# 兰州大学第一届“飞马杯”程序设计竞赛

## 题解

**2021.05.29**

| | |
| --- | --- |
| **规则** | ACM 个人赛 |
| **语言** | C/C++、Java、Python |
| **题目** | 12 |
| **时长** | 5 小时 |

**为了响应教育部提出“新工科”教育战略，培养和展示我大学生分析、解决问题和计算机编程的能力，鼓励和培养创新思维，丰富校园学术气氛，造就具有综合素质的面向 21 世纪的计算机人才，兰州大学将于 2021 年 05 月 29 日举办兰州大学第一届“飞马杯”程序设计竞赛，面向兰州大学全体学生。**

（感谢兰州大学信息科学与工程学院马俊老师提供的技术支持）

Accepted Wrong Answer Runtime Error Time Limit Exceeded
Memory Limit Exceeded Compile Error

# 目录

# A - ★★比赛新机制★★

**时间限制：** 1000ms
**空间限制：** 256MB
**难度：** ★★★☆☆ ☆☆☆☆☆

**标签：** 递推 前缀和

## 题解

记 $sum=\sum\limits_{i=1}^{n} a_i$，先考虑正序的情况：

若答题顺序为 $a_1,a_2,\dots,a_{n-1},a_n$，那么总罚时为 $S_1=na_1+(n-1)a_2+\dots+2a_{n-1}+a_n$；

若答题顺序为 $a_2,a_3,\dots,a_n,a_1$，那么总罚时为 $S_2=a_1+na_2+\dots+3a_{n-1}+2a_n$。

不难发现，$S_2=S_1-na_1+sum$，以此类推，可得 $S_{i+1}=S_i-na_i+sum$，反序同理。

于是，在每次递推时更新最小值即可。

时间复杂度 $O(n)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 500005
#define INF 123456789000000000
ll T, n, sum, isum, risum, a[maxn], ans;
int main()
{
    scanf("%lld", &T);
    while (T--) {
        scanf("%lld", &n);
        sum = isum = risum = 0;
        ans = INF;
        for (ll i = 1; i <= n; i++) {
            scanf("%lld", &a[i]);
            sum += a[i];
            isum += i * a[i];
            risum += (n - i + 1) * a[i];
        }
        for (ll i = n; i >= 1; i--) {
            isum += sum - n * a[i];
            ans = min(ans, isum);
        }
        for (ll i = 1; i <= n; i++) {
            risum += sum - n * a[i];
            ans = min(ans, risum);
        }
```

```
        printf("%lld\n", ans);
    }
    return 0;
}

```

# B - ★★体育课排队★★

**时间限制：** 4000ms
**空间限制：** 256MB
**难度：** ★★★★★ ★★☆☆☆

**标签：** 二分图 网络流 二分答案 Special Judge

## 题解

根据题意，考虑将 $n$ 位同学视为二分图的左部，将 $n$ 个位置视为二分图的右部，将该问题转化为二分图匹配问题。

由于匹配是同时开始进行的，因此每次二分图匹配的时间只与最长的那组匹配有关，要求该时间的最小值，使用 $KM$ 等方法难以解决。于是考虑二分答案，每次只将所有范围之内的边加入二分图，然后进行二分图匹配。若能够完全匹配，说明在当前时间约束下可以完成排队任务；否则应扩大时间约束。

使用匈牙利 $Hungary$ 算法，总时间复杂度 $O(n^3 \log n)$，无法通过。

注意到该图为二分图，可以用优化过的网络流算法进行二分图匹配，单次复杂度只有 $O(n^{\frac{5}{2}})$ ，相比匈牙利 $Hungary$ 算法的 $O(n^3)$ 具有明显优势。匹配完成后，利用残余网络按要求输出匹配情况。

时间复杂度 $O(n^{\frac{5}{2}} \log n)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 2005
#define maxm 2004005
#define INF 1234567890
int T, n, m, ss[maxn], sx[maxn], sy[maxn], sum[maxn], l, r, s, t, maxflow, cnt = 1, x[maxn], y[maxn], head[maxn], dis[maxn], cur[maxn], gap[maxn], ans[maxn];
struct Edge {int u, v, w, pre, next;} edge[maxm];
inline void add(int u, int v, int w) {
    edge[++cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].w = w;
    edge[cnt].pre = w;
    edge[cnt].next = head[u];
    head[u] = cnt;
    return;
}
inline int Dfs(int u, int lim) {
    if (!lim || u == t) return lim;
    int flow = 0, f;
    for (int i = cur[u]; i; i = edge[i].next) {
        int v = edge[i].v, w = edge[i].w;
        cur[u] = i;
        if (dis[v] + 1 == dis[u] && (f = Dfs(v, min(lim, w)))) {
            flow += f;
```

```
                lim -= f;
                edge[i].w -= f;
                edge[i ^ 1].w += f;
                if (!lim) return flow;
            }
        }
    gap[dis[u]]--;
    if (!gap[dis[u]]) dis[s] = t + 1;
    dis[u]++;
    gap[dis[u]]++;
    return flow;
}
inline void Bfs() {
    queue<int>q;
    for(int i = 1; i <= t; i++) {
        dis[i] = INF;
        gap[i] = 0;
    }
    dis[t] = 0;
    gap[0] = 1;
    q.push(t);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; i; i = edge[i].next) {
            int v = edge[i].v;
            if (dis[v] >= INF) {
                dis[v] = dis[u] + 1;
                gap[dis[v]]++;
                q.push(v);
            }
        }
    }
    return;
}
inline void ISAP() {
    Bfs();
    while (dis[s] < t) {
        for (int i = 1; i <= t; i++)
            cur[i] = head[i];
        maxflow += Dfs(s, INF);
    }
}
inline bool Check(int mid) {
    cnt = 1;
    maxflow = 0;
    for (int i = 1; i <= t; i++)
        head[i] = dis[i] = cur[i] = gap[i] = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            for (int k = 1; k <= ss[j]; k++) {
                int w = abs(sx[j] + k - 1 - x[i]) + abs(sy[j] - y[i]);
                if (w > mid) continue;
                add(i, n + sum[j-1] + k, 1);
                add(n + sum[j-1] + k, i, 0);
```

```
            }
    for(int i = 1; i <= n; i++) {
        add(s, i, 1);
        add(i, s, 0);
        add(i + n, t, 1);
        add(t, i + n, 0);
    }
    ISAP();
    if (maxflow == n) return 1;
    else return 0;
}
int main()
{
    cin >> T;
    while (T--) {
        cin >> n >> m;
        l = 0, r = 5000;
        for (int i = 1; i <= n; i++)
            cin >> x[i] >> y[i];
        for (int i = 1; i <= m; i++) {
            cin >> ss[i] >> sx[i] >> sy[i];
            sum[i] = sum[i-1] + ss[i];
        }
        s = 2 * n + 1;
        t = 2 * n + 2;
        while (l < r) {
            int mid = (l + r) >> 1;
            if (Check(mid)) r = mid;
            else l = mid + 1;
        }
        cout<< l << endl;
        Check(l);
        for (int i = 1; i <= n; i++)
            for (int j = head[i]; j; j = edge[j].next) {
                int v = edge[j].v, w = edge[j].w, pre = edge[j].pre;
                if (pre > w) ans[v - n] = i;
            }
        for(int i = 1; i <= m; i++)
            for(int j = 1; j <= ss[i]; j++)
                cout << ans[sum[i-1] + j] << " \n"[j == ss[i]];
    }
    return 0;
}

