LeetCode 1389. 按既定顺序创建目标数组

1389. 按既定顺序创建目标数组

难度 简单 2

给你两个整数数组 `nums` 和 `index`。你需要按照以下规则创建目标数组：

- 目标数组 `target` 最初为空。
- 按从左到右的顺序依次读取 `nums[i]` 和 `index[i]`，在 `target` 数组中的下标 `index[i]` 处插入值 `nums[i]`。
- 重复上一步，直到在 `nums` 和 `index` 中都没有要读取的元素。

请你返回目标数组。

题目保证数字插入位置总是存在。

示例 1：

输入： nums = [0,1,2,3,4], index = [0,1,2,2,1]
输出： [0,4,1,3,2]
解释：

nums	index	target
0	0	[0]
1	1	[0,1]
2	2	[0,1,2]
3	2	[0,1,3,2]
4	1	[0,4,1,3,2]

示例 2：

输入： nums = [1,2,3,4,0], index = [0,1,2,3,0]
输出： [0,1,2,3,4]
解释：

nums	index	target
1	0	[1]
2	1	[1,2]
3	2	[1,2,3]
4	3	[1,2,3,4]
0	0	[0,1,2,3,4]

手动自己实现

```
/*
如果有数就把原来有数的位置空出来
1.把indx[i] 到最后都后移一位
2.插入
*/
class Solution {
```

```
public:
    vector<int> createTargetArray(vector<int>& nums, vector<int>& index) {
        vector<int> target;

        for (int i = 0; i < nums.size(); i ++ ) {
            target.push_back(0);//插入一个位置防止数组越界
            for (int j = target.size() - 1; j > index[i]; j -- )
                target[j] = target[j - 1];//后移
            target[index[i]] = nums[i];//插入
        }

        return target;
    }
};
```

用insert函数

```
class Solution {
public:
    vector<int> createTargetArray(vector<int>& nums, vector<int>& index) {
        vector<int> target;

        for (int i = 0; i < nums.size(); i ++ )
            target.insert(target.begin() + index[i], nums[i]);//insert(插入的位置，数据)

        return target;
    }
};
```

LeetCode 1390. 四因数

1390. 四因数

难度 中等 6

给你一个整数数组 `nums`，请你返回该数组中恰有四个因数的这些整数的各因数之和。

如果数组中不存在满足题意的整数，则返回 `0` 。

示例：

```
输入：nums = [21,4,7]
输出：32
解释：
21 有 4 个因数：1, 3, 7, 21
4 有 3 个因数：1, 2, 4
7 有 2 个因数：1, 7
答案仅为 21 的所有因数的和。
```

提示：

- `1 <= nums.length <= 10^4`
- `1 <= nums[i] <= 10^5`

```cpp
class Solution {
public:
    int sumFourDivisors(vector<int>& nums) {
        int res = 0;
        for (auto x : nums) {
            int sum = 0, cnt = 0;
            for (int i = 1; i * i <= x; i ++ ) {
                if (x % i == 0) {
                    sum += i, cnt ++ ;
                    if (x / i != i) sum += x / i, cnt ++ ;//另外一个约数不一样，就加上！！！
                }
            }

            if (cnt == 4) res += sum;
        }

        return res;
    }
};
```

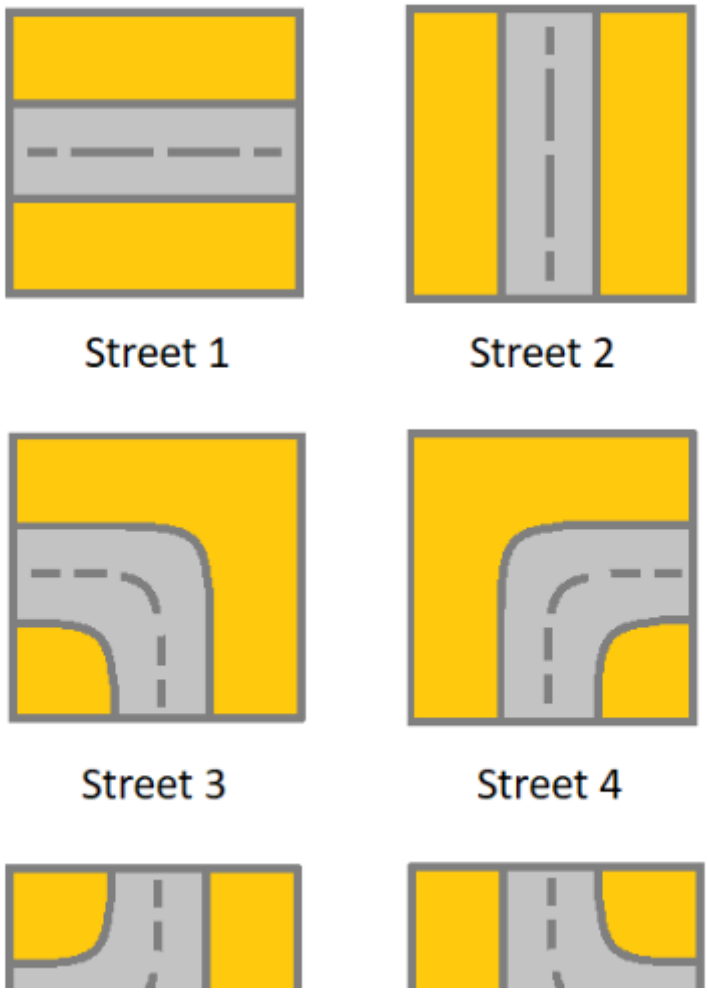
LeetCode 1391. 检查网格中是否存在有效路径

1391. 检查网格中是否存在有效路径

难度 中等 12

给你一个 *m* x *n* 的网格 `grid` 。网格里的每个单元都代表一条街道。`grid[i][j]` 的街道可以是：

- **1** 表示连接左单元格和右单元格的街道。
- **2** 表示连接上单元格和下单元格的街道。
- **3** 表示连接左单元格和下单元格的街道。
- **4** 表示连接右单元格和下单元格的街道。
- **5** 表示连接左单元格和上单元格的街道。
- **6** 表示连接右单元格和上单元格的街道。



