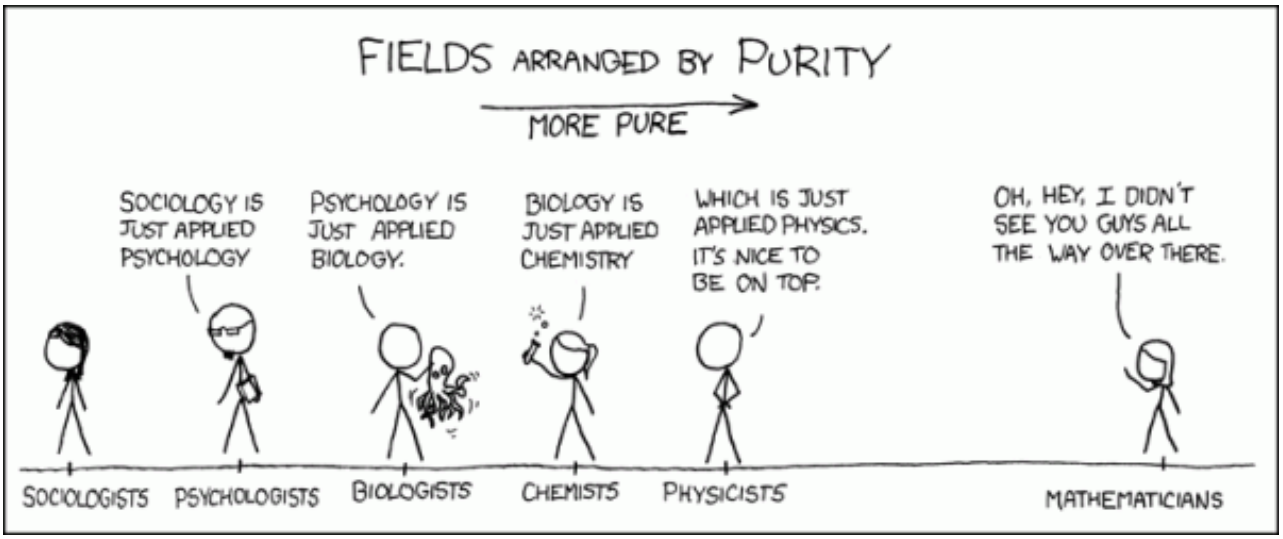
MATH SCRAPBOOK

Uwe Hoffmann

UWE HOFFMANN

MATH SCRAPBOOK

NOTES AND SOLVED PROBLEMS

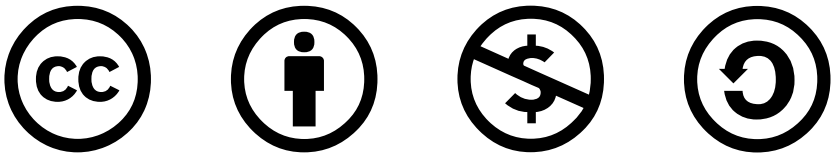


http://xkcd.com/435/

Formatted with LaTeX using the https://tufte-latex.github.io/tufte-latex/ template.

xkcd comics http://xkcd.com, used under CC license.

UWE@CODEMANIC.COM



Version January 2025

Dedicated to my family, in appreciation of their love and support

Preface

Collection of math/cs notes and problems written up over the years for my own amusement. Topics are at the undergraduate college level. Normally notes like these would be written with a pencil in a notebook. LaTeX makes it very easy to produce publishing quality typesetting of mathematical texts, so blame LaTeX for this book. Hopefully others will find some of these notes useful.

I'm an amateur but that hasn't prevented me from enjoying writing these notes just as not being Michael Jordan has never prevented me from enjoying pickup basketball. Describing and explaining has helped me understand the topics involved. Errors and misunderstandings are solely my fault. There is no original content in these notes and I tried to be complete about attributions and citations but if I missed something, I apologize.

Code snippets are mostly in Haskell and Mathematica. I'm not an expert in either but they should work.

Contents

1
Airplane Seating

Problem

A line of n airline passengers is waiting to board a plane. They each hold a ticket to one of the n seats on that flight. (For convenience, let's say that the ith passenger in line has a ticket for the seat number i.) Unfortunately, the first person in line is crazy, and will ignore the seat number on their ticket, picking a random seat to occupy. All of the other passengers are quite normal, and will go to their proper seat unless it is already occupied. If it is occupied they will then find a free seat to sit in, at random. What is the probability that the last (nth) person to board the plane will sit in their proper seat (#n)?

Any seat arrangement under the rules of the problem is a permutation π from the set S_n of permutations of size n. Let's define $A_n \subseteq S_n$ the subset of permutations of size n that are valid seat arrangements.

Let $B_n := \{\pi \in A_n : \pi(n) = n\}$ be the subset of A_n where the last person gets their proper seat. A strategy to solve the problem would be to count $|A_n|$ and $|B_n|$ and then divide them up to get the probability.

We will use the permutation cycle notation $(i_1, i_2, \ldots, i_k)$ for a cycle of length k that maps $i_1 \mapsto i_2 \mapsto \ldots i_k \mapsto i_1$. Also let ι_n be the identity permutation in S_n and let $A_n^* = A_n \setminus \{\iota_n\}$ and $B_n^* = B_n \setminus \{\iota_n\}$.

Let's characterize permutations in A_n^*.

Lemma 1.1. *A permutation $\pi \in A_n^*$ is a cycle of the form*

$$\pi = (1, i_1, i_2, \ldots, i_k) \text{ with } 2 \leq i_1 < i_2 < \ldots < i_k \leq n$$

Proof. Consider $\pi \in A_n^*$. Suppose $\pi(1) = 1$ then under the rules of the problem all other passengers can occupy their seat and $\pi = \iota_n$ which is a contradiction because A_n^* doesn't have the identity permutation.

So there exists a $i_1 \in \{2, \ldots, n\}$ with $\pi(1) = i_1$. i_1 cannot map to any $j < i_1$ because under the rules of the problem every $j < i_1$ maps to itself (every $j < i_1$ finds their seat unoccupied so they take it). So there exists a $i_2 \in \{2, \ldots, n\}$ with $i_2 > i_1$ and $i_1 \mapsto i_2$. And so on. This means that π has at least the cycle $(1, i_1, i_2, \ldots, i_k)$ with $2 \leq i_1 < i_2 < \ldots < i_k \leq n$. It cannot have any other cycles that don't have 1 in them because under the rules of the problem only passenger 1 can start a seat rearrangement and all passengers not affected by that rearrangement will occupy their seat.

□

Definition 1.2. Let $2^{\{2,\ldots,n\}}$ be the set of all subsets of $\{2, \ldots, n\}$. The function $\varphi : 2^{\{2,\ldots,n\}} \to S_n$ is defined as:

$$\begin{aligned} \varphi(\oslash) &= \iota_n \\ \varphi(\{i_1, i_2, \ldots, i_k\}) &= (1, i_1, i_2, \ldots, i_k) \\ &\text{assuming } 2 \leq i_1 < i_2 < \ldots < i_k \leq n \end{aligned}$$

φ is a valid function because for each subset there is only one cycle possible with the monotonically increasing ordering. From lemma 1.1 it then follows that $\varphi(2^{\{2,\ldots,n\}}) = A_n$, so $|A_n| = 2^{n-1}$.

For B_n we apply the same arguments, except we take out the n-th passenger. A permutation $\pi' \in B_n^*$ is a cycle of the form

$$\pi' = (1, i_1, i_2, \ldots, i_k) \text{ with } 2 \leq i_1 < i_2 < \ldots < i_k \leq n-1$$

and there is a function φ' defined as

$$\begin{aligned} \varphi'(\oslash) &= \iota_n \\ \varphi'(\{i_1, i_2, \ldots, i_k\}) &= (1, i_1, i_2, \ldots, i_k) \\ &\text{assuming } 2 \leq i_1 < i_2 < \ldots < i_k \leq n-1 \end{aligned}$$

that defines a bijection from $2^{\{2,\ldots,n-1\}}$ to B_n. It means that $|B_n| = 2^{n-2}$ for $n \geq 2$.

So the probability that the last (nth) person to board the plane will sit in their proper seat is $\frac{|B_n|}{|A_n|} = 0.5$ for $n \geq 2$.

2
Schröder-Bernstein Theorem

Bijections from one-to-one functions are the topic[1] in this note. The problem statement is known as the Schröder-Bernstein Theorem.

[1] Exercise 1.5.11 on page 32 from Stephen Abbott. *Understanding Analysis*. Springer, 2 edition, 2015. ISBN 978-1-4939-2711-1.

Problem

Let $f : X \to Y$ and $g : Y \to X$ be one-to-one functions. Then there exists a bijection $h : X \to Y$.

The given functions are one-to-one, so for subsets $f(X)$ and $g(Y)$ they are already bijections. This leads to the idea of partitioning X and Y such that we can compose a bijection h piece-wise from f and g^{-1} using the partitions. In particular given a subset $A \subseteq X$, we consider the sets A, $X \setminus A$, $f(A)$, $Y \setminus f(A)$ and $g(Y \setminus f(A))$. We want subsets $A \subseteq X$, such that $A \cap g(Y \setminus f(A)) = \emptyset$, as shown in figure 2.2. Let's define this as property P:

$$\forall A \subseteq X : P(A) \Leftrightarrow A \cap g(Y \setminus f(A)) = \emptyset$$

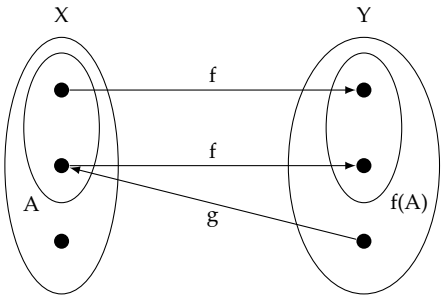


Figure 2.1: A violates $P(A)$

If we have a subset $A \subseteq X$ that satisfies $P(A)$, then we can define the bijection h:

$$h(x) = \begin{cases} f(x) & : x \in A \\ g^{-1}(x) & : x \in g(Y \setminus f(A)) \end{cases}$$

The domain of h is $A \cup g(Y \setminus f(A))$, which is not necessarily equal to X, so we are not done yet. Our goal therefore is to find a subset $A \subseteq X$ that satisfies $P(A)$ and for which $A \cup g(Y \setminus f(A)) = X$.

Let

$$\Lambda = \{A \subseteq X : P(A)\}$$

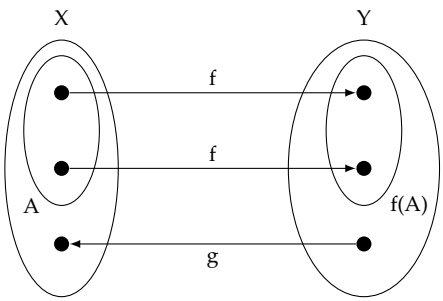


Figure 2.2: A satisfies $P(A)$

be the set of all subsets of X that satisfy property P and let $\bar{A}$ be the union of all such subsets

$$\bar{A} = \bigcup_{A \in \Lambda} A$$

Lemma 2.1. *$\bar{A}$ is the biggest subset of X that satisfies P.*

Proof. First we show that $\bar{A}$ satisfies P. Assume

$$\exists y \in Y \setminus f(\bar{A}) \text{ with } g(y) \in \bar{A}$$

Then there exists a set $A \in \Lambda$ with $g(y) \in A$ [2]. $A \subseteq \bar{A}$, so $f(A) \subseteq f(\bar{A})$. Therefore $Y \setminus f(\bar{A}) \subseteq Y \setminus f(A)$, so $y \in Y \setminus f(A)$. But this contradicts A satisfying property P, so no such y exists. It follows that $\bar{A}$ satisfies P too.

Assume there is a set A' that satisfies P and that is bigger than $\bar{A}$, so $\bar{A} \subseteq A'$. But $A' \in \Lambda$ and $\bar{A} = \bigcup_{A \in \Lambda}$, so $A' \subseteq \bar{A}$. That means $A' = \bar{A}$. □

[2] Because $\bar{A} = \bigcup_{A \in \Lambda}$.

With $\bar{A}$ we can define the partitions $X = \bar{A} \oplus (X \setminus \bar{A})$ and $Y = f(\bar{A}) \oplus (Y \setminus f(\bar{A}))$.

Lemma 2.2.

$$g(Y \setminus f(\bar{A})) = X \setminus \bar{A}$$

Proof. Because $\bar{A}$ satisfies P, we already know that

$$g(Y \setminus f(\bar{A})) \subseteq X \setminus \bar{A}$$

Now assume

$$\exists x \in X \setminus \bar{A} \text{ such that } \forall y \in Y \setminus f(\bar{A}) : g(y) \neq x$$

But then $\bar{A} \cup \{x\}$ satisfies P [3] and is bigger than $\bar{A}$. This contradicts lemma 2.1. So no such x exists and the lemma is proven. □

[3] We have

$$Y \setminus f(\bar{A} \cup \{x\}) \subseteq Y \setminus f(\bar{A})$$

so

$$\forall y \in Y \setminus f(\bar{A} \cup \{x\}) : g(y) \notin \bar{A} \cup \{x\}$$

We can now define the bjection $h : X \to Y$ with

$$h(x) = \begin{cases} f(x) & : x \in \bar{A} \\ g^{-1}(x) & : x \in X \setminus \bar{A} \end{cases}$$

which solves the problem in this section. [4]

[4] The solution uses a nifty proof strategy: maximize a mathematical structure so that its "complement" has no choice but to satisfy a certain property, ie not satisfying the property would contradict the maximality.

3
Bridge Crossings

Problem

Four people begin on the same side of a bridge. You must send them across to the other side in the fastest time possible. It is night. There is one flashlight. A maximum of two people can cross at a time. Any party who crosses, either one or two people, must have the flashlight to see. The flashlight must be walked back and forth, it cannot be thrown, etc. Each person walks at a different speed. A pair must walk together at the rate of the slower person's pace, based on this information: Person 1 takes $t_1 = 1$ minutes to cross, and the other persons take $t_2 = 2$ minutes, $t_3 = 5$ minutes, and $t_4 = 10$ minutes to cross, respectively.

Günter Rote[1] gives a very elegant solution to this puzzle.

[1] Günter Rote. *Crossing the Bridge at Night*. World Wide Web, http://page.mi.fu-berlin.de/~rote/Papers/pdf/Crossing+the+bridge+at+night.pdf, 2002

How many ways are there to let n people cross the bridge under the rules of the original puzzle ?

There are $\binom{n}{2}$ ways to send the first pair over to the other side, there are 2 ways to send the flashlight back with somebody from that side. Now there are $\binom{n-1}{2}$ ways to send the next pair over to the other side from the remaining $n-1$ people on this side and then there are 3 ways to send the flashlight back with somebody from that side etc.

Using the basic product counting principle from combinatorics we get the number of ways P to let n people cross the bridge

$$\begin{aligned} P &= \binom{n}{2} 2 \binom{n-1}{2} 3 \binom{n-2}{2} 4 \ldots (n-1) \binom{2}{2} \\ &= (n-1)! \prod_{k=0}^{n-2} \binom{n-k}{2} \end{aligned} \tag{3.1}$$

Taking the product from (3.1) and using the definition of a binomial coefficient we get:

$$\prod_{k=0}^{n-2} \binom{n-k}{2} = \prod_{k=0}^{n-2} \frac{(n-k)!}{2!(n-k-2)!} \tag{3.2}$$

With:

$$\begin{aligned} P_k &= \frac{(n-k)!}{2!(n-k-2)!} \quad \text{and} \\ p_k &= (n-k)! \end{aligned} \tag{3.3}$$

we get:

$$P_k = \frac{p_k}{2!p_{k+2}} \tag{3.4}$$

The product of these P_k can now be simplified to:

$$\begin{aligned} \prod_{k=0}^{n-2} P_k &= \prod_{k=0}^{n-2} \frac{(n-k)!}{2!(n-k-2)!} \\ &= \frac{1}{(2!)^{n-1}} \prod_{k=0}^{n-2} \frac{p_k}{p_{k+2}} \\ &= \frac{1}{2^{n-1}} \frac{p_0}{p_2} \frac{p_1}{p_3} \frac{p_2}{p_4} \cdots \frac{p_{n-3}}{p_{n-1}} \frac{p_{n-2}}{p_n} \\ &= \frac{1}{2^{n-1}} \frac{p_0 p_1}{p_{n-1} p_n} \\ &= \frac{1}{2^{n-1}} n!(n-1)! \end{aligned} \tag{3.5}$$

Using (3.5) we get the solution

$$P = \frac{n!((n-1)!)^2}{2^{n-1}} \tag{3.6}$$

For four people this comes to an astonishing 108 ways to cross the bridge under the rules of the puzzle.

Generating the ways

This section shows a small Haskell program that generates all the possible ways to cross the bridge. It has a helper function *pairs* that generates a list of all possible pairs from a set. It then defines two mutually recursive functions *bridgecrossleft* and *bridgecrossright* for crossing the bridge from the left side as pairs and for a flashlight carrier coming back from the right. The functions pass along the states on the left bank *lbs* and the right bank *rbs*. They generate all possible crossings in their respective direction given the current state. For pairs crossing from the left tuples have the respective pair and for people coming back from the right tuples have the same person in both positions of the tuple. The functions collect the resulting combinations in a list of lists of tuples *rs* (Fig. 3.1). *bridgecross* is the main function taking a list

and calling *bridgecrossleft* because we start on the left with all possible ways of crossing of the first pair.

Calling *bridgecross [1, 2, 3]* we get this result:

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

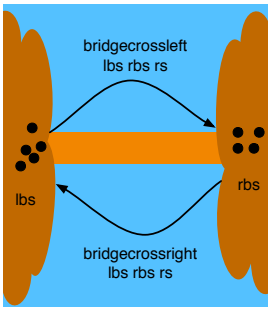


Figure 3.1: Two mutually recursive functions *bridgecrossleft* and *bridgecrossright*.

Listing 3.1: Haskell code

```
pairs :: [a] -> [(a, a)]

pairs xs = let
              rmap :: (a -> [a] -> [b]) -> [a] -> [b]

              rmap f (x:xs) = (f x xs) ++ (rmap f xs)

              rmap f [] = []

              mpairs :: (a -> [a] -> [(a, a)])

              mpairs x xs = map (\y -> (x, y)) xs

           in rmap mpairs xs

bridgecrossleft :: [Int] -> [Int] -> [(Int,Int)]
                                  -> [[(Int,Int)]]

bridgecrossleft lbs rbs rs
        = if (length lbs) >= 2 then
                let
                   ps = pairs lbs
                   f = (\(x,y) ->
                       (bridgecrossright
                          (filter (\z -> (z /= x)
                                      && (z /= y)) lbs)
                            (x:y:rbs) (rs ++ [(x, y)])))
                in foldl (++) [] (map f ps)
          else [rs]

bridgecrossright :: [Int] -> [Int] -> [(Int,Int)]
                                   -> [[(Int,Int)]]

bridgecrossright lbs rbs rs
         = if (length lbs) > 0 then
                let
                   f = (\x ->
                        (bridgecrossleft (x:lbs)
                           (filter (\z -> (z /= x)) rbs)
                               (rs ++ [(x, x)])))
                in foldl (++) [] (map f rbs)
            else [rs]

bridgecross :: [Int] -> [[(Int, Int)]]

bridgecross xs = bridgecrossleft xs [] []
```

4
Cat vs Dog

Bipartite graphs, network flows, matchings and vertex covers are the topics of the problem [1] in this note.

[1] Spotify. Cat vs dog. 2012. URL https://labs.spotify.com/puzzles/

Problem

The latest reality show has hit the TV: "Cat vs. Dog". In this show, a bunch of cats and dogs compete for the very prestigious Best Pet Ever title. In each episode, the cats and dogs get to show themselves off, after which the viewers vote on which pets should stay and which should be forced to leave the show.

Each viewer gets to cast a vote on two things: one pet which should be kept on the show, and one pet which should be thrown out. Also, based on the universal fact that everyone is either a cat lover (i.e. a dog hater) or a dog lover (i.e. a cat hater), it has been decided that each vote must name exactly one cat and exactly one dog.

Ingenious as they are, the producers have decided to use an advancement procedure which guarantees that as many viewers as possible will continue watching the show: the pets that get to stay will be chosen so as to maximize the number of viewers who get both their opinions satisfied. Calculate this maximum number of satisfied viewers.

At first glance this looks similar to a SAT problem [2], something like $(c_1 \wedge \neg d_3)$, $(c_3 \wedge \neg d_1)$, $(d_2 \wedge \neg c_2)$, ... where c_i are the cats and d_j are the dogs. The goal would be to pick the biggest subset of boolean expressions (votes) that are satisfied.

[2] Boolean satisfiability problem http://en.wikipedia.org/wiki/Boolean_satisfiability_problem

But SAT is about one boolean expression and about assigning values to boolean variables to satisfy it. Seems like SAT is fundamentally different and not a good approach in solving this problem. What if we want to visualize the boolean expressions and see the relationships between them, i.e. which ones are in conflict. Conflict between two boolean expressions means one expression has c_i and the other ex-

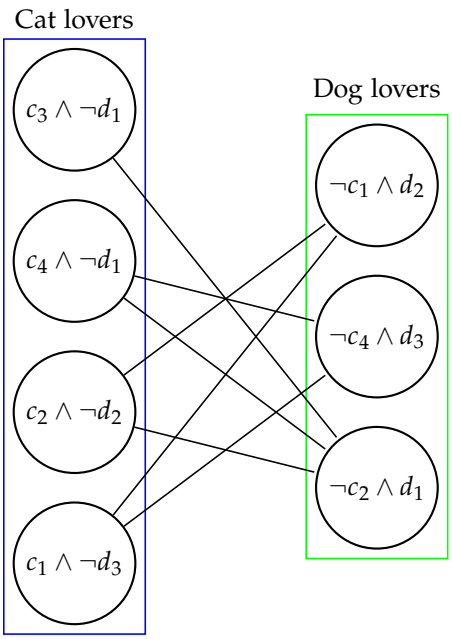


Figure 4.1: Votes form a bipartite graph. A graph is **bipartite** if the vertex set is partitioned into two subsets (blue and green in this case) such that no vertices in a subset are adjacent.

pression has $\neg c_i$ or one has d_j and the other $\neg d_j$. A good way to do that is with a graph as in Figure 4.1. The nodes in the graph are the boolean expressions and edges connect boolean expressions that are in conflict.

It becomes apparent that the graph is bipartite with cat lovers on one side and dog lovers on the other. That is good because a lot of graph algorithms are much simpler and work faster if the graphs are bipartite. But what algorithm should we use? We need to find the biggest subset of nodes in the graph that are not in conflict.

Sometimes it's easier to compute the complement of what we want: the smallest subset of nodes that are involved in conflicts. Removing these nodes and the edges they touch should leave us with a graph with only nodes and no edges, i.e. only votes without conflicts. Because we strive to remove the smallest subset of conflicting nodes we are left with the biggest subset of votes without conflicts.

The subset of nodes that are involved in conflicts is a vertex cover [3] for our bipartite graph.

We need to compute a minimum vertex cover. This will be a good excuse to learn about network flows in graphs, maximum flows and minimum cuts. This delightful detour will eventually bring us to maximum matchings[4] and then finally to minimum vertex covers.

We begin with **network flows** in graphs. We work with a directed graph $G = (V, E)$ that has two special vertices s and t called **source** and **target**. No edge goes into *source* and no edge comes out of *target*.

[3] A **vertex cover** is a subset of nodes such that each edge in a graph is incident to at least one vertex in the subset.

[4] A **matching** is a subset of edges such that no two edges in the subset share a vertex.

We also have a function $c : E \to \mathbb{R}_{\geq 0}$ that assigns a non-negative capacity to each edge. The graph G together with source s and target t and capacity function c form a **network** $(G = (V, E), s \in V, t \in V, c)$.

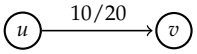


Figure 4.2: In figures we annotate an edge with flow and capacity as shown here. In this case $f(u \to v) = 10$ and $c(u \to v) = 20$. If only one number is annotating the edge then it's the capacity.

Definition 4.1. A function $f : E \to \mathbb{R}_{\geq 0}$ is a **flow** through network (G, s, t, c) if f satisfies the following constraints:

- *capacity constraint*: flow along an edge cannot exceed the capacity of the edge
$$\forall e \in E : f(e) \leq c(e)$$
- *conservation constraint*: incoming flow into a vertex (except for source and target) equals outgoing flow from the vertex
$$\forall v \in V \setminus \{s, t\} : \sum_u f(u \to v) = \sum_w f(v \to w)$$

Source s generates flow and target t consumes flow. The **value** of flow f, denoted $|f|$, is defined as

$$|f| = \sum_w f(s \to w) = \sum_v f(v \to t)$$

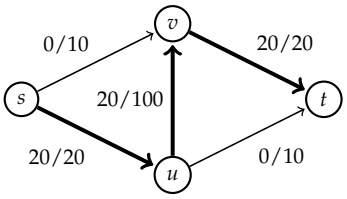


Figure 4.3: Example network flow. Here $|f| = 20$ and the whole flow is pumped along the path $s \to u \to v \to t$. In this exampe f **saturates** $s \to u$ and $v \to t$ and **avoids** $s \to v$ and $u \to t$.

Given a network (G, s, t, c) what is the maximum flow value that can be pumped through it? Figure 4.3 shows a flow of value 20 through an example network. It saturates the flow along one particular path and avoids the other edges. Is 20 the maximum flow value that can be achieved for this example network? Figure 4.4 shows the same network but now with a flow of value 30. Can we do better than 30? The answer is no, because that would exceed the outgoing capacity of source s or the incoming capacity of t.

For notational simplicity we assume functions f and c are defined on $V \times V$ and $f(u \to v) = c(u \to v) = 0$ if $u \to v$ is not an edge in $G = (V, E)$.

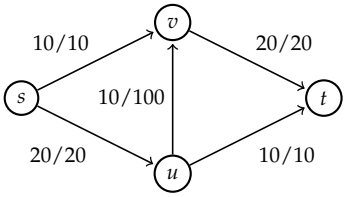


Figure 4.4: Same example network with a flow of value $|f| = 30$.

Our goal is to device an algorithm that constructs a flow with maximum value through a given network. To gauge the progress of our algorithm we need an upper bound for the maximum flow value. As said before the maximum value clearly cannot exceed the outgoing capacity of source s or the incoming capacity of t. But more generally if we sever the ties between source and target along some subset of edges such that there are no more paths from source to target then the maximum flow value cannot exceed the capacity of the cut. This seems like a useful concept to formalize.

Definition 4.2. In a network (G, s, t, c) a **cut** is a partition of the vertex set V into two subsets S and T, such that $V = S \cup T$, $S \cap T = \emptyset$ and $s \in S, t \in T$. The **capacity** of the cut (S, T), denoted $\|S, T\|$, is defined as

$$\|S, T\| = \sum_{v \in S} \sum_{w \in T} c(v \to w)$$

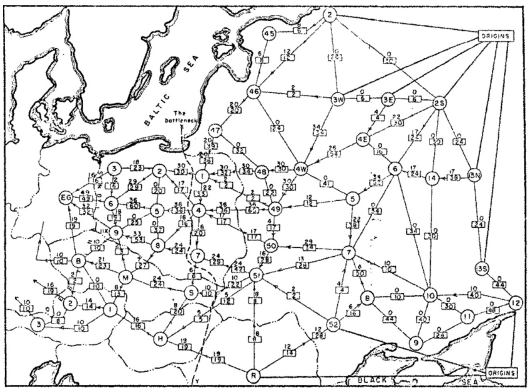


Figure 4.5: Alexander Schrijver. On the history of the transportation and maximum flow problems. 2002. URL http://homepages.cwi.nl/~lex/files/histtrpclean.pdf:
Network flows and minimum cuts played a role in the Cold War. The figure is a schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as "The bottleneck".

Theorem 4.3. *With network (G, s, t, c), for any flow f and any cut (S, T) we have*

$$|f| \leq \|S, T\|$$

Furthermore equality holds if and only if f saturates every edge from S to T and avoids every edge from T to S.

Proof.

$$\begin{aligned}
|f| &= \sum_w f(s \to w) && \text{(by definition)} \\
&= \sum_w f(s \to w) - \sum_v f(v \to s) && \text{(second sum terms are all zero)} \\
&= \sum_{u \in S} (\sum_w f(u \to w) - \sum_v f(v \to u)) && \text{(flow conservation constraint)} \\
&= \sum_{u \in S} (\sum_{w \in T} f(u \to w) - \sum_{v \in T} f(v \to u)) && \text{(edges in } S \text{ cancel each other out)} \\
&\leq \sum_{u \in S} \sum_{w \in T} f(u \to w) && \text{(because } f(v \to u) \geq 0) \\
&\leq \sum_{u \in S} \sum_{w \in T} c(u \to w) && \text{(flow capacity constraint)} \\
&= \|S, T\| && \text{(by definition)}
\end{aligned}$$

□

Theorem 4.3 tells us that if we keep increasing a flow and/or decreasing a cut we should eventually meet at a maximum flow that equals a minimum cut. But given a network how do we start? A first valid flow is $\forall e \in E : f(e) = 0$. We could then try a greedy strategy. Starting with source s find the path to t with the biggest capacity[5] and pump as much flow as we can through it as illustrated in Figure 4.3. Unfortunately we are stuck at that point. We cannot pump more flow out of s on $s \to v$ because that would violate flow conservation at v (we are at the maximum outgoing flow at v). The dashed edges in Figure 4.6 show some of our options. On edges where the current flow leaves residual capacity we can pump more and on edges where there

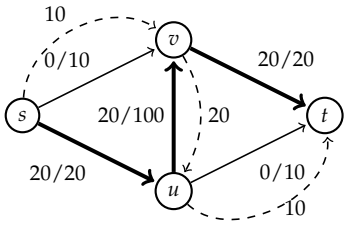


Figure 4.6: Dashed edges show how the greedy $s - t$ path flow can be augmented and reversed in order to increase overall flow.

[5] The capacity of a path is the minimum over the capacities of the edges forming the path.

is existing flow we can reverse it. Again, this concept seems worth formalizing.

Definition 4.4. A flow f in a network (G, s, t, c) induces a **residual network** (G_f, s, t, c_f) with **residual graph** G_f and **residual capacity** c_f in the following way:

- all vertices from G are vertices in G_f, also source s and target t are the same in G and G_f
- if $f(u \to v) > 0$ then G_f has an edge $(v \to u)$ with capacity

$$c_f(v \to u) = f(u \to v)$$

- if $f(u \to v) < c(u \to v)$ then G_f has an edge $(u \to v)$ with capacity

$$c_f(u \to v) = c(u \to v) - f(u \to v)$$

Figure 4.7 shows the residual network of our example network and flow. We observe that there is a simple path[6] $s \to v \to u \to t$ with capacity 10 from source s to target t in the residual graph. This path shows that there still is unused capacity for flow to be pushed from s to t. A simple path from s to t in G_f is called an **augmenting path**.

Theorem 4.5. *Given is a flow f in network (G, s, t, c). If there is an augmenting path in G_f with capacity F then the function $f' : V \times V \to \mathbb{R}_{\geq 0}$ defined as:*

$$f'(u \to v) = \begin{cases} f(u \to v) + F, & \text{if } u \to v \text{ is on the augmenting path} \\ f(u \to v) - F, & \text{if } v \to u \text{ is on the augmenting path} \\ f(u \to v), & \text{otherwise} \end{cases}$$

is a valid flow in network (G, s, t, c) with $|f'| = |f| + F$.

Proof. We need to check the capacity constraint and the conservation constraint.

Let's start with the capacity constraint. The definition of f' has three cases, so we check all three:

- Edge $u \to v$ is on the augmenting path:

$$\begin{aligned} f'(u \to v) &= f(u \to v) + F && \text{(by definition)} \\ &\leq f(u \to v) + c_f(u \to v) && \text{(by definition of } F\text{)} \\ &= f(u \to v) + c(u \to v) - f(u \to v) && \text{(by definition of } c_f\text{)} \\ &= c(u \to v) \end{aligned}$$

[6] A simple path is a path where every vertex on the path is visited only once.

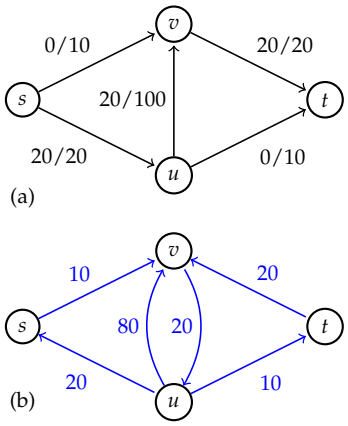


Figure 4.7:
(a) Example network with flow from Figure 4.3.
(b) Residual network (in blue) with edges annotated with their residual capacity.

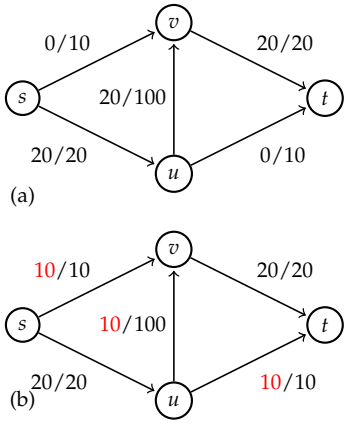


Figure 4.8:
(a) Example network with flow from Figure 4.3.
(b) Augmented flow (changed values in red) from augmenting path $s \to v \to u \to t$.

- Edge $v \to u$ is on the augmenting path:

$$\begin{aligned} f'(u \to v) &= f(u \to v) - F && \text{(by definition)} \\ &\geq f(u \to v) - c_f(u \to v) && \text{(by definition of } F\text{)} \\ &= f(u \to v) - f(u \to v) && \text{(by definition of } c_f\text{)} \\ &= 0 \end{aligned}$$

- Otherwise: In this case the flow of the edge hasn't changed so capacity constraint is satisfied.

Next is the conservation constraint. For vertices not on the augmenting path flow in and out of them hasn't changed, so conservation constraint is satisfied there. For a vertex v on the augmenting path we have four cases (since the augmenting path is simple and $v \neq s, v \neq t$):

- $u \to v$ on augmenting path and $v \to w$ on augmenting path: in this case one incoming edge into v changed by F and one outgoing changed by F, so conservation constraint holds for v
- $u \to v$ on augmenting path and $w \to v$ on augmenting path: in this case two incoming edges into v changed, one by F and the other by $-F$, so conservation constraint holds for v
- $v \to u$ on augmenting path and $w \to v$ on augmenting path: in this case one incoming edge into v changed by $-F$ and one outgoing changed by $-F$, so conservation constraint holds for v
- $v \to u$ on augmenting path and $v \to w$ on augmenting path: in this case two outgoing edges from v changed, one by $-F$ and the other by F, so conservation constraint holds for v

□

What happens when there is no augmenting path in G_f? As the Figure 4.9 hints we then have a maximum flow (in our example $|f| = 30$). The next theorem proves it.

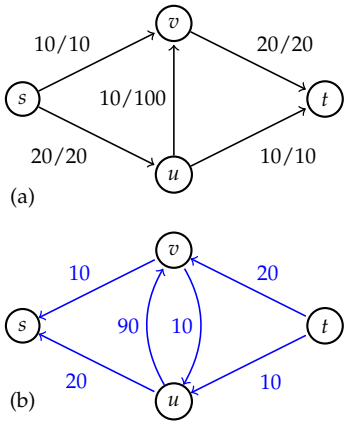


Figure 4.9:
(a) Example network with flow from Figure 4.4.
(b) Residual network (in blue) with edges annotated with their residual capacity.

Theorem 4.6. *Given is a flow f in network (G, s, t, c). If there is no augmenting path in G_f then f is a flow with maximum value.*

Proof. We define two subsets of V. The set S holds all the vertices of V that are reachable from s in G_f. Since there is no augmenting path in G_f we have $t \notin S$. We also define $T = V \setminus S$. Clearly (S, T) is a cut of our network. Also there is no G_f edge $u \to v$ with $u \in S$ and $v \in T$ because otherwise v would be reachable from somewhere in S but $v \notin S$, contradicting the definition of S. This means (by definition

of G_f) that f saturates every edge from S to T and avoids every edge from T to S. According to Theorem 4.3 we then have $|f| = \|S,T\|$ which means we have a maximum flow and minimum cut.

□

We can now piece together the following algorithm known as the **Ford-Fulkerson algorithm**:

Listing 4.1: Ford-Fulkerson algorithm

```
f = zero flow;
Gf = residual graph of f in G;

while (exists augmenting path in Gf):
   pa = choose any augmenting path;
   f = augment f with pa;
   Gf = residual graph of f in G;

return f
```

Delbert Ray Fulkerson was an American mathematician who co-developed the Ford–Fulkerson algorithm. https://en.wikipedia.org/wiki/D._R._Fulkerson

Theorem 4.7. *If the network has capacities in $\mathbb{N}_{\geq 0}$ then the Ford-Fulkerson algorithm terminates and returns the maximum flow in the network.*

Proof. We prove by induction that $f : V \times V \to \mathbb{N}_{\geq 0}$: The base case is the zero flow which is in $\mathbb{N}_{\geq 0}$. Assume the current flow values are in $\mathbb{N}_{\geq 0}$. The augmenting operation adds or subtracts a positive integer value from the current flow values and conforms to capacity constraints, so it keeps the augmented flow values in $\mathbb{N}_{\geq 0}$ which completes the induction.

The augmented flow f' modifies one outgoing edge from s by $F > 0$, so by the definition of the value of a flow we have $|f'| = |f| + F$. This means that augmenting strictly increases the value of the flow. We also know that flow values have an upper bound (by Theorem 4.3 any cut capacity is an upper bound). This means the algorithm has to eventually reach the maximum flow and terminate. □

This concludes our detour into network flows[7].
We should bring it back to our problem and the associated bipartite graph of conflicting votes. We want a minimal vertex cover and we would like to use the just derived Ford-Fulkerson algorithm to compute it. So we first have to transform our undirected bipartite graph into a network.

We have an undirected bipartite graph $G(V = X \cup Y, E)$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$ (X could be the votes of cat lovers and Y the votes of dog lovers in our problem or vice versa). We add a source s and a target t and construct a network (G', s, t, c) in the following way:

- vertex set of G' is $X \cup Y \cup \{s, t\}$
- $\forall u \in X$ add a directed edge $s \to u$ into edge set of G'
- $\forall v \in Y$ add a directed edge $v \to t$ into edge set of G'
- $\forall \{u, v\}$ undirected edge in G with $u \in X$ and $v \in Y$ add a directed edge $u \to v$ into edge set of G'
- unit capacity[8]: $\forall (u \to v) \in$ edge set of $G' : c(u \to v) = 1$

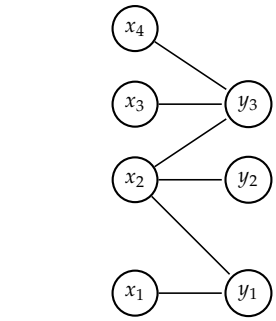
With a network (G', s, t, c) constructed from a bipartite graph G as described above (an example is shown in Figure 4.10) we have an equivalence between a matching in the bipartite graph and a flow in the network. The next theorem states this.

Theorem 4.8. *A matching M in G induces a flow f in G' such that $|f| = |M|$. Conversely a flow in G' induces a matching M in G such that $|M| = |f|$.*

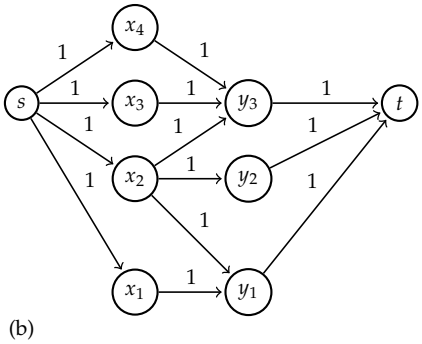
Proof. ($\Rightarrow$) We have a matching M in G, i.e. a subset of edges that don't share a vertex. From the construction of G' it follows that each of the edges in M can be extended to paths from s to t which will only meet in s and t. We define a function f that gives unit values to the edges along these paths and zero value to all other edges. We claim that f is a valid flow. It only assigns zero or unit values so it does satisfy the capacity constraint in G'. The paths don't intersect except in s and t (because M is a matching), so for any vertex along the path there is exactly one incoming edge with unit value and one outgoing edge with unit value. The rest of the edges have value zero so don't play a role in conservation. This then means that the conservation constraint is satisfied also and f is a flow. Each edge in M corresponds to one of the paths, so there are $|M|$ edges outgoing from s that have unit value. Hence $|f| = |M|$.
($\Leftarrow$) We have a flow f in G'. The flow either saturates or avoids an edge. We define the subset M of edges that are saturated by f and are

[7] We have just scratched the surface of the topic on network flows and algorithms computing maximum flows (an area of active research). For example by making smart choices when choosing the augmenting path we can improve the runtime of the algorithm (also we haven't analyzed the runtime). What happens when the capacities are not in $\mathbb{N}_{\geq 0}$. For details on all this and more see:
Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358
Jeff Erickson. *Algorithms, Etc*. 2015. URL http://jeffe.cs.illinois.edu/teaching/algorithms/.

[8] Unit capacity has the advantage that a flow either saturates the edge or avoids it.



(a)



(b)

Figure 4.10:
(a) Bipartite graph
(b) Network constructed from it.

between X and Y. We claim that M is a matching in G. Suppose it is not a matching. Then there exists a vertex that is shared by two edges in M. If this vertex is in X then it means it has two outgoing edges of unit value but only one incoming edge of unit value (from s). If this vertex is in Y then it means it has two incoming edges of unit value and only one outgoing edge of unit value (to t). In either case this is a contradiction to the conservation constraint of f. So M has to be a matching. The size of M is by its definition equal to the number of saturated edges from X to Y. But this number has to be equal to the number of edges of unit value going out of s (conservation constraint). Hence $|M| = |f|$. □

Theorem 4.8 let's us use the Ford-Fulkerson algorithm to compute a maximum matching in our bipartite graph G. Once we have a maximum matching we get the size of a minimum vertex cover with the following theorem, known as **König's theorem**[9]:

[9] For a short and elegant proof see: Romeo Rizzi. A short proof of König's matching theorem. *Journal of Graph Theory*, 33(3):138–139, 2000. URL https://math.dartmouth.edu/archive/m38s12/public_html/sources/Rizzi2000.pdf

Theorem 4.9. *In a bipartite graph G the size of a minimum vertex cover C equals the size of a maximum matching M.*

Proof. C is a vertex cover, so it covers all edges, which means it certainly covers a subset M of all edges. But M is a matching, so no two edges share a vertex. It follows that $|C| \geq |M|$.

From the maximum matching M we get the associated maximum flow f as described in Theorem 4.8. The residual graph G'_f of the associated network cannot have any augmenting paths.

We consider the minimum cut (S, T) associated with the maximum flow f. We define the following sets:

- $X_S = X \cap S, X_T = X \cap T$
- $Y_S = Y \cap S, Y_T = Y \cap T$
- $H = \{(u, v) \text{ edge in } G : u \in S, v \in T\}$
- $B = \{v \in Y_T : \exists u \in X_S \text{ with } (u, v) \text{ edge in } G\}$
- $D = X_T \cup Y_S \cup B$

D is a vertex cover: $X_T \subseteq D$ and $Y_S \subseteq D$, so D covers all edges that have endpoints in X_T or Y_S. The set B provides cover for H.

A vertex $u \in X_T$ is not reachable from s in G'_f. It means that f saturates $s \to u$ in G', so the saturated edge $s \to u$ crosses the (S, T) cut and counts towards $\|S, T\|$.

A vertex $v \in Y_S$ is reachable from s in G'_f. It means that f saturates $v \to t$ in G' (otherwise some vertex from T would be reachable from v and also from s in G'_f which is a contradiction). The saturated edge $v \to t$ crosses the (S, T) cut and counts towards $\|S, T\|$.

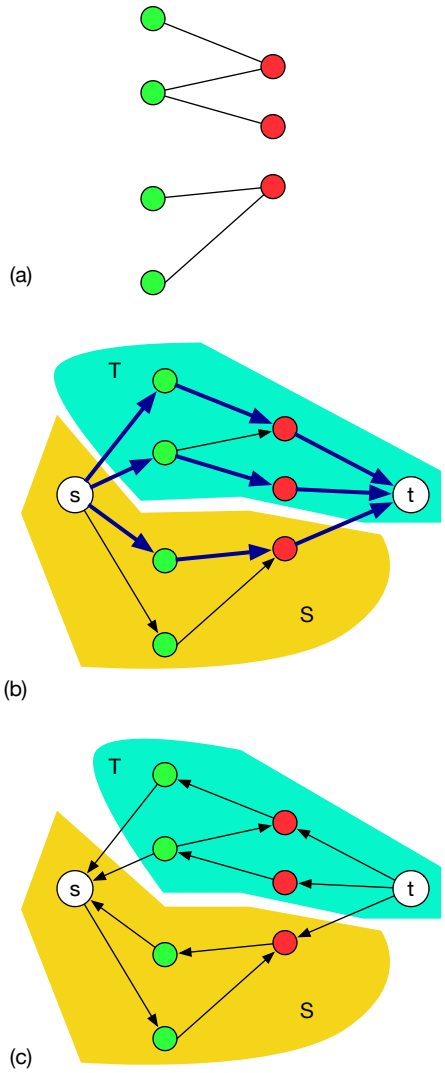


Figure 4.11:
(a) A bipartite graph $G(X \cup Y, E)$. X are green vertices, Y are red vertices.
(b) Maximum flow f (thicker arrows) and minimum cut (S, T) in the corresponding network G'. Thicker arrows between green and red vertices form the maximum matching.
(c) Same cut displayed with the corresponding residual graph G'_f.

f is a maximum flow so any edge from X_S to Y_T is saturated and counts towards $\|S, T\|$.

$$\|S, T\| = |X_T| + |Y_S| + |H|$$

Figure 4.11 shows this. In (b) there are three saturated (thick) arrows crossing the cut. The first two (counting from left to right) are due to X_T and the last one due to Y_S. In this example H is the empty set.

