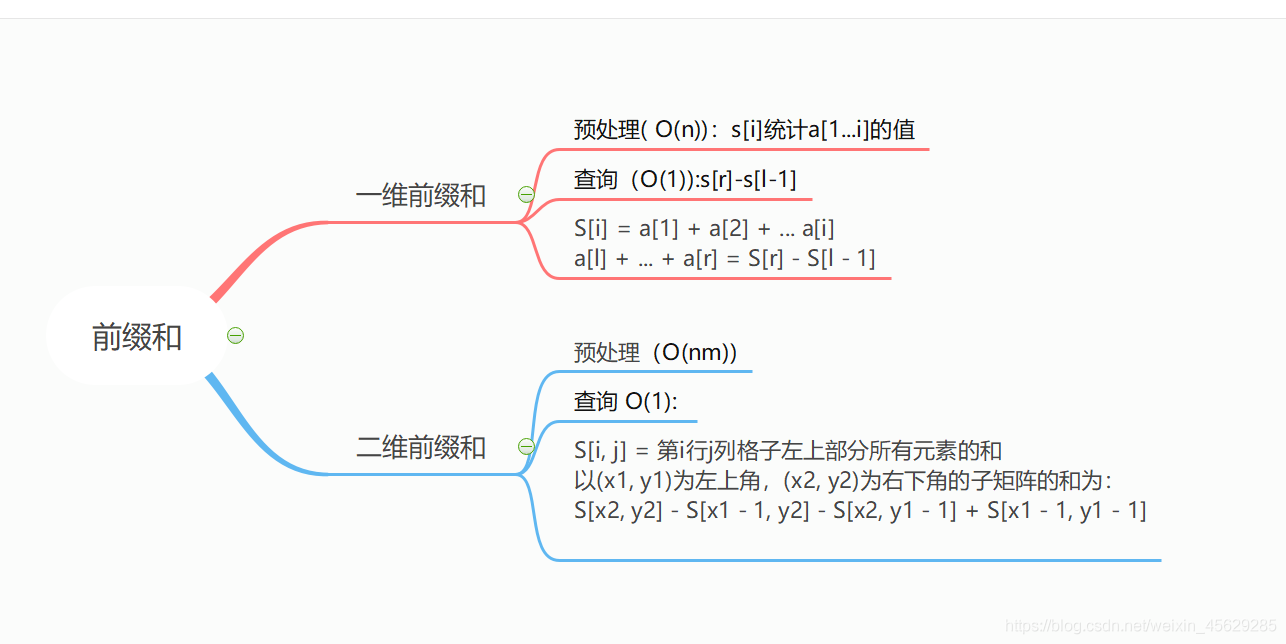
前缀和



前缀和可以使算法的时间复杂度大大降低

因为你不必继续遍历那些对答案实际无影响的数据

Eg1：一维前缀和

具体做法：

首先做一个预处理，定义一个sum[]数组，sum[i]代表a数组中前i个数的和。

求前缀和运算：

const int N=1e5+10;

int sum[N],a[N]; //sum[i]=a[1]+a[2]+a[3].....a[i];

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

{

sum[i]=sum[i-1]+a[i];

}

然后查询操作：

scanf("%d%d",&l,&r);

printf("%d\n", sum[r]-sum[l-1]);

对于每次查询，只需执行sum[r]-sum[l-1] ，时间复杂度为O(1)

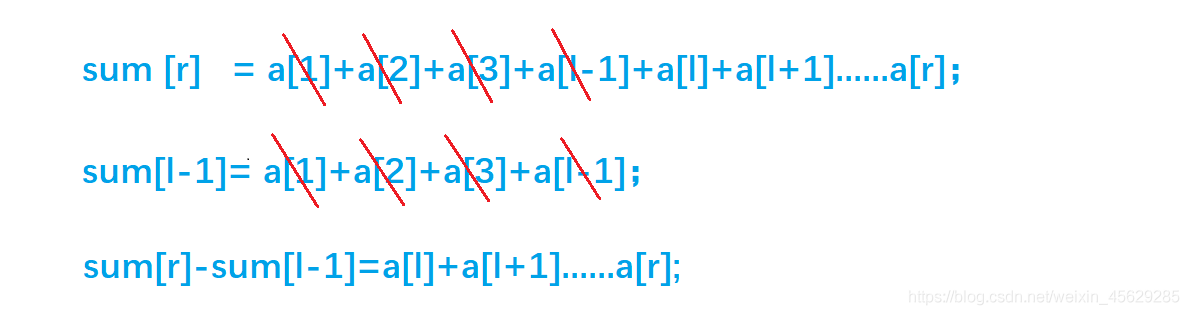
原理

sum[r] =a[1]+a[2]+a[3]+a[l-1]+a[l]+a[l+1]......a[r];

