2023 年牛客多校第二场题解

出题人：北京理工大学

1 A Link with Checksum

题意：定义如下的类 CRC 的校验和算法。

```
string Link_CRC(string D)
{
    P = 0x04C11DB7;
    int n = D.length();
    D += "00000000 00000000 00000000 00000000";
    for (int i = 1; i <= n; i++)
    {
        if (D 的最高位为 1)
        {
            去掉 D 的最高位;
            D 的最高 32 位和 P 异或;
        }
        else
            去掉 D 的最高位;
    }
    return D;
}
```

现给定一个长度为 n_1 比特的 01 串 A 和长度为 n_2 比特的 01 串 B，要求找到一个长度为 32 bit 的校验和 checksum，使得 Link_CRC$(A + \text{checksum} + B) = \text{checksum}$。$1 \le n_1, n_2 \le 10^5$。

解法：不难注意到，无论最高位是什么，都会被去掉。且 P 只有 26 位，显然不符合 CRC 运算中模数比余数长一位的规则。考虑令 $P = $ 0x104C11DB7（增加第 33 位为 1），同时修改规则使得这个类 CRC 变成一个真正的 CRC 运算：

```
string CRC(string D)
{
    P = 0x104C11DB7;
    int n = D.length();
    丢弃 D 的先导 0;
    while (位数 >= 33)
    {
        if (D 的最高位为 1)
            D 最高 33 位异或 P;
        去掉 D 的最高位;
    }
    取 D 最后 32 位作为 D;
    return D;
}
```

既然是传统 CRC，那么必然满足 CRC 的各种性质

1. 每个二进制位彼此独立。即满足形如 $\mathrm{CRC}(x \oplus y) = \mathrm{CRC}(x) \oplus \mathrm{CRC}(y)$。
2. $$\mathrm{CRC}(2x) = \begin{cases} 2\mathrm{CRC}(x), x \text{ 最高位为 } 0 \\ 2\mathrm{CRC}(x) \oplus P, x \text{ 最高位为 } 1 \end{cases}$$

利用上述两个性质，可以提前将每一位上有 1 的余数计算出来。由于校验和的长度固定，因而给出的 A 串和 B 串的 CRC 余数 r 可以提前计算。现在问题转化为求一长度为 32 的 01 串 x，需要满足 $r \oplus \mathrm{CRC}(x \times 2^{8n_2}) = x$。

由于串长仅 32，可以考虑折半搜索——枚举和存储高 16 位的情况，然后再枚举低 16 位的结果与之匹配。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 2000000;
const long long P = 0x104C11DB7ll;
long long bit[N + 5];
int a[N + 5], b[N + 5];
int main()
{
    bit[0] = 0x04C11DB7ll; // 注意：和朴素CRC不同，该方法的CRC的0x1（边界条件）的余数不是1，而是0x04C11DB7。
    for (int i = 1; i <= N; i++)
    {
        bit[i] = bit[i - 1] << 1;
        if (bit[i] >> 32) // 取最高位
            bit[i] ^= P;
    }
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = 1; i <= m; i++)
        scanf("%d", &b[i]);
    int pos = 0;
    long long AB = 0;
    for (int i = m; i >= 1; i--, pos += 8)
        for (int j = 7; j >= 0; j--)
            if (b[i] >> j & 1)
                AB ^= bit[pos + j];
    int startpos = pos;
    pos += 32;
    for (int i = n; i >= 1; i--, pos += 8)
        for (int j = 7; j >= 0; j--)
            if (a[i] >> j & 1)
                AB ^= bit[pos + j];
    // 折半搜索
    map<long long, long long> highpos; // 记录高 16 位的答案
    for (int i = 0; i < 1 << 16; i++)
```