We have

$$|M| = |f| = \|S, T\| = |X_T| + |Y_S| + |H| \geq |X_T| + |Y_S| + |B| \geq |D|$$

D is a vertex cover and C is a minimum vertex cover, so $|D| \geq |C|$. It follows that $|C| \geq |M| \geq |D| \geq |C|$ which means $|C| = |M|$. □

This solves the problem in this note. The number of satisfied viewers is $|V| - |M|$, where V is the set of vertices in the bipartite graph G of votes with their conflicts as edges and M is a maximum matching in G computed with the Ford-Fulkerson algorithm taking advantage of the min-max duality shown in Figure 4.12.

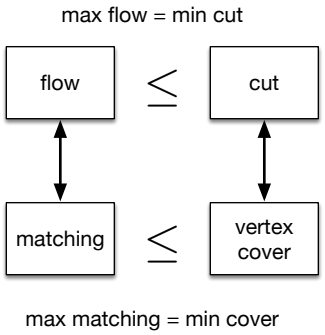


Figure 4.12: Min-max duality in bipartite graphs and corresponding networks.

5
Counting

Countable sets and counting schemes for infinite countable sets are the topics of the problem in this note.

Problem

Let $P = \{N \subset \mathbb{N} : N \text{ finite}\}$. Prove P is countable.

Let's revisit what it means for an infinite set to be countable: An infinite set M is countable if there is a bijection [1] from M to $\mathbb{N}$.

Given this definition, the problem statement is quite remarkable: the set of all the finite subsets of $\mathbb{N}$ is not "bigger" than $\mathbb{N}$.

[1] A bijection is a function that is one-to-one and onto.

Our strategy will be to start smaller and prove certain subsets of P are countable. We then expand it to P. We start by proving that the set of all subsets of $\mathbb{N}$ of size two is countable. We actually will prove something stronger, namely the set of ordered pairs of natural numbers is countable.

Theorem 5.1. *The set of ordered pairs $\mathbb{N} \times \mathbb{N}$ is countable.*

Mention puzzle 136 (Catching a Spy) from Levitin: Algorithmic Puzzles

Proof. We need a bijection from $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. There are many ways to do this [2].

The main idea we are going to use for our bijection is to order the pairs $(i, j) \in \mathbb{N} \times \mathbb{N}$ in rows, such that each pair in a row has the same value when summing the components of the pair. Figure 5.1 illustrates the idea. Row one has all pairs with components that sum up to two (in this case only one pair). Row two has all pairs with components that sum up to three, row three all pairs which sum to four, ... Notice also that in a row the pairs are sorted in increasing order of the first component.

[2] A very elegant way is described at http://www.math.upenn.edu/~wilf/website/recounting.pdf

$(1,1) \longrightarrow$ row 1

$(1,2), (2,1) \longrightarrow$ row 2

$(1,3), (2,2), (3,1) \longrightarrow$ row 3

$(1,4), (2,3), (3,2), (4,1) \longrightarrow$ row 4

...

We count the pairs from left to right in each row and go down the rows starting at the first row. For a given pair (i, j), how many pairs come before it in our counting scheme? It is in row $i + j - 1$, so there are $k : 1 \leq k < i + j - 1$ rows before it. Each row k has k pairs in it. This means there are

$$\sum_{k=1}^{i+j-2} k = \frac{(i+j-2)(i+j-1)}{2}$$

pairs in rows before our pair (i, j). There are $i - 1$ pairs before (i, j) in the same row. Therefore, our counting function is

$$f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \quad f(i,j) = i + \frac{(i+j-2)(i+j-1)}{2}$$

Suppose we have two pairs $(i_1, j_1) \neq (i_2, j_2)$. We have two cases:

- $i_1 + j_1 = i_2 + j_2$, same row, then $i_1 \neq i_2$, so $f(i_1, j_1) \neq f(i_2, j_2)$
- $i_1 + j_1 \neq i_2 + j_2$, different rows, so $f(i_1, j_1) \neq f(i_2, j_2)$

This means, f is one-to-one.

To prove that f is onto, we consider an arbitrary $n \in \mathbb{N}$ and find a pair (i, j) with $f(i, j) = n$. Working backwards and assuming we have a pair (i, j) with $f(i, j) = n$, it would fall on a row $r = i + j - 1$. In each row k there are k pairs, n is on row r, so

$$\sum_{k=1}^{r-1} k = \frac{r(r-1)}{2} < n \leq \sum_{k=1}^{r} k = \frac{r(r+1)}{2}$$

Solving for r we have:[3]

[3] Note that $\frac{1+\sqrt{1+8n}}{2} - \frac{-1+\sqrt{1+8n}}{2} = 1$

$$r^2 - r - 2n < 0, \quad r^2 + r - 2n \geq 0, \quad r = \left\lceil \frac{-1 + \sqrt{1+8n}}{2} \right\rceil$$

And then

$$i = n - \frac{r(r-1)}{2}, \quad j = r - i + 1$$

This means that given an arbitrary n, there exists a pair (i, j) with $f(i, j) = n$, so f is onto.

It follows that f is a bijection and $\mathbb{N} \times \mathbb{N}$ is countable. □

A corrollary to Theorem 5.1 let's us expand the countable subsets of P even more.

Corollary. *Set of all finite sequences of length k, $\mathbb{N}^k$ is countable.*[4]

[4] $\mathbb{N}^k$ is the set of sequences of length k, or the cartesian product $\mathbb{N} \times \mathbb{N} \times \ldots \mathbb{N}$. The set of pairs is $\mathbb{N}^2 = \mathbb{N} \times \mathbb{N}$.

Proof. Follows by induction on k: Assuming $\mathbb{N}^{k-1}$ is countable, then

$$\mathbb{N}^k = \mathbb{N}^{k-1} \times \mathbb{N}$$

is also countable according to Theorem 5.1. □

From the corrollary we now know[5] that the set of all subsets of $\mathbb{N}$ of size k is countable (it's a subset of $\mathbb{N}^k$). The problem in this section asks us to prove that P is countable, which means the union of all these countable sets is countable. The next theorem will prove just that.

[5] We keep using the fact that the set of all finite subsets of $\mathbb{N}$ of size k is a subset of the set of all sequences of size k. To see this impose an order on a set of size k and you get a sequence.

Theorem 5.2. *Let A_n, $n \in \mathbb{N}$ be countable sets. Then*

$$\bigcup_{n=1}^{\infty} A_n$$

is countable[6].

[6] Exercise 1.5.3 on page 30 from Stephen Abbott. *Understanding Analysis*. Springer, 2 edition, 2015. ISBN 978-1-4939-2711-1.

Proof. A_n is countable, so there exists a bijection $f_n : \mathbb{N} \to A_n$. We already know that $\mathbb{N} \times \mathbb{N}$ is countable, so there exists a bijection $g : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$.

We define $F : \mathbb{N} \to \bigcup_{n=1}^{\infty} A_n$

$$F(n) = f_i(j), \text{ where } (i, j) = g(n)$$

We claim that F is a bijection.

Take $n_1 \neq n_2$. Then $g(n_1) \neq g(n_2)$ and $(i_1, j_2) \neq (i_2, j_2)$, so

$$f_{i_1}(j_1) \neq f_{i_2}(j_2)$$

It means $F(n_1) \neq F(n_2)$ and F is one-to-one.

Now pick an arbitrary $a \in \bigcup_{n=1}^{\infty} A_n$. Then there exists $i \in \mathbb{N}$ with $a \in A_i$. [7] There also exists $j \in \mathbb{N}$ with $f_i(j) = a$. The pair (i, j) is in $\mathbb{N} \times \mathbb{N}$, so there exists $n \in \mathbb{N}$ with $g(n) = (i, j)$. It follows that $F(n) = a$ and F is onto. □

[7] We assume here the A_i are disjoint, if not we make them disjoint and their union stays the same.

Let us use this last theorem to prove the following statement[8]:

[8] Exercise 0.0.1 on page xiii from T. Tao. *An Introduction to Measure Theory*. Graduate Studies in Mathematics. American Mathematical Society, 2021. ISBN 9781470466404. URL https://books.google.com/books?id=k0lDEAAAQBAJ.

Theorem 5.3. *If $(x_\alpha)_{\alpha \in A}$ is a collection of numbers $(x_\alpha) \in [0, +\infty]$ such that $\sum_{\alpha \in A} x_\alpha < \infty$, then $x_\alpha = 0$ for all but at most countably many $\alpha \in A$, even if A itself is uncountable.*

Proof. We adopt the same definition of sum over the collection of numbers as in Terence Tao's book:

$$\sum_{\alpha \in A} x_\alpha = \sup\{\sum_{\alpha \in F} x_\alpha : F \subset A, F \text{ finite}\}$$

For each $n \in \mathbb{N}$ we define the subset $A_n \subset A$:

$$A_n = \{\alpha : \alpha \in A, x_\alpha \geq \frac{1}{n}\}$$

The sets A_n have to be finite because otherwise the sum $\sum_{\alpha \in A_n} x_\alpha$ would be an infinite sum of numbers not converging to zero, therefore it would diverge which is a contradiction to $\sum_{\alpha \in A} x_\alpha < \infty$.

We also know that $\bigcup_{n=1}^{\infty} A_n$ collect all the non-zero elements x_α and according to the previous theorem this union is countable. □

6
Fibolucci

Exercise 'Fibolucci' in *Programming, The Derivation of Algorithms*[1].

[1] A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990

Problem

Write a program that calculates the function

$$f(n) = \sum_{i=0}^{n} fib(i)fib(n-i), \text{ for } n \geq 0$$

where *fib* is the Fibonacci sequence defined by:

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n+2) = fib(n+1) + fib(n), \text{ for } n \geq 0$$

To solve the Fibolucci sum we adopt the same notation used in *Programming in the 1990s*[2]: The notation of function application is the "dot" notation with name of function, followed by arguments, each separated by a dot. The notation of quantified expressions has the operator followed by the bounded variables, then a colon followed by the range for the bounded variables and ended with a colon and the actual expression. So

[2] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

$$(\sum k : i \leq k < j : x_k)$$

corresponds to the more classical mathematical notation $\sum_{k=i}^{j-1} x_k$.

For our derivation steps in predicate calculus we will use the following notation:

$$
\begin{array}{ll}
 & A \\
= & \langle \text{ reason why A equals B } \rangle \\
 & B \\
\leq & \langle \text{ reason why B is less than C } \rangle \\
 & C
\end{array}
$$

We start by finding a recursive expression for f. We will use properties of quantified expressions as covered in Chapter 3 of *Programming in the 1990s*[3]. Since $fib.(0) = 0$ we can use an equivalent definition expression for f:

[3] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

$$f(n) = (\sum i : 1 \leq i < n : fib.i\ fib.(n-i))$$

We derive:

$$
\begin{array}{ll}
 & f.(n+2) \\
= & \langle \text{ definition of } f \rangle \\
 & (\sum i : 1 \leq i < n+2 : fib.i\ fib.(n+2-i)) \\
= & \langle \text{ range split, 1-point rule} \rangle \\
 & (\sum i : 1 \leq i < n+1 : fib.i\ fib.(n+2-i)) + fib.(n+1)\ fib.(1) \\
= & \langle\ fib.(1) = 1 \rangle \\
 & (\sum i : 1 \leq i < n+1 : fib.i\ fib.(n+2-i)) + fib.(n+1) \\
= & \langle \text{ definition of } fib \rangle \\
 & (\sum i : 1 \leq i < n+1 : fib.i\ (fib.(n+1-i) + fib.(n-i))) + fib.(n+1) \\
= & \langle \text{ splitting the term} \rangle \\
 & (\sum i : 1 \leq i < n+1 : fib.i\ fib.(n+1-i)) + \\
 & (\sum i : 1 \leq i < n+1 : fib.i\ fib.(n-i)) + fib.(n+1) \\
= & \langle \text{ definition of } f \rangle \\
 & f.(n+1) + (\sum i : 1 \leq i < n+1 : fib.i\ fib.(n-i)) + fib.(n+1) \\
= & \langle \text{ range split, 1-point rule, } fib.(0) = 0 \rangle \\
 & f.(n+1) + (\sum i : 1 \leq i < n : fib.i\ fib.(n-i)) + fib.(n+1) \\
= & \langle \text{ definition of } f \rangle \\
 & f.(n+1) + f.n + fib.(n+1)
\end{array}
$$

We get the recursive definition of f:

$$
\begin{aligned}
& f.0 = 0 \\
& f.1 = 0 \\
& f.(n+2) = fib.(n+1) + f.(n+1) + f.n, \text{ for } n \geq 0
\end{aligned}
$$

It is straightforward to write a program that computes f from this recursive definition, either iteratively with a loop that step by step computes next values of f starting with $f(2)$ and remembering the last two computed values of f and of *fib* for the next computations, or in Haskell by simply declaring the above recursions for f and *fib*. This will lead to a runtime of $O(n)$. But can we do better than linear ?

Let's look again at the recursive expressions of the two functions involved, leaving out the base cases and computing one additional next value:

$$
\begin{aligned}
f.(n+2) &= \mathit{fib}.(n+1) + f.(n+1) + f.n \\
f.(n+3) &= \mathit{fib}.(n+2) + f.(n+2) + f.(n+1) \\
\mathit{fib}.(n+2) &= \mathit{fib}.(n+1) + \mathit{fib}.n \\
\mathit{fib}.(n+3) &= \mathit{fib}.(n+2) + \mathit{fib}.(n+1)
\end{aligned}
$$

The key observation we can make here is that new values of the two functions are linear combinations of previously computed values. Linear combinations implies linear applications with matrix representations from linear algebra. How many previously computed values, i.e. how far back do we need to go: we need the last computed value last and the value computed before that, so 2 previous values. Looks like we could try something in a linear space of dimension 2.

Let's try first with *fib* which is simpler and doesn't depend on f. We define the function $\mathit{Fib} : \mathbb{N} \to \mathbb{N}^2$ into the two-dimensional space $\mathbb{N}^2$:

$$
\mathit{Fib}.n = \begin{pmatrix} \mathit{fib}.n \\ \mathit{fib}.(n+1) \end{pmatrix}, \text{ for } n \geq 0
$$

For a recursive expression for *Fib* we have:

$$
\begin{aligned}
& \mathit{Fib}.(n+1) \\
= \quad & < \text{definition of } \mathit{Fib} > \\
& \begin{pmatrix} \mathit{fib}.(n+1) \\ \mathit{fib}.(n+2) \end{pmatrix} \\
= \quad & < \text{definition of } \mathit{fib} > \\
& \begin{pmatrix} \mathit{fib}.(n+1) \\ \mathit{fib}.(n+1) + \mathit{fib}.n \end{pmatrix} \\
= \quad & < \text{matrix multiplication} > \\
& \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \mathit{fib}.n \\ \mathit{fib}.(n+1) \end{pmatrix} \\
= \quad & < \text{definition of } \mathit{Fib} > \\
& \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \mathit{Fib}.n
\end{aligned}
$$

So

$$
\mathit{Fib}.(n+1) = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \mathit{Fib}.n = \ldots = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} \mathit{Fib}.0
$$

The same approach can be used for f. We define a function $F : \mathbb{N} \to \mathbb{N}^4$ into the four-dimensional space $\mathbb{N}^4$:

$$F.n = \begin{pmatrix} fib.n \\ fib.(n+1) \\ f.n \\ f.(n+1) \end{pmatrix}, \text{ for } n \geq 0$$

For a recursive expression for F we have:

$$\begin{aligned}
& F.(n+1) \\
= \quad & < \text{definition of } F > \\
& \begin{pmatrix} fib.(n+1) \\ fib.(n+2) \\ f.(n+1) \\ f.(n+2) \end{pmatrix} \\
= \quad & < \text{definitions of } fib \text{ and } f > \\
& \begin{pmatrix} fib.(n+1) \\ fib.(n+1) + fib.n \\ f.(n+1) \\ f.(n+1) + f.n + fib.(n+1) \end{pmatrix} \\
= \quad & < \text{matrix multiplication} > \\
& \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} fib.n \\ fib.(n+1) \\ f.n \\ f.(n+1) \end{pmatrix} \\
= \quad & < \text{definition of } F > \\
& \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} F.n
\end{aligned}$$

and

$$F.(n+1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} F.n = \ldots = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}^{n+1} F.0$$

Calculating $F.n$ also calculates $f.n$ so if we can calculate $F.n$ faster than linear we also solve the original problem faster than linear. $F.n$ is basically an exponentiation so let's look at the exponentiation function $exp(x, n) = x^n$. The following recursive expression holds for exp:

$$exp.x.n = \begin{cases} exp.(x\ x).(n/2) & \text{if } n = 0 \bmod 2 \\ x\ exp.x.(n-1) & \text{if } n = 1 \bmod 2 \end{cases}$$

At least at every other step in the above recursion n is halved so computing $exp(x, n)$ has $O(log\ n)$ runtime which also implies $O(log\ n)$ runtime for F.

Before we write the actual code for computing F let's first see if we can find a more compact representation for the powers of matrix A involved in the computation:

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

We are searching for patterns in the powers of A:

$$A^2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 2 & 1 & 2 \end{pmatrix}, \; A^3 = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 1 & 2 \\ 2 & 5 & 2 & 3 \end{pmatrix}, \; A^4 = \begin{pmatrix} 2 & 3 & 0 & 0 \\ 3 & 5 & 0 & 0 \\ 2 & 5 & 2 & 3 \\ 5 & 10 & 3 & 5 \end{pmatrix}$$

We make the conjecture that A^k for any natural k is of the form:

$$A^k = \begin{pmatrix} a & b & 0 & 0 \\ b & a+b & 0 & 0 \\ c & d & a & b \\ e & f & b & a+b \end{pmatrix}, \text{ for some } a, b, c, d, e, f \in \mathbb{N} \tag{6.1}$$

and prove this by induction. The base case for $k = 1$ is established with values $(0, 1, 0, 0, 0, 1)$ for (a, b, c, d, e, f). Assuming that the conjecture holds for A^k we look at A^{k+1} and get:

$$A^{k+1} = A^k \, A = \begin{pmatrix} b & a+b & 0 & 0 \\ a+b & a+2b & 0 & 0 \\ d & b+c+d & b & a+b \\ f & a+b+e+f & a+b & a+2b \end{pmatrix}$$

so A^{k+1} has the same form as stated in the conjecture if we substitute $(b, a+b, d, b+c+d, f, a+b+e+f)$ for (a, b, c, d, e, f). This proves conjecture (6.1).

It means that in our program we can use a tuple representation (a, b, c, d, e, f) of 6 values instead of the whole 16 values to represent the powers of A. We need to define multiplication in this tuple space consistent with the matrix multiplication:

$$
\begin{aligned}
(a,b,c,d,e,f)(a',b',c',d',e',f') = \\
&(aa' + bb', \\
&\ ab' + b(a' + b'), \\
&\ ca' + db' + ac' + be', \\
&\ cb' + d(a' + b') + ad' + bf', \\
&\ ea' + fb' + bc' + (a + b)e', \\
&\ eb' + f(a' + b') + bd' + (a + b)f')
\end{aligned}
$$

We read this definition off the matrix multiplication:

$$
\begin{pmatrix} a & b & 0 & 0 \\ b & a+b & 0 & 0 \\ c & d & a & b \\ e & f & b & a+b \end{pmatrix}
\begin{pmatrix} a' & b' & 0 & 0 \\ b' & a'+b' & 0 & 0 \\ c' & d' & a' & b' \\ e' & f' & b' & a'+b' \end{pmatrix}
$$

The last expression we need is:

$$
A^n F.0 = \begin{pmatrix} a & b & 0 & 0 \\ b & a+b & 0 & 0 \\ c & d & a & b \\ e & f & b & a+b \end{pmatrix}
\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} =
\begin{pmatrix} b \\ a+b \\ d \\ f \end{pmatrix}
$$

so we are interested in d which corresponds to the $F.n$ coordinate of the vector.

Putting all the pieces together we get the final Haskell program:

Listing 6.1: Haskell code

```haskell
type Tuple6Ints = (Int, Int, Int, Int, Int, Int)

tmul :: Tuple6Ints -> Tuple6Ints -> Tuple6Ints

tmul (a, b, c, d, e, f) (a', b', c', d', e', f') =
    (a * a' + b * b',
     a * b' + b * (a' + b'),
     c * a' + d * b' + a * c' + b * e',
     c * b' + d * (a' + b') + a * d' + b * f',
     e * a' + f* b' + b * c' + (a + b) * e',
     e * b' + f* (a' + b') + b * d' + (a + b) * f')

fibexp :: Tuple6Ints -> Int -> Tuple6Ints

fibexp tuple n | n == 0 = error "undefined"
               | n == 1 = tuple
               | n `mod` 2 == 0 =
                   fibexp (tuple `tmul` tuple)
                          (n `div` 2)
               | n `mod` 2 == 1 =
                   tuple `tmul` (fibexp tuple (n - 1))
               | otherwise = error "wrong input"

fourth :: Tuple6Ints -> Int

fourth (a, b, c, d, e, f) = d

fibolucci :: Int -> Int

fibolucci n | n == 0 = 0
            | otherwise =
                fourth (fibexp (0, 1, 0, 0, 0, 1) n)
```

7
Grasshopper jumping

INDUCTION and integer inequalities are the topics of this note[1].

[1] For an extension to signed jumps see Géza Kós. On the grasshopper problem with signed jumps. *The American Mathematical Monthly*, 118:877–886, 2010. URL https://arxiv.org/abs/1008.2936

Problem

Let $a_1, a_2, \ldots, a_n$ be distinct positive integers and let M be a set of $n-1$ positive integers not containing $s = a_1 + a_2 + \ldots + a_n$. A grasshopper is to jump along the real axis, starting at the point 0 and making n jumps to the right with lengths $a_1, a_2, \ldots, a_n$ in some order. Prove that the order can be chosen in such a way that the grasshopper never lands on any point in M.

We use induction on n and we use the problem as our induction hypothesis with one modification: set M has at most $n-1$ elements.

The base case $n = 2$ is trivial.

Let $A = \{a_i : 1 \le i \le n\}$ and $M = \{m_i : 1 \le i < n\}$. Assume $a_1 < a_2 < \ldots < a_n$ and $m_1 < m_2 < \ldots < m_{n-1}$. For the induction step we have several cases.

Case: $a_n \in M$

There is an $l : 1 \le l < n : m_l = a_n$.

If $l = n-1$: there is an index k for which $a_k \notin M$. Then the order $\{k, n, \ldots\}$ never lands on any point in M because $a_k + a_n > m_{n-1}$.

If $l < n-1$: Define $M' = \{m_1, m_2, \ldots, m_{l-1}\} \cup \{m_{l+1} - a_n, \ldots, m_{n-1} - a_n\}$. Use integers $a_1, \ldots, a_{n-1}$ and M' as induction step to get an order $a_{\pi(1)}, \ldots, a_{\pi(n-1)}$ with $\pi \in S_{n-1}$.

$a_{\pi(1)} \notin M'$ and $a_{\pi(1)} < a_n$, so $a_{\pi(1)} \notin M$.

$a_{\pi(1)} \notin \{m_{l+1} - a_n, \ldots, m_{n-1} - a_n\}$, so $a_{\pi(1)} + a_n \notin \{m_{l+1}, \ldots, m_{n-1}\}$. Also $a_{\pi(1)} + a_n > a_n$ so $a_{\pi(1)} + a_n \notin \{m_1, m_2, \ldots, m_{l-1}\}$. That means $a_{\pi(1)} + a_n \notin M$.

We continue with similar reasoning with the rest: $a_{\pi(1)} + a_n + a_{\pi(2)} \notin M$ because $a_{\pi(1)} + a_{\pi(2)} \notin \{m_{l+1} - a_n, \ldots, m_{n-1} - a_n\}$, so $a_{\pi(1)} + a_n +$

$a_{\pi(2)} \notin \{m_{l+1}, \ldots, m_{n-1}\}$ and $a_{\pi(1)} + a_n + a_{\pi(2)} > a_n$ etc.

This means $\{\pi(1), n, \pi(2), \ldots, \pi(n-1)\}$ is a valid order.

Case: $a_n \notin M$

If there is an $m_i < a_n$ then we can use the induction step with integers $a_1, a_2, \ldots, a_{n-1}$ and set $M' = \{m_{i+1} - a_n, m_{i+2} - a_n, \ldots, m_{n-1} - a_n\}$ to find an order and prepend a_n to that order.

If not, then $\forall\, 1 \leq i < n : m_i > a_n$.

$\sum_{j=1}^{n-1} a_j \geq m_1$ because otherwise we could have used order $\{1, 2, \ldots, n\}$.

We have $a_1 < a_n < m_1$ and $\sum_{j=1}^{n-1} a_j \geq m_1$, so there exists an $1 \leq l < n-1$ such that $s' = \sum_{j=1}^{l} a_j < m_1$.

Define $M' = \{m_2 - a_n, m_3 - a_n, \ldots, m_{n-1} - a_n\}$ and use M' with the integers $a_1, a_2, \ldots, a_{n-1}$ in an induction step which gives us an order $\pi \in S_{n-1}$.

Since $a_{\pi(1)} < m_1$ and $\sum_{j=1}^{n-1} a_{\pi(j)} \geq m_1$ there exists an $1 < l \leq n-1$ such that $\sum_{j=1}^{l-1} a_{\pi(j)} < m_1$ and $\sum_{j=1}^{l} a_{\pi(j)} \geq m_1$.

We look at the order $\{\pi(1), \ldots, \pi(l-1), n, \pi(l), \ldots, \pi(n-1)\}$ and claim it is a valid order.

Indeed $\sum_{j=1}^{l-1} a_{\pi(j)} < m_1$, so jumps $\{\pi(1), \ldots, \pi(l-1)\}$ won't encounter anything from M. We also have

$$\sum_{j=1}^{l-1} a_{\pi(j)} + a_n > \sum_{j=1}^{l} a_{\pi(j)} \geq m_1$$

which means $\{\pi(1), \ldots, \pi(l-1), a_n\}$ will avoid m_1. It will also avoid anything from $M \setminus \{m_1\}$ because $\{\pi(1), \ldots, \pi(l-1)\}$ avoids anything from M'. The rest of the order is already bigger than m_1 and avoids $M \setminus \{m_1\}$ by induction.

8
Groovy numbers

Problem

$x \in \mathbb{R}$ is said to be a groovy number iff $\exists\ n \in \mathbb{N}$ such that $x = \sqrt{n} + \sqrt{n+1}$. Prove that if x is groovy, then $\forall r \in \mathbb{N} :\ x^r$ is groovy.

Binomial Expansion

In this section we explore a property of the binomial power expansion

$$(a+b)^r = \sum_{k=0}^{r} \binom{r}{k} a^{r-k} b^k$$

We define $\mathbb{N}_r = \{k \in \mathbb{N}_0 : 0 \leq k \leq r\}$ and its partition into two subsets $\mathbb{N}_r = \mathbb{E}_r \cup \mathbb{O}_r$, with $\mathbb{E}_r = \{k \in \mathbb{N}_r : k = 2u, u \in \mathbb{N}_0\}$ and $\mathbb{O}_r = \{k \in \mathbb{N}_r : k = 2u+1, u \in \mathbb{N}_0\}$. We then partition the binomial power expansion into two sums:

$$(a+b)^r = \sum_{k=0}^{r} \binom{r}{k} a^{r-k} b^k = \sum_{k \in \mathbb{E}_r} \binom{r}{k} a^{r-k} b^k + \sum_{k \in \mathbb{O}_r} \binom{r}{k} a^{r-k} b^k$$

Let

$$E(a,b,r) = \sum_{k \in \mathbb{E}_r} \binom{r}{k} a^{r-k} b^k \text{ and } O(a,b,r) = \sum_{k \in \mathbb{O}_r} \binom{r}{k} a^{r-k} b^k$$

Then

$$\begin{aligned}(a^2-b^2)^r &= (a+b)^r (a-b)^r \\ &= (E(a,b,r) + O(a,b,r))(E(a,-b,r) + O(a,-b,r))\end{aligned}$$

But

$$E(a,-b,r) = E(a,b,r) \text{ and } O(a,-b,r) = -O(a,b,r)$$

so

$$\begin{aligned}(a^2-b^2)^r &= (a+b)^r(a-b)^r \\ &= (E(a,b,r)+O(a,b,r))(E(a,-b,r)+O(a,-b,r)) \\ &= (E(a,b,r)+O(a,b,r))(E(a,b,r)-O(a,b,r)) \\ &= E(a,b,r)^2 - O(a,b,r)^2\end{aligned}$$

We therefore proved

Lemma 8.1.

$$(a^2-b^2)^r = E(a,b,r)^2 - O(a,b,r)^2$$

Solution

Using lemma 8.1 with $a=\sqrt{n}$ and $b=\sqrt{n+1}$, we get

$$(-1)^r = E(\sqrt{n},\sqrt{n+1},r)^2 - O(\sqrt{n},\sqrt{n+1},r)^2 \qquad \text{(L)}$$

Lemma 8.2.

$$\begin{aligned}&E(\sqrt{n},\sqrt{n+1},r)^2 \in \mathbb{N}, \\ &O(\sqrt{n},\sqrt{n+1},r)^2 \in \mathbb{N}\end{aligned}$$

Proof. We will look at two cases: r even and r odd.
Case 1. For $r=2u$ even we have

$$\begin{aligned}E(\sqrt{n},\sqrt{n+1},2u) &= \sum_{k=0}^{u}\binom{2u}{2k}(\sqrt{n})^{2u-2k}(\sqrt{n+1})^{2k} \\ &= \sum_{k=0}^{u}\binom{2u}{2k}(\sqrt{n})^{2(u-k)}(\sqrt{n+1})^{2k} \\ &= \sum_{k=0}^{u}\binom{2u}{2k}n^{u-k}(n+1)^k\end{aligned}$$

so $E(\sqrt{n},\sqrt{n+1},r)\in\mathbb{N}$, and therefore $E(\sqrt{n},\sqrt{n+1},r)^2\in\mathbb{N}$.

$$
\begin{aligned}
O(\sqrt{n},\sqrt{n+1},2u) &= \sum_{k=0}^{u-1}\binom{2u}{2k+1}(\sqrt{n})^{2u-2k-1}(\sqrt{n+1})^{2k+1} \\
&= \frac{\sqrt{n+1}}{\sqrt{n}}\sum_{k=0}^{u-1}\binom{2u}{2k+1}(\sqrt{n})^{2(u-k)}(\sqrt{n+1})^{2k} \\
&= \frac{\sqrt{n+1}}{\sqrt{n}}\sum_{k=0}^{u-1}\binom{2u}{2k+1}n^{2(u-k)}(n+1)^{k} \\
&= \sqrt{n(n+1)}\sum_{k=0}^{u-1}\binom{2u}{2k+1}n^{2(u-k)-1}(n+1)^{k}
\end{aligned}
$$

so $O(\sqrt{n},\sqrt{n+1},r)^2 \in \mathbb{N}$.
Case 2. $r = 2u+1$ is handled in a similar fashion by factoring out $\sqrt{n}$ and $\sqrt{n+1}$ with the remainder $\in \mathbb{N}$.

□

From lemma 8.2 and equation (L) it follows that $E(\sqrt{n},\sqrt{n+1},r)^2$ and $O(\sqrt{n},\sqrt{n+1},r)^2$ are consecutive natural numbers. Let

$$m = min(E(\sqrt{n},\sqrt{n+1},r)^2, O(\sqrt{n},\sqrt{n+1},r)^2) \in \mathbb{N}$$

Then

$$x^r = (\sqrt{n}+\sqrt{n+1})^r = \sqrt{m}+\sqrt{m+1}$$

9
Devil's chessboard

Hamming codes are used to solve the problem[1] in this note.

You, your friend, and the Devil play a game. You and the Devil are in the room with a chess board with 64 tokens on it, one on each square. Meanwhile, your friend is outside of the room. The token can either be on an up position or a down position, and the difference in position is distinguishable to the eye. The Devil mixes up the positions (up or down) of the tokens on the board and chooses one of the squares and calls it the magic square. Next, you may choose one token on a square and flip its position. Then, your friend comes in and must guess what the magic square was by looking on the squares on the board.[2]

[1] Michael Tong. Devil's chessboard. 2013. URL https://brilliant.org/discussions/thread/the-devils-chessboard/

[2] Details:

1. You **may** flip a token. As in, you are not forced to flip a token; you may choose to not flip a token.
2. You can't just tell your friend what square it is. Or point to it. Or text him it. Or... you get the point.
3. Your friend knows the strategy as well (you tell him beforehand).
4. If you don't get it right, the Devil takes your soul. High stakes.

Problem

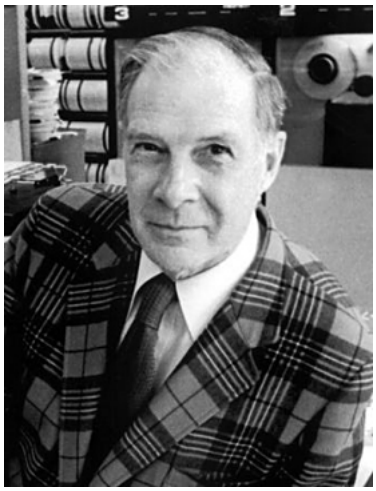
Show that there is a winning strategy such that your friend can always know what square the magic square is.

There might be solutions that exploit the chessboard geometry with its black and white fields. We will ignore the chessboard angle though and use this problem as an excuse to dive into the topic of linear codes. We will solve the problem by treating the token information as a 64-bit word and we will devise a winning strategy that involves a Hamming[3] code (a type of perfect linear code).

But first lets introduce linear codes. We operate in the field $\mathbb{F}_q$ of integers modulo a prime q.

Definition 9.1. A **linear code** C of words of length n is a subspace of the vector space $\mathbb{F}_q^n$. Let $\dim C = k$, then we say that C is a $[n,k]_q$ linear code.

Given a basis $\{\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}\}$ of C, we can build a matrix $G \in \mathbb{F}_q^{k\times n}$



[3] Richard Hamming was one of the founders of modern coding theory. http://en.wikipedia.org/wiki/Richard_Hamming

using the $\boldsymbol{c_i}$ basis vectors as rows. Then C is the row space of G and G is called a **generator matrix** of C. We have[4]

$$C = \{\boldsymbol{x}G : \boldsymbol{x} \in \mathbb{F}_q^k\},$$

so a code C is made from all linear combinations of the row vectors of its generator matrix.

Let G' be the row reduced echelon form of G. By definition G has full row rank, so G' has only nonzero rows. If $G' = \left[\, I_k \mid A_{k\times(n-k)} \,\right]$ for identity matrix I_k and some matrix A then the generator matrix G' is in **standard form**[5]. Row operations preserve the row space, so G' also generates C.

Definition 9.2. Given a $[n,k]_q$ linear code C, matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ is a **parity check matrix** for C, if $C = nullspace(H) = \{\boldsymbol{c} \in \mathbb{F}_q^n : H\boldsymbol{c}^T = 0\}$.

Theorem 9.3. *Given a $[n,k]_q$ linear code C and a generator matrix $G = \left[\, I_k \mid A_{k\times(n-k)} \,\right]$ for C in standard form, then $H = \left[\, -A^T_{(n-k)\times k} \mid I_{n-k} \,\right]$ is a parity check matrix[6] for C.*

Proof. Let $\boldsymbol{c} \in C$ be a code word from C. Then there exists an $\boldsymbol{x} \in \mathbb{F}_q^k$ such that $\boldsymbol{c} = \boldsymbol{x}G$. We have

$$\begin{aligned}
H\boldsymbol{c}^T &= H(\boldsymbol{x}G)^T \\
&= \left[-A^T_{(n-k)\times k} | I_{n-k}\right] \left(\boldsymbol{x}\left[I_k | A_{k\times(n-k)}\right]\right)^T \\
&= \left[-A^T_{(n-k)\times k} | I_{n-k}\right] \left[\frac{I_k}{A^T_{(n-k)\times k}}\right] \boldsymbol{x}^T \\
&= (-A^T + A^T)\boldsymbol{x}^T \\
&= 0
\end{aligned}$$

This means that $C \subseteq nullspace(H)$. We have $\dim C = k$ and

$$\dim nullspace(H) = n - rank(H) = n - n + k = k,$$

so $C = nullspace(H)$ and H is a parity check matrix for C.

□

What can we do if the generator matrix is not in standard form? Swapping columns in the generator matrix does not preserve the row space, so the linear code generated with the modified matrix is clearly not the same as the original code, but it is an equivalent code[7].

Definition 9.4. The **Hamming distance** $d(\boldsymbol{x},\boldsymbol{y})$ between two vectors $\boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_q^n$ is the number of positions in which the vectors differ. With $\boldsymbol{x} = x_1x_2\ldots x_n$ and $\boldsymbol{y} = y_1y_2\ldots y_n$ we have

$$d(\boldsymbol{x},\boldsymbol{y}) = |\{i : 1 \le i \le n : x_i \neq y_i\}|$$

[4] We treat vectors as row vectors in this section. That means that $\boldsymbol{x} \in \mathbb{F}_q^k$ is a matrix $\mathbb{F}_q^{1\times k}$.

[5] Not every generator matrix can be row reduced to the standard form. For example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

cannot.

[6] With G in standard form this theorem let's us construct a parity check matrix very easily. Also worth noting that in standard form we generate a code word from a message $\boldsymbol{x} \in \mathbb{F}_q^k$ by appending $n-k$ parity check bits to the message with $\boldsymbol{x}G$. We check if the transmitted and received word $\boldsymbol{y} \in \mathbb{F}_q^n$ is a valid code word by verifying $H\boldsymbol{y}^T = 0$. If true then the first k positions of $\boldsymbol{y}$ are the original message $\boldsymbol{x}$.

[7] A $[n,k_1]_q$ linear code C_1 is equivalent to a $[n,k_2]_q$ linear code C_2 if there is a permutation $\pi \in S_n$ such that when π is applied to the coordinate indices of all the code words from C_1, it produces all the code words from C_2. Equivalent linear codes have the same dimension $k_1 = k_2$.

The **Hamming weight** $w(\boldsymbol{x})$ is the number of positions that differ from zero:

$$w(\boldsymbol{x}) = |\{i : 1 \leq i \leq n : x_i \neq 0\}| = d(\boldsymbol{x}, \boldsymbol{0})$$

We will use the following properties of Hamming distances:

Lemma 9.5.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{x}) \geq 0$$
$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow x = y$$
$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$$
$$\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$$

Proof of Lemma 9.5
The first three properties are obvious from the definition of Hamming distance. For the last property let i be an index where $\boldsymbol{x}$ and $\boldsymbol{y}$ differ, so $x_i \neq y_i$. For vector $\boldsymbol{z}$ we can have the following cases for position i:

$$z_i = x_i \Rightarrow z_i \neq y_i$$
$$z_i = y_i \Rightarrow z_i \neq x_i$$
$$z_i \neq x_i \wedge z_i \neq y_i$$

In each of these cases the contribution of z_i to $d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$ is at least one, whereas on the left side position i contributes one to $d(\boldsymbol{x}, \boldsymbol{y})$. A similar analysis holds for indices i where $x_i = y_i$. □

Definition 9.6. The **minimum distance** of C is:

$$d(C) = \min\{d(\boldsymbol{x}, \boldsymbol{x'}) : \boldsymbol{x}, \boldsymbol{x'} \in C \wedge \boldsymbol{x} \neq \boldsymbol{x'}\} = \min\{w(\boldsymbol{x}) : \boldsymbol{x} \in C\}$$

The minimum distance is important enough that we add it to the characteristic notation of a linear code: $[n, k, d]_q$ is a linear code over field $\mathbb{F}_q$ with bit strings of length n, code dimension k and minimum distance between code words d.

The next lemma establishes a connection between the minimum distance of a linear code and one of its parity check matrix.

Lemma 9.7. *The minimum distance of a code C equals the minimum number of linearly dependent columns in one of its parity check matrices.*

Proof of Lemma 9.7
Let H be a parity check matrix of $[n, k, d]_q$ linear code C. There must be a code word $\boldsymbol{c}$ with $w(\boldsymbol{c}) = d$. $\boldsymbol{c}$ belongs to the nullspace of H, so

$$H\boldsymbol{c}^T = \boldsymbol{0}$$

But $H\boldsymbol{c}^T$ is a linear combination of column vectors of H, with d nonzero coefficients, so the column vectors in this linear combination are linearly dependent. □

So far we have worked with fields $\mathbb{F}_q$ of any prime q. Now we switch to the binary world $q = 2$ and $\mathbb{F}_2$. Our vectors are bit strings. We transmit these bit strings over a binary symmetric channel.

Definition 9.8. In a **binary symmetric channel** each bit sent has the same probability $p < \frac{1}{2}$ of being received incorrectly.

We send a code word $\boldsymbol{x} \in C$ from a $[n, k]_2$ linear code C over a binary symmetric channel and receive a bit string $\boldsymbol{y}$. If there were no transmission errors, then $y = x$. If there were errors, we want to find the most likely code word $\boldsymbol{x}$ that was transmitted given the errors in $\boldsymbol{y}$.

One decoding strategy[8] would be to choose a code word $\boldsymbol{x}$ with minimum Hamming distance over all code words from C to received bit string $\boldsymbol{y}$. This type of decoding is called *nearest neighbor decoding*. The chosen $\boldsymbol{x}$ is not always unique.

[8] Finding an appropriate code word for the transmitted bit string is called *decoding*. Finding the most likely code word is called *maximum likelihood decoding*.

Theorem 9.9. *In a binary symmetric channel with error probability $p < \frac{1}{2}$ the nearest neighbor decoding is a maximum likelihood decoding.*

Proof. Given a bit string $\boldsymbol{y} \in \mathbb{F}_2^n$ received through the channel, let $P_{\boldsymbol{y}}(\boldsymbol{x})$ be the probability that the code word $\boldsymbol{x}$ was sent when $\boldsymbol{y}$ was received. Because the channel is a binary symmetric channel, we have

$$P_y(x) = p^{d(x,y)}(1-p)^{n-d(x,y)}$$

Consider two code words x and x' such that $d(x,y) \leq d(x',y)$. Because $p < \frac{1}{2}$, we then have $P_y(x) \geq P_y(x')$. It follows that

$$\max_{x \in C} P_y(x) = \min_{x \in C} d(x,y)$$

so the likeliest code word is the nearest neighbor to y. □

For the rest of this section we use nearest neighbor decoding. We want to know if we can detect and possibly correct a transmission with errors. Let's define clearly what we mean by that. A transmission is a pair $(x,y) \in C \times \mathbb{F}_2^n$, where a code word x was sent and a bit strings y was received. It has $d(x,y)$ transmission errors. The nearest neighbor decoding $nnd(y)$ finds a code word (not necessarily unique) closest to y. The following holds by definition:

$$d(y, nnd(y)) = \min_{c \in C} d(y,c)$$

If no errors occurred in the transmission, then $x = y$ and also $d(x,y) = 0$ and $nnd(y) = x$. If errors in the transmission occurred we want to:

E.1 detect that errors happened, i.e. establish that $y \notin C$.

E.2 correct the errors, i.e. establish $nnd(y) = x$.

The next theorem describes the conditions for **E.1**.

Theorem 9.10. *Given a $[n,k,d]_2$ linear binary code C, we can detect that any transmission with up to e errors was erroneous if and only if $d > e$.*

Proof. ($\Rightarrow$) Let (x,y) be a transmission with $d(x,y) \leq e < d$ errors. Assume $y \in C$. Then $d(x,y) \leq e < d$ is a contradiction to d being the minimal distance of C. It follows that $y \notin C$.
($\Leftarrow$) We can detect that any transmission with up to e errors was erroneous. Assume $d \leq e$. Then there exist two code words $x \neq x'$ such that $d(x,x') \leq e$. Now consider transmission (x,x'). It's impossible to detect that it had errors because x' is a code word. This is a contradiction with the fact that we can detect that any transmission with up to e errors was erroneous. So $d > e$. □

For **E.2** we have this theorem:

Theorem 9.11. *Given a $[n,k,d]_2$ linear binary code C, we can correct any transmission with up to e errors if $d > 2e$.*

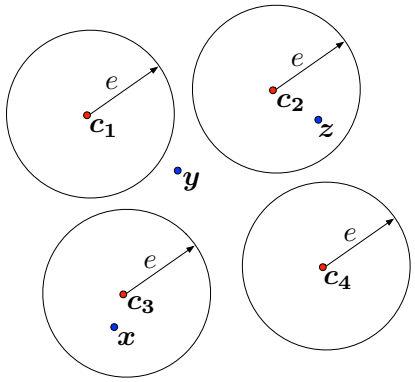


Figure 9.1: A Hamming sphere for code word c with radius e is the set $\{x : d(x,c) \leq e\}$. In this figure the spheres don't overlap, so vectors (blue dots) that fall within a sphere can be error-corrected to code words (red dots).

Proof. Let $(\boldsymbol{x}, \boldsymbol{y})$ be a transmission with $d(\boldsymbol{x}, \boldsymbol{y}) \leq e$ errors and $d > 2e$. Assume $nnd(\boldsymbol{y}) \neq \boldsymbol{x}$. Then $d(\boldsymbol{y}, nnd(\boldsymbol{y})) \leq e$ (otherwise $\boldsymbol{x}$ would be closer than $nnd(\boldsymbol{y})$ to $\boldsymbol{y}$). We have

$$d(\boldsymbol{x}, nnd(\boldsymbol{y})) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, nnd(\boldsymbol{y})) \leq e + e = 2e$$

which contradicts $d > 2e$. So $nnd(\boldsymbol{y}) = \boldsymbol{x}$.

