Algorithms for XCPC

Sshwy

2022 年 10 月 8 日

目录

1 代码头

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long lld;
typedef long double lf;
typedef unsigned long long uld;
typedef pair<int, int> pii;
#define fi first
#define se second
#define pb push_back
#define mk make_pair
#define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
#define ROF(i, a, b) for (int i = (a); i >= (b); --i)
namespace RA {
  int r(int p) { return 1ll * rand() * rand() % p; }
  int r(int L, int R) { return r(R - L + 1) + L; }
} // namespace RA
```

code/head.cpp

2 字符串

2.1 KMP 算法

```cpp
const int N = 1000;
int n, nex[N];
void getNext(char *s, int l) {
  nex[0] = -1;
  int i = 0, j = -1;
  while (i < l) {
    if (j == -1 || s[i] == s[j]) i++, j++, nex[i] = j;
    else j = nex[j];
  }
}
int kmp(char *s, char *t) {
  int ls = strlen(s), lt = strlen(t);
  getNext(s, ls);
  int i = 0, j = 0;
  while (i < ls && j < lt) {
    if (i == -1 || s[i] == t[j]) i++, j++;
    else i = nex[i];
  }
  if (i == ls) return j - i;
  else return -1;
}
```

code/kmp.cpp

2.2 Manacher 算法

```cpp
const int N = 11000100;
char s[N];
int ls, L, R, d[N], ans;
int main() {
  scanf("%s", s + 1);
  ls = strlen(s + 1);
  FOR(i, 1, ls) {
    if (i < R) d[i] = min(R - i, d[L + R - i]);
    while (
        0 < i - d[i] - 1 && i + d[i] + 1 <= ls && s[i - d[i] - 1] == s[i + d[i] + 1])
      ++d[i];
    if (i + d[i] > R) L = i - d[i], R = i + d[i];
    assert(L > 0);
    ans = max(ans, d[i] * 2 + 1);
  }
  L = R = 0;
  FOR(i, 1, ls) d[i] = 0;
  FOR(i, 1, ls - 1) {
    if (i <= R) d[i] = min(R - i, d[L + R - i - 1]);
    while (0 < i - d[i] && i + d[i] + 1 <= ls && s[i - d[i]] == s[i + d[i] + 1])
      ++d[i];
    if (i + d[i] > R) L = i - d[i] + 1, R = i + d[i];
    ans = max(ans, d[i] * 2);
  }
  printf("%d", ans);
  return 0;
}
```

code/manacher.cpp

2.3 后缀数组

```cpp
struct SA {
  char s[N];
  int l, sz, sa[N], rk[N];
  int t[N], bin[N], h[N], he[N]; // h,height
  void qsort() {
    for (int i = 0; i <= sz; i++) bin[i] = 0;
    FOR(i, 1, l) bin[rk[i]]++;
    FOR(i, 1, sz) bin[i] += bin[i - 1];
    ROF(i, l, 1) sa[bin[rk[t[i]]]--] = t[i];
  }
  void make() { // 记得先把 s 赋值 (1 起点)
    l = strlen(s + 1), sz = max(l, 127);
    for (int i = 1; i <= l; i++) t[i] = i, rk[i] = s[i];
    qsort();
    for (int j = 1; j <= l; j <<= 1) {
      int tot = 0;
      for (int i = l - j + 1; i <= l; i++) t[++tot] = i;
```

```
      for (int i = 1; i <= l; i++)
        if (sa[i] - j > 0) t[++tot] = sa[i] - j;
      qsort();
      memcpy(t, rk, sizeof(int) * (l + 1));
      rk[sa[1]] = tot = 1;
      for (int i = 2; i <= l; i++)
        rk[sa[i]] =
            t[sa[i - 1]] == t[sa[i]] && t[sa[i - 1] + j] == t[sa[i] + j] ? tot : ++tot;
    }
  }
  // 下面是 height 的部分
  int move(int x, int y, int len) {
    while (x + len <= l && y + len <= l && s[x + len] == s[y + len]) ++len;
    return len;
  }
  void calc_h() {
    for (int i = 1; i <= l; i++)
      h[i] = rk[i] == 1 ? 0 : move(i, sa[rk[i] - 1], max(h[i - 1] - 1, 0));
  }
  int st[N][20]; // h[sa[i]]~h[sa[i+2^j]] 中的最小值
  void make_st() {
    for (int i = 1; i <= l; i++) st[i][0] = h[sa[i]];
    for (int j = 1; (1 << j) <= l; j++) {
      int step = 1 << (j - 1);
      for (int i = 1; i + step <= l; i++) {
        st[i][j] = min(st[i][j - 1], st[i + step][j - 1]);
      }
    }
  }
  int lg2[N];
  void init_lg() { FOR(i, 2, l) lg2[i] = lg2[i / 2] + 1; }
  void prepare_lcp() { // 如果要 lcp 的话只用调用这个就行
    make();
    calc_h();
    make_st();
    init_lg();
  }
  int lcp(int x, int y) { // 返回长度
    if (x == y) return l - x + 1;
    x = rk[x], y = rk[y];
    if (x > y) swap(x, y);
    x++; // 取不到 x
    int step = lg2[y - x + 1];
    return min(st[x][step], st[y - (1 << step) + 1][step]);
  }
};
```

code/sa.cpp

2.4 后缀自动机

```
const int SZ = 2e6 + 500, ALP = 26;
```

```cpp

struct SAM {
  int tot, last;
  int tr[SZ][ALP], fail[SZ];
  int len[SZ], cnt[SZ], end[SZ];
  int s[SZ], ls;
  SAM() { tot = last = 1, len[1] = 0, fail[1] = 0; }
  void insert(char x) {
    s[++ls] = x;
    x -= 'a';
    int u = ++tot, p = last;
    len[u] = len[last] + 1, last = u;
    cnt[u] = 1, end[u] = ls; // u 的卫星信息
    while (p && tr[p][x] == 0) tr[p][x] = u, p = fail[p];
    if (!p) fail[u] = 1;
    else {
      int q = tr[p][x];
      if (len[q] == len[p] + 1) fail[u] = q;
      else {
        int cq = ++tot;
        len[cq] = len[p] + 1, fail[cq] = fail[q];
        end[cq] = end[q]; //如果需要，更新的cq的卫星信息
        memcpy(tr[cq], tr[q], sizeof(tr[q]));
        fail[q] = fail[u] = cq;
        while (p && tr[p][x] == q) tr[p][x] = cq, p = fail[p];
      }
    }
  }
  int a[SZ], bin[SZ], tim[SZ];
  void count() { //桶排，统计cnt
    FOR(i, 1, tot) bin[len[i]]++;
    FOR(i, 1, tot) bin[i] += bin[i - 1];
    FOR(i, 1, tot) a[bin[len[i]]--] = i;
    ROF(i, tot, 1) cnt[fail[a[i]]] += cnt[a[i]];
  }
  void print_node(int u) { //输出每个结点的状态
    printf("u=%d,cnt=%d,len=%d,fail=%d, ", u, cnt[u], len[u], fail[u]);
    FOR(i, end[u] - len[u] + 1, end[u]) putchar(s[i]);
    puts("");
  }
};
/*
 * cnt:状态出现次数;end: 状态的结尾位置
 * tim: 每个结点的时间戳
 */
```

code/sam.cpp

2.5 广义后缀自动机

```cpp
/******************heading******************/
const int N = 2e6 + 50, C = 12;
```

```
3
4  struct qxx {
5    int nex, t;
6  };
7  qxx e[N * 2];
8  int h[N], le = 1;
9  void add_path(int f, int t) { e[++le] = (qxx){h[f], t}, h[f] = le; }
10
11 int n, c, col[N], dg[N];
12
13 namespace T {
14   int tr[N][C], tr_tot = 1;
15   void dfs_add_trie(int u, int p, int tu) {
16     int cu = col[u];
17     if (!tr[tu][cu]) tr[tu][cu] = ++tr_tot;
18     tu = tr[tu][cu];
19     for (int i = h[u], v; v = e[i].t, i; i = e[i].nex) {
20       if (v == p) continue;
21       dfs_add_trie(v, u, tu);
22     }
23   }
24 } // namespace T
25 const int SZ = 2e6 + 5, ALP = 11;
26 int last[SZ];
27 struct SAM {
28   int tot;
29   int len[SZ], tr[SZ][ALP], fail[SZ];
30   int tnode[SZ];
31   // tnode表示trie上状态的左后一个字符的某一个结点。
32   SAM() { tot = 1, len[1] = 0, fail[1] = 0; }
33   void insert(int tu, int x) {
34     int v = T::tr[tu][x], p = last[tu], u = tr[p][x];
35     if (!u) {
36       u = ++tot;
37       tnode[u] = v;
38       len[u] = len[p] + 1;
39       while (p && !tr[p][x]) tr[p][x] = u, p = fail[p];
40     }
41     last[v] = u;
42     if (!p) fail[u] = 1;
43     else {
44       int q = tr[p][x];
45       if (len[q] == len[p] + 1) fail[u] = q;
46       else {
47         int cq = ++tot;
48         len[cq] = len[p] + 1;
49         fail[cq] = fail[q];
50         tnode[cq] = tnode[q];
51         memcpy(tr[cq], tr[q], sizeof(tr[q]));
52         fail[q] = fail[u] = cq;
53         while (p && tr[p][x] == q) tr[p][x] = cq, p = fail[p];
54       }
55     }
```

```
  }
  void print_node(int u) {
    printf("u=%2d,len=%2d,fail=%2d,tnode=%2d\n", u, len[u], fail[u], tnode[u]);
  }
  void count() {
    lld ans = 0;
    FOR(i, 1, tot) ans += len[i] - len[fail[i]];
    printf("%lld", ans);
  }
} sam;
queue<int> q;
void go() {
  last[1] = 1;
  q.push(1);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    FOR(i, 0, c - 1) {
      if (!T::tr[u][i]) continue;
      sam.insert(u, i);
      q.push(T::tr[u][i]);
    }
  }
  sam.count();
}
int main() {
  scanf("%d%d", &n, &c);
  FOR(i, 1, n) scanf("%d", &col[i]);
  FOR(i, 1, n - 1) {
    int u, v;
    scanf("%d%d", &u, &v);
    add_path(u, v), add_path(v, u);
    dg[u]++, dg[v]++;
  }
  FOR(i, 1, n) if (dg[i] == 1) T::dfs_add_trie(i, 0, 1);
  go();
  return 0;
}
```

code/general_sam.cpp

2.6 回文自动机

```
const int SZ = 5e5 + 500, ALP = 26;
struct PAM {
  int tot, last;
  int len[SZ], tr[SZ][ALP], fail[SZ];
  int s[SZ], ls; // 字符串的内容 -'0'
  int cnt[SZ], num[SZ]; //状态出现次数、fail树上的深度（有多少回文后缀）
  int newnode(int l) {
    ++tot, len[tot] = l, fail[tot] = 0, cnt[tot] = 0;
    FOR(i, 0, ALP - 1) tr[tot][i] = 0;
```

```
    return tot;
  }
  void clear() {
    tot = -1, newnode(0), newnode(-1), fail[0] = 1, last = 0,
    s[ls = 0] = -1; //减掉'a'后0就不是非匹配字符了，所以要整成-1
  }
  PAM() { clear(); }
  int getfail(int u) {
    // 将结点 u 的 fail 链状态上的状态尝试去用 s[ls] 扩展，返回这个可扩展的结点
    while (s[ls - len[u] - 1] != s[ls]) u = fail[u];
    return u;
  }
  void insert(char c) {
    s[++ls] = (c -= 'a');
    int cur = getfail(last);
    if (!tr[cur][c]) { // 如果没有转移就添加
      int u = newnode(len[cur] + 2);
      fail[u] = tr[getfail(fail[cur])][c];
      tr[cur][c] = u;
      // 在此处更新 tot 的卫星信息
      num[tot] = num[fail[tot]] + 1;
    }
    last = tr[cur][c];
    // 在此处更新 last 的卫星信息
    cnt[last]++;
  }
  void count() { //最后用来计算每个状态的出现次数
    ROF(i, tot, 0) cnt[fail[i]] += cnt[i];
  }
};
/*
 * 0 号结点表示 0 结点
 * 1 号结点表示 -1 结点，长度为 -1
 * last 记录上一次插入的字符所在结点的编号
 */
```

code/pam.cpp

3 数论与线性代数

3.1 EX-BSGS 算法

