中南民族大学2026除夕红包赛题解

题面地址：https://ac.nowcoder.com/acm/contest/128827

现在时间：[2026/02/16 22:30]

将揭示——

幕后真相：

本场比赛难度分布大致如下：

- Easy：除夕大礼包，我们仍未知道那天所探索的Tr的秘密，半加器？半减器！
- Normal：言灵编织未来，超超超超超优雅的超长二进制数字
- Hard：春之能量，芊大王学AE，闪光の哈撒韦
- Lunatic：无

A.芊大王学AE

出题人：xuanzelllza007

题面遍了一个故事，我们把故事抛开重新翻译一下这个题
给一个 $n*m*h$的由 01 构成三维图，要求包含切只包含一个联通图，同时要求每一个 1 不在同一个平面内呈十字。
因为 $n\ m\ h$ 很小很容易想到搜索整张图。这里我们采用dfs，可以轻松求出是否只含一个联通图。
接下来考虑后半的额外条件如何解决。我们在搜索的时候可以将 $dx\ dy$ 数组变得规律，让我们好访问一个 1 在一条直线上是否都存在相邻的 1 。**具体规律可以自己决定好写就行!** 于是我们在dfs遍历到每一个

1 就检查一遍是否存在两个方向都被两个 1 相夹，从而判断合不合法。
最后统计答案，我们可以直接在dfs的时候对那些没有和其他 1 相邻的面进行统计即可。另外值得注意的是，**最外面一层一样可以提供频道。**
下方给出std代码

```
#include<bits/stdc++.h>
using namespace std;
const int dx[6] = {1 , 0 , 0 , -1 , 0 , 0};
const int dy[6] = {0 , 1 , 0 , 0 , -1 , 0};
const int dz[6] = {0 , 0 , 1 , 0 , 0 , -1};
void solve()
{
    int n , m , h;
    cin >> n >> m >> h;
    vector<vector<vector<int>>> a(n + 2 , vector<vector<int>>(m + 2 , vector<int>
(h + 2 , 0))), vis(n + 2 , vector<vector<int>>(m + 2 , vector<int>(h + 2 , 0)));

    for(int k = 1 ; k <= h ; k ++)
        for(int i = 1 ; i <= n ; i ++)
            for(int j = 1 ; j <= m ; j ++)
                cin >> a[i][j][k];
    int ans = 0;
    bool iseffect = 1;
    auto dfs = [&](auto && self , int x , int y , int z) -> void
    {
        vis[x][y][z] = 1;
        int jud = 0;
        for(int i = 0 ; i < 3 ; i ++)
            jud += a[x + dx[i]][y + dy[i]][z + dz[i]] && a[x - dx[i]][y - dy[i]]
[z - dz[i]];
        if(jud >= 2){
            iseffect = 0;
            //cout << "4 connect! " << x << " " << y << " " << z << endl;
        }
        for(int i = 0 ; i < 6 ; i ++)
        {
            int newx = x + dx[i] , newy = y + dy[i] , newz = z + dz[i];
            if(newx > n || newx < 1 || newy > m || newy < 1 || newz > h || newz <
1 || !a[newx][newy][newz]){
                ans++;
                continue;
            }
            if(!vis[newx][newy][newz])self(self , newx , newy , newz);
        }
    };
    bool search = 0;
    for(int k = 1 ; k <= h ; k ++)
        for(int i = 1 ; i <= n ; i ++)
            for(int j = 1 ; j <= m ; j ++)
                if(a[i][j][k] && !vis[i][j][k]){
                    if(search){
                        iseffect = 0;
                    }
                    else {
                        dfs(dfs , i , j , k);
```

```
                        search = 1;
                    }
                }
    cout << (iseffect ? ans * 32 : -1) << endl;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0) , cout.tie(0);
    int t = 1;
    cin >> t;
    while (t--)
    {
        solve();
    }
    return 0;
}
```

时间复杂度和空间复杂度均为 $O(n*m*h)$

B.半加器？半减器！

出题人：连琰

首先考虑半加和半减的性质，由于不需要进位，所以可以拆位看成独立的2进制计算。发现半加和半减的效果是一样的。刚好满足按位异或的性质。
题目上又说是32位有符号位的运算，因此就是int型的异或运算，直接输出即可。

std：

```
#include <iostream>

void solve()
{
    int x, y;std::cin>>x>>y;
    std::cout << (x ^ y) << ' ' << (x ^ y) << '\\n';
}

int main()
{
    std::ios::sync_with_stdio(0);
    std::cin.tie(0);
    int T = 1;
    std::cin >> T;
    while (T--)
        solve();
    return 0;
}
```