□

Theorems 9.10 and 9.11 tell us that a large minimum distance $d(C)$ allows us to detect and correct more errors. But a large minimum distance between code words also limits the number of code words. The following theorem puts an upper bound on the number of code words given a minimum distance.

Theorem 9.12. *Given a $[n, k, 2t+1]_2$ linear binary code C, we have*

$$|C| \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}}$$

*This upper bound is called **Hamming bound**.*

Proof. Given a bit string $\boldsymbol{x}$ and an integer $i \leq n$, there are $\binom{n}{i}$ ways to choose the i positions at which $\boldsymbol{x}$ and and another bit string $\boldsymbol{y}$ differ. So there are $\binom{n}{i}$ bit strings $\boldsymbol{y}$ with $d(\boldsymbol{x}, \boldsymbol{y}) = i$. This means there are

$$\sum_{i=0}^{t} \binom{n}{i}$$

bit strings $\boldsymbol{y}$ with $d(\boldsymbol{x}, \boldsymbol{y}) \leq t$.

On the other hand, a bit string $\boldsymbol{y}$ with $d(\boldsymbol{y}, \boldsymbol{x}) \leq t$ to a code word $\boldsymbol{x}$ cannot have the same $d(\boldsymbol{y}, \boldsymbol{x'}) \leq t$ to a different code word $\boldsymbol{x'}$ because then

$$d(\boldsymbol{x}, \boldsymbol{x'}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{x'}) \leq t + t \leq 2t$$

which is a contradiction to $d(C) = 2t + 1$.

So for each code word, we have at most $\sum_{i=0}^{t} \binom{n}{i}$ bit strings with Hamming distance $\leq t$ and we cannot have the same bit strings near two different code words. We have 2^n bit strings, so

$$|C| \sum_{i=0}^{t} \binom{n}{i} \leq 2^n$$

□

A binary linear code that achieves equality in the Hamming bound 9.12 is called a **perfect code**.

We are now ready to define Hamming codes.

Definition 9.13. A Hamming code $\mathfrak{H}_r$ of order r (where r is a positive integer) is a binary linear code with the parity check matrix with columns that are all the $2^r - 1$ nonzero bit strings of length r.

Changing the order of the columns in the parity check matrix produces equivalent codes with the same minimum distance. So for easier analysis we now consider Hamming codes with parity check matrix in standard form, ie the last r columns form the identity matrix I_r, so $H = \left[A_{r\times(n-r)} \mid I_r \right]$, with $n = 2^r - 1$. From theorem 9.3 we then know the generator matrix is $G = \left[I_{n-r} \mid -A^T_{(n-r)\times r} \right] = \left[I_{n-r} \mid A^T_{(n-r)\times r} \right]$, since we operate in $\mathbb{F}_2$. We can see that $\dim \mathfrak{H}_r = n - r$. What is the minimum distance of $\mathfrak{H}_r$? All columns are nonzero and distinct, so no two columns are linearly dependent[9]. But consider the linear combination of the three columns

$$[1, 1, 0, \ldots, 0]^T + [1, 0, 0, \ldots, 0]^T + [0, 1, 0, \ldots, 0]^T = \mathbf{0}^T$$

They are linearly dependent. From lemma 9.7 it follows that $d(\mathfrak{H}_r) = 3$, so $\mathfrak{H}_r$ is a $[2^r - 1, 2^r - 1 - r, 3]_2$ binary linear code. According to theorem 9.11 it can correct transmissions with one error.

[9] Again, this is in $\mathbb{F}_2$. The sum of two distinct columns is always nonzero, so a linear combination that is zero has to have coefficients zero, hence linearly independent.

Theorem 9.14. *$\mathfrak{H}_r$ is a perfect code.*

Proof. The generator matrix has full row rank, so we need all linear combinations of the rows to get all the code words. This are binary words, so there are 2^{n-r} distinct linear combinations. It means $|\mathfrak{H}_r| = 2^{n-r}$.

Inserting into formula of theorem 9.12, we get[10]

$$2^{n-r} \sum_{i=0}^{1} \binom{n}{i} = 2^{n-r}(1 + n) = 2^{n-r}(1 + 2^r - 1) = 2^{n-r}2^r = 2^n$$

□

[10] With $t = 1$, because $d(\mathfrak{H}_r) = 3$.

This concludes our dive into linear codes and Hamming codes. Let's return to our problem and solve it using Hamming codes. The state of the chessboard is a binary word of length 64. We use $r = 6$, so Hamming code $\mathfrak{H}_6$. The word length is $2^6 - 1 = 63$. We agree that the devil choosing bit 64 is a special case which we handle later. For now imagine the chessboard as a 63-bit binary word and the devil only choosing a magic field between 1 and 63.

The winning strategy can be summarized as follows: the first player needs to modify the 63-bit word (by flipping at most one bit) in such a way that the magic field is the one bit error of a code word in $\mathfrak{H}_6$. Then the second player only has to come in, decode[11] the modified chessboard and point to the corrected error which is the same magic field.

Is this always possible? We know that Hamming codes are perfect codes, so any 63-bit word is at most one bit away from a code word. We have the following cases for the initial state of the chessboard:

- It happens to be a code word in $\mathfrak{H}_6$. Then the first player flips the magic field bit, producing an error there.
- It happens to be a 63-bit word that is a one bit error at the magic field. The first player doesn't flip any bit in this case.
- It happens to be a 63-bit word with a one bit error different from the magic field.

The last case needs a little thinking. Assume H is the parity check matrix for our $\mathfrak{H}_6$ Hamming code and assume the state of the chessboard is $\boldsymbol{x}$, which is one bit error from a code word $\boldsymbol{c_1}$. Also let $1 \leq m \leq 63$ be the magic field bit and $\boldsymbol{e_m}$ the unit vector with bit m set. The one bit error is different from the magic field, so $\boldsymbol{x} - \boldsymbol{c_1} \neq \boldsymbol{e_m}$. Let $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{e_m}$, which is also one bit away from a code word $\boldsymbol{c_2}$, with error bit k. So $\boldsymbol{y} = \boldsymbol{c_2} + \boldsymbol{e_k}$.

Now consider $\boldsymbol{x} - \boldsymbol{e_k}$:

$$H(\boldsymbol{x} - \boldsymbol{e_k}) = H(\boldsymbol{y} + \boldsymbol{e_m} - \boldsymbol{e_k}) = H(\boldsymbol{y} - \boldsymbol{e_k}) + H\boldsymbol{e_m} = H\boldsymbol{c_2} + H\boldsymbol{e_m} = H\boldsymbol{e_m}$$

So $\boldsymbol{x} - \boldsymbol{e_k}$ has one bit error at the magic field, which is what we want. Flipping bit k on the initial chessboard $\boldsymbol{x}$ achieves that.

In all three cases the modified chessboard is one bit away from a code word with the error at the magic field and the chessboard was modified by flipping at most one bit. The players agree that if the chessboard is a code word instead, then the devil chose bit 64 as the magic field, which handles the special case. Modifying the chessboard to get a code word can also be done by flipping at most one bit. This scales to any chessboard with size a power of two.

[11] Decoding is done as follows: $\boldsymbol{x}$ needs to be decoded. It is one bit away from a code word $\boldsymbol{c}$ with error at bit k. Let $\boldsymbol{e_k}$ be the unit vector with bit k set. So $\boldsymbol{x} = \boldsymbol{c} + \boldsymbol{e_k}$ and

$$H\boldsymbol{x} = H(\boldsymbol{c} + \boldsymbol{e_k}) = H\boldsymbol{e_k}$$

Since $\boldsymbol{e_k}$ is be the unit vector with bit k set, $H\boldsymbol{e_k}$ is column k from the parity check matrix H. To decode we calculate $H\boldsymbol{x}$ and look to see which column in H the result is. To save the lookup step we can be even more elegant. Instead of the parity check matrix in standard form, we choose a parity check matrix where column k is the bit representation of k. Instead of lookup we just reverse the bit representation back to the integer k.

What follows is a Mathematica session illustrating the strategy. We use a parity check matrix with column k the bit representation of integer k. This simplifies decoding as remarked in the side note 11 above.

The function *hamming* generates the parity check matrix for a Hamming code with a given r.

```
In[1]:= hamming[r_Integer] := Transpose[Table[IntegerDigits[i, 2, r], {i, 1, 2^r - 1}]]
```

For example

```
In[2]:= hamming[4] // MatrixForm
```

```
Out[2]=
```

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

We define r and the corresponding Hamming code h for our chessboard

```
In[3]:= r = 6;
```

```
In[4]:= h = hamming[6];
```

The function *pos* returns the unit vector with the error bit set from decoding the specified word.

```
In[5]:= pos[w_] := With[{s = Mod[h.w, 2]}, UnitVector[2^Length[s] - 1, FromDigits[s, 2]]]
```

Function *friendOne* implements the strategy part for the first friend. Given the initial state of the chessboard *cb* and a magic field *mf*, it returns a modified chessboard.

```
In[6]:= friendOne[cb_List, mf_Integer] := Module[{em, y, ey},
        em = UnitVector[2^r - 1, mf]; y = Mod[cb - em, 2]; ey = pos[y];
        z = Mod[cb - ey, 2]
        ]
```

Function *friendTwo* implements the strategy part for the second friend: decoding the specified chessboard and returning the index of the error bit which is also the magic field.

```
In[7]:= friendTwo[cb_List] := Position[pos[cb], 1][[1,1]]
```

This next function is returning random initial states for the chessboard.

```
In[8]:= rw := RandomInteger[1, {2^r - 1}]
```

We can now simulate one game with the devil.
cb is the initial (random) state of the chessboard.

```
In[9]:= cb = rw;
```

The magic field is some integer, the devil chose 23.

```
In[10]:= mf = 23;
```

The first friend enters the room, modifies the chessboard according to *friendOne*. The returned value is the modified chessboard.

```
In[11]:= cb2 = friendOne[cb, mf];
```

Let's check that the Hamming distance between initial and modified chessboard is at most one.

```
In[12]:= HammingDistance[cb, cb2]
Out[12]= 1
```

The second friend comes in and decodes with *friendTwo*, getting 23.

```
In[13]:= friendTwo[cb2]
Out[13]= 23
```

10
Maximum subsequence

Problem

Given a sequence of integer numbers $x_0, x_1, \ldots, x_{N-1}$ (not necessarily positive) find a subsequence $x_i, \ldots, x_{j-1}$ such that the sum of numbers in it is maximum over all subsequences of consecutive elements.

We adopt the same notation used in *Programming in the 1990s* [1] and *Programming, The Derivation of Algorithms*[2]: The notation of function application is the "dot" notation with name of function, followed by arguments, each separated by a dot. The notation of quantified expressions has the operator followed by the bounded variables, then a colon followed by the range for the bounded variables and ended with a colon and the actual expression. So

$$(\sum k : i \leq k < j : x_k)$$

corresponds to the more classical mathematical notation $\sum_{k=i}^{j-1} x_k$.
For our derivation steps in predicate calculus we will use the following notation:

$$\begin{array}{ll} & A \\ = & \{\text{reason why A equals B}\} \\ & B \\ \leq & \{\text{reason why B is less than C}\} \\ & C \end{array}$$

If all the numbers are positive then the maximum sum is the sum of the whole initial sequence. If all the numbers are negative then the maximum sum is 0 (by definition 0 is the sum over an empty range). So the interesting case is a sequence with positive and negative numbers in it.

[1] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

[2] A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990

We hope to find an algorithm that visits every number in the sequence only once, so with runtime $O(n)$. Let's introduce some notation: Let's introduce some notation[3] :

[3] Our problem can be stated as finding $f.N$ given $x_i \in \mathbb{Z}, 0 \leq i < N, N \in \mathbb{N}$.

$$f.n = (MAXi, j : 0 \leq i \leq j \leq n : s.i.j)$$

with

$$s.i.j = (\sum k : i \leq k < j : x_k).$$

We will use properties of quantified expressions as covered in Chapter 3 of *Programming in the 1990s*[4].

[4] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

$$\begin{array}{ll} & f.N \\ = & < \text{definition of f} > \\ & (MAXi, j : 0 \leq i \leq j \leq N : s.i.j) \\ = & < \text{range nesting} > \\ & (MAXj : 0 \leq j \leq N : (MAXi : 0 \leq i \leq j : s.i.j)) \\ = & < \text{defining } p.j = (MAXi : 0 \leq i \leq j : s.i.j) > \\ & (MAXj : 0 \leq j \leq N : p.j) \\ = & < \text{range split, 1-point rule} > \\ & (MAXj : 0 \leq j < N : p.j) \ max \ p.N \\ = & < \text{definition of f} > \\ & f.(N-1) \ max \ p.N \end{array}$$

We now have a recursive expression for f, which still depends on a newly introduced function p. Let's see if we can get a recursive expression for p too:

$$\begin{array}{ll} & p.N \\ = & < \text{definition of p} > \\ & (MAXi : 0 \leq i \leq N : s.i.N) \\ = & < \text{range split, 1-point rule} > \\ & (MAXi : 0 \leq i < N : s.i.N) \ max \ s.N.N \\ = & < \text{definition of s and s.N.N = o by definition of sum over empty range} > \\ & (MAXi : 0 \leq i < N : (\sum k : i \leq k < N : x_k)) \ max \ 0 \\ = & < \text{range split in sum} > \\ & (MAXi : 0 \leq i < N : (\sum k : i \leq k < N-1 : x_k) + x_{N-1}) \ max \ 0 \\ = & < + \text{ distributes over max} > \\ & (x_{N-1} + (MAXi : 0 \leq i < N : (\sum k : i \leq k < N-1 : x_k)) \ max \ 0 \\ = & < \text{definition of p} > \\ & (x_{N-1} + p.(N-1)) \ max \ 0 \end{array}$$

So $f.N = f.(N-1) \ max \ p.N$ and $p.N = (x_{N-1} + p.(N-1)) \ max \ 0$. The base cases are $f.0 = 0$ and $p.0 = 0$.

Armed with these recursive relations we can provide a Haskell program that solves the problem:

Listing 10.1: Haskell code

```
maxSum :: [Int] -> (Int, Int)

maxSum (x:xs) = let (a, b) = maxSum xs
                    c = x + b
                in (max c (max a 0), max c 0)

maxSum [] = (0, 0)
```

The maxSum function calculates the tuple $(f.N, p.N)$.

11
Minkowski Sum & Well-spaced triples

Fast Fourier Transform and using it to speed up polynomial multiplication is the topic of the two problems[1] in this note.

[1] Jeff Erickson. Algorithms — Extended Dance Remix: Fast Fourier Transforms. https://jeffe.cs.illinois.edu/teaching/algorithms/notes/A-fft.pdf, 2021. [Online; accessed 07-May-2022]

Problem

Given two sets of integers $X \subset \mathbb{Z}$ and $Y \subset \mathbb{Z}$, compute the size of the Minkowski sum: $X + Y = \{x + y : x \in X, y \in Y\}$ in $O(n \log n)$ time.

Solution. A pretty straightforward way of calculating the Minkowski sum is to generate all possible pairs (that is a nested loop, so $O(n^2)$) and then also making sure the resulting values form a set, so only occur once. This can be achieved by storing the values as we go in a balanced binary search tree. For each value the cost would be $O(\log n)$, so the straightforward solution has a runtime of $O(n^2 \log n)$ if using a binary search tree or $O(n^2)$ if using a hashtable.

Can we do better? This problem is an exercise in the FFT chapter of Jeff Erickson's Algorithms book. So the answer is: yes we can. To do so, we remember that multiplying two polynomials given in coefficient form can be done in $O(n \log n)$ using the Fast Fourier Transform.

But what polynomials should we consider? Let's explore polynomial multiplication with a couple of examples:

$$\begin{aligned}(1+x)(x^2+x^5) &= 1(x^2+x^5) + x(x^2+x^5) \\ &= x^2 + x^5 + x^3 + x^6\end{aligned}$$

$$\begin{aligned}(2x+x^4)(1+3x^3) &= 2x(1+3x^3) + x^4(1+3x^3) \\ &= 2x + 6x^4 + x^4 + 3x^7\end{aligned}$$

Notice how we multiplied each monomial[2] from the first polynomial with each mononial from the second polynomial. In the second example it is also visible that we haven't yet collected together all the monomials of degree four of the multiplication result to better demonstrate that each monomial from the first polynomial is multiplied with each monomial from the second polynomial.

When we multiply two monomials, their coefficients get multiplied and their exponents get added. Each monomial from one polynomial is paired with each monomial from the other polynomial in an operation (multiplication) and in that operation the exponents are added. This strongly suggests[3] that we should define a polynomial from one of the given sets of integers by making the members of the set be the exponents of the monomials that form the polynomial.

For example: if $X = \{2, 5, 6\}$ then the corresponding polynomial could be $x^2 + x^5 + x^6$. The polynomial coefficients have been arbitrarily chosen to all be one [4].

Thus we define the two polynomials $p_X(x)$ and $p_Y(x)$ from the given integer sets X and Y:

$$p_X(x) = \sum_{i \in X} x^i$$
$$p_Y(x) = \sum_{j \in Y} x^j$$

and we multiply them

$$p_X(x)p_Y(x) = \sum_{i \in X} \sum_{j \in Y} x^{i+j}$$

The exponents of the monomials of the polynomial product are the members of the Minkowski sum $X + Y$. The size of $X + Y$ is the number of monomials [5].

This gives us a way to calculate the size of $X + Y$ in $O(n \log n)$ because we can do the polynomial multiplication using FFT in $O(n \log n)$. □

[2] A monomial is an individual term of a polynomial A polynomial is a sum of monomials. In our example the monomials of $1 + x$ are 1 and x. The monomials of $x^2 + x^5$ are x^2 and x^5.

[3] In the problem each member of the first set is paired with each member of the second set in an operation (addition).

[4] There is one small wrinkle in this scheme. In the problem the given sets are sets of integers, so they can be negative. We cannot have monomials with negative exponents. For now let us assume X and Y only contain non-negative integers with the promise that at the end of this solution we will address how to drop this assumption.

[5] As promised: what can we do when X and Y contain negative integers. Let $d > 0$ be a constant such that both $d + X = \{d + a : a \in X\}$ and $d + Y = \{d + b : b \in Y\}$ have only non-negative members.

We define the polynomials slightly differently:

$$p_X(x) = \sum_{i \in X} x^{i+d}$$
$$p_Y(x) = \sum_{j \in Y} x^{j+d}$$

and we multiply them

$$p_X(x)p_Y(x) = \sum_{i \in X} \sum_{j \in Y} x^{i+j+2d}$$

Again, the size of $X + Y$ is the number of monomials.

Problem

Given is a bit string $B[1 \ldots n]$. A well-spaced triple is a triple (i, j, k) of indices such that $1 \leq i < j < k \leq n$ and $B[i] = B[j] = B[k] = 1$. Detect in $O(n \log n)$ time if bit string B contains a well-spaced triple.

Solution. Again the brute-force solution is easy to describe: for each middle index in the triple we consider all possible distances to left and right indices and check the condition. This is a nested loop with runtime $O(n^2)$.

We will try to improve on the brute-force by again employing polynomial multiplication even though it is not an obvious choice. After all we are only given one bit string, not two bit strings and multiplication needs two operands. But maybe we can derive a polynomial from the bit string and then square that polynomial which would be a polynomial multiplication.

Let's explore what would happen if we use the given bit string B directly as a coefficient vector of a polynomial

$$p_B(x) = \sum_{i=0}^{n-1} B[i+1]x^i$$

Squaring $p_B(x)$ we get

$$\begin{aligned}(p_B(x))^2 &= p_B(x)p_B(x) \\ &= (\sum_{i=0}^{n-1} B[i+1]x^i)(\sum_{j=0}^{n-1} B[j+1]x^j) \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} B[i+1]B[j+1]x^{i+j}\end{aligned}$$

For all $0 \leq j < n$ monomial x^j is present in $p_B(x)$ if $B[j+1] = 1$. If monomial x^j is present in $p_B(x)$ then monomial x^{2j} is present in $(p_B(x))^2$. This is because monomial x^j pairs with itself in the polynomial multiplication. So the coefficient of monomial x^{2j} in $(p_B(x))^2$ is at least one (namely $B[j+1]B[j+1] = 1$). Can it be larger than one? It would mean that some other monomial pairing has exponent sum equal to $2j$. Let i and k be the two indices for which $B[i+1] = 1$, $B[k+1] = 1$ and $i + k = 2j$. But $i + k = 2j$ is equivalent to $k - j = j - i$ which means that (i, j, k) form a well-spaced triple. This (i, k) monomial pairing appears twice in the polynomial multiplication (once for monomial x^i from the left and monomial x^k from the right and once reversed with monomial x^k from the left and monomial x^i from the right). That means that the well-spaced triple contributes the value two to the coefficient of x^{2j} in $(p_B(x))^2$ and we have found our criteria for detecting well-spaced triples: if $(p_B(x))^2$ has any monomials with even degree and coefficients greater or equal to three, then B has a well-spaced triple.

This gives us a way to detect well-spaced triples in $O(n \log n)$ because we can do the polynomial multiplication using FFT in $O(n \log n)$.

□

12
No consecutive integers

INTEGER EQUATIONS and multisets are the topics of the problem [1] in this note.

[1] Variation of Problem 1-72. on page 45 in N. Loehr. *Combinatorics*. Discrete Mathematics and Its Applications. CRC Press, 2017. ISBN 9781498780278

Problem

Determine the number of subsets of size k from set $\{1, 2, \ldots, n\}$ that do not contain consecutive integers.

The number of subsets of size k from set $\{1, 2, \ldots, n\}$ (without any constraints) is given by the binomial coefficient $\binom{n}{k}$. Each subset of size k can be represented as a word of length n from alphabet $\{\bigstar, []\}$ with k $[]$'s and $n-k$ $\bigstar$'s: if i is in the subset then the corresponding word has a $[]$ at position i otherwise it has a $\bigstar$ at position i. This representation is clearly a bijection. The constraint of no consecutive integers in a subset implies no adjacent $[]$'s in the corresponding word[2].

[2] For example given set $\{1, 2, 3, 4\}$ the subset $\{2, 4\}$ corresponds to $\bigstar\,[]\,\bigstar[]$. The subset $\{1, 2\}$ has consecutive integers and corresponds to $[]\,[]\,\bigstar\bigstar$. The reason why we chose $[]$ to indicate inclusion into a subset will become clear soon.

We will now associate words from $\{\bigstar, []\}^n$ with other combinatorial objects: the integer equations.

Definition 12.1. Given fixed integers $m > 0$ and $t \geq 0$ a sequence $(z_1, z_2, \ldots, z_m)$ is an **integer equation** if $\forall i : 1 \leq i \leq m : z_i \in \mathbb{N}_0$ and

$$\sum_{i=1}^{m} z_i = t$$

The number t is called the **target** of the integer equation.

Note that these are sequences and order matters. From an integer equation $(z_1, z_2, \ldots, z_m)$ we construct a $\{\bigstar, []\}^{t+m-1}$ word in the following way: start with z_1 number of $\bigstar$'s, then a $[]$, then z_2 number of $\bigstar$'s, then a $[]$ and so on finishing with the z_m number of $\bigstar$'s which are **not** followed by a $[]$. The word will contain exactly t $\bigstar$'s and they will

need exactly $m-1$ [] separators to know which stars belong to which z_i. It's easy to verify that this encoding is also a bijection[3].

[3] As an example let $m = 5$ and target $t = 10$. The sequence $(1,2,1,3,3)$ is an integer equation since

$$1+2+1+3+3=10$$

and it corresponds to the word

★ [] ★★ [] ★ [] ★★★ [] ★★★

This in turn corresponds to the subset $\{2,5,7,11\}$ of set $\{1,\ldots,14\}$.

Lemma 12.2. *If we set $t = n-k$ (the number of ★'s) and from $t+m-1 = n$ we get $m = k+1$ (the word length) then we can associate subsets of size k from set $\{1,2,\ldots,n\}$ with integer equations $(z_1,z_2,\ldots,z_{k+1})$ for target $n-k$. The constraint of not having consecutive integers in the subsets translates to integer equations $(z_1,z_2,\ldots,z_{k+1})$ where $z_i > 0$ except for z_1 and z_{k+1} (the first and the last in the sequence). This follows from the encoding not allowing adjacent []'s so there need to be ★'s separating the []'s.*

According to this association if we can count the number of integer equations with all but the first and last z_i strictly positive then we also have the number of subsets with no consecutive integers. To get there we will first count the number of anagrams, then the number of multisets, then the number of integer equations and finally the number of integer equations with all but the first and last z_i strictly positive. In what follows we will use n and k for other things before we bring it back in the end to our initial problem.

Let's start this journey with anagrams. Let $\{s_1,s_2,\ldots,s_k\}$ be an alphabet of distinct symbols. We can build words with these symbols, for example $s_2s_2s_1s_3s_3s_1$. As a notational convenience $s_is_is_i\ldots s_i = s_i^j$ if s_i appears j consecutive times in a word, so the example would be $s_2^2s_1s_3^2s_1$.

Definition 12.3. A word is an **anagram**[4] of $s_1^{n_1}s_2^{n_2}\ldots s_k^{n_k}$ with $n_i > 0$ if it is a word containing exactly n_i number of s_i symbols for each $1 \le i \le k$. We denote with $\mathcal{A}(s_1^{n_1}s_2^{n_2}\ldots s_k^{n_k})$ the set of all anagrams of $s_1^{n_1}s_2^{n_2}\ldots s_k^{n_k}$.

[4] For example given word a^2b the words *aba* and *baa* are anagrams of it. The word *abb* is not.

Theorem 12.4. *Given the set of anagrams $\mathcal{A}(s_1^{n_1}s_2^{n_2}\ldots s_k^{n_k})$ let $n = \sum_{i=1}^k n_k$. Then*

$$|\mathcal{A}(s_1^{n_1}s_2^{n_2}\ldots s_k^{n_k})| = \binom{n}{n_1,n_2,\ldots,n_k}$$

where $\binom{n}{n_1,n_2,\ldots,n_k}$ is the multinomial coefficient[5].

[5] The multinomial coefficient is defined as

$$\binom{n}{n_1,n_2,\ldots,n_k} = \frac{n!}{\prod_{i=1}^k n_i!}$$

Proof. We have n positions in our word that we need to fill with symbols. We are going to make the following choices: first we choose n_1 positions from those n positions where we fill in the symbol s_1. Then we choose the n_2 positions from the remaining unfilled positions where we fill in s_2 and so on. In total we make k such choices and the number of remaining unfilled positions at each stage is independent of the previous choices, so the multiplication rule applies. For our first symbol s_1 we have $\binom{n}{n_1}$ possibilities, for our second symbol we have $\binom{n-n_1}{n_2}$ possibilities and so on. Because of the multiplication rule the total number of choices is the product of all these binomial coefficients,

so

$$|\mathcal{A}(s_1^{n_1} s_2^{n_2} \dots s_k^{n_k})| = \prod_{i=1}^{k} \binom{n - (\sum_{j=1}^{i-1})}{n_i}$$

Expanding[6] the binomial coefficients on the right-hand side into factorials according to the binomial coefficient definition and simplifying the expression gives us the desired result. □

[6] After the binomial coefficients are expanded the product becomes a telescoping product that simplifies to exactly the multinomial coefficient.

We move on to **multisets**. Informally multisets are sets (order does not matter) where each element can appear more than once. So given a set A (the alphabet) a multiset is a tuple of A together with a function $\mu : A \mapsto \mathbb{N}$ that determines how often an element $a \in A$ appears in the multiset. For notational convenience we will use curly braces and list elements (with exponents if they appear more than once). For example $\{a^2, b, c^4\}$ is a multiset where a appears twice, b once and c four times. Note that order does not matter, so $\{a^2, b, c^4\}$ is the same multiset as $\{b, a^2, c^4\}$. The size of the multiset is the number of elements in it with elements appearing more than once counted accordingly, so

$$|(A, \mu)| = \sum_{a \in A} \mu(a)$$

Theorem 12.5. *The number of multisets of size k from an alphabet set of size n is*[7]

$$\binom{k + n - 1}{k}$$

[7] For example with alphabet set $\{a, b\}$ the multisets of size two are $\{a^2\}$, $\{b^2\}$, $\{a, b\}$, so there are three of them.

Proof. We will do an encoding of multisets to anagrams similar to what we did at the beginning of this section with ★'s and []'s. To avoid confusion with that previous encoding in this proof we will use the symbols ∘ and |.

Let $A = \{a_1, a_2, \dots, a_n\}$ be our alphabet. For a multiset (A, μ) with size k we define the following word[8] with symbols $\{\circ, |\}$:

$$\circ^{\mu(a_1)} | \circ^{\mu(a_2)} | \dots | \circ^{\mu(a_n)}$$

[8] The multisets from the previous example would be encoded as follows:

$$\{a^2\} \mapsto \circ\circ|$$
$$\{b^2\} \mapsto |\circ\circ$$
$$\{a, b\} \mapsto \circ|\circ$$

The first circles denote how often a_1 is in the multiset. They are separated by a | from the circles that denote how often a_2 is in the multiset and so on. In total there are k circles because the multiset has size k and there need to be $n - 1$ separators because the alphabet has size n and the circles for each element need to be kept apart. It's easy to see that we have defined a bijection from the set of multisets of size k with alphabet of size n to the set of anagrams $\mathcal{A}(\circ^k|^{n-1})$. From theorem 12.4 we already know how to count the size of $\mathcal{A}(\circ^k|^{n-1})$ and with the bijection it proves this theorem. □

Our next stop are the number of integer equations. Given m and t how many integer equations $(z_1, z_2, \dots, z_m)$ for target t are there?

Theorem 12.6. *The number of integer equations $(z_1, z_2, \ldots, z_m)$ for target t is*

$$\binom{t+m-1}{t}$$

Proof. We will associate a multiset with each integer equation[9]. The multiset will contain the element i z_i many times, for $1 \leq i \leq m$. Again it can be checked that this defines a bijection. These multisets belong to the set of multisets of size t from an alphabet of size m and theorem 12.5 counts them. By the bijection rule we have proven this theorem. □

[9] For example with $m = 5$ and target $t = 10$ the integer equation $(1, 2, 1, 3, 3)$ would correspond to multiset $\{1, 2^2, 3, 4^3, 5^3\}$.

We are almost done. In the beginning of this section we encoded our subsets without consecutive integers as integer equations with all but the first and last summand strictly positive. So we need to count these types of integer equations with this constraint.

Theorem 12.7. *The number of integer equations $(y_1, y_2, \ldots, y_m)$ for target t with $y_i > 0$ for all $1 < i < m$ is*

$$\binom{t+1}{m-1}$$

Proof. For an integer equation $(y_1, y_2, \ldots, y_m)$ we have $\sum_{i=1}^{m} y_i = t$ and $y_i > 0$ for all $1 < i < m$. So we can write

$$y_1 + \sum_{i=2}^{m-1} (y_i - 1) + y_m = t - (m-2)$$

This shows that we can transform the integer equations with the strictly positive constraints into normal integer equations without constraints but with a new target. This again is a bijection. We know how to count these from theorem 12.6. The new target is $t - m + 2$. Plugging it in we get

$$\binom{t-m+2+m-1}{m-1} = \binom{t+1}{m-1}$$

□

Using 12.7 and $t = n - k$ and $m = k + 1$ as described by our association 12.2 of subsets of size k without consecutive integers from set $\mathbb{N}_n$ to integer equations with all but the first and last strictly positive terms, we are finally able to solve the problem in this section. The answer is $\binom{n-k+1}{k}$.

13
Paying a dollar

Problem

In how many combinations of half-dollars, quarters, dimes, nickels and pennies can you pay out one dollar ? You can assume you have enough coins for any combination and any coins of one denomination are indistinguishable.

We will look at tuples (h, q, d, n, p) where h is the number of half-dollars, q the number of quarters, d the number of dimes, n the number of nickels and p the number of pennies. We want to consider all tuples (h, q, d, n, p) such that:

$$50h + 25q + 10d + 5n + p = 100$$

We want to determine how many such tuples exist. Let's establish what the possible values for h, q, d, n and p can be:

Denomination	Possible Values	Values range size
h	0, 1, 2	3
q	0, 1, 2, 3, 4	5
d	0, . . . , 10	11
n	0, . . . , 20	21
p	0, . . . , 100	101

Clearly h, q, d, n and p cannot take values outside of the ones listed because it would violate the requirement: $50h + 25q + 10d + 5n + p = 100$.

The multiplicity principle tells us there are $3 * 5 * 11 * 21 * 101 = 349965$ distinct tuples from the possible values.

But not all of them fulfill $50h + 25q + 10d + 5n + p = 100$. To figure out how many of them do let's introduce the following notation:

$$\#(h,q,d,n,p)_x = |\{(h,q,d,n,p) : 50h + 25q + 10d + 5n + p = x\}|$$

so $\#(h,q,d,n,p)_x$ is the number of tuples of half-dollars, quarters, dimes, nickels and pennies that add up to x. We are looking for $\#(h,q,d,n,p)_{100}$.

We also use $\#(q,d,n,p)_x$ for tuples of quarters, dimes, nickels and pennies that add up to x, $\#(d,n,p)_x$ for dimes, nickels and pennies that add up to x etc.

The half-dollar denomination has the smallest values range size so it's probably in our favor to start with it and break down the problem into smaller problems from there. It is clear that

$$\#(h,q,d,n,p)_{100} = \#(q,d,n,p)_{100} + \#(q,d,n,p)_{50} + \#(q,d,n,p)_0$$

because $\#(q,d,n,p)_{100}$ comes from h taking value 0, $\#(q,d,n,p)_{50}$ from h taking value 1 and $\#(q,d,n,p)_0$ from h taking value 2. Continuing down this path and breaking down the q cases:

$$\begin{aligned}\#(q,d,n,p)_{100} &= \#(d,n,p)_{100} + \#(d,n,p)_{75} + \\ &\quad \#(d,n,p)_{50} + \#(d,n,p)_{25} + \#(d,n,p)_0 \\ \#(q,d,n,p)_{50} &= \#(d,n,p)_{50} + \#(d,n,p)_{25} + \#(d,n,p)_0 \\ \#(q,d,n,p)_0 &= \#(d,n,p)_0\end{aligned}$$

This is getting tedious though. Maybe we can find a closed-form formula for $\#(d,n,p)_x$. Let's try to find one for $\#(n,p)_x$ first.

Lemma 13.1. *If $x = 5y$ then*

$$\#(n,p)_x = y + 1$$

Proof. n can take values in range $0,\ldots,y$ and for each value of n there is only one possible value of $p = x - 5n$. □

Lemma 13.2. *If x is a multiple of 5 then*

$$\#(d,n,p)_x = \begin{cases}(y+1)^2 & \text{if } x = 10y \\ (y+1)(y+2) & \text{if } x = 10y+5\end{cases}$$

Proof. Let's deal with the case $x = 10y$ first.

$$\begin{aligned}
\#(d,n,p)_x &= \sum_{k=0}^{y} \#(n,p)_{x-10k} \\
&= \sum_{k=0}^{y} \#(n,p)_{10(y-k)} \\
&= \sum_{k=0}^{y} \#(n,p)_{10k} \\
&= \sum_{k=0}^{y} (2k+1) \\
&= (y+1)^2
\end{aligned}$$

The case $x = 10y + 5$ is established in a similar fashion. □

We can now use the two lemmas to compute the number of combinations:

$$\begin{aligned}
\#(h,q,d,n,p)_{100} &= \#(d,n,p)_{100} + \#(d,n,p)_{75} + \\
&\quad 2\#(d,n,p)_{50} + 2\#(d,n,p)_{25} + 3\#(d,n,p)_0 \\
&= 11^2 + 8*9 + 2*6^2 + 2*3*4 + 3 \\
&= 292
\end{aligned}$$

14
Penn & Teller Full Deck of Cards

COUNTING WORDS WITH CONSTRAINTS is the topic of the problem in this note. This problem was posed by a coworker at a lunch discussion.

Problem

When you go see the Penn & Teller Magic Show in Las Vegas you can get a random card from the **Perfectly Ordinary Deck of Cards** at the entrance. How many times do you have to see the show to collect the full deck.

We assume the supply of cards at the entrance is endless and thoroughly shuffled. This allows us to work with a probability model of drawing with replacement where each card is equally likely to be drawn with probability $\frac{1}{52}$. At each visit we draw a card[1]. After k visits we have built up a sequence of cards which we model as a word $w_k = (c_1, c_2, \ldots, c_k)$ of size k from an alphabet of size 52 ($\forall c_i : c_i \in \{1, \ldots, 52\}$). Our random variable X is the number of visits needed to achieve a full deck. The probability $P(X = k)$ means that it took k visits to achieve the full deck.

The key observation is this: if it took k visits to achieve full deck then at visit $k-1$ the corresponding word of cards $w_{k-1} = (c_1, c_2, \ldots, c_{k-1})$ is missing just one card and all the other cards appear at least once in the word[2].

Let $\mathcal{A} = \{1, 2, \ldots, 52\}$ be our alphabet and $L_k(\mathcal{A}) = \{(c_1, c_2, \ldots, c_k) : \forall i : 1 \leq i \leq k : c_i \in \mathcal{A}\}$ the set of all the words of length k with letters (cards) from the alphabet $\mathcal{A}$. Let $M_k(c)$ be the set of words of length k where letter c does not occur in the word and every other letter occurs at least once:

[1] You can get two cards at each visit to the show in Las Vegas. Drawing only one card is a simplification to keep the expressions smaller. The case with two cards is similar but the expressions get a little bigger because you have more cases of the card sequence right before the visit that achieves full deck. We will point out the differences at the end of this note.

[2] It might be easier to see this with a concrete card. Imagine you are drawing an ace of spades at visit k and getting the full deck. This means that before the visit k you are still missing the ace of spades. If not, then drawing the ace of spades at visit k wouldn't complete the deck. Getting the full deck at visit k also means that you are not missing any other cards before the visit k, otherwise if one of the other cards would be missing then drawing an ace of spades again wouldn't complete the deck.

$$M_k(c) = \{(c_1, c_2, \ldots, c_k) : (\forall i : 1 \leq i \leq k : c_i \in \mathcal{A} \wedge c_i \neq c) \wedge (\forall d \in \mathcal{A} \setminus \{c\} : \exists i : 1 \leq i \leq k : c_i = d)\}$$

So now let us assume that at visit k we draw letter c and get the full deck. From our previous argument above we know that then the word w_{k-1} we built in the previous $k-1$ visits has to be in $M_{k-1}(c)$.

The probability that at visit k we draw letter c and get the full deck is thus the probability of drawing card c times the probability that $w_{k-1} \in M_{k-1}(c)$. It follows that:

$$P(X = k) = \sum_{c \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \frac{|M_{k-1}(c)|}{|L_{k-1}(\mathcal{A})|}$$

As we will see below, $|M_{k-1}(c)|$ is the same for all cards c, so for some fixed card c_0 we have $\forall c \in \mathcal{A} : |M_{k-1}(c)| = |M_{k-1}(c_0)|$. Then

$$P(X = k) = \frac{|M_{k-1}(c_0)|}{|L_{k-1}(\mathcal{A})|} \sum_{c \in \mathcal{A}} \frac{1}{|\mathcal{A}|} = \frac{|M_{k-1}(c_0)|}{|L_{k-1}(\mathcal{A})|}$$

We already know that

$$|L_{k-1}(\mathcal{A})| = |\mathcal{A}|^{k-1}$$

What is left to do to compute $P(X = k)$ is count $|M_{k-1}(c_0)|$. To avoid carrying around the $k-1$ we will count $|M_k(c_0)|$ instead and adjust afterwards.

How do we compute $|M_k(c_0)|$? We have two constraints on the words in $w_k = (c_1, c_2, \ldots, c_k) \in M_k(c_0)$:

$$\text{constraint } C_1 : \forall i : 1 \leq i \leq k : c_i \in \mathcal{A} \wedge c_i \neq c_0$$
$$\text{constraint } C_2 : \forall d \in \mathcal{A} \setminus \{c_0\} : \exists i : 1 \leq i \leq k : c_i = d$$

Constraint C_1 is easy to satisfy: we just use the alphabet without letter c_0: $\mathcal{A}_1 = \mathcal{A} \setminus \{c_0\}$.

For constraint C_2 we could try to eliminate all the subsets of $L_k(\mathcal{A}_1)$ with words missing one letter from $\mathcal{A}_1$, all the subsets with words missing two letters from $\mathcal{A}_1$ and so on all the way to subsets with words missing all but one letter from $\mathcal{A}_1$. Something like this:

$$|M_k(c_0)| = |L_k(\mathcal{A}_1)| - \sum_{\mathcal{B} \subset \mathcal{A}_1} |L_k(\mathcal{A}_1 \setminus \mathcal{B})|$$

But we have to be careful here. The subsets that we aim to eliminate are not disjoint and this would lead to overcounting[3]. So instead the correct way to count this is:

$$|M_k(c_0)| = \sum_{i=0}^{|\mathcal{A}_1|-1} (-1)^i \sum_{\mathcal{B}\subset\mathcal{A}_1, |\mathcal{B}|=i} |L_k(\mathcal{A}_1 \setminus \mathcal{B})|$$

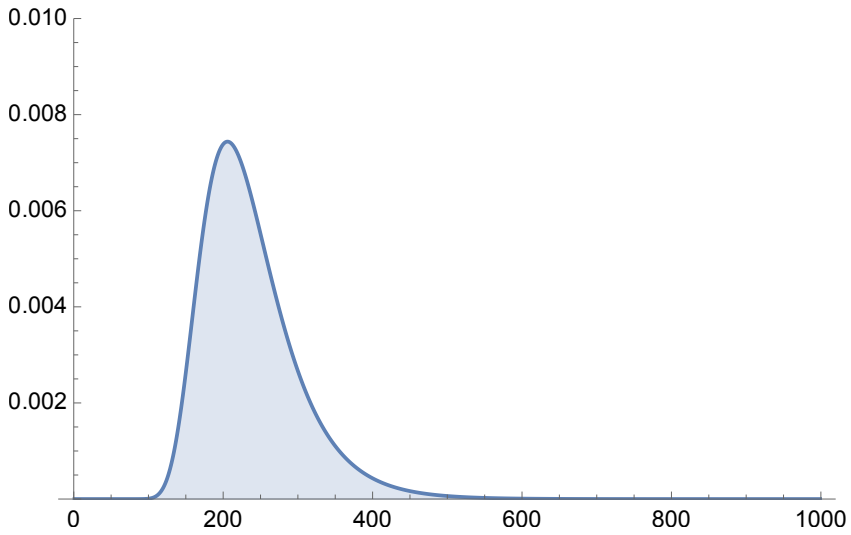
There are $\binom{|\mathcal{A}_1|}{i}$ subsets $\mathcal{B}$ of size i and for each $|L_k(\mathcal{A}_1 \setminus \mathcal{B})| = (|\mathcal{A}_1| - i)^k$. It follows that

$$|M_k(c_0)| = \sum_{i=0}^{|\mathcal{A}_1|-1} (-1)^i \binom{|\mathcal{A}_1|}{i} (|\mathcal{A}_1| - i)^k$$

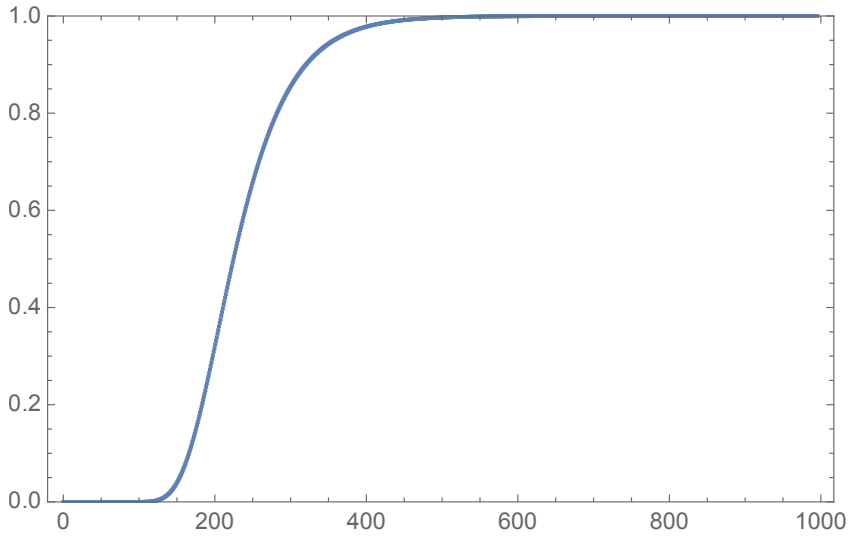
We have all the pieces now. We can adjust back to $k-1$ and also use $|\mathcal{A}| = 52$ to make a nice, closed formula for $P(X = k)$:

$$P(X = k) = \frac{1}{52^{k-1}} \sum_{i=0}^{50} (-1)^i \binom{51}{i} (51 - i)^{k-1}$$

Plugging this into Mathematica we can see the distribution and the cumulative distribution:



[3] Subsets with two missing letters are also subsets with one missing letter. Subsets with three missing letters are also subsets with two missing letters and subsets with one missing letter. And so on. It is indeed a use case for the inclusion-exclusion principle in combinatorics:
Wikipedia. Inclusion–exclusion principle — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Inclusion%E2%80%93exclusion%20principle&oldid=1086507513, 2022. [Online; accessed 07-May-2022].



It looks like around 400 visits will most likely get you a full deck. It might be cheaper to buy the full deck from the **Penn & Teller** online store.

It is always a good idea to double-check our result with a simulation[4]. The Python program below runs trials and records the number of visits necessary for a full deck:

[4] This is how I discovered a bug in my initial calculation. The listing is from my coworker that suggested the problem.

