tonosama Library

template

hash.sh

```
# 使い方: sh hash.sh -> コピペ -> Ctrl + D
# コメント・空白・改行を削除して md5 でハッシュする
g++ -dD -E -fpreprocessed - | tr -d '[:space:]' | md5sum | cut
-c-6
```

settings.sh

```
# CLion の設定
Settings → Build → CMake → Reload CMake Project
add_compile_options(-D_GLIBCXX_DEBUG)
# Caps Lock を Ctrl に変更
setxkbmap -option ctrl:nocaps
```

template.hpp

md5: 365d7f

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll INF = LLONG_MAX / 4;
#define rep(i, a, b) for (ll i = a; i < (b); i++)
#define all(a) begin(a), end(a)
ll sz(const auto& a) { return size(a); }
bool chmin(auto& a, auto b) {
   if (a <= b) return 0;
   a = b;
   return 1;
}
bool chmax(auto& a, auto b) {
   if (a >= b) return 0;
   a = b;
   return 1;
}

int main() {
   cin.tie(0)->sync_with_stdio(0);
   // your code here...
}
```

data-structure

BIT.hpp

md5: d8ec49

```cpp
struct BIT {
   vector<ll> a;
   BIT(ll n) : a(n + 1) {}
   void add(ll p, ll x) {  // A[p] += x
      p++;
      while (p < sz(a)) {
         a[p] += x;
         p += p & -p;
      }
   }
   ll sum(ll r) {
      ll s = 0;
      while (r > 0) {
         s += a[r];
         r -= r & -r;
      }
      return s;
   }
   ll sum(ll l, ll r) {  // sum of A[l, r)
      return sum(r) - sum(l);
   }
};
```

math

modint

BarrettReduction.hpp

md5: b4bd2c

```cpp
using u64 = uint64_t;
struct Barrett {  // mod < 2^32
   u64 m, im;
   Barrett(u64 mod) : m(mod), im(-1ULL / m + 1) {}
   // input: a * b < 2^64, output: a * b % mod
   u64 mul(u64 a, u64 b) const {
      a *= b;
      u64 x = ((__uint128_t)a * im) >> 64;
      a -= x * m;
      if ((ll)a < 0) a += m;
      return a;
   }
};
```

modint.hpp

md5: ade70b

```cpp
const ll mod = 998244353;
struct mm {
   ll x;
   mm(ll x_ = 0) : x(x_ % mod) {
      if (x < 0) x += mod;
   }
   friend mm operator+(mm a, mm b) { return a.x + b.x; }
   friend mm operator-(mm a, mm b) { return a.x - b.x; }
   friend mm operator*(mm a, mm b) { return a.x * b.x; }
   friend mm operator/(mm a, mm b) { return a * b.inv(); }
   // 4 行コピペ  Alt + Shift + クリックで複数カーソル
   friend mm& operator+=(mm& a, mm b) { return a = a.x + b.x; }
   friend mm& operator-=(mm& a, mm b) { return a = a.x - b.x; }
   friend mm& operator*=(mm& a, mm b) { return a = a.x * b.x; }
   friend mm& operator/=(mm& a, mm b) { return a = a * b.inv();
}
   mm inv() const { return pow(mod - 2); }
   mm pow(ll b) const {
      mm a = *this, c = 1;
      while (b) {
         if (b & 1) c *= a;
         a *= a;
         b >>= 1;
      }
      return c;
   }
};
```

FPS

FFT.hpp

md5: 10993d

