Problem 799: Champagne Tower

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

We stack glasses in a pyramid, where the

first

row has

1

glass, the

second

row has

2

glasses, and so on until the 100

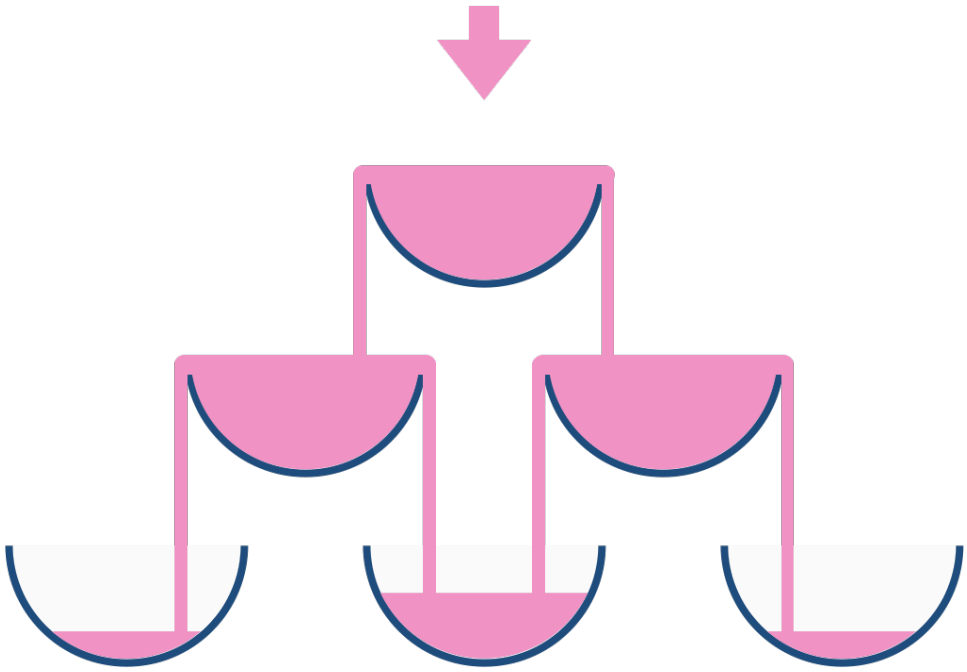
th

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

j

th

glass in the

i

th

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.

Example 3:

Input:

poured = 100000009, query_row = 33, query_glass = 17

Output:

1.00000

Constraints:

0 <= poured <= 10

9

0 <= query_glass <= query_row < 100

Code Snippets

C++:

```
class Solution {
public:
double champagneTower(int poured, int query_row, int query_glass) {

}
};
```

Java:

```
class Solution {
public double champagneTower(int poured, int query_row, int query_glass) {

}
}
```

Python3:

```
class Solution:
def champagneTower(self, poured: int, query_row: int, query_glass: int) ->
float:
```

Python:

```
class Solution(object):
def champagneTower(self, poured, query_row, query_glass):
"""
:type poured: int
:type query_row: int
:type query_glass: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number} poured
* @param {number} query_row
* @param {number} query_glass
* @return {number}
*/
var champagneTower = function(poured, query_row, query_glass) {

};
```

TypeScript:

```
function champagneTower(poured: number, query_row: number, query_glass:
number): number {

};
```

C#:

```
public class Solution {
public double ChampagneTower(int poured, int query_row, int query_glass) {

}
}
```

C:

```



double champagneTower(int poured, int query_row, int query_glass){
```

```
}
```

Go:

```
func champagneTower(poured int, query_row int, query_glass int) float64 {

}
```

Kotlin:

```
class Solution {
fun champagneTower(poured: Int, query_row: Int, query_glass: Int): Double {

}
}
```

Swift:

```
class Solution {
func champagneTower(_ poured: Int, _ query_row: Int, _ query_glass: Int) ->
Double {

}
}
```

Rust:

```
impl Solution {
pub fn champagne_tower(poured: i32, query_row: i32, query_glass: i32) -> f64
{

}
}
```

Ruby:

```
# @param {Integer} poured
# @param {Integer} query_row
# @param {Integer} query_glass
# @return {Float}
def champagne_tower(poured, query_row, query_glass)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $poured
* @param Integer $query_row
* @param Integer $query_glass
* @return Float
*/
function champagneTower($poured, $query_row, $query_glass) {

}
}
```

Scala:

```
object Solution {
def champagneTower(poured: Int, query_row: Int, query_glass: Int): Double = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
double champagneTower(int poured, int query_row, int query_glass) {

}
};
```

Java Solution:

```
/**
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public double champagneTower(int poured, int query_row, int query_glass) {

}
}
```

Python3 Solution:

```
"""
Problem: Champagne Tower
Difficulty: Medium
Tags: dp, stack

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def champagneTower(self, poured: int, query_row: int, query_glass: int) ->
float:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def champagneTower(self, poured, query_row, query_glass):
"""
:type poured: int
:type query_row: int
:type query_glass: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} poured
* @param {number} query_row
* @param {number} query_glass
* @return {number}
*/
var champagneTower = function(poured, query_row, query_glass) {

};
```

TypeScript Solution:

```
/**
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
```

```
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function champagneTower(poured: number, query_row: number, query_glass:
number): number {

};
```

C# Solution:

```
/*
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public double ChampagneTower(int poured, int query_row, int query_glass) {

}
}
```

C Solution:

```
/*
* Problem: Champagne Tower
* Difficulty: Medium
* Tags: dp, stack
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
double champagneTower(int poured, int query_row, int query_glass){

}
```

Go Solution:

```
// Problem: Champagne Tower
// Difficulty: Medium
// Tags: dp, stack
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func champagneTower(poured int, query_row int, query_glass int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun champagneTower(poured: Int, query_row: Int, query_glass: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func champagneTower(_ poured: Int, _ query_row: Int, _ query_glass: Int) ->
Double {

}
}
```

Rust Solution:

```
// Problem: Champagne Tower
// Difficulty: Medium
```

```
// Tags: dp, stack
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn champagne_tower(poured: i32, query_row: i32, query_glass: i32) -> f64
{

}
}
```

Ruby Solution:

```
# @param {Integer} poured
# @param {Integer} query_row
# @param {Integer} query_glass
# @return {Float}
def champagne_tower(poured, query_row, query_glass)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $poured
* @param Integer $query_row
* @param Integer $query_glass
* @return Float
*/
function champagneTower($poured, $query_row, $query_glass) {

}
}
```

Scala Solution:

```
object Solution {
def champagneTower(poured: Int, query_row: Int, query_glass: Int): Double = {

}
}
```