Contents

1 fanzhuan **2**

2 fastIO **3**

3 jls 快读 **5**

4 O3 **10**

5 大数 **11**

6 神奇的快读 **19**

7 纯快读 **26**

1 fanzhuan

```
auto f = [](unsigned int x) {
    x = ((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1)
        ;
    x = ((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2)
        ;
    x = ((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4)
        ;
    x = ((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8)
        ;
    x = ((x & 0xffff0000) >> 16) | ((x & 0x0000ffff) <<
        16);
    return x;
};
auto f2 = [f](unsigned int x, unsigned int k) {
    return f(x << (32 - k)) | (x >> k << k);
};
auto f=[](unsigned  long long x){
    x = ((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1)
        ;
    x = ((x & 0xcccccccc) >> 2) | ((x & 0x33333333) << 2)
        ;
    x = ((x & 0xf0f0f0f0) >> 4) | ((x & 0x0f0f0f0f) << 4)
        ;
    x = ((x & 0xff00ff00) >> 8) | ((x & 0x00ff00ff) << 8)
        ;
    x = ((x & 0xffff0000) >> 16) | ((x & 0x0000ffff) <<
        16);
    x=((x&0xffffffff000000))
};
```

2 fastIO

```
namespace IO {
#define getchar()(p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,
    stdin),p1==p2)?EOF:*p1++)
#define putchar(x)(*(p2=buf)+fwrite(buf,1,p2-buf,stdout))
    char buf[1 << 21], *p1 = buf, *p2 = buf;

    struct FastInput {
        template<class T>
        FastInput& operator>>(T &re) {
            re = 0;
            char ch = getchar();
            while (ch > '9' || ch < '0') ch = getchar();
            while (ch >= '0' && ch <= '9') re = 10 * re +
                ch - '0', ch = getchar();
            return *this;
        }
    };

    struct FastOutput {
        template<class T>
        FastOutput& operator<<(T x) {
            if (x == 0) {
                putchar('0');
                return *this;
            }
            static int st[60], top;
            top = 0;
            while (x) st[++top] = x % 10, x /= 10;
            for (int i = top; i >= 1; i--) putchar('0' +
               st[i]);
            return *this;
        }

        FastOutput& operator<<(char div) {
            putchar(div);
            return *this;
        }
    };

    FastInput cin;
    FastOutput cout;
}

using namespace IO;
```

```
#define cin IO::cin
#define cout IO::cout
```

