Problem 1066: Campus Bikes II

Problem Information

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

Problem Description

On a campus represented as a 2D grid, there are

n

workers and

m

bikes, with

n <= m

. Each worker and bike is a 2D coordinate on this grid.

We assign one unique bike to each worker so that the sum of the

Manhattan distances

between each worker and their assigned bike is minimized.

Return

the minimum possible sum of Manhattan distances between each worker and their assigned bike

.

The

Manhattan distance

between two points

p1

and

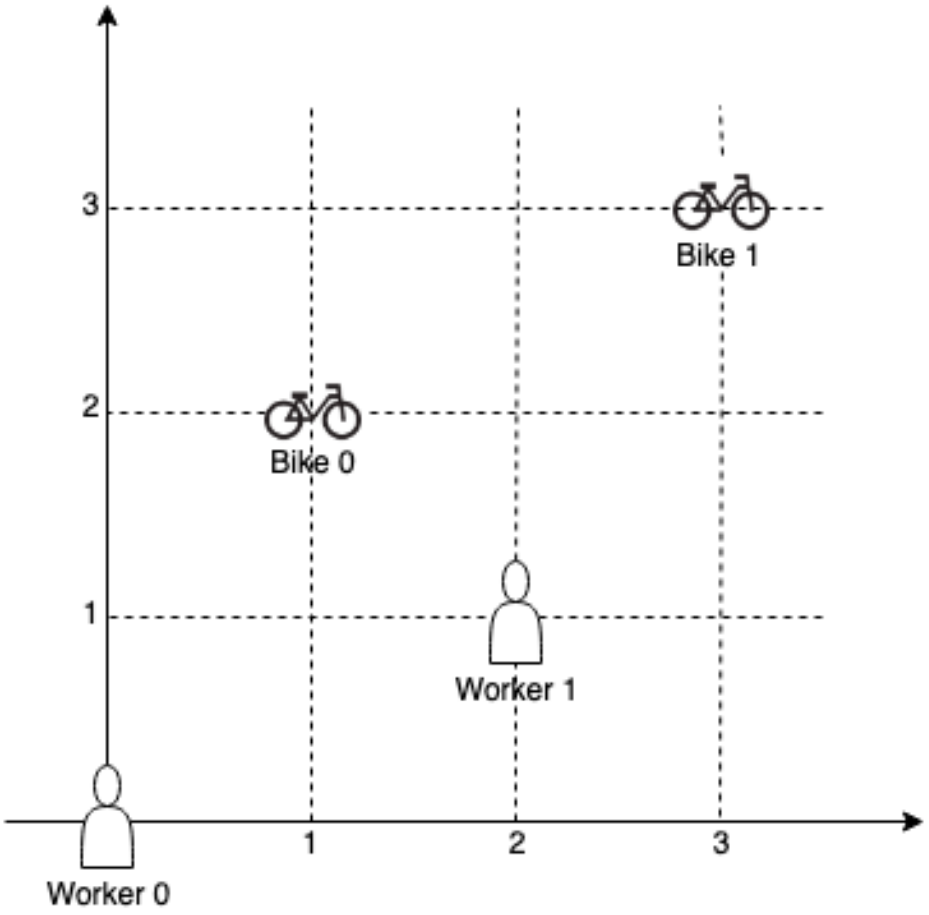
p2

is

Manhattan(p1, p2) = |p1.x - p2.x| + |p1.y - p2.y|

.

