目录

- 1 字符串
- 2 数学
  - 2.1 GCD & LCM
    - 2.1.1 非递归 GCD
    - 2.1.2 递归 GCD
    - 2.1.3 LCM
  - 2.2 扩展GCD
- 3 数据结构
  - 3.1 单调队列
  - 3.2 ZKW线段树
- 4 图论
- 5 动态规划
- 6 计算几何
- 7 其他
  - 7.1 高精度

1 字符串

2 数学

2.1 GCD & LCM

- require: c++98
- [x] 封装
- [ ] 已测试

gcd: 两个整数的最大公因数 (greatest common divisor)
lcm: 两个整数的最小公倍数 (least common multiple)

非递归版本参考于GNU `__gcd()` 源码。
递归版本参考于 `std::gcd()` 源码。

相关函数

- `std::gcd()`
  - 定义在 `<numeric>`
  - require: C++17
- `std::lcm()`
  - 定义在 `<numeric>`
  - require: C++17
- `__gcd()`
  - 定义在`<algorithm>`
  - require: C++98, GNU平台

2.1.1 非递归 GCD

```
template <typename T> T gcd(T m, T n) {
    while (n != 0) {
        T t = m % n;
        m = n;
        n = t;
    }
    return m;
}
```

2.1.2 递归 GCD

```
template <typename T> T gcd(T m, T n) {
    return m == 0 ? n : n == 0 ? m : gcd(n, m % n);
}
```

2.1.3 LCM

```
template <typename T> T lcm(T m, T n) {
    return (m != 0 && n != 0) ? (m / gcd(m, n)) * n : 0;
}
```

2.2 扩展GCD

3 数据结构

3.1 单调队列

- require: C++98
- ☑ 封装
- ☑ 已测试

这个版本相当于是对STL库中 `deque` 的重写，但引入了比较模板类。实际使用场景比较灵活，可能并不能照搬，但可以作为参考，并在STL的 `deque` 太慢时作为替换。

```
template <typename T, typename Cmp = less_equal<T> > struct Monoq {
    Cmp comp;
    const static int N = MAXN;
    T q[N];
    int ft, bk;
    inline Monoq() : ft(0), bk(0) {}
    inline bool empty() { return bk - ft <= 0; }
    inline int size() { return bk - ft; }
    inline T front() { return q[ft]; }
    inline T back() { return q[bk - 1]; }
    inline void push(T x) {
        while (!empty() && comp(x, back()))
            bk--;
        q[bk++] = x;
    }
    inline void pop_back() {
        if (!empty())
            bk--;
    }
    inline void pop_front() {
        if (!empty())
            ft++;
    }
    inline void clear() { ft = 0, bk = 0; }
    inline T *begin() { return q + ft; }
    inline T *end() { return q + bk; }
};

int a[50] = {3, 6, 7, 5, 3, 5, 6, 2, 9, 1, 2, 7, 0, 9, 3, 6, 0, 6, 2, 6};
struct MyCmp {
    inline bool operator()(int x, int y) {
        return a[x] <= a[y]; // strict increase monoq
        // return a[x] >= a[y]; // strict decrease monoq
    }
};

Monoq<int, MyCmp> q;
```

3.2 ZKW线段树

- require: C++98
- HDU1698
- [ ] 封装
- [x] 已测试

区间修改 + 区间查询

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

const int M = 1 << 17;
const int INF = 1e9;

int T[M + M + 1], lazy[M + M + 1];

void modify(int ll, int rr, int v) {
    ll += M - 1, rr += M + 1;
    for (int i = 20, l, r; i; i--) {
        l = ll >> i, r = rr >> i;
        if (lazy[l]) {
            lazy[l * 2] = lazy[l * 2 + 1] = lazy[l];
            T[l * 2] = T[l * 2 + 1] = lazy[l] * (1 << (i - 1));
            lazy[l] = 0;
        }
        if (lazy[r]) {
            lazy[r * 2] = lazy[r * 2 + 1] = lazy[r];
            T[r * 2] = T[r * 2 + 1] = lazy[r] * (1 << (i - 1));
            lazy[r] = 0;
        }
    }
    for (int l = ll, r = rr, num = 1; l > 1; l >>= 1, r >>= 1, num <<= 1) {
        if ((l ^ r ^ 1) > 1) {
            if (~l & 1)
                lazy[l ^ 1] = v, T[l ^ 1] = v * num;
            if (r & 1)
                lazy[r ^ 1] = v, T[r ^ 1] = v * num;
        }
        T[l >> 1] = T[l] + T[l ^ 1];
        T[r >> 1] = T[r] + T[r ^ 1];
    }
}

