5

Ranking and Unranking

This chapter studies the notions of *ranking* and *unranking* from a bijective viewpoint. Intuitively, our goal is to find algorithms that implement bijective maps between an n-element set of combinatorial objects and the set of integers $\underline{\mathbf{n}} = \{0, 1, 2, \ldots, n-1\}$. These algorithms will allow us to solve a variety of combinatorial problems. We begin the chapter by introducing some of these problems. Then we discuss bijective versions of the sum and product rules. These new rules provide a mechanical method for translating the counting arguments in earlier chapters into explicit bijections. Recursions derived using the sum and product rules can be treated in the same way; the resulting bijections are typically specified by recursive algorithms.

5.1 Ranking, Unranking, and Related Problems

Suppose S is a finite set of objects. We will study five fundamental combinatorial problems involving the set S: counting, listing, ranking, unranking, and random selection.

1. *Counting.* The counting problem asks us to compute the number of elements in the finite set S. We have already discussed tools to solve the counting problem in Chapter 1.

2. *Listing.* The listing problem asks us to list all the elements in the set S exactly once. There are many possible lists for a given set S; usually we produce lists that present the objects of S in a special order. Examples of such orderings are lexicographic orderings and Gray code orderings.

3. *Ranking.* Suppose we have specified a particular ordering for listing the elements of S. Given an object $x \in S$, the ranking problem asks us to calculate the position (or *rank*) of x on the list without actually listing all the objects preceding (or following) x on the list. It is often convenient to number the positions on the list starting with position 0.

4. *Unranking.* Suppose we have specified a particular ordering for listing the elements of S. Given an integer m with $0 \leq m < |S|$, the unranking problem asks us to calculate the object x in S that occupies position m on the list, without actually generating the whole list.

5. *Random Selection.* The random selection problem asks us to devise a way to choose a random element of S, where "random" means that each element of S is equally likely to be chosen. We assume that we are given a device that will produce random real numbers in the interval $[0, 1]$. Alternatively, we can assume that we have a device that, given a positive integer n, randomly picks an integer in the set $\underline{\mathbf{n}} = \{0, 1, \ldots, n-1\}$.

If we can solve the listing problem for S, then we can (in principle) solve the counting problem, the ranking problem, and the unranking problem by simply writing down the whole

list. However, we usually desire more efficient methods for counting S, ranking objects, and unranking integers. Also, this method is impractical if we are studying not just one finite set S but a whole infinite family of such sets.

Similarly, if we can solve the counting problem and unranking problem for S, then we can solve the random selection problem as follows. Use the random number generator described above, with $n = |S|$, to get a random integer between 0 and $|S| - 1$. Unrank this integer to get the random object in S. Note that if $|S|$ is very large, it may be difficult to implement a random number generator that uniformly chooses integers in the desired range. Thus, even if we can count S and unrank elements of S, it is valuable to find other ways to solve the random selection problem that avoid the random selection of integers in a huge range.

In bijective combinatorics, we try to solve the five enumeration problems posed above by constructing explicit *bijections*. For example, the ranking problem amounts to constructing a bijection $r : S \to \underline{\mathbf{n}}$ with specified properties. The unranking problem amounts to constructing the inverse bijection $u : \underline{\mathbf{n}} \to S$. The list of elements of S determined by the unranking map u is the sequence $(u(0), u(1), \ldots, u(n-1))$.

Here and below, "constructing" a map $h : A \to B$ means giving an algorithm that takes as input an element $a \in A$ and produces as output the corresponding element $h(a) \in B$. We must also prove that the proposed algorithm is indeed a bijection, and we would like to have algorithms that are as efficient as possible. Note that knowing an algorithm to compute a bijection $h : A \to B$ is not the same as knowing an algorithm to compute the inverse bijection $h^{-1} : B \to A$. We say that we have solved the counting problem for S *bijectively* if we have algorithms to compute both a bijection $u : \underline{\mathbf{n}} \to S$ and its inverse $r : S \to \underline{\mathbf{n}}$. Observe that the bijective counting problem is harder than the original enumerative counting problem: if S is complicated, we may be able to determine that $|S| = n$ without constructing any explicit bijections between S and $\underline{\mathbf{n}}$. By definition, saying that $|S| = n$ means that such bijections *exist*. But knowing this abstract existence statement is much weaker than actually giving constructions and algorithms that implement particular bijections.

As noted above, each bijection $u : \underline{\mathbf{n}} \to S$ provides one solution to the listing problem for S. If we are asked to list elements of S in a particular order, then we must construct an appropriate bijection u such that the list $(u(0), u(1), \ldots, u(n-1))$ contains the objects in S in the desired order. It is sometimes desirable to have an auxiliary *successor* algorithm that, given an object $u(i)$ in the list, outputs the next object $u(i+1)$ without ever explicitly computing i. We can then list the objects in S by starting with $u(0)$ and repeatedly invoking the successor algorithm. We will consider the construction of successor algorithms at the end of this chapter.

5.2 Bijective Sum Rule

We begin our study of ranking and unranking by revisiting the fundamental counting rules from Chapter 1. Our first rule lets us assemble ranking (resp. unranking) maps for two disjoint finite sets to obtain a ranking (resp. unranking) map for the union of these sets. Throughout this chapter, the notation $\underline{\mathbf{n}}$ will be used to denote the set $\{0, 1, \ldots, n-1\}$.

5.1. Bijective Sum Rule for Two Sets. Let S and T be disjoint finite sets.
(a) Given bijections $f : S \to \underline{\mathbf{n}}$ and $g : T \to \underline{\mathbf{m}}$, there is a canonical bijection $f + g : S \cup T \to \underline{\mathbf{n} + \mathbf{m}}$ defined by

$$(f+g)(x) = \begin{cases} f(x) & \text{for } x \in S; \\ g(x) + n & \text{for } x \in T. \end{cases}$$

(b) Given bijections $f' : \underline{\mathbf{n}} \to S$ and $g' : \underline{\mathbf{m}} \to T$, there is a canonical bijection $f' + g' : \underline{\mathbf{n+m}} \to S \cup T$ defined by

$$(f' + g')(k) = \begin{cases} f'(k) & \text{for } 0 \leq k < n \\ g'(k-n) & \text{for } n \leq k < n+m \end{cases}$$

(c) If $f' = f^{-1}$ and $g' = g^{-1}$, then $f' + g' = (f+g)^{-1}$.

We leave the detailed verification of this rule as an exercise. The disjointness of S and T is critical when showing that $f+g$ is a well-defined function and that $f'+g'$ is injective. Observe that the *order* in which we combine the bijections makes a difference: the bijection $f+g : S \cup T \to \underline{\mathbf{n+m}}$ is not the same as the bijection $g+f : S \cup T \to \underline{\mathbf{n+m}}$. Intuitively, the ranking bijection $f+g$ assigns earlier ranks to elements of S (using f to determine these ranks) and assigns later ranks to elements of T (using g); $g+f$ does the opposite. Similarly, the unranking map $f'+g'$ generates a list in which elements of S occur first, followed by elements of T; $g'+f'$ lists elements of T first, then S.

Iterating the bijective sum rule for two sets leads to the following general version of this rule.

5.2. Bijective Sum Rule for k Sets. Suppose $(S_1, \ldots, S_k)$ is an ordered list of pairwise disjoint finite sets. Let $S = \bigcup_{i=1}^{k} S_i$ be the union of these sets. Let $n_i = |S_i|$ and $n = |S| = n_1 + n_2 + \cdots + n_k$.
(a) Given bijections $f_i : S_i \to \underline{\mathbf{n_i}}$ for $1 \leq i \leq k$, there is a canonical bijection $f = \sum_{i=1}^{k} f_i : S \to \underline{\mathbf{n}}$ defined by $f(x) = f_i(x) + \sum_{j<i} n_j$ for $x \in S_i$.
(b) Given bijections $f_i' : \underline{\mathbf{n_i}} \to S_i$ for $1 \leq i \leq k$, there is a canonical bijection $f' = \sum_{i=1}^{k} f_i' : \underline{\mathbf{n}} \to S$ specified as follows. For each $x \in \underline{\mathbf{n}}$, there exists a unique i $(1 \leq i \leq k)$ such that $n_1 + \cdots + n_{i-1} \leq x < n_1 + \cdots + n_i$. Define $f'(x) = f_i'(x - [n_1 + \cdots + n_{i-1}]) \in S_i \subseteq S$.
(c) If $f_i' = f_i^{-1}$ for each i, then $f' = f^{-1}$.

As before, we leave the formal proof of the bijective sum rule to the reader. (One can give a direct proof that the maps in question are bijections, or use an induction argument involving the bijective sum rule for two sets to show that $\sum_{i=1}^{k} f_i = \sum_{i=1}^{k-1} f_i + f_k$.) Intuitively, $f_1 + \cdots + f_k$ is the ranking map that assigns elements of S_1 to positions 0 through $n_1 - 1$ using f_1, assigns elements of S_2 to positions n_1 through $n_1 + n_2 - 1$ using f_2, etc. The unranking map $f_1' + \cdots + f_k'$ generates a listing of S in which objects in S_1 occur first, then objects in S_2, and so on.

5.3 Bijective Product Rule

Now we consider bijective versions of the product rule, which generalize the familiar "base-b expansions" of natural numbers. Before introducing the bijective product rule, we recall the following theorem concerning integer division with remainder.

5.3. Theorem: Integer Division. Suppose a is any integer and b is a positive integer. There exist unique integers q and r such that

$$a = bq + r \text{ and } 0 \leq r < b.$$

Furthermore, there is an algorithm to compute q and r given a and b. The integers q and r are called the *quotient* and *remainder* when a is divided by b.

Proof. First assume $a \geq 0$. Consider the following algorithm. Define $a_0 = a$ and $i = 0$, and then loop as follows. If $a_i \geq b$, define $a_{i+1} = a_i - b \geq 0$, then replace i by $i+1$, and continue to loop. Otherwise, terminate with the answer $q = i$ and $r = a_i$. Now, this algorithm must terminate, since otherwise we would have an infinite strictly decreasing sequence $a_0 > a_1 > a_2 > \cdots$ of elements of $\mathbb{N}$, which is impossible. An induction on i shows that $a = bi + a_i$ for each i such that a_i is defined. Therefore, when the algorithm terminates, we will indeed have $a = bq + r$ and $0 \leq r < b$.

If $a < 0$, use the preceding algorithm to compute $q_1, r_1 \in \mathbb{Z}$ with $|a| = bq_1 + r_1$ and $0 \leq r_1 < b$. Then $a = -bq_1 - r_1$. If $r_1 = 0$, set $q = -q_1$ and $r = 0$; otherwise, set $q = -1 - q_1$ and $r = b - r_1$. One readily checks that $a = bq + r$ and $0 \leq r < b$. This completes the algorithmic proof of the existence of q and r.

For uniqueness of q and r, suppose we have $a = bq + r = bq' + r'$ where $q, r, q', r' \in \mathbb{Z}$ and $0 \leq r, r' < b$. Rearranging the given equations, we see that $b(q - q') = r' - r$. The right side is an integer strictly between $-b$ and b, whereas the left side is an integer multiple of b. The only such multiple of b is zero, so $q = q'$ and $r = r'$. □

5.4. Remark. The division algorithm used in the preceding proof is quite inefficient. In practice, one divides a by b using the "long division" algorithm (see 5.90).

5.5. Theorem: Base-b Expansions. Let $b > 1$ be a fixed positive integer. For every integer $a \geq 0$, there exists a unique sequence $(d_0, d_1, d_2, \ldots)$ of integers satisfying the following properties: $0 \leq d_i < b$ for all i; all but finitely many d_i's are zero; and

$$a = d_0 + d_1 b + d_2 b^2 + \cdots + d_i b^i + \cdots = \sum_{i \geq 0} d_i b^i.$$

We call $(d_0, d_1, \ldots)$ the *base-b expansion of a*, which can be written more concisely as $[a]_b = \cdots d_3 d_2 d_1 d_0$.

We only sketch the proof. The idea is to divide a by b repeatedly. The first remainder r is the last "digit" d_0; expanding the first quotient q in the same manner yields the remaining digits $\cdots d_3 d_2 d_1$. Uniqueness follows by induction, using the uniqueness assertion in 5.3. We will give another proof of this result later, by using the bijective product rule to rank and unrank words in the product set $\underline{\mathbf{b}}^k$.

We can generalize the preceding discussion by allowing the base b to change at each step. This idea leads to the general version of the bijective product rule, presented below. First, we consider the simpler version of this rule involving two sets.

5.6. Theorem: Bijective Product Rule for Two Sets. Suppose s and t are positive integers, and $n = st$. The map $p = p_{s,t} : \underline{\mathbf{s}} \times \underline{\mathbf{t}} \to \underline{\mathbf{n}}$ given by $p(i, j) = it + j$ is a bijection. To compute the inverse map $p'(u)$ (where $u \in \underline{\mathbf{n}}$), use division to write $u = qt + r$ where $0 \leq r < t$, and set $p'(u) = (q, r)$.

Proof. First we check that p does map $\underline{\mathbf{s}} \times \underline{\mathbf{t}}$ into $\underline{\mathbf{n}}$. Suppose $0 \leq i < s$ and $0 \leq j < t$ are integers. Then $0 \leq it + j$. Furthermore, since $i \leq s - 1$, we have $it + j \leq (s-1)t + j < (s-1)t + t = st$. Thus, $p(i, j) \in \underline{\mathbf{st}} = \underline{\mathbf{n}}$. Next we check that p' does map $\underline{\mathbf{n}}$ into $\underline{\mathbf{s}} \times \underline{\mathbf{t}}$. Given $u \in \underline{\mathbf{n}}$, the integers q and r in the definition of p' are well defined, by the existence and uniqueness assertions in 5.3. The condition on the remainder in 5.3 assures us that $r \in \underline{\mathbf{t}}$. Since $q < 0$ implies $u < 0$, while $q \geq s$ implies $u \geq st$, and we are assuming that $0 \leq u < n = st$, we conclude that $0 \leq q < s$. Thus, $q \in \underline{\mathbf{s}}$, so p' does map into the set $\underline{\mathbf{s}} \times \underline{\mathbf{t}}$. It is now routine to check that the maps p and p' are indeed two-sided inverses of each other, so both maps are bijections. □

5.7. Remark. Note that the bijection $p_{s,t}$ just constructed depends on the order of s and t. More explicitly, $p_{s,t} : \underline{\mathbf{s}} \times \underline{\mathbf{t}} \to \underline{\mathbf{n}}$ sends (i,j) to $it+j$, whereas $p_{t,s} : \underline{\mathbf{t}} \times \underline{\mathbf{s}} \to \underline{\mathbf{n}}$ sends (j,i) to $js+i$. Let $F : \underline{\mathbf{s}} \times \underline{\mathbf{t}} \to \underline{\mathbf{t}} \times \underline{\mathbf{s}}$ be the bijection given by $F((i,j)) = (j,i)$. The preceding formulas show that $p_{t,s} \circ F \neq p_{s,t}$ for $s \neq t$, although both functions are bijections from $\underline{\mathbf{s}} \times \underline{\mathbf{t}}$ to $\underline{\mathbf{st}}$.

