【第三十六课】有趣的莫比乌斯反演

1.solve.cpp

```cpp
/*************************************************************************
  > File Name: 1.solve.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

int is_prime(int x) {
    if (x <= 1) return 0;
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}

int gcd(int a, int b) {
    if (b) return gcd(b, a % b);
    return a;
}

int main() {
    int n, m, ans = 0;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (!is_prime(gcd(i, j))) continue;
            ans += 1;
            cout << i << " " << j << endl;
        }
    }
    cout << "total : " << ans << endl;
    return 0;
```

```
}
```

2.u.cpp

```cpp
/*************************************************************************
  > File Name: 2.u.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

#define MAX_N 10000
int mu[MAX_N + 5] = {0};
int prime[MAX_N + 5] = {0};

void init_prime(int n) {
    mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!prime[i]) {
            prime[++prime[0]] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= prime[0]; j++) {
            if (i * prime[j] > n) break;
            prime[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
            mu[i * prime[j]] = -mu[i];
        }
    }
    return ;
}

int main() {
    int n;
    cin >> n;
    init_prime(n);
    for (int i = 1; i <= n; i++) {
```

```
        cout << "mu[" << i << "] = " << mu[i] << endl;
    }
    return 0;
}
```

1447. 最简分数

给你一个整数 `n` ，请你返回所有 0 到 1 之间（不包括 0 和 1）满足分母小于等于 `n` 的 **最简** 分数 。分数可以以 **任意** 顺序返回。

示例 1：

```
输入：n = 2
输出：["1/2"]
解释："1/2" 是唯一一个分母小于等于 2 的最简分数。
```

```
class Solution {
public:
    int gcd(int a, int b) {
        if (b) return gcd(b, a % b);
        return a;
    }
    vector<string> simplifiedFractions(int n) {
        vector<string> ret;
        for (int a = 1; a < n; a++) {
            for (int b = a + 1; b <= n; b++) {
                if (gcd(a, b) - 1) continue;
                stringstream ss;
                ss << a << "/" << b;
                ret.push_back(ss.str());
            }
        }
        return ret;
    }
};
```

878. 第 N 个神奇数字

如果正整数可以被 A 或 B 整除，那么它是神奇的。

返回第 N 个神奇数字。由于答案可能非常大，**返回它模** `10^9 + 7` **的结果。**

示例 1：

```
输入：N = 1, A = 2, B = 3
输出：2
```

```
class Solution {
```

```
public:
    long long gcd(long long a, long long b) {
        if (b) return gcd(b, a % b);
        return a;
    }
    long long lcm(long long a, long long b) {
        return a * b / gcd(a, b);
    }
    long long f(long long n, long long a, long long b) {
        return n / a + n / b - n / lcm(a, b);
    }
    int nthMagicalNumber(int n, int a, int b) {
        long long l = 1, r = (long long)n * (long long)min(a, b), mid, mod_num = 1e9 + 7;
        while (l < r) {
            mid = (l + r) >> 1;
            if (f(mid, a, b) < n) l = mid + 1;
            else r = mid;
        }
        return l % mod_num;
    }
};
```

372. 超级次方

你的任务是计算 `ab` 对 `1337` 取模，`a` 是一个正整数，`b` 是一个非常大的正整数且会以数组形式给出。

示例 1：

```
输入：a = 2, b = [3]
输出：8
```

```
class Solution {
public:
    int power(int a, int b, int c) {
        int base = a % c, ans = 1;
        while (b) {
            if (b & 1) ans = ans * base % c;
            base = base * base % c;
            b >>= 1;
        }
        return ans;
    }
    int superPow(int a, vector<int>& b) {
        int base = a, ans = 1, mod_num = 1337;
        for (int i = b.size() - 1; i >= 0; --i) {
            ans = ans * power(base, b[i], mod_num) % mod_num;
            base = power(base, 10, mod_num);
```

