A⊕B 问题

考点：位运算,思维,二进制分块/数位DP。

区间 $[0,n]$ 可以按二进制拆分成若干个二进制高位不变、低位取遍所有值的区间段。如 $n=18$ 有 $[0,18]=[0,15]\cup[16,17]\cup[18,18]$。设二进制位长为 5，则第一个区间高位不变是 0，低 4 位取遍 $0/1$，即从 $(00000)_2$ 取到 $(01111)_2$；第二个区间高位不变是 1000，最低位变化，从 $(10000)_2$ 取到 $(10001)_2$；第三个区间全部高位不变 10000，没有变化的低位。按照这样的拆法，得到的低位段长一定各不相同，所以最多能拆出 $\log n$ 个区间。在实现时，可以逆序枚举 2^i，若 $2^i\le n$，代表 2^i 个不同的数 $[0,2^i-1]$ 可以取遍低 i 位，由此确定了一个低位段。那么下一次枚举时就等于删掉了这些数，故新的 $n'=n-2^i$。可以 $O(\log n)$ 实现枚举全部拆分段。

这样拆分后的每个区间有一个方便的性质，若区间每个数都要异或 x，则可以分成高位异或 x 与低位异或 x 讨论。设区间长为 L，考虑高位段时，可以认为有 L 个相同的数异或 x 的高位，所以将一次异或的结果乘以 L 即可；考虑低位段时，不论位运算结果如何，总和不变，即有 $\sum_{i=0}^{2^k-1} i\oplus x(x\le 2^i-1)=\sum_{i=0}^{2^k-1} i$。例如，低位是低两位，考虑区间 $[24,27]$ 的数：$(11000)_2,(11001)_2,(11010)_2,(11011)_2$。假设要同时异或 $(01110)_2$，得 $(10110)_2,(10111)_2,(10100)_2,(10101)_2$，可以看到高三位都是 101；而低两位从 $00,01,10,11$ 变成了 $10,11,00,01$，只是改变了顺序，而总和不变。所以对高低段分别都可以 $O(1)$ 计算出结果(低位用等差数列求和求出 $0+1+\cdots+2^i-1$)。

故总复杂度为 $O(\log n)$。注意取模公式。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
using ll = long long;
ll n, x, mod = 1e9 + 7, inv2 = (mod + 1) / 2;
signed main()
{
    sc(n), sc(x);
    ll ans = 0;
    for (ll i = 62, pw = 1LL << i, rem = n + 1, cnt = 0; i >= 0; --i, pw >>= 1)
    {
        if (pw <= rem)
        {
            ll lf = cnt, rf = cnt + pw - 1, num = rf - lf + 1;
            ll base = ((x ^ cnt) & (~(pw - 1)));
            ll sum = (pw % mod) * ((pw - 1 + mod) % mod) % mod * inv2 % mod;
            sum += (base % mod) * (num % mod) % mod;
            ans = (ans + sum) % mod;
            cnt += pw, rem -= pw;
        }
    }
    printf("%lld", ans);
    return 0;
}
```

本题也可以数位 DP，可自行尝试。

养秋

考点：贪心+小模拟。

1. 显然，购买了秋之后，越晚卖越好，即总是在第 23 天卖出。易证这样做比更早卖出更优。
2. 设在第 i 天购入秋，则该秋利润为 $23-i+1$ 天每天贡献一点，且卖出能回 6 点。故若价格低于 $23-i+1+6$，证明当前购入秋是赚的。
3. 显然如果能购入的话，越早购入越好。因此策略是每一天一开始贪心地买秋，直到发现不赚了或不够钱为止。
4. 由于 $11c^{25+n}=2\times 11c^n$，解方程得 $c=2^{\frac{1}{25}}$。最坏情况下，不赚钱时有 $11\times 2^{\frac{n}{25}}=23+6$ 解得 $n\approx 35$，故无论如何最多只需要购买约 35 个秋，可以用枚举法实现本题。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll mn = 40, days = 23;
ll g[mn], blue = 25, profit, h;
ll price()
{
    return 11 * pow(2, 1. * h / 25);
}
ll f(ll n)
{
    return 20 * (1 <= n && n <= 6) + 35 * (7 <= n && n <= 12) + 50 * (13 <= n && n <= 18) + 80 * (19 <= n);
}
signed main()
{
    cin.tie(0)->ios::sync_with_stdio(false);
    for (ll i = 1; i <= days; ++i)
    {
        cin >> g[i];
    }
    for (ll i = 1; i <= days; ++i)
    {
        ll expect = (days - i + 1) + 6;             //还能赚多少
        while (blue >= price() && price() < expect) //够钱买,能赚钱
        {
            blue -= price();
            profit -= price();
            ++h;
        }
        blue += h + f(i) + g[i];
        profit += h;
    }
    profit += 6 * h;
    cout << profit;
    return 0;
}
```

