1441. 用栈操作构建数组

1441. 用栈操作构建数组

难度 简单 2 收藏 分享 切换为英文 关注 反馈

给你一个目标数组 `target` 和一个整数 `n`。每次迭代，需要从 `list = {1,2,3..., n}` 中依序读取一个数字。

请使用下述操作来构建目标数组 `target` ：

- **Push**：从 `list` 中读取一个新元素， 并将其推入数组中。
- **Pop**：删除数组中的最后一个元素。
- 如果目标数组构建完成，就停止读取更多元素。

题目数据保证目标数组严格递增，并且只包含 `1` 到 `n` 之间的数字。

请返回构建目标数组所用的操作序列。

题目数据保证答案是唯一的。

示例 1：

```
输入：target = [1,3], n = 3
输出：["Push","Push","Pop","Push"]
解释：
读取 1 并自动推入数组 -> [1]
读取 2 并自动推入数组，然后删除它 -> [1]
读取 3 并自动推入数组 -> [1,3]
```

示例 2：

```
输入：target = [1,2,3], n = 3
输出：["Push","Push","Push"]
```

示例 3：

```
输入：target = [1,2], n = 4
输出：["Push","Push"]
解释：只需要读取前 2 个数字就可以停止。
```

```cpp
class Solution {
public:
    vector<string> buildArray(vector<int>& target, int n) {
        vector<string> ans;

        int i = 0, j = 1;
        while (i < target.size()) {
            ans.push_back("Push");
            if (target[i] == j) i++;
            else ans.push_back("Pop");
            j++;
        }
```

```
        return ans;
    }
};
```

1442. 形成两个异或相等数组的三元组数目

1442. 形成两个异或相等数组的三元组数目

难度 中等 8 收藏 分享 切换为英文 关注 反馈

给你一个整数数组 `arr` 。

现需要从数组中取三个下标 `i` 、 `j` 和 `k` ，其中 `(0 <= i < j <= k < arr.length)` 。

`a` 和 `b` 定义如下：

- `a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]`
- `b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]`

注意：^ 表示 **按位异或** 操作。

请返回能够令 `a == b` 成立的三元组 `(i , j , k)` 的数目。

示例 1：

输入： `arr = [2,3,1,6,7]`
输出： `4`
解释： 满足题意的三元组分别是 `(0,1,2)`, `(0,2,2)`, `(2,3,4)` 以及 `(2,4,4)`

示例 2：

输入： `arr = [1,1,1,1,1]`
输出： `10`

示例 3：

输入： `arr = [2,3]`
输出： `0`

示例 4：

输入： `arr = [1,3,5,7,9]`
输出： `3`

```
class Solution {
public:
    int countTriplets(vector<int>& arr) {
        int n = arr.size();
        vector<int> s(n + 1);
        s[0] = 0;

        for (int i = 1; i <= n; i++)
            s[i] = s[i - 1] ^ arr[i - 1];

        int ans = 0;
```

```
        for (int i = 1; i <= n; i++)
            for (int j = i + 1; j <= n; j++)
                for (int k = j; k <= n; k++)
                    if (s[j - 1] ^ s[i - 1] == s[k] ^ s[j - 1])
                        ans++;

        return ans;
    }
};
```

1443. 收集树上所有苹果的最少时间

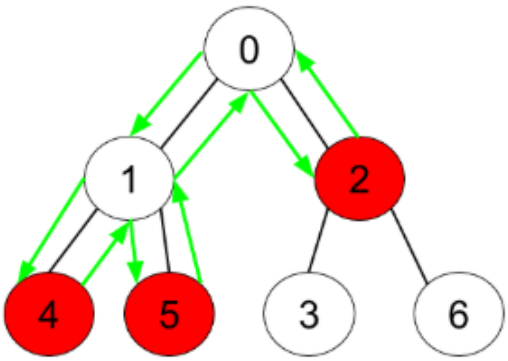
1443. 收集树上所有苹果的最少时间

难度 中等 15 收藏 分享 切换为英文 关注 反馈

给你一棵有 `n` 个节点的无向树，节点编号为 `0` 到 `n-1` ，它们中有一些节点有苹果。通过树上的一条边，需要花费 1 秒钟。你从 **节点 0** 出发，请你返回最少需要多少秒，可以收集到所有苹果，并回到节点 0 。

