

Heltion's CP-Teamplate

Heltion

November 7, 2019

Catalog

1	convolution	2
1.1	FFT in C	2
1.2	FFT in Fp	2
2	data structure	3
2.1	01Trie	3
2.2	persisting 01Trie	5
3	geometry	6
3.1	convex hull	6
4	linear algebra	6
4.1	matrix inverse in Fp	6
5	miscellaneous	7
5.1	check	7
5.2	makefile	8
6	number theory	8
6.1	Cipolla	8
6.2	Du's method	8
6.3	extended BSGS	9
6.4	extended CRT	10
6.5	extended Lucas	10
6.6	Min 25's method	11
6.7	Pollard Rho	12
7	numeric	13
7.1	ternary search with golden section	13

1 convolution

1.1 FFT in C

LibreOJ108: maxn = 100000, time = 117ms, memory = 11MB.

```
1 using R = double;
2 constexpr R PI = acos(-(R)1);
3 constexpr int maxn = 1 << 21;
4 struct C{
5     R a, b;
6     C operator + (const C& B) const{return {a + B.a, b + B.b};}
7     C operator - (const C& B) const{return {a - B.a, b - B.b};}
8     C operator * (const C& B) const{return {a * B.a - b * B.b, a * B.b + b * B.a};}
9     C operator / (const int& B) const{return {a / B, b / B};}
10    C conj() const{return {a, -b};}
11 };
12 void DFT(C* a, int n, bool idft = false){
13     static int r[maxn], on;
14     static C w[maxn];
15     if(on != n){
16         for(int i = 0; i < n; i += 1){
17             r[i] = i ? r[i >> 1] >> 1 | (i & 1 ? n >> 1 : 0) : 0;
18             w[i] = {cos(PI * i / n), sin(PI * i / n)};
19         }
20         on = n;
21     }
22     for(int i = 0; i < n; i += 1) if(i < r[i]) swap(a[i], a[r[i]]);
23     for(int m = 1; m <= n >> 1; m <<= 1)
24         for(int j = 0; j < n; j += m << 1)
25             for(int k = 0, p = 0; k < m; k += 1, p += n / m){
26                 C tmp = (idft ? w[p].conj() : w[p]) * a[j + k + m];
27                 a[j + k + m] = a[j + k] - tmp;
28                 a[j + k] = a[j + k] + tmp;
29             }
30     if(idft) for(int i = 0; i < n; i += 1) a[i] = a[i] / n;
31 }
32 int a[maxn], b[maxn], c[maxn];
33 C p[maxn];
34 int mul(int* a, int n, int* b, int m, int* c){
35     int k = 1;
36     while(k < n + m - 1) k <<= 1;
37     for(int i = 0; i < k; i += 1) p[i] = {i < n ? a[i] : 0., i < m ? b[i] : 0.};
38     DFT(p, k);
39     for(int i = 0; i < k; i += 1) p[i] = p[i] * p[i];
40     DFT(p, k, true);
41     for(int i = 0; i < n + m - 1; i += 1) c[i] = p[i].b * .5 + .5;
42     return n + m - 1;
43 }
```

1.2 FFT in Fp

LibreOJ108: maxn = 100000, time = 70ms, memory = 12MB.

