Changsha University of Science and Technology

ACM/ICPC Templates

Kenshin2438

The Alchemist of Pokémon

When you have eliminated the impossibles,
whatever remains, however improbable, must be the truth.

2023 年 1 月 17 日

目录

第一章　Misc

1.1　Debug

1.1.1　一些心得

当榜上过了很多人，但是你却没思路时，试试下面这些？

- 想一想**数据范围是否有特殊意义**。
- 如果是一些数学题，考虑**打表找规律，总比死磕要好**。
- 什么？是博弈？哦，两个聪明人的事，咱们不掺和。想不出就别死撑着了，**SG 函数**和 **Minimax 搜索**开冲！
- 多翻一下带来的板子，也许是自己不会的**人均算法**呢？
- 如果你的算法复杂度比较正确（且自己已经不能再优化了），考虑玄学（Miller-Rabin / Rho 随机化等等）；或者想想暴力优化？**能过这么多，总归是的有道理**。
- 选择放弃。**就你不会写那很可能就是你太菜了，换一个题自闭去**。
- 对着队友语言输出！然后把题交给队友。

1.1.2　Random Number

```cpp
std::mt19937 rng(__builtin_ia32_rdtsc());
template <typename T>
inline T randint(T l, T r) {
  return std::uniform_int_distribution<T>(l, r)(rng);
}
template <typename E>
inline E randreal(E l, E r) {
  return std::uniform_real_distribution<E>(l, r)(rng);
}
```

1.1.3 CMD 对拍 bat 脚本

```
@echo off

:loop
  gen.exe > _.in
  ac.exe < _.in > _.out
  bf.exe < _.in > _.ans
  fc _.out _.ans
if not errorlevel 1 goto loop
pause
goto loop
```

1.2 Int128

```
using i128 = __int128;

istream &operator>>(istream &is, i128 &v) {
  string s;
  is >> s, v = 0;
  for (const char &c : s) {
    if (c >= '0' && c <= '9') v = v * 10 + (c & 15);
  }
  if (s.front() == '-') v = -v;
  return is;
}

ostream &operator<<(ostream &os, const i128 &v) {
  if (v == 0) return (os << "0");
  i128 num = v;
  string s;
  if (v < 0) os << '-', num = -num;
  for (; num > 0; num /= 10) s.push_back(char(num % 10) + '0');
  return reverse(all(s)), (os << s);
}
```

1.3 ModInt

```
template <uint32_t mod> struct m32 {
  static_assert(mod < (1U << 31), "Modulus error!");

  using u32 = uint32_t;
  using u64 = uint64_t;
  using i64 = int64_t;

  u32 v = 0;
  template <typename T> u32 norm(T v) {
    return static_cast<u32>((v %= mod) < 0 ? mod + v : v);
  }
```

```cpp
  m32() = default;
  template <typename T> m32(T _ = 0) : v(norm(_)) {}
  ~m32() = default;

  m32 &operator=(const int &rhs) { return v = norm(rhs), *this; }
  m32 operator-() const {
    return v == 0 ? m32(0) : m32(mod - v);
  }
  m32 &operator+=(const m32 &rhs) {
    v += rhs.v;
    if (v >= mod) v -= mod;
    return *this;
  }
  m32 &operator-=(const m32 &rhs) {
    if (v < rhs.v) v += mod;
    v -= rhs.v;
    return *this;
  }
  m32 &operator*=(const m32 &rhs) {
    v = (u64)v * rhs.v % mod;
    return *this;
  }
  m32 &operator/=(const m32 &rhs) { return *this *= rhs.inv(); }
  m32 operator+(const m32 &rhs) const { return m32(*this) += rhs;
    }
  m32 operator-(const m32 &rhs) const { return m32(*this) -= rhs;
    }
  m32 operator*(const m32 &rhs) const { return m32(*this) *= rhs;
    }
  m32 operator/(const m32 &rhs) const { return m32(*this) /= rhs;
    }
  bool operator==(const m32 &rhs) const { return rhs.v == v; }
  bool operator!=(const m32 &rhs) const { return rhs.v != v; }
  m32 pow(i64 n) const {
    m32 x(*this), res(1);
    for (; n > 0; n >>= 1, x *= x)
      if (n & 1LL) res *= x;
    return res;
  }
  m32 inv() const {
    assert(v != 0);
    return pow(mod - 2);
  }

  static u32 get_mod() { return mod; }
  friend ostream &operator<<(ostream &os, const m32 &m) {
    return os << m.v;
  }
  friend istream &operator>>(istream &is, m32 &m) {
    long long a; is >> a; m = m32<mod>(a);
    return is;
  }
};

const int mod = 998244353;
using mint = m32<mod>;
```

1.4 Tree Hash

```
#include "../data_structure/hash_map.hpp"

struct tree_hash {
  int n; // vectex number
  vector<vector<int>> tree;
  vector<ull> H, G;
  // H(u) -> Hash value of subtree(u)
  // G(u) -> Hash value of tree(u-rooted)
  tree_hash(int _n) : n(_n), tree(n), H(n, 1), G(n) {};

  void build() {
    for (int i = 1; i < n; i++) {
      int u, v; cin >> u >> v;
      --u, --v;
      tree[u].emplace_back(v);
      tree[v].emplace_back(u);
    }
    function<void(int, int)> dfs = [&](int u, int fa) {
      for (const int &v : tree[u]) {
        if (v == fa) continue;
        dfs(v, u);
        H[u] += splitmix64(H[v] ^ SEED);
      }
    };
    dfs(0, 0);
    G[0] = H[0];
    function<void(int, int)> sol = [&](int u, int fa) {
      for (const int &v : tree[u]) {
        if (v == fa) continue;
        G[v] = H[v] + splitmix64((G[u] - splitmix64(H[v] ^ SEED))
            ^ SEED);
        sol(v, u);
      }
    };
    sol(0, 0);
  }
};
```