无向树的边由 `edges` 给出，其中 `edges[i] = [from_i, to_i]` ，表示有一条边连接 `from` 和 `to_i` 。除此以外，还有一个布尔数组 `hasApple` ，其中 `hasApple[i] = true` 代表节点 `i` 有一个苹果，否则，节点 `i` 没有苹果。

示例 1：



输入： `n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,true,true,false]`
输出： `8`
解释： 上图展示了给定的树，其中红色节点表示有苹果。一个能收集到所有苹果的最优方案由绿色箭头表示。

```
class Solution {
public:
    vector<vector<int>> tree;

    bool solve(int u, int fa, vector<bool> &hasApple, int &ans) {
        for (int v : tree[u])
            if (v != fa) {
                if (solve(v, u, hasApple, ans)) {
                    hasApple[u] = true;
                    ans += 2;
                }
            }
```

```
        return hasApple[u];
    }

    int minTime(int n, vector<vector<int>>& edges, vector<bool>& hasApple) {
        tree.resize(n);

        for (const auto &e : edges) {
            tree[e[0]].push_back(e[1]);
            tree[e[1]].push_back(e[0]);
        }

        int ans = 0;
        solve(0, -1, hasApple, ans);

        return ans;
    }
};
```

1444. 切披萨的方案数

1444. 切披萨的方案数

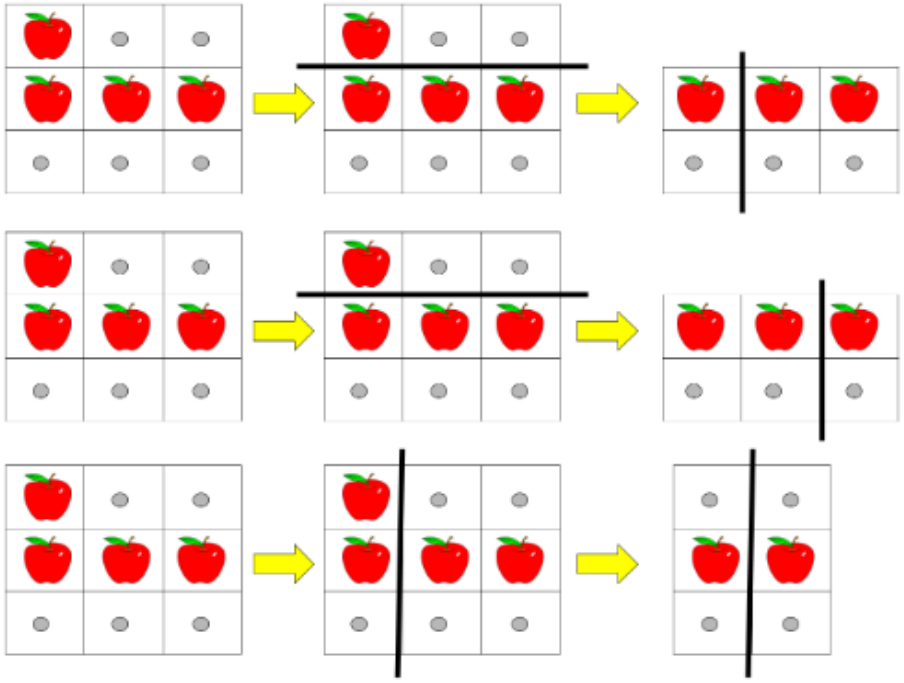
难度 困难 11 收藏 分享 切换为英文 关注 反馈

给你一个 `rows x cols` 大小的矩形披萨和一个整数 `k` ，矩形包含两种字符： `'A'` （表示苹果）和 `'.'` （表示空白格子）。你需要切披萨 `k-1` 次，得到 `k` 块披萨并送给别人。

切披萨的每一刀，先要选择是向垂直还是水平方向切，再在矩形的边界上选一个切的位置，将披萨一分为二。如果垂直地切披萨，那么需要把左边的部分送给一个人，如果水平地切，那么需要把上面的部分送给一个人。在切完最后一刀后，需要把剩下来的一块送给最后一个人。

请你返回确保每一块披萨包含 **至少** 一个苹果的切披萨方案数。由于答案可能是个很大的数字，请你返回它对 10^9 + 7 取余的结果。

示例 1：



输入： `pizza = ["A..","AAA","..."], k = 3`
输出： `3`
解释： 上图展示了三种切披萨的方案。注意每一块披萨都至少包含一个苹果。