Street 5

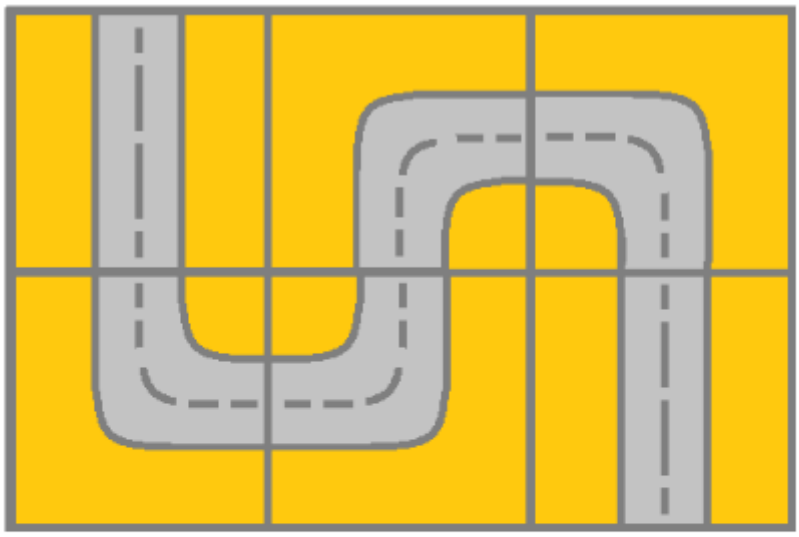
Street 6

你最开始从左上角的单元格 `(0,0)` 开始出发，网格中的「有效路径」是指从左上方的单元格 `(0,0)` 开始、一直到右下方的 `(m-1,n-1)` 结束的路径。**该路径必须只沿着街道走。**

注意：你 **不能** 变更街道。

如果网格中存在有效的路径，则返回 `true`，否则返回 `false` 。

示例 1：



输入： `grid = [[2,4,3],[6,5,2]]`
输出： `true`
解释： 如图所示，你可以从 `(0, 0)` 开始，访问网格中的所有单元格并到达 `(m - 1, n - 1)` 。

```
/*
上0右1下2右3
1：有道
1: 0 1 0 1
2: 1 0 1 0
3: 0 0 1 1
4: 0 1 1 0
5: 1 0 0 1
6: 1 1 0 0
判断能不能走过去
1.(x,y)街道的i方向是否为1
2.(x,y)下一个街道i方向的反方向是否为1
0 2是一对反方向---(第二位不一样)
1 3是一对反方向---(第二位不一样)
用 i ^ 2 看第二位
```

```
*/
class Solution {
public:
    bool hasValidPath(vector<vector<int>>& grid) {
        return dfs(0, 0, grid);
    }

    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    int state[6][4] = {
        {0, 1, 0, 1},
        {1, 0, 1, 0},
        {0, 0, 1, 1},
        {0, 1, 1, 0},
        {1, 0, 0, 1},
        {1, 1, 0, 0},
    };

    bool dfs(int x, int y, vector<vector<int>>& grid) {
        int n = grid.size(), m = grid[0].size();
        if (x == n - 1 && y == m - 1) return true;

        int k = grid[x][y];
        grid[x][y] = 0;

        for (int i = 0; i < 4; i ++ ) {
            int a = x + dx[i], b = y + dy[i];
            if (a < 0 || a >= n || b < 0 || b >= m || !grid[a][b]) continue;
            if (!state[k - 1][i] || !state[grid[a][b] - 1][i ^ 2]) continue;

            if (dfs(a, b, grid)) return true;
        }

        return false;
    }
};
```

LeetCode 1392. 最长快乐前缀

1392. 最长快乐前缀

难度 困难 13

「快乐前缀」是在原字符串中既是 **非空** 前缀也是后缀（不包括原字符串自身）的字符串。

给你一个字符串 `s`，请你返回它的 **最长快乐前缀**。

如果不存在满足题意的前缀，则返回一个空字符串。

示例 1：

> **输入：** s = "level"
> **输出：** "l"
> **解释：** 不包括 s 自己，一共有 4 个前缀（"l", "le", "lev", "leve"）和 4 个后缀（"l", "el", "vel", "evel"）。最长的既是前缀也是后缀的字符串是 "l" 。

示例 2：

> **输入：** s = "ababab"
> **输出：** "abab"
> **解释：** "abab" 是最长的既是前缀也是后缀的字符串。题目允许前后缀在原字符串中重叠。

示例 3：

> **输入：** s = "leetcodeleet"
> **输出：** "leet"

示例 4：

> **输入：** s = "a"
> **输出：** ""

```cpp
/*
非平凡最长和前缀相同的后缀
KMP
*/
class Solution {
public:
    string longestPrefix(string s) {
        int n = s.size();
        s = ' ' + s;//占位，使下标变成从1开始
        vector<int> next(n + 1);

        for (int i = 2, j = 0; i <= n; i ++ ) {
            while (j && s[i] != s[j + 1]) j = next[j];
            if (s[i] == s[j + 1]) j ++ ;
            next[i] = j;
        }
```

```
        return s.substr(1, next[n]);
    }
};
```