

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on September 17, 2017)

**1. Intro.** A quick program to output the “domination” or “majorization” relation when it is defined on permutations of multisets instead of on partitions.

Let’s say that digits are permuted. Then  $x_1 \dots x_n \succeq y_1 \dots y_n$  if and only if  $\sum_{i=1}^j [x_i \geq k] \geq \sum_{i=1}^j [y_i \geq k]$  for all  $j$  and  $k$ .

This relation is self-dual in the sense that  $x_1 \dots x_n \succeq y_1 \dots y_n$  if and only if  $x_n \dots x_1 \preceq y_n \dots y_1$ . And if the digits consist of equal quantities of the numbers 1 through  $k$ , then  $x_1 \dots x_n \succeq y_1 \dots y_n$  if and only if  $\bar{x}_1 \dots \bar{x}_n \preceq \bar{y}_1 \dots \bar{y}_n$ , where  $\bar{x} = k + 1 - x$ .

It’s emphatically *not* a lattice, in most cases.

Here I just compute the relation and its transitive reduction by brute force. When I learn better algorithms for transitive reduction, I can use this as an interesting example.

(Well, maybe not! In the examples I tried, we seem to have  $x$  covers  $y$  if and only if  $x$  differs from  $y$  by a transposition and  $x$  has exactly one more inversion than  $y$ . Furthermore, it appears that the covering relation on multiset permutations such as  $\{1, 1, 2, 2, 3\}$  is obtained by taking the relation on set permutations  $\{1, 1', 2, 2', 3\}$  and removing all cases in which  $1'$  occurs before 1 or  $2'$  before 2. Thus, some additional theory apparently lurks in the background, making this whole program unnecessary — except as a way to confirm the conjectures in further cases before I go ahead and find proofs.)

```
#define maxn 63      /* this many elements at most */
#define maxp 1000    /* this many perms at most */
#include <stdio.h>
#include <string.h>
char perm[maxp][maxn + 1]; /* the permutations */
char work[maxn + 1];        /* where I generate new ones */
char rel[maxp][maxp];      /* nonzero if  $x \prec y$  */
char red[maxp][maxp];      /* reduced relation */
main(int argc, char *argv[])
{
    register int i, j, k, l, ll, m, n, s, dom;
    <Set work to the string that is to be permuted, and check it 2>;
    <Generate the rest of the permutations 3>;
    <Compute the dominance relation 4>;
    <Do transitive reduction 5>;
    <Print the results 6>;
}
```

**2.**  $\langle$  Set *work* to the string that is to be permuted, and check it **2**  $\rangle \equiv$

```

if (argc  $\neq$  2) {
    fprintf(stderr, "Usage: %s digits_to_permute\n", argv[0]);
    exit(-1);
}
for (j = 0; argv[1][j]; j++) {
    if (j > maxn) {
        fprintf(stderr, "String too long (maxn=%d)!\n", maxn);
        exit(-2);
    }
    if (argv[1][j] < '0'  $\vee$  argv[1][j] > '9') {
        fprintf(stderr, "The string %s should contain digits only!\n", argv[1]);
        exit(-3);
    }
    if (j > 0  $\wedge$  argv[1][j - 1] > argv[1][j]) {
        fprintf(stderr, "The string %s should be nondecreasing!\n", argv[1]);
        exit(-4);
    }
    work[j + 1] = argv[1][j];
}
n = j;

```

This code is used in section 1.

**3.** Here I use ye olde Algorithm 7.2.1.2L.

$\langle$  Generate the rest of the permutations **3**  $\rangle \equiv$

```

m = 0;
l1: if (m  $\equiv$  maxp) {
    fprintf(stderr, "Too many permutations (maxp=%d)!\n", maxp);
    exit(-5);
}
for (j = 0; j < n; j++) perm[m][j] = work[j + 1];
m++;
l2: for (j = n - 1; work[j]  $\geq$  work[j + 1]; j-- ) ;
    if (j  $\equiv$  0) goto done;
l3: for (l = n; work[j]  $\geq$  work[l]; l-- ) ;
    s = work[j], work[j] = work[l], work[l] = s;
l4: for (k = j + 1, l = n; k < l; k++, l-- ) s = work[k], work[k] = work[l], work[l] = s;
goto l1;
done:

```

This code is used in section 1.

