Problem 2320: Count Number of Ways to Place Houses

Problem Information

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

Problem Description

There is a street with

n * 2

plots

, where there are

n

plots on each side of the street. The plots on each side are numbered from

1

to

n

. On each plot, a house can be placed.

Return

the number of ways houses can be placed such that no two houses are adjacent to each other on the same side of the street

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Note that if a house is placed on the

i

th

plot on one side of the street, a house can also be placed on the

i

th

plot on the other side of the street.

Example 1:

Input:

n = 1

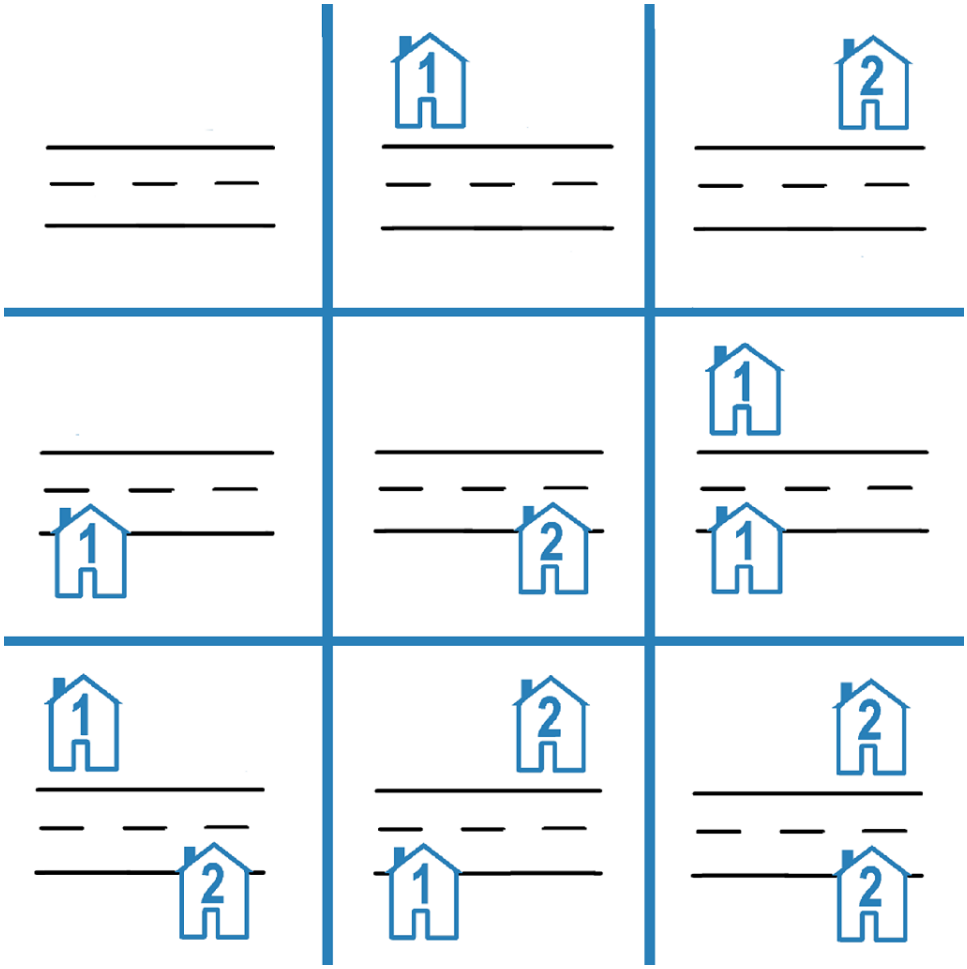
Output:

4

Explanation:

Possible arrangements: 1. All plots are empty. 2. A house is placed on one side of the street. 3. A house is placed on the other side of the street. 4. Two houses are placed, one on each side of the street.

Example 2:



Input:

n = 2

Output:

9

Explanation:

The 9 possible arrangements are shown in the diagram above.

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int countHousePlacements(int n) {

}
};
```

Java:

```
class Solution {
public int countHousePlacements(int n) {

}
}
```

Python3:

```
class Solution:
def countHousePlacements(self, n: int) -> int:
```

Python:

```
class Solution(object):
def countHousePlacements(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var countHousePlacements = function(n) {

};
```

TypeScript:

```
function countHousePlacements(n: number): number {

};
```

C#:

```
public class Solution {
public int CountHousePlacements(int n) {

}
}
```

C:

```
int countHousePlacements(int n) {

}
```

Go:

```
func countHousePlacements(n int) int {

}
```

Kotlin:

```
class Solution {
fun countHousePlacements(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func countHousePlacements(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_house_placements(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def count_house_placements(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function countHousePlacements($n) {

}
}
```

Dart:

```
class Solution {
int countHousePlacements(int n) {

}
}
```

Scala:

```
object Solution {
def countHousePlacements(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_house_placements(n :: integer) :: integer
def count_house_placements(n) do

end
end
```

Erlang:

```
-spec count_house_placements(N :: integer()) -> integer().
count_house_placements(N) ->
.
```

Racket:

```
(define/contract (count-house-placements n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int countHousePlacements(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int countHousePlacements(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Number of Ways to Place Houses
Difficulty: Medium
Tags: tree, dp

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def countHousePlacements(self, n: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def countHousePlacements(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @return {number}
*/
var countHousePlacements = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

function countHousePlacements(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int CountHousePlacements(int n) {

}
}
```

C Solution:

```
/*
* Problem: Count Number of Ways to Place Houses
* Difficulty: Medium
* Tags: tree, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

int countHousePlacements(int n) {

}
```

Go Solution:

```
// Problem: Count Number of Ways to Place Houses
// Difficulty: Medium
// Tags: tree, dp
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

func countHousePlacements(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countHousePlacements(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countHousePlacements(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Number of Ways to Place Houses
// Difficulty: Medium
// Tags: tree, dp
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn count_house_placements(n: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def count_house_placements(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function countHousePlacements($n) {

}
}
```

Dart Solution:

```
class Solution {
int countHousePlacements(int n) {

}
}
```

Scala Solution:

```
object Solution {
def countHousePlacements(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_house_placements(n :: integer) :: integer
def count_house_placements(n) do

end
end
```

Erlang Solution:

```
-spec count_house_placements(N :: integer()) -> integer().
count_house_placements(N) ->
.
```

Racket Solution:

```
(define/contract (count-house-placements n)
(-> exact-integer? exact-integer?)
)
```