int query(int l, int r) {
    int ansL = 0, ansR = 0, ln = 0, rn = 0, nn = 1;
    for (l += M - 1, r += M + 1; l ^ r ^ 1; l >>= 1, r >>= 1, nn <<= 1) {
        if (lazy[l])
            ansL = lazy[l] * ln;
        if (lazy[r])
            ansR = lazy[r] * rn;
        if (~l & 1)
```

```
            ansL += T[l ^ 1], ln += nn;
        if (r & 1)
            ansR += T[r ^ 1], rn += nn;
    }
    for (; l; l >>= 1, r >>= 1) {
        if (lazy[l])
            ansL = lazy[l] * ln;
        if (lazy[r])
            ansR = lazy[r] * rn;
    }
    return ansL + ansR;
}

int main() {
    ios::sync_with_stdio(false);
    int t;
    cin >> t;
    for (int ca = 1; ca <= t; ca++) {
        int n, q;
        cin >> n >> q;
        modify(1, n, 1);
        while (q--) {
            int x, y, z;
            cin >> x >> y >> z;
            modify(x, y, z);
        }
        printf("Case %d: The total value of the hook is %d.\n", ca,
               query(1, n));
    }
}
```

4 图论

5 动态规划

6 计算几何

7 其他

7.1 高精度

- require: C++11
- [x] 封装
- [ ] 已测试

高精度四则运算。

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

struct Unsigned_BigInt {
    ll k = 10; // base-k positional notation
    vector<ll> a;
    Unsigned_BigInt() {
        a.clear();
        a.push_back(0);
    }
    Unsigned_BigInt(ll v) {
        a.clear();
        a.push_back(abs(v));
        this->regular();
    }
    Unsigned_BigInt(string s) {
        a.clear();
        for (ll i = s.length() - 1; i >= 0 && s[i] != '-'; i--)
            a.push_back(s[i] - '0');
    }
    void regular() {
        for (ll i = 0; i < a.size(); i++)
            if (a[i] >= k || a[i] < 0) {
                if (i + 1 < a.size())
                    a[i + 1] += (a[i] >= 0 ? a[i] / k : (a[i] + 1) / k - 1);
                else
                    a.push_back(a[i] / k);
                a[i] = (a[i] % k + k) % k;
            }
    }
    void give(ll i, ll v) {
        if (i < a.size())
            a[i] += v;
        else
            a.push_back(v);
    }
    void shrink() {
        for (ll i = a.size() - 1; i >= 0 && a[i] == 0; i--)
            a.pop_back();
        if (a.empty())
            a.push_back(0);
    }
    bool operator<(const Unsigned_BigInt &b) const {
        if (a.size() == b.a.size()) {
            ll i = a.size() - 1;
            while (a[i] == b.a[i] && i >= 0)
                i--;
            return i >= 0 ? a[i] < b.a[i] : false;
        } else
            return a.size() < b.a.size();
    }
    Unsigned_BigInt operator+(const Unsigned_BigInt &b) const {
        Unsigned_BigInt c;
        for (ll i = 0; i < max(a.size(), b.a.size()); i++)
            c.give(i,
                   (i < a.size() ? a[i] : 0) + (i < b.a.size() ? b.a[i] : 0));
        c.regular();
        return c;
    }
    Unsigned_BigInt operator-(const Unsigned_BigInt &b) const {
        Unsigned_BigInt c;
        bool less = *this < b;
        for (ll i = 0; i < max(a.size(), b.a.size()); i++) {
            ll temp = (i < a.size() ? a[i] : 0) - (i < b.a.size() ? b.a[i] : 0);
            c.give(i, !less ? temp : -temp);
        }
        c.regular();
        c.shrink();
        return c;
    }
    Unsigned_BigInt operator*(const ll &b) const {
        ll bb = abs(b);
        Unsigned_BigInt c;
        for (ll i = 0; i < a.size(); i++)
