2025 蓝桥杯省赛c++B组个人题解

by ExtractStars - Codeforces

声明

本题解为退役蒻苟所写，不保证正确性，仅供参考。

花了大概2个半小时写完，感觉比去年省赛简单，难度大概等价于 codeforces dv4.5 吧

菜鸡不熟悉树上背包，调了一个多小时

A题

算一下弧长和半径即可

B题

我们需要为 2025 家分店分配客流上限，一个排列$A_1, A_2, \ldots, A_{2025}$

要求满足

1. $A_1, A_2, \ldots, A_{2025}$ 是 $1, 2, \ldots, 2025$ 的排列；
2. 对任意 $1 \le i, j \le 2025$ 有

   $$A_i \times A_j \le i \times j + 2025.$$

观察发现：

1. 取 $i = j$ 得

   $$A_i^2 \le i^2 + 2025 \quad \Longrightarrow \quad A_i \le \sqrt{i^2 + 2025}$$

   当 i 较小（例如 $i = 1$）时，$\sqrt{1 + 2025} \approx 45$，而当 i 足够大时，$\sqrt{i^2 + 2025}$ 与 i 仅相差很少。
2. 另外，取 $j = 2025$ 有

   $$A_i \times A_{2025} \le i \times 2025 + 2025$$

   由于用排列的性质可证明，必须把数值 2025 放在最后一个位置（若把 2025 放在比 2025 小的位置，例如 $i < 2025$，则考虑 $i = i$ 得到 $2025^2 \le i^2 + 2025$ 不可能成立）。所以必有 $A_{2025} = 2025$
3. 注意到对于任意 i（特别是 $i < 2025$），选取 $j = 2025$ 得

   $A_i \times 2025 \le i \times 2025 + 2025$，则$A_i \le i + 1, (1 \le i \le 2024)$

   这说明对于前 2024 个位置，客流上限最多只能比序号大 1。
4. 实际上可以证明，对于 $1 \le i \le 2024$
   - 要么保持 $A_i = i$（即“原位”），
   - 要么与其后一个位置交换，即令

     $$A_{i+1} = i$$

   并且这样的相邻交换不能重叠（即如果在位置 i 做了交换，则位置 $i + 1$ 就固定下来，不能再和 $i + 2$ 做交换）。

因此，问题转化为：对于 $1 \le i \le 2024$，有多少种方法从位置 1 到 2024 中选取一组不相邻的位置进行交换，即令 $A_i = i + 1$ 且 $A_{i+1} = i$，而对于没有发生交换的位置，必须保持 $A_i = i$。

设 $f(k)$ 为在一条长为 k 的线段上选取不相邻的位置的方案数。那么有经典递推

$f(0)=1,\quad f(1)=1,\quad f(k)=f(k-1)+f(k-2)\quad (k\geq 2)$ ，也就是斐波那契数列。

但注意这里的编号问题：

设 $F_1=F_2=1$；那么在小规模实验中，对于 $n=3$ 我们得到 3 种合法排列，对应 $f(2)=F_3=2$ 若最后位置固定。

但实际上，我们可以将整个排列构造看作在前 2024 个位置中做不相邻交换的选择，而方案数正好为 $F_{1013+1}=F_{1013}$

由于条件 $A_i\leq i+1$ 实际上只有在 $i\leq 1012$时可能取到 $i+1$，故有效的可交换区间仅在前 1012 个位置可以自由交换，每次交换固定两个位置。

总结证明可知，合法排列的数目就是在前 1012 个位置中选取不相邻位置进行交换的方案数。

C: 可分解的正整数 （1000）

问题描述
判断给定正整数能否表示为长度≥3的连续整数序列之和。

输入格式

- 第一行：正整数N。
- 第二行：N个正整数$A_1, A_2, \ldots, A_N$。

输出格式
可分解的正整数数量。

样例输入

```
3
3 6 15
```

样例输出

```
3
```

评测用例规模

$1\leq N\leq 10^5$，$1\leq A_i\leq 10^9$。

题解

打一下表发现 10^6 以内所有数除了 1 以外都可以这样分解，因此答案即为非 1 的数的数量

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

using namespace std;
void solve()
{
    int n;
    cin >> n;
    int ans = n;
    for (int i = 0; i < n; i++)
```