```

# C - ★★生命的游戏★★

| | |
|---|---|
| **时间限制:** | 1000ms |
| **空间限制:** | 256MB |
| **难度:** | ★★☆☆☆<br>☆☆☆☆☆ |

**标签:** 暴力 模拟

## 题解

按要求暴力模拟即可。

时间复杂度 $O(n^2k)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 105
int T, n, k, ans, a[maxn][maxn], now[maxn][maxn], s[maxn][maxn], di[] = {-1, -1, -1, 0, 0, 1, 1, 1}, dj[] = {-1, 0, 1, -1, 1, -1, 0, 1};
inline bool Check() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (a[i][j] != now[i][j]) return 0;
    return 1;
}
inline void Solve() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            s[i][j] = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (now[i][j]) {
                for (int k = 0; k < 8; k++) {
                    int ni = (i + di[k] + n) % n, nj = (j + dj[k] + n) % n;
                    s[ni][nj]++;
                }
            }
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (s[i][j] == 3) now[i][j] = 1;
            else if (s[i][j] < 2 || s[i][j] > 3) now[i][j] = 0;
}
int main()
{
    cin >> T;
    while (T--) {
        cin >> n >> k;
        ans = 0;
        for (int i = 0; i < n; i++)
```

```
            for (int j = 0; j < n; j++) {
                cin >> a[i][j];
                now[i][j] = a[i][j];
            }
        do {
            Solve();
            ans++;
        }
        while (!Check() && --k);
        if (k) cout << "YES" << endl << ans << endl;
        else cout << "NO" << endl;
    }
    return 0;
}
```

# D - ★★飞马祝福语★★

**时间限制：** 4000ms
**空间限制：** 256MB
**难度：** ★★★★★ ★★★☆☆

**标签：** 递推 动态规划 区间动态规划 矩阵 线段树

## 题解

设祝福信为 $s$，`FeiMa` 为 $x$，下标均从 $1$ 开始。

令 $dp[i][j]$ 表示 $s$ 的前 $i$ 位以 `FeiMa` 中第 $j$ 个字符结尾的不同子序列的数量($j=0$ 表示没有出现其中任何一个字符)，那么状态转移方程为：

$$dp[i][j]=\begin{cases} dp[i-1][j-1]+dp[i-1][j], & s[i]=x[j] \\ dp[i-1][j], & else \end{cases}$$

答案即为 $dp[n][5]$。

暴力求解，时间复杂度 $O(qn)$，无法通过。

考虑用矩阵进行状态转移，设 $D[i]=[\,dp[i][0] \quad dp[i][1] \quad dp[i][2] \quad dp[i][3] \quad dp[i][4] \quad dp[i][5]\,]$，

若 $s[i]=$ `F`，那么有：

$$D[i]=D[i-1]\begin{bmatrix} 1&1&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{bmatrix}$$

同理，若 $s[i]=$ `e`，那么有：

$$D[i]=D[i-1]\begin{bmatrix} 1&0&0&0&0&0\\ 0&1&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{bmatrix}$$

以此类推，根据 $s[i]$ 的不同取值，共有 $6$ 种转移矩阵。

于是将问题转化为矩阵的区间乘积维护，考虑将矩阵作为线段树结点的元素。

进行区间修改使用矩阵快速幂求值然后赋值，总时间复杂度 $O(6^3 q\log^2 n)$，无法通过。

考虑预处理出 $6$ 种转移矩阵的 $0\sim n$ 次方，在每次区间修改时可以直接 $O(6^2)$ 赋值。

时间复杂度 $O(6^3 q\log n)$。

另外，此题还有复杂度更优的算法：线段树维护区间动态规划。

设祝福信为 $s$，`FeiMa` 为 $x$，$s$ 的下标从 $1$ 开始，$x$ 的下标从 $0$ 开始。

使用线段树维护区间动态规划。令 $dp[rt][i][j]$ 表示线段树的 $rt$ 节点包含的字符子串范围内，拥有字符串 $x$ 的 $i$ 位到 $j$ 位区间的子序列数量（例如，$i=0,j=0$ 表示 $rt$ 节点中 `F` 的数量，$i=1,j=1$ 表示节点中 `e` 的数量，$i=2,j=3$ 表示节点中子序列为 `iM` 的数量），那么状态转移方程为：

$$dp[rt][i][j] = dp[rt << 1][i][j] + dp[rt << 1|1][i][j] + \sum_{k=i}^{j-1} dp[rt << 1][i][k] * dp[rt << 1|1][k+1][j]$$

