B| = |A \cup B| + |A \cap B|.$$

3.2 For three sets

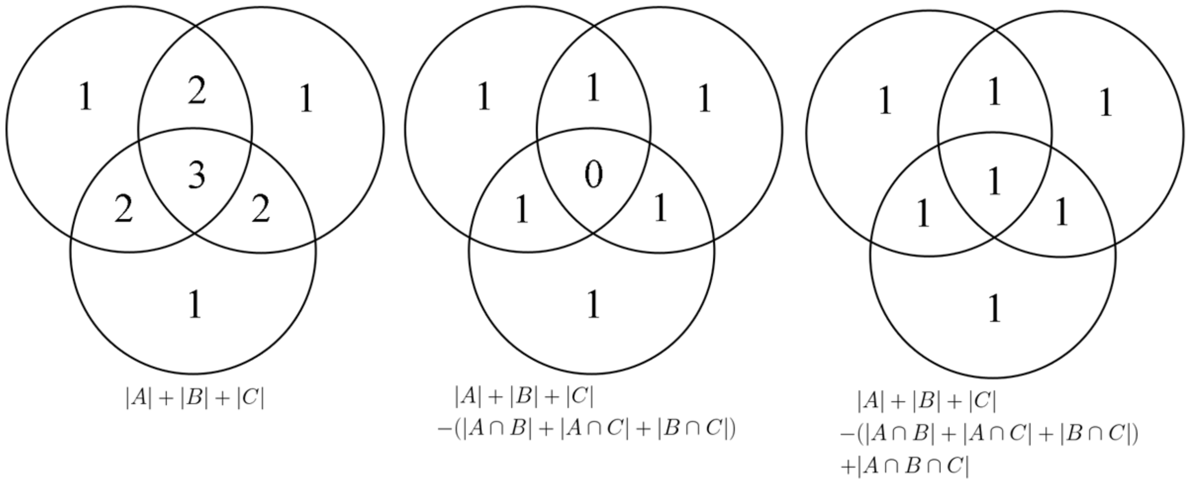


Figure 3.1: Inclusion/Exclusion for 3 sets

We can do a similar process for three sets. Start by adding up $|A|+|B|+|C|$ then subtract all the intersections of pairs then add back in $|A \cap B \cap C|$. For items in each region of the Venn diagram, think about how many times an item is added/subtracted overall.

E.g. items in A and B but not C will be added in twice in the first step ($|A|+|B|$) and subtracted once in the second step ($|A \cap B|$) so they are counted once.

Items in $A \cap B \cap C$ are added in $3-3+1=1$ times.

3.3 Binomial coefficients and I/E

Suppose we are looking at a union of n sets: $A_1 \cup \cdots \cup A_n$. Consider an element x belonging exactly to $A_1, \ldots, A_m$ and no other sets. If we do the same procedure of adding $|A_1|+\cdots+|A_n|$ then we are counting x a total of m times. Then, when we subtract all the pairwise intersections, $|A_i \cap A_j|$, we are subtracting from the count of x a total of $\binom{m}{2}$ because there are $\binom{m}{2}$ ways to choose two indices $\{i, j\} \subset \{1, \ldots, m\}$.

Thus, if we alternate: adding in single sets, subtracting pairs of intersections, adding triples, subtracting quadruples, etc. we are counting x a total of

$$\binom{m}{1} - \binom{m}{2} + \binom{m}{3} - \binom{m}{4} + \cdots \tag{3.1}$$

times.

Let's have a closer look at this. We know the Binomial Theorem says that

$$(1+x)^m = \binom{m}{0}x^0 + \binom{m}{1}x^1 + \binom{m}{2}x^2 + \cdots + \binom{m}{m}x^m.$$

Substituting $x=-1$, we get

$$0 = \binom{m}{0} - \binom{m}{1} + \binom{m}{2} - \cdots + \binom{m}{m}(-1)^m.$$

And since $\binom{m}{0} = 1$, if we move all the other terms to the other side of the equation, we see that Equation 3.1 evaluates to 1.

Note 1

For this problem we are counting the size of $A_1 \cup \cdots \cup A_n$ so each element is in at least one set ($m \geq 1$). This is important so that we can say that $0^m = 0$. In the context of the binomial theorem, $(1-1)^0 = \binom{0}{0} = 1$. This will be important later when we want to count elements not belonging to any set.

The general I/E formula is then

$$\begin{aligned}|A_1 \cup \dots \cup A_n| &= \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \dots \\ &= \sum_{t=1}^{n} (-1)^{t+1} \sum_{i_1 < \dots < i_t} |A_{i_1} \cap \dots \cap A_{i_t}|.\end{aligned}$$

Note: $(-1)^{t+1}$ takes the values $1, -1, 1, -1, \dots$ starting at $t = 1$.

3.4 Avoiding properties

Inclusion/Exclusion appears most frequently in combinatorics not as a means to count $|A_1 \cup \dots \cup A_n|$ directly but rather as a means to count everything *not* in $A_1 \cup \dots \cup A_n$.

For example, let $[m] = {1,...,m} and suppose we want to count the number of functions $f : [m] \to [n]$ which *don't* miss anything in the codomain. I.e. if A_i is the set of functions where $f(x)$ never equals i, then we want to count every function *not* in any set A_i.

Let X be the total set of functions. Then to count the functions avoiding $A_1 \cup \dots \cup A_n$ we do

$$\begin{aligned}|X \setminus (A_1 \cup \dots A_n)| &= |X| - |A_1 \cup \dots A_n| \\ &= |X| - \sum_i |A_i| + \sum_{i<j} |A_i \cap A_j| - \cdots\end{aligned}$$

Caution

The formula $|A \setminus B| = |A| - |B|$ works only when B is contained entirely inside A.

3.5 Simplifying notation.

Let $\mathscr{P} = \{P_1, \dots, P_n\}$ be a collection of sets representing negative properties—conditions we want to avoid. For a subset $S \subseteq \{1, \dots, n\}$, let $N_{\geq}(S)$ be the number of items satisfying at least those properties in S. I.e. $\bigcap_{i \in S} P_i$. We will say "x satisfies S" for short. Also, if $S = \{a, b, c\}$, let us write for example, $N_{\geq}(a, b, c)$ instead of $N_{\geq}(\{a, b, c\})$ for simplicity.

Let $N_{=}(S)$ be the number of x satisfying exactly those properties in S and no properties not in S. When $\mathscr{P}$ represents properties we wish to avoid, then $N_{=}(\varnothing)$ is the number of elements satisfying none of the properties.

Theorem 3.1 (Inclusion/Exclusion)**.**

$$\begin{aligned} N_=(\varnothing) &= N_\geq(\varnothing) - \sum_i N_\geq(i) + \sum_{i<j} N_\geq(i,j) - \cdots \\ &= \sum_S (-1)^{|S|} N_\geq(S). \end{aligned}$$

Proof. As in Section 3.3, we will break up the sum focusing on each element. So first, we will write

$$\sum_S (-1)^{|S|} N_\geq(S) = \sum_S (-1)^{|S|} \sum_{\substack{x \\ x \text{ satisfies } S}} 1$$

Here we replace the number $N_\geq(S)$ by a count of 1 for each element x which satisfies S. This introduces a second sum into the picture and that will allow us to swap the order of summations. Currently, the second sum is over all x with respect to the relation "x satisfies S." When we put the sum over x outside, we still have the relation "x satisfies S" but rather than summing over all x with this property, we sum over all S:

$$\sum_S (-1)^{|S|} \sum_{\substack{x \\ x \text{ satisfies } S}} 1 = \sum_x \sum_{\substack{S \\ x \text{ satisfies } S}} (-1)^{|S|}.$$

We also move the $(-1)^|S|$ inside the second sum because that quantity depends on $|S|$.

Next, just as we did in Section 3.3, let us say that x satisfies exactly $P_{i_1}, \ldots, P_{i_m}$ (m will depend on x) and let $S_x = \{i_1, \ldots, i_m\}$. This set is the largest set that x satisfies. Every other set that x satisfies will be a subset of S_x.

We now break up our sum based on the size of those subsets $S \subseteq S_x$, using the binomial coefficients to count the number of such S:

$$\sum_x \sum_{\substack{S \\ x \text{ satisfies } S}} (-1)^{|S|} = \sum_x \sum_{k=0}^{m} \underbrace{\binom{m}{k} (-1)^k}_{\substack{|S|=k \text{ and } x \text{ satisfies } S \\ \iff |S|=k \text{ and } S \subseteq S_x}} .$$

This, by the Binomial Theorem, is the same as

$$\sum_x (1-1)^m.$$

Remember here that m depends on x.

As discussed in Note 1, $0^m = 0$ if $m \geq 1$ but if $m = 0$ then $0^0 = 1$. Saying $m = 0$ means that x satisfies exactly 0 properties—which is what we are looking for. So the final simplification looks like

$$\sum_x 0^m = \sum_{\substack{x \\ m=0}} 1 = N_=(\varnothing).$$

□

3.6 Application 1: Surjections

As in Section 3.4, let X be the set of all functions from $[m]$ to $[n]$ and let P_i be the set of functions where i is never an output: $f(x) \neq i$ for any input x.

Lemma 3.1.

a) *The number of functions from an m element set to an n element set is n^m.*
b) *The number of functions from an m element set to an n element set that avoid k outputs is $(n-k)^m$.*

Proof.

a) There are n choices for $f(1)$ and n choices for $f(2)$, etc. So there are n^m choices in total for $f(1), \ldots, f(m)$.
b) If we are avoiding k outputs then there are only $n-k$ choices for each of $f(1), \ldots, f(m)$ so $(n-k)^m$.

□

With this in mind, we have $N_\geq(S) = (n-k)^m$ if $|S| = k$ (we avoid at least the specified k outputs). So by inclusion exclusion, the number of functions which avoid *no* outputs (i.e. surjections) is

$$\begin{aligned} &N_\geq(\varnothing) - \sum_i N_\geq(i) + \sum_{i<j} N_\geq(i,j) - \cdots \\ &\qquad = n^m - \sum_i (n-1)^m + \sum_{i<j} (n-2)^m - \cdots . \end{aligned}$$

And since there are $\binom{n}{k}$ ways to choose k outputs to avoid, we can also write this as

$$n^m - \binom{n}{1}(n-1)^m + \binom{n}{2}(n-2)^m - \cdots = \sum_k \binom{n}{k}(-1)^k(n-k)^m.$$

The textbook calls this number $S(m, n)$.

Note

In the above application, $N_{\geq}(S) = (n-k)^m$ only depended on the size k of S. This is true in many but not all examples.

3.7 Application 2: Derangements

Let X be the set of all permutations of $[n]$. We wish to count the permutations with no fixed points. So let P_i be the property that i is a fixed point. Then $N_{=}(\varnothing)$ is the number of permutations with zero fixed points. This number is called d_n in the textbook.

Lemma 3.2.

a) The number of permutations of $[n]$ is $n!$
b) The number of permutations with at least k specified fixed points is $(n-k)!$

Proof.

a) Discussed in Chapter 2 of the book.
b) To say that there are k specified fixed points means were are permuting the other $n-k$ items. Similar to (a), there are $(n-k)!$ ways to permute $n-k$ items.

□

Applying Inclusion/Exclusion, we thus have

$$d_n = n! - \binom{n}{1}(n-1)! + \binom{n}{2}(n-2)! - \cdots = \sum_k \binom{n}{k}(-1)^k(n-k)!.$$

Again, there are $\binom{n}{k}$ ways to choose a set S of k fixed points and each of these has the same number $N_{\geq}(S) = (n-k)!$.

4 Euler's Totient Function

Consider the number system of the integers mod n. This means we group the numbers based on whether they have the same remainder when divided by n. For example, $3 \cdot 7 \equiv 1 \pmod{10}$ because both 21 and 1 have the same remainder when divided by 10.

This number system works a lot like the integers. We have addition, subtraction and multiplication. The question is: when can we divide? For example, is it possible to solve $2x \equiv 3 \pmod{10}$ for x? I.e. is there a way to move the 2 to the other side of the equation? The answer is no, because $2x$ will never have a remainder of 3.

What about $7x \equiv 3 \pmod{10}$? This is trickier. Let's look at our multiples of 7:

$$0, 7, 14, 21, 28, 35, 42, 49, 56, 63$$

Ok so we see that $7 \cdot 9 \equiv 3 \pmod{10}$.

The next theorem gives the criterion for when we can "divide both sides by a."

Theorem 4.1. *We can solve the equation $ax \equiv b \pmod{n}$ for any b if and only if $\gcd(a, n) = 1$.*

Proof. First, suppose $\gcd(a, n) \neq 1$ and call this gcd d. Consider the equation $ax \equiv 1 \pmod{n}$. This means that ax has a remainder of 1 when divided by n or in other words ax is 1 more than a multiple of n: $ax = qn + 1$. Now, d is a divisor of both a and n by definition, which means $ax - qn$ should be a multiple of d. But $ax - qn = 1$ and we said $d \neq 1$, that is impossible. So $ax \equiv 1 \pmod{n}$ has no solution.

Next, suppose $\gcd(a, n) = 1$. By Theorem 3.9 of the Keller/Trotter book, also known as Bézout's theorem, we can find integers u and v such that $au + nv = \gcd(a, n) = 1$. If we rearrange, we get $au = 1 - nv$. So au and 1 differ by a multiple of n.

Let's apply this to our equation: take the equation $ax \equiv b \pmod{n}$ and multiply by u to get

$$aux = (1 - nv)x \equiv bu \pmod{n}$$

Since $(1 - nv)x$ and x differ by a multiple of n, they share the same remainder. Thus our solution is $x \equiv bu \pmod{n}$. $\square$

Example 4.1. Let's follow the steps to solve $7x \equiv 3 \pmod 1)0$. First, we do Euclid's algorithm to find the gcd of 7 and 10.

$$\begin{aligned} 10 &= 7 + 3 \\ 7 &= 2 \cdot 3 + 1 \end{aligned}$$

To get Bézout's identity: we solve each equation for the remainder term and then substitute the bigger remainder into the next equation:

$$\begin{aligned} 3 &= 10 - 7 \\ 1 &= 7 - 2 \cdot 3 = 7 - 2 \cdot (10 - 7) = 3 \cdot 7 - 2 \cdot 10. \end{aligned}$$

So 3 is our u. Multiply both sides of $7x \equiv 3 \pmod{10}$ by 3 and we get

$$\begin{aligned} 3 \cdot 7x &\equiv 3 \cdot 3 \pmod{10} \\ 1 \cdot x &\equiv 9 \pmod{10} \end{aligned}$$

4.1 The totient function

Numbers where $\gcd(a, n) = 1$ are called "coprime to n." They show up often when talking about modular arithmetic. We have a function, $\phi(n)$ called "Euler's totient function" which counts the number of integers less than n which are coprime to n. For example, $\phi(10) = 4$ counting $1, 3, 7, 9$.

Euler gave the following formula for $\phi(n)$ in terms of the prime factors p of n:

$$\phi(n) = n \cdot \prod_{\substack{p \mid n \\ p \text{ prime}}} \left(1 - \frac{1}{p}\right)$$

Here $p \mid n$ means p divides n or p is a factor of n.

This is the form we will prove in a minute, but let us also rewrite this formula in maybe a more helpful way. Say $n = p_1^{k_1} \cdots p_r^{k_r}$ is the prime factorization of n. Then

$$\phi(n) = p_1^{k_1} \cdots p_r^{k_r} \cdot \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right).$$

Now look at each pair of factors involving p_i:

$$p_i^{k_i} \left(1 - \frac{1}{p_i}\right) = p_i^{k_i} \left(\frac{p_i - 1}{p_i}\right) = p_i^{k_i - 1}(p_i - 1).$$

Putting that together, we get

$$\phi(n) = p_1^{k_1-1}(p_1-1)\cdots p_r^{k_r-1}(p_r-1).$$

Example 4.2. For $n = 18 = 2 \cdot 3^2$ we have

$$\phi(12) = 2^0(2-1)3^1(3-1) = 6.$$

You can double check this by listing all the numbers coprime to 18.

4.2 Applying Inclusion/Exclusion

For each of the primes p_i which are factors of n. Consider the property P_i which says that a is divisible by p_i. As we've been doing, for a subset of these primes, we want to count the number of a in $0, \dots, n-1$ which are divisible by at least those primes in S.

Lemma 4.1. *Let m be a divisor of n. The number of integers $a < n$ which are a multiple of m is n/m.*

Proof. Let's look at an example first. Suppose $n = 100$ and $m = 5$. Then we list the numbers from $0, \dots, 99$ and write the list mod 5. That is:

$$0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, \dots, 0, 1, 2, 3, 4.$$

Because $n-1$ is one less than n, our last number has a remainder of one less than 5. So we see every number $0, 1, 2, 3, 4$ exactly the same number of times. Since there are 5 possible remainders, we much see every remainder $100/5 = 20$ times. This includes a remainder of 0.

