A 小H的最简分式

模拟题，根据小学的分数加法的运算法则。

两分式相加，分母是两分式分母的**最小公倍数**，分子是各自分子和分母共同扩大的倍数相加，即：

$$\frac{a}{b}+\frac{c}{d}=\frac{a\times\frac{\operatorname{lcm}(b,d)}{b}+c\times\frac{\operatorname{lcm}(b,d)}{d}}{\operatorname{lcm}(b,d)}$$

题目要求化简为最简分数，所以直接对结果分子分母同时除以他们的**最大公约数**，记分子为 e，分母为 f，最后答案就是：

$$\frac{e}{f}=\frac{\frac{e}{\gcd(e,f)}}{\frac{f}{\gcd(e,f)}}$$

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int lcm(int a, int b) {
    return a * b / gcd(a, b);
}

void solve() {
    int a, b, c, d;
    std::cin >> a >> b >> c >> d;
    int f = lcm(b, d);
    int e = a * f / b + c * f / d;
    int g = gcd(f, e);
    std::cout << e / g << " " << f / g << "\n";
}

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

    int t;
    std::cin >> t;
    while (t --) {
        solve();
    }
```

```
    return 0;
}
```

B 小H的奇数序列

这是一道思维题。题目要求我们构造一个不重复的奇数数列，并且数列之和不超过 S，输出最大的长度。

贪心思路：给定了数列之和，要使得长度最大，我们数列里的数就要越小越好，即从最小的正奇数 1 开始。

$$1, 3, 5, 7, 9, \cdots$$

我们很容易观察出来这个是以 2 为公差的递增等差数列，所以前 n 项和公式为：

$$S = a_1 n + \frac{n(n-1)}{2} d$$

将 $S, a_1 = 1, d = 2$ 带入上式可得：

$$S = n^2$$

所以我们所求答案为：

$$n = \sqrt{S}$$

注意最后 n 应该是整数。

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

void solve() {
    int n;
    std::cin >> n;
    std::cout << (int)std::sqrt(n) << "\n";
}

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

    int t;
    std::cin >> t;
    while (t --) {
        solve();
    }

    return 0;
}
```

C 小H的序列统计

这是一个字符串统计类题目，一般都需要使用组合数学的思想。

给定一个字符串，请找出子序列 yxh 的个数。

暴力做法：3 重循环依次枚举这三个字符，然后累加，当然时间复杂度不可行。

考虑优化，之前的暴力是一个一个找的，我们能否一次找多个，比如固定某个位置的 yx 找这个后面的 h 出现的次数，这样可行，我们可以预处理一下关于 h 的个数前缀和，这样就优化了一重循环，时间复杂度还是 $O(n^2)$。

还需优化，我们根据上面的思路，我们这一次固定 x 的位置，那么我们就需要知道 x 前面的位置有多少 y，后面有多少 h。所以根据乘法原理我们这一个 x 的贡献就是前 y 和后 h 的乘积，即预处理前缀和 y 和 h，然后遍历到 x 位置的时候统计一下贡献即可。

时间复杂度为：$O(n)$

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

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

    int n;
    std::cin >> n;
    std::string s;
    std::cin >> s;
    std::vector<i64> y(n + 1), h(n + 1);
    for (int i = 1; i <= n; i++) {
        y[i] = y[i - 1] + (s[i - 1] == 'y');
        h[i] = h[i - 1] + (s[i - 1] == 'h');
    }

    i64 ans = 0;

    for (int i = 1; i <= n; i++) {
        if (s[i - 1] == 'x') {
            ans += y[i] * (h[n] - h[i]);
        }
    }
    std::cout << ans << "\n";

    return 0;
}
```

D 小H的公式推导(easy)

给定两个数列递推式，直接递推求解即可，注意取模的彻底，只进行一次初始化即可。

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

constexpr int N = 100010, mod = 998244353;

std::vector<i64> f(N), g(N);

void init(int n) {
    f[1] = 1;
    f[2] = 2;
    for (int i = 3; i < n; i++) {
        f[i] = (2 * f[i - 1] % mod + 3 * f[i - 2] % mod + i) % mod;
    }
    g[1] = 2;
    for (int i = 2; i < n; i++) {
        g[i] = (g[i - 1] % mod + f[i] % mod + (i64)i * i % mod) % mod;
    }
}

void solve() {
    int n;
    std::cin >> n;
    std::cout << g[n] % mod << "\n";
}

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

    init(N);

    int t;
    std::cin >> t;
    while (t --) {
        solve();
    }

    return 0;
}
```