Luogu3803: maxn = 1000000, time = 857ms, memory = 89MB.

```
1 using LL = long long;
2 constexpr LL mod = 998244353, G = 3;
3 constexpr int maxn = 1 << 21;
4 LL add(LL a, const LL& b){a += b; return a >= mod ? a - mod : a;}
5 LL sub(LL a, const LL& b){a -= b; return a < 0 ? a + mod : a;}
6 LL mul(const LL& a, const LL& b){return a * b % mod;}
7 LL power(LL a, LL r){
8     LL res = 1;
```

```

9     for(; r; r >>= 1, a = mul(a, a)) if(r & 1) res = mul(res, a);
10    return res;
11 }
12 void DFT(LL* a, int n, bool idft = false){
13     static int r[maxn], on;
14     static LL w[maxn], w_[maxn];
15     if(on != n){
16         r[1] = n >> 1; w[0] = w_[0] = 1; w[1] = power(G, (mod - 1) / n / 2); w_[1] =
            power(w[1], mod - 2);
17         for(int i = 2; i < n; i += 1){
18             r[i] = r[i >> 1] >> 1 | (i & 1 ? n >> 1 : 0);
19             w[i] = mul(w[i - 1], w[1]);
20             w_[i] = mul(w_[i - 1], w_[1]);
21         }
22         on = n;
23     }
24     for(int i = 0; i < n; i += 1) if(i < r[i]) swap(a[i], a[r[i]]);
25     for(int m = 1; m <= n >> 1; m <<= 1)
26         for(int j = 0; j < n; j += m << 1)
27             for(int k = 0, p = 0; k < m; k += 1, p += n / m){
28                 LL tmp = mul(idft ? w_[p] : w[p], a[j + k + m]);
29                 a[j + k + m] = sub(a[j + k], tmp);
30                 a[j + k] = add(a[j + k], tmp);
31             }
32     if(idft) for(int i = 0, _ = power(n, mod - 2); i < n; i += 1) a[i] = mul(a[i], _);
33 }
34 LL A[maxn], B[maxn], C[maxn];
35 int mul(LL* A, int n, LL* B, int m, LL* C){
36     int k = 1;
37     while(k < n + m - 1) k <<= 1;
38     DFT(A, k);
39     DFT(B, k);
40     for(int i = 0; i < k; i += 1) C[i] = mul(A[i], B[i]);
41     DFT(C, k, true);
42     return n + m - 1;
43 }

```

2 data structure

2.1 01Trie

Luogu3369: $n = 100000$, time = 67ms, memory = 5MB.

```

1
2 constexpr int maxn = 120000;
3 constexpr int maxb = 32;
4 struct _01Trie{
5     int count[maxn * maxb], ch[maxn * maxb][2], size = 1;
6     int in[maxb], out[maxb];
7     void encode(unsigned x){
8         x ^= 1U << 31;
9         for(int i = 0; i < maxb; i += 1) in[i] = x >> (maxb - 1 - i) & 1;
10    }
11    int decode(){
12        unsigned res = 0;
13        for(int i = 0; i < maxb; i += 1) res = res << 1 | out[i];
14        return res ^ 1U << 31;
15    }
16    void insert(int x){
17        encode(x);
18        int p = 1;

```

```

19     for(int c : in){
20         if(not ch[p][c]) ch[p][c] = size += 1;
21         count[p = ch[p][c]] += 1;
22     }
23 }
24 void erase(int x){
25     encode(x);
26     int p = 1;
27     for(int c : in) if(not count[p = ch[p][c]]) return;
28     p = 1;
29     for(int c : in) count[p = ch[p][c]] -= 1;
30 }
31 int rank(int x){
32     encode(x);
33     int p = 1, res = 1;
34     for(int c : in) res += c ? count[ch[p][0]] : 0, p = ch[p][c];
35     return res;
36 }
37 int k_th(int k){
38     int p = 1;
39     for(int &c : out)
40         if(k > count[ch[p][0]]) k -= count[ch[p][0]], p = ch[p][c = 1];
41         else p = ch[p][c = 0];
42     return decode();
43 }
44 int prev(int x){
45     encode(x);
46     int p = 1, q = 0;
47     for(int c : in){
48         if(c and count[ch[p][0]]) q = p;
49         p = ch[p][c];
50     }
51     if(not q) return -INT_MAX;
52     copy(in, in + maxb, out);
53     p = 1;
54     for(int& c : out)
55         if(q and p != q) p = ch[p][c];
56         else if(p == q) q = 0, p = ch[p][c = 0];
57         else p = ch[p][c = not not count[ch[p][1]]];
58     return decode();
59 }
60 int next(int x){
61     encode(x);
62     int p = 1, q = 0;
63     for(int c : in){
64         if(not c and count[ch[p][1]]) q = p;
65         p = ch[p][c];
66     }
67     if(not q) return INT_MAX;
68     copy(in, in + maxb, out);
69     p = 1;
70     for(int& c : out)
71         if(q and p != q) p = ch[p][c];
72         else if(p == q) q = 0, p = ch[p][c = 1];
73         else p = ch[p][c = not count[ch[p][0]]];
74     return decode();
75 }
76
77 };

