信息学奥赛笔记25

算法前置知识——求和杀手——前缀和

学前预备

在了解什么是前缀和前，先来看这样的一道题：

题目描述

现在有一个长度为n的数组a，现有m次查询，对于每次查询给定一个区间$[l_i, r_i]$，输出数组中下标区间$[l_i, r_i]$的和。

输入格式

第一行两个整数 n和m，分别表示数组的长度和需要查询的次数。

第二行n个整数，表示数组a。

接下来m行，每行两个整数l_i，r_i，表示查询区间。

输出格式

共m行，每行一个整数，表示查询的答案。

样例 #1

样例输入 #1

```
3 2
1 3 5
2 3
1 3
```

样例输出 #1

```
8
9
```

根据这一道题，我们可以很简单的写出一个暴力的代码。

暴力搜索(30分)

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, m, l, r, a[200001];
int main() {
    cin >> n >> m;
```

```
    for (int i = 1; i <= n; i++) {
        cin >> a[i]; // 数组读入
    }
    while (m--) {
        cin >> l >> r; // 输入区间
        if (l > r) swap(l, r);
        long long sum = 0;  // 累加器
        for (int j = l; j <= r; j++) {  // 对区间 [l, r] 求和
            sum += a[j];
        }
        cout << sum << endl;
    }
    return 0;
}
```

进度到算法部分，同学们一定要学会去计算**时间复杂度**。

时间复杂度

这是用来衡量代码的运行效率的一个名词。我们一般把一个程序所需要执行的语句数量写成一个多项式。

比如说这样的一个代码

```
for (int i = 0; i < n; i++) {
    cin >> a[i];
}
```

最坏的情况下需要执行`n`遍。所以我们定时间复杂度为$\mathrm{O}(n)$

再比如这样的一个代码

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        cout << j << " ";
    }
    cout << endl;
}
```

这个代码一共需要循环$1+2+\ldots+n=n*(n+1)/2=\frac{n^2+n}{2}$遍。

所以，我们能不能说时间复杂度就是$\mathrm{O}(\frac{n^2+n}{2})$呢?

这是**不行**的

时间复杂度是最坏情况下程序循环的**最高次方**，**省略系数**后的多项式结果。

在刚刚的式子当中，最高次方的结果为n^2。系数为$1/2$，所以时间复杂度就是$\mathrm{O}(n^2)$。

回到刚刚的题目

这道题的输入部分需要循环n遍，求和部分需要执行最坏情况下$m*n$遍，也就是当输入的结果全部是求`[1..n]`的和时。

所以时间复杂度为$\mathrm{O}(n+m*n)=\mathrm{O}((m+1)*n)\approx\mathrm{O}(m*n)$。

那么在最坏情况下，程序给出的$m\leq10^5$，$n\leq2\times10^5$。已经远超出了程序$1s$能够执行的时长。

但是我们已经按照题目的意思去把代码模拟出来了呀！！为什么这样做拿不到满分呢?

这就是算法

对于一道题目，我们不能再停留在仅仅只是用代码将程序写出来，把题意模拟出来的程度，而是我们要深入的去思考一个问题，不仅仅做出来，而是要用最高效的，循环次数最少得，使用变量最少的结果将它写出来，最终拿到满分。这才是学习信息学奥赛的目的，不断的优化，不断地创新。

前缀和(Prefix-Sum 算法)

假设我们现在有一个数组a，数组中的元素是$[a_1,a_2,\ldots,a_n]$

现在我们新创建另外一个数组b，使得$b_i=\sum_{j=1}^{i}a_j$

这个数学公式的含义拆开就是表示：$b_i=a_1+a_2+\ldots+a_i$。

现在我们把b数组写出来。

$$
\begin{aligned}
b_0&=0\\
b_1&=a_1\\
b_2&=a_1+a_2\\
b_3&=a_1+a_2+a_3\\
b_4&=a_1+a_2+a_3+a_4\\
b_5&=a_1+a_2+a_3+a_4+a_5\\
b_6&=a_1+a_2+a_3+a_4+a_5+a_6
\end{aligned}
$$