```
    {
        long long cur = 0;
        for (int j = 0; j < 16; j++)
            if (i >> j & 1)
                cur ^= bit[startpos + 16 + j];
        highpos[cur ^ ((long long)i << 16) ^ AB] = i;
    }
    // 枚举低位
    for (int i = 0; i < 1 << 16; i++)
    {
        long long cur = 0;
        for (int j = 0; j < 16; j++)
            if (i >> j & 1)
                cur ^= bit[startpos + j];
        if (highpos.count(cur ^ i))
        {
            printf("%lld", (highpos[cur ^ i] << 16) | i);
            return 0;
        }
    }
    printf("-1");
    return 0;
}
```

这样做的时间复杂度为 $\mathcal{O}(16n + 2^{16})$。

当然，本题的 CRC 位数不够多，当位数达到 64 或者正常 CRC 的 512 或 2048 位时，显然不支持折半搜索。这时不难注意到我们可以根据各位余数贡献独立的条件构建异或方程。我们可以统计余数的某一个二进制位可能会受哪些原输入串的二进制位影响（即原串上该位置有 1 会给余数的某一位上带来一次 1 的异或），从而列出方程。这样就可以使用线性基（01 高斯消元）以更快速的解决本题。

2 B Link with Railway Company

题意：给定一棵 n 个点的树状铁路网，树上每条边表示一段铁路，边权值表示该段铁路的维护费用。现在开行 m 条线路，经过树上的一条链，开行第 i 条线路获利 $x_i - y_i$。现在降本增效，选择其中一部分线路开行，选择开行的线路经过的每条铁路都需要投入维护费，如果该段铁路没有线路经过则可以废弃，问最大获利是多少。$1 \leq n, m \leq 10^4$。

解法：我们首先不考虑时间复杂度，我们首先考虑一个最基本的一个图模型，也就是就是当前的这条线路以及对应的这个线路上所有的铁路段，我们把它构成一个依赖关系，也就是说，我们必须这个线路上所有的铁路都要维护，这个时候才能够选择这条线路并进行运营获利。那这个是我们其实构成了一个很经典的一个问题，就是最大权闭合子图——即选择一个点必须选择它所有可到达的后继节点。这个问题可以通过如下的网络流建图得到：

1. 从源点出发向所有的正权点（有出边的点，本题中为线路代表的点）连流量为点权值的边；
2. 从所有没有出边的点（即负权点，本题中为铁路对应的边）向汇点连接边流量为点权的负数（即点权的绝对值）的边。
3. 如果存在依赖关系，则将依赖点（线路）向被依赖点（铁路）连接边流量为无穷大的边。

这样最大权值为所有正权点的和减去图的最小割（最大流）。

这样做的原理是，考虑其中某一条线路和它依赖的铁路集合。如果当前线路盈利能力较强，可以覆盖铁路线路的支出，那么当前最大流等于支出的代价，最后的贡献为线路收入减去支出的代价；如果线路盈利能力不足以覆盖铁路维护成本，则当前该线路的最大流的贡献等于盈利，最后对总收入的贡献抵消为零，即停止运营本条线路。

但是本题中依赖关系的边过多，直接根据依赖关系建立边数量可以达到 $O(nm)$，无法接受。但是不难注意到本题铁路线路构成一棵树，用树链剖分可以将线路对应的铁路构成 $O(\log n)$ 段树链，因而使用线段树优化建图可以实现 $O(m \log^2 n)$ 的边条数，从而可以通过本题。

3 D The Game of Eating

题意：n 个人 m 道菜，现在需要点 k 道菜，每个人轮流点一个。每个人对每个菜有一个评分 $\{a\}_{(i,j)=(1,1)}^{n,m}$，且每个人对不同菜的评分互不相同。每个人都按照这 k 个菜的自我评判标准最优（即如果点菜集合为 $\{p_1, p_2, \cdots . p_k\}$，则第 i 个人最大化 $\sum_{j=1}^{k} a_{i,p_j}$）点菜。问最终会点哪些菜。多组询问，$1 \le T \le 10^3$，$\sum n \le 2 \times 10^3$，$\sum m \le 2 \times 10^3$。

解法：如果正着贪心考虑点菜，那么会出现一个问题——我当前点的菜确实是我现在最喜欢的，但是如果我点我次喜欢的菜，然后后面有人也跟我同好喜欢我最喜欢的菜，那么我不如点我次喜欢的，这样我就能获得我最喜欢和次喜欢的菜了。

不妨倒着考虑这个问题——当现在还有一个菜的份额的时候，这个人一定会点自己最喜欢的菜——后续没人再帮他点菜了。再回退一步，当现在还剩两个菜的份额的时候，既然已知最后一个人会点他最喜欢的，我就点除了这个菜之外最喜欢的菜，依次类推。

因而最终就是倒着考虑最喜欢的菜，加入菜单即可。

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
void solve()
{
    int n, m, k;
    cin >> n >> m >> k;
    vector<vector<int>> a(n, vector<int>(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> a[i][j];
    vector<int> us(m, false);
    vector<int> ans;
    for (int i = k - 1; i >= 0; i--)
    {
        int now = i % n;
        int t = -1;
        for (int j = 0; j < m; j++)
            if (!us[j])
            {
                if (t == -1 || a[now][j] > a[now][t])
```

```
                    t = j;
            }
        us[t] = true;
        ans.push_back(t + 1);
    }
    sort(ans.begin(), ans.end());
    for (auto it : ans)
        cout << it << " ";
    cout << "\n";
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int T;
    cin >> T;
    while (T--)
        solve();
    return 0;
}
```

