CF 993 div3

https://codeforces.com/contest/2050

B. Transfusion

https://codeforces.com/contest/2050/problem/B

- 理解操作背后的性质。$Decrease\ a_{i-1}\ by\ 1, then\ increase\ a_{i+1}\ by\ 1.$ 和 $Decrease\ a_{i+1}\ by\ 1, then\ increase\ a_{i-1}\ by\ 1.$ 两个操作任选一，背后的性质是什么。
- 这两个操作都是对一个数 $+1$ ，对一个数 -1 ，所以不管这两个操作，以什么顺序执行，每个操作分别执行多少次，整个数组总和是不变化的。而最后的目标是使每个数相等，所以要满足 $sum\ \%\ n = 0$ 。而且最终每个元素的值是确定的即 $a[i] = sum\ /\ n$ 。
- 不仅如此，当选择 $i\ \%\ 2 = k$ 处的数进行操作时，那么受影响的是 $j\ (j\ \%\ 2 = 1 - k)$ 处的数，也就是说，下标为奇数的所有数组成的子序列和下标为偶数的所有数组成的子序列也和整个数组一样，满足 $odd\_sum\ \%\ odd\_cnt = 0$ 和 $even\_sum\ \%\ even\_cnt = 0$ 。并且 $(sum\ /\ n) * odd\_cnt = odd\_sum$ 且 $(sum\ /\ n) * even\_cnt = even\_sum$ 。
- 当一个数组满足上述条件时，便可以通过操作达到目标。

时间复杂度 $O(n)$

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

void f() {
    int n;
    cin >> n;
    vector<int> a(n + 1);
    long long sum = 0;
    long long odd = 0, even = 0, cnt_odd = 0, cnt_even = 0;
    for (int i = 1; i <= n; i ++) {
        cin >> a[i];
        sum += a[i];
        if ((i % 2) == 0) {
            even += a[i];
            cnt_even ++;
        } else {
            odd += a[i];
            cnt_odd ++;
        }
    }
    if ((sum % n) != 0) {
        cout << "NO" << endl;
        return ;
    }

    int target = sum / n;
```

```
    if ((target * cnt_even) == even && (target * cnt_odd) == odd) cout << "YES" << endl;
    else cout << "NO" << endl;
    return ;
}

int main() {
    int t;
    cin >> t;
    while (t --) f();
    return 0;
}
```

E. Three Strings

https://codeforces.com/contest/2050/problem/E

$f[i][j]$ 表示用 a 字符串的前 i 个字母和 b 字符串的前 j 个字母组成 c 字符串的前 $i+j$ 个字母时，最少需要的替换数目。

当选择 a 字符串的第 i 个字母放到 c 字符串末尾时，$f[i][j] = f[i-1][j] + (a[i] != c[i+j])$

当选择 b 字符串的第 j 个字母放到 c 字符串末尾时，$f[i][j] = f[i][j-1] + (b[j] != c[i+j])$

时间复杂度 $O(n^2)$

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

void f() {
    string a, b, c;
    cin >> a >> b >> c;
    int n = a.size();
    int m = b.size();
    vector<vector<int>> f(n + 1, vector<int>(m + 1, INT_MAX));
    f[0][0] = 0;

    for (int i = 1; i <= n; i ++) {
        f[i][0] = f[i - 1][0] + (a[i - 1] != c[i - 1]);
    }

    for (int j = 1; j <= m; j ++) {
        f[0][j] = f[0][j - 1] + (b[j - 1] != c[j - 1]);
    }

    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= m; j ++) {
            f[i][j] = min(f[i - 1][j] + (a[i - 1] != c[i + j - 1]), f[i][j - 1] + (b[j -
1] != c[i + j - 1]));
        }
    }
    cout << f[n][m] << "\n";
}
```

```
int main() {
    int t;
    cin >> t;
    while (t --) f();
    return 0;
}
```

F. Maximum modulo equality

https://codeforces.com/contest/2050/problem/F

推导：

对于一系列的查询 $[l,r]$ ，每次查询求最大的 m ， 使得 $a_l \% m = a_{l+1} \% m = \cdots = a_r \% m$

即：

$a_l \% m = c$

$a_{l+1} \% m = c$

$\cdots$

$a_r \% m = c$

令 $d_1 = a_{l+1} - a_l$, $d_2 = a_{l+2} - a_{l-1}$, $\ldots$, $d_{r-l-2} = a_r - a_{r-1}$

把上面的式子两两相减可以发现，$d_i \% m = 0$

也就是说，假设 m 满足 $a_l \% m = a_{l+1} \% m = \cdots = a_r \% m$ ，那么有
$d_1 \% m = d_2 \% m = \cdots = d_{r-l-2} \% m = 0$

所以 m 是这些 d_i 的公约数，而要求一个最大的 m ，那么就是求这些 d_i 的最大公约数。

我们可以用线段树在建树的阶段维护好每个区间的差值的最大公约数，在查询的时候，每次使用 $O(logn)$ 的时间就能查询出结果。注意一些细节，代码中处理出来，个数为 1 的区间的公约数会是差值本身，根据题目要求，特判一下 0 即可，对于一段区间都是相同的数字时，差值为 0 ，gcd 也是 0 。

