MATH 343 Applied Discrete Math
Supplementary Materials

Luis Goddyn, 99-3

This is supplementary material for MATH 343-3, Applied Discrete Math. You will find material regarding the following topics.

1. Generation of objects in minimal change order:
   (a) All subsets: Reflected Binary Gray code, recursive definition, transition sequence, successor rule, converting from RBC to binary and back.
   (b) All subsets: Specialized Gray codes: Balanced transition counts, Balanced linear time counts, Large minimum run length, Beckett codes.
   (c) Generating other objects recursively
      - All k-subsets (revolving door): Recursive definition, relation to RBC.
      - All permutations in S_n.
      - De Bruijn Orders: Euler tours, Fleury's algorithm, directed line graphs.
2. Generation of objects in lexicographic order:
   (a) Encoding objects as words over an ordered alphabet
   (b) Successor and Predecessor Rules for
      i. All subsets of an n-set.
      ii. All k-subsets of an n-set
      iii. All permutations
      iv. All partitions of an integer.
      v. All partitions of an integer in to a fixed number of parts..
3. Decompositions of complete graphs.
   (a) 1-factorization of K_{2n}
   (b) Hamilton decomposition of K_{n+1}
   (c) Near Hamilton decomposition of K_{2n}
   (d) Hamilton path decomposition of K_{2n}
   (e) Steiner Triple system.
   (f) Latin squares

1 Minimal Change Orders

Suppose we with to list all possible subsets of $[n] := \{0, \ldots, n-1\}$ such that consecutive subsets differ as little as possible. This may be needed in order to minimize the cost of a series of statistical tests, or to reduce the possibility of error in analogue-digital converters, as explained in class. The first question is: *What does "as little as possible" mean?*. For example, a small change may involve deleting an element from or adding a new element to the current subset. Alternatively, if we indicate a subset of $[n]$ by its $0,1$-indicator vector of length n, then we might regard a "left-shift", such as happens when entering digits on a microwave oven, to be a minimal change. Furthermore we might wish to generate only those subsets of $[n]$ having a particular size. Or perhaps we want other combinatorial objects such as permutations, partitions. Here are some notes which touch on these and related issues.

1.1 Reflected Binary Gray Code

A *binary word* is a finite sequence of zeros and ones. We shall indicate each subset $S \subseteq [n]$ by the binary word $(c_{n-1}, c_{n-2}, \ldots, c_0)$ where $c_i = 1$ if and only if $i \in S$. For example the subset $\{3,1,0\} \subseteq [5]$ is indicated by (01011). We define the matrices $G(n)$, $n = 1, 2, \ldots$ recursively by

$$G(1) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$G(n) = \begin{bmatrix} 0 & G(n-1) \\ 1 & G(n-1)^R \end{bmatrix}, \quad n = 2, 3, \ldots.$$

(Here G^R is the matrix obtained from G by writing the rows in reverse order so that the first row of G^R is the last row of G. The "0" denotes a column vector of zeros and the "1" denotes a column vector of ones.) For example we have $G(2) = \begin{bmatrix} 00 \\ 01 \\ 11 \\ 10 \end{bmatrix}$. It is not hard to see that the rows of $G(n)$, taken in order, form a list of n-bit binary words which indicate all subsets of $[n]$ in which successive sets differ in at most one element. For example $G(2)$ corresponds to the list of subsets $(\emptyset, \{0\}, \{0,1\}, \{1\})$ generated in that order. We notice that in fact the order given by $G(n)$ is *cyclic*. That is, the last and first rows of $G(n)$ are also related by a minimal change. Notice that in $G(2)$ the first two rows differ in bit position 0, the next two rows differ in position 1 while the third pair of rows differ in position 0 again. Thus we may efficiently record $G(2)$ by its *transition sequence* $(0,1,0)$. The transition sequence for $G(3)$ is $(0,1,0,2,0,1,0)$.

DEFINITION 1.1 An *n-bit binary word* is a sequence $c = (c_{n-1}, c_{n-2}, \ldots, c_0)$, where each c_i is 0 or 1. Each binary word c *indicates* a unique subset $S \subseteq [n]$ where $i \in S$ if and only if $c_i = 1$. A *Gray code* is a permutation (ordering) $C = (c^0, c^1, \ldots c^{N-1})$ of all $N = 2^n$ n-bit binary words, such that c^{j-1} and c^j differ in exactly one bit position ($j = 1, \ldots, N-1$). (Where needed, we may write c_i^j to denote the i-th bit of the j-th word in C.) A Gray code is *cyclic* if c^{N-1} and c^0 also differ in one bit. Suppose that c^j and c^{j+1} differ in bit position t^j, $j = 0, \ldots, N-2$. Then $(t^0, t^1, \ldots, t^{N-2})$ is called the *transition sequence* of the Gray code. The special Gray code obtained by listing the rows of $G(n)$ as described above is called the *Reflected Binary Gray code* or *RBC* for short.

Suppose we are given a Gray code $C = (c^0, c^1, \ldots, c^{N-1})$ either as a list of words or by some rule (such as with the RBC). There are three questions that one might want to be able answer quickly regarding C. First: *Is there a* **successor rule** *for the code?* That is, can one describe a quick procedure for deciding which word comes after a given word $c = (c_{n-1}, \ldots, c_0)$ in the Gray code C? Of course one can simply look up c in the list C and output the next word in C, but this is hardly "quick" as it may involve examining $N = 2^n$ words. What we desire is a rule which can be executed in time which is bounded by a polynomial in n. The problem of describing a successor rule for the RBC is an exercise below.

Second: *Is there an* **encoding rule** *for C?* Such a rule must be able to quickly tell me the binary word c^k for any integer $0 \leq k \leq N-1$.

Third: *Is there an* **decoding rule** *for C?* If I specify a n-bit word c, such a rule must be able to quickly tell me where c occurs in the code C. That is, the rule must determine the index k such that $c = c^k$.

We state here without proof a procedure for encoding and decoding the RBC. Let "$\oplus$" denote the "exclusive or" (XOR) operation so that $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. To encode an integer k with $0 \leq k \leq N-1$ into the word $c^k = (c_{n-1}, c_{n-2}, \ldots, c_0)$, we first convert k to an n-bit *binary number* $b = (b_{n-1}, b_{n-2}, \ldots, b_0)$ in the usual way. (That is the b_i satisfy $k = b_0 + 2b_1 + 4b^2 + \cdots + 2^{n-1}b_{n-1}$.) The highest (leftmost) bit of c^k is the same as that of b. The remaining bits are defined by taking "adjacent XORs". That is,

$$c_{n-1} = b_{n-1}$$

$$c_j = b_j \oplus b_{j+1}, \quad j = 0, 1, \ldots n-2.$$

For example, the 158th word in the 8-bit RBC is c_{157}. We write 157 in binary notation as $b = (10011101)$. Setting the first digit of c to equal 1 and taking adjacent XORs results in the RBC word $c_k = (11010011)$.

