Generating Subsets

September 17, 2001

1 Introduction

The number of subsets of $[n]$ is 2^n because each element in $[n]$ can independently be in or out of a subset. The number of subsets of $[n]$ with exactly k elements is $n!/k!(n-k)!$. To convince yourself of this first compute the number of length k sequences that can be made from the elements in $[n]$. There are

$$n \cdot (n-1) \cdot (n-2) \cdots (n-k+1) = \frac{n!}{(n-k)!}$$

ways of doing this because the first element in the sequence can be picked in n ways, the next can be picked in $(n-1)$ ways and so on. We are interested in subsets, not sequences and each of the $k!$ permutations of a sequence corresponds to the same subset. This implies that the number of size-k subsets of $[n]$ is $n!/k!(n-k)!$. This quantity is denoted by the symbol $\binom{n}{k}$ and often called a *binomial coefficient* because of its role in the *binomial theorem*. We will read this symbol as "n choose k" to emphasize its combinatorial origins.

Mathematica has a function called `Binomial` that returns n choose k. The experiment below illustrates the binomial theorem which tells us that the coefficient of the monomial x^k in the expansion of $(1+x)^n$ is $\binom{n}{k}$ for $k = 0, 1, \ldots, n$.

```
In[1]:= Table[Binomial[10, i], {i, 0, 10}]

Out[1]= {1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1}

In[2]:= Expand[(1+x)^{10}]

                        2         3         4         5         6         7
Out[2]= {1 + 10 x + 45 x  + 120 x  + 210 x  + 252 x  + 210 x  + 120 x  +

           8        9     10
      45 x  + 10 x  + x   }
```

2 Subsets in Lexicographic Order

If we agree to write a subset $[n]$ in increasing order of its elements, then the lexicographic ordering of subsets of $[n]$ is well defined. *Combinatorica* contains a function `LexicographicSubsets` that generates subsets in lexicographic order.

```
In[3]:= LexicographicSubsets[Range[4]]
```

```
Out[3]= {{}, {1}, {1, 2}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2, 4}, {1, 3},

>    {1, 3, 4}, {1, 4}, {2}, {2, 3}, {2, 3, 4}, {2, 4}, {3}, {3, 4}, {4}}
```

An easy recursive algorithm for generating the subsets of $[n]$ in lexicographic order is as follows. Generate the list L of subsets of $\{2, 3, \ldots, n\}$ in lexicographic order. Construct a list L' by prepending 1 to each subset in L. Append the list L to L' and move the empty set to the first position. The code for `LexicographicSubsets`, given below, is a faithful implementation of this algorithm.

```
LexicographicSubsets[{}] := { { } }

LexicographicSubsets[l_List] :=
       Module[{s = LexicographicSubsets[Rest[l]]},
              Join[{{}}, Map[Prepend[#, l[[1]]] &, s], Rest[s]]
       ]
```

Ranking and Unranking. Consider a subset $X \subseteq \{i, i+1, \ldots, n\}$. We will define a function $Rank(X, i, n)$ as the rank of X in the lexicographic ordering of subsets of $\{i, i+1, \ldots, n\}$. If $X = \emptyset$, $Rank(X, i, n) = 0$. Otherwise, consider the cases $i \in X$ and $i \notin X$ separately. If $i \in X$, then $Rank(X, i, n) = 1 + Rank(X - \{i\}, i+1, n)$. If $i \notin X$, then $Rank(X, i, n) = 2^{n-i} + Rank(X, i+1, n)$. The base case is when $i > n$ and in this case $Rank(X, i, n) = 0$. Note that since we are interested in ranking a subset $X \subseteq [n]$, we want to compute $Rank(X, 1, n)$. *Combinatorica* does not provide functions to rank and unrank subsets in lexicographic order, but *Mathematica* code implementing $Rank(X, i, n)$ is given below.

```
MyRankSubset[x_List, i_Integer, n_Integer] := 0 /; (i > n)
MyRankSubset[{}, i_Integer, n_Integer] := 0
MyRankSubset[x_List, i_Integer, n_Integer] :=
  If[MemberQ[x, i], 1 + Rank[Complement[x, {i}], i + 1, n],
    2^(n - i) + Rank[x, i + 1, n]]
```

