Generating Permutations and Subsets

The most important types of combinatorial objects are permutations, combinations, and subsets of a given set. There are n! permutations of {1, 2, . . . , n}. It is important in many instances to generate a list of such permutations.

Generating Permutations

We are given a sequence of numbers from 1 to n. Each permutation in the sequence that we need to generate should differ from the previous permutation by swapping just two adjacent elements of the sequence.

Input : n = 3

Output : 123 132 312 321 231 213

Input : n = 4

Output : 1234 1243 1423 4123 4132

1432 1342 1324 3124 3142 3412 4312

4321 3421 3241 3214 2314 2341 2431

4231 4213 2413 2143 2134

The Johnson and Trotter algorithm doesn't require to store all permutations of size n-1 and doesn't require going through all shorter permutations. Instead, it keeps track of the direction of each element of the permutation.

1. Find out the largest mobile integer in a particular sequence. **A directed integer is said to be mobile if it is greater than its immediate neighbor in the direction it is looking at**.
2. Switch this mobile integer and the adjacent integer to which its direction points.
3. Switch the direction of all the elements whose value is greater than the mobile integer value.
4. Repeat the step 1 until unless there is no mobile integer left in the sequence.

ALGORITHM *JohnsonTrotter*(n)

//Implements Johnson-Trotter algorithm for generating permutations
//Input: A positive integer n
//Output: A list of all permutations of $\{1, \ldots, n\}$
initialize the first permutation with $\overleftarrow{1}\ \overleftarrow{2} \ldots \overleftarrow{n}$
while the last permutation has a mobile element **do**
    find its largest mobile element k
    swap k with the adjacent element k's arrow points to
    reverse the direction of all the elements that are larger than k
    add the new permutation to the list

Here is an application of this algorithm for $n = 3$, with the largest mobile element shown in bold:

$$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{\mathbf{3}} \quad \overleftarrow{1}\ \overleftarrow{\mathbf{3}}\ \overrightarrow{2} \quad \overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{\mathbf{2}} \quad \overrightarrow{\mathbf{3}}\ \overrightarrow{2}\ \overleftarrow{1} \quad \overrightarrow{2}\ \overrightarrow{\mathbf{3}}\ \overleftarrow{1} \quad \overrightarrow{2}\ \overrightarrow{1}\ \overrightarrow{3}.$$

ALGORITHM *LexicographicPermute*(n)

//Generates permutations in lexicographic order
//Input: A positive integer n
//Output: A list of all permutations of $\{1, \ldots, n\}$ in lexicographic order
initialize the first permutation with $12 \ldots n$
while last permutation has two consecutive elements in increasing order **do**
  let i be its largest index such that $a_i < a_{i+1}$ //$a_{i+1} > a_{i+2} > \cdots > a_n$
  find the largest index j such that $a_i < a_j$ //$j \geq i + 1$ since $a_i < a_{i+1}$
  swap a_i with a_j //$a_{i+1}a_{i+2} \ldots a_n$ will remain in decreasing order
  reverse the order of the elements from a_{i+1} to a_n inclusive
  add the new permutation to the list

Generating Subsets

knapsack problem, which asks to find the most valuable subset of items that fits a knapsack of a given capacity. The exhaustive-search approach to solving this problem discussed there was based on generating all subsets of a given set of items. In this section, we discuss algorithms
for generating all 2^n subsets of an abstract set A

n	subsets							
0	$\varnothing$							
1	$\varnothing$	$\{a_1\}$						
2	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$				
3	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$

FIGURE 4.10 Generating subsets bottom up.

A more challenging question is whether there exists a minimal-change algorithm for generating bit strings so that every one of them differs from its immediate predecessor by only a single bit. (In the language of subsets, we want every subset to differ from its immediate predecessor by either an addition or a deletion, but not both, of a single element.) The answer to this question is yes. For example, for $n = 3$, we can get

$$000 \quad 001 \quad 011 \quad 010 \quad 110 \quad 111 \quad 101 \quad 100.$$

Such a sequence of bit strings is called Binary reflected Gray code.

ALGORITHM *BRGC*(n)

//Generates recursively the binary reflected Gray code of order n
//Input: A positive integer n
//Output: A list of all bit strings of length n composing the Gray code
if $n = 1$ make list L containing bit strings 0 and 1 in this order
else generate list $L1$ of bit strings of size $n - 1$ by calling $BRGC(n - 1)$
    copy list $L1$ to list $L2$ in reversed order
    add 0 in front of each bit string in list $L1$
    add 1 in front of each bit string in list $L2$
    append $L2$ to $L1$ to get list L
return L