To decode an RBC word $c^k = (c_{n-1}, c_{n-2}, \ldots, c_0)$, we take "successive XORs from the left", then convert the resulting binary number b into decimal notation. That is, we define $b = (b_{n-1}, b_{n-2}, \ldots, b_0)$ by

$$\begin{aligned} b_i &= c_{n-1} \oplus c_{n-2} \oplus \cdots \oplus c_i, \quad i = 0, 1, \ldots n-1 \\ &= b_{i+1} \oplus c_i, \qquad\qquad\qquad\quad i = 0, 1, \ldots n-2. \end{aligned}$$

and then set $k = b_0 + 2b_1 + 4b_2 + \cdots + 2^{n-1}b_{n-1}$. For example, decoding the 8-bit word $c = (11010011)$ results in $b = (10011101)$ which in decimal notation is $1 + 4 + 8 + 16 + 128 = 157$.

EXERCISE 1.1

1. *Write down the transition sequence of the* 4*-bit and* 5*-bit RBC codes.*
2. *List the words of a non-cyclic* 3*-bit Gray code starting at* (000)*, and write down its transition sequence.*
3. *Describe a rule for obtaining the transition sequence* T_{n+1} *of the* $(n+1)$*-bit RBC from the transition sequence* T_n *, of the* n*-bit RBC.*
4. *Describe a rule for obtaining the transition sequence* T_{n+1} *of a cyclic* $(n+1)$*-bit Gray code from the transition sequence* T_n *of a (non necessarily cyclic)* n*-bit Gray code* $(c^0, \ldots, c^{N-1})$*. In particular,* T_n *should be a subsequence of* T_{n+1}*. You must present an argument that your new Gray code is actually a Gray code and that it is cyclic.*
5. *Which word follows* (11010111) *in the* 8*-bit RBC?*
6. *Describe a successor rule for the RBC. That is, describe a quick way to obtain the word that follows any codeword* $c = (c_{n-1}, \ldots, c_0)$ *different from* $(100\cdots0)$ *in the RBC. Hint: Your rule will have two cases, depending on the number of bits in* c *which are equal to "*1*".*
7. *What is the* 728*th word,* c_{727}*, in the* 10*-bit RBC?*
8. *In what position does the word* (0111011100) *occur in the* 10*-bit RBC?*

1.2 Special Gray Codes

Balanced Transition Counts

Notice that in the 4-bit RBC, the left most changes only once whereas the last bit changes eight times. That is, there are eight "0"s in its transition sequence whereas there is only one "3". This imbalance may be a problem in some applications. We ask whether one can construct codes for which this "transition count" is more balanced. If we consider the final transition $c_{N-1} \mapsto c_0$, then the first digit actually changes twice. For any cyclic n-bit Gray code C we define its *transition count* to be the vector $(k_{n-1}, k_{n-2}, \ldots k_0)$ where k_i is the number of times the ith bit changes during a cyclic traversal of C. For example the transition count of the n-bit RBC is $(2, 2, 4, 8, \ldots, N/2)$. We leave as an exercise that for any Gray code, each k_i in its transition count is an even integer with $2 \leq k_i \leq N/2$, and the average value of the k_i is N/n.

There are applications for which the RBC is a poor choice due to the terrible imbalance of its transition count. This begs an obvious question: *Does there exist, for each n, and n-bit Gray code such that the k_i are all close to being equal?* Since the k_i are even, the best we can hope for is that each k_i is within 2 of their average, N/n.

If n is a power of two, say $n = 2^k$, then $N/n = 2^{2^k - k}$ is an even integer and we might even hope for a perfectly balanced transition sequence, where all $k_i = N/n$. For example, $t = (0, 1, 0)$ is a perfectly balanced transition sequence of a 2-bit code since there are exactly two 0s and two 1s in t (after adding the final 1 to make it cyclic). There is also a perfectly balanced 4-bit code (see exercise below). In fact D. Wagner and J. West showed in 1991 that perfectly balanced Gray codes exist for all $n = 2^k$. In 1996, G. Bhat and C. Savage showed that for every n there exists an n-bit Gray code with all transition counts within 2 of the average N/n. These constructions are neither easy to describe nor are they easy to prove that they result in balanced Gray codes.

Balanced Linear Time Counts

Suppose that we wish to test all 16 possible engines that one can build using two types of carburetors, two types of valves, two types of pistons, and two types of gaskets. There is one engine which is tested for one day, then modified and then tested for another day. After 16 days a statistical study determines the best combination of components to use. Clearly we should modify the engine as in a minimal change order so that total labour is minimized. However as an engine runs, its efficiency increases due to its 'breaking in'. Thus if some engine component is more often tested in the later half of the 16 days, then that component will have an unfair advantage. The RBC is a particularly poor choice of Gray code, since the first bit track is 0 on the first half and 1 on the second half in the sequence of words. We ask:

Does there exist, for each n, an n-bit Gray code such that no bit track has its ones 'bunched up' near the start or the end of the sequence of N words?

Formally, we calculate the *first moment* or *linear time count* of each bit track. That is, if we write the Gray code as a $N \times n$ matrix G as above, then the *linear time count* is defined to be the vector $\ell = vG$ where v is the row vector $(0, 1, \ldots, N-1)$. Note that ℓ is an n-bit row vector whose entries lie between $0+1+\cdots+N/2-1 = N(N-2)/8$ and $N/2+(N/2+1)+\cdots+(N-1) = N(3N-2)/8$. An ideal distribution of 1s on a bit track would have a linear time count of $N/4 + (N/4+1) + \cdots + (3N/4-1) = N(N-1)/4$. We define the *ideal time count* ITC(n) to be the n-vector $(t, t, \ldots, t)$ where $t = 2^n(2^n-1)/4$.

The linear time count of the 4-bit RBC is $(92, 60, 60, 60)$, which is quite 'far' from the ideal time count ITC(4) $= (60, 60, 60, 60)$. Although the "1"s in three of the bit tracks are perfectly balanced, those in the leftmost track heavily favour the end of the code sequence. There is another 4-bit Gray code, starting with (0000) and having transition sequence $(0, 1, 3, 2, 0, 1, 3, 1, 0, 1, 3, 2, 0, 1, 3)$ which has a more balanced time count, $(68, 60, 60, 52)$. This Gray code is much more suitable for the engine test described above.

How does one decide which of two time count vectors is better? Statisticians usually prefer a the time count which is closer to ITC(n) in *Euclidean distance*. For example the squared distance from the time count of RBC to ITC(4) is $(92-60)^2 + (60-60)^2 + (60-60)^2 + (60-60)^2 = 1024$ whereas $(68, 60, 60, 52)$ has squared distance only 128. Finding an n-bit Gray code with an optimal linear time count is a difficult unsolved problem. For $n \leq 6$, Gray codes with fairly good time counts have been found by computer search, but the best code is still unknown for $n = 7$.

