# 前缀和与差分

杨子凡

Dec 9, 2024

## 1 引入

画龙点睛，事半功倍。

差分算法的思想就是通过巧妙的修改差值而不是直接修改原数组，从而优化时间复杂度。利用差分，我们可以在常数时间复杂度内对一个区间进行修改，而不需要遍历整个区间。结合前缀和进行求和，我们可以在 $\mathcal{O}(1)$ 的时间内完成区间修改操作。算法中的这些优化技巧在实际问题中非常高效，尤其是在面对大数据量时。
本篇将介绍如何运用差分优化区间修改操作，并利用前缀和求解区间和。

## 2 差分的思想

差分是前缀和的逆过程，用于高效处理区间修改。给定一个数组 $a$，我们可以通过构造一个差分数组 $d$ 来记录数组相邻元素的差值。
假设 $d_i$ 记录的是数组 $a_i$ 与 $a_{i-1}$ 之间的差异，即：

$$d_i = a_i - a_{i-1}$$

利用差分数组，我们可以在常数时间内对区间进行修改。

## 3 区间修改

给定一个区间 $[l, r]$，我们需要将区间内的每个元素都加上一个常数 $x$。直接修改原数组会导致 $\mathcal{O}(r - l + 1)$ 的时间复杂度，而使用差分数组，我们可以通过如下操作：

- 对差分数组 $d_l$ 加上 $x$（表示区间开始处增加 $x$）。
- 对差分数组 $d_{r+1}$ 减去 $x$（表示区间结束后增加的部分被取消）。

这样，区间修改的时间复杂度就变为 $\mathcal{O}(1)$。

## 4 算法实现

还是来看一道例题，是洛谷的 P2367

```
#include <bits/stdc++.h>
#define F(_b, _e) for (int i = _b; i <= _e; i ++)
using namespace std;

const int MAXN = 5e6 + 5;
int a[MAXN];
int main() {
   int n, p; n = read(); p = read();
   int tmp = 0;
   F (1, n) {
      int k = read();
      a[i] = k - tmp;
      tmp = k;
   }
   F (1, p) {
      int l, r, d;
      l = read(); r = read(); d = read();
      a[l] += d;
      a[r + 1] -= d;
   }
   tmp = 0;
   int ans = 5e5;
   F (1, n) {
      tmp += a[i];
      ans = min(ans, tmp);
   }
   write(ans);
   return 0;
}
```

目前你已经掌握了前缀和的基本方法以及通过差分加前缀和的方法优化区间修改的时间复杂度。

但是，这样对于区间修改的时间复杂度的优化会导致查询的时间复杂度急剧上升，没差一个都要从第一项求前缀和到该项，那么，有没有什么方法可以使得修改和查询时间复杂度都降低呢？这将会是我们前缀和这一期第三篇的内容，下一篇将会有很大难度，各位做好心理准备。