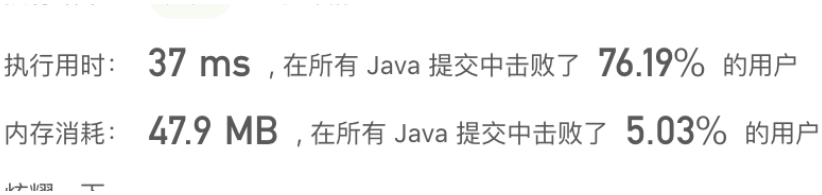
901 Online Stock Span




```java
public int[] dp;
public int[] nums;
public int index;
public StockSpanner() {
    dp = new int[110000];
    Arrays.fill(dp, 1);
    nums = new int[110000];
    index = 0;
}

public int next(int price) {
    nums[index] = price;

    //get res from dp
    int i = index - 1;
    while(i >= 0){
        if(nums[i] <= nums[index]){
            dp[index] += dp[i];
            i = i - dp[i];
        }else{
            break;
        }
    }

    index++;
    return dp[index - 1];
}

```

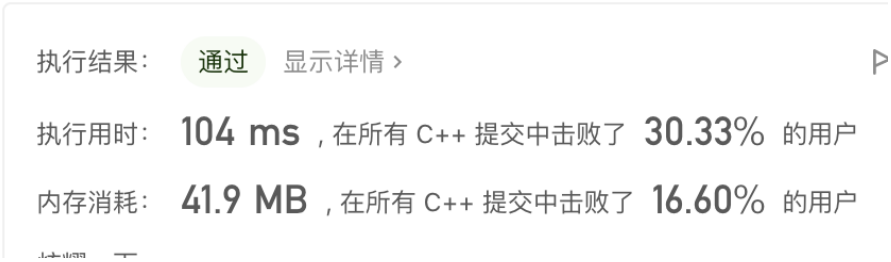
905 Sort Array By Parity

```go
func sortArrayByParity(A []int) []int {
  left := 0
  right := len(A) - 1

  for ; left < right; {
    for ; left < right && A[left] % 2 == 0 ; left++{}

    if left >= right{
      break
    }

    for ; left < right && A[right] % 2 != 0 ; right--{}

    if left >= right{
      break
    }

    A[left], A[right] = A[right], A[left]
    left++
    right--
  }

  return A
}
```

```java
   public int[] sortArrayByParity(int[] A) {
        int len = A.length;
        int left = 0, right = len - 1;
        while(left < right){
            while(left < len && A[left] % 2 == 0)
                left++;

            if(left >= right)
                break;

            while(right >= 0 && A[right] % 2 != 0)
                right--;

            if(left >= right)
                break;

            int temp = A[left];
            A[left]  = A[right];
            A[right] = temp;

            left++;
```

```
            right--;
        }

        return A;
    }
```

907 Sum of Subarray Minimum




```
/*
    author: guoguo

    本质上这道题目， 就是采用单调栈， 拿到 当前位置 i， 往左能扩展到哪里，往右能扩展到哪里
    如果采用 ON2 遍历 比较费时间
    因此 这里采用 单调栈， 找到左边最近的 不小于自己的
                          找到右边最近的，不小于自己的
*/
class Solution {
public:
    int sumSubarrayMins(vector<int>& arr) {
        int size = arr.size();
        long long res = 0;
        int mod = 1e9 + 7;

        vector<int> leftMin(size, -1);
        vector<int> rightMin(size, -1);
        stack<int> monoStack;

        for(int i = 0; i < size; i++){
            if(i == 0)
                monoStack.push(i);
            else{
                if(monoStack.empty() || arr[monoStack.top()] <= arr[i])
                    monoStack.push(i);
                else{
                    while(!monoStack.empty() && arr[monoStack.top()] > arr[i]){
                        leftMin[i] = monoStack.top();
                        if(leftMin[monoStack.top()] != -1)
                            leftMin[i] = leftMin[monoStack.top()];
```