Listing 14.1: Simulation

```
import random

def trial():
    cards = [False] * 52
    visits = 0
    while not all(cards):
        visits += 1
        cards[random.randint(0, 51)] = True
    return visits

def main():
    num_trials = 100000
    trials = [trial() for _ in range(0, num_trials)]
    trials.sort()
    print(f"n={num_trials}")
    print(f"p0:   {trials[0]}")
    print(f"p25:  {trials[num_trials >> 2]}")
    print(f"p50:  {trials[num_trials >> 1]}")
    print(f"p75:  {trials[(num_trials >> 2) * 3]}")
    print(f"p100: {trials[num_trials - 1]}")

if __name__ == "__main__":
    main()
```

The results confirm that at least we are not orders of magnitude off:

```
n=100000
p0:   95
p25:  190
p50:  225
p75:  269
p100: 822
```

As promised, what are the differences when two cards are drawn at each visit. The constraints on the words right before the visit that achieves full deck are a little bit more complicated: there could be one or two cards missing. When one card is missing, the missing card could come in once or twice on the last visit. That's more or less it. Working out a closed formula for this is left as an exercise to the reader.

15
Points on circle

Problem

N distinct points, numbered from 0 onwards, are located on a circle (in the rest of this problem all point numbers are taken **mod**N). Point $i+1$ is the clockwise neighbor of point i. An integer array, $dist[0 \ldots N)$, is given such that $dist.i$ is the distance (along the circle) between points i and $i+1$. Derive a program to determine whether four of these points form a rectangle.

We adopt the same notation used in *Programming in the 1990s* [1] and *Programming, The Derivation of Algorithms*[2]: The notation of function application is the "dot" notation with name of function, followed by arguments, each separated by a dot. The notation of quantified expressions has the operator followed by the bounded variables, then a colon followed by the range for the bounded variables and ended with a colon and the actual expression. So

$$(\sum k : i \leq k < j : x_k)$$

corresponds to the more classical mathematical notation $\sum_{k=i}^{j-1} x_k$.
For our derivation steps in predicate calculus we will use the following notation:

$$\begin{array}{ll} & A \\ = & \{\text{reason why A equals B}\} \\ & B \\ \leq & \{\text{reason why B is less than C}\} \\ & C \end{array}$$

We are asked to solve S in

$\|[$

[1] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

[2] A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990

con $N : \ int;\ \{N \geq 4\}$
  $dist(i : 0 \leq i < N) : \ int;\ \{\forall i : 0 \leq i < N : dist.i > 0\}$
var $r : bool$;
  S
$\{r : r \equiv (\exists$ 4 points that form a rectangle$)\}$
]||

Let's first develop a more manageable postcondition. Evidently four points that form a rectangle is equivalent to two pairs of diametral opposing points. We introduce a function for the set of all indices from point x to point y in clockwise direction along the circle:

$$I : [0, \ldots, N) \to [0, \ldots, N) \to 2^{[0,\ldots,N)}$$
$$I.x.y := \begin{cases} [x, \ldots, y) & , \ x \leq y \\ [x, \ldots, N) \bigcup [0, \ldots, y) & , \ x > y \end{cases}$$

Let C be the circumference of the circle. We define function

$$f : [0, \ldots, N) \to [0, \ldots, N) \to int$$
$$f.x.y := C - 2(\sum i : i \in I.x.y : dist.i)$$

We want to find the number of diametral opposing pairs of points:

||[
  con $N : \ int;\ \{N \geq 2\}$
    $dist(i : 0 \leq i < N) : \ int;\ \{\forall i : 0 \leq i < N : dist.i > 0\}$
  var $r : int$;
    S
  $\{r : r = (\#\, x, y : 0 \leq x < N,\ 0 \leq y < N : f.x.y = 0)\}$
]||

Lemma 15.1. *The function f is increasing in its first argument and decreasing in its second argument.*

Proof. f is increasing in its first argument:

$$\begin{array}{ll} & f.(x+1).y \\ = & \{\text{definition of } f\} \\ & C - 2(\sum i : i \in I.(x+1).y : dist.i) \\ = & \{I.(x+1).y = I.x.y \setminus \{x\}\} \\ & C - 2((\sum i : i \in I.x.y : dist.i) - dist.x) \\ = & \{\text{definition of } f\} \\ & f.x.y + 2dist.x \\ > & \{dist.x > 0\} \\ & f.x.y \end{array}$$

f is decreasing in its second argument:

$$\begin{array}{ll} & f.x.(y+1) \\ = & \{\text{definition of } f\} \\ & C - 2(\sum i : i \in I.x.(y+1) : dist.i) \\ = & \{I.x.(y+1) = I.x.y \cup \{y\}\} \\ & C - 2((\sum i : i \in I.x.y : dist.i) + dist.y) \\ = & \{\text{definition of } f\} \\ & f.x.y - 2dist.y \\ < & \{dist.y > 0\} \\ & f.x.y \end{array}$$

□

Looking at the postcondition

$$\{r : r = (\#\, x, y : 0 \leq x < N,\ 0 \leq y < N : f.x.y = 0)\}$$

we define the function

$$G.a.b = (\#\, x, y : a \leq x < N,\ b \leq y < N : f.x.y = 0)$$

and we will maintain the invariants:

$$\begin{array}{lcl} P_0 & : & G.0.0 = r + G.a.b \\ P_1 & : & 0 \leq a \leq N \\ P_2 & : & 0 \leq b \leq N \end{array}$$

The initial values $r, a, b := 0, 0, 0$ satisfy the invariants and

$$a = N \vee b = N \Rightarrow G.a.b = 0 \Rightarrow r = G.0.0$$

establishes the postcondition, so we can stop when $a = N \vee b = N$.
So far we have

```
[||
    con N : int; {N ≥ 4}
        dist(i : 0 ≤ i < N) : int; {∀i : 0 ≤ i < N : dist.i > 0}
    var a, b, r : int;
    a, b, r := 0, 0, 0;
    do a ≠ N ∧ b ≠ N
       S
    od
    {r : r = G.0.0}
||]
```

We need to increment a, b and maintain the invariants:

$$\begin{array}{ll} & G.a.b \\ = & \{\text{definition of } G\} \\ & (\#\, x,y : a \le x < N,\ b \le y < N : f.x.y = 0) \\ = & \{\text{range split } x = a\} \\ & G.(a+1).b + (\#y : b \le y < N : f.a.y = 0) \\ = & \{f \text{ is decreasing in second argument (15.1), and assume } f.a.b < 0\} \\ & G.(a+1).b \end{array}$$

so $f.a.b < 0 \Rightarrow G.a.b = G.(a+1).b$. Similarly

$$\begin{array}{ll} & G.a.b \\ = & \{\text{definition of } G\} \\ & (\#\, x,y : a \le x < N,\ b \le y < N : f.x.y = 0) \\ = & \{\text{range split } y = b\} \\ & G.a.(b+1) + (\#x : a \le y < N : f.x.b = 0) \\ = & \{f \text{ is increasing in second argument (15.1), and assume } f.a.b > 0\} \\ & G.a.(b+1) \end{array}$$

so $f.a.b > 0 \Rightarrow G.a.b = G.a.(b+1)$. Also for the case $f.a.b = 0$ we have

$$\begin{array}{ll} & r + G.a.b \\ = & \{\text{definition of } G\} \\ & r + (\#\, x,y : a \le x < N,\ b \le y < N : f.x.y = 0) \\ = & \{\text{range split } x = a\} \\ & r + G.(a+1).b + (\#y : b \le y < N : f.a.y = 0) \\ = & \{f \text{ is decreasing in second argument (15.1), and assume } f.a.b = 0\} \\ & (r+1) + G.(a+1).b \end{array}$$

Our program becomes

$$\begin{array}{l} [\!|\, [\\ \quad \textbf{con } N : \mathit{int};\ \{N \ge 4\} \\ \qquad \mathit{dist}(i : 0 \le i < N) : \mathit{int};\ \{\forall i : 0 \le i < N : \mathit{dist}.i > 0\} \\ \quad \textbf{var } a, b, r : \mathit{int}; \\ \quad a, b, r := 0, 0, 0; \\ \quad \textbf{do } a \ne N \wedge b \ne N \\ \qquad \textbf{if} \\ \qquad \Box\ f.a.b > 0 \to b := b + 1 \\ \qquad \Box\ f.a.b < 0 \to a := a + 1 \\ \qquad \Box\ f.a.b = 0 \to a, r := a + 1, r + 1 \\ \qquad \textbf{fi} \\ \quad \textbf{od} \\ \quad \{r : r = G.0.0\} \\ ]\, |\!| \end{array}$$

We cannot have f in the program text so the last thing we have to do is eliminate f. We do this by introducing a new variable $c : \mathit{int}$

and maintaining the additional invariant $P_3 : c = f.a.b$. Lemma 15.1 already showed us the expressions for f when the first or the second argument increase, so our final program looks like this[3]

[3] The program is bound by the function $2N - a - b$ so it is $O(N)$. The solution is an example of the slope search technique.

```
||[
    con N : int; {N ≥ 4}
        dist(i : 0 ≤ i < N) : int; {∀i : 0 ≤ i < N : dist.i > 0}
    var a, b, c, r : int;
    a, b, c, r := 0, 0, C, 0;
    do a ≠ N ∧ b ≠ N
        if
        □ c > 0 → b, c := b + 1, c − 2dist.b
        □ c < 0 → a, c := a + 1, c + 2dist.a
        □ c = 0 → a, c, r := a + 1, 2dist.a, r + 1
        fi
    od
    {r : r = G.0.0}
]||
```

16
Prison Cells

Problem

A prison has n cells with all cell doors shut initially. The warden is a little weird so he walks the whole row of cells and opens every cell door. Then he walks the whole row again and shuts every other cell door. Then he walks the whole row again and opens every third door then walks the row again and shuts every 4th door etc. You can assume that the doors are numbered 0 to $(n-1)$ and the warden always starts at zero and walks them in order. Which doors will stay open when the warden is done ?

Each time the warden walks the row of cells he toggles the state (open or close) of some of the cells. It is clear then that the number of toggles to one cell determines if it is open or closed in the end. In the beginning each cell door is closed so if the number of toggles is even then it stays closed, if it is odd then it is open at the end.

The goal then is to calculate the number of toggles for a cell. The cells are numbered 0 to $(n-1)$ so lets try to calculate the number of toggles for cell k. The first time the warden walks the row of cells he toggles each cell including our cell k. The second time he toggles cells $0, 2, 4, \ldots$. That means he toggles cell k if k is even. The third time around he toggles cells $0, 3, 6, \ldots$ so he toggles cell k if k is a multiple of 3. If we continue we see that the cell k gets toggled on the warden's d walk if k is a multiple of d or said differently if d divides k.

It follows that the number of toggles $T(k)$ for cell k is

$$T(k) = \sum_{d|k} 1.$$

This is already pretty good but for the expression above it's not so

obvious for which k $T(k)$ will be even and for which it will be odd. So we will make a short excursion into basic number theory in the hopes that we can transform the expression into something more revealing.

A little number theory

We say that two integers m and n are *relatively prime* if the only common divisors are ± 1 and we write $(m, n) = 1$ in that case.

Definition 16.1. A function $f : \mathbb{N} \to \Omega$ with Ω a field is said to be **weakly multiplicative** if

$$\forall\ m, n \in \mathbb{N} : (m, n) = 1 \ \Rightarrow\ f(mn) = f(m)f(n).$$

Theorem 16.2. *If f is a weakly multiplicative function then so is the function*

$$g(n) = \sum_{d|n} f(d).$$

Proof. Let $m_1, m_2 \in \mathbb{N}$ with $(m_1, m_2) = 1$. Let's define two sets

$$S_1 = \{d\ :\ d \mid m_1 m_2\},\ S_2 = \{d_1 d_2\ :\ d_1 \mid m_1 \wedge d_2 \mid m_2\}.$$

It is obvious that $S_2 \subseteq S_1$. On the other hand

$\forall x \in S_1 \rightsquigarrow x \mid m_1 m_2$ (by definition)
Let $k = (x, m_1)$, so $x = yk, m_1 = zk$, for some $y, z \in \mathbb{N}$ and $(y, z) = 1$
$x \mid m_1 m_2 \rightsquigarrow yk \mid zkm_2 \rightsquigarrow y \mid m_2$ because $(y, z) = 1$
This means $x = yk \in S_2$ because $y \mid m_2 \wedge k \mid m_1$.

So we have $S_1 = S_2$. We can now write

$$
\begin{aligned}
& g(m_1 m_2) \\
= & \quad < \text{definition of g} > \\
& (\sum d : d \mid m_1 m_2 : f(d)) \\
= & \quad < \text{index sets } S_1 = S_2 \text{ so we change bounded variables} > \\
& (\sum d_1, d_2 : d_1 \mid m_1 \wedge d_2 \mid m_2 : f(d_1 d_2)) \\
= & \quad < \text{f is weakly multiplicative and } (d_1, d_2) = 1 > \\
& (\sum d_1, d_2 : d_1 \mid m_1 \wedge d_2 \mid m_2 : f(d_1) f(d_2)) \\
= & \quad < \text{nesting} > \\
& (\sum d_1 : d_1 \mid m_1 : (\sum d_2 : d_2 \mid m_2 : f(d_1) f(d_2))) \\
= & \quad < \text{multiplication distributes over addition} > \\
& (\sum d_1 : d_1 \mid m_1 : f(d_1)(\sum d_2 : d_2 \mid m_2 : f(d_2))) \\
= & \quad < \text{definition of g} > \\
& (\sum d_1 : d_1 \mid m_1 : f(d_1) g(m_2)) \\
= & \quad < \text{multiplication distributes over addition} > \\
& (\sum d_1 : d_1 \mid m_1 : f(d_1)) g(m_2) \\
= & \quad < \text{definition of g} > \\
& g(m_1) g(m_2).
\end{aligned}
$$

which proves the theorem. □

The theorem tells us that the function $T(k)$ which is the number of toggles for cell k

$$T(k) = \sum_{d|k} 1.$$

is in fact a weakly multiplicative function because the function inside the sum (the constant function 1) is trivially a weakly multiplicative function.

A more detailed solution

If we use the unique prime factorization of k

$$k = p_1^{a_1} p_1^{a_1} \ldots p_h^{a_h}$$

and use the fact that $(p_i^{a_i}, p_j^{a_j}) = 1$ we get

$$T(k) = \prod_{i=1}^{h} T(p_i^{a_i}).$$

But it's easy to see that $T(p_i^{a_i}) = a_i + 1$ so we have

$$T(k) = \prod_{i=1}^{h} (a_i + 1).$$

When is $T(k)$ even ? When any of the a_i are odd. To find out if a cell is open or closed do the prime factorization and look at the exponents of the primes. If any of them is odd then the cell stays closed.

17
0-1 Sequences

Counting inversions is the topic of the problem [1] in this note.

[1] Tung Kam Chuen. 0-1 sequences. 2016. URL https://open.kattis.com/problems/sequences

Problem

You are given a sequence, in the form of a string with characters '0', '1', and '?' only. Suppose there are k '?'s. Then there are 2^k ways to replace each '?' by a '0' or a '1', giving 2^k different 0-1 sequences (0-1 sequences are sequences with only zeroes and ones).
For each 0-1 sequence, define its number of inversions as the minimum number of adjacent swaps required to sort the sequence in non-decreasing order. In this problem, the sequence is sorted in non-decreasing order precisely when all the zeroes occur before all the ones. For example, the sequence 11010 has 5 inversions. We can sort it by the following moves: $11010 \to 11001 \to 10101 \to 01101 \to 01011 \to 00111$.
Find the sum of the number of inversions of the 2^k sequences, modulo $10^9 + 7$.

There are two ways to count the necessary inversions to sort the 2^k 0-1 sequences: we could count for each '0' how many '1' to its left are marching by in the right direction on their way to being sorted. Or we could count for each '1' how many zeros to its right are marching by in the left direction on their way to being sorted.

We arbitrary choose the first way of counting the inversions.

In the sequence $\mathbf{b} = (b_0, b_1, \ldots, bn-1)$ with characters '0', '1', and '?' we will look at each position i where $\mathbf{b}[i] =$ '0' and each position i where $\mathbf{b}[i] =$ '?'.

We define $q(i)$ to be the number of question marks to the left of i and $o(i)$ to be the number of ones to the left of i:

$$q(i) = |\{j : 0 \leq j < i : \mathbf{b}[j] = \text{'?'}\}|$$
$$o(i) = |\{j : 0 \leq j < i : \mathbf{b}[j] = \text{'1'}\}|$$

Let $s(i)$ be the number of inversions coming from $\mathbf{b}[i]$. When $\mathbf{b}[i] =$ '1' we set $s(i) = 0$ so as to not overcount[2].

When $\mathbf{b}[i] =$ '0' we know that all 2^k 0-1 sequences will have $o(i)$ ones to the left of i. These definitely will count in $s(i)$. We also need to consider all ones coming from setting '?' into '1' to the left of i. There are $q(i)$ possibilities here. For each $j : 1 \leq j \leq q(i)$ we can turn j question marks into ones. We have to choose the subset of size j of positions from the set of $q(i)$ positions with question marks[3]. It follows that:

$$s(i) = 2^k o(i) + 2^{k-q(i)}(\sum_{j=1}^{q(i)} \binom{q(i)}{j} j)$$

There is a neat way to simplify the sum with the binomial above using a combinatorial proof: Given a set of people of size N, count in how many ways you can choose a team and from that team choose a leader. There are two ways to count here. In the first way count the number of ways to choose a leader: N ways. Then count the number of ways to choose the rest of the team, which is the number of subsets from the set of people without the leader, so 2^{N-1}. In the second way for each possible team size, count the number of possible teams and then count the number of possible leader in that team. Because both ways count the same things, we have:

$$N2^{N-1} = \sum_{j=1}^{N} \binom{N}{j} j$$

Applied to our $s(i)$ we get:

$$\begin{aligned} s(i) &= 2^k o(i) + 2^{k-q(i)} q(i) 2^{q(i)-1} \\ &= 2^k o(i) + 2^{k-1} q(i) \end{aligned}$$

For $\mathbf{b}[i] =$ '?' we do a similar calculation[4], with the only difference being the number of question marks to the right of i: $2^{k-q(i)-1}$ (one less than in the previous calculation, since position i is a question mark). We get:

$$s(i) = 2^k o(i) + 2^{k-2} q(i)$$

The two cases cover all the counts and we can write the following loop (in Go, leaving out the modulus optimizations):

[2] We chose to count ones marching right, passing zeros.

[3] As a convenience we label the $q(i)$ positions with question marks as position $1, 2, \ldots q(i)$.

[4] We instantiate this question mark as a zero. The case where this position gets instantiated as a one is covered by other zero positions.

```
seen_ones := 0
seen_qmarks := 0
num_inversions := 0

for i:=0; i < n; i++ {
   switch {
   case b[i] == '0':
     num_inversions += 2^k * seen_ones + 2^(k-1) * seen_qmarks
   case b[i] == '1': seen_ones++
   case b[i] == '?':
     num_inversions += 2^k * seen_ones + 2^(k-2) * seen_qmarks
     seen_qmarks++
   }
}
```

For a more complete implementation in C++, see
https://github.com/uwedeportivo/kattis/tree/main/sequences.

18
Last three digits before decimal point

Recurrence relations and modulo arithmetic are the topics of the problem [1] in this note.

[1] Cosmin Negruseri. Code-jam 2008 round 1a: Problem c: Numbers. 2008. URL https://code.google.com/codejam/contest/32016/dashboard#s=p2

Problem

Find the last three digits before the decimal point for the number $(3+\sqrt{5})^n$. For example, when $n = 5$, $(3+\sqrt{5})^5 = 3935.73982\ldots$, the answer is 935. For $n = 2$, $(3+\sqrt{5})^2 = 27.4164079\ldots$, the answer is 027. The value of n is in the range $2 \leq n \leq 2000000000$.

Looking at the numbers $(3+\sqrt{5})^n$, we can see that in general they are not integers. Ideally we would like to deal with integers. This sparks the idea of introducing the complement of $(3+\sqrt{5})$ into the mix, namely $(3-\sqrt{5})$. Let's look at the binomial expansion [2] of $(3+\sqrt{5})^n$:

$$(3+\sqrt{5})^n = \sum_{i=0}^{n} \binom{n}{i} 3^i (\sqrt{5})^{n-i}$$

[2] Binomial expansion:

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$$

Compare this to the binomial expansion of $(3-\sqrt{5})^n$:

$$(3-\sqrt{5})^n = \sum_{i=0}^{n} \binom{n}{i} 3^i (-\sqrt{5})^{n-i}$$

When $n-i$ is even, then $(\sqrt{5})^{n-i}$ and $(-\sqrt{5})^{n-i}$ are integers. When $n-i$ is odd, then the binomial terms for $(\sqrt{5})^{n-i}$ and $(-\sqrt{5})^{n-i}$ in the binomial expansions cancel each other out. So it follows that

$$\forall n \in \mathbb{N} : (3+\sqrt{5})^n + (3-\sqrt{5})^n \in \mathbb{N}$$

This is encouraging, so we define for all n:

$$a_n = (3+\sqrt{5})^n$$
$$b_n = (3-\sqrt{5})^n$$
$$c_n = a_n + b_n$$

We see that $\forall n \in \mathbb{N} : 0 < b_n < 1$, so $c_n = \lceil a_n \rceil$.

Concentrating on c_n, lets try to find the hundreds digit, the tens digit and the units digit of c_n.

Consider the polynomial:

$$(x-(3+\sqrt{5}))(x-(3-\sqrt{5})) = x^2 - 6x + 4$$

It leads to the recurrence relation: $f_n = 6f_{n-1} - 4f_{n-2}$, for which any linear combination of a_n and b_n is a solution [3]. We set the initial values of f_n such that the linear combination $c_n = a_n + b_n$ is the solution: $f_0 = 2, f_1 = 6$.
Therefore c_n satisfies the recurrence:

$$c_n = 6c_{n-1} - 4c_{n-2}$$
$$c_0 = 2$$
$$c_1 = 6$$

In theory we could just use this recurrence to compute c_n and then extract the hundreds digit, the tens digit and the units digit. Unfortunately this is not feasible for large n, since c_n grows quickly to very large values. But since we only need the last three digits of the values, we don't need to compute the values completely, computing them modulo 1000 will suffice.

Fortunately according to modulo arithmetic, the recurrence relation for c_n is still valid when doing modulo 1000. Let:

$$d_n \equiv c_n \mod 1000$$

and so

$$d_n \equiv 6d_{n-1} - 4d_{n-2} \mod 1000$$

Now consider the ordered pairs $(d_n, d_{n+1}), n \in \mathbb{N}$. Because $d_n \in \{0,1,2,\ldots,999\}$, there are only 10^6 distinct pairs of (d_n, d_{n+1}) possible. So it must be that there exist two indices $i, j \in \mathbb{N}^+$ such that:

$$(d_i, d_{i+1}) = (d_j, d_{j+1})$$

From the recurrence it follows that:

$$\forall k \in \mathbb{N} : (d_{i+k}, d_{i+k+1}) = (d_{j+k}, d_{j+k+1})$$

[3] In-depth treatment of recurrence relations can be found in Chapter 10, Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, 3rd edition, 1993. ISBN 0201549832

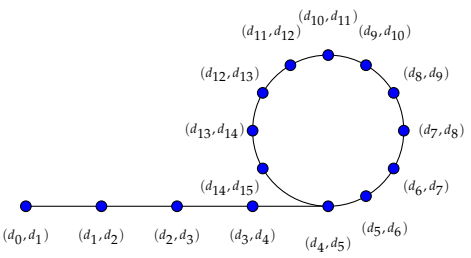


Figure 18.1: Periodic sequence of pairs preceded by a prefix of pairs.

The sequence of ordered pairs (d_n, d_{n+1}) is periodic with a period p of at most 10^6. We can construct a lookup table holding values of d_n from one period p and then compute d_n for large n by going into the lookup table at $n \mod p$.

The periodic part of the sequence doesn't necessarily start with the first pair in the sequence or with the second or the third etc... There might be a sequence prefix of ordered pairs that don't repeat before it goes into the sequence loop of repeating pairs. We write a function that computes the prefix and the period of the sequence using Floyd's cycle finding algorithm[4].

Luckily it turns out that in this case the prefix is $2, 6, 28$ and the periodic sequence has period 100.

With the lookup table we can compute d_n for any large n in constant time. d_n gives us the last three digits of c_n. Our goal though was to compute the last three digits before the decimal point of a_n.

We know $c_n = \lceil a_n \rceil$, so $c_n - 1 = \lfloor a_n \rfloor$. This means that to get the digits for a_n, we need to extract them from $d_n - 1$. Listing 18.1 has the complete Haskell implementation.

[4] Floyd's algorithm is described at https://en.wikipedia.org/wiki/Cycle_detection#Tortoise_and_hare. We use the Haskell implementation from https://wiki.haskell.org/Floyd's_cycle-finding_algorithm

Listing 18.1: Haskell code to compute last 3 digits

```
module Last3Digits(
    compute
) where

f :: Int -> Int -> Int
f a b = (6 * b - 4 * a) `mod` 1000

-- ds is our sequence of d_n
ds = 2 : 6 : zipWith (f) ds (tail ds)
-- this gives us the pairs
dps = zip ds (tail ds)

findCycle :: Eq a => [a] -> ([a],[a])
findCycle xxs = fCycle xxs xxs
  where fCycle (x:xs) (_:y:ys)
         | x == y             = fStart xxs xs
         | otherwise          = fCycle xs ys
        fCycle _      _       = (xxs,[]) -- not cyclic
        fStart (x:xs) (y:ys)
         | x == y             = ([], x:fLength x xs)
         | otherwise          = let (as,bs) = fStart xs ys in (x:as,bs)
        fLength x (y:ys)
         | x == y             = []
         | otherwise          = y:fLength x ys

tps = findCycle dps
-- ps is the prefix, cs the cycle of d_n
(ps, cs) = (map fst (fst tps), map fst (snd tps))
```

```
computeAux :: Int -> Int
computeAux n
    | n < (length ps) = ps!!n
    | otherwise       = cs!!((n - (length ps)) `mod` (length cs))

compute :: Int -> Int
compute n = computeAux n - 1
```

To check our computation we can use this Mathematica function:

```
In[14]:= last3Digits[n_Integer] := Mod[IntegerPart[(3 + Sqrt[5])^n], 1000]
```

19
How many trailing zeros in n!

Greatest dividing exponent and its properties is the topic of the problem in this note.

Problem

Write a program that calculates for an arbitrary positive integer n how many trailing zeros there are in $n!$.

Let's first try to figure out for any natural number n what the number of trailing zeros is. A useful concept here is the greatest dividing exponent [1]:

[1] Eric W. Weisstein. Greatest dividing exponent. From MathWorld—A Wolfram Web Resource. URL http://mathworld.wolfram.com/GreatestDividingExponent.html

Definition 19.1. The greatest dividing exponent $gde(n, b)$ of a base b with respect to a number n is the largest integer value of k such that $b^k \mid n$, where $b^k \leq n$.

Lemma 19.2.

$$gde(n, ab) = min(gde(n, a), gde(n, b)), \text{ with } (a, b) = 1$$

Proof. Assume $gde(n, a) \leq gde(n, b)$. Then $a^{gde(n,a)} \mid n$ and $b^{gde(n,a)} \mid n$, with $(a^{gde(n,a)}, b^{gde(n,a)}) = 1$, so $(ab)^{gde(n,a)} \mid n$. By definition of gde we then have $gde(n, a) \leq gde(n, ab)$.

We also have $(ab)^{gde(n,ab)} \mid n$, so $a^{gde(n,ab)} \mid n$. By definition of gde we then have $gde(n, a) \geq gde(n, ab)$.

It follows that $gde(n, a) = gde(n, ab)$. □

It's clear that the number of trailing zeros of n equals $gde(n, 10)$. From lemma 19.2 we are looking for $min(gde(n!, 2), gde(n!, 5))$.

Lemma 19.3.

$$gde(n!, p) = \sum_{k=1}^{\lfloor log_p n \rfloor} \left\lfloor \frac{n}{p^k} \right\rfloor, \textit{for a prime } p \leq n$$

Proof. We define the following subsets of $\{1, \ldots, n\}$:

$$M_p^k = \{i : 1 \leq i \leq n : p^k \mid i\}$$

For $k > \lfloor log_p n \rfloor$ the sets M_p^k are empty, so we only consider $k \leq \lfloor log_p n \rfloor$. Each member of one set M_p^k contributes k to $gde(n!, p)$, so the whole set contributes $k|M_p^k|$. From $p^k \mid i$ it follows that also $p^{k-1} \mid i$, so $M_p^k \subseteq M_p^{k-1}$ for $k = 2, \ldots, \lfloor log_p n \rfloor$. Being careful not to count the contributions more than once we get:

$$gde(n!, p) = \sum_{k=1}^{\lfloor log_p n \rfloor} |M_p^k|$$

With $|M_p^k| = \left\lfloor \frac{n}{p^k} \right\rfloor$ we conclude the proof. □

Lemma 19.4.

$$gde(n!, 2) \geq gde(n!, 5) \textit{ for any } n \geq 1$$

Proof. Plugging in the expression of *gde* from lemma 19.3 into the claim of this lemma we get:

$$gde(n!, 2) \geq gde(n!, 5) \Leftrightarrow \sum_{k=1}^{\lfloor log_2 n \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor \geq \sum_{k=1}^{\lfloor log_5 n \rfloor} \left\lfloor \frac{n}{5^k} \right\rfloor$$

We establish:

$$\begin{aligned} log_2 n \geq log_5 n &\Leftrightarrow log_2 n \geq log_2 n \; log_5 2 \\ &\Leftrightarrow 1 \geq log_5 2, \text{ which is true} \end{aligned}$$

For each $1 \leq k \leq \lfloor log_5 n \rfloor$ we have:

$$\left\lfloor \frac{n}{2^k} \right\rfloor \geq \left\lfloor \frac{n}{5^k} \right\rfloor$$

and for $\lfloor log_5 n \rfloor + 1 \leq k \leq \lfloor log_2 n \rfloor$ we have:

$$\left\lfloor \frac{n}{2^k} \right\rfloor > 0$$

Adding up the inequalities establishes the claim. □

From the three lemmas we found that:

$$\begin{aligned} \text{(number of trailing zeros in } n!) &= gde(n!, 10) \\ &= min(gde(n!, 2), gde(n!, 5)) \\ &= gde(n!, 5) \\ &= \sum_{k=1}^{\lfloor log_5 n \rfloor} \left\lfloor \frac{n}{5^k} \right\rfloor \end{aligned}$$

so our program needs to calculate the expression:

$$\sum_{k=1}^{\lfloor log_5 n \rfloor} \left\lfloor \frac{n}{5^k} \right\rfloor$$

The following small Haskell function does it:

Listing 19.1: Haskell code

```
gdefac :: Int -> Int

gdefac n = fst (until (\(x, y) -> y == 0)
                      (\(x, y) -> let
                                    y' = div y 5
                                  in (x + y', y'))
                      (0, n))
```

It works on tuples of numbers. It keeps dividing the second number in the tuple by 5 until zero and adding the division results together into the first number of the tuple. In the end it returns the first number in the tuple.

20
Twelve Coins

Coin weighings are the topics of the problem [1] in this note.

[1] Ethan Canin. *The Palace Thief Stories*, chapter Batorsag and Szerelem, page 87. Random House New York, 1994
or
Problem 1-111. on page 47 in N. Loehr. *Combinatorics*. Discrete Mathematics and Its Applications. CRC Press, 2017. ISBN 9781498780278

Problem

Of twelve coins, one is counterfeit and weighs either more or less than all the others. The others weigh the same. With a balance scale, on which one side may be weighed against the other, you are to use only three weighings to determine the counterfeit and its type (lighter or heavier).

We first present a hand-tailored solution for twelve coins.
Let M be the set of coins, $|M| = 12$. We have weighing function

$$w : M \to \{a, b\},\ a \neq b,\ a, b \in \mathbb{R}^+.$$

We have $|\{c \in M : w(c) = a\}| = 11$ and $|\{c \in M : w(c) = b\}| = 1$. We are asked to find $c_f \in M$ with $w(c_f) = b$ in three weighings.

For a subset $S \subseteq M$ we define

$$w(S) = \sum_{c \in S} w(c).$$

Let's partition M into 3 subsets S_0, S_1, S_2

$$\begin{aligned} &S_0 \cup S_1 \cup S_2 = M \\ &\forall\ 0 \leq i < 3 : |S_i| = 4 \\ &\forall\ 0 \leq i < j < 3 : S_i \cap S_j = \emptyset \end{aligned}$$

At this point we consume the first weighing:

1st weighing: compare $w(S_1)$ with $w(S_2)$

There are many variations of coin weighing problems. Some require identifying the type of the counterfeit (lighter or heavier), some don't. Some have more than one scale, some have more than one counterfeit, some don't state the existence of the counterfeit, some allow using scale weights or a known genuine coin. The Wikipedia page `https://en.wikipedia.org/wiki/Balance_puzzle` has a good overview. In this section we always want to find the counterfeit coin (we know there is exactly one counterfeit of unknown type) and its type and our scale is a balance scale with coins on both sides.

Case $w(S_1) = w(S_2)$

In this case $c_f \in S_0$. We partition S_0 into $S_0 = S_0^1 \cup S_0^3$ with $|S_0^1| = 1$ and $|S_0^3| = 3$. We also consider S_1^3, a subset of S_1 with $|S_1^3| = 3$. We consume the second weighing:

2nd weighing: compare $w(S_0^3)$ with $w(S_1^3)$

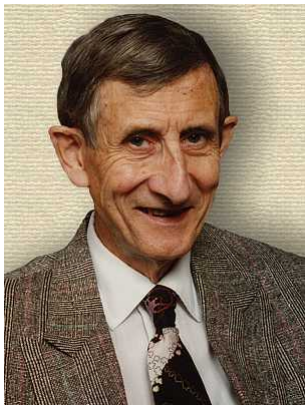
subcase 1: $w(S_0^3) = w(S_1^3)$. In this subcase $c_f \in S_0^1$ and we're done after just two weighings.

subcase 2: $w(S_0^3) > w(S_1^3)$. In this subcase S_0^3 has the counterfeit coin and $b > a$. We consume the third weighing: Let $S_0^3 = \{c_1, c_2, c_3\}$. We weigh c_1 against c_2.

3rd weighing: compare $w(c_1)$ with $w(c_2)$

If $w(c_1) = w(c_2)$ then $c_f = c_3$, if $w(c_1) > w(c_2)$ then $c_f = c_1$.

case 3: $w(S_0^3) < w(S_1^3)$. In this case S_0^3 has the counterfeit coin and $b < a$. Analog to previous case (replace heavy with light).



Freeman J. Dyson. https://en.wikipedia.org/wiki/Freeman_Dyson

Case $w(S_1) > w(S_2)$

In this case the counterfeit coin is either in S_1 or in S_2.
We consider 4 subsets:

$S_0^3 \subset S_0, |S_0^3| = 3$,

A with three coins from S_1,

B with one coin from S_2,

C with remaining coin from S_1: $C = S_1 \setminus A$.

We consume the second weighing:

2nd weighing: compare $w(S_0^3 \cup C)$ with $w(A \cup B)$

subcase 1: $w(S_0^3 \cup C) = w(A \cup B)$. In this subcase $c_f \in S_2 \setminus B$ and because $w(S_1) > w(S_2)$ we know that $b < a$. Let $\{c_1, c_2, c_3\} = S_2 \setminus B$ and we consume third weighing:

3rd weighing: compare $w(c_1)$ with $w(c_2)$

If $w(c_1) = w(c_2)$ then $c_f = c_3$, if $w(c_1) > w(c_2)$ then $c_f = c_2$.

subcase 2: $w(S_0^3 \cup C) < w(A \cup B)$. Assume $c_f \in C \subset S_1$. That would mean that $b > a$ because $w(S_1) > w(S_2)$ but that contradicts with $w(S_0^3 \cup C) = 3a + b < w(A \cup B) = 4a$. Assume $c_f \in B \subset S_2$. That

would mean that $b < a$ because $w(S_1) > w(S_2)$ but that contradicts also with $w(S_0^3 \cup C) = 4a < w(A \cup B) = 3a + b$. The only possibility remaining is $c_f \in A$. We use the ***third weighing*** analog to the previous case to find the counterfeit coin in a three-coin set using the fact that $b > a$.

subcase 3: $w(S_0^3 \cup C) > w(A \cup B)$. In this case the counterfeit coin can be either in B or in C. It cannot be in A according to a reasoning analog to previous case that leads to a contradiction. Both B and C only have one coin each so compare the coin in B with any good coin to find the counterfeit coin in this ***third weighing***.

This covers all the cases and we're done.

The solution brings up questions: how would it work for 13 coins and in general, how many weighings would be necessary for N coins. To answer these questions we present an elegant general solution[2] invented by Freeman J. Dyson.

[2] Freeman J. Dyson. Note 1931-The problem of the pennies. *Math. Gaz.*, 30:231–234, 1946
We follow an exposition of this solution by G. Shestopal.

In the hand-tailored solution for twelve coins we kept seeing a partition of coin sets into 3 subsets, two equally sized subsets that were weighed against each other and one subset that didn't participate in the current weighing. Depending on the result of the weighing and prior information from previous weighings we could narrow the search. Each weighing seems to contribute roughly three pieces of "information". A sequence of n weighings would generate 3^n amount of information that should somehow map to finding the counterfeit amongst N coins. This is not strictly accurate[3] because information deduced from weighings uses prior information from weighings before and the narrowing doesn't completely discard sets of coins but instead uses them as scale weights with known type (ie none are counterfeits). But this 3^n observation does suggest $3^n \approx N$ and point to the objective of finding a bijection between ternary codewords of length n and a set of size N.

[3] The hand-tailored solution is an **adaptive** solution: later weighings are set up according to the result of earlier weighings. Freeman J. Dyson's general solution is a **non-adaptive** solution if the number of coins is divisible by three: all the weighings are pre-determined regardless of their outcome.

Lets first consider N coins with N of the form $N = \frac{1}{2}(3^n - 3)$ for some $n \geq 2$. We will show how to find the counterfeit and its type in n weighings[4].

[4] Notice that $N = 12$ fits this form: $12 = \frac{1}{2}(3^3 - 3)$, so $n = 3$ weighings.

The set of codewords of length n from alphabet $\{0, 1, 2\}$ has size 3^n. We discard the three codewords with all digits equal: $0 \ldots 0$, $1 \ldots 1$ and $2 \ldots 2$. The set W of the remaining codewords has size $3^n - 3$. We split[5] W into $\frac{3^n-3}{2}$ pairs of complements:

[5] The expression $3^n - 3$ is the subtraction of two odd numbers, so it is even.

Definition 20.1. Two codewords $a_1 a_2 \ldots a_n \in W$ and $b_1 b_2 \ldots b_n \in W$ are **complements** of each other if

$$\forall i : 1 \leq i \leq n : a_i + b_i = 2$$

We use the notation $a_1^c a_2^c \ldots a_n^c$ for the complement of $a_1 a_2 \ldots a_n$.

Definition 20.2. The function $\delta : W \mapsto \{0,1,2\}^2$ finds the first two digits of a codeword that differ[6]

$$\delta(a_1a_2\dots a_n) = a_ia_{i+1} \text{ such that } \forall 1 \leq j < i : a_j = a_i \text{ and } a_i \neq a_{i+1}$$

[6] The function δ is well-defined because $0\dots0 \notin W$, $1\dots1 \notin W$ and $2\dots2 \notin W$.

Definition 20.3. Codeword $w \in W$ is called a **left** codeword if $\delta(w) \in \{10, 21, 02\}$ otherwise it is called a **right** codeword.

It is easy to see that the set of left codewords and the set of right codewords do not overlap and that if a codeword is a left codeword its complement is a right codeword[7]. A good mnemonic of left and right is this: $0 \to 1 \to 2$ moves digits to the right so $\{01, 12, 20\}$ corresponds to the right codewords and the right direction. Conversely $0 \leftarrow 1 \leftarrow 2$ moves digits to the left so $\{10, 21, 02\}$ corresponds to the left codewords and the left direction.

To summarize we now have $\frac{3^n-3}{2}$ pairs of codewords from W such that each pair has a left codeword and a right codeword that are complements of each other. Let P be the set of these pairs. We have $N = \frac{3^n-3}{2}$ coins. Let C be the set of coins. We pick an arbitrary bijection $\mu : C \mapsto P$ that assigns a pair to a coin, marking it with a left and a right codeword[8].

[7] For example let's prove that if $a_1a_2\dots a_n$ is a right codeword then $a_1^ca_2^c\dots a_n^c$ is a left codeword. If $a_j = a_{j+1}$ then $a_j^c = a_{j+1}^c$ and likewise if $a_j \neq a_{j+1}$ then $a_j^c \neq a_{j+1}^c$. So the index of the first two digits that differ is the same for a codeword and its complement. The complements of $\{01, 12, 20\}$ are exactly $\{21, 10, 02\}$ so the complement of a right codeword is a left codeword with differing digits at the same index as the right codeword.

We adopt the notation $\mu(c).right$ to denote the right codeword assigned to coin $c \in C$ and $\mu(c).left$ the left codeword. Also $\mu(c).right(i)$ denotes the i-th digit of the right codeword, so if $\mu(c).right = a_1a_2\dots a_n$ then $\mu(c).right(i) = a_i$. Analog $\mu(c).left(i)$ for the i-th digit of the left codeword assigned to coin c.

We define $\forall i : 1 \leq i \leq n$ and $\forall d : 0 \leq d \leq 2$ the subsets $C_i(d) \subset C$ with

$$C_i(d) = \{c \in C : \mu(c).right(i) = d\}$$

It is easy to see that $C = C_i(0) \cup C_i(1) \cup C_i(2)$ for each $1 \leq i \leq n$ and that $C_i(0) \cap C_i(1) = C_i(0) \cap C_i(2) = C_i(1) \cap C_i(2) = \emptyset$, so $C_i(0)$, $C_i(1)$ and $C_i(2)$ partition C.

Now we execute n weighings. For the i-th weighing we place the coins from $C_i(0)$ on the left pan of the scale and the coins from $C_i(2)$ on the right pan of the scale. We capture the n weighings in a codeword $x_1x_2\dots x_n \in W$: if in the i-th weighing the left pan sinks, then $x_i = 0$, if the weighing is balanced then $x_i = 1$ and if the right pan sinks then $x_i = 2$.

We claim that the weighing codeword $x_1x_2\dots x_n$ will be either the left or the right marker codeword of the counterfeit depending whether the counterfeit is lighter or heavier and prove the following theorem:

Theorem 20.4. *Let $x_1x_2\dots x_n$ be the result of the n weighings and let $c_f \in C$ be the counterfeit coin. Then $\mu(c_f).right = x_1x_2\dots x_n$ if c_f is heavier and $\mu(c_f).left = x_1x_2\dots x_n$ if c_f is lighter.*

[8] For example when $N = 12$ we could pick μ like this:

coin	left codeword	right codeword
c_1	211	011
c_2	100	122
c_3	022	200
c_4	212	010
c_5	101	121
c_6	020	202
c_7	210	012
c_8	102	120
c_9	021	201
c_{10}	221	001
c_{11}	110	112
c_{12}	002	220

$C_3(2)$ means the subset of coins for which the third digit in the right codeword is 2. In this example $C_3(2) = \{c_2, c_6, c_7, c_{11}\}$.

Proof. Assume the counterfeit c_f is lighter than the other coins (we deal with the case of c_f heavier afterwards).

There are three cases for the i-th weighing:

Case 1.(lighter) The scale is balanced, so $x_i = 1$. This means that c_f does not participate in the i-th weighing[9]: $c_f \notin C_i(0)$ and $c_f \notin C_i(2)$. Since $C_i(0)$, $C_i(1)$ and $C_i(2)$ partition C, we have $c_f \in C_i(1)$. By definition this means that $\mu(c_f).right(i) = x_i = 1$ and since right and left are complements it also means that $\mu(c_f).left(i) = x_i = 1$.

[9] Otherwise the scale wouldn't be balanced.

Case 2.(lighter) The left pan sinks, so $x_i = 0$. Here c_f participates and $c_f \in C_i(2)$ because it is lighter so on the right pan. By definition this means that $\mu(c_f).right(i) = 2$ and since $\mu(c_f).left(i)$ is the complement it also means $\mu(c_f).left(i) = x_i = 0$.

Case 3.(lighter) The pan sinks, so $x_i = 2$. In this case c_f also participates and $c_f \in C_i(0)$ because it is lighter so on the left pan. This means $\mu(c_f).right(i) = 0$ and by complement $\mu(c_f).left(i) = x_i = 2$.

For every i we have seen that $\mu(c_f).left(i) = x_i$, so $\mu(c_f).left = x_1x_2 \dots x_n$.

Now assume c_f is heavier. We proceed in analog fashion. There are three cases for the i-th weighing:

Case 1.(heavier) The scale is balanced, so $x_i = 1$. This means that c_f does not participate in the i-th weighing and we have $c_f \in C_i(1)$ and $\mu(c_f).right(i) = x_i = 1$.

Case 2.(heavier) The left pan sinks, so $x_i = 0$. Here c_f participates and $c_f \in C_i(0)$ because it is heavier so on the left pan. Then $\mu(c_f).right(i) = x_i = 0$.

Case 3.(heavier) The right pan sinks, so $x_i = 2$. In this case c_f also participates and $c_f \in C_i(2)$ because it is heavier so on the right pan. So $\mu(c_f).right(i) = x_i = 2$.

We see that for every i we have $\mu(c_f).right(i) = x_i$, so $\mu(c_f).right = x_1x_2 \dots x_n$. □

Theorem 20.4 shows that the n weighings detect the counterfeit coin and its type if there are $N = \frac{3^n-3}{2}$ coins.

If $N < \frac{3^n-3}{2}$ we have to be more careful how we mark coins with codewords[10].