5.8. Remark. The bijections in the product rule can also be built automatically using the bijective sum rule. In the sum rule, take $S_i = \{i\} \times \underline{\mathbf{t}}$ for $0 \leq i < s$, and let $f_i : S_i \to \underline{\mathbf{t}}$ be the bijection defined by $f_i(i,j) = j$ for all $j \in \underline{\mathbf{t}}$. The set $\underline{\mathbf{s}} \times \underline{\mathbf{t}}$ is the disjoint union of the S_i's, so the bijective sum rule furnishes a bijection $f = \sum_{i=0}^{s-1} f_i : \underline{\mathbf{s}} \times \underline{\mathbf{t}} \to \underline{\mathbf{st}}$. The formula for f gives $f(i,j) = f_i(i,j) + \sum_{0 \leq k < i} |S_i| = j + it = p_{s,t}(i,j)$. Similarly, $p_{t,s} \circ F$ can be obtained by adding up the bijections $\underline{\mathbf{s}} \times \{j\} \to \underline{\mathbf{s}}$ sending (i,j) to i. Since the bijective sum rule guarantees the invertibility of f, we see that the division algorithm 5.3 can be deduced as a consequence of 5.2.

5.9. Theorem. Suppose $f : A \to C$ and $g : B \to D$ are bijections. Then there is a canonical bijection $f \times g : A \times B \to C \times D$ given by

$$(f \times g)(a,b) = (f(a), g(b)) \qquad (a \in A, b \in B).$$

Proof. First, $f \times g$ is a function mapping $A \times B$ into $C \times D$. Also $f^{-1} \times g^{-1}$ (which sends (c,d) to $(f^{-1}(c), g^{-1}(d))$) is a function from $C \times D$ to $A \times B$. One checks immediately that $f^{-1} \times g^{-1}$ is the two-sided inverse of $f \times g$, so both maps are bijections. □

This result extends immediately to Cartesian products of finitely many sets.

5.10. Theorem. Suppose $f_i : S_i \to T_i$ are bijections, for $1 \leq i \leq k$. Then the map

$$f = f_1 \times f_2 \times \cdots \times f_k : S_1 \times \cdots \times S_k \to T_1 \times \cdots \times T_k,$$

given by $f(s_1, s_2, \ldots, s_k) = (f_1(s_1), f_2(s_2), \ldots, f_k(s_k))$, is a bijection with inverse $f_1^{-1} \times \cdots \times f_k^{-1}$.

5.11. Theorem: Bijective Product Rule for k Sets. Suppose $n_1, \ldots, n_k$ are given positive integers, and $n = n_1 n_2 \cdots n_k$. There are canonical bijections

$$p = p_{n_1,\ldots,n_k} : \underline{\mathbf{n_1}} \times \underline{\mathbf{n_2}} \times \cdots \times \underline{\mathbf{n_k}} \to \underline{\mathbf{n}}, \qquad p' = p'_{n_1,\ldots,n_k} : \underline{\mathbf{n}} \to \underline{\mathbf{n_1}} \times \underline{\mathbf{n_2}} \times \cdots \times \underline{\mathbf{n_k}}.$$

The map p is defined by

$$p(c_1, c_2, \ldots, c_k) = \sum_{i=1}^{k} c_i \prod_{j=i+1}^{k} n_j.$$

The inverse map p' is computed via the following algorithm. Given an input m with $0 \leq m < n$, divide m by n_k to get a quotient q_k and remainder r_k. Next, divide q_k by n_{k-1} to get a quotient q_{k-1} and remainder r_{k-1}. Continue in this way, dividing q_i by n_{i-1} to get a quotient q_{i-1} and remainder r_{i-1}, for $1 < i \leq k$. After completing all these divisions, set $p'(m) = (r_1, r_2, \ldots, r_k)$.

Proof. We use induction on k to show that p is a bijection. When $k = 1$, p is an identity map. When $k = 2$, the result follows from the bijective product rule for two sets. Finally, assume $k > 2$ and the result is already known for products of $k-1$ sets. Writing $n' = n_2 n_3 \cdots n_k$, observe that

$$\begin{aligned} p_{n_1,\ldots,n_k}(c_1, \ldots, c_k) &= c_1(n_2 \cdots n_k) + \sum_{i=2}^{k} c_i \prod_{j=i+1}^{k} n_j \\ &= c_1 n' + p_{n_2,\ldots,n_k}(c_2, \ldots, c_k) \\ &= p_{n_1,n'}(c_1, p_{n_2,\ldots,n_k}(c_2, \ldots, c_k)). \end{aligned}$$

This means that $p_{n_1,\ldots,n_k}$ is the composition of the two bijections

$$\mathrm{id}_{\underline{\mathbf{n_1}}} \times p_{n_2,\ldots,n_k} : \underline{\mathbf{n_1}} \times (\underline{\mathbf{n_2}} \times \cdots \times \underline{\mathbf{n_k}}) \to \underline{\mathbf{n_1}} \times \underline{\mathbf{n}}' \text{ and } p_{n_1,n'} : \underline{\mathbf{n_1}} \times \underline{\mathbf{n}}' \to \underline{\mathbf{n}}.$$

To show that p' is the inverse of p, it is sufficient to verify that $p'(p(c_1,\ldots,c_k)) = (c_1,\ldots,c_k)$, since we already know that $|\underline{\mathbf{n_1}} \times \cdots \times \underline{\mathbf{n_k}}| = |\underline{\mathbf{n}}|$. Again we use induction. Note that

$$p_{n_1,\ldots,n_k}(c_1,\ldots,c_k) = \sum_{i=1}^{k} c_i \prod_{i<j\le k} n_j = c_k + n_k \sum_{i=1}^{k-1} c_i \prod_{i<j<k} n_j.$$

This shows that the quotient and remainder when we divide $p(c_1,\ldots,c_k)$ by n_k are

$$q_k = \sum_{i=1}^{k-1} c_i \prod_{i<j\le k-1} n_j = p_{n_1,\ldots,n_{k-1}}(c_1,\ldots,c_{k-1})$$

and $r_k = c_k$, respectively. So the first division step successfully recovers c_k. The remaining divisions compute

$$p'_{n_1,\ldots,n_{k-1}}(p_{n_1,\ldots,n_{k-1}}(c_1,\ldots,c_{k-1})),$$

which equals $(c_1,\ldots,c_{k-1})$ by induction hypothesis. □

5.12. Example. Suppose $(n_1,n_2,n_3,n_4,n_5) = (4,6,5,4,2)$, so $n = n_1n_2n_3n_4n_5 = 960$. Then

$$p(3,1,0,2,1) = 3\cdot(6\cdot5\cdot4\cdot2) + 1\cdot(5\cdot4\cdot2) + 0\cdot(4\cdot2) + 2\cdot(2) + 1 = 765.$$

To compute $p^{-1}(222)$, first divide 222 by $n_5 = 2$ to get $q_5 = 111$ and $r_5 = 0$. Then divide 111 by $n_4 = 4$ to get $q_4 = 27$ and $r_4 = 3$. Then divide 27 by $n_3 = 5$ to get $q_3 = 5$ and $r_3 = 2$. Then divide 5 by $n_2 = 6$ to get $q_2 = 0$ and $r_2 = 5$. Finally, divide 0 by $n_1 = 4$ to get $q_1 = 0$ and $r_1 = 0$. We conclude that $p^{-1}(222) = (0,5,2,3,0)$.

5.13. Remark. If $n_1 = \cdots = n_k = b$, then $p^{-1}_{b,b,\ldots,b}(z)$ computes the base-b expansion of z (for $0 \le z < b^k$). Here, p and p^{-1} are bijections between the set of words $\{0,1,\ldots,b-1\}^k$ and the set of integers $\{0,1,2,\ldots,b^k-1\}$. Taking $b = 10$, we see that the decimal representation of positive integers is a special case of the bijective product rule.

5.14. Remark. Here is another algorithm for computing $p^{-1}_{n_1,\ldots,n_k}(m) = p^{-1}(p(c_1,\ldots,c_k))$ that recovers the numbers $(c_1,\ldots,c_k)$ from left to right. First, divide m by $n_2n_3\cdots n_k$ to obtain a quotient q_1 and a remainder r_1. Set $c_1 = q_1$, and recover $(c_2,\ldots,c_k)$ by recursively computing $p^{-1}_{n_2,\ldots,n_k}(r_1)$ in the same fashion. We let the reader prove that this algorithm does implement the inverse of $p_{n_1,\ldots,n_k}$.

We can now describe the general strategy for converting informal counting arguments based on the product rule to ranking and unranking algorithms. When using the product rule, we uniquely construct objects in a set S by making k choices, such that there are always n_i ways to make the ith choice. There is often a natural ordering of the choices that can be made at the ith stage, given the choices that have already been made; thus we can number the available choices at this stage $0,1,\ldots,n_i-1$. Then the informal construction process for manufacturing objects in S can be translated into a formal bijection $f : \underline{\mathbf{n_1}} \times \cdots \times \underline{\mathbf{n_k}} \to S$, where $f(c_1,\ldots,c_k)$ is the object constructed by making the choice numbered c_i at the ith stage, for $1 \le i \le k$. To solve the unranking problem for S, we use the composite bijection

$$f \circ p^{-1}_{n_1,\ldots,n_k} : \underline{\mathbf{n}} \to S.$$

Similarly, if we can find an algorithm that recovers the choices c_i from the final object $x \in S$, then we can compute f^{-1}. The ranking problem for S is then solved by the bijection

$$p_{n_1,\ldots,n_k} \circ f^{-1} : S \to \underline{\mathbf{n}}.$$

In the next several sections, we apply this idea to find ranking and unranking bijections for many commonly occurring families of combinatorial objects.

5.4 Ranking Words

5.15. Example: Four-Letter Words. Let S be the set of all four-letter words. We can build elements of S by choosing the first letter, then the second, third, and fourth letters. To describe this choice process formally as a bijection, let $A = \{a, b, \ldots, z\}$ be the alphabet. We identify a word $w = w_1w_2w_3w_4 \in S$ with the 4-tuple $(w_1, w_2, w_3, w_4) \in A \times A \times A \times A$. Choose a fixed bijection $f : A \to \underline{\mathbf{26}}$; for instance, we can use the standard alphabetical ordering given by

$$f(a) = 0,\ f(b) = 1,\ f(c) = 2, \ldots,\ f(y) = 24,\ f(z) = 25.$$

Then $f \times f \times f \times f$ is a bijection from $S = A \times A \times A \times A$ to $\underline{\mathbf{26}}^4$. Composing with the bijection $p : \underline{\mathbf{26}} \times \underline{\mathbf{26}} \times \underline{\mathbf{26}} \times \underline{\mathbf{26}} \to \underline{\mathbf{26^4}} = \underline{\mathbf{456,976}}$, we obtain a *ranking map* $r : S \to \underline{\mathbf{456,976}}$. For example,

$$r(\text{goop}) = p_{26,26,26,26}(6, 14, 14, 15) = 6 \cdot 26^3 + 14 \cdot 26^2 + 14 \cdot 26^1 + 15 = 115,299;$$

$$r(\text{pogo}) = p_{26,26,26,26}(15, 14, 6, 14) = 15 \cdot 26^3 + 14 \cdot 26^2 + 6 \cdot 26^1 + 14 = 273,274.$$

The inverse of r is the *unranking map* $u : \underline{\mathbf{456,976}} \to S$. To compute $u(x)$, we first express x in base 26 (this is what $p^{-1}_{26,26,26,26}$ does), and then use the inverse of f to convert back to letters. For example,

$$u(200,000) = u(11 \cdot 26^3 + 9 \cdot 26^2 + 22 \cdot 26 + 8) = f^{-1}(11)f^{-1}(9)f^{-1}(22)f^{-1}(8) = \text{ljwi}.$$

In general, if A is an m-letter alphabet, we can rank the set of k-letter words A^k by fixing a bijection $f : A \to \underline{\mathbf{m}}$ and computing

$$r(w_1w_2 \cdots w_k) = p_{m,m,\ldots,m}(f(w_1), \ldots, f(w_k)).$$

To unrank an integer $z \in \underline{\mathbf{m^k}}$, write $z = d_{k-1} \cdots d_0$ in base m and then replace each digit d_i by $f^{-1}(d_i)$.

5.16. Example: Three-Letter Words. Now consider $S = \{a, b, c\}^3$. Define $f : \{a, b, c\} \to \underline{\mathbf{3}}$ by $f(a) = 0$, $f(b) = 1$, and $f(c) = 2$. The ranking map $r : S \to \underline{\mathbf{27}}$ is defined by

$$r(w_1w_2w_3) = f(w_1) \cdot 9 + f(w_2) \cdot 3 + f(w_3).$$

To define the unranking map $u : \underline{\mathbf{27}} \to S$, write $z \in \underline{\mathbf{27}}$ as $z = d_2d_1d_0$ in base 3. Then

$$u(z) = u(d_2d_1d_0) = f^{-1}(d_2)f^{-1}(d_1)f^{-1}(d_0).$$

These bijections set up the following one-to-one correspondences between S, $\underline{\mathbf{3}} \times \underline{\mathbf{3}} \times \underline{\mathbf{3}}$, and $\underline{\mathbf{27}}$:

aaa ↔ 000=0	baa ↔ 100=9	caa ↔ 200=18
aab ↔ 001=1	bab ↔ 101=10	cab ↔ 201=19
aac ↔ 002=2	bac ↔ 102=11	cac ↔ 202=20
aba ↔ 010=3	bba ↔ 110=12	cba ↔ 210=21
abb ↔ 011=4	bbb ↔ 111=13	cbb ↔ 211=22
abc ↔ 012=5	bbc ↔ 112=14	cbc ↔ 212=23
aca ↔ 020=6	bca ↔ 120=15	cca ↔ 220=24
acb ↔ 021=7	bcb ↔ 121=16	ccb ↔ 221=25
acc ↔ 022=8	bcc ↔ 122=17	ccc ↔ 222=26

Notice that, as z runs from 0 to 26, the words in S are generated in alphabetical order.

More generally, using the bijective product rule will generate the elements of a set S in a certain *lexicographic order* that is determined by the nature of the "choice bijection" $\underline{\mathbf{n_1}} \times \underline{\mathbf{n_2}} \times \cdots \times \underline{\mathbf{n_k}} \to S$ used to build the objects in S from a sequence of choices. According to our definition of $p_{n_1,\ldots,n_k}$ and its inverse, the first choice in the sequence (which can occur in n_1 ways) is deemed "most significant," and the last choice (which can occur in n_k ways) is "least significant." If we unrank $0, 1, \ldots, n-1$ in this order, we obtain a list that begins with the $n_2 \cdots n_k$ objects that can be made by choosing zero in the first choice. Next we get all the objects that can be made by choosing one in the first choice, etc. Each such sublist is also arranged lexicographically, according to the choices made at stages $2, 3, \ldots, k$.

