There are 11 students and 11 courses.

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 probem, 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.

Steps:

read the input from file.

Initialize the variables. Assignments[][], courseToStudentMap, courseNameToIdMap.

Call the dp function which internally calls it recursively to count total combinations.

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