现在如果想求a数组中`[1, 4]`的和，答案就是$b_4-b_0=a_1+a_2+a_3+a_4-0$

如果想求`[2, 3]`的和，答案就是$b_3-b_1=a_1+a_2+a_3-(a1)=a_2+a_3$

如果想求`[3, 6]`的和，答案就是
$b_6-b_2=a_1+a_2+a_3+a_4+a_5+a_6-(a_1+a_2)=a_3+a_4+a_5+a_6$

如果想求`[l, r]`的和，答案就是
$b_r-b_{l-1}=a_1+a_2+\ldots+a_r-(a_1+a_2+\ldots+a_{l-1})=a_l+a_{l+1}+\ldots+a_r$。

数学证明

已知数组a的元素为$[a_1, a_2, \ldots, a_n]$，一个数组b使得$b_i = \sum_{j=1}^{i} a_j$

所以对于a数组`[l, r]`的和为$\sum_{i=l}^{r} a_i = \sum_{i=1}^{r} a_i - \sum_{i=1}^{l-1} a_i \quad (l <= r)$

即$b_r - b_{l-1}$。

代码思路

所以我们可以用一个**前缀和数组**，预处理出这个数组，当我们需要用到原数组的某一个区间的和时，不再访问原数组的元素，而是直接访问前缀和数组。

时间复杂度就从$O(n)$

```
cin >> l >> r;
long long sum = 0;
for (int i = l; i <= r; i++) {
    sum += a[i];
}
cout << sum << endl;
```

降低到了复杂度为$O(1)$

```
cin >> l >> r;
cout << sum[r] - sum[l - 1] << endl;
```

总时间复杂度就从$O(m * n + n)$ 降低到了$O(m + n)$。一个数量级的减少。

预处理前缀和数组

既然咱们要使用前缀和数组，就需要知道怎么预处理它，一般情况，我们有两种代码习惯都是可以预处理前缀和数组的，一个是变读入边处理，一个是单独遍历预处理。两种方法的核心思路都是一样的。

边读入边处理

```
int a[maxn], sum[maxn];
for (int i = 0; i < n; i++) { // 0..n - 1
    cin >> a[i];
    sum[i + 1] = sum[i] + a[i];
}
```

```
int a[maxn], sum[maxn];
for (int i = 1; i <= n; i++) {// 1..n
    cin >> a[i];
    sum[i] = sum[i - 1] + a[i];
}
```

这个代码虽然简单，但是其中需要各位同学注意两点细节

- 对于不同的代码习惯，`[1..n]`式循环和`[0..n-1]`式循环有不同的写法
- `sum[i] = sum[i - 1] + a[i]`。

第二点的公式中，我们可以通过刚刚结论得知，想要求当前这一项的前缀和，其实可以直接利用上一项的结果来进行操作。为什么呢?

因为$sum[i] = a_1 + a_2 + \ldots + a_i$，$sum[i-1] = a_1 + a_2 + \ldots + a_{i-1}$。

所以$sum[i] = sum[i-1] + a_i$。

但是最终我们的目的都是需要**使得前缀和数组的下标对齐到从`1`开始**。原因是什么?

因为求前缀和公式是$sum[r] - sum[l-1]$。

这个$l-1$必须要求从1开始，否则很容易导致数组下标的越界。所以各位同学在写前缀和的时候一定要注意！！前缀和数组需要从1开始存储。

单独遍历预处理

```
int a[maxn], sum[maxn];
for (int i = 0; i < n; i++) { // 0..n - 1
    cin >> a[i];
}
for (int i = 1; i <= n; i++) {
    sum[i] = sum[i - 1] + a[i - 1];
}
```

```
int a[maxn], sum[maxn];
for (int i = 1; i <= n; i++) { // 1..n
    cin >> a[i];
}
for (int i = 1; i <= n; i++) {
    sum[i] = sum[i - 1] + a[i];
}
```