答案即为 $dp[1][0][4]$。
时间复杂度 $O(5^3 q \log n)$。

## 参考代码

### C++ - 1

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 100005
#define p 998244353
struct Matrix {
    int a[6][6], n = 5;
    Matrix(const int & N = 5) : n(N) {memset(a, 0, sizeof(a));}
    void operator = (const Matrix & x) {
        n = x.n;
        for(int i = 0; i <= n; i++)
            for(int j = 0; j <= n; j++)
                a[i][j] = x.a[i][j];
    }
    Matrix operator * (const Matrix & x) const {
        Matrix ans = Matrix(n);
        for(int i = 0;i <= n; i++)
            for(int j = 0; j <= n; j++)
                for(int k = 0; k <= n; k++)
                    ans.a[i][j] = (ans.a[i][j] + 1ll * a[i][k] * x.a[k][j]) % p;
        return ans;
    }
};
int T, n, q, a[maxn];
char x;
map<char, int> m;
Matrix pre[6][maxn];
struct Tree {Matrix mul; int tag;}t[maxn << 2];
inline int ls(int k) {return k << 1;}
inline int rs(int k) {return k << 1 | 1;}
inline void push_up(int l, int r, int k) {
    t[k].mul = t[ls(k)].mul * t[rs(k)].mul;
}
inline void push_down(int l, int r, int k) {
    if (t[k].tag != -1) {
        t[ls(k)].tag = t[k].tag;
        t[rs(k)].tag = t[k].tag;
        int mid = (l + r) >> 1;
        t[ls(k)].mul = pre[t[k].tag][mid - l + 1];
        t[rs(k)].mul = pre[t[k].tag][r - mid];
        t[k].tag = -1;
```

```
    }
}
inline void build(int l, int r, int k) {
    t[k].tag = -1;
    if (l == r) {
        t[k].mul = pre[a[l]][1];
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, ls(k));
    build(mid + 1, r, rs(k));
    push_up(l, r, k);
}
inline void update(int nl, int nr, int l, int r, int k, int x) {
    if (nl <= l && r <= nr) {
        t[k].mul = pre[x][r - l + 1];
        t[k].tag = x;
        return;
    }
    push_down(l, r, k);
    int mid = (l + r) >> 1;
    if (nl <= mid) update(nl, nr, l, mid, ls(k), x);
    if (nr > mid) update(nl, nr, mid + 1, r, rs(k), x);
    push_up(l, r, k);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);cout.tie(0);
    for (int k = 0; k <= 5; k++) {
        for(int i = 0; i <= 5; i++)
            pre[k][0].a[i][i] = pre[k][1].a[i][i] = 1;
        if (k) pre[k][1].a[k-1][k] = 1;
    }
    for(int k = 0;k <= 5; k++)
        for(int i = 2;i < maxn; i++)
            pre[k][i] = pre[k][i - 1] * pre[k][1];
    m['F'] = 1, m['e'] = 2,m['i'] = 3,m['M'] = 4,m['a'] = 5;
    cin >> T;
    while (T--) {
        cin >> n >> q;
        for (int i = 1; i <= n; i++) {
            cin >> x;
            a[i] = m[x];
        }
        build(1, n, 1);
        int l, r;
        while (q--) {
            cin >> l >> r >> x;
            update(l, r, 1, n, 1, m[x]);
            cout << t[1].mul.a[0][5] << endl;
        }
    }
    return 0;
}
```