第二章　Number Theory

2.1　BSGS

```
// a^x EQUIV n (MOD mod), and gcd(a, mod) = 1
ll BSGS(ll a, ll n, ll mod) {
  a %= mod, n %= mod;
  if (n == 1LL || mod == 1LL) return 0LL;

  unordered_map<ll, ll> bs; // Attention !
  ll S = sqrt(mod) + 1;

  ll base = n;
  for (ll k = 0, val = base; k <= S; k++) {
    bs[val] = k, val = val * a % mod;
  }
  base = qpow(a, S, mod);
  for (ll x = 1, val = base; x <= S; x++) {
    if (bs.count(val)) return x * S - bs[val];
    val = val * base % mod;
  }
  return -1;  // No solution
}

ll inv(ll a, ll mod) {
  auto [res, _] = exgcd(a, mod);
  return (res % mod + mod) % mod;
}

// a^x EQUIV n (MOD mod), and gcd(a, mod) != 1
ll exBSGS(ll a, ll n, ll mod) {
  a %= mod, n %= mod;
  if (n == 1LL || mod == 1LL) return 0LL;

  ll k = 0, val = 1;
  for (ll g = __gcd(a, mod); g != 1LL; g = __gcd(a, mod)) {
    if (n % g != 0LL) return -1;  // No solution
    mod /= g, n /= g;
    val = val * (a / g) % mod, k++;
    if (val == n) return k;
  }
  ll res = BSGS(a, n * inv(val, mod) % mod, mod);
  return ~res ? res + k : res;
}
```

2.2 二次剩余

2.2.1 Cipolla

```
struct R {
  ll a, p, x, y;

  R(ll _a, ll _p) : a(_a), p(_p) { x = 1LL, y = 0LL; }
  void rand() {
    x = randint(0, p - 1);
    y = randint(0, p - 1);
  }
  R &operator*=(const R &rhs) {
    ll _x = (x * rhs.x + y * rhs.y % p * a) % p;
    ll _y = (x * rhs.y + y * rhs.x) % p;
    x = _x, y = _y;
    return *this;
  }

  void pow(ll n) {
    R res(a, p), b = *this;
    for (; n; n >>= 1, b *= b) {
      if (n & 1LL) res *= b;
    }
    x = res.x, y = res.y;
  }
};

ll Cipolla(ll a, ll p) {
  a = (a % p + p) % p;
  if (a == 0) return 0LL;
  // No Solution
  if (qpow(a, (p - 1) / 2, p) != 1LL) return -1LL;
  if (p % 4 == 3) return qpow(a, (p + 1) / 4, p);

  R t(a, p);
  while (true) {
    t.rand(), t.pow((p - 1) / 2);
    if (t.x == 0 && t.y != 0) {
      return qpow(t.y, p - 2, p);
    }
  }

  assert(false);
  return -1;
}
```

2.2.2 Tonelli Shanks

```
// 一般 O(log p); 最坏 O(log^2 p);
ll Tonelli_Shanks(ll a, ll p) {
  a = (a % p + p) % p;
```

```
  if (a == 0) return 0LL;
  // No Solution
  if (qpow(a, (p - 1) / 2, p) != 1LL) return -1LL;
  if (p % 4 == 3) return qpow(a, (p + 1) / 4, p);

  ll k = __builtin_ctzll(p - 1), h = p >> k, N = 2;
  // p = 1 + h * 2^k
  while (qpow(N, (p - 1) / 2, p) == 1) N++;
  // find a non-square mod p

  ll x = qpow(a, (h + 1) / 2, p);
  ll g = qpow(N, h, p);
  ll b = qpow(a, h, p);

  for (ll m = 0;; k = m) {
    ll t = b;
    for (m = 0; m < k && t != 1LL; m++) {
      t = t * t % p;
    }

    if (m == 0) return x;
    ll gs = qpow(g, 1 << (k - m - 1), p);

    g = gs * gs % p;
    b = b * g % p;
    x = x * gs % p;
  }

  assert(false);
  return -1;
}
```

2.3 Ex-gcd

得到的结果满足 $|x| + |y|$ 最小 (首要的) 同时有 x y (其次)。

```
template <typename T> pair<T, T> exgcd(T a, T b) {
  bool nega = (a < 0), negb = (b < 0);
  T x = 1, y = 0, r = 0, s = 1;
  while (b) {
    T t = a / b;
    r ^= x ^= r ^= x -= t * r;
    s ^= y ^= s ^= y -= t * s;
    b ^= a ^= b ^= a %= b;
  }
  return {nega ? -x : x, negb ? -y : y};
}
```

2.4 Miller-Rabin Test

```
using ull = unsigned long long;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
bool miller_rabin(ull n) {
  static const vector<ull> SPRP = {
    2, 325, 9375, 28178, 450775, 9780504, 1795265022
  };
  if (n == 1 || n % 6 % 4 != 1) return (n | 1) == 3;
  ll t = __builtin_ctzll(n - 1), k = (n - 1) >> t;
  for (const ull &a : SPRP) {
    ull tmp = modpow(a, k, n);
    if (tmp <= 1 || tmp == n - 1) continue;
    for (int i = 0; i <= t; i++) {
      if (i == t) return false;
      tmp = modmul(tmp, tmp, n);
      if (tmp == n - 1) break;
    }
  }
  return true;
}
```

2.5 Sieve

2.5.1 杜教筛

令 $f(x)$ 为一个积性函数，求 $S(n)=\sum_{i=1}^{n} f(i)$。考虑引入另一个函数 $g(n)$，同时有 $h(n)=f(n)*g(n)=\sum_{d|n} g(d)f(\frac{n}{d})$。

$$
\begin{aligned}
\sum_{i=1}^{n} h(i) &= \sum_{i=1}^{n}\sum_{d|i} g(d)f(\frac{i}{d}) \\
&= \sum_{d=1}^{n} g(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor} f(i) \\
&= \sum_{d=1}^{n} g(d)S(\lfloor\frac{n}{d}\rfloor) \\
\Rightarrow g(1)S(n) &= \sum_{i=1}^{n} h(i) - \sum_{i=2}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)
\end{aligned}
$$

因此，引入的函数需要满足 $\sum h(i), \sum g(i)$ 都容易求得。

2.5.2 Powerful Number 筛

令 $f(x)$ 为一个积性函数，求 $S(n)=\sum_{i=1}^{n} f(i)$。考虑引入一个拟合函数 $g(x)$，满足 $g(p)=f(p)$，且 $g(x)$ 为 **积性函数、前缀和易求**。

令 $h=f*g^{-1}$，即有 $f(n)=h(n)*g(n)$。可知：

$$f(p)=g(1)h(p)+h(1)g(p) \Rightarrow h(p)=0$$

所求的前缀和为:

$$\begin{aligned}S(n) &= \sum_{i=1}^{n} f(i) \\ &= \sum_{i=1}^{n} \sum_{d|i} h(d) g(\frac{i}{d}) \\ &= \sum_{d=1}^{n} h(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} g(i)\end{aligned}$$