            c.give(i, a[i] * bb);
        c.regular();
        return c;
    }
    Unsigned_BigInt operator*(const Unsigned_BigInt &b) const {
        Unsigned_BigInt c;
        for (ll i = 0; i < b.a.size(); i++)
            for (ll j = 0; j < a.size(); j++)
                c.give(j + i, a[j] * b.a[i]);
        c.regular();
        return c;
    }
    Unsigned_BigInt operator/(const ll &b) const {
        ll bb = abs(b);
        Unsigned_BigInt c = *this;
        ll dividend = 0;
        for (ll i = a.size() - 1; i >= 0; i--) {
            dividend = dividend * k + a[i];
            c.a[i] = dividend / bb;
            dividend %= bb;
        }
        c.shrink();
        return c;
    }
    ll operator%(const ll &b) const {
        ll r = 0;
        for (ll i = a.size() - 1; i >= 0; i--)
            r = ((r * k) % b + a[i]) % b;
        return r;
    }
    void print() {
        for (auto it = a.rbegin(); it != a.rend(); it++)
            // cout << *it;
            printf("%lld", *it);
        // cout << endl;
        printf("\n");
    }
};

struct BigInt {
    bool sign = false; // 0: +, 1: -
    Unsigned_BigInt num;
    BigInt() : sign(0), num(Unsigned_BigInt()) {}
    BigInt(ll v) : sign(v < 0), num(Unsigned_BigInt(v)) {}
    BigInt(string s) : sign(s[0] == '-'), num(Unsigned_BigInt(s)) {}
    BigInt(Unsigned_BigInt num, bool sign = false) : sign(sign), num(num) {}
    bool operator<(const BigInt &b) const {
        if (sign ^ b.sign)
            return sign;
        else
            return ((!sign) ? num < b.num : b.num < num);
    }
    BigInt operator+(const BigInt &b) const {
        if (sign)
            return -(-*this - b); // a + b == -(-a - b)
        if (b.sign)
            return *this - (-b); // a + b == a - (-b)
        return BigInt(num + b.num);
    }
    BigInt operator-() const { return BigInt(num, !sign); }
    BigInt operator-(const BigInt &b) const {
        if (sign)
            return -(-*this + b); // a - b == -(-a + b)
        if (b.sign)
            return *this + (-b); // a - b == a + (-b)
        return BigInt(num - b.num, num < b.num);
    }
    BigInt operator*(const ll &b) const {
        return BigInt(num * b, sign ^ b < 0);
    }
    BigInt operator*(const BigInt &b) const {
        return BigInt(num * b.num, sign ^ b.sign);
    }
    BigInt operator/(const ll &b) const {
        return BigInt(num / b, sign ^ b < 0);
    }
    ll operator%(const ll &b) const { return sign ? -(num % b) : num % b; }
    void print() {
        // cout << (!sign || (num.a.size() == 1 && num.a[0] == 0) ? "" : "-");
        printf(!sign || (num.a.size() == 1 && num.a[0] == 0) ? "" : "-");
        num.print();
    }
};

ostream &operator<<(ostream &os, const Unsigned_BigInt &b) {
    for (auto it = b.a.rbegin(); it != b.a.rend(); it++)
        os << *it;
    return os;
}
ostream &operator<<(ostream &os, const BigInt &b) {
    os << (!b.sign || (b.num.a.size() == 1 && b.num.a[0] == 0) ? "" : "-");
    os << b.num;
    return os;
}

int main() {
    string s1;
    string s2;
    // freopen("./test.in", "r", stdin);
    // freopen("./test.out", "w", stdout);

    while (cin >> s1 >> s2) {
        cout << (BigInt(s1) * BigInt(s2));
        // (BigInt(s1) * BigInt(s2)).print();
    }

    // fclose(stdin);
    // fclose(stdout);
}
```