

一维差分：

差分可以看成前缀和的逆运算。

差分数组：

首先给定一个原数组a：a[1], a[2], a[3],,,,,, a[n];

然后我们构造一个数组b ： b[1] ,b[2] , b[3],,,,,, b[i];

使得 a[i] = b[1] + b[2 ]+ b[3] +,,,,,, + b[i]

也就是说，a数组是b数组的前缀和数组，反过来我们把b数组叫做a数组的差分数组。换句话说，每一个a[i]都是b数组中从头开始的一段区间和。

考虑如何构造差分b数组？

最为直接的方法

如下：

a[0 ]= 0;

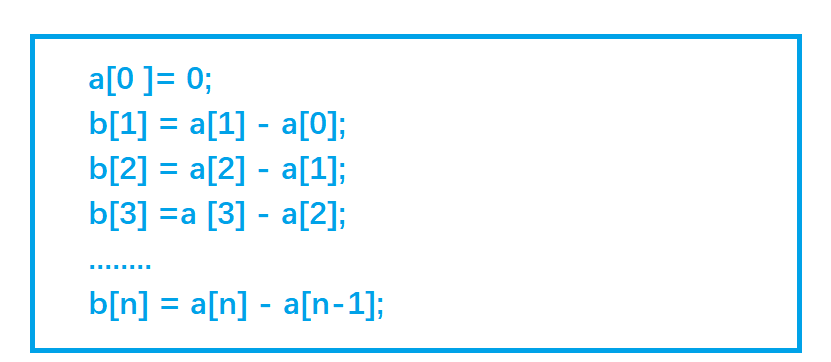
b[1] = a[1] - a[0];

b[2] = a[2] - a[1];

b[3] =a [3] - a[2];

........

b[n] = a[n] - a[n-1];

图示:

我们只要有b数组，通过前缀和运算，就可以在O(n) 的时间内得到a数组 。

知道了差分数组有什么用呢？ 别着急，慢慢往下看。

话说有这么一个问题：

给定区间[l ,r ]，让我们把a数组中的[ l, r]区间中的每一个数都加上c,即 a[l] + c , a[l+1] + c , a[l+2] + c ,,,,,, a[r] + c;

暴力做法是for循环l到r区间，时间复杂度O(n)，如果我们需要对原数组执行m次这样的操作，时间复杂度就会变成O(n\*m)。有没有更高效的做法吗? 考虑差分做法，(差分数组派上用场了)。

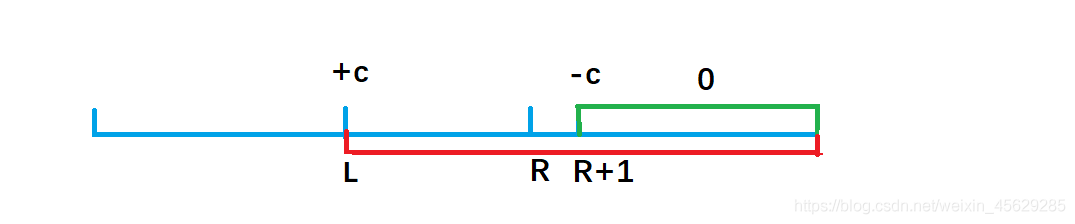
始终要记得，a数组是b数组的前缀和数组，比如对b数组的b[i]的修改，会影响到a数组中从a[i]及往后的每一个数。

首先让差分b数组中的 b[l] + c ,通过前缀和运算，a数组变成 a[l] + c ,a[l+1] + c,,,,,, a[n] + c;

然后我们打个补丁，b[r+1] - c, 通过前缀和运算，a数组变成 a[r+1] - c,a[r+2] - c,,,,,,,a[n] - c;