```cpp
constexpr ll pow_mod_constexpr(ll x, ll n, int m) {
   if (m == 1) return 0;
   uint _m = (uint)(m);
   ull r = 1;
   ull y = x % m;
   while (n) {
      if (n & 1) r = (r * y) % _m;
      y = (y * y) % _m;
      n >>= 1;
   }
   return r;
}

constexpr bool is_prime_constexpr(int n) {
   if (n <= 1) return false;
```

```
   if (n == 2 || n == 7 || n == 61) return true;
   if (n % 2 == 0) return false;
   ll d = n - 1;
   while (d % 2 == 0) d /= 2;
   constexpr ll bases[3] = {2, 7, 61};
   for (ll a : bases) {
      ll t = d;
      ll y = pow_mod_constexpr(a, t, n);
      while (t != n - 1 && y != 1 && y != n - 1) {
         y = y * y % n;
         t <<= 1;
      }
      if (y != n - 1 && t % 2 == 0) { return false; }
   }
   return true;
}
template<int n> constexpr bool is_prime =
is_prime_constexpr(n);

constexpr pair<ll, ll> inv_gcd(ll a, ll b) {
   a = safe_mod(a, b);
   if (a == 0) return {b, 0};

   ll s = b, t = a;
   ll m0 = 0, m1 = 1;

   while (t) {
      ll u = s / t;
      s -= t * u;
      m0 -= m1 * u;  // |m1 * u| <= |m1| * s <= b

      auto tmp = s;
      s = t;
      t = tmp;
      tmp = m0;
      m0 = m1;
      m1 = tmp;
   }
   if (m0 < 0) m0 += b / s;
   return {s, m0};
}

ull floor_sum_unsigned(ull n, ull m, ull a, ull b) {
   ull ans = 0;
   while (true) {
      if (a >= m) {
         ans += n * (n - 1) / 2 * (a / m);
         a %= m;
      }
      if (b >= m) {
         ans += n * (b / m);
         b %= m;
      }

      ull y_max = a * n + b;
      if (y_max < m) break;
      n = (ull)(y_max / m);
      b = (ull)(y_max % m);
      swap(m, a);
   }
   return ans;
}

template<class mint, int g =
internal::primitive_root<mint::mod()>,
internal::is_static_modint_t<mint>* = nullptr>
struct fft_info {
   static constexpr int rank2 =
countr_zero_constexpr(mint::mod() - 1);
   array<mint, rank2 + 1> root;   // root[i]^(2^i) == 1
   array<mint, rank2 + 1> iroot;  // root[i] * iroot[i] == 1

   array<mint, max(0, rank2 - 2 + 1)> rate2;
   array<mint, max(0, rank2 - 2 + 1)> irate2;

   array<mint, max(0, rank2 - 3 + 1)> rate3;
   array<mint, max(0, rank2 - 3 + 1)> irate3;

   fft_info() {
      root[rank2] = mint(g).pow((mint::mod() - 1) >> rank2);
      iroot[rank2] = root[rank2].inv();
      for (int i = rank2 - 1; i >= 0; i--) {
         root[i] = root[i + 1] * root[i + 1];
         iroot[i] = iroot[i + 1] * iroot[i + 1];
      }

      {
         mint prod = 1, iprod = 1;
         for (int i = 0; i <= rank2 - 2; i++) {
            rate2[i] = root[i + 2] * prod;
            irate2[i] = iroot[i + 2] * iprod;
            prod *= iroot[i + 2];
            iprod *= root[i + 2];
         }
      }
      {
         mint prod = 1, iprod = 1;
         for (int i = 0; i <= rank2 - 3; i++) {
            rate3[i] = root[i + 3] * prod;
            irate3[i] = iroot[i + 3] * iprod;
            prod *= iroot[i + 3];
            iprod *= root[i + 3];
         }
      }
   }
};

template<class mint, internal::is_static_modint_t<mint>* =
nullptr> void butterfly(vector<mint>& a) {
   int n = int(a.size());
   int h = internal::countr_zero((uint)n);

   static const fft_info<mint> info;

