Problem 2304: Minimum Path Cost in a Grid

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

integer matrix

grid

consisting of

distinct

integers from

0

to

m * n - 1

. You can move in this matrix from a cell to any other cell in the

next

row. That is, if you are in cell

(x, y)

such that

x < m - 1

, you can move to any of the cells

(x + 1, 0)

,

(x + 1, 1)

, ...,

(x + 1, n - 1)

.

Note

that it is not possible to move from cells in the last row.

Each possible move has a cost given by a

0-indexed

2D array

moveCost

of size

(m * n) x n

, where

moveCost[i][j]

is the cost of moving from a cell with value

i

to a cell in column

j

of the next row. The cost of moving from cells in the last row of

grid

can be ignored.

The cost of a path in

grid

is the

sum

of all values of cells visited plus the

sum

of costs of all the moves made. Return

the

minimum

cost of a path that starts from any cell in the

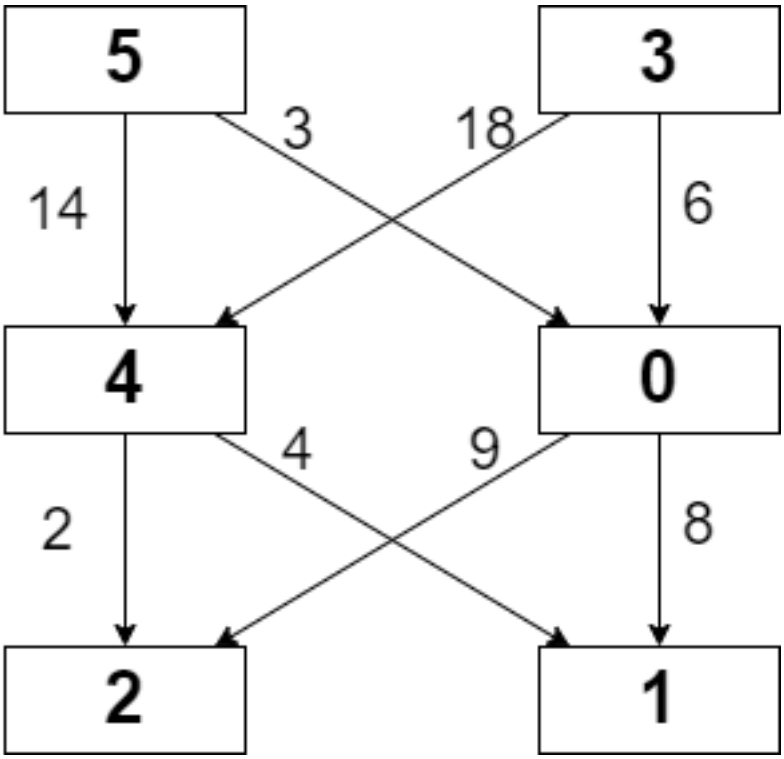
first

row and ends at any cell in the

last

row.

Example 1:



Input:

grid = [[5,3],[4,0],[2,1]], moveCost = [[9,8],[1,5],[10,12],[18,6],[2,4],[14,3]]

Output:

17

Explanation:

The path with the minimum possible cost is the path 5 -> 0 -> 1. - The sum of the values of cells visited is 5 + 0 + 1 = 6. - The cost of moving from 5 to 0 is 3. - The cost of moving from 0 to 1 is 8. So the total cost of the path is 6 + 3 + 8 = 17.

Example 2:

Input:

grid = [[5,1,2],[4,0,3]], moveCost = [[12,10,15],[20,23,8],[21,7,1],[8,1,13],[9,10,25],[5,3,2]]

Output:

6

Explanation:

The path with the minimum possible cost is the path 2 -> 3. - The sum of the values of cells visited is 2 + 3 = 5. - The cost of moving from 2 to 3 is 1. So the total cost of this path is 5 + 1 = 6.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 50

grid

consists of distinct integers from

0

to

m * n - 1

.

moveCost.length == m * n

moveCost[i].length == n

1 <= moveCost[i][j] <= 100

Code Snippets

C++:

```
class Solution {
public:
int minPathCost(vector<vector<int>>& grid, vector<vector<int>>& moveCost) {

}
};
```