01trie上动态点分治

考点：前缀和+贪心+滑动窗口/单调队列。

可以先叠一个前缀异或，设 $S_i=\oplus_{1\le j\le i}a_j$，可以 $O(n)$ 预处理 $S_i=S_{i-1}\oplus a_i$，将问题简化为：

$$\max_{k\le s\le n,s\le r\le n}\min_{r-s+1\le i\le r}S_i$$

即求所有长为 $s(k\le s\le n)$ (即长至少是 k)的区间 $[r-s+1,r],1\le r-s+1,s\le n$ 里，每个区间最小值的最大值。

观察易得，设 $k\le s_1<s_2\le n$，则区间长 s_2 的答案不会比区间长 s_1 的答案更优。因为区间越长，越难让 $\min$ 区间尽可能大。即 $k\le s\le n$ 的最大值必然在 $k=s$ 取得。易证，证略。故问题简化为：

$$\max_{k\le r\le n}\min_{r-k+1\le i\le r}S_i$$

即求所有长为 k 的区间的最小值的最大值。

不难想到，只需要用滑动窗口维护长为 k 的区间，动态维护最小值即可。因为若存在 $S_i\le S_{i-1}$，则之后的所有区间里，最小值将不再可能从 S_{i-1} 取得而只能在 S_i 取得，因此可以维护单调递增的双端队列即可。

复杂度 $O(n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll mn = 2e6 + 10;
ll n, k, ans, s[mn];
signed main()
{
    cin.tie(0)->ios::sync_with_stdio(false);
    cin >> n >> k;
    for (ll i = 1, a; i <= n; ++i)
    {
        cin >> a;
        s[i] = s[i - 1] ^ a;
    }
    deque<ll> q;
    for (ll i = 1; i <= n; ++i)
    {
        while (!q.empty() && q.front() <= i - k)
        {
            q.pop_front();
        }
        while (!q.empty() && s[q.back()] >= s[i])
        {
            q.pop_back();
        }
        q.push_back(i);
        if (i >= k)
        {
            ans = max(ans, s[q.front()]);
        }
    }
    cout << ans;
    return 0;
}
```

选数

考点：随机化/生日悖论。

设 $b_i = f(i)$，由于 a 是随机序列，f 运算没有破坏随机性，故可以认为 b 也是随机序列。

策略：不断枚举 x 并计算 $f(x)$，若发现相同的就马上输出。可以证明能够很快找出相同。

引理：生日悖论：一年有 n 天，有 k 人，生日均匀分布且相互独立。问两人生日相同的概率达到 p_0 至少要多少个人。

设生日互不相同，显然概率为 $p = \frac{n}{n} \times \frac{n-1}{n} \times \cdots \times \frac{n-k+1}{n}$，则有 $1 - p \ge p_0, p \le 1 - p_0$。根据 $1 + x \le e^x$ 有 $p \le e^{-\frac{1}{n}} \times e^{-\frac{2}{n}} \times \cdots \times e^{-\frac{k-1}{n}} = e^{-\frac{k(k-1)}{2n}} \le 1 - p_0$。

放到题目里，有 $n = 10^5$，令 $k = 10^3$，计算得 $e^{-\frac{k(k-1)}{2n}} \approx 0.007$，即有 $p_0 = 99.3\%$ 的概率在随机求 10^3 个 $f(x)$ 后找到相同值。故复杂度大约为 $O(nk) \approx 10^8$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 10;
using ll = long long;
ll seed, n, a[maxn];
ll nextRand()
{
    static ll x = seed;
    x ^= x << 11;
    x ^= x >> 45;
    x ^= x << 14;
    return (x % n + n) % n;
}
signed main()
{
    cin.tie(0)->ios::sync_with_stdio(false);
    cin >> n >> seed;
    for (int i = 0; i < n; ++i)
    {
        a[i] = nextRand();
    }
    map<ll, ll> m;
    for (int x = 0; x < n; ++x)
    {
        ll cnt = 0;
        for (ll i = 0; i < n; ++i)
        {
            cnt = (cnt + (a[i] ^ (i * x))) % n;
        }
        if (m.find(cnt) != m.end())
        {
            cout << m[cnt] << ' ' << x;
            return 0;
        }
        m[cnt] = x;
```

```
    }
    return 0;
}
```