Let $\pi : W \mapsto W$ be the function on the set of codewords W with $\pi(a_1a_2 \dots a_n) = b_1b_2 \dots b_n$ such that $\forall i : 1 \leq i \leq n : b_i \equiv (a_i + 1) \mod 2$.

Lemma 20.5. *The function π preserves the rightiness of codewords: if $w \in W$ is a right codeword, then $\pi(w)$ is also a right codeword.*

Proof. Let $a_1a_2 \dots a_n$ be a right codeword. By definition it means that $\delta(a_1a_2 \dots a_n) = a_ia_{i+1}$ with $\forall 1 \leq j < i : a_j = a_i$ and $a_ia_{i+1} \in \{01, 12, 20\}$. Let $\pi(a_1a_2 \dots a_n) = b_1b_2 \dots b_n$. This means that $\forall 1 \leq j < i : b_j \equiv a_j + 1 \equiv a_i + 1 \equiv b_i \mod 2$. So $\delta(b_1b_2 \dots b_n) = b_ib_{i+1}$. If $a_ia_{i+1} = 01$

[10] For example assume there are only ten coins and we arbitrarily assign the following codeword pairs:

coin	left codeword	right codeword
c_1	211	011
c_2	100	122
c_3	022	200
c_4	212	010
c_5	101	121
c_6	020	202
c_7	210	012
c_8	102	120
c_9	021	201
c_{10}	221	001

Then $C_2(2) = \{c_2, c_5, c_8\}$ and $C_2(0) = \{c_3, c_6, c_9, c_{10}\}$. This is a problem because the two sets that will be put on the scale in the two pans in the second weighing have different number of coins. We cannot deduce any information from this weighing anymore.

then $b_i b_{i+1} = 12$, if $a_i a_{i+1} = 12$ then $b_i b_{i+1} = 20$ and if $a_i a_{i+1} = 20$ then $b_i b_{i+1} = 01$. In all three cases $b_1 b_2 \dots b_n$ is also a right codeword. □

From the definition of π it is clear that $\pi^3(w) = w$. From this and lemma 20.5 it follows that π partitions the set of right codewords into subsets of size three: $\{w, \pi(w), \pi^2(w)\}$.

We now partition W into subsets of size six:

$$\{w, w^c, \pi(w), (\pi(w))^c, \pi^2(w), (\pi^2(w))^c\}$$

grouping π-generated right codewords and their left complements. The group with right codewords $00\dots01$, $11\dots12$ and $22\dots20$ we set aside and call the left-over group.

For the bijection $\mu : C \mapsto W$ we group coins in groups of three. For each group we pick a codeword group of six other than the left-over group and assign left and right complementing codewords to each coin in the group. If there are one or two coins left over from the grouping into threes, then we use the left-over codeword group to assign left and right codewords to those coins. If only one coin is left over we assign it the right codeword $11\dots12$ and if two are left over we assign the right codewords $00\dots01$ and $22\dots20$.

With μ defined this way if $N \equiv 1 \mod 3$ then the left-over coin has right codeword $11\dots12$ and will not participate in the first $n-1$ weighings. If $N \equiv 2 \mod 3$ the two left-over coins with right codewords $00\dots01$ and $22\dots20$ will both participate in every of the first $n-1$ weighings. This ensures that the left-over coins don't disrupt the first $n-1$ weighings. For a coin that is not in the left-over group it is easy to see that if it participates in a weighing then there is another coin in the same group (apply π twice to get to its right codeword) that also participates on the opposite pan. Overall we have satisfied the requirement $|C_i(0)| = |C_i(2)|$ for all $1 \le i < n$. The first $n-1$ weighings can proceed as before.

For the last weighing we have the following cases (here the solution turns adaptive, ie the setup for the last weighing depends on the outcome of the previous weighings):

The first two cases are for $N \equiv 1 \mod 3$, so one left-over coin c_l with right codeword $\mu(c_l).right = 11\dots12$.

Case 1. The first $n-1$ weighings yielded $x_1 x_2 \dots x_{n-1} = 11\dots1$ (the scale was balanced in all $n-1$ weighings). In this case we know the left-over coin is the counterfeit $c_l = c_f$ and we can just put it on one pan on the scale and any other coin on the other to determine its type.

Case 2. The first $n-1$ weighings yielded $x_1 x_2 \dots x_{n-1} \neq 11\dots1$. In this case we know the left-over coin is not the counterfeit. We can just leave it out and put $C_n(0)$ on the left pan and $C_n(2) \setminus \{c_l\}$ on the right

pan of the balance scale. The resulting complete weighing codeword will point to the counterfeit and its type.

The next cases are for for $N \equiv 2 \mod 3$, with two left-over coins c_{l0} and c_{l2} with right codewords $\mu(c_{l0}).right = 00\ldots01$ and $\mu(c_{l2}).right = 22\ldots20$.

Case 3. The first $n-1$ weighings yielded $x_1x_2\ldots x_{n-1} = 22\ldots2$. Both c_{l0} and c_{l2} participated in all $n-1$ weighings (according to their right codewords that start with $00\ldots0$ and $22\ldots2$ respectively). It means that either c_{l0} or c_{l2} is the counterfeit. In the last weighing we pit c_{l0} against any coin that is not c_{l2}. If the scale is balanced then c_{l2} is the counterfeit and it is heavier. If the scale tilts one way or the other it shows c_{l0} as a counterfeit and its type.

Case 4. The first $n-1$ weighings yielded $x_1x_2\ldots x_{n-1} = 00\ldots0$. Again both c_{l0} and c_{l2} participated in all $n-1$ weighings and again it means that either c_{l0} or c_{l2} is the counterfeit. In the last weighing we pit c_{l2} against any coin that is not c_{l0}. If the scale is balanced then c_{l0} is the counterfeit and it is heavier. If the scale tilts one way or the other it shows c_{l2} as a counterfeit and its type.

Case 5. The first $n-1$ weighings yielded $x_1x_2\ldots x_{n-1} \neq 00\ldots0$ and $x_1x_2\ldots x_{n-1} \neq 22\ldots2$. In this case both c_{l0} and c_{l2} cannot be counterfeits. In the last weighing we can just do $C_n(0)$ on the left pan and $C_n(2)$ on the right. The resulting complete weighing codeword will point to the counterfeit and its type.

This covers all the cases and shows that we can find the counterfeit coin and its type in n weighings if the number of coins $N \leq \frac{3^n-3}{2}$.

Is this optimal or does a strategy exist that needs less than n weighings?

To answer this question we want to find a lower bound for the number of weighings given N coins.

Theorem 20.6. *Given are N coins of which one is a counterfeit. If the counterfeit coin and its type can be found in n weighings using a non-adaptive strategy, then $2N \leq 3^n - 3$.*

Proof. There are n weighings producing a weighing codeword $x_1x_2\ldots x_n$ with $x_i = 0$ if left pan sinks in i-th weighing, $x_i = 1$ if i-th weighing is balanced and $x_i = 2$ if right pan sinks. We have 3^n possible codewords from n weighings.

Assume we have a non-adaptive strategy that finds the counterfeit coin and its type in n weighings. Since it is non-adaptive the information of which coin participates in which weighing on which pan is already pre-determined. Let's capture this information in a matrix $p \in \{0,1,2\}^{n\times N}$ which we call the **participation matrix**:

$$\forall 1 \leq i \leq n, 1 \leq j \leq N :$$
$$p_{ij} = \begin{cases} 0, & \text{coin } j \text{ on left pan in i-th weighing} \\ 1, & \text{coin } j \text{ not in i-th weighing} \\ 2, & \text{coin } j \text{ on right pan in i-th weighing} \end{cases}$$

The number of potential answers to the question of which coin is the counterfeit and what is its type is $2N$ (N coins and two possibilities for each coin - lighter or heavier). For notational convenience we define the index sequence $N_{\pm} = \{1, -1, 2, -2, \ldots, N, -N\}$. For all $1 \leq j \leq N$ index j means coin j is the counterfeit and it is heavier, index $-j$ means coin j is the counterfeit and it is lighter.

Given the participation matrix we define a matrix $a \in \{0, 1, 2\}^{n \times N_{\pm}}$. Cell a_{ij} tells us what result of weighing i keeps the answer that coin j is the heavier counterfeit as still a possibility. Cell $a_{i(-j)}$ tells us what result of weighing i keeps the answer that coin j is the lighter counterfeit as still a possibility. For example if $p_{ij} = 0$ then coin j is on the left pan in the i-th weighing and for coin j to be the counterfeit and heavier in the i-th weighing the left pan needs to sink, the corresponding weighing codeword component needs to be $x_i = 0$ and $a_{ij} = 0$.

In other words matrix a shows for each potential answer what needs to happen in each weighing so that the answer becomes the actual answer:

$$\forall 1 \leq i \leq n, 1 \leq j \leq N :$$
$$a_{ij} = \begin{cases} 0, & \text{if } p_{ij} = 0 \\ 1, & \text{if } p_{ij} = 1 \\ 2, & \text{if } p_{ij} = 2 \end{cases} \quad a_{i(-j)} = \begin{cases} 0, & \text{if } p_{ij} = 2 \\ 1, & \text{if } p_{ij} = 1 \\ 2, & \text{if } p_{ij} = 0 \end{cases}$$

The column j in matrix a is the weighing codeword required for making coin j and its type (from the sign of j) the actual answer.

For the strategy (represented by the participation matrix p) to work it needs to not rely on "luck", i.e. a given weighing codeword happens to be a column in matrix a. All the weighing codewords that can occur need to be columns in a exactly once.

Since the strategy is successful we know that each coin participates in at least one weighing[11], so the column vector $(1, 1, \ldots, 1)^T$ does not appear in p. This means that weighing codeword $x1x2 \ldots x_n = 11 \ldots 1$ cannot occur and we have $3^n - 1$ remaining weighing codewords that can happen.

[11] Otherwise there would be no way to find its type even if we find the counterfeit.

If $2N > 3^n - 1$ then it must be that a weighing codeword will appear more than once in a, so more than one potential answer could be the actual answer and we wouldn't know which one. So $2N \leq 3^n - 1$.

We observe that in each row i of a if $a_{ij} = 0$ then $a_{i(-j)} = 2$ and vice versa. That means that there are k_i zeros and k_i twos in row i, so

$2N - 2k_i$ ones. Also because an even number of coins participates in each weighing we have that k_i is even. So in each row we have an even number of zeros, ones and twos. There are 3^{n-1} possible weighing codewords that start with a one. That means at most 3^{n-1} columns in a can have a one in the first position. Since the number of ones is even and 3^{n-1} is odd we can have at most $3^{n-1} - 1$ ones in the first row. Similarly there are 3^{n-1} possible weighing codewords that start with a zero, so $k_1 \leq 3^{n-1} - 1$. We then have:

$$2N = 2N - 2k_1 + 2k_1 \leq 3^{n-1} - 1 + 2(3^{n-1} - 1) = 3^n - 3$$

□

Explain connection to Hamming codes for number of columns smaller than $3^n - 3$

21
Two decks of cards

Inclusion–exclusion principle and the number of derangements are the topics of the problem [1] in this section.

[1] Probability question on page ix in Preface of M. Beck and R. Geoghegan. *The Art of Proof: Basic Training for Deeper Mathematics*. Undergraduate Texts in Mathematics. Springer New York, 2010. ISBN 9781441970237

Problem

A deck of n different cards is shuffled and laid on the table by your left hand, face down. An identical deck of cards, independently shuffled, is laid at your right hand, also face down. You start turning up cards at the same rate with both hands, first the top card from both decks, then the next-to-top cards from both decks, and so on. What is the probability that you will simultaneously turn up identical cards from the two decks?

The shuffling implies equally likely outcomes so the probability is the number of outcomes with an identical card turning up divided by the number of total outcomes. The number of total outcomes is $(n!)^2$ since there are $n!$ possible shuffling outcomes of one deck (the number of permutations of S_n).

The set of outcomes where an identical card turns up seems harder to count. It feels easier to count its complement: the number of outcomes when no identical card comes up. There are $n!$ ways in which the first deck is shuffled. For a given permutation $\pi \in S_n$ of the first deck we need to count all the permutations $\rho \in S_n$ of the second deck for which $\forall i : 1 \leq i \leq n : \rho(i) \neq \pi(i)$. Let $A_\pi = \{\rho \in S_n : \forall i : 1 \leq i \leq n : \rho(i) \neq \pi(i)\}$.

Let $D_n = \{\tau \in S_n : \forall i : 1 \leq i \leq n : \tau(i) \neq i\}$. A permutations from D_n is called a **derangement**. We introduce the notation $!n = |D_n|$ for the number of derangements.

Lemma 21.1.

$$|A_\pi| = |D_n|$$

Proof. We need to present a bijection $f : D_n \to A_\pi$. We define $f(\tau) = \pi \circ \tau$. First we verify that f is well-defined, i.e. $f(\tau) \in A_\pi$.

$$\begin{aligned} &\forall i : 1 \leq i \leq n : \\ f(\tau)(i) &= (\pi \circ \tau)(i) \\ &= \pi(\tau(i)) \neq \pi(i) \\ &\text{because } \tau(i) \neq i \end{aligned}$$

Next we show that f is injective: $f(\tau_1) = f(\tau_2)$ implies $\pi \circ \tau_1 = \pi \circ \tau_2$. S_n is a group, so $\tau_1 = \tau_2$. Also f is surjective because: $\forall \rho \in A_\pi$ we have $\pi^{-1} \circ \rho \in D_n$ because $\rho(i) \neq \pi(i)$ implies $(\pi^{-1} \circ \rho)(i) \neq i$. $f(\pi^{-1} \circ \rho) = \rho$. □

From lemma 21.1 we now know that the number of outcomes when no identical card comes up is $!n \cdot n!$ and the probability requested in the problem is

$$P = 1 - \frac{!n \cdot n!}{n!^2} = 1 - \frac{!n}{n!}$$

What remains is to compute $!n$. Let us look again at the set of derangements: $D_n = \{\tau \in S_n : \forall i : 1 \leq i \leq n : \tau(i) \neq i\}$. It sometimes helps to consider the complement of a set when we have to compute its cardinality. To more precisely define the complement of D_n we will define the following subsets of S_n: $F_n(k) = \{\tau \in S_n : \tau(k) = k\}$. We then have:

$$D_n = S_n \setminus (\bigcup_{k=1}^{n} F_n(k))$$

For any given $k \in \{1, 2, \ldots, n\}$ we have $|F_n(k)| = (n-1)!$ (see footnote[2] why), so

$$|(\bigcup_{k=1}^{n} F_n(k))| = n(n-1)! = n!$$

[2] One position is fixed and the rest behave like a permutation in S_{n-1}.

But this can't be right. It would mean that $|D_n| = 0$ and $D_n = \emptyset$. But cleary D_n is not empty, for example the permutation:

$$\rho(i) = \begin{cases} i+1 & i < n \\ 1 & i = n \end{cases}$$

is a member of D_n. The problem here is that the $F_n(k)$ are not disjoint, so calculating the size of their union needs to be done more carefully. It turns out that this is a perfect use case of the inclusion-exclusion principle.

The **inclusion-exclusion principle** provides a method of counting the size of the union of subsets that are not necessarily disjoint.

We illustrate the method on a simple example of three sets A, B, C as in Figure 21.1. We would like to compute $|A \cup B \cup C|$. The expression $|A| + |B| + |C|$ would count the elements from $(A \cap B) \setminus (A \cap B \cap C)$, $(A \cap C) \setminus (A \cap B \cap C)$ and $(B \cap C) \setminus (A \cap B \cap C)$ twice and elements from $A \cap B \cap C$ three times. So $|A \cup B \cup C| < |A| + |B| + |C|$. To compensate we subtract the pairwise intersection sizes and our expression becomes $|A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C|$. This is almost right except for $A \cap B \cap C$ which we lost in the adjustment (it was counted three times and then subtracted three times). We add it back and get

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

In general, if we want to count $|\cup_{i=1}^{n} A_i|$ we start with $\sum_{i=1}^{n} |A_i|$ which includes pairwise intersections $A_i \cap A_j$ twice, so we exclude with $-(\sum_{1 \le i < j \le n} |A_i \cap A_j|)$. But this excludes the triple intersections so we include those with $\sum_{1 \le i < j < k \le n}^{n} |A_i \cap A_j \cap A_k|$. This overcounts quadruple intersections which we exclude etc. We stop with the exclusion or inclusion of the intersection of all A_i.

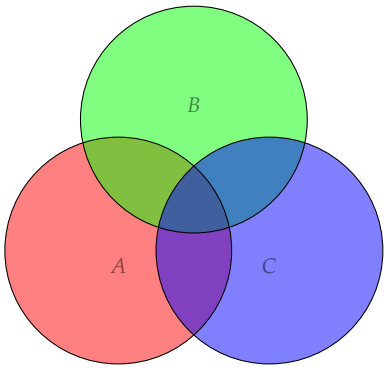


Figure 21.1: Union of three not necessarily disjoint sets.

Theorem 21.2. ***Inclusion-exclusion principle.***
Given n sets A_j, $1 \le j \le n$

$$|\bigcup_{j=1}^{n} A_j| = \sum_{k=1}^{n} (-1)^{k+1} (\sum_{J \subseteq \mathbb{N}_n,\ |J|=k} |\bigcap_{j \in J} A_j|)$$

Proof. We are going to prove the theorem by tracing the contributions of one element $a \in \bigcup_{j=1}^{n} A_j$ to the left-hand side and right-hand side of the equation. On the left it will be 1 since this is the union of sets and not multisets. For the right-hand side, we observe that there is a non-empty index set $I \subseteq \{1, 2, \ldots, n\}$ such that $\forall i \in I : a \in A_i$. The element a will contribute ± 1 from all the set intersections in which it appears, so all the terms in the sum where index set J satisfies $J \subseteq I$ (since these are intersections, a won't appear in any of the other terms). The hope is that the sum of all these ± 1 will be 1.

Let $m = |I|$. Then any index set J with size greater than m cannot be a subset of I, so the running index k of the sum only needs to go to m. For each k there are $\binom{m}{k}$ index subsets J of size k from I. In each a weighs in with 1, so the contributions of a add up to:

$$\sum_{k=1}^{m} (-1)^{k+1} \binom{m}{k}$$

We add and subtract $\binom{m}{0}$ to this sum and get

$$\begin{aligned}
\sum_{k=1}^{m}(-1)^{k+1}\binom{m}{k} &= \binom{m}{0} - \binom{m}{0} + \sum_{k=1}^{m}(-1)^{k+1}\binom{m}{k} \\
&= \binom{m}{0} - \sum_{k=0}^{m}(-1)^{k+1}\binom{m}{k} \\
&= \binom{m}{0} + \sum_{k=0}^{m}(-1)^{k}\binom{m}{k} \\
&= \binom{m}{0} + \sum_{k=0}^{m}1^{m-k}(-1)^{k}\binom{m}{k} \\
&= \binom{m}{0} + (1-1)^{m} \\
&= \binom{m}{0} = 1
\end{aligned}$$

On both sides of the equation in the theorem an arbitrary element a of the union of the sets contributes 1. This proves the inclusion-exclusion principle. □

Let us return to computing derangements. We now know how to compute $|(\bigcup_{k=1}^{n} F_n(k))|$ by using the inclusion-exclusion principle:

$$|(\bigcup_{k=1}^{n} F_n(k))| = \sum_{k=1}^{n}(-1)^{k+1}(\sum_{J\subseteq \mathbb{N}_n,\ |J|=k} |\bigcap_{j\in J} F_n(j)|)$$

For a given index set $J \subseteq \mathbb{N}_n$, $|J| = k$ the intersection contains all the permutations that are fixed in the positions $j \in J$, so the size of this intersection is $(n-k)!$. There are $\binom{n}{k}$ such index sets J of size k, so our expression becomes:

$$|(\bigcup_{k=1}^{n} F_n(k))| = \sum_{k=1}^{n}(-1)^{k+1}\binom{n}{k}(n-k)!$$

We then have

$$\begin{aligned}
!n &= n! - \sum_{k=1}^{n}(-1)^{k+1}\binom{n}{k}(n-k)! \\
&= n! + \sum_{k=1}^{n}(-1)^{k}\binom{n}{k}(n-k)! \\
&= \sum_{k=0}^{n}(-1)^{k}\binom{n}{k}(n-k)!
\end{aligned}$$

The probability of turning up identical cards from the two decks is

$$\begin{aligned}
1 - \frac{!n}{n!} &= 1 - \sum_{k=0}^{n} (-1)^k \binom{n}{k} \frac{(n-k)!}{n!} \\
&= 1 - \sum_{k=0}^{n} (-1)^k \frac{n!}{k!(n-k)!} \frac{(n-k)!}{n!} \\
&= 1 - \sum_{k=0}^{n} \frac{(-1)^k}{k!}
\end{aligned}$$

This probability converges fairly quickly to approximatively 0.7 so you have a 0.7 chance of turning up identical cards from the two decks.

22
While a

Loop invariants is the topic of the problem [1] in this note.

[1] Problem 4 on page 9 from A. Engel. *Problem-Solving Strategies*. Problem Books in Mathematics. Springer New York, 2013. ISBN 9781475789546. URL https://books.google.com/books?id=aUofswEACAAJ

Problem

We start with the state (a, b) where a, b are positive integers. To this initial state we apply the following algorithm:

```
while a > 0:
    if a < b:
        (a,b) = (2a, b - a)
    else:
        (a,b) = (a - b, 2b)
```

For which starting positions does the algorithm stop? In how many steps does it stop, if it stops? What can you tell about periods and tails?

We start with $a > 0$ and $b > 0$. We adopt the following notation: a_i, b_i are the values after $i \in \mathbb{N}_{\geq 0}$ times through the loop. Before the first time through the loop $a_0 = a$, $b_0 = b$. Let $n = a + b$.

Let's collect some invariants. We will prove all of them by induction on $i \in \mathbb{N}_{\geq 0}$.

Invariant 22.1.

$$\forall i \geq 0 : a_i + b_i = n$$

Proof. Base case $a_0 + b_0 = a + b = n$ holds by definition of n and (a_0, b_0). Assume $a_i + b_i = n$. For $a_{i+1} + b_{i+1}$ we have two cases:

Case $a_i < b_i$: Here we have $a_{i+1} = 2a_i$ and $b_{i+1} = b_i - a_i$. So

$$a_{i+1} + b_{i+1} = 2a_i + b_i - a_i = a_i + b_i = n$$

Case $a_i \geq b_i$: In this case we have $a_{i+1} = a_i - b_i$ and $b_{i+1} = 2b_i$. It follows

$$a_{i+1} + b_{i+1} = a_i - b_i + 2b_i = a_i + b_i = n$$

□

Invariant 22.2.

$$\forall i \geq 0 : b_i > 0$$

Proof. This follows almost immediately from definitions [2].

□

[2] Base case $b_0 = b > 0$ holds by definition of b. Assume $b_i > 0$. Again we have two cases. If $a_i < b_i$ then $b_{i+1} = b_i - a_i > 0$. If $a_i \geq b_i$ then $b_{i+1} = 2b_i > 0$.

Invariant 22.3.

$$\forall i \geq 0 : a_i \geq 0$$

Proof. This also follows from definitions [3].

□

[3] Base case $a_0 = a > 0$ holds by definition of a. Assume $a_i \geq 0$. Again we have two cases. If $a_i < b_i$ then $a_{i+1} = 2a_i \geq 0$. If $a_i \geq b_i$ then $a_{i+1} = a_i - b_i \geq 0$.

Invariant 22.4.

$$\forall i \geq 0 : a_i \equiv 2^i a \mod n$$

Proof. Base case $a_0 = a = 2^0 a$ trivially holds. Assume $a_i \equiv 2^i a \mod n$. For a_{i+1} we have two cases:

Case $a_i < b_i$: Here we have $a_{i+1} = 2a_i$. So

$$\begin{aligned} a_{i+1} &= 2a_i \\ &\equiv 2 \cdot 2^i a \mod n \\ &\equiv 2^{i+1} a \mod n \end{aligned}$$

Case $a_i \geq b_i$: In this case we have $a_{i+1} = a_i - b_i$. It follows

$$\begin{aligned} a_{i+1} &= a_i - b_i \\ &\equiv a_i + n - b_i \mod n \\ &\equiv a_i + a_i + b_i - b_i \mod n \\ &\equiv 2a_i \mod n \\ &\equiv 2 \cdot 2^i a \mod n \\ &\equiv 2^{i+1} a \mod n \end{aligned}$$

□

We will use these 4 invariants ($a_i \geq 0$, $b_i > 0$, $a_i + b_i = n$ and $a_i \equiv 2^i a \mod n$) to determine for which initial values a and b the loop terminates. To do so we consider $\frac{a}{n}$. Because $0 < a < n$ we know that $\frac{a}{n} \in (0, 1)$. We look at the expansion of $\frac{a}{n}$ in base 2.

Theorem 22.1. *If the expansion of $\frac{a}{n}$ is finite with k digits $d_i \in \{0, 1\}$*

$$\frac{a}{n} = \sum_{i=1}^{k} d_i 2^{-i}$$

then $a_k = 0$ and the loop terminates after k steps.

Proof. From

$$\frac{a}{n} = \sum_{i=1}^{k} d_i 2^{-i}$$

we get by multiplying both sides with $2^k n$:

$$2^k a = \sum_{i=1}^{k} n d_i 2^{k-i} \equiv 0 \mod n$$

Together with invariant 22.4 we get

$$a_k \equiv 2^k a \equiv 0 \mod n$$

and because $a_k \geq 0$, $b_k > 0$, $a_k + b_k = n$ we know that $0 \leq a_k < n$, so it must be that $a_k = 0$ and the loop terminates after at most k steps. To show that the loop terminates after exactly k steps, we need to show that $a_j > 0$ for $0 \leq j < k$. We will do this by finding a contradiction. Assume there exists a $j < k$ such that $a_j = 0$. Then it also holds that $2^j a \equiv 0 \mod n$.

From

$$\frac{a}{n} = \sum_{i=1}^{k} d_i 2^{-i}$$

we get by multiplying both sides with $2^j n$:

$$2^j a = \sum_{i=1}^{k} n d_i 2^{j-i} = \sum_{i=1}^{j} n d_i 2^{j-i} + \sum_{i=j+1}^{k} n d_i 2^{j-i} \equiv 0 \mod n$$

$2^j a \equiv 0 \mod n$, so $2^j a = nq$ for some $q \in \mathbb{Z}$. Then

$$q = \sum_{i=1}^{j} d_i 2^{j-i} + \sum_{i=j+1}^{k} d_i 2^{j-i}$$

We have $q \in \mathbb{Z}$, $\sum_{i=1}^{j} d_i 2^{j-i} \in \mathbb{Z}$, but $\sum_{i=j+1}^{k} d_i 2^{j-i} \notin \mathbb{Z}$, because $d_i \in \{0, 1\}$. This is a contradiction.

□

We arrived at a neat result: if the binary expansion of $\frac{a}{a+b}$ is finite with k digits, then the loop terminates after k steps.

What can we say if the expansion is not finite but instead has a repeating pattern with a prefix and a period (the only other option [4]) ? For starters, we can use a contradiction similar to the earlier one to prove that the loop does not terminate. Consider the infinite binary expansion:

[4] That is because $\frac{a}{a+b} \in \mathbb{Q}$. See below for why.

$$\frac{a}{n} = \sum_{i=1}^{\infty} d_i 2^{-i}$$

Assume there is a k for which $a_k = 0$. Then by multiplying the expansion with $2^k n$ we get:

$$2^k a = \sum_{i=1}^{k} n d_i 2^{k-i} + \sum_{i=k+1}^{\infty} n d_i 2^{k-i} \equiv 0 \mod n$$

So for some $q \in \mathbb{Z}$ such that $2^k a = nq$ we have

$$q = \sum_{i=1}^{k} d_i 2^{k-i} + \sum_{i=k+1}^{\infty} d_i 2^{k-i}$$

The left side and the first sum on the right both belong to $\mathbb{Z}$ but the second sum does not, which is a contradiction. This means, that $\forall k : a_k > 0$ and the loop does not terminate.

At this point we will do a small digression and prove some theorems about decimal expansion.

Theorem 22.2. *Given an integer $p > 1$, the series*

$$\sum_{i=1}^{\infty} \frac{d_i}{p^i}$$

with $d_i \in \{0, 1, \ldots, p-1\}$ converges to a value $x \in [0, 1]$.

Proof.

$$\sum_{i=1}^{n} \frac{d_i}{p^i} \leq \sum_{i=1}^{n} \frac{p-1}{p^i} \xrightarrow[n \to \infty]{} 1$$

so the series is bounded and will converge. □

Theorem 22.3. *For every $x \in [0, 1]$ there exists a decimal expansion with base $p > 1$ such that*

$$x = \sum_{i=1}^{\infty} \frac{d_i}{p^i}$$

with $d_i \in \{0, 1, \ldots, p-1\}$.

Proof. We divide the interval $[0, 1]$ into p intervals $[\frac{i}{p}, \frac{i+1}{p}]$ with $0 \leq i < p$. Since $[0, 1] = \bigcup_{i=0}^{p-1} [\frac{i}{p}, \frac{i+1}{p}]$ we know there exists at least one index i with $x \in [\frac{i}{p}, \frac{i+1}{p}]$. We set $d_1 = i$ and subdivide $[\frac{i}{p}, \frac{i+1}{p}]$ into p segments $[\frac{i}{p}, \frac{i+1}{p}] = \bigcup_{j=0}^{p-1} [\frac{d_1}{p} + \frac{j}{p^2}, \frac{d_1}{p} + \frac{j+1}{p^2}]$. x is in one of these subintervals and

we set d_2 to be the index of that subinterval and continue in this manner recursively defining all d_i. Because of the nested interval property with monotone decreasing length this converges to x.

Another way to prove it is like this:

The case where $x = 0$ is trivial (just set all $d_i = 0$).

For $x > 0$ we have:

The set $N_1 = \{k \in \mathbb{N}_0 : \frac{k}{p} < x\}$ is a set of non-negative integers strictly bounded above by p, so it has a largest element and we set $d_1 = \max(N_1)$. Then $x \leq \frac{d_1+1}{p}$ (otherwise $d_1 + 1 \in N_1$ and d_1 wouldn't be the largest element of N_1). We therefore have

$$\frac{d_1}{p} < x \leq \frac{d_1 + 1}{p}$$

We continue and look at $N_2 = \{k \in \mathbb{N}_0 : \frac{d_1}{p} + \frac{k}{p^2} < x\}$. Again the set N_2 is strictly bounded above by p and we set $d_2 = \max(N_2)$. Again we have:

$$\frac{d_1}{p} + \frac{d_2}{p^2} < x \leq \frac{d_1}{p} + \frac{d_2 + 1}{p^2}$$

Having defined $d_1, d_2, \ldots d_{n-1}$ we can recursively define $d_n = \max(N_n)$ with

$$N_n = \{k \in \mathbb{N}_0 : \sum_{i=1}^{n-1} \frac{d_i}{p^i} + \frac{k}{p^n} < x\}$$

Again $p \notin N_n$, so the definition is valid and the following inequalities hold:

$$\sum_{i=1}^{n} \frac{d_i}{p^i} < x \leq \sum_{i=1}^{n-1} \frac{d_i}{p^i} + \frac{d_n + 1}{p^n}$$

We define $u_n = \sum_{i=1}^{n} \frac{d_i}{p^i}$, $v_n = \sum_{i=1}^{n-1} \frac{d_i}{p^i} + \frac{d_n+1}{p^n}$ and $w_n = \frac{d_{n+1}+1}{p^{n+1}}$. u_n is monotone increasing and bounded above, so it converges. For v_n we have

$$\begin{aligned}
& v_n \geq v_{n+1} \\
\Leftrightarrow & \sum_{i=1}^{n-1} \frac{d_i}{p^i} + \frac{d_n + 1}{p^n} \geq \sum_{i=1}^{n} \frac{d_i}{p^i} + \frac{d_{n+1} + 1}{p^{n+1}} \\
\Leftrightarrow & \frac{d_n + 1}{p^n} \geq \frac{d_n}{p^n} + \frac{d_{n+1} + 1}{p^{n+1}} \\
\Leftrightarrow & \frac{1}{p^n} \geq \frac{d_{n+1} + 1}{p^{n+1}} \\
\Leftrightarrow & p \geq d_{n+1} + 1
\end{aligned}$$

which holds by definition of d_{n+1}. So v_n is monotone decreasing and bounded below, therefore it converges too. w_n converges to zero and $v_n = u_{n-1} + w_n$ therefore

$$lim_{n\to\infty} u_n = lim_{n\to\infty} v_n = x$$

□

Theorem 22.4. *Given is base $p > 1$ and*

$$x = \sum_{i=1}^{n} \frac{d_i}{p^i}$$

with $d_i \in \{0, 1, \ldots, p-1\}$ and $d_n \neq 0$. Then there are two base p expansions of x.

Proof. The first expansion is $x = \sum_{i=1}^{\infty} \frac{d_i}{p^i}$ with $d_i = 0$ for $i > n$. For the second expansion we define the following series:

$$y = \sum_{i=1}^{n-1} \frac{d_i}{p^i} + \frac{d_n - 1}{p^n} + \sum_{i=n+1}^{\infty} \frac{p-1}{p^i}$$

and prove that $y = x$. Then the two expansions are $0.d_1 d_2 \ldots d_n 00000 \ldots$ and $0.d_1 d_2 \ldots (d_n - 1)(p-1)(p-1)(p-1) \ldots$

To prove that $y = x$ we look at

$$\begin{aligned}
\sum_{i=n+1}^{\infty} \frac{p-1}{p^i} &= \frac{p-1}{p^n} \sum_{i=1}^{\infty} \frac{1}{p^i} \\
&= \frac{p-1}{p^n} (\sum_{i=0}^{\infty} \frac{1}{p^i} - 1) \\
&= \frac{p-1}{p^n} (\frac{p}{p-1} - 1) \\
&= \frac{p-1}{p^n} \frac{1}{p-1} \\
&= \frac{1}{p^n}
\end{aligned}$$

So y becomes

$$y = x - \frac{1}{p^n} + \frac{1}{p^n} = x$$

□

Theorem 22.5. *If we disallow series with infinitely repeated $(p-1)$ tail, any $x \in [0, 1]$ has a unique decimal expansion in base p.*

Proof. Assume two decimal expansions where both agree until index $k-1$ and index k is the first index where they differ.

$$x = \sum_{i=1}^{k-1} \frac{d_i}{p^i} + \frac{e_k}{p^k} + \sum_{i=k+1}^{\infty} \frac{e_i}{p^i}$$
$$y = \sum_{i=1}^{k-1} \frac{d_i}{p^i} + \frac{f_k}{p^k} + \sum_{i=k+1}^{\infty} \frac{f_i}{p^i}$$

Without loss of generality assume $e_k < f_k$.
We have

$$\begin{aligned} y - x &= \sum_{i=1}^{k-1} \frac{d_i}{p^i} + \frac{f_k}{p^k} + \sum_{i=k+1}^{\infty} \frac{f_i}{p^i} - \sum_{i=1}^{k-1} \frac{d_i}{p^i} - \frac{e_k}{p^k} - \sum_{i=k+1}^{\infty} \frac{e_i}{p^i} \\ &= \frac{f_k - e_k}{p^k} + \sum_{i=k+1}^{\infty} \frac{f_i}{p^i} - \sum_{i=k+1}^{\infty} \frac{e_i}{p^i} \\ &= \frac{f_k - e_k}{p^k} + \frac{1}{p^k}\left(\sum_{i=1}^{\infty} \frac{f_{k+i}}{p^i} - \sum_{i=1}^{\infty} \frac{e_{k+i}}{p^i}\right) \end{aligned}$$

We denote $u = \sum_{i=1}^{\infty} \frac{f_{k+i}}{p^i}$ and $v = \sum_{i=1}^{\infty} \frac{f_{k+i}}{p^i}$. Since we disallowed repeated $(p-1)$ tail, we know that $0 \leq u < 1$ and $0 \leq v < 1$, so $-1 < u - v < 1$. It follows that

$$0 \leq \frac{f_k - e_k - 1}{p^k} < y - x < \frac{f_k - e_k + 1}{p^k}$$

and $x \neq y$. □

Theorem 22.6. *$x \in [0,1] \cap \mathbb{Q}$ if and only if its decimal expansion in base $p > 1$ is either finite or has a prefix (of length zero or more) and an infinitely repeating non-zero length pattern tail.*

Proof.
($\Rightarrow$):
$x \in [0,1] \cap \mathbb{Q}$, so there exist $m, n \in \mathbb{N}$ with $m < n$ and $x = \frac{m}{n}$. We basically do the long division and present an expansion that will have a repeating tail (if it isn't finite). Let $k \in \mathbb{N}$ be the smallest integer such that $mp^k \geq n$ and we do division:

$$mp^k = nq + r$$

with $0 \leq r < n$. Because k is the smallest integer with $mp^k \geq n$ we have $np > mp^k$ (otherwise $k-1$ would be a smaller integer satisfying the same). That means $np > nq + r$ and thus $p > \frac{np-r}{n} > q$. This gives us $k-1$ zeros and the first non-zero digit in the expansion, namely q:

$$\begin{aligned}\frac{m}{n} &= \frac{1}{p^k}\frac{mp^k}{n} \\ &= \frac{1}{p^k}\frac{nq+r}{n} \\ &= \frac{q}{p^k} + \frac{r}{n}\end{aligned}$$

We repeat this process with $\frac{r}{n}$. There are only n possible remainders, so if it doesn't end with a remainder of zero it must eventually get a previously seen remainder and so the expansion will repeat itself. This creates an expansion with an infinitely repeating non-zero length pattern tail. Since it isn't finite, we can disallow repeating $(p-1)$ and from the expansion uniqueness theorem we have proved the $(\Rightarrow)$ direction.

$(\Leftarrow)$:

This direction is easy. If it is a finite sum, then it is rational since all the parts are rational. If it is infinite repeating we can eliminate the non-repeating prefix since it is finite and rational and shift the rest. So we can concentrate on a repeating series with a period of length $k-1$:

$$\begin{aligned}x &= \sum_{i=0}^{\infty}(\frac{1}{p^{ki}}\sum_{j=1}^{k-1}\frac{d_j}{p^j}) \\ &= (\sum_{j=1}^{k-1}\frac{d_j}{p^j})\sum_{i=0}^{\infty}\frac{1}{p^{ki}} \\ &= (\sum_{j=1}^{k-1}\frac{d_j}{p^j})(1+\sum_{i=1}^{\infty}\frac{1}{p^{ki}}) \\ &= (\sum_{j=1}^{k-1}\frac{d_j}{p^j})(1+\sum_{i=1}^{\infty}(\frac{1}{p^k})^i) \\ &= (\sum_{j=1}^{k-1}\frac{d_j}{p^j})(1+\frac{p^k}{p^k-1})\end{aligned}$$

which is a rational expression. □

We return to our problem. We now know the expansion of $\frac{a}{a+b}$ is repeating a period if it doesn't terminate. We will show that the loop also repeats a period of the same length.

Theorem 22.7. *If $\frac{a}{n}$ has an expansion in base p which repeats a period of k digits infinitely, then*

$$ap^k \equiv a \mod n$$

Proof. We have $\frac{a}{n} = 0.\overline{d_1 d_2 d_3 \dots d_k}$ which means

$$\begin{aligned}\frac{a}{n} &= 0.\overline{d_1 d_2 d_3 \dots d_k} \\ &= \sum_{i=1}^{k} \frac{d_i}{p^i} + \frac{1}{p^k}\left(\sum_{i=1}^{k} \frac{d_i}{p^i} + \frac{1}{p^k}\left(\sum_{i=1}^{k} \frac{d_i}{p^i} + \dots\right.\right. \\ &= \sum_{i=1}^{k} \frac{d_i}{p^i} + \frac{1}{p^k}\frac{a}{n}\end{aligned}$$

We multiply both sides by np^k and get

$$ap^k = \sum_{i=1}^{k} n d_i p^{k-i} + a$$

which proves the theorem. □

Theorem 22.8. *If $\frac{a}{n}$ has an expansion in base p which has a prefix and then repeats a period of k digits infinitely, then*

$$ap^k \equiv a \mod n$$

Proof. We have $\frac{a}{n} = 0.e_1 e_2 e_3 \dots e_l \overline{d_1 d_2 d_3 \dots d_k}$ which means

$$\begin{aligned}\frac{a}{n} &= 0.e_1 e_2 e_3 \dots e_l \overline{d_1 d_2 d_3 \dots d_k} \\ &= \sum_{i=1}^{l} \frac{e_i}{p^i} + \frac{1}{p^l}\left(0.\overline{d_1 d_2 d_3 \dots d_k}\right)\end{aligned}$$

This means

$$\frac{ap^l - \sum_{i=1}^{l} p^{l-i} e_i n}{n} = 0.\overline{d_1 d_2 d_3 \dots d_k}$$

We can then apply the previous theorem to a new $a' := ap^l - \sum_{i=1}^{l} p^{l-i} e_i n$ and see that

$$a' p^k \equiv a' \mod n$$

But $a' \equiv ap^l \mod n$, so

$$ap^{k+l} \equiv ap^l \mod n$$

or $ap^k \equiv a \mod n$. □

We combine this last result with the invariant 22.4 to see that $a_{i+k} = a_i$ and the loop repeats values with period k.

23
Divisible by three

Loop invariants and a constraint relaxation are used to solve this problem.

Problem

Show that an integer is divisible by three iff the sum of its digits in decimal representation is divisible by three.

The following proof is a delightful example of unconventional thinking[1].

[1] Unfortunately I don't know the origin of this proof and I don't remember where I first saw it.

We have to work with the digits in decimal representation of some integer n. Let's denote with $s(n)$ the sum of those digits. We will prove something stronger than the problem:

$$s(n) = n - 9k, \text{ for some } k \in \mathbb{Z}$$

The delightful twist that we are going to use is a relaxation of decimal representation: we will allow digits bigger than nine. We will then heal the representation back to decimal digits less than ten in a loop while maintaining $s(n) = n - 9k$ as a loop invariant.

We start with a representation with just one digit, namely n itself:

$$n = \sum_{i=0} d_i 10^i, \text{ with } d_0 = n \text{ and } \forall i > 0 : d_i = 0$$

This is not yet a valid decimal representation if $d_0 > 9$, but the loop invariant does hold: $s(n) = \sum_{i=0} d_i = n = n - 9 \cdot 0$. In a loop we now keep subtracting ten from d_0 and adding a carry-over one to d_1 until $d_0 \leq 9$. Each time through the loop d_1 increases by one and d_0 decreases by ten, so $s(n) = \sum_{i=0} d_i$ decreases by nine:

$$d_1 \leftarrow d_1 + 1$$
$$d_0 \leftarrow d_0 - 10$$
$$s(n) \leftarrow s(n) - 9$$

This means the loop invariant $s(n) = n - 9k$ is maintained during the healing of d_0. Eventually $d_0 \leq 9$. We then look at d_1 and heal it similarly, carrying over to d_2 and subtracting ten from it. Again the loop invariant holds. We repeat this for all digits until all are healed. The healing has to finish because n is a finite integer with a finite decimal representation. In the end we have a valid decimal representation and the loop invariant still holds which proves our problem.

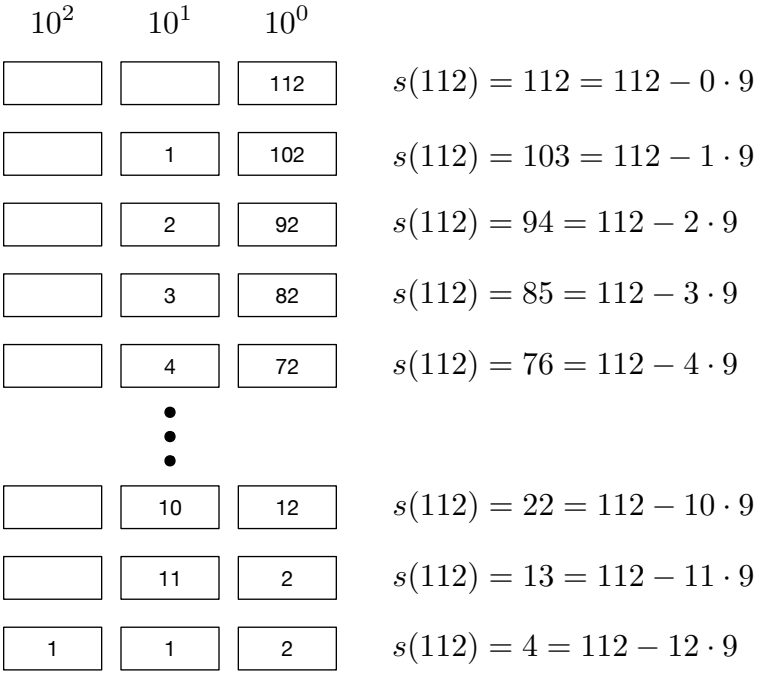


Figure 23.1: Example with $n = 112$. In each row the boxes represent the digits in the decimal representation (least significant on the right). In the first row the representation has only one digit, the number itself. Going down, each row is one step in the healing of the representation while maintaining the loop invariant $s(112) = 112 - k \cdot 9$.

Obviously one could just observe that $10 \equiv 1 \mod 3$, so also $10^k \equiv 1 \mod 3$ and the divisibility rule follows immediately.

$$n = \sum_{i=0} d_i 10^i \equiv \sum_{i=0} d_i \mod 3$$

24
Dutch National Flag

Problem 'Dutch National Flag' in *Programming, The Derivation of Algorithms*[1].

[1] A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990

Problem

Write a program that swaps elements of an array containing colors red, white and blue in such a way that the array's final state is in accordance with the Dutch National Flag.

We hope to solve this problem in linear time, going only once through the array in a loop.

