## 1217. 玩筹码（简单）

1. 题目描述

   数轴上放置了一些筹码，每个筹码的位置存在数组 `chips` 当中。

   你可以对 **任何筹码** 执行下面两种操作之一（**不限操作次数**，0 次也可以）：

   - 将第 `i` 个筹码向左或者右移动 2 个单位，代价为 **0**。
   - 将第 `i` 个筹码向左或者右移动 1 个单位，代价为 **1**。

   最开始的时候，同一位置上也可能放着两个或者更多的筹码。

   返回将所有筹码移动到同一位置（任意位置）上所需要的最小代价。

   **示例 1：**

   ```
   输入：chips = [1,2,3]
   输出：1
   解释：第二个筹码移动到位置三的代价是 1，第一个筹码移动到位置三的代价是 0，总代价为 1。
   ```

   **示例 2：**

   ```
   输入：chips = [2,2,2,3,3]
   输出：2
   解释：第四和第五个筹码移动到位置二的代价都是 1，所以最小总代价为 2。
   ```

   **提示：**

   - `1 <= chips.length <= 100`
   - `1 <= chips[i] <= 10^9`

2. 简单实现

   奇数可以以代价0变为任意奇数，偶数同理，故实际上花费的代价是改变数字的奇偶性的代价

```cpp
class Solution {
public:
    int minCostToMoveChips(vector<int>& chips) {
        int odd = 0, eval = 0;
        for(int i = 0; i < chips.size(); i++){
            if(chips[i] % 2 == 1)
                odd++;//奇数个数
            else
                eval++;//偶数个数
        }
        return min(odd, eval);
    }
};
```

## 1218. 最长定差子序列（中等）

1. 题目描述

给你一个整数数组 `arr` 和一个整数 `difference`，请你找出 `arr` 中所有相邻元素之间的差等于给定 `difference` 的等差子序列，并返回其中最长的等差子序列的长度。

**示例 1：**

```
输入：arr = [1,2,3,4], difference = 1
输出：4
解释：最长的等差子序列是 [1,2,3,4]。
```

**示例 2：**

```
输入：arr = [1,3,5,7], difference = 1
输出：1
解释：最长的等差子序列是任意单个元素。
```

**示例 3：**

```
输入：arr = [1,5,7,8,5,3,4,2,1], difference = -2
输出：4
解释：最长的等差子序列是 [7,5,3,1]。
```

**提示：**

- `1 <= arr.length <= 10^5`
- `-10^4 <= arr[i], difference <= 10^4`

2. 简单实现

设dp[i]表示以arr[i]结尾的最长等差子序列，则 `dp[i] = max(dp[j] + 1)`，其中满足`arr[j] + difference = arr[i]`，但如果线性搜索会是O(n^2)的复杂度，不符合要求，因此对符合条件的dp[j]的搜索改进为用二分法实现

```
typedef struct node{//存储arr内各个数字及其索引，用于排序
    int idx;
    int num;
};
bool cmp(const node& st1, const node& st2) {
    if(st1.num < st2.num)
        return true;
    else if(st1.num > st2.num)
        return false;
    else
        return st1.idx < st2.idx;
}
class Solution {
public:
    int longestSubsequence(vector<int>& arr, int difference) {
        int len = arr.size();
        vector<node> v = vector<node>(len);
        for(int i = 0; i < len; i++){
            v[i].idx = i;
```

```
            v[i].num = arr[i];
        }
        sort(v.begin(), v.end(), cmp);//对node的num升序排序

        vector<int> dp = vector<int>(len, 1);
        int ans = 0;
        for(int i = 1; i < len; i++){
            int aim = arr[i] - difference;//等差序列中arr[i]前面的数只能是aim
            int l = 0, r = len - 1;
            while(l < r){//二分法找v中最后一个aim
                int mid = l + (r - l) / 2;
                if(v[mid].num < aim)
                    l = mid + 1;
                else if(v[mid].num > aim)
                    r = mid;
                else{
                    if(mid == len - 1 || v[mid+1].num != aim){
                        l = mid;
                        break;
                    }
                    else
                        l = mid + 1;
                }
            }
            while(l >= 0 && v[l].num == aim){
                if(v[l].idx < i)//找到的数字的索引必须小于当前索引
                    dp[i] = max(dp[i], dp[v[l].idx] + 1);
                l--;
            }
            ans = max(ans, dp[i]);
        }
        return ans;
    }
};
```

3. 最优解法

用map记录以每个数结尾的定差子序列的最大长度就行了啊你个傻X

```
class Solution {
public:
    map<int,int> m;
    int ans = 0;
    int longestSubsequence(vector<int>& arr, int difference) {
        for(auto x: arr) {
            int dp = 1;
            if (m.count(x - difference))
                dp = m[x - difference] + 1;
            ans = max(dp, ans);
            m[x] = max(m.count(x)? m[x]:1, dp);//更新m
        }
        return ans;
    }
```

```
};
```