3 jls 快读

```
namespace iof {
    const int Buffer_MAXSIZE = 0xfffff;

    const char endl = '\n';

    class ifast {
    public:
        ifast &operator>>(char &ch);

        ifast &operator>>(char *s);

        ifast &operator>>(std::string &string);

        ifast &operator>>(bool &x) { return
           read_unsigned_integer(x); }

        ifast &operator>>(int &x) { return
           read_signed_integer(x); }

        ifast &operator>>(long long &x) { return
           read_signed_integer(x); }

        ifast &operator>>(unsigned int &x) { return
           read_unsigned_integer(x); }

        ifast &operator>>(unsigned long long &x) { return
            read_unsigned_integer(x); }

        void getline(std::string &string);

        void getline(char *s);

        auto hasMoreToken() { return front <= back; }

    private:
        template<typename T>
        ifast &read_signed_integer(T &x);

        template<typename T>
        ifast &read_unsigned_integer(T &x);

        char getchar();

        char buffer[Buffer_MAXSIZE] {};
        char *front = buffer;

        char *back = buffer;

        char next_char{};
    } fin;

    class ofast {
    public:
        ofast &operator<<(const char &c);

        ofast &operator<<(const char *c);

        ofast &operator<<(const std::string &s);

        ofast &operator<<(const bool &x) { return
           print_integer(x); }

        ofast &operator<<(const int &x) { return
           print_integer(x); }

        ofast &operator<<(const long long &x) { return
           print_integer(x); }

        ofast &operator<<(const unsigned int &x) { return
            print_integer(x); }

        ofast &operator<<(const unsigned long long &x) {
           return print_integer(x); }

        void flush();

        ~ofast() { flush(); }

    private:
        void putchar(const char &c) { *top++ = c; }

        char buffer[Buffer_MAXSIZE] {};

        char *top = buffer;

        template<typename T>
        ofast &print_positive_integer(const T &x);

        template<typename T>
        ofast &print_integer(const T &x);
    } fout;

    char ifast::getchar() {
        if (front == back) {
            back = buffer + fread(buffer, 1,
                Buffer_MAXSIZE, stdin);
            front = buffer;
        }
        return *(front++);
    }

    void ifast::getline(std::string &string) {
        string.clear();
        for (*this >> next_char; hasMoreToken() &&
            next_char != '\n'; next_char = getchar()) {
            string.push_back(next_char);
        }
    }

    void ifast::getline(char *s) {
        for (char *p = s; hasMoreToken() && next_char !=
            '\n'; next_char = getchar(), p++) {
            *p = next_char;
        }
    }

    ifast &ifast::operator>>(char &ch) {
        do {
            ch = getchar();
        } while (hasMoreToken() && (ch == '␣' || ch == '\
            n'));
        return *this;
    }

    ifast &ifast::operator>>(char *s) {
        *this >> next_char;
        for (char *p = s; hasMoreToken() && next_char !=
            '␣' && next_char != '\n'; next_char = getchar
            (), p++) {
            *p = next_char;
        }
        return *this;
    }

    ifast &ifast::operator>>(std::string &string) {
        *this >> next_char;
        string.clear();
        for (; hasMoreToken() && next_char != '␣' &&
            next_char != '\n'; next_char = getchar()) {
            string.push_back(next_char);
        }
        return *this;
    }

    template<typename T>
    ifast &ifast::read_unsigned_integer(T &x) {
        *this >> next_char;
        x = 0;
        do {
            x = (x << 1) + (x << 3) + next_char - '0';
            next_char = getchar();
        } while (hasMoreToken() && next_char != '␣' &&
            next_char != '\n');
        return *this;
    }

    template<typename T>
    ifast &ifast::read_signed_integer(T &x) {
        *this >> next_char;
        auto isNegative = next_char == '-';
        if (isNegative) {
            next_char = getchar();
        }
        x = 0;
        do {
            x = (x << 1) + (x << 3) + next_char - '0';
            next_char = getchar();
        } while (hasMoreToken() && next_char != '␣' &&
            next_char != '\n');
        if (isNegative) {
            x = -x;
        }
        return *this;
    }

    ofast &ofast::operator<<(const char &c) {
        putchar(c);
        return *this;
    }

    ofast &ofast::operator<<(const char *c) {
        for (auto *p = c; *p != '\0'; p++) {
            putchar(*p);
        }
        return *this;
    }

    ofast &ofast::operator<<(const std::string &s) {
        for (const auto &i : s) {
            putchar(i);
        }
        return *this;
    }

    void ofast::flush() {
        fwrite(buffer, 1, top - buffer, stdout);
        top = buffer;
    }

    template<typename T>
    ofast &ofast::print_integer(const T &x) {
        if (x > 0) {
            return print_positive_integer(x);
        } else if (x == 0) {
            putchar('0');
            return *this;
        } else {
            putchar('-');
            return print_positive_integer(-x);
        }
    }

    template<typename T>
    ofast &ofast::print_positive_integer(const T &x) {
        if (x > 9) {
            print_positive_integer(x / 10);
        }
        putchar(x % 10 + '0');
        return *this;
    }
}
#define cin myCin
#define cout myCout
iof::ifast myCin;
iof::ofast myCout;
```

4 O3

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,avx512f,avx512vl,avx512bw,
    avx512dq,avx512cd,avx512vbmi,avx512vbmi2,
    avx512vpopcntdq,avx512bitalg,bmi,bmi2,lzcnt,popcnt")
#pragma GCC optimize("Ofast")
```

5 大数