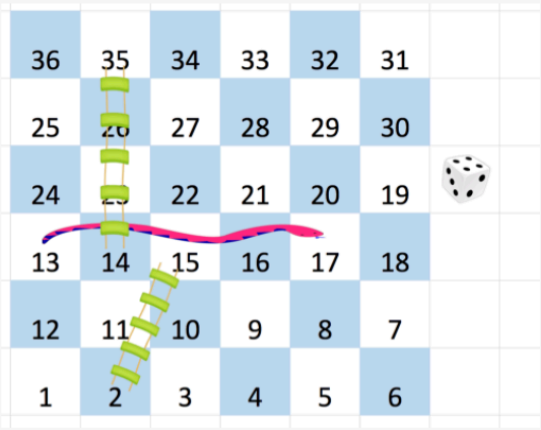
```
                        monoStack.pop();
                    }

                    monoStack.push(i);
                }
            }
        }

        monoStack = stack<int>();

        for(int i = size - 1; i >= 0; i--){
            if(i == size - 1)
                monoStack.push(i);
            else{
                if(monoStack.empty() || arr[monoStack.top()] < arr[i])
                    monoStack.push(i);
                else{
                    while(!monoStack.empty() && arr[monoStack.top()] >= arr[i]){
                        rightMin[i] = monoStack.top();
                        if(rightMin[monoStack.top()] != -1)
                            rightMin[i] = rightMin[monoStack.top()];
                        monoStack.pop();
                    }


                    monoStack.push(i);
                }
            }
        }

        for(int i = 0; i < size; i++){
            int left;
            if(leftMin[i] == -1)
                left = 1;
            else
                left = i - leftMin[i] + 1;
            int right;

            if(rightMin[i] == -1)
                right = 1;
            else
                right= rightMin[i] - i + 1;

            res += ((long)left * (long)right) * (long)arr[i];
        }

        return res % mod;
    }
};
```

909 Snakes and Lakes

909. Snakes and Ladders

难度 中等 39

On an N x N `board`, the numbers from `1` to `N*N` are written *boustrophedonically* **starting from the bottom left of the board**, and alternating direction each row. For example, for a 6 x 6 board, the numbers are written as follows:



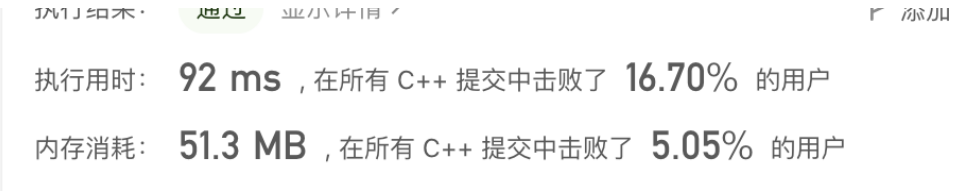
You start on square `1` of the board (which is always in the last row and first column). Each move, starting from square `x`, consists of the following:

- You choose a destination square `S` with number `x+1`, `x+2`, `x+3`, `x+4`, `x+5`, or `x+6`, provided this number is `<= N*N`.
  - (This choice simulates the result of a standard 6-sided die roll: ie., there are always **at most 6 destinations, regardless of the size of the board**.)
- If `S` has a snake or ladder, you move to the destination of that snake or ladder. Otherwise, you move to `S`.

A board square on row `r` and column `c` has a "snake or ladder" if `board[r][c] != -1`. The destination of that snake or ladder is

```
1
```

