# Problem 1036: Escape a Large Maze

## Problem Information

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

## Problem Description

There is a 1 million by 1 million grid on an XY-plane, and the coordinates of each grid square are

(x, y)

.

We start at the

source = [s

x

, s

y

]

square and want to reach the

target = [t

x

, t

y

]

square. There is also an array of

blocked

squares, where each

blocked[i] = [x

i

, y

i

]

represents a blocked square with coordinates

(x

i

, y

i

)

.

Each move, we can walk one square north, east, south, or west if the square is

not

in the array of

blocked

squares. We are also not allowed to walk outside of the grid.

Return

true

if and only if it is possible to reach the

target

square from the

source

square through a sequence of valid moves

.

Example 1:

Input:

blocked = [[0,1],[1,0]], source = [0,0], target = [0,2]

Output:

false

Explanation:

The target square is inaccessible starting from the source square because we cannot move. We cannot move north or east because those squares are blocked. We cannot move south or west because we cannot go outside of the grid.

Example 2:

,

Input:

blocked = [], source = [0,0], target = [999999,999999]

Output:

true

Explanation:

Because there are no blocked cells, it is possible to reach the target square.

Constraints:

0 <= blocked.length <= 200

blocked[i].length == 2

0 <= x

i

, y

i

< 10

6

source.length == target.length == 2

0 <= s

x

, s

y

, t

x

, t

y

< 10

6

source != target

It is guaranteed that

source

and

target

are not blocked.

## Code Snippets

**C++:**

```
class Solution {
public:
bool isEscapePossible(vector<vector<int>>& blocked, vector<int>& source,
vector<int>& target) {

}
};
```

**Java:**

```
class Solution {
public boolean isEscapePossible(int[][] blocked, int[] source, int[] target)
{

}
}
```

**Python3:**

```
class Solution:
def isEscapePossible(self, blocked: List[List[int]], source: List[int],
target: List[int]) -> bool:
```

**Python:**

```
class Solution(object):
def isEscapePossible(self, blocked, source, target):
"""
:type blocked: List[List[int]]
:type source: List[int]
:type target: List[int]
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number[][]} blocked
* @param {number[]} source
* @param {number[]} target
* @return {boolean}
*/
var isEscapePossible = function(blocked, source, target) {

};
```

**TypeScript:**

```
function isEscapePossible(blocked: number[][], source: number[], target:
number[]): boolean {

};
```

**C#:**

```
public class Solution {
public bool IsEscapePossible(int[][] blocked, int[] source, int[] target) {

}
}
```

**C:**

```
bool isEscapePossible(int** blocked, int blockedSize, int* blockedColSize,
int* source, int sourceSize, int* target, int targetSize) {

}
```

**Go:**

```
func isEscapePossible(blocked [][]int, source []int, target []int) bool {

}
```

**Kotlin:**

```
class Solution {
fun isEscapePossible(blocked: Array<IntArray>, source: IntArray, target:
IntArray): Boolean {

}
}
```

**Swift:**

```
class Solution {
func isEscapePossible(_ blocked: [[Int]], _ source: [Int], _ target: [Int])
-> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn is_escape_possible(blocked: Vec<Vec<i32>>, source: Vec<i32>, target:
```

```
Vec<i32>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[][]} blocked
# @param {Integer[]} source
# @param {Integer[]} target
# @return {Boolean}
def is_escape_possible(blocked, source, target)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $blocked
* @param Integer[] $source
* @param Integer[] $target
* @return Boolean
*/
function isEscapePossible($blocked, $source, $target) {

}
}
```

**Dart:**

```
class Solution {
bool isEscapePossible(List<List<int>> blocked, List<int> source, List<int>
target) {

}
}
```

**Scala:**

```
object Solution {
def isEscapePossible(blocked: Array[Array[Int]], source: Array[Int], target:
```

```
Array[Int]): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec is_escape_possible(blocked :: [[integer]], source :: [integer], target
:: [integer]) :: boolean
def is_escape_possible(blocked, source, target) do

end
end
```

**Erlang:**

```
-spec is_escape_possible(Blocked :: [[integer()]], Source :: [integer()],
Target :: [integer()]) -> boolean().
is_escape_possible(Blocked, Source, Target) ->
.
```

**Racket:**

```
(define/contract (is-escape-possible blocked source target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) boolean?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
bool isEscapePossible(vector<vector<int>>& blocked, vector<int>& source,
vector<int>& target) {

}
};
```

**Java Solution:**

```
/**
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean isEscapePossible(int[][] blocked, int[] source, int[] target)
{

}
}
```

**Python3 Solution:**

```
"""
Problem: Escape a Large Maze
Difficulty: Hard
Tags: array, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def isEscapePossible(self, blocked: List[List[int]], source: List[int],
target: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def isEscapePossible(self, blocked, source, target):
"""
:type blocked: List[List[int]]
:type source: List[int]
:type target: List[int]
:rtype: bool
"""
```

**JavaScript Solution:**

```
/**
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} blocked
* @param {number[]} source
* @param {number[]} target
* @return {boolean}
*/
var isEscapePossible = function(blocked, source, target) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function isEscapePossible(blocked: number[][], source: number[], target:
number[]): boolean {

};
```

**C# Solution:**

```
/*
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool IsEscapePossible(int[][] blocked, int[] source, int[] target) {

}
}
```

**C Solution:**

```
/*
* Problem: Escape a Large Maze
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool isEscapePossible(int** blocked, int blockedSize, int* blockedColSize,
int* source, int sourceSize, int* target, int targetSize) {

}
```

**Go Solution:**

```
// Problem: Escape a Large Maze
// Difficulty: Hard
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isEscapePossible(blocked [][]int, source []int, target []int) bool {

}
```

**Kotlin Solution:**

```
class Solution {
fun isEscapePossible(blocked: Array<IntArray>, source: IntArray, target:
IntArray): Boolean {

}
}
```

**Swift Solution:**

```
class Solution {
func isEscapePossible(_ blocked: [[Int]], _ source: [Int], _ target: [Int])
-> Bool {

}
}
```

**Rust Solution:**

```
// Problem: Escape a Large Maze
// Difficulty: Hard
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn is_escape_possible(blocked: Vec<Vec<i32>>, source: Vec<i32>, target:
Vec<i32>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} blocked
# @param {Integer[]} source
# @param {Integer[]} target
# @return {Boolean}
def is_escape_possible(blocked, source, target)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $blocked
* @param Integer[] $source
* @param Integer[] $target
* @return Boolean
*/
function isEscapePossible($blocked, $source, $target) {

}
}
```

### Dart Solution:

```
class Solution {
bool isEscapePossible(List<List<int>> blocked, List<int> source, List<int>
target) {

}
}
```

## Scala Solution:

```
object Solution {
def isEscapePossible(blocked: Array[Array[Int]], source: Array[Int], target:
Array[Int]): Boolean = {

}
}
```

## Elixir Solution:

```
defmodule Solution do
@spec is_escape_possible(blocked :: [[integer]], source :: [integer], target
:: [integer]) :: boolean
def is_escape_possible(blocked, source, target) do

end
end
```

## Erlang Solution:

```
-spec is_escape_possible(Blocked :: [[integer()]], Source :: [integer()],
Target :: [integer()]) -> boolean().
is_escape_possible(Blocked, Source, Target) ->
.
```

## Racket Solution:

```
(define/contract (is-escape-possible blocked source target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) boolean?)
)
```