Powerful Number: 由于 $h(p)=0$，且 h 为积性函数，则仅当 n 满足下面的条件时，$h(n)$ 才有贡献。

$$n=\prod_{i=1}^{s} p_i^{t_i}, \forall i \in [1,s], t_i > 1$$

** 关于 PN 的数目 **，从莫比乌斯函数的角度考虑，应该为 $n-\sum_{i=1}^{n} \mu^2(i)$，但是这样并不能很好的计算值。这里用 PN 的一个性质，$n \in PN, \exists a, b, \text{s.t.} n=a^2 b^3$，则结果为 $\sum_{a=1}^{\sqrt{n}} \sqrt[3]{\frac{n}{a^2}}$，用积分可以简单求值为 $O(\sqrt{n})$。

2.5.3 Min25 筛

第三章　Math

3.1　Minimax 搜索 alpha-beta 剪枝

```
inline int doMin(int step, int alpha, int beta);
inline int doMax(int step, int alpha, int beta);
inline bool assess() {} // 如果输赢（平局）已定，返回结果
int doMax(int step, int alpha, int beta) {
  if (assess()) return res;
  for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) {
    if (g[i][j] == 0) {
      g[i][j] = playerX;
      int now = doMin(step + 1, alpha, beta);
      g[i][j] = 0;

      if (now > alpha) alpha = now;
      if (alpha >= beta) return alpha;
    }
  }
  return alpha;
}
int doMin(int step, int alpha, int beta) {
  if (assess()) return res;
  for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) {
    if (g[i][j] == 0) {
      g[i][j] = playerO;
      int now = doMax(step + 1, alpha, beta);
      g[i][j] = 0;

      if (now < beta) beta = now;
      if (alpha >= beta) return beta;
    }
  }
  return beta;
}
int battle(int step, int alpha, int beta) {
  if (step & 1) { // playerO 最小化得分
    return doMin(step, alpha, beta);
  } else { // playerX 最大化得分
    return doMax(step, alpha, beta);
  }
}
```

3.2 Combination

```cpp
template <typename T> struct Combination {
  int n; vector<T> facs, ifacs, invs;

  inline void extend() {
    int m = n << 1;
    facs.resize(m), ifacs.resize(m), invs.resize(m);
    for (int i = n; i < m; i++) facs[i] = facs[i - 1] * i;
    ifacs[m - 1] = T(1) / facs[m - 1];
    invs[m - 1] = facs[m - 2] * ifacs[m - 1];
    for (int i = m - 2; i >= n; i--) {
      ifacs[i]= ifacs[i + 1] * (i + 1);
      invs[i] = ifacs[i] * facs[i - 1];
    }
    n = m;
  }

  Combination(int MAX = 0)
    : n(1), facs(1, T(1)), ifacs(1, T(1)), invs(1, T(1)) {
    while (n <= MAX) extend();
  }

  T fac(int i) {
    assert(i >= 0);
    while (n <= i) extend();
    return facs[i];
  }
  T ifac(int i) {
    assert(i >= 0);
    while (n <= i) extend();
    return ifacs[i];
  }
  T inv(int i) {
    assert(i >= 0);
    while (n <= i) extend();
    return invs[i];
  }

  T C(int n, int r) {
    if (n < 0 || n < r || r < 0) return T(0);
    return fac(n) * ifac(r) * ifac(n - r);
  }
  T P(int n, int r) {
    if (n < 0 || n < r || r < 0) return T(0);
    return fac(n) * ifac(n - r);
  }
  T S2(int n, int k) {
    T res(0);
    for (int i = 0; i <= k; i++) {
      T t = T(k - i).pow(n) * C(k, i);
      (i & 1) ? res -= t : res += t;
    }
    return res * ifac(k);
  }
```

```
  T B(int n, int k) { // sum_{i=0}^{k} S2(n, i)
    static vector<T> sum = { 1 };
    for (static int i = 1; i <= k; i++) {
      sum.push_back(sum.back() + (i & 1 ? -ifac(i) : ifac(i)));
    }
    T res(0);
    for (int i = 1; i <= k; i++) {
      res += sum[k - i] * T(i).pow(n) * ifac(i);
    }
    return res;
  }
};
Combination<mint> comb;
```

3.3 Stirling Number Query

```
struct StirlingNumber {
  const int P; // P is a small prime
  vector<vector<int>> C, S1, S2;

  StirlingNumber(int _P = 2) : P(_P) {
    C.resize(P, vector<int>(P, 0)), S1 = S2 = C;
    for (int i = 0; i < P; i++) {
      C[i][0] = 1, C[i][i] = 1;
      for (int j = 1; j < i; j++) {
        C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % P;
      }
    }
    for (int i = 0; i < P; i++) {
      S2[i][0] = 0, S2[i][i] = 1;
      for (int j = 1; j < i; j++) {
        S2[i][j] = (S2[i - 1][j] * j + S2[i - 1][j - 1]) % P;
      }
    }
    for (int i = 0; i < P; i++) {
      S1[i][0] = 0, S1[i][i] = 1;
      for (int j = 1; j < i; j++) {
        S1[i][j] = (S1[i - 1][j] * (P - i + 1) + S1[i - 1][j -
            1]) % P;
      }
    }
  }

  int getC(ll n, ll k) const {
    if (k < 0 || k > n) return 0;
    int res = 1;
    for (; n; n /= P, k /= P) {
      res = res * C[n % P][k % P] % P;
    }
    return res;
  }
  int getS1(ll n, ll k) const {
```

```
    if (k < 0 || k > n) return 0;
    if (n == 0) return 1;
    ll n1 = n / P, n0 = n % P;
    if (k < n1) return 0;
    ll i = (k - n1) / (P - 1), j = (k - n1) % (P - 1);
    if (j == 0 && n0 == P - 1) j = P - 1, i -= 1;
    if (i < 0 || i > n1 || j > n0) return 0;
    int res = S1[n0][j] * getC(n1, i) % P;
    if ((n1 ^ i) & 1) res = (P - res) % P;
    return res;
  }
  int getS2(ll n, ll k) const {
    if (k < 0 || k > n) return 0;
    if (n == 0) return 1;
    ll k1 = k / P, k0 = k % P;
    if (n < k1) return 0;
    ll i = (n - k1) / (P - 1), j = (n - k1) % (P - 1);
    if (j == 0) j = P - 1, i -= 1;
    if (j == P - 1 && k0 == 0) return getC(i, k1 - 1);
    else return getC(i, k1) * S2[j][k0] % P;
  }
};
```

3.4 一些数学结论

3.4.1 错排计数

$$D[n] = (n-1) \times (D[n-1] + D[n-2]), D[0] = 1, D[1] = 0$$

对于第 n 个要加入错排的数，

- 它可以和已经错排的 n-1 个数的其中一个进行交换，构成错排，于是有 $(n-1) \times D[n-1]$。
- 还可以与 n-1 个数中唯一一个没有加入错排的数（n-2 个构成错排，剩下一个待定）进行交换，这样就构成了 n 错排，而这 n-1 个数之中，每个数都可以作为那个唯一一个没有加入错排的数，故 $(n-1) \times D[n-2]$。