为啥还要打个补丁？

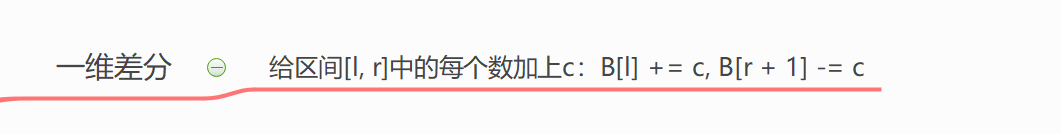
我们画个图理解一下这个公式的由来:



b[l] + c，效果使得a数组中 a[l]及以后的数都加上了c(红色部分)，但我们只要求l到r区间加上c, 因此还需要执行 b[r+1] - c,让a数组中a[r+1]及往后的区间再减去c(绿色部分)，这样对于a[r] 以后区间的数相当于没有发生改变。

因此我们得出一维差分结论：给a数组中的[ l, r]区间中的每一个数都加上c,只需对差分数组b做 b[l] + = c, b[r+1] - = c。时间复杂度为O(1), 大大提高了效率。

总结：



二维差分：

如果扩展到二维，我们需要让二维数组被选中的子矩阵中的每个元素的值加上c,是否也可以达到O(1)的时间复杂度。答案是可以的，考虑二维差分。

a[][]数组是b[][]数组的前缀和数组，那么b[][]是a[][]的差分数组

原数组： a[i][j]

我们去构造差分数组： b[i][j]

使得a数组中a[i][j]是b数组左上角(1,1)到右下角(i,j)所包围矩形元素的和。

如何构造b数组呢？

其实关于差分数组，我们并不用考虑其构造方法，因为我们使用差分操作在对原数组进行修改的过程中，实际上就可以构造出差分数组。

同一维差分，我们构造二维差分数组目的是为了 让原二维数组a中所选中子矩阵中的每一个元素加上c的操作，可以由O(n\*n)的时间复杂度优化成O(1)

已知原数组a中被选中的子矩阵为 以(x1,y1)为左上角，以(x2,y2)为右上角所围成的矩形区域;

始终要记得，a数组是b数组的前缀和数组，比如对b数组的b[i][j]的修改，会影响到a数组中从a[i][j]及往后的每一个数。

假定我们已经构造好了b数组，类比一维差分，我们执行以下操作

来使被选中的子矩阵中的每个元素的值加上c

b[x1][y1] + = c;

b[x1,][y2+1] - = c;

b[x2+1][y1] - = c;

b[x2+1][y2+1] + = c;

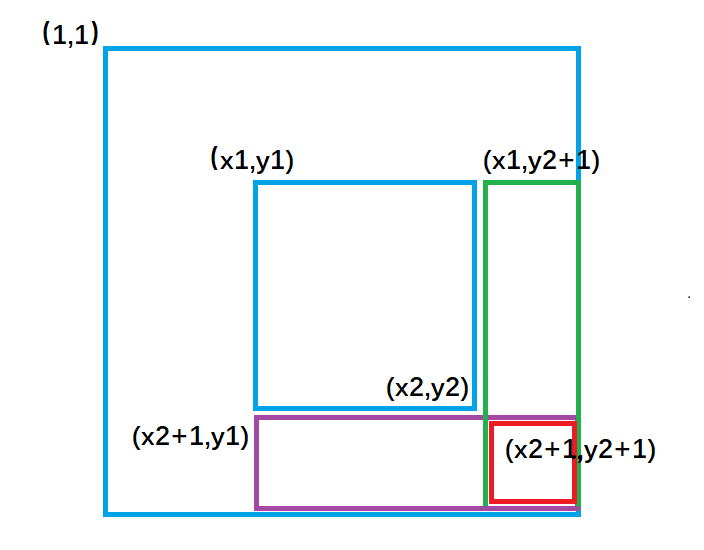
每次对b数组执行以上操作，等价于：

for(int i=x1;i<=x2;i++)

for(int j=y1;j<=y2;j++)

a[i][j]+=c;

