Combination Encoding

Nicholas Ormrod

University of Waterloo

njormrod@uwaterloo.ca

1. Problem

For a given n and k, there are $\binom{n}{k}$ ways to choose k of n elements. To space-efficiently represent an arbitrary choice of elements, it is desirable to encode choices as numbers between 0 and $\binom{n}{k} - 1$. The encoding should be efficiently computable and decodable.

... (motivation, abstract)

The reverse lexicographical encoding has several nice properties and allows for efficient computation.

	Encoding		Choice
	5	$\equiv$	$\{a_1, a_2\}$
$\binom{4}{2}$	4	$\equiv$	$\{a_1, a_3\}$
	3	$\equiv$	$\{a_1, a_4\}$
$a_1, a_2,$	2	$\equiv$	$\{a_2, a_3\}$
a_3, a_4	1	$\equiv$	$\{a_2, a_4\}$
	0	$\equiv$	$\{a_3, a_4\}$

Figure 1. Reverse lexicographical encoding for two elements chosen from a_1, a_2, a_3, a_4

1.1 Pascal's Diagonal

In the reverse lexicographical encoding, how many of the $\binom{n}{k}$ choices start with the first element, a_1? Quite simply, if a_1 is selected, then the other $k - 1$ selections must be made from the remaining $n - 1$ elements. This can be done in $\binom{n-1}{k-1}$ ways.

More generally, how many of the $\binom{n}{k}$ choices start with the i'th element? The lowest selected element is a_i, so the other $k - 1$ selections must be made from among the $n - i$ greater elements. This can be done in $\binom{n-i}{k-1}$ ways.

Since the first selected element is between a_1 and a_{n-k+1}, the following equation must be true:

$$\binom{n}{k} = \sum_{i=1}^{n-k+1} \binom{n-i}{k-1} \tag{1}$$

Visually, on Pascal's triangle, the underlined cell is the sum of the bolded cells:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & \mathbf{1} & & & \\
 & & 1 & & 3 & & \mathbf{3} & & 1 & & \\
 & 1 & & 4 & & \mathbf{6} & & 4 & & 1 & \\
1 & & 5 & & 10 & & \underline{\mathbf{10}} & & 5 & & 1
\end{array}$$

Mathematically, this property follows by Pascal's rule and induction.

Corollary: In Pascal's triangle, the cummulative sum of the i'th diagonal is represented by the $(i + 1)$'th diagonal.

2. Encoding

Since the encoding is lexicographic, the encodings are grouped foremost by their first selection. Hence a choice whose first selection is a_i has a larger encoding than all choices whose first selection is $a_{j>i}$.

We may use equation 1 to determine the number of choices whose first element is greater than a_i:

$$\begin{aligned}
& \{\text{\# of choices whose first selection is greater than } a_i\} \\
= & \sum_{j=i+1}^{n-k+1} \{\text{\# of choices whose first selection is } a_j\} \\
= & \sum_{j=i+1}^{n-k+1} \binom{n-j}{k-1} \\
= & \sum_{j=1}^{n-k+1-i} \binom{n-(j+i)}{k-1} \\
= & \binom{n-i}{k}
\end{aligned}$$

Therefore the set of all choices that start with a_i have encodings starting at $\binom{n-i}{k}$.

Within the group of choices that start with a_i, the relative ordering of two different choices can be determined by the lexicographic ordering of their tails. That is to say, the choice $\{a_i, a_{v_2}, \ldots, a_{v_k}\}$ preceeds $\{a_i, a_{w_2}, \ldots, a_{w_k}\}$ if and only if $\{a_{v_2}, \ldots, a_{v_k}\}$ preceeds $\{a_{w_2}, \ldots, a_{w_k}\}$.

Due to the reversing of the lexicographical ordering, the set of all possible tails is the same as the set of all $(k-1)$-choices whose encodings are between 0 and $\binom{n-i}{k-1} - 1$. This means we may use recursion to compute the offset within the group.

Algorithm 1 Recursive Encode

def $E(n, k, \{a_{v_1}, a_{v_2}, \ldots, a_{v_k}\})$

1: **if** $k = 0$ **then**
2:     **return** 0
3: **else**
4:     $groupstart \leftarrow \binom{n-v_1}{k}$
5:     $offset \leftarrow E(n, k-1, \{a_{v_2}, \ldots, a_{v_k}\})$
6:     **return** $groupstart + offset$
7: **end if**

In practice, one simple yet effective optimization is to change the base case to $k = 1$. When $k = 1$, the encoding e simply corresponds to the choice $\{n - e\}$.