Our array is $A[0, \ldots, n)$ with

$$\forall i : 0 \leq i < n : A[i] \in \{\blacksquare, \square, \blacksquare\}$$

The desired final state of the array has 3 contiguous regions: the red region, the white region and the blue region. Two indices r and w into the array are sufficient to show the extent of each region. We define post condition R

$$\begin{aligned} R \equiv\ & (\forall i : 0 \leq i < r : A[i] = \blacksquare) \\ \wedge\ & (\forall i : r \leq i < w : A[i] = \square) \\ \wedge\ & (\forall i : w \leq i < n : A[i] = \blacksquare) \end{aligned}$$

Our loop invariant P will be a relaxation[2] of the post condition R. We need to introduce a new index variable b to capture the notion of unprocessed region:

[2] Relaxation is a common technique to derive a useful loop invariant from a post condition. A common way to do the relaxation is to introduce a variable that in the beginning completely relaxes the condition and that then gradually changes and tightens the condition to its final desired form.

$$
\begin{aligned}
P \equiv\ & (\forall i : 0 \leq i < r : A[i] = \blacksquare) \\
& \wedge (\forall i : r \leq i < w : A[i] = \square) \\
& \wedge (\forall i : w \leq i < b : A[i] \text{ has not been processed yet}) \\
& \wedge (\forall i : b \leq i < n : A[i] = \blacksquare)
\end{aligned}
$$

This allows us to assign values to our indices r, w and b that satisfy P before the loop starts by extending the unprocessed region to be the whole array and making the red, white and blue regions empty[3]:

$$
\begin{aligned}
r &\leftarrow 0 \\
w &\leftarrow 0 \\
b &\leftarrow n
\end{aligned}
$$

[3] This is the key insight for solving the problem. Instead of having only three color regions to work with, we introduce a fourth region of unprocessed elements and we gradually shrink it. In the beginning this fourth region is the whole array and the color regions are all three empty. As the unprocessed region shrinks, the color regions start to grow in such a way that P always stays true. In the end the unprocessed region is empty and the three color regions are in their final desired state thanks to P always holding.

Our goal now is to maintain the loop invariant P while reducing the unprocessed region by processing array elements and swapping them until the unprocessed region is empty, so $b - w = 0$ or $b = w$. We then have $b = w \wedge P \Rightarrow R$. The swapping needs to happen in such a way that P always holds. We also want to make progress each time through the loop, so we want $b - w$ to get smaller each time through the loop. We achieve progress by either increasing w or decreasing b. As long as $b > w$ we go through the loop.

We will do a case analysis of the state at the region borders of processed and unprocessed regions of the array.

Let's start with a simple case: $A[w] = \square$. Then moving w one position to the right extends the white region, maintains P and shrinks $b - w$, so makes progress. We write this down as one case for the loop body:

$$\text{if } A[w] = \square \text{ then } w \leftarrow w + 1 \text{ endif}$$

Because we are inside the loop we know that $b > w$, so $A[b-1]$ exists, ie $b - 1$ is a valid index position. If $A[b-1] = \blacksquare$, then we can extend the blue region to the left and thus also shrink the unprocessed region while maintaining P:

$$\text{if } A[b-1] = \blacksquare \text{ then } b \leftarrow b - 1 \text{ endif}$$

We have a couple more cases to cover.

If $A[w] = \blacksquare$ then we can do our first swap: we swap $A[w]$ with $A[b-1]$ which will allow us after the swap to extend the blue region to its left as done in the previous case. As before this maintains P and is progress. Let's capture this for the body of our loop:

$$\text{if } A[w] = \blacksquare \text{ then } A[w] \leftrightarrow A[b-1] \,;\, b \leftarrow b-1 \text{ endif}$$

If $A[w] = \blacksquare$ then we can do the following swap: we swap $A[r]$ with $A[w]$. We can then extend the red region to its right. Whatever the white region was (empty or not), it also shifts to the right by one (as a region it hasn't changed, just the first white element might have moved to be the last element of the white region if the white region was not empty):

$$\text{if } A[w] = \blacksquare \text{ then } A[w] \leftrightarrow A[r] \,;\, r \leftarrow r+1 \,;\, w \leftarrow w+1 \text{ endif}$$

These cases[4] are sufficient to allow us to make progress while maintaining P. Because $b - w$ is finite and we reduce it each time through the loop, the loop will terminate. Our final program (in Go syntax) is:

```
r := 0
w := 0
b := 0
for w < b {
    switch {
    case A[w] == White: w = w + 1
    case A[b-1] == Blue: b = b - 1
    case A[w] == Blue: swap(A[w], A[b-1])
                       b = b - 1
    case A[w] == Red: swap(A[w], A[r])
                      r = r + 1
                      w = w + 1
    }
}
```

The loop invariant P is maintained throughout and when the loop exits, we have $b = w$ which establishes R.

[4] These cases are biased towards making progress from the left to the right. One can make similar choices that cover more cases on the right border of the unprocessed region.

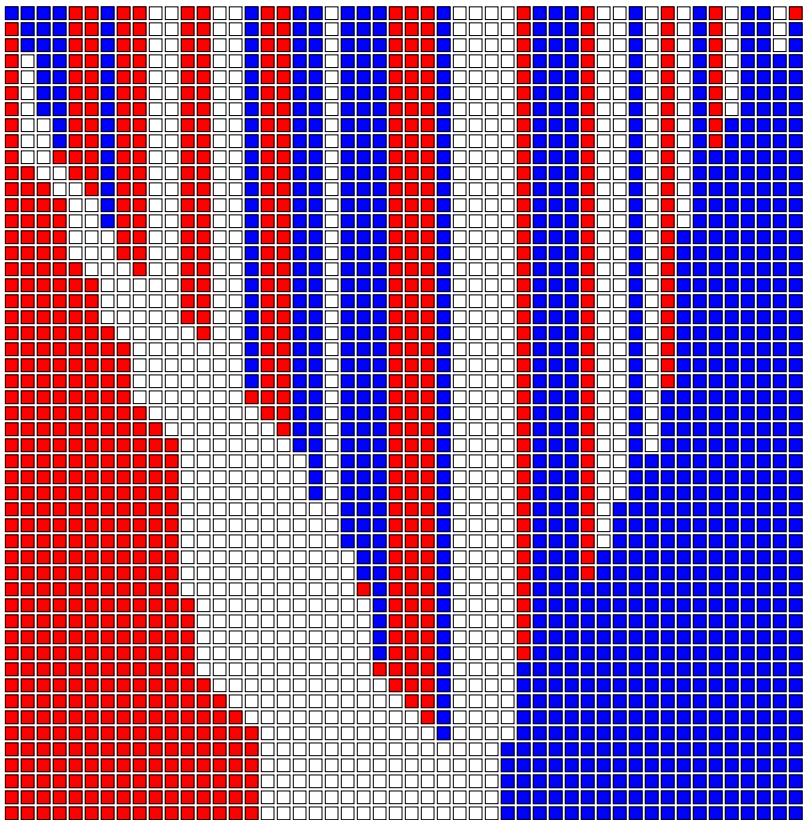


Figure 24.1: Example of processing an array. The first row is the initial state of the array. Each row below is one time through the loop. It is interesting to see that even though condition R is satisfied towards the end while $b > w$, the loop still has to process elements until $b = w$, not doing any more swaps but bringing w and b ever closer.

25
Bernoulli Inequality

In this note we explore some variants of the *Bernoulli Inequality* by following this exercise[1].

[1] Exercise 1.18 on page 10 in N.L. Carothers. *Real Analysis*. Cambridge University Press, 2000. ISBN 9780521497565. URL https://books.google.com/books?id=4VFDVy1NFiAC

Problem

Given $a > 0$ show that $(1+a)^r > 1 + ar$ for any rational exponent $r > 1$.

We need to declare what properties of the real numbers we are allowed to use. This exercise is in the beginning of Real Analysis, so we are not allowed to deploy any 'heavy machinery' like derivatives, convex functions etc. We assume the usual properties of $\mathbb{R}$ as an ordered field, but we have not shown yet that m-th roots exist for any positive real[2].

[2] We are actually going to sketch that out in this note because we need it. We have not defined exponentiation by a rational exponent yet.

First we prove the inequality for natural numbers $n > 1$.

Theorem 25.1. *Given $a > -1$ and $a \neq 0$, the inequality $(1+a)^n > 1 + na$ holds for any integer $n > 1$.*

Proof. We are going to use induction to prove this inequality. For $n = 2$ we have

$$1 + 2a + a^2 > 1 + 2a$$

which covers the base case. Assume that the inequality holds for n. For the induction step we have:

$$\begin{aligned}(1+a)^{n+1} &= (1+a)(1+a)^n > (1+a)(1+na) \\ &= 1 + na + a + na^2 \\ &= 1 + (n+1)a + na^2 > 1 + (n+1)a\end{aligned}$$

□

The next theorem might look like it is coming out of nowhere but it is a step in the exercise and there is a connection with the Bernoulli inequality.

Theorem 25.2. *The sequence $e_n = (1 + \frac{x}{n})^n$ is increasing for any $x > 0$.*

Proof. We are actually going to use theorem 25.1 to prove this theorem. There are two straightforward ways to prove that a sequence e_n increases. One way is to show that $e_{n+1} - e_n > 0$ and the other way is to show that $\frac{e_{n+1}}{e_n} > 1$. The second way requires $e_n > 0$ which is the case here. We are choosing the second way because ratios more closely connect with multiplication and exponents and we hope to find opportunities to simplify the expressions.

$$\begin{aligned}
\frac{e_{n+1}}{e_n} &= \frac{(1 + \frac{x}{n+1})^{n+1}}{(1 + \frac{x}{n})^n} \\
&= (1 + \frac{x}{n})(\frac{(1 + \frac{x}{n+1})}{(1 + \frac{x}{n})})^{n+1} \\
&= (1 + \frac{x}{n})(\frac{n+1+x}{n+x}\frac{n}{n+1})^{n+1} \\
&= (1 + \frac{x}{n})(\frac{(n+1)n+nx}{(n+x)(n+1)})^{n+1} \\
&= (1 + \frac{x}{n})(\frac{(n+1)(n+x)-x}{(n+x)(n+1)})^{n+1} \\
&= (1 + \frac{x}{n})(1 - \frac{x}{(n+x)(n+1)})^{n+1}
\end{aligned}$$

The last part of this long chain of equalities has a form that suggests theorem 25.1. We have to make sure that $\frac{-x}{(x+n)(n+1)}$ satisfies the conditions of that theorem.

$$\begin{aligned}
\frac{-x}{(x+n)(n+1)} > -1 &\Leftrightarrow x < (x+n)(n+1) \\
&\Leftrightarrow x < nx + x + n^2 + n \\
&\Leftrightarrow 0 < nx + n^2 + n
\end{aligned}$$

which $x > 0$ satisfies, so we can apply theorem 25.1. It follows that

$$\begin{aligned}
\frac{e_{n+1}}{e_n} &> (1 + \frac{x}{n})(1 - (n+1)\frac{x}{(n+x)(n+1)}) \\
&= (1 + \frac{x}{n})(1 - \frac{x}{n+x}) \\
&= (1 + \frac{x}{n})(\frac{n}{n+x}) \\
&= 1
\end{aligned}$$

□

Next we need m-th roots for any positive real number.

Theorem 25.3. *Let $x \geq 0$ be a positive real number and let $n \geq 1$ be an integer. Then the set $R := \{y \in \mathbb{R} : y \geq 0, y^n \leq x\}$ is not empty and bounded above.*

Proof. $0 \in R$, so R is not empty. To find upper bounds for R we will look at two cases: $x > 1$ and $x \leq 1$.

Let us start with $x > 1$. Then x itself is an upper bound because any $y > x$ would have $y^n > x$.

For $x \leq 1$ we find that 1 is an upper bound because if $y > 1$ then $y^n > 1 \geq x$, which is a contradiction.

□

Because of completeness we know that $sup(R)$ exists. We will denote $x^{\frac{1}{n}} := sup(R)$. We still have a little work to do. We are only going to prove properties of $x^{\frac{1}{n}}$ necessary for our inequality problem.

Theorem 25.4.

(i) $(x^{\frac{1}{n}})^n = x$

(ii) $x^{\frac{1}{n}} \geq 0$

(iii) $x_1 > x_2 \Leftrightarrow x_1^{\frac{1}{n}} > x_2^{\frac{1}{n}}$

(iv) $(x^{\frac{1}{n}})^{\frac{1}{m}} = x^{\frac{1}{mn}}$

Proof. For notational simplicity, we define $z := x^{\frac{1}{n}} = sup(R)$.

For (i) we prove by contradiction that $z^n < x$ and $z^n > x$ are impossible.

First assume $z^n < x$. Then $x - z^n > 0$. For any small $0 < \epsilon < 1$ we have:

$$\begin{aligned}(z+\epsilon)^n &= \sum_{i=0}^{n} \binom{n}{i} \epsilon^i z^{n-i} \\ &= z^n + \sum_{i=1}^{n} \binom{n}{i} \epsilon^i z^{n-i} \\ &= z^n + \epsilon \sum_{i=1}^{n} \binom{n}{i} \epsilon^{i-1} z^{n-i}\end{aligned}$$

Since $\epsilon < 1$ we can replace all the ϵ^{i-1} in the sum with 1 to get the inequality:

$$(z+\epsilon)^n \leq z^n + \epsilon \sum_{i=1}^{n} \binom{n}{i} z^{n-i}$$

We have the identity:

$$\sum_{i=1}^{n} \binom{n}{i} z^{n-i} = z^{n+1} - z^n$$

so our inequality becomes:

$$(z+\epsilon)^n \leq z^n + \epsilon(z^{n+1} - z^n)$$

Now choose ϵ such that

$$\epsilon < \frac{x - z^n}{z^{n+1} - z^n}$$

and we have

$$\begin{aligned}(z+\epsilon)^n &\leq z^n + \epsilon(z^{n+1} - z^n) \\ &< z^n + (\frac{x - z^n}{z^{n+1} - z^n})(z^{n+1} - z^n) \\ &= z^n + x - z^n = x\end{aligned}$$

This means that $(z+\epsilon)^n \in R$ but $z + \epsilon > z = sup(R)$, a contradiction. So z^n cannot be smaller than x.

Next assume $z^n > x$. Then $z^n - x > 0$. We proceed similarly to the previous case. For any small $0 < \epsilon < 1$ we have:

$$\begin{aligned}(z-\epsilon)^n &= \sum_{i=0}^{n} \binom{n}{i} (-1)^i \epsilon^i z^{n-i} \\ &= z^n + \sum_{i=1}^{n} \binom{n}{i} (-1)^i \epsilon^i z^{n-i} \\ &= z^n - \epsilon \sum_{i=1}^{n} \binom{n}{i} (-1)^{i-1} \epsilon^{i-1} z^{n-i}\end{aligned}$$

Since $\epsilon < 1$ we can replace all the ϵ^{i-1} in the sum with 1 to get the inequality:

$$\begin{aligned}(z-\epsilon)^n &\geq z^n - \epsilon \sum_{i=1}^{n} \binom{n}{i} z^{n-i} \\ &\geq z^n - \epsilon(z^{n+1} - z^n)\end{aligned}$$

Again choose ϵ such that

$$\epsilon < \frac{z^n - x}{z^{n+1} - z^n}$$

$$\begin{aligned}(z-\epsilon)^n &\geq z^n - \epsilon(z^{n+1} - z^n) \\ &> z^n - (\frac{z^n - x}{z^{n+1} - z^n})(z^{n+1} - z^n) \\ &= z^n + x - z^n = x\end{aligned}$$

Because $z - \epsilon < z$ there must exist $y \in R$ such that $z - \epsilon < y$. We then have

$$x < (z - \epsilon)^n < y^n \leq x$$

which is a contradiction. So z^n cannot be greater than x either. The only possibility left is $z^n = x$.

Both (ii) and (iii) follow from the identity:

$$(a^n - b^n) = (a - b)(\sum_{i=0}^{n-1} a^{n-1-i} b^i)$$

For (iv) we raise both sides to the power of mn:

$$\begin{aligned}((x^{\frac{1}{n}})^{\frac{1}{m}})^{mn} &= (((x^{\frac{1}{n}})^{\frac{1}{m}})^m)^n \\ &= (x^{\frac{1}{n}})^n \\ &= x \\ &= (x^{\frac{1}{mn}})^{mn}\end{aligned}$$

□

We are almost ready to define exponentiation by a positive rational exponents. We need one more theorem:

Theorem 25.5. *Given $p, q, p', q' \in \mathbb{N}$ such that $pq' = p'q$ and with any real number $x > 0$ we have*

$$(x^{\frac{1}{q}})^p = (x^{\frac{1}{q'}})^{p'}$$

Proof. We have $pq' = p'q$. We define $y = x^{\frac{1}{pq'}} = x^{\frac{1}{p'q}}$.

We know from equality (iv) in theorem 25.4 that

$$y = (x^{\frac{1}{q'}})^{\frac{1}{p}} = (x^{\frac{1}{q}})^{\frac{1}{p'}}$$

so

$$y^p = x^{\frac{1}{q'}}, \text{ and } y^{p'} = x^{\frac{1}{q}}$$

We then have

$$(x^{\frac{1}{q}})^p = (y^{p'})^p = (y^p)^{p'} = (x^{\frac{1}{q'}})^{p'}$$

□

We can now define exponentiation by $r \in \mathbb{Q}$, $r > 0$. Let $r = \frac{p}{q}$ and $x > 0$. Then $x^r := (x^{\frac{1}{q}})^p$ and we know this is well defined.

We are ready to prove the Bernoulli inequality for rational exponents.

The exponent $r = \frac{p}{q}$ is greater than one, so $p > q$.

We know from theorem 25.2 that e_n is increasing, so:

$$(1 + \frac{x}{p})^p > (1 + \frac{x}{q})^q$$

We choose $x = ap$ and have:

$$(1 + a)^p > (1 + ar)^q$$

We take the q-th root and we know from property (iii) of theorem 25.4 that

$$(1 + a)^r > 1 + ar$$

which proves the Bernoulli inequality for rational exponents. We cannot prove it yet for any real exponent without resorting to limits which forces us to lose the inequality strictness. Instead let us close this note with three applications of the Bernoulli inequality[3].

[3] Exercises 1.19 and 1.20 on page 10 in N.L. Carothers. *Real Analysis*. Cambridge University Press, 2000. ISBN 9780521497565. URL https://books.google.com/books?id=4VFDVy1NFiAC

Theorem 25.6. *For $0 < c < 1$ we have $c^n \to 0$.*

Proof.

$$\begin{aligned}\frac{1}{c^n} &= (\frac{1}{c})^n \\ &> 1 + n(\frac{1}{c} - 1) \\ &> n(\frac{1}{c} - 1)\end{aligned}$$

so

$$0 < c^n < \frac{1}{n}\frac{c}{1-c}$$

and c^n is being squeezed into converging to zero. □

Theorem 25.7. *For $c > 0$ we have $\sqrt[n]{c} \to 1$.*

Proof. We have two cases: $c \geq 1$ and $c < 1$. Let us first deal with $c \geq 1$:

$$\begin{aligned}(\sqrt[n]{c})^n &> 1 + n(\sqrt[n]{c} - 1) \\ c - 1 &> n(\sqrt[n]{c} - 1) \\ \frac{c-1}{n} + 1 &> \sqrt[n]{c} \geq 1\end{aligned}$$

and $\sqrt[n]{c}$ is squeezed into converging to one.

For the case $c < 1$ we consider its reciprocal $\sqrt[n]{\frac{1}{c}}$ and the result follows from the previous case. □

Theorem 25.8. *For $a_i > 0$, $1 \leq i \leq n$ we have*[4]

$$\sqrt[n]{\prod_{i=1}^{n} a_i} \leq \frac{1}{n} \sum_{i=1}^{n} a_i$$

[4] This is known as the AGM inequality, or Arithmetic Geometric Mean inequality.

Proof. We are going to use Bernoulli and induction (as the exercise hint suggests). For $n = 2$ we have

$$\begin{aligned}
&\sqrt{a_1 a_2} \leq \frac{1}{2}(a_1 + a_2) \\
&\Leftrightarrow a_1 a_2 \leq \frac{1}{4}(a_1^2 + 2a_1 a_2 + a_2^2) \\
&\Leftrightarrow 4a_1 a_2 \leq a_1^2 + 2a_1 a_2 + a_2^2 \\
&\Leftrightarrow 0 \leq (a_1 - a_2)^2
\end{aligned}$$

This takes care of the base case. For the induction step we assume AGM holds for n. We introduce some notation to simplify our expressions: $s_n := \sum_{i=1}^{n} a_i$, $\bar{a}_n := \frac{s_n}{n}$, $p_n := \prod_{i=1}^{n} a_i$ and finally $\bar{g}_n := \sqrt[n]{p_n}$.

We assume $\bar{g}_n \leq \bar{a}_n$ and have to prove $\bar{g}_{n+1} \leq \bar{a}_{n+1}$.

We consider $(\frac{\bar{a}_{n+1}}{\bar{a}_n})^{n+1}$ and have:

$$\begin{aligned}
(\frac{\bar{a}_{n+1}}{\bar{a}_n})^{n+1} &= (\frac{n}{n+1}\frac{s_{n+1}}{s_n})^{n+1} \\
&> 1 + (n+1)(\frac{n}{n+1}\frac{s_{n+1}}{s_n} - 1) \\
&= 1 + (n+1)\frac{ns_{n+1} - ns_n - s_n}{(n+1)s_n} \\
&= 1 + \frac{na_{n+1} - s_n}{s_n} \\
&= \frac{na_{n+1}}{s_n} \\
&= \frac{a_{n+1}}{\bar{a}_n}
\end{aligned}$$

so

$$(\bar{a}_{n+1})^{n+1} > a_{n+1}(\bar{a}_n)^n \geq a_{n+1} p_n = p_{n+1}$$

which concludes the induction step and proves AGM. □

26
Completeness

COMPLETENESS and related properties[1] are the topic in this section.

[1] Exercise 2.6.7 on page 71 from Stephen Abbott. *Understanding Analysis*. Springer, 2 edition, 2015. ISBN 978-1-4939-2711-1.

Consider the function $f : \mathbb{Q} \to \mathbb{Q}$ defined as follows:

$$f(x) = \begin{cases} -1 & : x^2 < 2 \\ 1 & : \text{otherwise} \end{cases}$$

Even though $\forall x \in \mathbb{Q} : f'(x) = 0$ the function f is not constant. Furthermore f is continuous in $\mathbb{Q}$ and $f(0) = -1 < 0$ and $f(2) = 1 > 0$ but there is no $c \in \mathbb{Q}$ for which $f(c) = 0$, so the *Intermediate Value Property* doesn't hold[2].

[2] The Ancient Greeks already discovered that $\sqrt{2} \notin \mathbb{Q}$.

Clearly $\mathbb{R}$ has an additional property which distinguishes it from $\mathbb{Q}$. This property cannot be deduced from the ordered field axioms[3] because those are shared by $\mathbb{Q}$ and $\mathbb{R}$ and we would be able to deduce it for $\mathbb{Q}$ too. It needs to be an additional property. The **Dedekind Completeness Property** is most commonly used as this additional property. We want to explore in this section how Dedekind Completeness relates to other properties also tied to what makes $\mathbb{R}$ different from $\mathbb{Q}$.

[3] We mean here the axioms of Addition and Multiplication (Commutativity, Associativity, etc) and Order axioms (Trichotemy, Transitivity, etc). See http://homepages.math.uic.edu/~kauffman/axioms1.pdf

The properties we consider are[4]:

[4] For a more detailed view on this topic and counterexamples of ordered fields without some of these properties see J. Propp. Real Analysis in Reverse. *ArXiv e-prints*, April 2012. URL https://arxiv.org/abs/1204.4483.

Dedekink Completeness Property **DDC**: Every non-empty real set bounded from above has a least upper bound.

Cut Property **CP**: Let A and B be two non-empty subsets of $\mathbb{R}$ with $A \cap B = \emptyset$ and $A \cup B = \mathbb{R}$ such that $\forall a \in A$ and $b \in B : a < b$. Then there exists a cutpoint $c \in \mathbb{R}$ such that $\forall a \in A$ and $b \in B : a \leq c \leq b$.

Archimedean Property **AP**: $\forall x \in \mathbb{R} : \exists n \in \mathbb{N}$ with $n > x$.

Nested Interval Property **NIP**: Given sequence of non-empty intervals $I_n, n \in \mathbb{N}$ with $I_{n+1} \subseteq I_n$, then $\cap_{n \in \mathbb{N}} I_n \neq \emptyset$.

Monotone Convergence Property **MC**: A bounded monotone sequence converges.

Bolzano-Weierstrass Property **BW**: A bounded sequence has a convergent subsequence.

Cauchy Criterion **CC**: A sequence converges if and only if it is a Cauchy sequence.

Ratio Test **RT**: If $\lim_{n\to\infty} \frac{|a_{n+1}|}{|a_n|} = L < 1$ then $\sum_{n=1}^{\infty} a_n$ converges[5].

Intermediate Value Property **IV**: Given is a continuous function $f : [a, b] \to \mathbb{R}$ with $f(a) < 0$ and $f(b) > 0$. Then there exists $c \in [a, b]$ with $f(c) = 0$.

[5] The *Ratio Test* and the *Intermediate Value Property* feel like higher level properties that use infinite series and continuos functions. We will see in the following theorems how they relate to the other properties.

Theorem 26.1. *DDC* $\Leftrightarrow$ *CP*

Proof. ($\Rightarrow$) We have A and B two non-empty subsets of $\mathbb{R}$ with $A \cap B = \emptyset$ and $A \cup B = \mathbb{R}$ such that $\forall a \in A$ and $b \in B : a < b$. B is non-empty, so there exists $b \in B$. This b is an upper bound of A, so A is bound from above. By the Dedekind Completeness Property *DDC* there exists a least upper bound c. We claim that c is the desired cutpoint. Since c is the least upper bound we already have $A \leq c$. Assume $\exists b' \in B$ with $b' < c$. But b' is an upper bound of A (since $A < B$) which means $c \leq b'$ because c is the least upper bound. This is a contradiction, so $\forall b' \in B : b' \geq c$. It follows that $A \leq c \leq B$ and c is the cutpoint.
($\Leftarrow$) We are given a non-empty set $A \subset \mathbb{R}$ bound from above, so there exists $b \in \mathbb{R} : A \leq b$. We define B be the set of upper bounds of A and let $A' = \mathbb{R} \setminus B$. Both A' and B are non-empty, $A' < B$ and $A' \cup B = \mathbb{R}$[6]. By the Cut Property *CP* there exists a cutpoint c with $A' \leq c \leq B$. We claim that c is the least upper bound of A. Assume there exists $a \in A$ with $c < a$. Then for $c' = \frac{c+a}{2}$ we have $c < c' < a$. This implies that $c' \in B$ so c' is an upper bound of A which contradicts with $c' < a$. We therefore have $\forall a \in A : a \leq c$ and c is an upper bound of A. Now assume there exists another upper bound d with $d < c$. But then $d \in A'$ which contradicts the definition of A' and B. So for all d upper bound of A we have $d \geq c$. This makes c the least upper bound of A. □

[6] The set A is bounded from above so B is non-empty. If $A = \{a\}$ then A' is non-empty (for example $(a-1) \in A'$). If $|A| > 1$ then one of the elements in A cannot be an upper bound of A which also implies A' is non-empty. By definition $A' \cup B = \mathbb{R}$. Assume there exists $a' \in A'$ and $b' \in B$ such that $a' \geq b'$. This would make a' an upper bound of A, so $a' \in B$, a contradiction. It follows that $A' < B$.

Theorem 26.2. *DDC* $\Leftrightarrow$ *NIP* + *AP*[7]

Proof. ($\Rightarrow$) We have nested intervals $I_n = [a_n, b_n]$ with $I_{n+1} \subseteq I_n$. It follows that for all $n \in \mathbb{N}$ we have $a_{n+1} \geq a_n$ and $b_{n+1} \leq b_n$. Assume there exists $i, j \in \mathbb{N}$ such that $b_i < a_j$. We have three cases:

- $i = j$: then $a_i \leq b_i$ for interval I_i contradicting $b_i < a_j$.
- $i < j$: then $b_i \geq b_j$ which yields the inequality chain $b_j \leq b_i < a_j$, contradicting $a_j \leq b_j$ for interval I_j.

[7] The Nested Intervals Property *NIP* is not enough to achieve Dedekind Completeness *DDC*. For examples of fields that are not Archimedean see J. Propp. Real Analysis in Reverse. *ArXiv e-prints*, April 2012. URL https://arxiv.org/abs/1204.4483. This theorem only shows that if the Archimedean Property *AP* also holds then we can get back from *NIP* to *DDC*.

- $i > j$: then $a_i \geq a_j$ which yields the inequality chain $b_i < a_j \leq a_i$, contradicting $a_i \leq b_i$ for interval I_i.

This means that for all $i, j \in \mathbb{N}$ we have $a_j \leq b_i$. In other words, the b_n are upper bounds for the set $A = \{a_n : n \in \mathbb{N}\}$.

The set A is bound from above and non-empty, so according to DDC there exists a least upper bound c. Since it is an upper bound we already have $\forall n \in \mathbb{N} : a_n \leq c$. Since c is the least upper bound and all b_n are upper bounds we also have $c \leq b_n$. It follows that $\forall n \in \mathbb{N} : c \in I_n$ or $c \in \cap_{n\in\mathbb{N}} I_n$. This proves $DDC \Rightarrow NIP$.

Assume there exists $x \in \mathbb{R}$ such that $\forall n \in \mathbb{N} : n \leq x$. This means that $\mathbb{N}$ is bound from above. Let c be the least upper bound for $\mathbb{N}$. We have

$$\forall n \in \mathbb{N} : n+1 \in \mathbb{N} \Rightarrow n+1 \leq c \Rightarrow n \leq c-1$$

$c-1$ is an upper bound, c is the least upper bound so $c \leq c-1$, a contradiction. This proves $DDC \Rightarrow AP$.
($\Leftarrow$) Consider the non-empty set $S \subseteq \mathbb{R}$ bounded from above by $b_0 \in \mathbb{R}$.

We want to apply NIP, so we define nested intervals around the upper bounds of S.

Proof Part 26.2.1. S is non-empty, so there exists $a_0 \in S$. Define $I_0 = [a_0, b_0]$. The strategy now is to halve the interval and narrow it down but remain with the right endpoint of each interval "on top of" S and with the left endpoint in S.

Consider $m = \frac{a_0+b_0}{2}$. If $[m, b_0] \cap S = \emptyset$ then let $a_1 = a_0$ and $b_1 = m$. If on the other hand $\exists s \in [m, b_0] \cap S$ then let $a_1 = s$ and $b_1 = b_0$. Define $I_1 = [a_1, b_1]$. Repeat this process to define all $I_n, n \in \mathbb{N}$.

The intervals I_n have the following properties:

P1 : $I_{n+1} \subseteq I_n$. This is visible from the definition of I_{n+1}. Its endpoints are either endpoints of I_n or are points from inside I_n.

P2 : $\forall n \in \mathbb{N} : b_n$ upper bound of S. We show this by induction on n. By choice b_0 is an upper bound. Now assume that b_n is an upper bound. If $b_{n+1} = b_n$ then it is an upper bound. If $b_{n+1} = \frac{a_n+b_n}{2}$ then because $S \cap [b_{n+1}, b_n] = \emptyset$ and it also follows that b_{n+1} is an upper bound [8].

P3 : $\forall n \in \mathbb{N}$: I_n non-empty. This also follows by induction and by the field axioms of $\mathbb{R}$.

P4 : $\forall n \in \mathbb{N} : a_n \in S$. This follows by induction and definition of left endpoints.

P5 : $\forall n \in \mathbb{N} : |I_n| \leq \frac{b_0-a_0}{2^n}$. [9]

[8] Assume b_{n+1} is not an upper bound of S, so there exists $s' \in S$ with $s' > b_{n+1}$. But by induction b_n is an upper bound, which means $b_{n+1} < s' \leq b_n$, so $s' \in [b_{n+1}, b_n]$, which contradicts $S \cap [b_{n+1}, b_n] = \emptyset$.

[9] We show this by induction on n. Base case $n = 0$ holds by definition of I_0. Assume $|I_n| \leq \frac{b_0-a_0}{2^n}$. For I_{n+1} we observe that its length is either half that of I_n or less than half when $[\frac{a_n+b_n}{2}, b_n] \cap S \neq \emptyset$

$P1$ and $P3$ satisfy the requirements of NIP, so we know $\alpha \in \cap_{n\in\mathbb{N}} I_n$ exists.

We want to show that $\alpha = supS$.

Proof Part 26.2.2. Assume α is not an upper bound of S. Then there exists $s \in S$ with $s > \alpha$. Let $\epsilon = s - \alpha > 0$. Using the Archimedean property we choose $m \in \mathbb{N}$ such that $I_m = [a_m, b_m]$ with $|I_m| < \epsilon$ [10]. Then $\alpha \in I_m$, but $s \notin I_m$ and furthermore $b_m < s$. This is a contradiction to property $P2$, so α is an upper bound of S.

[10] We use property $P5$. From $|I_m| \leq \frac{b_0-a_0}{2^m} < \epsilon$, we get $m > log_2(\frac{b_0-a_0}{\epsilon})$.

Now assume α is not the smallest upper bound of S. Then there exists an upper bound β of S with $\beta < \alpha$. Let $\epsilon = \alpha - \beta > 0$. Again we choose $m \in \mathbb{N}$ such that $I_m = [a_m, b_m]$ with $|I_m| < \epsilon$. That pushes a_m between β and α: $\beta < a_m \leq \alpha$. But according to property $P4$, $a_m \in S$, so $\beta < a_m$ contradicts the fact that β is an upper bound of S. So α is the smallest upper bound of S: $\alpha = supS$. This proves $NIP + AP \Rightarrow DDC$

□

Theorem 26.3. $DDC \Leftrightarrow MC$

Proof. ($\Rightarrow$) Given is a monotone increasing sequence (a_n) bound from above. We define $A = \{a_n : n \in \mathbb{N}\}$, a set that is bound from above. From DDC it follows that least upper bound c of A exists. We want to show that $\lim_{n\to\infty} a_n = c$. For all $\epsilon > 0$ we have $c - \epsilon < c$, so $c - \epsilon$ cannot be an upper bound of A (c is the least upper bound). That means that there exists $n_0 \in \mathbb{N}$ with $a_{n_0} > c - \epsilon$. Since the sequence is monotone increasing, we have

$$\forall n \geq n_0 : a_n \geq a_{n_0} > c - \epsilon \Rightarrow |c - a_n| < \epsilon$$

which proves $a_n \to c$.
($\Leftarrow$) We first want to show $MC \Rightarrow AP$. Given MC assume that AP doesn't hold, so there exists $x \in \mathbb{R}$ bigger than any natural number. This means x is an upper bound for the sequence $a_n = n$, a monotone increasing sequence. From MC it then follows that a_n converges to a limit c. The sequence $b_n = n + 1$ is a_n shifted to the left, so it is also convergent with the same limit c. Taking the limit on the sequence equation $b_n = a_n + 1$ we get $c = c + 1$, a contradiction. So $MC \Rightarrow AP$.

To show that $MC \Rightarrow DDC$ we are given non-empty set S with $a_0 \in S$ bound from above by $b_0 \in \mathbb{R}$. We define the same nested intervals as in the Proof Part 26.2.1 of the proof of Theorem 26.2.

The same properties P_1 to P_5 for I_n as stated in Proof Part 26.2.1 hold. The sequence (a_n) is in S and monotone increasing and the sequence (b_n) is made of upper bounds of S and is monotone decreasing. (a_n) is bound from above and monotone so according to MC it converges to a limit α.

We want to show that $\alpha = supS$. We will use the exact same argument as in the Proof Part 26.2.2 of the proof of Theorem 26.2[11]. This proves $MC \Rightarrow DDC$ □

[11] The only difference in the two proofs is that in this proof MC ensures the existence of α and in the previous proof it was NIP.

Theorem 26.4. $DDC \Leftrightarrow BW + AP$[12]

[12] Once again Bolzano-Weierstrass BW is not enough to get back to Dedekind Completeness DDC. We need the field to be Archimedean AP.

Proof. ($\Rightarrow$) We have already seen $DDC \Rightarrow AP$ (Theorem 26.2).

Proof Part 26.4.1. To prove $DDC \Rightarrow BW$ we are given a bounded sequence (s_n):

$$\exists a_0, b_0 \in \mathbb{R} \text{ such that } \forall n \in \mathbb{N} : a_0 \leq s_n \leq b_0$$

We define interval $I_0 = [a_0, b_0]$ and divide it in half at $c = \frac{a_0+b_0}{2}$. At least one of the two intervals $[a_0, c]$, $[c, b_0]$ has an infinite number of elements of the sequence s_n[13]. Define I_1 to be either $[a_0, c]$ or $[c, b_0]$ with an infinite number of elements of s_n. We repeat this process recursively, defining I_m to be one of the halves of I_{m-1} that has an infinite number of elements of (s_n). We get a sequence of nested intervals (I_m) of decreasing length $|I_m| = \frac{a_0+b_0}{2^m}$.

[13] Otherwise (s_n) would not be an infinite sequence.

We define $f : \mathbb{N} \to \mathbb{N}$ recursively as

$$\begin{cases} f(1) & = 1 \\ f(n) & = min\{i > f(n-1) : s_i \in I_{n-1}\} \end{cases}$$

The set $\{i > f(n-1) : s_i \in I_{n-1}\}$ is a non-empty, infinite subset[14] of $\mathbb{N}$, so its minimum exists and f is well defined and by definition strictly monotone increasing. We define subsequence (s'_n) as $s'_n = s_{f(n)}$, well defined because f is strictly monotone increasing.

[14] By definition of I_{n-1} there are an infinite number of elements s_i in I_{n-1}, so there are an infinite number of indices i in $\{i > f(n-1) : s_i \in I_{n-1}\}$. Also any non-empty subset of $\mathbb{N}$ has a smallest element.

Proof Part 26.4.2. From DDC we know that NIP holds so $\alpha \in \cap_{m\in\mathbb{N}} I_m$ exists. We claim that $s'_n \to \alpha$.

Because of AP we have for all $\epsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that $|I_{n_0}| < \epsilon$. We have $\alpha \in I_{n_0}$ and for all $n > f^{-1}(n_0) : s'_n \in I_{n_0}$. This means for all $n > f^{-1}(n_0) : |s'_n - \alpha| < \epsilon$ and (s'_n) is a subsequence of (s_n) that converges to α.

($\Leftarrow$)

Proof Part 26.4.3. We are going to prove this direction by going through NIP. Given nested non-empty intervals $I_{n+1} \subseteq I_n$ we define sequence (s_n) by choosing an arbitrary element from each I_n and setting it to be s_n. According to BW there exists a subsequence (s'_n) of (s_n) that converges $s'_n \to c$. We claim that $c \in \cap_{n\in\mathbb{N}} I_n$.

Proof Part 26.4.4. Assume $c \notin \cap_{n\in\mathbb{N}} I_n$. Then there must exist $n_0 \in \mathbb{N}$ such that $c \notin I_{n_0} = [a_{n_0}, b_{n_0}]$. Either $c < a_{n_0}$ or $c > b_{n_0}$. Let's consider $c < a_{n_0}$ (the other case is very similar). $\epsilon = \frac{a_{n_0}-c}{2} > 0$. We have $s'_n \to c$, so there exists n_1 such that $\forall n > n_1 : |s'_n - c| < \epsilon$. So for

$\forall n > max(n_0, n_1) : s'_n < c + \epsilon < a_{n_0}$. But (s'_n) is a subsequence of (s_n) so there must exist $m \in \mathbb{N}$ with $f^{-1}(m) > max(n_0, n_1)$. We have $s'_m = s_{f^{-1}(m)} \in I_{f^{-1}(m)}$. So $s'_m \in I_{f^{-1}(m)} \subseteq I_{n_0}$ and $s'_m < a_{n_0}$ which is a contradiction. This means $c \in \cap_{n\in\mathbb{N}} I_n$ and $BW \Rightarrow NIP$ which together with AP gets us to DDC according to Theorem 26.2.

□

Theorem 26.5. *$DDC \Leftrightarrow CC + AP$*[15]

Proof. ($\Rightarrow$) We have already seen $DDC \Rightarrow AP$ (Theorem 26.2). To prove $DDC \Rightarrow CC$ we are given a Cauchy sequence (a_n). We first show that (a_n) is bounded. From the definition of a Cauchy sequence[16] we get for $\epsilon = 1$ there exists $N \in \mathbb{N}$ such that $\forall m \geq N : |a_n - a_N| < 1 \Rightarrow |a_n| < 1 + |a_N|$. Define $M = max\{|a_1|, |a_2|, \ldots, |a_{N-1}|, |a_N| + 1\}$ and we have $\forall n \in \mathbb{N} : |a_n| < M$.

The Cauchy sequence (a_n) is bounded so using $DDC \Rightarrow BW$ from Theorem 26.4 we know there is a subsequence of (a_n) that converges. Let $f : \mathbb{N} \to \mathbb{N}$ be the strictly monotone increasing function that defines the converging subsequence $a'_n = a_{f(n)}$ and let $\lim_{n\to\infty} a'_n = c$.

For all $\epsilon > 0$ we have:

$$\exists n_1 \in \mathbb{N} \text{ such that } \forall n \geq n_1 : |a_n - a_{n_1}| < \frac{\epsilon}{2}$$

and then

$$\exists n_2 \geq f^{-1}(n_1) \text{ such that } \forall n \geq n_2 : |a'_n - c| < \frac{\epsilon}{2}$$

So

$$\begin{aligned} \forall n \geq n_2 : |a_n - c| &= |a_n - a'_{n_2} + a'_{n_2} - c| \leq |a_n - a'_{n_2}| + |a'_{n_2} - c| \\ &= |a_n - a_{f(n_2)}| + |a'_{n_2} - c| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon \end{aligned}$$

It means (a_n) converges to c and $DDC \Rightarrow CC + AP$.

($\Leftarrow$) We will show that $CC + AP \Rightarrow BW$. We are given a bounded sequence (s_n) and we use the same subsequence construction as in the Proof Part 26.4.1 of Theorem 26.4. We claim that the so constructed subsequence (s'_n) is a Cauchy sequence. Indeed for all $\epsilon > 0$ there exists $N \in \mathbb{N}$ such that $|I_N| < \epsilon$ (again we need AP here). We then have:

$$\forall m, n \geq N : s'_n, s'_m \in I_N \Rightarrow |s'_n - s'_m| \leq |I_N| < \epsilon$$

So (s'_n) is a Cauchy sequence and by CC it converges which means that (s_n) has a convergent subsequence. □

Finish up.

[15] As seen before with NIP and BW the Cauchy Criterion CC is not enough to get back to Dedekind Completeness DDC. We need the field to be Archimedean AP.

[16] A sequence (a_n) is a Cauchy sequence if $\forall \epsilon > 0 : \exists N \in \mathbb{N}$ such that $\forall m, n \geq N : |a_m - a_n| < \epsilon$.

27
Enigma

Enigma Machines and how the internal wiring of their rotors was reverse-engineered is the topic of this section. We will follow a simplified version of Rejewski's description [1] of his work.

An Enigma Machine applies a series of permutations to each typed letter, mapping it to another letter (which lights up on the Lampboard, see Figure 27.1), thus encrypting a message[2].

Electrical current flows from the typed letter through the plugboard, then the right rotor, the middle rotor and the left rotor. It then enters the reflector and goes back in reverse order through the same components ending up on the lampboard where the corresponding encrypted letter lights up. The plugboard, rotors and reflector have internal wirings which correspond to permutations in S_{26}[3]. The resulting permutation applied to a letter by the Enigma Machine is the product[4]:

$$P^{-1}N_k^{-1}M_k^{-1}L_k^{-1}RL_kM_kN_kP$$

The rotors rotate after each typed letter in the style of an odometer: the right rotor rotates one position after each typed letter, the middle rotor rotates one position after each full-circle rotation of the right rotor and the left rotor rotates one position after each full-circle rotation of the middle rotor. Rotating the rotors changes the permutations they will apply to a letter, so their permutations are indexed by k in the product above and in the Figure 27.1. We will see later how we can model these rotations with permutations.

The reflector pairs each letter with another (always different) letter, thus it is a product of 13 disjoint transpositions. A permutation made out of only disjoint transpositions is called a **proper involution**. We will see why the Enigma Machine designers chose a proper involution for the reflector.

First though we need to collect some facts about permutations that we will use in our Enigma Machine analysis.

[1] Marian Rejewski. How Polish mathematicians broke the Enigma cipher. *IEEE Annals of the History of Computing*, 3(3): 213–234, 1981. ISSN 1058-6180

[2] Enigma machines were used by the Nazis in WWII to encrypt/decrypt messages. The machines are rotor-based electromechanical typewriters. http://en.wikipedia.org/wiki/Enigma_machine has a detailed description of their internals.

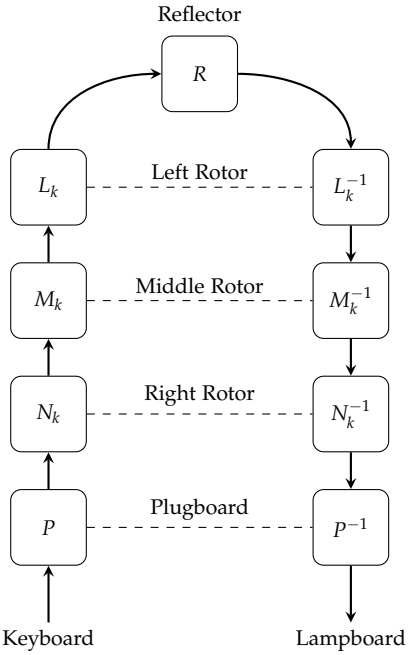


Figure 27.1: Enigma Permutations

[3] S_{26} is the symmetric group of permutations of $\{1, 2, \ldots, 26\}$.