```
namespace EXBSGS {
  const lld SZ = 433337;
  struct hash_map {
    struct data {
      lld u, v, nex;
    };
    data e[SZ];
    lld h[SZ], le;
    lld hash(lld u) { return (u % SZ + SZ) % SZ; }
```

```
    lld &operator[](lld u) {
      lld hu = hash(u);
      for (lld i = h[hu]; i; i = e[i].nex)
        if (e[i].u == u) return e[i].v;
      return e[++le] = (data){u, -1, h[hu]}, h[hu] = le, e[le].v;
    }
    void clear() { memset(h, 0, sizeof(h)), le = 0; }
  } h;
  lld gcd(lld a, lld b) { return b ? gcd(b, a % b) : a; }
  lld mul(lld a, lld b, lld p) {
    if (p <= 1000000000ll) return 1ll * a * b % p;
    if (p <= 100000000000ll)
      return (((a * (b >> 20) % p) << 20) % p + a * (b & ((1 << 20) - 1))) % p;
    lld d = floor(a * (long double)b / p);
    lld res = (a * b - d * p) % p;
    if (res < 0) res += p;
    return res;
  }
  lld pw(lld a, lld m, lld p) {
    lld res = 1;
    while (m) m & 1 ? res = mul(res, a, p) : 0, a = mul(a, a, p), m >>= 1;
    return res;
  }
  lld exgcd(lld a, lld b, lld &x, lld &y) {
    if (!b) return x = 1, y = 0, a;
    lld t = exgcd(b, a % b, y, x);
    return y = y - (a / b) * x, t;
  }
  lld inv(lld a, lld p) {
    lld b, t, g = exgcd(a, p, b, t);
    if (g > 1) return -1;
    return ((b % p) + p) % p;
  }
  lld bsgs(lld a, lld b, lld p) {
    a %= p, b %= p, h.clear();
    if (b == 1) return 0;
    if (!a && !b) return 1;
    if (!a) return -1;
    lld t = sqrt(p) + 0.5, cur = b, q = 1;
    FOR(i, 0, t) h[cur] = i, cur = mul(cur, a, p);
    cur = pw(a, t, p);
    FOR(i, 0, t) {
      if (h[q] != -1 && i * t - h[q] >= 0) return i * t - h[q];
      q = mul(q, cur, p);
    }
    return -1;
  }
  lld exbsgs(lld a, lld b, lld p) {
    lld d = 0, f = 1, g;
    while ((g = gcd(a, p)) > 1) {
      if (b % g) return -1;
      ++d, f = mul(f, g, p), b /= g, p /= g;
    }
```

```
    lld ia = inv(a, p);
    f = mul(f, pw(ia, d, p), p);
    b = mul(b, f, p);
    lld res = bsgs(a, b, p);
    return ~res ? res + d : -1;
  }
} // namespace EXBSGS
/*
 * exbsgs: 求a^x=b mod p 的最小非负整数解。-1表示无解
 */
```

code/exbsgs.cpp

3.2 Pollard-Rho 和 Miller

```
namespace math {
  inline int powmod(int a, int b, int mod) {
    int res = 1;
    for (; b; b >>= 1, a = 1ll * a * a % mod)
      if (b & 1) res = 1ll * res * a % mod;
    return res;
  }
  inline LL mul(LL a, LL b, LL p) {
    if (p <= 1000000000ll) return 1ll * a * b % p;
    if (p <= 1000000000000ll)
      return (((a * (b >> 20) % p) << 20) % p + a * (b & ((1 << 20) - 1))) % p;
    LL d = floor(a * (long double)b / p);
    LL res = (a * b - d * p) % p;
    if (res < 0) res += p;
    return res;
  }
  inline LL powmod(LL a, LL b, LL mod) {
    LL res = 1;
    for (; b; b >>= 1, a = mul(a, a, mod))
      if (b & 1) res = mul(res, a, mod);
    return res;
  }
  inline bool check(LL a, LL x, LL times, LL n) {
    LL tmp = powmod(a, x, n);
    while (times--) {
      LL last = mul(tmp, tmp, n);
      if (last == 1 && tmp != 1 && tmp != n - 1) return 0;
      tmp = last;
    }
    return tmp == 1;
  }
  int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
  const int S = 8;
  inline bool Miller(LL n) {
    FOR(i, 0, S) {
      if (n == base[i]) return 1;
      if (n % base[i] == 0) return 0;
```

```
    }
    LL x = n - 1, times = 0;
    while (!(x & 1)) times++, x >>= 1;
    FOR(_, 0, S) if (!check(base[_], x, times, n)) return 0;
    return 1;
  }
#define mytz __builtin_ctzll
  inline LL gcd(LL a, LL b) {
    if (!a) return b;
    if (!b) return a;
    register int t = mytz(a | b);
    a >>= mytz(a);
    do {
      b >>= mytz(b);
      if (a > b) {
        LL t = b;
        b = a, a = t;
      }
      b -= a;
    } while (b);
    return a << t;
  }
#define F(x) ((mul(x, x, n) + c) % n)
  inline LL rho(LL n, LL c) {
    LL x = 1ll * rand() * rand() % n, y = F(x);
    while (x ^ y) {
      LL w = gcd(abs(x - y), n);
      if (w > 1 && w < n) return w;
      x = F(x), y = F(y), y = F(y);
    }
    return 1;
  }
  inline LL calc(LL x) {
    if (Miller(x)) return x;
    LL fsf = 0; // while((fsf=rho(x,rand()%x))==1);
    while ((fsf = rho(x, 2)) == 1)
      ;
    return max(calc(fsf), calc(x / fsf));
  }
} // namespace math
/*
 * 不知从何处拉来的强大板子
 * func.powmod(int,int,int) 32位整型快速幂
 * func.mul(LL,LL,LL) 64位长整型数的乘法运算
 * func.powmod(LL,LL,LL) 64位长整型数的快速幂
 */
```

code/math.cpp

3.3 线性基

```
const int N = 100;
```

```cpp
long long p[N];
void insert(long long x) {
  ROF(i, 62, 0) {
    if (!((x >> i) & 1)) continue; //判断第i位是否为0
    if (!p[i]) {
      p[i] = x;
      break;
    } //不能张成，添加到线性基中
    x ^= p[i]; //去掉可以张成的维度（第i维消元）
  }
}
```

code/linear_basis.cpp

```cpp
int n;
long long b[70], lb;
bool insert(long long x) { //消成对角矩阵
  ROF(i, 63, 0) {
    if (!(x >> i & 1)) continue;
    if (!b[i]) {
      b[i] = x, lb++;
      ROF(j, i - 1, 0) if (b[i] >> j & 1) b[i] ^= b[j]; //消掉i的其他元
      FOR(j, i + 1, 63) if (b[j] >> i & 1) b[j] ^= b[i]; //消掉其他元的i
      return 1;
    }
    x ^= b[i];
  }
  return 0;
}
void print() {
  int len = 0;
  FOR(i, 0, 63) if (b[i]) len = i;
  FOR(i, 0, len) {
    ROF(j, len, 0) { printf("%lld", b[i] >> j & 1); }
    puts("");
  }
}
```

code/linear_basis2.cpp

```cpp
struct basis {
  long long b[61];
  basis() { memset(b, 0, sizeof(b)); }
  bool insert(long long x) {
    ROF(i, 60, 0) {
      if (x >> i & 1) {
        if (b[i]) x ^= b[i];
        else return b[i] = x, 1;
      }
    }
    return 0;
  }
  long long qmax() {
    long long res = 0;
```

```
    ROF(i, 60, 0) if ((res ^ b[i]) > res) res ^= b[i];
    return res;
  }
  basis operator+(basis bi) {
    basis res = bi;
    FOR(i, 0, 60) if (b[i]) res.insert(b[i]);
    return res;
  }
};
/*
 * 上三角矩阵
 * 重载了合并的运算符
 */
```

code/linear_basis3.cpp

3.4 Min 25

```
// by Yao
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
#define ROF(i, a, b) for (int i = (a); i >= (b); --i)

typedef long long LL;

/**
 * Min_25 筛
 *
 * ref: https://notes.sshwy.name/Math/Min_25/
 *
 * N: 要筛的值的上界
 * CNT: f1 的项数
 * COEF[CNT]: f1 对应的系数。const 指定义为全局变量数组（里面的元素可改）
 * ef1: 计算 f1 的点值。第一个参数是点，第二个参数是要存储的数组
 * esf1: 计算 f1 前缀和的点值
 * f_pe: 计算 f 在质数幂处的值
 * MOD: 模数
 */
// Usage (https://www.luogu.com.cn/problem/P5325):
const LL N = 1e10;
const int P = 1e9 + 7, I6 = (P + 1) / 6, I2 = (P + 1) / 2;

int coef[] = {P - 1, 1};

void f1(LL x, int *result) {
  x %= P;
  result[0] = x % P;
  result[1] = x * 1ll * x % P;
}
void prefixSumF1(LL x, int *result) {
  x %= P;
```

```
  result[0] = x * (x + 1) % P * I2 % P;
  result[1] = x * (x + 1) % P * (x * 2 + 1) % P * I6 % P;
}
int f_pe(int p, int e, LL pe) {
  pe %= P;
  return 1ll * pe * (pe - 1) % P;
}

int main() {
  long long n;
  scanf("%lld", &n);

  Min25<N, 2, coef, f1, prefixSumF1, f_pe, P> Sieve;
  int ans = Sieve.sieve(n);

  printf("%d\n", ans);
  return 0;
}
// Min_25
template <const LL N, const int CNT, const int COEF[CNT], void (*ef1)(LL, int *),
    void (*esf1)(LL, int *), int (*f_pe)(int, int, LL), const int MOD>
struct Min25 {
  vector<vector<int>> g, h, hs;
  vector<int> id[2], pn;
  vector<LL> val;
  LL n;
  int SQRT_N, sqrt_n, tot, lp;

  Min25() {
    SQRT_N = 2 * (sqrt(N) + 5);
    tot = lp = 0;
    g.resize(SQRT_N, vector<int>(CNT, 0));
    h.resize(SQRT_N, vector<int>(CNT, 0));
    hs.resize(SQRT_N, vector<int>(CNT, 0));
    id[0].resize(SQRT_N, 0), id[1].resize(SQRT_N, 0);
    val.resize(SQRT_N, 0), pn.resize(SQRT_N, 0);
  }
  // Min_25
  void init() {
    sqrt_n = sqrt(n) + 3;
    vector<bool> co(SQRT_N, false);
    co[0] = co[1] = 1;
    for (int i = 2; i <= sqrt_n; i++) {
      if (!co[i]) pn[++lp] = i;
      for (int j = 1; j <= lp && 1ll * i * pn[j] <= sqrt_n; j++) {
        co[i * pn[j]] = 1;
        if (i % pn[j] == 0) break;
      }
    }
    for (LL pos = 1, nex, w; pos <= n; pos = nex + 1) {
      nex = n / (n / pos), w = n / pos, val[++tot] = w;
      w <= sqrt_n ? id[0][w] = tot : id[1][n / w] = tot;
    } // FOR(i, 1, tot) assert(I(val[i]) == i);
```

```
}
void calc_h() {
  int tmp[CNT];
  FOR(i, 1, lp) {
    ef1(pn[i], tmp);
    FOR(j, 0, CNT - 1) {
      h[i][j] = tmp[j];
      hs[i][j] = (hs[i - 1][j] + tmp[j]) % MOD;
    }
  }
}
int H(int i) { // 计算 sum_j f(p_j) (j <= i)  // assert(i <= lp);
  int res = 0;
  FOR(j, 0, CNT - 1) res = (res + 1ll * hs[i][j] * COEF[j]) % MOD;
  return res;
}
int I(LL x) { return x <= sqrt_n ? id[0][x] : id[1][n / x]; }
void calc_g() {
  FOR(i, 1, tot) { //当i=0
    int tmp[CNT];
    esf1(val[i], tmp);
    FOR(j, 0, CNT - 1) {
      g[i][j] = (tmp[j] - 1 + MOD) % MOD; // 对于积性函数来说必然有 f(1) = 1
    }
  }
  FOR(i, 1, lp) { // pn[i]
    FOR(j, 1, tot) {
      if (1ll * pn[i] * pn[i] > val[j]) break;
      int k = I(val[j] / pn[i]);
      FOR(t, 0, CNT - 1) {
        g[j][t] =
            (g[j][t] - 1ll * h[i][t] * (g[k][t] - hs[i - 1][t]) % MOD + MOD) % MOD;
      }
    }
  }
}
int G(LL x) { // 计算 sum f(p) (p <= x 且 p 是质数)
  int res = 0;
  FOR(i, 0, CNT - 1) res = (res + 1ll * g[I(x)][i] * COEF[i]) % MOD;
  return res;
}
int S(int i, LL m) {
  if (m < pn[i] || m <= 1) return 0;
  LL res = (G(m) - H(i - 1) + MOD) % MOD;
  FOR(j, i, lp) {
    if (1ll * pn[j] * pn[j] > m) break;
    LL pje = 1, pje1 = pn[j];
    FOR(e, 1, 100) {
      pje *= pn[j], pje1 *= pn[j];
      if (pje1 > m) break;
      res += 1ll * f_pe(pn[j], e, pje) * S(j + 1, m / pje) % MOD +
             f_pe(pn[j], e + 1, pje1);
      res %= MOD;
```

