Problem 2127: Maximum Employees to Be Invited to a Meeting

Problem Information

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

Problem Description

A company is organizing a meeting and has a list of

n

employees, waiting to be invited. They have arranged for a large

circular

table, capable of seating

any number

of employees.

The employees are numbered from

0

to

n - 1

. Each employee has a

favorite

person and they will attend the meeting

only if

they can sit next to their favorite person at the table. The favorite person of an employee is

not

themself.

Given a

0-indexed

integer array

favorite

, where

favorite[i]

denotes the favorite person of the

i

th

employee, return

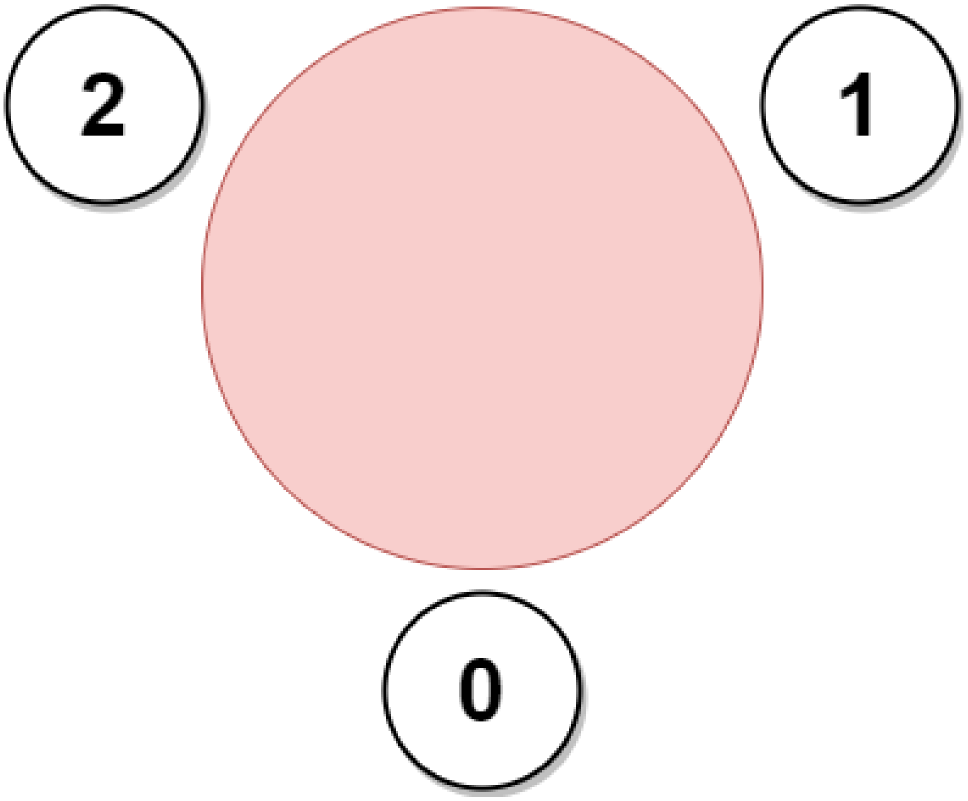
the

maximum number of employees

that can be invited to the meeting

.

Example 1:



Input:

favorite = [2,2,1,2]

Output:

3

Explanation:

The above figure shows how the company can invite employees 0, 1, and 2, and seat them at the round table. All employees cannot be invited because employee 2 cannot sit beside employees 0, 1, and 3, simultaneously. Note that the company can also invite employees 1, 2, and 3, and give them their desired seats. The maximum number of employees that can be invited to the meeting is 3.

Example 2:

Input:

favorite = [1,2,0]

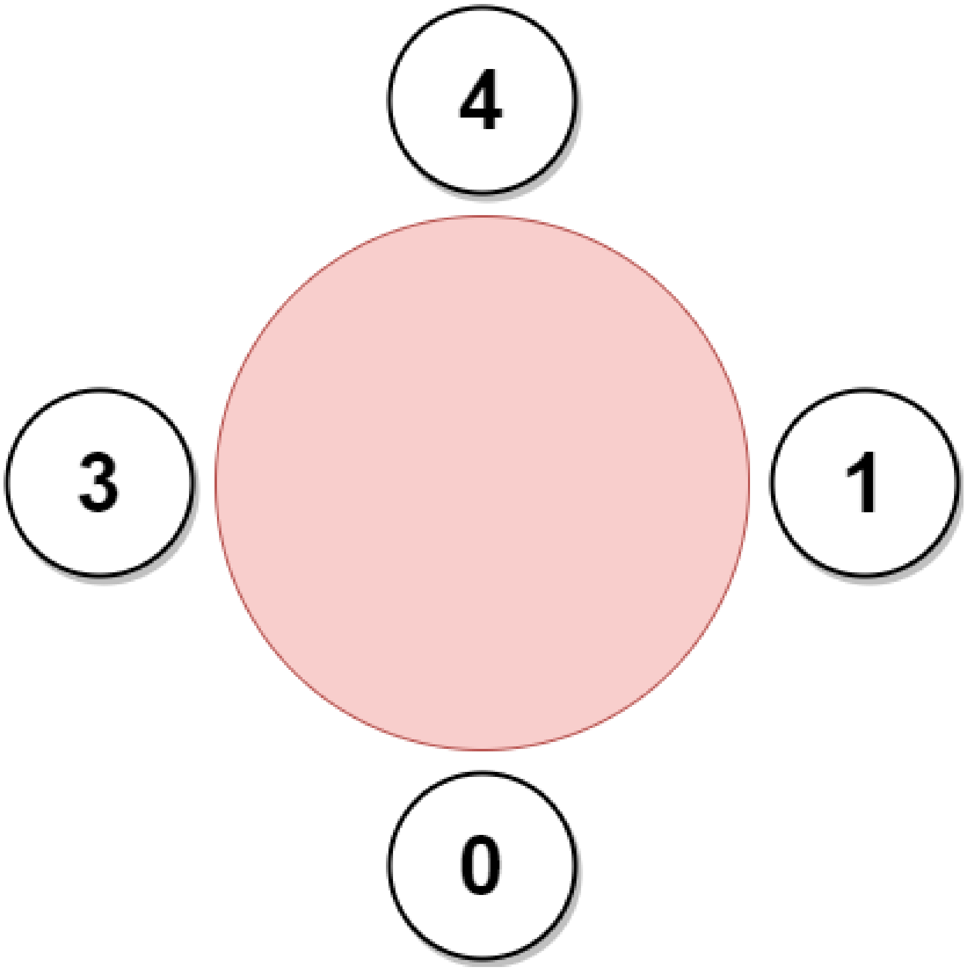
Output:

