

Algorithms for XCPC

Sshwy

2022 年 10 月 6 日

目录

1	代码头	1
2	字符串	1
2.1	KMP 算法	1
2.2	Manacher 算法	2
2.3	后缀数组	2
2.4	后缀自动机	3
2.5	广义后缀自动机	4
2.6	回文自动机	6
3	数论与线性代数	7
3.1	EX-BSGS 算法	7
3.2	Pollard-Rho 和 Miller	9
3.3	线性基	10
3.4	Min 25	12
3.5	Min 25 杰哥	15
3.6	二次剩余 Cipolla	16
4	多项式相关	16
4.1	FFT	16
4.2	NTT	17
4.3	FWT	18
4.4	全家桶	19
5	网络流	21
5.1	MCMF 最大费用最大流	21
5.2	DINIC 算法求最大流	22
6	数据结构	23
6.1	左偏树	23
6.2	Splay	23
6.3	非旋转 Treap	25
6.4	LCT	26
7	其他	28
7.1	计算几何	28
7.2	欧拉序求 LCA	39

7.3 IO 优化	40
---------------------	----

1 代码头

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long lld;
4 typedef long double lf;
5 typedef unsigned long long uld;
6 typedef pair<int, int> pii;
7 #define fi first
8 #define se second
9 #define pb push_back
10 #define mk make_pair
11 #define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
12 #define ROF(i, a, b) for (int i = (a); i >= (b); --i)
13 namespace RA {
14     int r(int p) { return 1ll * rand() * rand() % p; }
15     int r(int L, int R) { return r(R - L + 1) + L; }
16 } // namespace RA
```

code/head.cpp

2 字符串

2.1 KMP 算法

```
1 const int N = 1000;
2 int n, nex[N];
3 void getNext(char *s, int l) {
4     nex[0] = -1;
5     int i = 0, j = -1;
6     while (i < l) {
7         if (j == -1 || s[i] == s[j]) i++, j++, nex[i] = j;
8         else j = nex[j];
9     }
10 }
11 int kmp(char *s, char *t) {
12     int ls = strlen(s), lt = strlen(t);
13     getNext(s, ls);
14     int i = 0, j = 0;
15     while (i < ls && j < lt) {
16         if (i == -1 || s[i] == t[j]) i++, j++;
17         else i = nex[i];
18     }
19     if (i == ls) return j - i;
20     else return -1;
21 }
```

code/kmp.cpp

2.2 Manacher 算法

```

1  const int N = 11000100;
2  char s[N];
3  int ls, L, R, d[N], ans;
4  int main() {
5      scanf("%s", s + 1);
6      ls = strlen(s + 1);
7      FOR(i, 1, ls) {
8          if (i < R) d[i] = min(R - i, d[L + R - i]);
9          while (
10             0 < i - d[i] - 1 && i + d[i] + 1 <= ls && s[i - d[i] - 1] == s[i + d[i] + 1])
11             ++d[i];
12         if (i + d[i] > R) L = i - d[i], R = i + d[i];
13         assert(L > 0);
14         ans = max(ans, d[i] * 2 + 1);
15     }
16     L = R = 0;
17     FOR(i, 1, ls) d[i] = 0;
18     FOR(i, 1, ls - 1) {
19         if (i <= R) d[i] = min(R - i, d[L + R - i - 1]);
20         while (0 < i - d[i] && i + d[i] + 1 <= ls && s[i - d[i]] == s[i + d[i] + 1])
21             ++d[i];
22         if (i + d[i] > R) L = i - d[i] + 1, R = i + d[i];
23         ans = max(ans, d[i] * 2);
24     }
25     printf("%d", ans);
26     return 0;
27 }

```

code/manacher.cpp

2.3 后缀数组

```

1  struct SA {
2      char s[N];
3      int l;
4      int sa[N], rk[N];
5      int t[N], bin[N], sz;
6      int h[N], he[N]; // h,height
7      void qsort() {
8          for (int i = 0; i <= sz; i++) bin[i] = 0;
9          for (int i = 1; i <= l; i++) bin[rk[i]]++;
10         for (int i = 1; i <= sz; i++) bin[i] += bin[i - 1];
11         for (int i = l; i >= 1; i--) sa[bin[rk[t[i]]]--] = t[i];
12     }
13     void make(char *s) {
14         l = strlen(s + 1), sz = max(l, 'z' - '0' + 1);
15         for (int i = 1; i <= l; i++) t[i] = i, rk[i] = s[i] - '0' + 1;
16         qsort();
17         for (int j = 1; j <= l; j <= 1) {

```

```

18     int tot = 0;
19     for (int i = l - j + 1; i <= l; i++) t[++tot] = i;
20     for (int i = 1; i <= l; i++)
21         if (sa[i] - j > 0) t[++tot] = sa[i] - j;
22     qsort();
23     memcpy(t, rk, sizeof(int) * (l + 1));
24     rk[sa[1]] = tot = 1;
25     for (int i = 2; i <= l; i++)
26         rk[sa[i]] =
27             t[sa[i - 1]] == t[sa[i]] && t[sa[i - 1] + j] == t[sa[i] + j] ? tot : ++tot;
28     }
29 }
30 int move(int x, int y, int len) {
31     while (x + len <= l && y + len <= l && s[x + len] == s[y + len]) ++len;
32     return len;
33 }
34 void calc_h() {
35     for (int i = 1; i <= l; i++)
36         h[i] = rk[i] == 1 ? 0 : move(i, sa[rk[i] - 1], max(h[i - 1] - 1, 0));
37 }
38 int st[N][16]; // h[sa[i]]~h[sa[i+2^j]] 中的最小值
39 void make_st() {
40     for (int i = 1; i <= l; i++) st[i][0] = h[sa[i]];
41     for (int j = 1; (1 << j) <= l; j++) {
42         int step = 1 << (j - 1);
43         for (int i = 1; i + step <= l; i++) {
44             st[i][j] = min(st[i][j - 1], st[i + step][j - 1]);
45         }
46     }
47 }
48 int lcp(int x, int y) { // 返回长度
49     if (x == y) return l - x + 1;
50     x = rk[x], y = rk[y];
51     if (x > y) swap(x, y);
52     x++; // 取不到 x
53     int step = log(y - x + 1) / log(2);
54     return min(st[x][step], st[y - (1 << step) + 1][step]);
55 }
56 };

```

code/sa.cpp

2.4 后缀自动机

```

1  const int SZ = 2e6 + 500, ALP = 26;
2
3  struct SAM {
4      int tot, last;
5      int tr[SZ][ALP], fail[SZ];
6      int len[SZ], cnt[SZ], end[SZ];
7      int s[SZ], ls;
8      SAM() { tot = last = 1, len[1] = 0, fail[1] = 0; }

```

```

9   void insert(char x) {
10       s[++ls] = x;
11       x -= 'a';
12       int u = ++tot, p = last;
13       len[u] = len[last] + 1, last = u;
14       cnt[u] = 1, end[u] = ls; // u 的卫星信息
15       while (p && tr[p][x] == 0) tr[p][x] = u, p = fail[p];
16       if (!p) fail[u] = 1;
17       else {
18           int q = tr[p][x];
19           if (len[q] == len[p] + 1) fail[u] = q;
20           else {
21               int cq = ++tot;
22               len[cq] = len[p] + 1, fail[cq] = fail[q];
23               end[cq] = end[q]; // 如果需要, 更新的cq的卫星信息
24               memcpy(tr[cq], tr[q], sizeof(tr[q]));
25               fail[q] = fail[u] = cq;
26               while (p && tr[p][x] == q) tr[p][x] = cq, p = fail[p];
27           }
28       }
29   }
30   int a[SZ], bin[SZ], tim[SZ];
31   void count() { // 桶排, 统计cnt
32       FOR(i, 1, tot) bin[len[i]]++;
33       FOR(i, 1, tot) bin[i] += bin[i - 1];
34       FOR(i, 1, tot) a[bin[len[i]]--] = i;
35       ROF(i, tot, 1) cnt[fail[a[i]]] += cnt[a[i]];
36   }
37   void print_node(int u) { // 输出每个结点的状态
38       printf("u=%d,cnt=%d,len=%d,fail=%d, ", u, cnt[u], len[u], fail[u]);
39       FOR(i, end[u] - len[u] + 1, end[u]) putchar(s[i]);
40       puts("");
41   }
42   };
43   /*
44   * cnt: 状态出现次数; end: 状态的结尾位置
45   * tim: 每个结点的时间戳
46   */

```

code/sam.cpp

2.5 广义后缀自动机

