

# 累積和 偶奇



<https://leetcode.com/problems/number-of-sub-arrays-with-odd-sum> の Prefix Sum with Odd-Even Counting の方法がよく分かりませんので、解説してください

LeetCode の「[Number of Sub-arrays With Odd Sum](#)」に対する **Prefix Sum with Odd-Even Counting** のアプローチを解説します。

## 問題概要

与えられた整数配列 `arr` について、部分配列の要素の合計が奇数になるものの個数を求める。

### 制約

- `1 <= arr.length <= 10^5`
- `1 <= arr[i] <= 100`

→  $O(N^2)$  の解法では間に合わないので、 $O(N)$  で解く必要がある

## Prefix Sum with Odd-Even Counting の考え方

### ポイント

1. 累積和 (prefix sum) を利用する
2. prefix sum の偶奇を管理してカウント
3. prefix sum の性質を利用して部分配列の奇数合計を数える

### Prefix Sum の偶奇による考え方

- `prefix[i]` を `arr[0]` から `arr[i]` までの累積和とする。
- 部分配列 `arr[l] ... arr[r]` の合計 `sum(l, r)` は

$$sum(l, r) = prefix[r] - prefix[l - 1]$$

- この値が奇数になるための条件:
  - `prefix[r] - prefix[l-1]` が奇数  $\Leftrightarrow$  `prefix[r]` と `prefix[l-1]` の偶奇が異なる
  - つまり、`prefix[r]` が偶数のとき `prefix[l-1]` が奇数 であるような `(l-1, r)` の組み合わせ数を数える

### 前処理で数え上げる

- `odd\_count` : `prefix[i]` が 奇数 の個数
- `even\_count` : `prefix[i]` が 偶数 の個数

最初に `prefix[0] = 0` (偶数) と考え、`even\_count = 1` から始める。

各 `arr[i]` を走査しながら、累積和 `prefix` を更新し、

- `prefix` が偶数なら `even\_count` を増やす
- `prefix` が奇数なら `odd\_count` を増やす

この時点で、現在の `prefix` の奇偶に応じて、部分配列の奇数合計をカウント する。

## コード実装 (Python)

```
python

def numOfSubarrays(arr):
    MOD = 10**9 + 7
    odd_count = 0
    even_count = 1 # prefix sum = 0 initially, which is even
    prefix_sum = 0
    result = 0

    for num in arr:
        prefix_sum += num
        if prefix_sum % 2 == 0:
            result += odd_count # 偶数 - 奇数 = 奇数
            even_count += 1
        else:
            result += even_count # 奇数 - 偶数 = 奇数
            odd_count += 1

    result %= MOD # mod を取る

    return result
```

## コードの流れ

1. `prefix\_sum` を 0 から始める。
2. `even\_count = 1` と初期化。（最初の `prefix = 0` を考慮）
3. 配列を順番に走査：
  - `prefix\_sum` を更新
  - 偶数なら `odd\_count` を加算 (`even - odd = odd` のペアを数える)
  - 奇数なら `even\_count` を加算 (`odd - even = odd` のペアを数える)
  - `prefix\_sum` の偶奇に応じて `even\_count` / `odd\_count` を増やす
4. 結果を `10^9 + 7` で割って返す。

## 時間計算量・空間計算量

- 時間計算量：O(N) (1 回のループで済む)
- 空間計算量：O(1) (定数の変数しか使用しない)

## なぜこの方法が高速なのか？

通常の部分配列の総和を求める方法は  $O(N^2)$  ですが、このアプローチでは：

- **prefix sum の偶奇のみを管理**
- ループ 1 回で `even_count` / `odd_count`` を数えて効率的にペアを数える

これにより、 $O(N)$  で問題を解くことができる。

## まとめ

1. **prefix sum の偶奇** に着目する
2. 累積的に `even_count` / `odd_count`` を数える
3. ペアの性質 (`even - odd = odd` / `odd - even = odd``) を利用
4.  $O(N)$  の計算量で解決 できる

このテクニックは「**prefix sum の偶奇管理**」として、他の類似問題（部分配列の偶奇を問う問題）にも応用できます。



A network error occurred. Please check your connection and try again. If this issue persists please contact us through our help center at [help.openai.com](https://help.openai.com).



