Chapter 4 Practice for Combinatorics

Yonghui Wu

School of Computer Science, Fudan University

yhwu@fudan.edu.cn

Wechat: 13817360465

Chapter 4 Practice for Combinatorics

- Combinatorics
  - the branch of mathematics studying the enumeration, combination, and permutation of sets of elements and the mathematical relations that characterize their properties.

Chapter 4 Practice for Combinatorics

4.1 Generating Permutations

- 4.1.1 Generating the Next Permutation Based on Lexicographic Order
- 4.1.2 Generating All Permutations Based on Lexicographic Order

4.1.1 Generating the Next Permutation Based on Lexicographic Order

- Lexicographic Order is generating the next permutation based on the alphabetical order of their component elements.
- Suppose the current permutation is $(p)=p_1\ldots p_{i-1}p_i\ldots p_n$. The method generating the next permutation (q) based on Lexicographic Order is as follows.

Step 1: Find the longest suffix that is non-increasing by scanning the sequence from right to left. The element immediately to the left of the suffix is called "the first element". If there is no such an element, the sequence is non-increasing and is the last permutation. That is, find such an index i that $i = \max\{j \mid p_{j-1} < p_j, p_j \geq p_{j+1}\}$, and p_{i-1} is the first element.

Step 2: Find the rightmost successor to the first element in the suffix. Because the first element is less than the head of the suffix, some elements in the suffix are greater than the first element. In the suffix, the rightmost successor to the first element is the smallest element greater than the first element. We call the element "the second element". That is, find such an index j that $j = \max\{k \mid k \geq i, p_{i-1} < p_k\}$.

Step 3: Swap p_{i-1} and p_j, and get a new sequence $p_1 \ldots p_{i-2} \boxed{p_j} p_i p_{i+1} \ldots p_{j-1} \boxed{p_{i-1}} p_{j+1} \ldots p_n$.

Step 4: Reverse the subsequence after the original index of the first element. The next permutation is $(q) = p_1 \cdots p_{i-2}\ p_j\ \boxed{p_n \ldots p_{j+1} p_{i-1} p_{j-1} \cdots.\ p_{i+1}\ p_i.}$

- Suppose the current permutation is (p)=2763541. Based on Lexicographic Order, the next permutation (q)=2764135.
- (1) $276\underline{35}41$: Find the first element, and $p_{i-1}p_i$ is 35.
- (2) $27635\underline{4}1$: Find the second element: 4.
- (3) $276\underline{4}5\underline{3}1$: Swap the first element and the second element.
- (4) $2764\underline{135}$: Reverse the subsequence after the original index of the first element. And get the next permutation (q).

4.1.1.1 ID Codes

- Source: New Zealand Contest 1991
- IDs for online judges: POJ 1146, UVA 146

- It is 2084 and the year of Big Brother has finally arrived, albeit a century late. In order to exercise greater control over its citizens and thereby to counter a chronic breakdown in law and order, the Government decides on a radical measure--all citizens are to have a tiny microcomputer surgically implanted in their left wrists. This computer will contains all sorts of personal information as well as a transmitter which will allow people's movements to be logged and monitored by a central computer. (A desirable side effect of this process is that it will shorten the dole queue for plastic surgeons.)

- An essential component of each computer will be a unique identification code, consisting of up to 50 characters drawn from the 26 lower case letters. The set of characters for any given code is chosen somewhat haphazardly. The complicated way in which the code is imprinted into the chip makes it much easier for the manufacturer to produce codes which are rearrangements of other codes than to produce new codes with a different selection of letters. Thus, once a set of letters has been chosen all possible codes derivable from it are used before changing the set.

- For example, suppose it is decided that a code will contain exactly 3 occurrences of `a', 2 of `b' and 1 of `c', then three of the allowable 60 codes under these conditions are:
- abaabc
- abaacb
- ababac
- These three codes are listed from top to bottom in alphabetic order. Among all codes generated with this set of characters, these codes appear consecutively in this order.

- Write a program to assist in the issuing of these identification codes. Your program will accept a sequence of no more than 50 lower case letters (which may contain repeated characters) and print the successor code if one exists or the message `No Successor' if the given code is the last in the sequence for that set of characters.

- **Input**
- Input will consist of a series of lines each containing a string representing a code. The entire file will be terminated by a line consisting of a single #.
- **Output**
- Output will consist of one line for each code read containing the successor code or the words `No Successor'.