This pattern works generally: we repeat remainders of $0, \dots, m-1$ and we end on $m-1$ because $n-1$ is one less than a multiple of m. $\square$

So putting this into our inclusion-exclusion notation, if $S = \{i_1, \dots, i_k\}$, then

$$N_{\geq}(S) = \frac{n}{p_{i_1} \cdots p_{i_k}}.$$

4.3 Wrapping up

By Inclusion/Exclusion, the number of whole numbers $a < n$ not divisible by any p_i is

$$N_{\geq}(\varnothing) - \sum_i N_{\geq}(i) + \sum_{i<j} N_{\geq}(i,j) - \cdots$$

and that is

$$n - \sum_i \frac{n}{p_i} + \sum_{i<j} \frac{n}{p_i p_j} - \cdots = n\left(1 - \sum_i \frac{1}{p_i} + \sum_{i<j} \frac{1}{p_i p_j} - \cdots\right).$$

We should be able to factor a $(1 - 1/p_1)$ out of this so let's work at giving that a go by separating terms including p_1 from those that don't:

$$\begin{aligned}
1 &- \sum_i \frac{1}{p_i} + \sum_{i<j} \frac{1}{p_i p_j} - \cdots \\
&= 1 - \frac{1}{p_1} - \sum_{1<i} \frac{1}{p_i} + \frac{1}{p_1} \sum_{1<j} \frac{1}{p_j} + \sum_{1<i<j} \frac{1}{p_i p_j} - \cdots \\
&= \left(1 - \frac{1}{p_1}\right) - \left(\frac{1}{p_1} \sum_{1<i} \frac{1}{p_i} - \frac{1}{p_1} \sum_{1<j} \frac{1}{p_j}\right) + \left(\sum_{1<i<j} \frac{1}{p_i p_j} - \frac{1}{p_1} \sum_{1<i<j} \frac{1}{p_i p_j}\right) - \cdots \\
&= \left(1 - \frac{1}{p_1}\right)\left(1 - \sum_{1<i} \frac{1}{p_i} + \sum_{1<i<j} \frac{1}{p_i p_j} - \cdots\right).
\end{aligned}$$

Note: in the second term in the second to last line, the sum over all $j > 2$ is the same as the sum over all $i > 2$ just with a different name given to the variable.

So we are able to factor out $(1 - 1/p_1)$ to get a similar expression with one fewer primes. Similarly, we can factor out $(1 - 1/p_2)$ and so on until there are no more primes. This gives us Euler's product representation for the totient function.

Part III

Permutations

5 Permutations

A permutation of a finite set X can be thought of in a few ways. The set of all permutations of $\{1, \ldots, n\}$ will be denoted S_n. This is also called *the symmetric group*.

5.1 As a shuffle of the symbols

E.g. 53214 is a shuffle of $X = \{1, 2, 3, 4, 5\}$.

This gives us one way of counting the number of permutations. For the first position in the shuffle, we may write any of the n numbers. For the second position, we have $n - 1$ numbers to choose from—we cannot repeat the first. Likewise the third has $n - 2$ to choose from, not repeating the first or second. In this way, the number of permutations is

$$n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1, \text{ denoted } n!.$$

5.2 As a function from X to itself

E.g. The shuffle 53214 may be thought of as a function $\pi : X \to X$ where $\pi(i)$ equals the i-th number in the shuffle:

$$\pi(1) = 5, \pi(2) = 3, \pi(3) = 2, \pi(4) = 1, \pi(5) = 4.$$

These functions are often represented by a table:

i	1	2	3	4	5
$\pi(i)$	5	3	2	1	4

Something new added by the table/function representation is that we can talk about the inverse function which undoes the shuffle:

$j = \pi(i)$	5	3	2	1	4
$i = \pi^{-1}(j)$	1	2	3	4	5

or reordering the columns:

j	1	2	3	4	5
$\pi^{-1}(j)$	4	3	2	5	1

5.2.1 Composition of functions

Given two permutations, $\pi_1, \pi_2 : X \to X$, we can compose them to get a new function $(\pi_1 \circ \pi_2)$. One way to compute this is via the following procedure:

1. Write the two functions as tables.
2. Reorder the columns of the outermost function in the composition to align with the output of the innermost function.
3. Stack the tables on top of each other.

E.g. Take π_2 to be the function represented by the shuffle 53214 and take π_1 to be represented by the shuffle 13254. So as a table

i	1	2	3	4	5
$\pi_1(i)$	1	3	2	5	4

Now we shuffle the columns so the top is 53214:

i	5	3	2	1	4
$\pi_1(i)$	4	2	3	1	5

Then stack this with π_2:

i	1	2	3	4	5
$j = \pi_2(i)$	5	3	2	1	4
$\pi_1(j)$	4	2	3	1	5

The bottom row is $\pi_1 \circ \pi_2$.

Exercise 5.1. Compute this in the other order: $\pi_2 \circ \pi_1$. *Observe that the order matters.*

Note: in some sources (e.g. books, software), compositions are written as a product in the reverse order. Sometimes there is an option provided to reverse the multiplication order. Here are those computations in the SageMath computer algebra system.

```
sage.combinat.permutation.Permutations.options(mult='r2l')
π1 = Permutation([1,3,2,5,4])
π2 = Permutation([5,3,2,1,4])
π1 * π2
```

```
[4, 2, 3, 1, 5]
```

```
π2 * π1
```

```
[5, 2, 3, 4, 1]
```

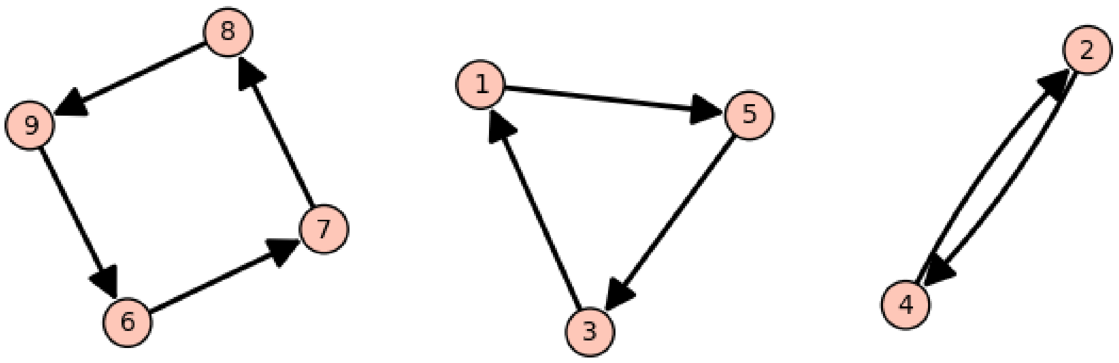
6 Visual representations of permutations

6.1 As a collection of cycles

Let's look at a longer permutation now: 541237896. As in Section 5.2, we can view this as a function where 1 goes to 5, 2 goes to 4 etc. Following the path of a single number we get a cycle: 1 goes to 5 goes to 3 goes to 1. Likewise 2 goes to 4 goes to 2. And so on.

The entire permutation can be broken up into cycles:

```
π = Permutation([5,4,1,2,3,7,8,9,6])
π.to_digraph().plot(vertex_size=1500)
```



6.1.1 Notation

The cycle $1 \to 5 \to 3 \to 1$ can be written as (153). In general, $(a_1, \ldots, a_n)$ represents the cycle $a_1 \to a_2 \to \cdots \to a_n \to a_1$. We often omit the commas when every number is a single digit.

Here is how to compute the cycles in SageMath:

```
π.cycle_string()
```

```
'(1,5,3)(2,4)(6,7,8,9)'
```

The answer is given as a product (i.e. composition) of cycles.

Elements which do not go anywhere—also called *fixed-points*—are represented by cycles of length 1 E.g. (123)(4) represents the two cycles $1 \to 2 \to 3 \to 1$ and $4 \to 4$. We often omit length 1 cycles from the notation so $(123)(4) = (123)$ as a permutation of $1, 2, 3, 4$.

Exercise 6.1. Compute the cycle decomposition for 341859672. You can verify your answer in SageMath.

```
π = Permutation([3,4,1,8,5,9,6,7,2])
π.cycle_string()
```

Note: as mentioned earlier: SageMath omits the cycle (5) representing the fixed point $\pi(5) = 5$.

6.1.2 Multiplying Cycles

To multiply cycles, we read from right to left. If our current number appears in the cycle, we shift over to the right (wrapping back to the start if necessary). E.g. $(2, 5)(1, 5, 3)$ would move 1 to 5 with the first cycle then from 5 to 2 with the second cycle.

You can either do this procedure and produce a shuffle like:

i	1	2	3	4	5
$(1, 5, 3)$	5	-	1	-	3
$(2, 5)$	2	5	-	-	-
Answer	2	5	1	4	3

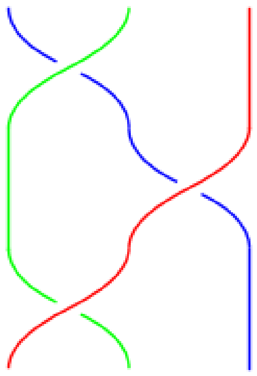
Here a “-” means the value is unchanged at this step.

Or you can combine this with the procedure of Section 6.1 and produce a product in cycle form again. Here that is $(1, 2, 5, 3)$ because $1 \to 5 \to 2$ then $2 \to 5$ then $5 \to 3$ then $3 \to 1$.

6.2 Braids

Another visual representation used frequently in the mathematical study of knots is that of crossing lines. Let’s look at an example:

```
B.<a,b> = BraidGroup(3)
plot(a * b * a)
```



This represents a permutation where the first end ends up in the third position ($1 \to 3$), the second end ends up in the second position ($2 \to 2$) and the third end ends up in the first position ($3 \to 1$). Overall, 1 and 3 are swapped, so this permutation is (13).

6.2.1 Transpositions

Cycles of length 2 are called *transpositions*. E.g. (14) is a transposition which switches 1 and 4. The picture above represents a composition of 3 transpositions $(13) = (12)(23)(12)$.

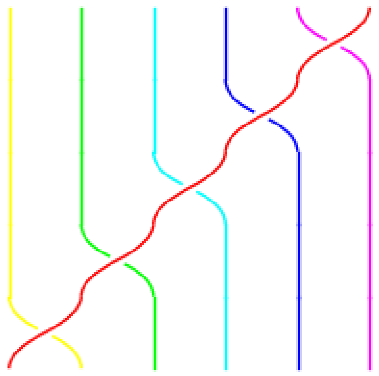
Theorem 6.1 (Product of transpositions)**.**

a) Every permutation can be written as a product of transpositions
b) Every permutation can be written as a product of transpositions of adjacent elements.

Another way to say this is that by swapping pairs of elements at a time, we can obtain any possible shuffle.

Proof. We know that a permutation may be written in terms of cycles. So if we can show that any cycle can be written as a successive sequence of swaps we are good. We will give a visual demonstration of this fact:

```
B.<t1,t2,t3,t4,t5> = BraidGroup(6)
plot(t1 * t2 * t3 * t4 * t5)
```



So the cycle $(123456) = (12)(23)(34)(45)(56)$. Remember that compositions are read from right to left: e.g. $(f \circ g)(i) = f(g(i))$ means first do g then do f.

This kind of decomposition generalizes: you can replace (123456) with any cycle of any length. E.g. $(1456) = (14)(45)(56)$:

```
sage.combinat.permutation.Permutations.options(mult='r2l', display='cycle')
P = Permutation
P((1,4)) * P((4,5)) * P((5,6))
```

```
(1,4,5,6)
```

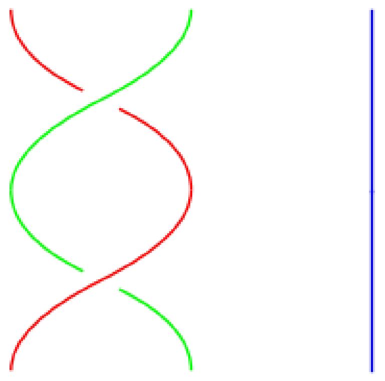
$\square$

For an alternative proof, this decomposition into a sequence of adjacent swaps is exactly how the sorting algorithm BubbleSort works. We can sort any list using BubbleSort which does only adjacent swaps. So the shuffle is obtained by reversing those swaps to go from sorted to shuffled.

6.2.2 Braids versus permutations

You may have noticed in what we did, the word “braid” was used. Braids are similar to permutation except that we keep track of which strand goes above and which strand goes below. E.g. the transposition (12) applied twice looks like

```
B.<a,b> = BraidGroup(3)
plot(a * a)
```



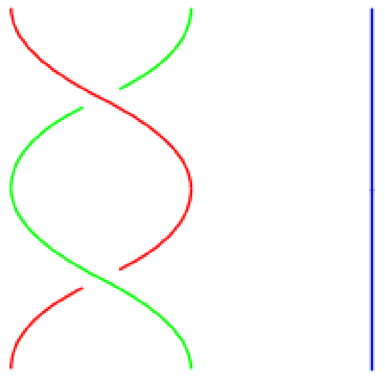
As a permutation this is the trivial shuffle 123. But as a braid it is still twisted.

While we are only focused on permutations rather than braids here, we still make use of braids because SageMath is able to create diagrams for us.

6.2.2.1 The Braid Group in SageMath

The generators of the braid group are adjacent swaps. So the line `B.<a, b> = BraidGroup(3)` sets `a` to a swap of $1, 2$ and sets `b` to a swap of $2, 3$. We can obtain the reverse swap with `a^-1` or `b^-1`

```
plot(a^-1 * a^-1)
```



When converting from braids to permutations, we ignore whether a strand goes over or under and just focus on the swapping.

7 Generators

In Theorem 6.1 we saw that every cycle and hence every permutation can be written as a product of adjacent swaps: $(12), (23), (34), \ldots, (n-1, n)$. Here we will identify some relations that hold among these generating elements.

Let us call $\tau_i = (i, i+1)$ the swap of i and $i+1$.

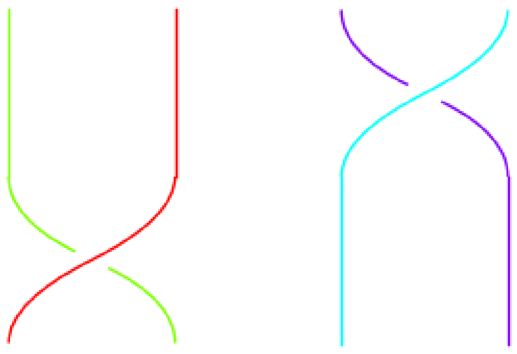
7.1 Squaring relation

We have $\tau_i^2 = 1$ where 1 represents the identity permutation (no shuffling). This says that if we swap i and $i+1$ and then swap again, we get back to a sorted list.

7.2 Commutating

From Exercise 5.1, we saw that in general $\pi_1\pi_2 \neq \pi_2\pi_1$. Nonetheless, if we are swapping disjoint sets of pairs like (12) followed by (34) then there is no interaction between the swaps. So the order doesn't matter: $(12)(34) = (34)(12)$. Specifically, τ_i and τ_j commute ($\tau_i\tau_j = \tau_j\tau_i$) provided i and j are at least 2 apart.

```
B.<a,b,c> = BraidGroup(4)
plot(a * c)
```



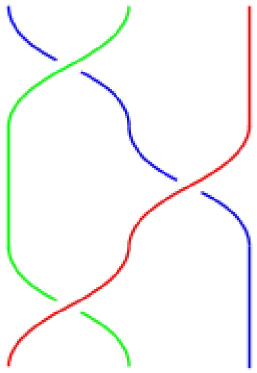
```
a * c == c * a
```

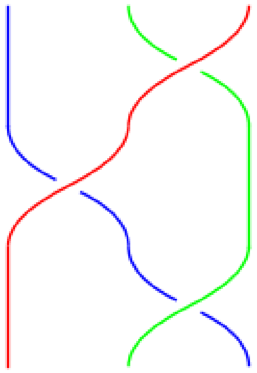
```
True
```

7.3 ABA = BAB

At the top of Section 6.2 we saw that (12)(23)(12) = (13). We also have (23)(12)(23) = (13). Compare the following pictures.

```
B.<a,b> = BraidGroup(3)
plot(a * b * a)
plot(b * a * b)
```





We can see visually that the middle green strand is sliding from one side of the blue/red crossing to the other.

7.4 Other relations?

It turns out, every way to simplify or manipulate products of transpositions can be reduced to exactly these three rules:

1. $\tau_i^2 = 1$
2. $\tau_i\tau_j = \tau_j\tau_i$ for $|i - j| \geq 2$ (at least two apart)
3. $\tau_i\tau_{i+1}\tau_i = \tau_{i+1}\tau_i\tau_{i+1}$

The proof of this has two stages. One which we have already seen. First, you show that every permutation can be written as a product of transpositions of adjacent elements (Theorem 6.1). This shows that you can use $\tau_1, \ldots, \tau_n$ and these rules to write every element of S_n. I.e. the number of objects generated by these rules is at least $n!$.

The second step is showing that the number of objects generated by these rules is no more than $n!$ so that it is exactly the same as S_n. This requires more tools than we have available to us (i.e. group theory). A proof for those in-the-know may be found here for instance.

We will take it as a fact that these rules describe S_n exactly.

8 Sign of a Permutation

We established in Theorem 6.1 that every permutation can be written as a product of transpositions. But more than that, in all our examples, the number of transpositions required to perform a shuffle was always consistently odd or consistently even. For instance, $(13) = (12)(23)(12)$ has an odd number of transpositions on both sides.

If we look at Section 7.4, we said that every possible way to manipulate a permutation can be reduced to three rules, each of which preserves the parity (odd or even) of the number of transpositions.

In case you were unsatisfied with relying on the fact that every relation reduces to the three given relations, we will present two different proofs that the parity is always consistent: one here, one in the next section.

For a visualization of this proof, see https://www.youtube.com/watch?v=p6kCYbKIMak starting at 13:50.

8.1 Step 1: reducing to the identity

Suppose we have two different ways to write a given permutation as a product of transpositions:

$$\pi = \tau_1\tau_2\ldots\tau_m = \tau_1'\tau_2'\ldots\tau_n'.$$

(Here the letter τ means any transposition, not just for adjacent elements.)