```

1   /*****heading*****/
2   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         }
56     }
57     void print_node(int u) {
58         printf("u=%2d, len=%2d, fail=%2d, tnode=%2d\n", u, len[u], fail[u], tnode[u]);
59     }
60     void count() {
61         lld ans = 0;
62         FOR(i, 1, tot) ans += len[i] - len[fail[i]];

```



```

63     printf("%lld", ans);
64 }
65 } sam;
66 queue<int> q;
67 void go() {
68     last[1] = 1;
69     q.push(1);
70     while (!q.empty()) {
71         int u = q.front();
72         q.pop();
73         FOR(i, 0, c - 1) {
74             if (!T::tr[u][i]) continue;
75             sam.insert(u, i);
76             q.push(T::tr[u][i]);
77         }
78     }
79     sam.count();
80 }
81 int main() {
82     scanf("%d%d", &n, &c);
83     FOR(i, 1, n) scanf("%d", &col[i]);
84     FOR(i, 1, n - 1) {
85         int u, v;
86         scanf("%d%d", &u, &v);
87         add_path(u, v), add_path(v, u);
88         dg[u]++, dg[v]++;
89     }
90     FOR(i, 1, n) if (dg[i] == 1) T::dfs_add_trie(i, 0, 1);
91     go();
92     return 0;
93 }

```

code/general_sam.cpp

2.6 回文自动机

```

1  const int SZ = 5e5 + 500, ALP = 26;
2  struct PAM {
3      int tot, last;
4      int len[SZ], tr[SZ][ALP], fail[SZ];
5      int s[SZ], ls; // 字符串的内容 - '0'
6      int cnt[SZ], num[SZ]; // 状态出现次数、fail树上的深度 (有多少回文后缀)
7      int newnode(int l) {
8          ++tot, len[tot] = l, fail[tot] = 0, cnt[tot] = 0;
9          FOR(i, 0, ALP - 1) tr[tot][i] = 0;
10         return tot;
11     }
12     void clear() {
13         tot = -1, newnode(0), newnode(-1), fail[0] = 1, last = 0,
14         s[ls = 0] = -1; // 减掉'a'后0就不是非匹配字符了, 所以要整成-1
15     }
16     PAM() { clear(); }

```

```

17 int getfail(int u) {
18     // 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
19     while (s[ls - len[u] - 1] != s[ls]) u = fail[u];
20     return u;
21 }
22 void insert(char c) {
23     s[++ls] = (c -= 'a');
24     int cur = getfail(last);
25     if (!tr[cur][c]) { // 如果没有转移就添加
26         int u = newnode(len[cur] + 2);
27         fail[u] = tr[getfail(fail[cur])][c];
28         tr[cur][c] = u;
29         // 在此处更新 tot 的卫星信息
30         num[tot] = num[fail[tot]] + 1;
31     }
32     last = tr[cur][c];
33     // 在此处更新 last 的卫星信息
34     cnt[last]++;
35 }
36 void count() { //最后用来计算每个状态的出现次数
37     R0F(i, tot, 0) cnt[fail[i]] += cnt[i];
38 }
39 };
40 /*
41  * 0 号结点表示 0 结点
42  * 1 号结点表示 -1 结点，长度为 -1
43  * last 记录上一次插入的字符所在结点的编号
44  */

```

code/pam.cpp

3 数论与线性代数

3.1 EX-BSGS 算法

```

1 namespace EXBSGS {
2     const lld SZ = 433337;
3     struct hash_map {
4         struct data {
5             lld u, v, nex;
6         };
7         data e[SZ];
8         lld h[SZ], le;
9         lld hash(lld u) { return (u % SZ + SZ) % SZ; }
10        lld &operator[](lld u) {
11            lld hu = hash(u);
12            for (lld i = h[hu]; i; i = e[i].nex)
13                if (e[i].u == u) return e[i].v;
14            return e[++le] = (data){u, -1, h[hu]}, h[hu] = le, e[le].v;
15        }
16        void clear() { memset(h, 0, sizeof(h)), le = 0; }

```

```

17 } h;
18 lld gcd(lld a, lld b) { return b ? gcd(b, a % b) : a; }
19 lld mul(lld a, lld b, lld p) {
20     if (p <= 1000000000ll) return 1ll * a * b % p;
21     if (p <= 1000000000000ll)
22         return (((a * (b >> 20) % p) << 20) % p + a * (b & ((1 << 20) - 1))) % p;
23     lld d = floor(a * (long double)b / p);
24     lld res = (a * b - d * p) % p;
25     if (res < 0) res += p;
26     return res;
27 }
28 lld pw(lld a, lld m, lld p) {
29     lld res = 1;
30     while (m) m & 1 ? res = mul(res, a, p) : 0, a = mul(a, a, p), m >>= 1;
31     return res;
32 }
33 lld exgcd(lld a, lld b, lld &x, lld &y) {
34     if (!b) return x = 1, y = 0, a;
35     lld t = exgcd(b, a % b, y, x);
36     return y = y - (a / b) * x, t;
37 }
38 lld inv(lld a, lld p) {
39     lld b, t, g = exgcd(a, p, b, t);
40     if (g > 1) return -1;
41     return ((b % p) + p) % p;
42 }
43 lld bsgs(lld a, lld b, lld p) {
44     a %= p, b %= p, h.clear();
45     if (b == 1) return 0;
46     if (!a && !b) return 1;
47     if (!a) return -1;
48     lld t = sqrt(p) + 0.5, cur = b, q = 1;
49     FOR(i, 0, t) h[cur] = i, cur = mul(cur, a, p);
50     cur = pw(a, t, p);
51     FOR(i, 0, t) {
52         if (h[q] != -1 && i * t - h[q] >= 0) return i * t - h[q];
53         q = mul(q, cur, p);
54     }
55     return -1;
56 }
57 lld exbsgs(lld a, lld b, lld p) {
58     lld d = 0, f = 1, g;
59     while ((g = gcd(a, p)) > 1) {
60         if (b % g) return -1;
61         ++d, f = mul(f, g, p), b /= g, p /= g;
62     }
63     lld ia = inv(a, p);
64     f = mul(f, pw(ia, d, p), p);
65     b = mul(b, f, p);
66     lld res = bsgs(a, b, p);
67     return ~res ? res + d : -1;
68 }
69 } // namespace EXBSGS

```

```

70  /*
71  * exbsgs: 求  $a^x = b \pmod p$  的最小非负整数解。-1表示无解
72  */

```

code/exbsgs.cpp

3.2 Pollard-Rho 和 Miller

```

1  typedef long long lld;
2  namespace math {
3      inline int powmod(int a, int b, int mod) {
4          int res = 1;
5          for (; b >>= 1, a = 1ll * a * a % mod)
6              if (b & 1) res = 1ll * res * a % mod;
7          return res;
8      }
9      inline lld mul(lld a, lld b, lld p) {
10         if (p <= 1000000000ll) return 1ll * a * b % p;
11         if (p <= 1000000000000ll)
12             return (((a * (b >> 20) % p) << 20) % p + a * (b & ((1 << 20) - 1))) % p;
13         lld d = floor(a * (long double)b / p);
14         lld res = (a * b - d * p) % p;
15         if (res < 0) res += p;
16         return res;
17     }
18     inline lld powmod(lld a, lld b, lld mod) {
19         lld res = 1;
20         for (; b >>= 1, a = mul(a, a, mod))
21             if (b & 1) res = mul(res, a, mod);
22         return res;
23     }
24     inline bool check(lld a, lld x, lld times, lld n) {
25         lld tmp = powmod(a, x, n);
26         while (times--) {
27             lld last = mul(tmp, tmp, n);
28             if (last == 1 && tmp != 1 && tmp != n - 1) return 0;
29             tmp = last;
30         }
31         return tmp == 1;
32     }
33     int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
34     const int S = 8;
35     inline bool Miller(lld n) {
36         FOR(i, 0, S) {
37             if (n == base[i]) return 1;
38             if (n % base[i] == 0) return 0;
39         }
40         lld x = n - 1, times = 0;
41         while (!(x & 1)) times++, x >>= 1;
42         FOR(_, 0, S) if (!check(base[_], x, times, n)) return 0;
43         return 1;
44     }

```