[4] We use the convention of permutation product as function composition, so for $A, B \in S_{26}$ we have $AB = A \circ B$ and $AB(x) = A(B(x))$.

Theorem 27.1. *Every permutation can be written as a product of disjoint cycles. This product is unique (ignoring cycle order and order of elements in cycle).*

Proof. Let $\pi \in S_n$ be a permutation.

We start by choosing an arbitrary $x_1 \in \{1, \ldots, n\}$ and define for it the set

$$T_{x_1} = \{x_1, \pi(x_1), \pi^2(x_1), \ldots\}$$

$\{1, \ldots, n\}$ is finite, $T_{x_1} \subseteq \{1, \ldots, n\}$, so T_{x_1} is finite too. This means that sooner or later there exist $i < j$ with $\pi^j(x_1) = \pi^i(x_1)$ or $x_1 = \pi^{j-i}(x_1)$. Then $ord(x_1) = j - i$ is the order of x_1. It follows that

$$T_{x_1} = \{x_1, \pi(x_1), \pi^2(x_1), \ldots, \pi^{ord(x_1)-1}(x_1)\}$$

and T_{x_1} implies the cycle $(x_1, \pi(x_1), \pi^2(x_1), \ldots, \pi^{ord(x_1)-1}(x_1))$. T_{x_1} is called the π-orbit of x_1. Let's denote this cycle

$$\langle x_1 \rangle = (x_1, \pi(x_1), \pi^2(x_1), \ldots, \pi^{ord(x_1)-1}(x_1))$$

We now choose an arbitrary $x_2 \in \{1, \ldots, n\} \setminus T_{x_1}$. If there is no such x_2 we stop this process and jump to the section in the proof after all x_k have been chosen. We similarly define T_{x_2} and cycle $\langle x_2 \rangle$.

T_{x_2} and T_{x_1} are disjoint[5].

We continue and choose an arbitrary $x_3 \in \{1, \ldots, n\} \setminus (T_{x_1} \cup T_{x_2})$, and in general an arbitrary

$$x_k \in \{1, \ldots, n\} \setminus (\bigcup_{i=1}^{k-1} T_{x_i})$$

Since all the T_{x_i} are non-empty and $\{1, \ldots, n\}$ is finite, we eventually have to stop. We then have chosen $x_1, x_2, \ldots, x_k$ and the corresponding sets $T_{x_1}, T_{x_2}, \ldots, T_{x_k}$ and cycles $\langle x_1 \rangle, \langle x_2 \rangle, \ldots, \langle x_k \rangle$.

The sets T_{x_i} and their corresponding cycles are by construction pairwise disjoint. We also have $\{1, \ldots, n\} = \bigcup_{i=1}^{k} T_{x_i}$.

We define the permutation ρ as the product of the cycles chosen above:

$$\rho = \prod_{i=1}^{k} \langle x_i \rangle$$

and show that $\rho = \pi$.

For all $y \in \{1, \ldots, n\}$ there exists a unique $1 \leq i \leq k$ such that $y \in T_{x_i}$.[6]

So $y = \pi^j(x_i)$ for some index $0 \leq j < ord(x_i)$. Since the cycles are disjoint, only cycle $\langle x_i \rangle$ from ρ affects y. We have

[5] Assume $y \in T_{x_2} \cap T_{x_1}$. Then $y = \pi^i(x_1)$ and $y = \pi^j(x_2)$. It follows that $x_2 \in T_{x_1}$ or $x_1 \in T_{x_2}$, either one of which contradicts how x_2 was chosen. Another way to see this is by defining the following relationship: $\forall a, b \in S_n : a \sim b \equiv \exists n \in \mathbb{N} : b = \pi^n(a)$. It's not hard to see that $a \sim b$ so defined is an equivalence relationship and with it the T_{x_i} become equivalence classes and partition S_n.

[6] Because $\{1, \ldots, n\} = \bigcup_{i=1}^{k} T_{x_i}$ and $T_{x_1}, T_{x_2}, \ldots, T_{x_k}$ are pairwise disjoint and form a partition of $\{1, \ldots, n\}$.

$$\begin{aligned}\rho(y) &= \langle x_i \rangle(y) \\ &= \langle x_i \rangle(\pi^j(x_i)) \\ &= \pi^{j+1}(x_i) \\ &= \pi(\pi^j(x_i)) \\ &= \pi(y)\end{aligned}$$

□

Given two permutations $\pi, \rho \in S_n$, the product $\rho\pi\rho^{-1}$ is called a conjugate of π.

Theorem 27.2. *Conjugation preserves cycle structure, i.e. conjugates have cycles of the same length with the same multiplicity.*

Incidentally Theorem 27.2 is the reason why the products $\pi\rho$ and $\rho\pi$ have the same cycle structure. Even though in general $\pi\rho \neq \rho\pi$, $\pi\rho$ and $\rho\pi$ are conjugate. This was the question asked in Exercise 5.5 on page 34 from Michael Artin. *Algebra*. Addison Wesley, 2 edition, 2010. ISBN 0132413779.

Proof. Consider $\pi, \rho \in S_n$. From Theorem 27.1 we know that π is a product of disjoint cycles $\pi = \prod_{i=1}^{k} \rho_i$. For the conjugate $\rho\pi\rho^{-1}$ we can write:

$$\begin{aligned}\rho\pi\rho^{-1} &= \rho\rho_1\rho_2\rho_3 \dots \rho_k\rho^{-1} \\ &= \rho\rho_1(\rho^{-1}\rho)\rho_2(\rho^{-1} \dots \rho)\rho_k\rho^{-1} \\ &= (\rho\rho_1\rho^{-1})(\rho\rho_2\rho^{-1}) \dots (\rho\rho_k\rho^{-1}) \\ &= \prod_{i=1}^{k} \rho\rho_i\rho^{-1}\end{aligned}$$

so it is enough to prove the theorem for a cycle.

Let $\rho = (a_1, a_2, \dots, a_r)$ be a cycle of length r. We have

$$(\rho\rho\rho^{-1})(\rho(a_i)) = (\rho\rho)(a_i) = \rho(a_{i+1})$$

so $\rho\rho\rho^{-1}$ will have the cycle $(\rho(a_1), \rho(a_2), \dots, \rho(a_r))$ with length r. Now assume that x is moved by $\rho\rho\rho^{-1}$, so $(\rho\rho\rho^{-1})(x) \neq x$. It follows that $(\rho\rho^{-1})(x) \neq \rho^{-1}(x)$ or $\rho(\rho^{-1}(x)) \neq \rho^{-1}(x)$. This means that $\rho^{-1}(x) \in (a_1, a_2, \dots, a_r)$ and $x \in (\rho(a_1), \rho(a_2), \dots, \rho(a_r))$. It follows that $\rho\rho\rho^{-1} = (\rho(a_1), \rho(a_2), \dots, \rho(a_r))$.

□

We have seen that an Enigma Machine permutation E is the product

$$\begin{aligned}E &= P^{-1}N_k^{-1}M_k^{-1}L_k^{-1}RL_kM_kN_kP \\ &= (L_kM_kN_kP)^{-1}R(L_kM_kN_kP) \\ &= QRQ^{-1}\end{aligned}$$

with $Q = (L_kM_kN_kP)^{-1}$. This means that E is a conjugate of the reflector permutation R, and according to Theorem 27.2 has the same

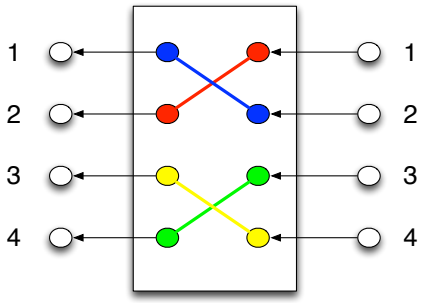


Figure 27.2: Initial Rotor

cycle structure as R. So E is a proper involution (because R is) and also $E^{-1} = E$. The same Enigma Machine configuration was used to encrypt and decrypt a message, which was probably why the Enigma Machine designers chose a proper involution for R and ultimately for E.

Let's analyse the rotor motion on the example in Figure 27.2. It shows a small rotor with an internal wiring doing a permutation from S_4. It has 4 inputs/outputs and permutation $(12)(34)$. Rotating it down one position as in Figure 27.3 doesn't change its internal wiring but shifts the inputs/outputs. Input one is now connected to the yellow wire instead of the red, input two to the red wire instead of the blue, etc. The resulting permutation is $(14)(23)$. The inputs have been shifted according to (4321) and the outputs according to (1234), so $(14)(23) = (1234)(12)(34)(1432)$. In general one rotation of a rotor is equivalent to conjugating it with the full cycle permutation σ, in other words we have $N_{k+1} = \sigma N_k \sigma^{-1}$. To see why this is true, consider input x touches the red wire in the rotor after the rotation. We don't know yet where the rotor will map x. We do know that if x touches the red wire in the rotor after the rotation, then $\sigma^{-1}(x)$ is touching the red wire before the rotation (because all inputs and outputs have been shifted down). Also we know where the rotor maps any input y before the rotation, namely to $N_k(y)$. So $\sigma^{-1}(x)$ is mapped to $N_k(\sigma^{-1}(x))$. And any output from before the rotation is shifted down once after the rotation, in this case to $\sigma(N_k(\sigma^{-1}(x)))$ after the rotation. Collecting this tracing into one expression, we have $N_{k+1}(x) = \sigma(N_k(\sigma^{-1}(x)))$.

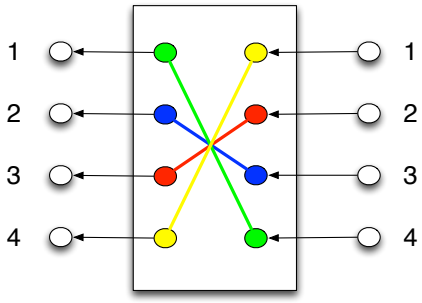
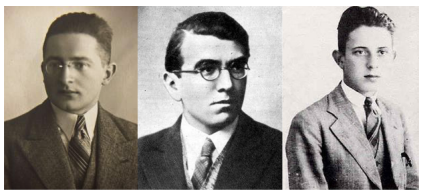


Figure 27.3: Rotor after one rotation

Assuming only the right rotor moves, the first six Enigma permutations are:

$$\begin{aligned}
A &= P^{-1}N_0^{-1}M_0^{-1}L_0^{-1}RL_0M_0N_0P \\
B &= P^{-1}\sigma N_0^{-1}\sigma^{-1}M_0^{-1}L_0^{-1}RL_0M_0\sigma N_0\sigma^{-1}P \\
C &= P^{-1}\sigma^2 N_0^{-1}\sigma^{-2}M_0^{-1}L_0^{-1}RL_0M_0\sigma^2 N_0\sigma^{-2}P \\
D &= P^{-1}\sigma^3 N_0^{-1}\sigma^{-3}M_0^{-1}L_0^{-1}RL_0M_0\sigma^3 N_0\sigma^{-3}P \\
E &= P^{-1}\sigma^4 N_0^{-1}\sigma^{-4}M_0^{-1}L_0^{-1}RL_0M_0\sigma^4 N_0\sigma^{-4}P \\
F &= P^{-1}\sigma^5 N_0^{-1}\sigma^{-5}M_0^{-1}L_0^{-1}RL_0M_0\sigma^5 N_0\sigma^{-5}P
\end{aligned} \tag{27.1}$$

The first six permutations are important because of how the German Nazis chose to operate Enigma. It was known to the code breakers[7] that after configuring Enigma to its daily settings and before sending a message, an operator would send a block of three letters twice. The three letters encoded a message key and because transmission lines were deemed unreliable, these three letters would be sent twice. This means that for each message transmission the input to permutations A and D was the same letter (similar for B and E and for C and F). The


[7] Marian Rejewski, Henryk Zygalski and Jerzy Różycki. http://en.wikipedia.org/wiki/Marian_Rejewski

code breakers had access to two months of intercepted messages and daily key settings. So they could determine that an unknown letter u was mapped by A to the observed letter x and by D to the observed letter y, so $A(u) = x$ and $D(u) = y$. Because A and D are each proper involutions[8] it also holds that $A(x) = u$ and $D(y) = u$. It follows that $AD(y) = A(D(y)) = A(u) = x$. So AD maps one observed letter to another observed letter. With enough messages in a given day, each letter of the alphabet will be observed which then completely defines AD and similarly BE and CF. So for a given day AD, BE and CF were known permutations. The goal now is to factor AD into A and D.

We need a way to compute how many possible factorizations there are and a way to generate all possibilities. To accomplish this we need to collect some properties of products of proper involutions. We will use a simplified approach similar to the approach described in chapter 3.8 of Lawrence and Zorzitto [2021][9].

[8] This is one example of why choosing a proper involution as the encryption permutation was a bad idea for Enigma Machines. As it turns out it was fatally bad: It was the main weakness that allowed the British *bombe machine* built at Bletchley Park by Alan Turing and Gordon Welchman to decrypt Enigma encrypted messages. http://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma#British_bombe

[9] J.W. Lawrence and F.A. Zorzitto. *An Introduction to Abstract Algebra: A Comprehensive Introduction*. Cambridge Mathematical Textbooks. Cambridge University Press, 2021. ISBN 9781108836654. URL https://books.google.com/books?id=PvQgEAAAQBAJ

Theorem 27.3. *Let $\pi = \tau\rho$ be the product of proper involutions τ and ρ and let $x \in \{1, \ldots, n\}$. Then the π-orbits of x and $\rho(x)$ are disjoint and have equal length.*

Proof. Reminder here that the π-orbit of x is

$$T_x = \{x, \pi(x), \pi^2(x), \ldots, \pi^{ord(x)-1}(x)\}$$

Assume the two orbits are not disjoint and $y \in T_x \cap T_{\rho(x)}$. For some integers i and j we have $y = \pi^i(x) = \pi^j(\rho(x))$. Let $m = ord(\rho(x))$ and let $(k-1)m < j \leq km$ for some k. Then

$$\pi^{i+km-j}(x) = \pi^{km}(\rho(x)) = \rho(x)$$

Let $n = i + km - j$ and so

$$\rho(\pi^n(x)) = \rho^2(x) = x$$

because ρ is a proper involution (so it is its own inverse).
We now have two cases: n can be even or odd.
When $n = 2l$:

$$\begin{aligned}
\rho\pi^n &= \rho\pi^l\pi^l \\
&= \rho \underbrace{(\tau\rho)(\tau\rho)\ldots(\tau\rho)}_{l\text{-times}} \pi^l \\
&= \underbrace{(\rho\tau)(\rho\tau)\ldots(\rho\tau)}_{l\text{-times}} \rho\pi^l \\
&= \pi^{-l}\rho\pi^l
\end{aligned}$$

This means that $\rho\pi^n$ is a conjugate of ρ and thus it is a proper involution and cannot have x mapping to itself. We have a contradiction.

When $n = 2l + 1$:

$$\begin{aligned}\rho\pi^n &= \rho\pi^l\tau\rho\pi^l \\ &= \rho\underbrace{(\tau\rho)(\tau\rho)\dots(\tau\rho)}_{l\text{-times}}\tau\rho\pi^l \\ &= \underbrace{(\rho\tau)(\rho\tau)\dots(\rho\tau)}_{l\text{-times}}\rho\tau\rho\pi^l \\ &= (\rho\pi)^{-1}\tau(\rho\pi^l)\end{aligned}$$

This means that $\rho\pi^n$ is a conjugate of τ and thus it is a proper involution and cannot have x mapping to itself. Again we have a contradiction.

We just showed that the π-orbits of x and $\rho(x)$ are disjoint. To show that the orbits have the same length, we again reach for this useful identity: for any integer m we have $\rho\pi^m = \pi^{-m}\rho$. This is because

$$\begin{aligned}\rho\pi^m &= \rho\underbrace{(\tau\rho)(\tau\rho)\dots(\tau\rho)}_{m\text{-times}} \\ &= \underbrace{(\rho\tau)(\rho\tau)\dots(\rho\tau)}_{m\text{-times}}\rho \\ &= \pi^{-m}\rho\end{aligned}$$

The identity allows for this equivalence:

$$\pi^{-m}(\rho(x)) = \rho(x) \Leftrightarrow \rho(x) = \rho(\pi^m(x)) \Leftrightarrow \pi^m(x) = x$$

It means that the π^{-1}-orbit of $\rho(x)$ has the same length as the π-orbit of x. But π^{-1}-orbit and π-orbit of an element are the same[10]. So the π-orbits of x and $\rho(x)$ have the same length. □

[10] Just walk the cycle backwards.

Theorem 27.4. *Let $\pi = \tau\rho$ be the product of proper involutions τ and ρ and let $x \in \{1, \dots, n\}$. Then the π-orbits of $\tau(x)$ and $\rho(x)$ are equal. In addition to that, the π-orbit of x is mapped by τ and ρ onto this common π-orbit of $\tau(x)$ and $\rho(x)$.*

Proof. Keeping in mind that a proper inversion is its own inverse, we have:

$$\begin{aligned}\rho(x) &= \rho(x) \\ &= \rho(\tau^2(x)) \\ &= (\rho\tau)(\tau(x)) \\ &= \pi^{-1}(\tau(x))\end{aligned}$$

so then $\pi(\rho(x)) = \tau(x)$ and $\tau(x)$ is in the π-orbit of $\rho(x)$.
Using the $\rho\pi^m = \pi^{-m}\rho$ identity again, we see that

$$\rho(\pi^m(x)) = \pi^{-m}(\rho(x))$$

so ρ maps the π-orbit of x onto the π-orbit of $\rho(x)$.
To see where τ maps the π-orbit of x we need a similar identity, so lets deduce it:

$$\begin{aligned}
\tau\pi^m &= \tau \underbrace{(\tau\rho)(\tau\rho)\dots(\tau\rho)}_{m\text{-times}} \\
&= \tau\tau \underbrace{(\rho\tau)(\rho\tau)\dots(\rho\tau)}_{m-1\text{-times}}\rho \\
&= \pi^{-m+1}\rho \\
&= \pi^{-m+1}\pi^{-1}\tau \\
&= \pi^{-m}\tau
\end{aligned}$$

We can use this identity for:

$$\begin{aligned}
\tau(\pi^m(x)) &= \pi^{-m}(\tau(x)) \\
&= \pi^{-m}(\pi\rho)(x) \\
&= \pi^{1-m}(\rho(x))
\end{aligned}$$

so τ also maps the π-orbit of x onto the π-orbit of $\rho(x)$.

□

Theorem 27.5. *Let $\pi = \tau\rho$ be the product of proper involutions τ and ρ and let $x \in \{1, \dots, n\}$. Let $y \notin T_x \cup T_{\rho(x)}$. Then $\rho(y) \notin T_x \cup T_{\rho(x)}$.*

Proof. Assume $\rho(y) \in T_x \cup T_{\rho(x)}$. Two cases:
First case: $\rho(y) \in T_x$. Then $\rho(y) = \pi^m(x)$ for some $m \in \mathbb{N}$. Keeping in mind again that the a proper involution is its own inverse, we apply ρ to both sides to get

$$y = (\rho\pi^m)(x) = \pi^{-m}(\rho(x))$$

so $y \in T_{\rho(x)}$ which is a contradiction.
Second case: $\rho(y) \in T_{\rho(x)}$. We proceed similarly:
$\rho(y) = \pi^m(\rho(x))$ for some $m \in \mathbb{N}$ so

$$y = (\rho\pi^m)(\rho(x)) = \pi^{-m}(\rho\rho)(x) = \pi^{-m}(x)$$

so $y \in T_x$ which is a contradiction.

□

Theorem 27.6. *Let $\pi = \tau\rho$ be the product of proper involutions τ and ρ. Then the cycle lengths of π that are greater than one come in even numbers.*

Proof. Let (ab) be a cycle of τ. We have two cases:

Case 1: (ab) is also a cycle of ρ. Then the product π has cycles (a) and (b) of length one.

Case 2: (ab) is not a cycle of ρ. Then it must have a cycle (ac_1) for some c_1. In τ there must be a cycle (c_1c_2) for some c_2. In ρ again there must be a cycle (c_2c_3) for some c_3, ... (remember, ρ and τ are proper involutions, so each element participates in one and only one 2-cycle). We stop with a cycle $(c_{2k}b)$ in ρ, which eventually must happen. Then the product π has cycles $(c_{2k}c_{2k-2}\dots c_2a)$ and $(c_1c_3c_{2k-1}b)$ of length k. □

We are ready to tackle the factorization. To recap, we have a permutation π that we know and we also know it is a product of two proper involutions. Our goal is to find out how many possible factorizations into two proper involutions there are and how do we generate all the factorizations (because we need the factors to determine the first rotor wiring).

Theorem 27.7. *Let $\pi \in S_{2n}$ be a permutation composed of just two disjoint cycles of length n. Then π has exactly n factorizations into two proper involutions.*

Proof. Pick an $a \in \{1, \dots, 2n\}$. It is part of one of the two cycles. We are looking for possible $\pi = \tau\rho$ factorizations, with both τ and ρ being proper involutions. The two π-orbits of the two cycles are $A := T_a = \{a, \pi(a), \dots, \pi^{n-1}(a)\}$ and $B := S_{2n} \setminus A$.

We are going to construct all the possible ρ using the previous theorems as constraints (once a possible ρ is constructed, it also fully determines the other factor, τ).

For example, because of theorem 27.3 we have to pick some element from B for $\rho(a)$: $\rho(a) \in B$. We will argue that once this choice has been made, the complete factorization has been determined. Let's see why. What value should $\rho(\pi(a))$ take ? Again, using the identity $\rho\pi^m = \pi^{-m}\rho$, we get $\rho(\pi(a)) = \pi^{-1}(\rho(a))$. By repeatedly using the identity as we move π-forward in the cycle with a, we move π-backwards in the other cycle and at each stop we make another pair for the proper involution ρ.

Figure 27.4 shows the process for an example with two cycles of length four. After setting an a and picking where to map $\rho(a)$, everything else is determined (the labels in the figure show the expressions determining the relationships).

There are n ways to pick an element from B, hence we can construct n different ρ, so n different factorizations $\pi = \tau\rho$. It doesn't matter

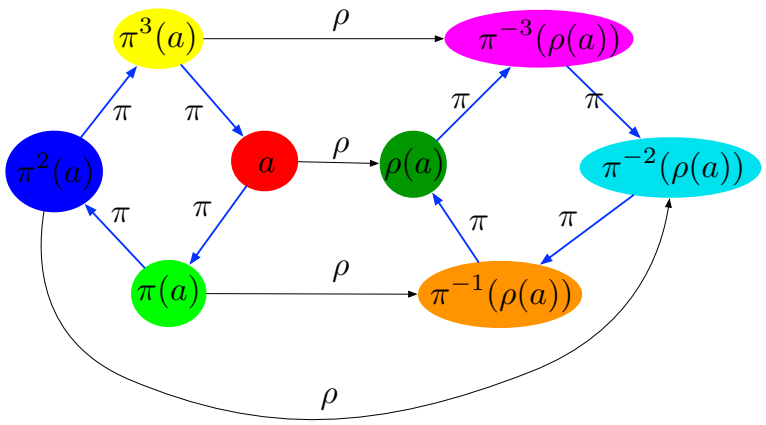


Figure 27.4: Constructing ρ.

which a we start with. Through the cycle-wise rotation in cycle with a and counter-cycle rotation in the cycle with $\rho(a)$, we see all n possible ρ constructions, regardless which a is our anchor. Does it matter from which cycle we choose the anchor ? It doesn't because again the same factorizations would be produced if all the ρ-arrows in figure 27.4 were reversed.

□

Now theorem 27.6 assures us that any product of two proper involutions has cycle lengths greater than one occurring an even number of times. We can always pair up two cycles of the same length. Theorems 27.4 and 27.5 help us isolate the pairings and construct the factors by restricting ourselves to each pairing and using the construction from theorem 27.7 to build the possible factors for each paired restriction. We multiply all the restricted τ's to get the unrestricted τ and multiply all the restricted ρ's to get the unrestricted ρ.

So how many factorizations are there for a given product π? Lets say π has $2m_k$ cycles of length k. In how many ways can we pair up these $2m_k$ cycles?

Theorem 27.8. *The number of ways W to form m pairs from the integers $\{1, 2, \ldots, 2m\}$ is*

$$W = \frac{(2m)!}{2^m m!}$$

Proof. Integer one can be paired with $2m - 1$ other integers. Picking an unpaired remaining integer, it can be paired with $2m - 3$ other integers, etc.

It follows that

$$
\begin{aligned}
W &= (2n-1)(2n-3)\ldots 5\cdot 3\cdot 1 \\
&= (2n-1)(2n-3)\ldots 5\cdot 3\cdot 1\cdot \frac{(2m)(2n-2)(2n-4)\ldots 4\cdot 2}{(2m)(2n-2)(2n-4)\ldots 4\cdot 2} \\
&= \frac{(2m)!}{(2m)(2n-2)(2n-4)\ldots 4\cdot 2} \\
&= \frac{(2m)!}{2^m m!}
\end{aligned}
$$

□

Which means that if π has $2m_k$ cycles of length k, we can produce

$$\frac{k^{m_k}(2m_k)!}{2^{m_k}m_k!}$$

factorizations restricted to those cycles. Multiplying over all the possibly cycle lengths greater than one gives us the number of factorizations

$$\prod_{k \text{ cycle length}} \frac{k^{m_k}(2m_k)!}{2^{m_k}m_k!}$$

We return to the first six Enigma permutations 27.1. After using the factorization to factor AD, BE and CF, we know possible solutions for A, B, C, D, E and F. The plugboard settings P were found out from French spies, σ is the full cycle permutation. We can drop the subscripts from M and L because we assume they don't rotate for the first six typed letters and from N because we know how to express its rotations. We get:

$$
\begin{aligned}
A &= P^{-1}N^{-1}M^{-1}L^{-1}RLMNP \\
B &= P^{-1}\sigma^{-1}N^{-1}\sigma M^{-1}L^{-1}RLM\sigma^{-1}N\sigma P \\
C &= P^{-1}\sigma^{-2}N^{-1}\sigma^2 M^{-1}L^{-1}RLM\sigma^{-2}N\sigma^2 P \\
D &= P^{-1}\sigma^{-3}N^{-1}\sigma^3 M^{-1}L^{-1}RLM\sigma^{-3}N\sigma^3 P \\
E &= P^{-1}\sigma^{-4}N^{-1}\sigma^4 M^{-1}L^{-1}RLM\sigma^{-4}N\sigma^4 P \\
F &= P^{-1}\sigma^{-5}N^{-1}\sigma^5 M^{-1}L^{-1}RLM\sigma^{-5}N\sigma^5 P
\end{aligned}
\tag{27.2}
$$

The unknowns in equations 27.2 are M, L, R and N. Our goal is to compute N. To simplify working with these equations, we define $G = M^{-1}L^{-1}RLM$, move as many known permutations as we can to the left side of the equations and name the left sides U, V, W, X, Y, Z:

$$
\begin{aligned}
U &:= PAP^{-1} = N^{-1}GN \\
V &:= \sigma PBP^{-1}\sigma^{-1} = N^{-1}\sigma G\sigma^{-1}N \\
W &:= \sigma^2 PCP^{-1}\sigma^{-2} = N^{-1}\sigma^2 G\sigma^{-2}N \\
X &:= \sigma^3 PDP^{-1}\sigma^{-3} = N^{-1}\sigma^3 G\sigma^{-3}N \\
Y &:= \sigma^4 PEP^{-1}\sigma^{-4} = N^{-1}\sigma^4 G\sigma^{-4}N \\
Z &:= \sigma^5 PFP^{-1}\sigma^{-5} = N^{-1}\sigma^5 G\sigma^{-5}N
\end{aligned}
\tag{27.3}
$$

We now multiply subsequent equations to get the following five equations:

$$
\begin{aligned}
UV &= N^{-1}G\sigma G\sigma^{-1}N \\
VW &= N^{-1}\sigma G\sigma G\sigma^{-2}N \\
WX &= N^{-1}\sigma^2 G\sigma G\sigma^{-3}N \\
XY &= N^{-1}\sigma^3 G\sigma G\sigma^{-4}N \\
YZ &= N^{-1}\sigma^4 G\sigma G\sigma^{-5}N
\end{aligned}
\tag{27.4}
$$

We eliminate G by inserting VW into the first equation, WX into the second etc:

$$
\begin{aligned}
UV &= N^{-1}\sigma^{-1}NVWN\sigma N \\
VW &= N^{-1}\sigma^{-1}NWXN\sigma N \\
WX &= N^{-1}\sigma^{-1}NXYN\sigma N \\
XY &= N^{-1}\sigma^{-1}NYZN\sigma N
\end{aligned}
\tag{27.5}
$$

We define the new unknown $H = N^{-1}\sigma^{-1}N$ and get

$$
\begin{aligned}
UV &= H(VW)H^{-1} \\
VW &= H(WX)H^{-1} \\
WX &= H(XY)H^{-1} \\
XY &= H(YZ)H^{-1}
\end{aligned}
\tag{27.6}
$$

So UV, VW etc are conjugated by H. Each of the four equations in 27.6 usually yielded several dozen solutions for H and usually there is only one common solution to all four equations. This gave the code breakers H and thus N, the internal wiring of the right rotor. The second rotor was cracked the same way because the German Nazis switched rotor positions every 3 months[11] and a new rotor slid into the rightmost position. Rejewski and his team had daily keys for two months which happened to overlap with one rotor switching. They didn't have daily keys for a longer period that would span two rotor

Figure 27.5: An Enigma on display at the *Museum für Kommunikation Frankfurt* http://www.mfk-frankfurt.de

[11] It is amazing how little things in cryptography can trip up security of a system and open the doors to attackers. The German Nazis no doubt believed that by switching rotors they would increase the number of possible permutations (correct) and thus increase the security of their system (incorrect).

switchings, so they couldn't use this method to deduce the wiring of the third rotor. It's not clear how Rejewski and his colleagues cracked the wiring of the third rotor and the wiring of the reflector[12], but they did. Using only two months worth of daily keys and intercepted messages the Polish cryptologists were able to deduce the internal wirings of the rotors of the Enigma Machine and with that were able to build a functioning replica of it. This achievement jumpstarted the effort of the British team at Bletchley Park and eventually resulted in the capability of the Allied Forces to listen in on all the transmissions encrypted with Enigma.

[12] For more details and possible solutions, see J. Vábek. On Rejewski's solution of Enigma cipher. In *PROCEEDINGS OF WDS 2006*. MATFYZPRESS, 2006 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.186.9963&rank=1.

28
Burnside Pólya Counting

Burnside Pólya Counting is the topic of this note. Group theory is very rich in structure[1]. Its use in counting combinatorial objects is a very cool application and a fine excuse to explore it.

[1] It's so rich that there are often many ways to prove some properties. I will try to write this note mostly from memory and will probably use awkward detours where more efficient ways are available.

Motivating Example

Suppose you want to count in how many ways you can color a paper strip (like in figure 28.1) with k cells using n colors. That is pretty simple: each cell can be colored in n ways independent of any other cell and there are k cells, so there are n^k ways to color the strip of paper. Now let us throw in a wrinkle: from a counting perspective a strip rotated by 180° is considered the same as the original strip, so the two strips seen in figure 28.2 should be counted as one strip.

Figure 28.1: Paper strip with eleven cells colored with four colors.

You would say that's fine. We just divide by two and get $\frac{n^k}{2}$. Each color sequence and its inverse are considered one strip. That is almost right. There exist color sequence palindromes like the one in figure 28.3. In those cases rotating the strip does not result in a new color sequence, so it is not correct to divide those by two since there is only one color sequence associated with a strip. We would be undercounting.

Figure 28.2: These two strips are the same and contribute one to the counting. The bottom strip has the reversed color sequence of the top strip.

So you say fine: we first put those to the side and then divide the rest by two. Our counting total becomes counting color sequences that are not palindromes and dividing that number by two and counting all color sequences that are palindromes. Let's do that. Let S be the set of colored paper strips with color sequences that are not palindromes and let P be the set of colored paper strips with color sequences that are palindromes.

Figure 28.3: Strip with a color sequence that is the same when read backwards.

There are n^k color sequences and $|P|$ of them are palindromes. So

$$\text{number of ways to color a strip} = |S| + |P| = \frac{n^k - |P|}{2} + |P| = \frac{n^k}{2} + \frac{|P|}{2}$$

We need to determine the number of palindromes. The color choice of the first cell also determines the color of the last cell (because it needs to be the same when read backwards). Similarly the color choice of the second cell determines the color of the next to last cell and so on. So we only have half the choices of an unconstrained color sequence. Therefore

$$|P| = \begin{cases} n^{\frac{k}{2}}, \text{ when } k \text{ is even} \\ n \times n^{\frac{k-1}{2}} = n^{\frac{k+1}{2}}, \text{ when } k \text{ is odd} \end{cases}$$

By using the ceiling function we can collapse the two cases into:

$$|P| = n^{\lceil \frac{k}{2} \rceil}$$

and get

$$\text{number of ways to color a strip} = \frac{n^k}{2} + \frac{n^{\lceil \frac{k}{2} \rceil}}{2}$$

This wasn't too bad but one could imagine that more complicated counting scenarios with objects where different configurations are considered the same can become quite tricky without a systematic approach[2]. Let's look back at the colored paper strips. Maybe we can tickle out a systematic approach.

[2] For example colored necklaces where rotation and flipping is considered the same object. Or counting how many different molecules you can form with a given number of carbon, hydrogen and bromine atoms.

We reasoned with colored sequences. A colored sequence and its flipped counterpart were assigned to a strip. Let's look closer at the flipping. Flipping a sequence is an action on the sequence. How do these actions combine? Flipping it again brings it back to the original sequence, so flipping it twice is like doing nothing. Seems like we also need something that represents doing nothing. This points to the additive group $\mathbb{Z}_2$ with zero being the action of doing nothing and one the action of flipping. But what is an action? From what we just described, an action binds an element of the group $\mathbb{Z}_2$ with an element of the set of colored sequences (let's name this set C) and returns a new element of C. It is a function:

$$\Phi : \mathbb{Z}_2 \times C \mapsto C$$

To be consistent with group structure we want to impose restrictions on the function Φ and require that it satisfy two properties:

Firstly, the action of the neutral element of the group (in our case it is zero) should not change the color sequence:

$$\forall c \in C : \Phi(0, c) = c$$

And secondly, a sequence of actions should be consistent with the group operation:

$$\forall c \in C \text{ and } \forall g, h \in \mathbb{Z}_2 : \Phi(g + h, c) = \Phi(g, \Phi(h, c))$$

With our simple group of only two elements we get $\forall c \in C$:

$$\Phi(0, c) = c$$
$$\Phi(1, c) = \bar{c} \text{ where } \bar{c} \text{ is the reversed color sequence of } c$$

The number of ways to color a strip is a sum of two expressions:

$$\text{number of ways to color a strip} = \frac{n^k}{2} + \frac{|P|}{2}$$

The numerator in the first expression is n^k, the size of C, which is also the size of the subset of elements left unchanged by the action of the neutral group element zero (since that is the full set C according to our first restriction on Φ). The numerator in the second expression is $|P|$, the size of the subset of elements left unchanged by the action of the group element one (the flipping). This coincides with the set of color sequences that are palindromes. The denominator in both cases is two, the size of the group.

We are going to make a **bold statement** and posit that this counting holds for any group and any set: the number of object classes is the sum of the sizes of subsets that are invariant to the action of a group element, with the sum taken over all group elements and then divided by the size of the group.

But first we have to be more precise in describing what we mean by the general case and what we mean by object classes.

Defining Group Action, Orbit and Stabilizer

Given is a finite set X and a finite group $(G, \circ)$ with neutral element $e \in G$ and g^{-1} the inverse of g.

Definition 28.1. Group $(G, \circ)$ acts on set X through a function $\Phi : G \times X \mapsto X$ iff Φ satisfies

$$\forall x \in X : \Phi(e, x) = x$$
$$\forall x \in X, \forall g, h \in G : \Phi(g \circ h, x) = \Phi(g, \Phi(h, x))$$

The group action immediately implies some interesting subsets of X and G. Lets define them:

Definition 28.2. The **orbit** of $x \in X$ is a subset $O_x \subset X$ of all the group actions from G on x:

$$O_x = \{\Phi(g, x) : g \in G\}$$

Orbits are precisely the object classes we mentioned above that we want to count. In the motivating example above, the orbits are the color strips and the set X is C, the set of color sequences. So our goal is to count the number of orbits.

Definition 28.3. The **stabilizer** of $x \in X$ is a subset $S_x \subset G$ of all the group elements of G that keep x unchanged:

$$S_x = \{g \in G : \Phi(g, x) = x\}$$

In the motivating example above, for a given color sequence, the stabilizer is either the whole group $\mathbb{Z}_2$ if the sequence is a palindrome, or the stabilizer is the one element set containing just zero if the color sequence is not a palindrome.

Theorem 28.4. *The orbits of a group action partition the set X.*

Proof. We will show that group action induces an equivalence relationship $\sim$ on X. We define

$$\forall x, y \in X : x \sim y \text{ iff } y \in O_x$$

and show that it is an equivalence relationship. Since $\Phi(e, x) = x$ we have $x \in O_x$, so $\sim$ is reflexive.

Now assume $x \sim y$, which implies $y \in O_x$, so there is a $g \in G$ such that $y = \Phi(g, x)$. Then

$$\begin{aligned}\Phi(g^{-1}, y) &= \Phi(g^{-1}, \Phi(g, x)) \\ &= \Phi(g^{-1} \circ g, x) \\ &= \Phi(e, x) \\ &= x\end{aligned}$$

so $x \in O_y$ and $y \sim x$, ie the relationship is symmetric.

For transitivity, assume $x \sim y$ and $y \sim z$, so there are $g, h \in G$ such that $y = \Phi(g, x)$ and $z = \Phi(h, y)$. We have

$$\begin{aligned}z &= \Phi(h, y) \\ &= \Phi(h, \Phi(g, x)) \\ &= \Phi(h \circ g, x)\end{aligned}$$

so $z \in O_x$ and $x \sim z$.

□

Theorem 28.5. *The stabilizer S_x of $x \in X$ is a subgroup of G.*

Proof. For all $g, h \in S_x$ we have:

$$\Phi(g \circ h^{-1}, x) = \Phi(g, \Phi(h^{-1}, x))$$

but

$$\begin{aligned}\Phi(h^{-1}, x) &= \Phi(h^{-1}, \Phi(h, x)) \text{ because } h \in S_x \\ &= \Phi(h^{-1} \circ h, x) \\ &= \Phi(e, x) \\ &= x\end{aligned}$$

so

$$\Phi(g \circ h^{-1}, x) = \Phi(g, \Phi(h^{-1}, x)) = \Phi(g, x) = x$$

because g is also in S_x. It follows that $g \circ h^{-1} \in S_x$.

□

To recap the two important properties: the orbits partition X and a stabilizer is a subgroup of G.

We are now going to make a slight detour into the world of left cosets.

Left cosets

Given is a group G and a subgroup U (for notational simplicity in this subsection we will use multiplicative notation for the group operation and say 1_G is the neutral element). We define the following relationship[3] in G:

$$\forall g, h \in G : g \sim h \text{ iff } g^{-1}h \in U$$

[3] Note that even though we use the same $\sim$ symbol in this subsection, it is a different relationship from the relationship in X in the previous subsection.

We will show that this relationship is an equivalence relationship. For reflexivity, it's clear that $g^{-1}g = 1_G \in U$, so $g \sim g$. For symmetry we have $g \sim h$, so $g^{-1}h \in U$. But the inverse of an element from the subgroup is also in the subgroup, so $(g^{-1}h)^{-1} = h^{-1}g \in U$, therefore $h \sim g$. For transitivity assume $g \sim h$ and $h \sim f$, then $g^{-1}f = (g^{-1}h)(h^{-1}f)$. Both $g^{-1}h$ and $h^{-1}f$ are in U according to our assumption, so their composition is too. Therefore $g \sim f$.

Lets denote the set of equivalence classes from this relationship with G/U. Let $gU = \{gu : u \in U\}$.

For every $g \in G$ let $[g]$ be the equivalence class for which g is a representative. We prove that $gU = [g]$:

Let $h \in gU$. Then there exists an $u \in U$ such that $h = gu$, so $g^{-1}h = u$ and $g \sim h$ according to the definition of $\sim$. This means that

$gU \subseteq [g]$. Likewise let $h \in [g]$. Then $g \sim h$ and $g^{-1}h = u$ for some $u \in U$. This means that $h = gu$ and $h \in gU$, so $[g] \subseteq gU$.

For every $g \in G$ the function $f_g : U \mapsto gU$ with $f(u) = gu$ is a bijection[4]. This means that all equivalence classes have the same size, namely $|U|$ and we have:

[4] Easy to see using the group axioms.

$$|G| = |U||^{G}/_{U}|$$

This concludes our small detour[5]. Let's go back to the group actions and use what we just established.

[5] There is a lot more to explore about cosets. I did say that group theory is rich in structure. We only pulled in what was absolutely needed to continue.

Burnside's Lemma

We already know that a stabilizer is a subgroup of G. We can now use

$$|G| = |S_x||^{G}/_{S_x}|$$

But what are the elements of $^{G}/_{S_x}$? We know from the detour subsection that the equivalence classes have the form gS_x for some $g \in G$. Now assume $g \notin S_x$. That means $\Phi(g, x) = y$ for some $y \neq x$ in X. But that y belongs in the orbit of x. There are $|O_x|$ such distinct y and therefore $|O_x|$ equivalence classes. We have just proved the **orbit-stabilizer theorem**:

Theorem 28.6. *For every $x \in X$ we have $|G| = |S_x||O_x|$.*

With the help of this theorem we can count the number of orbits. We remember that the orbits partition X. In each orbit O_x we can assign $\frac{1}{|O_x|}$ to each member of the orbit. Summing up these assigned fractions results in the value one for each orbit. So summing up the fractions over all the elements of X counts the number of orbits as seen in figure 28.4. The number of orbits becomes:

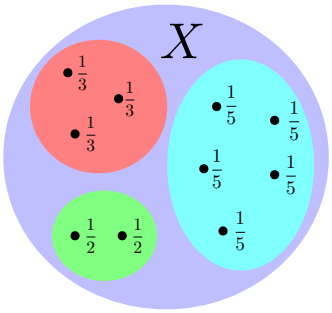


Figure 28.4: The set X with three orbits. Each element in an orbit is assigned the fraction one over the size of the orbit.

$$\#orbits = \sum_{x \in X} \frac{1}{|O_x|}$$

From the orbit-stabilizer theorem we know that $\frac{1}{|O_x|} = \frac{|S_x|}{|G|}$, so:

$$\#orbits = \frac{1}{|G|} \sum_{x \in X} |S_x|$$

This is already pretty good but has the disadvantage that we sum over the elements of X, which can be large. We want to instead sum over the elements of G. Let's look at the definition of S_x again:

$$S_x = \{g \in G : \Phi(g, x) = x\}$$

Evaluating the size of S_x is equivalent to counting all pairs $(g, x) \in G \times X$ where $\Phi(g, x) = x$ for one particular x and all g. Summing up

all these sizes is equivalent to doing it for all x and all g. We can use an indicator function to express this: $1_\Phi : G \times X \mapsto \{0,1\}$ with

$$1_\Phi(g,x) = \begin{cases} 1, \text{ when } \Phi(g,x) = x \\ 0, \text{ when } \Phi(g,x) \neq x \end{cases}$$

Then $|S_x| = \sum_{g \in G} 1_\Phi(g,x)$ and

$$\#orbits = \frac{1}{|G|} \sum_{x \in X} \sum_{g \in G} 1_\Phi(g,x)$$

We can now invert the double sum order and write

$$\#orbits = \frac{1}{|G|} \sum_{g \in G} \sum_{x \in X} 1_\Phi(g,x)$$

Collecting all the indicator values for one fixed g is the same as evaluating the size of a subset of X of elements that g keeps unchanged. Lets denote sets like this **fixsets**:

$$Fix(g) = \{x \in X : \Phi(g,x) = x\}$$

so $|Fix(g)| = \sum_{x \in X} 1_\Phi(g,x)$.
Using this we arrive at the **Burnside Lemma**:

$$\#orbits = \frac{1}{|G|} \sum_{g \in G} |Fix(g)|$$

Applications of the Burnside Lemma

As a first application of our lemma, lets revisit the motivating example of color strips and verify that we get the same answer. According to the lemma we have

$$\text{number of ways to color a strip} = \frac{1}{|\mathbb{Z}_2|}(|Fix(0)| + |Fix(1)|)$$

Clearly $Fix(0)$ are all the color sequences, so $|Fix(0)| = n^k$ and $Fix(1)$ are the color sequences left unchanged by flipping, ie palindromes, so $|Fix(1)| = n^{\lceil \frac{k}{2} \rceil}$. Our answer checks out.

The second example is a square tablecloth of five by five cells to be colored with four colors. Here rotations by 90°, 180° and 270° are considered the same tablecloth as seen in figure 28.5. This is very similar to our first example, but here the group acting on the set of color sequences is the cyclic group $\mathbb{Z}_4$. In how many ways can we color the tablecloth? To apply Burnside we need the sizes of four fixsets. We again have $|Fix(0)| = 4^{25}$, ie all color sequences are unchanged by not rotating.

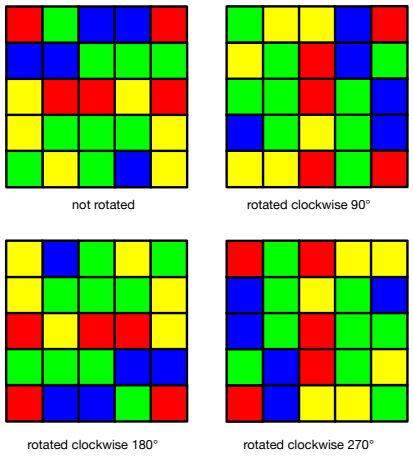


Figure 28.5: These four configurations are considered to be the same tablecloth.