时间复杂度 $O(nlogn + q * logn)$

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

void buildtree(int k, int l, int r, vector<int>& a, vector<int>& f) {
    if(l == r) {
        f[k] = a[l];
        return ;
    }
    int mid = (l + r) >> 1;
    buildtree(k * 2, l, mid, a, f);
    buildtree(k * 2 + 1, mid + 1, r, a, f);
    f[k] = gcd(f[k * 2], f[k * 2 + 1]);
}
```

```
int query(int k, int l, int r, int s, int t, vector<int>& f) {
    if(l == s && r == t) return f[k];
    int mid = (l + r) >> 1;
    if(t <= mid) {
        return query(k * 2, l, mid, s, t, f);
    } else if(s > mid) {
        return query(k * 2 + 1, mid + 1, r, s, t, f);
    } else {
        return gcd(query(k * 2, l, mid, s, mid, f), query(k * 2 + 1, mid + 1, r, mid + 1,
t, f));
    }
}

void f() {
    int n, q;
    cin >> n >> q;
    vector<int> aa(n + 1);
    vector<int> a(n + 1);
    vector<int> f(4 * n);
    for (int i = 1; i <= n; i ++) cin >> aa[i];
    for (int i = 1; i < n; i ++) a[i] = abs(aa[i + 1] - aa[i]);
    buildtree(1, 1, n, a, f);
    while (q --) {
        int l, r;
        cin >> l >> r;
        if (l == r) cout << 0 << " ";
        else cout << query(1, 1, n, l, r - 1, f) << " ";
    }
    cout << endl;
}

int main() {
    int t;
    cin >> t;
    while (t --) f();
    return 0;
}
```

G. Tree Destruction【没做出来】

https://codeforces.com/contest/2050/problem/G

差一点冷静分析。

也是抓住题目所说的动作，它背后带来的影响。

删除 a 到 b 路径上的所有点，对于联通分量的数量变化（或者说贡献）的情况是怎么样的呢?

- 如果删除的是路径的端点 u ，那么对连通分量数量的贡献是 $d[u]-1$，其中 d 表示一个点的度数。这个怎么来的呢，因为是个无向树，所以一个点删除后，它会贡献出数量等于它的度数的连通分量，但是如果它是端点，说明有一条“度”的路径是要被删除的，那个联通分量会被删除掉。
- 同上，如果删除的是路径的中间点 u ，对连通分量数量的贡献是 $d[u]-2$ ，其中 d 表示一个点的度数。
- 我们可以统一起来，计算每个点的 $d[u]-2$ 最后对结果补回两个端点多删除的两个 1 ，也就是 $+2$ 。
- 具体做法就是一个树形 dp ，$dfs(u,fa)$ 表示当前遍历到的是 u 节点，它的父亲节点是 fa ，自底删除到 u 时贡献出来的联通分量个数最大是多少。
- 那么对于 $dfs(u,fa)$，显然 $dfs(u,fa)=max(dfs(p,fa))+d[u]$ ，其中 p 是 u 的儿子们。
- 对于最终要求的答案，我们用全局变量 ans 来维护。
- 对于 u 它既可以是自底向上一直删除的一条路到目前为止的“终点”，它也可以是自底向上删除的某两条路的最近公共祖先 lca ，所以对于第一种情况，只需要一个最大的 $d[p]$ 即可，对于第二种情况，只需要最大的两个 $d[p]$ 即可，也就是说，$ans=max(ans,\ d[u]+d[p1],\ d[u]+d[p1]+d[p2])$ ， 对于叶子节点就是没有儿子的情况，代码中对各种情况都做了处理。

时间复杂度 $O(n)$ ，预处理贡献为 $d[i]-2$ ，每个点遍历一次，用两个变量记录子树贡献的最大值和最小值，最后对答案补回 $+2$ 。

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> edges;
vector<int> d;
int ans;

int dfs(int u, int fa) {
    vector<int> t;
    int max_1 = 0, max_2 = 0;
    for (auto p : edges[u]) {
        if (p != fa) {
            int res = dfs(p, u);
            if (res > 0) {
                if (res > max_1) {
                    max_2 = max_1;
                    max_1 = res;
                } else if (res > max_2) {
                    max_2 = res;
                }
            }
        }
    }

    int single = d[u] + max_1;
    int lca = d[u] + max_1 + max_2;
    ans = max(ans, single);
    ans = max(ans, lca);
    return single;
}
```

```
void f() {
    int n;
    cin >> n;
    edges.clear();
    edges.resize(n);
    d.clear();
    d.resize(n, 0);
    for(int i = 0; i < n - 1; i ++) {
        int x, y;
        cin >> x >> y;
        x -= 1;
        y -= 1;
        edges[x].push_back(y);
        edges[y].push_back(x);
        d[x] ++;
        d[y] ++;
    }
    ans = -0x3f3f3f3f;
    for(int i = 0; i < n; i ++) d[i] -= 2;
    dfs(0, -1);
    cout << ans + 2 << endl;
    return ;
}

int main() {
    int t;
    cin >> t;
    while(t --) f();
    return 0;
}
```