Large Minimum Run Length

The rightmost bit track of the RBC has the pattern $0110011001100\cdots110$ (this is the rightmost column of $G(n)$). Thus there are only two "1"s before it changes back to "0". This can be a problem in applications

as described in class. Let C be a cyclic Gray code. We define a *run in bit track i* of C to be a maximal list of consecutive code words $c^{j+1}, c^{j+2}, \ldots, c^{j+k}$ such that the ith bit of these words are all equal. That is,

$$1 - c_i^j = c_i^{j+1} = c_i^{j+2} = \cdots = c_i^{j+k} = 1 - c_i^{j+k+1}.$$

The *length* of such a run is defined to be k. For example, all the runs in the rightmost bit track of the RBC have length 2. A *run* in C is any run in any of the bit tracks of C. The *minimum run length* of C, written $\mathrm{MRL}(C)$ is the smallest length among all of its runs. We ask:

For each $n \geq 2$, what is the largest minimum run length that a cyclic n-bit Gray code can have?

It is not hard to show that every cyclic n-bit Gray code C satisfies $\mathrm{MRL}(C) \leq n - 1$ if $n \geq 3$ (see exercise below). For example, a cyclic 5-bit code with transition sequence

$$(0, 3, 1, 4, 2, 3, 1, 4, 0, 3, 2, 4, 1, 3, 2, 4, 0, 3, 1, 4, 2, 3, 1, 4, 0, 3, 2, 4, 1, 3, 2)$$

has minimum run length 4. In 1988, Goddyn, Lawrence and Nemeth constructed cyclic codes C with $\mathrm{MRL}(C) = 2n/3$. Later, Goddyn and Gvozdjak improved this to $n - 5\log_2 n$. That is, for every $\epsilon > 0$, there exists an $n \geq 2$ and a cyclic n-bit Gray code C where $\mathrm{MRL}(C)/n > 1 - \epsilon$. These constructions are rather involved and complicated.

Beckett Gray Codes

Playwrite Samuel Beckett had a passion for order and symmetry which is expressed in some of his absurdist plays. One play, "Quad", is divided into sixteen time periods. At the end of each period, one of four actors either enters or exits the stage. Apparently, Beckett's intension was to have each of the sixteen subsets of actors appear on stage exactly once. Also, the play should begin and end with an empty stage. This is clearly a cyclic 4-bit Gray code. However, Beckett's scripting demands that whenever an actor leaves the stage, it must be the most 'exhausted' of those currently on stage. That is, whenever an actor leaves the stage, sie must be the unique actor who has been on stage the longest without a break.

In technical terms, Beckett sought a cyclic Gray code with the special property that whenever a bit changes from 1 to 0, it must be the bit that currently has the longest run of 1s. There is no restriction on which bit can change from 0 to 1. We shall call such a code a *Beckett Gray code*.

There is an alternative model for Beckett's restriction. We imagine the stage to be a queue where actors enter from the left and exit to the right. Each move consists of either adding a new actor to the back of the queue or popping the front actor off the queue. There is an obvious question.

For which n does there exist an n-bit Beckett Gray code?

It is known that n-bit cyclic Beckett Gray codes exist for $n \in \{1, 2, 5, 6\}$, and that they do not exist for $n \in \{3, 4\}$. The status for $n \geq 7$ is unknown at this time. Thus Beckett is unable to cast his play as described. Indeed there are subsets of actors which appear multiple times in his play!

Beckett codes appear to be very difficult to find or to work with mathematically. All Beckett Gray codes with $n \leq 5$, and several hundred thousand codes with $n = 6$ have been found by a computer search of B. Stevens, but the case $n = 7$ already appears almost hopeless. The search becomes only slightly easier if we relax Beckett's 'cyclic' requirement. For example, one non-cyclic 7-bit Beckett code was found by F. Ruskey after several month's computing time. The search space for $n = 7$ is so large that finding such codes seems to require as much luck as computing skill.

EXERCISE 1.2

1. *Let $(k_{n-1}, \ldots k_0)$ be the transition count of a cyclic n-bit code. Show that each k_i is an even integer with $2 \leq k_i \leq N/2$ and average value N/n.*
2. *(Bonus) Show that the average length of a run in a cyclic n-bit Gray code equals n.*
3. *Find a cyclic 4-bit gray code having a perfectly balanced transition count. List your Gray code and draw it as a path on a diagram of the 4-cube.*
4. *What is the squared distance from the linear time count of the n-bit RBC to ITC(n)?*
5. *Find a 3-bit Gray code with the optimal linear time count ('optimal'='smallest distance to ITC(3)').*
6. *Show that for any $n \geq 3$, the minimum run length of any cyclic n-bit Gray code is at most $n - 1$.*
7. *(Bonus) Find a 5-bit Beckett Gray code.*

1.3 Generating Other Objects Recursively

Generating k-subsets

Suppose we need to generate all possible $\binom{n}{k}$ k-subsets of an n-set. Here, a 'minimal change' would be to generate each successive set from the previous one by deleting one element and adding a new element. One might imagine a 'revolving door' to a room holding k people, and a move consists of one person leaving the room through the door simultaneously to another person entering the room. Equivalently, we wish to find a $\binom{n}{k}$ by n binary matrix such that every word having exactly k ones appears as a row, and such that consecutive rows differ in exactly two bit positions.

We begin by attempting to find a "natural" minimal change order which would be analogous to the RBC $G(n)$. There is a well known recurrence relation

$$\binom{n}{0} = \binom{n}{n} = 1, \quad n \geq 0$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \quad 1 \leq k \leq n$$

for the binomial coefficients. We might hope to use this to find a natural minimal change order for the k-subsets, in a similar way that $G(n)$ is related to the recurrence $2^n = 2^{n-1} + 2^{n-1}$. In fact we can do this as follows.

$$G(n,0) = \overbrace{[0\ 0\ \cdots\ 0]}^{n}, \qquad G(n,n) = \overbrace{[1\ 1\ \cdots\ 1]}^{n}, \quad \text{for } n \geq 1 \tag{1}$$

$$G(n,k) = \begin{bmatrix} 0 & G(n-1,k) \\ 1 & G(n-1,k-1)^R \end{bmatrix}, \qquad \text{for } n > k > 0. \tag{2}$$

Again, G^R is the matrix G with reversed row order, as defined in Section 1.1. For example we calculate

$$G(2,1) = \begin{bmatrix} 0 & G(1,1) \\ 1 & G(1,0)^R \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

A more complicated example follows:

$$\begin{aligned} G(4,2) &= \begin{bmatrix} 0 & G(3,2) \\ 1 & G(3,1)^R \end{bmatrix} = \begin{bmatrix} 0 & \begin{bmatrix} 0 & G(2,2) \\ 1 & G(2,1)^R \end{bmatrix} \\ 1 & \begin{bmatrix} 0 & G(2,1) \\ 1 & G(2,0)^R \end{bmatrix}^R \end{bmatrix} \\ &= \begin{bmatrix} 0 & \begin{bmatrix} 0 & G(2,2) \\ 1 & G(2,1)^R \end{bmatrix} \\ 1 & \begin{bmatrix} 1 & G(1,1)^R \\ 0 & G(2,1) \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \end{aligned}$$

In the following, we write $[0^s 1^t]$ for $[\overbrace{0\ 0\ \cdots\ 0}^{s}\ \overbrace{1\ 1\ \cdots\ 1}^{t}]$.