3.4.2 Pick 定理

$$S = a + \frac{b}{2} - 1$$

其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，S 表示多边形的面积。

3.4.3 约瑟夫环

一共有 n 个人，每数到 k 的人离开，求第 m 个离开的人的编号 (1-indexed)。

```
ll Josephus(ll n, ll m, ll k) {
  if (k == 1) return m;
  ll index = (k - 1) % (n - m + 1);
  for (ll cur = n - m + 2; cur <= n; cur++) {
    index = (index + k) % cur;
    ll tmp = min(n - cur, (cur - index - 1) / k) - 1;
    if (tmp > 0) cur += tmp, index += tmp * k;
  }
  return index + 1;
}
```

第四章 Data Structure

4.1 Linear Basis

```
// 实数线性基

template<int mod = 998244353>
struct Basis {
  vector<vector<int>> B;

  Basis(int n) { B.resize(n, vector<int>(n)); }
  int inv(int x) {
    return x == 1 ? 1 : (mod - mod / x) * 1LL * inv(mod % x) %
        mod;
  }
  bool insert(vector<int> v) {
    for (int i = v.size() - 1; i >= 0; i--) {
      if (v[i] == 0) continue;
      if (B[i][i] == 0) {
        B[i] = v;
        return true;
      }
      int t = v[i] * 1LL * inv(B[i][i]) % mod;
      for (int j = i; j >= 0; j--) {
        v[j] = (v[j] - B[i][j] * 1LL * t % mod + mod) % mod;
      }
    }
    return false;
  }
};

// 异或线性基

template<typename T> struct Basis {
  vector<T> B;
  int sz = 0;
  bool free = false;

  Basis(int n) : B(n) {};
  void insert(T v) {
    for (int i = B.size() - 1; i >= 0; i--) {
      if ((v >> i & 1) == 0) continue;
      if (B[i]) v ^= B[i];
      else {
```

```
        for (int j = i - 1; j >= 0; j--)
          if (v >> j & 1) v ^= B[j];
        for (int j = i + 1; j < (int) B.size(); j++)
          if (B[j] >> i & 1) B[j] ^= v;
        B[i] = v, sz += 1;
        return;
      }
    }
    free = true;
  }
  T max() {
    T res = 0;
    for (int i = 0; i < (int) B.size(); i++) res ^= B[i];
    return res;
  }
  T kth(long long k) { // k start from 1-indexed
    if (free) k -= 1; // free -> 0
    if (k >= (1LL << sz)) return -1;
    T res = 0;
    for (int i = 0, t = 0; i < (int) B.size(); i++) {
      if (B[i] == 0) continue;
      if (k >> t & 1) res ^= B[i];
      t += 1;
    }
    return res;
  }
  long long rank(T v) { // rank start from 0-indexed
    long long res = 0;
    for (int i = 0, t = 0; i < (int) B.size(); i++) {
      if (B[i] == 0) continue;
      if (v >> i & 1) res |= 1LL << t;
      t += 1;
    }
    return res;
  }
  int size() { return sz; }
};
```

4.2 Chtholly Tree

```
struct ODT {
  map<int, int> mp;
  ODT(int _ = 0, int unit = 0) { mp[_ - 1] = unit; }
  // initialize [_, +infty)
  int get(int x) { return prev(mp.upper_bound(x))->second; }
  void split(int x) { mp[x] = get(x); }
  void assign(int l, int r, int v) {  // assign [l, r), value v
    split(l), split(r);
    auto it = mp.find(l);
    while (it->first != r) it = mp.erase(it);
    mp[l] = v;
  }
};
```

4.3 Disjoint Set Union

```
struct DSU {
  vector<int> p;

  DSU() = default;
  DSU(int n) : p(n, -1) {}
  ~DSU() = default;

  int find(int x) { return p[x] < 0 ? x : p[x] = find(p[x]); }
  int size(int x) { return -p[find(x)]; }
  bool same(int u, int v) { return find(u) == find(v); }
  bool merge(int u, int v) {
    u = find(u), v = find(v);
    if (u == v) return false;

    p[u] += p[v], p[v] = u;
    return true;
  }
};
```

4.3.1 Rollback-DSU

```
struct Rollback_DSU {
  vector<int> fa, sz;
  vector<pair<int&, int>> his_fa, his_sz;

  Rollback_DSU() = default;
  Rollback_DSU(int n) : fa(n), sz(n) { init(); }
  ~Rollback_DSU() = default;

  void init() {
    fill(all(sz), 1), iota(all(fa), 0);
    his_sz.clear(), his_fa.clear();
  }
  int find(int x) {
    while (x != fa[x]) x = fa[x];
    return x;
  }
  bool same(int u, int v) {
    return find(u) == find(v);
  }
  bool merge(int u, int v) {
    u = find(u), v = find(v);
    if (u == v) return false;
    if (sz[u] < sz[v]) swap(u, v);
    his_sz.emplace_back(sz[u], sz[u]);
    his_fa.emplace_back(fa[v], fa[v]);
```

```
    sz[u] += sz[v], fa[v] = u;
    return true;
  }
  int history() { return his_fa.size(); }
  void rollback(int h) {
    while ((int) his_fa.size() > h) {
      his_sz.back().first = his_sz.back().second;
      his_fa.back().first = his_fa.back().second;
      his_fa.pop_back(), his_sz.pop_back();
    }
  }
};
```

4.3.2 Weighted-DSU

```
struct Weighted_DSU {
  vector<int> p, val;

  Weighted_DSU() = default;
  Weighted_DSU(int n) : p(n), val(n) { iota(all(p), 0); }
  ~Weighted_DSU() = default;

  int find(int x) {
    if (x == p[x]) return x;
    int fa = find(p[x]);
    val[x] += val[p[x]];
    return p[x] = fa;
  }
  bool same(int x, int y) { return find(x) == find(y); }
  int weight(int x) {
    find(x);
    return val[x];
  }
  int diff(int x, int y) {  // query (a[y] - a[x])
    return weight(y) - weight(x);
  }
  bool merge(int x, int y, int dif = 0) {
    // add constraint (a[y] - a[x] = dif)
    dif = weight(y) - weight(x) - dif;
    x = find(x), y = find(y);
    if (x == y) return false;
    p[x] = y, val[x] = dif;
    return true;
  }
};
```

