

Problem 1989: Maximum Number of People That Can Be Caught in Tag

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are playing a game of tag with your friends. In tag, people are divided into two teams: people who are "it", and people who are not "it". The people who are "it" want to catch as many people as possible who are not "it".

You are given a

0-indexed

integer array

team

containing only zeros (denoting people who are

not

"it") and ones (denoting people who are "it"), and an integer

dist

. A person who is "it" at index

i

can catch any

one

person whose index is in the range

$[i - \text{dist}, i + \text{dist}]$

(

inclusive

) and is

not

"it".

Return

the

maximum

number of people that the people who are "it" can catch

.

Example 1:

Input:

team = [0,1,0,1,0], dist = 3

Output:

2

Explanation:

The person who is "it" at index 1 can catch people in the range $[i - \text{dist}, i + \text{dist}] = [1 - 3, 1 + 3] = [-2, 4]$. They can catch the person who is not "it" at index 2. The person who is "it" at index 3 can catch people in the range $[i - \text{dist}, i + \text{dist}] = [3 - 3, 3 + 3] = [0, 6]$. They can catch the person who is not "it" at index 0. The person who is not "it" at index 4 will not be caught because the people at indices 1 and 3 are already catching one person.

Example 2:

Input:

team = [1], dist = 1

Output:

0

Explanation:

There are no people who are not "it" to catch.

Example 3:

Input:

team = [0], dist = 1

Output:

0

Explanation:

There are no people who are "it" to catch people.

Constraints:

$1 \leq \text{team.length} \leq 10$

5

0 <= team[i] <= 1

1 <= dist <= team.length

Code Snippets

C++:

```
class Solution {
public:
    int catchMaximumAmountofPeople(vector<int>& team, int dist) {

    }
};
```

Java:

```
class Solution {
    public int catchMaximumAmountofPeople(int[] team, int dist) {

    }
}
```

Python3:

```
class Solution:
    def catchMaximumAmountofPeople(self, team: List[int], dist: int) -> int:
```

Python:

```
class Solution(object):
    def catchMaximumAmountofPeople(self, team, dist):
        """
        :type team: List[int]
        :type dist: int
        :rtype: int
        """
```

JavaScript:

```

/**
 * @param {number[]} team
 * @param {number} dist
 * @return {number}
 */
var catchMaximumAmountofPeople = function(team, dist) {

};

```

TypeScript:

```

function catchMaximumAmountofPeople(team: number[], dist: number): number {

};

```

C#:

```

public class Solution {
    public int CatchMaximumAmountofPeople(int[] team, int dist) {

    }
}

```

C:

```

int catchMaximumAmountofPeople(int* team, int teamSize, int dist) {

}

```

Go:

```

func catchMaximumAmountofPeople(team []int, dist int) int {

}

```

Kotlin:

```

class Solution {
    fun catchMaximumAmountofPeople(team: IntArray, dist: Int): Int {

    }
}

```

Swift:

```
class Solution {  
    func catchMaximumAmountofPeople(_ team: [Int], _ dist: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn catch_maximum_amountof_people(team: Vec<i32>, dist: i32) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} team  
# @param {Integer} dist  
# @return {Integer}  
def catch_maximum_amountof_people(team, dist)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $team  
     * @param Integer $dist  
     * @return Integer  
     */  
    function catchMaximumAmountofPeople($team, $dist) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int catchMaximumAmountofPeople(List<int> team, int dist) {
```

```
}  
}
```

Scala:

```
object Solution {  
  def catchMaximumAmountofPeople(team: Array[Int], dist: Int): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec catch_maximum_amount_of_people(team :: [integer], dist :: integer) ::  
    integer  
  def catch_maximum_amount_of_people(team, dist) do  
  
  end  
end
```

Erlang:

```
-spec catch_maximum_amount_of_people(Team :: [integer()], Dist :: integer())  
-> integer().  
catch_maximum_amount_of_people(Team, Dist) ->  
.
```

Racket:

```
(define/contract (catch-maximum-amount-of-people team dist)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```

Solutions

C++ Solution:

```

/*
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 catchMaximumAmountofPeople(vector<int>& team, int dist) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 catchMaximumAmountofPeople(int[] team, int dist) {

    }
}

```

Python3 Solution:

```

"""
Problem: Maximum Number of People That Can Be Caught in Tag
Difficulty: Medium
Tags: array, greedy

```

```

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 catchMaximumAmountofPeople(self, team: List[int], dist: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def catchMaximumAmountofPeople(self, team, dist):
        """
        :type team: List[int]
        :type dist: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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[]} team
 * @param {number} dist
 * @return {number}
 */
var catchMaximumAmountofPeople = function(team, dist) {

};

```

TypeScript Solution:

```
/**
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 catchMaximumAmountofPeople(team: number[], dist: number): number {

};
```

C# Solution:

```
/*
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 CatchMaximumAmountofPeople(int[] team, int dist) {

    }
}
```

C Solution:

```
/*
 * Problem: Maximum Number of People That Can Be Caught in Tag
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 catchMaximumAmountofPeople(int* team, int teamSize, int dist) {

}

```

Go Solution:

```

// Problem: Maximum Number of People That Can Be Caught in Tag
// Difficulty: Medium
// Tags: array, greedy
//
// 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 catchMaximumAmountofPeople(team []int, dist int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun catchMaximumAmountofPeople(team: IntArray, dist: Int): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func catchMaximumAmountofPeople(_ team: [Int], _ dist: Int) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Maximum Number of People That Can Be Caught in Tag
// Difficulty: Medium

```

```

// Tags: array, greedy
//
// 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 catch_maximum_amount_of_people(team: Vec<i32>, dist: i32) -> i32 {

    }
}

```

Ruby Solution:

```

# @param {Integer[]} team
# @param {Integer} dist
# @return {Integer}
def catch_maximum_amount_of_people(team, dist)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $team
     * @param Integer $dist
     * @return Integer
     */
    function catchMaximumAmountofPeople($team, $dist) {

    }

}

```

Dart Solution:

```

class Solution {
    int catchMaximumAmountofPeople(List<int> team, int dist) {

    }
}

```

```
}
```

Scala Solution:

```
object Solution {  
  def catchMaximumAmountofPeople(team: Array[Int], dist: Int): Int = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec catch_maximum_amountof_people(team :: [integer], dist :: integer) ::  
    integer  
  def catch_maximum_amountof_people(team, dist) do  
  
  end  
end
```

Erlang Solution:

```
-spec catch_maximum_amountof_people(Team :: [integer()], Dist :: integer())  
-> integer().  
catch_maximum_amountof_people(Team, Dist) ->  
.
```

Racket Solution:

```
(define/contract (catch-maximum-amountof-people team dist)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```