Java:

```
class Solution {
public int minPathCost(int[][] grid, int[][] moveCost) {

}
}
```

Python3:

```
class Solution:
def minPathCost(self, grid: List[List[int]], moveCost: List[List[int]]) ->
int:
```

Python:

```
class Solution(object):
def minPathCost(self, grid, moveCost):
"""
:type grid: List[List[int]]
:type moveCost: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
```

```
* @param {number[][]} moveCost
* @return {number}
*/
var minPathCost = function(grid, moveCost) {

};
```

TypeScript:

```
function minPathCost(grid: number[][], moveCost: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinPathCost(int[][] grid, int[][] moveCost) {

}
}
```

C:

```
int minPathCost(int** grid, int gridSize, int* gridColSize, int** moveCost,
int moveCostSize, int* moveCostColSize) {

}
```

Go:

```
func minPathCost(grid [][]int, moveCost [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minPathCost(grid: Array<IntArray>, moveCost: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minPathCost(_ grid: [[Int]], _ moveCost: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_path_cost(grid: Vec<Vec<i32>>, move_cost: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer[][]} move_cost
# @return {Integer}
def min_path_cost(grid, move_cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[][] $moveCost
* @return Integer
*/
function minPathCost($grid, $moveCost) {

}
}
```

Dart:

```
class Solution {
int minPathCost(List<List<int>> grid, List<List<int>> moveCost) {
```

```
}
}
```

Scala:

```
object Solution {
def minPathCost(grid: Array[Array[Int]], moveCost: Array[Array[Int]]): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec min_path_cost(grid :: [[integer]], move_cost :: [[integer]]) :: integer
def min_path_cost(grid, move_cost) do

end
end
```

Erlang:

```
-spec min_path_cost(Grid :: [[integer()]], MoveCost :: [[integer()]]) ->
integer().
min_path_cost(Grid, MoveCost) ->
.
```

Racket:

```
(define/contract (min-path-cost grid moveCost)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Path Cost in a Grid
* 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 minPathCost(vector<vector<int>>& grid, vector<vector<int>>& moveCost) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Path Cost in a Grid
* 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 minPathCost(int[][] grid, int[][] moveCost) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Path Cost in a Grid
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 minPathCost(self, grid: List[List[int]], moveCost: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minPathCost(self, grid, moveCost):
"""
:type grid: List[List[int]]
:type moveCost: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Path Cost in a Grid
* 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[][]} grid
* @param {number[][]} moveCost
* @return {number}
*/
var minPathCost = function(grid, moveCost) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Path Cost in a Grid
* 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 minPathCost(grid: number[][], moveCost: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Path Cost in a Grid
* 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 MinPathCost(int[][] grid, int[][] moveCost) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Path Cost in a Grid
* 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 minPathCost(int** grid, int gridSize, int* gridColSize, int** moveCost,
int moveCostSize, int* moveCostColSize) {

}
```

Go Solution:

```
// Problem: Minimum Path Cost in a Grid
// 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 minPathCost(grid [][]int, moveCost [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minPathCost(grid: Array<IntArray>, moveCost: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minPathCost(_ grid: [[Int]], _ moveCost: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Path Cost in a Grid
// 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 min_path_cost(grid: Vec<Vec<i32>>, move_cost: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer[][]} move_cost
# @return {Integer}
def min_path_cost(grid, move_cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[][] $moveCost
* @return Integer
*/
function minPathCost($grid, $moveCost) {

}
}
```

Dart Solution:

```
class Solution {
int minPathCost(List<List<int>> grid, List<List<int>> moveCost) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minPathCost(grid: Array[Array[Int]], moveCost: Array[Array[Int]]): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_path_cost(grid :: [[integer]], move_cost :: [[integer]]) :: integer
def min_path_cost(grid, move_cost) do

end
end
```

Erlang Solution:

```
-spec min_path_cost(Grid :: [[integer()]], MoveCost :: [[integer()]]) ->
integer().
min_path_cost(Grid, MoveCost) ->
.
```

Racket Solution:

```
(define/contract (min-path-cost grid moveCost)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```