4.4 Fenwick Tree

```
struct FenwickTree {
  vector<ll> s;
```

```
  FenwickTree(int n) : s(n) {}
  void update(int pos, ll dif) {
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos - 1];
    return res;
  }
};

struct FenwickTree {
  vector<ll> s, d;

  FenwickTree() = default;
  FenwickTree(int n) : s(n), d(n) {}

  void add(int p, ll dif) {
    ll val = (p + 1) * dif;
    for (; p < s.size(); p |= p + 1) {
      s[p] += val, d[p] += dif;
    }
  }
  ll query(int i) {
    ll S = 0LL, D = 0LL;
    for (int p = i; p > 0; p &= p - 1) {
      S += s[p - 1], D += d[p - 1];
    }
    return D * (i + 1) - S;
  }
  void range_add(int l, int r, ll val) {  // [l, r)
    add(l, val), add(r, -val);
  }
  ll range_query(int l, int r) {  // [l, r)
    return query(r) - query(l);
  }
};
```

4.5 Sparse Table

```
template <typename T> struct ST {
  vector<vector<T>> st;
  T f(const T &a, const T &b) {
    return min<T>(a, b);
  }

  ST() = default;
  ST(const vector<T> &v) : st(1, v) {
    for (int pw = 1, k = 1; (pw << 1) <= sz(v); pw <<= 1, k++) {
      st.emplace_back(sz(v) - (pw << 1) + 1);
      for (int i = 0; i < sz(st[k]); i++) {
        st[k][i] = f(st[k - 1][i], st[k - 1][i + pw]);
      }
```

```
    }
  }
  ~ST() = default;

  T query(int l, int r) {  // query [l, r)
    int dep = 31 - __builtin_clz(r - l);
    return f(st[dep][l], st[dep][r - (1 << dep)]);
  }
};
```

4.6 Hash Map

```
using ull = unsigned long long;

#include <bits/extc++.h>
const int SEED = chrono::steady_clock::now().time_since_epoch().
    count();
struct chash {  // To use most bits rather than just the lowest
    ones:
  const ull C = ll(4e18 * acos(0)) | 71;  // large odd number
  ll operator()(ll x) const { return __builtin_bswap64((x ^ SEED)
      * C); }
};
using HashMap = __gnu_pbds::gp_hash_table<ll, int, chash>;

ull splitmix64(ull x) {
  // http://xorshift.di.unimi.it/splitmix64.c
  x += 0x9e3779b97f4a7c15;
  x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
  x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
  return x ^ (x >> 31);
}
```

4.7 Ordered Set

```
// tr.insert(val);      //插入元素
// tr.erase(iterator); //删除元素
// tr.order_of_key();  //求k在树中是第几大
// tr.find_by_order(); //求树中的第k大
// tr.lower_bound();   //求前驱
// tr.upper_bound();   //求后继

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <typename T>
struct ordered_set {
```

```
  tree<T, null_type, less<T>, rb_tree_tag,
     tree_order_statistics_node_update> s;

  ordered_set() = default;
  ordered_set(const vector<T> &v) : s(all(v)) {}
  ~ordered_set() = default;

  void insert(const T &x) { s.insert(x); }
  void erase(const T &x) { s.erase(x); }
  int rank(const T &x) { return s.order_of_key(x); }
  T nth_element(const int &k) { return *s.find_by_order(k); }
};
```

4.8 Segment Tree

```
template <typename T, T (*ut)(), T (*f)(T, T)>
struct SegTree {
 private:
  int n, _log;
  vector<T> val;
  void _update(int t) { val[t] = f(val[t << 1 | 0], val[t << 1 |
     1]); }

 public:
  SegTree() = default;
  SegTree(const vector<T> &v) {
    n = 1, _log = 0;
    while (n < (int)v.size()) n <<= 1, _log++;
    val.resize(n << 1, ut());
    for (int i = 0; i < (int)v.size(); i++) val[i + n] = v[i];
    for (int i = n - 1; i > 0; i--) _update(i);
  }
  ~SegTree() = default;

  void set(int p, const T &dif) {
    val[p += n] = dif;
    for (int i = 1; i <= _log; i++) _update(p >> i);
  }
  T get(int p) { return val[p + n]; }
  T query(int l, int r) {
    if (l == r) return ut();
    T L = ut(), R = ut();
    for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) L = f(L, val[l++]);
      if (r & 1) R = f(val[--r], R);
    }
    return f(L, R);
  }
};

namespace SegTreeUtil {  // Utilization
template <typename T>
T Merge(T a, T b) {}
```

```
template <typename T>
T Init() {
  return T();
}
template <typename T>
struct Tree : SegTree<T, Init<T>, Merge<T>> {
  using base = SegTree<T, Init<T>, Merge<T>>;
  Tree(const vector<T> &v) : base(v) {}
};
}  // namespace SegTreeUtil
using SegTreeUtil::Tree;
```

4.9 KD Tree

```
const int N = (int) 5e5 + 10;
int n, root, cmpk;

struct node {
  int pos[2];
  int son[2];
  int min[2];
  int max[2];
  int id;
  bool operator < (const node &rhs) const {
    return pos[cmpk] < rhs.pos[cmpk];
  }
} KDT[N];

void update(int nd) {
  for (int k = 0; k < 2; k++) {
    KDT[nd].min[k] = KDT[nd].max[k] = KDT[nd].pos[k];
    for (int i = 0; i < 2; i++) if (KDT[nd].son[i]) {
      int s = KDT[nd].son[i];
      KDT[nd].min[k] = min(KDT[nd].min[k], KDT[s].min[k]);
      KDT[nd].max[k] = max(KDT[nd].max[k], KDT[s].max[k]);
    }
  }
}

int build(int l, int r, int k) {
  int m = (l + r) >> 1; cmpk = k;
  nth_element(KDT + l, KDT + m, KDT + r + 1);
  KDT[m].son[0] = KDT[m].son[1] = 0;
  if (l != m) KDT[m].son[0] = build(l, m - 1, k ^ 1);
  if (r != m) KDT[m].son[1] = build(m + 1, r, k ^ 1);
  return update(m), m;
}

void KDT_build() {
  for (int i = 1; i <= n; i++) {
    cin >> KDT[i].pos[0] >> KDT[i].pos[1];
    KDT[i].id = i;
```

```
  }
  root = build(1, n, 0);
}

vector<int> ans;
int st[2], ed[2];

bool judge(const node &nd) {
  return nd.pos[0] >= st[0]
      && nd.pos[0] <= ed[0]
      && nd.pos[1] >= st[1]
      && nd.pos[1] <= ed[1];
}

bool check(const node &nd) {
  return nd.min[0] <= ed[0]
      && nd.max[0] >= st[0]
      && nd.min[1] <= ed[1]
      && nd.max[1] >= st[1];
}

void query(int nd) {
  if (judge(KDT[nd])) ans.push_back(KDT[nd].id);
  for (int i = 0; i < 2; i++) if (KDT[nd].son[i]) {
    int s = KDT[nd].son[i];
    if (check(KDT[s])) query(s);
  }
}

void SingleTest(int TestCase) {
  cin >> n; KDT_build();
  int q; cin >> q;
  while (q--) { // 找到矩形区域内的所有点
    cin >> st[0] >> ed[0];
    cin >> st[1] >> ed[1];
    ans.clear(), query(root), sort(all(ans));
    for (int id : ans) cout << id - 1 << '\n';
  }
}
```