THEOREM 1.2 *For $0 \leq k \leq n$, the the rows of $G(n,k)$ represent a cyclic minimal change ordering of all k-subsets of $[n]$.*

PROOF. Since $G(n,k)$ is recursively defined, we shall use induction on n. The proof becomes easier if we use the following stronger induction hypothesis.

$H(n)$: For $k = 0, 1, \ldots, n$ the following two statements hold: (a) $G(n,k)$ represents a cyclic minimal change order. (b) $G(n,k)$ begins with $[0^{n-k}1^k]$, and ends with either $[1\ 0^{n-k}1^{k-1}]$ (if $k \leq 0$) or $[0^n]$ (if $k = 0$).

The hypothesis $H(1)$ holds true since $G(1,0) = [0]$ and $G(1,1) = [1]$. Let $n \geq 2$ and suppose that $H(n-1)$ holds true. If $k = 0$ or $k = n$ then $G(n,k)$ equals $[0^n]$ or $[1^n]$ so statements (a) and (b) of $H(n)$ hold true in this case. We may assume for the rest of this proof that $1 \leq k \leq n-1$.

By the recurrence (2), the first row of $G(n,k)$ is the first row of $[0\ G(n-1,k)]$ which by induction hypothesis equals $[0\ 0^{(n-1)-k}1^k] = [0^{n-k}1^k]$. Similarly, the last row of $G(n,k)$ is the first row of $[1\ G(n-1,k-1)]$ which, by induction hypothesis, equals $[1\ 0^{(n-1)-(k-1)}1^{k-1}] = [1\ 0^{n-k}1^{k-1}]$. We have now proved statement (b) of $H(n)$.

It remains to prove statement (a) when $1 \leq k \leq n-1$. By induction hypothesis, each of $G(n-1,k)$ and $G(n-1,k-1)^R$ are minimal change orders, so we need only check that the two 'boundary transitions' in (2) are minimal changes. The first and last rows of $G(n,k)$ were determined in the previous paragraph, and we easily check that they differ in only two bit positions.

Finally we check the 'middle' boundary of $G(n,k)$. By induction hypothesis, the last row of $[0\ G(n-1,k)]$ is $[0\ 1\ 0^{n-k}1^{k-1}]$. In comparison, the first row of $[1\ G(n-1,k-1)^R]$ is the last row of $[1\ G(n-1,k-1)]$ which is either $[1\ 1\ 0^{n-k}1^{k-2}]$ (if $k \geq 2$) or $[1\ 0^n]$ (if $k = 1$). In either case, only two bits have changed. □

We finish with a note on computation. Recursive constructions are attractive as they are often easy to code as a recursive algorithms. However it is common for such code to have poor running times. For example, a naïve encoding of $G(n,k)$ might result in smaller matrices $G(n',k')$ $(n' < n,\ k' < k)$ being unnecessarily recomputed several times. For example, $G(n-2,k-1)$ is needed in the constructions of both $G(n-1,k)$ and $G(n-1,k-1)$. Thus $G(n-2,k-1)$ might be calculated twice when computing $G(n,k)$. Unless memory is in short supply, each $G(n',k')$ should be stored for future reference in a lookup table the first time it is calculated. Here is an encoding of $G(n,k)$ in MAPLE which illustrates the resulting time savings.

```
> with(linalg):

> # Reverse rows of matrix M #
  Rev:=proc(M)
       rowdim(M); submatrix(M, [seq(%-i,i=0..%-1)], 1..coldim(M))
  end:

> # Minimum change order matrix for k-subsets of [n] #
  G:=proc(n,k)
     if   k=0 then RETURN( matrix(1,n,0) )
     elif k=n then RETURN( matrix(1,n,1) )
     else G(n-1,k);   G(n-1,k-1);
          concat( vector(rowdim(%%),0) , %% );
          concat( vector(rowdim(%%),1) , %% );
          RETURN( stackmatrix(%%,Rev(%)) )
     fi
  end:

> G(4,2)
                        [ 0 0 1 1 ]
                        [ 0 1 1 0 ]
                        [ 0 1 0 1 ]
                        [ 1 1 0 0 ]
                        [ 1 0 1 0 ]
                        [ 1 0 0 1 ]

> # Timing test (in seconds) #
  time( G(8,4) );
                           6.180

> # Ask procedure G to remember all computed values in a lookup table #
  G:=subsop(3=remember, eval(G)):

> time( G(8,4) );
                           1.859
```

Generating Permutations

A *permutation* of a set X is a one-to-one and onto function $\pi : X \to X$. We define S_n, $n \geq 1$ to be the set of permutations on $[n] = \{0, 1, \ldots, n-1\}$. Thus $|S_n| = n!$. There are several standard ways of denoting a permutation. Consider the permutation $\pi \in S_5$ where $\pi(0) = 3, \pi(1) = 4, \pi(2) = 2, \pi(3) = 0$, and $\pi(4) = 1$. We may write this in *matrix notation* as

$$\pi = \begin{bmatrix} 0 & 1 & \ldots & n-1 \\ \pi(0) & \pi(1) & \ldots & \pi(n-1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 0 & 1 \end{bmatrix}.$$

We may even write just the last row of this matrix: [3 4 2 0 1]. Alternatively, we may write π in *cyclic notation*. To do this, we iterate π starting with 0 until just before we reach 0 once again. We place the resulting sequence in parentheses, $(0\ \pi(0)\ \pi(\pi(0))\ \ldots)$. Now we determine the least integer in $[n]$ which has not yet appeared and repeat the process. When $\pi = [\ 3\ \ 4\ \ 2\ \ 0\ \ 1\ ]$ we obtain $(0\ 3)(1\ 4)(2)$. Each iteration of the above process results in a *cycle* in π. For example, $\pi = (0\ 3)(1\ 4)(2)$ has two cycles of length 2 and one cycle of length 1. It is common practice to not bother writing out cycles of length 1, so we may write $\pi = (0\ 3)(1\ 4)$ in this example.

A minimal change for a permutation might consist of a "swap" (also called a *transposition*) of two columns in the matrix notation. For example [3 4 2 0 1] and [3 0 2 4 1] are related by a transposition. Alternatively, there is a notion which is "more minimal" than this. We may allow only *adjacent transpositions* in which only consecutive entries in the matrix may be swapped, such as [3 4 2 0 1] $\mapsto$ [3 2 4 0 1].