5.17. Example: Words with Restrictions. Let S be the set of four-letter words $w_1w_2w_3w_4$ that begin and end with consonants and have a vowel in the second position. Choosing letters from left to right and using the product rule, we see that $|S| = 21{\cdot}5{\cdot}26{\cdot}21 = 57,330$. Let C, V, and A denote the set of consonants, vowels, and all letters, respectively. The usual alphabetical order defines bijections $C \to \underline{\mathbf{21}}$, $V \to \underline{\mathbf{5}}$, and $A \to \underline{\mathbf{26}}$; for example,

$$V \to \underline{\mathbf{5}} \text{ via } a \mapsto 0,\ e \mapsto 1,\ i \mapsto 2,\ o \mapsto 3,\ u \mapsto 4.$$

We obtain a ranking map $r : S \to \underline{\mathbf{57,330}}$ by defining

$$r(w_1w_2w_3w_4) = p_{21,5,26,21}(w_1', w_2', w_3', w_4'),$$

where w_i' denotes the image of w_i under the appropriate bijection. For example,

$$r(\text{host}) = p_{21,5,26,21}(5, 3, 18, 15) = 5 \cdot (5 \cdot 26 \cdot 21) + 3 \cdot (26 \cdot 21) + 18 \cdot (21) + 15 = 15,681.$$

We unrank by applying $p^{-1}_{21,5,26,21}$ and then decoding to letters. For example, repeated division shows that

$$p^{-1}_{21,5,26,21}(44001) = (16, 0, 15, 6),$$

and therefore $u(44001) = \text{vapj}$. This unranking method generates the words in S in alphabetical order.

5.18. Example: License Plates. A California license plate consists of a digit, followed by three letters, followed by three digits. We can use the preceding ideas to rank and unrank license plates. For instance,

$$r(\text{3PZY292}) = p_{10,26,26,26,1000}(3, 15, 25, 24, 292) = 63,542,292.$$

5.5 Ranking Permutations

In the examples considered so far, the choices made at each stage of the product rule did not depend on what choices were made in previous stages. This section studies the more complicated situation where the available choices do depend on what happened earlier. We illustrate this situation by solving the ranking and unranking problems for permutations.

Suppose A is an n-letter alphabet. Recall that a k-permutation of A is a word $w = w_1 w_2 \cdots w_k$, where the w_i's are *distinct* elements of A. Let S be the set of all k-permutations of A. Using the ordinary product rule, we build elements w in S by choosing $w_1 \in A$ in n ways, then choosing $w_2 \in A \sim \{w_1\}$ in $n-1$ ways, and so on. At the ith stage (where $1 \leq i \leq k$), we choose $w_i \in A \sim \{w_1, w_2, \ldots, w_{i-1}\}$ in $n-(i-1)$ ways. Thus, $|S| = n(n-1)\cdots(n-k+1) = (n)\downarrow_k$. Notice that the set of choices available at the ith stage depends on the choices made earlier, but the *cardinality* of this set is independent of previous choices. (This last fact is a key hypothesis of the product rule.)

Let us rephrase the preceding counting argument to obtain a bijection between S and the product set $\underline{\mathbf{n}} \times \underline{\mathbf{n-1}} \times \cdots \times \underline{\mathbf{n-k+1}}$. Fix a total ordering $x = (x_0, x_1, \ldots, x_{n-1})$ of the letters in A; equivalently, fix a bijection $x : \underline{\mathbf{n}} \to A$. Suppose $w = w_1 w_2 \cdots w_k \in S$. We must map w to a k-tuple $(j_1, j_2, \ldots, j_k)$, where $0 \leq j_i < n-(i-1)$. To compute j_1, locate w_1 in the sequence $(x_0, x_1, \ldots, x_{n-1})$, let j_1 be the number of letters preceding w_1 in the sequence, and then erase w_1 from the sequence to get a new sequence x'. To compute j_2, find w_2 in the sequence x', let j_2 be the number of letters preceding it, and then erase w_2 to get a new sequence x''. Continue similarly to generate the remaining j_i's. This process is reversible, as demonstrated in the next example, so we have defined the desired bijection. Combining this bijection (and its inverse) with the maps $p_{n,n-1,\ldots,n-k+1}$ and $p^{-1}_{n,n-1,\ldots,n-k+1}$, we obtain the desired ranking and unranking maps. One may verify that these maps correspond to the alphabetic ordering of permutations specified by the given total ordering of the alphabet A.

5.19. Example. Let $n = 8$, $k = 5$, and A = (a,b,c,d,e,f,g,h) with the usual alphabetical ordering. Let w = cfbgd $\in S$. We compute $(j_1, \ldots, j_5)$ as follows:

2 letters precede c in (a,b,c,d,e,f,g,h), so	$j_1 = 2$;
4 letters precede f in (a,b,d,e,f,g,h), so	$j_2 = 4$;
1 letter precedes b in (a,b,d,e,g,h), so	$j_3 = 1$;
3 letters precede g in (a,d,e,g,h), so	$j_4 = 3$;
1 letter precedes d in (a,d,e,h), so	$j_5 = 1$.

Thus, cfbgd $\mapsto (2,4,1,3,1)$. The rank of this word is therefore

$$p_{8,7,6,5,4}(2,4,1,3,1) = 2\cdot(7\cdot6\cdot5\cdot4) + 4\cdot(6\cdot5\cdot4) + 1\cdot(5\cdot4) + 3\cdot(4) + 1 = 2193.$$

Next, let us unrank the integer 982. First, repeated division gives

$$p^{-1}_{8,7,6,5,4}(982) = (1,1,1,0,2).$$

Since $j_1 = 1$, the first letter of the desired word must be b. Removing b from the alphabet gives (a,c,d,e,f,g,h). Since $j_2 = 1$, the second letter of the desired word is c. Removing c from the previous list gives (a,d,e,f,g,h). Continuing in this way, we see that 982 unranks to give the word bcdag.

5.20. Example. Let S be the set of permutations of $(1,2,3,4,5,6)$. Using the procedure above to rank the permutation $(4,6,2,1,5,3)$, we first compute $(j_1, \ldots, j_6) = (3,4,1,0,1,0)$ and then calculate $p_{6,5,4,3,2,1}(3,4,1,0,1,0) = 463$. To unrank the integer 397, first calculate $p^{-1}_{6,5,4,3,2,1}(397) = (3,1,2,0,1,0)$. Then use these position numbers to recover the permutation $(4,2,5,1,6,3)$.

5.6 Ranking Subsets

In 1.42, we used the product rule to prove that the number of k-element subsets of an n-element set is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. This enumeration result was obtained *indirectly*, by enumerating k-permutations of an n-element set in two ways and then dividing the resulting equation by $k!$. In general, the operation of division presents serious problems when attempting to construct bijections. Therefore, we will adopt a different approach to the problem of ranking and unranking subsets. Instead of using the bijective product rule, we will apply the bijective sum rule to the recursion characterizing the binomial coefficients. This will lead us to recursive algorithms for ranking and unranking subsets.

For convenience, write $C(n,k)$ for the number of k-element subsets of an n-element set. In 2.25, we saw that these numbers satisfy the recursion

$$C(n,k) = C(n-1,k) + C(n-1,k-1) \qquad (0 < k \leq n) \tag{5.1}$$

with initial conditions $C(n,0) = 1$. This recursion came from a combinatorial argument involving the sum rule. Using the bijective sum rule instead will lead directly to recursively defined bijections for ranking and unranking. For each alphabet A, introduce the temporary notation $S_k(A)$ to denote the set of all k-element subsets of A. We assume that all alphabets to be considered are equipped with some fixed total ordering that allows us to rank and unrank individual letters of the alphabet. Suppose $A = (x_0, x_1, \ldots, x_{n-1})$ is such an alphabet with n letters. We can write $S_k(A)$ as the disjoint union of sets T and U, where T consists of all subsets that do not contain x_{n-1} and U consists of all subsets that contain x_{n-1}. Note that $T = S_k(A \sim \{x_{n-1}\})$, and U corresponds to $S_{k-1}(A \sim \{x_{n-1}\})$ via a bijection that deletes x_{n-1} from a subset belonging to U. We can use recursion to obtain ranking and unranking maps for $S_k(A \sim \{x_{n-1}\})$ and $S_{k-1}(A \sim \{x_{n-1}\})$, as these involve subsets drawn from smaller alphabets. Then we combine these maps using the bijective sum rule to get ranking and unranking maps for $S_k(A)$.

Writing out the definitions, we arrive at the following recursive ranking algorithm for mapping a subset $B \in S_k(A)$ to an integer:

- If $k = 0$ (so $B = \emptyset$), then return the answer 0.
- If $k > 0$ and the last letter x in A does not belong to B, then return the ranking of B relative to the set $S_k(A \sim \{x\})$, which we compute recursively using this very algorithm.
- If $k > 0$ and the last letter x in A does belong to B, let i be the ranking of $B' = B \sim \{x\}$ relative to the set $S_{k-1}(A \sim \{x\})$ (computed recursively), and return the answer $i + C(n-1,k)$. Note that $C(n-1,k)$ can be computed using the recursion (5.1) for binomial coefficients.

The inverse map is the following recursive unranking algorithm that maps an integer m to a subset $B \in S_k(A)$:

- If $k = 0$ (so m must be zero), then return $\emptyset$.
- If $k > 0$ and $0 \leq m < C(n-1,k)$, then return the result of unranking m relative to the set $S_k(A \sim \{x\})$, where x is the last letter of A.
- If $k > 0$ and $C(n-1,k) \leq m < C(n,k)$, then let B' be the unranking of $m - C(n-1,k)$ relative to the set $S_{k-1}(A \sim \{x\})$, and return $B' \cup \{x\}$.

5.21. Example. Let $A = (a, b, c, d, e, f, g, h)$, and let us rank the subset $B = \{c, d, f, g\} \in S_4(A)$. Since $h \notin B$, we recursively proceed to rank B relative to the 7-letter alphabet $A_1 = (a, b, c, d, e, f, g)$. The new last letter g belongs to B, so we must add $C(7-1, 4) = 15$ to the rank of $B_1 = \{c, d, f\}$ relative to (a, b, c, d, e, f). The last letter f belongs to B_1, so we must add $C(6-1, 3) = 10$ to the rank of $B_2 = \{c, d\}$ relative to (a, b, c, d, e). This is the same as the rank of B_2 relative to (a, b, c, d), which is $C(4-1, 2) = 3$ plus the rank of $\{c\}$ relative to (a, b, c), which is $C(3-1, 1) = 2$ plus the rank of $\emptyset$ relative to (a, b). Two more reductions reveal that the latter rank is zero. Adding up the contributions, we see that the rank of B is

$$\binom{3-1}{1} + \binom{4-1}{2} + \binom{6-1}{3} + \binom{7-1}{4} = 30.$$

Generalizing the pattern in the previous example, we can give the following non-recursive formula for the rank of a subset.

5.22. Sum Formula for Ranking Subsets. If $A = (x_0, x_1, \ldots, x_{n-1})$ and $B = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ where $i_1 < i_2 < \cdots < i_k$, then the rank of B relative to $S_k(A)$ is $\sum_{j=1}^{k} \binom{i_j}{j}$.

A routine induction argument can be used to prove this formula formally.

5.23. Example. Now we illustrate the recursive unranking algorithm. Let us unrank the integer 53 to obtain an object $B \in S_4(A)$, where $A = (a, b, c, d, e, f, g, h)$. Here $n = 8$ and $k = 4$. Since $C(7, 4) = 35 \leq 53$, we know that $h \in B$. We proceed by unranking $53 - 35 = 18$ to get a 3-element subset of (a, b, c, d, e, f, g). Now $C(6, 3) = 20 > 18$, so g does not lie in the subset. We proceed to unrank 18 to get a 3-element subset of (a, b, c, d, e, f). Now $C(5, 3) = 10 \leq 18$, so f does belong to B. We continue, unranking $18 - 10 = 8$ to get a two-element subset of (a, b, c, d, e). Since $C(4, 2) = 6 \leq 8$, $e \in B$ and we continue by unranking 2. We have $C(3, 1) = 3 > 2$, so $d \notin B$. But at the next stage $C(2, 1) = 2 \leq 2$, so $c \in B$. We conclude, finally, that $B = \{c, e, f, h\}$.

As before, we can describe this algorithm iteratively instead of recursively.

5.24. Unranking Algorithm for Subsets. Suppose $A = (x_0, x_1, \ldots, x_{n-1})$ and we are unranking an integer m to get a k-element subset B of A. Repeatedly perform the following steps until k becomes zero: let i be the largest integer such that $C(i, k) \leq m$; declare that $x_i \in B$; replace m by $m - C(i, k)$ and decrement k by 1.

We close with a remark about the ordering of subsets associated to the ranking and unranking algorithms described above. Let x be the last letter of A. If we unrank the integers $0, 1, 2, \ldots$ in this order to obtain a listing of $S_k(A)$, we will obtain all k-element subsets of A not containing x first, and all k-element subsets of A containing x second. Each of these sublists is internally ordered in the same way according to the next-to-last letter of A, and so on recursively. In contrast, if we had used the bijective sum rule on the recursion

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

(in which the order of the summands is swapped), then the ordering rules at each level of this hierarchy would be reversed. Similarly, the reader can construct variant ranking algorithms in which the first letter of the alphabet is considered "most significant," etc. Some of these variants are explored in the exercises.

5.7 Ranking Anagrams

Next we study the problem of ranking and unranking anagrams. Recall that $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ is the set of all words of length $n = n_1 + \cdots + n_k$ consisting of n_i copies of a_i for $1 \leq i \leq k$. We have seen (§1.9) that these sets are counted by the multinomial coefficients:

$$|\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})| = \binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1! n_2! \cdots n_k!}.$$

There are at least three ways of deriving this formula. One way counts permutations of n distinct letters in two ways, and solves for the number of anagrams by division. This method is not easily converted into a ranking algorithm. A second way uses the product rule, choosing the positions for the n_1 copies of a_1, then the positions for the n_2 copies of a_2, and so on. Combining the bijective product rule with the ranking algorithm for subsets presented earlier, this method does lead to a ranking algorithm for anagrams. A third way to count anagrams involves finding recursions involving the multinomial coefficients (§2.5). This is the approach we will pursue here.

Let $C(n; n_1, \ldots, n_k)$ be the number of rearrangements of n letters, where there are n_i letters of type i. Classifying words by their first letter leads to the recursion

$$C(n; n_1, \ldots, n_k) = \sum_{i=1}^{k} C(n-1; n_1, \ldots, n_i - 1, \ldots, n_k).$$

Applying the bijective sum rule to this recursion, we are led to recursive ranking and unranking algorithms for anagrams.

Here are the details of the algorithms. We recursively define ranking maps

$$r = r_{a_1^{n_1} \cdots a_k^{n_k}} : \mathcal{R}(a_1^{n_1} \cdots a_k^{n_k}) \to \underline{\mathbf{m}},$$

where $m = (n_1 + \cdots + n_k)!/(n_1! \cdots n_k!)$. If any n_i is negative, r is the function with graph $\emptyset$. If all n_i's are zero, r is the function sending the empty word to 0. To compute $r(w)$ in the remaining case, suppose a_i is the first letter of w. Write $w = a_i w'$. Return the answer

$$r(w) = \sum_{j<i} C(n-1; n_1, \ldots, n_j - 1, \ldots, n_k) + r_{a_1^{n_1} \cdots a_i^{n_i - 1} \cdots a_k^{n_k}}(w'),$$

where $r(w')$ is computed recursively by the same algorithm.

Next, we define the corresponding unranking maps

$$u = u_{a_1^{n_1} \cdots a_k^{n_k}} : \underline{\mathbf{m}} \to \mathcal{R}(a_1^{n_1} \cdots a_k^{n_k}).$$

Use the only possible maps if some $n_i < 0$ or if all $n_i = 0$. Otherwise, to unrank $s \in \underline{\mathbf{m}}$, first find the maximal index i such that $n_i > 0$ and $\sum_{j<i} C(n-1; n_1, \ldots, n_j - 1, \ldots, n_k) \leq s$; let s' be the difference between s and this sum. Recursively compute

$$w' = u_{a_1^{n_1} \cdots a_i^{n_i - 1} \cdots a_k^{n_k}}(s');$$

finally, return the answer $w = a_i w'$. This unranking algorithm induces a listing of the anagrams in $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ "in alphabetical order" relative to the alphabet ordering $a_1 < a_2 < \cdots < a_k$.