918 Maximum Sum Circular Subarray




```
1
2  class Solution {
3  public:
4      int maxSubarraySumCircular(vector<int>& nums) {
5          int size= nums.size();
6          int doubleSize = 2 * size;
7          int temp = nums[0];
8
9          vector<int> prefixSum(doubleSize, 0);
10         for(int i = 0; i < doubleSize; i++){
11             int pos = i % size;
12             if(i == 0)
13                 prefixSum[i] = nums[i];
14             else
15                 prefixSum[i] += prefixSum[i - 1] + nums[pos];
16
17             temp = nums[pos] > temp ? nums[pos] : temp;
18         }
19
20         int index = doubleSize - 1;
21         deque<pair<int, int>> maxStack;
22         vector<int> lenMax(size + 1, 0);
23         while(index >= 0){
24             if(index >= size){
25                 while(!maxStack.empty() && maxStack.back().first <
   prefixSum[index])
26                     maxStack.pop_back();
27
28                 if(maxStack.empty() || maxStack.back().first >= prefixSum[index])
29                     maxStack.emplace_back(prefixSum[index], index);
30
31                 index--;
32             }else{
33                 if(!maxStack.empty() && maxStack.front().second == index + size)
34                     maxStack.pop_front();
35
36                 int curMax = maxStack.front().first;
37                 lenMax[index] = curMax;
38
39                 while(!maxStack.empty() && maxStack.back().first <
   prefixSum[index])
```

```
                    maxStack.pop_back();

                maxStack.emplace_back(prefixSum[index], index);

                index--;
            }
        }

        int res = prefixSum[size - 1];
        for(int i = 0; i < size; i++){
            if(res < lenMax[i] - prefixSum[i])
                res = lenMax[i] - prefixSum[i];
        }



        return res > temp ? res : temp;
    }
};
```

922 Sort Array By Parity II

执行用时： 28 ms ，在所有 Go 提交中击败了 52.94% 的用户

内存消耗： 6.2 MB ，在所有 Go 提交中击败了 80.88% 的用户

```
func sortArrayByParityII(nums []int) []int {
  even, odd := 0, 1

  for ;odd < len(nums) && even < len(nums);{
    for;odd < len(nums) && nums[odd] % 2 != 0;{
      odd += 2
    }

    for ;even < len(nums) && nums[even] % 2 == 0;{
      even += 2
    }

    if odd >= len(nums) || even >= len(nums){
      break
```

```
    }

    nums[odd],nums[even] = nums[even], nums[odd]
    odd  += 2
    even += 2
  }

  return nums
}
```

937 Reorder Data in Log Files

```
    public String[] reorderLogFiles(String[] logs) {
        List<String> nums = new ArrayList<>();
        List<String> alpha = new ArrayList<>();

        for(int i = 0 ; i < logs.length; i++){
            String log = logs[i];

            int pos = log.indexOf(" ");
            char ch  = log.charAt(pos + 1);
            if('0' <= ch && ch <= '9')
                nums.add(log);
            else
                alpha.add(log);
        }

        alpha.sort(new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                int pos1 = o1.indexOf(" ");
                int pos2 = o2.indexOf(" ");

                String sub1 = o1.substring(pos1 + 1);
                String sub2 = o2.substring(pos2 + 1);

                if (sub1.compareTo(sub2) == 0) {
                    return o1.substring(0, pos1).compareTo(o2.substring(0, pos2));
                } else {
                    return sub1.compareTo(sub2);
```

```
                }
            }
        });

        String[] res = new String[logs.length];
        for(int i = 0; i < logs.length; i++){
            if(i < alpha.size())
                res[i] = alpha.get(i);
            else
                res[i] = nums.get(i - alpha.size());
        }

        return res;
    }
```

934 Shortest Bridge

执行用时： 236 ms ，在所有 C++ 提交中击败了 5.32% 的用户

内存消耗： 29.6 MB ，在所有 C++ 提交中击败了 8.56% 的用户

```
class Solution {
public:
    vector<vector<int>> dir{{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
    int row, col;
    int shortestBridge(vector<vector<int>>& grid) {
        row = grid.size();
        col = grid[0].size();

        queue<pair<int, int>> myQueue1;
        queue<pair<int, int>> myQueue2;

        unordered_set<string> set1;
        unordered_set<string> set2;
        bool jump = false;

        for(int i = 0; i < row && !jump; i++){
            for(int j = 0; j < col && !jump; j++){
                if(grid[i][j] == 1){
                    dfs(grid, i, j);
                    jump = !jump;
                }
            }
        }
```