`MyRankSubset` is tested in the following experiment. The rank of the subset $\{3, 4, 5, 2, 1, 11, 8, 13\}$ in the list of lexicographically ordered subsets of $[14]$ turns out to be 444 and this is confirmed by calling `LexicographicSubsets[Range[14]]` and examining the 445 th element.

```
In[3]:= MyRankSubset[{3, 4, 5, 2, 1, 11, 8, 13}, 1, 14]

Out[3]= 444

In[4]:= l = LexicographicSubsets[Range[14]][[445]]

Out[4]= {1, 2, 3, 4, 5, 8, 11, 13}
```

Given a rank $r \in \{0, 1, \ldots, 2^{n-i+1} - 1\}$, we can unrank it to get the corresponding subset $X \subseteq \{i, i+1, \ldots, n\}$ by essentially reversing the algorithm for rank. If $r = 0$, then $X = \emptyset$. So suppose that $r > 0$. From the discussion of how we compute $Rank(X, i, n)$, we see that r is either $1 + Rank(X', i+1, n)$ or $2^{n-i} + Rank(X', i+1, n)$, where $X' \subseteq \{i+1, i+2, \ldots, n\}$. In the former case, $i \in X$ and $Rank(X', i+1, n) \in \{0, 1, \ldots, 2^{n-i} - 1\}$. In the latter case, $i \notin X$ and $Rank(X', i+1, n) \in \{1, 2, \ldots, 2^{n-i} - 1\}$. This means that if $r \in [2^{n-i}]$ then i belongs to X and the rest of the elements in X can be obtained by unranking $r - 1$. If $r \in \{2^{n-i}+1, \ldots, 2^{n-i+1} - 1\}$ then i does not belong to X and the elements of X can be obtained by unranking $r - 2^{n-i}$. I'll leave the implementation of this idea as an exercise for you.

Random Subset. Generating a random subset of $[n]$ is easy. Flip a coin independently for each element $i \in [n]$ to decide if i should belong to the subset. The result is a subset $X \subseteq [n]$ selected with probability $1/2^n$.

3 Gray Codes

There is an obvious bijection between subsets of $[n]$ and binary n-vectors (that is, elements of $\{0,1\}^n$). To each subset $X \subseteq [n]$ associate a binary n-vector $v = (v_1, v_2, \ldots, v_n)$ such that $i \in X$ if and only if $v_i = 1$. This implies that generating subsets of $[n]$ is equivalent to generating binary n-vectors. These binary n-vectors can be generated in many different orders. For example, they can be generated in lexicographic order or in the increasing order of their decimal equivalents. But the ordering of these binary vectors that is most well known is the *binary reflected Gray code*. This ordering is named after Frank Gray, a researcher at Bell Labs who received a patent for Gray codes in 1953. The problem he was trying to solve was the following. Suppose you want to transmit a finite string of bits using an analogue transmission device. You might take the bit string, compute the decimal equivalent, and transmit a signal of that strength or frequency. A small error in the signal strength means a small change in the number received. However, a small change in the number received could mean a huge error in the corresponding bit string. For example, if 256 is the intended number and 255 is received, that means a difference in 9 bits. Gray solved this problem by finding an ordering on the set of binary n-vectors in which each binary vector differs from the previous in exactly one bit. Then it is simply a matter of transmitting the rank of a binary n-vector and then unranking it at the receiver's end.

