CS 6301 : {016}

Implementation of Advanced DS&A

Project 2 (Type 1)

Enumeration of permutations and combination.

Submitted By : Arjun Gopal
NET ID : AXG145630
09/12/2014
CS 6301 : Section 016

Project Requirement

1. Write a program that visits all permutations/combinations of k objects from a set of n distinct objects, numbered 1..n. The program takes input from the console/terminal/stdin and prints output to stdout.

   Input specification:
   Each line of input has 3 integers: n, k, and v.
   n is between 3 and 1000; k is an integer between 0 and n; v is in {0,1,2,3}.

   Output specification:
   if v = 0, the program outputs one line with 2 integers: the number of permutations of k objects out of n distinct objects, and the time taken by the program in milliseconds (rounded to integer). Even though the program is not printing the permutations, it should visit all permutations.

   if v = 1, the program again outputs one line with 2 integers, like the above case, except it outputs the number of combinations of k out of n.

   if v = 2 (verbose output version of v=0), the program should output all the actual permutations, one line at a time, and then the output of v=0.

   if v = 3 (verbose output of v=1), output all the actual combinations, and then the output of v=1.

   Sample input:
   ```
   4 2 2
   ```

   Sample output:
   ```
   1 2
   1 3
   1 4
   2 1
   2 3
   2 4
   3 1
   3 2
   3 4
   4 1
   4 2
   4 3
   12 0
   ```

2. Write a program that visits all permutations of n (not necessarily distinct) objects. The program reads from its standard input (stdin) and writes the output to stdout.

   Input specification:
   First line of input has 2 integers, n and v.
   The value of n is between 0 and 1000; v is either 0 or 1.
   The next n integers in the input (separated by spaces and/or newlines) are the n elements.

```
Output specification:
If v > 0, then print each distinct permutation in one line of output.
If v = 0, only one line of output is generated (below).  But your
program should visit all the permutations, though it does not output them.
The last line of output has the number of permutations visited and the
time taken by the program in milliseconds (rounded to an integer).

Sample input:
4 1
1 2 3 2

Sample output:
1 2 2 3
1 2 3 2
1 3 2 2
2 1 2 3
2 1 3 2
2 2 1 3
2 2 3 1
2 3 1 2
2 3 2 1
3 1 2 2
3 2 1 2
3 2 2 1
12 0
```

Introduction

<u>Problem 1</u>

The problem 1 is to visit permutations and combinations of r items from a set of n items (nPr and nCr). Permutation problem was addressed using Algorithm L which was described in Donald Knuth's "The Art of Computer Programming". There were some alterations made to the algorithm as the base algorithm is to calculate all the permutations of n items and we need to permute r items from n. Details are mentioned detailed later in this document. The second problem (combinations) is addressed using a recursive approach. Also readings (time) are given for the various test cases.

<u>Problem 2</u>

The problem 2 is to calculate the distinct permutations of an array with duplicates items. This problem is also addressed with Algorithm L explained by Donald Knuth. Test cases with tables showing the time taken for each is given.

The implementations are done using Java. The code can be compiled as follows

Javac AXG145630Problem1.java

Javac AXG145630Problem2.java

The input will be read from the console according to the specifications mentioned as a part of project requirement.

nP0 and nC0 will return 1 as the count as there is only 1 way of selecting 0 items from n items which is nothing. But it has no permutation or combination sequence to print and so it will not print any sequence for verbose 1 and 3.

Implementation Details

Problem 1

Permutations : Algorithm L was implemented with little alterations so that it can permute r items from n items. The modified algorithm is mentioned below.

Input is array a[1…n] . Compute nPr (Permutation of r items from n)

```
L1. Visit permutation a1 a2 .. ar
L2. Rotate the array from r+1
L3. Set j as n-1. If a[j] >= a[j+1] decrease j repeatedly by 1 until a[j]<a[j+1]. Terminate the algorithm if j<0
L3. Set rightIndex = j+1 and m = j+2 . increment m till m>=n. update rightIndex to m if (a[m]>a[j] and a[m]<a[rightIndex])
L4. Exchange items at index j and rightIndex
L5. If(r-1) > j rotate array from j and then from r+1.  Return to L1
```

Combinations : This problem is addressed with a recursive approach. We have to compute combinations of r items from n (nCr)

The recursive method takes the following parameters

arr[] : the main array of size n

aux[] : the auxiliary array of size r

begin : initialized to 0

n : the total no of items

r : initialized to 0

k : no of items that has to be selected from n

v : verbose input

in each recursive call , there is a loop from begin to n and auxiliary array is populated with items from main array . when the last slot for auxiliary array is reached, all the remaining items (which are not visited yet) is added there and printed in iterations.

Algorithm

S1 : initialize begin = 0, r=0

S2: if(k<0) return. Base case

S3 : if r== k (we are at the last slot). Loop from begin to n and put all the values at aux[r] and print aux.

S3: if (r<k), then there are more slots in aux to fill. Loop I from begin to m (m reduces by one in each recursion). in the first recursion m will be equal to n. In each loop put aux[r] = arr[i] and recursively call the method by incrementing begin and r.

This algorithm is very efficient in performance as it will not go and visit all the combinations of n item. No of recursive calls is proportional to k.

Problem 2 is a direct implementation of Algorithm L. Here the input is an array which contains duplicate. The array is sorted (double pivot quick sort) ,used by java for sorting arrays of primitive data types, before passing to algorithm L.

Execution Statistics.

Problem 1

Input	**Output**	**Time taken (in millisecond)**
10 10 0	3628800 47	47
11 11 0	39916800 328	328
12 12 0	479001600 3580	3580
13 13 0	6227020800 45612	45612
5 3 0	60 0	0
50 3 0	117600 17	17
5 2 1	10 0	0
50 2 1	1225 0	0
50 4 1	230300 1	1
90 8 1	77515521435 83008	83008

Table 1 : Execution Time readings for problem 1

Problem 2

Input	Output	Time Taken (in milliseconds)
7 0 1 2 2 3 4 3 2	420 1	1
9 0 11 21 12 21 4 21 11 4 11	5040 1	1

Table 2 : Execution Time readings for problem 2

Conclusion with comments

Both the problems were implemented considering all the edge cases. The performance of both is adaptive. For example for 4P2 it will visit only 12 permutations and not all the 24. The project took 2 days to implement including preparing the report and executing the test cases.

References :

The Art of Computer Programming – Donald E. Knuth