```
      }
    }
    return res;
  }

  int sieve(LL _n) {
    n = _n, init(), calc_h(), calc_g();
    return (S(1, n) + 1) % MOD;
  }
};
```

code/min_25.cpp

3.5 Min 25 杰哥

```
typedef long long LL;

LL a[N], T, n;
int pr[N], id1[N], id2[N], flag[N], g[N], sum[N], ncnt, m;

int ID(LL x) { return x <= T ? id1[x] : id2[n / x]; }
int calc(LL x) {
  return x %= mod, x * (x + 1) / 2 % mod - 1;
} // 算完全积性函数的前缀和
int f1(LL x) { return x %= mod, x * (x - 1) % mod; } // 算积性函数在质数位置的取值
int f(LL j, int c) {} // 计算 f(j)，其中 j 是某个质数的 c 次方

void init() {
  T = sqrt(n + 0.5);
  for (int i = 2; i <= T; i++) {
    if (!flag[i]) pr[++ncnt] = i, sum[ncnt] = (sum[ncnt - 1] + f1(i)) % mod;
    for (int j = 1; j <= ncnt && (LL)i * pr[j] <= T; j++) {
      flag[i * pr[j]] = 1;
      if (i % pr[j] == 0) break;
    }
  }
  for (LL l = 1; l <= n; l = n / (n / l) + 1) {
    a[++m] = n / l;
    if (a[m] <= T) id1[a[m]] = m;
    else id2[n / a[m]] = m;
    g[m] = calc(a[m]);
  }
  for (int i = 1; i <= ncnt; i++)
    for (int j = 1; j <= m && (LL)pr[i] * pr[i] <= a[j]; j++)
      g[j] = (g[j] - (LL)f1(pr[i]) * (g[ID(a[j] / pr[i])] - sum[i - 1]) % mod + mod) %
             mod;
}

int solve(LL n, int m) {
  if (n < pr[m]) return 0;
  int res = (g[ID(n)] * (LL)2 * t - (LL)sum[m - 1] * 2 * t) % mod;
  res = (res + mod) % mod;
```

```
  for (int i = m; i <= ncnt && (LL)pr[i] * pr[i] <= n; i++)
    for (LL j = pr[i], c = 1; j * pr[i] <= n; j *= pr[i], c++)
      Inc(res, ((LL)solve(n / j, i + 1) * f(j, c) + f(j * pr[i], c + 1)) % mod);
  return res;
}
```

code/min_25_jie.cpp

3.6 二次剩余 Cipolla

```
int pw(int a, int m, int p) {
  int res = 1;
  while (m) m & 1 ? res = 1ll * res * a % p : 0, a = 1ll * a * a % p, m >>= 1;
  return res;
}
struct sqrtNum {
  int W, P;
  int a, b; // a+b*sqrt(W)
  sqrtNum(int _a, int _b, int _w, int _p) { a = _a, b = _b, W = _w, P = _p; }
  sqrtNum operator*(const sqrtNum &x) const {
    return sqrtNum((1ll * a * x.a % P + 1ll * b * x.b % P * W) % P,
        (1ll * a * x.b % P + 1ll * b * x.a) % P, W, P);
  }
};
int quad_res(int n, int p) {
  n %= p;
  assert(p & 1);
  if (pw(n, (p - 1) / 2, p) == p - 1) return -1; // no solution
  if (n == 0) return 0;
  int a;
  srand(clock() + time(0));
  do a = rand() % p;
  while (pw((a * 1ll * a % p - n + p) % p, (p - 1) / 2, p) == 1);
  int w2 = (a * 1ll * a % p - n + p) % p;
  sqrtNum q(a, 1, w2, p), qm(1, 0, w2, p);
  for (int m = (p + 1) / 2; m; m >>= 1, q = q * q)
    if (m & 1) qm = qm * q;
  assert(qm.b == 0);
  return qm.a;
}
```

code/cipolla.cpp

3.7 特征多项式

```
int pw(int a, int m) {
  int res = 1;
  while (m) m & 1 ? res = 1ll * res * a % P : 0, a = 1ll * a * a % P, m >>= 1;
  return res;
}
```

```cpp

typedef vector<int> Poly;

Poly operator-(const Poly &p, const Poly &q) {
  Poly res = p;
  res.resize(max(p.size(), q.size()), 0);
  for (long unsigned i = 0; i < q.size(); ++i) res[i] = (res[i] - q[i] + P) % P;
  return res;
}
Poly operator+(const Poly &p, const Poly &q) {
  Poly res = p;
  res.resize(max(p.size(), q.size()), 0);
  for (long unsigned i = 0; i < q.size(); ++i) res[i] = (res[i] + q[i]) % P;
  return res;
}
Poly operator*(const Poly &p, const Poly &q) {
  if (!p.size() || !q.size()) return Poly();
  Poly res(p.size() + q.size() - 1, 0);
  for (long unsigned i = 0; i < p.size(); i++)
    for (long unsigned j = 0; j < q.size(); j++)
      res[i + j] = (res[i + j] + p[i] * 1ll * q[j] % P) % P;
  return res;
}
Poly operator%(const Poly &p, const Poly &q) { // mod
  assert(q.size());
  Poly res = p;
  while (res.size() >= q.size()) {
    int d = res.size() - q.size();
    int rate = res.back() * 1ll * pw(q.back(), P - 2) % P;
    for (long unsigned i = 0; i < q.size(); i++)
      res[i + d] = (res[i + d] - q[i] * 1ll * rate % P + P) % P;
    assert(res.back() == 0);
    res.pop_back();
  }
  return res;
}

int t[N][N];
int det(int g[N][N], int n) {
  memcpy(t, g, sizeof(t)), g = t;
  int res = 1;
  FOR(i, 1, n) {
    if (!g[i][i]) FOR(j, i + 1, n) if (g[j][i]) {
        FOR(k, i, n) swap(g[i][k], g[j][k]);
        res = P - res;
        break;
      }
    if (!g[i][i]) return 0;
    FOR(j, 1, n) if (i != j) {
      int rate = g[j][i] * 1ll * pw(g[i][i], P - 2) % P;
      FOR(k, i, n) g[j][k] = (g[j][k] - g[i][k] * 1ll * rate % P + P) % P;
    }
  }
```

```
  FOR(i, 1, n) res = res * 1ll * g[i][i] % P;
  return res;
}

int t2[N][N];
Poly p[N];
Poly characteristic_polynomial(int M[N][N], int n) { //求M的特征多项式
  memcpy(t2, M, sizeof(t2)), M = t2;
  int x = det(M, n);
  // 1. 化为上海森堡矩阵
  FOR(j, 1, n - 1) { //列
    if (!M[j + 1][j]) FOR(i, j + 2, n) if (M[i][j]) {
        FOR(k, 1, n) swap(M[j + 1][k], M[i][k]); // R[j+1] <-> R[i]
        FOR(k, 1, n) swap(M[k][j + 1], M[k][i]); // C[j+1] <-> C[i]
        break;
      }
    //把第j列的第j+1行以下的位置全部消元
    FOR(i, j + 2, n) if (M[i][j]) {
      int rate = M[i][j] * 1ll * pw(M[j + 1][j], P - 2) % P;
      FOR(k, 1, n)
      M[i][k] = (M[i][k] - M[j + 1][k] * 1ll * rate % P + P) %
                P; // R[i]   = R[i]-rate*R[j+1]
      FOR(k, 1, n)
      M[k][j + 1] =
          (M[k][j + 1] + M[k][i] * 1ll * rate % P) % P; // C[j+1] = C[j+1]+rate*C[i]
    }
  }
  assert(x == det(M, n));
  // 2. 计算特征多项式: 即(xI-A)的行列式
  p[n + 1] = Poly(1, 1);
  ROF(i, n, 1) {
    Poly s, t;

    t = Poly(1, P - M[i][n]);
    if (i == n) t.pb(1); // x-M[n][n]
    s = p[n + 1] * t;
    ROF(j, n, i + 1) {
      t = Poly(1, P - M[i][j - 1]);
      if (i == j - 1) t.pb(1);
      s = t * p[j] - Poly(1, P - M[j][j - 1]) * s;
    }
    p[i] = s;
  }
  return p[1];
}
```

code/characteristic_polynomial.cpp

3.8 中国剩余定理 & exgcd

```
LL mul(LL a, LL b, LL p) { // 这个函数貌似目前只支持非负整数
  if (a <= 1000000000 && b <= 1000000000) return a * b % p;
```

```
  LL res = 0;
  while (b) {
    if (b & 1) res = (res + a) % p;
    a = a * 2 % p, b >>= 1;
  }
  return res;
}
LL exgcd(LL a, LL b, LL &x, LL &y) {
  if (!b) return x = 1, y = 0, a;
  LL t = exgcd(b, a % b, y, x);
  return y = y - (a / b) * x, t;
}
// v_i(A, B): x = A (mod B)
pair<LL, LL> go(vector<pair<LL, LL>> v) {
  LL b = 0, a = 1; // x=0 mod 1
  for (auto p : v) {
    LL a1, b1, k, k1;
    a1 = p.second;
    b1 = p.first;
    LL g = exgcd(a, a1, k, k1), d = ((b1 - b) % a1 + a1) % a1;
    if (d % g) return make_pair(-1, -1);
    k = mul(k, d / g, a1);
    // 然后合并方程
    b = b + a * k, a = a / g * a1, b = (b + a) % a;
  }
  return make_pair((b + a) % a, a);
}
```

code/crt.cpp

3.9 类欧几里得算法

```
/**
 * 类欧几里德算法
 * 计算 sum (0 <= i <= n) floor((a*i+b)/c)
 */
int f(int a, int b, int c, int n) {
  if (a == 0) { return (b / c) * (n + 1); }
  if (a < c && b < c) {
    int m = (a * n + b) / c;
    return m * n - f(c, c - b - 1, a, m - 1);
  }
  return f(a % c, b % c, c, n) + (b / c) * (n + 1) + (a / c) * (n * (n + 1) / 2);
}
```

code/euclideanoid.cpp

3.10 自然数幂和

```
const int P = 1e9 + 7, K = 1005;
```

```cpp
int pw(int a, int m) {
  int res = 1;
  while (m) m & 1 ? res = 1ll * res * a % P : 0, a = 1ll * a * a % P, m >>= 1;
  return res;
}
int fnv[K], d[K];
void work(int k, int *d) { // prepare for 0^k + 1^k + 2^k + ...
  FOR(i, 0, k + 1) d[i] = pw(i, k);
  FOR(i, 1, k + 1) ROF(j, k + 1, i + 1) d[j] = (d[j] - d[j - 1] + P) % P;
  fnv[0] = 1;
  FOR(i, 1, k + 1) fnv[i] = 1ll * fnv[i - 1] * pw(i, P - 2) % P;
}
int calc(int n, int k, int *d) { // 0^k + ... + n^k
  int res = 0, coef = 1;
  FOR(i, 0, k + 1) {
    res = (res + 1ll * coef * fnv[i] % P * d[i]) % P;
    coef = 1ll * coef * (n - i) % P;
  }
  return res;
}
```

code/faulhaber.cpp

4 多项式相关

4.1 FFT

