

Grandyang

仰天长啸仗剑红尘，冬去春来寒暑几更...

博客园

首页

新随笔

联系

订阅

管理

随笔 - 1094 文章 - 1 评论 - 1068

公告



Get a FREE visitor map for your site!

昵称：Grandyang
园龄：5年5个月
粉丝：232
关注：29
[+加关注](#)

< 2017年9月 >						
日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

最新随笔

1. [LeetCode] Find K Closest Elements 寻找K个最近元素
2. [LeetCode] Judge Route Circle 判断路线绕圈
3. [LeetCode] Maximum Binary Tree 最大二叉树
4. [LeetCode] Two Sum IV - Input is a BST 两数之和为四 - 输入是二叉搜索树
5. [LeetCode] Find Duplicate Subtrees 寻找重复树
6. [LeetCode] Print Binary Tree 打印二叉树
7. [LeetCode] 4 Keys Keyboard 四键的键盘
8. [LeetCode] 2 Keys Keyboard 两键的键盘
9. [LeetCode] Dota2 Senate 刀塔二参议院
10. [LeetCode] Replace Words 替换单词

随笔分类

3D Visualization(10)

[LeetCode] Trapping Rain Water II 收集雨水之二

Given an $m \times n$ matrix of positive integers representing the height of each unit cell in a 2D elevation map, compute the volume of water it is able to trap after raining.

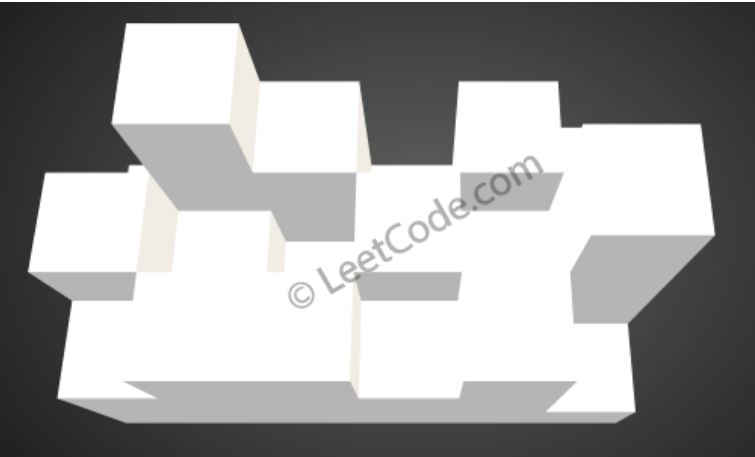
Note:
Both m and n are less than 110. The height of each unit cell is greater than 0 and is less than 20,000.

Example:

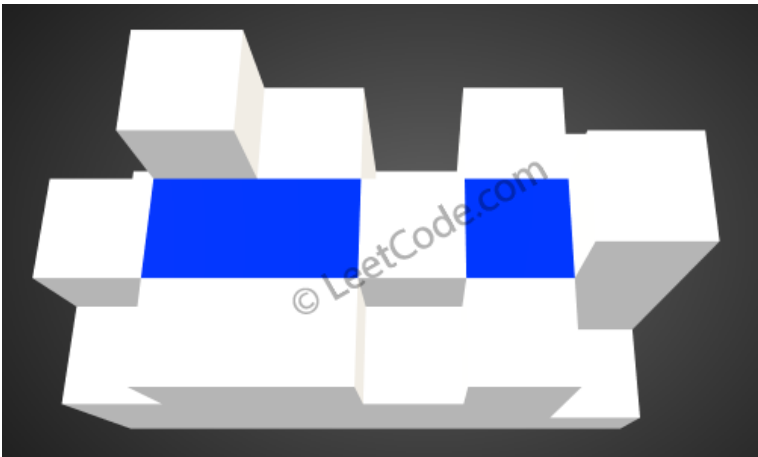
Given the following 3x6 height map:

```
[
  [1,4,3,1,3,2],
  [3,2,1,3,2,4],
  [2,3,3,2,3,1]
]
```

Return 4.