There exists a recursive definition of a minimal change order for S_n. We shall indicate a minimal change order of S_n by a $n! \times n$ matrix $P(n)$ where each row is a distinct permutation in S_n. We define $P(1)$ to be the matrix [0]. The larger matrices $P(n)$, $n \geq 2$ are defined inductively. If $\pi = [a_0\ a_1\ \ldots\ a_{n-1}]$ is a permutation of $[n]$ then we define the *left expansion* $\pi^{\leftarrow n}$ and the *right expansion* $\pi^{\rightarrow n}$ to be the matrices

$$\pi^{\leftarrow n} = \begin{bmatrix} a_0 & a_1 & \ldots & a_{n-1} & n \\ a_0 & a_1 & \ldots & n & a_{n-1} \\ & & \vdots & & \\ a_0 & n & \ldots & a_{n-2} & a_{n-1} \\ n & a_0 & \ldots & a_{n-2} & a_{n-1} \end{bmatrix}, \qquad \pi^{\rightarrow n} = \begin{bmatrix} n & a_0 & \ldots & a_{n-2} & a_{n-1} \\ a_0 & n & \ldots & a_{n-2} & a_{n-1} \\ & & \vdots & & \\ a_0 & a_1 & \ldots & n & a_{n-1} \\ a_0 & a_1 & \ldots & a_{n-1} & n \end{bmatrix}.$$

For example, $[1\ 0\ 2]^{\leftarrow n} = \begin{bmatrix} 1 & 0 & 2 & 4 \\ 1 & 0 & 4 & 2 \\ 1 & 4 & 0 & 2 \\ 4 & 1 & 0 & 2 \end{bmatrix}$. Finally, if $P(n) = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_{n!} \end{bmatrix}$, where each π_i is a row permutation in S_n, we define $P(n+1)$ to be the $(n+1)! \times n+1$ matrix

$$P(n+1) = \begin{bmatrix} \pi_1^{\leftarrow n} \\ \pi_2^{\rightarrow n} \\ \vdots \\ \pi_{n!-1}^{\leftarrow n} \\ \pi_{n!}^{\rightarrow n} \end{bmatrix}.$$

For example, starting with $P(1) = [0]$, this gives

$$P(2) = \left[0^{\leftarrow 1}\right] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad P(3) = \begin{bmatrix} [\,0 & 1\,]^{\leftarrow 2} \\ [\,1 & 0\,]^{\rightarrow 2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \\ 2 & 0 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}.$$

De Bruijn Sequences

A *binary De Bruijn cycle (or sequence) of order* n is a cyclic sequence of digits 0 and 1 such that each of

the $N := 2^n$ n-bit binary words occurs exactly once consecutively along this cycle. An example for $n = 4$ is given by the sequence (0 0 0 0 1 0 1 1 0 1 0 0 1 1 1 1). The first four digits are (0 0 0 0). The second through fifth digits give the word (0 0 0 1). Continuing in this manner, we obtain all sixteen 4-bit words exactly once, the last one being (1 0 0 0). The resulting ordering is a cyclic minimal change ordering of all subsets of an n-set, where now a "left shift" is regarded to be a "minimal change". The student may recognize this operation when entering successive digits into the keyboard of a standard microwave oven.

Such a sequence exists for all $n \geq 1$. This was first proved in 1894 by Flye-Sainte Marie and rediscovered by De Bruijn in 1946. De Bruijn sequences are useful for designing *Position-to-Analog* converters as it can be used, for example, to determine the position of a circular drum, as shown in class.

How to find a De Bruijn sequence of order n:

1. Draw a directed graph D_n whose vertices are the $(n-1)$-bit binary words. (There are $N/2$ such words.) We draw an arc directed from $(c_0c_1 \ldots c_{n-2})$ to $(d_0d_1 \ldots d_{n-2})$ if and only if $c_i = d_{i-1}$, $i = 1, 2, \ldots, n-2$. We can label this arc with the n-tuple $(c_0c_1 \ldots c_{n-2}d_{n-2})$. For example, when $n = 7$ there would be an arc from (011010) to both (110100) and (110101). These arcs would be labeled with (0110100) and (0110101), respectively. Notice that there are exactly N arcs in D_n and that they are all labeled with distinct n-bit binary words.

2. Now we find an *Euler tour* of D_n. That is, we find a closed trail in D_n which traverses each of the arcs (in the right direction) exactly once, ending at the same vertex you started with.

3. Write down the first digit c_0 of the label of each arc in the order that you traverse them. The resulting binary sequence has length N and is a De Bruijn sequence of order n.

Try a few small examples ($n = 2, 3, 4$) to convince yourself that this works. This construction begs the following question. How does one find an Euler tour in D_n and how do we know that one even exists? It is a standard result in Graph Theory that a directed graph D has an Euler tour if and only if D is connected and for some $d \geq 1$ each vertex has exactly d incoming arcs and d outgoing arcs. It is easy to see that D_n is connected and that each vertex of D_n has exactly 2 incoming and 2 outgoing arcs? To find such a tour we use Fleury's algorithm which we now describe.

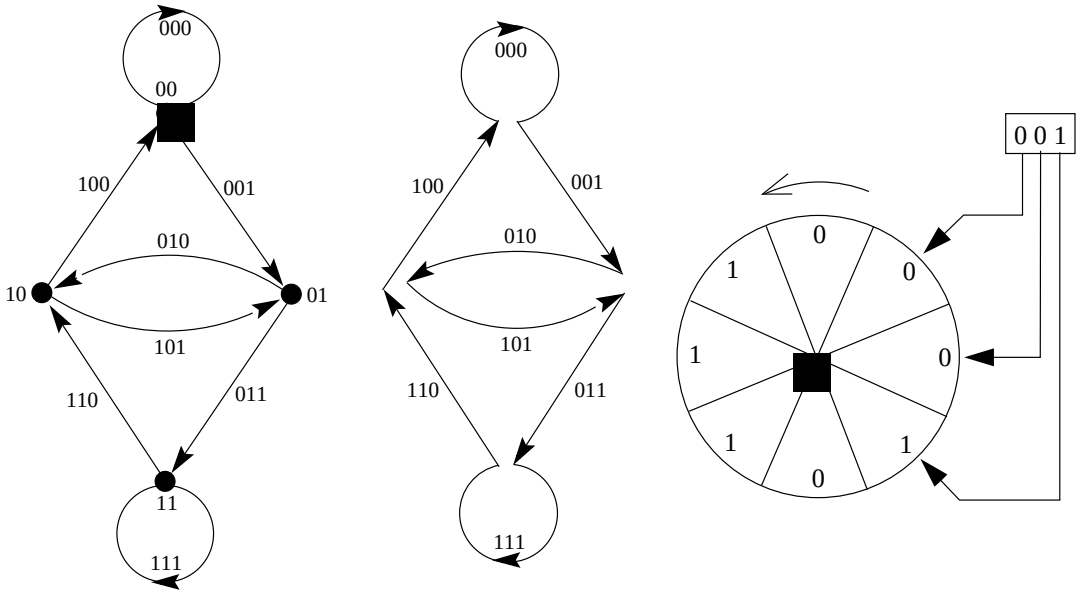
1. Begin at any vertex v_0 and let a_1 be any arc which leaves v_0, and let v_1 be the head of this arc. Delete a_1 from the current graph.