```

45 #define mytz __builtin_ctzll
46 inline lld gcd(lld a, lld b) {
47     if (!a) return b;
48     if (!b) return a;
49     register int t = mytz(a | b);
50     a >>= mytz(a);
51     do {
52         b >>= mytz(b);
53         if (a > b) {
54             lld t = b;
55             b = a, a = t;
56         }
57         b -= a;
58     } while (b);
59     return a << t;
60 }
61 #define F(x) ((mul(x, x, n) + c) % n)
62 inline lld rho(lld n, lld c) {
63     lld x = 1ll * rand() * rand() % n, y = F(x);
64     while (x ^ y) {
65         lld w = gcd(abs(x - y), n);
66         if (w > 1 && w < n) return w;
67         x = F(x), y = F(y), y = F(y);
68     }
69     return 1;
70 }
71 inline lld calc(lld x) {
72     if (Miller(x)) return x;
73     lld fsf = 0; // while((fsf=rho(x,rand()%x))==1);
74     while ((fsf = rho(x, 2)) == 1)
75         ;
76     return max(calc(fsf), calc(x / fsf));
77 }
78 } // namespace math
79 /*
80  * 不知从何处拉来的强大板子
81  * func.powmod(int,int,int) 32位整型快速幂
82  * func.mul(lld,lld,lld) 64位长整型数的乘法运算
83  * func.powmod(lld,lld,lld) 64位长整型数的快速幂
84  */

```

code/math.cpp

3.3 线性基

```

1  const int N = 100;
2  long long p[N];
3  void insert(long long x) {
4      R0F(i, 62, 0) {
5          if (!(x >> i) & 1) continue; //判断第i位是否为0
6          if (!p[i]) {
7              p[i] = x;

```

```

8         break;
9     } //不能张成，添加到线性基中
10    x ^= p[i]; //去掉可以张成的维度（第i维消元）
11    }
12 }

```

code/linear_basis.cpp

```

1  int n;
2  long long b[70], lb;
3  bool insert(long long x) { //消成对角矩阵
4      R0F(i, 63, 0) {
5          if (!(x >> i & 1)) continue;
6          if (!b[i]) {
7              b[i] = x, lb++;
8              R0F(j, i - 1, 0) if (b[i] >> j & 1) b[i] ^= b[j]; //消掉i的其他元
9              F0R(j, i + 1, 63) if (b[j] >> i & 1) b[j] ^= b[i]; //消掉其他元的i
10             return 1;
11         }
12         x ^= b[i];
13     }
14     return 0;
15 }
16 void print() {
17     int len = 0;
18     F0R(i, 0, 63) if (b[i]) len = i;
19     F0R(i, 0, len) {
20         R0F(j, len, 0) { printf("%lld", b[i] >> j & 1); }
21         puts("");
22     }
23 }

```

code/linear_basis2.cpp

```

1  struct basis {
2      long long b[61];
3      basis() { memset(b, 0, sizeof(b)); }
4      bool insert(long long x) {
5          R0F(i, 60, 0) {
6              if (x >> i & 1) {
7                  if (b[i]) x ^= b[i];
8                  else return b[i] = x, 1;
9              }
10         }
11         return 0;
12     }
13     long long qmax() {
14         long long res = 0;
15         R0F(i, 60, 0) if ((res ^ b[i]) > res) res ^= b[i];
16         return res;
17     }
18     basis operator+(basis bi) {
19         basis res = bi;
20         F0R(i, 0, 60) if (b[i]) res.insert(b[i]);

```

```

21     return res;
22 }
23 };
24 /*
25  * 上三角矩阵
26  * 重载了合并的运算符
27  */

```

code/linear_basis3.cpp

3.4 Min 25

```

1 // by Yao
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define FOR(i, a, b) for (int i = (a); i <= (b); ++i)
5 #define ROF(i, a, b) for (int i = (a); i >= (b); --i)
6
7 typedef long long LL;
8
9 /**
10  * Min_25 筛
11  *
12  * ref: https://notes.sshwy.name/Math/Min\_25/
13  *
14  * N: 要筛的值的上界
15  * CNT: f1 的项数
16  * 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）
17  * 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
18  * esf1: 计算 f1 前缀和的点值
19  * f_pe: 计算 f 在质数幂处的值
20  * MOD: 模数
21  */
22 // Usage (https://www.luogu.com.cn/problem/P5325):
23 const LL N = 1e10;
24 const int P = 1e9 + 7, I6 = (P + 1) / 6, I2 = (P + 1) / 2;
25
26 int coef[] = {P - 1, 1};
27
28 void f1(LL x, int *result) {
29     x %= P;
30     result[0] = x % P;
31     result[1] = x * 1ll * x % P;
32 }
33 void prefixSumF1(LL x, int *result) {
34     x %= P;
35     result[0] = x * (x + 1) % P * I2 % P;
36     result[1] = x * (x + 1) % P * (x * 2 + 1) % P * I6 % P;
37 }
38 int f_pe(int p, int e, LL pe) {
39     pe %= P;
40     return 1ll * pe * (pe - 1) % P;

```

```

41 }
42
43 int main() {
44     long long n;
45     scanf("%lld", &n);
46
47     Min25<N, 2, coef, f1, prefixSumF1, f_pe, P> Sieve;
48     int ans = Sieve.sieve(n);
49
50     printf("%d\n", ans);
51     return 0;
52 }
53 // Min_25
54 template <const LL N, const int CNT, const int COEF[CNT], void (*ef1)(LL, int *),
55           void (*esf1)(LL, int *), int (*f_pe)(int, int, LL), const int MOD>
56 struct Min25 {
57     vector<vector<int>> g, h, hs;
58     vector<int> id[2], pn;
59     vector<LL> val;
60     LL n;
61     int SQRT_N, sqrt_n, tot, lp;
62
63     Min25() {
64         SQRT_N = 2 * (sqrt(N) + 5);
65         tot = lp = 0;
66         g.resize(SQRT_N, vector<int>(CNT, 0));
67         h.resize(SQRT_N, vector<int>(CNT, 0));
68         hs.resize(SQRT_N, vector<int>(CNT, 0));
69         id[0].resize(SQRT_N, 0), id[1].resize(SQRT_N, 0);
70         val.resize(SQRT_N, 0), pn.resize(SQRT_N, 0);
71     }
72     // Min_25
73     void init() {
74         sqrt_n = sqrt(n) + 3;
75         vector<bool> co(SQRT_N, false);
76         co[0] = co[1] = 1;
77         for (int i = 2; i <= sqrt_n; i++) {
78             if (!co[i]) pn[++lp] = i;
79             for (int j = 1; j <= lp && 1ll * i * pn[j] <= sqrt_n; j++) {
80                 co[i * pn[j]] = 1;
81                 if (i % pn[j] == 0) break;
82             }
83         }
84         for (LL pos = 1, nex, w; pos <= n; pos = nex + 1) {
85             nex = n / (n / pos), w = n / pos, val[++tot] = w;
86             w <= sqrt_n ? id[0][w] = tot : id[1][n / w] = tot;
87         } // FOR(i, 1, tot) assert(I(val[i]) == i);
88     }
89     void calc_h() {
90         int tmp[CNT];
91         FOR(i, 1, lp) {
92             ef1(pn[i], tmp);
93             FOR(j, 0, CNT - 1) {

```



```

94     h[i][j] = tmp[j];
95     hs[i][j] = (hs[i - 1][j] + tmp[j]) % MOD;
96 }
97 }
98 }
99 int H(int i) { // 计算 sum_j f(p_j) (j <= i) // assert(i <= lp);
100     int res = 0;
101     FOR(j, 0, CNT - 1) res = (res + 1ll * hs[i][j] * COEF[j]) % MOD;
102     return res;
103 }
104 int I(LL x) { return x <= sqrt_n ? id[0][x] : id[1][n / x]; }
105 void calc_g() {
106     FOR(i, 1, tot) { // 当 i=0
107         int tmp[CNT];
108         esf1(val[i], tmp);
109         FOR(j, 0, CNT - 1) {
110             g[i][j] = (tmp[j] - 1 + MOD) % MOD; // 对于积性函数来说必然有 f(1) = 1
111         }
112     }
113     FOR(i, 1, lp) { // pn[i]
114         FOR(j, 1, tot) {
115             if (1ll * pn[i] * pn[i] > val[j]) break;
116             int k = I(val[j] / pn[i]);
117             FOR(t, 0, CNT - 1) {
118                 g[j][t] =
119                     (g[j][t] - 1ll * h[i][t] * (g[k][t] - hs[i - 1][t]) % MOD + MOD) % MOD;
120             }
121         }
122     }
123 }
124 int G(LL x) { // 计算 sum f(p) (p <= x 且 p 是质数)
125     int res = 0;
126     FOR(i, 0, CNT - 1) res = (res + 1ll * g[I(x)][i] * COEF[i]) % MOD;
127     return res;
128 }
129 int S(int i, LL m) {
130     if (m < pn[i] || m <= 1) return 0;
131     LL res = (G(m) - H(i - 1) + MOD) % MOD;
132     FOR(j, i, lp) {
133         if (1ll * pn[j] * pn[j] > m) break;
134         LL pje = 1, pje1 = pn[j];
135         FOR(e, 1, 100) {
136             pje *= pn[j], pje1 *= pn[j];
137             if (pje1 > m) break;
138             res += 1ll * f_pe(pn[j], e, pje) * S(j + 1, m / pje) % MOD +
139                 f_pe(pn[j], e + 1, pje1);
140             res %= MOD;
141         }
142     }
143     return res;
144 }
145
146 int sieve(LL _n) {

```