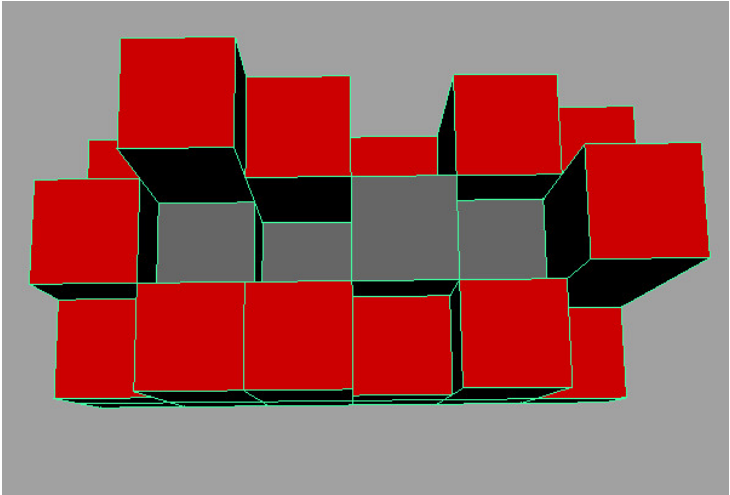
The above image represents the elevation map $[[1, 4, 3, 1, 3, 2], [3, 2, 1, 3, 2, 4], [2, 3, 3, 2, 3, 1]]$ before the rain.



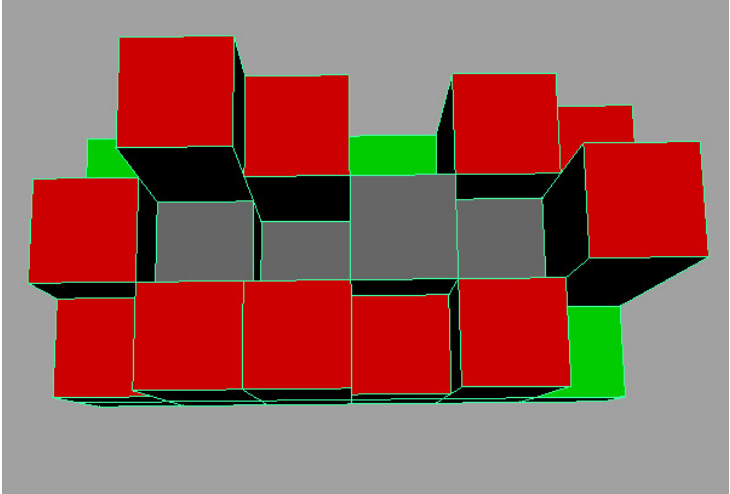
After the rain, water are trapped between the blocks. The total volume of water trapped is 4.

这道题是之前那道Trapping Rain Water的拓展，由2D变3D了，感觉很叼。但其实解法跟之前的完全不同了，之前那道题由于是二维的，我们可以用双指针来做，而这道三维的，我们需要用BFS来做，解法思路很巧妙，下面我们就以题目中的例子来进行分析讲解，多图预警，手机流量党慎入：

首先我们应该能分析出，能装水的底面肯定不能在边界上，因为边界上的点无法封闭，那么所有边界上的点都可以加入queue，当作BFS的启动点，同时我们需要一个二维数组来标记访问过的点，访问过的点我们用红色来表示，那么如下图所示：



我们再想想，怎么样可以成功的装进去水呢，是不是周围的高度都应该比当前的高度高，形成一个凹槽才能装水，而且装水量取决于周围最小的那个高度，有点像木桶原理的感觉，那么为了模拟这种方法，我们采用模拟海平面上升的方法来做，我们维护一个海平面高度mx，初始化为最小值，从1开始往上升，那么我们BFS遍历的时候就需要从高度最小的格子开始遍历，那么我们的queue就不能使用普通队列了，而是使用优先级队列，将高度小的放在队首，最先取出，这样我们就可以遍历高度为1的三个格子，用绿色标记出来了，如下图所示：



向周围BFS搜索的条件是不能越界，且周围格子未被访问，那么可以看出上面的第一个和最后一个绿格子无法进行进一步搜索，只有第一行中间那个绿格子可以搜索，其周围有一个灰格子未被访问过，将其加入优先队列

- Adobe Software(2)
- Algorithms(7)
- C/C++, Java, Python(32)
- CareerCup(150)
- CUDA/OpenCL(1)
- Digital Image Processing(3)
- Entertainment(6)
- GTK+/VTK/ITK/FLTK(20)
- IOS(3)
- LaTex(3)
- LeetCode(594)
- LintCode(101)
- MatLab(10)
- Maya / 3ds Max(10)
- MySQL(2)
- OpenCV(37)
- Point Grey Research(11)
- Qt(49)
- Useful Links(33)