```

2.2 persisting 01Trie

Luogu3835: $n = 500000$, time = 599ms, memory = 176MB.

```
1
2 constexpr int maxn = 540000;
3 constexpr int maxb = 32;
4 struct Persisting_01Trie{
5     int root[maxn], count[maxn * maxb], ch[maxn * maxb][2], size;
6     int in[maxb], out[maxb];
7     void encode(unsigned x){
8         x ^= 1U << 31;
9         for(int i = 0; i < maxb; i += 1) in[i] = x >> (maxb - 1 - i) & 1;
10    }
11    int decode(){
12        unsigned res = 0;
13        for(int i = 0; i < maxb; i += 1) res = res << 1 | out[i];
14        return res ^ 1U << 31;
15    }
16    void insert(int u, int v, int x){
17        encode(x);
18        int pu = root[u], pv = root[v] = size += 1;
19        for(int c : in){
20            ch[pv][c ^ 1] = ch[pu][c ^ 1];
21            pu = ch[pu][c];
22            pv = ch[pv][c] = size += 1;
23            count[pv] = count[pu] + 1;
24        }
25    }
26    void erase(int u, int v, int x){
27        encode(x);
28        int pu = root[u], pv = root[v] = pu;
29        for(int c : in) if(not count[pu = ch[pu][c]]) return;
30        pu = root[u], pv = root[v] = size += 1;
31        for(int c : in){
32            ch[pv][c ^ 1] = ch[pu][c ^ 1];
33            pu = ch[pu][c];
34            pv = ch[pv][c] = size += 1;
35            count[pv] = count[pu] - 1;
36        }
37    }
38    int rank(int u, int v, int x){
39        encode(x);
40        int pu = root[v] = root[u], res = 1;
41        for(int c : in) res += c ? count[ch[pu][0]] : 0, pu = ch[pu][c];
42        return res;
43    }
44    int k_th(int u, int v, int k){
45        int pu = root[v] = root[u];
46        for(int &c : out)
47            if(k > count[ch[pu][0]]) k -= count[ch[pu][0]], pu = ch[pu][c = 1];
48            else pu = ch[pu][c = 0];
49        return decode();
50    }
51    int prev(int u, int v, int x){
52        encode(x);
53        int pu = root[v] = root[u], pv = 0;
54        for(int c : in){
55            if(c and count[ch[pu][0]]) pv = pu;
56            pu = ch[pu][c];
57        }
58        if(not pv) return -INT_MAX;
59        copy(in, in + maxb, out);
```

```

60     pu = root[u];
61     for(int& c : out)
62         if(pv and pu != pv) pu = ch[pu][c];
63         else if(pu == pv) pv = 0, pu = ch[pu][c = 0];
64         else pu = ch[pu][c = not not count[ch[pu][1]]];
65     return decode();
66 }
67 int next(int u, int v, int x){
68     encode(x);
69     int pu = root[v] = root[u], pv = 0;
70     for(int c : in){
71         if(not c and count[ch[pu][1]]) pv = pu;
72         pu = ch[pu][c];
73     }
74     if(not pv) return INT_MAX;
75     copy(in, in + maxb, out);
76     pu = root[u];
77     for(int& c : out)
78         if(pv and pu != pv) pu = ch[pu][c];
79         else if(pu == pv) pv = 0, pu = ch[pu][c = 1];
80         else pu = ch[pu][c = not count[ch[pu][0]]];
81     return decode();
82 }
83
84 };

