Top-20 Training Program (BackTracking Thinking)

Apply the Backtracking thinking discussed in class to solve the following problems.

Problem1: Generate Permutations

Write an efficient program that displays all the n-length permutations using characters given in an n-length string.

Problem2: All Subsets Sum

Given a sorted array of n integers and sum S, write an efficient method that displays all combinations of given array elements that sum to S. Also, write a method that solves the above problem if the array is not sorted?

Problem3: Bar Color Alignment

You are given a set of N "color bars," each bar containing N colors (3 <= N <= 12). The colors on a color bar can be changed by rotating them one step to the right, end around. For example, Red-Yellow-Green-Blue can be rotated to Blue-Red-Yellow-Green. The input will be two dimensional NxN array in which each row contains numbers representing the colors on that color bar. Colors will be represented by the integers 0 through 11. Rotate the color bars until every column contains every color, and no column contains the same color more than once. For example, for N = 5, the input might be:

```
3 4 1 0 2
3 1 2 4 0
4 0 3 2 1
3 1 2 4 0
3 1 0 2 4
```

Your program must return true if such solution possible otherwise return false.

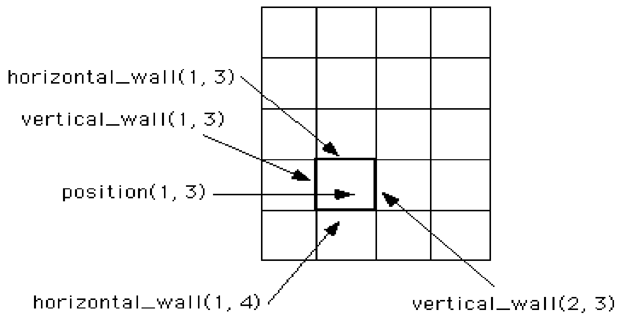
Problem4: Maze Solving

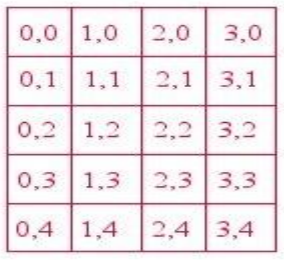
The maze is represented as an n x m grid (a 2-D array named maze_postion[n][m]). Each square in the grid can take on an integer value. The value of each square is initialized to zero except for the "end" of the maze, which is set to a value of 2. As the program moves through the maze, it should set the values of squares on the current path to 1 (this will allow the print-out to show the path when the program reaches the goal).

Walls of the maze are represented by two 2-D arrays: horizontal_wall[n][m+1], and vertical_wall[n+1][m]. The values of the elements of these arrays are false for those positions with no wall and true for positions where there is a wall. The positions of the walls relative to the grid are as follows:For a grid cell at position x, y,

- The horizontal wall directly above it is at horizontal_wall[x][y].
- The horizontal wall directly below it is at horizontal_wall[x][y+1].
- The vertical wall directly to the left is at vertical_wall[x][y].
- The vertical wall directly to the right is at vertical_wall[x+1][y].

Note that an increase in y indicates a move downward in the maze. The relationship of the walls and the gridlines is diagrammed below; the grid on the right shows the corresponding x and y coordinates for each grid position:





0,0	1,0	2,0	3,0
0,1	1,1	2,1	3,1
0,2	1,2	2,2	3,2
0,3	1,3	2,3	3,3
0,4	1,4	2,4	3,4

You must write a program to find a path from the start (at positions 0,0) to the end (marked/displayed with a $). As you move through the maze, you are not allowed to move through a wall. Thus, before you make any moves you must test whether there is a wall in the way. Your task is to write the function solveMaze() which will solve the maze and print out the solution if it finds one. It will return a value of true if it finds the end of the maze, and false if it does not find the end (a grid space with value of 2).

Example:

```
---------------
|     |        |
      ---   ---
|  |  |        |
      ------
|  |  |  |     |
         ---
|  |           |
         ---
|  |  |  |   $ |
---------------

---------------
|X  X |        |
      ---   ---
|  |X |        |
      ------
|  |X |  |     |
         ---
|  |X  X  X  X |
         ---
|  |  |  |   $ |
---------------
```

Problem5: Cryptarithmatic Puzzle Solving

Write an efficient program that solves the following cryptographic puzzle:

```
  S END
+MORE
--------------
MONEY
--------------
```

The goal here is to assign each letter a digit from 0 to 9 so that the arithmetic works out correctly. The rules are that all occurrences of a letter must be assigned the same digit, and no digit can be assigned to more than one letter. Your method will take three strings as arguments and returns the valid assignment for all the characters available in puzzle if solution exists.