E 小H的方格路径

这道题是一道典型的动态规划题目，但是一看数据范围开不了这么大的数组。所以递推求方数这种方法失效了。

我们可以在小数据内打表观察这个表格的性质。

i/j	1	2	3	4	5
1	1	1	1	1	1
2	1	2	3	4	5
3	1	3	6	10	15
4	1	4	10	20	35
5	1	5	15	35	70

这是通过递推的方式计算出的表格，看副对角线的每个数有啥关系。把表顺时针转 45° 你会惊奇的发现竟然是个杨辉三角。

这样我们的答案就是杨辉三角里的某个数了，我们就可以通过其他方式得到，那么就是组合数！

> 补充组合数的表示方法：$\binom{n}{k} = C_n^k$ 在算法竞赛中我们一般使用左侧表示方式表示组合数。

我们的杨辉三角如下：

i/j	1	2	3	4	5
1	$\binom{0}{0}$	$\binom{1}{1}$	$\binom{2}{2}$	$\binom{3}{3}$	$\binom{4}{4}$
2	$\binom{1}{0}$	$\binom{2}{1}$	$\binom{3}{2}$	$\binom{4}{3}$	$\binom{5}{4}$
3	$\binom{2}{0}$	$\binom{3}{1}$	$\binom{4}{2}$	$\binom{5}{3}$	$\binom{6}{4}$
4	$\binom{3}{0}$	$\binom{4}{1}$	$\binom{5}{2}$	$\binom{6}{3}$	$\binom{7}{4}$
5	$\binom{4}{0}$	$\binom{5}{1}$	$\binom{6}{2}$	$\binom{7}{3}$	$\binom{8}{4}$

所以我们要求第 (i, j) 位置的答案就是：

$$\binom{i+j-2}{j-1}$$

我们只需要预处理阶乘和阶乘逆元就可以得到答案了，组合数公式为：

$$\binom{a}{b} = \frac{a!}{b!(a-b)!}$$

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

constexpr int N = 400010, mod = 998244353;

i64 fac[N], invfac[N];

i64 power(i64 a, i64 b) {
```

```
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

i64 inv(i64 x) {
    return power(x, mod - 2);
}

void init(int n) {
    fac[0] = 1;
    invfac[0] = 1;
    for (int i = 1; i < n; i++) {
        fac[i] = fac[i - 1] * i % mod;
        invfac[i] = invfac[i - 1] * inv(i) % mod;
    }
}

i64 C(i64 a, i64 b) {
    if (a < b || b < 0) {
        return 0;
    }
    return (fac[a] * invfac[a - b] % mod * invfac[b]) % mod;
}

void solve() {
    int x, y;
    std::cin >> x >> y;
    std::cout << C(x + y - 2, y - 1) << "\n";
}

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

    init(N);