Our goal is to show that m and n have the same parity (both are odd or both are even). Notice that if we multiply both sides by τ_1, we get

$$\tau_1\cdot\tau_1\tau_2\ldots\tau_m = \tau_1\cdot\tau_1'\tau_2'\ldots\tau_n'$$

but $\tau_1\cdot\tau_1$ means swap the same pair twice, and doing this twice is the same as not doing it at all. Therefore the $\tau_1\cdot\tau_1$ cancels and we are left with

$$\tau_2\ldots\tau_m = \tau_1\cdot\tau_1'\tau_2'\ldots\tau_n'.$$

Keep doing this from left to right until we have

$$1 = \tau_m \dots \tau_2 \tau_1 \tau_1' \tau_2' \dots \tau_n'.$$

So saying "m and n have the same parity" is equivalent to saying "1 cannot be written as an odd number of transpositions."

8.2 Step 2: setting up the induction hypothesis

Let's introduce some letters now for our transpositions. Say

$$1 = (a_1 b_1)(a_2 b_2) \cdots (a_k b_k) \tag{8.1}$$

where $a_i \neq b_i$ for all i (we're always swapping two different things).

We know that k isn't 1 since a single transposition is not the same as the identity. So look now at $k \geq 2$. We will show that we can always rewrite this product using $k - 2$ transpositions.

There are a couple ways to phrase this next part of the proof. First, we could say: take as an inductive hypothesis that for all $k' < k$ if the identity is written as a product of k' transpositions then k' is even. In this way, going from k to $k' = k - 2$ we know that $k - 2$ is even and hence so is k.

Another way we could say this: we are always reducing by 2 every time so $k - 2 - 2 - 2 - \dots$ will either end up at 0 or 1 and we know it can't end up at 1.

8.3 Step 3: reorganizing the product

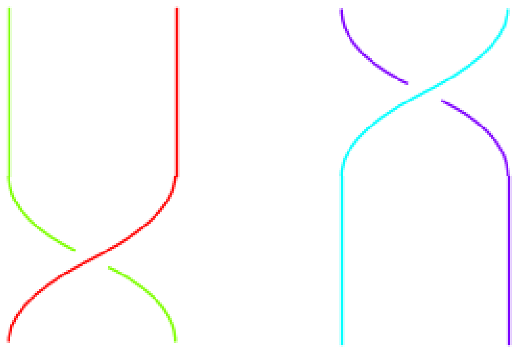
Look back at Equation 8.1. Since $(ab) = (ba)$, to keep things consistent, we will assume that $a_i < b_i$ always. Let a be the smallest number appearing in any transposition. E.g. for $(23)(34)(23)(24)$ we have $a = 2$.

We will slide all the the transpositions with an a in them to the right (towards the beginning of the composition). Now we can't just move things without changing the product so we have to be strategic.

Let's say we have in the middle of our expression $(uv)(xy)$ where $u = a$ and $x \neq a$ and so we want to move (uv) to the right of (xy).

8.3.1 Case 1: disjoint

If u, v, x, y are all distinct, then the two swaps can be done in either order and $(uv)(xy) = (xy)(uv)$.

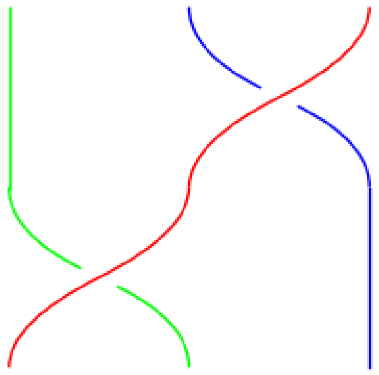


8.3.2 Case 2: two numbers in common

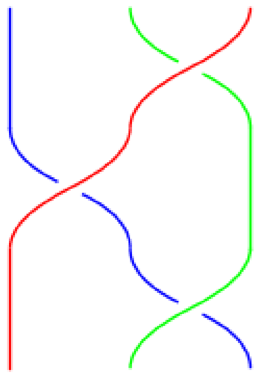
If $(uv) = (xy)$ then $(uv)(xy) = (uv)^2 = 1$ and we have reduced the number of transpositions by 2 as we said we would.

8.3.3 Case 3: larger number in common

Consider the product $(12)(23) = (123)$ (swap $2, 3$ first then $1, 2$)



Compare this with the identity $(23)(12)(23) = (13)$ that we worked out in Chapter 7:



Multiply both sides on the left by (23) and simplify using $(23)(23) = 1$ to get $(12)(23) = (23)(13)$.

In this way, the smallest number is always moving to the right. The rule is $(uv)(xy) = (xy)(uv')$ where v' is the number in (xy) which wasn't in common.

Exercise 8.1. Work out the diagrams to show that $(12)(23) = (23)(13)$. (I would show you in SageMath, but the software doesn't draw (13) very well.)

8.3.4 Case 4: smaller number in common

Using the previous case, we move all the terms having $u = a$ to the right. So now we have a bunch of terms $(av_1)(av_2)\dots(av_r)$ in the right of our product. We know we have at least two pairs because if we swap a out of position, something later on has to swap it back into position.

Also, if any of the adjacent pairs are equal, we can cancel them as in Case 2. Otherwise, we do something similar to Case 3 where

$$(12)(23)(12) = (13) \implies (12)(23) = (13)(12)$$

by multiplying on the right by (12)

The feature of this identity is that we have one fewer 1's, or more generally, one fewer a's. And we can keep doing this until we are left with either a single a (impossible) or we eventually find a pair that cancels.

Exercise 8.2. Draw the diagram for this identity.

8.4 Summary

We showed that we can push all the transpositions containing a 1 to the right and then moving those transpositions past each other until we had fewer and fewer 1's and eventually there must be a pair with both numbers in common because we can't just move 1 a single time in our sequence of transpositions.

Example 8.1. Start with (12)(23)(12)(13). Use Case 3 to swap the first and second: (23)(13)(12)(13). Now use Case 4 to swap the second and third: (23)(23)(13)(13). Now cancel using Case 2.

8.5 Notation/Computation

If π is a permutation, common notations for its parity or sign are: $\operatorname{sgn}\pi$ and $(-1)^\pi$. We say the sign is $+1$ if it is an even length product and the sign is -1 if it is an odd length product.

We have the identity $\operatorname{sgn}(\pi_1\pi_2) = \operatorname{sgn}\pi_1 \operatorname{sgn}\pi_2$ because, for example, multiplying two odd length products creates an even length product $((-1)\cdot(-1) = +1)$

One way to compute this is by decomposing π into cycles. We saw in Theorem 6.1 how to write cycles in terms of transpositions: $(12345) = (12)(23)(34)(45)$. The observation here is that odd length cycles become an even length product of transpositions and vice versa.

So the algorithm is:

1. Convert the permutation into a product of cycles.
2. Write a $+1$ if the cycle has an odd length and a -1 if it has an even length.
3. Multiply those numbers together to find the sign.

Example 8.2. Consider the shuffle 376819254. We can write this as (1369485)(27) this is a product of a cycle of length 7 and a cycle of length 2. Therefore the sign is $(+1)(-1) = -1$.

This computation in SageMath:

```
π = Permutation([3,7,6,8,1,9,2,5,4])
sign(π)
```

```
-1
```

9 Inversions

We present a second (or third depending if you count the proof we didn't prove) proof that the number of transpositions is always ever even or always odd.

Let π be a permutation. An **inversion** for π is a pair of outputs which are not in sorted order. I.e. we have $i < j$ to begin with and $\pi(i) > \pi(j)$ to end with.

In counting inversions, we look at where $\pi(i)$ is greater than $\pi(j)$ for some j that comes after i.

Example 9.1. Consider the permutation represented by 31254.

We have $\pi(1) = 3$ and this is greater than both $\pi(2) = 1, \pi(3) = 2$.

We have $\pi(2) = 1$ but this is not greater than anything that comes after.

We have $\pi(3) = 2$ but this is not greater than anything that comes after.

We have $\pi(4) = 5$ which is greater than $\pi(5) = 4$.

So this permutation has 3 inversions: $(1 < 2) \to (\pi(1) > \pi(2))$ and $(1 < 3) \to (\pi(1) > \pi(3))$ and $(4 < 5) \to (\pi(4) > \pi(5))$.

We can use SageMath to check our work:

```
π = Permutation([3,1,2,5,4])
π.inversions()
```

```
[(1, 2), (1, 3), (4, 5)]
```

9.1 Parity of the number of inversions

Inversions as a concept have a few uses in combinatorics. Relevant to us now is that the number of inversions has the same parity as the permutation. To show this, we will consider how a single transposition affects this parity. But since the total number can go up or down, let us define a quantity which we can analyze to describe the parity.

For the identity permutation, define

$$V(1) = \prod_{i<j}(j - i)$$

E.g. for $n = 4$ this is $(2-1)(3-1)(4-1)(3-2)(4-2)(4-3)$. We're not interested in the absolute value here but rather the sign—which for the identity permutation is $+1$.

More generally, define

$$V(\pi) = \prod_{i<j}(\pi(j) - \pi(i)).$$

Note that we will have a factor of $V(\pi)$ which is negative whenever $i < j$ and $\pi(i) > \pi(j)$. So the sign of $V(\pi)$ tells us the parity of the number of inversions.

What's useful here is that factoring out the various -1's and reordering, we can write $V(\pi) = \pm V(1)$ where it is a $+1$ if we have an even number of inversions and a -1 if we have an odd number.

9.2 Analysis

We want to show that a single transposition changes $V(1)$ to $-V(1)$. Then a sequence of odd length will have a sign of -1 and one of even length will have a sign of $+1$, matching what we did in Chapter 8.

Suppose we apply the transposition (ij) with $i < j$. Right away, we get one inversion because now $\pi(i) > \pi(j)$. Let's look at the other numbers. Suppose $x < i < j$. Then after swapping, we still have x on the left of i, j\$ so no inversions here. Likewise, if $i < j < x$ then x will still be on the right afterwards.

The last case is that $i < x < j$. Then we get two inversions going from i, x, j to j, x, i. One between i and x and one between x and j since both these pairs are now out of order.

To summarize, this single transposition gives us:

- 0 inversions for anything left of i or right of j,
- 2 inversions for each number in between,
- 1 inversion for the pair i, j

So overall we get an odd number of inversions for each swap. This shows that the number of inversions is odd if and only if the number of swaps is odd.

10 Exercises

1. Convert $2, 12, 3, 5, 4, 9, 8, 7, 6, 10, 1, 11$ to a product of cycles and draw the associated digraph.

Hint

Reference: Section 6.1

- start with 1
  - there is a 2 in position 1
  - there is a 12 in position 2
  - keep going... x in position 12... until you get back to 1
- write down the first cycle $(1, 2, 12, \dots)$ (don't repeat the 1 at the end)
- do the same procedure with the next number which hasn't been seen so far (i.e. 3)
- keep going until every number is part of a cycle
- optionally, throw away cycles of length 1

2. Convert $(1, 8, 12)(2, 3, 6, 7, 9)(4, 10, 11)$ to a shuffle of $1, \dots, 12$

Hint

Reference: Section 6.1.2
Since the cycles are disjoint (don't share any numbers), the procedure here is:

- look for each number from 1 to 12
- write down the number following it in its cycle (e.g. 1 is followed by 8 so we write an 8 in the first position of the shuffle, 2 is followed by 3, 3 by 6, etc.)
- if a number does not appear (i.e. 5), then the permutation leaves it unchanged and it remains in its starting position (so you should have a 5 in position 5 at the end)

3. Show that $(12)(23)(34)(23)(12) = (14)$ by drawing the braid diagram.

4. Compute the sign of the permutations in 1. and 2.

Problem 1: ______________ Problem 2: ______________

Hint

Reference: Section 8.5
Regardless of whether the cycles are disjoint or not (here they are), odd cycles are even and even cycles are odd (in terms of number of swaps). So multiply 1 for each odd length cycle and -1 for each even length cycle. For example, $(1, 8, 2)$ has an odd length (even number of swaps) so we multiply $+1$ for it.

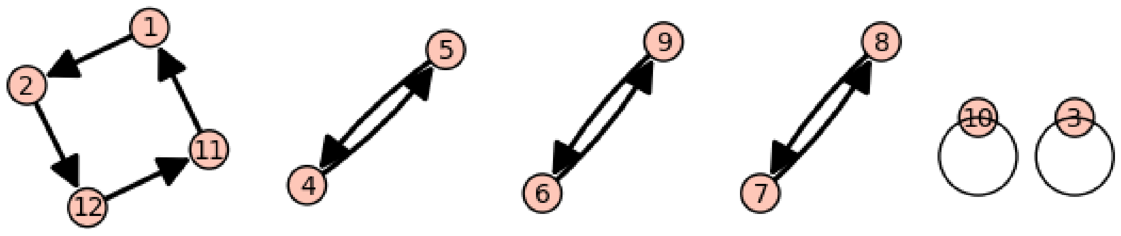
5. Write (182635) as a product of transpositions.

Hint

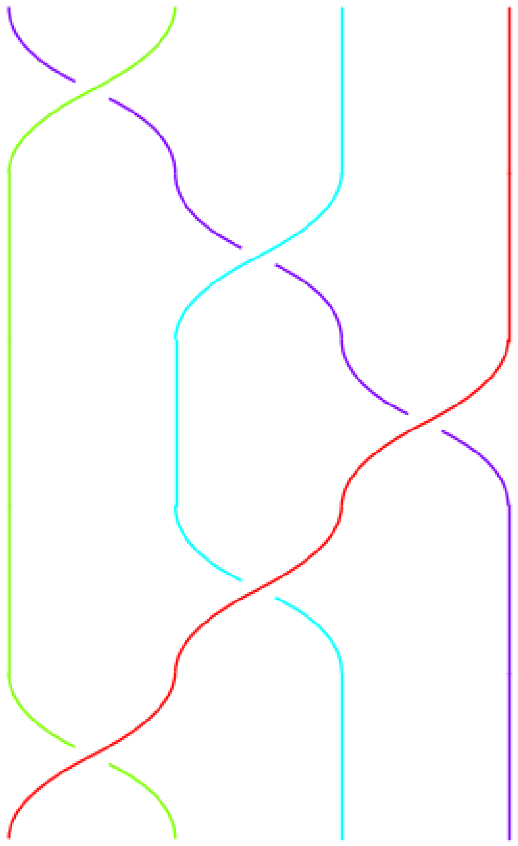
Reference: Theorem 6.1
The rule is like $(a, b, c, d) = (a, b)(b, c)(c, d)$. The transpositions are adjacent pairs.

Solutions

1. `(1, 2, 12, 11)(4, 5)(6, 9)(7, 8)`



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



3.
4. both even (+1)
5. `(1, 8)(8, 2)(2, 6)(6, 3)(3, 5)`

Part IV

Generating Functions

11 Weight Functions

11.1 Recap: Strings

Recall that an **alphabet** is a set of characters like $\{0,1\}$ is the alphabet for binary. A **string** is an ordered sequence of characters from a fixed alphabet. E.g. 011011 and 00011 are binary strings.

Additionally, ε will denote the empty string and multiplication of strings will be done by concatenation. E.g. $011 \cdot 000 = 011000$.

For most examples that follow, we will work with binary strings of 0s and 1s unless otherwise specified.

11.2 Definition

Let S be a set of strings (e.g. all binary strings or all ternary strings). A **weight function** on S is a function $f : S \to \mathbf{Z}$ satisfying:

- $f(x) \geq 0$
- $f(uv) = f(u) + f(v)$ when u and v are two strings.

Example 11.1. The length l of a string is a weight. E.g. $l(011) = 3, l(011 \cdot 10011) = l(011) + l(10011)$.

Example 11.2. $f(s) =$ the number of 1s in s is a weight. E.g. $f(011) = 2$ and $f(011 \cdot 10011) = f(011) + l(10011)$.

11.2.1 Key property

The definition of a weight means they behave kind of like a logarithm. Therefore, if we do $x^{f(s)}$, we get a function which satisfies

$$x^{f(uv)} = x^{f(u)} \cdot x^{f(v)}.$$

This is why we are going to have a bunch of xs all over the next few sections.

12 Generating Functions

12.1 Definition

Let L be a subset of strings. These is also called a **language**. Let f be a weight function. We define the **(ordinary) generating function** of L or **OGF** as

$$\Phi_L(x) = \Phi_L^f(x) = \sum_{s \in L} x^{f(s)}.$$

ℹ Note

It is common in mathematical definitions to write something like

$$\Phi_L(x) = \Phi_L^f(x) = \cdots .$$

This means that I am calling attention to the fact that Φ depends on f but also saying that because f isn't going to change much, we are going to drop the f from the notation most or all of the time.
It is also common to call generating functions by other letters, like $A(x)$ but this needs to be introduced first by saying that A is the OGF wheras Φ_L means this implicitly.

Example 12.1. Let L be the subset $\{01, 1110, 11, 0, 1, 111000, \varepsilon\}$ of binary strings. Let f be the length weight. The OGF of L is

$$\begin{aligned}
\Phi_L(x) &= x^{f(01)} + x^{f(1110)} + x^{f(11)} + x^{f(0)} + x^{f(1)} + x^{f(111000)} + x^{f(\varepsilon)} \\
&= x^2 + x^4 + x^2 + x^1 + x^1 + x^6 + x^0 \\
&= 1 + 2x + 2x^2 + x^4 + x^6
\end{aligned}$$

12.2 Monomials

Inside $\Phi_L(x)$, each monomial $a_n x^n$ (after collecting terms) represents that L has a_n strings where f equals n (e.g. the length equals n).

Example 12.2. For the language L of Example 12.1 above, we have for instance, 2 strings of legnth 1, represented by the monomial $2x^1$. We have 1 string of length 6 represented by the monomial $1x^6$. We have no strings of length 5, represented by the lack of a monomial with x^5. We could also say that the coefficient of x^5 is 0.