我们画个图去理解一下这个过程：

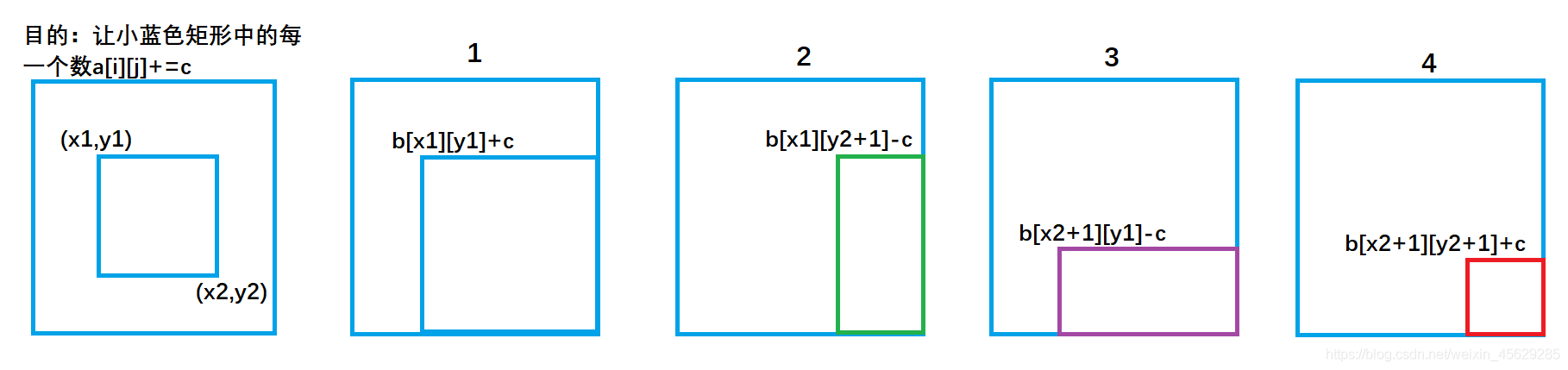


b[x1][ y1 ] +=c ; 对应图1 ,让整个a数组中蓝色矩形面积的元素都加上了c。

b[x1,][y2+1]-=c ; 对应图2 ,让整个a数组中绿色矩形面积的元素再减去c，使其内元素不发生改变。

b[x2+1][y1]- =c ; 对应图3 ,让整个a数组中紫色矩形面积的元素再减去c，使其内元素不发生改变。

b[x2+1][y2+1]+=c; 对应图4,让整个a数组中红色矩形面积的元素再加上c，红色内的相当于被减了两次，再加上一次c，才能使其恢复。



我们将上述操作封装成一个插入函数:

void insert(int x1,int y1,int x2,int y2,int c)

{ //对b数组执行插入操作，等价于对a数组中的(x1,y1)到(x2,y2)之间的元素都加上了c

b[x1][y1]+=c;

b[x2+1][y1]-=c;

b[x1][y2+1]-=c;

b[x2+1][y2+1]+=c;

}

我们可以先假想a数组为空，那么b数组一开始也为空，但是实际上a数组并不为空，因此我们每次让以(i,j)为左上角到以(i,j)为右上角面积内元素(其实就是一个小方格的面积)去插入 c=a[i][j]，等价于原数组a中(i,j) 到(i,j)范围内 加上了 a[i][j] ,因此执行n\*m次插入操作，就成功构建了差分b数组.

这叫做曲线救国。

代码如下：

for(int i=1;i<=n;i++)

{

for(int j=1;j<=m;j++)

{

insert(i,j,i,j,a[i][j]); //构建差分数组

}

}

当然关于二维差分操作也有直接的构造方法，公式如下：

b[i][j]=a[i][j]−a[i−1][j]−a[i][j−1]+a[i−1][j−1]

二维差分数组的构造同一维差分思维相同，因次在这里就不再展开叙述了。

总结：