## C++ - 2

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int mod = 998244353;
const int maxn = 1e5+100;
int T,n,m,q;
char ss[maxn];
char pp[] = "FeiMa";
ll pro[maxn<<2][5][5];
char lazy[maxn<<2];
void build(int rt,int l,int r)
{
    lazy[rt] = 0;
    if(l==r)
    {
        for(int i=0; i<5; i++) for(int j=i; j<5; j++) pro[rt][i][j] = 0;
        for(int i=0; i<5; i++) pro[rt][i][i] = ss[l]==pp[i];
        return;
    }
    int mid = (l+r)>>1;
    build(rt<<1,l,mid);
    build(rt<<1|1,mid+1,r);
    for(int i=0; i<5; i++)
    {
        for(int j=i; j<5; j++)
        {
            pro[rt][i][j] = (pro[rt<<1][i][j]+pro[rt<<1|1][i][j])%mod;
            for(int k=i; k<j; k++) pro[rt][i][j] = (pro[rt][i][j]+pro[rt<<1][i][k]*pro[rt<<1|1][k+1][j])%mod;
        }
    }
}
void push_down(int rt,int l,int r)
{
    if(!lazy[rt]) return;
    lazy[rt<<1] = lazy[rt<<1|1] = lazy[rt];
    int mid = (l+r)>>1;
    int sz = mid-l+1;
    for(int i=0; i<5; i++) for(int j=i; j<5; j++) pro[rt<<1][i][j] = 0;
    for(int i=0; i<5; i++) pro[rt<<1][i][i] = (lazy[rt]==pp[i])*sz;
    sz = r-mid;
    for(int i=0; i<5; i++) for(int j=i; j<5; j++) pro[rt<<1|1][i][j] = 0;
    for(int i=0; i<5; i++) pro[rt<<1|1][i][i] = (lazy[rt]==pp[i])*sz;
    lazy[rt] = 0;
}
void update(int rt,int l,int r,int fr,int to,char ch)
{
    if(fr<=l && r<=to)
    {
        for(int i=0; i<5; i++) for(int j=i; j<5; j++) pro[rt][i][j] = 0;
```

```
        lazy[rt] = ch;
        int sz = r-l+1;
        for(int i=0; i<5; i++) pro[rt][i][i] = (ch==pp[i])*sz;
        return;
    }
    push_down(rt,l,r);
    int mid = (l+r)>>1;
    if(mid>=fr) update(rt<<1,l,mid,fr,to,ch);
    if(mid+1<=to) update(rt<<1|1,mid+1,r,fr,to,ch);
    for(int i=0; i<5; i++)
    {
        for(int j=i; j<5; j++)
        {
            pro[rt][i][j] = (pro[rt<<1][i][j]+pro[rt<<1|1][i][j])%mod;
            for(int k=i; k<j; k++) pro[rt][i][j] = (pro[rt][i][j]+pro[rt<<1][i][k]*pro[rt<<1|1][k+1][j])%mod;
        }
    }
}
char ch[2];
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%s",&n,&q,ss + 1);
        build(1,1,n);
        int l,r;
        while(q--)
        {
            scanf("%d%d%s",&l,&r,ch);
            update(1,1,n,l,r,ch[0]);
            printf("%lld\n",pro[1][0][4]);
        }
    }
    return 0;
}
```

# E - ★★序列大团结★★

| | |
|---|---|
| **时间限制：** | 1000ms |
| **空间限制：** | 256MB |
| **难度：** | ★★★★★☆☆☆☆☆ |

**标签：** 哈希Hash

## 题解

先考虑序列中无法被 $k$ 整除的数，设 $s[i][v]$ 为序列前 $i$ 项中数值 $v$ 出现的次数模 $k$ 的余数，当 $s[i]$ 和 $s[j]$ $(i<j)$ 完全相同时(即每个数值 $v$ 出现的次数模 $k$ 的余数都相同)，子序列 $E_{i+1}, E_{i+2}, \ldots, E_j$ 就是“团结”的。

考虑使用滚动数组，将 $s[i][v]$ 优化至 $s[v]$，从首项开始遍历并维护 $s$，同时对 $s$ 的内容进行 $Hash$ 处理，将处理后的结果保存，记 $num[hash(s)]$ 为到目前为止 $hash(s)$ 出现的次数，那么以当前项为右端点的团结的子序列的数量为 $num[hash(s)]-1$。$hash(s)$ 可用 $map$ 存储。

而能够被 $k$ 整除的数，不影响 $hash$ 值，但也同样需要计算答案。

时间复杂度 $O(n\log n)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
#define p 998244353
ull T, n, k, x, ans, now, hs;
map<ull, int>num;
map<int, int>s;
inline ull ksm(ull x, ull k) {
    ull res = 1;
    for(; k; k >>= 1, x = x * x)
        if (k & 1) res = res * x;
    return res;
}
int main(){
    cin >> T;
    while (T--) {
        cin >> n >> k;
        num.clear();
        s.clear();
        ans = now = 0;
        num[0] = 1;
        for (int i = 1; i <= n; i++) {
            cin >> x;
            if (x % k) {
                hs = ksm(p, x);
                now -= hs * s[x];
                s[x] = (s[x] + 1) % k;
```

```
                now += hs * s[x];
            }
            ans += num[now];
            num[now]++;
        }
        cout << ans << endl;
    }
    return 0;
}
```