随笔档案

- 2017年9月 (6)
- 2017年8月 (10)
- 2017年7月 (12)
- 2017年6月 (21)
- 2017年5月 (26)
- 2017年4月 (18)
- 2017年3月 (22)
- 2017年2月 (23)
- 2017年1月 (13)
- 2016年12月 (26)
- 2016年11月 (30)
- 2016年10月 (30)
- 2016年9月 (24)
- 2016年8月 (40)
- 2016年7月 (31)
- 2016年6月 (33)
- 2016年5月 (30)
- 2016年4月 (70)
- 2016年3月 (32)
- 2016年2月 (32)
- 2016年1月 (25)
- 2015年12月 (3)
- 2015年11月 (36)
- 2015年10月 (43)
- 2015年9月 (51)
- 2015年8月 (46)
- 2015年7月 (45)
- 2015年6月 (29)
- 2015年5月 (28)
- 2015年4月 (42)
- 2015年3月 (55)
- 2015年2月 (61)
- 2015年1月 (27)
- 2014年12月 (8)
- 2014年11月 (27)
- 2014年10月 (35)
- 2014年9月 (5)

积分与排名

积分 - 1517096
排名 - 30

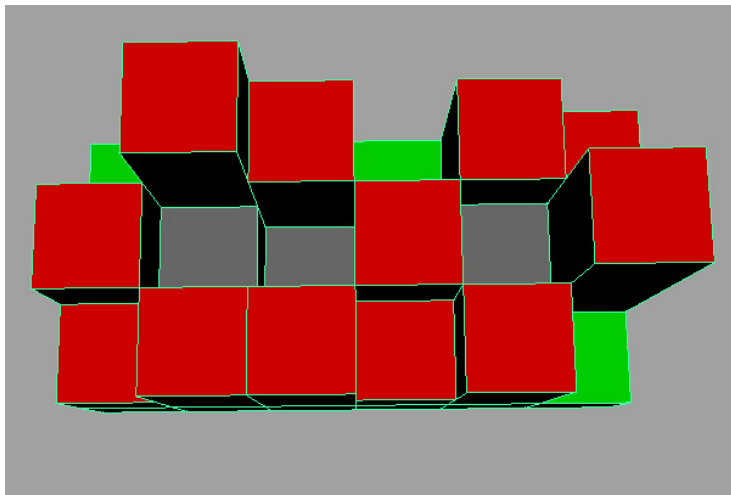
最新评论

阅读排行榜

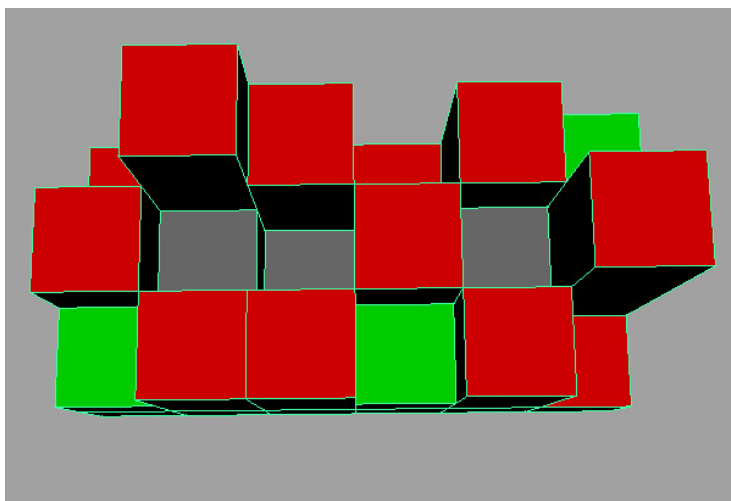
评论排行榜

推荐排行榜

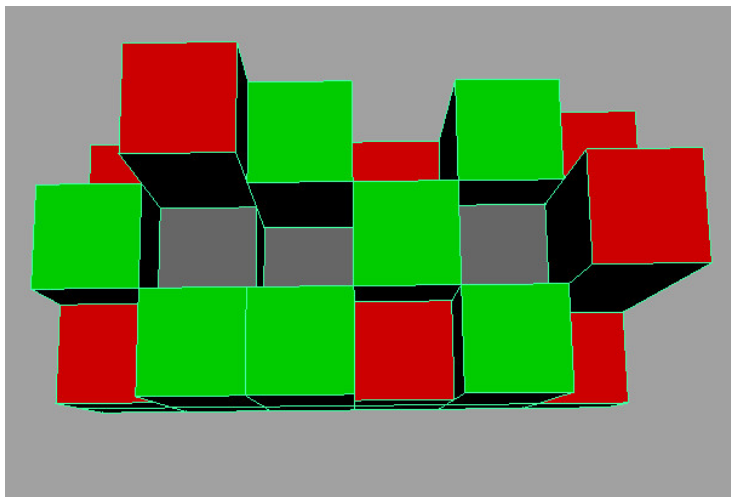
queue中，然后标记为红色，如下图所示：



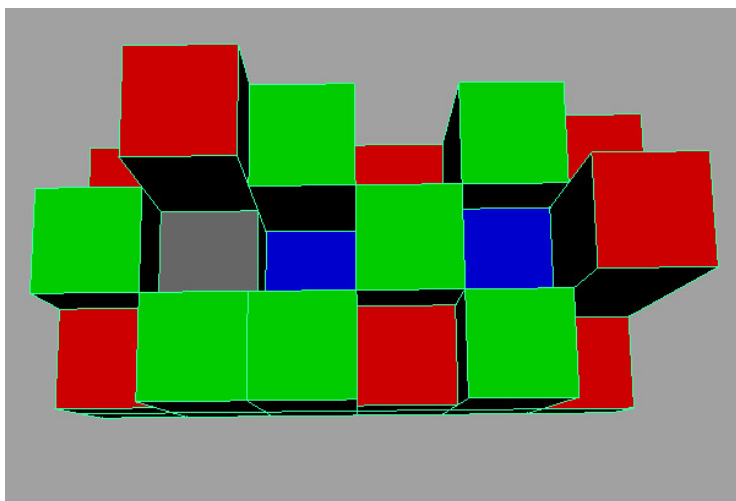
那么优先队列queue中高度为1的格子遍历完了，此时海平面上升1，变为2，此时我们遍历优先队列queue中高度为2的格子，有3个，如下图绿色标记所示：



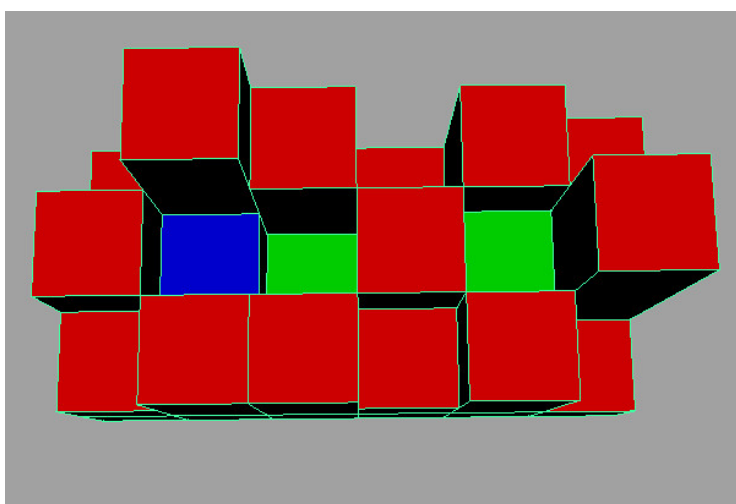
我们发现这三个绿格子周围的格子均已被访问过了，所以不做任何操作，海平面继续上升，变为4，遍历所有高度为4的格子，如下图绿色标记所示：