Exercise 12.1. Let L be the set of all binary strings of length ≤ 5 which have no 11.

a) for $n = 0, 1, 2, 3, 4, 5$, list all the strings with no 11
b) write down the generating function with respect to the length weight
c) write down the generating function with respect to the "number of 1s" weight

12.2.1 Using generating functions to count

The monomials $a_n x^n$ are created by adding up all the words in L of weight n. So another way to describe the generating function is

$$\Phi_L(x) = \sum_{s \in L} x^{f(s)} = \sum_{n=0}^{\infty} \#\{s \in L : f(s) = n\} x^n. \tag{12.1}$$

This means the general strategy for getting an answer from a generating function is to—by some process—write it as as a power series $\sum a_n x^n$ making sure that it's x^n. Then a_n is the answer to the question: how many strings of weight n are there where (*whatever conditions we set*).

Theorem 12.1 (Addition Theorem). *If L and L' are disjoint, then $\Phi_{L \cup L'}(x) = \Phi_L(x) + \Phi_{L'}(x)$.*

Proof. To say that L and L' are disjoint implies that

$$\#\{s \in L \cup L' : f(s) = n\} = \#\{s \in L : f(s) = n\} + \#\{s \in L' : f(s) = n\}.$$

Substituting this into Equation 12.1 yields

$$\begin{aligned} \Phi_{L \cup L'}(x) &= \sum_{n=0}^{\infty} \#\{s \in L : f(s) = n\} x^n + \sum_{n=0}^{\infty} \#\{s \in L' : f(s) = n\} x^n \\ &= \Phi_L(x) + \Phi_{L'}(x). \end{aligned}$$

□

12.3 Infinite series

We've looked at examples where L is a finite set. In many examples, L will be an infinite set. For instance consider the infinite set of all strings of 0s. I.e. $\{\varepsilon, 0, 00, 000, 0000, \dots\}$. For this language, we have the generating function

$$A = 1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x}.$$

Geometric series formula

To prove this formula, we take A and multiply everything by x because that creates a lot of similar terms:

$$\begin{aligned} A &= 1 + x + x^2 + x^3 + \cdots \\ xA &= \phantom{1 + {}} x + x^2 + x^3 + \cdots \\ A - xA &= 1 \end{aligned}$$

From which we solve for $A = \dfrac{1}{1-x}$.

Example 12.3. Let L is the language of all binary strings, i.e. $L = \{\varnothing, 0, 1, 00, 01, 10, 11, 000, \dots\}$. We know that there are 2^n binary strings of length n so the OGF of L is

$$\Phi_L(x) = \sum_{n=0}^{\infty} 2^n x^n = \sum_{n=0}^{\infty} (2x)^n = \frac{1}{1-2x}.$$

Note

In a calculus or analysis course, we would want to specify here that $|x| < \frac{1}{2}$ for this series to converge. In combinatorics, we treat these series *purely algebraically*. That means as an infinite list of monomials without care whether or not the series converges. People use the term **formal power series** to refer to this concept.

12.3.1 When do we care about convergence?

For the most part, *we don't*. Meaning an infinite series is still valid even if it doesn't converge other than when $x = 0$. However, there are arguements using analysis which only apply to convergent power series. For instance, the fact that the series

$$1 + x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + \cdots = \frac{1}{1 - x - x^2}$$

converges for $|x| < \frac{1}{\phi}$ where $\varphi = \frac{1+\sqrt{5}}{2}$ is connected to the fact that the Fibonnaci numbers are approximately $c\varphi^n$ for some constant c.

Exercise 12.2. In fact, one can show that

$$\mathrm{Fib}_n = \mathrm{round}\left(\frac{\varphi^n}{\sqrt{5}}\right).$$

Use a calculator or a computer to check this works!

13 Multiplication of Generating Functions

Suppose L, L' are two languages. We define the concatenated language LL' by concatenating a string from L and a string from L' in all possible ways:

$$LL' = \{uv : u \in L, v \in L'\}.$$

The utility of this is that many common languages of strings can be written as concatenations of smaller, more simpler languages. And, it will turn out, we can compute the OGF of the more complex language out of the OGFs for the simpler languages.

13.1 Multiplication of Monomials

Suppose we have a language L which consists of 10 strings, each of length 5. Suppose we also have a language L' consisting of 6 strings, each of length 7. Concatenating any pair of strings will always yield a string of length 12 and there are 60 such pairs. This computation is represented by the following multiplication of generating functions:

$$(10x^5) \cdot (6x^7) = 60x^{12}.$$

13.2 A step further

Now suppose L is the same as before: 10 strings of length 5, and this time L' consists of 3 strings of length 4 and 7 strings of length 8. Now if we concatenate to form LL' we are either combining a string of length 5 with one of length 4 (in $10 \cdot 3$ ways) or a string of length 5 with one of length 8 (in $10 \cdot 7$ ways). Again, this computation is reflected in the product of polynomials

$$10x^5 \cdot (3x^4 + 7x^8) = 30x^9 + 70x^{13}.$$

Exercise 13.1. Suppose L consists of 3 strings of length 4 and 8 of length 6. Suppose L' consists of 2 strings of length 1 and 5 strings of length 4. What will LL' consist of?

13.3 Multiplication theorem

Theorem 13.1 (Multiplication Theorem)**.** *Let L and L' be two different languages. Let f be a common weight function like the length of a string. Then*

$$\Phi_{LL'}(x) = \Phi_L(x)\Phi_{L'}(x).$$

Proof. We have

$$\begin{aligned}\Phi_{LL'}(x) &= \sum_{w\in LL'} x^{f(w)} \\ &= \sum_{u\in L}\sum_{v\in L'} x^{f(uv)} \\ &= \sum_{u\in L}\sum_{v\in L'} x^{f(u)}x^{f(v)}\end{aligned}$$

using Section 11.2.1. Continuing, we get

$$\begin{aligned}&= \sum_{u\in L} x^{f(u)} \sum_{v\in L'} x^{f(v)} \\ &= \Phi_L(x)\Phi_{L'}(x).\end{aligned}$$

□

Tip

One might wonder whether $\sum a_i \sum b_j$ means $(\sum a_i)(\sum b_j)$ or $\sum(a_i \sum b_j)$. It turns out these are the same, and it is explained by the distributive property of multiplication. For example,

$$(a_1 + a_2 + a_3)(b_1 + b_2) = a_1(b_1 + b_2) + a_2(b_1 + b_2) + a_3(b_1 + b_2).$$

13.4 Strings with 0s followed by 1s

Let L be the language of strings with only 0s and let L' be the language of strings with only 1s. In Section 12.3, we saw that these languages have generating functions $\frac{1}{1-x}$ (weighting by length). By theorem Theorem 13.1, the language LL' of all strings of 0s followed by 1s has generating function

$$\frac{1}{(1-x)^2}.$$

We can compute this in a couple ways. First, we ask the question: "how many strings are there of length n consisting of a string of 0s followed by a string of 1s?" E.g. for $n = 4$ there are 5 such strings:

$$0000, 0001, 0011, 0111, 1111.$$

We can look at this as: there are either $0, 1, 2, 3, 4$ ones. In general, there will be $n + 1$ strings of length n (corresponding to the monomial $(n + 1)x^n$). Therefore

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty}(n+1)x^n = 1 + 2x + 3x^2 + 4x^3 + \cdots .$$

13.4.1 Calculus?

The monomials we've just computed look suspiciously like derivatives: 1 is the derivative of x, $2x$ is the derivative of x^2, $3x^2$ is the derivative of x^3, and so on. I.e., we have

$$\frac{1}{(1-x)^2} = \sum_{n=0}^{\infty} \frac{d}{dx} x^n.$$

Tip

The sums $\sum nx^{n-1}$ and $\sum(n + 1)x^n$ are equal. In words: each sum is adding up all monomials whose coefficient is exactly 1 more than the exponent. Algebraically, we can transform the first sum to the second by letting $m = n - 1$ and $n = m + 1$. Then

$$\sum nx^{n-1} = \sum(m + 1)x^m.$$

When working with OGFs, we often want to make such tranformations so that the question of "how many strings of length n" can be read directly from the summand $a_n x^n$: there are a_n strings of length n.

In our calculus courses, we learn that we can often take the derivative of a series term-by-term. Or in other words, we can safely move the derivative operator from inside the term to outside and vice versa:

$$\sum_{n=0}^{\infty} \frac{d}{dx} x^n = \frac{d}{dx} \sum_{n=0}^{\infty}\sum_{n=0}^{\infty} x^n.$$

This we recognize as the geometric series from Section 12.3. So that means we should expect

$$\frac{1}{(1-x)^2} = \frac{d}{dx}\frac{1}{1-x}.$$

Indeed, that is the case.

The following is a result of analysis that is very potent for many OGF problems.

Theorem 13.2. *If $\sum a_n x^n$ converges to a function $f(x)$ (and the radius of convergence is not zero), then*

$$\sum \frac{d}{dx} a_n x^n = f'(x)$$

(and the radius of convergence is preserved).

In Section 12.3.1, we said that these generating functions are still valid even when they don't converge. But there is utility here in that if we have an expression to represent an infinite sum like how $\frac{1}{1-x}$ represents the geometric series, then we can compute the derivative of the expression and of the infinite series.

Example 13.1. Given that

$$\frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + \cdots = \sum nx^{n-1},$$

we can compute the derivative on both sides to obtain

$$\frac{2}{(1-x)^3} = 2 + 6x + 12x^2 + 20x^3 + \cdots = \sum n(n-1)x^{n-2}.$$

If we divide by 2 we might recognize this as

$$\frac{1}{(1-x)^3} = \sum \frac{n(n-1)}{2} x^{n-2} = \sum \binom{n}{2} x^{n-2} = \sum \binom{n+2}{2} x^n.$$

14 Stars and Bars Revisited

Suppose we have 2 people and n identical objects to distribute. Let's say for now that a person gets 0 or more objects but all n objects are distributed. An example arrangement we might create with stars and bars would look like

$$* * * * * \mid * * *.$$

The key insight is that if we change this to

$$0\,0\,0\,0\,0 \mid 1\,1\,1.$$

Now it looks like the problem we studied in Section 13.4. The bar here is no longer necessary to know where the first group ends and the second begins.

Now we can check that our stars-and-bars formula agrees with the formula computed in Section 13.4. Indeed, that formula is

$$\binom{n+1}{1} = n+1.$$

14.1 From 2 people to k people

Let's copy the key insight for the $k = 2$ case. A distribution of n objects to k people, with each receiving 0 or more, looks like a string

$$1 \cdots 1 2 \cdots 2 3 \cdots 3 \cdots \cdots k \cdots k.$$

Here the number of is in the string is the number of objects person i receives. E.g.

$$1122223555$$

means person 1 receives 2 objects, person 2 receives 4, person 3 receives 1, person 5 receives 3, and everyone else (including person 4) receives none.

14.1.1 Applying the multiplication theorem

Let L_i be the language of strings consisting of 0 or more of the character i. Then the stars-and-bars language for k people, is $L_1 L_2 \cdots L_k$. We also know from Section 12.3 that each of these language has a generating function of $\frac{1}{1-x}$. It follows that our stars-and-bars language has a generating function of

$$\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n.$$

We know these are the coefficients because we know there are $\binom{n+k-1}{k-1}$ stars-and-bars strings of length n (i.e. that many ways to distribute the n objects among k people).

Exercise 14.1. Use proof by induction to show that

$$\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n.$$

Use the derivative to go from case k to case $k+1$. Take the geometric series as the base case ($k = 1$).

Hint: the derivative is going to reduce the exponent from x^n to x^{n-1} so to get a step ahead of that, we can add 1 everywhere we see n:

$$\frac{1}{(1-x)^k} = \sum_{n+1=0}^{\infty} \binom{n+k}{k-1} x^{n+1}.$$

14.2 Variations of Stars and Bars

How do do the stars and bars problem where every person gets at least 1 object? Let's look back at our work:

1. Broke up the problem into a concatenation $L_1 L_2 \cdots L_k$
2. $L_i = \{\varepsilon, i, ii, iii, \dots\}$
3. The generating function of L_i is $\frac{1}{1-x}$
4. Use the multiplication principle to conclude that the generating function of $L_1 L_2 \cdots L_k$ is $\frac{1}{(1-x)^k}$

The second step is where we need to start making changes; if each person gets at least 1 object, then L_i should instead be $\{i, ii, iii, \dots\}$ with generating function

$$x^{f(i)} + x^{f(ii)} + x^{f(iii)} + \cdots = x + x^2 + x^3 + \cdots = \frac{x}{1-x}.$$

General Geometric Series

In general, a geometric series will have some initial term a and then each succesive term is multiplied by a common ratio r. We can factor out the initial term a to evaluate:

$$a + ar + ar^2 + ar^3 + \cdots = a(1 + r + r^2 + r^3 + \cdots) = \frac{a}{1-r}.$$

Mnemonic: "initial term over 1 minus the common ratio." E.g. $3x+6x^3+12x^5+24x^7+\cdots$ has an inital term of $3x$ and a common ratio of $2x^2$. So this evaluates to $\frac{3x}{1-2x^2}$.

Substituting this new generating function into our previous steps, we find that the stars-and-bars generating function with everyone getting at least 1 object is

$$\left(\frac{x}{1-x}\right)^k = \frac{x^k}{(1-x)^k} = x^k \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^{n+k}.$$

In order to get the number of solutions with m objects, we need the coefficient of x^m, so we need $n + k = m$. Changing $n + k$ to m, we get

$$\frac{x^k}{(1-x)^k} = \sum_{m=k}^{\infty} \binom{m-1}{k-1} x^m.$$

This matches our result of $\binom{m-1}{k-1}$ ways to distribute m objects to k people, with each person receiving at least 1.

14.2.1 Other restrictions

Let's take the problem

$$x_1 + x_2 + \cdots + x_k = n,$$

where $x_i \geq 1$. Now consider a restriction like $5 \leq x_1 \leq 13$. This changes the language L_1 to $\{11111, 111111, \cdots, 1111111111111\}$ and the generating function of L_1 to

$$x^5 + x^6 + \cdots + x^{13} = \frac{x^5 - x^{14}}{1-x}.$$

Finite Geometric Sums

Suppose we have a finite geometric sum with initial term a, common ratio r and proceeding for some number of steps. Then using the same method as Section 12.3, we can compute the sum

$$\begin{aligned} S &= a + ar + ar^2 + \cdots + ar^m \\ rS &= \quad ar + ar^2 + \cdots + ar^m + ar^{m+1} \\ S - rS &= a \qquad\qquad\qquad\qquad - ar^{m+1}. \end{aligned}$$

This yields the formula $S = \frac{a - ar^{m+1}}{1-r}$. Mnemonic: take the initial term and subtract from it the term which would come after the last term (first minus next) over $1 - r$.

Combining this with the generating functions $\frac{x}{1-x}$ for $L_2, L_3, \ldots, L_k$, our restricted problem has an overal generating function of

$$\frac{x^5 - x^{14}}{1-x} \cdot \left(\frac{x}{1-x}\right)^{k-1} = \frac{x^{k-1}(x^5 - x^{14})}{(1-x)^k}.$$

We will cover how to read the number of solutions from such a generating function in the next section. For now, let's address one last modification.

General Method

Let L_1 represent the possible number of objects person 1 can receive. E.g.

- if person 1 receives 0 or more, then $L_1 = \{\varepsilon, 1, 11, 111, \ldots\}$—with OGF $x^0 + x^1 + x^2 + \cdots = \frac{1}{1-x}$
- if person 1 receives 1 or more, then $L_1 = \{1, 11, 111, \ldots\}$—with OGF $x^1 + x^2 + x^3 + \cdots = \frac{x}{1-x}$
- if person 1 receives $0, 1, 2$, then $L_1 = \{\varepsilon, 1, 11\}$—with OGF $x^0 + x^1 + x^2$
- if person 1 receives an even number, then $L_1 = \{\varepsilon, 11, 1111, \ldots\}$—with OGF $x^0 + x^2 + x^4 + \cdots = \frac{1}{1-x^2}$.

Do this for each person and then multiply the OGFs together for the full problem.

14.3 Slack variables

Consider the problem

$$x_1 + x_2 + \cdots + x_k \leq n$$

with $x_i \geq 0$.

The method for solving this is to add a slack variable y with $y \geq 0$. This adds to our proceedure a $k+1$-st language L_y. This gives us the generating function

$$\frac{1}{(1-x)^{k+1}} = \sum_{n=0}^{\infty} \binom{n+k}{k} x^n$$

Indeed, the coefficient $\binom{n+k}{k}$ matches our previous solution to this problem.

If on the other hand we had $x_i \geq 1$ then the generating functions for the L_i languages is $\frac{x}{1-x}$ and we multiply this still by $\frac{1}{1-x}$ for L_y. Here there is an asymmetry between L_y, which contains the empty string, and the languages L_i which don't. So for this problem, we have

$$\frac{x^k}{(1-x)^{k+1}} = x^k \sum_{n=0}^{\infty} \binom{n+k}{k} x^n = \sum_{n=0}^{\infty} \binom{n+k}{k} x^{n+k} = \sum_{n=k}^{\infty} \binom{n}{k} x^n.$$

14.4 Summary

a) For $x_1 + \cdots + x_k = n$ and $x_i \geq 0$, we have the OGF

$$\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty} \binom{n+k-1}{k-1} x^n.$$

b) For $x_1 + \cdots + x_k = n$ and $x_i \geq 1$, we have the OGF

$$\frac{x^k}{(1-x)^k} = \sum_{n=k}^{\infty} \binom{n-1}{k-1} x^n.$$

c) For $x_1 + \cdots + x_k \leq n$ and $x_i \geq 0$, we have the OGF

$$\frac{1}{(1-x)^{k+1}} = \sum_{n=0}^{\infty} \binom{n+k}{k} x^n.$$

d) For $x_1 + \cdots + x_k = n$ and $x_i \geq 1$, we have the OGF

$$\frac{x^k}{(1-x)^{k+1}} = \sum_{n=k}^{\infty} \binom{n}{k} x^n.$$

15 Coefficient Extraction

In this section, we develop some techniques for extracting coefficients from generating functions. Remember that coefficient of x^n represents the number of solutions of weight n. So knowing that the generating function is, for example,

$$A(x) = \frac{x^{k-1}(x^5 - x^{14})}{(1-x)^k} = \sum a_n x^n,$$

our goal is to find a_n. A common notation for this is $a_n = [x^n]A(x)$.

