**Design:**

There are 11 students and 11 courses. We have to find all possible combinations of assigning course of his/her choice to students, such that every one gets at least 1 course of his choice.

We initially started with backtracking approach. Idea was to start from any one student and assign all the possible courses of his choice. Then call the method again with remaining courses and remaining students and with set of remaining courses.

If any branch of tree try to assign the already assigned course then stop and return. But the solution had student\*!course complexity.

So while trying to find a better solution, we read through the various **resources** on internet. Dp with bit masking seems to present a better solution to the problem.

So the idea here is to keep a 2-D array DP[2TotalStudents][TotalCourses], where DP[s][c] will store the answer only if the course from id 1 to c have been used. Also, let's say s when denoted in binary be b1,b2...bn. If bp(1 ≤ p ≤ n) is 1, it means that student with id == p has been assigned a course and all the students with bit 1 are assigned distinct courses.

As a standard DP problem, we will develop a recursive solution DP,.

Function findPossibleAssignments(mask, courseid) which means that courses till id courseid have been assigned and mask denotes which students have got the course. At each step we assign course to each possible student based on his/her choice and recursively calculate the number of possible assignments.

In order to represent courses as subset, we are using bitMask. For 11 students, all possible ways to assign & not assign student are 211.

In the subset i-th student has a course if and only if the i-th bit of the mask is set i.e., it equals to 1. As an example, the mask 1000010100 means that the subset of the set [1… 11] consists of elements 3, 5 and 11. We know that for a set of N students there are total 2N subsets thus 2N combinations are possible, one representing each subset.

So the dp[i][j] represent course j assignment to student i. In a way we look at possible assignments but use dp to re-use the computed results.

**Steps:**

1. read the input from file.
2. Initialize the variables. Assignments[][], courseToStudentMap, courseNameToIdMap.
3. Call the dp function which internally calls it recursively to count total combinations.

**Runtime complexity:**

Worst case complexity : O(TotalCourse \* 2TotalStudents)

**Resources referred :**

<https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/>

<https://www.geeksforgeeks.org/bitmasking-and-dynamic-programming-set-1-count-ways-to-assign-unique-cap-to-every-person/>

<https://medium.freecodecamp.org/unmasking-bitmasked-dynamic-programming-25669312b77b>

**Files attached:**

DP with Bit Masking: CourseAssignment.java

Backtracking approach: CourseDistributorDP.java