2025 SYNU 4月周赛 Round V

题解报告

A. 二进制博弈

tag

博弈论

解题思路

结论：如果一个数是三的倍数，那么Orange先手必败，否则Orange先手必胜。

如果你打表前几个数，你会发现这个结论。

考虑证明博弈策略：

我们观察 2^k 的性质：

$$2^k \equiv \begin{cases} 1 \pmod 3 & \text{若 } k \text{ 为偶数} \\ 2 \pmod 3 & \text{若 } k \text{ 为奇数} \end{cases}$$

因此，假设当前的 n 是3的倍数，那么经过单次操作后石头堆变成 $n-2^k$，由于 n 是3的倍数，2^k 对3同余1或者2，因此做差之后的结果一定也对3同余1或者2，而此时对手只需要拿走 $2^{k'}$ 使得剩余的石头 n' 再次回到3的倍数即可。**因此，我们每轮(2人各行动一次算一轮)博弈实际上就是让n减少3的若干倍**，由于 n 是3的倍数，且因为Orange是先手，所以最后0一定会落到Orange手上，使得其必败。同理，若n不是3的倍数，反之则反。

参考代码

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    cin.tie(0) -> sync_with_stdio(0);
    int t;
    cin >> t;
    while(t--)
    {
        int n;
        cin >> n;
        cout << (n % 3 == 0 ? "No" : "Yes") << '\n';
    }
}
```

B. 质数分解

tag

筛法，双指针，哈希，枚举

解题思路

考虑用质数筛筛出 $1-4e7$ 中所有的质数 p_i，对于每个询问 x，本质上是询问在 p_i 序列中，是否存在一段区间和 $[p_l,...,p_r]$ 等于 x。可以考虑用双指针，或者哈希算法 $O(n)$ 求出。

给出双指针的思路，考虑枚举右端点，维护区间和 sum，每次若区间和大于 x 则左移左端点并消除贡献，直到 $sum \le x$，然后再判断 sum 是否等于 x 即可。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
// #define int long long
#define endl '\n'
#define all(_x) _x.begin(), _x.end()
#define range(_x, _st, _ed) (_x.begin() + _st), (_x.begin() + _ed)
#define rep(_x, _y, _z) for (int _x = _y; _x <= _z; _x++)
#define matrix(_x, _y, _z) vector<vector<int>>(_x, vector<int>(_y, _z))
#define debug(_x) cout << #_x << '=' << _x << endl
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using mat = vector<vector<int>>;
using u64 = unsigned long long;
constexpr int N = 4e7 + 10;
// dont use umap!!!

void solve()
{
    int x;
    cin >> x;
    int ans = 0;
    vector<int> primes, vis(x + 1);
    vector<i64> pre(1);
    int j = 0;
    for(int i = 2; i <= x; i ++)
    {
        if(!vis[i])
        {
            primes.push_back(i);
            pre.push_back(pre.back() + i);
            while(j < pre.size() and pre.back() - pre[j] > x) j++;
            if(pre.back() - pre[j] == x) ans ++;
        }
        for(auto p : primes)
        {
            if(i * p > x) break;
            vis[i * p] = 1;
            if(i % p == 0) break;
        }
    }
    cout << ans << '\n';
}

signed main()
{
    IOS;
    int _ = 1;
    cin >> _;
    while (_--) solve();

    return 0;
}
```

对于哈希做法给出思路，由于要判断一段区间和是否等于 x，考虑对 p_i 作前缀和 pre，每次枚举右端点并用哈希表标记 pre_r，枚举过程中判断 $pre_r - x$ 是否被标记即可。

C. 幽谷传响II

tag

单调栈，分类讨论

解题思路

假设没有 k 的限制，那么本题的做法与 P1397幽谷传响 做法一致，你可以在 【题解】CEIT 3月周赛 中，找到该题的做法。

在P1397的基础上，本题实际上是对子数组的范围进行了限制。在P1397的题解做法基础上，我们假设 a_i 是所在子数组的最小值，其作为子数组最小值的最大范围为 (L, R)，注意是开区间。

我们对其与 k 的限制进行分类讨论：

- $R - L - 1 < k$，即本身最大范围不超过 k，那么此时跟没有限制一样，贡献为 $(i - L) \times (R - i) \times a_i$。
- 若 $R - L - 1 > k$
- 因为子数组必须包含 a_i ，因此其合法子数组的左端点必须大于等于 $i - k + 1$，同理右端点必须小于等于 $i + k - 1$。钦定 $L = \max(L, i - k), R = \min(R, i + k)$。
- - 若子数组左端点大于 $R - k$，那么右端点可以在 $[i, R)$ 中任取，当前情况的贡献为 $i - (R - k) \times (i - (R - k))$。
  - 若子数组左端点小于等于 $R - k$：
  - - 若左端点在 $L + 1$，右端点可以取 $[i, L + k]$，贡献为 $L + k - i + 1$
    - 若左端点在 $L + 2$，右端点可以取 $[i, L + k + 1]$，贡献为 $L + k - i + 2$
    - 若左端点在 $L + 3$，右端点可以取 $[i, L + k + 2]$，贡献为 $L + k - i + 3$
    - ...
    - 若左端点在 $R - k$，右端点可以取 $[i, R - 1]$，贡献为 $R - i$