Algorithm 2 Recursive Decode

def $D(n, k, e)$
if $k = 0$ **then**
  return []
else
  $v \leftarrow \min\{i : \binom{n-i}{k} \leq e\}$
  $offset \leftarrow e - \binom{n-v}{k}$
  return $a_v : D(n, k-1, offset)$
end if

2.1 Deltas

In the reverse lexicographical encoding, a lower encoding corresponds to a greater lexeme. This means that the lowest encoding always corresponds to the greatest lexeme: $\{a_{n-k+1}, \ldots, a_n\}$.

In general, if encoding e corresponds to $\{a_{v_1}, \ldots, a_{v_k}\}$ with respect to n and k, then it corresponds to $\{a_{v_1+j}, \ldots, a_{v_k+j}\}$ with respect to $(n+j)$ and k. Thus, to encode all possible k-choices of the n elements $\{a_{j+1}, a_{j+2}, \ldots, a_{j+n}\}$, we may use the same encoding and decoding function by replacing n with $(n+j)$. This is particularly useful to get a zero-indexed implementation to work with one-indexed data.

If encoding e corresponds to $\{a_{v_1}, \ldots, a_{v_k}\}$ with respect to n and k, where each v_i is at least $j+1$, then e corresponds to $\{a_{v_1-j}, \ldots, a_{v_k-j}\}$ with respect to $(n-j)$ and k. This subtractive encoding can be used to work with deltas. That is to say, the choice $\{a_{v_1}, a_{v_2}, \ldots, a_{v_k}\}$ is represented by $\{(v_1 - 0), (v_2 - v_1), \ldots, (v_n - v_{n-1})\}$. This is particularly useful when dealing with compositions of fixed size, such as the representation of dice rolls.

Encoding		Choice		Deltas
5	$\equiv$	$\{a_1, a_2\}$	$\equiv$	$\{1, 1\}$
4	$\equiv$	$\{a_1, a_3\}$	$\equiv$	$\{1, 2\}$
3	$\equiv$	$\{a_1, a_4\}$	$\equiv$	$\{1, 3\}$
2	$\equiv$	$\{a_2, a_3\}$	$\equiv$	$\{2, 1\}$
1	$\equiv$	$\{a_2, a_4\}$	$\equiv$	$\{2, 2\}$
0	$\equiv$	$\{a_3, a_4\}$	$\equiv$	$\{3, 1\}$

Figure 2. Delta Representation

Algorithm 3 Recursive Encode Deltas

def $E_\delta(n, k, \{\delta_1, \delta_2, \ldots, \delta_k\})$
if $k = 0$ **then**
  return 0
else
  $groupstart \leftarrow \binom{n-\delta_1}{k}$
  $offset \leftarrow E_\delta(n - \delta_1, k-1, \{\delta_2, \ldots, \delta_k\})$
  return $groupstart + offset$
end if

3. Observations

1. Any encoding or decoding algorithm takes $\Omega(k)$ time, since they accept as input or produce as output a list of k values.
2. Any specific k-choice of n elements can be represented by the $(n-k)$-choice of elements which were not selected - its dual. We may convert a choice to its dual in $O(n)$ time.
3. We may assume that $k \leq \frac{n}{2}$. If not, then we may convert to/from the dual space in $O(n) \leq O(2k) \leq \Omega(k)$ time.
4. Computing binomial coefficients from scratch, according to [1], takes $O(\min(k, n-k))$ time. Since we may assume that $k \leq \frac{n}{2}$, this is just $O(k)$.
5. $\log \binom{n}{k} = \Theta(k \log \frac{n}{k})$ when $k \leq \frac{n}{2}$.
6. The function $i \mapsto \binom{n-i}{k}$ is monotone. Hence finding the minimum i such that $\binom{n-i}{k} \leq e$, as in algorithm 2, may be accomplished with a search algorithm.

4. Analysis

The slowest component of the reverse lexicographical encoding is the need to compute binomial coefficients. If the binomial coefficients were pre-computed, then encoding can be done in $O(k)$ time and decoding can be done in $O(k \log \frac{n}{k})$ time. Given that the input sizes for these two functions are k and $\log \binom{n}{k}$, respectively, these are excellent runtimes.[1]

An alternate approach exists that takes $O(n)$ time, including time spent computing binomial coefficients.

Full C++ source code and tests for each method can be found at **??**.

4.1 Weighted Binary Search

The encoding algorithm, algorithm 1, contains k recursive calls. Each call computes one binomial coefficient, and does $O(1)$ extra work. Thus, if the binomial is computable in $O(1)$, encode is $O(k)$.

The decode algorithm, algorithm 2, is slightly different. It needs to search for the smallest i such that $\binom{n-i}{k} \leq e$. A regular binary search will take $O(\log n)$ steps; however, a slight modification will produce an amortized $O(\log \frac{n}{k})$.