```
const int base = 1000000000;
const int base_digits = 9; // 分解为九个数位一个数字
struct bigint {
    vector<int> a;
    int sign;

    bigint() : sign(1) {}
    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }
    bigint(long long v) {
        *this = v;
    }
    bigint(const string &s) {
        read(s);
    }
    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }
    void operator=(long long v) {
        a.clear();
        sign = 1;
        if (v < 0) sign = -1, v = -v;
        for (; v > 0; v = v / base) {
            a.push_back(v % base);
        }
    }

    // 基础加减乘除
    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;
            for (int i = 0, carry = 0; i < (int)max(a.
                size(), v.a.size()) || carry; ++i) {
                if (i == (int)res.a.size()) {
                    res.a.push_back(0);
                }
                res.a[i] += carry + (i < (int)a.size() ?
                    a[i] : 0);
                carry = res.a[i] >= base;
                if (carry) {
```

```
                res.a[i] -= base;
            }
        }
        return res;
    }
    return *this - (-v);
}
bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int)v.a.
                size() || carry; ++i) {
                res.a[i] -= carry + (i < (int)v.a.
                    size() ? v.a[i] : 0);
                carry = res.a[i] < 0;
                if (carry) {
                    res.a[i] += base;
                }
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}
void operator*=(int v) {
    check(v);
    for (int i = 0, carry = 0; i < (int)a.size() ||
        carry; ++i) {
        if (i == (int)a.size()) {
            a.push_back(0);
        }
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
    }
    trim();
}
void operator/=(int v) {
    check(v);
    for (int i = (int)a.size() - 1, rem = 0; i >= 0;
        --i) {
        long long cur = a[i] + rem * (long long)base;
        a[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
}
int operator%(int v) const {
    if (v < 0) {
        v = -v;
    }
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i) {
        m = (a[i] + m * (long long)base) % v;
    }
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}
void operator-=(const bigint &v) {
    *this = *this - v;
}
bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}
bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}
void operator%=(const int &v) {
    *this = *this % v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign) return sign < v.sign;
    if (a.size() != v.a.size()) return a.size() *
        sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i]) return a[i] * sign < v.a[
            i] * sign;
    return false;
}
bool operator>(const bigint &v) const {
    return v < *this;
```

```
}
bool operator<=(const bigint &v) const {
    return !(v < *this);
}
bool operator>=(const bigint &v) const {
    return !(*this < v);
}
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}
void check(int v) { // 检查输入的是否为负数
    if (v < 0) {
        sign = -sign;
        v = -v;
    }
}
void trim() { // 去除前导零
    while (!a.empty() && !a.back()) a.pop_back();
    if (a.empty()) sign = 1;
}
bool isZero() const { // 判断是否等于零
    return a.empty() || (a.size() == 1 && !a[0]);
}
friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}
void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int)s.size() && (s[pos] == '-' || s
        [pos] == '+')) {
        if (s[pos] == '-') sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -=
        base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j
             <= i; j++) x = x * 10 + s[j] - '0';
        a.push_back(x);
    }
    trim();
}
friend istream &operator>>(istream &stream, bigint &v
    ) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}
friend ostream &operator<<(ostream &stream, const
    bigint &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int)v.a.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill('0')
            << v.a[i];
    return stream;
}

/* 大整数乘除大整数部分 */
typedef vector<long long> vll;
bigint operator*(const bigint &v) const { // 大整数乘
    大整数
    vector<int> a6 = convert_base(this->a,
        base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits,
        6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size()) a.push_back(0);
    while (b.size() < a.size()) b.push_back(0);
    while (a.size() & (a.size() - 1)) a.push_back(0),
         b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int)c.size(); i
        ++) {
```

```
        long long cur = c[i] + carry;
        res.a.push_back((int)(cur % 1000000));
        carry = (int)(cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
friend pair<bigint, bigint> divmod(const bigint &a1,
                                   const bigint &b1)
                                      { // 大整数除大
                                      整数，同时返回
                                      答案与余数
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());
    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b
           .a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r
           .a[b.a.size() - 1];
        int d = ((long long)base * s1 + s2) / b.a.
           back();
        r -= b * d;
        while (r < 0) r += b, --d;
        q.a[i] = d;
    }
    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}
static vector<int> convert_base(const vector<int> &a,
    int old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) +
        1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++) p[i] = p[
       i - 1] * 10;
    vector<int> res;
    long long cur = 0;
```

