Codeforces Div3 1065's Solution

A.Shizuku Hoshikawa and Farm Legs

思路分析:第一题其实很简单，就一个数学问题鸡兔同笼，只不过要求出所有解，并且排序去重,直接上代码部分

知识点: 数学

Code

```c
#include <stdio.h>
#include <string.h>

#define HASH_SIZE 200000

void solve() {
    long long n;
    scanf("%lld", &n);

    int hash[HASH_SIZE] = {0};
    int cnt = 0;

    for (long long i = 0; i <= n / 2; i++) {
        if ((n - 2 * i) % 4 == 0 && hash[i] == 0) {
            hash[i] = 1;
            cnt++;
        }
    }

    printf("%d\n", cnt);
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        solve();
    }
    return 0;
}
```

B.Yuu Koito and Minimum Absolute Sum

思路分析:这个题我最开始做的时候卡了很久，我在想如何构造满足一个这么长式子的数组，但其实很简单，注意到 **b数组** 是差值数组, 既然是差值数组求和, 那么可以对原式进行化简, $b_1 + b_2 + b_3 + \ldots + b_n$, 由于 $b_i = a_i - a_{i-1}$ 那么我们可以先展开所有的 b_i , 那么 $|a_2 - a_1 + a_3 - a_2 + \ldots + a_n - a_{n-1}|$, 化简得到 $|a_n - a_1|$, 所以说我们其实是要求这个的 **min**, 那么其实思路就很简单了, 既然你都要求字典序最小, 那么中间的所有 **-1** 都可以变成最小的 **0** , 然后如果 $a_n = -1$ 或者 $a_0 = -1$, 那么我们只需要将他们变成相同的值, 如果都是-1, 那么就是都为 0, 如果都不为-1, 那么就是输出他们相减的绝对值

知识点: 数学

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 300007
#define MOD 1000000007

void solve() {
    long long n;
    scanf("%lld", &n);

    long long a[N];
    for (long long i = 0; i < n; ++i) {
        scanf("%lld", &a[i]);
    }

    long long ans;
    if (a[0] != -1 && a[n-1] != -1) {
        ans = llabs(a[n-1] - a[0]);
    } else {
        ans = 0;
    }

    if (ans == 0) {
        if (a[0] == -1 && a[n-1] == -1) {
            a[0] = 0;
            a[n-1] = 0;
        } else if (a[0] == -1) {
            a[0] = a[n-1];
        } else if (a[n-1] == -1) {
            a[n-1] = a[0];
        }
    }

    for (long long i = 0; i < n; ++i) {
        if (a[i] == -1) {
            a[i] = 0;
        }
    }

    printf("%lld\n", ans);
    for (long long i = 0; i < n; ++i) {
        printf("%lld ", a[i]);
    }
    printf("\n");
}

int main() {
    int T;
```

```
    scanf("%d", &T);
    while (T--) {
        solve();
    }
    return 0;
}
```

C1.Renako Amaori and XOR Game (easy version)

思路分析: 由题意很容易就知道如果按照最优策略, 如果他们没有移动之前的得分相等的话, 那么其实做不做操作都一样, 因为根据位运算的性质, 就是做一次操作之后就会把自己的得分和对方互换或者不动, 如果初始得分不一样, 那么就是看谁最后操作, 因为你从前面就可以知道, 操作就是交换双方的得分, 所以说决定权在最后一个人身上, 由于只能交互同一个索引下的不同的数才为有效操作

知识点: 位运算, 数学

Code

```cpp
#include <stdio.h>
#include <stdlib.h>

#define N 300007
#define MOD 1000000007

void solve() {
    long long n;
    scanf("%lld", &n);

    long long a[N], b[N];
    long long resa = 0, resb = 0;

    for (long long i = 1; i <= n; ++i) {
        scanf("%lld", &a[i]);
        resa ^= a[i];
    }

    for (long long i = 1; i <= n; ++i) {
        scanf("%lld", &b[i]);
        resb ^= b[i];
    }

    long long cnt1 = 0, cnt2 = 0;
    for (long long i = 1; i <= n; ++i) {
        if (a[i] != b[i]) {
            if (i % 2 == 1) {
                cnt1++;
                cnt2 = 0;
            } else {
                cnt2++;
                cnt1 = 0;
            }
```

```
        }
    }

    if (resa == resb) {
        printf("Tie\n");
    } else {
        if (cnt1 > cnt2) {
            printf("Ajisai\n");
        } else if (cnt1 < cnt2) {
            printf("Mai\n");
        } else {
            if (resa > resb) {
                printf("Ajisai\n");
            } else if (resa < resb) {
                printf("Mai\n");
            } else {
                printf("Tie\n");
            }
        }
    }
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        solve();
    }
    return 0;
}
```

C2. Renako Amaori and XOR Game (hard version)

思路分析: 这个题其实就是上一个题目的按位计算, 我们可以知道对于每一位来说就是 0,1, 所以说按位扩展就行

知识点: 数学, 位运算

Code