```

147     n = _n, init(), calc_h(), calc_g();
148     return (S(1, n) + 1) % MOD;
149 }
150 };

```

code/min_25.cpp

3.5 Min 25 杰哥

```

1  typedef long long LL;
2
3  LL a[N], T, n;
4  int pr[N], id1[N], id2[N], flag[N], g[N], sum[N], ncnt, m;
5
6  int ID(LL x) { return x <= T ? id1[x] : id2[n / x]; }
7  int calc(LL x) {
8      return x % mod, x * (x + 1) / 2 % mod - 1;
9  } // 算完全积性函数的前缀和
10 int f1(LL x) { return x % mod, x * (x - 1) % mod; } // 算积性函数在质数位置的取值
11 int f(LL j, int c) {} // 计算 f(j), 其中 j 是某个质数的 c 次方
12
13 void init() {
14     T = sqrt(n + 0.5);
15     for (int i = 2; i <= T; i++) {
16         if (!flag[i]) pr[++ncnt] = i, sum[ncnt] = (sum[ncnt - 1] + f1(i)) % mod;
17         for (int j = 1; j <= ncnt && (LL)i * pr[j] <= T; j++) {
18             flag[i * pr[j]] = 1;
19             if (i % pr[j] == 0) break;
20         }
21     }
22     for (LL l = 1; l <= n; l = n / (n / l) + 1) {
23         a[++m] = n / l;
24         if (a[m] <= T) id1[a[m]] = m;
25         else id2[n / a[m]] = m;
26         g[m] = calc(a[m]);
27     }
28     for (int i = 1; i <= ncnt; i++)
29         for (int j = 1; j <= m && (LL)pr[i] * pr[i] <= a[j]; j++)
30             g[j] = (g[j] - (LL)f1(pr[i]) * (g[ID(a[j] / pr[i])] - sum[i - 1]) % mod + mod) %
31                 mod;
32 }
33
34 int solve(LL n, int m) {
35     if (n < pr[m]) return 0;
36     int res = (g[ID(n)] * (LL)2 * t - (LL)sum[m - 1] * 2 * t) % mod;
37     res = (res + mod) % mod;
38     for (int i = m; i <= ncnt && (LL)pr[i] * pr[i] <= n; i++)
39         for (LL j = pr[i], c = 1; j * pr[i] <= n; j *= pr[i], c++)
40             Inc(res, ((LL)solve(n / j, i + 1) * f(j, c) + f(j * pr[i], c + 1)) % mod);
41     return res;

```

42 | }

code/min_25_jie.cpp

3.6 二次剩余 Cipolla

```

1 int pw(int a, int m, int p) {
2     int res = 1;
3     while (m) m & 1 ? res = 1ll * res * a % p : 0, a = 1ll * a * a % p, m >>= 1;
4     return res;
5 }
6 struct sqrtNum {
7     int W, P;
8     int a, b; // a+b*sqrt(W)
9     sqrtNum(int _a, int _b, int _W, int _P) { a = _a, b = _b, W = _W, P = _P; }
10    sqrtNum operator*(const sqrtNum &x) const {
11        return sqrtNum((1ll * a * x.a % P + 1ll * b * x.b % P * W) % P,
12                        (1ll * a * x.b % P + 1ll * b * x.a) % P, W, P);
13    }
14 };
15 int quad_res(int n, int p) {
16     n %= p;
17     assert(p & 1);
18     if (pw(n, (p - 1) / 2, p) == p - 1) return -1; // no solution
19     if (n == 0) return 0;
20     int a;
21     srand(clock() + time(0));
22     do a = rand() % p;
23     while (pw((a * 1ll * a % p - n + p) % p, (p - 1) / 2, p) == 1);
24     int w2 = (a * 1ll * a % p - n + p) % p;
25     sqrtNum q(a, 1, w2, p), qm(1, 0, w2, p);
26     for (int m = (p + 1) / 2; m; m >>= 1, q = q * q)
27         if (m & 1) qm = qm * q;
28     assert(qm.b == 0);
29     return qm.a;
30 }

```

code/cipolla.cpp

4 多项式相关

4.1 FFT

```

1 namespace FFT {
2     const int N = (1 << 21) + 5;
3     const double PI = acos(-1);
4     struct cpx {
5         double a, b;
6         cpx(double _a = 0, double _b = 0) { a = _a, b = _b; }

```

```

7   cpx operator+(cpx c) { return cpx(a + c.a, b + c.b); }
8   cpx operator-(cpx c) { return cpx(a - c.a, b - c.b); }
9   cpx operator*(cpx c) { return cpx(a * c.a - b * c.b, a * c.b + b * c.a); }
10  };
11  int tr[N], d;
12  void dft(cpx f[], int len, int typ) {
13      for (int i = 0; i < len; i++)
14          if (i < tr[i]) swap(f[i], f[tr[i]]);
15      for (int j = 1; j < len; j <= 1) {
16          cpx wn(cos(PI / j), sin(typ * PI / j));
17          for (int i = 0; i < len; i += j < 1) {
18              cpx w(1, 0), u, v;
19              for (int k = i; k < i + j; k++, w = w * wn)
20                  u = f[k], v = f[k + j] * w, f[k] = u + v, f[k + j] = u - v;
21          }
22      }
23      if (typ == -1)
24          for (int i = 0; i < len; i++) f[i].a /= len;
25  }
26  int init(int l) {
27      d = 0;
28      int len = 1;
29      while (len < (l < 1)) len <= 1, d++;
30      for (int i = 1; i < len; i++) tr[i] = (tr[i > 1] > 1) | (i & 1) < (d - 1);
31      return len;
32  }
33 } // namespace FFT
34 typedef FFT::cpx dft[FFT::N];
35 /*
36  * L5 : ai+b
37  * L15:  $w_{(2*j)}$  的单位根,  $2*PI/(2*j)=PI/j$ 
38  * L22: 实部
39  * L28: 蝴蝶变换预处理
40  * L32: 定义dft类型, 表示一个cpx的数组
41  * dft(f,len,typ): len 必须是  $2^k$ 
42  * 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
43  */

```

code/fft.cpp

4.2 NTT

```

1  namespace NTT {
2      const int N = (1 < 21) + 5, P = 998244353;
3      int pw(int a, int m) {
4          int res = 1;
5          while (m) m & 1 ? res = 1ll * res * a % P : 0, a = 1ll * a * a % P, m >>= 1;
6          return res;
7      }
8      int tr[N], d;
9      void dft(int f[], int len, int typ) {
10         FOR(i, 0, len - 1) if (i < tr[i]) swap(f[i], f[tr[i]]);

```

```

11     for (int j = 1; j < len; j <= 1) {
12         int wn = pw(3, (P - 1) / (j < 1) * typ + P - 1);
13         for (int i = 0; i < len; i += j < 1) {
14             int w = 1, u, v;
15             for (int k = i; k < i + j; k++, w = 1ll * w * wn % P) {
16                 u = f[k], v = 1ll * w * f[k + j] % P;
17                 f[k] = u + v, f[k] < P ? 0 : f[k] -= P;
18                 f[k + j] = u - v, f[k + j] < 0 ? f[k + j] += P : 0;
19             }
20         }
21     }
22     if (typ == -1) {
23         int x = pw(len, P - 2);
24         for (int i = 0; i < len; i++) f[i] = 1ll * f[i] * x % P;
25     }
26 }
27 int init(int l) {
28     d = 0;
29     int len = 1;
30     while (len < (l < 1)) len <= 1, ++d;
31     for (int i = 1; i < len; i++) tr[i] = (tr[i > 1] > 1) | (i & 1) < (d - 1);
32     return len;
33 }
34 } // namespace NTT
35 typedef int dft[NTT::N];
36 /*
37  * 必须保证初始时f里的值非负!
38  */

```

code/ntt.cpp

4.3 FWT