```
        int cur_digits = 0;
        for (int i = 0; i < (int)a.size(); i++) {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back((int)(cur % p[new_digits]))
                    ;
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int)cur);
        while (!res.empty() && !res.back()) res.pop_back
            ();
        return res;
    }
    static vll karatsubaMultiply(const vll &a, const vll
       &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

        for (int i = 0; i < k; i++) a2[i] += a1[i];
        for (int i = 0; i < k; i++) b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int)a1b1.size(); i++) r[i]
            -= a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) r[i]
            -= a2b2[i];
        for (int i = 0; i < (int)r.size(); i++) res[i + k
            ] += r[i];
        for (int i = 0; i < (int)a1b1.size(); i++) res[i]
             += a1b1[i];
        for (int i = 0; i < (int)a2b2.size(); i++) res[i
            + n] += a2b2[i];
        return res;
    }

    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    bigint operator/(const bigint &v) const {
        return divmod(*this, v).first;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }
    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }
    void operator%=(const bigint &v) {
        *this = *this % v;
    }
};
```

6 神奇的快读

```
#ifndef __OY_FASTIO__
#define __OY_FASTIO__
#define cin OY::IO::InputHelper::get_instance()
#define cout OY::IO::OutputHelper::get_instance()
#define endl '\n'
#ifndef INPUT_FILE
#ifdef OY_LOCAL
#define INPUT_FILE "in.txt"
#else
#define INPUT_FILE ""
#endif
#endif
#ifndef OUTPUT_FILE
#ifdef OY_LOCAL
#define OUTPUT_FILE "out.txt"
#else
#define OUTPUT_FILE ""
#endif
#endif
namespace OY {
    namespace IO {
        using size_type = size_t;
        static constexpr size_type INPUT_BUFFER_SIZE = 1
           << 16, OUTPUT_BUFFER_SIZE = 1 << 16,
           MAX_INTEGER_SIZE = 20, MAX_FLOAT_SIZE = 50;
        static constexpr char input_file[] = INPUT_FILE,
           output_file[] = OUTPUT_FILE;
        struct InputHelper {
            FILE *m_file_ptr;
            char m_buf[INPUT_BUFFER_SIZE], *m_end, *
               m_cursor;
            bool m_ok;
            InputHelper &set_bad() { return m_ok = false,
                *this; }
            template <size_type BlockSize>
            void _reserve() {
                size_type a = m_end - m_cursor;
                if (a >= BlockSize) return;
                memmove(m_buf, m_cursor, a), m_cursor =
                    m_buf;
                size_type b = a + fread(m_buf + a, 1,
                    INPUT_BUFFER_SIZE - a, m_file_ptr);
                if (b < INPUT_BUFFER_SIZE) m_end = m_buf
                    + b, *m_end = EOF;
```