When binary searching for the j'th element, v_j, make the first partition at $l + \frac{n}{k}$, where l, a lower bound for v_j, is v_{j-1}. If the check passes, implying $v_j \leq l + \frac{n}{k}$, then $v_j \in \left(l, l + \frac{n}{k}\right]$ - a range of size $\frac{n}{k}$. If the check fails, implying $v_j > l + \frac{n}{k}$, then v_j and all future elements are greater than $l + \frac{n}{k}$ - update the lower bound to $l + \frac{n}{k}$ and repeat.

Algorithm 4 Recursive Binary Decode

def $D(n, k, e, l = 0)$
if $k = 0$ **then**
  return []
else if $\binom{n-(l+\frac{n}{k})}{k} > e$ **then**
  return $D(n, k, e, l + \frac{n}{k})$
else
  $v \leftarrow \ldots$ // binary search for $\min\{i : \binom{n-i}{k} \leq e\}$ in the range $\left(l, l + \frac{n}{k}\right]$
  $offset \leftarrow e - \binom{n-v}{k}$
  return $v : D(n, k-1, offset, v)$
end if

In this method, when a partition finally passes, a regular binary search finds the correct value in at most $\log \frac{n}{k}$ further iterations. Hence this method performs $O(k \log \frac{n}{k} + \{\text{\# of fails}\})$ total steps. But each failure increases the lower bound by $\frac{n}{k}$.[2] Since the upper bound is n, and the initial lower bound is 0, there can be at most k failures. The total cost is therefore $O(k \log \frac{n}{k})$ to find all k elements.

[1] Technically, encode should alse be $O(\log \binom{n}{k})$, since it returns an encoding of that many bits.

[2] Or $\frac{n}{k-i}$, which is greater than $\frac{n}{k}$.

4.2 Bottom-Up Linear Search

When the binomial coefficients must be computed on the fly, the regular methods slow down by a factor of k. In these circumstances, the unlikely hero is linear search.

When using linear search to find the smallest i such that $\binom{n-i}{k} \leq e$, the value $\binom{n-i}{k}$ may be computed incrementally in $O(1)$ time, under the identity

$$\binom{n}{k} \quad = \quad \frac{n}{n-k+1} \cdot \binom{n-1}{k}$$

The coefficients may even be reused in the recursive calls. When the j'th element is determined to be i, then the $j+1$'th element must be at least $i+1$. Hence the first binomial in the $j+1$'th linear search may be computed from the last binomial in the j'th search, under the identity

$$\binom{n}{k} \quad = \quad \frac{k+1}{n+1} \cdot \binom{n+1}{k+1}$$

The first coefficient of the first search, $\binom{n-1}{k}$, must be computed in $O(k)$ time.

This pattern of reusing the coefficients has the property that, on each update, the numerator decreases by 1. Since $O(1)$ work is done for each coefficient, and there are n valid numerators, decode by linear search takes $O(n+k) = O(n)$ time.

A similar approach may be used to encode sequences.

Algorithm 5 Imperative Linear Decode

def $D(n, k, e)$
- $ret \leftarrow []$
- $j \leftarrow 1$ // lower bound for the next element
- $c \leftarrow \binom{n-j}{k}$ // invariant: $c = \binom{n-j}{k-i+1}$
- **for** $i \leftarrow 1..k-1$ **do**
  - **while** $c > e$ **do**
    - $c \leftarrow \frac{c \cdot (n-j-k+i-1)}{n-j}$
    - $j \leftarrow j+1$
  - **end while**
  - $ret.push\_back(j)$
  - $e \leftarrow e - c$
  - $c \leftarrow \frac{c \cdot (k-i+1)}{n-j}$
  - $j \leftarrow j+1$
- **end for**
- $ret.push\_back(n-e)$ // $k = 1$ optimization
- **return** ret

5. Conclusion

The reverse lexicographical encoding may be used to compactly and efficiently encode and decode specific combinations.

With cached coefficients ($O(k^2)$ space overhead) the encoding runs in $O(k)$ time and decoding runs in $O(k \log \frac{n}{k})$ time. In the absence of cached coefficients both encoding and decoding can be run in $O(n)$ time.

The algorithms are short, easy to implement, and easy to test. They may easily be modified to deal with encodings of different indices, or deltas.

Code may be found at **??**.

References

[1] Yannis Manolopoulos, *Binomial Coefficient Computation: Recursion or Iteration?* SIGCSE Bulletin, Vol 34, No. 4, 2002 December.

[2] Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed. Cambridge: The MIT Press, 2009.