Generating All k-subsets Of a n-set

Blaž Sovdat*

March 20, 2014

Abstract

This note describes a simple algorithm for generating all subsets of size k of a given set of n elements.

1 Introduction

One of the fundamental algorithmic combinatorial problems is generation of all subsets of a given set. Many efficient algorithms are known for this problem. In this note we restric ourselves to generating all subsets of size k of a given set of size n.

In the next section we describe a very simple algorithm for generating all $\binom{n}{k}$ subsets of size k, denoted k-subsets, of a given n-set in lexiographic order,[1]. We also include C implementation of the algorithm, assuming, without loss of generality, that a n-set is $[n] := \{0, 1, \ldots, n-1\}$. (We can always use elements of $[n]$ as indices of some other "set" of size n.)

2 Simple algorithm for generating all k-subsets of a n-set

In this section we give a simple algorithm that, given a n-set, finds all its k-subsets.

2.1 Informal description.

We now give intuitively appealing description of algorithm 1 using binary string α with k ones and $n-k$ zeros, with nonzero indices of α indicating members of the k-subset. The problem of generating all k-subsets of n-set is the one of generating all binary strings $\alpha \in \{0,1\}^n$ of length n with exactly k ones. We can think our algorithm starts with string $\alpha_0 := 11\ldots10\ldots00$ with k initial ones, followed by $n-k$ zeros. (The algorithm does not store α explicitly; see the next subsection.)

Suppose we are in m-th iteration and that α_m is the current k-subset. We then find the rightmost nonzero index i (the largest i such that $\alpha_m[i] = 1$ and there is $j > i$ with $\alpha_m[j] = 0$) that we can move to the right. We now move this element to the right (the next configuration will thus be $\alpha_{m+1}[i] = 0$ and $\alpha_{m+1}[i+1] = 1$) and shift all elements $\ell > i$ "as left as possible". (See example below.) The resulting α_{m+1} represents a new k-subset.

We keep repeating the above step until $\alpha_m[n-1] = \alpha_m[n-2] = \ldots = \alpha_m[n-k+1] = 1$, which is (lexiographically) the last k-subset.

The following example illustrates how our algorithm would generate all 2-subsets of a 5-set.

*Email: `blaz.sovdat@gmail.com`

[1] Lexiographic order from the viewpoint of indices.

Example. Consider set $S := \{a, b, c, d, e\}$ and suppose we want to generate all pairs of element from S. Below are α's that the algorithm would generate, with S_α being the set that α represents. (For now ignore the array p; it is what algorithm actually keeps during its execution; see the next subsection.)

Step 1) $\alpha = 11000$; $p = [0, 1]$; $S_\alpha = \{a, b\}$.

Step 2) $\alpha = 10100$; $p = [0, 2]$; $S_\alpha = \{a, c\}$.

Step 3) $\alpha = 10010$; $p = [0, 3]$; $S_\alpha = \{a, d\}$.

Step 4) $\alpha = 10001$; $p = [0, 4]$; $S_\alpha = \{a, e\}$.

Step 5) $\alpha = 01100$; $p = [1, 2]$; $S_\alpha = \{b, c\}$.

Step 6) $\alpha = 01010$; $p = [1, 3]$; $S_\alpha = \{b, d\}$.

Step 7) $\alpha = 01001$; $p = [1, 4]$; $S_\alpha = \{b, e\}$.

Step 8) $\alpha = 00110$; $p = [2, 3]$; $S_\alpha = \{c, d\}$.

Step 9) $\alpha = 00101$; $p = [2, 4]$; $S_\alpha = \{c, e\}$.

Step 10) $\alpha = 00011$; $p = [3, 4]$; $S_\alpha = \{d, e\}$.

2.2 Formal description

For its working, the algorithm does not need to store α explicitly. It suffices to keep indices of nonzero entries in an array p of size k.

Algorithm 1 Generating all k-subsets of a given n-set in lexiographic order.

Let $p := [0, 1, \dots, k-1]$.
while true **do** ▷ Forever
    Visit current k-subset $\{S_{p[i]} \mid 0 \le i < k\}$.
    if $p[0] = n - k$ **then break** ▷ This was the last subset; we are done
    Find $i := \max\{i \mid p[i] + k - i \ne n\}$, the rightmost element we can still move to the right.
    Set $j := 2$ and $r := p[i]$, and increment $p[i] := p[i] + 1$.
    for $i < \ell < k$ **do**
        Set $p[\ell] := r + j$ and then increment $j := j + 1$. ▷ Also note that we have $j = 2 + (\ell - (i + 1))$

2.3 Analysis

We now give simple analysis of the above algorithm. TODO: Proof of corretness; running time.

2.4 Implementation.

Listing 1 is C implementation of algorithm 1.

Listing 1: C implementation of algorithm 1

```c
#include<stdlib.h>
#include<stdio.h>

void subs(int n, int k);

int main(int argc, char **argv)
{
  if(argc != 3) return 1;
  int n, k;

  n = atoi(argv[1]); k = atoi(argv[2]);
  subs(n, k);

  return 0;
}

void subs(int n, int k)
{
  int *p = (int *)malloc(sizeof(int)*k);
  int i, j, r;

  for(i = 0; i < k; ++i) p[i] = i; // initialize our ``set''
  // the algorithm
  while(1)
   { // visit the current k-subset
    for(i = 0; i < k; ++i)
      printf("%d ", p[i]+1);
    printf("\n");

    if(p[0] == n-k) break; // if this is the last k-subset, we are done

    for(i = k-1; i >= 0 && p[i]+k-i == n; --i); // find the right element
    r = p[i]; ++p[i]; j = 2; // exchange them
    for(++i; i < k; ++i, ++j) p[i] = r+j; // move them
   }
  free(p);
}
```

Compile the above code with `g++ -O3 subs.c -o subs`.

Acknowledgements

I discovered this algorithm on a long train ride home in 2011, but only now decided to write it up as a note. I thank Uroš Čibej for pointing out that essentially the same algorithm is already described in Knuth's TAOCP Vol. 4, Fasc. 3.