```

3 geometry

3.1 convex hull

```

1 using R = double;
2 constexpr int maxn = 100000;
3 struct P{
4     R x, y;
5     P operator - (const P& B){return {x - B.x, y - B.y};}
6 }p[maxn], h[maxn + 1];
7 R cross(const P& A, const P& B){return A.x * B.y - A.y * B.x;}
8 int convex_hull(P *p, P* h, int n){
9     sort(p, p + n, [](const P& A, const P& B){
10         return A.x == B.x ? A.y < B.y : A.x < B.x;
11     });
12     int hn = 0;
13     for(int i = 0; i < n; i += 1){
14         while(hn >= 2 and cross(p[i] - h[hn - 2], h[hn - 1] - h[hn - 2]) >= 0) hn -= 1;
15         h[hn ++] = p[i];
16     }
17     int tmp = hn;
18     for(int i = n - 1; ~i; i -= 1){
19         while(hn > tmp and cross(p[i] - h[hn - 2], h[hn - 1] - h[hn - 2]) >= 0) hn -=
20             1;
21         h[hn ++] = p[i];
22     }
23     return hn - 1;

```

4 linear algebra

4.1 matrix inverse in \mathbb{F}_p

```

1 using LL = long long;
2 constexpr int maxn = 400;
3 constexpr LL mod = 1000000007;
4 LL A[maxn][maxn], B[maxn][maxn];
5 LL power(LL a, LL r){
6     LL res = 1;
7     for(; r; r >>= 1, a = a * a % mod)
8         if(r & 1) res = res * a % mod;
9     return res;
10 }
11 void sub(LL &a, LL b){
12     a -= b;
13     if(a < 0) a += mod;
14 }
15 bool matrix_inverse(int n, LL A[][maxn], LL B[][maxn]){
16
17     for(int i = 0; i < n; i += 1)
18         for(int j = 0; j < n; j += 1) B[i][j] = i == j;
19     for(int i = 0; i < n; i += 1){
20         int k = -1;
21         for(int j = i; j < n and not~k; j += 1) if(A[j][i]) k = j;
22         if(not~k) return false;
23         if(k != i) {
24             for(int j = i; j < n; j += 1) swap(A[i][j], A[k][j]);
25             for(int j = 0; j <= i; j += 1) swap(B[i][j], B[k][j]);
26         }
27         LL inv = power(A[i][i], mod - 2);
28         for(int j = i; j < n; j += 1) A[i][j] = A[i][j] * inv % mod;
29         for(int j = 0; j <= i; j += 1) B[i][j] = B[i][j] * inv % mod;
30         for(int j = 0; j < n; j += 1) if(j != i){
31             LL tmp = A[j][i];
32
33             for(int k = i; k < n; k += 1) sub(A[j][k], tmp * A[i][k] % mod);
34             for(int k = 0; k <= i; k += 1) sub(B[j][k], tmp * B[i][k] % mod);
35         }
36     }
37     return true;
38 }

```

5 miscellaneous

5.1 check

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(int argc, char** argv){
4     if(argc > 1) srand(atoi(argv[1]));
5     //...
6 }
7
8 make $1
9 make $2
10 make $3
11 i=0
12 while true
13 do
14     ./ $3 $i > $1.in
15     ./ $1 < $1.in > $1.out
16     ./ $2 < $1.in > $2.out
17     if(diff $1.out $2.out)

```

```

18     then
19         echo "OK at "$i
20     else
21         echo "WA at "$i
22         break
23     fi
24     let "i++"
25 done

```

5.2 makefile

```

1 %:%.cpp
2     g++ $< -o $@ -O2 -std=c++14

```

6 number theory

6.1 Cipolla

```

1 using LL = long long;
2 LL power(LL a, LL r, const LL& p){
3     LL res = 1;
4     for(; r; r >>= 1, a = a * a % p)
5         if(r & 1) res = res * a % p;
6     return res;
7 }
8 struct Fp2{
9     LL x, y, w, p;
10    Fp2 operator * (const Fp2& B) const{
11        assert(w == B.w and p == B.p);
12        Fp2 res = {0, 0, w, p};
13        res.x = (x * B.x + y * B.y % p * w) % p;
14        res.y = (x * B.y + y * B.x) % p;
15        return res;
16    }
17 };
18 LL sqrt(LL n, LL p){
19     if(n == 0) return 0;
20     if(power(n, (p ^ 1) >> 1, p) != 1) return -1;
21     LL x, w;
22     do x = rand(), w = (x * x + p - n) % p;
23     while(power(w, (p ^ 1) >> 1, p) == 1);
24     Fp2 a = {x, 1, w, p}, res = {1, 0, w, p};
25     for(LL r = (p + 1) >> 1; r; r >>= 1, a = a * a)
26         if(r & 1) res = res * a;
27     return res.x;
28 }

```

6.2 Du's method

