

## ccpc桂林更新 2023. 10. 26

## fft

```

1  const int N=1e6+10;
2  const long double Pi=acos(-1);
3  int n,m,limit;
4  int l, r[1<<20];
5
6  const int EX = 20;
7  const long double pi = 3.1415926535897932384626433832795028841971;
8  // Complex number
9  pair<long double, long double> operator+(const pair<long double, long double> &a, const pair<long
double, long double> &b) {
10     return {a.x + b.x, a.y + b.y};
11 }
12 pair<long double, long double> operator-(const pair<long double, long double> &a, const pair<long
double, long double> &b) {
13     return {a.x - b.x, a.y - b.y};
14 }
15 pair<long double, long double> operator*(const pair<long double, long double> &a, const pair<long
double, long double> &b) {
16     return {a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x};
17 }
18 void FFT(pair<long double, long double> *a, int op) {
19     for(int i=0; i<limit; i++) {
20         if(i<r[i]) swap(a[i], a[r[i]]);
21     }
22     for(int mid=1; mid<limit; mid<=<1) {
23         pair<long double, long double> W(cos(Pi/mid), op*sin(Pi/mid));
24         for(int r=mid<<1, j=0; j<limit; j+=r) {
25             pair<long double, long double> w(1, 0);
26             for(int l=0; l<mid; l++, w=w*W) {
27                 pair<long double, long double> x=a[j+l], y=w*a[j+mid+l];
28                 a[j+l]=x+y; a[j+mid+l]=x-y;
29             }
30         }
31     }
32 }
33
34 vector<int> operator*(const vector<int> &a, const vector<int> &b) {
35     int n = a.size(), m = b.size();
36     vector<int> ans(n + m - 1);
37     n--, m--;
38     limit = 1;
39     l = 0;
40     while (limit <= n + m)
41         limit <=< 1, l++;
42     for (int i = 1; i <= limit; i++)
43         r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
44     pair<long double, long double> *c = new pair<long double, long double>[limit]();

```

```

45     for (int i = 0; i <= n; i++)
46         c[i].x = a[i];
47     for (int i = 0; i <= m; i++)
48         c[i].y = b[i];
49     FFT(c, 1);
50     for (int i = 0; i < limit; i++)
51         c[i] = c[i] * c[i];
52     FFT(c, -1);
53     for (int i = 0; i <= n + m; i++)
54         ans[i] = c[i].y / (limit << 1) + 0.5;
55     delete[] c;
56     return ans;
57 }

```

## jsjh

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct Point { double x, y; };          // 点
5  using Vec = Point;                     // 向量
6  struct Line { Point P; Vec v; };        // 直线 (点向式)
7  struct Seg { Point A, B; };             // 线段 (存两个端点)
8  struct Circle { Point O; double r; };   // 圆 (存圆心和半径)
9
10 const Point O = {0, 0};                 // 原点
11 const Line Ox = {0, {1, 0}}, Oy = {0, {0, 1}}; // 坐标轴
12 const double PI = acos(-1), EPS = 1e-9;
13
14 bool eq(double a, double b) { return abs(a - b) < EPS; } // ==
15 bool gt(double a, double b) { return a - b > EPS; }      // >
16 bool lt(double a, double b) { return a - b < -EPS; }     // <
17 bool ge(double a, double b) { return a - b > -EPS; }     // >=
18 bool le(double a, double b) { return a - b < EPS; }      // <=
19
20 Vec r90a(Vec v) { return {-v.y, v.x}; }                  // 逆时针旋转90度的向量
21 Vec r90c(Vec v) { return {v.y, -v.x}; }                  // 顺时针旋转90度的向量
22 Vec operator+(Vec u, Vec v) { return {u.x + v.x, u.y + v.y}; } // 向量加向量
23 Vec operator-(Vec u, Vec v) { return {u.x - v.x, u.y - v.y}; } // 向量减向量
24 Vec operator*(double k, Vec v) { return {k * v.x, k * v.y}; } // 数乘
25 double operator*(Vec u, Vec v) { return u.x * v.x + u.y * v.y; } // 点乘
26 double operator^(Vec u, Vec v) { return u.x * v.y - u.y * v.x; } // 叉乘
27 double len(Vec v) { return sqrt(v.x * v.x + v.y * v.y); } // 向量长度
28 double slope(Vec v) { return v.y / v.x; }                 // 斜率 // NOTE 不要用isinf判断斜率
    不存在, 用后面的paral_y
29
30 // 两向量的夹角余弦
31 // DEPENDS len, V*V
32 double cos_t(Vec u, Vec v) { return u * v / len(u) / len(v); }
33
34 // 归一化向量 (与原向量方向相同的单位向量)
35 // DEPENDS len
36 Vec norm(Vec v) { return {v.x / len(v), v.y / len(v)}; }
37

```