    int t;
    std::cin >> t;
    while (t --) {
        solve();
    }

    return 0;
}
```

F 小H的区间加数

由题意可知这是一道区间修改，最后区间查询的问题。因为只有一次查询，所以我们可以不上线段树或者树状数组，直接用数组做。

思考这个问题：对每个区间里的数依次加 $1^2, 2^2, 3^2, \cdots$，可以类比到我们学差分的时候是对每个数加 x。二者是否有相同点。

首先看普通的差分：将 $[2,5]$ 这个区间的每个数加 1

$$0,1,1,1,1,0,0,\cdots$$

差分一次得：

$$0,1,0,0,0,-1,0,\cdots$$

我们可以看到对差分数组 b 的影响就是 $b_l+1, b_{r+1}-1$，两次操作

那么对于每次加的数不是 1 呢

$$0,1,4,9,16,0,0,0,0,\cdots$$

多次差分：

$$0,1,3,5,7,-16,0,0,0,\cdots$$
$$0,1,2,2,2,-23,16,0,0,\cdots$$
$$0,1,1,0,0,-25,39,-16,0,\cdots$$

我们看到做了三次差分后，变得好像有规律了，影响了 $l, l+1, r+1, r+2, r+3$ 这五个地方的值。所以我们每次加的操作可以等价操作这 5 个位置的值。

那么我们来看看每个位置有什么方法可以快速计算出影响。

$b_l+1, b_{l+1}+1$ 这两个是可以一眼看出的，剩下的三个我们推导一下，我们记区间长度为（$len=r-l+1$）：

在 $r+1$ 位置，相当于减去了 $len^2+2\times len+1$

在 $r+2$ 位置，相当于加上了 $2\times len^2+2\times len-1$

在 $r+3$ 位置，相当于减去了 len^2

所以我们每次就修改这 5 个位置，最后 3 次前缀和就可以复原数组了，然后在一次前缀和计算数组的和。注意取模计算减法！！！

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

constexpr int mod = 998244353, N = 200010;

std::vector<i64> a(N);
```

```
void getsum(int n) {
    for (int i = 1; i <= n; i++) {
        a[i] = (a[i] + a[i - 1]) % mod;
    }
}

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

    int n, m;
    std::cin >> n >> m;
    while (m--) {
        int l, r;
        std::cin >> l >> r;
        a[l]++;
        a[l + 1]++;
        a[r + 1] = (a[r + 1] % mod + mod - ((i64)(r - l + 1) * (r - l + 1) % mod +
2LL * (r - l) % mod + 3) % mod) % mod;
        a[r + 2] = (a[r + 2] % mod + 2 * (i64)(r - l + 1) * (r - l + 1) % mod + 2LL
* (r - l) % mod + 1) % mod;
        a[r + 3] = (a[r + 3] % mod + mod - ((i64)(r - l + 1) * (r - l + 1) % mod)) %
mod;
    }

    getsum(n);
    getsum(n);
    getsum(n);
    getsum(n);

    std::cout << a[n] % mod << "\n";

    return 0;
}
```

方法二：将区间修改改为单点修改。

其实我们可以直接将这个区间里需要增加的值直接计算出来，通过平方和公式：

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

这样我们可以直接将增加的值直接加到 r 的位置，在 $r+1$ 的位置减去。只需要执行 2 次操作，然后前缀和一次还原。

> 注意：在取模运算做除法运算时，需要将除法变为乘法，即乘以这个数在 mod 下的乘法逆元。

参考代码：

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

using u32 = unsigned;
using i64 = long long;
```

```cpp
using u64 = unsigned long long;
using u128 = unsigned __int128;

constexpr int mod = 998244353, N = 200010;

std::vector<i64> a(N);

i64 power(i64 a, i64 b) {
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

i64 inv(i64 x) {
    return power(x, mod - 2);
}

void getsum(int n) {
    for (int i = 1; i <= n; i++) {
        a[i] = (a[i] + a[i - 1]) % mod;
    }
}

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

    int n, m;
    std::cin >> n >> m;
    while (m--) {
        int l, r;
        std::cin >> l >> r;
        i64 len = r - l + 1;
        i64 add = (len * (len + 1) % mod * (len * 2 + 1) % mod * inv(6)) % mod;
        a[r] = (a[r] + add) % mod;
        a[r + 1] = (a[r + 1] % mod + mod - add) % mod;
    }

    getsum(n);
    getsum(n);

    std::cout << a[n] % mod << "\n";