```
}
template <typename Tp, typename
    BinaryOperation>
InputHelper &fill_integer(Tp &ret,
    BinaryOperation op) {
     if (!isdigit(*m_cursor)) return set_bad()
         ;
     ret = op(Tp(0), *m_cursor - '0');
     size_type len = 1;
     while (isdigit(*(m_cursor + len))) ret =
         op(ret * 10, *(m_cursor + len++) - '0'
         );
     m_cursor += len;
     return *this;
}
explicit InputHelper(const char *
    inputFileName) : m_ok(true), m_cursor(
    m_buf + INPUT_BUFFER_SIZE), m_end(m_buf +
    INPUT_BUFFER_SIZE) { m_file_ptr = *
    inputFileName ? fopen(inputFileName, "rt")
     : stdin; }
~InputHelper() { fclose(m_file_ptr); }
static InputHelper &get_instance() {
     static InputHelper s_obj(input_file);
     return s_obj;
}
static bool is_blank(char c) { return c == '␣
    ' || c == '\t' || c == '\n' || c == '\r';
    }
static bool is_endline(char c) { return c ==
    '\n' || c == EOF; }
const char &getchar_checked() {
     _reserve<1>();
     return *m_cursor;
}
const char &getchar_unchecked() const {
    return *m_cursor; }
void next() { ++m_cursor; }
template <typename Tp, typename std::
    enable_if<std::is_signed<Tp>::value & std
    ::is_integral<Tp>::value>::type * =
    nullptr>
InputHelper &operator>>(Tp &num) {
     while (is_blank(getchar_checked())) next
         ();
     _reserve<MAX_INTEGER_SIZE>();
    if (getchar_unchecked() != '-') return
        fill_integer(num, std::plus<Tp>());
    next();
    return fill_integer(num, std::minus<Tp>()
        );
}
template <typename Tp, typename std::
    enable_if<std::is_unsigned<Tp>::value &
    std::is_integral<Tp>::value>::type * =
    nullptr>
InputHelper &operator>>(Tp &num) {
    while (is_blank(getchar_checked())) next
        ();
    _reserve<MAX_INTEGER_SIZE>();
    return fill_integer(num, std::plus<Tp>())
        ;
}
template <typename Tp, typename std::
    enable_if<std::is_floating_point<Tp>::
    value>::type * = nullptr>
InputHelper &operator>>(Tp &num) {
    bool neg = false, integer = false,
        decimal = false;
    while (is_blank(getchar_checked())) next
        ();
    _reserve<MAX_FLOAT_SIZE>();
    if (getchar_unchecked() == '-') {
        neg = true;
        next();
    }
    if (!isdigit(getchar_unchecked()) &&
        getchar_unchecked() != '.') return
        set_bad();
    if (isdigit(getchar_unchecked())) {
        integer = true;
        num = getchar_unchecked() - '0';
        while (next(), isdigit(
            getchar_unchecked())) num = num *
            10 + (getchar_unchecked() - '0');
    }
    if (getchar_unchecked() == '.')
        if (next(), isdigit(getchar_unchecked
            ())) {
            if (!integer) num = 0;
            decimal = true;
            Tp unit = 0.1;
```

