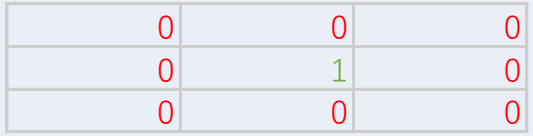
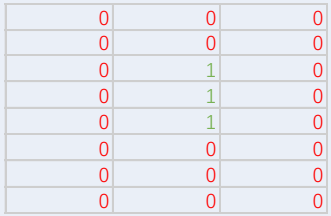
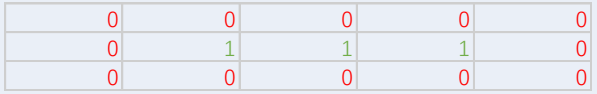
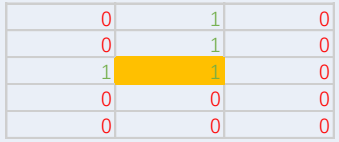
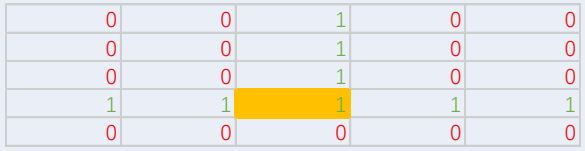
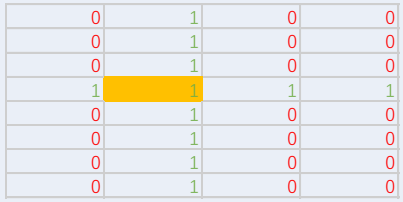
4

题目：给定一M*N 的0,1矩阵，找到其中最大的以1为边的矩形

求解思路1：生成矩形

分类		举例	作用
孤立点			对形成矩阵无任何作用
边点	纵边型		可能是矩形某条边上的一点 但不可能是矩形的顶点
	横边型		
角点	拐点型		可能是矩形的一个顶点
	T字型		
	十字型		

算法思想：

1. 找出所有的角点。
2. 然后用这些角点去生成矩阵。
3. 然后从所有生成的矩形中找出面积最大的。

注意，角点也是一种边点，只不过角点即是纵边点，也是横边点，这也是角点的判断依据。如果一个点是边点但不是角点，那么我称之为纯边点。

算法步骤

1. 构造生成图结构
    1. 横边检测：检测每一行，标记每一个1的分类(是横孤立点,还是某个横边的点)
    2. 纵边检测：检测每一行，进一步标记每一个1的分类（是纵孤立点，还是某个纵边的点）
    3. 注意：如果一个点即是横边的点，优势纵边的点，那么这个点就是角点。
2. BFS所有角点构造极大联通子图，并生成单位矩形。
3. 合并单位矩形成为更大矩形，求解出个联通子图最大矩形。
4. 比较各联通子图，求解出最终MAX矩形。

数据结构简单示意:

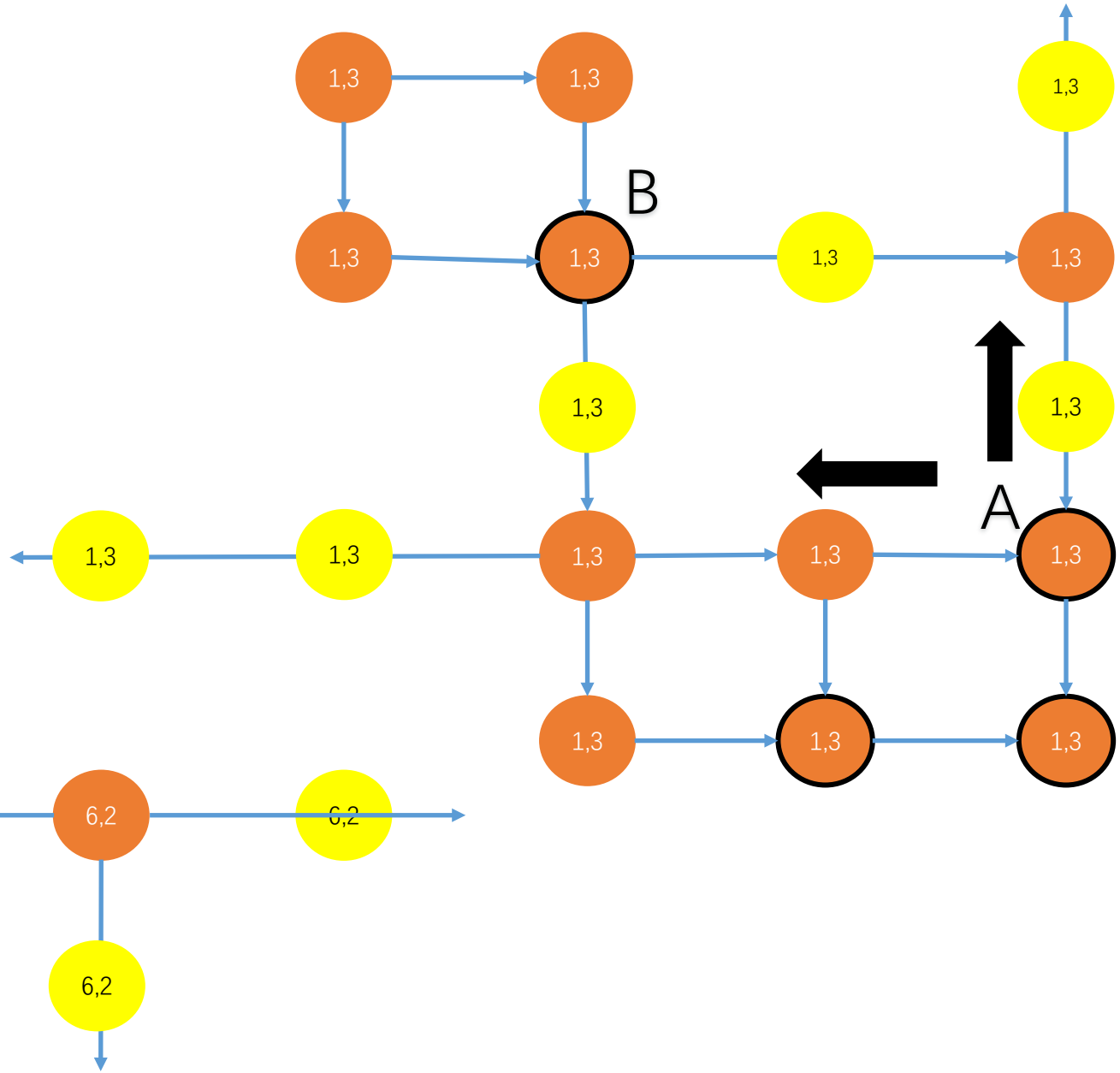
原0,1矩阵

	1	2	3	4	5	6			
1	1	0	1	1	0	1			孤立点
2	0	0	1	1	1	1			纯边点
3	0	0	0	1	0	1			角点
4	0	1	1	1	1	1			
5	0	0	0	1	1	1			
6	1	1	1	0	0	0			
7	0	1	0	0	0	1			

数据结构简单示意

1. 可以看出此例中，会形成两个极大联通子图，
2. 注意每个圆圈中的数字即为其所属的极大联通子图的编号(用起始点的位置进行编号)
3. 生成图总是从未被纳入极大联通子图中找最左最上的角点来开始生成。
4. 某个极大联通子图生成过程是一个BFS过程。
5. BFS过程中当遇到双父亲节点时候（图中黑圈角点）就说明遇到了一个单位矩形（无内涵矩形），这时需要从当前双父亲节点向祖先回溯寻找该单位矩阵的起始节点，例如对于A点来说，其单位矩阵的起始节点即为B.
6. 双父亲节点的回溯过程可沿着任意父亲路径进行，如图中箭头所示，这里关键在于定义起始节点的独特特征：
    1. 一定位于双父节点的左上侧。
    2. 一定具有向右的孩子，和向下的孩子
7. 通过双父节点和对应的起始节点就能够确定一个单位矩形，继续BFS过程，可以找到这个极大联通子图的所有单位矩形。
8. 可以观察到，有些边同时位于2个矩形中，所以接下来我们可以通过重合边进行矩形合并（类似于并查集合并）进而构造出更大的矩形，直至不能合并，及所有矩形都达到了极大值。
9. 从所有极大值举行中找出该极大联通子图的最大矩形。
10. 找出所有极大联通子图的最大值，即为极大值。

生成图



性能讨论

1. 该算法整体上式生成思路，类似于动态规划，而不是搜索思路。生成思路往往能够避免大量重复计算，本质上是以时间换空间，从这个角度该算法可能具有较好的时间复杂度。
2. 可以看到，在求解最大矩形的时候，该算法首先根据联通性，将矩阵进行划分，进而分治，求解每个部分的最大值。对于规模比较大的问题，分治策略往往可快速降低时间复杂度。
3. 本算法并未直接生成所有矩形，而是只生成单位矩形，再利用重合边进行矩形组合。这种操作充分利用了图形的自带几何性质，相比生成所有矩形的算法，极大的减少了矩形的生成量，进而极大的提升算法性能。
4. 数学上一个求解一个函数的最大值的基本思路是：先找到各个独立的定义域的多个极大值，再通过比较找出最大值，再求出整个定义域的最大值。本算法与这个函数求最大值具有极高相似性，本算法是先求出各个独立联通子图的多个极大值(矩形合并的极限)，再通过比较求出最大值，再求出所有联通子图中的最大值。