```

1  /*****heading*****/
2  #define int lld
3  const int N = 1 << 17, P = 998244353;
4  int n, in;
5  int a[N], b[N], c[N];
6
7  int pw(int a, int m, int p) {
8      int res = 1;
9      while (m) m & 1 ? res = res * a % p : 0, a = a * a % p, m >>= 1;
10     return res;
11 }
12 void fwt_and(int *f, int tag) {
13     for (int j = 1; j < n; j <= 1)
14         for (int i = 0; i < n; i += j < 1)
15             for (int k = i; k < i + j; k++) f[k] = (f[k] + f[k + j] * tag) % P;
16     FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
17 }
18 void fwt_or(int *f, int tag) {
19     for (int j = 1; j < n; j <= 1)

```

```

20     for (int i = 0; i < n; i += j << 1)
21         for (int k = i; k < i + j; k++) f[k + j] = (f[k + j] + f[k] * tag) % P;
22     FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
23 }
24 void fwt_xor(int *f, int tag) {
25     for (int j = 1; j < n; j <= 1)
26         for (int i = 0; i < n; i += j << 1)
27             for (int k = i; k < i + j; k++) {
28                 int x = f[k], y = f[k + j];
29                 f[k] = (x + y) % P, f[k + j] = (x - y) % P;
30             }
31     if (tag == -1) FOR(i, 0, n - 1) f[i] = f[i] * in % P;
32     FOR(i, 0, n - 1) f[i] += f[i] < 0 ? P : 0;
33 }
34 /*
35  * fwt:tag=1
36  * ifwt:tag=-1
37  * in=pw(n,P-2,P) (inverse element if n)
38  */

```

code/fwt.cpp

4.4 全家桶

```

1  const int N = 300010, mod = 998244353;
2  typedef long long LL;
3  int Pow(int x, int y) {
4      int res = 1;
5      for (; y; y >>= 1, x = (LL)x * x % mod)
6          if (y & 1) res = (LL)res * x % mod;
7      return res;
8  }
9
10 int Wn[2][17][N << 1], r[N << 2];
11
12 void Pre(int len) {
13     for (int mid = 2, step = 0; mid <= len; mid <= 1, step++) {
14         int t1 = Pow(3, (mod - 1) / mid), t2 = Pow(t1, mod - 2);
15         Wn[0][step][0] = Wn[1][step][0] = 1, Wn[1][step][1] = t1, Wn[0][step][1] = t2;
16         for (int i = 2; i < (mid >> 1); i++) {
17             Wn[1][step][i] = (LL)Wn[1][step][i - 1] * t1 % mod;
18             Wn[0][step][i] = (LL)Wn[0][step][i - 1] * t2 % mod;
19         }
20     }
21 }
22
23 void Inc(int &x, int y) { x += y, x -= x >= mod ? mod : 0; }
24 int Sub(int x, int y) {
25     int t = x - y;
26     return t < 0 ? t + mod : t;
27 }
28

```

```

29 void NTT(vector<int> &a, int len, int type) {
30     for (int i = 0; i < len; i++)
31         if (i < r[i]) swap(a[i], a[r[i]]);
32     for (int mid = 2, step = 0; mid <= len; mid <= 1, step++)
33         for (int i = 0; i < len; i += mid)
34             for (int j = i; j < i + (mid >> 1); j++) {
35                 int t = (LL)Wn[type][step][j - i] * a[j + (mid >> 1)] % mod;
36                 a[j + (mid >> 1)] = Sub(a[j], t), Inc(a[j], t);
37             }
38     if (!type)
39         for (int i = 0, inv = Pow(len, mod - 2); i < len; i++) a[i] = (LL)a[i] * inv % mod;
40 }
41
42 void GetR(int len, int l) {
43     for (int i = 1; i < len; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << l - 1);
44 }
45
46 vector<int> mul(vector<int> a, vector<int> b) {
47     int t = a.size() + b.size() - 2, len = 1, l = 0;
48     while (len <= t) len <= 1, l++;
49     GetR(len, l), a.resize(len + 1), b.resize(len + 1);
50     NTT(a, len, 1), NTT(b, len, 1);
51     for (int i = 0; i < len; i++) a[i] = (LL)a[i] * b[i] % mod;
52     NTT(a, len, 0), a.resize(min(n + 1, t + 1));
53     return a;
54 }
55
56 vector<int> Inv(const vector<int> &a, int n) {
57     if (n == 1) return {Pow(a[0], mod - 2)};
58     vector<int> b = Inv(a, (n + 1) / 2);
59     int len = 1, l = 0;
60     while (len <= n * 2) len <= 1, l++;
61     GetR(len, l), b.resize(len);
62     vector<int> tmp(len);
63     for (int i = 0; i < n && i < a.size(); i++) tmp[i] = a[i];
64     NTT(tmp, len, 1), NTT(b, len, 1);
65     for (int i = 0; i < len; i++)
66         b[i] = (2 - (LL)tmp[i] * b[i] % mod + mod) * b[i] % mod;
67     NTT(b, len, 0), b.resize(n);
68     return b;
69 }
70
71 int inv[N];
72
73 vector<int> Ln(const vector<int> &a, int n) {
74     vector<int> a1, inva = Inv(a, n);
75     for (int i = 1; i < n && i < a.size(); i++) a1.push_back((LL)a[i] * i % mod);
76     a1 = mul(a1, inva), a1.resize(n);
77     for (int i = n - 1; i >= 1; i--)
78         a1[i] = (LL)a1[i - 1] * inv[i] % mod, assert(inv[i]);
79     return a1[0] = 0, a1;
80 }
81

```

```

82 vector<int> Exp(const vector<int> &a, int n) {
83     if (n == 1) return {1};
84     vector<int> tmp = Exp(a, (n + 1) / 2), ln = Ln(tmp, n), tmp1;
85     for (int i = 0; i < n; i++) tmp1.push_back((a[i] - ln[i] + mod) % mod);
86     tmp1[0]++, tmp = mul(tmp, tmp1), tmp.resize(n);
87     return tmp;
88 }

```

code/poly.cpp

5 网络流

5.1 MCMF 最大费用最大流

```

1 namespace MCMF {
2     struct qxx {
3         int nex, t, v, c;
4     };
5     qxx e[N];
6     int h[N], le = 1;
7     void add_path(int f, int t, int v, int c) {
8         e[++le] = (qxx){h[f], t, v, c}, h[f] = le;
9     }
10    void add_flow(int f, int t, int v, int c) {
11        add_path(f, t, v, c), add_path(t, f, 0, -c);
12    }
13
14    bool vis[N];
15    queue<int> q;
16    int d[N], pre[N], incf[N];
17    int s, t;
18    bool spfa() {
19        memset(d, -1, sizeof(d));
20        q.push(s), d[s] = 0, incf[s] = INF, incf[t] = 0;
21        while (!q.empty()) {
22            int u = q.front();
23            q.pop();
24            vis[u] = 0;
25            for (int i = h[u]; i; i = e[i].nex) {
26                const int v = e[i].t, w = e[i].v, c = e[i].c;
27                if (!w || d[v] >= d[u] + c) continue;
28                d[v] = d[u] + c, incf[v] = min(incf[u], w), pre[v] = i;
29                if (!vis[v]) q.push(v), vis[v] = 1;
30            }
31        }
32        return incf[t];
33    }
34    int maxflow, maxcost;
35    void update() {
36        maxflow += incf[t];
37        for (int u = t; u != s; u = e[pre[u] ^ 1].t) {

```



```

38     e[pre[u]].v -= incf[t], e[pre[u] ^ 1].v += incf[t];
39     maxcost += incf[t] * e[pre[u]].c;
40 }
41 }
42 void go() {
43     while (spfa()) update();
44 }
45 } // namespace MCMF
46 /*
47  * 注意，本板子是最大费用最大流！
48  */

```

code/mcmf.cpp

5.2 DINIC 算法求最大流