4 E Square

题意：给定 x，求一个数 y 使得 $0 \le y \le 10^9$ 且 y^2 的最高若干位恰好为 x。$0 \le x \le 10^9$，多组询问，$1 \le T \le 10^5$。

解法：首先特判 $x = 0$ 时 $y = 0$。

可以考虑枚举 y^2 的位数是多少。枚举 k，寻找满足 $\left[\sqrt{x \times 10^k}, \sqrt{(x+1) \times 10^k}\right)$ 的 y 即可。注意此处 $\sqrt{x \times 10^k}$ 可能很大，因而最好使用二分答案来确定该区间上下界。单组时间复杂度 $\mathcal{O}(10 \log 10^{18})$。

```
#include <bits/stdc++.h>
using namespace std;
long long cal(long long x) // 二分开根号
{
    long long l = 1, r = 1'000'000'000, ans = -1;
    while (l <= r)
    {
        long long mid = (l + r) >> 1;
        if (mid * mid >= x)
        {
            ans = mid;
            r = mid - 1;
        }
        else
            l = mid + 1;
    }
```

```
    return ans;
}
int main()
{
    int t;
    long long x, y;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%lld", &x);
        if (!x)
        {
            printf("0\n");
            continue;
        }
        long long ans = -1;
        // 枚举位数
        for (long long th = 1, i = 1; i <= 10; i++, th *= 10)
        {
            long long curl = x * th;
            long long low = cal(curl);
            if (low * low / th == x)
            {
                ans = low;
                break;
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

5 F Link with Chess Game

题意：长度为 n 的链上给定三个颜色不同的棋子的位置，这些棋子可以重叠。两个人一轮移动其中一枚棋子，不能出现之前已经出现的状态，不能操作者输，问结果。多组测试数据，$1 \le T \le 10^5$，$1 \le n \le 10^5$。

解法：首先我们需要考虑一个基本博弈模型：二分图博弈。二分图博弈是指双方在二分图上某个点放置一枚棋子，双方轮流移动棋子，且不能将棋子移动到已经走过的点。结论是，如果起始点不在最大匹配（即去掉该点，则匹配数减 1）上，则先手必败。

考虑现在这个模型。我们可以将当前的状态放置在一个棱长为 n 的正方体上，每移动某一种颜色的棋子相当于是在某个维度上移动一步。首先不难发现，棱长为 n 的正方体上所有整点和与坐标轴平行的边构成一个二分图（可以用黑白染色的结论证明）。那么考虑起始的点是不是在最大匹配上。

对于 n 为偶数的情况，显然每个点都在完美匹配上——考虑固定 z 坐标，仅在和 xOy 平行的平面上，该正方体 n 个 z 轴剖面正方形都是可以完美匹配的。因而这种情况先手必胜。

对于 n 为奇数的情况，则要考察起始点的情况。

1. 如果起始点是黑点（三个维度加起来的和为奇数），则一定存在一种匹配方式使得它不在完美匹配点上，因而这种情况先手必败。
2. 如果当前点为白点，即三个维度加起来的和为偶数，则必然先手必胜。因为黑点的个数比白点多，因而所有的白点都可以被匹配，也就是它一定在完美匹配上。