C.超超超超超优雅的超长二进制数字

出题人：Whalica

当题目中出现 **每个位置可能有不同的情况，且不同位置间的元素可以相互影响** 的时候，**可以考虑** 动态规划，而本题的条件恰好可以设计出合理的转移方程。

具体地，我们通常会设 $dp_{i,j}$ 表示从高位到低位考虑到第 i 位时，以 $j \in \{0,1\}$ 结尾时，共有 $dp_{i,j}$ 个二进制数满足要求。这样一来，我们就可以用往低位插数的方式来进行转移：

- 当第 i 位为 0 时，对更高一位的数没有任何要求，所以更高一位可以是 0 或 1，转移方程为：`dp[i][0] = dp[i - 1][0] + dp[i - 1][1]`。
- 当第 i 位为 1 时，此时因为题目限制，两个 1 不能相邻，所以更高一位的数不能是 1，转移方程为：`dp[i][1] = dp[i - 1][0]`。

如此从最高位到最低位转移完后，答案就是 `dp[n][0] + dp[n][1]`，答案对 998244353 取模。复杂度 $O(n)$。

std:

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

using i64 = long long;
using u64 = unsigned long long;

constexpr i64 P = 998244353;

void solve() {
    int n;
    std::cin >> n;

    std::vector<std::array<i64, 2>> dp(n + 1, {0, 0});
    dp[1] = {1, 1}; // 当 i == 1 时，此时可以是单独的 0 或 1，所以方案数都是 1 种。
    for (int i = 2; i <= n; i++) {
        dp[i][0] = (dp[i - 1][0] + dp[i - 1][1]) % P;
        dp[i][1] = dp[i - 1][0];
    }

    std::cout << (dp[n][0] + dp[n][1]) % P << "\\n";
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    std::cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

D.闪光の哈撒韦

出题人：西江

本题的本质是一个尼姆游戏，每组飞弹到古斯塔夫的距离减去一等价于尼姆游戏中的每堆石子数量。异或判断尼姆积即可。博弈论必学经典模型

后退次数和离战区边缘距离是多余数据。因为当某方后退d距离时，另一方可以前进d距离，使得局面不变。而又因为不限制前进次数但限制了后退次数，该游戏总会结束。所以在最优策略下，后退无法影响游戏状态。

另外由于上限卡到了2^{31}，不开ll是过不了的。

代码：

```
#include <bits/stdc++.h>
using namespace std;
#define int long long

int n,k1,k2;

void solve(){
    cin>>n>>k1>>k2;
    int a,b,c;
    int t,res;
    for(int i=1;i<=n;++i){
        cin>>a>>b>>c;
        t=abs(a-c)-1;
        if(i==1)res=t;
        else res^=t;
    }
    cout<<(res?"让他们见识一下吧,马夫蒂!":"我变弱了,是因为琪琪吗?")<<endl;
}

signed main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    int _=1;
    cin>>_;

    while(_--)solve();

    return 0;
}
```

E.言灵编织未来

出题人：雨沐

这道题的核心在于如何**在满足回文结构和特定子串（"promise"）的同时，通过贪心策略精确凑出目标权值。**

1. 核心构造逻辑

首先，我们需要明确目标：构造一个回文串 t'，其包含子串 `promise`，且总权值等于原串 t 的权值之和 W。

- **权值计算**：遍历原串 t，计算 $W = \sum(t_i - \text{'a'} + 1)$。
- **结构基准**：由于必须包含 `promise` 且要是回文串，最简单的办法是构造一个以 `promise` 为核心的回文中心。
- **奇偶性匹配**：回文串除了中心字符外，两侧字符必须成对出现。这意味着两侧增加的权值必然是偶数。因此，我们必须根据 W 的奇偶性来选择不同权值的"核心"。

---

2. 核心状态与预设

为了应对总权值的奇偶性，我们预设两个基本的回文"种子"：

- **方案 A（奇数核心）**：`"promisesimorp"`
  - **权值和**：**185**（奇数）。
  - **适用场景**：当目标权值 W 为奇数时。
- **方案 B（偶数核心）**：`"promiseesimorp"`
  - **权值和**：**190**（偶数）。
  - **适用场景**：当目标权值 W 为偶数时。