```

1  const int N = 5e5 + 5, M = 5e5 + 5, INF = 0x3f3f3f3f;
2  struct qxx {
3      int nex, t, v;
4  };
5  qxx e[M];
6  int h[N], cnt = 1;
7  void add_path(int f, int t, int v) { e[++cnt] = (qxx){h[f], t, v}, h[f] = cnt; }
8  void add_flow(int f, int t, int v) { add_path(f, t, v), add_path(t, f, 0); }
9  void add_dual_flow(int f, int t, int v) { add_path(f, t, v), add_path(t, f, v); }
10
11 namespace DINIC {
12     int s, t, maxflow, d[N];
13     queue<int> q;
14     bool bfs() {
15         memset(d, 0, sizeof(d));
16         q.push(s), d[s] = 1;
17         while (!q.empty()) {
18             int u = q.front();
19             q.pop();
20             for (int i = h[u]; i; i = e[i].nex) {
21                 const int &v = e[i].t, &w = e[i].v;
22                 if (!d[v] && w) d[v] = d[u] + 1, q.push(v);
23             }
24         }
25         return d[t];
26     }
27     int dinic(int u, int flow) {
28         if (u == t) return flow;
29         int k, rest = flow;
30         for (int i = h[u]; i && rest; i = e[i].nex) {
31             const int &v = e[i].t, &w = e[i].v;
32             if (!w || d[v] != d[u] + 1) continue;
33             k = dinic(v, min(rest, w));
34             if (k) e[i].v -= k, e[i ^ 1].v += k, rest -= k;
35             else d[v] = 0;
36         }

```

```

37     return flow - rest;
38 }
39 void go() {
40     while (bfs())
41         for (int i; i = dinic(s, INF);) maxflow += i;
42 }
43 } // namespace DINIC
44 /*
45  * add_dual_flow: 无向边的流 (即双向可流)
46  */

```

code/dinic.cpp

6 数据结构

6.1 左偏树

```

1  int lc[SZ], rc[SZ], val[SZ], dep[SZ], tot;
2  int new_node(int v) {
3      ++tot, val[tot] = v, lc[tot] = rc[tot] = 0, dep[tot] = 0;
4      return tot;
5  }
6  int merge(int x, int y) {
7      if (!x || !y) return x + y;
8      if (val[x] > val[y]) swap(x, y);
9      rc[x] = merge(rc[x], y);
10     if (dep[lc[x]] < dep[rc[x]]) swap(lc[x], rc[x]);
11     dep[x] = dep[rc[x]] + 1;
12     return x;
13 }
14 int getmin(int x) { return val[x]; }
15 int pop(int x) {
16     val[x] = -1;
17     return merge(lc[x], rc[x]);
18 }

```

code/leftist.cpp

6.2 Splay

```

1  const int SZ = 1e5 + 5;
2
3  struct Splay {
4      int tot, ch[SZ][2], key[SZ], sz[SZ], pa[SZ], root;
5      int rv[SZ];
6      int new_node(int v) {
7          ++tot, ch[tot][0] = ch[tot][1] = pa[tot] = 0, key[tot] = v;
8          sz[tot] = rv[tot] = 0;
9          return tot;

```

```

10 }
11 Splay() {
12     tot = 0;
13     new_node(-INF), new_node(INF);
14     pa[1] = 2, ch[2][0] = 1, root = 2;
15 }
16 bool get(int u) { return ch[pa[u]][1] == u; }
17 void pushup(int u) { sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1; }
18 void node_rv(int u) {
19     if (!u) return;
20     swap(ch[u][0], ch[u][1]);
21     rv[u] ^= 1;
22 }
23 void pushdown(int u) {
24     if (rv[u]) {
25         node_rv(ch[u][0]);
26         node_rv(ch[u][1]);
27         rv[u] = 0;
28     }
29 }
30 void rotate(int u) {
31     int p = pa[u], pp = pa[p], gu = get(u);
32     pushdown(u);
33     ch[pp][get(p)] = u, pa[u] = pp;
34     ch[p][gu] = ch[u][gu ^ 1], pa[ch[u][gu ^ 1]] = p;
35     ch[u][gu ^ 1] = p, pa[p] = u;
36     pushup(p), pushup(u);
37 }
38 void splay(int u, int v) {
39     while (pa[u] != v) {
40         int p = pa[u];
41         if (pa[p] != v) rotate(get(u) == get(p) ? p : u);
42         rotate(u);
43     }
44     if (!v) root = u;
45 }
46 bool find(int v) { // return if v exists. Based on key
47     if (!root) return 0;
48     int u = root;
49     while (key[u] != v && ch[u][key[u] < v]) pushdown(u), u = ch[u][key[u] < v];
50     splay(u, 0);
51     return key[root] == v;
52 }
53 void insert(int v) {
54     if (!root) return root = new_node(v), void();
55     int u = root, nu = new_node(v);
56     while (ch[u][key[u] < v]) pushdown(u), u = ch[u][key[u] < v];
57     ch[u][key[u] < v] = nu, pa[nu] = u;
58     splay(nu, 0);
59 }
60 int kth(int rk) {
61     ++rk;
62     if (!root) return 0;

```

```

63     int u = root;
64     while (sz[ch[u][0]] + 1 != rk) {
65         pushdown(u);
66         if (rk <= sz[ch[u][0]]) u = ch[u][0];
67         else rk -= sz[ch[u][0]] + 1, u = ch[u][1];
68     }
69     splay(u, 0);
70     return u;
71 }
72 };

```

code/splay.cpp

6.3 非旋转 Treap

```

1  #include <cstdio>
2  using namespace std;
3  const int N = 1e5 + 5, SZ = N;
4  int n;
5
6  struct Treap {
7      int seed = 1, root, tot;
8      int ch[SZ][2];
9      int val[SZ], rnd[SZ], sz[SZ]; // sz: 子树大小
10
11      int rrand() { return seed = seed * 482711; }
12      void pushup(int u) { sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1; }
13      void split(int u, int key, int &x, int &y) {
14          if (!u) x = y = 0;
15          else {
16              if (val[u] <= key) x = u, split(ch[u][1], key, ch[u][1], y);
17              else y = u, split(ch[u][0], key, x, ch[u][0]);
18              pushup(u);
19          }
20      }
21      int merge(int x, int y) { // x < y
22          if (!x || !y) return x + y; // 返回 x, y 或 0
23          if (rnd[x] < rnd[y]) return ch[x][1] = merge(ch[x][1], y), pushup(x), x;
24          else return ch[y][0] = merge(x, ch[y][0]), pushup(y), y;
25      }
26      void insert(int v) { // 插入 v
27          int x, y, u = ++tot;
28          val[u] = v, sz[u] = 1, rnd[u] = rrand();
29          split(root, v, x, y);
30          root = merge(merge(x, u), y);
31      }
32      void del(int v) {
33          int x, y, z;
34          split(root, v - 1, x, y); // 所有的 v 就被分在 y 中
35          split(y, v, y, z); // 所有的 v 仍被分在 y 中
36          if (!y) return; // 不存在 v 这个权值
37          y = merge(ch[y][0], ch[y][1]); // 根节点的不要了

```

```

38     root = merge(x, merge(y, z));
39 }
40 int rank(int v) { //即相同的数中，第一个数的排名
41     int x, y, res;
42     split(root, v - 1, x, y);
43     res = sz[x] + 1, root = merge(x, y);
44     return res;
45 }
46 int kth(int k) { //查询排名为k的数
47     int u = root;
48     while (k != sz[ch[u][0]] + 1) {
49         if (k <= sz[ch[u][0]]) u = ch[u][0];
50         else k -= sz[ch[u][0]] + 1, u = ch[u][1];
51     }
52     return val[u];
53 }
54 int pre(int v) { return kth(rank(v) - 1); } //严格前驱
55 int suc(int v) { return kth(rank(v) + 1); } //严格后继
56 } treap;
57
58 int main() {
59     scanf("%d", &n);
60     for (int i = 1; i <= n; i++) {
61         int opt, x;
62         scanf("%d%d", &opt, &x);
63         if (opt == 1) treap.insert(x);
64         else if (opt == 2) treap.del(x);
65         else if (opt == 3) printf("%d\n", treap.rank(x));
66         else if (opt == 4) printf("%d\n", treap.kth(x));
67         else if (opt == 5) printf("%d\n", treap.pre(x));
68         else printf("%d\n", treap.suc(x));
69     }
70     return 0;
71 }

```

code/FHQ_treap.cpp

6.4 LCT