```
#include <stdio.h>
#include <stdlib.h>

#define N 300007
#define MOD 1000000007
#define MAX_BIT 30

void solve() {
    long long n;
    scanf("%lld", &n);

    long long a[N], b[N], d[N];
```

```
    long long res = 0;

    for (long long i = 1; i <= n; ++i) {
        scanf("%lld", &a[i]);
    }

    for (long long i = 1; i <= n; ++i) {
        scanf("%lld", &b[i]);
        d[i] = a[i] ^ b[i];
        res ^= d[i];
    }

    if (res == 0) {
        printf("Tie\n");
        return;
    }

    int k = 0;
    for (int bit = MAX_BIT; bit >= 0; --bit) {
        if (res & ((long long)1 << bit)) {
            k = bit;
            break;
        }
    }

    long long q = -1;
    for (long long i = n; i >= 1; --i) {
        if (((d[i] >> k) & 1) == 1) {
            q = i;
            break;
        }
    }

    if (q == -1) {
        printf("Tie\n");
    } else if (q % 2 == 1) {
        printf("Ajisai\n");
    } else {
        printf("Mai\n");
    }
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        solve();
    }
    return 0;
}
```

D/F.Rae Taylor and Trees (easy/hard version)

思路分析: 题目意思很简单, 不要被吓到了, 其实就是一个区间内有很多不连续单调区间, 然后要你判断是否能将他们通过特定的规则连在一起, 规则: 如果 下一个区间的$mx >=$ 上一区间的r ,所以说我们可以通过维护一个前缀数组 $pref$, 后缀数组 $suff$ 来记录区间的最大值以及每一个区间的终点, 这个 $easy$ 版本 其实就比 $hard$ 多了一个操作, 就是如果存在这个数的话, 需要你找出链接每一段区间的数值并将其输出就行

知识点: 前后缀和预处理

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 300007

typedef struct {
    int val;
    int pos;
} Pair;

// 比较函数：用于获取两个Pair中的最大值（val大的优先；val相等时pos大的优先，与原代码max逻辑一致）
Pair max_pair(Pair a, Pair b) {
    if (a.val > b.val) return a;
    if (a.val < b.val) return b;
    return (a.pos > b.pos) ? a : b;
}

void solve() {
    int n;
    scanf("%d", &n);

    int p[N];
    int pre[N];
    Pair suf[N + 2];

    pre[0] = n;
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &p[i]);
        pre[i] = (pre[i-1] < p[i]) ? pre[i-1] : p[i];
    }

    suf[n+1].val = 0;
    suf[n+1].pos = 0;
    for (int i = n; i >= 1; --i) {
        Pair curr = {p[i], i};
        suf[i] = max_pair(curr, suf[i+1]);
    }

    for (int i = 2; i <= n; ++i) {
```

```
        if (pre[i-1] > suf[i].val) {
            printf("No\n");
            return;
        }
    }

    printf("Yes\n");
    int l = 1;
    while (l <= n) {
        int r = suf[l].pos;
        for (int i = l; i < r; ++i) {
            printf("%d %d\n", p[i], p[r]);
        }
        if (l > 1) {
            printf("%d %d\n", pre[l-1], p[r]);
        }
        l = r + 1;
    }
}

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        solve();
    }
    return 0;
}
```

E.Anisphia Wynn Palettia and Good Permutations

思路分析: 其实这也是一个数学构造题, 题意很简单就是任意一对连续三元组, 如果他们之间两两配对 $gcd(a,b) == 1$ 的话, 则证明他不是一个好的索引, 我们需要构造一个 n 长度的序列, 然后这个序列中不超过 6 对坏索引, 其实数论里面有一点, 相邻偶数对一定 $gcd > 1$, 相邻奇数对一定 $gcd == 1$, 根据这个知识又因为, 如果你有从 $1-n$ 的数,那么其中偶数有 $n/2$个, 奇数有 $n/2$, 所以说其实每一对偶数对中间加一个奇数组成的三元组一定是好的索引, 那么我们可以消去 $n/4$个奇数, 又因为其实 每个一个奇数必然会是 3 的倍数, 所以说 可以把 3 的数选出来, 如果还剩下奇数全放在后面,其实必然就可以少于 6 对坏索引, 那么我们其实可以证明出来的

知识点: 数论, gcd

Code

```
#include <stdio.h>

#define MAX_N 1000

void solve() {
    int n;
    scanf("%d", &n);

```

```
    int threes[MAX_N], evens[MAX_N], odds[MAX_N];
    int t_cnt = 0, e_cnt = 0, o_cnt = 0;
    for (int i = 1; i <= n; ++i) {
        if (i % 3 == 0) {
            threes[t_cnt++] = i;
        } else if (i % 2 == 0) {
            evens[e_cnt++] = i;
        } else {
            odds[o_cnt++] = i;
        }
    }

    int res[MAX_N * 2];
    int res_idx = 0;
    int odd_idx = 0;

    for (int i = 0; i < e_cnt; i += 2) {
        res[res_idx++] = evens[i];
        if (i + 1 < e_cnt) {
            res[res_idx++] = evens[i + 1];
        }
        if (odd_idx < o_cnt) {
            res[res_idx++] = odds[odd_idx++];
        }
    }

    for (int i = 0; i < t_cnt; i += 2) {
        res[res_idx++] = threes[i];
        if (i + 1 < t_cnt) {
            res[res_idx++] = threes[i + 1];
        }
        if (odd_idx < o_cnt) {
            res[res_idx++] = odds[odd_idx++];
        }
    }

    while (odd_idx < o_cnt) {
        res[res_idx++] = odds[odd_idx++];
    }

    for (int i = 0; i < res_idx; ++i) {
        printf("%d ", res[i]);
    }
    printf("\n");
}

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        solve();
    }
    return 0;
}
```