```cpp
namespace FFT {
  const int N = (1 << 21) + 5;
  const double PI = acos(-1);
  struct cpx {
    double a, b;
    cpx(double _a = 0, double _b = 0) { a = _a, b = _b; }
    cpx operator+(cpx c) { return cpx(a + c.a, b + c.b); }
    cpx operator-(cpx c) { return cpx(a - c.a, b - c.b); }
    cpx operator*(cpx c) { return cpx(a * c.a - b * c.b, a * c.b + b * c.a); }
  };
  int tr[N], d;
  void dft(cpx f[], int len, int typ) {
    for (int i = 0; i < len; i++)
      if (i < tr[i]) swap(f[i], f[tr[i]]);
    for (int j = 1; j < len; j <<= 1) {
      cpx wn(cos(PI / j), sin(typ * PI / j));
      for (int i = 0; i < len; i += j << 1) {
        cpx w(1, 0), u, v;
        for (int k = i; k < i + j; k++, w = w * wn)
          u = f[k], v = f[k + j] * w, f[k] = u + v, f[k + j] = u - v;
      }
    }
    if (typ == -1)
      for (int i = 0; i < len; i++) f[i].a /= len;
```

```
  }
  int init(int l) {
    d = 0;
    int len = 1;
    while (len < (l << 1)) len <<= 1, d++;
    for (int i = 1; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1) << (d - 1);
    return len;
  }
} // namespace FFT
typedef FFT::cpx dft[FFT::N];
/*
 * L5 : ai+b
 * L15: w_(2*j) 的单位根，2*PI/(2*j)=PI/j
 * L22: 实部
 * L28: 蝴蝶变换预处理
 * L32: 定义dft类型，表示一个cpx的数组
 * dft(f,len,typ): len 必须是 2^k
 * init(l): l 是参与运算的多项式的长度之和，返回一个2^k的长度
 */
```

code/fft.cpp

4.2 NTT

```
namespace NTT {
  const int N = (1 << 21) + 5, P = 998244353;
  int pw(int a, int m) {
    int res = 1;
    while (m) m & 1 ? res = 1ll * res * a % P : 0, a = 1ll * a * a % P, m >>= 1;
    return res;
  }
  int tr[N], d;
  void dft(int f[], int len, int typ) {
    FOR(i, 0, len - 1) if (i < tr[i]) swap(f[i], f[tr[i]]);
    for (int j = 1; j < len; j <<= 1) {
      int wn = pw(3, (P - 1) / (j << 1) * typ + P - 1);
      for (int i = 0; i < len; i += j << 1) {
        int w = 1, u, v;
        for (int k = i; k < i + j; k++, w = 1ll * w * wn % P) {
          u = f[k], v = 1ll * w * f[k + j] % P;
          f[k] = u + v, f[k] < P ? 0 : f[k] -= P;
          f[k + j] = u - v, f[k + j] < 0 ? f[k + j] += P : 0;
        }
      }
    }
    if (typ == -1) {
      int x = pw(len, P - 2);
      for (int i = 0; i < len; i++) f[i] = 1ll * f[i] * x % P;
    }
  }
  int init(int l) {
    d = 0;
```

```
    int len = 1;
    while (len < (l << 1)) len <<= 1, ++d;
    for (int i = 1; i < len; i++) tr[i] = (tr[i >> 1] >> 1) | (i & 1) << (d - 1);
    return len;
  }
} // namespace NTT
typedef int dft[NTT::N];
/*
 * 必须保证初始时f里的值非负!
 */
```

code/ntt.cpp

4.3 FWT

```
/******************heading******************/
#define int lld
const int N = 1 << 17, P = 998244353;
int n, in;
int a[N], b[N], c[N];

int pw(int a, int m, int p) {
  int res = 1;
  while (m) m & 1 ? res = res * a % p : 0, a = a * a % p, m >>= 1;
  return res;
}
void fwt_and(int *f, int tag) {
  for (int j = 1; j < n; j <<= 1)
    for (int i = 0; i < n; i += j << 1)
      for (int k = i; k < i + j; k++) f[k] = (f[k] + f[k + j] * tag) % P;
  FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
}
void fwt_or(int *f, int tag) {
  for (int j = 1; j < n; j <<= 1)
    for (int i = 0; i < n; i += j << 1)
      for (int k = i; k < i + j; k++) f[k + j] = (f[k + j] + f[k] * tag) % P;
  FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
}
void fwt_xor(int *f, int tag) {
  for (int j = 1; j < n; j <<= 1)
    for (int i = 0; i < n; i += j << 1)
      for (int k = i; k < i + j; k++) {
        int x = f[k], y = f[k + j];
        f[k] = (x + y) % P, f[k + j] = (x - y) % P;
      }
  if (tag == -1) FOR(i, 0, n - 1) f[i] = f[i] * in % P;
  FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
}
/*
 * fwt:tag=1
 * ifwt:tag=-1
 * in=pw(n,P-2,P) (inverse element if n)
```

```
38 */
```

code/fwt.cpp

4.4 全家桶

```cpp
const int N = 300010, mod = 998244353;
typedef long long LL;
int Pow(int x, int y) {
  int res = 1;
  for (; y; y >>= 1, x = (LL)x * x % mod)
    if (y & 1) res = (LL)res * x % mod;
  return res;
}

int Wn[2][17][N << 1], r[N << 2];

void Pre(int len) {
  for (int mid = 2, step = 0; mid <= len; mid <<= 1, step++) {
    int t1 = Pow(3, (mod - 1) / mid), t2 = Pow(t1, mod - 2);
    Wn[0][step][0] = Wn[1][step][0] = 1, Wn[1][step][1] = t1, Wn[0][step][1] = t2;
    for (int i = 2; i < (mid >> 1); i++) {
      Wn[1][step][i] = (LL)Wn[1][step][i - 1] * t1 % mod;
      Wn[0][step][i] = (LL)Wn[0][step][i - 1] * t2 % mod;
    }
  }
}

void Inc(int &x, int y) { x += y, x -= x >= mod ? mod : 0; }
int Sub(int x, int y) {
  int t = x - y;
  return t < 0 ? t + mod : t;
}

void NTT(vector<int> &a, int len, int type) {
  for (int i = 0; i < len; i++)
    if (i < r[i]) swap(a[i], a[r[i]]);
  for (int mid = 2, step = 0; mid <= len; mid <<= 1, step++)
    for (int i = 0; i < len; i += mid)
      for (int j = i; j < i + (mid >> 1); j++) {
        int t = (LL)Wn[type][step][j - i] * a[j + (mid >> 1)] % mod;
        a[j + (mid >> 1)] = Sub(a[j], t), Inc(a[j], t);
      }
  if (!type)
    for (int i = 0, inv = Pow(len, mod - 2); i < len; i++) a[i] = (LL)a[i] * inv % mod;
}

void GetR(int len, int l) {
  for (int i = 1; i < len; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l - 1);
}

vector<int> mul(vector<int> a, vector<int> b) {
```

```cpp
  int t = a.size() + b.size() - 2, len = 1, l = 0;
  while (len <= t) len <<= 1, l++;
  GetR(len, l), a.resize(len + 1), b.resize(len + 1);
  NTT(a, len, 1), NTT(b, len, 1);
  for (int i = 0; i < len; i++) a[i] = (LL)a[i] * b[i] % mod;
  NTT(a, len, 0), a.resize(min(n + 1, t + 1));
  return a;
}

vector<int> Inv(const vector<int> &a, int n) {
  if (n == 1) return {Pow(a[0], mod - 2)};
  vector<int> b = Inv(a, (n + 1) / 2);
  int len = 1, l = 0;
  while (len <= n * 2) len <<= 1, l++;
  GetR(len, l), b.resize(len);
  vector<int> tmp(len);
  for (int i = 0; i < n && i < a.size(); i++) tmp[i] = a[i];
  NTT(tmp, len, 1), NTT(b, len, 1);
  for (int i = 0; i < len; i++)
    b[i] = (2 - (LL)tmp[i] * b[i] % mod + mod) * b[i] % mod;
  NTT(b, len, 0), b.resize(n);
  return b;
}

int inv[N];

vector<int> Ln(const vector<int> &a, int n) {
  vector<int> a1, inva = Inv(a, n);
  for (int i = 1; i < n && i < a.size(); i++) a1.push_back((LL)a[i] * i % mod);
  a1 = mul(a1, inva), a1.resize(n);
  for (int i = n - 1; i >= 1; i--)
    a1[i] = (LL)a1[i - 1] * inv[i] % mod, assert(inv[i]);
  return a1[0] = 0, a1;
}

vector<int> Exp(const vector<int> &a, int n) {
  if (n == 1) return {1};
  vector<int> tmp = Exp(a, (n + 1) / 2), ln = Ln(tmp, n), tmp1;
  for (int i = 0; i < n; i++) tmp1.push_back((a[i] - ln[i] + mod) % mod);
  tmp1[0]++, tmp = mul(tmp, tmp1), tmp.resize(n);
  return tmp;
}
```

code/poly.cpp

5 图论

5.1 MCMF 最大费用最大流

```cpp
namespace MCMF {
  struct qxx {
```

```
    int nex, t, v, c;
  };
  qxx e[N];
  int h[N], le = 1;
  void add_path(int f, int t, int v, int c) {
    e[++le] = (qxx){h[f], t, v, c}, h[f] = le;
  }
  void add_flow(int f, int t, int v, int c) {
    add_path(f, t, v, c), add_path(t, f, 0, -c);
  }

  bool vis[N];
  queue<int> q;
  int d[N], pre[N], incf[N];
  int s, t;
  bool spfa() {
    memset(d, -1, sizeof(d));
    q.push(s), d[s] = 0, incf[s] = INF, incf[t] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      vis[u] = 0;
      for (int i = h[u]; i; i = e[i].nex) {
        const int v = e[i].t, w = e[i].v, c = e[i].c;
        if (!w || d[v] >= d[u] + c) continue;
        d[v] = d[u] + c, incf[v] = min(incf[u], w), pre[v] = i;
        if (!vis[v]) q.push(v), vis[v] = 1;
      }
    }
    return incf[t];
  }
  int maxflow, maxcost;
  void update() {
    maxflow += incf[t];
    for (int u = t; u != s; u = e[pre[u] ^ 1].t) {
      e[pre[u]].v -= incf[t], e[pre[u] ^ 1].v += incf[t];
      maxcost += incf[t] * e[pre[u]].c;
    }
  }
  void go() {
    while (spfa()) update();
  }
} // namespace MCMF
/*
 * 注意，本板子是最大费用最大流!
 */
```

code/mcmf.cpp

5.2 DINIC 算法求最大流

```
const int N = 5e5 + 5, M = 5e5 + 5, INF = 0x3f3f3f3f;
```

```cpp
struct qxx {
  int nex, t, v;
};
qxx e[M];
int h[N], cnt = 1;
void add_path(int f, int t, int v) { e[++cnt] = (qxx){h[f], t, v}, h[f] = cnt; }
void add_flow(int f, int t, int v) { add_path(f, t, v), add_path(t, f, 0); }
void add_dual_flow(int f, int t, int v) { add_path(f, t, v), add_path(t, f, v); }

namespace DINIC {
  int s, t, maxflow, d[N];
  queue<int> q;
  bool bfs() {
    memset(d, 0, sizeof(d));
    q.push(s), d[s] = 1;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (int i = h[u]; i; i = e[i].nex) {
        const int &v = e[i].t, &w = e[i].v;
        if (!d[v] && w) d[v] = d[u] + 1, q.push(v);
      }
    }
    return d[t];
  }
  int dinic(int u, int flow) {
    if (u == t) return flow;
    int k, rest = flow;
    for (int i = h[u]; i && rest; i = e[i].nex) {
      const int &v = e[i].t, &w = e[i].v;
      if (!w || d[v] != d[u] + 1) continue;
      k = dinic(v, min(rest, w));
      if (k) e[i].v -= k, e[i ^ 1].v += k, rest -= k;
      else d[v] = 0;
    }
    return flow - rest;
  }
  void go() {
    while (bfs())
      for (int i; i = dinic(s, INF);) maxflow += i;
  }
} // namespace DINIC
/*
 * add_dual_flow:无向边的流（即双向可流）
 */
```

code/dinic.cpp

5.3 朱刘算法

```cpp
/**
 * Chu-Liu/Edmonds' algorithm
```

```
 * 计算有向图（允许重边、不允许自环）给定根的最小权外向生成树（最小树形图）
 * vector<Edge> buildFrom(n, r, ve): n 个点，边集是 ve，根是 r 的最小权外向生成树
 *   若无解则返回一个空的 vector
 *   要求 ve 非空
 *
 * Usage:
 */
const int N = 115, M = 10004;

