

循环铭文 (correct) 题解

思路分析

合法字符串必须出现在无限循环串 `YesYesYes...` 中。因此相邻字符的转移只能是：

- `Y` 后面只能接 `e`；
- `e` 后面只能接 `s`；
- `s` 后面只能接 `Y`。

同时，字符串中的每个字符都必须属于 `Y`、`e`、`s`。逐位检查即可。

另一种更短的写法是构造一个长度超过 50 的 `YesYes...` 字符串，然后用 `find` 判断给定字符串是否为它的子串。由于题目保证 $|s| \leq 50$ ，构造足够长即可。

复杂度

时间复杂度 $O(\sum |s|)$ ，空间复杂度 $O(1)$ 。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

void check(string s) {
    if (s[0] != 'Y' && s[0] != 'e' && s[0] != 's') {
        cout << "NO\n";
        return;
    }
    for (int i = 0; i + 1 < (int)s.size(); i++) {
        if ((s[i] != 'Y' && s[i] != 'e' && s[i] != 's') ||
            (s[i] == 'Y' && s[i + 1] != 'e') ||
            (s[i] == 'e' && s[i + 1] != 's') ||
            (s[i] == 's' && s[i + 1] != 'Y')) {
            cout << "NO\n";
            return;
        }
    }
    cout << "YES\n";
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t;
    cin >> t;
    while (t--) {
        string s;
        cin >> s;
```

```
        check(s);  
    }  
    return 0;  
}
```

能量补给 (worm) 题解

思路分析

为了让选择的数量尽可能少，每次都应优先选能量值大的晶块。因此先将数组按从大到小排序，再求前缀和 $sum[i]$ ，表示选择前 i 枚最大晶块可以得到的总能量。

对于一次需求 x ：

- 如果 $sum[n] < x$ ，说明所有晶块加起来也不够，输出 -1 ；
- 否则在单调递增的前缀和数组中二分查找第一个 $sum[i] \geq x$ 的位置，答案就是 i 。

复杂度

时间复杂度 $O(n \log n + q \log n)$ ，空间复杂度 $O(n)$ 。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

const int N = 200005;
int n, q, a[N];
long long sum[N];

bool cmp(int x, int y) {
    return x > y;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> q;
    for (int i = 1; i <= n; i++) cin >> a[i];
    sort(a + 1, a + n + 1, cmp);
    for (int i = 1; i <= n; i++) {
        sum[i] = sum[i - 1] + a[i];
    }
    while (q--) {
        int x;
        cin >> x;
        if (sum[n] < x) {
            cout << -1 << '\n';
        } else {
            cout << lower_bound(sum + 1, sum + n + 1, x) - sum << '\n';
        }
    }
    return 0;
}
```

共振频率 (iteration) 题解

思路分析

由于 $\gcd(a, b) \leq a, b$ ，新加入的数字一定不会超过初始数组中的最大值。值域不大，可以枚举每个可能的数字 x ，判断它是否能够被生成。

若 x 能由若干已有数字通过 gcd 操作得到，那么参与生成的数字都必须是 x 的倍数，并且这些倍数的整体 gcd 必须恰好为 x 。

因此对每个尚未出现在初始数组中的 x ：

1. 枚举 $2x, 3x, 4x, \dots$ 这些 x 的倍数；
2. 只考虑初始数组中出现过的倍数；
3. 维护这些倍数的 gcd，若最后结果等于 x ，则 x 可以被加入。

已经在初始数组中的数字不能算作“新加入”，直接跳过。

复杂度

时间复杂度 $O(M \log M)$ (M 为最大值，调和级数枚举)，空间复杂度 $O(M)$ 。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

const int M = 2000005;
int cnt[M], maxValue, ans;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        cnt[x]++;
        maxValue = max(maxValue, x);
    }
    for (int x = 1; x <= maxValue; x++) {
        if (cnt[x]) continue;
        int g = 0;
        for (int y = 2 * x; y <= maxValue; y += x) {
            if (cnt[y]) g = __gcd(g, y);
        }
        if (g == x) ans++;
    }
    cout << ans << '\n';
}
```

```
return 0;
```

```
}
```

载荷排行 (allowance) 题解

思路分析

只有当前装载量严格大于你的设备装载量的其他设备，才会排在你前面。若要让这样一台设备无法参评，需要花费 $w_i - t_i + 1$ 单位货物。

贪心思路是：在所有当前排在你前面的设备中，优先处理花费最小的那台。每处理一台，你的装载量会减少，相应地，原本排在你后面的设备也可能变成排在你前面，所以需要动态维护。

做法如下：

1. 将其他设备按装载量 t_i 升序排序；
2. 用指针从装载量最大的设备开始，把所有 $t_i > t$ 的设备加入小根堆，堆中存放它们的处理花费 $w_i - t_i + 1$ ；
3. 当前堆大小加 1 就是此时你的名次，用它更新答案；
4. 如果堆顶花费不超过你当前剩余装载量，就处理这台设备，并减少自己的装载量；否则无法继续，结束。

过程中每台设备最多入堆一次、出堆一次。

复杂度

时间复杂度 $O(n \log n)$ ，空间复杂度 $O(n)$ 。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
const int M = 300005;
pair<ll, ll> car[M];
priority_queue<ll, vector<ll>, greater<ll>> pq;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    ll t, w;
    cin >> t >> w;
    for (int i = 2; i <= n; i++) {
        cin >> car[i].first >> car[i].second;
    }
    sort(car + 2, car + n + 1);

    int ans = n, i = n;
    while (true) {
        while (i >= 2 && car[i].first > t) {
            pq.push(car[i].second - car[i].first + 1);
            i--;
        }
    }
```

```
    }  
    ans = min(ans, (int)pq.size() + 1);  
    if (!pq.empty() && t >= pq.top()) {  
        t -= pq.top();  
        pq.pop();  
    } else {  
        break;  
    }  
}  
cout << ans << '\n';  
return 0;  
}
```