Problem 3710: Maximum Partition Factor

Problem Information

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

Problem Description

You are given a 2D integer array

points

, where

points[i] = [x

i

, y

i

]

represents the coordinates of the

i

th

point on the Cartesian plane.

The

Manhattan distance

between two points

points[i] = [x

i

, y

i

]

and

points[j] = [x

j

, y

j

]

is

|x

i

- x

j

| + |y

i

- y

j

|

.

Split the

n

points into

exactly two non-empty

groups. The

partition factor

of a split is the

minimum

Manhattan distance among all unordered pairs of points that lie in the same group.

Return the

maximum

possible

partition factor

over all valid splits.

Note: A group of size 1 contributes no intra-group pairs. When

n = 2

(both groups size 1), there are no intra-group pairs, so define the partition factor as 0.

Example 1:

Input:

points = [[0,0],[0,2],[2,0],[2,2]]

Output:

4

Explanation:

We split the points into two groups:

{[0, 0], [2, 2]}

and

{[0, 2], [2, 0]}

.

In the first group, the only pair has Manhattan distance

|0 - 2| + |0 - 2| = 4

.

In the second group, the only pair also has Manhattan distance

|0 - 2| + |2 - 0| = 4

.

The partition factor of this split is

min(4, 4) = 4

, which is maximal.

Example 2:

Input:

points = [[0,0],[0,1],[10,0]]

Output:

11

Explanation:

We split the points into two groups:

{[0, 1], [10, 0]}

and

{[0, 0]}

.

In the first group, the only pair has Manhattan distance

|0 - 10| + |1 - 0| = 11

.

The second group is a singleton, so it contributes no pairs.

The partition factor of this split is

11

, which is maximal.

Constraints:

2 <= points.length <= 500

points[i] = [x

i

, y

i

]

-10

8

<= x

i

, y

i

<= 10

8

Code Snippets

C++:

```
class Solution {
public:
```

```
int maxPartitionFactor(vector<vector<int>>& points) {

}
};
```

Java:

```
class Solution {
public int maxPartitionFactor(int[][] points) {

}
}
```

Python3:

```
class Solution:
def maxPartitionFactor(self, points: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxPartitionFactor(points: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxPartitionFactor(int[][] points) {

}
}
```

C:

```
int maxPartitionFactor(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func maxPartitionFactor(points [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxPartitionFactor(points: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxPartitionFactor(_ points: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_partition_factor(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} points
# @return {Integer}
def max_partition_factor(points)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function maxPartitionFactor($points) {

}
}
```

Dart:

```
class Solution {
int maxPartitionFactor(List<List<int>> points) {

}
}
```

Scala:

```
object Solution {
def maxPartitionFactor(points: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_partition_factor(points :: [[integer]]) :: integer
def max_partition_factor(points) do
```

```
end
end
```

Erlang:

```
-spec max_partition_factor(Points :: [[integer()]]) -> integer().
max_partition_factor(Points) ->
.
```

Racket:

```
(define/contract (max-partition-factor points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Partition Factor
* Difficulty: Hard
* Tags: array, graph, 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 maxPartitionFactor(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Partition Factor
```

```
* Difficulty: Hard
* Tags: array, graph, 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 maxPartitionFactor(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Partition Factor
Difficulty: Hard
Tags: array, graph, 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 maxPartitionFactor(self, points: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Partition Factor
* Difficulty: Hard
* Tags: array, graph, 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[][]} points
* @return {number}
*/
var maxPartitionFactor = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Partition Factor
* Difficulty: Hard
* Tags: array, graph, 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 maxPartitionFactor(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Partition Factor
* Difficulty: Hard
* Tags: array, graph, 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 MaxPartitionFactor(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Partition Factor
* Difficulty: Hard
* Tags: array, graph, 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 maxPartitionFactor(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Partition Factor
// Difficulty: Hard
// Tags: array, graph, 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 maxPartitionFactor(points [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxPartitionFactor(points: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxPartitionFactor(_ points: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Partition Factor
// Difficulty: Hard
// Tags: array, graph, 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 max_partition_factor(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @return {Integer}
def max_partition_factor(points)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $points
* @return Integer
*/
function maxPartitionFactor($points) {

}
}
```

Dart Solution:

```
class Solution {
int maxPartitionFactor(List<List<int>> points) {

}
}
```

Scala Solution:

```
object Solution {
def maxPartitionFactor(points: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_partition_factor(points :: [[integer]]) :: integer
def max_partition_factor(points) do

end
end
```

Erlang Solution:

```
-spec max_partition_factor(Points :: [[integer()]]) -> integer().
max_partition_factor(Points) ->
.
```

Racket Solution:

```
(define/contract (max-partition-factor points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```