# F - ★★飞马分隔符★★

| | |
|---|---|
| **时间限制:** | 1000ms |
| **空间限制:** | 256MB |
| **难度:** | ★★☆☆☆<br>☆☆☆☆☆ |

**标签:** 模拟 字符串 贪心

## 题解

从头到尾遍历，每次遇到字符 `a`，判断前面是否顺次出现过一组 `F`、`e`、`i`、`M`，若出现过，则在此处分隔，更新答案，并清空记录。

时间复杂度 $O(n)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
int T, n, ans, flag;
string s;
int main()
{
    cin >> T;
    while (T--) {
        cin >> n >> s;
        ans = flag = 0;
        for (int i = 0; i < n; i++) {
            if (s[i] == 'F' && flag == 0) flag++;
            else if (s[i] == 'e' && flag == 1) flag++;
            else if (s[i] == 'i' && flag == 2) flag++;
            else if (s[i] == 'M' && flag == 3) flag++;
            else if (s[i] == 'a' && flag == 4) {
                ans++;
                flag = 0;
            }
        }
        cout << ans << endl;
    }
    return 0;
}

```

# G - ★★糖果魔法阵★★

**时间限制：** 2000ms
**空间限制：** 256MB
**难度：** ★★★★★ ★★★★☆

**标签：** 数学 数论 递推 二次剩余 光速幂 扩展欧拉定理 逆元

## 题解

根据题意，定义 $G_n = \dfrac{G_{n-1}^4 + 12G_{n-1}^2 + 4}{4G_{n-1}(G_{n-1}^2 + 2)}$，求出两个实数不动点为 $\sqrt{2}, -\sqrt{2}$。

构造 $\dfrac{G_n - \sqrt{2}}{G_n + \sqrt{2}} = \dfrac{G_{n-1}^4 - 4\sqrt{2}G_{n-1}^3 + 12G_{n-1}^2 - 8\sqrt{2}G_{n-1} + 4}{G_{n-1}^4 + 4\sqrt{2}G_{n-1}^3 + 12G_{n-1}^2 + 8\sqrt{2}G_{n-1} + 4} = \left(\dfrac{G_{n-1} - \sqrt{2}}{G_{n-1} + \sqrt{2}}\right)^4 = \left(\dfrac{G_0 - \sqrt{2}}{G_0 + \sqrt{2}}\right)^{4^n}$

推出通项为 $G_n = \sqrt{2}\dfrac{(G_0 + \sqrt{2})^{4^n} + (G_0 - \sqrt{2})^{4^n}}{(G_0 + \sqrt{2})^{4^n} - (G_0 - \sqrt{2})^{4^n}}$

由于 $2$ 是模 $998244353$ 的二次剩余，将 $G_0 = 1$ 与 $\sqrt{2} \equiv 116195171 \pmod{998244353}$ 代入得：

$$G_n = 116195171\frac{116195172^{4^n} + 882049183^{4^n}}{116195172^{4^n} - 882049183^{4^n}} \bmod 998244353$$

考虑指数 $4^n$，根据欧拉定理，可对其取模，即 $4^n \iff 4^n \bmod \varphi(998244353) \iff 4^n \bmod 998244352$

考虑指数 $n$，根据扩展欧拉定理，当 $n \geq \varphi(998244352) = 402653184$ 时可对其取模，即 $n \iff n \bmod 402653184 + 402653184$。

使用快速幂求出每天结束时的 $G_k$，进而求出下一天的魔力值，时间复杂度 $O(n \log P)$，无法通过。

考虑对 $4^n$ 进行模 $998244352$ 的光速幂处理，对 $116195172^n$ 和 $882049183^n$ 进行模 $998244353$ 的光速幂处理。

在每天结束时，用光速幂求出相应的 $(116195172^{4^k} - 882049183^{4^k})mG_k \bmod 998244353$，便可得知下一天的情况。

直到最后一天结束时，用光速幂和费马小定理求出 $G_n$。

时间复杂度 $O(n + q\log P)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 31625
#define two 116195171
#define ka 116195172
#define kb 882049183
#define phi 402653184
#define p 998244353
ll s, d, q, n, a[maxn + 5][2], b[maxn + 5][2], c[maxn + 5][2], k, t, sum, ans;
```

```
bool flag;
inline void Init() {
    a[0][0] = a[0][1] = b[0][0] = b[0][1] = c[0][0] = c[0][1] = 1;
    for (ll i = 1; i <= maxn; i++) {
        a[i][0] = a[i - 1][0] * ka % p;
        b[i][0] = b[i - 1][0] * kb % p;
        c[i][0] = c[i - 1][0] * 4 % (p - 1);
    }
    for (ll i = 1; i <= maxn; i++) {
        a[i][1] = a[i - 1][1] * a[maxn][0] % p;
        b[i][1] = b[i - 1][1] * b[maxn][0] % p;
        c[i][1] = c[i - 1][1] * c[maxn][0] % (p - 1);
    }
}
inline ll ksm(ll x, ll k) {
    ll res = 1 % p; x %= p;
    for (; k; k >>= 1, x = x * x % p)
        if (k & 1) res = res * x % p;
    return res;
}
int main()
{
    Init();
    cin >> s >> d >> q;
    while (q--) {
        cin >> n;
        sum = s;
        k = flag = 0;
        for (ll i = 1; i <= n; i++) {
            k += sum / d;
            if (k >= phi) flag = 1;
            k %= phi;
            if(flag) k += phi;
            t = c[k % maxn][0] * c[k / maxn][1] % (p - 1);
            sum = (((a[t % maxn][0] * a[t / maxn][1] % p) + (b[t % maxn][0] * b[t / maxn][1] % p)) * two % p) * i % p;
        }
        ans = (sum * ksm(n, p - 2) % p) * ksm((((a[t % maxn][0] * a[t / maxn][1] % p) - (b[t % maxn][0] * b[t / maxn][1] % p) + p) % p) % p, p - 2)%p;
        cout << ans << endl;
    }
    return 0;
}

