Problem 892: Surface Area of 3D Shapes

Problem Information

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

Problem Description

You are given an

n x n

grid

where you have placed some

1 x 1 x 1

cubes. Each value

v = grid[i][j]

represents a tower of

v

cubes placed on top of cell

(i, j)

.

After placing these cubes, you have decided to glue any directly adjacent cubes to each other, forming several irregular 3D shapes.

Return

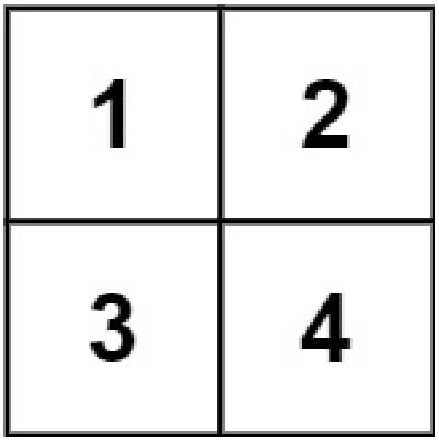
the total surface area of the resulting shapes

.

Note:

The bottom face of each shape counts toward its surface area.

Example 1:



1	2
3	4

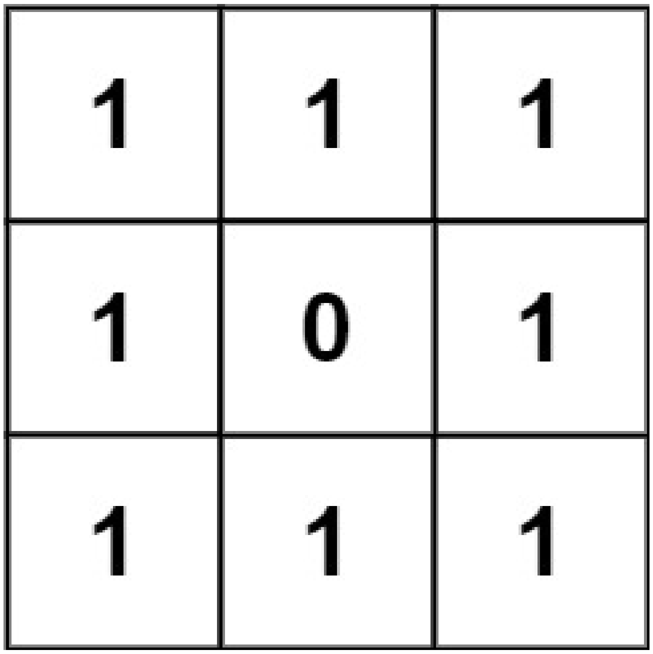
Input:

grid = [[1,2],[3,4]]

Output:

34

Example 2:



1	1	1
1	0	1
1	1	1

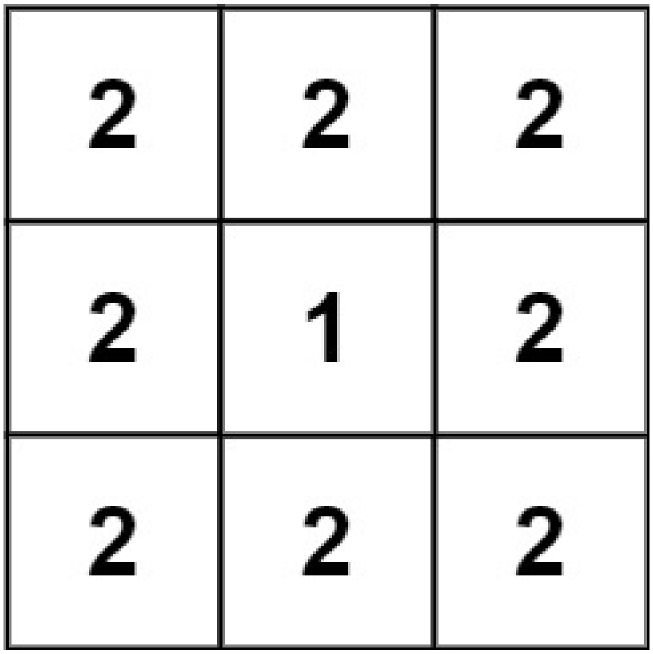
Input:

grid = [[1,1,1],[1,0,1],[1,1,1]]

Output:

32

Example 3:



2	2	2
2	1	2
2	2	2

Input:

grid = [[2,2,2],[2,1,2],[2,2,2]]

Output:

46

Constraints:

n == grid.length == grid[i].length

1 <= n <= 50

0 <= grid[i][j] <= 50

Code Snippets

C++:

```
class Solution {
public:
int surfaceArea(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int surfaceArea(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def surfaceArea(self, grid: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def surfaceArea(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
var surfaceArea = function(grid) {

};
```

TypeScript:

```
function surfaceArea(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int SurfaceArea(int[][] grid) {

}
}
```

C:

```
int surfaceArea(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func surfaceArea(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun surfaceArea(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func surfaceArea(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn surface_area(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def surface_area(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/

function surfaceArea($grid) {

}
}
```

Dart:

```
class Solution {
int surfaceArea(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def surfaceArea(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec surface_area(grid :: [[integer]]) :: integer
def surface_area(grid) do

end
end
```

Erlang:

```
-spec surface_area(Grid :: [[integer()]]) -> integer().
surface_area(Grid) ->
.
```

Racket:

```
(define/contract (surface-area grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int surfaceArea(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int surfaceArea(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Surface Area of 3D Shapes
Difficulty: Easy
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def surfaceArea(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def surfaceArea(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} grid
* @return {number}
*/
var surfaceArea = function(grid) {
};
```

TypeScript Solution:

```
/**
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function surfaceArea(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SurfaceArea(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Surface Area of 3D Shapes
* Difficulty: Easy
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int surfaceArea(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Surface Area of 3D Shapes
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func surfaceArea(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun surfaceArea(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func surfaceArea(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Surface Area of 3D Shapes
// Difficulty: Easy
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn surface_area(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def surface_area(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function surfaceArea($grid) {

}
}
```

Dart Solution:

```
class Solution {
int surfaceArea(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def surfaceArea(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec surface_area(grid :: [[integer]]) :: integer
def surface_area(grid) do

end
end
```

Erlang Solution:

```
-spec surface_area(Grid :: [[integer()]]) -> integer().
surface_area(Grid) ->
.
```

Racket Solution:

```
(define/contract (surface-area grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```