第五章 String

5.1 Suffix Array

5.1.1 SA-doubling

在 $\mathcal{O}(n \log n)$ 下求出 SA 数组。

```cpp
struct SuffixArray_doubling {
  vector<int> sa, lcp;
  SuffixArray_doubling(const string &s, int lim = 256) {
    int n = (int) s.length() + 1;
    vector<int> x(all(s) + 1), y(n), ws(max(n, lim));
    sa.resize(n), iota(all(sa), 0);
    for (int d = 0, p = 0; p < n; d = max(1, d << 1), lim = p) {
      p = d, iota(all(y), n - d);
      for (int i = 0; i < n; i++) if (sa[i] >= d) y[p++] = sa[i]
         - d;
      fill(all(ws), 0);
      for (int i = 0; i < n; i++) ws[x[i]]++;
      for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
      for (int i = n - 1; i >= 0; i--) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      for (int i = 1; i < n; i++) {
        int a = sa[i - 1], b = sa[i];
        x[b] = (y[a] == y[b] && y[a + d] == y[b + d]) ? p - 1 : p
          ++;
      }
    }
    vector<int> rk(n); // sa[0] = s.length()
    for (int i = 1; i < n; i++) rk[sa[i]] = i;
    lcp.resize(n); // longest common prefixes
    // lcp[i] = LCP(sa[i], sa[i - 1])
    for (int i = 0, d = 0, p = 0; i < (int) s.length(); i++) {
      p -= p > 0, d = sa[rk[i] - 1];
      while (s[i + p] == s[d + p]) p++;
      lcp[rk[i]] = p;
    }
    sa.erase(begin(sa)), lcp.erase(begin(lcp)), lcp[0] = s.length
      ();
  }
};
```

5.1.2 SA-IS

使用诱导排序，在 $\mathcal{O}(n)$ 时间复杂度下求出 SA 数组。

```
struct SuffixArray {
  vector<int> SA, LCP;

  SuffixArray(const string &s, char first = 'a', char last = 'z')
      {
    get_sa(s, first, last), get_lcp(s);
  }

  vector<int> SA_IS(const vector<int> &v, int K) {
    const int n = sz(v);
    vector<int> SA(n), lms;
    vector<bool> sl(n, false);
    for (int i = n - 2; i >= 0; i--) {
      sl[i] = (v[i] == v[i + 1] ? sl[i + 1] : v[i] > v[i + 1]);
      if (sl[i] && !sl[i + 1]) lms.push_back(i + 1);
    }
    reverse(all(lms));
    const auto induced_sort = [&](const vector<int> &LMS) {
      vector<int> l(K, 0), r(K, 0);
      for (const int &x : v) {
        if (x + 1 < K) l[x + 1]++;
        ++r[x];
      }
      partial_sum(all(l), begin(l));
      partial_sum(all(r), begin(r));
      fill(all(SA), -1);
      for_each(rall(LMS), [&](const int &p) { SA[--r[v[p]]] = p;
          });
      for (const int &p : SA) if (p >= 1 && sl[p - 1]) {
        SA[l[v[p - 1]]++] = p - 1;
      }
      fill(all(r), 0);
      for (const int &x : v) ++r[x];
      partial_sum(all(r), begin(r));
      for_each(rall(SA) - 1, [&](const int &p) {
        if (p >= 1 && !sl[p - 1]) SA[--r[v[p - 1]]] = p - 1;
      });
    };
    induced_sort(lms);
    vector<int> new_lms(sz(lms)), new_v(sz(lms));
    for (int i = 0, k = 0; i < n; i++) {
      if (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1]) {
        new_lms[k++] = SA[i];
      }
    }
    int cur = SA.back() = 0;
    for (int k = 1; k < sz(new_lms); k++) {
      int i = new_lms[k - 1], j = new_lms[k];
      if (v[i] != v[j]) { SA[j] = ++cur; continue; }
      bool flag = false;
      for (int a = i + 1, b = j + 1;; a++, b++) {
        if (v[a] != v[b]) { flag = true; break; }
```

```
        if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
          flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b - 1])
              );
          break;
        }
      }
      SA[j] = (flag ? ++cur : cur);
    }
    for (int i = 0; i < sz(lms); i++) new_v[i] = SA[lms[i]];
    if (cur + 1 < sz(lms)) {
      auto lms_SA = SA_IS(new_v, cur + 1);
      for (int i = 0; i < sz(lms); i++) new_lms[i] = lms[lms_SA[i
          ]];
    }
    return induced_sort(new_lms), SA;
  }

  void get_sa(const string &s, char first = 'a', char last = 'z')
      {
    vector<int> v(sz(s) + 1);
    copy(all(s), begin(v));
    for (auto &&x : v) x -= first - 1;
    v.back() = 0;
    this->SA = SA_IS(v, last - first + 2);
    this->SA.erase(begin(this->SA));
  }

  void get_lcp(const string &s) {
    int n = sz(s);
    vector<int> rank(n), lcp(n);
    for (int i = 0; i < n; i++) rank[SA[i]] = i;
    for (int i = 0, p = 0; i < n; i++, p ? p--: 0) {
      if (rank[i] == 0) { p = 0; continue; }
      int j = SA[rank[i] - 1];
      while (i + p < n && j + p < n && s[i + p] == s[j + p]) p++;
      lcp[rank[i]] = p;
    }
    this->LCP = move(lcp);
  }
};
```

