1217. Minimum Cost to Move Chips to The Same Position

Easy Greedy Array Math

Leetcode Link

Problem Description

In this problem, we are given `n` chips each positioned on an axis at some integer points as provided by an array `position[i]`. The goal is to move all chips to the same position on the axis. While moving chips, we need to minimize the cost. Luckily, moving a chip by 2 positions (either left or right) doesn't incur any cost (`cost = 0`). However, moving a chip by just 1 position (left or right) would cost us 1 unit (`cost = 1`).

Our objective is to determine the minimum cost required to move all the chips to a single position on the axis.

Intuition

The intuition here relies on the understanding of even and odd numbers: moving a chip between two even positions or between two odd positions has no cost; it's free. However, moving a chip from an even position to an odd position (or vice versa) comes with a cost of 1.

Given that moving chips any number of even spaces is free, all the even-positioned or all the odd-positioned chips can be considered effectively at the same point. Therefore, we should look to move all chips to whichever of these is the minority (either all to an odd or all to an even position), as this would minimize overall cost.

To get there, we can:

1. Count the number of chips in odd positions (`a`) and the number of chips in even positions (`b`).
2. Realize that to minimize cost, we want to move the smaller group to the larger group's position, since moving each chip costs 1 unit.
3. Hence, our minimum cost will be the smaller of the two counts, `min(a, b)`.

The reasoning behind the solution is simple once we realize that the even and odd movements are decoupled due to the cost structure of the problem.

Solution Approach

The implementation of the solution involves a simple algorithm that is based on counting and leverages the properties of even and odd numbers as discussed.

Here's how the solution is implemented:

1. Initialize a counter `a` for the number of chips at odd positions by using the Python `sum()` function with a generator expression. We use `p % 2` to determine if the position `p` is odd since this will return 1 for odd numbers and 0 for even numbers. `a = sum(p % 2 for p in position)` iterates over all positions and adds up the ones (which correspond to odd positions).

2. Calculate `b`, the number of chips at even positions. Since we already know the total number of chips `n` (represented by `len(position)`), the count of chips at even positions can be found by subtracting the count of odd-positioned chips from the total number. Therefore, `b = len(position) - a`.

3. To determine the minimum cost of moving all chips to the same position, we take the minimum of `a` and `b`, because it is cheaper to move the minority group. The cost incurred will be equal to the number of chips which are at the positions of minority parity (either even or odd). Thus, `return min(a, b)` gives us the desired minimum cost.

No complex data structures are needed for this solution, and there is no need for any special patterns. The entire algorithm relies on an understanding of parity and counting, with a fundamental use of arithmetic operations.

This approach complements a greedy-style strategy, which seeks to minimize each move's cost step by step, naturally leading us to the overall minimum cost for the problem.

Example Walkthrough

Let's consider a small example to illustrate the solution approach.

Say we are given `n = 5` chips with the following positions: `position = [2, 2, 2, 3, 3]`. Here are the steps to find the minimum cost to move all chips to a single position:

1. Count the number of chips in odd positions:
   - In the array `position`, the positions `3` and `3` are odd (since `3 % 2 = 1`).
   - So we have `a = 2` chips at odd positions.
2. Count the number of chips in even positions:
   - Since there are `n = 5` chips in total and `a = 2` of them are at odd positions, the remaining `b = n - a = 5 - 2 = 3` chips are at even positions.
3. Determine the minimum cost:
   - To move all chips to a single position, we need to move all chips to whichever position has more chips (either even or odd) to minimize the cost.
   - In this case, we have more chips at even positions (`b = 3` chips) than at odd positions (`a = 2` chips).
   - So, we decide to move all odd-positioned chips to the even position.
   - Since moving a chip from an odd to an even position costs `1` unit, and we have `a = 2` odd-positioned chips, the total minimum cost will be `a * 1 = 2 * 1 = 2`.