```

# H - ★★温暖的力量★★

**时间限制：** 1000ms
**空间限制：** 256MB
**难度：** ★☆☆☆☆☆☆☆☆☆

**标签：** 数学

## 题解

要求将 $n$ 分为尽可能多的质数的和。

当 $n$ 为大于 $3$ 的奇数时，可以分为若干个 $2$ 和一个 $3$；当 $n$ 为大于 $3$ 的偶数时，可以分为若干个 $2$。

不难得出答案：

$$Answer = \begin{cases} -1, & n \leq 3 \\ \left\lfloor \frac{n}{2} \right\rfloor, & n > 3 \end{cases}$$

时间复杂度 $O(T)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
int T, n;
int main()
{
    cin >> T;
    while (T--) {
        cin >> n;
        if (n <= 3) cout << -1 << endl;
        else cout << n / 2 << endl;
    }
    return 0;
}

```

# I - ★★平形四边行★★

| | |
|---|---|
| **时间限制：** | 1000ms |
| **空间限制：** | 256MB |
| **难度：** | ★★★★★<br>☆☆☆☆☆ |

**标签：** 暴力 Special Judge

## 题解

四个点能形成“平形四边行”的充要条件是，存在一种方案，将四个点均分为两组，每组的两个点形成一条线段，这两条线段的中点重合。

注意到桶的大小只有 $4000 \times 4000$，即第 $4000 \times 4000 + 1$ 个点落下时，一定存在重合的中点，从而构成一组解。另外，桶不可能被完全填满，因此复杂度远远小于预期。

于是 $n^2$ 暴力枚举每组点，存储所形成线段的中点，直到存在重合的中点，同时应注意过滤掉存在交集的两组点。

时间复杂度 $O(4000^2)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 100005
int n, x[maxn], y[maxn];
pair<int, int>v[4005][4005];
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> x[i] >> y[i];
        for (int j = 1; j < i; j++) {
            int xx = x[i] + x[j] + 2000;
            int yy = y[i] + y[j] + 2000;
            if (i == v[xx][yy].first || j == v[xx][yy].first) continue;
            if (i == v[xx][yy].second || j == v[xx][yy].second) continue;
            if (v[xx][yy].first) {
                cout << "YES" << endl << i << ' ' << j << ' ' << v[xx][yy].first << ' ' << v[xx][yy].second << endl;
                return 0;
            }
            v[xx][yy] = make_pair(i,j);
        }
    }
    cout << "NO" << endl;
    return 0;
}

```

# J - ★★翻滚吧硬币★★

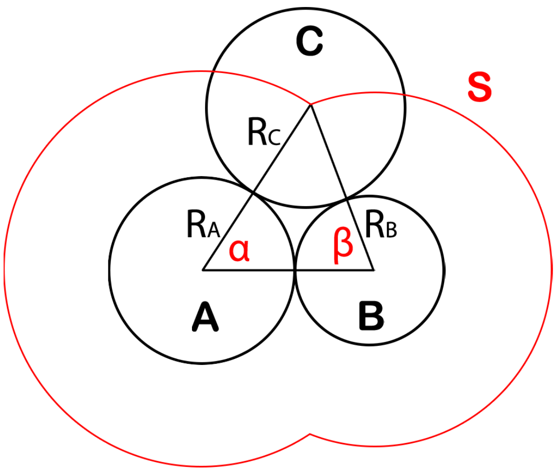
**时间限制：** 1000ms
**空间限制：** 256MB
**难度：** ★★★★☆ ☆☆☆☆☆

**标签：** 数学 计算几何 Special Judge

## 题解

硬币悖论问题。

如图，三枚硬币 $A, B, C$，半径分别为 $R_A, R_B, R_C$。




现在，让硬币 $C$ 沿着硬币 $A, B$ 的边界翻滚，硬币 $C$ 的圆心的运动轨迹为曲线 $S$，记其长度为 $s$，那么翻滚的圈数即为 $\frac{s}{2\pi R_C}$。