6 G Link with Centrally Symmetric Strings

题意：定义镜像回文——回文中心是空或 `o,s,z,x`，然后 `o,s,z,x` 自身跟自身匹配，`(b,q),(d,p),(n,u)` 匹配。给定串 S，问是否可以将 S 分解为若干镜像回文串的拼接。多组询问，$\sum |S| \le 10^6$。

解法：考虑一个最简单的 DP 状态：f_i 表示前缀 i 是否可以被拆分成若干个镜像回文串的拼接。那么其实只需要去从前缀 i 的最长回文后缀 j 转移（即，最小的 j 满足 $[j+1,i]$ 为镜像回文串）即可。这是因为如果考虑前缀 i 更短的回文后缀 k，那么它一定可以被从前缀 j 处再加两段回文串拼接而成——$[j+1, j+1+(i-k-1)-1]$，$[j+1+(i-k-1),k]$，$[k+1,i]$。也就是说 $f_j = f_k$。所以每个点只需要从其最长回文后缀处转移即可。CF1827C 和本题在这个处理上非常类似。

基于这个性质，我们只需要找到每个点对应的最长回文后缀是多少，可以使用 Manacher 算法实现——使用并查集，在 Manacher 算法执行匹配过程的时候，维护每个字符最远匹配的点。

但是我们需要对传统 Manacher 算法进行改造才能适用于本题镜像回文的情况。首先是对字符串字符匹配的判定——`zxso` 是自身和自身匹配，而剩下几个配对的字符之间需要互相匹配。同时 `b,p,d,q,n,u` 不能作为回文串中心，即以他们为中心的回文半径为 0。

7 H 0 and 1 in BIT

题意：对一个 01 串定义两种操作：A 表示将这个串 01 反转；B 表示将这个 01 串视为一个二进制数（左侧为最低位），然后执行加一操作。q 次询问，给出一个仅由 AB 构成的长度为 n 的操作序列 S，每次给出一个 01 串，问执行完 $S[l;r]$ 后该 01 串变成什么。$1 \le n, q \le 2 \times 10^5$，每次询问的 01 串长度不超过 50，强制在线。

解法：考虑补码的性质——对于一个二进制数 x，$-x$ 等效于每个 01 位翻转，然后执行加一操作。因而如果将原串视为一个二进制数 x，则 B 操作可以视为 $x \leftarrow -x-1$，而 A 操作则为 $x \leftarrow x+1$。

根据这个转化，我们发现，可以考虑每个字符对于最终答案的影响，例如 $BABA$ 中第一个 B 对答案的影响是 $+1$（因为后面有两个 A），第一个 A 对答案的影响是 $+1$（因为后面有一个 A），第二个 B 对答案的影响是 -1（因为后面有一个 A），第二个 A 对答案的影响是 -1（因为后面什么都没了）。且顺便也能发现对于一段确定的区间 (l,r) 无论输入的 x 是什么，变化都是 x 先乘以 1 或 -1（取决于这段里 A 的奇偶性），在模 2^{64} 意义下加或减同一个常数，我们就是要求出每个区间的这个常数，记所求的东西为 $f(l,r)$。

转化后的问题可以用矩阵加线段树或加倍增或前缀求矩阵逆来维护，但本题也可以使用前缀和。

记 $cnt(l,r)$ 表示 (l,r) 中 A 的数量，可以前缀和预处理。同时我们也可以前缀和预处理出前缀的答案 $f(1,1), f(1,2), \cdots, f(1,n-1), f(1,n)$。

对于一次询问 (l,r)：若 $cnt(l,r)$ 是偶数，则有 $f(1,l-1)+f(l,r)=f(1,r)$；若 $cnt(l,r)$ 是奇数，则有 $-f(1,l-1)+f(l,r)=f(1,r)$，因此都可以解出 $f(l,r)$。另外注意当 $cnt(l,r)$ 是奇数时，要把输入的 x 先乘以 -1。

本题和 AGC044C 类似，但是本题要求强制在线，有兴趣可以了解一下。

8 I Link with Gomoku

题意：给定 $n \times m$ 的棋盘上面玩五子棋，黑手先，轮流下棋使得最终局面为平局，没有禁手。多组测试数据，满足 $\sum n \times m \leq 10^6$。

解法：由于需要是一个可以一步一步下出来的局面，因而黑白棋个数需要考虑——当 $n \times m$ 为偶数时要相同，为奇数时黑比白多一个。

尽可能考虑在两行内实现黑白棋子个数相同，如果列数为偶数时能够一行内就相同。不难得到这样的双行构造：