Analysis

- The successor code is the next permutation based on Lexicographic Order. Therefore, the algorithm is as follows. Suppose the given code is $s_0s_1s_2\ldots\ldots s_{l-1}$.
    - Find the index i that $i=\max\{j \mid s_{j-1}<s_j\}$.
    - If $i==0$, then the given code is the last in the sequence for that set of characters, output “No Successor”, and exit; else
    - On the right of “the first element”, find the smallest character greater than it. That is, find such an index j that $j=\max\{k \mid s_{i-1}<s_k\}$;
    - Swap s_{i-1} and s_j, and get $s_0\ldots s_{i-2}\, s_j\, s_i s_{i+1}\ldots s_{j-1}\, s_{i-1}\, s_{j+1}\ldots s_{l-1}$;
    - Reverse the substring after s_j, and get the successor code $(q)=s_0\ldots s_{i-2}s_js_{l-1}\ldots s_{j+1}s_{i-1}s_{j-1}\ldots s_{i+1}\,s_i$.

4.1.2 Generating All Permutations Based on Lexicographic Order

- Based on 4.1.1, the method generating all permutations for a finite set with *n* elements is as follows.
  - First, sort the *n* elements in ascending order, the permutation is the first permutation.
  - Then the method generating the next permutation based on Lexicographic order is used repeatedly until the last permutation is generated.

4.1.2.1 Generating Fast, Sorted Permutation

- Source: TCL Programming Contest， 2001
- ID for Online Judge: UVA 10098

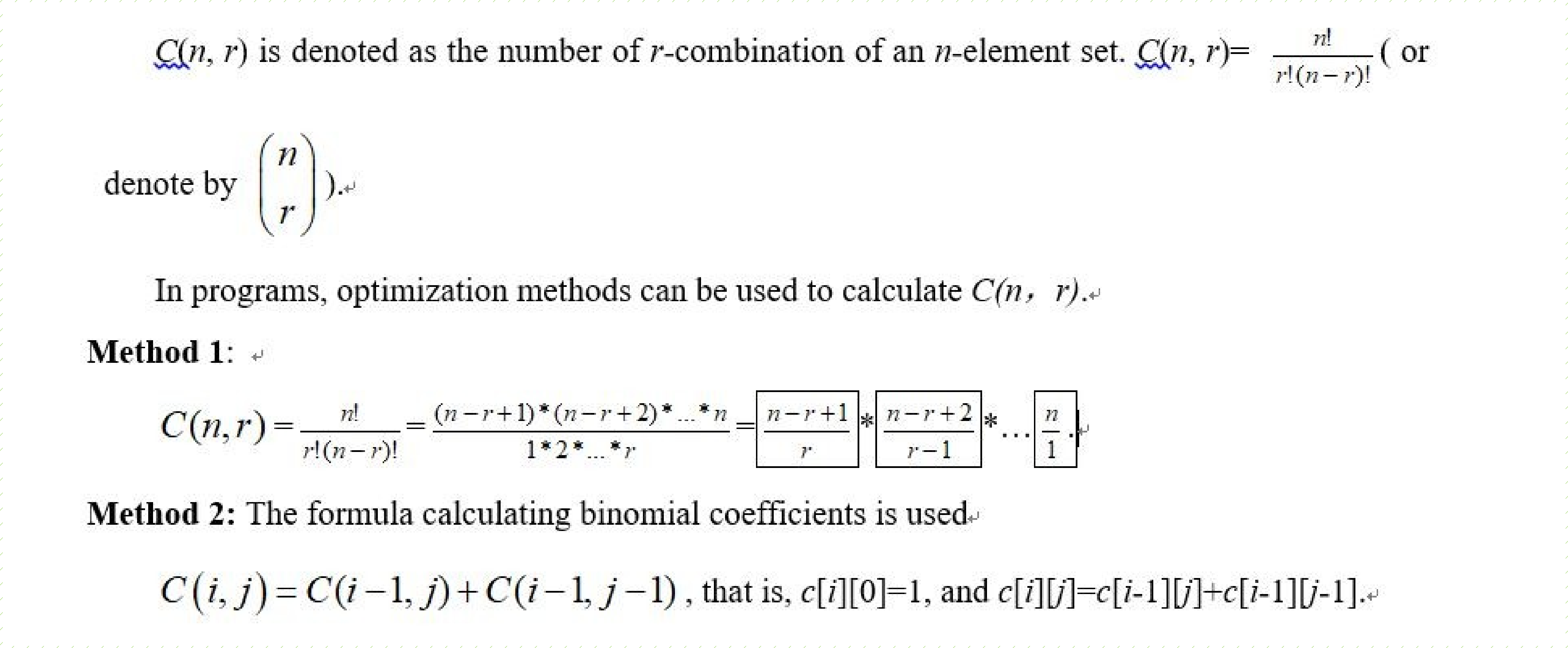
- Generating permutation has always been an important problem in computer science. In this problem you will have to generate the permutation of a given string in ascending order. Remember that your algorithm must be efficient.