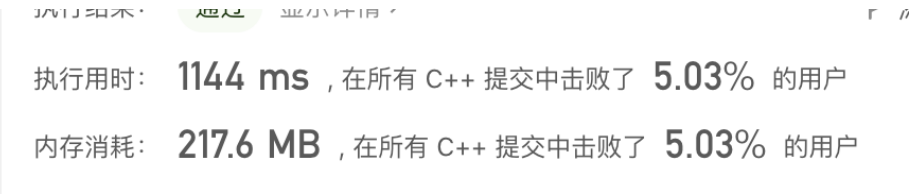
```
        // island 1 and island 2
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 1){
                    myQueue1.push({i, j});
                    set1.insert(to_string(i) + "@" + to_string(j));
                }else if(grid[i][j] == 2){
                    myQueue2.push({i, j});
                    set2.insert(to_string(i) + "@" + to_string(j));
                }
            }
        }

        if(myQueue2.size() == 0 || myQueue1.size() == 0)
            return 0;

        int count = 0;
        while(true){
            int size1 = myQueue1.size();
            for(int i = 0; i < size1; i++){
                pair<int, int> curPos1 = myQueue1.front(); myQueue1.pop();

                for(int k = 0; k < 4; k++){
                    int newX = curPos1.first + dir[k][0];
                    int newY = curPos1.second + dir[k][1];
                    string symbol = to_string(newX) + "@" + to_string(newY);

                    if(set2.count(symbol) != 0)
                        return 2 * count;

                    if(isInRange(newX, newY) && set1.count(symbol) == 0){
                        myQueue1.push({newX, newY});
                        set1.insert(symbol);
                    }
                }
            }

            int size2 = myQueue2.size();
            for(int i = 0; i < size2; i++){
                pair<int, int> curPos2 = myQueue2.front(); myQueue2.pop();

                for(int k = 0; k < 4; k++){
                    int newX = curPos2.first  + dir[k][0];
                    int newY = curPos2.second + dir[k][1];
                    string symbol = to_string(newX) + "@" + to_string(newY);

                    if(set1.count(symbol) != 0)
                        return 2 * count + 1;
```

```
                    if(isInRange(newX, newY) && set2.count(symbol) == 0){
                        myQueue2.push({newX, newY});
                        set2.insert(symbol);
                    }
                }
            }

            count++;
        }

        return -1;
    }

    inline bool isInRange(int i ,int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }

    void dfs(vector<vector<int>>& grid, int i, int j){
        grid[i][j] = 2;

        for(int k = 0; k < 4; k++){
            int newX = i + dir[k][0];
            int newY = j + dir[k][1];

            if(isInRange(newX, newY) && grid[newX][newY] == 1){
                dfs(grid, newX, newY);
            }
        }
    }
};

```

935 Knight Dialer

```
ref : https://leetcode-cn.com/problems/knight-dialer/solution/4zhuang-tai-dong-tai-gui-hua-pythonjie-kong-jian-f/
class Solution {
public:
    int mod = 1e9 + 7;
    int knightDialer(int n) {
        if(n == 1)
            return 10;
```

```
        vector<long> dp(4, 1);
        for(int i = 2; i <= n; i++){
            vector<long> newDP(4, 0);
            newDP[0] = (dp[1] % mod + dp[2] % mod) % mod;
            newDP[1] = (2 * dp[0]) % mod;
            newDP[2] = ((2 * dp[0]) % mod + dp[3] % mod) % mod;
            newDP[3] = (2 * dp[2]) % mod;

            swap(dp, newDP);
        }

        return (4 * dp[0] + 2 * dp[1] + 2 * dp[2] + dp[3]) % mod;
    }
};
```