由弧长公式可得 $s = 2(\pi - \alpha)(R_A + R_C) + 2(\pi - \beta)(R_B + R_C)$，其中 $\alpha, \beta$ 可由余弦定理推出：

$$\begin{cases} \alpha = \arccos\left(\frac{(R_A+R_B)^2+(R_A+R_C)^2-(R_B+R_C)^2}{2(R_A+R_B)(R_A+R_C)}\right) \\ \beta = \arccos\left(\frac{(R_B+R_A)^2+(R_B+R_C)^2-(R_A+R_C)^2}{2(R_B+R_A)(R_B+R_C)}\right) \end{cases}。$$

要想使得圈数最小，只需固定半径较小的两个硬币，让半径最大的硬币沿其边界翻滚即可。

时间复杂度 $O(T)$。

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
const long double pi=acosl(-1);
int T, r[5];
long double ans, a, b, c, x, y;
int main()
{
```

```
    cin >> T;
    while (T--) {
        cin >> r[1] >> r[2] >> r[3];
        sort(r + 1, r + 3 + 1);
        a = r[1] + r[2], b = r[1] + r[3], c = r[2] + r[3];
        x = acosl((a * a + b * b - c * c) / (2.0l * a * b));
        y = acosl((a * a + c * c - b * b) / (2.0l * a * c));
        ans = ((pi - x) * b + (pi - y) * c) / (pi * r[3]);
        cout << fixed << setprecision(15) << ans << endl;
    }
    return 0;
}
```

# K - ★★快乐苹果树★★

| | |
|---|---|
| **时间限制:** | 1000ms |
| **空间限制:** | 256MB |
| **难度:** | ★★★★★ ★★★★☆ |

**标签:** 数学 逆元 概率论 数学期望 树链剖分 树状数组 线段树

## 题解

定义 $size[i]$ 表示以结点 $i$ 为根的子树的大小，$son[i]$ 表示结点 $i$ 的 所有直接子结点(距离为 1)组成的集合，$sons[i]$ 表示以结点 $i$ 为根的子树内所有节点组成的集合。

对于操作一，可分为以下两种情况:

- 当选取结点 $u \in sons[w], w \in son[F]$，那么点集 $\{v \mid v \in T, lca(u,v) = F\}$ 表示的是 $sons[F] - sons[w]$，此时该操作与 $u$ 无关，只与 $w$ 有关。结点 $u \in sons[w]$ 的概率为 $\frac{size[w]}{size[F]}$，于是点集 $sons[F] - sons[w]$ 中所有结点的快乐值增加 $\frac{size[w]}{size[F]} \frac{1}{size[F] - size[w]} k$。
- 当选取结点 $u = F$ 时，概率为 $\frac{1}{size[F]}$，点集 $\{v \mid v \in T, lca(u,v) = F\}$ 表示的是 $sons[F]$，于是点集 $sons[F]$ 中所有结点的快乐值增加 $\frac{1}{size[F]} \frac{1}{size[F]} k$。

预处理出各子树大小以及对应逆元，暴力求解，总时间复杂度 $O(n + qn^2)$，无法通过。

考虑使点集 $sons[F]$ 中所有结点的快乐值都加上该值，然后再使点集 $sons[w]$ 中所有结点的快乐值减去该值。

如此一来，对于每次操作一，使点集 $sons[F]$ 中所有结点的快乐值增加 $\frac{1}{size[F]} \frac{1}{size[F]} k + \sum\limits_{w \in son[F]} \frac{size[w]}{size[F]} \frac{1}{size[F] - size[w]} k$,

随后，遍历 $w \in son[F]$，使点集 $sons[w]$ 中所有结点的快乐值减去 $\frac{size[w]}{size[F]} \frac{1}{size[F] - size[w]} k$。

总时间复杂度 $O(n + qn)$，无法通过。

考虑将子树按大小分块，在每次进行子树更新时使用线段树或树状数组维护，总时间复杂度 $O(n \log n + q\sqrt{n} \log n)$，可以通过，但不稳定且代码实现较为困难。

考虑进行树链剖分，将轻儿子 $x$ 懒标记的值放置在 $F$ 结点即 $top[fa[x]]$ 结点，重儿子 $\log n$ 更新。查询时，遍历 $top[fa[x]]$ 节点，减去对应的懒标记的值。

时间复杂度 $O(n + q \log n)$。

## 参考代码

### C++

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 100005
#define maxm 200005
```

```
#define p 998244353
ll T, n, q, inv[maxn], t[maxn], tag[maxn], fa[maxn], se[maxn], son[maxn], dfn[maxn], top[maxn], id, cnt, head[maxn], s1[maxn], s2[maxn], s3[maxn];
struct Edge {ll u, v, next;} edge[maxm];
inline void add(ll u, ll v) {
    edge[++cnt].u = u;
    edge[cnt].v = v;
    edge[cnt].next = head[u];
    head[u] = cnt;
}
inline void Get_Inv() {
    inv[1] = 1;
    for (ll i = 2; i < maxn; i++)
        inv[i] = (p - p / i) * inv[p % i] % p;
}
inline ll lowbit(ll x) {return x & (-x);}
inline void change(ll x, ll val)
{
    for (ll i = x; i <= n; i += lowbit(i))
        t[i] = (t[i] + val) % p;
}
inline ll query(ll x) {
    ll res = 0;
    for (ll i = x; i; i -= lowbit(i))
        res = (res + t[i]) % p;
    return res;
}
inline void update(ll l, ll r, ll val) {
    change(l, val);
    change(r + 1, p - val);
}
inline void Dfs1(int u, int f) {
    fa[u] = f;
    se[u] = 1;
    son[u] = s3[u] = 0;
    for (ll i = head[u]; i; i = edge[i].next) {
        ll v = edge[i].v;
        if (v == f) continue;
        Dfs1(v, u);
        se[u] += se[v];
        if (se[v] > se[son[u]]) son[u] = v;
    }
    s1[u] = inv[se[u]];
    for (ll i = head[u]; i; i = edge[i].next) {
        ll v = edge[i].v;
        if (v == f) continue;
        s2[v] = (se[v] * s1[u] % p) * inv[se[u] - se[v]] % p;
        s3[u] = (s3[u] + s2[v]) % p;
    }
}
inline void Dfs2(int u, int t) {
    top[u] = t;
    dfn[u] = ++id;
    if (!son[u]) return;
    Dfs2(son[u], t);
```

```
    for (ll i = head[u]; i; i = edge[i].next) {
        ll v = edge[i].v;
        if (v == son[u] || v == fa[u]) continue;
        Dfs2(v, v);
    }
}
int main()
{
    Get_Inv();
    cin >> T;
    while (T--) {
        cin >> n >> q;
        cnt = id = 0;
        for (ll i = 0; i <= n; i++)
            head[i] = t[i] = tag[i] = 0;
        ll u, v;
        for (ll i = 1; i < n; i++) {
            cin >> u >> v;
            add(u, v);
            add(v, u);
        }
        Dfs1(1, 0);
        Dfs2(1, 1);
        ll op, f, k;
        while (q--) {
            cin >> op;
            if (op == 1) {
                cin >> f >> k;
                update(dfn[f], dfn[f] + se[f] - 1, (s3[f] * k % p + (s1[f] * s1[f] % p) * k % p) % p);
                if (son[f])
                    update(dfn[son[f]], dfn[son[f]] + se[son[f]] - 1, (p - s2[son[f]] * k % p) % p);
                tag[f] = (tag[f] + k) % p;
            }
            else {
                cin >> f;
                ll res = query(dfn[f]);
                for(ll i = top[f]; fa[i]; i = top[fa[i]])
                    res = (res + (p - s2[i] * tag[fa[i]] % p) % p) % p;
                cout << res << endl;
            }
        }
    }
    return 0;
}
```