```
    {
        int x;
        cin >> x;
        if (x == 1)
            ans--;
    }
    cout << ans << "\n";
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    // std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```

D: 产值调整（1000）

问题描述

三矿山产值A, B, C每年按以下规则调整K次：

1. $A' = \lfloor (B + C)/2 \rfloor$
2. $B' = \lfloor (A + C)/2 \rfloor$
3. $C' = \lfloor (A + B)/2 \rfloor$

输入格式

- 第一行：测试用例数T。
- 每行：A, B, C, K。

输出格式

调整后的A, B, C。

样例输入

```
2
10 20 30 1
5 5 5 3
```

样例输出

```
25 20 15
5 5 5
```

评测用例规模

$1 \le T \le 10^5$，$1 \le A, B, C, K \le 10^9$。

题解

观察发现 A, B, C 都在向 $\frac{A+B+V}{3}$ 收敛，且速度很快，所以模拟一下直到三个相等即可。

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

using namespace std;

using ll = long long;
void solve()
{
    ll A, B, C, K;
    cin >> A >> B >> C >> K;
    while (K--)
    {
        ll NA = (B + C) >> 1;
        ll NB = (A + C) >> 1;
        ll NC = (A + B) >> 1;
        A = NA, B = NB, C = NC;
        if (A == B && B == C)
            break;
    }
    cout << A << " " << B << " " << C << "\n";
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```

E: 画展布置（1200）

问题描述
从N幅画中选M幅排列，最小化艺术价值变化程度$L = \sum_{i=1}^{M-1} |B_{i+1}^2 - B_i^2|$。

输入格式

- 第一行：N和M。
- 第二行：N个艺术价值$A_1, A_2, \ldots, A_N$。

输出格式
L的最小值。

评测用例规模

$2 \le M \le N \le 10^5; 1 \le A_i \le 10^5$。

题解

题目要求从 N 幅画中选出 M 幅，并排成一列，使得艺术价值变化程度$L = \sum_{i=1}^{M-1} \left| B_{i+1}^2 - B_i^2 \right|$ 最小。

注意到只要将选中的画按某种顺序排列，如果能让各个相邻画作的 “平方值” 单调变化，则$L = B_M^2 - B_1^2$,

其中 B_1^2 和 B_M^2 分别是选中画作中最小和最大的平方值。

因为对于任意一个选定的集合，无论中间顺序如何，若将它们重新排序为“平方值”递增的顺序，其变化总和必定为 $\max(B_i^2) - \min(B_i^2)$

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

using namespace std;

using ll = long long;
void solve()
{
    int N, M;
    cin >> N >> M;
    vector<ll> A(N), S(N);
    for (int i = 0; i < N; i++)
    {
        cin >> A[i];
        S[i] = A[i] * A[i];
    }

    sort(S.begin(), S.end());
    ll ans = LLONG_MAX;
    for (int i = 0; i + M - 1 < N; i++)
    {
        ll diff = S[i + M - 1] - S[i];
        if (diff < ans)
            ans = diff;
    }

    cout << ans << "\n";
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    // std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```

F: 水质检测（1200)

问题描述
在$2 \times n$河床上添加最少检测器，使所有检测器连通。

输入格式

- 两行：每行长度为n的字符串，`#`表示检测器，`.`表示空白。

输出格式
最少需添加的检测器数。

样例输入

```
.# #.....#
.# .# .#...
```

样例输出
`5`

评测用例规模

$1 \leq n \leq 10^6$

题解

记录一下上一个联通块是只有上还是只有下还是两个都有以及它的位置即可

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

using namespace std;

using ll = long long;
void solve()
{
    string s1, s2;
    cin >> s1 >> s2;
    int n = s1.size();
    int sta = 0, p = 0;
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        if (sta)
        {
            if (s1[i] == '#' && s2[i] == '#')
            {
                ans += i - p - 1;
                sta = 3;
                p = i;
            }
            else if (s1[i] == '#')
            {
                if (sta == 2)
                    ans += i - p;
```