```

1  const int SZ = 1e6 + 6;
2
3  int tot, ch[SZ][2], f[SZ], val[SZ], sz[SZ], rv[SZ];
4  int sxor[SZ];
5
6  int new_node(int v) {
7      ++tot, ch[tot][0] = ch[tot][1] = f[tot] = rv[tot] = 0;
8      sz[tot] = 1, val[tot] = v, sxor[tot] = v;
9      return tot;
10 }
11 void pushup(int u) {
12     sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + 1;
13     sxor[u] = sxor[ch[u][0]] ^ sxor[ch[u][1]] ^ val[u];

```

```

14 }
15 void noderv(int u) {
16     if (u) rv[u] ^= 1;
17 }
18 void nodeassign(int u, int v) { val[u] = v, pushup(u); }
19 void pushdown(int u) {
20     if (rv[u]) swap(ch[u][0], ch[u][1]), noderv(ch[u][0]), noderv(ch[u][1]), rv[u] = 0;
21 }
22 bool isroot(int u) { return ch[f[u]][0] != u && ch[f[u]][1] != u; }
23 bool get(int u) { return ch[f[u]][1] == u; }
24 void rotate(int u) {
25     int p = f[u], pp = f[p], k;
26     pushdown(p), pushdown(u), k = get(u); // k的赋值必须在pushdown后!
27     if (!isroot(p)) ch[pp][get(p)] = u; //!!!
28     ch[p][k] = ch[u][!k], f[ch[u][!k]] = p;
29     ch[u][!k] = p, f[p] = u, f[u] = pp;
30     pushup(p), pushup(u);
31 }
32 void splay(int u) {
33     pushdown(u);
34     for (int p; p = f[u], !isroot(u); rotate(u))
35         if (!isroot(p)) rotate(get(p) == get(u) ? p : u);
36 }
37 void access(int u) {
38     for (int p = 0; u; p = u, u = f[u]) splay(u), ch[u][1] = p, pushup(u);
39 }
40 void makeroot(int u) { access(u), splay(u), noderv(u); }
41 bool check_link(int x, int y) {
42     makeroot(x), access(y), splay(x), splay(y);
43     return !(isroot(x) && isroot(y));
44 }
45 void link(int x, int y) { makeroot(x), f[x] = y; }
46 bool check_edge(int x, int y) {
47     if (!check_link(x, y)) return 0;
48     makeroot(x), access(y), splay(y);
49     if (ch[y][0] != x || ch[x][1]) return 0;
50     return 1;
51 }
52 void cut(int x, int y) {
53     makeroot(x), access(y), splay(y), ch[y][0] = f[x] = 0, pushup(y);
54 }
55 void assign(int x, int y) { splay(x), nodeassign(x, y); }
56 int query(int x, int y) { return makeroot(x), access(y), splay(y), sxor[y]; }
57 /*
58  * 模板: Luogu3690
59  * new_node: 新建权值为 v 的结点
60  * pushup: 信息更新
61  * pushdown: 标记下传, 主要是翻转标记
62  * noderv: 对某一个结点施加标记。
63  * 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
64  * get 的时候会出锅
65  * nodeassign: 模板题需要
66  * isroot: 是否是所在Splay的根

```

```

67 * get: 是Splay上左儿子还是右儿子
68 * print: 调试函数
69 * 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
70 * 儿子不能乱赋值
71 * 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
72 * 先pushdown。
73 * 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
74 * 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)
75 * access完之后这个点不一定是所在splay的根, 需要手动splay一下
76 * 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。
77 * check_link: 判断两个点是否连通。
78 * link: 连接两个不连通的点
79 * check_edge: 判断两个点是否直连通 (有没有边)
80 * cut: 删掉 (x,y) 的边。
81 * assign: 模板题需要
82 * query: 模板题需要
83 * 提醒: 在修改了ch指针后要考虑是否pushup
84 */

```

code/lct.cpp

7 其他

7.1 计算几何

```

1 /**
2  * Computing Geometry Library
3  * ref1: https://onlinejudge.u-aizu.ac.jp/courses/library/4/CGL/all
4  * ref2: https://darkbzoj.tk/problem/2178
5  * @author Sshwy
6  */
7 #include <bits/stdc++.h>
8 using namespace std;
9 #define pb push_back
10 #define FOR(i, a, b) for (int i = (int)(a); i <= (int)(b); ++i)
11 #define ROF(i, a, b) for (int i = (int)(a); i >= (int)(b); --i)
12
13 namespace cg {
14     typedef long double vtyp;
15     const vtyp eps = 1e-9;
16     const vtyp PI = 3.1415926535897932626;
17
18     bool isZero(vtyp x) { return -eps < x && x < eps; }
19     bool eq(vtyp x, vtyp y) { return isZero(x - y); }
20     bool neq(vtyp x, vtyp y) { return !eq(x, y); }
21     bool lt(vtyp x, vtyp y) { return !eq(x, y) && x < y; }
22     bool gt(vtyp x, vtyp y) { return !eq(x, y) && x > y; }
23     bool le(vtyp x, vtyp y) { return eq(x, y) || x < y; }
24     bool ge(vtyp x, vtyp y) { return eq(x, y) || x > y; }
25
26     struct vec {

```

```

27     vtyp x, y;
28     vec() { x = y = 0; }
29     vec(vtyp _x, vtyp _y) { x = _x, y = _y; }
30
31     vec operator+(const vec V) const { return vec(x + V.x, y + V.y); }
32     vec operator-(const vec V) const { return vec(-x, -y); }
33     vec operator-(const vec V) const { return *this + (-V); }
34     vec operator*(const vtyp a) const { return vec(x * a, y * a); }
35     friend vec operator*(const vtyp a, const vec v) { return v * a; }
36     vec operator/(const vtyp a) const { return vec(x / a, y / a); }
37     operator bool() const { return !(isZero(x) && isZero(y)); }
38     bool operator==(const vec V) const { return bool(*this - V) == 0; }
39     bool operator!=(const vec V) const { return bool(*this - V) != 0; }
40     bool operator<(const vec V) const { return x == V.x ? y < V.y : x < V.x; }
41     bool operator>(const vec V) const { return x == V.x ? y > V.y : x > V.x; }
42
43     vtyp length() const { return sqrt(x * x + y * y); }
44     /**
45      * 方向角, 单位 rad
46      */
47     vtyp ang() const { return atan2(y, x); }
48     /**
49      * 方向向量
50      * @return 0向量或者一个单位向量
51      */
52     vec dir() const {
53         if (*this) {
54             vtyp len = length();
55             // vtyp ang = atan2(y,x); return vec(cos(ang), sin(ang));
56             return vec(x / len, y / len);
57         } else return vec(0, 0);
58     }
59     // void read(){ scanf("%Lf%Lf",&x,&y); }
60 };
61 typedef vec point;
62
63 vec r90_clockwise(const vec v) { // 顺时针旋转 90 度
64     return vec(v.y, -v.x);
65 }
66
67 struct line {
68     point p1, p2;
69     line(point _p1, point _p2) { p1 = _p1, p2 = _p2; }
70     line operator+(point p) { return line(p1 + p, p2 + p); } // shift
71     line operator-(point p) { return line(p1 - p, p2 - p); }
72     vec dir() const { return (p2 - p1).dir(); }
73 };
74 typedef line segment;
75
76 istream &operator>>(istream &in, vec &v) { return in >> v.x >> v.y, in; }
77 ifstream &operator>>(ifstream &in, vec &v) { return in >> v.x >> v.y, in; }
78 ostream &operator<<(ostream &out, const vec &v) {
79     return out << v.x << " " << v.y, out;

```



```

80     }
81     ofstream &operator<<(ofstream &out, const vec &v) {
82         return out << v.x << " " << v.y, out;
83     }
84     /**
85      * 点积
86      *  $a \cdot b == |a||b|\cos \theta$ 
87      */
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;
194     else return false;
195 }
196 }
197 /**
198  * 两个点的距离
199  */
200 vtyp distance(point a, point b) { return (b - a).length(); }
201 /**
202  * 点到直线的距离
203  */
204 vtyp distance(line L, point p) { return (p - projection(L, p)).length(); }
205 /**
206  * 两个线段的距离
207  */
208 vtyp distance(segment a, segment b) {
209     if (check_segment_intersection(a, b)) return 0;
210     vtyp res = distance(a.p1, b.p1);
211     res = min(res, distance(a.p1, b.p2));
212     res = min(res, distance(a.p2, b.p1));
213     res = min(res, distance(a.p2, b.p2));
214     point o;
215     if (o = projection(b, a.p1), on_segment(b, o)) res = min(res, distance(a.p1, o));
216     if (o = projection(b, a.p2), on_segment(b, o)) res = min(res, distance(a.p2, o));
217     if (o = projection(a, b.p1), on_segment(a, o)) res = min(res, distance(b.p1, o));
218     if (o = projection(a, b.p2), on_segment(a, o)) res = min(res, distance(b.p2, o));
219     return res;
220 }
221 /**
222  * 求简单多边形面积
223  * @param g 多边形顶点集
224  */
225 vtyp area(const vector<point> &g) {
226     vtyp res = 0;
227     for (unsigned i = 0; i < g.size(); i++) {
228         res += det(g[i], g[(i + 1) % g.size()]);
229     }
230     res /= 2;
231     return abs(res);
232 }
233 /**
234  * 判断是否是凸包
235  * @param g 多边形顶点集
236  */
237 bool is_convex(const vector<point> &g) {
238     if (g.size() < 3) return true;

```