   int len = 0;  // a[i, i+(n>>len), i+2*(n>>len), ..] is
transformed
   while (len < h) {
      if (h - len == 1) {
         int p = 1 << (h - len - 1);
         mint rot = 1;
         for (int s = 0; s < (1 << len); s++) {
            int offset = s << (h - len);
            for (int i = 0; i < p; i++) {
               auto l = a[i + offset];
               auto r = a[i + offset + p] * rot;
               a[i + offset] = l + r;
               a[i + offset + p] = l - r;
            }
            if (s + 1 != (1 << len)) rot *=
info.rate2[countr_zero(~(uint)(s))];
         }
         len++;
      } else {
         int p = 1 << (h - len - 2);
         mint rot = 1, imag = info.root[2];
         for (int s = 0; s < (1 << len); s++) {
            mint rot2 = rot * rot;
            mint rot3 = rot2 * rot;
            int offset = s << (h - len);
            for (int i = 0; i < p; i++) {
               auto mod2 = 1ULL * mint::mod() * mint::mod();
               auto a0 = 1ULL * a[i + offset].val();
               auto a1 = 1ULL * a[i + offset + p].val() *
rot.val();
               auto a2 = 1ULL * a[i + offset + 2 * p].val() *
rot2.val();
               auto a3 = 1ULL * a[i + offset + 3 * p].val() *
rot3.val();
               auto a1na3imag = 1ULL * mint(a1 + mod2 -
a3).val() * imag.val();
               auto na2 = mod2 - a2;
               a[i + offset] = a0 + a2 + a1 + a3;
               a[i + offset + 1 * p] = a0 + a2 + (2 * mod2 -
(a1 + a3));
               a[i + offset + 2 * p] = a0 + na2 + a1na3imag;
               a[i + offset + 3 * p] = a0 + na2 + (mod2 -
a1na3imag);
            }
            if (s + 1 != (1 << len)) rot *=
info.rate3[countr_zero(~(uint)(s))];
         }
         len += 2;
      }
   }
}
```

```
template<class mint, internal::is_static_modint_t<mint>* =
nullptr> void butterfly_inv(vector<mint>& a) {
   int n = int(a.size());
   int h = internal::countr_zero((uint)n);

   static const fft_info<mint> info;

   int len = h;  // a[i, i+(n>>len), i+2*(n>>len), ..] is
transformed
   while (len) {
      if (len == 1) {
         int p = 1 << (h - len);
         mint irot = 1;
         for (int s = 0; s < (1 << (len - 1)); s++) {
            int offset = s << (h - len + 1);
            for (int i = 0; i < p; i++) {
               auto l = a[i + offset];
               auto r = a[i + offset + p];
               a[i + offset] = l + r;
               a[i + offset + p] = (ull)(mint::mod() + l.val()
- r.val()) * irot.val();
               ;
            }
            if (s + 1 != (1 << (len - 1))) irot *=
info.irate2[countr_zero(~(uint)(s))];
         }
         len--;
      } else {
         int p = 1 << (h - len);
         mint irot = 1, iimag = info.iroot[2];
         for (int s = 0; s < (1 << (len - 2)); s++) {
            mint irot2 = irot * irot;
            mint irot3 = irot2 * irot;
            int offset = s << (h - len + 2);
            for (int i = 0; i < p; i++) {
               auto a0 = 1ULL * a[i + offset + 0 * p].val();
               auto a1 = 1ULL * a[i + offset + 1 * p].val();
               auto a2 = 1ULL * a[i + offset + 2 * p].val();
               auto a3 = 1ULL * a[i + offset + 3 * p].val();

               auto a2na3iimag = 1ULL * mint((mint::mod() + a2
- a3) * iimag.val()).val();

               a[i + offset] = a0 + a1 + a2 + a3;
               a[i + offset + 1 * p] = (a0 + (mint::mod() - a1)
+ a2na3iimag) * irot.val();
               a[i + offset + 2 * p] = (a0 + a1 + (mint::mod()
- a2) + (mint::mod() - a3)) * irot2.val();
               a[i + offset + 3 * p] = (a0 + (mint::mod() - a1)
+ (mint::mod() - a2na3iimag)) * irot3.val();
            }
            if (s + 1 != (1 << (len - 2))) irot *=
info.irate3[countr_zero(~(uint)(s))];
         }
         len -= 2;
      }
   }
}