Warning

Be careful! This is not the same thing as $A(n)$. Plugging in n is not the way to find the coefficient. For example, the coefficient of x^2 in the quadratic $f(x) = 2x^2 + 3x + 1$ is $[x^2]f(x) = 2$ and not $f(2)$ which equals 15.

15.1 Shifts

Multiplying $A(x)$ by x^k shifts the sequence left or right (depending on whether k is positive or negative):

$$x^k \sum a_n x^n = \sum a_n x^{n+k} = \sum a_{n-k} x^n.$$

The second equals sign is obtained either by substituting $n \leftarrow n + k$ or by noticing that the coefficient is always k less than the exponent.

Substituting inside a sum

Consider a summation like

$$\sum_{k=2}^{\infty} k(k-1)x^{k-2}.$$

If we want to write this as a sum where x^k is the exponent in our summand, then we can either

1) add 2 everywhere we see k:

$$\sum_{k+2=2}^{\infty}(k+2)(k+1)x^k = \sum_{k=0}^{\infty}(k+2)(k+1)x^k,$$

2) make a substitution of $n = k-2$ and $k = n+2$ to get

$$\sum_{n+2=2}^{\infty}(n+2)(n+1)x^n.$$

Both methods are equivalent.

Putting this together, we have the following result.

Theorem 15.1 (Coefficient Shift). *If $A(x) = \sum a_n x^n$ then*

$$[x^n]x^k A(x) = [x^{n-k}]A(x) = a_{n-k}.$$

"The coefficient of x^n in $x^k A(x)$ is a_{n-k}."

15.2 Sums and differences

Suppose $A(x) = \sum a_n x^n, B(x) = \sum b_n x^n$ are two power series. Then for any constants λ, μ, we have

$$\lambda A(x) + \mu B(x) = \lambda \sum a_n x^n + \mu \sum b_n x^n = \sum (\lambda a_n + \mu b_n) x^n.$$

In terms of our $[x^n]$ operator, we record the following result.

Theorem 15.2 (Linear Combinations).

$$[x^n](\lambda A(x) + \mu B(x)) = \lambda [x^n]A(x) + \mu [x^n]B(x).$$

15.3 Derivatives

As we saw earlier,

$$A'(x) = \sum n a_n x^{n-1} = \sum (n+1) a_{n+1} x^n.$$

Theorem 15.3 (Derivatives and Antiderivatives).

$$[x^n]A'(x) = (n+1)[x^{n+1}]A(x).$$

Or for antiderivatives,

$$[x^n]A(x) = \frac{1}{n}[x^{n-1}]A'(x).$$

15.4 Newton's Binomial Theorem

We've seen the Binomial Theorem,

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

And, in the previous section, we worked out power series for $(1-x)^{-n}$. These kinds of functions are quite common in combinatorics, and not just for integer exponents either.

Definition 15.1. The general binomial coefficient is

$$\binom{r}{k} = \frac{r(r-1)(r-2)\cdots(r-k+1)}{k!}$$

and we define this for any real (or complex) number r.

What Newton observed, is that when you take the derivative of $f(x) = (1+x)^r$ a few times, you get

$$r(1+x)^{r-1} \text{ then } r(r-1)(1+x)^{r-2} \text{ then } r(r-1)(r-2)(1+x)^{r-3}, \ldots$$

Theorem 15.4 (Newton's Binomial Theorem)**.** *The Taylor series of* $(1+x)^r$ *is*

$$\sum_{k=0}^{\infty} \binom{r}{k} x^k.$$

Proof. We've seen above that for $f(x) = (1+x)^r$, we have

$$f^{(k)}(x) = r(r-1)(r-2)\cdots(r-k+1)(1+x)^{r-k}.$$

Plugging this into the formula for a Taylor series,

$$(1+x)^r = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} x^k = \sum_{k=0}^{\infty} \frac{r(r-1)(r-2)\cdots(r-k+1)}{k!} = \sum_{k=0}^{\infty} \binom{r}{k} x^k.$$

□

Note

These series converge for at least $|x| < 1$ (and so techniques of calculus apply to analyzing these series). Convergence for $x = \pm 1$ depends on r. And of course, if r is a whole number then the series is finite and converges everywhere.

15.5 Examples

Let's finally get to solving the problem of Section 14.2.1. Recall that we had the generating function

$$A(x) = \frac{x^{k-1}(x^5 - x^{14})}{(1-x)^k} = \frac{x^{k+4} - x^{k+13}}{(1-x)^k}.$$

Now we're going to combine this with the first formula from Section 14.4 to get

$$\begin{aligned}
[x^n]\frac{x^{k+4} - x^{k+13}}{(1-x)^k} &= [x^n]x^{k+4}\frac{1}{(1-x^k)} - [x^n]x^{k+13}\frac{1}{(1-x)^k} \\
&= [x^{n-k-4}]\frac{1}{(1-x)^k} - [x^{n-k-13}]\frac{1}{(1-x)^k} \\
&= \binom{(n-k-4)+k-1}{k-1} - \binom{(n-k-13)+k-1}{k-1} \\
&= \binom{n-5}{k-1} - \binom{n-14}{k-1}.
\end{aligned}$$

Coefficient extraction with the formulas

The formula from Section 14.4 says

$$\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty}\binom{n+k-1}{k-1}x^n.$$

That means that the coefficient of x^n is

$$[x^n]\frac{1}{(1-x)^k} = \binom{n+k-1}{k-1}.$$

If we want to apply this to a more complex power of x, we do so by substituting where we see n, for example,

$$[x^{n+1}]\frac{1}{(1-x)^k} = \binom{(n+1)+k-1}{k-1} = \binom{n+k}{k-1}.$$

15.5.1 Negative Binomial Theorem

Comparing Theorem 15.4 and formula (a.) of Section 14.4 yields an interesting result:

$$\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty}\binom{n+k-1}{k-1}x^n = \sum_{n=0}^{\infty}\binom{-k}{n}(-x)^n. \tag{15.1}$$

Note that we are looking at $(1-x)^{-k}$ not $(1+x)^{-k}$ hence the $(-x)$ on the right.

The next theorem is most often stated with n and k swapped. We also use the identity $\binom{a+b}{a} = \binom{a+b}{b}$.

Theorem 15.5 (Negative Binomial Theorem)**.**

$$\binom{-k}{n}(-1)^n = \binom{n+k-1}{k-1} = \binom{n+k-1}{n}.$$

Or, with n and k in the usual position:

$$\binom{-n}{k}(-1)^k = \binom{n+k-1}{n-1} = \binom{n+k-1}{k}.$$

Like many things in combinatorics, it's interesting to give a second proof of this beyond the generating function proof in Equation 15.1.

Proof.

$$\begin{aligned}\binom{-n}{k}(-1)^k &= \frac{(-n)(-n-1)(-n-2)\cdots(-n-k+1)}{k!}(-1)^k \\ &= \frac{(n)(n+1)(n+2)\cdots(n+k-1)}{k!} \\ &= \frac{(n+k-1)(n+k-2)\cdots(n+1)(n)}{k!} \\ &= \binom{n+k-1}{k}.\end{aligned}$$

In line 2, we distribute one -1 to each of the k factors in the numerator. □

15.5.2 Central Binomial Theorem

We've seen the binomial theorem with a whole number exponent. We've seen it with a negative integer exponent. The next place to go is fractions. Let's start with $n = \frac{1}{2}$. For reference, here are the first couple values of this:

```
for k in range(4):
    print(binomial(1/2, k))
```

```
1
1/2
-1/8
1/16
```

It would seem that these are ± 1 over some power of 2 but if we continue our computation, we find that the next value is $-5/128$.

In order to get to the bottom of this, we need to work this out from the definition:

$$\begin{aligned}\binom{1/2}{k} &= \frac{\frac{1}{2}(\frac{1}{2}-1)(\frac{1}{2}-2)\cdots(\frac{1}{2}-k+1)}{k!} \\ &= \frac{\frac{1}{2}(-\frac{1}{2})(-\frac{3}{2})(-\frac{5}{2})\cdots(-\frac{2k-3}{2})}{k!} \\ &= \frac{(-\frac{1}{2})^{k-1}(1)(3)(5)\cdots(2k-3)}{2k!} \\ &= (-1)^{k-1}\frac{(1)(3)(5)\cdots(2k-3)}{2^k k!}.\end{aligned}$$

This is a pretty good formula. We can also write it in terms of the double factorial which is the product of all the odd numbers up to $2k-3$. (It's called *double* because the factors decrease by 2 each time instead of 1.)

But like in the Wikipedia article, we can write double factorials in terms of more usual factorials. We do this by adding in the missing even numbers to the numerator and balancing that out in the denominator:

$$\begin{aligned}\binom{1/2}{k} &= (-1)^{k-1}\frac{(1)(3)\cdots(2k-3)\cdot(2)(4)\cdots(2k-2)}{2^k k!\cdot(2)(4)\cdots(2k-2)} \\ &= (-1)^{k-1}\frac{(1)(2)(3)\cdots(2k-3)(2k-2)}{2^k k!\cdot 2^{k-1}(1)(2)\cdots(k-1)} \\ &= (-1)^{k-1}\frac{(2k-2)!}{2^{2k-1}k!(k-1)!} \\ &= (-1)^{k-1}\frac{1}{2^{2k-1}k}\binom{2k-2}{k-1}\end{aligned}$$

Let's do this once more with $n=-1/2$ but going through the steps a bit quicker:

$$\begin{aligned}\binom{-1/2}{k} &= \frac{(-\frac{1}{2})(-\frac{3}{2})(-\frac{5}{2})\cdots(-\frac{2k-1}{2})}{k!}\\ &= (-1)^k\frac{1\cdot 3\cdot 5\cdots(2k-1)}{2^k k!}\\ &= (-1)^k\frac{(2k)!}{2^k k!\cdot(2)(4)\cdots(2k)}\\ &= (-1)^k\frac{(2k)!}{2^k k!\cdot 2^k k!}\\ &= (-1)^k\frac{1}{4^k}\binom{2k}{k}.\end{aligned}$$

Let us record these results as a theorem.

Theorem 15.6 (Half Binomials)**.**

$$\binom{1/2}{k} = (-1)^{k-1}\frac{1}{2^{2k-1}k}\binom{2k-2}{k-1} \text{ and } \binom{-1/2}{k} = \frac{(-1)^k}{4^k}\binom{2k}{k}.$$

The first formula is only valid for $k \geq 1$. If $k = 0$ then we get $\binom{1/2}{0} = 1$ (product of zero terms in the numerator equals 1 and the denominator is $0! = 1$).

Example 15.1. Take the second formula, and use it with Theorem 15.4. We will put $(-4x)^k$ instead of x^k to cancel the $(-1/4)^k$. That is,

$$\begin{aligned}\frac{1}{\sqrt{1-4x}} = (1-4x)^{-1/2} &= \sum_{k=0}^{\infty}\binom{-1/2}{k}(-4x)^k\\ &= \sum_{k=0}^{\infty}\frac{(-1)^k}{4^k}\binom{2k}{k}(-4x)^k\\ &= \sum_{k=0}^{\infty}\binom{2k}{k}x^k\end{aligned}$$

This is a neat little result in and of itself but there is deeper implications here. Consider the number of binary strings of length $2k$ with an equal number of 0s and 1s. That means there are k of each, and that gives us the formula $\binom{2k}{k}$. These are called the **central binomial coefficients** because they appear right down the center of Pascal's triangle.

For those of you who have taken computer science or mathematical linguistics, there is a concept known as a "regular language"—languages which can be parsed by a regular expresion[1].

[1]We'll define many of these terms in a later section for those of you who haven't seen them before.

For these regular languages, we will see later on, that they always yeild a *rational* generating function. Here we have a non-rational generating function and indeed this language of strings with equal numbers of 0s and 1s is not a regular language.

Going beyond regular languages, the Chomsky–Schützenberger Theorem says that if you have an unambiguous context-free grammar, then the generating function for that language is *algebraic*. That is what we are looking at with $\frac{1}{\sqrt{1-4x}}$. This is an algebraic function because it is a square root of a rational function. Indeed, the language of strings with equal numbers of 0s and 1s is a context-free language.

15.5.3 Catalan Numbers

If you remember the formula for the Catalan numbers even vaguely, you'll recognize the $\binom{2k}{k}$. If you squint real hard, the first formula of Theorem 15.6 looks vaguely similar once you deal with the $(-1)^k$ and the 4^k as we did in Example 15.1. Let's dig into that.

First, recall the decomposition we had for Dyck paths.

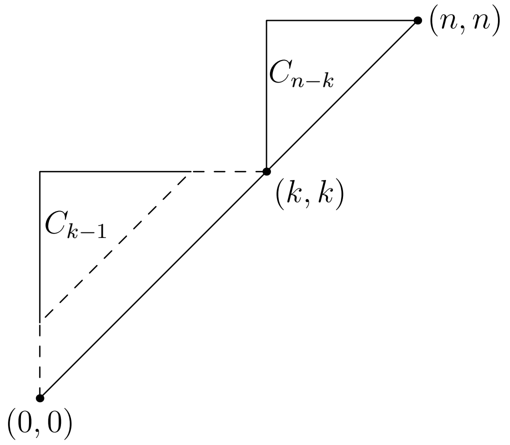


Figure 15.1: Dyck path decomposition based on the first place the path hits the line $y = x$

In terms of "the language of Dyck paths," what this says is that a Dyck path of length n is equal to

1) a shift one step up from the diagonal
2) a Dyck path of length $k - 1$
3) a Dyck path of length $n - k$

Here's the beauty of generating functions: as long as we keep track of the shifts, we don't need to keep track of the lengths of the subwords or subpaths. So we can describe the language of Dyck paths in simpler terms: it is a shift, together with two smaller Dyck paths. Therefore, if we represent our shift by the monomial x, and our Catalan generating function by $C(x)$, what we have here is $xC(x)^2$.

We almost have it, we just need to be careful of the edge case. If $n = 0$, the path from $(0, 0)$ to $(0, 0)$ is a valid Dyck path which involves no shift. So a Dyck path is either

a) this length 0 Dyck path, which has OGF $x^0 = 1$
b) a shift, together with two Dyck paths, with OGF $xC(x)^2$.

Therefore, $C(x)$ satisfies the equation

$$C(x) = 1 + xC(x)^2.$$

If you solve this quadratic equation using the quadratic formula, we find that

$$C(x) = \frac{1 \pm \sqrt{1-4x}}{2x}. \tag{15.2}$$

In order to get numbers out of this, we use Theorem 15.4 to expand the square root (separating the $k = 0$ term so we can apply the formula from Theorem 15.6):

$$\begin{aligned}\sqrt{1-4x} &= 1 + \sum_{k=1}^{\infty} \binom{1/2}{k} (-4x)^k \\ &= 1 + \sum_{k=1}^{\infty} (-1)^{k-1} \frac{1}{2^{2k-1}k} \binom{2k-2}{k-1} (-4x)^k \\ &= 1 + \sum_{k=1}^{\infty} \frac{-2}{k} \binom{2k-2}{k-1} x^k.\end{aligned}$$

The presence of the $-$ sign indicates that we want to use the negative square root in Equation 15.2. Doing this, we obtain

$$\begin{aligned}-\sqrt{1-4x} &= -1 + \sum_{k=1}^{\infty} \frac{2}{k} \binom{2k-2}{k-1} x^k \\ 1 - \sqrt{1-4x} &= \sum_{k=1}^{\infty} \frac{2}{k} \binom{2k-2}{k-1} x^k \\ \frac{1-\sqrt{1-4x}}{2x} &= \sum_{k=1}^{\infty} \frac{1}{k} \binom{2k-2}{k-1} x^{k-1} \\ &= \sum_{k=0}^{\infty} \frac{1}{k+1} \binom{2k}{k} x^k\end{aligned}$$

(adding 1 to each k to get from $k - 1$ to k).

We've now found another way to get our Catalan number formula.