うーん、よくわかりません。

`sum(l,r)=prefix[r]-prefix[l-1]` までは分かりますが、`prefix[r] - prefix[l-1]` が奇数  $\leftrightarrow$  `prefix[r]` と `prefix[l-1]` の偶奇が異なる から分からなくなります。

[1,2,3,4,5] を使って示してもらえますか？

## Prefix Sum の偶奇が異なると奇数になる理由

まず、`sum(l, r) = prefix[r] - prefix[l-1]` について、`sum(l, r)` が奇数になる条件を考えます。

- 偶数 - 偶数 = 偶数
- 奇数 - 奇数 = 偶数
- 偶数 - 奇数 = 奇数
- 奇数 - 偶数 = 奇数

つまり、`sum(l, r)` が 奇数 になるのは `prefix[r]` と `prefix[l-1]` の偶奇が異なる ときです。

## 配列 `[1,2,3,4,5]` で考える

### Prefix Sum を計算

<code>i`</code>	<code>arr[i]`</code>	<code>prefix[i]`</code>	偶奇
0	-	0	偶数

`i`	`arr[i]`	`prefix[i]`	偶奇
1	1	1	奇数
2	2	3	奇数
3	3	6	偶数
4	4	10	偶数
5	5	15	奇数

(※ `prefix[0] = 0` は累積和の初期値として仮定)

## 部分配列の例

### 1. `(l, r) = (1, 1)` の場合 (`[1]`)

- $\text{sum}(1, 1) = \text{prefix}[1] - \text{prefix}[0] = 1 - 0 = 1$  (奇数)
- `prefix[1]` (奇数) と `prefix[0]` (偶数) は異なる → **奇数になる!**

### 2. `(l, r) = (1, 2)` の場合 (`[1,2]`)

- $\text{sum}(1, 2) = \text{prefix}[2] - \text{prefix}[0] = 3 - 0 = 3$  (奇数)
- `prefix[2]` (奇数) と `prefix[0]` (偶数) は異なる → **奇数になる!**

### 3. `(l, r) = (1, 3)` の場合 (`[1,2,3]`)

- $\text{sum}(1, 3) = \text{prefix}[3] - \text{prefix}[0] = 6 - 0 = 6$  (偶数)
- `prefix[3]` (偶数) と `prefix[0]` (偶数) は同じ → **偶数になる**

### 4. `(l, r) = (2, 3)` の場合 (`[2,3]`)

- $\text{sum}(2, 3) = \text{prefix}[3] - \text{prefix}[1] = 6 - 1 = 5$  (奇数)
- `prefix[3]` (偶数) と `prefix[1]` (奇数) は異なる → **奇数になる!**

### 5. `(l, r) = (2, 4)` の場合 (`[2,3,4]`)

- $\text{sum}(2, 4) = \text{prefix}[4] - \text{prefix}[1] = 10 - 1 = 9$  (奇数)
- `prefix[4]` (偶数) と `prefix[1]` (奇数) は異なる → **奇数になる!**

## Prefix Sum の偶奇をカウント

1. 現在の `prefix\_sum` が偶数なら、奇数の `prefix` の個数 (`odd\_count`) だけ  $\text{sum}(l, r)$  は奇数になる。
2. 現在の `prefix\_sum` が奇数なら、偶数の `prefix` の個数 (`even\_count`) だけ  $\text{sum}(l, r)$  は奇数になる。

## 計算の流れ

配列 `[1,2,3,4,5]` をスキャンしながら以下を管理：

- ``odd_count``: ``prefix[i]`` が **奇数** の数
- ``even_count``: ``prefix[i]`` が **偶数** の数 (初期状態で ``prefix[0] = 0`` なので ``even_count = 1``)