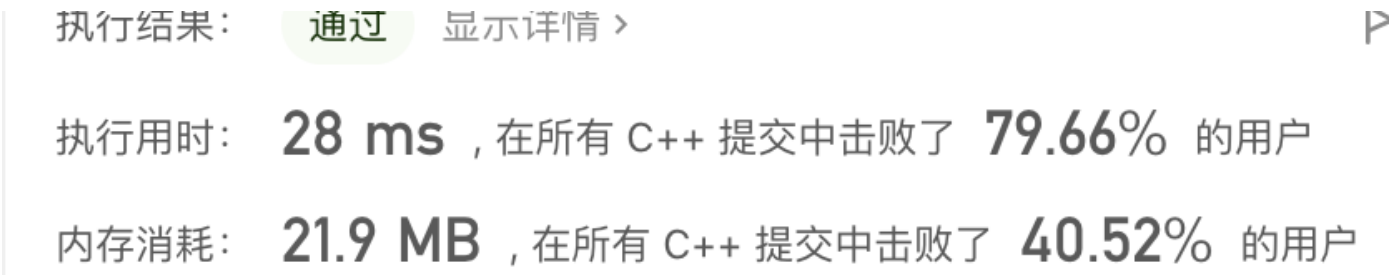

```
class Solution {
public:
    int mod = 1000000007;
    int knightDialer(int n) {
        unordered_map<int, vector<int>> lookup;
        lookup[1] = {6, 8};
        lookup[2] = {7, 9};
        lookup[3] = {4, 8};
        lookup[4] = {3, 9, 0};
        lookup[5] = {};
        lookup[6] = {0, 1, 7};
        lookup[7] = {2, 6};
        lookup[8] = {1, 3};
        lookup[9] = {2, 4};
        lookup[0] = {4, 6};

        unordered_map<int, long> map;
        for(int i = 0; i < 10; i++)
            map[i] = 1;

        for(int i = 2; i <= n; i++){
            unordered_map<int, long> temp;

```

```
            for(int j = 0; j < 10; j++){
                for(int nextDest : lookup[j]){
//                    cout << "in j " << j  << " next dest" << nextDest << "map ->
" << map[j] << endl;
                    temp[nextDest] += map[j];
                    temp[nextDest] %= mod;
                }
            }

            map = temp;
        }

        long res = 0;
        for(int i = 0; i < 10; i++)
            res += map[i];

        return (int)(res % mod);
    }
};
```

938 Range Sum of BST

```
    int sum = 0;
    public int rangeSumBST(TreeNode root, int low, int high) {
        preorder(root, low, high);
        return sum;
    }

    private void preorder(TreeNode root, int low, int high){
        if(root == null)
            return;

        if(root.val >= low && root.val <= high)
            sum += root.val;

        preorder(root.left, low, high);
        preorder(root.right, low, high);
    }
```

941Valid Mountain Array

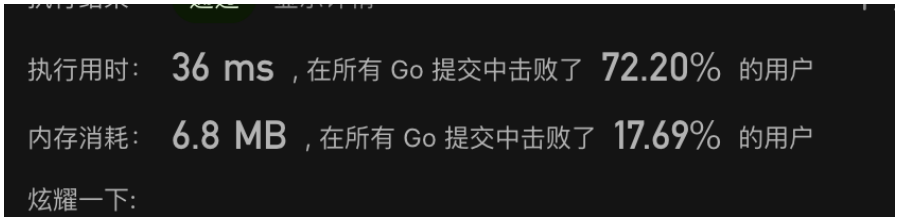



```cpp
class Solution {
public:
    bool validMountainArray(vector<int>& arr) {
        int len = arr.size();
        if(len < 3)
            return false;

        int index = 1;
        while(index < len - 1 && arr[index] > arr[index - 1])
            index++;

        if(index == 1){
            if(arr[index] <= arr[index - 1])
                return false;
        }else if(index == len - 1){
            return arr[index - 1] > arr[index] ;
        }

        while(index < len - 1 && arr[index] > arr[index + 1])
            index++;

        return index == len - 1;
    }
};
```

945 Minimum Increment to Make Array Unique

```cpp
class Solution {
public:
    int minIncrementForUnique(vector<int>& nums) {
        int len = nums.size();

        sort(nums.begin(), nums.end());
        int count = 0;
        int curNum = -1;
        for(int i = 0; i < len; i++){
            if(curNum >= nums[i]){
                count += curNum + 1 - nums[i];
                curNum += 1;
            }else{
                curNum = nums[i];
            }
        }

        return count;
    }
};
```