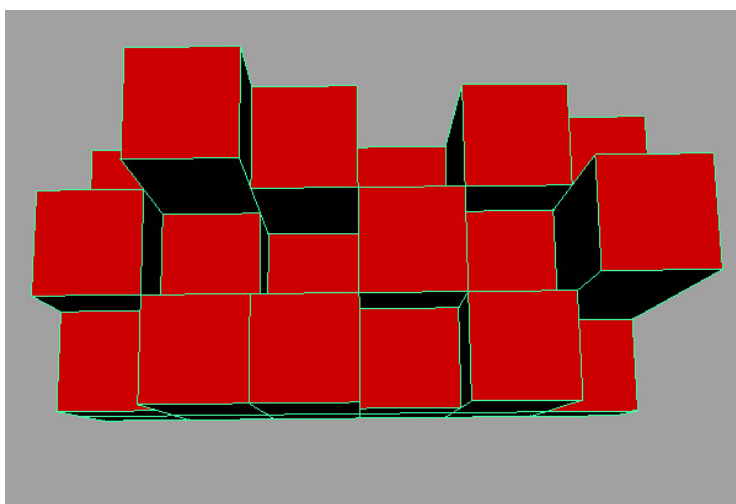
由于我们没有特别声明高度相同的格子在优先队列queue中的顺序，所以应该是随机的，其实谁先遍历到都一样，对结果没啥影响，我们就假设第一行的两个绿格子先遍历到，那么那么周围各有一个灰格子可以遍历，这两个灰格子比海平面低了，可以存水了，把存水量算出来加入结果res中，如下图所示：



上图中这两个遍历到的蓝格子会被加入优先队列queue中，由于它们的高度小，所以下一次从优先队列queue中取格子时，它们会被优先遍历到，那么左边的那个蓝格子进行BFS搜索，就会遍历到其左边的那个灰格子，由于其高度小于海平面，也可以存水，将存水量算出来加入结果res中，如下图所示：



等两个绿格子遍历结束了，它们会被标记为红色，蓝格子遍历会先被标记红色，然后加入优先队列queue中，由于其周围格子全变成红色了，所有不会有任何操作，如下图所示：



此时所有的格子都标记为红色了，海平面继续上升，继续遍历完优先队列queue中的格子，不过已经不会对结果有任何影响了，因为所有的格子都已经访问过了，此时等循环结束后返回res即可，参见代码如下：



```
class Solution {
public:
    int trapRainWater(vector<vector<int>>& heightMap) {
        if (heightMap.empty()) return 0;
        int m = heightMap.size(), n = heightMap[0].size(), res = 0, mx = INT_MIN;
```

```
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q;
vector<vector<bool>> visited(m, vector<bool>(n, false));
vector<vector<int>> dir{{0,-1},{-1,0},{0,1},{1,0}};
for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
        if (i == 0 || i == m - 1 || j == 0 || j == n - 1) {
            q.push({heightMap[i][j], i * n + j});
            visited[i][j] = true;
        }
    }
}
while (!q.empty()) {
    auto t = q.top(); q.pop();
    int h = t.first, r = t.second / n, c = t.second % n;
    mx = max(mx, h);
    for (int i = 0; i < dir.size(); ++i) {
        int x = r + dir[i][0], y = c + dir[i][1];
        if (x < 0 || x >= m || y < 0 || y >= n || visited[x][y]) continue;
        visited[x][y] = true;
        if (heightMap[x][y] < mx) res += mx - heightMap[x][y];
        q.push({heightMap[x][y], x * n + y});
    }
}
return res;
};
```



类似题目：

[Trapping Rain Water](#)

参考资料：

https://discuss.leetcode.com/topic/60914/concise-c-priority_queue-solution

[LeetCode All in One 题目讲解汇总\(持续更新中...\)](#)

分类: [LeetCode](#)

好文要顶

关注我

收藏该文



Grandyang

关注 - 29

粉丝 - 232

3

0

+加关注

« 上一篇: [\[LeetCode\] Queue Reconstruction by Height 根据高度重建队列](#)

» 下一篇: [\[LeetCode\] Valid Word Abbreviation 验证单词缩写](#)

posted @ 2016-10-03 12:32 Grandyang 阅读(4016) 评论(4) 编辑 收藏

评论列表

#1楼 2016-12-19 14:59 Joker_88

非常感谢！

支持(0) 反对(0)

#2楼 2016-12-29 00:43 perthblank

借宝地替自己和没看明白的同学再详细解释一下，为什么
if (heightMap[x][y] < mx) res += mx - heightMap[x][y];
这一行 work..