```

1 using LL = long long;
2 constexpr int maxp = 120000;
3 constexpr int maxn = 1 << 22;
4 constexpr LL mod = 1000000007;
5 int p[maxn], ps[maxp], pg[maxp], pn, m;
6 bool cp[maxn];
7 int phi[maxn];
8 LL v[maxp << 1], s[maxp << 1], sphi[maxn], on;
9 LL add(LL x, const LL& y){x += y; return x >= mod ? x - mod : x;}

```

```

10 LL sub(LL x, const LL& y){x -= y; return x < 0 ? x + mod : x;}
11 LL id(LL n){return n <= m ? ps[n] : pg[on / n];}
12 LL sfg(LL n){n %= mod; return n * (n + 1) / 2 % mod;}//...
13 LL sg(LL n){return n % mod;}//...
14 LL S(LL n){
15     m = sqrt(on = n) + 1;
16     phi[1] = sph[1] = 1;
17     for(int i = 2; i < maxn; i += 1){
18         if(not cp[i]) p[pn++] = i, phi[i] = i - 1;
19         for(int j = 0; j < pn and i * p[j] < maxn; j += 1){
20             cp[i * p[j]] = true;
21             if(i % p[j] == 0){
22                 phi[i * p[j]] = phi[i] * p[j];
23                 break;
24             }
25             phi[i * p[j]] = phi[i] * (p[j] - 1);
26         }
27         sph[i] = add(sph[i - 1], phi[i]);
28     }
29     int gn = 0;
30     for(LL L = 1, R; L <= n; L = R + 1){
31         R = n / (n / L);
32         if(n / L <= m) ps[n / L] = gn += 1;
33         else pg[R] = gn += 1;
34         v[gn] = n / L;
35     }
36     for(int i = gn; i; i -= 1)
37         if(v[i] < maxn) s[i] = sph[v[i]];
38     else{
39         s[i] = sfg(v[i]);
40         for(LL L = 2, R; L <= v[i]; L = R + 1){
41             R = v[i] / (v[i] / L);
42             s[i] = sub(s[i], sub(sg(R), sg(L - 1)) * s[id(v[i] / L)] % mod);
43         }
44     }
45     return s[1];
46 }

```

6.3 extended BSGS

```

1 LL gcd(LL a, LL b){
2     return b ? gcd(b, a % b) : a;
3 }
4 LL exBSGS(LL a, LL b, LL p){
5     LL res = 0, k = 1, d;
6     if(b == 1) return 0;
7     if(not a) return b ? -1 : 1;
8     for(;k != b and (d = gcd(a, p)) != 1; res += 1){
9         if(b % d) return -1;
10        p /= d, b /= d;
11        k = k * (a / d) % p;
12    }
13    if(k == b) return res;
14    unordered_map<LL, LL> mp;
15    LL x = 1, y, M = sqrt(p) + 1;
16    for(int i = 0; i < M; i += 1, x = x * a % p) mp[b * x % p] = i;
17    y = k * x % p;
18    for(int i = 1; i <= M; i += 1, y = y * x % p)
19        if(mp.count(y)) return res + i * M - mp[y];
20    return -1;
21 }

```

6.4 extended CRT

```
1 LL exgcd(LL a, LL b, LL& x, LL& y){
2     if(not b) return x = 1, y = 0, a;
3     LL d = exgcd(b, a % b, x, y), t = x;
4     return x = y, y = t - a / b * y, d;
5 }
6 bool excrt(int n, LL* m, LL* r, LL& M, LL& R){
7     __int128 IM = 1, IR = 0;
8     for(int i = 0; i < n; i += 1){
9         LL x, y, d = exgcd(IM, m[i], x, y);
10        if((r[i] - IR) % d) return false;
11        IR += ((r[i] - IR) * x / d) % (m[i] / d) * IM;
12        IM = IM / d * m[i];
13        IR = (IR % IM + IM) % IM;
14    }
15    M = IM;
16    R = IR;
17    return true;
18 }
```

6.5 extended Lucas

