Problem 1259: Handshakes That Don't Cross

Problem Information

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

Problem Description

You are given an

even

number of people

numPeople

that stand around a circle and each person shakes hands with someone else so that there are

numPeople / 2

handshakes total.

Return

the number of ways these handshakes could occur such that none of the handshakes cross

.

Since the answer could be very large, return it

modulo

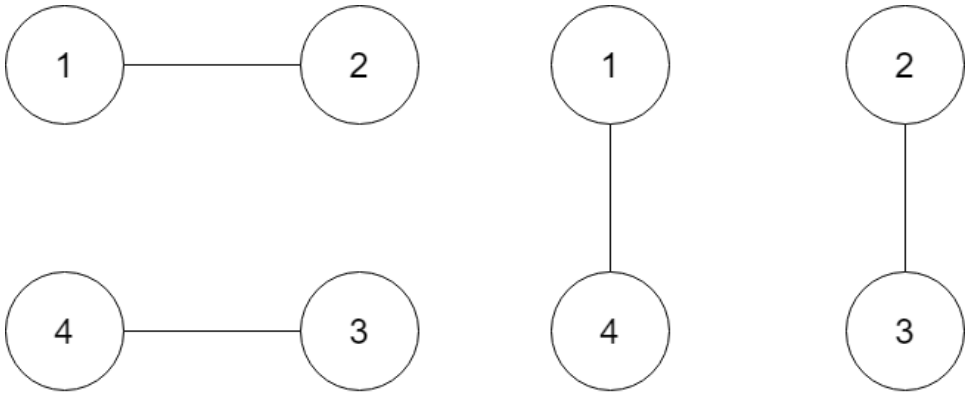
10

9

+ 7

.

Example 1:



[1] [2]

Input:

numPeople = 4

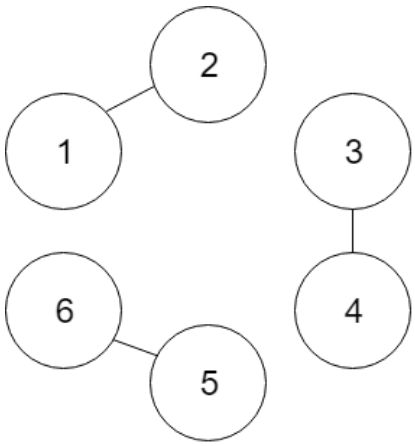
Output:

2

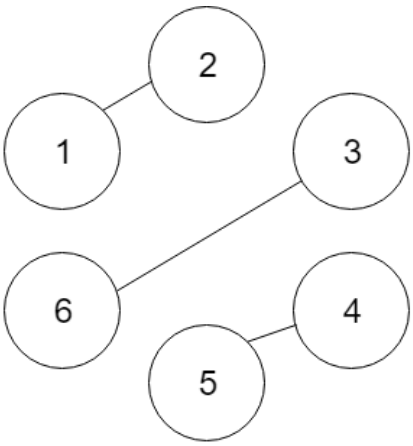
Explanation:

There are two ways to do it, the first way is [(1,2),(3,4)] and the second one is [(2,3),(4,1)].

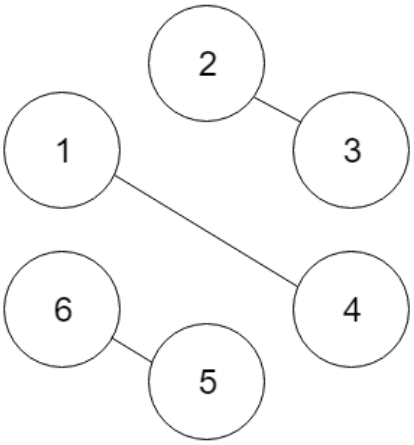
Example 2:



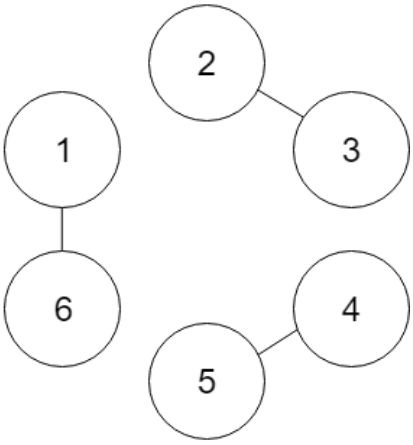
[1]