```

38 // 与原向量平行且横坐标大于等于0的单位向量
39 // DEPENDS d*V, len
40 Vec pnorm(Vec v) { return (v.x < 0 ? -1 : 1) / len(v) * v; }
41
42 // 线段的方向向量
43 // DEPENDS V-V
44 // NOTE 直线的方向向量直接访问属性v
45 Vec dvec(Seg l) { return l.B - l.A; }
46
47 // 两点式直线
48 // DEPENDS V-V
49 Line line(Point A, Point B) { return {A, B - A}; }
50
51 // 斜截式直线
52 Line line(double k, double b) { return {{0, b}, {1, k}}; }
53
54 // 点斜式直线
55 Line line(Point P, double k) { return {P, {1, k}}; }
56
57 // 线段所在直线
58 // DEPENDS V-V
59 Line line(Seg l) { return {l.A, l.B - l.A}; }
60
61 // 给定直线的横坐标求纵坐标
62 // NOTE 请确保直线不与y轴平行
63 double at_x(Line l, double x) { return l.P.y + (x - l.P.x) * l.v.y / l.v.x; }
64
65 // 给定直线的纵坐标求横坐标
66 // NOTE 请确保直线不与x轴平行
67 double at_y(Line l, double y) { return l.P.x - (y - l.P.y) * l.v.x / l.v.y; }
68
69 // 点到直线的垂足
70 // DEPENDS V-V, V*V, d*V
71 Point pedal(Point P, Line l) { return l.P - (l.P - P) * l.v / (l.v * l.v) * l.v; }
72
73 // 过某点作直线的垂线
74 // DEPENDS r90c
75 Line perp(Line l, Point P) { return {P, r90c(l.v)}; }
76
77 // 角平分线
78 // DEPENDS V+V, len, norm
79 Line bisec(Point P, Vec u, Vec v) { return {P, norm(u) + norm(v)}; }
80
81 // 线段的方向向量
82 // DEPENDS V-V
83 // NOTE 直线的方向向量直接访问属性v
84 Vec dvec(Seg l) { return l.B - l.A; }
85
86 // 线段中点
87 Point midp(Seg l) { return {(l.A.x + l.B.x) / 2, (l.A.y + l.B.y) / 2}; }
88
89 // 线段中垂线
90 // DEPENDS r90c, V-V, midp
91 Line perp(Seg l) { return {midp(l), r90c(l.B - l.A)}; }
92

```

```

93 // 向量是否互相垂直
94 // DEPENDS eq, V*V
95 bool verti(Vec u, Vec v) { return eq(u * v, 0); }
96
97 // 向量是否互相平行
98 // DEPENDS eq, V^V
99 bool paral(Vec u, Vec v) { return eq(u ^ v, 0); }
100
101 // 向量是否与x轴平行
102 // DEPENDS eq
103 bool paral_x(Vec v) { return eq(v.y, 0); }
104
105 // 向量是否与y轴平行
106 // DEPENDS eq
107 bool paral_y(Vec v) { return eq(v.x, 0); }
108
109 // 点是否在直线上
110 // DEPENDS eq
111 bool on(Point P, Line l) { return eq((P.x - l.P.x) * l.v.y, (P.y - l.P.y) * l.v.x); }
112
113 // 点是否在线段上
114 // DEPENDS eq, len, V-V
115 bool on(Point P, Seg l) { return eq(len(P - l.A) + len(P - l.B), len(l.A - l.B)); }
116
117 // 两个点是否重合
118 // DEPENDS eq
119 bool operator==(Point A, Point B) { return eq(A.x, B.x) && eq(A.y, B.y); }
120
121 // 两条直线是否重合
122 // DEPENDS eq, on(L)
123 bool operator==(Line a, Line b) { return on(a.P, b) && on(a.P + a.v, b); }
124
125 // 两条线段是否重合
126 // DEPENDS eq, P==P
127 bool operator==(Seg a, Seg b) { return (a.A == b.A && a.B == b.B) || (a.A == b.B && a.B == b.A); }
128
129 // 以横坐标为第一关键词、纵坐标为第二关键词比较两个点
130 // DEPENDS eq, lt
131 bool operator<(Point A, Point B) { return lt(A.x, B.x) || (eq(A.x, B.x) && lt(A.y, B.y)); }
132
133 // 直线与圆是否相切
134 // DEPENDS eq, V^V, len
135 bool tangency(Line l, Circle C) { return eq(abs((C.O ^ l.v) - (l.P ^ l.v)), C.r * len(l.v)); }
136
137 // 圆与圆是否相切
138 // DEPENDS eq, V-V, len
139 bool tangency(Circle C1, Circle C2) { return eq(len(C1.O - C2.O), C1.r + C2.r); }
140
141 // 两点间的距离
142 // DEPENDS len, V-V
143 double dis(Point A, Point B) { return len(A - B); }
144
145 // 点到直线的距离
146 // DEPENDS V^V, len
147 double dis(Point P, Line l) { return abs((P ^ l.v) - (l.P ^ l.v)) / len(l.v); }

```