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

1512. 好数对的数目

给你一个整数数组 `nums` 。

如果一组数字 `(i,j)` 满足 `nums[i]` == `nums[j]` 且 `i` < `j` ，就可以认为这是一组 **好数对** 。

返回好数对的数目。

示例 1：

```
输入：nums = [1,2,3,1,1,3]
输出：4
解释：有 4 组好数对，分别是 (0,3), (0,4), (3,4), (2,5) ，下标从 0 开始
```

```
class Solution {
public:
    int numIdenticalPairs(vector<int>& nums) {
        int ans = 0, cnt[101] = {0};
        for (auto x : nums) {
            ans += cnt[x];
            cnt[x] += 1;
        }
        return ans;
    }
};
```

1359. 有效的快递序列数目

给你 `n` 笔订单，每笔订单都需要快递服务。

请你统计所有有效的 收件/配送 序列的数目，确保第 `i` 个物品的配送服务 `delivery(i)` 总是在其收件服务 `pickup(i)` 之后。

由于答案可能很大，请返回答案对 `10^9 + 7` 取余的结果。

示例 1：

```
输入：n = 1
输出：1
解释：只有一种序列 (P1, D1)，物品 1 的配送服务（D1）在物品 1 的收件服务（P1）后。
```

```cpp
class Solution {
public:
    int countOrders(int n) {
        long long ans = 1, mod_num = 1e9 + 7;
        for (int i = 2; i <= n; i++) {
            ans = ans * (2 * i * i - i) % mod_num;
        }
        return ans;
    }
};
```

60. 排列序列

给出集合 `[1,2,3,...,n]`，其所有元素共有 `n!` 种排列。

按大小顺序列出所有排列情况，并一一标记，当 `n = 3` 时, 所有排列如下：

1. `"123"`
2. `"132"`
3. `"213"`
4. `"231"`
5. `"312"`
6. `"321"`

给定 `n` 和 `k`，返回第 `k` 个排列。

示例 1：

```
输入：n = 3, k = 3
输出："213"
```

```cpp
class Solution {
public:
    int pick(int n, int mark[]) {
        int i = 0, cnt = 0;
        do {
            i += 1;
            cnt += (mark[i] == 0);
        } while (cnt < n);
        mark[i] = 1;
        return i;
    }
    string getPermutation(int n, int k) {
        int base = 1, mark[n + 1];
        memset(mark, 0, sizeof(mark));
        for (int i = 1; i < n; i++) base *= i;
        stringstream ss;
        for (int i = n; i >= 1; --i) {
```

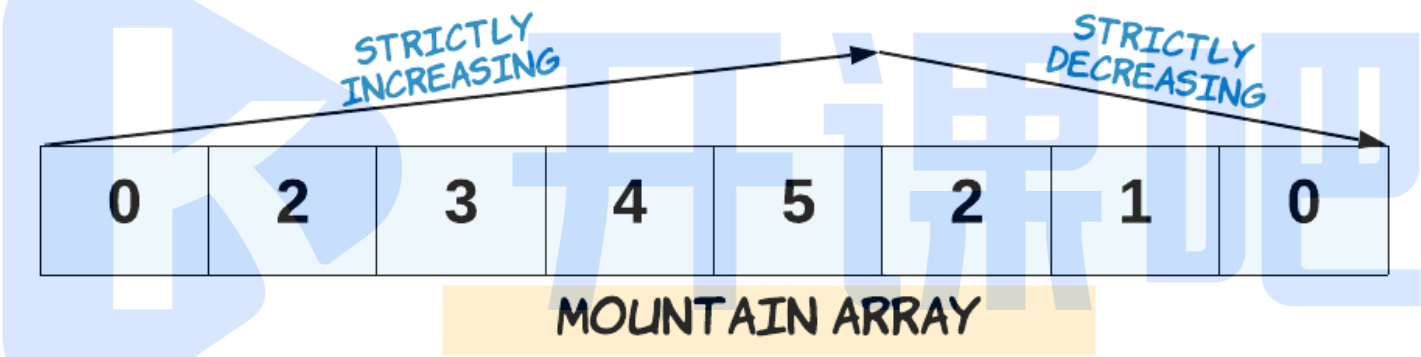
```
            int ind = 1 + (k - 1) / base;
            ss << pick(ind, mark);
            k -= (ind - 1) * base;
            if (i > 1) base /= (i - 1);
        }
        return ss.str();
    }
};
```