$Fix(1)$ is the set of color sequences unchanged by rotating by 90°. If we divide the cloth into four quadrants (assigning shared boundary cells like in figure 28.6), it's clear that a 90°rotation forces the contents of each quadrant to move to the next quadrant. So if the color sequence is supposed to stay unchanged then all four quadrants must have the same contents. The exception is the cell in the middle of the tablecloth. It is unchanged by any rotation. Therefore $|Fix(1)| = 4 \times 4^6$.

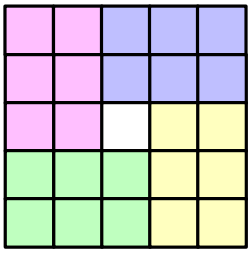


Figure 28.6: The four quadrants of the tablecloth.

$Fix(2)$ is the set of color sequences unchanged by rotating by 180°. The quadrants across from each other end up exchanging contents which means those contents have to be equal for the color sequence to by unchanged by the rotation. Therefore $|Fix(2)| = 4 \times 4^{12}$.

And finally $Fix(3)$ is the set of color sequences unchanged by rotating by 270°. The quadrants exchange contents with the quadrant before them (in clockwise order). Therefore $|Fix(3)| = 4 \times 4^6$.

$$\begin{aligned}\text{number of ways to color tablecloth} &= \frac{1}{4}(4 \times 4^6 + 4 \times 4^{12} + 4 \times 4^6) \\ &= 2 \times 4^6 + 4^{12} \\ &= 16785408\end{aligned}$$

The Burnside Lemma is very cool but it doesn't offer any help determining the sizes of the fixsets. For this we will use **Pólya's method of counting**, which we will explore in the next subsection.

Pólya's method of counting

We will change the setting slightly. The set X is still the set of color sequences of length k, but now we make explicit that these sequences are actually mappings of sequence position to a color from a set C of colors, so $x \in X$ is a function $x : \{1, 2, \ldots, k\} \mapsto C$. And the group G will now by a subgroup of the symmetric group S_k of permutations of length k. We define the group action to be the following function composition:

$$\forall \pi \in S_k, \forall x \in X : \Phi(\pi, x) = x \circ \pi^{-1}$$

It's easy to check that $x \circ \pi^{-1} \in X$ and the identity permutation doesn't change x. For the second group action restriction we consider two permutations $\pi, \tau \in S_k$ and check:

$$\begin{aligned}\Phi(\pi \circ \tau, x) &= x \circ (\pi \circ \tau)^{-1} \\ &= x \circ \tau^{-1} \circ \pi^{-1} \\ &= \Phi(x, \tau) \circ \pi^{-1} \\ &= \Phi(\pi, \Phi(x, \tau))\end{aligned}$$

Informally the group action of π on x assigns the color $x(i)$ to position $\pi(i)$ in the new color sequence $\Phi(\pi, x)$. That is because

$$\Phi(\pi, x)(\pi(i)) = (x \circ \pi^{-1})(\pi(i)) = x(i)$$

Permutations simplify computing the fixsets needed to apply Burnside's lemma. The reason for this is the decomposition of a permutation into disjoint cycles. For a color sequence x to be a member of the fixset $Fix(\pi)$ of permutation π, it needs to assign the same color to every position from a cycle of π. Therefore the cycle structure of a permutation directly determines the size of its fixset.

Lets use this fact on an example. Consider a necklace with six beads, each bead can be any of seven colors. How many necklaces can we make? Since the beads of a necklace are on a closed loop string, it makes sense to model it like this: we use color sequences of length six from seven colors. The first bead of a necklace can move from first to last and all other beads shift over one position. We want to find a group G for which the orbit of a color sequence is considered the same necklace. The group generated by the full cycle $G =< (1,2,3,4,5,6) >$ achieves this (because it keeps rotating beads on the closed loop string as seen in figure 28.7). We use Mathematica to list out the permutations of this group in cycle notation:

```
In[1]:= GroupElements[
  PermutationGroup[{Cycles[{{1, 2, 3, 4, 5, 6}}]}]]

Out[1]= {Cycles[{}],
         Cycles[{{1, 2, 3, 4, 5, 6}}],
         Cycles[{{1, 3, 5}, {2, 4, 6}}],
         Cycles[{{1, 4}, {2, 5}, {3, 6}}],
         Cycles[{{1, 5, 3}, {2, 6, 4}}],
         Cycles[{{1, 6, 5, 4, 3, 2}}]}
```

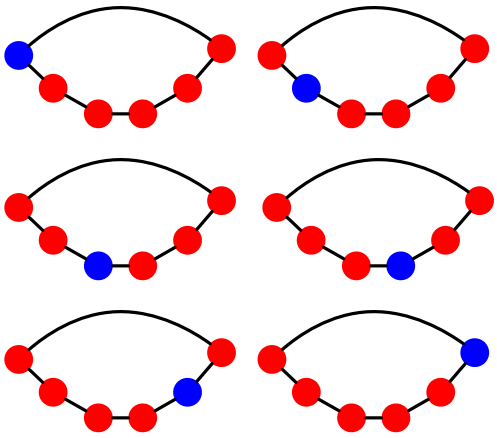


Figure 28.7: These six color sequences are considered the same necklace. By repeatedly applying the full cycle we move the blue bead around.

The fixset sizes are (in cycle notation):

$$\begin{aligned} |Fix((1)(2)(3)(4)(5)(6))| &= 7^6 \\ |Fix((1,2,3,4,5,6))| &= 7 \\ |Fix((1,3,5)(2,4,6))| &= 7^2 \\ |Fix((1,4)(2,5)(3,6))| &= 7^3 \\ |Fix((1,5,3)(2,6,4))| &= 7^2 \\ |Fix((1,6,5,4,3,2))| &= 7 \end{aligned}$$

Using Burnside we get:

$$\text{number of necklaces} = \frac{1}{6}(7^6 + 7 + 7^2 + 7^3 + 7^2 + 7) = 19684$$

As always in mathematics, there is way more to say about this subject (one can for example impose restrictions on the colorings). But I will stop here for this note.

29
Two algebraic delights

Algebraic representations are the topic of this note. Transforming a structure into a corresponding algebraic representation enables easier reasoning and unlocks simpler proofs.

We are going to explore two examples of transformations that I call little algebraic delights. They allow reasoning with ordinary algebraic operations on mathematical objects that are not algebraic at first glance. The first example will use indicator functions[1] to prove set identities and the second example will use formal languages[2] to count combinatorial objects.

[1] D. Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2002. ISBN 9780521002899. URL https://books.google.com/books?id=B7Ch-c2G21MC

[2] Ö. Eğecioğlu and A.M. Garsia. *Lessons in Enumerative Combinatorics*. Graduate Texts in Mathematics. Springer International Publishing, 2021. ISBN 9783030712501. URL https://books.google.com/books?id=5BMuEAAAQBAJ

Indicator functions of sets

When you deal with sets, you usually have to do Boolean algebra. Proving identities of expressions of set operations can become really tedious. Let's say we want to prove that the symmetric difference is associative, so given three sets A, B, C we have

$$(A \triangle B) \triangle C = A \triangle (B \triangle C)$$

where the symmetric difference is defined as

$$A \triangle B = (A \setminus B) \cup (B \setminus A)$$

The usual approach of proving is to show that the set on the left hand side $(A \triangle B) \triangle C$ is a subset of the right hand side $A \triangle (B \triangle C)$ and vice versa, by tediously following an $x \in (A \triangle B) \triangle C$ and showing that it is also in $A \triangle (B \triangle C)$ and then the reverse.

Instead of that approach let's try something different. Let $\mathcal{U} = A \cup B \cup C$ be the union of all the sets participating in the identity we want to prove (our universe). We define an indicator function $\mathbb{I}_S : \mathcal{U} \to \{0, 1\}$ for a subset $S \subseteq \mathcal{U}$ of this universe as:

$$\mathbb{I}_S(x) = \begin{cases} 1 & : x \in S \\ 0 & : x \notin S \end{cases}$$

We can combine indicator functions with arithmetic operations in a point-wise manner in the field $\mathbb{Z}_2$. It's also clear that sets are in a one-to-one correspondence with their indicator function.

Let's look at the indicator functions of some set operations[3]:

$$\begin{aligned} &\forall x \in \mathcal{U} : \\ &\mathbb{I}_{S\cup T}(x) = max(\mathbb{I}_S(x), \mathbb{I}_T(x)) \\ &\mathbb{I}_{S\cap T}(x) = \mathbb{I}_S(x) \cdot \mathbb{I}_T(x) \\ &\mathbb{I}_{S^c}(x) = 1 - \mathbb{I}_S(x) \\ &\mathbb{I}_{S\setminus T}(x) = \mathbb{I}_S(x) \cdot (1 - \mathbb{I}_T(x)) \\ &\mathbb{I}_{S\triangle T}(x) = \mathbb{I}_S(x) + \mathbb{I}_T(x) = \mathbb{I}_S(x) - \mathbb{I}_T(x) \end{aligned}$$

[3] These identities are easy to prove. Just remember, the operations are modulo two and are point-wise, so have to hold for every x in the universe. For example, to prove the last identity $\mathbb{I}_{S\triangle T}(x) = \mathbb{I}_S(x) + \mathbb{I}_T(x)$ we can observe that the symmetric difference shouldn't include the intersection of the two sets, ie when both indicator functions are equal to one. The sum of $1 + 1$ is zero modulo two so that works out, etc... Also remember $-1 = 1$ in $\mathbb{Z}_2$.

Let's omit the x in these point-wise expressions:

$$\begin{aligned} \mathbb{I}_{S\cup T} &= max(\mathbb{I}_S, \mathbb{I}_T) \\ \mathbb{I}_{S\cap T} &= \mathbb{I}_S \cdot \mathbb{I}_T \\ \mathbb{I}_{S^c} &= 1 - \mathbb{I}_S \\ \mathbb{I}_{S\setminus T} &= \mathbb{I}_S \cdot (1 - \mathbb{I}_T) \\ \mathbb{I}_{S\triangle T} &= \mathbb{I}_S + \mathbb{I}_T = \mathbb{I}_S - \mathbb{I}_T \end{aligned}$$

Since we agreed that we will work with the indicator functions of the sets, we could just drop the $\mathbb{I}$ from the notation[4]:

$$\begin{aligned} S \cup T &= max(S, T) \\ S \cap T &= S \cdot T \\ S^c &= 1 - S \\ S \setminus T &= S \cdot (1 - T) \\ S\triangle T &= S + T = S - T \end{aligned}$$

[4] To parse expressions where we dropped the symbol $\mathbb{I}$, we have to group set operations and imagine an $\mathbb{I}$ in front of them, ie set operations have grouping precedent over arithmetic operations. For example $S \cup T \cdot V$ means the point-wise multiplication $\mathbb{I}_{S\cup T} \cdot \mathbb{I}_V$.

Given the indicator function equivalent of the symmetric difference, our initial $(A\triangle B)\triangle C = A\triangle(B\triangle C)$ becomes the almost trivial $(A + B) + C = A + (B + C)$.

The expression for the union has *max* which is sometimes convenient but sometimes gets in the way of point-wise arithmetic. But we can get rid of it by observing that the union is the symmetric difference plus the intersection:

$$S \cup T = S \triangle T + S \cdot T = S + T + S \cdot T$$

Let's deploy our new-found powers to something more complicated and try to prove that

$$(\bigcap_{i=1}^{n} A_i) \triangle (\bigcap_{i=1}^{n} B_i) \subseteq \bigcup_{i=1}^{n} (A_i \triangle B_i)$$

Two things before we start: our universe expanded to $\mathcal{U} = (\bigcup_{i=1}^{n} A_i) \cup (\bigcup_{i=1}^{n} B_i)$ and we have for subsets $S, T \subseteq \mathcal{U}$:

$$S \subseteq T \Leftrightarrow \forall x \in \mathcal{U} : \mathbb{I}_S(x) \leq \mathbb{I}_T(x)$$

Given that inequality and the fact that the range of indicator functions is $\{0, 1\}$, when the right-hand side is one then the inequality is trivially true. The only interesting case is when the right-hand side is zero. We use the *max* expression for union and have

$$\bigcup_{i=1}^{n} (A_i \triangle B_i) = max_{i=1}^{n} (A_i - B_i)$$

This max can only be zero iff all $A_i = B_i$. But in that case we also have the left-hand side zero because the left-hand side is

$$(\bigcap_{i=1}^{n} A_i) \triangle (\bigcap_{i=1}^{n} B_i) = \prod_{i=1}^{n} A_i - \prod_{i=1}^{n} B_i$$

which concludes our proof.

Formal Languages to count combinatorial objects

We will do a very quick, (ahem) informal introduction to Formal Languages[5]. We start with an alphabet $\mathcal{A}$ which is an ordered set of symbols. We want it ordered so that we can do lexicographic ordering of words from that alphabet. Speaking of words: they are sequences of symbols from the alphabet. Concatenation of two words w_1 and w_2 is denoted by $w_1 \cdot w_2$ and defined as you would expect. The empty word ϵ has length zero and is the neutral element of concatenation. A word w_2 is a prefix of a word w_1 iff there is a word w_3 such that $w_1 = w_2 \cdot w_3$. The set of all words (including the empty word ϵ) from alphabet $\mathcal{A}$ is denoted $\mathcal{A}^*$ and the set of all words excluding ϵ is $\mathcal{A}^+$. A subset $\mathcal{L} \subseteq \mathcal{A}^*$ is called a language.

[5] This should be very familiar for all of us computer science majors.

We have a couple of ways to form new languages from given ones. One way is concatenation. Given $\mathcal{L}_1, \mathcal{L}_2$:

$$\mathcal{L}_1 \cdot \mathcal{L}_2 = \{w_1 \cdot w_2 : w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2\}$$

The other way is the set union $\mathcal{L}_1 \cup \mathcal{L}_2$ which becomes more interesting when $\mathcal{L}_1$ and $\mathcal{L}_2$ are disjoint.

So far so good. Now comes the cool stuff. Given a language $\mathcal{L}$ we define its listing series as:

$$s\mathcal{L} = \sum_{w \in \mathcal{L}} w$$

Some notes on this notation: This is a formal sum and should not be thought of as the normal addition of numbers. The name *listing series* hints at its special nature: it lists out the words of a language in a chain. Using the plus symbol as the chain separator (as opposed to say the comma) might be confusing in the beginning but it will pay off later when it is combined with the multiplication symbol used for concatenation. The words in the sum are usually listed out in lexicographic order. The usefulness of the listing series will become apparent when we relate it to concatenation. Let's look at an example: $\mathcal{L}_1 = \{a, b\}$:

$$s\mathcal{L} = aa + ab + ba + bb$$

We take a second language $\mathcal{L}_2 = \{c\}$ and now list out the concatenation (the listing operator s has lower precedent than the concatenation operator, saving us round braces):

$$\begin{aligned} s\mathcal{L}_1 \cdot \mathcal{L}_2 &= aac + abc + bac + bbc \\ &= (aa + ab + ba + bb) \cdot c \\ &= (s\mathcal{L}_1) \cdot (s\mathcal{L}_2) \end{aligned}$$

Treating $+$ and $\cdot$ in a strictly symbolic, algebraic way, it looks like $\cdot$ distributes over $+$ and preserves the correct meaning of listing series of the languages involved in the expression. Note that unlike with the multiplication of numbers, concatenation is not commutative, so ab and ba are different and $aa + ab + ba + bb$ is not $aa + 2ab + bb$ (that 2 in the last expression doesn't even make sense). We have to be careful not to cross the line and conflate the operations with the familiar numeric ones, but if we are careful we can now manipulate languages algebraically as if they were finite sums of terms (or even infinite sums as we will see).

Before we can put this to good use, let us also introduce another notational convenience: exponentiation. We have seen that ab and ba are not the same, also aab and baa are different. But as a convenience we can abbreviate aa to a^2 and in general a word $aa \ldots a$ of length n formed with one single symbol a as a^n. That way aab and baa can be written a^2b and ba^2 respectively.

Exponentiation can be expanded to languages: $\mathcal{L}^n$ is the language formed by concatenating $\mathcal{L}$ with itself n times. We also agree that $\mathcal{L}^0 = \{\epsilon\}$.

One last thing before we start: what should the placeholder symbol for the empty word in a listing series be? Well, since the empty word is the neutral element of concatenation and we use "multiplication" as our concatenation operator, it is befitting to use 1 for the empty word and this also fits with distribution of concatenation over listing series and our "exponentiation". For a neat example: consider $\mathcal{A} = \{a\}$ and list out $\mathcal{A}^*$(purely symbolic[6]):

$$s\mathcal{A}^* = 1 + a + a^2 + a^3 + \ldots = \sum_{i=0}^{\infty} a^i = \frac{1}{1-a}$$

[6] With a wink towards calculus, we agree that $\frac{1}{1-a}$ symbolizes the listing series of $\mathcal{A}^*$.

We're ready to do some interesting combinatorics. Until now we used the subscript on a language symbol like $\mathcal{L}_1$ just to make it an individual and distinguish it from another language $\mathcal{L}_2$ in this exposition. From now on we will give it meaning: given a fixed alphabet we say $\mathcal{L}_n$ is the language of all the words with length n from that alphabet.

It's not that hard to prove that when $n = p + q$ with $n, p, q \in \mathbb{N}$ then (over the same alphabet):

$$\begin{aligned} \mathcal{L}_n &= \mathcal{L}_p \cdot \mathcal{L}_q \\ s\mathcal{L}_n &= (s\mathcal{L}_p) \cdot (s\mathcal{L}_q) \end{aligned}$$

This is really powerful because it is a rich source of recursions for both the listing series generation and for counting the number of words in the language.

Consider the alphabet of two symbols: $\mathcal{A} = \{a, b\}$. Let's list out some languages of different lengths from this alphabet:

$$\begin{aligned} s\mathcal{L}_0 &= 1 \\ s\mathcal{L}_1 &= a + b \\ s\mathcal{L}_2 &= aa + ab + ba + bb \\ s\mathcal{L}_3 &= aaa + aab + aba + baa + abb + bab + bba + bbb \\ &\ldots \end{aligned}$$

We observe that $s\mathcal{L}_n = s(a \cdot \mathcal{L}_{n-1}) + s(b \cdot \mathcal{L}_{n-1})$ and if $\mathcal{L}_{n-1}$ is listed in lexicographic order, then this recursion even preserves the lexicographic order. In essence, it gives us an algorithm to generate words of a given length in lexicographic order[7].

[7] It also lets us count the number of words in $\mathcal{L}_n$: if $b_n = |\mathcal{L}_n|$ is the number of words in $\mathcal{L}_n$, then it satisfies the recursion: $b_n = 2b_{n-1}$ and we know $b_0 = 1$. So no surprise here: $b_n = 2^n$.

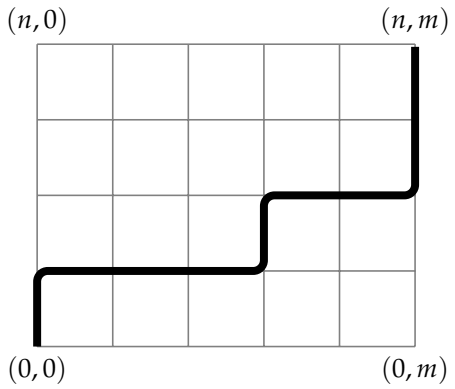


Figure 29.1: Lower left corner of grid is $(0,0)$ and upper right corner is (n,m). Paths are going either up North or right East, always along an edge in the grid.

Let us expand the language subscript notation with even more meaning. Same two-symbol alphabet $\mathcal{A} = \{a, b\}$, and now $\mathcal{L}_{n,k}$ means language of words of length n with exactly k b's in them. Then[8]:

$$s\mathcal{L}_{n,k} = (s\ a \cdot \mathcal{L}_{n-1,k}) \cdot (s\ b \cdot \mathcal{L}_{n-1,k-1})$$

If we denote the size of language $\mathcal{L}_{n,k}$ with $C_{n,k} = |\mathcal{L}_{n,k}|$ then we have the recursion:

$$C_{n,k} = C_{n-1,k} + C_{n-1,k-1}$$

and the astute reader recognizes this as the recursion of the binomial coefficients and the Pascal triangle.

If we find a one-to-one correspondence between the combinatorial objects that we want to reason about and words in a language, then we can deploy this formal language machinery to generate the objects (using listing series) and also count them.

As an example of such a correspondence, consider a $n \times m$ lattice grid and East/North lattice paths on that grid.

How many such paths are there that go from $(0,0)$ to (n,m) ? We can make a one-to-one correspondence from the set of possible paths to a language with the two-symbol alphabet $\{E, N\}$ (E for East, N for North). Each path has to have $n+m$ segments with n North-going segments and m East-going segments. The corresponding language $\mathcal{L}$ over the alphabet $\mathcal{A} = \{E, N\}$ consists of words of length $n+m$ with exactly n N's (and therefore m E's). We recognize again the binomial coefficient and have the number of paths

$$|\mathcal{L}| = C_{n+m,n}$$

This was a simple example but the general idea stays the same: find a bijection between the combinatorial objects and words in a language over some alphabet and then switch to algebraic series manipulation

[8] If a word of length n with exactly k b's starts with an a then it must be followed by a word of length $n-1$ with k b's. If on the other hand it starts with a b then it must be followed by a word of length $n-1$ with $k-1$ b's.

of the words. For example, if it is a 3-dimensional lattice grid, then we expand the alphabet to three symbols and proceed. Sometimes the language used in the correspondence has very interesting restrictions such as in a two-symbol alphabet language where the number of a's and b's has to be equal and any prefix of the word has to have at least as many a's as b's[9].

[9] Words from such a language are called Dyck words. These words have correspondence to many combinatorial objects.

30
Sequences and Series

Select exercises on sequences and series from Chapter 3 of the *Lectures on Real Analysis* textbook[1].

[1] F. Lárusson. *Lectures on Real Analysis*. Australian Mathematical Society Lecture Series. Cambridge University Press, 2012. ISBN 9781107026780. URL https://books.google.com/books?id=koj-IrXXwocC

Exercise 3.17, page 35

(a) Let $a \geq 0$ and $n \in \mathbb{N}$, $n \geq 2$. Show that

$$(1+a)^n \geq \frac{1}{2}n(n-1)a^2$$

(b) Show that $n^{\frac{1}{n}} \to 1$ as $n \to \infty$.

Solution. (a) Using the binomial expansion, we get

$$(1+a)^n = \sum_{k=0}^{n} \binom{n}{k} a^k = 1 + na + \frac{1}{2}n(n-1)a^2 + \ldots \geq \frac{1}{2}n(n-1)a^2$$

(b) Using the inequality from (a) with $a = n^{\frac{1}{n}} - 1$ we get

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

So $\frac{2}{n-1} \geq (n^{\frac{1}{n}} - 1)$ and $n^{\frac{1}{n}} \to 1$. □

Exercise 3.18, page 35

Consider the recursively defined sequence (a_n) with $a_1 = 3$ and $a_{n+1} = \frac{a_n}{2} + \frac{3}{a_n}$. Show that (a_n) converges and find its limit.

Solution. Let's first prove by induction that $\forall n \in \mathbb{N} : 2 < a_n \leq 3$:

It's true for $a_1 = 3$. Assume it is true for a given n and let's do the induction step.

$$a_{n+1} = \frac{a_n}{2} + \frac{3}{a_n} > \frac{2}{2} + \frac{3}{3} = 2$$

Also

$$a_{n+1} = \frac{a_n}{2} + \frac{3}{a_n} \leq \frac{3}{2} + \frac{3}{2} = 3$$

At least we know (a_n) is bounded. Let us spy a little and assume (a_n) does converge, say to limit L. Then L must satisfy:

$$L = \frac{L}{2} + \frac{3}{L}$$

which works out to $L = \sqrt{6}$.

Let's try with a simpler sequence (b_n) such that $a_n = b_n\sqrt{6}$.

$$\begin{aligned} a_{n+1} = b_{n+1}\sqrt{6} &= \frac{a_n}{2} + \frac{3}{a_n} \\ &= \frac{b_n\sqrt{6}}{2} + \frac{3}{b_n\sqrt{6}} \\ &= \frac{b_n\sqrt{6}}{2} + \frac{\sqrt{6}}{2b_n} \end{aligned}$$

So (b_n) satisfies $b_{n+1} = \frac{1}{2}(b_n + \frac{1}{b_n})$. We prove that (b_n) is monoton decreasing:

$$\begin{aligned} b_{n+1} \leq b_n &\Leftrightarrow \\ \frac{1}{2}(b_n + \frac{1}{n}) \leq b_n &\Leftrightarrow \\ b_n^2 + 1 \leq 2b_n^2 &\Leftrightarrow \\ b_n^2 \geq 1 &\Leftrightarrow \\ b_n \geq 1 & \end{aligned}$$

We use the AGM inequality[2] and show:

$$b_{n+1} = \frac{1}{2}(b_n + \frac{1}{b_n}) \geq \sqrt{b_n \frac{1}{b_n}} = 1$$

So (b_n) is monoton decreasing and bounded below by 1, so (b_n) converges, and so does (a_n): $b_n \to 1$ and $a_n \to \sqrt{6}$.

□

[2] For positive x and y we have $(\sqrt{x} + \sqrt{y})^2 \geq 0$ which when expanded ends up at $\frac{x+y}{2} \geq \sqrt{xy}$.

Exercise 3.23, page 36

Let $\sum a_n$ be a series. Set $a_n^+ = max\{0, a_n\}$ and $a_n^- = min\{0, a_n\}$. Consider the series $\sum a_n^+$ and $\sum a_n^-$.
(a) Prove that $\sum a_n$ is absolutely convergent if and only if $\sum a_n^+$ and $\sum a_n^-$ both converge. Then $\sum a_n = \sum a_n^+ + \sum a_n^-$.
(b) Prove that if $\sum a_n$ is conditionally convergent, then $\sum a_n^+$ and $\sum a_n^-$ both diverge.

Solution. We will use the partial sums:

$$s_n = \sum_{k=1}^{n} a_k, \quad s_n^a = \sum_{k=1}^{n} |a_k|$$
$$s_n^+ = \sum_{k=1}^{n} a_k^+, \quad s_n^- = \sum_{k=1}^{n} a_k^-$$

(a) ($\Rightarrow$)

We have $\forall n \in \mathbb{N} : |a_n| \geq a_n^+$ and $|a_n| \geq (-1)a_n^-$. Using the comparison test we find $\sum a_n^+$ and $\sum a_n^-$ converge.

($\Leftarrow$) $\sum a_n^+$ and $\sum a_n^-$ converge, so then also $\sum a_n^+ + (-1)\sum a_n^-$ converges. But $s_n^a = s_n^+ + (-1)s_n^-$, so $\sum |a_n|$ converges too.

(b) $\sum a_n$ converges conditionally. If both $\sum a_n^+$ and $\sum a_n^-$ converge, then from (a) we would have $\sum a_n$ converges absolutely, contradicting the premise. So at least one of $\sum a_n^+$ or $\sum a_n^-$ must diverge.

Assume $\sum a_n^+$ diverges (the other case is similar). s_n^+ is monotonically increasing and divergent, so it is unbounded. We have $s_n^+ = s_n - s_n^-$ and s_n is bounded. It follows that s_n^- has to be unbounded, so $\sum a_n^-$ diverges also.

□

Exercise 3.24, page 36

Let $\sum a_n$ be a conditionally convergent series. Prove that for every $\sigma \in \mathbb{R}$ there is a rearrangement of $\sum a_n$ that converges to σ.

Solution. We will construct this rearrangement.

We know from the previous exercise that both $\sum a_n^+$ and $\sum a_n^-$ diverge and both s_n^+ and s_n^- are unbounded.

Assume first that $\sigma > 0$ (the other case is similar). Since s_n^+ is unbounded, there exists[3] a $N_1 \in \mathbb{N}$ such that

[3] This N_1 has to exist because s_n^+ is unbounded. If it was only zeros it would converge and be bounded.

$$\sum_{k=1}^{N_1-1} a_k^+ \leq \sigma$$
$$\sum_{k=1}^{N_1} a_k^+ > \sigma$$

Let $d_1 = |\sum_{k=1}^{N_1} a_k^+ - \sigma|$. We see that $0 < d_1 \leq |a_{N_1}^+|$. Our rearrangement will start with the first N_1 terms from $\sum a_n^+$. For the next terms we turn to $\sum a_n^-$. s_n^- is also unbounded, so there exists a $M_1 \in \mathbb{N}$ such that

$$\sum_{k=1}^{M_1-1} a_k^- \geq d_1$$
$$\sum_{k=1}^{M_1} a_k^- < d_1$$

We add the first M_1 terms from $\sum a_n^-$ to the rearrangement. Let $d_2 = |\sum_{k=1}^{N_1} a_k^+ + \sum_{k=1}^{M_1} a_k^- - \sigma|$. We see that $0 < d_2 \leq |a_{M_1}^-|$.

Next we go back to $\sum a_n^+$ for more terms. The tail of $\sum a_n^+$ starting at $N_1 + 1$ is also unbounded, so there must exist a N_2 such that

$$\sum_{k=N_1+1}^{N_2-1} a_k^+ \leq d_2$$
$$\sum_{k=N_1+1}^{N_2} a_k^+ > d_2$$

We add the terms $\sum_{k=N_1+1}^{N_2} a_k^+$ to the rearrangement and define

$$d_3 = |\sum_{k=1}^{N_1} a_k^+ + \sum_{k=1}^{M_1} a_k^- + \sum_{k=N_1+1}^{N_2} a_k^+ - \sigma|$$

We see that $0 < d_3 \leq |a_{N_2}^+|$.

We go back down with the help of terms from the tail of $\sum a_n^-$ starting at M_1, a tail that is also unbounded. There must exist a M_2 such that

$$\sum_{k=M_1+1}^{M_2-1} a_k^+ \geq d_3$$
$$\sum_{k=M_1+1}^{M_2} a_k^+ < d_3$$

We add the terms $\sum_{k=M_1+1}^{M_2} a_k^-$ to the rearrangement and define

$$d_4 = |\sum_{k=1}^{N_1} a_k^+ + \sum_{k=1}^{M_1} a_k^- + \sum_{k=N_1+1}^{N_2} a_k^+ + \sum_{k=M_1+1}^{M_2} a_k^- - \sigma|$$

We see that $0 < d_4 \leq |a_{M_2}^-|$.

We continue in this way, switching between terms in $\sum a_n^+$ and $\sum a_n^-$, constructing a rearrangement of $\sum a_n$ that has partial sums that have distance d_n from σ.

The sequence (d_n) of distances is bounded by $(|a_n|)$ and $\sum a_n$ is a conditionally convergent series, so $a_n \to 0$. That means that $d_n \to 0$ and the rearrangement converges to σ.

□

Exercise 3.30, page 37

Show that there is a sequence (a_n) such that for every real number x, there is a subsequence of (a_n) converging to x.

Solution. At first glance this seems quite a fantastical premise. How can there be a sequence that for every real number contains a subsequence converging to that number? Isn't $\mathbb{R}$ uncountable? Well, the best way to prove the existence of such a sequence is to construct it.

First we want to make our life easier: we use the fact that there exists a bijection between the interval $(0,1)$ and $\mathbb{R}$. There are many bijections between these two sets to choose from and we will choose a continuous one:

$$F : \mathbb{R} \to (0,1)$$
$$F(x) = \frac{1}{1+e^x}$$

and its inverse

$$F^{-1} : (0,1) \to \mathbb{R}$$
$$F^{-1}(x) = \ln(\frac{1-x}{x})$$

If we can construct subsequences that converge to $x \in (0,1)$ then we can use F^{-1} to map them over to $y \in \mathbb{R}$ and because of continuity the mapping of the subsequence will converge to y. The construction idea is to map a given $n \in \mathbb{N}$ to a pair $(i,j) \in \mathbb{N} \times \mathbb{N}$. This (i,j) pair will have the following meaning: j subdivides $(0,1)$ into j subintervals of length $\frac{1}{j}$ and i will select which of those j subintervals we mean. A given $x \in (0,1)$ will fall into one of them and its corresponding (i,j) pair will determine the n we use in the subsequence. Increasing the j and then choosing the corresponding i subinterval containing x will get us closer and closer to x.

This is the construction idea. We still have to deal with the technicalities.

First we want a bijection from $\mathbb{N}$ to a subset of $\mathbb{N} \times \mathbb{N}$ where the pairs (i,j) satisfy $i \leq j$. We use a similar approach to the one we used in a previous note: https://sagenhaft.space/posts/math_notes/counting/counting.pdf.

We order the pairs $(i,j) \in \mathbb{N} \times \mathbb{N}$ satisfying $i \leq j$ in rows, such that row r has pairs $(1,r),(2,r),\ldots,(r,r)$. Figure 30.1 illustrates the idea. Our bijection will count going down the rows and going left to right in each row. So the order is $(1,1),(1,2),(2,2),(1,3),(2,3),(3,3),\ldots$

$(1,1)$ ⟶ row 1

$(1,2),(2,2)$ ⟶ row 2

$(1,3),(2,3),(3,3)$ ⟶ row 3

$(1,4),(2,4),(3,4),(4,4)$ ⟶ row 4

$\vdots$

Figure 30.1: Going from n to (i,j) with $i \leq j$.

Lets first deduce the inverse, going from (i, j) to n in that order. For a given (i, j) we know we are in row j at pair i in that row. Each row k before row j has k pairs in it, therefore the corresponding position n in the counting order is:

$$\begin{aligned} n &= \sum_{k=1}^{j-1} k + i \\ &= \frac{j(j-1)}{2} + i \end{aligned}$$

We can test this: in the Figure 30.1, pair $(2, 4)$ should be the eighth pair. $\frac{4\times 3}{2} + 2 = 8$, so it checks out. We denote $M = \{(i, j) \in \mathbb{N} \times \mathbb{N} : i \leq j\}$ and define the function f:

$$\begin{aligned} f &: M \to \mathbb{N} \\ f(i, j) &= \frac{j(j-1)}{2} + i \end{aligned}$$

It is easy to prove that f is a bijection. Suppose we have two pairs $(i_1, j_1) \neq (i_2, j_2)$. If $j_1 \neq j_2$ then they are in different rows. If $j_1 = j_2$ then we must have $i_1 \neq i_2$, so again their mapping is different. It follows that f is injective.

Given $n \in \mathbb{N}$, can we find (i, j) such that $f(i, j) = n$? The nth pair falls on some row r. There are $\frac{r(r-1)}{2}$ pairs in the rows before row r and $\frac{r(r+1)}{2}$ pairs in the first r rows. Therefore:

$$\frac{r(r-1)}{2} < n \leq \frac{r(r+1)}{2}$$

The two relevant values for these two quadratic inequalities are $\frac{1+\sqrt{1+8n}}{2}$ and $\frac{-1+\sqrt{1+8n}}{2}$ because we have to stay positive. Notice that their difference is $\frac{1+\sqrt{1+8n}}{2} - \frac{-1+\sqrt{1+8n}}{2} = 1$, so there is only one positive integer satisfying both inequalities (as we hoped) and that positive integer is our sought after row r:

$$r = \left\lceil \frac{-1 + \sqrt{1 + 8n}}{2} \right\rceil$$

Lets verify this for fun again, making sure that the eighth pair is on row four:

$$\left\lceil \frac{-1 + \sqrt{1 + 8 \times 8}}{2} \right\rceil = \left\lceil \frac{-1 + \sqrt{65}}{2} \right\rceil = \lceil 3.53113 \rceil = 4$$

We know that $j = r$ and then $i = n - \frac{j(j-1)}{2}$. This means that f is surjective and therefore a bijection.

The inverse $f^{-1}(n)$ is:

$f^{-1} : \mathbb{N} \to M$

$f^{-1}(n) = (i, j)$, where $j = \left\lceil \frac{-1 + \sqrt{1 + 8n}}{2} \right\rceil$ and $i = n - \frac{j(j-1)}{2}$

For a given pair (i, j) lets divide interval $(0, 1)$ into j non-overlapping intervals:

$$(0, \frac{1}{j}], (\frac{1}{j}, \frac{2}{j}], \ldots, (\frac{j-2}{j}, \frac{j-1}{j}], (\frac{j-1}{j}, 1)$$

Except for the last subinterval, all other subintervals are left-exclusive and right-inclusive. The last one is open on both ends. This is just a technicality, but we now have a set of intervals that don't intersect and their union is $(0, 1)$.

A given $x \in (0, 1)$ will fall into one of these subintervals. We will use this fact shortly.

We are ready to define our sequence (a_n):

$$a_n = \ln(\frac{j-i}{i}), \text{ where } j = \left\lceil \frac{-1 + \sqrt{1 + 8n}}{2} \right\rceil \text{ and } i = n - \frac{j(j-1)}{2}$$

For any $x \in \mathbb{R}$ we first get $y = F(x) = \frac{1}{1+e^x}$ which places us in interval $(0, 1)$. We choose the following subsequence of (a_{n_k}): choose the n_k so that the corresponding (i, j) pair according to our bijection f^{-1} is the ith interval of the division of $(0, 1)$ into j non-overlapping intervals that contains y. Keep increasing j and selecting the corresponding (a_{n_k}) according to this criteria. This subsequence converges to x.

This construction is not unique. We made pretty arbitrary choices along the way. There are more than one sequence (a_n) with the desired property.

□

31
Existence of n-th root

Proving the existence of n-th root is the topic of this note.

This note is an experiment in including PDF pages. The included PDF pages are scans of handwriting with a fountain pen. The proof is a modification of the proof of the existence of the square root from Chapter 2 of the *Lectures on Real Analysis* textbook[1] to the n-th root case.

[1] F. Lárusson. *Lectures on Real Analysis*. Australian Mathematical Society Lecture Series. Cambridge University Press, 2012. ISBN 9781107026780. URL https://books.google.com/books?id=koj-IrXXwocC

-1-

There is $s \in \mathbb{R}$ with $s^n = y$ for $y > 0$ and $n \geq 2$

Proof:

consider set $A = \{ x \in \mathbb{R} : x^n < y \}$

$0 \in A \Rightarrow A \neq \emptyset$

$\forall x \in A: \; x < y \Rightarrow y$ upper bound

$\Rightarrow \; s := \sup A$ exists

Claim: $s^n = y$

We show this by eliminating the possibilities $s^n < y$, $s^n > y$

Assume $s^n < y$

We will work with expression $(s+\varepsilon)^n$

$$(s+\varepsilon)^n = \sum_{k=0}^{n} \binom{n}{k} s^{n-k} \varepsilon^k$$

$$= s^n + \sum_{k=1}^{n} \binom{n}{k} s^{n-k} \varepsilon^k$$

$$= s^n + \varepsilon \cdot \sum_{k=1}^{n} \binom{n}{k} s^{n-k} \varepsilon^{k-1}$$

we will make sure that

$0 < \varepsilon < 1$

$$\Rightarrow \; (s+\varepsilon)^n < s^n + \varepsilon \underbrace{\sum_{k=1}^{n} \binom{n}{k} s^{n-k}}_{B :=}$$

-2-

Now choose ε such that

$$0 < \varepsilon < \min\left(1, \frac{y - s^n}{B}\right)$$

$$\Rightarrow (s+\varepsilon)^n < s^n + \varepsilon \cdot B < s^n + \frac{y-s^n}{B} \cdot B = y$$

$\Rightarrow s + \varepsilon \in A$ ↯ contradicts $s = \sup A$

Assume $s^n > y$

We will work with expression $(s-\varepsilon)^n$

$$(s-\varepsilon)^n = \sum_{k=0}^{n} \binom{n}{k} s^{n-k} (-\varepsilon)^k$$

$$= s^n + \sum_{k=1}^{n} \binom{n}{k} s^{n-k} (-\varepsilon)^k$$

$$= s^n + \sum_{\substack{k=1 \\ k \equiv 0 \bmod 2}}^{n} \binom{n}{k} s^{n-k} \varepsilon^k - \sum_{\substack{k=1 \\ k \equiv 1 \bmod 2}}^{n} \binom{n}{k} s^{n-k} \varepsilon^k$$

$$> s^n - \sum_{\substack{k=1 \\ k \equiv 1 \bmod 2}}^{n} \binom{n}{k} s^{n-k} \varepsilon^k$$

$$= s^n - \varepsilon \cdot \sum_{\substack{k=1 \\ k \equiv 1 \bmod 2}}^{n} \binom{n}{k} s^{n-k} \varepsilon^{k-1}$$

Again, we make sure $0 < \varepsilon < 1$

$$\Rightarrow (s-\varepsilon)^n > s^n - \varepsilon \underbrace{\sum_{\substack{k=1 \\ k \equiv 1 \bmod 2}}^{n} \binom{n}{k} s^{n-k}}_{C :=}$$

-3-

Now we choose ε such that

$$0 < \varepsilon < \min\left(1, \frac{s^n - y}{c}\right)$$

$$\Rightarrow (s-\varepsilon)^n > s^n - \varepsilon \cdot c > s^n - \frac{s^n - y}{c} \cdot c = y$$

$\Rightarrow$ $s-\varepsilon$ is upper bound of A

$\hookrightarrow$ contradicts $s = \sup A$

So $s^n = y$ ∎

32
Bibliography

Stephen Abbott. *Understanding Analysis*. Springer, 2 edition, 2015. ISBN 978-1-4939-2711-1.

Michael Artin. *Algebra*. Addison Wesley, 2 edition, 2010. ISBN 0132413779.

M. Beck and R. Geoghegan. *The Art of Proof: Basic Training for Deeper Mathematics*. Undergraduate Texts in Mathematics. Springer New York, 2010. ISBN 9781441970237.

Ethan Canin. *The Palace Thief Stories*, chapter Batorsag and Szerelem, page 87. Random House New York, 1994.

N.L. Carothers. *Real Analysis*. Cambridge University Press, 2000. ISBN 9780521497565. URL https://books.google.com/books?id=4VFDVy1NFiAC.

Tung Kam Chuen. 0-1 sequences. 2016. URL https://open.kattis.com/problems/sequences.

Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990.

Freeman J. Dyson. Note 1931-The problem of the pennies. *Math. Gaz.*, 30:231–234, 1946.

Ö. Eğecioğlu and A.M. Garsia. *Lessons in Enumerative Combinatorics*. Graduate Texts in Mathematics. Springer International Publishing, 2021. ISBN 9783030712501. URL https://books.google.com/books?id=5BMuEAAAQBAJ.

A. Engel. *Problem-Solving Strategies*. Problem Books in Mathematics. Springer New York, 2013. ISBN 9781475789546. URL https://books.google.com/books?id=aUofswEACAAJ.

Jeff Erickson. *Algorithms, Etc*. 2015. URL http://jeffe.cs.illinois.edu/teaching/algorithms/.

Jeff Erickson. Algorithms — Extended Dance Remix: Fast Fourier Transforms. https://jeffe.cs.illinois.edu/teaching/algorithms/notes/A-fft.pdf, 2021. [Online; accessed 07-May-2022].

Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, 3rd edition, 1993. ISBN 0201549832.

A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990.

Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358.

Géza Kós. On the grasshopper problem with signed jumps. *The American Mathematical Monthly*, 118:877–886, 2010. URL https://arxiv.org/abs/1008.2936.

F. Lárusson. *Lectures on Real Analysis*. Australian Mathematical Society Lecture Series. Cambridge University Press, 2012. ISBN 9781107026780. URL https://books.google.com/books?id=koj-IrXXwocC.

J.W. Lawrence and F.A. Zorzitto. *An Introduction to Abstract Algebra: A Comprehensive Introduction*. Cambridge Mathematical Textbooks. Cambridge University Press, 2021. ISBN 9781108836654. URL https://books.google.com/books?id=PvQgEAAAQBAJ.

N. Loehr. *Combinatorics*. Discrete Mathematics and Its Applications. CRC Press, 2017. ISBN 9781498780278.

Cosmin Negruseri. Codejam 2008 round 1a: Problem c: Numbers. 2008. URL https://code.google.com/codejam/contest/32016/dashboard#s=p2.

D. Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2002. ISBN 9780521002899. URL https://books.google.com/books?id=B7Ch-c2G21MC.

J. Propp. Real Analysis in Reverse. *ArXiv e-prints*, April 2012. URL https://arxiv.org/abs/1204.4483.

Marian Rejewski. How Polish mathematicians broke the Enigma cipher. *IEEE Annals of the History of Computing*, 3(3):213–234, 1981. ISSN 1058-6180.

Romeo Rizzi. A short proof of König's matching theorem. *Journal of Graph Theory*, 33(3):138–139, 2000. URL https://math.dartmouth.edu/archive/m38s12/public_html/sources/Rizzi2000.pdf.

Günter Rote. *Crossing the Bridge at Night*. World Wide Web, http://page.mi.fu-berlin.de/~rote/Papers/pdf/Crossing+the+bridge+at+night.pdf, 2002.

Alexander Schrijver. On the history of the transportation and maximum flow problems. 2002. URL http://homepages.cwi.nl/~lex/files/histtrpclean.pdf.

Spotify. Cat vs dog. 2012. URL https://labs.spotify.com/puzzles/.

T. Tao. *An Introduction to Measure Theory*. Graduate Studies in Mathematics. American Mathematical Society, 2021. ISBN 9781470466404. URL https://books.google.com/books?id=k0lDEAAAQBAJ.

Michael Tong. Devil's chessboard. 2013. URL https://brilliant.org/discussions/thread/the-devils-chessboard/.

J. Vábek. On Rejewski's solution of Enigma cipher. In *PROCEEDINGS OF WDS 2006*. MATFYZPRESS, 2006.

Eric W. Weisstein. Greatest dividing exponent. From MathWorld—A Wolfram Web Resource. URL http://mathworld.wolfram.com/GreatestDividingExponent.html.

Wikipedia. Inclusion–exclusion principle — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Inclusion%E2%80%93exclusion%20principle&oldid=1086507513, 2022. [Online; accessed 07-May-2022].

33
Index

Todo list