G.Sakura Adachi and Optimal Sequences

思路分析: 倒序贪心, 由于我们对所有一起 $*2$, 所以说只有操作 1 的答案有贡献, 然后我们可以对 $\times 2$ 进行分层进行计算贡献, 每一层的贡献使用组合数学公式计算, 由于对答案取模, 所以说我们需要使用 逆元 与 费马小定理 计算贡献, 逆元使用快速幂求解

知识点: 逆元，费马小定理, 组合数学, 位运算, 贪心

Code

```cpp
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#define MOD 1000003LL
#define N 300007LL
#define MAX_K 60LL
typedef int64_t ll;
typedef long double ld;

ll fact[MOD];
ll a[N];
ll b[N];
ll c[N];
ll cnt[MAX_K];

ll inv(ll a) {
    ll res = 1LL;
    ll b = MOD - 2LL;
    while (b > 0LL) {
        if (b % 2LL == 1LL) {
            res = res * a % MOD;
        }
        a = a * a % MOD;
        b /= 2LL;
    }
    return res;
}

int __lg(ll x) {
    if (x == 0LL) return -1;
    return 63 - __builtin_clzll(x);
}

ll accumulate(ll *arr, int n) {
    ll sum = 0LL;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    return sum;
}

void solve() {
```

```
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%lld", &a[i]);
    }
    for (int i = 0; i < n; i++) {
        scanf("%lld", &b[i]);
    }

    ll k_val = __lg(b[0] / a[0]);
    for (int i = 1; i < n; i++) {
        ll current = __lg(b[i] / a[i]);
        if (current < k_val) {
            k_val = current;
        }
    }
    ll x = k_val;

    memset(cnt, 0, sizeof(cnt));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < k_val; j++) {
            cnt[j] += (b[i] & 1LL);
            b[i] >>= 1LL;
        }
        c[i] = b[i] - a[i];
    }

    ll ans = 1LL;
    for (int j = 0; j < k_val; j++) {
        x += cnt[j];
        ans = ans * fact[cnt[j]] % MOD;
    }

    ll res = accumulate(c, n);
    if (res < MOD) {
        ans = ans * fact[res] % MOD;
    } else {
        ans = 0LL;
    }

    for (int i = 0; i < n; i++) {
        x += c[i];
        ans = ans * inv(fact[c[i]]) % MOD;
    }

    printf("%lld %lld\n", x, ans);
}

int main() {
    fact[0] = 1LL;
    for (ll i = 1LL; i < MOD; i++) {
        fact[i] = fact[i - 1] * i % MOD;
    }

```

```
    int T;
    scanf("%d", &T);
    while (T--) {
        solve();
    }

    return 0;
}
```

H.Shiori Miyagi and Maximum Array Score

思路分析: 动态规划 + 树状数组快速维护区间max, 枚举 n 的每个状态, 使用树状数组优化 $logn$ 时间内更新与得到区间最大值

知识点: 树状数组, 动态规划

Code

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>

#define MOD 1000000007LL
#define N 300007LL
typedef int64_t ll;

ll t[N];  // 树状数组
int n, m;

// lowbit函数
ll lowbit(ll x) {
    return x & -x;
}

// 更新树状数组
void update(ll k, ll v) {
    ll max_k = m - n + 1;  // 树状数组最大下标
    for (; k <= max_k; k += lowbit(k)) {
        if (v > t[k]) {
            t[k] = v;
        }
    }
}

// 查询树状数组
ll query(ll k) {
    ll res = 0;
    for (; k > 0; k -= lowbit(k)) {
        if (t[k] > res) {
            res = t[k];
        }
    }
```

```
    return res;
}

void clear() {
    memset(t, 0, sizeof(t));
}

ll count_factor(ll b, ll x) {
    ll ans = 0;
    while (x % b == 0) {
        ans++;
        x /= b;
    }
    return ans;
}

void solve() {
    scanf("%lld %lld", &n, &m);

    clear();

    for (ll i = 2; i <= n; i++) {
        ll start_j = ((m - n + i) / i) * i;
        for (ll j = start_j; j >= i; j -= i) {
            ll pos = j - i + 1;  x
            ll val = query(pos) + count_factor(i, j);
            update(pos, val);
        }
    }

    printf("%lld\n", query(m - n + 1));
}

int main() {
    int T = 1;
    scanf("%d", &T);
    while (T--) {
        solve();
    }

    return 0;
}
```