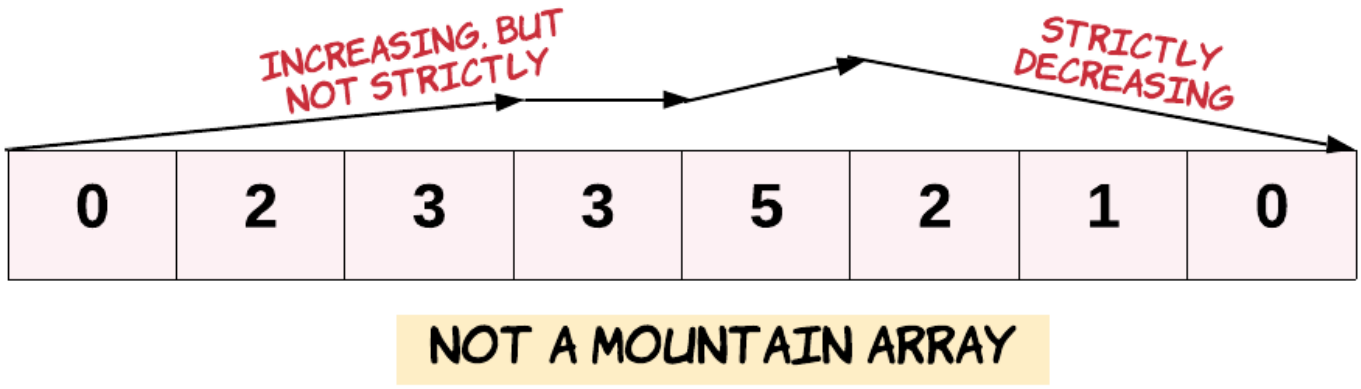
941. 有效的山脉数组

给定一个整数数组 `arr`，如果它是有效的山脉数组就返回 `true`，否则返回 `false`。

让我们回顾一下，如果 A 满足下述条件，那么它是一个山脉数组：

- `arr.length >= 3`
- 在 `0 < i < arr.length - 1` 条件下，存在 `i` 使得：
  - `arr[0] < arr[1] < ... arr[i-1] < arr[i]`
  - `arr[i] > arr[i+1] > ... > arr[arr.length - 1]`





示例 1：

```
输入：arr = [2,1]
输出：false
```

```cpp
class Solution {
public:
    bool validMountainArray(vector<int>& arr) {
        int i = 0, j = arr.size() - 1;
        while (i < j && arr[i] < arr[i + 1]) ++i;
        while (i < j && arr[j] < arr[j - 1]) --j;
        return i == j && i != 0 && i != arr.size() - 1;
    }
};
```