5.25. Example. Let us compute the rank of the word $w =$ abbcacb in $\mathcal{R}(a^2b^3c^2)$; here $n = 7$, $n_1 = 2$, $n_2 = 3$, and $n_3 = 2$. Erasing the first letter a, we see that the rank of w equals zero plus the rank of $w_1 =$ bbcacb; now $n = 6$, $n_1 = 1$, $n_2 = 3$, and $n_3 = 2$. Erasing b, we must now add $\binom{5}{0,3,2} = 10$ to the rank of $w_2 =$ bcacb; now $n = 5$, $n_1 = 1$, $n_2 = 2$, and $n_3 = 2$. Erasing the next b, we must add $\binom{4}{0,2,2} = 6$ to the rank of $w_3 =$ cacb; now $n = 4$, $n_1 = 1$, $n_2 = 1$, and $n_3 = 2$. Erasing c, we must add $\binom{3}{0,1,2} + \binom{3}{1,0,2} = 6$ to the rank of $w_4 =$ acb; now $n = 3$, $n_1 = 1$, $n_2 = 1$, and $n_3 = 1$. Continuing in this way, one sees that the rank of acb is 1. Thus, the rank of the original word is $10 + 6 + 6 + 1 = 23$.

Next, let us unrank 91 to obtain a word w in $\mathcal{R}(a^2b^3c^2)$. To determine the first letter of w, note that $0 \le 91$, $\binom{6}{1,3,2} = 60 \le 91$, but $\binom{6}{1,3,2} + \binom{6}{2,2,2} = 150 > 91$. Thus, the first letter is b, and we continue by unranking $91 - 60 = 31$ to obtain a word in $\mathcal{R}(a^2b^2c^2)$. This time, we have $0 \le 31$, $\binom{5}{1,2,2} = 30 \le 31$, but $\binom{5}{1,2,2} + \binom{5}{2,1,2} = 60 > 31$. So the second letter is b, and we continue by unranking $31 - 30 = 1$ to obtain a word in $\mathcal{R}(a^2b^1c^2)$. It is routine to check that the next two letters are both a, and we continue by unranking 1 to obtain a word in $\mathcal{R}(b^1c^2)$. The word we get is cbc, so the unranking of 91 is the word bbaacbc.

5.8 Ranking Integer Partitions

In this section, we devise ranking and unranking algorithms for integer partitions by applying the bijective sum rule to the recursion 2.42. Let $P(n,k)$ be the set of integer partitions of n with largest part k, and let $p(n,k) = |P(n,k)|$. Recall from 2.42 that these numbers satisfy

$$p(n,k) = p(n-k,k) + p(n-1,k-1) \qquad (n,k > 0).$$

The first term on the right counts elements of $P(n,k)$ in which the largest part occurs at least twice (deleting the first copy of this part gives a bijection onto $P(n-k,k)$). The second term on the right counts elements of $P(n,k)$ in which the largest part occurs exactly once (reducing this part by one gives a bijection onto $P(n-1,k-1)$). Combining these bijections with the bijective sum rule, we obtain recursively determined ranking maps $r = r_{n,k} : P(n,k) \to \underline{\mathbf{p(n,k)}}$. To find $r_{n,k}(\mu)$, consider three cases. If μ has only one part (which happens when $n = k$), return 0. If $k = \mu_1 = \mu_2$, return $r_{n-k,k}((\mu_2, \mu_3, \ldots))$. If $k = \mu_1 > \mu_2$, return $p(n-k,k) + r_{n-1,k-1}((\mu_1 - 1, \mu_2, \ldots))$. The unranking maps $u = u_{n,k} : \underline{\mathbf{p(n,k)}} \to P(n,k)$ operate as follows. To compute $u(m)$ where $0 \le m < p(n,k)$, consider two cases. If $0 \le m < p(n-k,k)$, recursively compute $\nu = u_{n-k,k}(m)$ and return the answer $\mu = (k, \nu_1, \nu_2, \ldots)$. If $p(n-k,k) \le m < p(n,k)$, recursively compute $\nu = u_{n-1,k-1}(m - p(n-k,k))$ and return the answer $\mu = (\nu_1 + 1, \nu_2, \nu_3, \ldots)$.

5.26. Example. Let us compute $r_{8,3}(\mu)$, where $\mu = (3,3,1,1)$. Since $\mu_1 = \mu_2$, the rank will be $r_{5,3}(\nu)$, where $\nu = (3,1,1)$. Next, since $\nu_1 \neq \nu_2$, we have

$$r_{5,3}(3,1,1) = p(2,3) + r_{4,2}(2,1,1) = r_{4,2}(2,1,1).$$

The first two parts of the new partition are again different, so

$$r_{4,2}(2,1,1) = p(2,2) + r_{3,1}(1,1,1) = 1 + r_{3,1}(1,1,1).$$

After several more steps, we find that $r_{3,1}(1,1,1) = 0$, so $r_{8,3}(\mu) = 1$. Thus μ is the second partition in the listing of $P(8,3)$ implied by the ranking algorithm; the first partition in this list, which has rank 0, is $(3,3,2)$.

Next, let us compute $\mu = u_{10,4}(6)$. First, $p(6,4) = 2 \leq 6$, so μ will be obtained by adding one to the first part of $\nu = u_{9,3}(4)$. Second, $p(6,3) = 3 \leq 4$, so ν will be obtained by adding one to the first part of $\rho = u_{8,2}(1)$. Third, $p(6,2) = 3 > 1$, so ρ will be obtained by adding a new first part of length 2 to $\xi = u_{6,2}(1)$. Fourth, $p(4,2) = 2 > 1$, so ξ will be obtained by adding a new first part of length 2 to $\zeta = u_{4,2}(1)$. Fifth, $p(2,2) = 1 \leq 1$, so ζ will be obtained by adding one to the first part of $\omega = u_{3,1}(0)$. We must have $\omega = (1,1,1)$, this being the unique element of $P(3,1)$. Working our way back up the chain, we successively find that

$$\zeta = (2,1,1), \quad \xi = (2,2,1,1), \quad \rho = (2,2,2,1,1), \quad \nu = (3,2,2,1,1),$$

and finally $\mu = u_{10,4}(6) = (4,2,2,1,1)$.

Now that we have algorithms to rank and unrank the sets $P(n,k)$, we can apply the bijective sum rule to the identity

$$p(n) = p(n,n) + p(n,n-1) + \cdots + p(n,1)$$

to rank and unrank the set $P(n)$ of all integer partitions of n.

5.27. Example. Let us enumerate all the integer partitions of 6. We obtain this list of partitions by concatenating the lists associated to the sets

$$P(6,6),\ P(6,5),\ \cdots,\ P(6,1),$$

written in this order. In turn, each of these lists can be constructed by applying the unranking maps $u_{6,k}$ to the integers $0, 1, 2, \ldots, p(6,k) - 1$. The reader can verify that this procedure leads to the following list:

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

$$(2,2,2),\ (2,2,1,1),\ (2,1,1,1,1),\ (1,1,1,1,1,1).$$

One may also check that the list obtained in this way presents the integer partitions of n in decreasing lexicographic order (as defined in 10.36).

5.9 Ranking Set Partitions

Next, we consider the ranking and unranking of set partitions (which are counted by Stirling numbers of the second kind and Bell numbers). The recursion for Stirling numbers involves both addition and multiplication, so our recursive algorithms will use both the bijective sum rule and the bijective product rule.

Let $SP(n,k)$ be the set of all set partitions of $\{1, 2, \ldots, n\}$ into exactly k blocks, and let $S(n,k) = |SP(n,k)|$ be the associated Stirling number of the second kind. Recall from 2.52 that

$$S(n,k) = S(n-1,k-1) + kS(n-1,k) \quad (n,k > 0).$$

The first term counts set partitions in $SP(n,k)$ such that n is in a block by itself; removal of this block gives a bijection onto $SP(n-1,k-1)$. The second term counts set partitions π in $SP(n,k)$ such that n belongs to a block with other elements. Starting with any set partition π' in $SP(n-1,k)$, we can build such a set partition $\pi \in SP(n,k)$ by adding n

to any of the k nonempty blocks of π'. We number the blocks of π' using $0, 1, \ldots, k-1$ by arranging the minimum elements of these blocks in increasing order. For example, if $\pi' = \{\{6,3,5\},\{2\},\{1,7\},\{8,4\}\}$, then block 0 of π' is $\{1,7\}$, block 1 is $\{2\}$, block 2 is $\{3,5,6\}$, and block 3 is $\{4,8\}$.

The ranking maps $r = r_{n,k} : SP(n,k) \to \underline{\mathbf{S(n,k)}}$ are defined recursively as follows. Use the only possible maps if $k \leq 0$ or $k > n$. For $0 < k \leq n$, compute $r_{n,k}(\pi)$ as follows. If $\{n\} \in \pi$, return the answer $r_{n-1,k-1}(\pi \sim \{\{n\}\})$. Otherwise, let π' be obtained from π by deleting n from whatever block contains it, and let i be the index of the block of π' that used to contain n. Return $S(n-1,k-1) + p_{k,S(n-1,k)}(i, r_{n-1,k}(\pi'))$.

Similarly, we define the unranking maps $u = u_{n,k} : \underline{\mathbf{S(n,k)}} \to SP(n,k)$ as follows. Assume $n, k > 0$ and we are computing $u_{n,k}(m)$. If $0 \leq m < S(n-1,k-1)$, then return $u_{n-1,k-1}(m) \cup \{\{n\}\}$. If $S(n-1,k-1) \leq m < S(n,k)$, first compute $(i,j) = p^{-1}_{k,S(n-1,k)}(m - S(n-1,k-1))$. Next, calculate the partition $\pi' = u_{n-1,k}(j)$ by unranking j recursively, and finally compute π by adding n to the ith block of π'.

5.28. Example. Let us compute the rank of $\pi = \{\{1,7\},\{2,4,5\},\{3,8\},\{6\}\}$ relative to the set $SP(8,4)$. In the first stage of the recursion, removal of the largest element 8 from block 2 leaves the set partition $\pi' = \{\{1,7\},\{2,4,5\},\{3\},\{6\}\}$. Therefore,

$$r_{8,4}(\pi) = S(7,3) + 2S(7,4) + r_{7,4}(\pi') = 301 + 2 \cdot 350 + r_{7,4}(\pi').$$

(See Figure 2.21 for a table of Stirling numbers, which were calculated using the recursion for $S(n,k)$.) In the second stage, removing 7 from block 0 leaves the set partition $\pi'' = \{\{1\},\{2,4,5\},\{3\},\{6\}\}$. Hence,

$$r_{7,4}(\pi') = S(6,3) + 0S(6,4) + r_{6,4}(\pi'') = 90 + r_{6,4}(\pi'').$$

In the third stage, removing the block $\{6\}$ leaves the set partition $\pi^{(3)} = \{\{1\},\{2,4,5\},\{3\}\}$, and

$$r_{6,4}(\pi'') = r_{5,3}(\pi^{(3)}).$$

In the fourth stage, removing 5 from block 1 leaves the set partition $\pi^{(4)} = \{\{1\},\{2,4\},\{3\}\}$, and

$$r_{5,3}(\pi^{(3)}) = S(4,2) + 1S(4,3) + r_{4,3}(\pi^{(4)}) = 7 + 6 + r_{4,3}(\pi^{(4)}).$$

In the fifth stage, removing 4 from block 1 leaves the set partition $\pi^{(5)} = \{\{1\},\{2\},\{3\}\}$, and

$$r_{4,3}(\pi^{(4)}) = S(3,2) + 1S(3,3) + r_{3,3}(\pi^{(5)}) = 3 + 1 + r_{3,3}(\pi^{(5)}).$$

But $r_{3,3}(\pi^{(5)})$ is zero, since $|SP(3,3)| = 1$. We deduce in sequence

$$r_{4,3}(\pi^{(4)}) = 4,\ r_{6,4}(\pi'') = r_{5,3}(\pi^{(3)}) = 17,\ r_{7,4}(\pi') = 107,\ r_{8,4}(\pi) = 1108.$$

Next, let us compute $u_{7,3}(111)$. The input 111 weakly exceeds $S(6,2) = 31$, so we must first compute $p^{-1}_{3,90}(111 - 31) = (0, 80)$. This means that 7 will go in block 0 of $u_{6,3}(80)$. Now $80 \geq S(5,2) = 15$, so we compute $p^{-1}_{3,25}(80 - 15) = (2, 15)$. This means that 6 will go in block 2 of $u_{5,3}(15)$. Now $15 \geq S(4,2) = 7$, so we compute $p^{-1}_{3,6}(15 - 7) = (1, 2)$. This means that 5 will go in block 1 of $u_{4,3}(2)$. Now $2 < S(3,2) = 3$, so 4 is in a block by itself in $u_{4,3}(2)$. To find the remaining blocks, we compute $u_{3,2}(2)$. Now $2 \geq S(2,1) = 1$, so we compute $p^{-1}_{2,1}(2 - 1) = (1, 0)$. This means that 3 goes in block 1 of $u_{2,2}(0)$. Evidently, $u_{2,2}(0) = \{\{1\},\{2\}\}$. Using the preceding information to insert elements $3, 4, \ldots, 7$, we conclude that

$$u_{7,3}(111) = \{\{1,7\},\{2,3,5\},\{4,6\}\}.$$

The ranking/unranking procedure given here lists the objects in $SP(n,k)$ in the following order. Set partitions with n in its own block appear first. Next come the set partitions with n in block zero (i.e., n is in the same block as 1); then come the set partitions with n in block one, etc. By applying the sum and product bijections in different orders, one can obtain different listings of the elements of $SP(n,k)$.

Let $SP(n)$ be the set of all set partitions of n, so $|SP(n)|$ is the nth Bell number. The preceding results lead to ranking and unranking algorithms for this collection, by applying the bijective sum rule to the disjoint union

$$SP(n) = SP(n,1) \cup SP(n,2) \cup \cdots \cup SP(n,n).$$

Another approach to ranking $SP(n)$ is to use the recursion in 2.53. Details of this approach are left as an exercise.

5.10 Ranking Card Hands

We now apply the preceding ideas to the problem of ranking and unranking certain poker hands. We will use the bijective sum and product rules to transform the counting arguments from §1.13 into ranking and unranking bijections.

In this section, we define Deck = Suits × Values, where

$$\text{Suits} = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\};$$

$$\text{Values} = \{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}.$$

(A slightly different definition was used in §1.13.) The displayed orderings of the suits and values determine ranking and unranking bijections Suits $\leftrightarrow \underline{\mathbf{4}}$ and Values $\leftrightarrow \underline{\mathbf{13}}$. For example, $r(\diamondsuit) = 1 = r(2)$; 11 unranks to the value Q; and 3 unranks to the suit $\spadesuit$. Combining these maps with the map $p_{4,13}$ from the bijective product rule, we obtain ranking and unranking bijections Deck $\leftrightarrow \underline{\mathbf{52}}$. Since we are thinking of Deck as the product set Suits × Values, the suit of a card is more significant than its value when ranking. With these conventions, the list of cards generated by unranking $0, 1, \ldots, 51$ in this order runs as follows:

$$A\clubsuit, 2\clubsuit, \ldots, K\clubsuit, A\diamondsuit, 2\diamondsuit, \ldots, K\diamondsuit, A\heartsuit, \ldots, K\heartsuit, A\spadesuit, \ldots, Q\spadesuit, K\spadesuit.$$

Naturally, all ranking and unranking results in the examples below depend on this chosen ordering of the deck.