## 1219. 黄金矿工（中等）

1. 题目描述

   你要开发一座金矿，地质勘测学家已经探明了这座金矿中的资源分布，并用大小为 `m * n` 的网格 `grid` 进行了标注。每个单元格中的整数就表示这一单元格中的黄金数量；如果该单元格是空的，那么就是 `0`。

   为了使收益最大化，矿工需要按以下规则来开采黄金：

   - 每当矿工进入一个单元，就会收集该单元格中的所有黄金。
   - 矿工每次可以从当前位置向上下左右四个方向走。
   - 每个单元格只能被开采（进入）一次。
   - **不得开采**（进入）黄金数目为 `0` 的单元格。
   - 矿工可以从网格中 **任意一个** 有黄金的单元格出发或者是停止。

   **示例 1：**

   ```
   输入：grid = [[0,6,0],[5,8,7],[0,9,0]]
   输出：24
   解释：
   [[0,6,0],
    [5,8,7],
    [0,9,0]]
   一种收集最多黄金的路线是：9 -> 8 -> 7。
   ```

   **示例 2：**

   ```
   输入：grid = [[1,0,7],[2,0,6],[3,4,5],[0,3,0],[9,0,20]]
   输出：28
   解释：
   [[1,0,7],
    [2,0,6],
    [3,4,5],
    [0,3,0],
    [9,0,20]]
   一种收集最多黄金的路线是：1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7。
   ```

   **提示：**

   - `1 <= grid.length, grid[i].length <= 15`
   - `0 <= grid[i][j] <= 100`
   - 最多 **25** 个单元格中有黄金。

2. 简单实现

   dfs即可

   ```
   class Solution {
   public:
       vector<vector<int>> dir = {{-1,0}, {1,0}, {0,-1}, {0,1}};
       int ans;
   ```

```
    void dfs(vector<vector<int>>& grid, int x, int y, int temp){
        int back = grid[x][y];
        grid[x][y] = 0;
        temp += back;
        bool flag = true;
        for(int i = 0; i < 4; i++){
            int x2 = x + dir[i][0];
            int y2 = y + dir[i][1];
            if(0 <= x2 && x2 < grid.size()
               && y2 >= 0 && y2 < grid[0].size()
               && grid[x2][y2] > 0){
                dfs(grid, x2, y2, temp);
                flag =  false;
            }
        }
        if(flag)
            ans = max(ans, temp);
        grid[x][y] = back;
        temp -= back;
    }
    int getMaximumGold(vector<vector<int>>& grid) {
        ans = 0;
        for(int i = 0; i < grid.size(); i++)
            for(int j = 0; j < grid[0].size(); j++)
                if(grid[i][j] > 0)
                    dfs(grid, i, j, 0);
        return ans;
    }
};
```

## 1220. 统计元音字母序列的数目（困难）

1. 题目描述

给你一个整数 `n`，请你帮忙统计一下我们可以按下述规则形成多少个长度为 `n` 的字符串：

- 字符串中的每个字符都应当是小写元音字母（`'a'`, `'e'`, `'i'`, `'o'`, `'u'`）
- 每个元音 `'a'` 后面都只能跟着 `'e'`
- 每个元音 `'e'` 后面只能跟着 `'a'` 或者是 `'i'`
- 每个元音 `'i'` 后面 **不能** 再跟着另一个 `'i'`
- 每个元音 `'o'` 后面只能跟着 `'i'` 或者是 `'u'`
- 每个元音 `'u'` 后面只能跟着 `'a'`

由于答案可能会很大，所以请你返回 模 `10^9 + 7` 之后的结果。

**示例 1：**

```
输入：n = 1
输出：5
解释：所有可能的字符串分别是："a", "e", "i" , "o" 和 "u"。
```

**示例 2：**

```
输入：n = 2
输出：10
解释：所有可能的字符串分别是："ae", "ea", "ei", "ia", "ie", "io", "iu", "oi", "ou" 和
"ua"。
```

**示例 3：**

```
输入：n = 5
输出：68
```

**提示：**

- `1 <= n <= 2 * 10^4`

2. 简单实现

令`dp[0...4][n]`依次代表以aeiou为开头的长为n的字符串的个数，则根据规则有`dp[0][n] = dp[1][n-1], dp[1][n] = dp[0][n-1] + dp[2][n-1]`等，因为`dp[0...4][n]`只与`dp[0...4][n-1]`有关，因此用两个一维数组记录即可

```
class Solution {
public:
    int MOD = 1e9 + 7;
    int countVowelPermutation(int n) {
        vector<long> dp1 = vector<long>(5, 1);//用long防止相加溢出
        vector<long> dp2 = vector<long>(5, 0);
        while(--n){
            dp2[0] = dp1[1] % MOD;
            dp2[1] = (dp1[0] + dp1[2]) % MOD;
            dp2[2] = (dp1[0] + dp1[1] + dp1[3] + dp1[4]) % MOD;
            dp2[3] = (dp1[2] + dp1[4]) % MOD;
            dp2[4] = dp1[0] % MOD;
            dp1 = dp2;
        }
        return (dp1[0] + dp1[1] + dp1[2] + dp1[3] + dp1[4]) % MOD;
    }
};
```