```
偶数时：
xoxoxoxoxoxoxo
oxoxoxoxoxoxox
奇数时：
xoxoxoxoxoxox
oxoxoxoxoxoxo
```

但是如果简单这样拼接会导致斜行出现五连（下方大写的 `X`）：

```
xoxoxoxoXoxoxo
oxoxoxoXoxoxox
xoxoxoXoxoxoxo
oxoxoXoxoxoxox
xoxoXoxoxoxoxo
oxoxoxoxoxoxox
```

为避免这种情况，可以考虑让 $3-4$ 两行交换：

```
xoxoxoxoxoxoxo
oxoxoxoxoxoxox
oxoxoxoxoxoxox
xoxoxoxoxoxoxo
xoxoxoxoxoxoxo
oxoxoxoxoxoxox
oxoxoxoxoxoxox
xoxoxoxoxoxoxo
```

这样就可以避免斜行的问题了。

最后注意一个 corner case 就是这样安排之后可能 `o` 比 `x` 更多，这个时候需要交换二者以实现黑手先（黑比白多）。

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

```cpp
const int N = 1000;
int s[N + 5][N + 5];
int main()
{
    int t, n, m;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &n, &m);
        int cntx = 0, cnto = 0;
        for (int i = 0; i < n; i++)
        {
            int cur = (i % 4 <= 1) ? 0 : 1; // 模4为23的时候要交换两行
            int op = (i & 1) ^ cur;
            for (int j = 1; j <= m; j++)
            {
                if (op == 0)
                    s[i + 1][j] = 1, cntx++;
                else
                    s[i + 1][j] = 0, cnto++;
                op ^= 1;
            }
        }
        int rev = 0;
        if (cntx < cnto) // 交换ox
            rev = 1;
        for (int i = 1; i <= n; i++, puts(""))
            for (int j = 1; j <= m; j++)
                if (s[i][j] ^ rev)
                    printf("x");
                else
                    printf("o");
    }
    return 0;
}
```

9 K Box

题意：给定长度为 n 的序列 $\{a\}_{i=1}^{n}$，a_i 表示第 i 个盒子被盖子保护起来之后可以获得的价值。现在有一些地方有盖子，每个盖子至多只能往左或往右移动一格，问能被盖子保护起来的价值最大值。$1 \le n \le 10^6$，$1 \le a_i \le 10^9$。

解法：显然，不同盖子之间相对位置关系不会发生变化，而每个盖子只有三种状态——往左移动一格（用 0 表示），不动（1），右移一格（2）。设 $f_{i,j}$ 表示前 i 个盖子在 j 状态时所能保护的最大值，记第 i 个盖子在 p_i 处，则不难得到下面的 dp 方程：

$$f_{i,j} = \max_{k=0,1,2}(f_{i-1,k} + a_{p_i+j-1}), s.t. \, p_{i-1} + k - 1 \le p_i + j - 1$$

总时间复杂度 $\mathcal{O}(n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000000;
long long f[N + 5][3], a[N + 5];
int pos[N + 5], cnt, b[N + 5];
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%lld", &a[i]);
    for (int i = 1; i <= n; i++)
    {
        scanf("%d", &b[i]);
        if (b[i])
            pos[++cnt] = i;
    }
    if (!cnt)
    {
        printf("0");
        return 0;
    }
    memset(f, 0xcf, sizeof(f)); // 初始化为全-inf
    f[1][1] = a[pos[1]];
    if (pos[1] > 1)
        f[1][0] = a[pos[1] - 1];
    if (pos[1] < n)
        f[1][2] = a[pos[1] + 1];
    for (int i = 2; i <= cnt; i++)
        for (int j = 0; j <= 2; j++)
            for (int k = 0; k <= 2; k++)
                if (pos[i - 1] + k - 1 < pos[i] + j - 1 && pos[i] + j - 1 <= n)
                    // 不越界，盖子之间相对位置关系保持
                    f[i][j] = max(f[i][j], f[i - 1][k] + a[pos[i] + j - 1]);
    printf("%lld", max({f[cnt][0], f[cnt][1], f[cnt][2]}));
    return 0;
}
```