3

Explanation:

Each employee is the favorite person of at least one other employee, and the only way the company can invite them is if they invite every employee. The seating arrangement will be the same as that in the figure given in example 1: - Employee 0 will sit between employees 2 and 1. - Employee 1 will sit between employees 0 and 2. - Employee 2 will sit between employees 1 and 0. The maximum number of employees that can be invited to the meeting is 3.

Example 3:



Input:

favorite = [3,0,1,4,1]

Output:

4

Explanation:

The above figure shows how the company will invite employees 0, 1, 3, and 4, and seat them at the round table. Employee 2 cannot be invited because the two spots next to their favorite employee 1 are taken. So the company leaves them out of the meeting. The maximum number of employees that can be invited to the meeting is 4.

Constraints:

n == favorite.length

2 <= n <= 10

5

0 <= favorite[i] <= n - 1

favorite[i] != i

Code Snippets

C++:

```
class Solution {
public:
int maximumInvitations(vector<int>& favorite) {

}
};
```

Java:

```
class Solution {
public int maximumInvitations(int[] favorite) {

}
}
```

Python3:

```
class Solution:
def maximumInvitations(self, favorite: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumInvitations(favorite: number[]): number {

};
```

C#:

```
public class Solution {
public int MaximumInvitations(int[] favorite) {

}
}
```

C:

```
int maximumInvitations(int* favorite, int favoriteSize) {

}
```

Go:

```
func maximumInvitations(favorite []int) int {

}
```

Kotlin:

```
class Solution {
fun maximumInvitations(favorite: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumInvitations(_ favorite: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_invitations(favorite: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} favorite
# @return {Integer}
def maximum_invitations(favorite)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $favorite
* @return Integer
*/
function maximumInvitations($favorite) {

}
}
```

Dart:

```
class Solution {
int maximumInvitations(List<int> favorite) {

}
}
```

Scala:

```
object Solution {
def maximumInvitations(favorite: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_invitations(favorite :: [integer]) :: integer
def maximum_invitations(favorite) do

end
end
```

Erlang:

```
-spec maximum_invitations(Favorite :: [integer()]) -> integer().
maximum_invitations(Favorite) ->
.
```

Racket:

```
(define/contract (maximum-invitations favorite)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 maximumInvitations(vector<int>& favorite) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 maximumInvitations(int[] favorite) {
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Employees to Be Invited to a Meeting
Difficulty: Hard
Tags: array, graph, sort, search

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 maximumInvitations(self, favorite: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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[]} favorite
* @return {number}
*/
var maximumInvitations = function(favorite) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 maximumInvitations(favorite: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 MaximumInvitations(int[] favorite) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Employees to Be Invited to a Meeting
* Difficulty: Hard
* Tags: array, graph, sort, search
*
* 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 maximumInvitations(int* favorite, int favoriteSize) {

}
```

Go Solution:

```
// Problem: Maximum Employees to Be Invited to a Meeting
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// 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 maximumInvitations(favorite []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumInvitations(favorite: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumInvitations(_ favorite: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Employees to Be Invited to a Meeting
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// 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 maximum_invitations(favorite: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} favorite
# @return {Integer}
def maximum_invitations(favorite)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $favorite
* @return Integer
*/
function maximumInvitations($favorite) {

}
}
```

Dart Solution:

```
class Solution {
int maximumInvitations(List<int> favorite) {

}
}
```

Scala Solution:

```
object Solution {
def maximumInvitations(favorite: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_invitations(favorite :: [integer]) :: integer
def maximum_invitations(favorite) do

end
end
```

Erlang Solution:

```
-spec maximum_invitations(Favorite :: [integer()]) -> integer().
maximum_invitations(Favorite) ->
.
```

Racket Solution:

```
(define/contract (maximum-invitations favorite)
(-> (listof exact-integer?) exact-integer?)
)
```