5.2 KMP / KMP-automaton

```
// 前缀函数 lps[i] -> 前缀串s[0..i]的最长匹配的真前后缀
// lps[i] = max{k : s[0..k-1]==s[i-k+1..i]}

vector<int> KMP(const string &s, const string &pat) {
  string t = pat + '\0' + s;
  vector<int> lps(sz(t), 0);
  for (int i = 1; i < sz(t); i++) {
    int g = lps[i - 1];
    while (g && t[i] != t[g]) g = lps[g - 1];
```

```
    lps[i] = g + (t[i] == t[g]);
  }
  vector<int> match;
  for (int i = sz(t) - sz(s); i < sz(t); i++) {
    if (lps[i] == sz(pat)) {
      match.push_back(i - 2 * sz(pat));
    }
  }
  return match;
}

// 根据前缀函数建自动机
// aut[l][c] -> 前缀l后面添加字符c后 的前缀函数

void LPS_automaton(const string &s) {
  vector<int> lps(sz(s), 0);
  vector<array<int, 26>> aut(sz(s));
  aut[0][s[0] - 'a'] = 1;
  for (int i = 1; i < sz(s); i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i]) {
        aut[i][c] = aut[lps[i - 1]][c];
      } else {
        aut[i][c] = i + ('a' + c == s[i]);
      }
    }
    lps[i] = aut[lps[i - 1]][s[i] - 'a'];
  }
};
```

5.3 Z Algorithm

```
// z[i] = LCP(s, s.substr(i))
vector<int> Z_algorithm(const string &s) {
  int n = (int) s.length();
  vector<int> z(n); z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n; i++) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i] += 1;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

第六章 Graph

6.1 Low Link

```
class LowLink {
 private:
  vector<int> articulation_points;
  vector<pair<int, int>> bridges;

  int n, m, idx;
  vector<vector<int>> G;
  vector<int> dfn, low;

  void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++idx;
    int cnt = 0, ok = 0;
    for (int v : G[u]) {
      if (dfn[v] == 0) {
        cnt++, tarjan(v, u);
        low[u] = min(low[u], low[v]);
        if (fa != -1 && dfn[u] <= low[v]) ok = 1;
        if (dfn[u] < low[v]) {
          bridges.push_back(minmax(v, u));
        }
      } else if (dfn[v] < dfn[u] && v != fa) {
        low[u] = min(low[u], dfn[v]);
      }
    }
    if (ok || (fa == -1 && cnt > 1)) {
      articulation_points.push_back(u);
    }
  }

 public:
  LowLink(int _n, int _m) : n(_n), m(_m), G(n) {
    for (int u, v, _ = 0; _ < m; _++) {
      cin >> u >> v;
      G[u].emplace_back(v);
      G[v].emplace_back(u);
    }
  }
  pair<vector<int>, vector<pair<int, int>>> low_link() {
    dfn.assign(n, 0);
    low.assign(n, 0);
```

```
    articulation_points.clear();
    bridges.clear();
    idx = 0;  // init
    for (int i = 0; i < n; i++) {
      if (dfn[i] == 0) tarjan(i, -1);
    }
    return {articulation_points, bridges};
  }
};
```

6.2 SCC

```
struct SCC {
  vector<int> scc;
  int n, m;
  vector<vector<int>> G, rG;

  SCC(int _n, int _m) : n(_n), m(_m), G(n), rG(n) {
    for (int u, v, _ = 0; _ < m; _++) {
      cin >> u >> v; --u, --v;
      G[u].emplace_back(v);
      rG[v].emplace_back(u);
    }
  }
  pair<int, vector<int>> kosaraju() {
    vector<int> seq;
    vector<bool> vis(n, false);

    function<void(int)> dfs = [&](int u) {
      vis[u] = true;
      for (const int &v : G[u]) {
        if (vis[v] == false) dfs(v);
      }
      seq.emplace_back(u);
    };
    for (int i = 0; i < n; i++) if (!vis[i]) dfs(i);

    vis.assign(n, false);
    int scc_num = 0;
    function<void(int)> rdfs = [&](int u) {
      vis[u] = 1, scc[u] = scc_num;
      for (int v : rG[u]) {
        if (vis[v] == false) rdfs(v);
      }
    };
    for (int i = n - 1; i >= 0; i--) {
      if (!vis[seq[i]]) rdfs(seq[i]), ++scc_num;
    }
    return {scc_num, scc};
  }
};
```

6.3 Manhattan MST

```
#include "../data_structure/disjoint_set_union.hpp"

tuple<ll, vector<pair<int, int>>> ManMST(vector<int> xs, vector<
    int> ys) {
  vector<int> id(xs.size());
  iota(all(id), 0);
  vector<tuple<ll, int, int>> edges;
  for (int s = 0; s < 2; s++) {
    for (int t = 0; t < 2; t++) {
      sort(all(id),
           [&](int i, int j) { return xs[i] + ys[i] < xs[j] + ys[
               j]; });
      map<int, int> sweep;
      for (int i : id) {
        for (auto it = sweep.lower_bound(-ys[i]); it != sweep.end
            ();
             it = sweep.erase(it)) {
          int j = it->second;
          if (xs[i] - xs[j] < ys[i] - ys[j]) break;
          int w = abs(xs[i] - xs[j]) + abs(ys[i] - ys[j]);
          edges.emplace_back(w, i, j);
        }
        sweep[-ys[i]] = i;
      }
      swap(xs, ys);
    }
    for (auto &x : xs) x = -x;
  }
  ll mst = 0;
  DSU dsu = DSU(xs.size());
  vector<pair<int, int>> P;
  sort(all(edges));
  for (auto [w, u, v] : edges) {
    if (dsu.merge(u, v)) {
      mst += w, P.emplace_back(u, v);
    }
  }
  return {mst, P};
}
```

6.4 Dinic

```
const int maxn = 2e4 + 10;
const int inf = 0x3f3f3f3f;
template <typename T> struct Dinic_Graph {
  int n, s, t;
  struct edge {
    int from, to, rev;
    T c;
    edge(int _from, int _to, int _rev, T _c) {
```

```
      from = _from, to = _to, rev = _rev, c = _c;
    }
  };
  vector<edge> G[maxn];
  queue<int> Q;
  T d[maxn];

  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) G[i].clear();
    while (!Q.empty()) Q.pop();
  }

  inline void ins(int u, int v, T c) {
    debug(u, v, c);
    G[u].emplace_back(u, v, 0, c);
    G[v].emplace_back(v, u, 0, 0);
    G[u].back().rev = int(G[v].size()) - 1;
    G[v].back().rev = int(G[u].size()) - 1;
  }

  inline bool bfs() {
    for (int i = 0; i <= n; i++) d[i] = inf;
    Q.push(s);
    d[s] = 0;
    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      for (edge& e : G[v]) {
        if (e.c > 0 && d[e.to] > d[v] + 1) {
          d[e.to] = d[v] + 1;
          Q.push(e.to);
        }
      }
    }
    return d[t] != inf;
  }

  inline T dfs(int v, T flow) {
    if (v == t || flow == 0) return flow;
    T res = 0, cur = 0;
    for (edge &e : G[v]) {
      if (e.c > 0 && d[e.to] == d[v] + 1) {
        if ((cur = dfs(e.to, min(flow, e.c))) > 0) {
          res += cur;
          flow -= cur;
          e.c -= cur;
          G[e.to][e.rev].c += cur;
        }
      }
    }
    if (res == 0) d[v] = -1;
    return res;
  }

