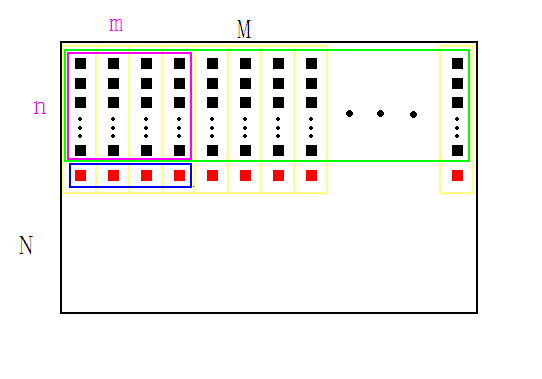
Boxfilter的原理有点类似Integral Image，而且比它还要快，但是实现步骤比较复杂。在计算矩形特征之前，Boxfilter与Integral Image都需要对图像进行初始化（即对数组A赋值），不同于Integral Image, Boxfilter的数组A中的每个元素的值是该像素邻域内的像素和（或像素平方和），在需要求某个矩形内像素和的时候，直接访问数组中对应的位置就可以了。因此可以看出它的复杂度是O(1)。



Boxfilter的初始化过程如下：

1、给定一张图像，宽高为（M,N），确定待求矩形模板的宽高(m,n)，如图紫色矩形。图中每个黑色方块代表一个像素，红色方块是假想像素。

2、开辟一段大小为M的数组，记为buff, 用来存储计算过程的中间变量，用红色方块表示

3、将矩形模板（紫色）从左上角（0，0）开始，逐像素向右滑动，到达行末时，矩形移动到下一行的开头（0，1），如此反复，每移动到一个新位置时，计算矩形内的像素和，保存在数组A中（数组A大小和原图一样，初值为0）。以(0,0)位置为例进行说明：首先将绿色矩形内的每一列像素求和，结果放在buff内（红色方块），再对蓝色矩形内的像素求和，结果即为紫色特征矩形内的像素和，把它存放到数组A（对应位置中，考虑模板尺寸影响，边缘像素舍弃）中，如此便完成了第一次求和运算。

4、每次紫色矩形向右移动时，实际上就是求对应的蓝色矩形的像素和，此时只要把上一次的求和结果减去蓝色矩形内的第一个红色块，再加上它右面的一个红色块，就是当前位置的和了，用公式表示 sum[i] = sum[i-1] - buff[x-1] + buff[x+m-1]

5、当紫色矩形移动到行末时，需要对buff进行更新。因为整个绿色矩形下移了一个像素，所以对于每个buff[i], 需要加上一个新进来的像素，再减去一个出去的像素，然后便开始新的一行的计算了。