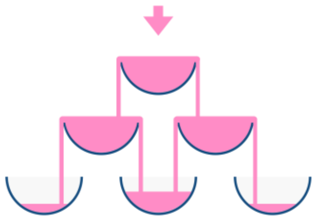
142.We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2 glasses, and so on until the 100th row. Each glass holds one cup of champagne. Then, some champagne is poured into the first glass at the top. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, any excess champagne will fall equally to the left and right of those glasses, and so on. (A glass at the bottom row has its excess champagne fall on the floor.) For example, after one cup of champagne is poured, the top most glass is full. After two cups of champagne are poured, the two glasses on the second row are half full. After three cups of champagne are poured, those two cups become full - there are 3 full glasses total now. After four cups of champagne are poured, the third row has the middle glass half full, and the two outside glasses are a quarter full, as pictured below.



Now after pouring some non-negative integer cups of champagne, return how full the jth glass in the ith row is (both i and j are 0-indexed.)

Example 1:

Input: poured = 1, query_row = 1, query_glass = 1
Output: 0.00000
Explanation: We poured 1 cup of champange to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

Example 2:

Input: poured = 2, query_row = 1, query_glass = 1
Output: 0.50000
Explanation: We poured 2 cups of champange to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champange.

AIM: To solve the above problem

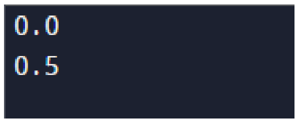
PROGRAM:

```
def champagneTower(poured, query_row, query_glass):
    dp = [[0.0] * (i + 1) for i in range(query_row + 1)]
    dp[0][0] = poured

    for i in range(query_row):
        for j in range(len(dp[i])):
            if dp[i][j] > 1:
                excess = dp[i][j] - 1
                dp[i + 1][j] += excess / 2.0
                dp[i + 1][j + 1] += excess / 2.0

    return min(1.0, dp[query_row][query_glass])

print(champagneTower(1, 1, 1))
print(champagneTower(2, 1, 1))
```

OUTPUT: 

```
0.0
0.5
```

TIME COMPLEXITY: O(query_row^2)