```

239     int flag = 0;
240     for (unsigned i = 0; i < g.size(); i++) {
241         int j = (i + 1) % g.size(), k = (i + 2) % g.size();
242         vtyp sig = det(g[j] - g[i], g[k] - g[i]);
243         if (lt(sig, 0)) {
244             if (flag == 1) return false;
245             else flag = -1;
246         }
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 }
300 /**
301  * 判断点p与多边形的包含关系
302  * @param g 多边形顶点集
303  * @return 0 表示在多边形外, 1 表示在边上, 2表示在多边形内
304  */
305 int polygon_point_containment(vector<point> g, point p) {
306     line L(vec(p.x - 1, p.y), p); // 水平方向的射线
307     int cnt = 0;
308     for (unsigned i = 0; i < g.size(); i++) {
309         int j = (i + 1) % g.size();
310         line e(g[i], g[j]);
311         if (on_segment(e, p)) return 1;
312         if (parallel(L, e)) {
313             // do nothing.
314         } else if (online_front(L, g[i])) {
315             if (g[i].y > g[j].y) ++cnt;
316         } else if (online_front(L, g[j])) {
317             if (g[j].y > g[i].y) ++cnt;
318         } else {
319             point o = intersection(L, e);
320             if (on_segment(e, o) && online_front(L, o)) ++cnt;
321         }
322     }
323     if (cnt % 2) return 2;
324     return 0;
325 }
326
327 struct circle {
328     point o;
329     vtyp r;
330     circle() { r = 0; }
331     circle(point _o, vtyp _r) { o = _o, r = _r; }
332 };
333 /**
334  * 判断两个圆的位置关系 (切线数量)
335  * @param a 第一个圆
336  * @param b 第二个圆
337  * @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
338  */
339 int check_circle_intersection(circle a, circle b) {
340     vtyp d = distance(a.o, b.o);
341     if (gt(d, a.r + b.r)) return 4;
342     if (eq(d, a.r + b.r)) return 3;
343     if (gt(d, abs(a.r - b.r))) return 2;
344     if (eq(d, abs(a.r - b.r))) return 1;

```

```

345     return 0;
346 }
347 /**
348  * 判断圆和点的位置关系
349  * @return 0 表示包含, 1 表示在圆上, 2 表示在圆外
350  */
351 int check_circle_point_containment(circle c, point p) {
352     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
512  * @param angle 弓形所对的圆心角, 单位 rad
513  */
514 vtyp circular_segment_area(vtyp r, vtyp angle) {
515     return r * r * (angle - sin(angle)) / vtyp(2);
516 }
517 /**
518  * 求两个圆交面积
519  */
520 vtyp circle_intersection_area(circle c1, circle c2) {
521     vtyp ans = 0;
522     auto typ = check_circle_intersection(c1, c2);
523     if (typ <= 1) {
524         ans += PI * min(c1.r, c2.r) * min(c1.r, c2.r);
525     } else if (check_circle_intersection(c1, c2) < 3) {
526         auto pt = circle_intersection(c1, c2);
527         auto t1 = angle(pt.first - c1.o, pt.second - c1.o);
528         auto t2 = angle(pt.first - c2.o, pt.second - c2.o);
529         point p = intersection(line(c1.o, c2.o), line(pt.first, pt.second));
530         if (online_front(segment(c1.o, c2.o), p)) {
531             ans += circular_segment_area(c2.r, 2 * PI - t2);
532         } else {
533             ans += circular_segment_area(c2.r, t2);
534         }
535         if (online_front(segment(c2.o, c1.o), p)) {
536             ans += circular_segment_area(c1.r, 2 * PI - t1);
537         } else {
538             ans += circular_segment_area(c1.r, t1);
539         }
540     }
541     return ans;
542 }
543 } // namespace cg
544 using cg::circle;
545 using cg::line;
546 using cg::point;
547 using cg::segment;
548
549 // int main(){
550 //     circle c1, c2;
551 //     cin >> c1.o >> c1.r >> c2.o >> c2.r;
552 //     auto ans = cg::circle_intersection_area(c1, c2);
553 //     cout << setiosflags(ios::fixed) << setprecision(9) << ans << endl;
554 //     return 0;

```

555 | // }

code/geometry.cpp

7.2 欧拉序求 LCA

```

1  const int N = 1e5 + 5, M = 2e5 + 5;
2
3  struct qxx {
4      int nex, t;
5  };
6  qxx e[N * 2];
7  int h[N], le;
8  void add_path(int f, int t) { e[++le] = (qxx){h[f], t}, h[f] = le; }
9
10 int bg[N], ed[N], totime;
11 int dep[N];
12 int st[N * 2][20];
13
14 void dfs(int u, int p) {
15     dep[u] = dep[p] + 1;
16     bg[u] = ++totime;
17     st[totime][0] = u;
18     for (int i = h[u]; i; i = e[i].nex) {
19         const int v = e[i].t;
20         if (v == p) continue;
21         dfs(v, u);
22         st[++totime][0] = u;
23     }
24     ed[u] = totime;
25 }
26 void bin_exp() {
27     FOR(j, 1, 19) {
28         if ((1 << j) > totime) break;
29         int ilim = totime - (1 << j) + 1;
30         FOR(i, 1, ilim) {
31             st[i][j] = dep[st[i][j - 1]] < dep[st[i + (1 << j - 1)][j - 1]]
32                 ? st[i][j - 1]
33                 : st[i + (1 << j - 1)][j - 1];
34         }
35     }
36 }
37 int lca(int u, int v) {
38     int l = bg[u], r = bg[v];
39     if (l > r) l ^= r ^= l ^= r;
40     int j = log(r - l + 1) / log(2);
41     return dep[st[l][j]] < dep[st[r - (1 << j) + 1][j]] ? st[l][j]
42         : st[r - (1 << j) + 1][j];
43 }

```

code/euler_lca.cpp

7.3 IO 优化

```

1 namespace IO {
2     char nc() {
3         static char bf[100000], *p1 = bf, *p2 = bf;
4         return p1 == p2 && (p2 = (p1 = bf) + fread(bf, 1, 100000, stdin), p1 == p2)
5             ? EOF
6             : *p1++;
7     }
8     int rd() {
9         int res = 0;
10        char c = getchar();
11        while (!isdigit(c)) c = getchar();
12        while (isdigit(c)) res = res * 10 + c - '0', c = getchar();
13        return res;
14    }
15    int tmp[20], lt;
16    char OBF[100000], *OBP = OBF;
17    void flush() { fwrite(OBF, 1, OBP - OBF, stdout), OBP = OBF; }
18    void wrch(char x) {
19        if (OBP == OBF + 99999) flush();
20        *OBP = x, ++OBP;
21    }
22    void wr(int x) {
23        lt = 0;
24        while (x) tmp[++lt] = x % 10, x /= 10;
25        while (lt > 0) wrch(tmp[lt] + '0'), --lt;
26    }
27    void wr(long long x) {
28        lt = 0;
29        while (x) tmp[++lt] = x % 10, x /= 10;
30        while (lt > 0) wrch(tmp[lt] + '0'), --lt;
31    }
32    void wr(const char *s) {
33        while (*s) wrch(*s), ++s;
34    }
35    void wr(char x) { wrch(x); }
36 } // namespace IO
37 /*
38  * nc(): 读取下一个字符, 如果缓冲区没了就刷一次缓冲区
39  * rd(): 读入一个整数 (含义为 read %d
40  * flush(): 输出时把缓冲区清空。记得在程序结尾的时候flush一下
41  * wrch(): 输出单个字符
42  * wr(): 输出一个整数/LL/字符串/单个字符
43  */

```

code/io.cpp