```

148
149 // 平行直线间的距离
150 // DEPENDS d*V, V^V, len, pnorm
151 // NOTE 请确保两直线是平行的
152 double dis(Line a, Line b) { return abs((a.P ^ pnorm(a.v)) - (b.P ^ pnorm(b.v))); }
153
154 // 平移
155 // DEPENDS V+V
156 Line operator+(Line l, Vec v) { return {l.P + v, l.v}; }
157 Seg operator+(Seg l, Vec v) { return {l.A + v, l.B + v}; }
158
159 // 旋转
160 // DEPENDS V+V, V-V
161 Point rotate(Point P, double rad) { return {cos(rad) * P.x - sin(rad) * P.y, sin(rad) * P.x +
    cos(rad) * P.y}; }
162 Point rotate(Point P, double rad, Point C) { return C + rotate(P - C, rad); } //
    DEPENDS ^1
163 Line rotate(Line l, double rad, Point C = 0) { return {rotate(l.P, rad, C), rotate(l.v, rad)}; } //
    DEPENDS ^1, ^2
164 Seg rotate(Seg l, double rad, Point C = 0) { return {rotate(l.A, rad, C), rotate(l.B, rad, C)}; } //
    DEPENDS ^1, ^2
165
166 // 对称
167 // 关于点对称
168 Point reflect(Point A, Point P) { return {P.x * 2 - A.x, P.y * 2 - A.y}; }
169 Line reflect(Line l, Point P) { return {reflect(l.P, P), l.v}; } // DEPENDS ^1
170 Seg reflect(Seg l, Point P) { return {reflect(l.A, P), reflect(l.B, P)}; } // DEPENDS ^1
171 // 关于直线对称
172 // DEPENDS V-V, V*V, d*V, pedal
173 // NOTE 向量和点在这里的表现不同, 求向量关于某直线的对称向量需要用reflect_v
174 Point reflect(Point A, Line ax) { return reflect(A, pedal(A, ax)); } // DEPENDS ^1
175 Vec reflect_v(Vec v, Line ax) { return reflect(v, ax) - reflect(0, ax); } // DEPENDS ^1, ^4
176 Line reflect(Line l, Line ax) { return {reflect(l.P, ax), reflect_v(l.v, ax)}; } // DEPENDS ^1, ^4,
    ^5
177 Seg reflect(Seg l, Line ax) { return {reflect(l.A, ax), reflect(l.B, ax)}; } // DEPENDS ^1, ^4
178
179 // 直线与直线交点
180 // DEPENDS eq, d*V, V*V, V+V, V^V
181 vector<Point> inter(Line a, Line b)
182 {
183     double c = a.v ^ b.v;
184     if (eq(c, 0)) return {};
185     Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
186     return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
187 }
188
189 // 直线与圆交点
190 // DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
191 vector<Point> inter(Line l, Circle C)
192 {
193     Point P = pedal(C.O, l);
194     double h = len(P - C.O);
195     if (gt(h, C.r)) return {};
196     if (eq(h, C.r)) return {P};
197     double d = sqrt(C.r * C.r - h * h);

```