```
                else
                    ans += i - p - 1;
                sta = 1;
                p = i;
            }
            else if (s2[i] == '#')
            {
                if (sta == 1)
                    ans += i - p;
                else
                    ans += i - p - 1;
                sta = 2;
                p = i;
            }
        }
        else
        {
            if (s1[i] == '#' || s2[i] == '#')
            {
                p = i;
                if (s1[i] == '#' && s2[i] == '#')
                    sta = 3;
                else if (s1[i] == '#')
                    sta = 1;
                else
                    sta = 2;
            }
        }
    }
    cout << ans << "\n";
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    // std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```

G: 生产车间（1800）

问题描述

树形结构设备网络，叶节点生产材料，根节点打包成品。调整节点使所有节点产能不超限，求根节点最大打包量。

输入格式

- 第一行：设备数n。
- 第二行：各节点权值$w_1, w_2, \ldots, w_n$。

- 后续$n-1$行：树边。

输出格式

根节点的最大成品量。

样例输入

```
9
9 7 3 7 1 6 2 2 7
1 2
1 3
2 4
2 5
2 6
6 7
6 8
6 9
```

样例输出

`8`

评测用例规模

$1 \leq n \leq 10^3; 1 \leq w_i \leq 10^3$

题解

跑一下树上背包即可

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

using namespace std;

using ll = long long;

void solve()
{
    int n;
    cin >> n;
    vector<int> w(n + 1);
    for (int i = 1; i <= n; i++)
    {
        cin >> w[i];
    }
    vector<vector<int>> adj(n + 1);
    for (int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    function<int(int, int, int)> dfs = [&](int u, int now, int fa) -> int
```

```
    {
        now = min(now, w[u]);
        if (adj[u].size() == 1)
            return w[u] > now ? 0 : w[u];
        vector<int> f(now + 1);
        for (auto &v : adj[u])
        {
            if (v == fa)
                continue;
            vector<int> g = f;
            for (int i = 1; i <= now; i++)
            {
                int val = dfs(v, i, u);
                f[i] = max(g[i], g[i - val] + val);
            }
        }
        return f[now];
    };
    cout << dfs(1, w[1], 0) << '\n';
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    // std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```

H: 装修报价 （1600）

问题描述

老王计划装修房子，于是联系了一家装修公司。该公司有一套自动报价系统，只需用户提供 N 项装修相关费用 $A_1, A_2, \ldots, A_N$，系统便会根据这些费用生成最终的报价。

然而，当老王提交数据后，他发现这套系统的运作方式并不透明：系统只会给出一个最终报价，而不会公开任何运算过程或中间步骤。公司对此解释称，这套系统会依据某种内部算法，在每对相邻数字之间插入 $+$（加法）、$-$（减法）或 $\oplus$（异或）运算符，并按照特定优先级规则计算结果：异或运算优先级最高，其次是加减。

为了验证系统报价是否合理，老王决定模拟其运作方式，尝试每种可能的运算符组合，计算出所有可能出现的结果的总和。如果最终报价明显超出这个范围，他就有理由怀疑系统存在异常或误差。

现在，请你帮老王算出所有可能的结果的总和。由于该总和可能很大，你只需提供其对 $10^9 + 7$ 取余后的结果即可。

输入格式

- 第一行输入一个整数 N，表示装修相关费用的项数。
- 第二行输入 N 个非负整数 $A_1, A_2, \ldots, A_N$，表示各项费用。

输出格式

输出一个整数，表示所有可能的总和对 10^9+7 取余后的结果。

示例输入

```
3
0 2 5
```

示例输出

```
11
```

示例说明

对于输入样例中的三个数 $A=[0,2,5]$，所有可能的运算符组合共有 9 种。计算结果如下：

- $0 \oplus 2 \oplus 5 = 7$
- $0 \oplus 2 + 5 = 7$
- $0 \oplus 2 - 5 = -3$
- $0 + 2 \oplus 5 = 7$
- $0 + 2 + 5 = 7$
- $0 + 2 - 5 = -3$
- $0 - 2 \oplus 5 = -7$
- $0 - 2 + 5 = 3$
- $0 - 2 - 5 = -7$