sum[l-1]=a[1]+a[2]+a[3]+a[l-1];

sum[r]-sum[l-1]=a[l]+a[l+1]+......+a[r];

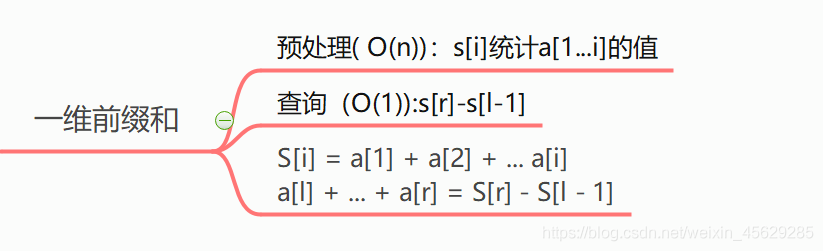
图解



这样，对于每个询问，只需要执行 sum[r]-sum[l-1]。输出原序列中从第l个数到第r个数的和的时间复杂度变成了O(1)。

我们把它叫做一维前缀和。

总结：

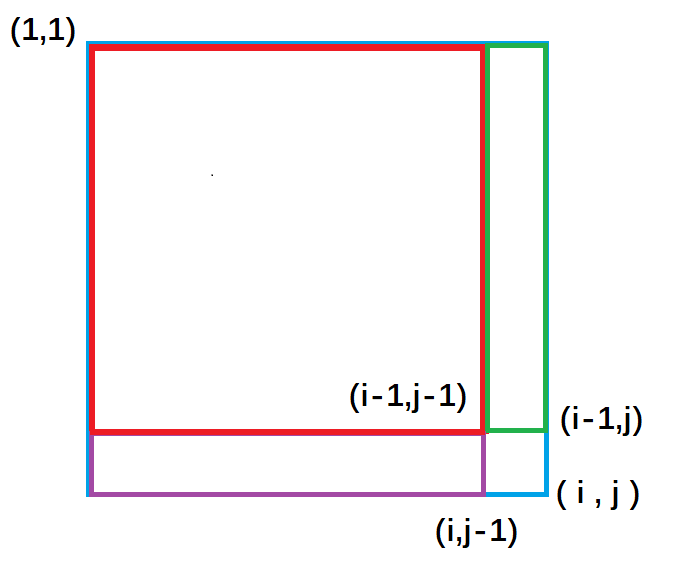


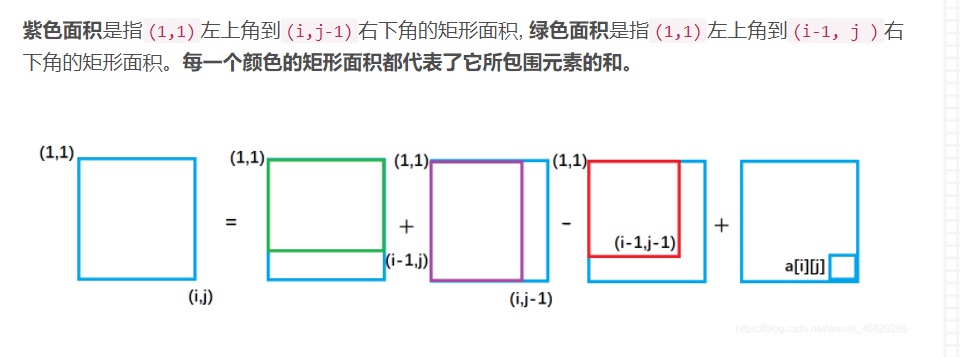
二维前缀和：

输入一个n行m列的整数矩阵，再输入q个询问，每个询问包含四个整数x1, y1, x2, y2，表示一个子矩阵的左上角坐标和右下角坐标。对于每个询问输出子矩阵中所有数的和。