<code>`i`</code>	<code>`arr[i]`</code>	<code>`prefix_sum`</code>	偶奇	<code>`even_count`</code>	<code>`odd_count`</code>	<code>`result`</code>
0	-	0	偶数	1	0	0
1	1	1	奇数	1	1	<code>`1`</code> ( <code>`even_count`</code> を加算)
2	2	3	奇数	1	2	<code>`1+1=2`</code> ( <code>`even_count`</code> を加算)
3	3	6	偶数	2	2	<code>`2+2=4`</code> ( <code>`odd_count`</code> を加算)
4	4	10	偶数	3	2	<code>`4+2=6`</code> ( <code>`odd_count`</code> を加算)
5	5	15	奇数	3	3	<code>`6+3=9`</code> ( <code>`even_count`</code> を加算)

最終的に、**9** 個の部分配列が奇数の合計を持つ。

## 結論

``prefix_sum[r] - prefix_sum[l-1]`` が奇数になるのは ``prefix[r]`` と ``prefix[l-1]`` の偶奇が異なるとき！  
この性質を利用し、累積和の偶奇カウントを保持しながら ``O(N)`` で求めることができる。



``prefix_sum[r] - prefix_sum[l-1]`` が奇数になるのは `prefix[r]` と `prefix[l-1]` の偶奇が異なるとき！` は理解できたのですが、

``Prefix Sum`` の偶奇をカウント

現在の `prefix_sum`` が偶数なら、奇数の `prefix`` の個数 (`odd_count``) だけ `sum(l, r)`` は奇数になる。  
現在の `prefix_sum`` が奇数なら、偶数の `prefix`` の個数 (`even_count``) だけ `sum(l, r)`` は奇数になる。`

これがよく分かりません。

理解が進んできたようですね！では、もう少し噛み砕いて説明します。

## 考え方の確認

「部分配列 `sum(l, r) = prefix[r] - prefix[l-1]`` が奇数になるのは、``prefix[r]`` と ``prefix[l-1]`` の偶奇が異なるとき」まで理解できていますね。

このことを使って、**奇数になる部分配列の数を数える方法** を考えます。

## なぜ `prefix\_sum` の偶奇をカウントすればいいのか？

現在の `prefix\_sum[r]` を計算したとき、その `prefix\_sum[r]` の偶奇が分かれば、それまでに登場した `prefix\_sum[l-1]` の個数を使って、奇数になる部分配列の数を求めることができます。

### 偶数の `prefix\_sum[r]` が登場した場合

- `prefix[r]` が偶数の場合、`prefix[l-1]` が **奇数** であれば  $\text{sum}(l, r) = \text{prefix}[r] - \text{prefix}[l-1]$  は奇数になる。
- したがって、**奇数の** `prefix[l-1]` (すなわち `odd\_count`) の数だけ `sum(l, r)` が奇数になる部分配列がある。

### 奇数の `prefix\_sum[r]` が登場した場合

- `prefix[r]` が奇数の場合、`prefix[l-1]` が **偶数** であれば  $\text{sum}(l, r) = \text{prefix}[r] - \text{prefix}[l-1]$  は奇数になる。
- したがって、**偶数の** `prefix[l-1]` (すなわち `even\_count`) の数だけ `sum(l, r)` が奇数になる部分配列がある。

## 具体例で説明

配列 `[1, 2, 3, 4, 5]` について、以下の表を作ってみましょう。

`i`	`arr[i]`	`prefix_sum`	偶奇	`even_count`	`odd_count`	奇数部分配列の個数 (`result`)
0	-	0	偶数	1	0	0
1	1	1	奇数	1	1	`1` (even_count を加算)
2	2	3	奇数	1	2	`1+1=2` (even_count を加算)
3	3	6	偶数	2	2	`2+2=4` (odd_count を加算)
4	4	10	偶数	3	2	`4+2=6` (odd_count を加算)
5	5	15	奇数	3	3	`6+3=9` (even_count を加算)