有的时候我们可能需要把前缀和数组单拎出来来求，比如说需要等待数组排序后求前缀和，或者原本数组需要有限经过另一个预处理的时候。代码不是一成不变的。各位同学一定不要去背代码，而是理解加记忆。

课堂习题

[U430347] 前缀和(模板) https://www.luogu.com.cn/problem/U430347

题目描述

现在有一个长度为n的数组a，现有m次查询，对于每次查询给定一个区间$[l_i, r_i]$，输出数组中下标区间$[l_i, r_i]$的和。

输入格式

第一行两个整数 n和m，分别表示数组的长度和需要查询的次数。

第二行n个整数，表示数组a。

接下来m行，每行两个整数l_i，r_i，表示查询区间。

输出格式

共m行，每行一个整数，表示查询的答案。

样例 #1

样例输入 #1

```
3 2
1 3 5
2 3
1 3
```

样例输出 #1

```
8
9
```

提示

对于30%的数据，有$1 \leq n \leq 10^3$，$1 \leq m \leq 10$，$|a[i]| \leq 1000$，$1 \leq l_i, r_i \leq n$

对于100%的数据，有$1 \leq n \leq 2 \times 10^5$，$1 \leq m \leq 10^5$，$|a[i]| \leq 10^9$，$1 \leq l_i, r_i \leq n$

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, m, l, r, a[200001];
long long sum[200001]; // 由于前缀和数组需要储存若干个数的和，可能要存的值异常大，一般情况都开long long
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) { // 0..n - 1
        cin >> a[i];
        sum[i + 1] = sum[i] + a[i];   // 前缀和预处理
    }
    while (m--) {
        cin >> l >> r;
        if (l > r) swap(l, r); // 细节要做到
        cout << sum[r] - sum[l - 1] << endl; // 解前缀和
    }
    return 0;
}

```

时空复杂度

时间复杂度$O(n+m)$，n表示数组的长度，m表示询问的次数

空间复杂度$O(n)$，n表示数组的长度。

[U430393] 游戏币 https://www.luogu.com.cn/problem/U430393

题目背景

游乐场组织了一场送游戏币的活动，规则如下：

1.每张游戏币券所能兑换的游戏币**数目不等**，并且所有的游戏币券摞成一摞。

2.每次抽取只能从最底下或者上面抽取一张游戏币券，不能从中间抽取。

题目描述

小明现在参与这场活动。现在有n张游戏币券摞成一摞，小明只能抽k次游戏币券，小明希望拿到最多的游戏币，如果按照规则来拿，输出最多能获得游戏币数量。

输入格式

第一行两个整数，n，k,分别表示游戏币券的数量以及抽取的次数。

第二行n个整数，表示从上到下每一张券可兑换的游戏币数量。

输出格式

一个整数，表示能获得的最大游戏币数量。

样例 #1

样例输入 #1

```
7 3
1 2 3 4 5 6 1
```

样例输出 #1

```
12
```

样例 #2

样例输入 #2

```
7 7
9 7 7 9 7 7 9
```

样例输出 #2

```
55
```

提示

设a_i代表第i张游戏币券的所能兑换的游戏币数。

对于30%的数据，有$1 \le n \le 10^3$,$1 \le k \le 10$，$1 \le a_i \le 3\times10^4$

对于100%的数据，有$1 \le n \le 6\times10^5$,$1 \le k \le 4\times10^5$，$1 \le a_i \le 2\times10^9$

思路分析

> 读完题后，需要先明确一点，要从`n`张牌里拿`k`张，`k`得先防止溢出，`k = min(n, k)`。
>
> 从两头拿牌，所以枚举从左端拿`i(i <= k)`张牌，那就需要从牌尾拿走`k - i`张牌。当前的得分就是$(a_1 + a_2 + \ldots + a_i) + (a_{n-k+i+1} + \ldots + a_n)$。
>
> 所以左边的和就是$sum[i]$，末尾的和为$[n-k+i+1, n]$的和，也就是$sum[n] - sum[n-k+i]$。
>
> 最终答案取枚举的结果的最大值即可。所以可以写出前缀和代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
long long n, k, x, ans, sum[600001];
int main() {
    cin >> n >> k;
    k = min(n, k);
    for (int i = 1; i <= n; i++) {
        cin >> x;
        sum[i] += sum[i - 1] + x;  // 输入 + 前缀和预处理，完全式变读入边处理
    }
    for (int i = 0; i <= k; i++) { // i 表示枚举从左侧拿 i 张牌
        ans = max(ans, sum[i] + sum[n] - sum[n - k + i]);
    }
    cout << ans;
    return 0;
}
```