Theorem 15.7 (Catalan OGF)**.** *The generating function for the Catalan numbers is*

$$C(x) = \frac{1-\sqrt{1-4x}}{2x} = \sum_{k=0}^{\infty} \frac{1}{k+1}\binom{2k}{k} x^k.$$

16 Exercises

1. Given that $\dfrac{1}{1-ax} = \sum_{n=0}^{\infty} a^n x^n$, convert the following into summation notation (in terms of x^n rather than x^{n+k})

   a) $\dfrac{1}{1-x} + \dfrac{1}{1+x}$
   b) $\dfrac{x^3}{1+2x}$
   c) $\displaystyle\int_0^x \frac{1}{1-t}\, dt = \log\left(\frac{1}{1-x}\right)$ (hint: take the antiderivative/integral term by term)
   d) $\dfrac{3}{2-x}$ (hint: convert to $\dfrac{a}{1-bx}$)
   e) $\dfrac{1+x}{1-x^3}$ (hint: split the fraction into two fractions)

Hints

a) Start with

$$\frac{1}{1-y} = \sum_{n=0}^{\infty} y^n$$

where maybe $y = x^3$ and we get x^{3n} or $y = -x$ and we get $(-x)^n$.

b) If we multiply by x^3 and get $\sum a^n x^{n+3}$ then subtract 3 from n everywhere to get $\sum a^{n-3} x^n$. For the bottom of the summation, we would change from $n = 0$ to $n - 3 = 0$ or $n = 3$.

c) If we integrate $\sum x^n$ term by term, we get $\sum x^{n+1} n + 1$.

d) Convert to $\dfrac{3/2}{1-x/2}$.

e) You can write your answer as $\sum a_n x^n$ where $a_n = \ldots$ (formula will depend on if $n = 3k, 3k+1, 3k+2$).

2. Convert each sequence to a closed form generating function. Use the most obvious choice of form for the general term of the sequence. Sequences start at $n = 0$.

   a) $2, 2, 2, 2, 2, 2, \ldots$

b) $0, 1, 0, 1, 0, 1, \ldots$

c) $1, 0, 1, 0, 1, 0, \ldots$

d) $0, 1, 2, 3, 4, 5, \ldots$ (hint: factor out an x from $\sum nx^n$ so that the terms are $\frac{d}{dx}x^n$, then move the derivative operator outside the summation [but after the x that was factored out])

e) $1, 3, 5, 7, 9, 11, \ldots$ (hint: use part d)

Hint

a) Write it first as $\sum 2x^n$ and factor out the 2.

b) Write it first as $x + x^3 + x^5 + \ldots$

c) Write it first as $1 + x^2 + x^4 + \ldots$

d) $\dfrac{d}{dx}\dfrac{1}{1-x} = \dfrac{1}{(1-x)^2} = \sum nx^{n-1}$

3. Using the formulas from section Section 14.4, find

a) $\displaystyle\sum_{n=0}^{\infty} \binom{n+2}{2} x^n$

b) $\displaystyle\sum_{n=5}^{\infty} \binom{n-1}{4} x^n$

Hint

a) Can be plugged into the formula directly

b) Look for the formula with $n-1$ and then figure out how to choose the right k.

4. Find the indicated coefficients.

a) $[x^{10}]\dfrac{1}{1+2x}$

b) $[x^{20}]\dfrac{1}{(1-x)^7}$

c) $[x^{20}]\dfrac{x^3}{(1-x)^{10}}$

d) $[x^{85}]\dfrac{2x^5}{1-3x^5}$ (hint: if the sum is $\sum(*)x^{5n+5}$ then the coefficient of x^{85} occurs where $5n + 5 = 85$)

Hint

a) Use the formula $\sum(-2)^n x^n$ and take the coefficient of x^{10}

b) Use the formula from Section 14.4.

c) First use the shift formula from Theorem 15.1 to write it as $[x^{17}]\dfrac{1}{(1-x)^{10}}$.

d) The sum is $\sum 2 \cdot 3^n x^{5n+5}$

5. Write down a generating function for the number of ways to make change for n dollars using $1, 5$ and 10 dollar bills. (Hint: if we are receiving multiples of \$5 bills, then the generating function should be $x^0 + x^5 + x^{10} + \cdots$ where the weight function of a bill is its dollar value.)

Hint

If the options for \$5 bills are $0, 1, 2, \ldots$ then we have $1 + x^5 + x^{10} + x^{15} + \ldots$ where the coefficient gives the number of choices (only 1 way to have k bills of the same type) and the exponent gives the weight or how much it contributes. You should get geometric series for each bill value.

6. Repeat the previous problem but now suppose we are only allowed to give out a maximum of one \$5 bill.

Hint

Similar to problem 5. but now there are only 0 or 1 choices for the \$5 bill. So $1 + x^5$ is the generating function for that denomination.

Solutions

1. a) $\displaystyle\sum_{n=0}^{\infty}((-1)^n + 1)x^n$

   b) $\displaystyle\sum_{n=3}^{\infty}(-2)^{n-3}x^n$

   c) $\displaystyle\sum_{n=1}^{\infty}\frac{x^n}{n}$

   d) $\displaystyle\sum_{n=0}^{\infty}\frac{3}{2^{n+1}}x^n$

   e) $\displaystyle\sum_{n=0}^{\infty}a_n x^n$ where $a_n = 0$ if $n = 3k + 2$ and $a_n = 1$ otherwise (so the sequence $1, 1, 0, 1, 1, 0, 1, 1, 0, \ldots$).

2. a) $\dfrac{2}{1-x}$

   b) $\dfrac{x}{1-x^2}$

   c) $\dfrac{1}{1-x^2}$

d) $\dfrac{x}{(1-x)^2}$

e) $\sum(2n+1)x^n = 2\sum nx^n + \sum x^n = \dfrac{2}{(1-x)^2} + \dfrac{1}{1-x}$.

3. a) $\dfrac{1}{(1-x)^3}$

b) $\dfrac{x^5}{(1-x)^5}$

4. a) $(-2)^{10}$

b) $\dbinom{26}{6}$

c) $\dbinom{26}{9}$

d) $2 \cdot 3^{16}$

5. $\dfrac{1}{(1-x)(1-x^5)(1-x^{10})}$

6. $\dfrac{1+x^5}{(1-x)(1-x^{10})}$

Part V

Regular Languages

17 Regular Languages

At this point we've seen numerous examples where a language is composed of a sequence of blocks.

- A binary string is a sequence of 0s and 1s
- A Dyck path is a sequence of smaller paths, shifted properly
- A tiling of a $2 \times n$ grid using dominoes is a sequence of vertical dominoes or pairs of horizontal dominoes
- Equivalently: we looked at sequences of 1s and 2s which sum to n; both give the Fibonacci numbers
- Binary strings with no 11—also counted by the Fibonacci numbers—can be broken into sequences of blocks although the blocks are more complex than the previous example

17.1 Kleene Star

Let L be a language that does not contain the empty string (for reasons of unique representation). The language which repeats L in a sequence is called the Kleene star of L, denoted L^*. An element of L^* is some finite sequence

$$u_1 u_2 \cdots u_k, \text{ where } u_i \in L.$$

Another name for such a sequence of length k is L^k (concatenating k strings from L). Thus,

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots .$$

It's also common notation to write these unions with a + sign to mean "or" (with a heavy implication that we want this to be disjoint). Also note that $L^0 = \{\varepsilon\}$ because concatenating 0 strings yields an empty string. So we have

$$L^* = \varepsilon + L + L^2 + L^3 + \cdots .$$

17.1.1 Kleene Plus

A related construction considers only the sequences of positive length. This is called the Kleene plus operator and it is defined by

$$L^+ = L + L^2 + L^3 + \cdots .$$

17.2 Relation to Generating Functions

Theorem 12.1 and Theorem 13.1 imply the following theorem.

Theorem 17.1 (Kleene star and plus)**.** *Provided the sets* $L^0, L^1, L^2, \ldots$ *are* disjoint*,*

$$\begin{aligned}\Phi_{L^*} &= \Phi_{L^0} + \Phi_{L^1} + \Phi_{L^2} + \Phi_{L^3} + \cdots \\ &= \Phi_L^0 + \Phi_L^1 + \Phi_L^2 + \Phi_L^3 + \cdots \\ &= \frac{1}{1-\Phi_L}.\end{aligned}$$

Similarly,

$$\begin{aligned}\Phi_{L^+} &= \Phi_L^1 + \Phi_L^2 + \Phi_L^3 + \cdots \\ &= \frac{\Phi_L}{1-\Phi_L}.\end{aligned}$$

17.3 Examples

Consider our sequences of 1s and 2s which gave us the Fibonacci sequence. Here, unlike many previous examples, we don't want to weight sequences by length but rather by sum (we're looking for sequences whose sum is n not whose length is n). So let f denote the sum of a sequence, e.g. $f(122) = 1+2+2 = 5$.

Let $L = \{1, 2\}$. Then L^* represents all sequences of 1s and 2s. By theorem Theorem 17.1, we have

$$\Phi_{L^*} = \frac{1}{1-\Phi_L} = \frac{1}{1-(x^{f(1)} + x^{f(2)})} = \frac{1}{1-x-x^2}. \tag{17.1}$$

We know that the number of these sequences which sum to n is counted by the Fibonacci sequence, so like we stated Section 12.3.1, we have

```
taylor(1/(1 - x - x^2), x, 0, 10)
```

$89\,x^{10} + 55\,x^9 + 34\,x^8 + 21\,x^7 + 13\,x^6 + 8\,x^5 + 5\,x^4 + 3\,x^3 + 2\,x^2 + x + 1$

17.3.1 Avoiding '11'

If we want to avoid 11 we need to make sure that everyone 1 has at least one 0 next to it. We can do this with the language 0^+1 which guarantees a zero on the left. Taking sequences of these, we get $(0^+1)^*$.

We're getting close to the description. We also need to allow a potential initial 1 in case because that's the place in the string which might have 0s only on the right. So $(\varepsilon + 1)(0^+1)^*$. Lastly, we can have as many zeros as we want after the last 1 so $L = (\varepsilon + 1)(0^+1)^*0^*$.

Here we're weighting by length again, so

- $\Phi_{\varepsilon+1} = x^0 + x^1$
- $\Phi_{0^+1} = \frac{x}{1-x} \cdot x = \frac{x^2}{1-x}$
- $\Phi_{(0^+1)^*} = \frac{1}{1-\frac{x^2}{1-x}}$
- $\Phi_{0^*} = \frac{1}{1-x}$

Putting that all together, we have

$$\Phi_L = (1+x) \cdot \frac{1}{1 - \frac{x^2}{1-x}} \cdot \frac{1}{1-x} = \frac{1+x}{1-x-x^2}.$$

This is similar to Equation 17.1 but the numerator shifts the sequence over a step:

```
taylor((1 + x)/(1 - x - x^2), x, 0, 10)
```

$144\,x^{10} + 89\,x^9 + 55\,x^8 + 34\,x^7 + 21\,x^6 + 13\,x^5 + 8\,x^4 + 5\,x^3 + 3\,x^2 + 2\,x + 1$

17.4 Even number of 1s

Heuristically, half of all binary strings should have an even number of 1s and half should have an odd number. We'll prove this using generating functions.

For a start, the language of binary strings can be written as $(0+1)^*$ (sequences of 0s or 1s). If we want to guarantee that every 1 is partnered with a second 1 later on, then we can modify this to $L = (0 + 10^*1)^*$. Now we compute

$$\Phi_{10^*1} = x \cdot \frac{1}{1-x} \cdot x = \frac{x^2}{1-x}.$$

So therefore,

$$\begin{aligned}\Phi_L &= \frac{1}{1-(x+\frac{x^2}{1-x})}\\ &= \frac{1-x}{(1-x)-x(1-x)-x^2}\\ &= \frac{1-x}{1-2x}\\ &= \frac{1}{1-2x}-\frac{x}{1-2x}\\ &= \sum_{n=0}^{\infty} 2^n x^n - \sum_{n=0}^{\infty} 2^n x^{n+1}\\ &= \sum_{n=0}^{\infty} 2^n x^n - \sum_{n=1}^{\infty} 2^{n-1} x^n.\end{aligned}$$

And that means that the number of such strings is

$$\begin{cases} 2^n - 2^{n-1} = 2^{n-1} & \text{if } n \geq 1 \\ 1 & \text{if } n = 0 \end{cases}.$$

17.5 Regular Languages

We describe **regular languages** inductively:

- a finite language is regular
- if L, L' are regular then so is $L + L'$
- if L, L' are regular then so is LL'
- if L, L' are regular then so are L^* and L^+

That is, a regular language is everything you can create out of a finite description using unions, concatenations, Kleene stars and pluses, and letters from the alphabet.

Such a description using these operations is known as a **regular expression**. This concept is closely related to regular expressions in computer programming although computer programming includes many more operations such as the `?` operator where $L? = \varepsilon + L$.

Note

Some regular expression operators in computer programming allow one to create languages which are not regular. For instance, back-references allow one to ask whether a previous part of the string has been repeated. So computer programming regular expressions exceeds what formal language regular expressions can describe. See Wikipedia for more info.

18 Finite Automata

As we saw in Section 17.3, it can be a bit tricky to come up with a regular expression for a language. Often it's easier to draw what is called a **finite state automaton** (FSA) instead. For instance, for the no 11s language, we make states representing the last character seen as we move through the string and if the last character was a 1 and we see a second 1, we know the string should be thrown out.

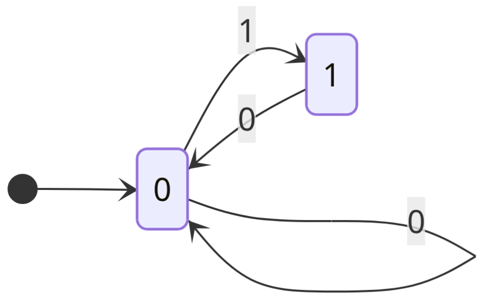


Figure 18.1: The Fibonacci automaton

18.1 Interpreting a FSA

The way to use a FSA is to input a string one digit at a time, let's say from left to right. So if we take the string 010011 and input it into the automaton in Figure 18.1, we start in state 0. The arrow

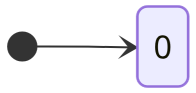


Figure 18.2: Start arrow

indicates which state to start in.

Then we read the first character in our string, which is 0 and this tells us to follow the edge labeled 0 from our current state, which puts us back in state 0.

Next we read a 1 and this tells us to follow the edge labeled 1 and puts us in state 1. Then another 0 puts us back in state 0. Then state 0 again. Then state 1.

With the last 1, we try to follow an edge labeled 1 out of state 1, but there is no such edge. So at this point we fail and that means the string 010011 is rejected by the automaton.

18.2 A FSA for even '1's

To create this automaton, we need one state to represent an even number of 1s and one state to represent an odd number of 1s. To distinguish between success states and failure states, we will use a thick border on the success states.

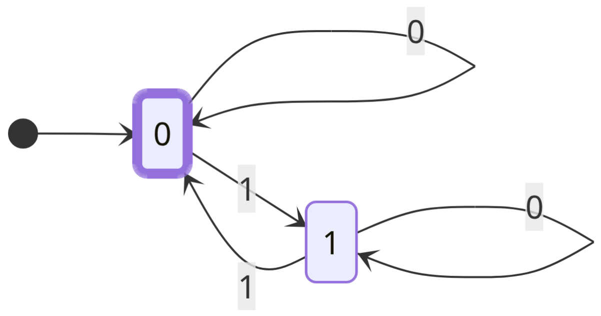


Figure 18.3: Automaton for an even number of 1s

These success or failure states are used at the end of reading a string. If we finish in a success state, the string is good. If we finish in a failure state, the string is bad.

18.3 State Elimination

Let's have a look at how to turn a FSA into a regular expression. To do this, we will modify our automata step by step, marking each path by what regular expression goes through each state.

18.3.1 One in, one out

For example, a state like

Can be replaced by

In order to enter state S we need to see an a, then we can use b as many times as we want, then we need to see a c to leave.

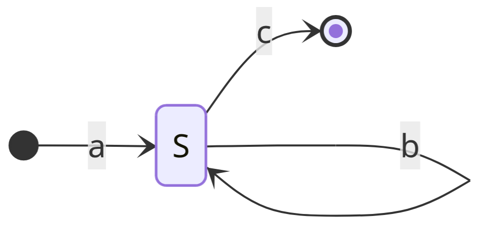


Figure 18.4: Pre-state elimination

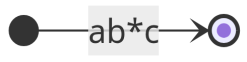


Figure 18.5: Post-state elimination

18.3.2 Parallel edges

Given two parallel edges, we can combine them using + meaning “or”.

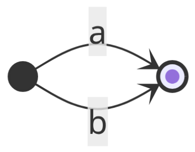


Figure 18.6: Pre-state elimination for parallel edges

This says to get from the first state to the second, we can take either a or b.

18.3.3 Multiple ins or outs

Suppose we have multiple in-edges or out-edges on a state. We can replace the state with a path from each input to each output and record the expression for that path according to Section 18.3.1.

18.3.4 Trinary Example

Consider the language of trinary strings containing at least one 1 and one 2. Let’s setup states recording whether we’ve yet seen a 1, a 2 or both.

In order to eliminate every state, we make sure to have a starting and ending node that we can get an answer going from start to finish. There is only one accepting state here, “012”, which can go to the ending node whenever we finish.