可持久化线段树分治

考点：链表。

即求：给定 h 数组($0 \le h_i < n$ 且值域连续，即若出现了 $h_i = x$，必然出现 $h_j = 1, 2, \cdots, x-1$)，有 n 次询问，第 $k(0 \le k < n)$，每次问 h 有多少个区间 $[l, r]$ 满足 $h_i \ge k(l \le i \le r)$，区间长为 $m-1$，设每个区间的贡献甲为 C_m^2，贡献乙为数组 a 的区间 $[l-1, r]$ 任选两个值相乘的最大值，求区间贡献甲之和及贡献乙的最大值。

转换到 h 数组上，若一个子区间 $[l, r]$ 都满足 $h \ge k$，则区间 $[l-1, r]$ 的每个子区间都可以选择，枚举左右端点共有 C_n^2 个方案，且区间 $[l-1, r]$ 的最大值和次大值组成最值(也有可能是最小负数和次小负数，但不能是最大值和最小值相乘(它们有可能是同一个玩意))。

ST 表会超时(不仅是初始化)，且无法做到 $O(1)$ 询问次小值(次小不满足可重复贡献)，线段树同理超时。考虑使用区间合并，即从大到小枚举 k，设一开始每个区间长度为 1。每次对满足 $h_i = k$ 的区间，将其与附近的区间合并，更新贡献，维护最大最小次大次小值。

实现细节：使用双链表维护区间，可以 $O(1)$ 查询每个区间相邻的两区间，如果相邻区间有效(即未被合并且已经遍历过即 $\ge k$)，那么将当前区间与相邻区间合并(共四种情况，合并左右，合左，合右，不合)，若合并注意删除节点。总合并次数是 $O(n)$，故复杂度是 $O(n)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll mn = 1e6 + 10, inf = 2e9;
ll n, a[mn], b[mn], cnt, mx = -2e18, ncnt[mn], nmx[mn];
vector<ll> bin[mn];
ll nx[mn], pr[mn];
void rmnode(ll i) { nx[pr[i]] = nx[i], pr[nx[i]] = pr[i]; }
struct segment
{ //合并处理的区间为[l,r],实际计算包含的区间为[l-1,r]
    ll l = -1, r = -1, mx1 = -inf, mx2 = -inf, mi1 = inf, mi2 = inf;
    void add(ll i) //最值维护
    {
        if (a[i] > mx1) // mx1:最大值
        {
            mx2 = mx1, mx1 = a[i];
        }
        else if (a[i] > mx2) // mx2:次大值
        {
            mx2 = a[i];
        }
        if (a[i] < mi1) // mi1:最小值
        {
            mi2 = mi1, mi1 = a[i];
        }
        else if (a[i] < mi2) // mi2:次小值
        {
            mi2 = a[i];
```