时空复杂度

时间复杂度$O(n+k)$，n表示数组的长度，k表示拿走的牌的数目。

空间复杂度$O(n)$，需要对n个数求前缀和存储。

[U430831] 奇数统计 https://www.luogu.com.cn/problem/U430831

题目描述

现在有一个长度为n的数组a，现有m次查询，对于每次查询给定一个区间$[l_i，r_i]$，输出数组中下标区间$[l_i，r_i]$的奇数个数的和。

输入格式

第一行两个整数 n和m，分别表示数组的长度和需要查询的次数。

第二行n个整数，表示数组a。

接下来m行，每行两个整数$l_i，r_i$，表示查询区间。

输出格式

共m行，每行一个整数，表示查询的答案。

样例 #1

样例输入 #1

```
9 1
83 84 81 95 97 25 54 20 42
3 6
```

样例输出 #1

```
4
```

样例 #2

样例输入 #2

```
3 2
48 98 81
2 1
2 3
```

样例输出 #2

```
0
1
```

提示

对于30%的数据，有$1 \leq n \leq 10^3$，$1 \leq m \leq 10$，$|a[i]| \leq 1000$，$1 \leq l_i, r_i \leq n$

对于100%的数据，有$1 \leq n \leq 2 \times 10^5$，$1 \leq m \leq 10^5$，$a_i \leq |a[i]| \leq 10^9$，$1 \leq l_i, r_i \leq n$

思路分析

> 和模板题没有太大差别，区别在于前缀和数组不再统计$[1..i]$的数的和，而是统计$[1..i]$的奇数个数。

```cpp
#include <bits/stdc++.h>
using namespace std;
int sum[200001];  // 这题比较特殊，就算全为奇数，前缀和数组记录的数最大也不超过n，所以不用开long long
int main() {
    int n, m, x, l, r;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> x;
        sum[i] = sum[i - 1];
        if (x % 2) sum[i]++; // 如果是奇数，则[1..i]的奇数个数 + 1
    }
    while (m--) {
        cin >> l >> r;
        if (l > r) swap(l, r); // 细节
        cout << sum[r] - sum[l - 1] << endl;
    }
    return 0;
}
```

时空复杂度

时间复杂度$O(n+m)$，n表示数组的长度，m表示询问的次数

空间复杂度$O(n)$，n表示数组的长度。

总结

前缀和从某种意义上来说并不能称之为是**算法(Algorithm)**，更多的是一种思想，一种数学手段降低时间复杂度的一种技巧。

这也是$CSP-J$比赛中常见的一种数据预处理手段。

更重要的是，前缀和也是很多其他算法和处理方式的前置知识，例如：差分数组，后缀数组(SA)，线段树等一些高难度算法的一种降维技巧。

在这些技巧里，虽然写法不同，主要的解决的问题范围也不同，但是大体上都有一个相同的目标：

以空间换时间

通过建立前缀和数组，我们降低了查询某个区间的数的和的时间，因为我们已经预先将部分的和算出来了，相当于是一种提前准备的思路。代价就是需要占用更多的计算机空间，对于计算机的空间，这是可以通过经济手段提升的，而且成本也非常低，时间可是十分宝贵的东西。所以将来大家会学习到很多**以空间换时间**的思路，对于一个程序员，一个竞赛选手来说，这是血赚的。