Recall that a poker hand is a five-element subset of Deck. To rank or unrank such hands, we can use the bijections Deck $\leftrightarrow \underline{\mathbf{52}}$ to reduce to the problem of ranking and unranking five-element subsets of $\underline{\mathbf{52}}$. This problem was solved in §5.6. More interesting ranking and unranking problems arise if we restrict attention to certain special kinds of hands, like a full house. We discuss some of these problems next; other examples are treated in the exercises.

5.29. Example: Four-of-a-kind hands. Let S be the set of all four-of-a-kind poker hands. Recall (§1.13) that we can build a hand $H \in S$ (via the ordinary product rule) by picking one of the 13 values $v \in$ Values, and then picking one of the 48 cards in Deck $\sim$ (Suits $\times \{v\}$). The rank of the hand H, relative to this particular construction method for S, is

$$p_{13,48}(r(v), r'(c)),$$

where r : Values $\to \underline{\mathbf{13}}$ is the ranking function for card values, and r' is the ranking function

on Deck $\sim$ (Suits $\times\{v\}$) induced by the usual rank function on Deck. More precisely, $r'(c)$ is the number of cards preceding c in the standard ordering of the deck after throwing out the four cards of value v. If $c = (s_1, v_1)$, one checks that $r'(c) = 12r(s_1) + r(v_1) - \chi(v_1 > v)$.

For example, let us rank the four-of-a-kind hand $H = \{5\spadesuit, 8\heartsuit, 5\diamondsuit, 5\clubsuit, 5\heartsuit\}$. Here, $v = 5$ and $c = 8\heartsuit = (\heartsuit, 8)$. In the full deck, c has rank $13 \cdot 2 + 7 = 33$. But in the deck with the 5's removed, c has rank $r'(c) = 12 \cdot 2 + 6 = 30$. Accordingly, $r(H) = p_{13,48}(4, 30) = 4 \cdot 48 + 30 = 222$.

To illustrate unranking, let us compute $u(600)$. First, $p^{-1}_{13,48}(600) = (12, 24)$. It follows that $v = u(12) = K$. Next, unranking 24 relative to the deck with the four kings deleted gives us the card $c = A\heartsuit$. Therefore, $u(600) = \{K\clubsuit, K\diamondsuit, K\heartsuit, K\spadesuit, A\heartsuit\}$.

5.30. Example: Full house hands. Let S be the set of all full house hands. Recall (§1.13) that we can build a hand $H \in S$ from the data (x, B, y, C), where $x \in$ Values, B is a three-element subset of Suits, $y \in$ Values $\sim \{x\}$, and C is a two-element subset of Suits. For example, the data

$$(x, B, y, C) = (J, \{\clubsuit, \diamondsuit, \spadesuit\}, 9, \{\clubsuit, \heartsuit\})$$

generate the full house hand $H = \{J\clubsuit, J\diamondsuit, J\spadesuit, 9\clubsuit, 9\heartsuit\}$. The rank of this hand is

$$p_{13,4,12,6}(r(x), r(B), r_x(y), r(C)),$$

where we use the same letter r to denote various ranking functions on the set of choices available at each stage. We write $r_x(y)$ to emphasize that the ranking function for $y \in$ Values $\sim \{x\}$ depends on the value of the previous choice x. For the sample choice sequence considered above, we get

$$r(H) = p_{13,4,12,6}(10, 1, 8, 1) = 2880 + 72 + 48 + 1 = 3001.$$

(We are using the ranking functions for k-element subsets of Suits, which were discussed in §5.6.) Observe that the answer depends critically on the precise ordering of the choices we made in the counting argument. If we had chosen the data in the order (x, y, B, C), for example, then we would obtain a different answer for $r(H)$.

To illustrate unranking, let us compute $u(515)$. First,

$$p^{-1}_{13,4,12,6}(515) = (1, 3, 1, 5).$$

Continuing to unrank, $x = u(1) = 2$, $B = u(3) = \{\diamondsuit, \heartsuit, \spadesuit\}$, $y = u_x(1) = 3$ (since the value 2 has been deleted), and $C = u(5) = \{\heartsuit, \spadesuit\}$. So

$$u(515) = \{2\diamondsuit, 2\heartsuit, 2\spadesuit, 3\heartsuit, 3\spadesuit\}.$$

5.31. Example: Two-pair hands. Let S be the set of two-pair poker hands. This time, we build $H \in S$ from data (B, C, D, z), where B is a two-element subset of Values, C is a two-element subset of Suits, D is a two-element subset of Suits, and $z = (s, v)$ is a card such that the value v is not in B. Using the bijective product rule, we have

$$r(H) = p_{78,6,6,44}(r(B), r(C), r(D), r_B(z)).$$

For example, let us find the rank of $H = \{2\spadesuit, 2\clubsuit, 9\clubsuit, 9\heartsuit, K\diamondsuit\}$, which arises from the data

$$(B, C, D, z) = (\{2, 9\}, \{\clubsuit, \spadesuit\}, \{\clubsuit, \heartsuit\}, (\diamondsuit, K)).$$

The ranking formula developed in §5.6 gives $r(B) = \binom{1}{1} + \binom{8}{2} = 29$; similarly, $r(C) = 3$ and $r(D) = 1$. After removing all 2's and 9's from the deck, the new rank of $K\diamondsuit$ is $r_B((\diamondsuit, K)) = 11 + 10 = 21$. So, finally,

$$r(H) = 29 \cdot 6 \cdot 6 \cdot 44 + 3 \cdot 6 \cdot 44 + 1 \cdot 44 + 21 = 46,793.$$

5.32. Example: Ordinary hands. In §1.13, we built an "ordinary" poker hand H by choosing a five-element subset $V(H)$ of Values that avoided one of the ten possible value sets for a straight, and then choosing a word in Suits^5 not all of whose letters are equal (to avoid flushes). This argument showed that there are $(C(13,5)-10)\cdot(4^5-4)=1,302,540$ ordinary poker hands. How can we find a ranking algorithm for this collection of card hands?

Let Y be the set of all five-element subsets of Values, and let $Z=\text{Suits}^5$. We have already found ranking functions $r_Y: Y\to \underline{\mathbf{C(13,5)}}$ and $r_Z: Z\to \underline{\mathbf{4^5}}$. To take the prohibited conditions into account, let $Y'=\{\{A,2,3,4,5\},\{2,3,4,5,6\},\ldots\}$ be the set of ten objects in Y corresponding to straights, and let $Z'=\{sssss : s\in\text{Suits}\}$ be the four objects in Z corresponding to flushes. We can get ranking functions $r'_Y: Y\sim Y'\to \underline{\mathbf{(C(13,5)-10)}}$ and $r'_Z: Z\sim Z'\to \underline{\mathbf{4^5-4}}$ by setting

$$r'_Y(C)=r_Y(C)-|\{C'\in Y' : r_Y(C')<r_Y(C)\}|.$$

This formula is practical since there are only ten possibilities for C', and we can compute the ranks of these objects in advance. They are:

$$0,5,20,55,125,251,461,791,1278,1286.$$

For example, $r(\{3,4,5,6,7\})=C(2,1)+C(3,2)+C(4,3)+C(5,4)+C(6,5)=20$ and $r(\{10,J,Q,K,A\})=C(0,1)+C(9,2)+C(10,3)+C(11,4)+C(12,5)=1278$. Now, the rank function on $Y\sim Y'$ can be computed via the formula

$$r'_Y(C)=\begin{cases} r_Y(C)-1 & \text{if } 0<r_Y(C)<5;\\ r_Y(C)-2 & \text{if } 5<r_Y(C)<20;\\ \cdots & \cdots\\ r_Y(C)-9 & \text{if } 1278<r_Y(C)<1286.\end{cases}$$

Similarly, we can set

$$r'_Z(w)=r_Z(w)-|\{w'\in Z' : r_Z(w')<r_Z(w)\}|.$$

In this case, we precompute

$$\begin{aligned} r(\clubsuit\clubsuit\clubsuit\clubsuit\clubsuit)=p_{4,4,4,4,4}(0,0,0,0,0) &= 0;\\ r(\diamondsuit\diamondsuit\diamondsuit\diamondsuit\diamondsuit)=p_{4,4,4,4,4}(1,1,1,1,1) &= 341;\\ r(\heartsuit\heartsuit\heartsuit\heartsuit\heartsuit)=p_{4,4,4,4,4}(2,2,2,2,2) &= 682;\\ r(\spadesuit\spadesuit\spadesuit\spadesuit\spadesuit)=p_{4,4,4,4,4}(3,3,3,3,3) &= 1023.\end{aligned}$$

Since these numbers form an arithmetic progression, we can write

$$r'_Z(w)=r_Z(w)-\lceil r_Z(w)/341\rceil \qquad (w\in Z\sim Z').$$

Finally, the overall ranking function for a hand H constructed from the pair (C,w) is given by $r(H)=p_{1277,1020}(r'_Y(C),r'_Z(w))$. For example, let us compute the rank of the hand $H=\{A\clubsuit,4\clubsuit,7\heartsuit,9\clubsuit,10\diamondsuit\}$. For this hand, $C=\{A,4,7,9,10\}$ and $w=\clubsuit\clubsuit\heartsuit\clubsuit\diamondsuit$. We calculate

$$r_Y(C)=\binom{0}{1}+\binom{3}{2}+\binom{6}{3}+\binom{8}{4}+\binom{9}{5}=219; \qquad r'_Y(C)=219-5=214;$$

$$r_Z(w)=2\cdot 4^2+1=33; \qquad r'_Z(w)=33-1=32;$$

$$r(H)=p_{1277,1020}(214,32)=214\cdot 1020+32=218,312.$$

As another example, let us unrank $1,000,000$. First, $p^{-1}_{1277,1020}(10^6) = (980, 400)$. The number 980 is between 791 and 1278 in the list of ranks of objects in Y', so we recover $r_Y(C) = 980+8 = 988$. Unranking 988 produces the subset $\{4, 5, 8, 9, 12\}$ of $\underline{\mathbf{13}}$, which translates into the value set $\{5, 6, 9, 10, K\}$. Next, since 400 lies between 341 and 682, we recover $r_Z(w) = 402$. Repeated division by 4 produces the base-4 number 12102, which translates to the value sequence $z = \diamondsuit\heartsuit\diamondsuit\clubsuit\heartsuit$. In conclusion, $u(10^6) = \{5\diamondsuit, 6\heartsuit, 9\diamondsuit, 10\clubsuit, K\heartsuit\}$.

This example shows that applications of the difference rule can be difficult to translate into ranking and unranking algorithms. Our success here depended on the fact that the sizes of the sets being subtracted were quite small, so that their effect on the ranking could be specified by a relatively brief case analysis.

5.33. Remark. In general, the orderings of special card hands obtained above do *not* necessarily arise by restricting the usual ordering of all five-card hands to the given subcollection. Rather, these orderings arise from the particular ordered sequence of choices used to generate these hands. Considerable cleverness may be required to find a ranking algorithm for generating a particular subcollection of card hands in lexicographic order.

5.11 Ranking Dyck Paths

Recall that a *Dyck path* of order n is a lattice path from $(0, 0)$ to (n, n) that never goes below the line $y = x$. These objects are counted by the Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$, which satisfy the recursion

$$C_n = \sum_{k=1}^{n} C_{k-1}C_{n-k} \qquad (n > 0)$$

and initial condition $C_0 = 1$ (see 2.33). Recall that the recursion classifies Dyck paths ending at (n, n) based on the first point (k, k) at which the path returns to the line $y = x$ after leaving the origin (see Figure 2.10). If we use words $w \in \{N, E\}^{2n}$ to encode Dyck paths, the first-return recursion corresponds to the factorization $w = Nw_1Ew_2$, where w_1 encodes a Dyck path of order $k - 1$, and w_2 encodes a Dyck path of order $n - k$.

By applying the bijective sum and product rules to the preceding recursion, we can obtain recursive ranking and unranking algorithms for Dyck paths. For each $n \geq 0$, let D_n be the set of words encoding Dyck paths of order n. Define ranking maps $r_n : D_n \to \underline{\mathbf{C_n}}$ as follows. When $n = 0$, r_0 maps the empty word to the integer 0. For $n > 0$, suppose $w \in D_n$ has first-return factorization $w = Nw_1Ew_2$, where $w_1 \in C_{k-1}$ and $w_2 \in C_{n-k}$ for some k. Recursively compute

$$r_n(w) = \sum_{j=1}^{k-1} C_{j-1}C_{n-j} + p_{C_{k-1},C_{n-k}}(r_{k-1}(w_1), r_{n-k}(w_2)).$$

The unranking maps $u_n : \underline{\mathbf{C_n}} \to D_n$ are defined as follows. First, $u_0(0)$ is the empty word. Given $n > 0$ and $z \in \underline{\mathbf{C_n}}$, find the unique integer $k \leq n$ with

$$\sum_{j<k} C_{j-1}C_{n-j} \leq z < \sum_{j\leq k} C_{j-1}C_{n-j}.$$

Next, compute

$$(x, y) = p^{-1}_{C_{k-1},C_{n-k}}\left(z - \sum_{j<k} C_{j-1}C_{n-j}\right).$$

Recursively determine the words $w_1 = u_{k-1}(x)$ and $w_2 = u_{n-k}(y)$, and return the answer $u_n(z) = Nw_1Ew_2$.

5.34. Example. Let us compute the rank of the Dyck path w = NNENNEEENE. The first-return factorization of w is $w = Nw_1Ew_2$ where w_1 = NENNEE and w_2 = NE. Here, $n = 5$, $k - 1 = 3$, $k = 4$, and $n - k = 1$. The ranking formula gives

$$r_5(w) = C_0C_4 + C_1C_3 + C_2C_2 + p_{C_3,C_1}(r_3(w_1), r_1(w_2)).$$

Now $r_1(w_2) = 0$ since w_2 is the only Dyck path of order 1. As for $r_3(w_1)$, we proceed recursively. The required factorization of w_1 is $w_1 = Nw_3Ew_4$ where w_3 is the empty word and w_4 = NNEE. At this stage of the recursive computation, we have $n = 3$, $k - 1 = 0$, $k = 1$, and $n - k = 2$. Therefore,

$$r_3(w_1) = p_{C_0,C_2}(r_0(w_3), r_2(w_4)).$$

Now $r_0(w_3) = 0$; as for w_4, we have (writing ϵ for the empty word)

$$r_2(w_4) = C_0C_1 + p_{C_1,C_0}(r_1(NE), r_0(\epsilon)) = 1 + p_{1,1}(0, 0) = 1.$$

Recall that the first few Catalan numbers are

$$C_0 = 1,\ C_1 = 1,\ C_2 = 2,\ C_3 = 5,\ C_4 = 14,\ C_5 = 42,\ C_6 = 132,\ C_7 = 429.$$

Working our way back through the recursive calculations, we find that $r_3(w_1) = p_{1,2}(0, 1) = 1$ and

$$r_5(w) = 14 + 5 + 4 + p_{5,1}(1, 0) = 23 + 1 = 24.$$

5.35. Remark. The recursive ranking and unranking calculations can be simplified slightly by precomputing the ranks of Dyck paths of small order, which occur over and over again in the calculations. Observe that our ranking method for D_n lists the Dyck paths in the following order: first, all paths whose first return to $y = x$ is at $(1, 1)$; second, all paths whose first return is at $(2, 2)$; and so on. Within each of these sublists, the ordering of paths is determined recursively (with the aid of the bijective product rule). Using these observations, we can quickly enumerate Dyck paths of order at most 3 in the order implied by the unranking algorithm. For $n = 0$, the list consists of just the empty word. For $n = 1$, we get: NE. For $n = 2$, we get: NENE, NNEE. For $n = 3$, we get:

NENENE, NENNEE, NNEENE, NNENEE, NNNEEE.