```
1 using LL = long long;
2 struct PLL{LL x, y;};
3 LL power(LL a, LL r, LL p){
4     LL res = 1;
5     for(; r; r >>= 1, a = a * a % p) if(r & 1) res = res * a % p;
6     return res;
7 }
8 LL inverse(LL a, PLL p){
9     return power(a, p.y / p.x * (p.x - 1) - 1, p.y);
10 }
11 PLL fact(LL n, PLL p){
12     if(n == 0) return {0, 1};
13     PLL res = fact(n / p.x, p);
14     res.x += n / p.x;
15     for(LL i = 1; i <= n % p.y; i += 1) if(i % p.x) res.y = res.y * i % p.y;
16     if((p.x == 4 or p.y != 2) and ((n / p.y) & 1)) res.y = p.y - res.y;
17     return res;
18 }
19 LL exLucas(LL m, LL n, PLL p){
20     PLL fm = fact(m, p), fn = fact(n, p), fm_n = fact(m - n, p);
21     LL res = fm.y * inverse(fn.y, p) % p.y * inverse(fm_n.y, p) % p.y;
22     for(int i = 0; i < fm.x - fn.x - fm_n.x and res; i += 1) res = res * p.x % p.y;
23     return res;
24 }
25 LL exLucas(LL m, LL n, LL p){
26     vector<PLL> vp;
27     LL np = p;
28     for(int i = 2; i * i <= np; i += 1) if(np % i == 0){
29         vp.push_back({i, 1});
30         for(; np % i == 0; np /= i) vp.back().y *= i;
31     }
32     if(np > 1) vp.push_back({np, np});
33     vector<LL> r;
34     for(auto p : vp) r.push_back(exLucas(m, n, p));
35     LL res = 0;
36     for(int i = 0; i < (int)r.size(); i += 1)
37         res = (res + r[i] * (p / vp[i].y) % p * inverse(p / vp[i].y, vp[i])) % p;
38     return res;
39 }
```

6.6 Min 25's method

```

1 using LL = long long;
2 constexpr int maxn = 120000;
3 constexpr LL mod = 1000000007;
4 int p[maxn], pn, ps[maxn], pg[maxn], m;
5 bool cp[maxn];
6 LL sp[3][maxn], g[3][maxn << 1], v[maxn << 1], on;
7 LL add(LL x, const LL& y){x += y; return x >= mod ? x - mod : x;}
8 LL sub(LL x, const LL& y){x -= y; return x < 0 ? x + mod : x;}
9 LL s1(LL n){n %= mod; n = n * (n + 1) / 2 % mod; return n ? n - 1 : mod - 1;}
10 LL s2(LL n){n %= mod; n = n * (n + 1) % mod * (2 * n + 1) % mod * 166666668 % mod;
    return n ? n - 1 : mod - 1;}
11 LL id(LL n){return n <= m ? ps[n] : pg[on / n];}
12 LL S(LL n, int k){
13     if(k == 0){
14         m = sqrt(on = n) + 1;
15         for(int i = 2; i <= m; i += 1){
16             if(not cp[i]){
17                 p[pn += 1] = i;
18                 sp[1][pn] = add(sp[1][pn - 1], i);
19                 sp[2][pn] = add(sp[2][pn - 1], (LL)i * i % mod);
20                 //...
21             }
22             for(int j = 1; j <= pn and i * p[j] <= m; j += 1){
23                 cp[i * p[j]] = true;
24                 if(i % p[j] == 0) break;
25             }
26         }
27         int gn = 0;
28
29         for(LL L = 1, R; L <= n; L = R + 1){
30             R = n / (n / L);
31             if(n / L > m) pg[R] = gn += 1;
32             else ps[n / L] = gn += 1;
33             v[gn] = n / L;
34             g[1][gn] = s1(v[gn]);
35             g[2][gn] = s2(v[gn]);
36             //...
37         }
38
39         for(int i = 1; i <= pn; i += 1){
40             LL p2 = (LL)p[i] * p[i];
41             for(int j = 1; j <= gn and p2 <= v[j]; j += 1){
42                 int x = id(v[j] / p[i]);
43                 g[1][j] = sub(g[1][j], p[i] * sub(g[1][x], sp[1][i - 1]) % mod);
44                 g[2][j] = sub(g[2][j], p2 % mod * sub(g[2][x], sp[2][i - 1]) % mod);
45                 //...
46             }
47         }
48     }
49     if(p[k] >= n) return 0;
50     int x = id(n);
51     LL res = sub(sub(g[2][x], sp[2][k]), sub(g[1][x], sp[1][k]));
52     //...
53     for(int i = k + 1; i <= pn and (LL)p[i] * p[i] <= n; i += 1){
54         LL a = p[i], b = a;
55         for(int r = 1; a <= n; r += 1, a *= p[i], b = b * p[i] % mod)
56             res = add(res, b * (b - 1) % mod * (S(n / a, i) + (r != 1)) % mod);

```