977 Squares of a Sorted Array




```go
func sortedSquares(nums []int) []int {
    index := 0
    for ; index < len(nums); index++{
        if index > 0 && nums[index] * nums[index - 1] <= 0{
            break
        }
    }

    res := make([]int, 0)

    if index == len(nums){
        if nums[index - 1] > 0{
```

```
            for index = 0; index < len(nums); index++{
                res = append(res, nums[index] * nums[index])
            }
        }else{
            for index = len(nums) - 1; index >= 0; index--{
                res = append(res, nums[index] * nums[index])
            }
        }

    }else{
        left := index - 1
        right := index

        for ;left >= 0 || right < len(nums);{
            if left < 0{
                res = append(res, nums[right] * nums[right])
                right++
            }else if right >= len(nums){
                res = append(res, nums[left] * nums[left])
                left--
            }else if nums[left] * nums[left] < nums[right] * nums[right]{
                res = append(res, nums[left] * nums[left])
                left--
            }else{
                res = append(res, nums[right] * nums[right])
                right++
            }
        }

    }

    return res
}

```

979 Distribute Coins in Binary Tree

```
class Solution {
public:
    int count = 0;
    int distributeCoins(TreeNode* root) {
        postorder(root);
        return count;
```

```
    }

    int postorder(TreeNode* root){
        if(root == nullptr)
            return 0;

        int left = postorder(root->left);
        int right = postorder(root->right);

        int cur = root->val - 1 + left + right;

        count += abs(cur);
        return cur;

    }
};
```

984 String Without AAA

```
/*
    简单的思路，就是当 a和b个数相等的时候，最舒服
    直接 ababababa

    剩下的就是往里面插a & b
    当然a > b 和 b > a 是不同的情况，需要多加注意
*/

class Solution {
public:
    string strWithout3a3b(int a, int b) {
        int minFreq = std::min(a, b);
        string res = "";
        for(int i = 0; i < minFreq; i++) {
            if(a > b)
                res += "ab";
            else
                res += "ba";
        }

        int count = 0;
        if(a > b){
            for(int i = 0; count != a - b && i < res.size();i++){
                if(res[i] == 'b') {
```

```
                    res.insert(res.begin() + i, 'a');
                    count++;
                    i += 2;
                }
            }
        }else if(a < b){
            for(int i = 0; count != b - a && i < res.size(); i++){
                if(res[i] == 'a') {
                    res.insert(res.begin() + i, 'b');
                    count++;
                    i += 2;
                }
            }
        }

        if(count != abs(a - b)){
            for(int i = 0; i < abs(a - b) - count; i++){
                if(a > b)
                    res += "a";
                else
                    res += "b";
            }
        }
        return res;
    }
};

```

```
/*
    更好的 PQ 解法
    ref https://github.com/wisdompeak/LeetCode

    整体思路是 每次从堆中拿两个元素,
    比如 a > b
    那么我就尽量多拿a, 拿2个a 一个b
    res += aab
*/
class Solution {
public:
    string strWithout3a3b(int a, int b) {
        priority_queue<pair<int, int>> pq;
        pq.push({a, 'a'});
        pq.push({b, 'b'});

        string res = "";

```

```
        while(!pq.empty()){
            if(pq.size() == 1){
                int freq = pq.top().first;
                if(freq > 2)
                    return "";
                else{
                    for(int i = 0; i < freq; i++)
                        res.push_back(pq.top().second);
                    return res;
                }
            }

            auto x = pq.top(); pq.pop();
            auto y = pq.top(); pq.pop();

            int k = std::min(1 + x.first - y.first, 2);
            for(int i = 0; i < k; i++)
                res.push_back(x.second);
            res.push_back(y.second);

            x.first -= k;
            y.first -= 1;
            if(x.first > 0)   pq.push({x.first, x.second});
            if(y.first > 0)   pq.push({y.first, y.second});

        }

        return res;
    }
};
```