289. 生命游戏

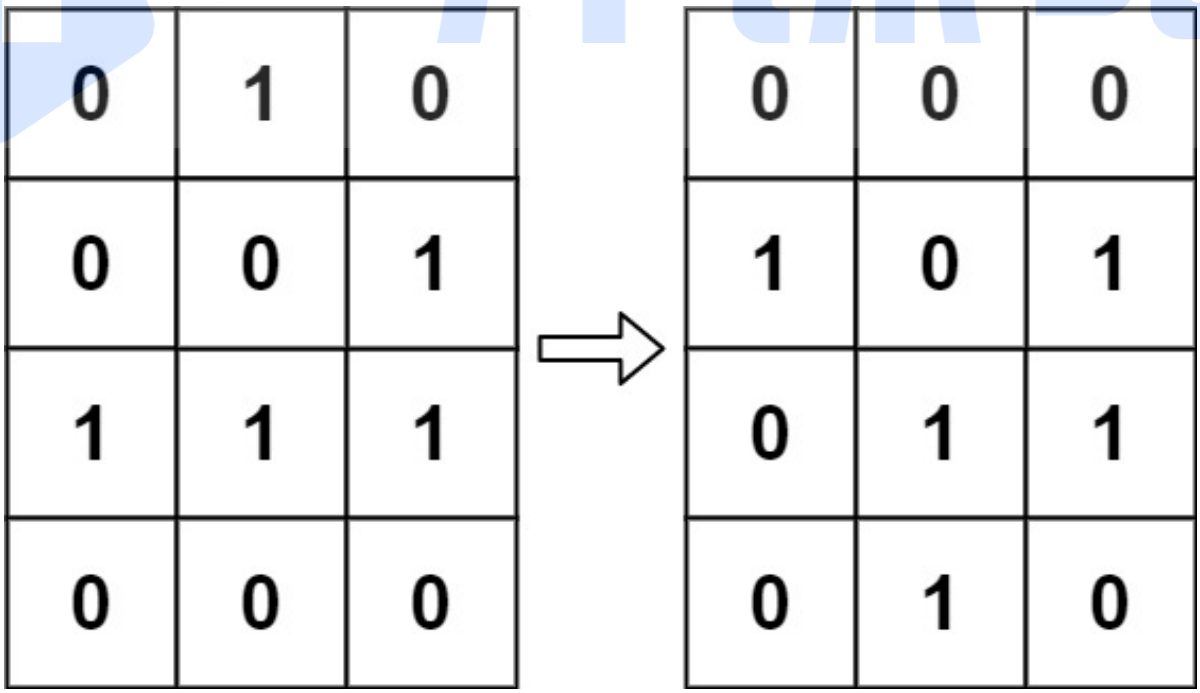
根据 百度百科 ，生命游戏，简称为生命，是英国数学家约翰·何顿·康威在 1970 年发明的细胞自动机。

给定一个包含 m × n 个格子的面板，每一个格子都可以看成是一个细胞。每个细胞都具有一个初始状态：1 即为活细胞（live），或 0 即为死细胞（dead）。每个细胞与其八个相邻位置（水平，垂直，对角线）的细胞都遵循以下四条生存定律：

1. 如果活细胞周围八个位置的活细胞数少于两个，则该位置活细胞死亡；
2. 如果活细胞周围八个位置有两个或三个活细胞，则该位置活细胞仍然存活；
3. 如果活细胞周围八个位置有超过三个活细胞，则该位置活细胞死亡；
4. 如果死细胞周围正好有三个活细胞，则该位置死细胞复活；

下一个状态是通过将上述规则同时应用于当前状态下的每个细胞所形成的，其中细胞的出生和死亡是同时发生的。给你 `m x n` 网格面板 `board` 的当前状态，返回下一个状态。

示例 1：




```
输入：board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
输出：[[0,0,0],[1,0,1],[0,1,1],[0,1,0]]
```

```cpp
class Solution {
public:
    void gameOfLife(vector<vector<int>>& board) {
        vector<vector<int>> cnt(board);
        int dir[8][2] = {
            0, 1, 1, 0,
            0, -1, -1, 0,
            1, 1, -1, -1,
            1, -1, -1, 1
        };
        int n = board.size(), m = board[0].size();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                cnt[i][j] = 0;
                for (int k = 0; k < 8; k++) {
                    int x = i + dir[k][0], y = j + dir[k][1];
                    if (x < 0 || x >= n) continue;
                    if (y < 0 || y >= m) continue;
                    cnt[i][j] += board[x][y];
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (board[i][j]) {
                    board[i][j] = (cnt[i][j] == 2 || cnt[i][j] == 3);
                } else {
                    board[i][j] = (cnt[i][j] == 3);
                }
            }
        }
        return ;
    }
};
```

754. 到达终点数字

在一根无限长的数轴上，你站在`0`的位置。终点在`target`的位置。

每次你可以选择向左或向右移动。第 n 次移动（从 1 开始），可以走 n 步。

返回到达终点需要的最小移动次数。

示例 1:

```
输入：target = 3
输出：2
解释：
第一次移动，从 0 到 1 。
第二次移动，从 1 到 3 。
```

```cpp
class Solution {
public:
    int reachNumber(int target) {
        target = abs(target);
        int k = floor(sqrt(2 * target));
        while ((1 + k) * k / 2 < target) ++k;
        int delta = (1 + k) * k / 2 - target;
        while (delta & 1) k += 1, delta += k;
        return k;
    }
};
```