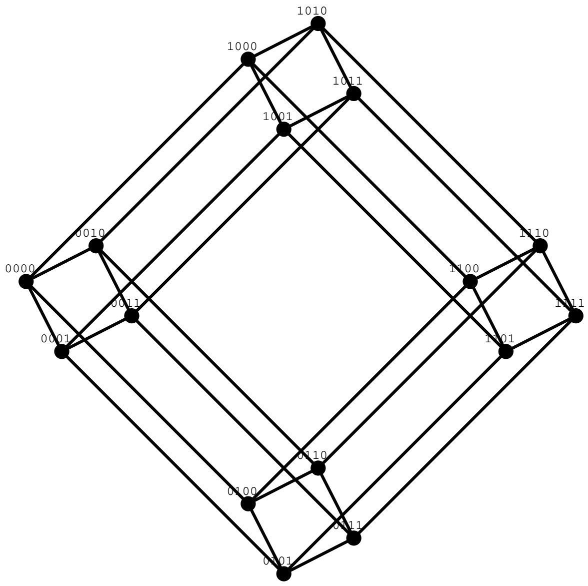
To generate binary n-vectors in Gray code order, first generate the list L of binary $(n-1)$-vectors in Gray code order. Let L_0 be the list obtained from L by prepending 0 to each vector in L and let L_1 be the list obtained from L by prepending 1 to each vector in L. Append to L_0 the reverse of L_1 to get the list of binary n-vectors. To see that the resulting list satisfies the minimum change property, note that within L_0 and within L_1, the left most element is fixed and our induction hypothesis tells us that the remaining $(n-1)$ bits differ only in 1 position from one vector to the next. At the boundary between L_0 and L_1, because of reversing L_1, all elements except the first are identical and the first element changes from 0 to 1. The Gray code order corresponds to listing subsets in an order such that each subset is obtained from the previous by adding or deleting one element. *Combinatorica* contains a function `GrayCodeSubsets` that lists subsets in Gray code order.

```
In[5]:= GrayCodeSubsets[4]

Out[5]= {{}, {4}, {3, 4}, {3}, {2, 3}, {2, 3, 4}, {2, 4}, {2}, {1, 2},
        {1, 2, 4}, {1, 2, 3, 4}, {1, 2, 3}, {1, 3}, {1, 3, 4}, {1, 4}, {1}}
```

The code for `GrayCodeSubsets` is given below.

```
GrayCodeSubsets[ { } ] := { {} }

GrayCodeSubsets[l_List] :=
       Module[{s = GrayCodeSubsets[Take[l, Length[l] - 1]]},
              Join[s,  Map[Append[#, Last[l]] &, Reverse[s]]]
       ]
```

Define the *n-dimensional hypercube* as a graph $H_n = (V_n, E_n)$ with vertex set $V_n = \{0,1\}^n$ and edge set E_n containing edges that connect vertices that differ in exactly one bit. The 4-dimensional hypercube is shown below.



A Hamiltonian path in H_n corresponds to a minimum change ordering on binary n-vectors. Therefore, the Gray code construction proves that H_n has a Hamiltonian path. In fact, the first and the last vectors in the Gray code ordering also differ by a single bit and therefore the Gray code construction proves that H_n has a Hamiltonian cycle. Of course, there is no reason to believe that the Gray code ordering is the only Hamiltonian cycle in the graph. The following experiment shows that the binary reflected Gray code is just one of 2688 distinct Hamiltonian cycles. The question: how many distinct Hamiltonian cycles does H_n have, is open. No one knows the number of distinct Hamiltonian cycles in H_n, even asymptotically!

```
In[6]:= Length[ HamiltonianCycle[ Hypercube[4], All] ]

Out[6]= 2688
```

Ranking and Unranking Gray codes. Given a binary vector $v = (v_1, v_2, \ldots, v_n)$, how do we compute its rank in Gray code order? Suppose that v has rank r. There is a pretty connection between the binary representation of r and the bits in v that provides the basis for algorithms for ranking and unranking Gray codes. The connection is expressed by the formula

$$v_i = (b_i + b_{i-1}) \bmod 2 \tag{1}$$

that holds for all $i = 1, 2, \ldots, n$, assuming that $b_0 = 0$. This immediately provides a way of unranking r to obtain a binary n-vector v. Compute the binary representation $(b_1, b_2, \ldots, b_n)$ of r and then use formula (1) to compute the v_i's. It is also easy to invert the formula to get the equation

$$b_i = (v_i + v_{i-1} + \cdots + v_1) \bmod 2 \tag{2}$$

that holds for all $i = 1, 2, \ldots, n$. Using this formula, we can compute the rank of a binary n-vector in Gray code.