所有结果的总和为：

$$7+7+(-3)+7+7+(-3)+(-7)+3+(-7)=11$$

11 对 10^9+7 取余后的值依然为 11，因此，输出结果为 11。

评测用例规模

$1 \le N \le 10^5, 1 \le A_i \le 10^9$

题解

注意到：

- 在所有 $\oplus$ 连续的段内，其结果就是该段内所有数的异或值；
- 在相邻段之间的运算符为加或减，由于加减具有线性性质（先计算异或段，后做加减运算）可以发现，最终结果实际上为各“段”值的加权和，其中只有最左边那一段的符号“固定为 +”，而后续各段由于加减符号正负会相互抵消后求和。

具体看“分段”：

- 定义：设在相邻位置处如果选用非 $\oplus$ 运算符，则视为“断开”，形成新段。
- 因为每个位置独立选运算符，所以可以将所有可能的运算符组合看成对 $N-1$ 个空位的选择，每个位置可以“接续”（即选 $\oplus$）或“断开”（即选加或减），而“断开”时又有 2 种符号选择。

观察发现：

- 若整个序列中没有断开，则只有一段，其结果为
  $G = A_1 \oplus A_2 \oplus \cdots \oplus A_N$
- 若第一个断开出现在位置 j（也就是说从 A_1 到 A_j 连续使用 $\oplus$），则第一段的“段值”为
  $G_1 = A_1 \oplus A_2 \oplus \cdots \oplus A_j$

而后面不论如何选择（剩余位置随意），在加减阶段由于正负相互抵消，其对总和的贡献在对所有符号取和时，只有第一段的值会“留下”。
具体地：

- 固定第一断开出现在 j 的情况下，对于前 $j-1$ 个间隙，必须全选 $\oplus$（1 种方式）；
- 第 j 个空位选断开，有 2 种符号选择；
- 对于位置 $j+1, \ldots, N-1$ 每个空位可任意选（3 种方式），总数为3^{N-1-j}.

故满足“第一断开位置为 j "的方案数为$2 \cdot 3^{N-1-j}$.

对于这些方案，最终计算（加减累加时）会固定地把第一段 G_1 加入结果中（其他段各自正负总和为 0）。

于是，把所有方案按照是否出现断开以及第一断开的位次分情况讨论，最后所有可能最终结果的总和 S 为

$$S = \underbrace{G}_{\text{无断开}} + \sum_{j=1}^{N-1} \left(2 \cdot 3^{N-1-j}\right) \cdot \left(A_1 \oplus A_2 \oplus \cdots \oplus A_j\right).$$

这里 $G = A_1 \oplus A_2 \oplus \cdots \oplus A_N$。

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

using namespace std;

using ll = long long;

const ll mod = 1000000007;
ll qmi(ll x, ll k, ll p = mod)
{
    x %= p;
    ll res = 1;
    while (k > 0)
    {
        if (k & 1)
            res = (res * x) % p;
        x = (x * x) % p;
        k >>= 1;
    }
    return res;
}
void solve()
{
    int n;
    cin >> n;
    vector<ll> a(n + 1);
    vector<ll> preXor(n + 1, 0);
    for (int i = 1; i <= n; i++)
```

```
    {
        cin >> a[i];
        preXor[i] = preXor[i - 1] ^ a[i];
    }
    vector<ll> p3(n);
    p3[0] = 1;
    for (int i = 1; i < n; i++)
    {
        p3[i] = (p3[i - 1] * 3) % mod;
    }
    ll ans = preXor[n] % mod;
    for (int i = 1; i < n; i++)
    {
        ll t = ((2ll * p3[n - 1 - i]) % mod * preXor[i]) % mod;
        ans = (ans + t) % mod;
    }
    cout << (ans % mod + mod) % mod << "\n";
}
int main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    int _ = 1;
    // std::cin >> _;
    while (_--)
    {
        solve();
    }
    return 0;
}
```