算法问题

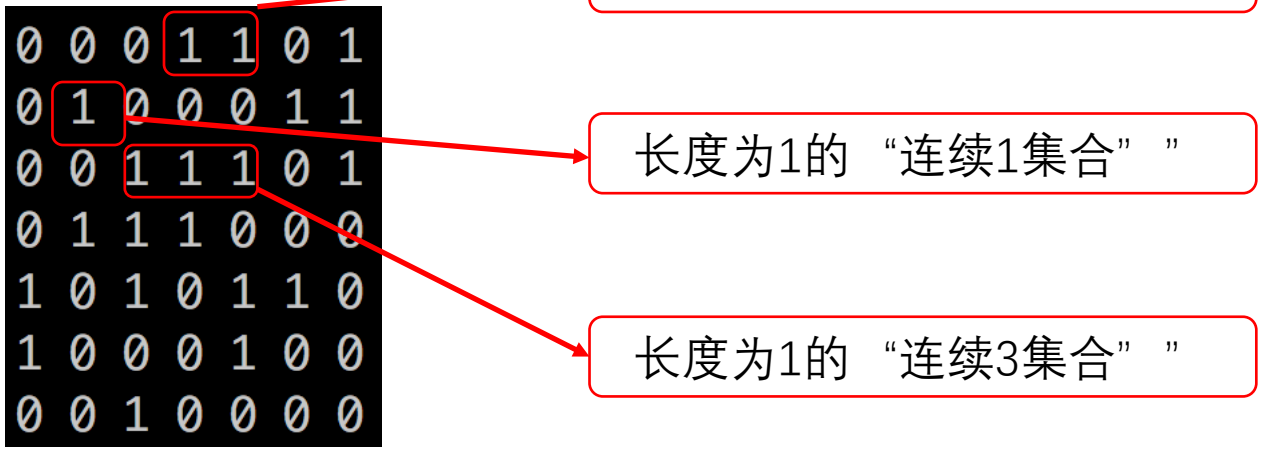
1. 本算法核心问题是缺乏数理逻辑上的完备性证明，虽然根据直观推理该算法能够找出最大矩形，但是受限于我的数学功底，目前我还无法严格证明这点，所以该算法无法严格正确。

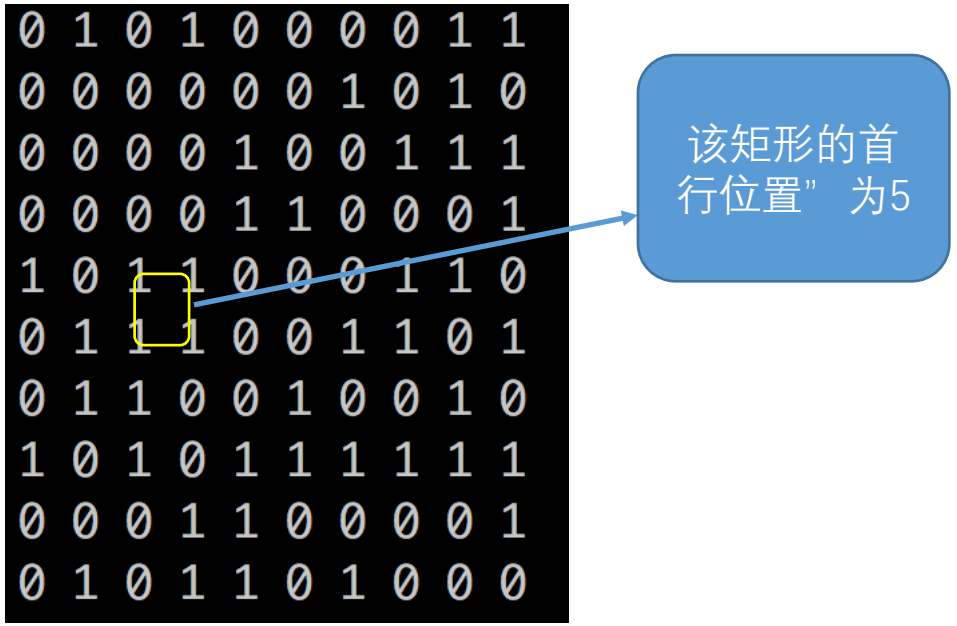
求解思路2：分类讨论

算法思想：

题目要求根据在01矩阵中找到以1位边的矩形，可以用矩形组成要素进行分类，先求解出每个类别的最大矩形，再比较求解出整体最大值，具体过程为：（矩阵行数为M，列数为N）