2. Suppose that we have so far constructed a tour $(v_0, a_1, v_1, a_2, \ldots, a_k, v_k)$. We are free to choose the next arc, a_{k+1}, among those which have tail v_k and which belong to the current graph, subject only to the condition that, after deleting a_{k+1}, there remains a directed path from any arc remaining in the graph back to v_0. That is, deleting a_{k+1} should not "isolate" any arc from the starting vertex v_0. Let v_{k+1} be the head of a_{k+1}. Delete a_{k+1} from the current graph.

3. If there are still some arcs remaining in the current graph, then go to step 2.

I will not prove here that Fleury's algorithm always works as this is not a graph theory course, but you are expected know how the algorithm runs.

Here is an example of how to generate an 3-bit De Bruijn sequence. First we make the graph D_3 which has four vertices and eight arcs labeled as shown in diagram (a). Then we find an Euler tour using Fleury's algorithm, as in diagram (b). Finally, we take the first digits of the arcs traversed in the tour, and write them on a circular disk as in diagram (c). Here, the disk is in position [0 0 1].



EXERCISE 1.3

1. *For a few values of n and k do the following: Write out the RBC matrix $G(n)$ as in Section 1.1. Select those rows of $G(n)$ which contain exactly k ones (without permuting the rows). Compare your resulting matrix with $G(n,k)$. After making your tests, propose a conjecture about what you have observed.*

2. *(Bonus) Find a successor rule for $G(n,k)$.*

3. *Think of a definition of "minimal change" for k-subsets of $[n]$ which is more "minimal" than the definition used for $G(n,k)$. Find a ordering of the 2-subsets of $[4]$ which is a cyclic minimal change order in your new stronger sense.*

4. *Write the permutation $[05736412] \in S_8$ in cyclic notation.*

5. *Write the permutation $(0347)(25) \in S_8$ in matrix notation.*

6. *Write out the matrix $P(4)$.*

7. *Write out a proof that $P(n)$ is a cyclic listing of S_n in minimal change order, where minimal change is defined to be "adjacent transposition".*

8. *(Bonus) Find and describe a successor rule for $P(n)$.*

9. *Find a binary De Bruijn sequence of order 4.*

10. *A* ternary *De Bruijn sequence of order n is similar to a binary one, except that the alphabet of symbols is $\{0,1,2\}$ instead of $\{0,1\}$ and we now want all 3^n n-tuples to appear in the sequence. Modify the above construction to find a ternary De Bruijn sequence of order 3.*

11. *Another variation of the De Bruijn problem is to find a cyclic sequence of length $\binom{n}{k}$ such that every k-subset of $[n]$ appears exactly once as a consecutive string along the sequence. We call such an object an $\binom{n}{k}$-*universal sequence*. For example the sequence (0 1 2) is a $\binom{3}{2}$-universal sequence since the three pairs [0 1], [1 2], [2 0] appear consecutively exactly once along the sequence. Find a $\binom{5}{2}$-universal sequence. Show that a $\binom{5}{3}$-universal sequence does not exist.*

2 Lexicographic Generation

Let A be an alphabet whose elements are linearly ordered by some relation "$\leq$". We call the pair $(A, \leq)$ an *ordered alphabet*. For example, $([4], \leq)$ is an ordered alphabet, where $[4] = \{0, 1, 2, 3\}$ and "$\leq$" has its usual meaning. Let A^* be the set of words (of any finite length) that can be formed using A. For example, [3 0 3 2 2] is a word of length 5 belonging to [4]*. Using "$\leq$", we can define another linear ordering "$\preceq$" on A^* which is very similar to how words are ordered in an ordinary dictionary. This ordering is called the *lexicographic ordering* on A^* induced by the ordered alphabet $(A, \leq)$.

DEFINITION 2.1 *Let $(A, \leq)$ be a linearly ordered alphabet. The lexicographic ordering of A^* induced by $(A, \leq)$ is the linear ordering "$\preceq$" defined as follows. Let $\alpha = [a_1\ a_2\ \ldots\ a_s]$, $\beta = [b_1\ b_2\ \ldots\ b_t]$ be words in A^*. Then $\alpha \preceq \beta$ if an only if there is some index k such that*

- $0 \leq k \leq \min\{s, t\}$,
- $a_i = b_i$ *for* $1 \leq i \leq k$, *and*
- *either* $k = s$ *or* $(k < t$ *and* $a_{k+1} < b_{k+1})$.

We write $\alpha \prec \beta$ if $\alpha \preceq \beta$ and $\alpha \neq \beta$

This definition might seem overly complicated, but it is necessary to handle the cases where the words have different lengths. For example, if $a < b < c$ then we have $[b] \prec [b\ a] \prec [b\ a\ a] \prec [b\ b] \prec [b\ c] \prec [c]$.

In applications where we wish to generate combinatorial objects, but where it is not important that they be in minimal change order, it is often easy to describe and implement successor and predecessor rules for generating the objects in lexicographic order.

Before this can be done, it is necessary to *uniquely encode* each combinatorial object as a word over some ordered alphabet. After the resulting words are lexicographically ordered, we should then decode each word back into the corresponding combinatorial object. We give some examples in this section.

Selecting an Encoding for Lexicographic Listing
Suppose we wish to list the subsets of the set $\{a, 2, W\}$ in lexicographic order. First we must decide how to encode a subset such as $\{W, 2\}$ as a word over some ordered alphabet. One way is to use the alphabet $A = \{a, 2, W\}$, and to order these symbols according to, say, their ASCII representation:

$$2 < W < a.$$

To encode a subset, we can sort its elements into $\leq$-increasing order and form a word in A^*. For example, the subset $\{W, 2\}$ would be encoded as the word $[2\ W] \in A^*$. The eight subsets of $\{2, a, W\}$ result in eight words in A^*. We note that the encoding rule is injective, but not bijective. That is, different subsets encode to different words, but some words in A^*, such as $[W\ W]$ and $[W\ 2]$ do not correspond to subsets of $\{a, 2, W\}$. We can now sort the resulting eight words into lexicographic order:

$$[\,] \prec [2] \prec [2\ W] \prec [2\ W\ a] \prec [2\ a] \prec [W] \prec [W\ a] \prec [a]. \tag{3}$$