DirectedMST<N, M> DMST;

int n, m, r;

vector<Edge> E;

int main() {
  scanf("%d%d%d", &n, &m, &r);
  FOR(i, 1, m) {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    E.push_back(Edge(u, v, w));
  }
  auto Et = DMST.buildFrom(n, r, E);

  if (Et.empty()) {
    puts("-1");
  } else {
    int ans = 0;
    for (auto e : Et) ans += e.w;
    printf("%d\n", ans);
  }

  return 0;
}
// Algorithm
struct Edge {
  int u, v, w, ow;
  Edge(int _u, int _v, int _w) { u = _u, v = _v, w = ow = _w; }
  void reset() { w = ow; }
};

template <const int N, const int M> struct DirectedMST {
  int nd[N], tnd[N], fa[N], pre[N], In[N], Time[M], totTime, onCir[N], totCir;
  vector<int> toggle[M];

  int get(int u) { return fa[u] == u ? u : fa[u] = get(fa[u]); }
  int getNode(int u) { return nd[u] == u ? u : nd[u] = getNode(nd[u]); }

  bool work(const int n, const int root, vector<Edge> &ve) {
    bool flag = false;
    fill(In, In + n + 1, -1), fill(onCir, onCir + n + 1, 0);
    totCir = 0;

```

```
56      for (unsigned i = 0; i < ve.size(); i++) {
57        int u = getNode(ve[i].u), v = getNode(ve[i].v);
58        if (u == v) continue;
59        if (In[v] == -1 || ve[In[v]].w > ve[i].w) In[v] = i;
60      }
61
62      FOR(i, 1, n) fa[i] = i;
63
64      FOR(i, 1, n) if (i != root && getNode(i) == i) {
65        if (In[i] == -1) return false;
66        Edge e = ve[In[i]];
67        int u = getNode(e.u), v = getNode(e.v);
68        if (u == v) continue;
69        if (get(u) == get(v)) {
70          ++totCir;
71          for (int z = u; z != -1; z = z == v ? -1 : getNode(ve[In[z]].u))
72            onCir[z] = totCir, tnd[z] = v, Time[In[z]] = ++totTime; // assert(z);
73          flag = true;
74        } else {
75          fa[get(u)] = get(v);
76        }
77      }
78
79      for (unsigned i = 0; i < ve.size(); i++) {
80        auto &e = ve[i];
81        int u = getNode(e.u), v = getNode(e.v);
82        if (u == v) continue;
83        if (onCir[v] && onCir[v] == onCir[u]) continue;
84        if (onCir[v]) toggle[i].push_back(In[v]), e.w -= ve[In[v]].w;
85      }
86
87      FOR(i, 1, n) if (onCir[i]) nd[i] = tnd[i]; // assert(getNode(i) == i);
88
89      return flag;
90    }
91    vector<Edge> buildFrom(int n, int root, vector<Edge> ve) {
92      assert(!ve.empty());
93      vector<Edge> vt;
94      FOR(i, 1, n) nd[i] = i;
95      fill(Time, Time + ve.size() + 1, 0);
96      totTime = 0;
97
98      while (work(n, root, ve))
99        ;
100
101     FOR(i, 1, n) if (getNode(i) == i && i != root) {
102       if (In[i] == -1) return vt; // empty
103       Time[In[i]] = ++totTime;
104     }
105     vector<int> SortByTime(totTime + 1, -1);
106     for (unsigned i = 0; i < ve.size(); i++)
107       if (Time[i]) SortByTime[Time[i]] = i;
108
```

```
    ROF(i, totTime, 1) {
      int x = SortByTime[i];
      if (Time[x])
        for (int y : toggle[x]) Time[y] = 0;
    }

    for (unsigned i = 0; i < ve.size(); i++) {
      ve[i].reset();
      if (Time[i]) vt.push_back(ve[i]);
    } // assert(vt.size() == n - 1);
    return vt;
  }
};
```

code/dmst.cpp

5.4 KM 算法

```
// by Yao
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
#define ROF(i, a, b) for (int i = (a); i >= (b); --i)

const int N = 404;
const long long INF = 1e18;

int n, nl, nr, m;
long long w[N][N], hl[N], hr[N], slack[N];
int pre[N], toR[N], toL[N], q[N], ql, qr;
bool vl[N], vr[N];

bool push(int v) { // v in L。找到🄵广路则返回 true
  vl[v] = true;
  if (toR[v]) { // 存在与 v 匹配的点，就入队
    q[++qr] = toR[v];
    vr[toR[v]] = true;
    return false;
  }
  while (v) { // 找到🄵广路，就🄵广
    toR[v] = pre[v];
    swap(v, toL[pre[v]]);
  }
  return true;
}
void bfs(int s) { // s in R
  fill(vl + 1, vl + n + 1, false);
  fill(vr + 1, vr + n + 1, false);
  fill(slack + 1, slack + n + 1, INF);
  ql = qr = 0, q[++qr] = s, vr[s] = true;
  while (1) {
```

```
    while (ql < qr) {
      int u = q[++ql]; // u in R
      FOR(v, 1, n) if (!vl[v]) {
        long long d = hl[v] + hr[u] - w[v][u];
        if (d == 0) { // 是相等子图里的点
          pre[v] = u;
          if (push(v)) return;
        } else if (slack[v] >= d) {
          slack[v] = d, pre[v] = u;
        }
      }
    }
    long long d = INF;
    FOR(i, 1, n) if (!vl[i] && d > slack[i]) d = slack[i];
    FOR(i, 1, n) {
      if (vl[i]) hl[i] += d;
      else slack[i] -= d;
      if (vr[i]) hr[i] -= d;
    }
    FOR(i, 1, n) if (!vl[i] && !slack[i] && push(i)) return;
  }
}
void KM() {
  FOR(i, 1, n) {
    hl[i] = *max_element(w[i] + 1, w[i] + n + 1);
    hr[i] = 0;
  }
  fill(slack + 1, slack + n + 1, INF);
  FOR(i, 1, n) { bfs(i); }
}
int main() {
  scanf("%d%d%d", &nl, &nr, &m);
  FOR(i, 1, m) {
    int u, v, ww;
    scanf("%d%d%d", &u, &v, &ww);
    w[u][v] = ww; // w[u,v] 和 w[v,u] 不是一个东西
  }
  n = max(nl, nr);
  KM();
  long long ans = 0;
  FOR(i, 1, n) ans += hl[i] + hr[i];
  printf("%lld\n", ans);
  // 不能用 toR[i] > nr 来判断是否有匹配，因为 0 权边是不存在的
  FOR(i, 1, nl) printf("%d%c", w[i][toR[i]] == 0 ? 0 : toR[i], " \n"[i == nl]);
  return 0;
}
```

code/km.cpp

5.5 Tarjan SCC

```
struct qxx {
```

```cpp
  int nex, t;
} e[M];
int h[N], le;
void add_path(int f, int t) { e[++le] = (qxx){h[f], t}, h[f] = le; }
#define FORe(i, u, v) for (int i = h[u], v; v = e[i].t, i; i = e[i].nex)

int dfn[N], low[N], totdfn;
int s[N], tp;
bool in_s[N];
int scc[N], totscc; //每个点所属SCC标号
int sz[N]; //每个SCC的大小
void dfs(int u) {
  low[u] = dfn[u] = ++totdfn;
  s[++tp] = u, in_s[u] = 1;
  FORe(i, u, v) {
    if (!dfn[v]) dfs(v), low[u] = min(low[u], low[v]);
    else if (in_s[v]) low[u] = min(low[u], dfn[v]);
  }
  if (dfn[u] == low[u]) {
    ++totscc;
    while (s[tp] != u) scc[s[tp]] = totscc, in_s[s[tp]] = 0, --tp, ++sz[totscc];
    scc[s[tp]] = totscc, in_s[s[tp]] = 0, --tp, ++sz[totscc];
  }
}
// FOR(i,1,n)if(!dfn[i])dfs(i);
```

code/tarjan_scc.cpp

6 数据结构

6.1 左偏树

```cpp
int lc[SZ], rc[SZ], val[SZ], dep[SZ], tot;
int new_node(int v) {
  ++tot, val[tot] = v, lc[tot] = rc[tot] = 0, dep[tot] = 0;
  return tot;
}
int merge(int x, int y) {
  if (!x || !y) return x + y;
  if (val[x] > val[y]) swap(x, y);
  rc[x] = merge(rc[x], y);
  if (dep[lc[x]] < dep[rc[x]]) swap(lc[x], rc[x]);
  dep[x] = dep[rc[x]] + 1;
  return x;
}
int getmin(int x) { return val[x]; }
int pop(int x) {
  val[x] = -1;
  return merge(lc[x], rc[x]);
```

```
}
```

code/leftist.cpp

6.2 Splay

```
const int SZ = 1e5 + 5;

struct Splay {
  int tot, ch[SZ][2], key[SZ], sz[SZ], pa[SZ], root;
  int rv[SZ];
  int new_node(int v) {
    ++tot, ch[tot][0] = ch[tot][1] = pa[tot] = 0, key[tot] = v;
    sz[tot] = rv[tot] = 0;
    return tot;
  }
  Splay() {
    tot = 0;
    new_node(-INF), new_node(INF);
    pa[1] = 2, ch[2][0] = 1, root = 2;
  }
  bool get(int u) { return ch[pa[u]][1] == u; }
  void pushup(int u) { sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1; }
  void node_rv(int u) {
    if (!u) return;
    swap(ch[u][0], ch[u][1]);
    rv[u] ^= 1;
  }
  void pushdown(int u) {
    if (rv[u]) {
      node_rv(ch[u][0]);
      node_rv(ch[u][1]);
      rv[u] = 0;
    }
  }
  void rotate(int u) {
    int p = pa[u], pp = pa[p], gu = get(u);
    pushdown(u);
    ch[pp][get(p)] = u, pa[u] = pp;
    ch[p][gu] = ch[u][gu ^ 1], pa[ch[u][gu ^ 1]] = p;
    ch[u][gu ^ 1] = p, pa[p] = u;
    pushup(p), pushup(u);
  }
  void splay(int u, int v) {
    while (pa[u] != v) {
      int p = pa[u];
      if (pa[p] != v) rotate(get(u) == get(p) ? p : u);
      rotate(u);
    }
    if (!v) root = u;
  }
  bool find(int v) { // return if v exists. Based on key
```

```
    if (!root) return 0;
    int u = root;
    while (key[u] != v && ch[u][key[u] < v]) pushdown(u), u = ch[u][key[u] < v];
    splay(u, 0);
    return key[root] == v;
  }
  void insert(int v) {
    if (!root) return root = new_node(v), void();
    int u = root, nu = new_node(v);
    while (ch[u][key[u] < v]) pushdown(u), u = ch[u][key[u] < v];
    ch[u][key[u] < v] = nu, pa[nu] = u;
    splay(nu, 0);
  }
  int kth(int rk) {
    ++rk;
    if (!root) return 0;
    int u = root;
    while (sz[ch[u][0]] + 1 != rk) {
      pushdown(u);
      if (rk <= sz[ch[u][0]]) u = ch[u][0];
      else rk -= sz[ch[u][0]] + 1, u = ch[u][1];
    }
    splay(u, 0);
    return u;
  }
};
```

code/splay.cpp

6.3 非旋转 Treap

```
#include <cstdio>
using namespace std;
const int N = 1e5 + 5, SZ = N;
int n;

struct Treap {
  int seed = 1, root, tot;
  int ch[SZ][2];
  int val[SZ], rnd[SZ], sz[SZ]; // sz:子树大小

  int rrand() { return seed = seed * 482711; }
  void pushup(int u) { sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1; }
  void split(int u, int key, int &x, int &y) {
    if (!u) x = y = 0;
    else {
      if (val[u] <= key) x = u, split(ch[u][1], key, ch[u][1], y);
      else y = u, split(ch[u][0], key, x, ch[u][0]);
      pushup(u);
    }
  }
  int merge(int x, int y) { // x<y
```