同一维前缀和一样，我们先来定义一个二维数组s[][], s[i][j]表示二维数组中，左

上角(1,1)到右下角( i,j )所包围的矩阵元素的和。接下来推导二维前缀和的公式。



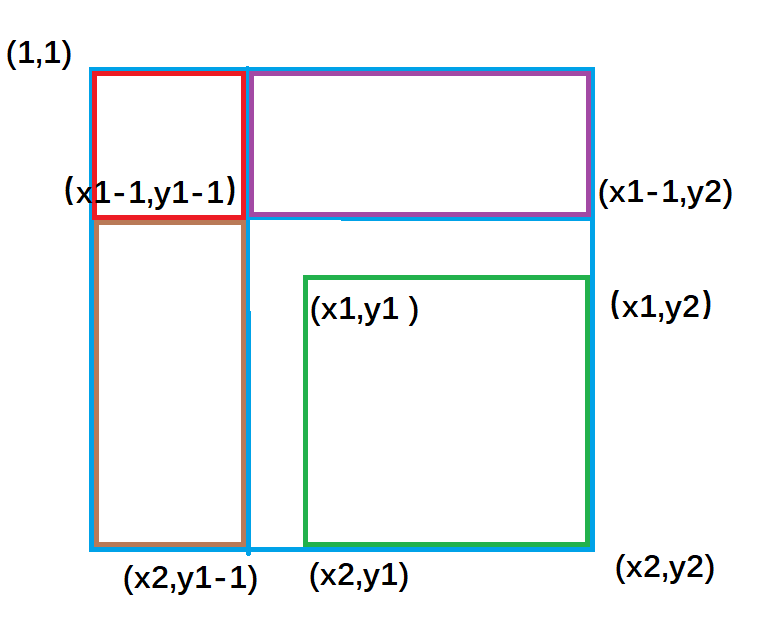


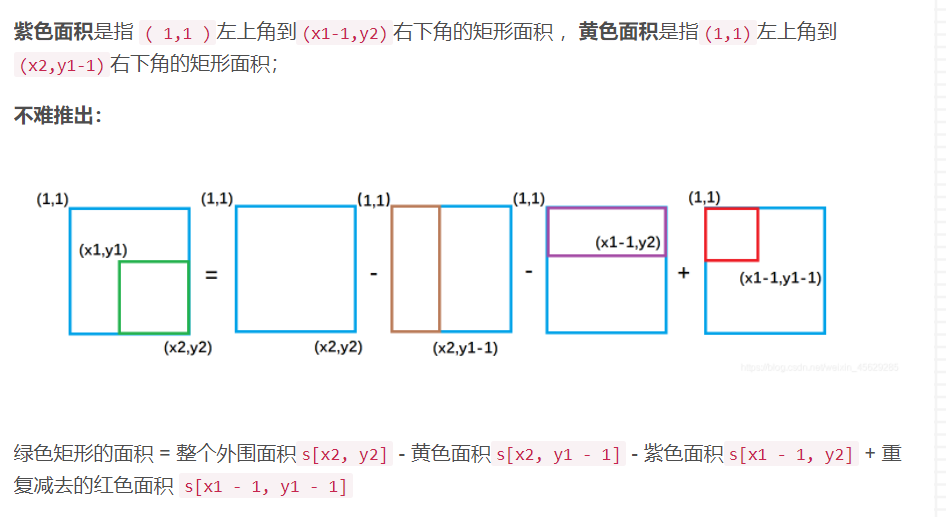
从图中我们很容易看出，整个外围蓝色矩形面积s[i][j] = 绿色面积s[i-1][j] + 紫色面积s[i][j-1] - 重复加的红色的面积s[i-1][j-1]+小方块的面积a[i][j];

因此得出二维前缀和预处理公式

s[i] [j] = s[i-1][j] + s[i][j-1 ] + a[i] [j] - s[i-1][ j-1]

接下来回归问题去求以(x1,y1)为左上角和以(x2,y2)为右下角的矩阵的元素的和。

如图：



****因此二维前缀和的结论为：****

以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：  
s[x2, y2] - s[x1 - 1, y2] - s[x2, y1 - 1] + s[x1 - 1, y1 - 1]

****总结：****