    return 0;
}
```

G 小H的公式推导(hard)

这道题的数据范围比D的大，并且不能通过递推的方式求解。

所以我们可以用一个终极大招，矩阵加速求解递推式。这是一种很常见的求数据范围太大的递推式的方式。

什么是矩阵加速呢。那么首先请出我们的斐波那契数列来演示一下。

$f_n = f_{n-1} + f_{n-2}$ 这是斐波那契的递推式，常规方法是 $O(n)$ 的时间求解，在数据太大时就失效了。

我们从线性代数的角度思考，这个 f_n, f_{n-1}, f_{n-2} 之间的关系。可以自己假定一个关系：我们可以通过一个变换，将 f_{n-2} 变到 f_{n-1} 变到 f_n 即

$$f_{n-2} \to f_{n-1} \to f_n$$

我们都是用相同的变换，在线性代数里，变换通常是一个矩阵，那么我们可以思考

$$\begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} \to \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$$

很明显，我们左乘一个系数矩阵，就可以变成右边的矩阵了，即：

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$$

那么即使我们要求下一项，我们也可以再在这个基础上乘系数矩阵：

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix}$$

又矩阵具有乘法结合律，所以我们可以写成这种形式：

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix}$$

初始条件是 $f_1 = 1, f_2 = 1$，所以具体的表示式：

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$$

那么我们的时间复杂度就是计算这个矩阵乘法多次幂的时间复杂度了，又因为矩阵多次幂也可以用快速幂计算，那么时间复杂度为：$O(m^3 \log k)$

m 为系数矩阵维度，k 为指数。

那么回到这个题，我们有两个递推式，f_n, g_n 我们需要找到他们之间的关系。

我们先看对应关系，应该是先算 f 再算 g，我们可以合起来一起算。

可以先写出来看看：

$$\begin{pmatrix} g_{n-1} \\ f_n \\ f_{n-1} \\ \dots \end{pmatrix} \to \begin{pmatrix} g_n \\ f_{n+1} \\ f_n \\ \dots \end{pmatrix}$$

只有这些肯定是不够的，因为还有常数 1，跟随次数变化的 n ，我们需要添加额外的信息。

首先看 g_n，它需要 g_{n-1}, f_n, n^2，所以要增加一项 n^2

看 f_{n+1}，需要 $f_n, f_{n-1}, n+1$，所以要增加一项 n，一项常数 1

> 为什么不直接是 $n+1$，因为在 n^2 变为 $(n+1)^2$ 的平方的时候不好直接变换，将 $(n+1)^2 = n^2 + 2n + 1$ 这样就可以直接从 n^2 加上某些项变为 $(n+1)^2$

所以我们的对应关系就是：

$$\begin{pmatrix} g_{n-1} \\ f_n \\ f_{n-1} \\ n \\ n^2 \\ 1 \end{pmatrix} \to \begin{pmatrix} g_n \\ f_{n+1} \\ f_n \\ n+1 \\ (n+1)^2 \\ 1 \end{pmatrix}$$

我们现在来填系数矩阵：

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 2 & 3 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} g_{n-1} \\ f_n \\ f_{n-1} \\ n \\ n^2 \\ 1 \end{pmatrix} = \begin{pmatrix} g_n \\ f_{n+1} \\ f_n \\ n+1 \\ (n+1)^2 \\ 1 \end{pmatrix}$$

最后将初始值带入即可得到最后的表达式：

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 2 & 3 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{n-1} \begin{pmatrix} g_1 \\ f_2 \\ f_1 \\ 2 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} g_n \\ f_{n+1} \\ f_n \\ n+1 \\ (n+1)^2 \\ 1 \end{pmatrix}$$

最后的答案就是系数矩阵的第一行和初始矩阵对应相乘再相加。

时间复杂度为：$O(m^3 \log k)$

参考代码：

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

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;
using u128 = unsigned __int128;

constexpr int mod = 998244353, N = 200010;

struct Matrix {
    std::vector<std::vector<i64>> matrix;
    int n;
    Matrix(int _n): n(_n) {
        matrix.assign(n, std::vector<i64>(n, 0));
```

```
    }
    static Matrix getIdentity(int n) {
        Matrix res(n);
        for (int i = 0; i < n; i++) {
            res.matrix[i][i] = 1;
        }
        return res;
    }
    const Matrix operator*(const Matrix &b) const {
        Matrix res(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    res.matrix[i][j] = (res.matrix[i][j] % mod + matrix[i][k] *
b.matrix[k][j] % mod) % mod;
                }
            }
        }
        return res;
    }
};

Matrix power(Matrix base, i64 b) {
    Matrix res = Matrix::getIdentity(base.n);
    while (b) {
        if (b & 1) res = res * base;
        b >>= 1;
        base = base * base;
    }
    return res;
}

void solve() {
    int n;
    std::cin >> n;
    Matrix B(6);
    B.matrix = {
        {1, 1, 0, 0, 1, 0},  // g(n) = g(n-1) + f(n) + n^2
        {0, 2, 3, 1, 0, 1},  // f(n+1) = 2f(n) + 3f(n-1) + n+1
        {0, 1, 0, 0, 0, 0},  // f(n) = f(n)
        {0, 0, 0, 1, 0, 1},  // n = n + 1
        {0, 0, 0, 2, 1, 1},  // n^2 = n^2 + 2n + 1
        {0, 0, 0, 0, 0, 1}   // 常数 1
    };

    std::vector<i64> v = {2, 2, 1, 2, 4, 1}; // 初始状态：g(1), f(2), f(1), n=2,
n^2=4, 常数1
    B = power(B, n - 1);

    i64 ans = 0;
    for (int i = 0; i < 6; i++) {
        ans = (ans + B.matrix[0][i] * v[i] % mod) % mod;
```

```
    }
    std::cout << ans << "\n";
}

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

    int t;
    std::cin >> t;
    while (t--) {
        solve();
    }

    return 0;
}
```