```cpp
    if (!x || !y) return x + y; //返回x，y或0
    if (rnd[x] < rnd[y]) return ch[x][1] = merge(ch[x][1], y), pushup(x), x;
    else return ch[y][0] = merge(x, ch[y][0]), pushup(y), y;
  }
  void insert(int v) { //插入v
    int x, y, u = ++tot;
    val[u] = v, sz[u] = 1, rnd[u] = rrand();
    split(root, v, x, y);
    root = merge(merge(x, u), y);
  }
  void del(int v) {
    int x, y, z;
    split(root, v - 1, x, y); //所有的v就被分在y中
    split(y, v, y, z); //所有的v仍被分在y中
    if (!y) return; //不存在v这个权值
    y = merge(ch[y][0], ch[y][1]); //根节点的不要了
    root = merge(x, merge(y, z));
  }
  int rank(int v) { //即相同的数中，第一个数的排名
    int x, y, res;
    split(root, v - 1, x, y);
    res = sz[x] + 1, root = merge(x, y);
    return res;
  }
  int kth(int k) { //查询排名为k的数
    int u = root;
    while (k != sz[ch[u][0]] + 1) {
      if (k <= sz[ch[u][0]]) u = ch[u][0];
      else k -= sz[ch[u][0]] + 1, u = ch[u][1];
    }
    return val[u];
  }
  int pre(int v) { return kth(rank(v) - 1); } //严格前驱
  int suc(int v) { return kth(rank(v + 1)); } //严格后继
} treap;

int main() {
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) {
    int opt, x;
    scanf("%d%d", &opt, &x);
    if (opt == 1) treap.insert(x);
    else if (opt == 2) treap.del(x);
    else if (opt == 3) printf("%d\n", treap.rank(x));
    else if (opt == 4) printf("%d\n", treap.kth(x));
    else if (opt == 5) printf("%d\n", treap.pre(x));
    else printf("%d\n", treap.suc(x));
  }
  return 0;
}
```

code/FHQ_treap.cpp

6.4 LCT

```cpp
const int SZ = 1e6 + 6;

int tot, ch[SZ][2], f[SZ], val[SZ], sz[SZ], rv[SZ];
int sxor[SZ];

int new_node(int v) {
  ++tot, ch[tot][0] = ch[tot][1] = f[tot] = rv[tot] = 0;
  sz[tot] = 1, val[tot] = v, sxor[tot] = v;
  return tot;
}
void pushup(int u) {
  sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1;
  sxor[u] = sxor[ch[u][0]] ^ sxor[ch[u][1]] ^ val[u];
}
void noderv(int u) {
  if (u) rv[u] ^= 1;
}
void nodeassign(int u, int v) { val[u] = v, pushup(u); }
void pushdown(int u) {
  if (rv[u]) swap(ch[u][0], ch[u][1]), noderv(ch[u][0]), noderv(ch[u][1]), rv[u] = 0;
}
bool isroot(int u) { return ch[f[u]][0] != u && ch[f[u]][1] != u; }
bool get(int u) { return ch[f[u]][1] == u; }
void rotate(int u) {
  int p = f[u], pp = f[p], k;
  pushdown(p), pushdown(u), k = get(u); // k的赋值必须在pushdown后!
  if (!isroot(p)) ch[pp][get(p)] = u; //!!!
  ch[p][k] = ch[u][!k], f[ch[u][!k]] = p;
  ch[u][!k] = p, f[p] = u, f[u] = pp;
  pushup(p), pushup(u);
}
void splay(int u) {
  pushdown(u);
  for (int p; p = f[u], !isroot(u); rotate(u))
    if (!isroot(p)) rotate(get(p) == get(u) ? p : u);
}
void access(int u) {
  for (int p = 0; u; p = u, u = f[u]) splay(u), ch[u][1] = p, pushup(u);
}
void makeroot(int u) { access(u), splay(u), noderv(u); }
bool check_link(int x, int y) {
  makeroot(x), access(y), splay(x), splay(y);
  return !(isroot(x) && isroot(y));
}
void link(int x, int y) { makeroot(x), f[x] = y; }
bool check_edge(int x, int y) {
  if (!check_link(x, y)) return 0;
  makeroot(x), access(y), splay(y);
  if (ch[y][0] != x || ch[x][1]) return 0;
  return 1;
}
```

```
void cut(int x, int y) {
  makeroot(x), access(y), splay(y), ch[y][0] = f[x] = 0, pushup(y);
}
void assign(int x, int y) { splay(x), nodeassign(x, y); }
int query(int x, int y) { return makeroot(x), access(y), splay(y), sxor[y]; }
/*
 * 模板：Luogu3690
 * new_node: 新建权值为 v 的结点
 * pushup: 信息更新
 * pushdown: 标记下传，主要是翻转标记
 * noderv: 对某一个结点施加标记。
 *     LCT的标记不同于线段树，必须在下传的时候再更新当前结点的信息。不然
 *     get 的时候会出锅
 * nodeassign: 模板题需要
 * isroot: 是否是所在Splay的根
 * get: 是Splay上左儿子还是右儿子
 * print: 调试函数
 * rotate: 双旋，注意与Splay的双旋不同，要判f[u]是不是root，不然f[f[u]]的
 *     儿子不能乱赋值
 * splay: 把当前结点旋转到当前Splay的根结点，要用到isroot函数。一开始
 *     先pushdown。
 * access: 把当前结点到根的路径连成一个Splay，注意这个Splay只包含当前结点
 *     到根这段路径上的点，不包括当前结点子树的那一段（非到叶结点的树链）
 *     access完之后这个点不一定是所在splay的根，需要手动splay一下
 * makeroot: 把当前结点变成原树的根，这个结点也会顺便变成所在Splay的根。
 * check_link: 判断两个点是否连通。
 * link: 连接两个不连通的点
 * check_edge: 判断两个点是否直连通（有没有边）
 * cut: 删掉 (x,y) 的边。
 * assign: 模板题需要
 * query: 模板题需要
 * 提醒：在修改了ch指针后要考虑是否pushup
 */
```

code/lct.cpp

6.5 点分治

```
const int N = 1e5 + 5;
struct qxx {
  int nex, t, v;
} e[N * 2];
int h[N], le = 1;
void add_path(int f, int t, int v) { e[++le] = {h[f], t, v}, h[f] = le; }
int n;
bool Cut[N];
int sz[N], sm[N];
void Size(int u, int p) {
  sz[u] = 1, sm[u] = 0;
  for (int i = h[u]; i; i = e[i].nex) {
    int v = e[i].t;
    if (!Cut[v] && v != p) { Size(v, u), sz[u] += sz[v], sm[u] = max(sm[u], sz[v]); }
```

```
  }
}
int Core(int u, int p, int T) {
  int res = u, mx = max(T - sz[u], sm[u]);
  for (int i = h[u]; i; i = e[i].nex) {
    int v = e[i].t;
    if (!Cut[v] && v != p) {
      int x = Core(v, u, T), y = max(T - sz[x], sm[x]);
      if (y < mx) res = x, mx = y;
    }
  }
  return res;
}
void Solve(int u, int p) {
  Size(u, p);
  int core = Core(u, p, sz[u]);
  // do sth ...
  Cut[core] = 1;
  for (int i = h[core]; i; i = e[i].nex) {
    int v = e[i].t;
    if (!Cut[v]) Solve(v, core);
  }
}
```

code/centroid_decomposition.cpp

6.6 笛卡尔树

```
int n, a[N], root;
int s[N], tp, lc[N], rc[N];

void al(int u, int v) { lc[u] = v; }
void ar(int u, int v) { rc[u] = v; }

void build() {
  FOR(i, 1, n) {
    while (tp > 1 && a[s[tp - 1]] <= a[i]) ar(s[tp - 1], s[tp]), --tp;
    if (tp > 0 && a[s[tp]] <= a[i]) al(i, s[tp]), --tp;
    s[++tp] = i;
  }
  while (tp > 1) ar(s[tp - 1], s[tp]), --tp;
  root = s[1];
}
```

code/cartesian_tree.cpp

6.7 树链剖分

```
vector<int> g[N];
int sz[N], dep[N], big[N], top[N], fa[N], totdfn;
```

```cpp
int b[N], id[N];
void dfs1(int u, int p) {
  sz[u] = 1, dep[u] = dep[p] + 1, big[u] = 0, fa[u] = p;
  for (int v : g[u])
    if (v != p)
      dfs1(v, u), sz[u] += sz[v], (!big[u] || sz[big[u]] < sz[v]) ? big[u] = v : 0;
}
void dfs2(int u, int p, int tp) {
  top[u] = tp, b[++totdfn] = u, id[u] = totdfn;
  if (big[u]) dfs2(big[u], u, tp);
  for (int v : g[u])
    if (v != p && v != big[u]) dfs2(v, u, v);
}
int lca(int u, int v) {
  while (top[u] != top[v])
    dep[top[u]] < dep[top[v]] ? swap(u, v), 0 : 0, u = fa[top[u]];
  return dep[u] < dep[v] ? u : v;
}
int kthanc(int u, int k) {
  while (dep[u] - k < dep[top[u]]) k -= dep[u] - dep[top[u]] + 1, u = fa[top[u]];
  return b[id[top[u]] + dep[u] - dep[top[u]] - k];
}
```

code/heavy_light_decomposition.cpp

6.8 长链剖分 & K 级祖先

```cpp
vector<int> g[N];
int dep[N], lon[N], hei[N], fa[N][20], f0[N], htop[N];
vector<int> up[N], dn[N];
void dfs1(int u, int p) {
  dep[u] = dep[p] + 1, lon[u] = 0, fa[u][0] = p, f0[u] = p;
  for (int j = 1; fa[u][j - 1] && j < 20; j++) fa[u][j] = fa[fa[u][j - 1]][j - 1];
  for (int v : g[u])
    if (v != p) dfs1(v, u), (!lon[u] || hei[lon[u]] < hei[v]) ? lon[u] = v : 0;
  hei[u] = lon[u] ? hei[lon[u]] + 1 : 1;
}
void dfs3(int u, int p, int htp) {
  htop[u] = htp;
  if (u == htp) {
    for (int v = u; v; v = lon[v]) dn[u].pb(v);
    for (int v = u; v && up[u].size() < dn[u].size(); v = f0[v]) up[u].pb(v);
  }
  if (lon[u]) dfs3(lon[u], u, htp);
  for (int v : g[u])
    if (v != p && v != lon[u]) dfs3(v, u, v);
}
int highbit[N];
int kthanc(int u, int k) {
  if (dep[u] <= k) return 0;
  if (k == 0) return u;
  u = fa[u][highbit[k]], k -= 1 << highbit[k];
```

```
  int d = dep[u] - k - dep[htop[u]];
  if (d >= 0) return dn[htop[u]][d];
  else return up[htop[u]][-d];
}
// FOR(i,1,n)highbit[i]=i==1?0:highbit[i>>1]+1;
```

code/kth_ancestor.cpp

6.9 虚树

```
const int N = 5e5 + 5;

int n;
vector<int> g[N];
int dep[N], fa[N][20];
int dfn[N], totdfn;
void dfs(int u, int p) {
  dep[u] = dep[p] + 1, fa[u][0] = p, dfn[u] = ++totdfn;
  FOR(j, 1, 19) {
    fa[u][j] = fa[fa[u][j - 1]][j - 1];
    if (fa[u][j] == 0) break;
  }
  for (int v : g[u])
    if (v != p) dfs(v, u);
}
int lca(int x, int y) {
  if (dep[x] < dep[y]) swap(x, y);
  ROF(j, 19, 0) if (dep[x] - (1 << j) >= dep[y]) x = fa[x][j];
  if (x == y) return x;
  ROF(j, 19, 0) if (fa[x][j] != fa[y][j]) x = fa[x][j], y = fa[y][j];
  return fa[x][0];
}
int distance(int x, int y) { return dep[x] + dep[y] - dep[lca(x, y)] * 2; }

typedef pair<int, int> pii;
vector<pii> vt[N];
void VT_addpath(int u, int v, int w) {
  vt[u].pb({v, w});
  vt[v].pb({u, w});
}

int s[N], tp;
void build_VT(vector<int> &V) { //建虚树并把点集更新到V里
  for (int u : V) vt[u].clear();
  sort(V.begin(), V.end(), [](int x, int y) { return dfn[x] < dfn[y]; });
  V.resize(unique(V.begin(), V.end()) - V.begin());
  s[tp = 1] = lca(V.front(), V.back());
  vector<int> Vt;
  for (int u : V) {
    int z = lca(s[tp], u);
    if (u == z) continue;
    while (tp > 1 && dep[s[tp - 1]] >= dep[z])
```

```
      VT_addpath(s[tp], s[tp - 1], distance(s[tp], s[tp - 1])), --tp;
    if (s[tp] != z) VT_addpath(s[tp], z, distance(s[tp], z)), s[tp] = z, Vt.pb(z);
    s[++tp] = u;
  }
  while (tp > 1) VT_addpath(s[tp], s[tp - 1], distance(s[tp], s[tp - 1])), --tp;
  V.pb(s[1]);
  V.insert(V.end(), Vt.begin(), Vt.end());
  sort(V.begin(), V.end(), [](int x, int y) { return dfn[x] < dfn[y]; });
  V.resize(unique(V.begin(), V.end()) - V.begin());
}
```

code/virtual_tree.cpp

7 其他

7.1 计算几何