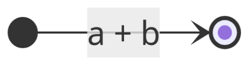


Figure 18.7: Post-state elimination for parallel edges

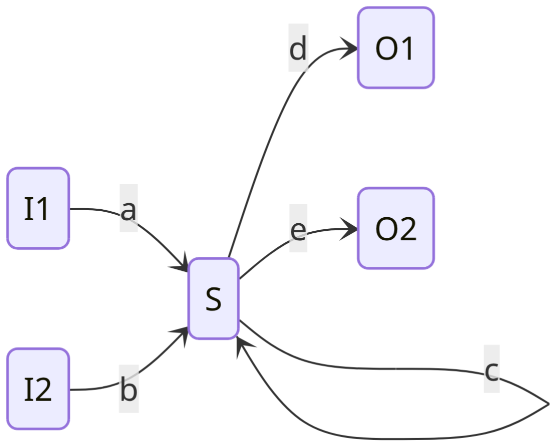


Figure 18.8: Pre-state elimination for multiple edges

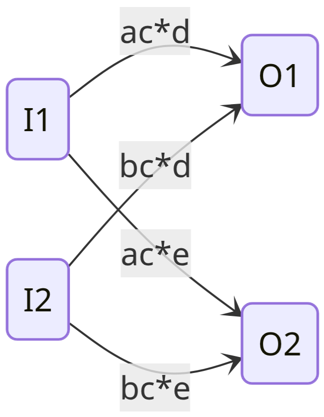


Figure 18.9: Pre-state elimination for multiple edges

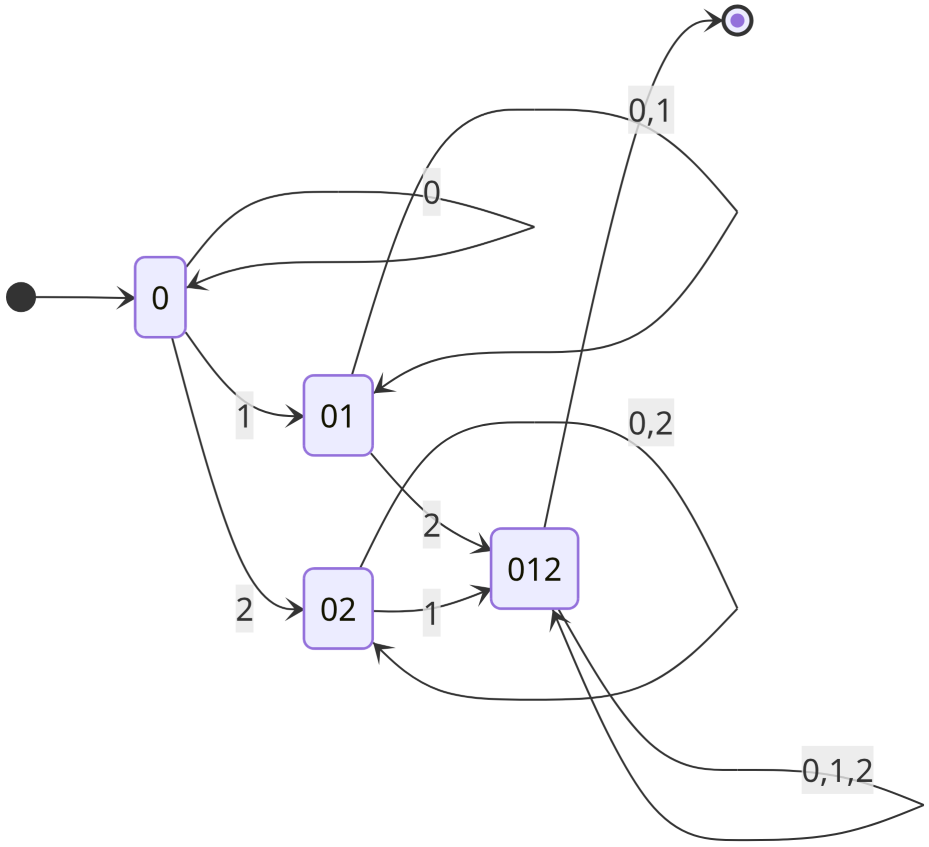


Figure 18.10: FSA for trinary strings with at least one 1 and one 2

Next, we eliminate states “01” and “02” using Section 18.3.1.

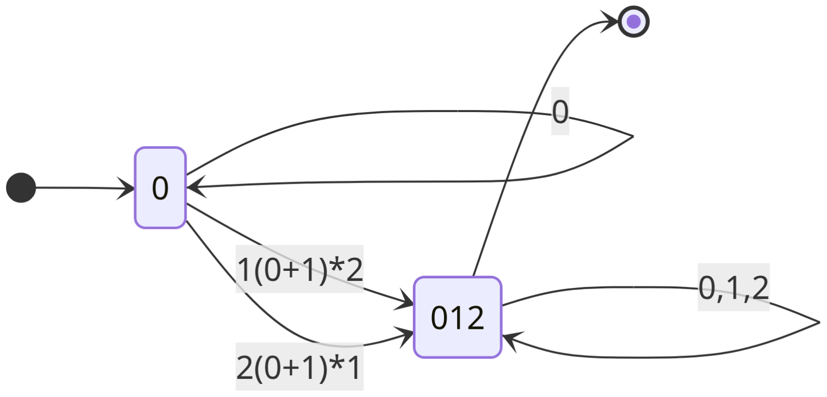


Figure 18.11: FSA for trinary strings with at least one 1 and one 2, begin state elimination

Then we use Section 18.3.2 and finish off with Section 18.3.1 to eliminate the remaining states.

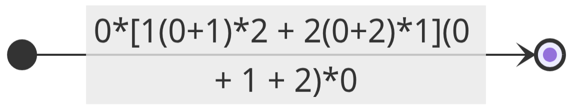


Figure 18.12: FSA for trinary strings with at least one 1 and one 2, finish state elimination

This gives us the regular expression

$$0^*[1(0+1)^*2+2(0+2)^*1](0+1+2)^*.$$

18.4 Fibonacci Example

Consider the language of binary strings with no “11.” We can make an automata with three states: a good state where we can take anything, a state representing that we’ve last seen a “1” and one which represents that we’ve seen two “1”s in a row.

As before, we have a start and end state represented by just circles.

So first, we can eliminate the state for “11” since going there will never lead to a valid string.

The easiest state to eliminate here is state “1” because it has the fewest number of arrows and no loops. There is one path into state “1” and two out. So following Section 18.3.3, we replace these with the path $1\varepsilon = 1$ from state “0” to the end and with the path 10 from 0 back to itself.

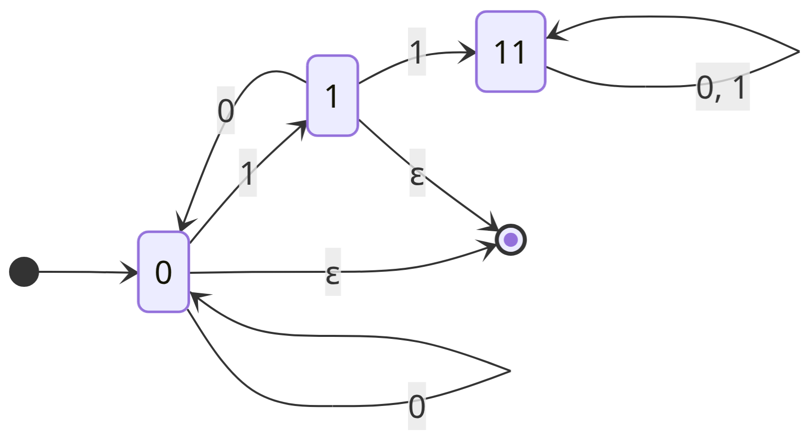


Figure 18.13: FSA for binary strings with no 11

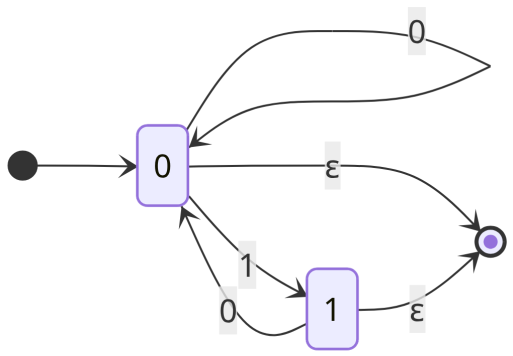


Figure 18.14: FSA for binary strings with no 11, no sink state

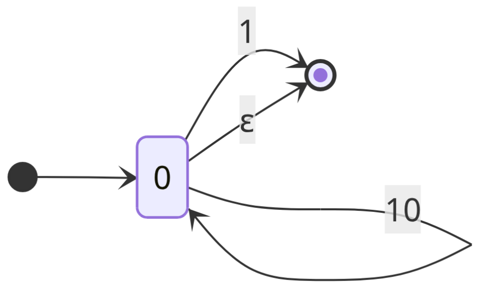


Figure 18.15: FSA for binary strings with no 11, eliminate state 1

We now have some parallel loops and parallel edges, which we can add together to get a loop of $0 + 10$ and an edge of $\varepsilon + 1$. So the regular expression we come to is

$$(0 + 10)^*(\varepsilon + 1).$$

19 Putting Everything Together

We've done some examples where we take a description and create a finite state automaton. We've done state elimination to create regular expressions. So let's look one last time at how to turn these regular expressions into formulae.

19.1 Trinary Example

In Section 18.3.4, we had the regular expression

$$0^*[1(0+1)^*2+2(0+2)^*1](0+1+2)^*.$$

Now we could break this into sublanguages like $L_0 = \{0\}$ and $\Phi_{L_0} = x^{f(0)} = x^1$ but we can streamline this a bit.

1. Take your regular expression.
2. Replace each character like $0, 1, 2$ with $x^{f(0)}, x^{f(1)}, x^{f(2)}$. If we're weighting by length these should all just be x.
3. Replace A^* by $\frac{1}{1-A}$.

Let's illustrate:

$$\begin{aligned}
&0^*[1(0+1)^*2+2(0+2)^*1](0+1+2)^* \\
&\mapsto x^*[x(x+x)^*x+x(x+x)^*x](x+x+x)^* \\
&= x^*[2x^2(2x)^*](3x)^* \\
&= \frac{1}{1-x}\frac{2x^2}{1-2x}\frac{1}{1-3x} \\
&= \frac{2x^2}{(1-x)(1-2x)(1-3x)}.
\end{aligned}$$

19.1.1 Getting a formula

On it's face, it's hard to tell what the taylor series is for this function. So we first ask Sage to give us a partial fraction decomposition.

```
%display latex
A = 2x^2 / ((1 - x)*(1 - 2x)*(1 - 3x))
A.partial_fraction()
```

$$-\frac{1}{3\,x-1}+\frac{2}{2\,x-1}-\frac{1}{x-1}$$

Sage typically puts the x's first but it's more useful to us to write this as

$$\frac{1}{1-3x}-\frac{2}{1-2x}+\frac{1}{1-x}.$$

We use the important identity $\frac{1}{1-ax}=\sum a^n x^n$ to convert this to

$$\sum_{n=0}^{\infty}3^n x^n-2\sum_{n=0}^{\infty}2^n x^n+\sum_{n=0}^{\infty}x^n=\sum_{n=0}^{\infty}(3^n-2\cdot 2^n+1)x^n.$$

Conclusion: there are $3^n-2\cdot 2^n+1$ trinary strings of length n with at least one 1 and one 2.

Note 2

We can also obtain this answer by Inclusion/Exclusion. Let X denote the set of all trinary strings of length n. Let A be strings with no 1 and let B be strings with no 2. Then $|X|=3^n$ and $|A|=|B|=2^n$ since each represents strings with two choices of character. We want $|X|-|A\cup B|=|X|-|A|-|B|+|A\cap B|$ using the formula we had in Section 3.4. So $3^n-2^n-2^n+1$ which matches what we have using generating functions.

We're beginning to see how generating functions tie together our material on binomial coefficients, strings, inclusion/exclusion and also recurrence relations.

19.2 Fibonacci Example

At the end of Section 18.4, we had the regular expression $(0+10)^*(\varepsilon+1)$. We repeat our method to turn this into a generating function. First, we should point out that ε will become $x^{f(\varepsilon)}=x^0=1$. So

$$(x+x^2)^*(1+x)=\frac{1+x}{1-x-x^2}.$$

Partial fractions don't help us out here. At least…not easily.

On the other hand, we know that the Fibonacci numbers satisfy the recurrence $f_n = f_{n-1} + f_{n-2}$. To make the connection clearer, let us write this as

$$f_n - f_{n-1} - f_{n-2}.$$

19.3 More Examples

Consider the sequence generated by $a_0 = 0, a_1 = 0, a_2 = 1$ and $a_n + 4a_{n-1} - 2a_{n-2} + 5a_{n-3}$ for $n \geq 3$.

We can compute a generating function $\sum a_n x^n$ in Sage using the following commands.

ℹ Note

Per the Sage documentation, the coefficients are given in ascending order (from $n-3$ to $n-2$ to $n-1$):

$$a_{n+3} = -5a_n + 2a_{n+1} - 4a_{n+2}.$$

```
C = CFiniteSequences(ZZ) # sequences over the integers
C.from_recurrence([-5, 2, -4], [0, 0, 1]).ogf()
```

$$\frac{x^2}{5x^3 - 2x^2 + 4x + 1}$$

The numerator is hard to predict as it depends on the initial values as well as the recurrence. But pay attention to that denominator:

$$1 + 4x - 2x^2 + 5x^3$$

These coefficients match our recurrence!

19.4 General Result

Suppose a Taylor series $\sum a_n x^n$ can be written as a rational function $P(x)/Q(x)$ where:

1. $\deg P < \deg Q$
2. $Q(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_k x^k$ (degree k)

Then the coefficients a_n satisfy the recurrence relation

$$c_0 a_n + c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} = 0, \text{ for } n \geq k$$

(And the reverse holds as well!)

Note

The condition that $\deg P < \deg Q$ isn't crucial to this. It takes care of a few things. First, if $\deg P \geq \deg Q$ we could do long division to simplify.
Second, if we have something like $x/(1-x) = -1 + 1/(1-x)$, then the recurrence relation we get from the denominator is $a_n = a_{n-1}$ but if we write down this Taylor series, the coefficients are $0, 1, 1, 1, 1, \ldots$. So the recurrence relation $a_n = a_{n-1}$ only holds if $n \geq 2$.
The big takeaway here is: making $\deg P \geq \deg Q$ alters the initial terms and the recurrence relation takes longer to kick in.

Proof. Let $A = \sum a_n x^n = P(x)/Q(x)$. Multiplying both sides by $Q(x)$ we should find that $AQ = P$ (a polynomial). So

$$\begin{aligned} AQ &= (c_0 + c_1 x + \cdots + c_k x^k) \sum a_n x^n \\ &= c_0 \sum a_n x^n + c_1 \sum a_n x^{n+1} + \cdots + c_k \sum a_n x^{n+k} \\ &= c_0 \sum a_n x^n + c_1 \sum a_{n-1} x^n + \cdots + c_k \sum a_{n-k} x^n. \end{aligned}$$

Now this is supposed to be a polynomial. And it is, if and only if the coefficients are 0 for n bigger than $\deg P$ and in particular if $n \geq k = \deg Q$. So for $n \geq k$, we have

$$c_0 a_n + c_1 a_{n-1} + \cdots + c_k a_{n-k}.$$

And the steps here are reversible: if we satisfy this recurrence then AQ is a polynomial of degree less than k. □

19.5 Epilogue

When our generating function comes apart nicely into partial fractions, we can use the identity $\frac{1}{1-ax} = \sum a^n x^n$ to extract the coefficient of a^n. More generally, we can handle generating functions like $1/(1-ax)^k$ using the formulas of Section 14.4.

For something like $x/(1-x-x^2)$, partial fractions *can* be used to obtain

$$\frac{x}{1-x-x^2} = \frac{1}{\sqrt{5}} \left(\frac{1}{1-\phi x} - \frac{1}{1-\psi x} \right); \text{ where } \phi, \psi = \frac{1 \pm \sqrt{5}}{2}.$$

With the resulting formula

$$f_n = \frac{1}{\sqrt{5}}(\phi^n - \psi^n).$$

Known as *Binet's formula* which we saw a version of in Section 12.3.1. This formula is super cool but computationally leaves a lot to be desired as multiplying large decimal numbers together can be quite expensive.

The fastest way to compute the Fibonacci sequence comes right from the recurrence definition. With a little bit of linear algebra, we can turn that linear recurrence into

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix} = \begin{pmatrix} 0f_n + f_{n+1} \\ f_n + f_{n+1} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_{n+2} \end{pmatrix}.$$

So the Fibonacci numbers and in fact, all of these linear recurrence relations, can be computed via matrix multiplication. This has the immediate advantage that we can use the identity $A^{2n} = (A^n)^2$ to cut down our number of steps quite significantly. I.e. we have an algorithm where if n has k bits in binary, then the algorithm takes $\mathcal{O}(k) = \mathcal{O}(\log_2 n)$ matrix multiplications.

Tip

The Sage commands for this look like

```
C = CFiniteSequences(ZZ)
Fib = C.from_recurrence([1, 1], [0, 1])
Fib.closed_form()
```

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

20 Transfer Matrix Method

Let D be a directed graph on the vertices $1, \dots, n$. The **adjacency matrix** of D is the matrix $A = A(D)$ where

$$A_{ij} = \begin{cases} 1 & \text{if } i \to j \text{ is a directed edge,} \\ 0 & \text{otherwise} \end{cases}$$

These adjacency matrices allow for loop edges $i \to i$ and can be made to allow for parallel edges if we let A_{ij} be the number of parallel edges from $i \to j$.

20.1 Example

Consider the graph

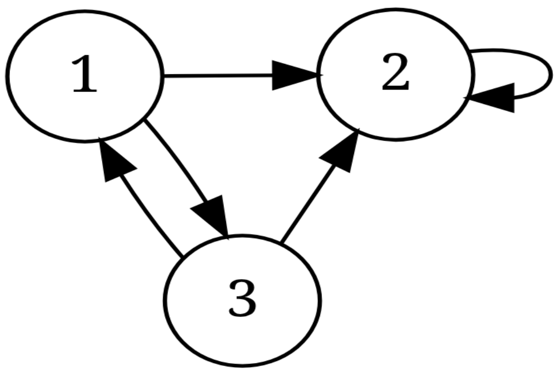


Figure 20.1: A digraph on 1,2,3 with edges 12, 13, 22, 31, 32

The adjacency matrix of this digraph is

$$A = \begin{array}{c c} & \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \tag{20.1}$$

For instance, row 1 says that vertex 1 has edges $1 \to 2$ and $1 \to 3$.