这一行的条件是访问到了某一个没visit的块高度为 heightMap[x][y]，并且此时的全局水位mx比之高，结果是此块能积水 mx - heightMap[x][y]

1. 为什么能积水，即当前这一块不会通过比自己更矮的块流出去：因为遍历开始讲边缘的块都放入了优先队列，因此如果对于当前块，存在一条通向边缘的路径使得路径上的点高度都<当前块高度，那一定在水位上升到比它高之前被遍历到，与条件冲突

2. 为什么积水不会 $< mx - \text{heightMap}[x][y]$: 即为什么不存在一个高度 $h < mx$, $h > \text{heightMap}[x][y]$, 使得在 x, y 处 $> h$ 的水会流出。如果存在, 那这个 h 所在块将存在一条到边缘的路径使得路径上的所有块高度都 $< h$ (否则无法流出)。根据优先队列以及BFS, h 一定可以在水位上升到 $> h$ 的高度之前被访问, 使得访问到 h 的时候水位也为 h 。同样, 也存在一条从 h 所在块到当前块的路径使得路径上所有块高度都 $< h$ 且 $> \text{heightMap}[x][y]$, 使得水能够从 h 块流出。那么在水位为 h 的时候, 当前块就会被访问到, 当水位上升至 $mx > h$ 时, 当前块已经访问了, 与条件冲突

3. 为什么积水不会 $> mx - \text{heightMap}[x][y]$, 即设当前水位 $mx = h1$ 为围绕在当前块周围的高度, 为什么不会在当前 mx 对应边缘的外层存在一层更高 $h2$ 的边缘, 使得在当前块上能积水 $h2 > h1 = mx$: 如果存在这种情况, 那么访问到 $h2$ 边缘层的时候, mx 已经上升到 $h2$ 的高度, 也就是访问到当前块时, $mx = h2 > h1$, 与条件冲突

支持(1) 反对(0)

#3楼[楼主] 2016-12-29 01:46 Grandyang

@ perthblank
写的不错, 赞一个~

支持(0) 反对(0)

#4楼 2017-03-29 17:40 fgvity

@ perthblank
首先感谢2楼的兄弟
在2楼兄弟给出的解释的基础上, 我想到了一个更通俗的解释。
每次更新 mx 就好比把边界外面的水灌到高度为 $mx + 0.1$ (比 mx 高一点点, 让水能淹过高度为 mx 的块)
队列里面存的块呢, 就是那些能碰到水的块 (被打湿了), 也就是图中红色的块

这样一来, 每当 mx 上升, 新的被打湿的块的水一定是通过那些已经被打湿的, 并且高度为 mx 的块流过去的, 那么通过BFS就能更新队列。

并且如果一个块被打湿了, 同时高度还小于 mx , 那么就说明它周围一定有高度为 mx 的块当边界, 这样才能保证它能够撑到现在才被打湿 (如果没有那么高的边界的话, 水早就应该把它淹过去了, 因为它的高度是小于 mx 的)

既然有 mx 的高度的边界, 那么这一块能存的水就是 $mx - \text{height}$ 了

而且它也不能存更多的水了, 因为假如我们把这整个玩意从水里捞出来, 再假如这一块存的水竟然比 $mx - \text{height}$ 多, 那就代表这一块有比 mx 还高的边界, 那就代表它能够撑得更久, 那他就不应该在 mx 的时候被打湿。

所以在 mx 时被打湿的块能存的水就是 $\max(0, mx - \text{height})$

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问网站首页](#)。

- 【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】腾讯云上实验室 1小时搭建人工智能应用
- 【推荐】可嵌入您系统的“在线Excel”! SpreadJS 纯前端表格控件
- 【推荐】阿里云“全民云计算”优惠升级



美团云
GPU 云主机 5折起

M60 原价1660/月 现价830/月 P40 原价3400/月 现价2100/月

查看详情

最新IT新闻:

- 马云蔡崇信拟减持总计约40亿美元阿里股票
 - 贾跃亭建信托给女儿留5亿元? 乐视回应: 全部造假
 - 曾经被投资人嘲讽为“最笨CEO”, 如今公司估值超过10亿
 - 专访魅蓝总裁李楠: 魅蓝要开线下店 还要做成潮牌
 - 微软Edge浏览器达到新的里程碑 登陆3.3亿台设备
- » 更多新闻...



极光统计 多维洞察用户增长运营指标

最新知识库文章:

- [Google 及其云智慧](#)
 - [做到这一点，你也可以成为优秀的程序员](#)
 - [写给立志做码农的大学生](#)
 - [架构腐化之谜](#)
 - [学会思考，而不只是编程](#)
- » [更多知识库文章...](#)

历史上的今天:

2015-10-03 [CareerCup] 10.2 Data Structures for Large Social Network 大型社交网站的数据结构

Copyright ©2017 Grandyang