```
                num += unit * (getchar_unchecked
                    () - '0');
                while (next(), isdigit(
                    getchar_unchecked())) num += (
                    unit *= 0.1) * (
                    getchar_unchecked() - '0');
            }
        if (!integer && !decimal) return set_bad
            ();
        if (neg) num = -num;
        return *this;
    }
    InputHelper &operator>>(char &c) {
        while (is_blank(getchar_checked())) next
            ();
        if (getchar_checked() == EOF) return
            set_bad();
        c = getchar_checked(), next();
        return *this;
    }
    InputHelper &operator>>(std::string &s) {
        while (is_blank(getchar_checked())) next
            ();
        if (getchar_checked() == EOF) return
            set_bad();
        s.clear();
        do {
            s += getchar_checked();
            next();
        } while (!is_blank(getchar_checked()) &&
            getchar_unchecked() != EOF);
        return *this;
    }
    explicit operator bool() { return m_ok; }
};
struct OutputHelper {
    FILE *m_file_ptr = nullptr;
    char m_buf[OUTPUT_BUFFER_SIZE], *m_end, *
        m_cursor;
    char m_temp_buf[MAX_FLOAT_SIZE], *
        m_temp_buf_cursor, *m_temp_buf_dot;
    uint64_t m_float_reserve, m_float_ratio;
    void _write() { fwrite(m_buf, 1, m_cursor -
        m_buf, m_file_ptr), m_cursor = m_buf; }
    template <size_type BlockSize>
    void _reserve() {
        size_type a = m_end − m_cursor;
        if (a >= BlockSize) return;
        _write();
    }
    OutputHelper(const char *outputFileName,
       size_type prec = 6) : m_cursor(m_buf),
       m_end(m_buf + OUTPUT_BUFFER_SIZE),
       m_temp_buf_cursor(m_temp_buf) { m_file_ptr
        = *outputFileName ? fopen(outputFileName,
        ”wt”) : stdout, precision(prec); }
    static OutputHelper &get_instance() {
        static OutputHelper s_obj(output_file);
        return s_obj;
    }
    ~OutputHelper() { flush(), fclose(m_file_ptr)
       ; }
    void precision(size_type prec) {
       m_float_reserve = prec, m_float_ratio =
       uint64_t(std::pow(10, prec)),
       m_temp_buf_dot = m_temp_buf + prec; }
    OutputHelper &flush() { return _write(),
       fflush(m_file_ptr), *this; }
    void putchar(const char &c) {
        if (m_cursor == m_end) _write();
        *m_cursor++ = c;
    }
    template <typename Tp, typename std::
       enable_if<std::is_signed<Tp>::value & std
       ::is_integral<Tp>::value>::type * =
       nullptr>
    OutputHelper &operator<<(Tp ret) {
        _reserve<MAX_INTEGER_SIZE>();
        size_type len = 0;
        if (ret >= 0)
            do *(m_cursor + len++) = '0' + ret %
                10, ret /= 10;
            while (ret);
        else {
            putchar('−');
            do *(m_cursor + len++) = '0' − ret %
                10, ret /= 10;
            while (ret);
        }
        for (size_type i = 0, j = len − 1; i < j
           ;) std::swap(*(m_cursor + i++), *(
           m_cursor + j−−));
    m_cursor += len;
    return *this;
}
template <typename Tp, typename std::
   enable_if<std::is_unsigned<Tp>::value &
   std::is_integral<Tp>::value>::type * =
   nullptr>
OutputHelper &operator<<(Tp ret) {
    _reserve<MAX_INTEGER_SIZE>();
    size_type len = 0;
    do *(m_cursor + len++) = '0' + ret % 10,
        ret /= 10;
    while (ret);
    for (size_type i = 0, j = len - 1; i < j
        ;) std::swap(*(m_cursor + i++), *(
        m_cursor + j--));
    m_cursor += len;
    return *this;
}
template <typename Tp, typename std::
   enable_if<std::is_floating_point<Tp>::
   value>::type * = nullptr>
OutputHelper &operator<<(Tp ret) {
    if (ret < 0) {
        putchar('-');
        return *this << -ret;
    }
    ret *= m_float_ratio;
    uint64_t integer = ret;
    if (ret - integer >= 0.4999999999)
        integer++;
    do {
        *m_temp_buf_cursor++ = '0' + integer
            % 10;
        integer /= 10;
    } while (integer);
    if (m_temp_buf_cursor > m_temp_buf_dot) {
        do putchar(*--m_temp_buf_cursor);
        while (m_temp_buf_cursor >
            m_temp_buf_dot);
        putchar('.');
    } else {
        putchar('0'), putchar('.');
        for (size_type i = m_temp_buf_dot -
            m_temp_buf_cursor; i--;) putchar('
            0');
```

```
                }
                do putchar(*--m_temp_buf_cursor);
                while (m_temp_buf_cursor > m_temp_buf);
                return *this;
            }
            OutputHelper &operator<<(const char &ret) {
                putchar(ret);
                return *this;
            }
            OutputHelper &operator<<(const char *ret) {
                while (*ret) putchar(*ret++);
                return *this;
            }
            OutputHelper &operator<<(const std::string &
                ret) { return *this << ret.data(); }
        };
        InputHelper &getline(InputHelper &ih, std::string
            &line) {
            line.clear();
            if (ih.getchar_checked() == EOF) return ih.
                set_bad();
            while (!InputHelper::is_endline(ih.
                getchar_checked())) line += ih.
                getchar_unchecked(), ih.next();
            if (ih.getchar_unchecked() != EOF) ih.next();
            return ih;
        }
    }
}
using OY::IO::getline;

#endif /*__OY_FASTIO__*/
```

7 纯快读

```
namespace IO {
#define getchar()(p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,
    stdin),p1==p2)?EOF:*p1++)
    char buf[1 << 21], *p1 = buf, *p2 = buf;
    struct ifast {
        template<class T>
        ifast& operator>>(T &re) {
            re = 0;
            char ch = getchar();
            while (!isdigit(ch)) ch = getchar();
            while (isdigit(ch)) re = re*10 + ch - '0', ch
                = getchar();
            return *this;
        }
    };
    ifast cin;
}
using namespace IO;
#define cin IO::cin
```