<https://leetcode.com/problems/minimum-cost-homecoming-of-a-robot-in-a-grid/description/>

There is an m x n grid, where (0, 0) is the top-left cell and (m - 1, n - 1) is the bottom-right cell. You are given an integer array startPos where startPos = [startrow, startcol] indicates that **initially**, a **robot** is at the cell (startrow, startcol). You are also given an integer array homePos where homePos = [homerow, homecol] indicates that its **home** is at the cell (homerow, homecol).

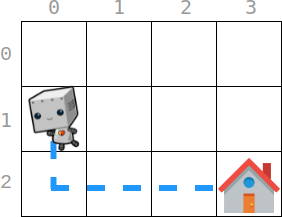
The robot needs to go to its home. It can move one cell in four directions: **left**, **right**, **up**, or **down**, and it can not move outside the boundary. Every move incurs some cost. You are further given two **0-indexed** integer arrays: rowCosts of length m and colCosts of length n.

If the robot moves **up** or **down** into a cell whose **row** is r, then this move costs rowCosts[r].

If the robot moves **left** or **right** into a cell whose **column** is c, then this move costs colCosts[c].

Return *the****minimum total cost****for this robot to return home*.

**Example 1:**



**Input:** startPos = [1, 0], homePos = [2, 3], rowCosts = [5, 4, 3], colCosts = [8, 2, 6, 7]

**Output:** 18

**Explanation:** One optimal path is that:Starting from (1, 0)

-> It goes down to (**2**, 0). This move costs rowCosts[2] = 3.

-> It goes right to (2, **1**). This move costs colCosts[1] = 2.

-> It goes right to (2, **2**). This move costs colCosts[2] = 6.

-> It goes right to (2, **3**). This move costs colCosts[3] = 7.

The total cost is 3 + 2 + 6 + 7 = 18

**Example 2:**

**Input:** startPos = [0, 0], homePos = [0, 0], rowCosts = [5], colCosts = [26]

**Output:** 0

**Explanation:** The robot is already at its home. Since no moves occur, the total cost is 0.

**Constraints:**

m == rowCosts.length

n == colCosts.length

1 <= m, n <= 105

0 <= rowCosts[r], colCosts[c] <= 104

startPos.length == 2

homePos.length == 2

0 <= startrow, homerow < m

0 <= startcol, homecol < n

**Attempt 1: 2025-06-08**

**Solution 1: Greedy (10 min)**

class Solution {

    public int minCost(int[] startPos, int[] homePos, int[] rowCosts, int[] colCosts) {

        int totalCost = 0;

        int curRow = startPos[0];

        int curCol = startPos[1];

        int homeRow = homePos[0];

        int homeCol = homePos[1];

        while(curRow != homeRow) {

            if(curRow < homeRow) {

                curRow++;

            } else {

                curRow--;

            }

            totalCost += rowCosts[curRow];

        }

        while(curCol != homeCol) {

            if(curCol < homeCol) {

                curCol++;

            } else {

                curCol--;

            }

            totalCost += colCosts[curCol];

        }

        return totalCost;

    }

}

Time Complexity: O(m + n)

Space Complexity: O(1)

**Refer to**

**All shortest paths have the same cost**

<https://leetcode.com/problems/minimum-cost-homecoming-of-a-robot-in-a-grid/solutions/1598941/java-c-python-all-shortest-paths-have-the-same-cost/>

**Intuition**

It's a brain-teaser,

all shortest paths have the same cost.

**Prove**

We have 0 <= rowCosts[r], colCosts[c] <= 10^4,

that means we don't go duplicated path.

From the view of row index, the best path will be go directly from start x to home x

From the view of col index, the best path will be go directly from start y to home y

**Explanation**

Firstly move rows, from startPos[0] to homePos[0].

Secondly move cols, from startPos[1] to homePos[1].

Sum up the cost for every step.

**Complexity**

Time O(n+m)

Space O(1)

**Java**

public int minCost(int[] startPos, int[] homePos, int[] rowCosts, int[] colCosts) {

int res = 0, i = startPos[0], j = startPos[1], x = homePos[0], y = homePos[1];

while (i != x) {

i += (x - i) / Math.abs(x - i);

res += rowCosts[i];

}

while (j != y) {

j += (y - j) / Math.abs(y - j);

res += colCosts[j];

}

return res;

}

**Refer to Deepseek**

This problem requires calculating the minimum cost for a robot to return home from its starting position in a grid, moving only right or down, with specific row and column costs.

**Approach**

We'll use a greedy approach since:

The robot can only move right or down

The path is deterministic (must move toward home in both directions)

Costs are additive based on movement direction

**Solution Code**

class Solution {

public int minCost(int[] startPos, int[] homePos, int[] rowCosts, int[] colCosts) {

int totalCost = 0;

int currentRow = startPos[0];

int currentCol = startPos[1];

int homeRow = homePos[0];

int homeCol = homePos[1];

// Move vertically toward home row

while (currentRow != homeRow) {

if (currentRow < homeRow) {

currentRow++;

} else {

currentRow--;

}

totalCost += rowCosts[currentRow];

}

// Move horizontally toward home column

while (currentCol != homeCol) {

if (currentCol < homeCol) {

currentCol++;

} else {

currentCol--;

}

totalCost += colCosts[currentCol];

}

return totalCost;

}

}

**Explanation**

**Vertical Movement**:

Determine direction (up or down) based on start/home positions

Move one row at a time, accumulating row costs

Stop when reaching home row

**Horizontal Movement**:

Determine direction (left or right) based on start/home positions

Move one column at a time, accumulating column costs

Stop when reaching home column

**Cost Calculation**:

Only costs of passed rows/columns are added (start position cost excluded)

Order of movement doesn't affect total cost (can do horizontal first if preferred)

**Time and Space Complexity**

**Time Complexity**: O(m + n) where m = row distance, n = col distance

**Space Complexity**: O(1) - constant extra space used

This solution efficiently computes the minimum cost by leveraging the grid's structure and movement constraints. The greedy approach works perfectly here since there's only one optimal path given the movement restrictions.

**Refer to**

[L62.Unique Paths (Ref.L63,L64,L174,L2087,L2304,L2400)](note://E33F0B728ABE4CCF8B7E554A219CF1EA)