### 各ステップの動作

1. `prefix\_sum = 0` (**偶数**) で始めるので、`even\_count = 1` で初期化。
2. `arr[1] = 1` を加えて `prefix\_sum = 1` (**奇数**) :
  - `even\_count = 1` なので、**1 個の奇数部分配列** ([1]) が作れる → `result = 1`。
  - `odd\_count` を 1 増やす。
3. `arr[2] = 2` を加えて `prefix\_sum = 3` (**奇数**) :
  - `even\_count = 1` なので、**1 個の奇数部分配列** ([2,3]) が作れる → `result = 2`。
  - `odd\_count` を 2 に増やす。

4. `arr[3] = 3` を加えて `prefix\_sum = 6` (偶数) :
  - `odd\_count = 2` なので、**2 個の奇数部分配列** ([3] と [1,2,3]) が作れる → `result = 4`。
  - `even\_count` を 2 に増やす。
5. `arr[4] = 4` を加えて `prefix\_sum = 10` (偶数) :
  - `odd\_count = 2` なので、**2 個の奇数部分配列** ([4,5] と [2,3,4,5]) が作れる → `result = 6`。
  - `even\_count` を 3 に増やす。
6. `arr[5] = 5` を加えて `prefix\_sum = 15` (奇数) :
  - `even\_count = 3` なので、**3 個の奇数部分配列** ([5], [3,4,5], [1,2,3,4,5]) が作れる → `result = 9`。
  - `odd\_count` を 3 に増やす。

## なぜこの方法が効率的なのか？

通常、すべての `(l, r)` を走査するには  $O(N^2)$  かかるが、この方法では `prefix\_sum` の偶奇をカウントするだけで  $O(N)$  で解ける。

- `prefix\_sum` の偶奇だけを管理すればいいので、`even\_count` / `odd\_count` の 2 つの変数で済む (定数空間  $O(1)$ ) 。
- 1 回のループ ( $O(N)$ ) で奇数部分配列の数をカウントできる。

## まとめ

### 重要ポイント

- `prefix[r] - prefix[l-1]` が奇数になるのは、`prefix[r]` と `prefix[l-1]` の偶奇が異なるとき。
- `prefix[r]` の偶奇を見て、すでに出現した `prefix[l-1]` の偶奇の個数 (`even\_count` / `odd\_count`) を使って一気にカウントできる。
- `prefix\_sum` が偶数なら `odd\_count` を加算、奇数なら `even\_count` を加算することで、 $O(N)$  で計算できる。

この方法を理解すれば、部分配列の偶奇を問う他の問題にも応用できるので、非常に便利なテクニックです！



```
`class Solution:
    def numOfSubarrays(self, arr: List[int]) -> int:
        mod = 10 ** 9 + 7
        odd_count = 0

        # An empty subarray [] total is 0. That is even.
        # That's why we set 1.
        even_count = 1

        prefix_sum = 0
        result = 0
        n = len(arr)

        for r in range(n):
            prefix_sum += arr[r]
```

```

        if prefix_sum % 2 == 0:
            # There are many subarrays, which consists "odd-subarray" and "even-subarray"
            # Since prefix[r] is even,
            # we consider only the "odd_count" which represents the number of "odd_subarray" so
            #
            # Prefix[r] - Prefix[l - 1] (= total of the [l, R]), which the "L" takes "0...r".
            result += (odd_count + even_count) - even_count # = odd_count
            even_count += 1
        else:
            result += (odd_count + even_count) - odd_count # even_count
            odd_count += 1

    result %= mod
    return result`

