1975. Maximum Matrix Sum

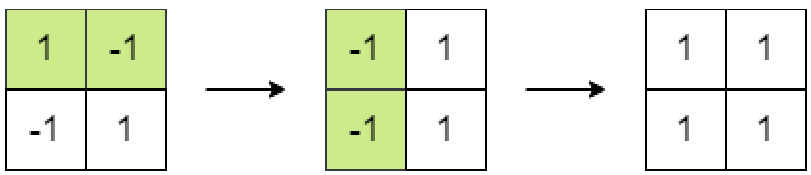
You are given an `n x n` integer `matrix`. You can do the following operation **any** number of times:

- Choose any two **adjacent** elements of `matrix` and **multiply** each of them by `-1`.

Two elements are considered **adjacent** if and only if they share a **border**.

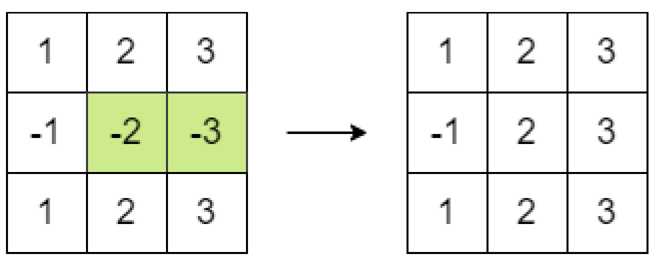
Your goal is to **maximize** the summation of the matrix's elements. Return *the **maximum** sum of the matrix's elements using the operation mentioned above.*

Example 1:



```
Input: matrix = [[1,-1],[-1,1]]
Output: 4
Explanation: We can follow the following steps to reach sum equals 4:
- Multiply the 2 elements in the first row by -1.
- Multiply the 2 elements in the first column by -1.
```

Example 2:



```
Input: matrix = [[1,2,3],[-1,-2,-3],[1,2,3]]
Output: 16
Explanation: We can follow the following step to reach sum equals 16:
- Multiply the 2 last elements in the second row by -1.
```

Constraints:

- `n == matrix.length == matrix[i].length`
- `2 <= n <= 250`
- -10^5 `<= matrix[i][j] <=` 10^5

Intuition

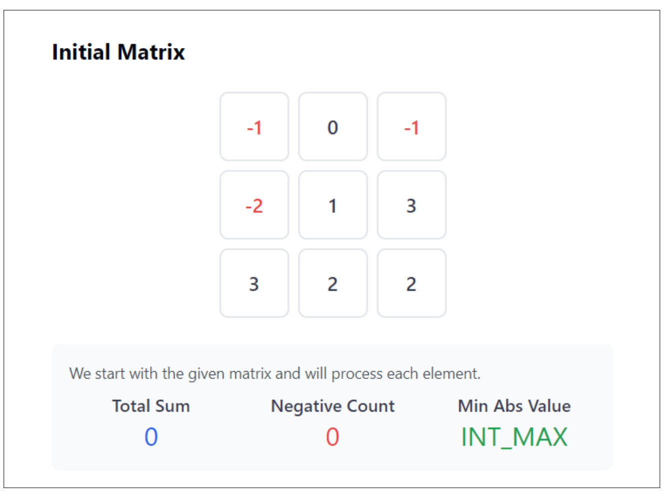
To maximize the matrix sum, let's first imagine the ideal situation: if every element in the matrix were positive, we would have the highest possible sum. Since we can flip pairs of adjacent elements by multiplying them by -1, we could, in theory, make all values positive if we wanted. So, we start by calculating the sum of the absolute values of all elements, as this would be the ideal maximum sum if all elements were positive.

Next, we need to think about when flipping doesn't work perfectly. Specifically, if there's an odd number of negative elements, it won't be possible to make everything positive because one negative will always remain. This observation leads us to a simple rule: i**f there's an even count of negative numbers, we can flip them all to positive values. But if the count is odd, one number has to stay negative, which means the sum can't be quite as high as in the ideal case.**

To minimize the impact of this remaining negative, we want it to be the smallest number in the matrix. So, while calculating the absolute sum, we also track the smallest absolute value. This way, if we end up with an odd count of negatives, we can subtract twice this smallest value from the total. This subtraction accounts for the one unavoidable negative element and keeps the final sum as high as possible.

Why subtract twice the smallest absolute value?

For an odd count of negative numbers, flipping a negative number to positive adds that number's absolute value to the total sum. For example, if we had flipped -1 to $+1$, it would increase the sum by $+1$. However, since we can't flip this number (due to the odd count of negatives), we need to "remove" this potential gain. This is why we subtract twice the smallest absolute value: once to account for the gain we didn't get and again because we didn't flip it.



Processing (0)

-1	0	-1
-2	1	3
3	2	2

Add |0| to sum, update minAbs to 0

Total Sum	Negative Count	Min Abs Value
1	1	0

Processing (-1)

-1	0	-1
-2	1	3
3	2	2

Add |−1| to sum, increment negative count

Total Sum	Negative Count	Min Abs Value
2	2	0

Processing (-2)

-1	0	-1
-2	1	3
3	2	2

Add |−2| to sum, increment negative count

Total Sum	Negative Count	Min Abs Value
4	3	0

Processing (1)

-1	0	-1
-2	1	3
3	2	2

Add |1| to sum

Total Sum	Negative Count	Min Abs Value
5	3	0

Processing (3)

-1	0	-1
-2	1	3
3	2	2

Add |3| to sum

Total Sum	Negative Count	Min Abs Value
8	3	0

Processing (3)

-1	0	-1
-2	1	3
3	2	2

Add |3| to sum

Total Sum	Negative Count	Min Abs Value
11	3	0

Processing (2)

-1	0	-1
-2	1	3
3	**2**	2

Add |2| to sum

Total Sum	Negative Count	Min Abs Value
13	3	0

Processing (2)

-1	0	-1
-2	1	3
3	2	**2**

Add |2| to sum

Total Sum	Negative Count	Min Abs Value
15	3	0

Final Result

-1	0	-1
-2	1	3
3	2	2

Since negative count (3) is odd, subtract 2 * minAbs (0) = 0. Final sum remains 15

Total Sum	Negative Count	Min Abs Value
15	3	0

1975. Maximum Matrix Sum

```
/*
   Pattern recognition
   Time compelxity: O(n^2)
   Space complexity: O(1)
*/
class Solution {
public:
   long long maxMatrixSum(std::vector<std::vector<int>>& matrix){
      int n=matrix.size();

      long long ans=0;
      int count_neg=0; // Count negative numbers
      int min_abs_val=INT_MAX; // Determine the min absolute value
      for(int i=0;i<n;++i){
         for(int j=0;j<n;++j){
            int e=matrix[i][j];
            count_neg+=int(e<0);
            ans+=std::abs(e);
            min_abs_val=std::min(min_abs_val,abs(e));
         }
      }
      return count_neg%2?ans-2*min_abs_val:ans;
   }
};
```