1. 首先根据矩形的“首行位置”进行分类，可分成M类。
2. 确定“首行位置”,可根据每行的“连续1集合”进行分类。
3. 确定“首行位置”与“连续1集合”，再根据矩形高度进行分类。





数据结构简单示意

```
0 1 0 1
1 0 1 0
0 1 0 0
0 1 1 1
```

原0,1矩阵

```
(0,0,-1,-1,0)  (0,1,1,1,1)  (0,2,-1,-1,0)  (0,3,3,3,1)
(1,0,0,0,1)  (1,1,-1,-1,0)  (1,2,2,2,1)  (1,3,-1,-1,0)
(2,0,-1,-1,0)  (2,1,1,1,2)  (2,2,-1,-1,0)  (2,3,-1,-1,0)
(3,0,-1,-1,0)  (3,1,1,3,1)  (3,2,1,3,1)  (3,3,1,3,1)
```

nodeArray:

```
[0,1,1,1,]  [0,3,3,1,]
[1,0,0,1,]  [1,2,2,1,]
[2,1,1,2,]
[3,1,3,1,]
```

rowInfs:

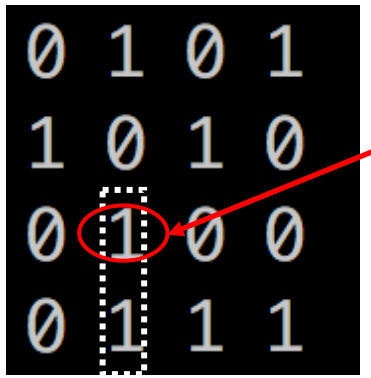
数据结构nodeArray:

```
(0,0,-1,-1,0)  (0,1,1,1,1)  (0,2,-1,-1,0)  (0,3,3,3,1)
(1,0,0,0,1)  (1,1,-1,-1,0)  (1,2,2,2,1)  (1,3,-1,-1,0)
(2,0,-1,-1,0)  (2,1,1,1,2)  (2,2,-1,-1,0)  (2,3,-1,-1,0)
(3,0,-1,-1,0)  (3,1,1,3,1)  (3,2,1,3,1)  (3,3,1,3,1)
```

数据结构nodeArray:

```
(0,0,-1,-1,0)  (0,1,1,1,1)  (0,2,-1,-1,0)  (0,3,3,3,1)
(1,0,0,0,1)  (1,1,-1,-1,0)  (1,2,2,2,1)  (1,3,-1,-1,0)
(2,0,-1,-1,0)  (2,1,1,1,2)  (2,2,-1,-1,0)  (2,3,-1,-1,0)
(3,0,-1,-1,0)  (3,1,1,3,1)  (3,2,1,3,1)  (3,3,1,3,1)
```

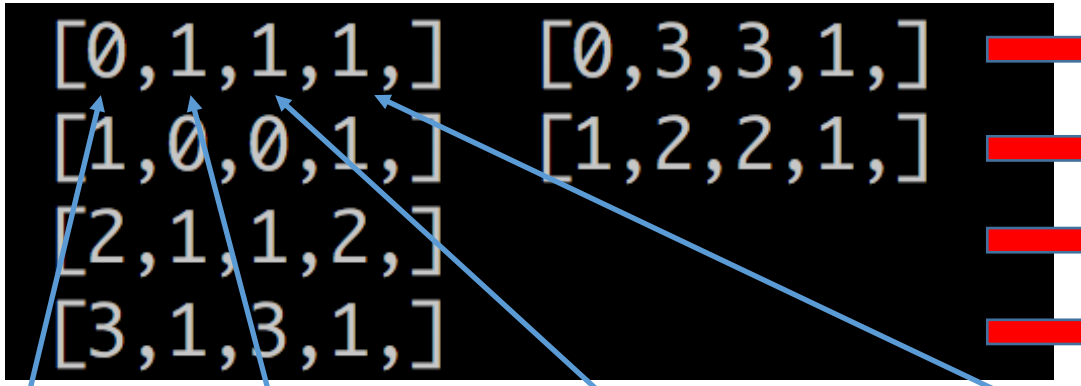
节点行号 | 节点列号 | 节点归属的“连续1集合”的左起点 | 节点归属的“连续1集合”的右终点 | 节点高度



该节点的高度为2

节点高度定义

数据结构rowInfs:



每行的“连续1集合”信息

行号

连续1集合”的左起点

“连续1集合”的右终点

“连续1集合”内部的最高节点高度

算法问题

时间复杂度太高，若MN是同一个量级的，则该算法的上界可达到O(n^3)