1155. 掷骰子的N种方法

这里有 `d` 个一样的骰子，每个骰子上都有 `f` 个面，分别标号为 `1, 2, ..., f`。

我们约定：掷骰子的得到总点数为各骰子面朝上的数字的总和。

如果需要掷出的总点数为 `target`，请你计算出有多少种不同的组合情况（所有的组合情况总共有 `f^d` 种），**模** `10^9 + 7` 后返回。

示例 1：

```
输入：d = 1, f = 6, target = 3
输出：1
```

```cpp
class Solution {
public:
    int numRollsToTarget(int d, int f, int target) {
        int dp[d + 1][target + 1], mod_num = 1e9 + 7;
        memset(dp, 0, sizeof(dp));
        dp[0][0] = 1;
        for (int i = 1; i <= d; i++) {
            for (int j = i; j <= target; j++) {
                for (int k = 1; k <= f; k++) {
                    if (j < k) break;
                    dp[i][j] += dp[i - 1][j - k];
                    dp[i][j] %= mod_num;
                }
            }
        }
        return dp[d][target];
```

```
    }
};
```

132. 分割回文串 II

给你一个字符串 `s`，请你将 `s` 分割成一些子串，使每个子串都是回文。

返回符合要求的 **最少分割次数** 。

示例 1：

```
输入：s = "aab"
输出：1
解释：只需一次分割就可将s分割成 ["aa","b"] 这样两个回文子串。
```

```
class Solution {
public:
    void extract(string &s, int i, int j, vector<vector<int>> &ind) {
        while (i >= 0 && s[i] == s[j]) {
            ind[j + 1].push_back(i);
            --i, ++j;
        }
        return ;
    }
    int minCut(string s) {
        vector<vector<int>> ind(s.size() + 1);
        for (int i = 0; i < s.size(); i++) {
            extract(s, i, i, ind);
            extract(s, i, i + 1, ind);
        }
        int dp[s.size() + 1];
        dp[0] = 0;
        for (int i = 1, I = s.size(); i <= I; i++) {
            dp[i] = i;
            for (auto j : ind[i]) {
                dp[i] = min(dp[i], dp[j] + 1);
            }
        }
        return dp[s.size()] - 1;
    }
};
```

1147. 段式回文

段式回文 其实与 一般回文 类似，只不过是最小的单位是 一段字符 而不是 单个字母。

举个例子，对于一般回文 "`abcba`" 是回文，而 "`volvo`" 不是，但如果我们把 "`volvo`" 分为 "`vo`"、"`l`"、"`vo`" 三段，则可以认为 “`(vo)(l)(vo)`” 是段式回文（分为 3 段）。

给你一个字符串 `text`，在确保它满足段式回文的前提下，请你返回 **段**的 **最大数量** `k`。

如果段的最大数量为 `k`，那么存在满足以下条件的 `a_1, a_2, ..., a_k`：

- 每个 `a_i` 都是一个非空字符串；
- 将这些字符串首位相连的结果 `a_1 + a_2 + ... + a_k` 和原始字符串 `text` 相同；
- 对于所有`1 <= i <= k`，都有 `a_i = a_{k+1 - i}`。

示例 1：

```
输入：text = "ghiabcdefhelloadamhelloabcdefghi"
输出：7
解释：我们可以把字符串拆分成 "(ghi)(abcdef)(hello)(adam)(hello)(abcdef)(ghi)"。
```

```cpp
class Solution {
public:
    int getResult(string &text, int l, int r) {
        int n = r - l + 1;
        if (n <= 1) return n;
        for (int i = 1, I = n / 2; i <= I; i++) {
            bool flag = true;
            for (int j = l, k = r - i + 1, t = 0; k <= r; j++, k++) {
                if (text[j] == text[k]) continue;
                flag = false;
                break;
            }
            if (flag) return getResult(text, l + i, r - i) + 2;
        }
        return 1;
    }
    int longestDecomposition(string text) {
        return getResult(text, 0, text.size() - 1);
    }
};
```