994 Rotting-Orange

执行结果: 通过 显示详情 >

执行用时: 12 ms , 在所有 C++ 提交中击败了 31.73% 的用户

内存消耗: 12.8 MB , 在所有 C++ 提交中击败了 45.55% 的用户

炫耀一下:

```cpp
//进行优化
class Solution {
public:
    vector<vector<int>> dir{{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
    int row;
    int col;
    int orangesRotting(vector<vector<int>>& grid) {
        row = grid.size();
        col = grid[0].size();

        queue<pair<int, int>> myQueue;
        int fresh = 0;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 2){
                    myQueue.push({i, j});
                    grid[i][j] = -1;
                }else if(grid[i][j] == 1)
                    fresh++;
            }
        }

        if(fresh == 0)
            return 0;
        auto isInRange =  [&](int i, int j){return i >= 0 && j >= 0 && i < row && j < col;};

        int res = 0;
        int round = 0;
        while(!myQueue.empty()){
            int size = myQueue.size();
            for(int i = 0; i < size; i++){
                pair<int, int> curPos = myQueue.front();
                myQueue.pop();

                for(int k = 0; k < dir.size(); k++){
                    int newX = curPos.first + dir[k][0];
                    int newY = curPos.second + dir[k][1];
                    string symbol = to_string(newX) + "@" + to_string(newY);

                    if(isInRange(newX, newY) && grid[newX][newY] == 1){
                        fresh--;

                        if(fresh == 0)
                            return round + 1;
                        grid[newX][newY] = -1;
                        myQueue.push({newX, newY});
                    }
                }
```

```
            }

            round++;
        }

        return fresh == 0 ? round : -1;
    }


};
```

执行结果：通过 显示详情

执行用时：24 ms，在所有 C++ 提交中击败了 9.97% 的用户

内存消耗：13.8 MB，在所有 C++ 提交中击败了 5.99% 的用户

```
class Solution {
public:
    vector<vector<int>> dir{{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
    int row;
    int col;
    int orangesRotting(vector<vector<int>>& grid) {
        row = grid.size();
        col = grid[0].size();

        queue<pair<int, int>> myQueue;
        unordered_set<string> set;
        int fresh = 0;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 2){
                    myQueue.push({i, j});
                    set.emplace(to_string(i) + "@" + to_string(j));
                }else if(grid[i][j] == 1)
                    fresh++;
            }
        }

        if(fresh == 0)
            return 0;

        int res = 0;
        int round = 0;
        while(!myQueue.empty()){
            int size = myQueue.size();
```

```
            for(int i = 0; i < size; i++){
                pair<int, int> curPos = myQueue.front();
                myQueue.pop();

                for(int k = 0; k < dir.size(); k++){
                    int newX = curPos.first + dir[k][0];
                    int newY = curPos.second + dir[k][1];
                    string symbol = to_string(newX) + "@" + to_string(newY);

                    if(isInRange(newX, newY) && set.count(symbol) == 0 &&
grid[newX][newY] != 0){
                        set.emplace(symbol);

                        if(grid[newX][newY] == 1)
                            fresh--;

                        if(fresh == 0)
                            return round + 1;
                        grid[newX][newY] = 2;
                        myQueue.push({newX, newY});
                        set.emplace(symbol);
                    }
                }
            }

            round++;
        }

        return fresh == 0 ? round : -1;
    }

    bool isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
};
```