---

3. 具体填充策略

一旦选定了核心（$Core$），剩余需要填充的权值为 $R = W - \text{Weight}(Core)$。由于奇偶性已对齐，$R$ 必定是一个非负偶数。

- **对半分摊**：将剩余权值平分为两份，即每侧需要填充 $half = R/2$。
- **贪心填充**：为了保证 t' 的长度不超过 $2|t|$，我们需要用最少的字符凑出 $half$。
  - 优先使用权值最大的字符 `'z'`（权值 26）。
  - 计算需要多少个 `'z'`：`num_z = half / 26`。
  - 计算余下的权值：`rem = half % 26`，若 `rem > 0`，则补一个对应的字符。
- **最终拼接**：

  $t' = \text{Side}^R + Core + \text{Side}$

  其中 Side 是由上述贪心策略生成的字符串，Side^R 是其翻转。

复杂度分析

- **时间复杂度**：$O(|t|)$。

  计算原串权值需要 $O(|t|)$，构造新串的过程由于长度限制在 $2|t|$ 以内，填充和拼接操作也是线性的。

- **空间复杂度**：$O(|t|)$。

  主要用于存储构造出来的目标字符串。

参考代码(C++)

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

using namespace std;
using i64 = long long;

string F(string t, int sum)
{
    string res = "";
    sum >>= 1;
    while (sum)
    {
        int x = min(26, sum);
        sum -= x;
        res += char(x + 'a' - 1);
    }
    string rvres(res.rbegin(), res.rend());
    return rvres + t + res;
}
void solve()
{
    string s, t = "";
    cin >> s;
    int sum = 0;
    for (int i = 0; i < s.size(); i++)
    {
        sum += s[i] - 'a' + 1;
    }
    string pre[2] = {"promisesimorp", "promiseesimorp"};
    int arr[2] = {185, 190};
    if (sum & 1)
    {
        t = pre[0];
        sum -= 185;
    }
    else
    {
        t = pre[1];
        sum -= 190;
    }

    string ans = F(t, sum);
    cout << ans << '\n';

}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    int _T;
    cin >> _T;
    while (_T--)
    {
        solve();
```

```
    }
    return 0;
}
```

F.我们仍未知道那天所探索的Tr的秘密

出题人：Rorislux

```
// Created on: 2026-02-09 18:56:18

#include <bits/stdc++.h>

using namespace std;
using i64 = long long;

#if !defined(ONLINE_JUDGE) && defined(LOCAL)
// #if !defined(ONLINE_JUDGE)
#include "D:\\VSC P\\ide\\cpp\\cpphead\\helper.h"
#else
#define dbg(...) ;
#define local_go_m(x) \\
    int c;            \\
    cin >> c;         \\
    while (c--)       \\
    x()
#define local_go(x) x()
#endif

void go()
{
    i64 n, m, x, y;
    cin >> n >> m >> x >> y;
    auto qpow = [&](i64 a, i64 b)
    {
        i64 res = 1;
        while (b)
        {
            if (b & 1)
                res = res * a % 998244353;
            a = a * a % 998244353;
            b >>= 1;
        }
        return res;
    };
    i64 k = n + m - x - y;
    i64 res = qpow(2, k);
    res -= (x == 0 ? 1 : 0);
    cout << res << endl;
}

int main()
{
```

```
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    local_go_m(go);
    // local_go(go);
    return 0;
}
```

由题意可知，有 $k = n + m - x - y$ 个种子未被确定，对于每个种子有选或不选两种可能，即有 **r** $es = 2^k$ 种种子组合，特别注意，当 $x = 0$ 时， res 中全不选的情况要排除，即 res—

观察数据范围后，发现需使用**快速幂**通过本题

G.春之能量

出题人：CubeCat

题目要求转换一下就是把一棵树拆成很多条链，使得链上所有点的权值之和大于等于 k ，问这样的拆分方式是否存在

对于这类树上的问题，可以从根开始考虑，也就是不考虑当前节点和父亲之间的关系。

可以容易发现对于当前的节点，只存在4种情况：

1. 所有子节点都可以在不与当前节点连接的情况下满足要求，这也就表明于不与父节点相连对于节点没有影响。但是题目要求值大于某个数，所以肯定是越大越好，所以挑一个值(子节点所在链)最大的和父节点相连就是最好的了
2. 有1个不满足要求，此时直接与父节点相连即可
3. 有2个不满足，这里是本题难点。有两个的话就只能尝试把这两个链通过当前节点组合，如果组合后还是不满足就可以直接给出 "No" 了。同时需要注意的是这样连了之后就是闭合的了，也就是不能再向上传递了，代码中用up数组体现这一点
4. 有3个及以上肯定也是不行的，直接是"No"

同时有一个共同的性质是，无论如何只要不是上面第3种情况都是可以向上传递的

这个树虽然没有明确指明根，但是事实上最后结果和根的选取是无关的，只要你 dfs 的条件关系没错最后结果肯定是对的，代码是选取1为根

