Problem 1453: Maximum Number of Darts Inside of a Circular Dartboard

Problem Information

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

Problem Description

Alice is throwing

n

darts on a very large wall. You are given an array

darts

where

darts[i] = [x

i

, y

i

]

is the position of the

i

th

dart that Alice threw on the wall.

Bob knows the positions of the

n

darts on the wall. He wants to place a dartboard of radius

r

on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

Given the integer

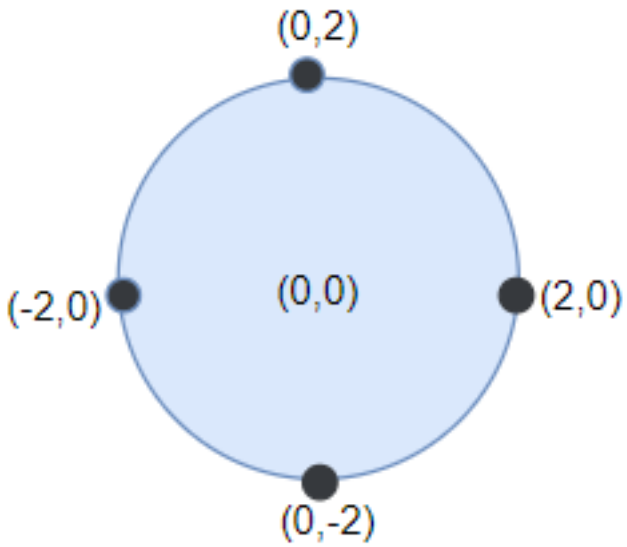
r

, return

the maximum number of darts that can lie on the dartboard

.

Example 1:



Input:

darts = [[-2,0],[2,0],[0,2],[0,-2]], r = 2

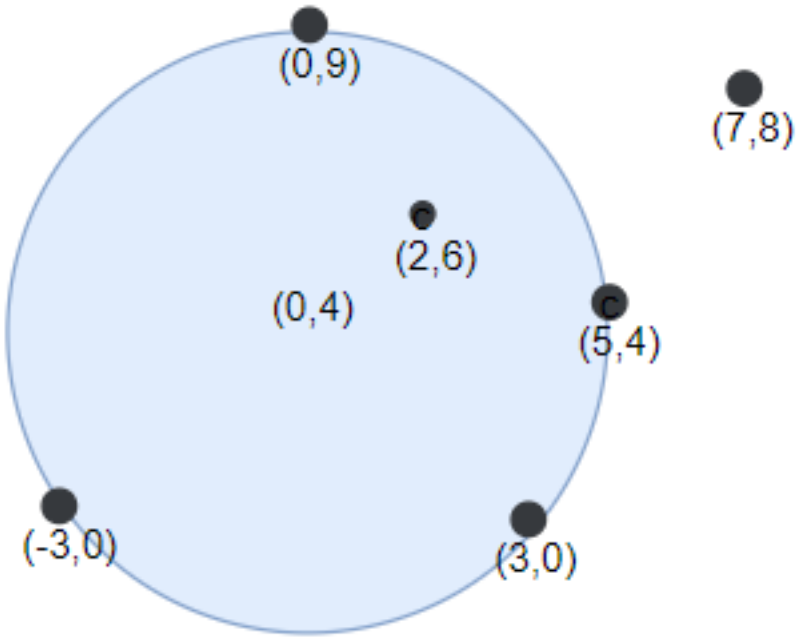
Output:

4

Explanation:

Circle dartboard with center in (0,0) and radius = 2 contain all points.

Example 2:



Input:

darts = [[-3,0],[3,0],[2,6],[5,4],[0,9],[7,8]], r = 5

Output:

5

Explanation:

Circle dartboard with center in (0,4) and radius = 5 contain all points except the point (7,8).

Constraints:

1 <= darts.length <= 100

darts[i].length == 2

-10

4

<= x

i

, y

i

<= 10

4

All the

darts

are unique

1 <= r <= 5000

Code Snippets

C++:

```
class Solution {
public:
int numPoints(vector<vector<int>>& darts, int r) {
```

```
}
};
```

Java:

```
class Solution {
public int numPoints(int[][] darts, int r) {

}
}
```

Python3:

```
class Solution:
def numPoints(self, darts: List[List[int]], r: int) -> int:
```

Python:

```
class Solution(object):
def numPoints(self, darts, r):
"""
:type darts: List[List[int]]
:type r: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} darts
* @param {number} r
* @return {number}
*/
var numPoints = function(darts, r) {

};
```

TypeScript:

```
function numPoints(darts: number[][], r: number): number {

};
```

C#:

```
public class Solution {
public int NumPoints(int[][] darts, int r) {

}
}
```

C:

```
int numPoints(int** darts, int dartsSize, int* dartsColSize, int r) {

}
```

Go:

```
func numPoints(darts [][]int, r int) int {

}
```

Kotlin:

```
class Solution {
fun numPoints(darts: Array<IntArray>, r: Int): Int {

}
}
```

Swift:

```
class Solution {
func numPoints(_ darts: [[Int]], _ r: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_points(darts: Vec<Vec<i32>>, r: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} darts
# @param {Integer} r
# @return {Integer}
def num_points(darts, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $darts
* @param Integer $r
* @return Integer
*/
function numPoints($darts, $r) {

}
}
```

Dart:

```
class Solution {
int numPoints(List<List<int>> darts, int r) {

}
}
```

Scala:

```
object Solution {
def numPoints(darts: Array[Array[Int]], r: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_points(darts :: [[integer]], r :: integer) :: integer
```

```
def num_points(darts, r) do

end
end
```

Erlang:

```
-spec num_points(Darts :: [[integer()]], R :: integer()) -> integer().
num_points(Darts, R) ->
.
```

Racket:

```
(define/contract (num-points darts r)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
* 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 numPoints(vector<vector<int>>& darts, int r) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
* 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 numPoints(int[][] darts, int r) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Darts Inside of a Circular Dartboard
Difficulty: Hard
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 numPoints(self, darts: List[List[int]], r: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numPoints(self, darts, r):
"""
:type darts: List[List[int]]
:type r: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
* 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[][]} darts
* @param {number} r
* @return {number}
*/
var numPoints = function(darts, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
* 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 numPoints(darts: number[][], r: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
```

```
* 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 NumPoints(int[][] darts, int r) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Darts Inside of a Circular Dartboard
* Difficulty: Hard
* 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 numPoints(int** darts, int dartsSize, int* dartsColSize, int r) {

}
```

Go Solution:

```
// Problem: Maximum Number of Darts Inside of a Circular Dartboard
// Difficulty: Hard
// 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 numPoints(darts [][]int, r int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun numPoints(darts: Array<IntArray>, r: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numPoints(_ darts: [[Int]], _ r: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Darts Inside of a Circular Dartboard
// Difficulty: Hard
// 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 num_points(darts: Vec<Vec<i32>>, r: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} darts
# @param {Integer} r
# @return {Integer}
def num_points(darts, r)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $darts
* @param Integer $r
* @return Integer
*/
function numPoints($darts, $r) {

}
}
```

Dart Solution:

```
class Solution {
int numPoints(List<List<int>> darts, int r) {

}
}
```

Scala Solution:

```
object Solution {
def numPoints(darts: Array[Array[Int]], r: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_points(darts :: [[integer]], r :: integer) :: integer
def num_points(darts, r) do

end
end
```

Erlang Solution:

```
-spec num_points(Darts :: [[integer()]], R :: integer()) -> integer().
num_points(Darts, R) ->
.
```

Racket Solution:

```
(define/contract (num-points darts r)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```