Marcus Mann

Theodore Walsh

28 January 2022

APMA 3100-004

Project 1: An Exploration of Counting Methods

On my honor as a student, I have neither given nor received aid on this assignment.

Theodore Walsh

//sign name here

1.

a) (astronomy, calculus, French, history) , (astronomy, calculus, French, literature) , (astronomy, calculus, French, organic chemistry) , (astronomy, calculus, French, physics) , (astronomy, calculus, history, literature) , (astronomy, calculus, history, organic chemistry) , (astronomy, calculus, history, physics) , (astronomy, calculus, literature, organic chemistry) , (astronomy, calculus, literature, physics) , … (history, literature, organic chemistry, physics)

b) (astronomy, calculus, French, history), (astronomy, calculus, history, French), (astronomy, history, calculus, French), (astronomy, history, French, calculus), (astronomy, French, history, calculus), (astronomy, French, calculus, history),

(calculus, astronomy, French, history), (calculus, astronomy, history, French), …

(French, astronomy, history, calculus),…

(history, French, astronomy, calculus),…

(astronomy, literature, organic chemistry, physics),…

c) The provided code counts the book orders as the iterator checks between for loops eliminates all duplicate combinations.

d)

import java.util.ArrayList;  
import java.util.List;

List<String> classes = new ArrayList<>();  
classes.add("astronomy");  
classes.add("calculus");  
classes.add("French");  
classes.add("history");  
classes.add("literature");  
classes.add("organic chemistry");  
classes.add("physics");  
  
int orders = 0;  
for (int i=1; i<=classes.size(); i++) {  
 for (int j=1; j<=classes.size(); j++) {  
 if (i == j) {  
 break;  
 }  
 for (int k=1; k<=classes.size(); k++) {  
 if ((i == k) || (j == k)) {  
 break;  
 }  
 for (int l=1; l<=classes.size(); l++) {  
 if ((l == i) || (l == j) || (l==k)) {  
 break;  
 }  
 orders++;  
 if (orders <= 20) {  
 System.*out*.println(classes.get(l - 1) + ", " + classes.get(k - 1) + ", " + classes.get(j - 1) + ", " + classes.get(i - 1));  
 }  
 }  
 }  
 }  
}  
  
System.*out*.println("Total orders: " + orders);

astronomy, calculus, French, history

astronomy, calculus, French, literature

astronomy, calculus, history, literature

astronomy, French, history, literature

calculus, French, history, literature

astronomy, calculus, French, organic chemistry

astronomy, calculus, history, organic chemistry

astronomy, French, history, organic chemistry

calculus, French, history, organic chemistry

astronomy, calculus, literature, organic chemistry

astronomy, French, literature, organic chemistry

calculus, French, literature, organic chemistry

astronomy, history, literature, organic chemistry

calculus, history, literature, organic chemistry

French, history, literature, organic chemistry

astronomy, calculus, French, physics

astronomy, calculus, history, physics

astronomy, French, history, physics

calculus, French, history, physics

astronomy, calculus, literature, physics

Total orders: 35

2.

If a requested password requires 8 digits of which 3 must be zeroes and the remaining 5 must be ones, we can represent the positions of the zeroes in the password array as solutions to the problem. For instance, the password string 00011111 can be represented as 123. Using Java, with 3 nested for- loops that skip over duplicate entries where one loop’s iterator equals another’s, that is, a zero cannot be in the same ith position multiple times, the solutions to the problem can be represented by the following code:

int counter = 0;  
System.*out*.print("{ ");  
for (int i=1; i<=8; i++) {  
 for (int j=1; j<=8; j++) {  
 if (j == i) {  
 break;  
 }  
 for (int k=1; k<=8; k++) {  
 if (k == i || k == j) {  
 break;  
 }  
 System.*out*.print("(" + k + "" + j + "" + i + ") ");  
 counter++;  
 if (counter %10 == 0) {  
 System.*out*.println();  
 }  
 }  
 }  
}  
System.*out*.println("}");  
System.*out*.println("Total passwords: " + counter);

This results in the 56 values that indicate the location of the zeroes in the password.

{ (123) (124) (134) (234) (125) (135) (235) (145) (245) (345)

(126) (136) (236) (146) (246) (346) (156) (256) (356) (456)

(127) (137) (237) (147) (247) (347) (157) (257) (357) (457)

(167) (267) (367) (467) (567) (128) (138) (238) (148) (248)

(348) (158) (258) (358) (458) (168) (268) (368) (468) (568)

(178) (278) (378) (478) (578) (678) }

Total passwords: 56

If this problem were to be expanded to a 30-digit password that required 13 0’s and 17 1’s, adapting the previous solution would be problematic as each of the 13 nested loops would need to be tested against each other to prevent 0’s ending in duplicate positions. This seems rather labor intensive and antithetical to the programming mantra. A recursive solution is probably possible though admittedly I could not discover how to make it work correctly. Programming the math directly was also problematic as it involves calculating the value of 30! which is many orders of magnitude larger than what can naturally be represented by a 64-bit integer, so Java’s BigInteger Class was used which cleverly turns the large value into an array of single-digit values.

import java. math.BigInteger;

int length = 30;  
int zeroes = 13;  
System.*out*.println("Total passwords: " + *passwords*(length, zeroes));

public static BigInteger passwords(int length, int zeroes) {  
  
 BigInteger numerator = BigInteger.*valueOf*(length);  
 for (int i=length-1; i > (length-zeroes); i-- ) {  
 numerator = numerator.multiply(BigInteger.*valueOf*(i));  
 }  
  
 BigInteger denominator = BigInteger.*valueOf*(zeroes);  
 for (int i=zeroes-1; i>0; i--) {  
 denominator = denominator.multiply(BigInteger.*valueOf*(i));  
 }

return numerator.divide(denominator);  
 }

Total passwords: 119759850

With the issue of the integer size solved, the values of the numerator and denominator were calculated using the formula for combinations, since each 0 is not unique, where The numerator is calculated first by only calculating the values of to reduce the number of calculations required and the overall size of the numerator. Next the denominator Is calculated, and then the two are divided using BigInteger’s appropriate method to get the solution of 119,759,850 passwords.