Here is an experiment the shows the above connection between the v_i's and the b_i's. Each row in the matrix shown below consists of two vectors. The first element in row i is the binary 4-vector v of rank $i-1$ and the second element is the binary representation b of $i-1$. You can verify that $v_i = (b_i + b_{i-1}) \bmod 2$ for the vectors v and b in each row.

```
In[7]:= Transpose[{Map[ Table[If[MemberQ[#, i], 1, 0], {i, 4}] &,
          GrayCodeSubsets[Range[4]]] ,
        Table[IntegerDigits[i, 2, 4], {i, 0, 15}]}] // ColumnForm

Out[7]= {{0, 0, 0, 0}, {0, 0, 0, 0}}
        {{0, 0, 0, 1}, {0, 0, 0, 1}}
        {{0, 0, 1, 1}, {0, 0, 1, 0}}
        {{0, 0, 1, 0}, {0, 0, 1, 1}}
        {{0, 1, 1, 0}, {0, 1, 0, 0}}
        {{0, 1, 1, 1}, {0, 1, 0, 1}}
        {{0, 1, 0, 1}, {0, 1, 1, 0}}
        {{0, 1, 0, 0}, {0, 1, 1, 1}}
        {{1, 1, 0, 0}, {1, 0, 0, 0}}
        {{1, 1, 0, 1}, {1, 0, 0, 1}}
        {{1, 1, 1, 1}, {1, 0, 1, 0}}
        {{1, 1, 1, 0}, {1, 0, 1, 1}}
        {{1, 0, 1, 0}, {1, 1, 0, 0}}
        {{1, 0, 1, 1}, {1, 1, 0, 1}}
        {{1, 0, 0, 1}, {1, 1, 1, 0}}
        {{1, 0, 0, 0}, {1, 1, 1, 1}}
```

How do we prove the connection expressed in formula (1)? For $n = 1$, the formula is true. We suppose that the formula is true for some $n-1 > 0$ and prove it for n. So let $v = (v_1, v_2, \ldots, v_n)$ be a binary n-vector of rank $r \in \{0, 1, \ldots, 2^n - 1\}$ and let $b = (b_1, b_2, \ldots, b_n)$ be the binary representation of r. There are two cases, depending on whether $v_1 = 0$ or $v_1 = 1$.

(i) Suppose that $v_1 = 0$. This implies that v is in the first half of the Gray code, which in turn implies that $r \in \{0, 1, \ldots, 2^{n-1} - 1\}$, which in turn implies that $b_1 = 0$. This means that $v_1 = (b_1 + b_0) \bmod 2$ and we only have to verify formula (1) for $i = 2, 3, \ldots, n$. Now note that the $(n-1)$-binary vector $(v_2, v_3, \ldots, v_n)$ has rank r and r has binary representation $(b_2, b_3, \ldots, b_n)$. The induction hypothesis tells us that $v_i = (b_i + b_{i-1}) \bmod 2$ for $i = 2, 3, \ldots, n$.

(ii) Suppose that $v_1 = 1$. This implies that v is in the second half of the Gray code, which in turn implies that $r \in \{2^{n-1}, \ldots, 2^n - 1\}$, which in turn implies that $b_1 = 1$. This means that $v_1 = (b_1 + b_0) \bmod 2$ and we only have to verify formula (1) for $i = 2, 3, \ldots, n$. Because the second half of the Gray code is obtained by "reflecting" a copy of the Gray code for binary $(n-1)$-vectors, the binary $(n-1)$-vector $(b_2, b_3, \ldots, b_n)$ has rank $2^{n-1} - r$. Since r has representation $(b_1, b_2, \ldots, b_n)$, $2^{n-1} - r$ has representation $(1 - b_1, 1 - b_2, \ldots, 1 - b_n)$. The induction hypothesis tells us that

$$v_i = ((1 - b_i) + (1 - b_{i-1})) \bmod 2 = (b_i + b_{i-1}) \bmod 2$$

for $i = 2, 3, \ldots, n$.

Combinatorica has functions `RankGrayCodeSubset` and `UnrankGrayCodeSubset` that implement formulae (1) and (2). In the experiment below, we compute the millionth subset of [100] in Gray code order and rank it to get 999999.