# L - ★★星星收集者★★

| | |
|---|---|
| **时间限制：** | 1000ms |
| **空间限制：** | 256MB |
| **难度：** | ★★★★★<br>★☆☆☆☆ |

**标签：** 博弈论 动态规划

## 题解

令星星的总和为 $sum$。
首先，因为双方的星星总和一定是 $sum$，如果 $x \geq \frac{sum}{2}$，那么当 $a \geq \frac{sum}{2}$ 时 $L.St$ 就获胜了。
反之，如果 $x < \frac{sum}{2}$，当 $a < \frac{sum}{2}$ 时 $L.St$ 获胜。
其余情况 $L.Ts$ 获胜。
那么我们从 $L.St$ 的角度考虑，他的策略根据 $sum$ 与 $x$ 的关系，只有两种：

- 抓尽可能少的星星
- 抓尽可能多的星星（也就是全部抓取）

$L.Ts$ 需要让 $L.St$ 在这两种策略下都不能获胜。
令添加星星数为 $k$，首先可以发现，当$x \geq \frac{sum}{2}$时，$L.St$ 必胜（全部抓取），所以可以得到 $k > x + x - sum$。
当 $x \geq \frac{sum}{2}$ 时，考虑 $L.Ts$ 在添加星星时的策略。

我们发现，在此情况下，双方都想最小化自己的星星数。那么 $L.Ts$ 会尽可能的将 $k$ 分配给 $L.St$，只要将 $k$ 全部分配给第一个盒子就能做到。
我们用动态规划可以求出在原先状态下，a最少可以抓取多少星星。
令这个值为 $v$，则能得到 $\frac{k+sum}{2} < v + k$。
综合两种情况就能得到 $k$ 的最小值

## 参考代码

### C++

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define maxn 200005
ll n, x, a[maxn], sum[maxn], dp[maxn], last;
int main()
{
    cin >> n >> x;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        sum[i] = sum[i - 1] + a[i];
    }
    last = n;
    dp[n] = a[n];
    for (ll i = n - 1; i; i--) {
        dp[i] = sum[n] - sum[i - 1] - dp[last];
```

```
        if (dp[i] > dp[last]) last = i;
    }
    cout << max(max(0ll, sum[n] - dp[1] - dp[1] + 1), max(0ll, 2 * x + 1 - sum[n])) << endl;
    return 0;
}

```