```
/**
 * Computing Geometry Library
 * ref1: https://onlinejudge.u-aizu.ac.jp/courses/library/4/CGL/all
 * ref2: https://darkbzoj.tk/problem/2178
 * @author Sshwy
 */
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
#define FOR(i, a, b) for (int i = (int)(a); i <= (int)(b); ++i)
#define ROF(i, a, b) for (int i = (int)(a); i >= (int)(b); --i)

namespace cg {
  typedef long double vtyp;
  const vtyp eps = 1e-9;
  const vtyp PI = 3.1415926535897932626;

  bool isZero(vtyp x) { return -eps < x && x < eps; }
  bool eq(vtyp x, vtyp y) { return isZero(x - y); }
  bool neq(vtyp x, vtyp y) { return !eq(x, y); }
  bool lt(vtyp x, vtyp y) { return !eq(x, y) && x < y; }
  bool gt(vtyp x, vtyp y) { return !eq(x, y) && x > y; }
  bool le(vtyp x, vtyp y) { return eq(x, y) || x < y; }
  bool ge(vtyp x, vtyp y) { return eq(x, y) || x > y; }

  struct vec {
    vtyp x, y;
    vec() { x = y = 0; }
    vec(vtyp _x, vtyp _y) { x = _x, y = _y; }

    vec operator+(const vec V) const { return vec(x + V.x, y + V.y); }
    vec operator-() const { return vec(-x, -y); }
    vec operator-(const vec V) const { return *this + (-V); }
    vec operator*(const vtyp a) const { return vec(x * a, y * a); }
```

```cpp
    friend vec operator*(const vtyp a, const vec v) { return v * a; }
    vec operator/(const vtyp a) const { return vec(x / a, y / a); }
    operator bool() const { return !(isZero(x) && isZero(y)); }
    bool operator==(const vec V) const { return bool(*this - V) == 0; }
    bool operator!=(const vec V) const { return bool(*this - V) != 0; }
    bool operator<(const vec V) const { return x == V.x ? y < V.y : x < V.x; }
    bool operator>(const vec V) const { return x == V.x ? y > V.y : x > V.x; }

    vtyp length() const { return sqrt(x * x + y * y); }
    /**
     * 方向角，单位 rad
     */
    vtyp ang() const { return atan2(y, x); }
    /**
     * 方向向量
     * @return 0向量或者一个单位向量
     */
    vec dir() const {
      if (*this) {
        vtyp len = length();
        // vtyp ang = atan2(y,x); return vec(cos(ang), sin(ang));
        return vec(x / len, y / len);
      } else return vec(0, 0);
    }
    // void read(){ scanf("%Lf%Lf",&x,&y); }
  };
  typedef vec point;

  vec r90_clockwise(const vec v) { // 顺时针旋转 90 度
    return vec(v.y, -v.x);
  }

  struct line {
    point p1, p2;
    line(point _p1, point _p2) { p1 = _p1, p2 = _p2; }
    line operator+(point p) { return line(p1 + p, p2 + p); } // shift
    line operator-(point p) { return line(p1 - p, p2 - p); }
    vec dir() const { return (p2 - p1).dir(); }
  };
  typedef line segment;

  istream &operator>>(istream &in, vec &v) { return in >> v.x >> v.y, in; }
  ifstream &operator>>(ifstream &in, vec &v) { return in >> v.x >> v.y, in; }
  ostream &operator<<(ostream &out, const vec &v) {
    return out << v.x << " " << v.y, out;
  }
  ofstream &operator<<(ofstream &out, const vec &v) {
    return out << v.x << " " << v.y, out;
  }
  /**
   * 点积
   * a dot b == |a||b|cos theta
   */
```

```
88  vtyp dot(const vec a, const vec b) { return a.x * b.x + a.y * b.y; }
89  /**
90   * 叉积
91   * 两个向量围成的有向面积
92   */
93  vtyp det(const vec a, const vec b) { return a.x * b.y - a.y * b.x; }
94  /**
95   * 向量夹角
96   * @return 一个[0, PI) 内的数表示角度, 单位 rad
97   */
98  vtyp angle(vec a, vec b) {
99    if (det(a, b) < 0) swap(a, b);
100   vtyp res = b.ang() - a.ang();
101   if (res < 0) res += 2 * PI;
102   return res;
103 }
104
105 /**
106  * 投影
107  * @param L 直线
108  * @param p 要求投影的点
109  * @return p 在 L 上的投影坐标 (即垂足)
110  */
111 point projection(line L, point p) {
112   vec d = L.p2 - L.p1;
113   return L.p1 + (dot(d, p - L.p1) / d.length()) * d.dir();
114 }
115 /**
116  * 对称点
117  * @param L 直线
118  * @param p 点
119  * @return p 关于直线 L 的对称点
120  */
121 point reflection(line L, point p) {
122   point o = projection(L, p);
123   return vtyp(2) * (o - p) + p;
124 }
125
126 /**
127  * 判断向量是否平行
128  */
129 bool parallel(vec a, vec b) { return isZero(det(a, b)); }
130 /**
131  * 判断直线是否平行
132  */
133 bool parallel(line a, line b) { return parallel(a.p2 - a.p1, b.p2 - b.p1); }
134 /**
135  * 判断向量是否垂直
136  */
137 bool orthogonal(vec a, vec b) { return isZero(dot(a, b)); }
138 /**
139  * 判断直线是否垂直
140  */
```

```
141   bool orthogonal(line a, line b) { return orthogonal(a.p2 - a.p1, b.p2 - b.p1); }
142   /**
143    * 判断点 p 是否在直线L上
144    */
145   bool online(line L, point p) { return parallel(L.p2 - L.p1, p - L.p1); }
146   /**
147    * 判断两直线是否重合
148    */
149   bool coincident(line a, line b) { return online(a, b.p1) && online(a, b.p2); }
150   /**
151    * 判断点 p 是否与有向线段共线且在反向延长线上
152    */
153   bool online_back(segment sl, point p) {
154     vec a = sl.p2 - sl.p1, b = p - sl.p1;
155     return parallel(a, b) && lt(dot(a, b), 0);
156   }
157   /**
158    * 判断点 p 是否与有向线段共线且在正向延长线上
159    */
160   bool online_front(segment sl, point p) {
161     vec a = sl.p1 - sl.p2, b = p - sl.p2; // 倒过来
162     return parallel(a, b) && lt(dot(a, b), 0);
163   }
164   /**
165    * 判断点 p 是否在线段上（含端点）
166    */
167   bool on_segment(segment sl, point p) {
168     return online(sl, p) && !online_back(sl, p) && !online_front(sl, p);
169   }
170   /**
171    * 两条直线的交点
172    * 需确保两条直线不平行
173    */
174   point intersection(line a, line b) {
175     assert(!parallel(a, b));
176     vtyp x = det(a.p1 - b.p1, b.p2 - b.p1);
177     vtyp y = det(b.p2 - b.p1, a.p2 - b.p1);
178     return a.p1 + (a.p2 - a.p1) * x / (x + y);
179   }
180   /**
181    * 判断两个线段是否相交（含边界）
182    */
183   bool check_segment_intersection(segment a, segment b) {
184     if (cg::coincident(a, b)) {
185       if (on_segment(a, b.p1) || on_segment(a, b.p2) || on_segment(b, a.p1) ||
186           on_segment(b, a.p2))
187         return true;
188       else return false;
189     } else if (cg::parallel(a, b)) {
190       return false;
191     } else {
192       point o = cg::intersection(a, b);
193       if (cg::on_segment(a, o) && cg::on_segment(b, o)) return true;
```

```
      else return false;
    }
  }
  /**
   * 两个点的距离
   */
  vtyp distance(point a, point b) { return (b - a).length(); }
  /**
   * 点到直线的距离
   */
  vtyp distance(line L, point p) { return (p - projection(L, p)).length(); }
  /**
   * 两个线段的距离
   */
  vtyp distance(segment a, segment b) {
    if (check_segment_intersection(a, b)) return 0;
    vtyp res = distance(a.p1, b.p1);
    res = min(res, distance(a.p1, b.p2));
    res = min(res, distance(a.p2, b.p1));
    res = min(res, distance(a.p2, b.p2));
    point o;
    if (o = projection(b, a.p1), on_segment(b, o)) res = min(res, distance(a.p1, o));
    if (o = projection(b, a.p2), on_segment(b, o)) res = min(res, distance(a.p2, o));
    if (o = projection(a, b.p1), on_segment(a, o)) res = min(res, distance(b.p1, o));
    if (o = projection(a, b.p2), on_segment(a, o)) res = min(res, distance(b.p2, o));
    return res;
  }
  /**
   * 求简单多边形面积
   * @param g 多边形顶点集
   */
  vtyp area(const vector<point> &g) {
    vtyp res = 0;
    for (unsigned i = 0; i < g.size(); i++) {
      res += det(g[i], g[(i + 1) % g.size()]);
    }
    res /= 2;
    return abs(res);
  }
  /**
   * 判断是否是凸包
   * @param g 多边形顶点集
   */
  bool is_convex(const vector<point> &g) {
    if (g.size() < 3) return true;
    int flag = 0;
    for (unsigned i = 0; i < g.size(); i++) {
      int j = (i + 1) % g.size(), k = (i + 2) % g.size();
      vtyp sig = det(g[j] - g[i], g[k] - g[j]);
      if (lt(sig, 0)) {
        if (flag == 1) return false;
        else flag = -1;
      }
```

```
247       if (gt(sig, 0)) {
248         if (flag == -1) return false;
249         else flag = 1;
250       }
251     }
252     return true;
253   }
254   /**
255    * 求凸包
256    * @param g 多边形顶点集
257    */
258   vector<point> convex(vector<point> g) {
259     sort(g.begin(), g.end());
260     if (g.size() < 3) return g;
261
262     vector<bool> vis(g.size(), false);
263     vector<int> s(g.size() + 1, 0);
264     int ls = 0;
265
266     for (unsigned i = 0; i < g.size(); i++) {
267       while (ls > 1 && lt(det(g[s[ls - 1]] - g[s[ls - 2]], g[i] - g[s[ls - 1]]), 0))
268         --ls;
269       s[ls] = i, ++ls;
270     }
271     FOR(i, 0, ls - 1) vis[s[i]] = true;
272     vis[0] = false;
273     for (int i = g.size() - 1; i >= 0; i--)
274       if (!vis[i]) {
275         while (ls > 1 && lt(det(g[s[ls - 1]] - g[s[ls - 2]], g[i] - g[s[ls - 1]]), 0))
276           --ls;
277         s[ls] = i, ++ls;
278       }
279     assert(s[0] == s[ls - 1]);
280
281     vector<point> cvx;
282     FOR(i, 0, ls - 2) cvx.pb(g[s[i]]);
283     return cvx;
284   }
285   /**
286    * 求点集的最远点对距离（正确性还不太懂，也许有锅）
287    * @param v 点集
288    */
289   vtyp diameter(const vector<point> &v) {
290     vector<point> g = convex(v);
291     vtyp dist = 0;
292     unsigned pos = 0;
293     for (unsigned i = 0; i < g.size(); i++) {
294       while (pos + 1 < g.size() && distance(g[i], g[pos]) < distance(g[i], g[pos + 1]))
295         ++pos;
296       dist = max(dist, distance(g[i], g[pos]));
297     }
298     return dist;
299   }
```

```cpp
/**
 * 判断点p与多边形的包含关系
 * @param g 多边形顶点集
 * @return 0 表示在多边形外，1 表示在边上，2表示在多边形内
 */
int polygon_point_containment(vector<point> g, point p) {
  line L(vec(p.x - 1, p.y), p); // 水平方向的射线
  int cnt = 0;
  for (unsigned i = 0; i < g.size(); i++) {
    int j = (i + 1) % g.size();
    line e(g[i], g[j]);
    if (on_segment(e, p)) return 1;
    if (parallel(L, e)) {
      // do nothing.
    } else if (online_front(L, g[i])) {
      if (g[i].y > g[j].y) ++cnt;
    } else if (online_front(L, g[j])) {
      if (g[j].y > g[i].y) ++cnt;
    } else {
      point o = intersection(L, e);
      if (on_segment(e, o) && online_front(L, o)) ++cnt;
    }
  }
  if (cnt % 2) return 2;
  return 0;
}

struct circle {
  point o;
  vtyp r;
  circle() { r = 0; }
  circle(point _o, vtyp _r) { o = _o, r = _r; }
};
/**
 * 判断两个圆的位置关系（切线数量）
 * @param a 第一个圆
 * @param b 第二个圆
 * @return 0 表示包含，1 表示内切，2 表示相交，3 表示外切，4 表示相离
 */
int check_circle_intersection(circle a, circle b) {
  vtyp d = distance(a.o, b.o);
  if (gt(d, a.r + b.r)) return 4;
  if (eq(d, a.r + b.r)) return 3;
  if (gt(d, abs(a.r - b.r))) return 2;
  if (eq(d, abs(a.r - b.r))) return 1;
  return 0;
}
/**
 * 判断圆和点的位置关系
 * @return 0 表示包含，1 表示在圆上，2 表示在圆外
 */
int check_circle_point_containment(circle c, point p) {
  vtyp d = distance(c.o, p);
```