```

57     //...
58 }
59 return add(res, not k);
60 }

```

6.7 Pollard Rho

Miller Rabin: LibreOJ143: $T = 100000, n = 10^{18}$, time = 346ms.

Pollard Rho: Luogu4718: $T = 350, n = 10^{18}$, time = 1160ms.

```

1 using LL = long long;
2 LL gcd(LL a, LL b){
3     return b ? gcd(b, a % b) : a;
4 }
5 LL mul(LL a, LL b, LL p){
6     return(__int128)a * b % p;
7     //return (a * b - (LL)(a / (long double)p * b + 1e-3) * p + p) % p;
8 }
9 LL power(LL a, LL r, LL p){
10    LL res = 1;
11    for(; r; a = mul(a, a, p), r >>= 1)
12        if(r & 1) res = mul(res, a, p);
13    return res;
14 }
15 bool Miller_Rabin(LL n){
16    static LL p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
17    if(n == 1) return false;
18    if(n == 2) return true;
19    if(not(n & 1)) return false;
20    LL d = n - 1, r = 0;
21    for(;not(d & 1); d >>= 1) r += 1;
22    bool res = true;
23    for(int i = 0; i < 9 and p[i] < n and res; i += 1){
24        LL x = power(p[i], d, n);
25        if(x == 1 or x == n - 1) continue;
26        for(int j = 1; j < r; j += 1){
27            x = mul(x, x, n);
28            if(x == n - 1) break;
29        }
30        if(x != n - 1) res = false;
31    }
32    return res;
33 }
34 void Pollard_Rho(LL n){
35    if(n == 1) return;
36    if(Miller_Rabin(n)){
37        //n is prime factor
38        return;
39    }
40    LL d = n;
41    while(d == n){
42        d = 1;
43        for(LL k = 1, y = 0, x = 0, s = 1, c = rand() % n; d == 1; k <<= 1, y = x, s = 1){
44            for(int i = 1; i <= k; i += 1){
45                x = (mul(x, x, n) + c) % n;
46                s = mul(s, abs(x - y), n);
47                if(not(i % 127) or i == k){
48                    d = gcd(s, n);
49                    if(d != 1) break;
50                }
51            }

```

```

52     }
53 }
54 Pollard_Rho(d);
55 Pollard_Rho(n / d);
56 }

```

7 numeric

7.1 ternary search with golden section

```

1 constexpr double Phi = (sqrt(5) - 1) / 2;
2 constexpr int step = 50;
3 double f(double x){
4     //cost function
5 }
6 void ternary_search(double f(double), double L, double R){
7     //find minimum
8     double mL = Phi * L + (1 - Phi) * R;
9     double mR = Phi * R + (1 - Phi) * L;
10    double fmL = f(mL), fmR = f(mR);
11    for(int i = 0; i < step; i += 1)
12        if(fmL > fmR){
13            L = mL;
14            mL = mR;
15            fmL = fmR;
16            fmR = f(mR = Phi * R + (1 - Phi) * L);
17        }
18        else{
19            R = mR;
20            mR = mL;
21            fmR = fmL;
22            fmL = f(mL = Phi * L + (1 - Phi) * R);
23        }
24 }

```