Example 1:



Input:

workers = [[0,0],[2,1]], bikes = [[1,2],[3,3]]

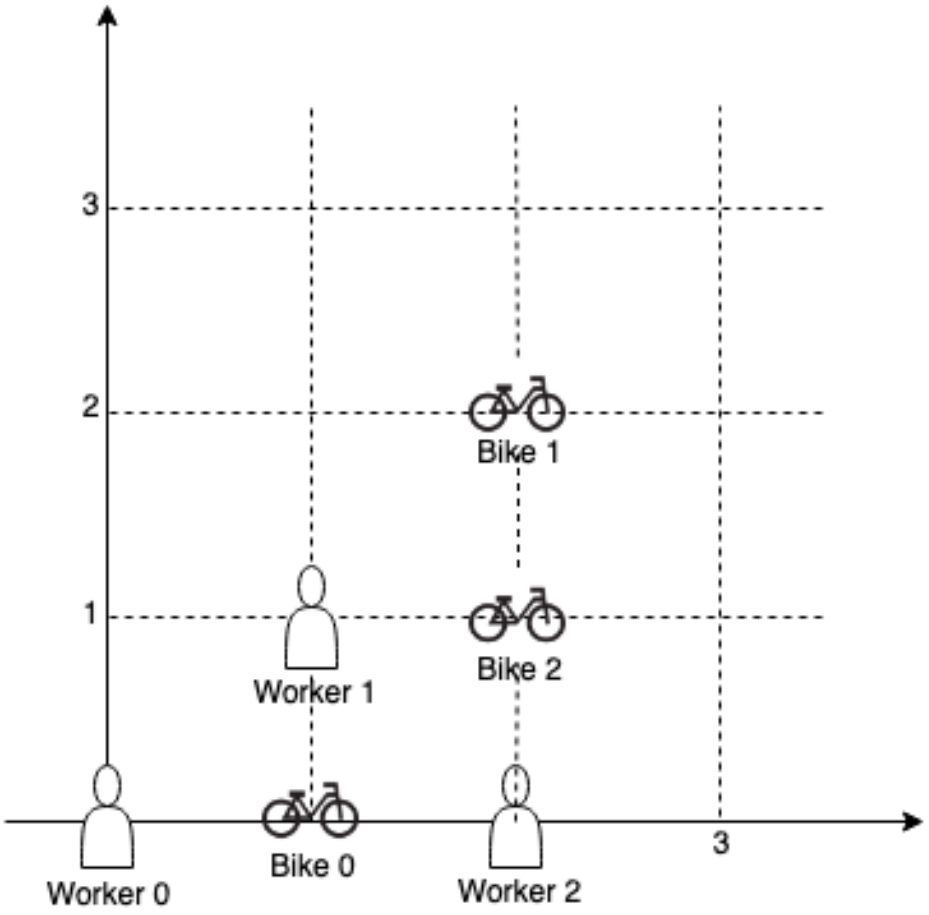
Output:

6

Explanation:

We assign bike 0 to worker 0, bike 1 to worker 1. The Manhattan distance of both assignments is 3, so the output is 6.

Example 2:



Input:

workers = [[0,0],[1,1],[2,0]], bikes = [[1,0],[2,2],[2,1]]

Output:

4

Explanation:

We first assign bike 0 to worker 0, then assign bike 1 to worker 1 or worker 2, bike 2 to worker 2 or worker 1. Both assignments lead to sum of the Manhattan distances as 4.

Example 3:

Input:

workers = [[0,0],[1,0],[2,0],[3,0],[4,0]], bikes = [[0,999],[1,999],[2,999],[3,999],[4,999]]

Output:

4995

Constraints:

n == workers.length

m == bikes.length

1 <= n <= m <= 10

workers[i].length == 2

bikes[i].length == 2

0 <= workers[i][0], workers[i][1], bikes[i][0], bikes[i][1] < 1000

All the workers and the bikes locations are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int assignBikes(vector<vector<int>>& workers, vector<vector<int>>& bikes) {

}
};
```

Java:

```
class Solution {
public int assignBikes(int[][] workers, int[][] bikes) {
```

```
}
}
```

Python3:

```
class Solution:
def assignBikes(self, workers: List[List[int]], bikes: List[List[int]]) ->
int:
```

Python:

```
class Solution(object):
def assignBikes(self, workers, bikes):
"""
:type workers: List[List[int]]
:type bikes: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} workers
* @param {number[][]} bikes
* @return {number}
*/
var assignBikes = function(workers, bikes) {