```

こういうことですかね。（英語が得意ではないですが）

はい、ほぼ合っています！コードのコメントの意図を補足しながら確認しますね。

## コードのポイント

### 1. `even\_count = 1` の理由

```

python

even_count = 1 # 空の部分配列 [] の和は 0 なので偶数

```

- 初期状態では `prefix\_sum = 0`（偶数）。
- したがって、累積和が **偶数の prefix を 1 個持っている** と考える。

### 2. `prefix\_sum` を更新

```

python

prefix_sum += arr[r]

```

- `arr[r]` を足して累積和を更新。

### 3. 偶数の `prefix\_sum` が出た場合

```

python

if prefix_sum % 2 == 0:
    result += (odd_count + even_count) - even_count # = odd_count
    even_count += 1

```

- `prefix\_sum` が偶数のとき、`sum(l, r) = prefix[r] - prefix[l-1]` が奇数になるのは `prefix[l-1]` が奇数のとき。
- `odd\_count` の分だけ奇数の部分配列があるので、それを加算する。



## 4. 奇数の `prefix\_sum` が出た場合

```
python

else:
    result += (odd_count + even_count) - odd_count # = even_count
    odd_count += 1
```

- `prefix\_sum` が奇数のとき、`sum(l, r) = prefix[r] - prefix[l-1]` が奇数になるのは `prefix[l-1]` が偶数のとき。
- `even\_count` の分だけ奇数の部分配列があるので、それを加算する。

## コードの改良版

より直感的に理解できるように書き直すと、次のようになります：

```
python

from typing import List

class Solution:
    def numOfSubarrays(self, arr: List[int]) -> int:
        MOD = 10**9 + 7
        odd_count = 0
        even_count = 1 # prefix_sum = 0 は偶数と考える
        prefix_sum = 0
        result = 0

        for num in arr:
            prefix_sum += num

            if prefix_sum % 2 == 0:
                result += odd_count # 奇数の prefix を持つ部分配列の数を加算
                even_count += 1 # 偶数 prefix のカウントを更新
            else:
                result += even_count # 偶数の prefix を持つ部分配列の数を加算
                odd_count += 1 # 奇数 prefix のカウントを更新

        result %= MOD # mod 10^9+7 を適用

        return result
```

- `result += (odd\_count + even\_count) - even\_count` → `result += odd\_count`
- `result += (odd\_count + even\_count) - odd\_count` → `result += even\_count`
- このように単純化できる！

## このコードの流れ

配列 `[1,2,3,4,5]` で動作を確認します。

`i`	`arr[i]`	`prefix_sum`	偶奇	`even_count`	`odd_count`	`result` (奇数部分配列の個数)
0	-	0	偶数	1	0	0

`i`	`arr[i]`	`prefix_sum`	偶奇	`even_count`	`odd_count`	`result` (奇数部分配列の個数)
1	1	1	奇数	1	1	`1` (even_count を加算)
2	2	3	奇数	1	2	`1+1=2` (even_count を加算)
3	3	6	偶数	2	2	`2+2=4` (odd_count を加算)
4	4	10	偶数	3	2	`4+2=6` (odd_count を加算)
5	5	15	奇数	3	3	`6+3=9` (even_count を加算)

## 結論

あなたのコードは正しく動作しますし、考え方も合っています！

ただし、``result += (odd_count + even_count) - even_count`` という表現は ``result += odd_count`` に簡略化できるので、それを使うとコードがより明確になります。

このアプローチを理解すれば、**累積和を使った偶奇判定の問題** に応用できるので、ぜひ使ってみてください！🚀



理解できました。ありがとうございます。

理解できたようで良かったです！😊🎉

この**累積和の偶奇カウント**の考え方は、部分配列の偶奇判定だけでなく、他の「条件付き部分配列カウント」の問題にも応用できます。たとえば：

- 「奇数の個数が  $k$  個以下の部分配列を数える」
- 「XOR の累積和を使って特定の条件を満たす部分配列を数える」 など、色々なパターンで活用できるので、ぜひ試してみてください！

また疑問があれば、いつでも聞いてくださいね！😊🚀

ChatGPT can make mistakes. Check important info.