  T Dinic(int _s, int _t) {
```

```
    s = _s, t = _t;
    T res = 0, cur = 0;
    while (bfs()) {
      while ((cur = dfs(s, inf)) > 0) {
        res += cur;
      }
    }
    return res;
  }
};
Dinic_Graph<int> G;
```

第七章　Dynamic Programing

7.1　Incremental Convex Hull Trick

凸包优化技巧

```
// https://codeforces.com/contest/1715/problem/E
// https://loj.ac/p/2035
// https://ac.nowcoder.com/acm/problem/20352

struct Line {
  mutable ll k, b, x;
  bool operator < (const Line &rhs) const { return k < rhs.k; }
  bool operator < (const ll &rhs) const { return x < rhs; }
};
// Incremental Convex Hull Trick
struct CHT : multiset<Line, less<>> {
  static const ll inf = numeric_limits<ll>::max();
  // for doubles, use inf = 1/.0, div(a, b) = a / b
  ll div(ll a, ll b) { return a / b - (ll)((a ^ b) < 0 && a % b);
      }
  bool isect(iterator A, iterator B) { // judge intersect
    if (B == end()) return A->x = inf, false;
    if (A->k == B->k) A->x = A->b > B->b ? inf : -inf;
    else A->x = div(B->b - A->b, A->k - B->k);
    return A->x >= B->x;
  }
  void add(ll k, ll b) {
    iterator C = insert({k, b, 0}), A, B = A = C++;
    while (isect(B, C)) C = erase(C);
    if (A != begin() && isect(--A, B)) isect(A, B = erase(B));
    while ((B = A) != begin() && (--A)->x >= B->x) isect(A, erase
       (B));
  }
  ll get(ll x) const { // get max kx+b
    assert(!empty()); // debug
    Line l = *lower_bound(x);
    return l.k * x + l.b;
  }
};
```

第八章　Polynomial

8.1　FFT-mod

```
using C = complex<double>;
void FFT(vector<C> &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);
  for (static int k = 2; k < n; k <<= 1) {
    R.resize(n), rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    for (int i = k; i < 2 * k; i++) {
      rt[i] = R[i] = i & 1 ? R[i >> 1] * x : R[i >> 1];
    }
  }
  vector<int> rev(n);
  for (int i = 0; i < n; i++) {
    rev[i] = rev[i >> 1] >> 1 | (i & 1) << (L - 1);
  }
  for (int i = 0; i < n; i++) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += k * 2) {
      for (int j = 0; j < k; j++) {
        C z = rt[j + k] * a[i + j + k];
        a[i + j + k] = a[i + j] - z, a[i + j] += z;
      }
    }
  }
}

using Poly = vector<ll>;
template <int M>
Poly convMod(const Poly &a, const Poly &b) {
  if (a.empty() || b.empty()) return {};
  Poly res(sz(a) + sz(b) - 1);
  int B = 32 - __builtin_clz(sz(res));
  int n = 1 << B, S = sqrt(M);
  vector<C> L(n), R(n), outs(n), outl(n);
  for (int i = 0; i < sz(a); i++) {
    L[i] = C(int(a[i] / S), int(a[i] % S));
  }
```

```
  for (int i = 0; i < sz(b); i++) {
    R[i] = C(int(b[i] / S), int(b[i] % S));
  }
  FFT(L), FFT(R);
  for (int i = 0; i < n; i++) {
    int j = (n - i) & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  FFT(outl), FFT(outs);
  for (int i = 0; i < sz(res); i++) {
    ll A = ll(imag(outs[i]) + 0.5) % M;
    ll B = (ll(imag(outl[i]) + 0.5) + ll(real(outs[i]) + 0.5)) %
        M * S % M;
    ll C = ll(real(outl[i]) + 0.5) % M * (S * S % M) % M;
    res[i] = (A + B + C) % M;
  }
  return res;
}
```

8.2 Lagrange

```
ll lagrange(ll N) { // (x, y)
  ll res = 0LL;
  for (int i = 1; i <= k + 2; i++) {
    ll tmp = y[i];
    for (int j = 1; j <= k + 2; j++) {
      if (i == j) continue;
      tmp = (N - x[j]) % mod * inv(x[i] - x[j]) % mod * tmp % mod
          ;
    }
    res = (res + mod + tmp) % mod;
  }
  return res;
}
```

8.2.1 横坐标是连续整数的拉格朗日插值

给出 n, k ，求 $\sum_{i=1}^{n} i^k$ 对 $10^9 + 7$ 取模的值。

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

using ll = long long;
#define all(a) begin(a), end(a)

const int mod = 1e9 + 7;

ll qpow(ll x, ll n, ll mod) {
  ll res = 1LL;
  for (x %= mod; n; n >>= 1, x = x * x % mod) {
```

```
    if (n & 1LL) res = res * x % mod;
  }
  return (res + mod) % mod;
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  int n, k; cin >> n >> k;

  vector<ll> y(k + 3, 0);
  for (int i = 1; i <= k + 2; i++) {
    y[i] = (y[i - 1] + qpow(i, k, mod)) % mod;
  }
  if (n <= k + 2) return cout << y[n] << '\n', 0;

  vector<ll> inv(k + 3, 1);
  for (int i = 2; i <= k + 2; i++) {
    inv[i] = inv[mod % i] * (mod - mod / i) % mod;
  }
  for (int i = 2; i <= k + 2; i++) {
    inv[i] = inv[i - 1] * inv[i] % mod;
  }

  vector<ll> pre(k + 4, 1), suf(k + 4, 1);
  for (int i = 1; i <= k + 2; i++) pre[i] = pre[i - 1] * (n - i)
     % mod;
  for (int i = k + 2; i >= 1; i--) suf[i] = suf[i + 1] * (n - i)
     % mod;

  ll ans = 0;
  for (int i = 1; i <= k + 2; i++) {
    ll P = inv[i - 1] * inv[k + 2 - i] % mod * y[i] % mod;
    ll Q = pre[i - 1] * suf[i + 1] % mod;
    if ((k + 2 - i) & 1) Q = mod - Q;
    ans = (ans + P * Q % mod) % mod;
  }
  cout << ans << '\n';
  return 0;
}
```