};
```

TypeScript:

```
function assignBikes(workers: number[][], bikes: number[][]): number {

};
```

C#:

```
public class Solution {
public int AssignBikes(int[][] workers, int[][] bikes) {
```

```
}
}
```

C:

```
int assignBikes(int** workers, int workersSize, int* workersColSize, int**
bikes, int bikesSize, int* bikesColSize) {

}
```

Go:

```
func assignBikes(workers [][]int, bikes [][]int) int {

}
```

Kotlin:

```
class Solution {
fun assignBikes(workers: Array<IntArray>, bikes: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func assignBikes(_ workers: [[Int]], _ bikes: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn assign_bikes(workers: Vec<Vec<i32>>, bikes: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} workers
# @param {Integer[][]} bikes
# @return {Integer}
def assign_bikes(workers, bikes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $workers
* @param Integer[][] $bikes
* @return Integer
*/
function assignBikes($workers, $bikes) {

}
}
```

Dart:

```
class Solution {
int assignBikes(List<List<int>> workers, List<List<int>> bikes) {

}
}
```

Scala:

```
object Solution {
def assignBikes(workers: Array[Array[Int]], bikes: Array[Array[Int]]): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec assign_bikes(workers :: [[integer]], bikes :: [[integer]]) :: integer
def assign_bikes(workers, bikes) do
```

```
end
end
```

Erlang:

```
-spec assign_bikes(Workers :: [[integer()]], Bikes :: [[integer()]]) ->
integer().
assign_bikes(Workers, Bikes) ->
.
```

Racket:

```
(define/contract (assign-bikes workers bikes)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int assignBikes(vector<vector<int>>& workers, vector<vector<int>>& bikes) {

}
};
```

Java Solution:

```
/**
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int assignBikes(int[][] workers, int[][] bikes) {

}
}
```

Python3 Solution:

```
"""
Problem: Campus Bikes II
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def assignBikes(self, workers: List[List[int]], bikes: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def assignBikes(self, workers, bikes):
"""
:type workers: List[List[int]]
:type bikes: List[List[int]]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} workers
* @param {number[][]} bikes
* @return {number}
*/
var assignBikes = function(workers, bikes) {

};
```

TypeScript Solution:

```
/**
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function assignBikes(workers: number[][], bikes: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int AssignBikes(int[][] workers, int[][] bikes) {

}
}
```

C Solution:

```
/*
* Problem: Campus Bikes II
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int assignBikes(int** workers, int workersSize, int* workersColSize, int**
bikes, int bikesSize, int* bikesColSize) {

}
```

Go Solution:

```
// Problem: Campus Bikes II
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func assignBikes(workers [][]int, bikes [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun assignBikes(workers: Array<IntArray>, bikes: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func assignBikes(_ workers: [[Int]], _ bikes: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Campus Bikes II
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn assign_bikes(workers: Vec<Vec<i32>>, bikes: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} workers
# @param {Integer[][]} bikes
```

```
# @return {Integer}
def assign_bikes(workers, bikes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $workers
* @param Integer[][] $bikes
* @return Integer
*/
function assignBikes($workers, $bikes) {

}
}
```

Dart Solution:

```
class Solution {
int assignBikes(List<List<int>> workers, List<List<int>> bikes) {

}
}
```

Scala Solution:

```
object Solution {
def assignBikes(workers: Array[Array[Int]], bikes: Array[Array[Int]]): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec assign_bikes(workers :: [[integer]], bikes :: [[integer]]) :: integer
def assign_bikes(workers, bikes) do
```

```
end
end
```

Erlang Solution:

```
-spec assign_bikes(Workers :: [[integer()]], Bikes :: [[integer()]]) ->
integer().
assign_bikes(Workers, Bikes) ->
.
```

Racket Solution:

```
(define/contract (assign-bikes workers bikes)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```