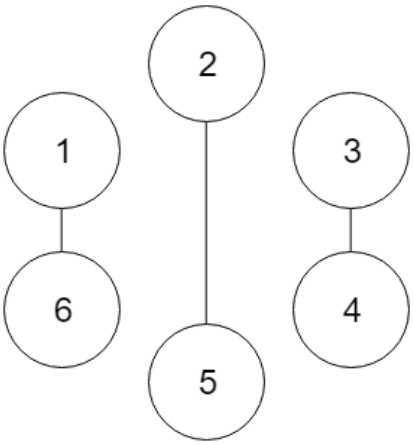
[2]



[3]



[4]



[5]

Input:

numPeople = 6

Output:

5

Constraints:

2 <= numPeople <= 1000

numPeople

is even.

Code Snippets

C++:

```
class Solution {
public:
int numberOfWays(int numPeople) {

}
};
```

Java:

```
class Solution {
public int numberOfWays(int numPeople) {

}
}
```

Python3:

```
class Solution:
def numberOfWays(self, numPeople: int) -> int:
```

Python:

```
class Solution(object):
def numberOfWays(self, numPeople):
"""
:type numPeople: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} numPeople
* @return {number}
*/
var numberOfWays = function(numPeople) {

};
```

TypeScript:

```
function numberOfWays(numPeople: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfWays(int numPeople) {

}
}
```

C:

```
int numberOfWays(int numPeople) {

}
```

Go:

```
func numberOfWays(numPeople int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfWays(numPeople: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfWays(_ numPeople: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_ways(num_people: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num_people
# @return {Integer}
def number_of_ways(num_people)

end
```

PHP:

```
class Solution {

/**
* @param Integer $numPeople
* @return Integer
*/
function numberOfWays($numPeople) {

}
}
```

Dart:

```
class Solution {
int numberOfWays(int numPeople) {

}
```

```
}
```

Scala:

```
object Solution {
def numberOfWays(numPeople: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_ways(num_people :: integer) :: integer
def number_of_ways(num_people) do

end
end
```

Erlang:

```
-spec number_of_ways(NumPeople :: integer()) -> integer().
number_of_ways(NumPeople) ->
.
```

Racket:

```
(define/contract (number-of-ways numPeople)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
*
```

```
* 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:
int numberOfWays(int numPeople) {

}
};
```

Java Solution:

```
/**
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
*
* 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 int numberOfWays(int numPeople) {

}
}
```

Python3 Solution:

```
"""
Problem: Handshakes That Don't Cross
Difficulty: Hard
Tags: dp, math

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 numberOfWays(self, numPeople: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfWays(self, numPeople):
"""
:type numPeople: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
*
* 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} numPeople
* @return {number}
*/
var numberOfWays = function(numPeople) {

};
```

TypeScript Solution:

```
/**
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
```

```
*
* 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 numberOfWays(numPeople: number): number {

};
```

C# Solution:

```
/*
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
*
* 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 int NumberOfWays(int numPeople) {

}
}
```

C Solution:

```
/*
* Problem: Handshakes That Don't Cross
* Difficulty: Hard
* Tags: dp, math
*
* 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
*/

int numberOfWays(int numPeople) {
```

```
}
```

Go Solution:

```
// Problem: Handshakes That Don't Cross
// Difficulty: Hard
// Tags: dp, math
//
// 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 numberOfWays(numPeople int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfWays(numPeople: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfWays(_ numPeople: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Handshakes That Don't Cross
// Difficulty: Hard
// Tags: dp, math
//
// 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 number_of_ways(num_people: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num_people
# @return {Integer}
def number_of_ways(num_people)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $numPeople
* @return Integer
*/
function numberOfWays($numPeople) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfWays(int numPeople) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfWays(numPeople: Int): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_ways(num_people :: integer) :: integer
def number_of_ways(num_people) do

end
end
```

Erlang Solution:

```
-spec number_of_ways(NumPeople :: integer()) -> integer().
number_of_ways(NumPeople) ->
.
```

Racket Solution:

```
(define/contract (number-of-ways numPeople)
(-> exact-integer? exact-integer?)
)
```