4. We use the fact that dominance is a subset of (reverse) lexicographic order. In other words, if  $x_1 \dots x_n$  is lexicographically less than  $y_1 \dots y_n$  we cannot have  $x_1 \dots x_n \succeq y_1 \dots y_n$ .

⟨ Compute the dominance relation 4 ⟩  $\equiv$

```

for ( $l = 0$ ;  $l < m$ ;  $l++$ )
  for ( $ll = l + 1$ ;  $ll < m$ ;  $ll++$ ) {
     $dom = 0$ ;
    for ( $k = work[n] + 1$ ;  $k \leq work[1]$ ;  $k++$ )
      for ( $j = 0$ ;  $j < n$ ;  $j++$ ) {
        for ( $i = s = 0$ ;  $i \leq j$ ;  $i++$ )  $s += (perm[l][i] \geq k ? 1 : 0) - (perm[ll][i] \geq k ? 1 : 0)$ ;
        if ( $s > 0$ ) goto fin;
        if ( $s < 0$ )  $dom = 1$ ;
      }
    if ( $dom$ )  $rel[l][ll] = 1$ ;
  fin: continue;
}

```

This code is used in section 1.

5. Hey, I'm just using brute force today.

⟨ Do transitive reduction 5 ⟩  $\equiv$

```

for ( $l = 0$ ;  $l < m$ ;  $l++$ )
  for ( $ll = l + 1$ ;  $ll < m$ ;  $ll++$ ) {
    if ( $rel[l][ll]$ ) {
      for ( $j = l + 1$ ;  $j < ll$ ;  $j++$ )
        if ( $rel[l][j] \wedge rel[j][ll]$ ) goto nope;
       $red[l][ll] = 1$ ;
    }
  nope: continue;
}

```

This code is used in section 1.

6. ⟨ Print the results 6 ⟩  $\equiv$

```

for ( $l = 0$ ;  $l < m$ ;  $l++$ ) {
   $printf("%s\sqcup", perm[l])$ ;
  for ( $ll = l + 1$ ;  $ll < m$ ;  $ll++$ )
    if ( $red[l][ll]$ )  $printf("\sqcup%s", perm[ll])$ ;
   $printf("\n")$ ;
}

```

This code is used in section 1.

**7. Index.**

*argc*: [1](#), [2](#).  
*argv*: [1](#), [2](#).  
*dom*: [1](#), [4](#).  
*done*: [3](#).  
*exit*: [2](#), [3](#).  
*fin*: [4](#).  
*fprintf*: [2](#), [3](#).  
*i*: [1](#).  
*j*: [1](#).  
*k*: [1](#).  
*l*: [1](#).  
*ll*: [1](#), [4](#), [5](#), [6](#).  
*l1*: [3](#).  
*l2*: [3](#).  
*l3*: [3](#).  
*l4*: [3](#).  
*m*: [1](#).  
*main*: [1](#).  
*maxn*: [1](#), [2](#).  
*maxp*: [1](#), [3](#).  
*n*: [1](#).  
*nope*: [5](#).  
*perm*: [1](#), [3](#), [4](#), [6](#).  
*printf*: [6](#).  
*red*: [1](#), [5](#), [6](#).  
*rel*: [1](#), [4](#), [5](#).  
*s*: [1](#).  
*stderr*: [2](#), [3](#).  
*work*: [1](#), [2](#), [3](#), [4](#).

- ⟨ Compute the dominance relation [4](#) ⟩ Used in section [1](#).
- ⟨ Do transitive reduction [5](#) ⟩ Used in section [1](#).
- ⟨ Generate the rest of the permutations [3](#) ⟩ Used in section [1](#).
- ⟨ Print the results [6](#) ⟩ Used in section [1](#).
- ⟨ Set *work* to the string that is to be permuted, and check it [2](#) ⟩ Used in section [1](#).

DOMINATION

	Section	Page
Intro .....	<a href="#">1</a>	1
Index .....	<a href="#">7</a>	4