- **Input**
- The first line of the input contains an integer *n*, which indicates how many strings to follow. The next *n* lines contain *n* strings. Strings will only contain alpha numerals and never contain any space. The maximum length of the string is 10.
- **Output**
- For each input string print all the permutations possible in ascending order. Not that the strings should be treated, as case sensitive strings and no permutation should be repeated. A blank line should follow each output set.

Analysis

- Suppose the leng of string s is l. Therefore, $l!$ permutations in ascending order are required to output. The algorithm is as follows.
- The first permutation is gotten by sorting string s in ascending order. For the current permutation s,
- (1) $i=\max\{\, j \mid s_{j-1} \geq s_j \,\}$;
- (2) $j=\max\{\, k \mid s_{i-1} < s_k\}$;
- (3) Swap s_{i-1} and s_j, and get $s_0\ldots s_{i-2}\underline{s_j}s_is_{i+1}\ldots s_{j-1}\underline{s_{i-1}}s_{j+1}\ldots s_{l-1}$;
- (4) Reverse the substring after s_j, and get the next permutation $(q)=s_0\ldots s_{i-2}s_j\underline{s_{l-1}\ldots s_{j+1}s_{i-1}s_{j-1}\ldots s_{i+1}s_i}$;
- The above process is repeated until $i==0$. All the permutations possible in ascending order are generated.

4.2 Enumeration of Permutations and Combinations



$C(n, r)$ is denoted as the number of r-combination of an n-element set. $C(n, r)= \frac{n!}{r!(n-r)!}$ (or denote by $\binom{n}{r}$).

In programs, optimization methods can be used to calculate $C(n，r)$.

Method 1:

$$C(n,r)=\frac{n!}{r!(n-r)!}=\frac{(n-r+1)*(n-r+2)*\ldots*n}{1*2*\ldots*r}=\boxed{\frac{n-r+1}{r}}*\boxed{\frac{n-r+2}{r-1}}*\ldots\boxed{\frac{n}{1}}.$$

Method 2: The formula calculating binomial coefficients is used

$C(i,j)=C(i-1,j)+C(i-1,j-1)$, that is, $c[i][0]=1$, and $c[i][j]=c[i-1][j]+c[i-1][j-1]$.