Finally we decode these words back into subsets, and list these subsets in order:

$$\emptyset, \{2\}, \{2, W\}, \{2, W, a\}, \{2, a\}, \{W\}, \{W, a\}, \{a\}.$$

Another way is to encode a subset $\{W, 2\} \leq \{a, 2, W\}$ by its 3-bit binary indicator word $[b_a\ b_2\ b_W] = [0\ 1\ 1]$. This will give us a different ordering of the words and subsets:

$$[0\ 0\ 0] \prec [0\ 0\ 1] \prec [0\ 1\ 0] \prec [0\ 1\ 1] \prec [1\ 0\ 0] \prec [1\ 0\ 1] \prec [1\ 1\ 0] \prec [1\ 1\ 1]$$

$$\emptyset, \{W\}, \{2\}, \{2, W\}, \{a\}, \{a, W\}, \{a, 2\}, \{a, 2, W\}.$$

We see that the encoding rule can greatly affect the resulting lexicographic ordering. Once we have defined an encoding rule, we may try to find useful routines such as successor and predecessor rules for the ordering (3). We investigate the above two encodings and encodings of other combinatorial objects below.

All Subsets of $[n]$ in Dictionary Order

It is convenient to assume that the set whose subsets we wish to list is $[n] = \{0, 1, \ldots, n-1\}$. For listing them in *dictionary order* we encode each subset of $[n]$ as a word in $([n], \leq)$ as described in the first part of the previous section. For example the subset $\{2, 0\} \subseteq [3]$ is encoded as the word [0 2]. We seek a successor rule for the induced lexicographic ordering. For example, if $n = 3$ we would like a successor rule for the ordering

$$[\,] \prec [0] \prec [0\ 1] \prec [0\ 1\ 2] \prec [0\ 2] \prec [1] \prec [1\ 2] \prec [2].$$

Informally, to find the successor of a subset whose rightmost element is r, we append the element $r+1$ to the set unless $r = n-1$, in which case we delete r from the set, and increment the previous element. The predecessor rule is roughly the reverse of this. We formally describe here these two rules.

RULE 2.2 *Let $\alpha = [a_1\ a_2\ \ldots\ a_k]$ be the encoding of a subset of $[n]$, encoded such that $a_1 \leq a_2 \leq \cdots \leq a_k$ as described above.*
The lexicographic **successor** *of α is determined as follows:*

1. *if $\alpha = [\,]$, then output $[0]$.*
2. *if $k \geq 1$ and $a_k < n-1$, then output $[a_1\ a_2\ \ldots\ a_k\ (a_k+1)]$.*
3. *if $\alpha = [n-1]$, then output* `No Successor`.
4. *if $k \geq 2$ and $a_k = n-1$, then output $[a_1\ a_2\ \ldots\ a_{k-2}\ (a_{k-1}+1)]$.*

The lexicographic **predecessor** *of α is determined as follows:*

1. *if $\alpha = [\,]$, then output* `No Predecessor`.
2. *if $\alpha = [0]$, then output $[\,]$.*
3. *if $k = 1$ and $a_1 > 0$, then output $[(a_1 - 1)\ (n-1)]$.*
4. *if $k \geq 2$ and $a_k = a_{k-1} + 1$, then output $[a_1\ a_2\ \ldots\ a_{k-1}]$.*
5. *if $k \geq 2$ and $a_k > a_{k-1} + 1$, then output $[a_1\ a_2\ \ldots\ a_{k-1}\ (a_k - 1)\ (n-1)]$.*

All Subsets of $[n]$ in Binary Counting Order

Here the alphabet is $A = \{0, 1\}$ with $0 < 1$ and we encode each subset of $[n]$ by its n-bit indicator vector $[a_{n-1}\ a_{n-2}\ \ldots\ a_0]$. Now lexicographic ordering is just the regular counting order of binary numbers. The successor and predecessor rules are just the rules for adding and subtracting 1 in binary.

RULE 2.3 *Let $\alpha = [a_{n-1}\ a_{n-2}\ \ldots\ a_0]$ be the binary indicator vector encoding of a subset $S \subseteq [n]$, where $a_k \in \{0, 1\}$ indicates whether $k \in S$ as described above.*
The lexicographic **successor** *of α is determined as follows:*

1. *if $\alpha = [1\ 1\ \ldots\ 1]$, then output* `No Successor`.
2. *find the least (rightmost) index k such that $a_k = 0$ and output $[a_{n-1}\ a_{n-2}\ \ldots\ a_{k+1}\ 1\ 0\ \ldots\ 0]$.*

The lexicographic **predecessor** *of α is determined as follows:*

1. *if $\alpha = [0\ 0\ \ldots\ 0]$, then output* `No Predecessor`.
2. *find the least (rightmost) index k such that $a_k = 1$ and output $[a_{n-1}\ a_{n-2}\ \ldots\ a_{k+1}\ 0\ 1\ \ldots\ 1]$.*

All k-Subsets of $[n]$ in Dictionary Order

Again we assume our ordered alphabet to be $([n], \leq)$ and encode a k-subset of $[n]$ by writing its elements in increasing order. Thus the lexicographically first subset is encoded by $[0\ 1\ \ldots\ (k-1)]$ and the last subset is encoded by $[(n-k)\ (n-k+1)\ \ldots\ (n-1)]$.

The successor rule can be informally stated as follows. "Find the rightmost digit that can be incremented without duplicating a digit, increase it and pad it on the right with consecutive digits in increasing order." For example, the successor of 5-subset $\{0, 2, 4, 8, 9\} \subseteq [10]$ is found by scanning the word from the right to locate the digit "4", then increasing it and "filling in" on the right to get $\{0, 2, 5, 6, 7\}$. Finding the predecessor requires finding the rightmost digit that can be decremented without duplicating a letter. Formally we write the rules as follows.

RULE 2.4 *Let $\alpha = [a_1\ a_2\ \ldots\ a_k]$ be the encoding of a k-subset of $[n]$, encoded such that $a_1 \le a_2 \le \cdots \le a_k$ as described above.*
The lexicographic **successor** *of α is determined as follows:*

1. *if $a_1 = n - k$, then output* `No Successor`.
2. *find the greatest (rightmost) index i such that $a_i < n - i$*
   and output $[a_1\ a_2\ \ldots\ a_{i-1}\ (a_i + 1)\ (a_i + 2)\ \ldots\ (a_i + k - i + 1)]$.

We leave the calculation of a predecessor rule as an exercise.

