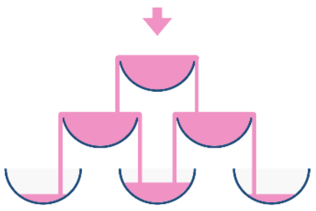
1.17 CHAMPAGNE TOWER – GLASS FILL SIMULATION

Question:

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.)

AIM:

Given a number of champagne cups poured into the top glass of a pyramid (100 rows), determine how full the glass at position (query_row, query_glass) is after pouring.

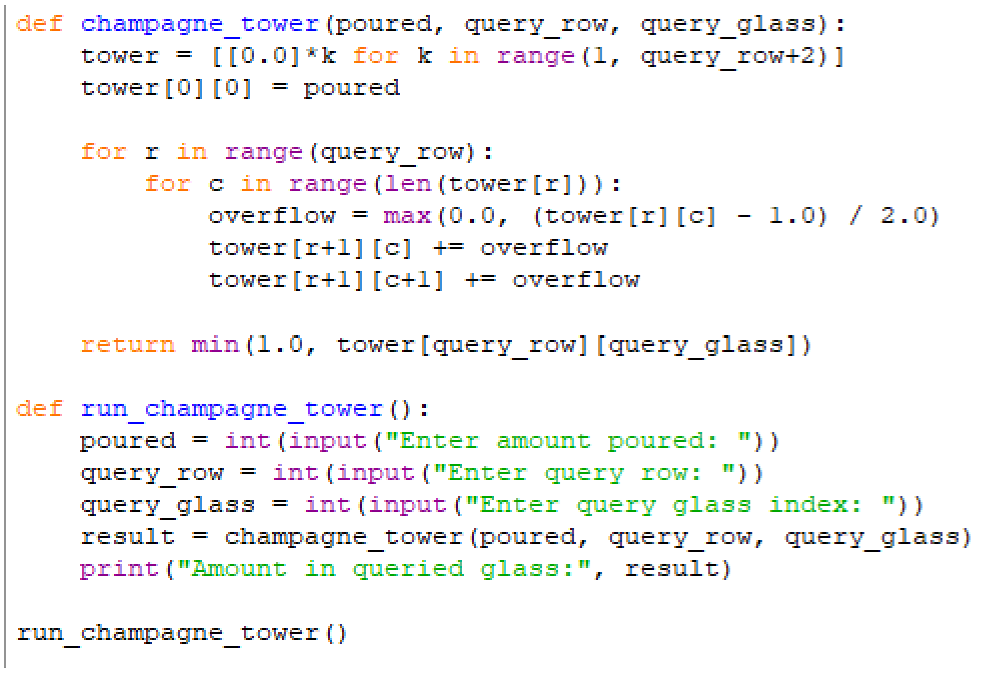
ALGORITHM:

1. Create a 2D array tower with size [query_row + 2][query_row + 2] (to avoid overflow indexing).
2. Pour poured cups into tower[0][0].
3. For each row r from 0 to query_row:
   - For each glass c in row r:
   - If tower[r][c] > 1, compute excess = tower[r][c] - 1.

- Distribute excess / 2 to tower[r+1][c] and tower[r+1][c+1].
- Clamp current tower[r][c] to 1.

4. Return the minimum of 1 and tower[query_row][query_glass].

PROGRAM:

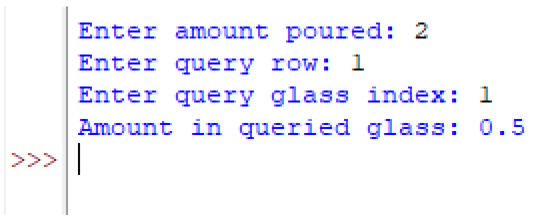


```
def champagne_tower(poured, query_row, query_glass):
    tower = [[0.0]*k for k in range(1, query_row+2)]
    tower[0][0] = poured

    for r in range(query_row):
        for c in range(len(tower[r])):
            overflow = max(0.0, (tower[r][c] - 1.0) / 2.0)
            tower[r+1][c] += overflow
            tower[r+1][c+1] += overflow

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

def run_champagne_tower():
    poured = int(input("Enter amount poured: "))
    query_row = int(input("Enter query row: "))
    query_glass = int(input("Enter query glass index: "))
    result = champagne_tower(poured, query_row, query_glass)
    print("Amount in queried glass:", result)

run_champagne_tower()
```

Input:

poured = 2, query_row = 1, query_glass = 1

Output:



```
    Enter amount poured: 2
    Enter query row: 1
    Enter query glass index: 1
    Amount in queried glass: 0.5
>>> |
```

RESULT:

Thus the program is successfully executed, and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: $O(n^2)$
- Space Complexity: $O(n^2)$