1749. Maximum Absolute Sum of Any Subarray

You are given an integer array `nums`. The **absolute sum** of a subarray `[numsl, numsl+1, ..., numsr-1, numsr]` is `abs(numsl + numsl+1 + ... + numsr-1 + numsr)`.

Return *the* ***maximum*** *absolute sum of any* ***(possibly empty)*** *subarray of* `nums`.

Note that `abs(x)` is defined as follows:

- If `x` is a negative integer, then `abs(x) = -x`.
- If `x` is a non-negative integer, then `abs(x) = x`.

Example 1:

```
Input: nums = [1,-3,2,3,-4]
Output: 5
Explanation: The subarray [2,3] has absolute sum = abs(2+3) = abs(5) = 5.
```

Example 2:

```
Input: nums = [2,-5,1,-4,3,-2]
Output: 8
Explanation: The subarray [-5,1,-4] has absolute sum = abs(-5+1-4) = abs(-8) = 8.
```

Constraints:

- $1 \leq nums.length \leq 10^5$
- $-10^4 \leq nums[i] \leq 10^4$

1749. Maximum Absolute Sum of Any Subarray

Overview

We need to find the maximum absolute sum of any subarray within the given integer array `nums`. A **subarray** is a contiguous segment of the array, and the **absolute sum** of a subarray is simply the absolute value of the sum of its elements.

Formally, for a subarray `[nums[l], nums[l + 1], ..., nums[r]]`, its absolute sum is:

$$|\sum_{i=l}^{r} nums[i]|$$

We need to find the subarray whose sum, when taken in absolute value, is the highest among all possible subarrays (including the possibility of choosing an empty subarray, which has a sum of `0`).

Mathematically, we are looking for:

$$max\left(max_{l \le r}|\sum_{i=l}^{r} nums[i]|\right)$$

where `l` and `r` define the boundaries of a valid subarray.

A common pitfall in this problem is overlooking that both a subarray with a large positive sum and a subarray with a large negative sum contribute to the answer, as we take the absolute value.

Intuition

A brute-force approach to solving this problem involves considering all possible subarrays of the given array and comparing their sums to find the one with the maximum absolute value. While this brute-force approach works, it requires repeatedly summing subarrays, making it computationally expensive. Instead of recalculating sums every time, we can optimize the process using prefix sums, a common technique for handling subarray problems efficiently.

The idea behind prefix sums is that if we precompute cumulative sums up to each index, we can quickly determine the sum of any subarray. The prefix sum at index i represents the total sum of elements from the beginning of the array up to i. This allows us to determine the sum of any subarray between indices l and r by taking the difference prefix_sum[r+1] - prefix_sum[l], eliminating the need for repeated summation.

1749. Maximum Absolute Sum of Any Subarray

```
/*
   Prefix sum,prefix max positive,prefix min negative
   Time complexity: O(n)
   Space complexity: o(1)
*/
class Solution {
   public:
      int maxAbsoluteSum(std::vector<int>& nums) {
         int prefix_sum=0,max_pos_sum=0, min_neg_sum=0,ans=0;
         for(auto& e: nums){
            max_pos_sum=std::max(max_pos_sum,prefix_sum);
            min_neg_sum=std::min(min_neg_sum,prefix_sum);

            prefix_sum+=e;

            ans=(prefix_sum>=0)
               ?std::max(ans,prefix_sum-min_neg_sum)
               :std::max(ans,max_pos_sum-prefix_sum);
         }

         return ans;
      }
};
```