All Partitions of n in Dictionary Order
A *partition of n* is multiset of positive integers whose sum is n. The order of the summands does not matter. For example $3+1+1$ and $1+3+1$ are the same partition of 5. There are exactly seven partitions of 5, namely

$$1+1+1+1+1\,,\quad 1+1+1+2\,,\quad 1+1+3\,,\quad 1+2+2\,,\quad 1+4\,,\quad 2+3\,,\quad 5. \tag{4}$$

There is no known simple formula for the number $P(n)$ of partitions of n, so an easy way to count them is simply to generate them all. One way is to list them in all in "dictionary order", as we have done in (4). Each sum is encoded by the word $[a_1\ a_2\ \ldots\ a_k]$ after sorting the summands into nondecreasing order $a_1 + a_2 + \cdots + a_k$, $(a_1 \le a_2 \le \cdots \le a_k)$. The lexicographically fist word is a list of n ones, $[1\ 1\ \ldots\ 1]$, and the last word is $[n]$. A successor rule is not hard to deduce for this ordering. Informally we replace two largest parts, $a_{k-1} + a_k$, with a list of parts $x + x + \cdots + x + y$ which sum to $a_{k-1} + a_k$ such that $x = a_{k-1} + 1$ and $x \le y < 2x$. For example, to find the successor of the partition [1 3 4 4 23] of 35, we set $x = 4 + 1 = 5$. To compute y we divide $4 + 23$ by x and find the remainder $r = 2$. Then we set $y = x + r = 7$. We have found that $4 + 13 = 5 + 5 + 5 + 5 + 7$ and the successor is [1 3 4 5 5 5 5 7]. For another example the successor of [1 3 3 6] is computed by finding $x = 3 + 1 = 4$ and that $3 + 6$ has a remainder of 5 when divided by 4, thus $y = 5 + 4 = 9$. Thus the successor is [1 3 9]. Notice that the list of x's is empty in this case.

RULE 2.5 *Let $\alpha = [a_1\ a_2\ \ldots\ a_k]$ be the encoding of a partition of n, encoded such that $a_1 \le a_2 \le \cdots \le a_k$ as described above.*
The lexicographic **successor** *of α is determined as follows:*

1. *if $\alpha = [n]$, then output* `No Successor`.
2. *Find the unique partition of $a_{k-1} + a_k$ which has the form $x + x + \cdots + x + y$ where $x = a_{k-1} + 1$ and $x \le y < 2x$. (We compute y by dividing x into $a_{k-1} + a_k$, then adding x to the remainder.)*
   Output $[a_1\ a_2\ \ldots\ a_{k-2}\ x\ x\ \ldots\ x\ y]$.

We leave the calculation of the predecessor rule as an exercise.

All Partitions of n into k parts in Dictionary Order
This time we would like to list the set $P(n, k)$ of partitions of n into exactly k distinct parts. Suppose we encode a partition as in the previous section. Then for example $P(11, 6)$ in lexicographic order would be.

$$[1\ 1\ 1\ 1\ 1\ 6],\ [1\ 1\ 1\ 1\ 2\ 5],\ [1\ 1\ 1\ 1\ 3\ 4],\ [1\ 1\ 1\ 2\ 2\ 4],\ [1\ 1\ 1\ 2\ 3\ 3],\ [1\ 1\ 2\ 2\ 2\ 3],\ [1\ 2\ 2\ 2\ 2\ 2] \tag{5}$$

Informally, the lexicographic successor is obtained by scanning backward until we find an entry a_i which differs from a_k by at least two. Then we replace entries $a_i, a_{i+1}, \ldots a_k$ with $a_i + 1$, and add to a_k any remaining amount needed to to bring the sum up to n.

RULE 2.6 *Let $\alpha = [a_1\ a_2\ \ldots\ a_k]$ be the encoding of a partition of n into exactly k parts, encoded such that $a_1 \le a_2 \le \cdots \le a_k$ as described above.*
The lexicographic **successor** *of α is determined as follows:*

1. *if $a_k - a_1 \le 1$, then output* `No Successor`.
2. *if $a_k - a_1 \ge 2$, then locate the greatest (rightmost) index i such that $a_k - a_i \ge 2$.*
   Output $[a_1\ a_2\ \ldots\ a_{i-1}\ (a_i+1)\ (a_i+1)\ \ldots\ (a_i+1)\ r]$ where $r = (a_i + a_{i+1} + \cdots + a_k) - (k-i)(a_i+1)$.

We leave the calculation of the predecessor rule as an exercise.

EXERCISE 2.1

1. *Find and describe a predecessor rule for the k subsets of $[n]$ in dictionary order, as encoded in the above text.*

2. *Find and describe a predecessor rule for the partitions of n in dictionary order, as encoded in the above text.*

3. *Suppose that we encode a k-subset of $[n]$ as a word in $[n^*]$ whose entries are in nonincreasing (rather than nondecreasing) order. For example would encode $\{3,5,5,6\}$ as $[6\ 5\ 5\ 3]$ rather than $[3\ 5\ 5\ 6]$.*

   (a) *Using this encoding, list the 4-subsets of $[6]$ in dictionary order.*

   (b) *Find and describe a successor rule for this ordering of the k-subsets of $[n]$.*

4. *Suppose that we encode a partition of n as a word in $[n]^*$ whose entries are in nonincreasing (rather than nondecreasing) order. For example would encode $1+2+2+3$ as $[3\ 2\ 2\ 1]$ rather than $[1\ 2\ 2\ 3]$.*

   (a) *Using this encoding, list the partitions of 7 in lexicographic order.*

   (b) *Find and describe a successor rule for this ordering of the partitions of n.*

5. *A* partition of the set $[n]$ *is a collection $\{A_1, A_2, \ldots, A_k\}$ of disjoint nonempty subsets $A_i \subseteq [n]$ $(i = 1, \ldots, k)$, whose union is $[n]$. For example, $\{\{1,4\},\{0\},\{2,3,5,6\}\}$ is a partition of $[7]$, which we shall write as $(1,4)(0)(2,3,5,6)$ for convenience. There are exactly five partitions of $[3]$, namely: $(0,1,2)$, $(0)(1,2)$, $(1)(0,2)$, $(2)(0,1)$ and $(0)(1)(2)$. You should notice the difference between the partitions of n and the partitions of $[n]$.*

   (a) *Find an encoding rule for the set of partitions of $[n]$. Be very clear as to what your ordered alphabet $(A, \leq)$ is.*

   (b) *List the partitions of $[4]$ in lexicographic order, using your encoding.*

   (c) *(Bonus) Find and describe a successor rule for the lexicographic ordering of your encoded partitions.*

6. *Suppose you wish to generate all possible simple undirected graphs with vertex set $[n]$ but having no loops nor multiple edges, and where each edge is either red, blue or green. Find an encoding rule for such objects, and give some examples of encoding and decoding.*

7. *Suppose you wish to generate all simple graphs having $[n]$ vertices,* up to isomorphism. *That is, you only wish to list one graph out of each isomorphism class. What problems do you come up against when trying to come up with an encoding rule for such objects? Roughly, how does Brendan McKay's program "nauty" (see* `http://cs.anu.edu.au/people/bdm/nauty/`*) relate to this problem?*