```
353     if (lt(d, c.r)) return 0;
354     if (eq(d, c.r)) return 1;
355     return 2;
356   }
357   /**
358    * 求三角形内切圆
359    * @param a 三角形第一个顶点
360    * @param b 三角形第二个顶点
361    * @param c 三角形第三个顶点
362    * @return 一个 circle 表示内切圆
363    */
364   circle incircle(point a, point b, point c) {
365     vtyp r =
366         abs(det(a - b, a - c)) / (distance(a, b) + distance(a, c) + distance(b, c));
367     line C(a, b), B(a, c);
368     vec shiftC = (c - projection(C, c)).dir() * r;
369     vec shiftB = (b - projection(B, b)).dir() * r;
370     point o = intersection(C + shiftC, B + shiftB);
371     return circle(o, r);
372   }
373   /**
374    * 求三角形外接圆
375    * @param a 三角形第一个顶点
376    * @param b 三角形第二个顶点
377    * @param c 三角形第三个顶点
378    * @return 一个 circle 表示外接圆
379    */
380   circle outcircle(point a, point b, point c) {
381     vec vc = r90_clockwise(a - b), vb = r90_clockwise(a - c);
382     point mc = (a + b) / vtyp(2), mb = (a + c) / vtyp(2);
383     point o = intersection(line(mc, mc + vc), line(mb, mb + vb));
384     vtyp r = (o - a).length();
385     return circle(o, r);
386   }
387   /**
388    * 圆点到直线的距离
389    */
390   vtyp distance(line L, circle c) { return distance(L, c.o); }
391   /**
392    * 求直线和圆的交点。如果相切那么返回两个相同的点
393    * 不会检查是否有交点。要求你提前判定
394    * @return 一个 pair 表示两个交点
395    */
396   pair<point, point> circle_line_intersection(line L, circle c) {
397     vtyp d = distance(L, c);
398     d = sqrt(max(vtyp(0), c.r * c.r - d * d));
399     vec shift = L.dir() * d;
400     point mid = projection(L, c.o);
401     return make_pair(mid - shift, mid + shift);
402   }
403   /**
404    * 求两圆的交点。如果相切那么返回两个相同的点
405    * 不会检查是否有交点。要求你提前判定
```

```
406     * @return 一个 pair 表示两个交点
407     */
408   pair<point, point> circle_intersection(circle c1, circle c2) {
409     assert(check_circle_intersection(c1, c2) > 0);
410     assert(check_circle_intersection(c1, c2) < 4);
411     vec oo = c2.o - c1.o, ooo = r90_clockwise(oo);
412     vtyp d = oo.length();
413     vtyp cosT = (c1.r * c1.r + d * d - c2.r * c2.r) / (2 * c1.r * d);
414     point p = c1.r * cosT * oo.dir() + c1.o;
415     vec shift = c1.r * sqrt(1 - cosT * cosT) * ooo.dir();
416     return make_pair(p + shift, p - shift);
417   }
418   /**
419    * 求圆外或圆上一点到圆的切线。
420    * 不会检查是否在圆外。要求你提前判定
421    * @return 一个 pair 表示两个切点，如果是圆上的点那么返回两个相同的点
422    */
423   pair<point, point> circle_point_tangent(circle c, point p) {
424     assert(check_circle_point_containment(c, p) != 0);
425     vec op = p - c.o, oop = r90_clockwise(op);
426     vtyp d = op.length();
427     vtyp x = c.r * c.r / d;
428     point mid = c.o + op.dir() * x;
429     vec shift = oop.dir() * sqrt(c.r * c.r - x * x);
430     return make_pair(mid + shift, mid - shift);
431   }
432   /**
433    * 两个大小不同的圆的外位似中心
434    * 若这两个圆不是包含关系，那么可以理解为是两条外公切线的交点
435    */
436   point circle_outer_homothetic_center(circle c1, circle c2) {
437     assert(neq(c1.r, c2.r));
438     if (gt(c1.r, c2.r)) swap(c1, c2);
439     point p = (c1.o - c2.o) * c1.r / (c2.r - c1.r) + c1.o;
440     return p;
441   }
442   /**
443    * 两个大小不同的圆的内位似中心
444    * 若这两个圆是相离或者外切关系，那么可以理解为是两条内公切线的交点
445    */
446   point circle_inner_homothetic_center(circle c1, circle c2) {
447     point p = (c2.o - c1.o) * c1.r / (c2.r + c1.r) + c1.o;
448     return p;
449   }
450   /**
451    * 求两圆外公切线
452    * 要求两圆不能是包含关系。
453    * 如果是内切的话那么返回两条相同的线（指line的两个点分别相同）
454    */
455   pair<line, line> circle_outer_common_tangent(circle c1, circle c2) {
456     assert(check_circle_intersection(c1, c2) != 0);
457     if (neq(c1.r, c2.r)) {
458       point p = circle_outer_homothetic_center(c1, c2);
```

```
459       auto pt = circle_point_tangent(c1, p);
460       if (pt.first == pt.second) {
461         vec oo = r90_clockwise(c1.o - c2.o);
462         line t(p + oo, p);
463         return make_pair(t, t);
464       } else {
465         return make_pair(line(p, pt.first), line(p, pt.second));
466       }
467     } else {
468       vec oo = c1.o - c2.o, ooo = r90_clockwise(oo);
469       vec shift = ooo.dir() * c1.r;
470       line t(c2.o, c1.o);
471       return make_pair(t + shift, t - shift);
472     }
473   }
474   /**
475    * 求两圆内公切线
476    * 要求两圆要么相离要么外切。
477    * 如果是外切的话那么返回两条相同的线（指line的两个点分别相同）
478    */
479   pair<line, line> circle_inner_common_tangent(circle c1, circle c2) {
480     assert(check_circle_intersection(c1, c2) >= 3);
481     point p = circle_inner_homothetic_center(c1, c2);
482     auto pt = circle_point_tangent(c1, p);
483     if (pt.first == pt.second) {
484       vec oo = r90_clockwise(c1.o - c2.o);
485       line t(p + oo, p);
486       return make_pair(t, t);
487     } else {
488       return make_pair(line(p, pt.first), line(p, pt.second));
489     }
490   }
491   /**
492    * 求两圆所有公切线，去重
493    */
494   vector<line> circle_common_tangent(circle c1, circle c2) {
495     vector<line> res;
496     int typ = check_circle_intersection(c1, c2);
497     if (typ > 0) {
498       auto pt = circle_outer_common_tangent(c1, c2);
499       res.pb(pt.first);
500       if (pt.first.p2 != pt.second.p2) res.pb(pt.second);
501     }
502     if (typ >= 3) {
503       auto pt = circle_inner_common_tangent(c1, c2);
504       res.pb(pt.first);
505       if (pt.first.p2 != pt.second.p2) res.pb(pt.second);
506     }
507     return res;
508   }
509   /**
510    * 求弓形面积
511    * @param r 半径
```

```cpp
   * @param angle 弓形所对的圆心角，单位 rad
   */
  vtyp circular_segment_area(vtyp r, vtyp angle) {
    return r * r * (angle - sin(angle)) / vtyp(2);
  }
  /**
   * 求两个圆交面积
   */
  vtyp circle_intersection_area(circle c1, circle c2) {
    vtyp ans = 0;
    auto typ = check_circle_intersection(c1, c2);
    if (typ <= 1) {
      ans += PI * min(c1.r, c2.r) * min(c1.r, c2.r);
    } else if (check_circle_intersection(c1, c2) < 3) {
      auto pt = circle_intersection(c1, c2);
      auto t1 = angle(pt.first - c1.o, pt.second - c1.o);
      auto t2 = angle(pt.first - c2.o, pt.second - c2.o);
      point p = intersection(line(c1.o, c2.o), line(pt.first, pt.second));
      if (online_front(segment(c1.o, c2.o), p)) {
        ans += circular_segment_area(c2.r, 2 * PI - t2);
      } else {
        ans += circular_segment_area(c2.r, t2);
      }
      if (online_front(segment(c2.o, c1.o), p)) {
        ans += circular_segment_area(c1.r, 2 * PI - t1);
      } else {
        ans += circular_segment_area(c1.r, t1);
      }
    }
    return ans;
  }
} // namespace cg
using cg::circle;
using cg::line;
using cg::point;
using cg::segment;

// int main(){
//   circle c1, c2;
//   cin >> c1.o >> c1.r >> c2.o >> c2.r;
//   auto ans = cg::circle_intersection_area(c1, c2);
//   cout << setiosflags(ios::fixed) << setprecision(9) << ans << endl;
//   return 0;
// }
```

code/geometry.cpp

7.2 欧拉序求 LCA

```cpp
const int N = 1e5 + 5, M = 2e5 + 5;

struct qxx {
```

```
  int nex, t;
};
qxx e[N * 2];
int h[N], le;
void add_path(int f, int t) { e[++le] = (qxx){h[f], t}, h[f] = le; }

int bg[N], ed[N], totime;
int dep[N];
int st[N * 2][20];

void dfs(int u, int p) {
  dep[u] = dep[p] + 1;
  bg[u] = ++totime;
  st[totime][0] = u;
  for (int i = h[u]; i; i = e[i].nex) {
    const int v = e[i].t;
    if (v == p) continue;
    dfs(v, u);
    st[++totime][0] = u;
  }
  ed[u] = totime;
}
void bin_exp() {
  FOR(j, 1, 19) {
    if ((1 << j) > totime) break;
    int ilim = totime - (1 << j) + 1;
    FOR(i, 1, ilim) {
      st[i][j] = dep[st[i][j - 1]] < dep[st[i + (1 << j - 1)][j - 1]]
                     ? st[i][j - 1]
                     : st[i + (1 << j - 1)][j - 1];
    }
  }
}
int lca(int u, int v) {
  int l = bg[u], r = bg[v];
  if (l > r) l ^= r ^= l ^= r;
  int j = log(r - l + 1) / log(2);
  return dep[st[l][j]] < dep[st[r - (1 << j) + 1][j]] ? st[l][j]
                                                      : st[r - (1 << j) + 1][j];
}
```

code/euler_lca.cpp

7.3 IO 优化

```
namespace IO {
  char nc() {
    static char bf[100000], *p1 = bf, *p2 = bf;
    return p1 == p2 && (p2 = (p1 = bf) + fread(bf, 1, 100000, stdin), p1 == p2)
               ? EOF
               : *p1++;
  }
```

```cpp
  int rd() {
    int res = 0;
    char c = getchar();
    while (!isdigit(c)) c = getchar();
    while (isdigit(c)) res = res * 10 + c - '0', c = getchar();
    return res;
  }
  int tmp[20], lt;
  char OBF[100000], *OBP = OBF;
  void flush() { fwrite(OBF, 1, OBP - OBF, stdout), OBP = OBF; }
  void wrch(char x) {
    if (OBP == OBF + 99999) flush();
    *OBP = x, ++OBP;
  }
  void wr(int x) {
    lt = 0;
    while (x) tmp[++lt] = x % 10, x /= 10;
    while (lt > 0) wrch(tmp[lt] + '0'), --lt;
  }
  void wr(long long x) {
    lt = 0;
    while (x) tmp[++lt] = x % 10, x /= 10;
    while (lt > 0) wrch(tmp[lt] + '0'), --lt;
  }
  void wr(const char *s) {
    while (*s) wrch(*s), ++s;
  }
  void wr(char x) { wrch(x); }
} // namespace IO
/*
 * nc():读取下一个字符，如果缓冲区没了就刷一次缓冲区
 * rd():读入一个整数 (含义为 read %d
 * flush():输出时把缓冲区清空。记得在程序结尾的时侯flush一下
 * wrch():输出单个字符
 * wr():输出一个整数/LL/字符串/单个字符
 */
```

code/io.cpp