996 Number of Squareful Array

执行结果： 通过 显示详情

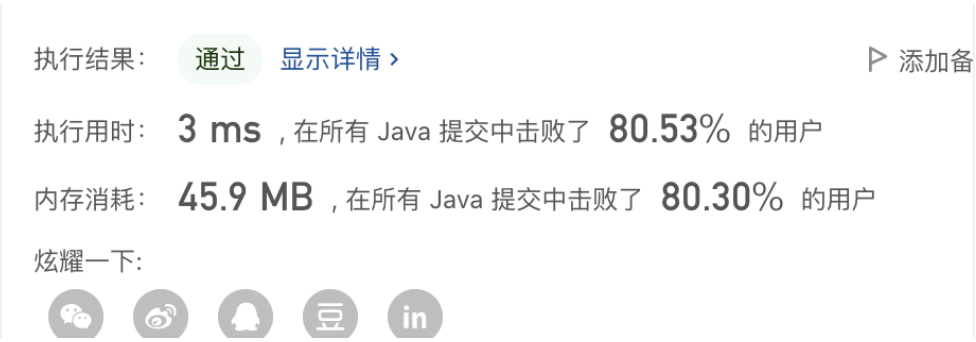
执行用时： 4 ms ，在所有 C++ 提交中击败了 57.86% 的用户

内存消耗： 7.9 MB ，在所有 C++ 提交中击败了 16.07% 的用户

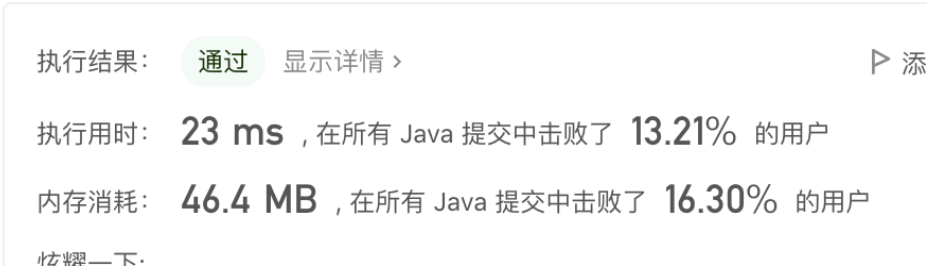
```
/*
    如果不能枚举， 就自己构造
```

```cpp
*/
class Solution {
public:
    int count = 0;
    int numSquarefulPerms(vector<int>& nums) {
        int size = nums.size();
        vector<bool> visited(size, false);
        sort(nums.begin(), nums.end());
        vector<int> path;
        backtrack(nums, visited, path, 0);

        return count;
    }

    void backtrack(vector<int>& nums, vector<bool> visited, vector<int> path,int cur){
        if(cur == nums.size()){
            count++;
            return;
        }

        for(int i = 0; i < visited.size(); i++){
            if(visited[i] || (i > 0 && nums[i] == nums[i - 1] && !visited[i - 1]))
                continue;

            if(cur == 0){
                path.push_back(nums[i]);
                visited[i] = true;

                backtrack(nums, visited, path, cur + 1);

                path.pop_back();
                visited[i] = false;
            }else{
                int sqr = std::sqrt(nums[i] + path[path.size() - 1]);
                if(sqr * sqr == nums[i] + path[path.size() - 1]){
                    path.push_back(nums[i]);
                    visited[i] = true;

                    backtrack(nums, visited, path, cur + 1);

                    path.pop_back();
                    visited[i] = false;
                }
            }

        }
    }
};
```

997 Find the Town Judge




```java
public int findJudge(int n, int[][] trust) {
    int[] in  = new int[n + 1];
    int[] out = new int[n + 1];

    for(int i = 0; i < trust.length; i++){
        int A = trust[i][0];
        int B = trust[i][1];

        in[B]++;
        out[A]++;
    }

    for(int i = 1; i <= n; i++){
        if(in[i] == n - 1 && out[i] == 0)
            return i;
    }

    return -1;
}
```




```java
public int findJudge(int n, int[][] trust) {
    if(n == 1 && trust.length == 0)
        return 1;
```

```
        Map<Integer, Integer> map = new HashMap<>();
        Set<Integer> set = new HashSet<>();

        for(int i = 0; i < trust.length; i++) {
            map.put(trust[i][1], map.getOrDefault(trust[i][1], 0) + 1);
            set.add(trust[i][0]);
        }

        List<Integer> judges = new ArrayList<>();
        for(Integer potentialJudge : map.keySet()){
            if(map.get(potentialJudge) == n - 1 && !set.contains(potentialJudge))
                return potentialJudge;
        }

        return -1;
    }
```