    观察上述式子，不难发现，贡献构成了一组等差数列，因此贡献之和为:

$$(L + R + k - 2i + i)(R - L - k)/2$$

因此，总贡献为 $((R - i)(i - (R - k)) + (L + R + k - 2i + i)(R - L - k)/2)a_i$

综上所述，答案为上述两种情况之和。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define IOS                         \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);                   \
    cout.tie(0);
// #define int long long
#define endl '\n'
#define all(_x) _x.begin(), _x.end()
#define matrix(_x, _y, _z) vector<vector<int>>(_x, vector<int>(_y, _z))
#define debug(_x) cout << #_x << '=' << _x << endl
using i64 = long long;
using i128 = __int128;
using pii = pair<int, int>;
using mat = vector<vector<int>>;
using u64 = unsigned long long;
constexpr int N = 2e5 + 10, mod = 1e9 + 7;
// dont use umap!!!

i64 qpow(i64 a,i64 k,i64 p=mod)
{
    i64 res=1;
    while(k)
    {
        if(k & 1) res = res * a % p;
        k >>= 1ll;
        a = a * a % p;
    }
    return res;
}

signed main()
{
    IOS;
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    // 左边界 left[i] 为左侧严格小于 a[i] 的最近元素位置（不存在时为 -1）
    vector<int> left(n, -1);
    // 右边界 right[i] 为右侧小于等于 a[i] 的最近元素位置（不存在时为 n）
    vector<int> right(n, n);
    stack<int> st;
    st.push(-1); // 哨兵，方便赋值 left
    for (int i = 0; i < n; i++)
    {
        int x = a[i];
        while (st.size() > 1 && x <= a[st.top()])
        {
            right[st.top()] = i; // i 是栈顶的右边界
            st.pop();
        }
        left[i] = st.top();
        st.push(i);
    }

    i64 res = 0;
    for (int i = 0; i < n; i++)
    {
        int x = a[i], l = left[i], r = right[i];
        if (r - l - 1 <= k)
        {
            i64 cnt = 1LL * (i - l) * (r - i) % mod;
            res += x * cnt % mod; // 累加贡献
```

```
            res %= mod;
        }
        else
        {
            l = max(l, i - k);
            r = min(r, i + k);
            // 左端点 > r-k 的子数组个数
            i64 cnt = 1LL * (r - i) * (i - (r - k)) % mod;
            // 左端点 <= r-k 的子数组个数
            i64 cnt2 = 1LL * (l + r + k - i * 2 + 1) * (r - l - k) % mod * qpow(2, mod - 2) % mod;
            res += x * (cnt + cnt2) % mod; // 累加贡献
            res %= mod;
        }
    }
    cout << res << '\n';
    return 0;
}
```

D. Orange的集合选数

tag

动态规划，组合计数

解题思路

不妨考虑动态规划。

我们设 $f[i][j]$ 表示前 i 个数字能够构成的元素个数为 j 的所有子集和的平方。

设 $g[i][j]$ 表示前 i 个数字能够构成的元素个数为 j 的所有子集和。

然后我们考虑第 i 个元素加入或者不加入:

- 第 i 个元素不加入：

$$f[i][j] = f[i-1][j]$$
$$g[i][j] = g[i-1][j]$$

- 第 i 个元素加入，考虑加入 a_i，从前 $i-1$ 个数中，选出大小为 $j-1$ 的子集个数为 $\binom{i-1}{j-1}$，每个集合都加入一个 a_i，因此其贡献为 $\binom{i-1}{j-1} \times a_i$，因此得到 g 的转移表达式，同时根据平方和公式可得 f 的转移表达式:

$$f[j][j] = f[i-1][j-1] + 2 \times g[i-1][j-1] \times a_i + \binom{i-1}{j-1} a_i^2$$
$$g[i][j] = g[i-1][j-1] + \binom{i-1}{j-1} a_i$$

最终答案为 $G(i) = f[n][i]$。

参考代码

```cpp
#include <bits/stdc++.h>
using namespace std;
constexpr int mod = 1e9 + 7, N = 3e3 + 5;
int a[N], n;
int f[N][N], g[N][N], C[N][N];
int main()
{
    ios::sync_with_stdio(false);
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> a[i];
    }
    C[0][0] = 1;
    for (int i = 1; i <= n; i++)
    {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++)
        {
            C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
        }
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= i; j++)
        {
            f[i][j] = (f[i - 1][j] + ((f[i - 1][j - 1] + 2ll * g[i - 1][j - 1] % mod * a[i] % mod) % mod + 1ll * C[i - 1][j - 1] * a[i] %
            g[i][j] = (g[i - 1][j] + (g[i - 1][j - 1] + 1ll * C[i - 1][j - 1] * a[i] % mod) % mod) % mod;
        }
    }
    for (int i = 1; i <= n; i++)
        cout << f[n][i] << ' ';
    return 0;
}
```