5.36. Example. Let us unrank 211 to obtain a Dyck path of order $n = 7$. From the recursion, we have

$$429 = C_7 = 1 \cdot 132 + 1 \cdot 42 + 2 \cdot 14 + 5 \cdot 5 + 14 \cdot 2 + 42 \cdot 1 + 132 \cdot 1.$$

The first step in unranking is to calculate the partial sums on the right side until we find the first one larger than the given input 211. We find that

$$C_0C_6 + C_1C_5 + C_2C_4 = 202 \le 211 < 227 = C_0C_6 + C_1C_5 + C_2C_4 + C_3C_3.$$

Therefore $k = 4$, $k-1 = 3 = n-k$, and $(x, y) = p_{5,5}^{-1}(211 - 202) = (1, 4)$. Using the previous example, we have $w_1 = u_3(x)$ = NENNEE and $w_2 = u_3(y)$ = NNNEEE. It follows that

$$u_7(211) = \text{N NENNEE E NNNEEE.}$$

5.12 Ranking Trees

We know from §3.7 that there are n^{n-2} rooted trees on the vertex set $\{1, 2, \ldots, n\}$ rooted at vertex 1. Let B be the set of such trees; we seek ranking and unranking algorithms for B. One way to obtain these algorithms is to use the bijective proof of 3.47. In that proof, we described a bijection $\phi' : B \to A$, where A is the set of all functions $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that $f(1) = 1$ and $f(n) = n$. Let C be the set of words of length $n-2$ in the alphabet $\{0, 1, \ldots, n-1\}$. The map $\psi : A \to C$ such that $\psi(f) = w_1 \cdots w_{n-2}$ with $w_i = f(i+1)-1$, is evidently a bijection. Furthermore, the map $p_{n,n,\ldots,n}$ used in the bijective product rule gives a bijection from C to $\underline{\mathbf{n^{n-2}}}$. Composing all these bijections, we get the desired ranking algorithm. Inverting the bijections gives an unranking algorithm.

5.37. Example. Consider the rooted tree T shown in Figure 3.9. In 3.49, we computed $\phi'(T)$ to be the function g defined by

$$g : 1 \mapsto 1,\ 2 \mapsto 2,\ 3 \mapsto 2,\ 4 \mapsto 9,\ 5 \mapsto 9,\ 6 \mapsto 7,\ 7 \mapsto 6,\ 8 \mapsto 9,\ 9 \mapsto 9.$$

Here, $\psi(g)$ is the word 1188658. Applying the map $p_{9,9,\ldots,9}$ (or equivalently, interpreting the given word as a number written in base 9), we find that $r(T) = 649,349$.

This application shows how valuable a bijective proof of a counting result can be. If we have a bijection from a complicated set of objects to a "nice" set of objects (such as functions or words), we can compose the bijection with standard ranking maps to obtain ranking and unranking algorithms for the complicated objects. In contrast, if a counting result is obtained by some intricate algebraic manipulation, it may not be so straightforward to extract an effective ranking mechanism.

5.13 Successors and Predecessors

Suppose S is a finite set of n objects, and we wish to list all the elements of S in a certain order. If we know an appropriate unranking bijection $u : \underline{\mathbf{n}} \to S$, then we can generate the desired list by computing $u(0), u(1), \ldots, u(n-1)$ in succession. However, if the unranking map u is complicated, this method of listing S may not be very efficient.

In many applications, if we know the object $z = u(i)$ that occupies a particular position on the list, it may be possible to compute the object that immediately precedes or follows z on the list, without ever explicitly computing i or applying the algorithm defining u to the inputs $i-1$ or $i+1$. We call $u(i-1)$ the *predecessor* of z (relative to the listing determined by u), and we call $u(i+1)$ the *successor* of z. Reversing the ordering of the elements of S interchanges predecessors and successors; so, in what follows, we need only consider successors.

The *successor problem* asks for an efficient algorithm for finding the successor of a given object z relative to a given ordering. We could solve this problem by ranking z, adding 1, and unranking, but we typically want more elegant solutions. If we can solve the successor problem, and if we know what the first object on the list is, then we will have a new method for listing all the elements of S. Namely, we start at the first object and then repeatedly invoke the successor algorithm until the last object is reached. In computer programming, one often uses this general strategy to loop through all elements of some set of combinatorial objects.

As a typical example, we develop a successor algorithm for listing the words in $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ in alphabetical order. This example includes as special cases the problem of listing all k-element subsets of an n-element set (which can be encoded as words in $\mathcal{R}(0^{n-k}1^k)$) and the problem of listing all permutations of k letters (take each $n_k = 1$). At the outset, we fix an ordered alphabet $A = \{a_1 < \cdots < a_k\}$. Our algorithm will consist of three functions, called "first," "last," and "next." The "first" and "last" functions return the first and last words in $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ relative to the alphabetical ordering; explicitly, we have

$$\text{first}(n_1, \ldots, n_k) = a_1^{n_1} a_2^{n_2} \cdots a_k^{n_k};$$

$$\text{last}(n_1, \ldots, n_k) = a_k^{n_k} a_{k-1}^{n_{k-1}} \cdots a_1^{n_1}.$$

In the case $n_1 + \cdots + n_k = 0$, both functions return the empty word.

The successor function "next" takes as input the integers $n_1, \ldots, n_k \geq 0$, as well as a word $z \in \mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$. The output of "next" is either the successor z' of z in the alphabetical ordering, or a special flag (called "done") indicating that z was the last word on the list. The operation of "next" is based on the observation that the alphabetical list of words consists of all words starting with a_1 (if $n_1 > 0$), then all words starting with a_2 (if $n_2 > 0$), and so on. Within each of these sublists, the words are ordered in the same way based on their second letters, and so on recursively. So we can define $\text{next}(n_1, \ldots, n_k, z)$ using the following recursive algorithm.

- Base Case: If $n_1 + \cdots + n_k \leq 0$ or if $z = \text{last}(n_1, \ldots, n_k)$, return "done."
- Recursion: Suppose $z = a_i z'$. Recursively compute $w' = \text{next}(n_1, \ldots, n_i - 1, \ldots, n_k, z')$. If w' is not the special "done" flag, return $a_i w'$ as the answer. Otherwise, find the smallest index $j > i$ with $n_j > 0$. If no such index exists, return "done"; else return the concatenation of a_j and $\text{first}(n_1, \ldots, n_j - 1, \ldots, n_k)$.

5.38. Example. Consider $z = \text{bdcaccb} \in \mathcal{R}(a^1b^2c^3d^1)$. To compute the successor of z, we are first directed to find the successor of $z' = \text{dcaccb} \in \mathcal{R}(a^1b^1c^3d^1)$. Continuing recursively, we are led to consider the words caccb, then accb, then ccb. But ccb is the last word in $\mathcal{R}(a^0b^1c^2d^0)$, so $\text{next}(0, 1, 2, 0, \text{ccb})$ returns "done." Returning to the calculation of $\text{next}(1, 1, 2, 0, \text{accb})$, we seek the next available letter after 'a', which is 'b'. We concatenate 'b' and $\text{first}(1, 0, 2, 0) = \text{acc}$ to obtain $\text{next}(1, 1, 2, 0, \text{accb}) = \text{bacc}$. Then $\text{next}(1, 1, 3, 0, \text{caccb}) = \text{cbacc}$. Continuing similarly, we eventually obtain the final output "bdcbacc" as the successor of z.

If we apply the "next" function to this new word, we strip off initial letters one at a time until we reach the suffix "cc," which is the last word in its class. So, to compute $\text{next}(1, 0, 2, 0, \text{acc})$, we must find the next *available* letter after 'a', namely 'c', and append to this the word $\text{first}(1, 0, 1, 0) = \text{ac}$. Thus, $\text{next}(1, 0, 2, 0, \text{acc}) = \text{cac}$, and working back up the recursive calls leads to a final answer of "bdcbcac." This example shows why we need to remember the values $n_1, \ldots, n_k$ in each recursive call to "next."

5.39. Example. If we use the given method to list the permutations of $\{1, 2, 3, 4\}$, we obtain:

$$1234,\ 1243,\ 1324,\ 1342,\ 1423,\ 1432,\ 2134,\ 2143,\ \ldots,\ 4321.$$

5.40. Example. Using the encoding of subsets as binary words (see 1.38), we can list all 2-element subsets of $\{1, 2, 3, 4, 5\}$ by running through the words in $\mathcal{R}(1^20^3)$. (For convenience, we choose the alphabet ordering $1 < 0$ here.) The words are:

$$11000,\ 10100,\ 10010,\ 10001,\ 01100,\ 01010,\ 01001,\ 00110,\ 00101,\ 00011.$$

The associated subsets are:

$$\{1,2\},\ \{1,3\},\ \{1,4\},\ \{1,5\},\ \{2,3\},\ \{2,4\},\ \{2,5\},\ \{3,4\},\ \{3,5\},\ \{4,5\}.$$

In general, the method used here lists k-element subsets of $\{1,2,\ldots,n\}$ in lexicographic order. Using the ordering of the letters $0 < 1$ would have produced the reversal of the list displayed above. In contrast, the ranking method discussed in §5.6 lists the subsets according to a different ordering, in which all subsets not containing n are listed first, followed by all subsets that do contain n, and so on recursively. The latter method produces the following list of subsets:

$$\{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,4\},\ \{2,4\},\ \{3,4\},\ \{1,5\},\ \{2,5\},\ \{3,5\},\ \{4,5\}.$$

5.41. Example: Successor Algorithm for Dyck Paths. Let us design a successor algorithm for generating Dyck paths of order n, based on the "first-return" recursion (§2.7). As above, we will use three routines called "first," "last," and "next." The routine first(n) returns the path $(NE)^n$, which returns to the diagonal as early as possible, while the routine last(n) returns the path N^nE^n, which returns to the diagonal as late as possible. If $n \geq 0$ and w encodes a Dyck path of order n, then next(n,w) will return the next Dyck path in the chosen ordering, or "done" if w is the last path. This routine works as follows:

- Base Case: If $n \leq 1$ or $w = \text{last}(n)$, then return "done."
- Recursion: Find the first-return factorization $w = Nw_1Ew_2$ of w, where $w_1 \in D_{k-1}$ and $w_2 \in D_{n-k}$ for some k between 1 and n. Now consider subcases.
  - Calculate $w_2' = \text{next}(n-k, w_2)$. If this path exists, return Nw_1Ew_2' as the answer.
  - Otherwise, find $w_1' = \text{next}(k-1, w_1)$. If this path exists, return $N\ w_1'\ E\ \text{first}(n-k)$ as the answer.
  - Otherwise, increase k by 1. If the new k is $\leq n$, return $N\ \text{first}(k-1)\ E\ \text{first}(n-k)$ as the answer.
  - Otherwise, return "done."

For example, let us compute next(7,N NENNEE E NNNEEE). The given input w factorizes as $w = Nw_1Ew_2$ where $w_1 =$ NENNEE and $w_2 =$ NNNEEE. Here, $k = 4$, $k-1 = 3$, and $n-k = 3$. By inspection, $w_2 = \text{last}(3)$, so we proceed to the next subcase. We are directed to compute $w_1' = \text{next}(3, w_1)$. Here, w_1 factors as $w_1 = Nw_3Ew_4$, where w_3 is the empty word and $w_4 =$ NNEE. Again, w_4 is the last Dyck path of order 2, and this time w_3 is also the last Dyck path of order 0. So we increase k by 1, and return

$$w_1' = \text{N first(1) E first(1)=N NE E NE.}$$

Using this result in the original calculation, we obtain

$$\text{next}(w) = \text{N } w_1' \text{ E first(3) = N NNEENE E NENENE.}$$

5.14 Random Selection

We now briefly revisit the problem of random selection of combinatorial objects. Suppose S is a set of n objects, and we wish to randomly select an element of S. If we have an

unranking function $u : \underline{\mathbf{n}} \to S$, we can select the object by generating a random integer in $\underline{\mathbf{n}}$ and unranking it. However, it may be impractical to generate such an integer if n is very large. If the objects in S are generated by a recursive process, we can often get around this problem by making several random choices that are used to build an object in S in stages.

5.42. Example: Subsets. As a typical example, consider the question of randomly choosing a k-element subset of $A = \{x_1, \ldots, x_n\}$. Earlier (§5.6), we developed ranking and unranking algorithms for this problem based on the recursion

$$C(n,k) = C(n-1,k) + C(n-1,k-1). \tag{5.2}$$

A slight adjustment of the methods used before will lead to an efficient solution for the random selection problem.

Recall that the term $C(n-1,k)$ in the recursion counts the k-element subsets of A that do not contain x_n, while the term $C(n-1,k-1)$ counts subsets that do contain x_n. If we choose a random k-element subset of A, the probability that it will not contain x_n is $C(n-1,k)/C(n,k) = (n-k)/n$, and the probability that it will contain x_n is $C(n-1,k-1)/C(n,k) = k/n$. This suggests the following recursively defined random selection procedure. To choose a k-element subset A of $\{x_1, \ldots, x_n\}$, use a random number generator to obtain a real number $r \in [0,1]$. If $r \leq k/n$, declare that x_n belongs to A, and recursively select a random $(k-1)$-element subset of $\{x_1, \ldots, x_{n-1}\}$ by the same method. If $r > k/n$, declare that x_n does not belong to A, and recursively select a random k-element subset of $\{x_1, \ldots, x_{n-1}\}$. The base cases occur when $k = 0$ or $k = n$, in which case there is only one possible subset to select.

5.43. Example: Set Partitions. A similar method can be used to randomly select a k-element set partition of an n-element set $A = \{x_1, \ldots, x_n\}$. Recall that these objects are counted by the Stirling numbers $S(n,k)$, which satisfy the recursion

$$S(n,k) = S(n-1,k-1) + kS(n-1,k) \qquad (n,k > 0).$$

Recall that the first term on the right counts set partitions with x_n in a block by itself, while the second term counts set partitions in which x_n occurs in a block with other elements. The base cases of the selection procedure occur when $k = 0$ or $k = n$; here, there is at most one possible object to select. For the main case, assume $n > 0$ and $0 < k < n$. Choose a random real number $r \in [0,1]$. Consider the quantity $r_0 = S(n-1,k-1)/S(n,k)$, which can be computed (at least approximately) using the Stirling recursion. Note that r_0 is the probability that a random set partition of A into k blocks will have x_n in a block by itself. If $r \leq r_0$, recursively select a set partition of $\{x_1, \ldots, x_{n-1}\}$ into $k-1$ blocks, and append the block $\{x_n\}$ to this set partition to obtain the answer. In the alternative case $r > r_0$, recursively select a set partition $\{B_1, \ldots, B_k\}$ of $\{x_1, \ldots, x_{n-1}\}$ into k blocks. Now, choose a random integer i in the range $\{1, \ldots, k\}$, and insert n into block B_i to obtain the answer. Such an integer i can be found, for example, by choosing another random real number $s \in [0,1]$, multiplying by k, and rounding up to the nearest integer.