template<class mint, internal::is_static_modint_t<mint>* =
nullptr>
vector<mint> convolution_naive(const vector<mint>& a, const
vector<mint>& b) {
   int n = int(a.size()), m = int(b.size());
   vector<mint> ans(n + m - 1);
   if (n < m) {
      for (int j = 0; j < m; j++) {
         for (int i = 0; i < n; i++) { ans[i + j] += a[i] *
b[j]; }
      }
   } else {
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < m; j++) { ans[i + j] += a[i] *
b[j]; }
      }
   }
   return ans;
}

template<class mint, internal::is_static_modint_t<mint>* =
nullptr>
vector<mint> convolution_fft(vector<mint> a, vector<mint> b) {
   int n = int(a.size()), m = int(b.size());
   int z = (int)internal::bit_ceil((uint)(n + m - 1));
   a.resize(z);
   internal::butterfly(a);
   b.resize(z);
   internal::butterfly(b);
   for (int i = 0; i < z; i++) { a[i] *= b[i]; }
   internal::butterfly_inv(a);
   a.resize(n + m - 1);
   mint iz = mint(z).inv();
   for (int i = 0; i < n + m - 1; i++) a[i] *= iz;
   return a;
}

template<class mint, internal::is_static_modint_t<mint>* =
nullptr>
vector<mint> convolution(const vector<mint>& a, const
vector<mint>& b) {
   int n = int(a.size()), m = int(b.size());
   if (!n || !m) return {};

   int z = (int)internal::bit_ceil((uint)(n + m - 1));
   assert((mint::mod() - 1) % z == 0);

   if (min(n, m) <= 60) return convolution_naive(a, b);
   return internal::convolution_fft(a, b);
}

vector<ll> convolution_ll(const vector<ll>& a, const
vector<ll>& b) {
   int n = int(a.size()), m = int(b.size());
   if (!n || !m) return {};

   static constexpr ull MOD1 = 754974721;  // 2^24
   static constexpr ull MOD2 = 167772161;  // 2^25
   static constexpr ull MOD3 = 469762049;  // 2^26
   static constexpr ull M2M3 = MOD2 * MOD3;
   static constexpr ull M1M3 = MOD1 * MOD3;
   static constexpr ull M1M2 = MOD1 * MOD2;
   static constexpr ull M1M2M3 = MOD1 * MOD2 * MOD3;

   static constexpr ull i1 = internal::inv_gcd(MOD2 * MOD3,
MOD1).second;
   static constexpr ull i2 = internal::inv_gcd(MOD1 * MOD3,
MOD2).second;
   static constexpr ull i3 = internal::inv_gcd(MOD1 * MOD2,
MOD3).second;

   static constexpr int MAX_AB_BIT = 24;
   static_assert(MOD1 % (1ull << MAX_AB_BIT) == 1, "MOD1 isn't
enough to support an array length of 2^24.");
   static_assert(MOD2 % (1ull << MAX_AB_BIT) == 1, "MOD2 isn't
enough to support an array length of 2^24.");
   static_assert(MOD3 % (1ull << MAX_AB_BIT) == 1, "MOD3 isn't
enough to support an array length of 2^24.");
   assert(n + m - 1 <= (1 << MAX_AB_BIT));

   auto c1 = convolution<MOD1>(a, b);
   auto c2 = convolution<MOD2>(a, b);
   auto c3 = convolution<MOD3>(a, b);

   vector<ll> c(n + m - 1);
   for (int i = 0; i < n + m - 1; i++) {
      ull x = 0;
      x += (c1[i] * i1) % MOD1 * M2M3;
      x += (c2[i] * i2) % MOD2 * M1M3;
      x += (c3[i] * i3) % MOD3 * M1M2;
      ll diff = c1[i] - internal::safe_mod((ll)(x), (ll)
(MOD1));
      if (diff < 0) diff += MOD1;
      static constexpr ull offset[5] = {0, 0, M1M2M3, 2 *
M1M2M3, 3 * M1M2M3};
      x -= offset[diff % 5];
      c[i] = x;
   }

   return c;
}
```

graph

flow

string

geometry

memo