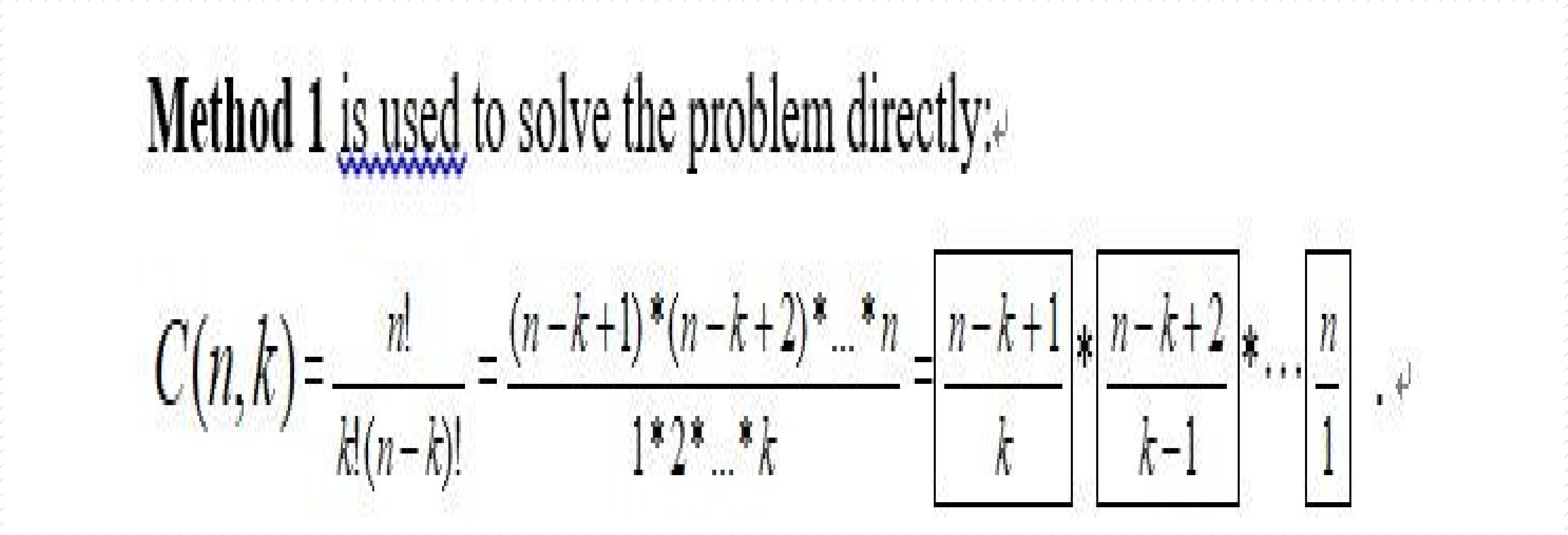
4.2.1.1 Binomial Showdown

- **Source: Ulm Local 1997**
- **IDs for Online Judges: POJ 2249, ZOJ 1938, UVA 530**

- In how many ways can you choose k elements out of n elements, not taking order into account? Write a program to compute this number.

- **Input**
- The input file will contain one or more test cases. Each test case consists of one line containing two integers *n* (*n*>=1) and *k* (0<=*k*<=*n*). Input is terminated by two zeroes for *n* and *k*.
- **Output**
- For each test case, print one line containing the required number. This number will always fit into an integer, i.e. it will be less than 2^{31}.
- **Warning:** Don't underestimate the problem. The result will fit into an integer - but if all intermediate results arising during the computation will also fit into an integer depends on your algorithm. The test cases will go to the limit.

Analysis



Method 1 is used to solve the problem directly:

$$C(n,k)=\frac{n!}{k!(n-k)!}=\frac{(n-k+1)*(n-k+2)*...*n}{1*2*...*k}=\boxed{\frac{n-k+1}{k}}*\boxed{\frac{n-k+2}{k-1}}*\ldots\boxed{\frac{n}{1}}\ .$$

4.2.1.2 Combinations

- IDs for Online Judges: POJ 1306， UVA 369

- Computing the exact number of ways that *N* things can be taken *M* at a time can be a great challenge when *N* and/or *M* become very large. Challenges are the stuff of contests. Therefore, you are to make just such a computation given the following:
- Given: $5 \le N \le 100$; $5 \le M \le 100$; $M \le N$
- Compute the **EXACT** value of: $C = N! / (N-M)!M!$
- You may assume that the final value of *C* will fit in a 32-bit Pascal LongInt or a C long. For the record, the exact value of 100! is:
- 93,326,215,443,944,152,681,699,238,856,266,700,490,715,968,264,381,621, 468,592,963,895,217,599,993,229,915,608,941,463,976,156,518,286,253, 697,920,827,223,758,251,185,210,916,864,000,000,000,000,000,000,000,000

- **Input**
- The input to this program will be one or more lines each containing zero or more leading spaces, a value for *N*, one or more spaces, and a value for *M*. The last line of the input file will contain a dummy *N*, *M* pair with both values equal to zero. Your program should terminate when this line is read.
- **Output**
- The output from this program should be in the form:
- *N* things taken *M* at a time is *C* exactly.

Analysis

- Suppose $c[i][j]$ is $C(i, j)$.
- Based on the formula calculating binomial coefficients, $c[i][j]=c[i-1][j]+c[i-1][j-1]$.
- Initially, $c[i][0]=1$, $0\leq i\leq 101$.
- Then, based on the above formula, $c[i][j]$ can be calculated, $1\leq i\leq 100$ ， $1\leq j\leq 100$.
- Finally, for each test case N, M, Output c[N][M].