```
        }
    }
    pair<ll, ll> getans() { return make_pair(((r - l + 1) + 1) * (r - l + 1) / 2, max(mx1 * mx2, mi1 * mi2)); }
    void update(ll w)
    {
        segment ext = *this;
        ext.add(ext.l - 1);
        auto pr = ext.getans();
        cnt += w * pr.first, mx = max(mx, pr.second);
    }
    segment() {}
    segment(ll lf) { l = lf, r = lf, add(lf), update(1); }
    segment(ll xl, ll xr) : l(xl), r(xr) {}
    friend segment smerge(segment l, segment r)
    {
        l.update(-1), r.update(-1);
        segment t = segment(l.l, r.r);
        if (l.mx1 > r.mx1)
        {
            t.mx1 = l.mx1;
            t.mx2 = max(l.mx2, r.mx1);
        }
        else
        {
            t.mx1 = r.mx1;
            t.mx2 = max(r.mx2, l.mx1);
        }
        if (l.mi1 < r.mi1)
        {
            t.mi1 = l.mi1;
            t.mi2 = min(l.mi2, r.mi1);
        }
        else
        {
            t.mi1 = r.mi1;
            t.mi2 = min(r.mi2, l.mi1);
        }
        t.update(1);
        return t;
    }
} s[mn];
void solve()
{
    for (ll i = 1; i <= n; ++i)
    { //初始化长为n的链表,每个点代表长为1的,左端点为i的区间
        nx[i] = i + 1, pr[i] = i - 1;
    }
    for (ll i = 1; i <= n; ++i)
    {
        bin[b[i]].push_back(i);
    }
    for (ll i = n; i >= 1; --i)
    {
        for (auto v : bin[i])
```

```
        {
            ll vl = pr[v], vr = nx[v];
            s[v] = segment(v);
            if (s[vl].r + 1 == s[v].l && s[v].r + 1 == s[vr].l)
            {
                s[v] = smerge(s[v], s[vr]);
                s[vl] = smerge(s[vl], s[v]);
                rmnode(v), rmnode(vr);
            }
            else if (s[vl].r + 1 == s[v].l)
            {
                s[vl] = smerge(s[vl], s[v]), rmnode(v);
            }
            else if (s[v].r + 1 == s[vr].l)
            {
                s[v] = smerge(s[v], s[vr]), rmnode(vr);
            }
        }
        ncnt[i] = cnt, nmx[i] = mx;
    }
    for (ll i = 1; i <= n; ++i)
    {
        cout << ncnt[i] << ' ' << (nmx[i] == -2e18 ? 0 : nmx[i]) << '\n';
    }
}
signed main()
{
    cin.tie(0)->ios::sync_with_stdio(false);
    cin >> n;
    for (ll i = 1; i <= n; ++i)
    {
        cin >> a[i];
    }
    for (ll i = 2; i <= n; ++i)
    {
        cin >> b[i];
    }
    solve();
    return 0;
}
```

数位和

签到题。

f 的最大值当且仅当 x 的每个数位都是 9。故一次 f 运算后，$f(x)$ 最大为 $9n = 9 \times 10^5$。所以本题不需要使用高精度。

由此可知，经过很少次数的 f 运算后，$f^k(x)$ 必然收敛为个位数，且个位数满足 $f(x) = x$，所以当发现 $f(x)$ 为个位数时，马上 break 即可。

复杂度为 $O(n)$。

```
#include <bits/stdc++.h>
```

```
using namespace std;
using ll = long long;
const ll mn = 1e5 + 10;
ll n, k, x;
char x0[mn];
signed main()
{
    cin.tie(0)->ios::sync_with_stdio(false);
    cin >> n >> k >> (x0 + 1);
    --k;
    for (ll i = 1; i <= n; ++i)
    {
        x += x0[i] - '0';
    }
    for (ll i = 1, v; i <= k; ++i)
    {
        if (x < 10)
        {
            break;
        }
        v = x, x = 0;
        for (; v; v /= 10)
        {
            x += v % 10;
        }
    }
    cout << x << '\n';
    return 0;
}
```