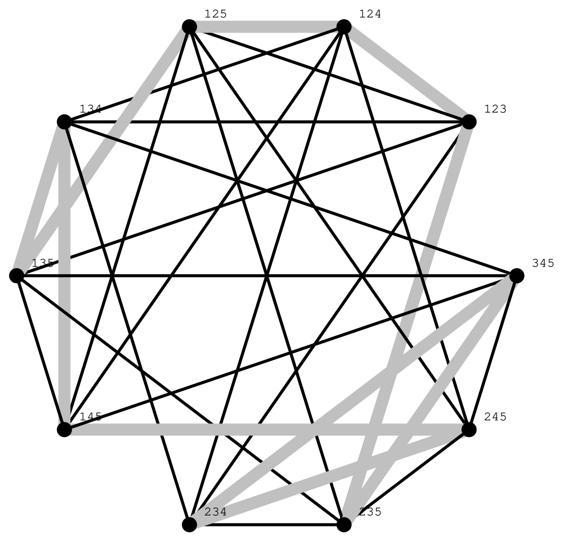
```
In[8]:= UnrankGrayCodeSubset[999999, Range[100]]

Out[8]= {81, 85, 86, 87, 91, 92, 95}

In[9]:= RankGrayCodeSubset[Range[100], %]

Out[9]= 999999
```

4 Generating k-subsets

Let us now turn to the question of generating all subsets of $[n]$ of a fixed size k. We will use the term *k-subsets* to refer to a subset of size k. Generating k-subsets in lexicographic order is easy. Implement the recursive algorithm underlying the well-known identity of binomial numbers

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

and we are done. The more interesting question is whether k-subsets can be generated in some minimum change order, similar to the Gray code order we used for subsets. The change that was allowed in going from one subset to the next in Gray code order was a single insertion or a single deletion. A single insertion or a single deletion will change the size of a subset and hence we need to allow a larger change than that in order to go from one k-subset to another k-subset. This motivates the question: can k-subsets be listed in an order so that in going from one k-subset to the next we perform exactly one insertion and one deletion?



The above figure shows a graph whose vertices are 3-subsets of [5] and whose edges connect pairs of 3-subsets that can be obtained from each other by a single insertion followed by a single deletion. The figure also shows a Hamiltonian cycle in the graph indicating that it is possible to list 3-subsets of [5] in minimum change order. In the example below, this graph is constructed for 3-subsets of [6] and tested for Hamiltonicity.

```
In[10]:= HamiltonianQ[
            MakeGraph[KSubsets[Range[6], 3],
                     ((Length[Complement[#1, #2]] === 1) &&
                     (Length[Complement[#2, #1]] === 1)) &
            ]
         ]

Out[10]= True
```

As these examples indicate, yes it is possible to enumerate k-subsets in minimum change order. More importantly, there is a simple algorithm to do this. Let $L_{n,k}$ denote the list of k-subsets of $[n]$ written in minimum change order with the k-subset $\{1, 2, \ldots, k\}$ being the first and the k-subset $\{1, 2, \ldots, n-1, k\}$ being the last. Then, $L_{n,k}$ can be constructed from $L_{n-1,k}$ and $L_{n-1,k-1}$ as follows. Append n to each subset in $L_{n-1,k-1}$, reverse this list, and append it to $L_{n-1,k}$. Within each list, $L_{n-1,k}$ and $L_{n-1,k-1}$ the minimum change condition is satisfied. The induction hypothesis tells us that $L_{n-1,k}$ starts with $\{1, 2, \ldots, k\}$ and ends with $\{1, 2, \ldots, k-1, n-1\}$. After we append n and reverse it, the list $L_{n-1,k-1}$, starts with $\{1, 2, \ldots, k-2, n-1, n\}$ and ends with $\{1, 2, \ldots, k-1, n\}$. So the two subsets at the boundary between the two lists are $\{1, 2, \ldots, k-1, n-1\}$ and $\{1, 2, \ldots, k-2, n-1, n\}$. The latter can be obtained from the former by the deletion of $k-1$ and the insertion of n. *Combinatorica* has a function `GrayCodeKSubsets` that implements this function. Here is an example illustrating this function.

```
In[11]:= GrayCodeKSubsets[6, 3]

Out[11]= {{1, 2, 3}, {1, 3, 4}, {2, 3, 4}, {1, 2, 4}, {1, 4, 5}, {2, 4, 5},

>    {3, 4, 5}, {1, 3, 5}, {2, 3, 5}, {1, 2, 5}, {1, 5, 6}, {2, 5, 6},

>    {3, 5, 6}, {4, 5, 6}, {1, 4, 6}, {2, 4, 6}, {3, 4, 6}, {1, 3, 6},

>    {2, 3, 6}, {1, 2, 6}}
```