5.44. Example: Permutations. As a final example, consider the problem of randomly generating a permutation of $\{1, 2, \ldots, n\}$. We can convert the standard counting argument (based on the product rule) into a random selection procedure. However, some care is required, since the available choices at each stage depend on what choices were made in previous stages.

Recall that one method for building a permutation $w = w_1 w_2 \cdots w_n$ of $\{1, 2, \ldots, n\}$ is to choose w_1 to be any letter (in n ways), then choosing w_2 to be any letter other than w_1 (in $n-1$ ways), then choosing w_3 to be any letter other than w_1 or w_2 (in $n-2$ ways),

and so on. The associated random generation algorithm would operate as follows. Generate random integers $i_1 \in \{1, 2, \ldots, n\}$, $i_2 \in \{1, 2, \ldots, n-1\}$, ..., $i_n \in \{1\}$. Define $w_1 = i_1$. Define w_2 to be the i_2th smallest element in the set $\{1, 2, \ldots, n\} \sim \{w_1\}$. In general, define w_j to be the i_jth smallest element in the set $\{1, 2, \ldots, n\} \sim \{w_1, \ldots, w_{j-1}\}$.

This computation can become rather messy, since we must repeatedly scan through the remaining letters to determine the i_jth smallest one. Furthermore, the method is really no different from unranking a random integer between 0 and $n! - 1$. An alternative recursive generation method proceeds as follows. If $n = 1$, return $w = 1$ as the answer. If $n > 1$, first recursively generate a random permutation $w' = w'_1 \cdots w'_{n-1}$ of $\{1, 2, \ldots, n-1\}$. Next, generate a random integer $w_n \in \{1, 2, \ldots, n\}$. Now make a single scan through the previously chosen letters w'_i, and let $w_i = w'_i$ if $w'_i < w_n$, $w_i = w'_i + 1$ if $w'_i \geq w_n$. This determines the final answer $w = w_1 w_2 \cdots w_n$. We let the reader check that every permutation is equally likely to be generated when using this method.

Summary

- *Definitions.* Let S be a set of n objects. A *ranking map* for S is a bijection $r : S \to \underline{\mathbf{n}}$, where $\underline{\mathbf{n}} = \{0, 1, 2, \ldots, n-1\}$. An *unranking map* for S is a bijection $u : \underline{\mathbf{n}} \to S$. Given a particular total ordering of the elements of S, a *successor map* for S is a function that maps each $z \in S$ to the element immediately following z in the ordering (if any).

- *Bijective Sum Rule.* Let $S_1, \ldots, S_k$ be disjoint finite sets with union S. Given bijections $f_i : S_i \to \underline{\mathbf{n_i}}$, there is a bijection $f = \sum_i f_i : S \to \underline{\mathbf{n}}$ (where $n = n_1 + \cdots + n_k$) given by

$$f(x) = \sum_{j<i} n_j + f_i(x) \qquad (x \in S_i).$$

  The map f depends on the given ordering of the f_i's. To compute $f^{-1}(z)$, find the unique index i such that $\sum_{j<i} n_j \leq z < \sum_{j \leq i} n_j$, and let $f^{-1}(z) = f_i^{-1}(z - \sum_{j<i} n_j)$.

- *Bijective Product Rule.* Given positive integers $n_1, \ldots, n_k$ with product n, there is a bijection $p = p_{n_1, n_2, \ldots, n_k} : \underline{\mathbf{n_1}} \times \underline{\mathbf{n_2}} \times \cdots \times \underline{\mathbf{n_k}} \to \underline{\mathbf{n}}$ given by

$$p(c_1, c_2, \ldots, c_k) = c_1 n_2 \cdots n_k + c_2 n_3 \cdots n_k + \cdots + c_{k-1} n_k + c_k \qquad (0 \leq c_i < n_i).$$

  To compute $p^{-1}(z)$, let q_k and r_k be the quotient and remainder when z is divided by n_k. Then let q_{k-1} and r_{k-1} be the quotient and remainder when q_k is divided by n_{k-1}. Continue similarly; then $p^{-1}(z) = (r_1, r_2, \ldots, r_k)$. If all n_i's equal the same integer b, $p^{-1}(z)$ is the base-b expansion of z.

- *Ranking k-Permutations.* Suppose we are ranking k-permutations $w = w_1 w_2 \cdots w_k$ of an ordered alphabet $A = (x_0, x_1, \ldots, x_{n-1})$. To find $r(w)$, first compute $(j_1, \ldots, j_k)$ by letting j_i be the number of letters preceding w_i in the given ordering of A that are different from $w_1, \ldots, w_{i-1}$. Then calculate $r(w) = p_{n, n-1, \ldots, n-k+1}(j_1, \ldots, j_k)$. To unrank z, apply p^{-1} to recover $(j_1, \ldots, j_k)$, and then recover $w_1, \ldots, w_k$ from left to right by letting w_i be the $(j_i + 1)$th smallest letter in $A \sim \{w_1, \ldots, w_{i-1}\}$.

- *Rank Formula for Subsets.* Suppose we are ranking k-element subsets of an ordered alphabet $A = (x_0, x_1, \ldots, x_{n-1})$. If $B = \{x_{i_1} < x_{i_2} < \cdots < x_{i_k}\}$, then we can take $r(B) = \sum_{j=1}^{k} \binom{i_j}{j}$. We can unrank a given integer z by a greedy strategy that recovers

$i_k, \ldots, i_1$ (in this order) by choosing the largest possible value that will not cause the partial sum so far to exceed z. This ranking method leads to a listing of k-element subsets in which all subsets containing x_{n-1} appear after all subsets not containing x_{n-1}; each sublist is ordered in the same way relative to x_{n-2}, etc.

- *Rank Formula for Anagrams.* The following recursive formula can be used to rank words $w \in \mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ in alphabetical order: if $w = a_i w'$, then

$$r(w) = \sum_{j<i} C(n_1 + \cdots + n_k - 1; n_1, \ldots, n_j - 1, \ldots, n_k) + r(w').$$

  To unrank a given integer z, choose i as large as possible so that the sum in the previous formula does not exceed z; subtract the sum for this choice of i from z; unrank the result recursively; and prepend the letter a_i to obtain the final answer.

- *Successor Algorithm for Anagrams.* Consider all words in $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$ in alphabetical order. To find the word immediately following w, first write $w = a_i w'$. If w' is not the last word in its rearrangement class, recursively compute its successor (say z'), and return $a_i z'$ as the successor of w. Otherwise, find the first $j > i$ with $n_j > 0$, and return $a_j b'$ as the successor of w, where $b' = a_1^{n_1} \cdots a_j^{n_j - 1} \cdots a_k^{n_k}$.

- *Random Selection Algorithms.* Suppose we want to randomly select an object from a given set S. If an unranking map $u : \underline{\mathbf{n}} \to S$ is available, we can generate a random integer $z \in \underline{\mathbf{n}}$ and return $u(z)$. Alternatively, if the objects in S can be built up in stages, we can make a random choice at each stage to decide how to build the object. For instance, to build a random k-subset B of $\{1, 2, \ldots, n\}$, we can include n in B with probability k/n, and then choose the remaining elements of B recursively.

Exercises

5.45. Suppose $f : \{a, b, c\} \to \underline{\mathbf{3}}$ and $g : \{d, e\} \to \underline{\mathbf{2}}$ are defined by $f(a) = 1$, $f(b) = 2$, $f(c) = 0$, $g(d) = 1$, $g(e) = 0$. Compute the bijections $f + g$ and $g + f$.

5.46. Compute (a) $p_{7,5}(4, 3)$; (b) $p_{7,5}(3, 4)$; (c) $p_{5,7}(4, 3)$; (d) $p_{5,7}(3, 4)$; (e) $p_{7,5}^{-1}(22)$; (f) $p_{5,7}^{-1}(22)$.

5.47. Find (a) $p_{2,2,2,2,2}(0, 1, 1, 0, 1)$; (b) $p_{2,2,2,2,2}^{-1}(29)$; (c) $p_{7,7,7}(3, 0, 6)$; (d) $p_{7,7,7}^{-1}(306)$; (e) $p_{10,10,10}^{-1}(306)$.

5.48. Compute: (a) $p_{5,4,3,2,1}(3, 3, 0, 1, 0)$; (b) $p_{5,4,3,2,1}^{-1}(111)$; (c) $p_{3,6,2,6}(2, 5, 0, 4)$; (d) $p_{3,6,2,6}^{-1}(150)$; (e) $p_{6,2,6,3}^{-1}(150)$; (f) $p_{6,6,3,2}^{-1}(150)$.

5.49. Consider the product set $X = \underline{\mathbf{3}} \times \underline{\mathbf{4}}$. (a) View X as the disjoint union of the sets $X_i = \{i\} \times \underline{\mathbf{4}}$, for $i = 0, 1, 2$. Let $f_i : X_i \to \underline{\mathbf{4}}$ be the bijection $f_i(i, y) = y$. Compute the bijections $f_0 + f_1 + f_2$ and $f_2 + f_1 + f_0$, which map X to $\underline{\mathbf{12}}$. (b) View X as the disjoint union of the sets $X^{(j)} = \underline{\mathbf{3}} \times \{j\}$, for $j = 0, 1, 2, 3$. Let $g_j : X^{(j)} \to \underline{\mathbf{3}}$ be the bijection $g_j(x, j) = x$. Compute the bijection $g_0 + g_1 + g_2 + g_3 : X \to \underline{\mathbf{12}}$. (c) Compute the bijection $p_{3,4} : X \to \underline{\mathbf{12}}$. Is this one of the maps found in (a) or (b)? (d) Let $t : X \to \underline{\mathbf{4}} \times \underline{\mathbf{3}}$ be the bijection $t(i, j) = (j, i)$. Compute the bijection $p_{4,3} \circ t : X \to \underline{\mathbf{12}}$. Is this one of the maps found in (a) or (b)?

5.50. Rank the following four-letter words: (a) alto; (b) zone; (c) rank; (d) four; (e) word.

5.51. Unrank the following numbers in $\underline{\mathbf{26^4}}$ to obtain four-letter words: (a) 115,287; (b) 396,588; (c) 392,581; (d) 338,902; (e) 275,497.

5.52. (a) Rank the six-letter word "unrank." (b) Unrank 199,247,301 to get a 6-letter word. (c) What happens if we unrank 199,247,301 to get a k-letter word where $k > 6$?

5.53. A fraternity name consists of either two or three capital Greek letters. Recall that there are 24 letters in the Greek alphabet, ordered as follows:

$$\text{ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ}.$$

Assume an ordering of fraternity names consisting of all two-letter names in alphabetical order, followed by all three-letter names in alphabetical order. Compute the rank of (a) ΦΒΚ; (b) ΔΔ; (c) ΔΔΔ; (d) ΑΧΩ. Now, unrank: (e) 144; (f) 1440; (g) 13931.

5.54. Repeat (a)–(g) in 5.53, assuming the names are ordered so that all three-letter names precede all two-letter names, with names of each length in alphabetical order.

5.55. Repeat (a)–(g) in 5.53, assuming the names are ordered in alphabetical order (so that, for example, ΔΔ is immediately preceded by ΔΓΩ and immediately followed by ΔΔΑ).

5.56. Consider the set of four-digit even numbers (no leading zeroes allowed) that do not contain the digit 6. (a) Use the product rule to count this set. (b) Find a ranking bijection that will list these numbers in increasing numerical order. (c) Use (b) to rank 1234, 2500, and 9708. (d) Now unrank 1234, 2501, and 666.

5.57. Consider five-letter palindromes, ranked in alphabetical order. (a) Rank the palindromes LEVEL and MADAM. (b) Unrank 1581 and 12,662. (c) Find the first and last palindromes in the ranking that are real English words.

5.58. A Virginia license plate consists of three uppercase letters followed by four digits. For arcane bureaucratic reasons, license plate 0 is ZZZ-9999, followed by ZZZ-9998, ..., ZZZ-0000, ZZY-9999, etc. Use this system to rank the license plates: (a) ZCF-2073; (b) JXB-2007; (c) ABC-1234. Now unrank: (d) 7,777,777; (e) 123,456,789.

5.59. Repeat the previous exercise assuming a new ordering, honoring the 400th anniversary of Jamestown, where license plate 0 is JAM-1607, and license plates count forward in lexicographic order ("wrapping around" from ZZZ-9999 to AAA-0000).

5.60. Let $A = \{a, b, c, d, e, f\}$. (a) Compute the ranks of bfdc and fdac among all 4-permutations of A. (b) Unrank 232 to get a 4-permutation of A. (c) Compute the rank of ecafdb among all permutations of A. (d) Unrank 583 to get a permutation of A.

5.61. (a) Compute the rank of 42153 among all permutations of $\{1, 2, \ldots, 5\}$. (b) Unrank 46 to obtain a permutation of $\{1, 2, \ldots, 5\}$.

5.62. (a) Compute the rank of 36281745 among all permutations of $\{1, 2, \ldots, 8\}$. (b) Unrank 23,419 to obtain a permutation of $\{1, 2, \ldots, 8\}$.

5.63. Let $A = \{a, b, c, d, e, f, g, h\}$. (a) Use the ranking formula for $S_4(A)$ in §5.6 to rank the subsets $\{a, c, e, g\}$, $\{b, c, d, h\}$, and $\{d, e, f, h\}$. (b) Unrank 30, 40, and 50 to obtain 4-element subsets of A.

5.64. (a) Devise a ranking algorithm for k-element subsets of an n-element alphabet based on the recursion $C(n,k) = C(n-1,k-1)+C(n-1,k)$, which differs from the recursion in §5.6 due to the reversal of the order of terms on the right side. (b) Describe informally the order in which the ranking algorithm in (a) will produce the k-element subsets. (c) Answer the ranking and unranking questions in the previous exercise using this new ranking algorithm.

5.65. (a) Find the ranks of bbccacba and cabcabbc in the set $\mathcal{R}(a^2b^3c^3)$, ordered alphabetically. (b) Unrank 206 and 497 to get anagrams in $\mathcal{R}(a^2b^3c^3)$.

5.66. (a) Compute the rank of MISSISSIPPI among the set of all anagrams in $\mathcal{R}(I^4MP^2S^4)$ (listed alphabetically). (b) Which anagram in this set has rank 33,333?

5.67. (a) Use the rank functions $r_{n,k}$ in §5.8 to rank the integer partitions $(3,3,3)$, $(5,2,2)$, and $(4,3,2,1)$. (b) Compute $u_{12,3}(6)$, $u_{15,4}(22)$, and $u_{20,6}(47)$.