20.2 Matrix Multiplication

In linear algebra, we define the product of two matrices A and B by

$$(AB)_{ij} = \sum_t A_{it} B_{tj}.$$

Visually, this means we take the i-th row of A and take the dot product with the j-th row of B. For instance, if A is the adjacency matrix in Equation 20.1, then

$$A_{12} = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 2.$$

This product has a combinatorial meaning:

$$(A^2)_{ij} = \sum_t A_{it} A_{tj} = \sum_t \begin{cases} 1 & \text{if } i \to t \to j \text{ are edges,} \\ 0 & \text{else.} \end{cases}$$

In other words: A^2 **gives the number of walks of length** 2.

Theorem 20.1. *If A is the adjacency matrix for some digraph D, then $(A^n)_{ij}$ is the number of walks of length n from i to j.*

Proof. We prove this by induction. First, the base case: if $n = 1$ then $(A^1)_{ij} = A_{ij}$ is by definition the number of walks (single edges) from i to j.

For the inductive step, we assume that (for every i and j) $(A^{n-1})_{ij}$ counts the number of walks from i to j of length $n - 1$. Then

$$(A^n)_{ij} = (A^{n-1}A)_{ij} = \sum_t (A^{n-1})_{it} A_{tj}.$$

Here, the first factor, $(A^{n-1})_{it}$ counts the number of walks from i to t of length $n - 1$ and A_{tj} is the number of edges from t to j. So the whole term $(A^{n-1})_{it} A_{tj}$ counts the number of walks from i to t to j. So this is every walk of length n from i to j. □

20.2.1 Example

For the digraph in Figure 20.1, how many walks are there of length 3 from vertex 1 to vertex 2? We can do $1, 2, 2, 2$; or $1, 3, 2, 2$; or $1, 3, 1, 2$. So there are 3 walks of length 3. Therefore, we expect that A^3 will have a 3 in row 1, column 2. Let's use SageMath to check this.

```
%display latex
A = Matrix([[0, 1, 1], [0, 1, 0], [1, 1, 0]])
A^3
```

$$\begin{pmatrix} 0 & 3 & 1 \\ 0 & 1 & 0 \\ 1 & 3 & 0 \end{pmatrix}$$

Examples of other things this matrix tells us: there are 0 walks from 1 to 1 of length 3; there is only 1 walk of length 3 from 1 to 3, namely $1, 3, 1, 3$.

20.3 Generating Functions

Recall that when we started with generating functions, a monomial like ax^n meant "a objects of length or size n." So if we do $A^n x^n$ then the entries will represent "$(A^n)_{ij}$ walks of length n." Then the generating function for this is

$$\begin{aligned} M &= A^0x^0 + A^1x^1 + A^2x^2 + A^3x^3 + \cdots \\ &= I + Ax + A^2x^2 + A^3x^3 + \cdots \\ &= (I - Ax)^{-1} \end{aligned}$$

Where we use a similar formula as our geometric series, $\frac{1}{1-ax}$, except a is now a matrix.

For the digraph in Figure 20.1, this generating function is

```
M = (identity_matrix(3) - A*x)^-1
M.simplify_rational()
```

$$\begin{pmatrix} -\frac{1}{x^2-1} & \frac{x}{x^2-2\,x+1} & -\frac{x}{x^2-1} \\ 0 & -\frac{1}{x-1} & 0 \\ -\frac{x}{x^2-1} & \frac{x}{x^2-2\,x+1} & -\frac{1}{x^2-1} \end{pmatrix}$$

Theorem 20.2. *Each entry M_{ij} of this matrix is the generating function for the number of walks from vertex i to vertex j.*

For instance, the generating function for walks from vertex 1 to vertex 3 is

$$\frac{x}{1-x^2} = x + x^3 + x^5 + x^7 + \cdots .$$

This agrees with what we can see from the diagram: there is exactly one walk of length n for any odd number n and none of even length.

Of course, the true utility of this is when we can't do the calculation in our heads of the number of walks. For instance, the number of walks from vertex 1 to vertex 2 of length n is given by the generating function

$$\frac{x}{1-2x+x^2} = \frac{x}{(1-x)^2} = \sum_{n=0}^{\infty} nx^n$$

(using the formula from Section 14.4). So there are n walks of length n.

20.4 Fibonacci

We can use this method to simplify a lot of the work we had to do with state elimination. Have a look at our Fibonacci automaton again, simplified down to 2 states.

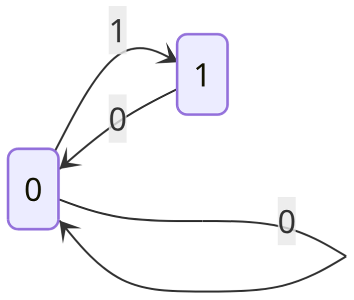


Figure 20.2: FSA for binary strings with no 11, no sink state

A valid string is a walk from vertex 0 to either vertex 0 or vertex 1 (depending on the last bit of the string). So we want $M_{00} + M_{01}$ for the number of walks from vertex 0 to vertex 0 plus from vertex 0 to vertex 1.

Here's how to accomplish that in SageMath. We start with the adjacency matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Then compute $M = (I - A)^{-1}$

```
A = Matrix([[1, 1], [1, 0]])
I = identity_matrix(2)
M = (I - A*x)^-1
OGF = M[0][0] + M[0][1]
OGF.simplify_rational()
```

$$-\frac{x+1}{x^2+x-1}$$

And this matches our answer in Section 19.2.

20.5 Recurrence Relations

With all this matrix algebra, the results we get can get pretty gnarly. Sometimes it's enough to get a recurrence relation to describe our answer. By Section 19.4, we know that the recurrence is given by the denominator.

We will use now a theorem from linear algebra. ::: {#thm-matrix-inverse} The inverse of a matrix A is given by

$$A^{-1} = \frac{1}{\det A} \operatorname{Adj}(A)$$

where the adjugate matrix $\operatorname{Adj}(A)$ has entries given by determinants of A deleting one row and one column. :::

What's important to us here is that the denominator is $\det A$. So the denominator of our generating functions is

$$\det(I - Ax)$$

This is related to the characteristic polynomial but with the coefficients reversed.

20.5.1 Example

For our Fibonacci FSA in Figure 20.2, the denominator is

```
det(I - A*x)
```

$$-x^2 - x + 1$$

This tells us the recurrence relation is

$$\begin{array}{cccc} 1 & -1x & -1x^2 & \\ \downarrow & \downarrow & \downarrow & \\ a_n & -a_{n-1} & -a_{n-2} & = 0 \end{array}$$

Note

In Section 20.3, we saw that some of the denominators were $1 - x^2 = (1-x)(1+x)$ and some were $1 - x$ and some were $(1-x)^2$. The denominator we calculate using $\det(I - Ax)$ is $(1-x)^2(1+x) = 1 - x - x^2 + x^3$ leading to the recurrence $a_n - a_{n-1} - a_{n-2} + a_{n-3}$.

All of the quantities will satisfy the longer recurrence but they also individually satisfy a shorter recurrence which may be different from their siblings.

21 Exercises

1. For each of the automata below, eliminate the state "E" by:

   a) first identify all the arrows pointing into "E" and out of "E"

   b) replace each path $A \xrightarrow{u} E \xrightarrow{v} B$ by $A \xrightarrow{ul^*v} B$ where l is the loop at "E".

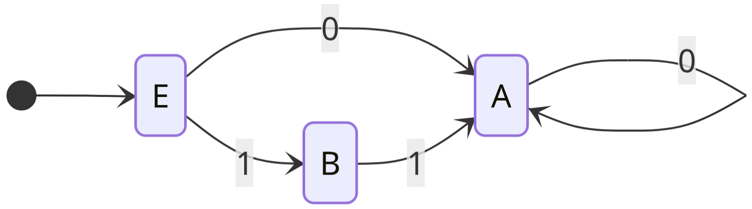


Figure 21.1: FSA for exercise 1

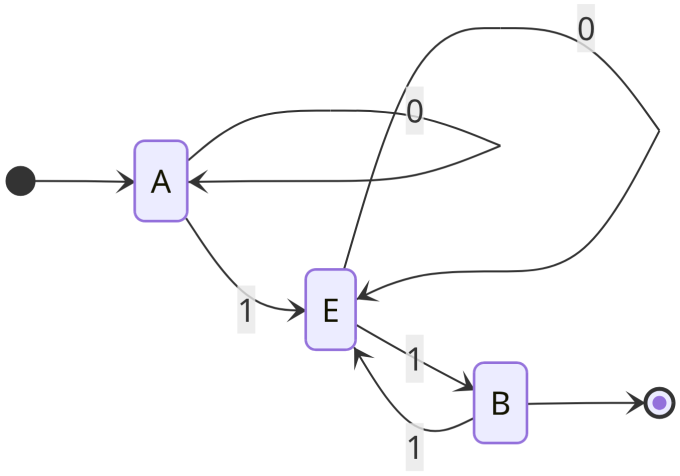


Figure 21.2: FSA for exercise 1

2. Write the described language using $+, \cdot$ (concatenation) and * (Kleene star or plus)

   a) $\{\varepsilon, 0, 1, 11, 111, 1111, \cdots\}$

   b) Strings of 0s with at least two 0s

   c) Strings of 0s with an even number of 0s

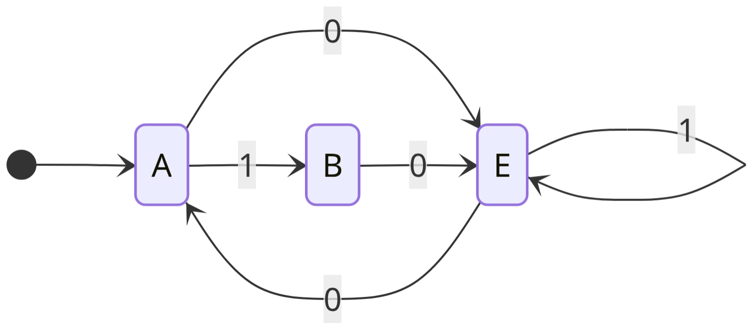


Figure 21.3: FSA for exercise 1

d) Strings of 0s with an odd number of 0s

Hint

a) How does this compare to $\{\varepsilon, 1, 11, 111, 1111, \dots\} = 1^*$?
b) Set the two 0s aside first (compare to $a^+ = aa^*$)
c) Think in terms of repeating 00
d) Combine ideas from b. and c.

3. Consider the language $L = 0^*(11)^*$.

   a) List all words in L of length ≤ 6.
   b) Let a_n be the number of words in L of length n. Using part a, guess a formula for a_n (which may depend on whether n is even or odd).
   c) Can you prove your formula?

Hint

In words: L describes languages with some number of 0s followed by an even number of 1s. So for instance, if $n = 9$, the possible number of 1s is $0, 2, 4, 6, 8$ and there would be 5 possible strings. The formula should look something like $n/2$ or $(n+1)/2$.

4. Consider a tiling problem where we tile a $2 \times n$ grid using horizontal or vertical dominos (1×2 or 2×1) or a 2×2 square. Let $\Sigma = \{V, H, S\}$ where V represents a vertical domino, H represents a horizontal domino, and S represents a square. A tiling can be described by the language $L = \Sigma^*$.

   a) Let f be the weight function describing the width. So $f(V) = 1, f(H) = f(S) = 2$. What is the generating function for Σ?
   b) What is the generating function for $L = \Sigma^*$?

c) Using SageMath's Taylor series method, compute the number of such tilings of a 2×10 grid. Note: you have to write `2*x` rather than `2x` in SageMath.
d) Using our techniques from recursion, what is a recursive formula for the number of tilings t_n? Compare this to the denominator of your OGF like in Section 19.4.

Hint

The generating function of Σ is

$$\Phi_\Sigma = \sum_{w \in \Sigma} x^{f(w)} = x^{f(V)} + x^{f(H)} + x^{f(S)}.$$

For b. use Theorem 17.1.
For d. use the fact that a tiling of length n is either: * a tiling of length $n-1$ followed by a vertical domino, * a tiling of length $n-2$ followed by either a pair of horizontal dominoes or a square (so 2 options).

5. Draw a finite automata for:

   a) the language $0^*(11)^*$
   b) the language $(0+11)^*$

The automata in these notes were drawn using the Mermaid Javascript library. You may wish to copy the following code to mermaid.live and modify it to create the FSA you need.

```
stateDiagram-v2
    direction LR
    classDef accept stroke-width:5px
    [*] --> 0
    0 --> 0 : 0
    0 --> 1 : 1
    1 --> 11 : 1
    11 --> 11 : 1

    class 0, 11 accept
```

Hint

The FSA for part a should require only slight modification of the sample Mermaid diagram. Likewise, part b should require only slight modification from part a.

6. Take the FSA from the previous question and perform state elimination to check if you recover the regular expression you started with. Remember to add in a termination state before you start eliminating.

Hint

Use the ab^*c formula described in Section 18.3.1 on each path $A \to S \to B$ to eliminate S.

7. In Example 8.18 of Keller and Trotter's book, they show using exponential generating functions that the number of ternary strings with an even number of 0s is $(3^n+1)/2$. In this exercise, we will get this answer using ordinary generating functions.

   a) Draw a FSA for this language (you can do it in two states, not including the start and end).
   b) Perform state elimination to obtain a regular expression.
   c) Convert that regular expression to an OGF.
   d) Using SageMath, find a partial fraction decomposition for the OGF.
   e) Verify that the Taylor series is $\sum \frac{3^n+1}{2}x^n$.

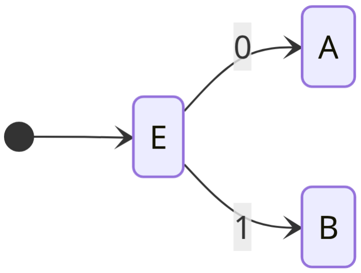
Hint

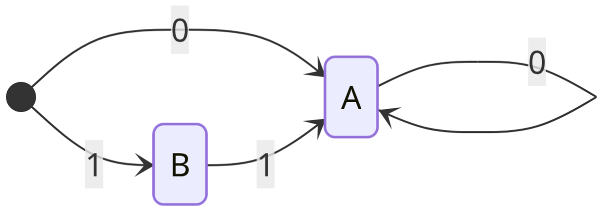
a) The start state will represent an even number of 0s seen so far and another state will represent an odd number.
b) Eliminate the odd state using the elimination rule: `in(loop)*out`.
c) Replace $0, 1, 2$ by x^1 since each of those characters has length 1.
d) Don't forget to use `2*x` rather than `2x`.
e) Use the formula $\frac{a}{1-r} = \sum ar^n$.

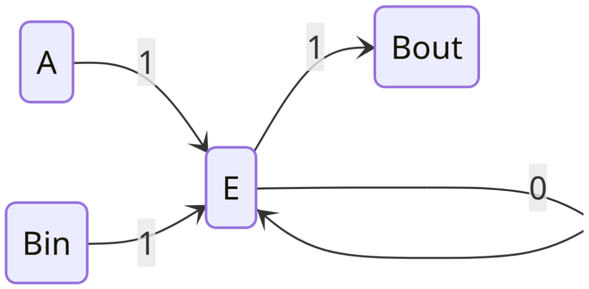
Solutions

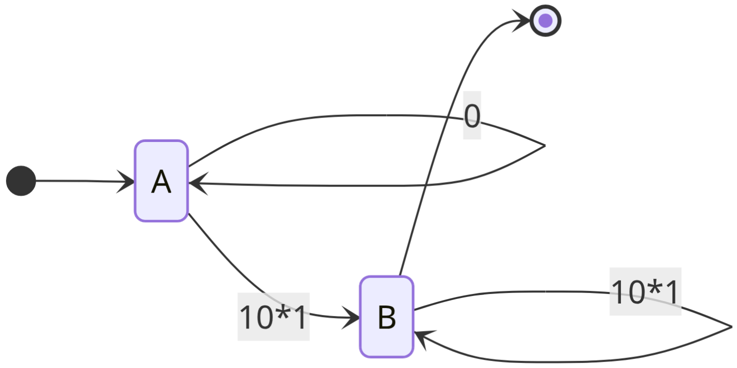
1.

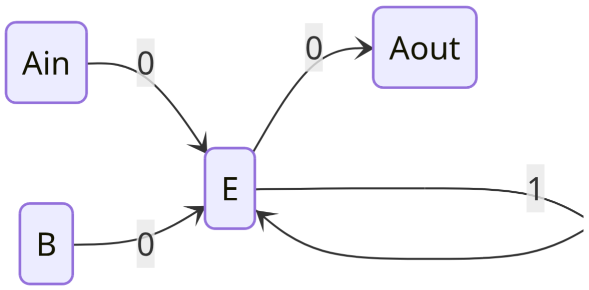
2. a) $0 + 1^*$
   b) 000^* or 00^+
   c) $(00)^*$
   d) $0(00)^*$

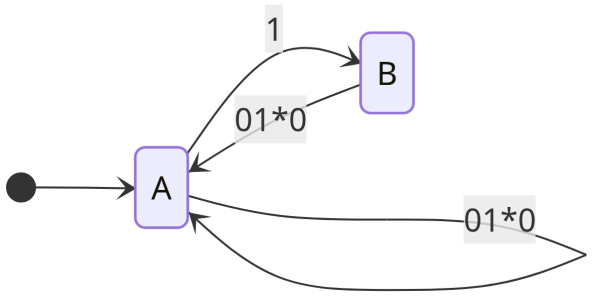

E
0
A
1
B


0
A
0
1
B
1


A
1
1
Bout
E
0
Bin
1


0
A
10*1
B
10*1


Ain
0
0
Aout
E
1
B
0


1
B
01*0
A
01*0