```
#include <bits/stdc++.h>
#define fore(i, l, r) for (int i = l; i < r; i++)
using namespace std;
#define endl '\n'
#define inf 0x3f3f3f3f
typedef pair<int, int> pii;

void solve() {
    int n, k;
    cin >> n >> k;
    vector<vector<int>> p(n + 1);
    vector<int> val(n + 1);
    int u, v;
    fore(i, 1, n + 1) cin >> val[i];
    fore(i, 1, n) {
        cin >> u >> v;
```

```
        p[u].emplace_back(v);
        p[v].emplace_back(u);
    }
    vector<int> dp(n + 1, 0);
    int flag = true;
    vector<int> up(n + 1, true);
    auto dfs = [&](auto&& dfs, int s, int fa) -> void {
        dp[s] = val[s];
        vector<int> can, must;
        for(int to : p[s]) {
            if(to == fa) continue;
            dfs(dfs, to, s);
            if(up[to] == false) continue;
            if(dp[to] < k) must.emplace_back(dp[to]);
            else can.emplace_back(dp[to]);
        }

        sort(can.begin(), can.end(), greater<int>());
        if(must.size() >= 3) {
            flag = false;
            return;
        }else if(must.size() == 2) {
            up[s] = false;
            if(must[0] + must[1] + val[s] < k) {
                flag = false;
                return;
            }
        }else if(must.size() == 1) {
            dp[s] += must[0];
        }else {
            if(!can.empty()) dp[s] += can[0];
        }

        if(s == 1 && dp[s] < k && up[s] == true) flag = false;
    };

    dfs(dfs, 1, 0);
    if(flag) cout << "Yes" << endl;
    else cout << "No" << endl;
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    solve();
    return 0;
}
```

H.除夕大红包

出题人：去玩而已哦

一、题目简述

给定一个红包总金额 `total`（单位：分）和参与人数 `n`，需要将红包分给 `n` 个人，规则如下：

1. 每个人至少分到 1 分
2. 如果不能平均分，多出来的钱从第一个人开始，每人多分 1 分，直到分完

要求输出每个人最终能分到的金额。

二、解题思路

这道题的核心其实就是**整数除法和取余数**。

1. 平均分配的基础金额

如果将 `total` 平均分给 `n` 个人，每个人至少可以得到：

```
base = total / n
```

这是整数除法，自动向下取整。

2. 不能平分的剩余金额

平均分配后，可能会剩下一部分钱，数量为：

```
remainder = total % n
```

这表示还有 `remainder` 分钱没有分完。

3. 按规则分配剩余的钱

根据题目规则：

- 前 `remainder` 个人，每人可以在 `base` 的基础上多拿 1 分
- 后面的人员，只拿 `base` 分

也就是说：

- 第 `0` 到 `remainder - 1` 个人：`base + 1`
- 第 `remainder` 到 `n - 1` 个人：`base`

三、算法步骤

1. 读入 `total` 和 `n`
2. 计算：
    - `base = total / n`
    - `remainder = total % n`
3. 从第一个人遍历到第 `n` 个人：
    - 如果当前编号小于 `remainder`，输出 `base + 1`
    - 否则输出 `base`

4. 按要求用空格分隔输出结果

四、复杂度分析

- **时间复杂度**：`O(n)`
  需要依次输出 `n` 个结果
- **空间复杂度**：`O(1)`
  只使用了常数级额外变量，没有使用数组或额外容器

五、参考实现（C++）

```cpp
#include <iostream>
using namespace std;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    long long total;
    int n;

    // 读取输入
    cin >> total >> n;

    // 计算每人基础金额和余数
    long long base = total / n;         // 平均每人分得的基础金额
    long long remainder = total % n;    // 无法平分的余数

    // 输出结果
    for (int i = 0; i < n; i++)
    {
        if (i < remainder)
        {
            // 前 remainder 个人多分 1 分
            cout << base + 1;
        }
        else
        {
            // 其他人分得基础金额
            cout << base;
        }

        // 输出空格（最后一个人后不输出空格）
        if (i < n - 1)
        {
            cout << " ";
        }
    }
    cout << endl;

    return 0;
```

```
}
```

幕后的幕后

感谢大家对本题解的阅读，也感谢大家的参赛

希望大家能在校赛的时候再创辉煌

如有发现疏漏，请联系群内管理员