5.68. Use the rank function $r_{8,4}$ from §5.8 to list all integer partitions of 8 into 4 parts.

5.69. Enumerate all the integer partitions of 7, following the method used in 5.27.

5.70. Use the rank functions from §5.9 to list all set partitions of $\{1,2,3,4,5\}$ into 3 blocks.

5.71. (a) Use the algorithms in §5.9 to rank the following set partitions relative to the set $SP(n,k)$: $\{\{1,3\},\{2,4,5\}\}$; $\{\{1,5,7\},\{2\},\{3,4,8\},\{6\}\}$. (b) Unrank 247 to obtain a set partition in $SP(7,4)$. (c) Unrank 1492 to obtain a set partition in $SP(8,4)$.

5.72. (a) Rank the four-of-a-kind hand $\{3\clubsuit, 8\heartsuit, 8\diamondsuit, 8\spadesuit, 8\clubsuit\}$ (see 5.29). (b) Unrank 264 to get a four-of-a-kind hand.

5.73. (a) Rank the full house hand $\{3\clubsuit, 3\heartsuit, 3\diamondsuit, 9\spadesuit, 9\diamondsuit\}$ (see 5.30). (b) Unrank 3082 to get a full house hand.

5.74. (a) Rank the full house hand $\{A\spadesuit, A\heartsuit, A\diamondsuit, K\heartsuit, K\clubsuit\}$ (see 5.30). (b) Unrank 483 to get a full house hand.

5.75. (a) Rank the two-pair hand $\{A\diamondsuit, A\heartsuit, 7\spadesuit, Q\diamondsuit, Q\spadesuit\}$ (see 5.31). (b) Unrank 71,031 to get a two-pair hand.

5.76. (a) Rank the hand $\{5\clubsuit, 7\diamondsuit, 8\spadesuit, 10\heartsuit, J\heartsuit\}$ among all possible poker hands. (b) Rank the same hand among all "ordinary" poker hands (see 5.32). (c) Unrank 1,159,403 to get one of the $\binom{52}{5}$ possible poker hands. (d) Unrank 1,159,403 to get an ordinary poker hand.

5.77. Use the ranking algorithm in §5.11 to list all Dyck paths of order (a) 4; (b) 5.

5.78. (a) Rank the Dyck path NNNENEENNEENNENEEE (see §5.11). (b) Unrank 52 to get a Dyck path of order 6. (c) Unrank 335 to get a Dyck path of order 7.

5.79. (a) Use the method of §5.12 to find the rank of the rooted tree

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

(b) Unrank 1,609,765 to obtain a rooted tree on 9 vertices rooted at vertex 1.

5.80. Use the algorithm in §5.13 to find the successor of each word in the appropriate set of anagrams: (a) ccbabdc; (b) 3641275; (c) 01101011; (d) 33212312; (e) UKULELE.

5.81. Write an algorithm to find the predecessor of a given word $w \in \mathcal{R}(a_1^{n_1}\cdots a_k^{n_k})$ in the alphabetical ordering of anagrams. Use this to find the predecessor of each word in the previous exercise.

5.82. (a) Use the successor algorithm in §5.13 to find the first four successors of the Dyck path NNNENEENNEENNENEEE. (b) Write a predecessor algorithm for Dyck paths. (c) Use (b) to compute the first four predecessors of the Dyck path in (a).

5.83. Give careful proofs of the bijective sum rules (5.1 and 5.2).

5.84. Prove that for all $a \in \mathbb{Z}$ and all nonzero $b \in \mathbb{Z}$, there exist unique $q, r \in \mathbb{Z}$ with $a = bq + r$ and $0 \leq r < |b|$. Describe an algorithm for computing q and r given a and b.

5.85. Prove that for all $a \in \mathbb{Z}$ and all nonzero $b \in \mathbb{Z}$, there exist unique $q, r \in \mathbb{Z}$ with $a = bq + r$ and $-|b|/2 < r \leq |b|/2$. Describe an algorithm for computing q and r given a and b.

5.86. Suppose $0 \neq b \in \mathbb{Z}$ and $S \subseteq \mathbb{Z}$. Find a necessary and sufficient condition on S that will make the following statement true: for all $a \in \mathbb{Z}$, there exist unique $q, r \in \mathbb{Z}$ with $a = bq + r$ and $r \in S$. How could one find q and r given a, b, and S?

5.87. Division Algorithm for Polynomials. (a) Suppose F is a field, $g \in F[x]$ is a nonzero polynomial, and $f \in F[x]$ is any polynomial. Prove there exist unique polynomials $q, r \in F[x]$ with $f = qg + r$ and either $r = 0$ or $\deg(r) < \deg(g)$. Describe an algorithm for computing q and r given f and g. (b) Show that (a) can fail if F is a commutative ring that is not a field. (c) Is (a) true for all commutative rings F if we assume g is monic?

5.88. Verify 5.14.

5.89. Given $a, b \in \mathbb{Z}$ with $b > 0$, let $a \bmod b$ be the unique remainder $r \in \underline{\mathbf{b}}$ such that $a = bq + r$ for some $q \in \mathbb{Z}$. (a) Given $s, t > 0$, consider the map $f : \underline{\mathbf{st}} \to \underline{\mathbf{s}} \times \underline{\mathbf{t}}$ given by $f(x) = (x \bmod s, x \bmod t)$ for $x \in \underline{\mathbf{st}}$. Prove that f is a bijection iff $\gcd(s, t) = 1$. (b) Generalize (a) to maps from $\underline{\mathbf{s_1 s_2 \cdots s_k}}$ to $\underline{\mathbf{s_1}} \times \underline{\mathbf{s_2}} \times \cdots \times \underline{\mathbf{s_k}}$.

5.90. Complexity of Binary Arithmetic. Let x and y be k-bit numbers (this means the base-2 expansions of x and y have zeroes beyond the first k positions). (a) Show that there is an algorithm to compute the base-2 expansion of $x + y$ (or $x - y$) that requires at most ck bit operations, for some constant c. (b) Show that the base-2 expansion of xy can be computed in at most ck^2 bit operations, for some constant c. (See 7.175 for a faster method.) (c) If $y > 0$, there exist unique $q, r \in \mathbb{Z}$ with $x = qy + r$ and $0 \leq r < y$. Show that q and r can be computed from x and y in at most ck^2 bit operations, for some constant c.

5.91. Binary Exponentiation. Suppose n is a k-bit number, $x \in \underline{\mathbf{n}}$, and e is an m-bit number. Show that we can compute $x^e \bmod n$ in at most ck^2m bit operations, for some constant c.

5.92. Euclid's Algorithm for Computing GCD's. (a) Show that for all nonzero $x \in \mathbb{Z}$, $\gcd(x, 0) = x$. Show that if $x, y, q, r \in \mathbb{Z}$ satisfy $y \neq 0$ and $x = qy + r$ then $\gcd(x, y) = \gcd(y, r)$. (b) Use (a) and 5.85 to develop an algorithm that will compute the gcd of two n-bit numbers using at most n integer divisions (hence at most cn^3 bit operations, for some constant c).

5.93. (a) Devise a ranking algorithm for k-element multisets of an n-element ordered alphabet based on the recursion 2.26. Use this to rank the multiset $[b, b, c, d, d, d]$ over the alphabet $\{a, b, c, d, e\}$ and to unrank 132 to get a 6-element multiset over this alphabet. (b) Repeat part (a), but use a ranking algorithm based on one of the bijections in §1.11.

5.94. Fix $k \in \mathbb{N}^+$. Prove that every $m \in \mathbb{N}$ can be written in exactly one way in the form $m = \sum_{j=1}^{k} \binom{i_j}{j}$, where $0 \leq i_1 < i_2 < \cdots < i_k$.

5.95. Fix $k \in \mathbb{N}^+$. Use 5.94 to find an explicit formula for a bijection $f : \mathbb{N}^k \to \mathbb{N}$ (cf. 1.149).

5.96. (a) Devise a ranking algorithm for derangements based on the recursion 4.24. (b) List all derangements of $\{1, 2, 3, 4\}$ in the order specified by your ranking algorithm. (c) Compute the rank of $3527614 \in D_7$. (d) Unrank 1776 to obtain a derangement in D_7.

5.97. Devise algorithms to rank and unrank partitions of n into k *distinct* parts. Use your algorithms to rank the partition $(10, 7, 6, 3, 1)$ and unrank 10 to get a partition of 20 into 3 distinct parts.

5.98. Suppose we rewrite the recursion for Stirling numbers in the form

$$S(n, k) = S(n-1, k)k + S(n-1, k-1) \qquad (n, k > 0).$$

(a) Use the bijective product and sum rules (taking terms in the order written here) to devise ranking and unranking algorithms for set partitions in $SP(n, k)$. (b) Rank the partition $\pi = \{\{1, 7\}, \{2, 4, 5\}, \{3, 8\}, \{6\}\}$ and unrank 111 to obtain an element of $SP(7, 3)$ (cf. 5.28). (c) Repeat 5.71 using the new ranking algorithm.

5.99. Use the recursion in 2.53 to develop ranking and unranking algorithms for the set $SP(n)$ of all set partitions of an n-element set. Find the rank of $\{\{1, 2, 4\}, \{3, 5, 6\}, \{7, 8\}\}$. Which set partition of 8 has rank 1394?

5.100. Find ranking and unranking algorithms for three-of-a-kind poker hands. Use these algorithms to rank the three-of-a-kind hand $\{4\heartsuit, 4\clubsuit, 4\diamondsuit, 9\clubsuit, K\spadesuit\}$ and to unrank 21,751.

5.101. Find ranking and unranking algorithms for one-pair poker hands. Use these algorithms to rank the one-pair hand $\{2\heartsuit, 2\clubsuit, 7\diamondsuit, 8\diamondsuit, 10\diamondsuit\}$ and to unrank 497,079.

5.102. (a) Find ranking and unranking algorithms for straight poker hands (including straight flushes). Use these algorithms to rank the straight hand $\{4\heartsuit, 5\heartsuit, 6\clubsuit, 7\diamondsuit, 8\diamondsuit\}$ and to unrank 1574. (b) Repeat part (a) for the set of straight hands that are not flushes.

5.103. (a) Find ranking and unranking algorithms for flush poker hands (including straight flushes). Use these algorithms to rank the flush hand $\{3\heartsuit, 7\heartsuit, 10\heartsuit, J\heartsuit, K\heartsuit\}$ and to unrank 4716. (b) Repeat part (a) for the set of flush hands that are not straights.

5.104. Develop ranking and unranking algorithms for 231-avoiding permutations. Find the rank of $w = 1\,5\,2\,4\,3\,11\,7\,6\,10\,8\,9$. Unrank 231 to get a 231-avoiding permutation of length 7.

5.105. Develop ranking and unranking algorithms for the set of subsets of $\{1, 2, \ldots, n\}$ that do not contain two consecutive integers (cf. 2.130(b)). For $n = 10$, rank the subset $\{2, 5, 7, 10\}$ and unrank 42.

5.106. (a) Use the pruning bijections in §3.12 to develop ranking and unranking algorithms for the set of trees with vertex set $\{v_1, \ldots, v_n\}$ such that $\deg(v_i) = d_i$ for all i (where the d_i are positive integers summing to $2n - 2$). (b) Given $(d_1, \ldots, d_9) = (1, 2, 1, 1, 1, 2, 3, 1, 4)$, find the rank of the tree shown in Figure 3.16. (c) Unrank 129 to obtain a tree with the degrees d_i from part (b).

5.107. Write a successor algorithm for listing integer partitions of n into k parts in lexicographic order (see 10.36). Use your algorithm to find the successors of $(9, 3)$, $(7, 4, 2, 1)$, and $(3, 3, 1, 1, 1)$.

5.108. Write a successor algorithm for listing all integer partitions of n in lexicographic order (see 10.36). Use your algorithm to find the sucessors of $(9, 3)$, $(7, 4, 2, 1)$, and $(3, 3, 1, 1, 1)$.

5.109. Write a successor algorithm for listing set partitions of $\{1, 2, \ldots, n\}$ into k blocks, using any convenient ordering. Find the successor and predecessor of the set partition $\{\{1, 7\}, \{2\}, \{3, 5, 6\}, \{4, 8\}\}$.

5.110. (a) Write a successor algorithm for listing full-house poker hands, using any convenient ordering. (b) Use your algorithm to determine the successor of the hand $\{J\clubsuit, J\diamondsuit, J\spadesuit, 9\clubsuit, 9\heartsuit\}$. (c) What is the predecessor of the hand in (b)?

5.111. (a) Write a successor algorithm for listing one-pair poker hands, using any convenient ordering. (b) Use your algorithm to find the successor of the hand $\{2\heartsuit, 2\clubsuit, 7\diamondsuit, 8\diamondsuit, 10\diamondsuit\}$. (c) What is the predecessor of the hand in (b)?

5.112. Describe a successor algorithm for ranking rooted trees with vertex set $\{1, 2, \ldots, n\}$ rooted at vertex 1. Compute the successor and predecessor of the tree shown in Figure 3.9.

5.113. Devise a random selection algorithm for choosing anagrams in $\mathcal{R}(a_1^{n_1} \cdots a_k^{n_k})$.

5.114. (a) Devise a random selection algorithm for choosing integer partitions of n into k parts. (b) Write an algorithm for choosing a random integer partition of n.

5.115. Devise a random selection algorithm for choosing a derangement of n letters.

5.116. Confirm that the random selection algorithm for permutations described at the end of §5.14 will generate every permutation in S_n with equal probability.

5.117. Consider the following proposed algorithm for randomly selecting a permutation $w \in S_n$. Initially, set $w_i = i$ for $1 \leq i \leq n$. Next, for $1 \leq i \leq n$, exchange w_i and w_j, where j is chosen randomly in $\{1, 2, \ldots, n\}$. Does this method produce every element of S_n with equal probability? Explain.

5.118. Modify the algorithm in the previous exercise by exchanging w_i with w_j, where j is chosen randomly in $\{1, 2, \ldots, i\}$ at each stage. Does this method produce every element of S_n with equal probability? Explain.

5.119. Consider the following proposed algorithm for randomly selecting a permutation $w \in S_n$. Choose $w_1 \in \{1, 2, \ldots, n\}$ at random. For $i = 2, \ldots, n$ in turn, repeatedly choose $w_i \in \{1, 2, \ldots, n\}$ at random until a value different from $w_1, \ldots, w_{i-1}$ is obtained. Argue informally that this algorithm will produce every permutation in S_n with equal likelihood, but that the expected number of random choices needed to generate one permutation in S_n is

$$n(1 + 1/2 + 1/3 + \cdots + 1/n) \approx n \ln n.$$

5.120. Devise a ranking algorithm for 4-letter words in which Q is always followed by U (so Q cannot be the last letter). Use your algorithm to rank AQUA and QUIT and to unrank 1000. Can you find an algorithm that generates these words in alphabetical order? Can you generalize to n-letter words?

5.121. Devise a ranking algorithm for 5-letter words that never have two consecutive vowels. Use your algorithm to rank BILBO and THIRD and to unrank 9999. Can you find an algorithm that generates these words in alphabetical order? Can you generalize to n-letter words?

Notes

Our presentation of ranking and random selection places great emphasis on bijections constructed automatically by repeated use of the bijective sum and product rules. For a somewhat different approach based on a multigraph model, see the papers [141, 142]. Other discussions of ranking and related problems can be found in the texts [10, 100, 128]. An encyclopedic treatment of algorithms for generating combinatorial objects may be found in Knuth's comprehensive treatise [78].