Following the implementation of the solution:

1. `a` is initialized using `sum(p % 2 for p in position)` which gives us `2`, representing the chips at odd positions.
2. `b` is calculated as `len(position) - a`, which amounts to `5 - 2 = 3`, representing the chips at even positions.
3. Since moving chips by 2 positions incurs no cost, we can move the even-positioned chips around freely. So we only need to move the chips at the odd positions to one of the even positions.
4. Thus, the minimum cost is obtained by taking the minimum of `a` and `b`, which in this case is `min(2, 3) = 2`.

Hence, the minimum cost required to move all the chips to a single position in this example is `2`.

Python Solution

```python
class Solution:
    def minCostToMoveChips(self, positions: List[int]) -> int:
        # Count the number of chips in odd positions
        odd_chip_count = sum(position % 2 for position in positions)

        # Calculate the number of chips in even positions
        even_chip_count = len(positions) - odd_chip_count

        # The minimum cost to move the chips will be the smaller of the two counts
        # because moving chips within even or odd indices is free
        # and moving a chip between even and odd index costs 1.
        # So, we choose the smaller group to move to the other.
        return min(odd_chip_count, even_chip_count)

```

Java Solution

```java
class Solution {

    // This method calculates the minimum cost to move all chips to the same position.
    public int minCostToMoveChips(int[] positions) {
        // Initialize counters for odd and even positions
        int oddCount = 0;
        int evenCount = 0;

        // Loop through each chip's position
        for (int position : positions) {
            // If the position is odd, increment the odd counter
            if (position % 2 != 0) {
                oddCount++;
            } else {
                // If the position is even, increment the even counter
                evenCount++;
            }
        }

        // The cost of moving chips is 0 between even positions, or between odd positions.
        // It only costs 1 to move from even to odd or vice versa.
        // Return the minimum of odd and even counts since we want to move all chips
        // to the position (even or odd) that has the least number of chips to minimize the cost.
        return Math.min(oddCount, evenCount);
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // For std::min

class Solution {
public:
    // Method to calculate minimum cost to move chips to the same position.
    int minCostToMoveChips(vector<int>& positions) {
        int oddCount = 0; // Counter for chips at odd positions

        // Loop through each chip position
        for (auto& p : positions) {
            // Increment oddCount if the current position is odd
            oddCount += p & 1; // Using bitwise AND to determine oddness
        }

        int evenCount = positions.size() - oddCount; // Chips at even positions

        // The cost of moving chips from even positions to odd (or vice versa) is zero,
        // so we only need to move all chips to either an odd position or an even position.
        // We choose the position type (odd or even) with fewer chips to minimize the cost.
        return std::min(oddCount, evenCount);
    }
};

```

Typescript Solution

```typescript
/**
 * This function calculates the minimum cost to move all chips to the same position.
 * Moving chips between positions with the same parity (even-to-even or odd-to-odd) is free,
 * while moving chips between positions with different parity (even-to-odd or odd-to-even) costs 1.
 * @param {number[]} positions - An array of integers representing the positions of chips.
 * @return {number} - The minimum cost required to move all chips to the same position.
 */
function minCostToMoveChips(positions: number[]): number {
    let oddCount: number = 0; // Initialize a counter for chips on odd positions.
    // Iterate over each position, incrementing oddCount for chips on odd positions.
    for (const position of positions) {
        oddCount += position % 2;
    }
    let evenCount: number = positions.length - oddCount; // Calculate the number of chips on even positions.
    // The cost is the smaller of moving all even-position chips to an odd position,
    // or moving all odd-position chips to an even position.
    return Math.min(oddCount, evenCount);
}

```

Time and Space Complexity

Time Complexity

The time complexity of the given code is determined by the iteration over the `position` array to calculate the count of chips at even and odd positions, which is `a` and `b` respectively. This iteration is a single pass through all elements of the array, therefore the time complexity is `O(n)`, where `n` is the number of elements in the `position` array.

Space Complexity

The space complexity of this algorithm is `O(1)`. This is because the extra space required by the algorithm does not increase with the size of the input array. The only additional space used is for the variables `a` and `b`, which are used to store the counts of chips at even and odd positions, respectively, regardless of the input size.