```

198     Vec vec = d / len(l.v) * l.v;
199     return {P + vec, P - vec};
200 }
201
202 // 圆与圆的交点
203 // DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
204 vector<Point> inter(Circle C1, Circle C2)
205 {
206     Vec v1 = C2.O - C1.O, v2 = r90c(v1);
207     double d = len(v1);
208     if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d)) return {};
209     if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r))) return {C1.O + C1.r / d * v1};
210     double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
211     double h = sqrt(C1.r * C1.r - a * a);
212     Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
213     return {C1.O + av + hv, C1.O + av - hv};
214 }
215
216 // 三角形的重心
217 Point barycenter(Point A, Point B, Point C)
218 {
219     return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
220 }
221
222 // 三角形的外心
223 // DEPENDS r90c, V*V, d*V, V-V, V+V
224 // NOTE 给定圆上三点求圆, 要先判断是否三点共线
225 Point circumcenter(Point A, Point B, Point C)
226 {
227     double a = A * A, b = B * B, c = C * C;
228     double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
229     return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
230 }
231
232 // 三角形的内心
233 // DEPENDS len, d*V, V-V, V+V
234 Point incenter(Point A, Point B, Point C)
235 {
236     double a = len(B - C), b = len(A - C), c = len(A - B);
237     double d = a + b + c;
238     return 1 / d * (a * A + b * B + c * C);
239 }
240
241 // 三角形的垂心
242 // DEPENDS V*V, d*V, V-V, V^V, r90c
243 Point orthocenter(Point A, Point B, Point C)
244 {
245     double n = B * (A - C), m = A * (B - C);
246     double d = (B - C) ^ (A - C);
247     return 1 / d * r90c(n * (C - B) - m * (C - A));
248 }

```

```

1  const int N=2e5+10;
2  const int mod=998244353;
3  template<const int T>
4  struct ModInt {
5      const static int mod = T;
6      int x;
7      ModInt(int x = 0) : x(x % mod) {}
8      ModInt(long long x) : x(int(x % mod)) {}
9      int val() { return x; }
10     ModInt operator + (const ModInt &a) const { int x0 = x + a.x; return ModInt(x0 < mod ? x0 : x0 -
mod); }
11     ModInt operator - (const ModInt &a) const { int x0 = x - a.x; return ModInt(x0 < 0 ? x0 + mod :
x0); }
12     ModInt operator * (const ModInt &a) const { return ModInt(1LL * x * a.x % mod); }
13     ModInt operator / (const ModInt &a) const { return *this * a.inv(); }
14     void operator += (const ModInt &a) { x += a.x; if (x >= mod) x -= mod; }
15     void operator -= (const ModInt &a) { x -= a.x; if (x < 0) x += mod; }
16     void operator *= (const ModInt &a) { x = 1LL * x * a.x % mod; }
17     void operator /= (const ModInt &a) { *this = *this / a; }
18     friend ostream &operator<<(ostream &os, const ModInt &a) { return os << a.x; }
19
20     ModInt pow(int64_t n) const {
21         ModInt res(1), mul(x);
22         while(n) {
23             if (n & 1) res *= mul;
24             mul *= mul;
25             n >>= 1;
26         }
27         return res;
28     }
29
30     ModInt inv() const {
31         int a = x, b = mod, u = 1, v = 0;
32         while (b) {
33             int t = a / b;
34             a -= t * b; swap(a, b);
35             u -= t * v; swap(u, v);
36         }
37         if (u < 0) u += mod;
38         return u;
39     }
40
41 };
42 typedef ModInt<998244353> mint;

```

## ntt

```

1  // vector2 ± ä ~ ó
2  using i64 = long long;
3  constexpr int mod = 998244353;
4  int norm(int x) {
5      if (x < 0) {
6          x += mod;
7      }

```

```

8     if (x >= mod) {
9         x -= mod;
10    }
11    return x;
12 }
13 template<class T>
14 T power(T a, int b) {
15     T res = 1;
16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     int val() const {
27         return x;
28     }
29     Z operator-() const {
30         return Z(norm(mod - x));
31     }
32     Z inv() const {
33         assert(x != 0);
34         return power(*this, mod - 2);
35     }
36     Z &operator*=(const Z &rhs) {
37         x = i64(x) * rhs.x % mod;
38         return *this;
39     }
40     Z &operator+=(const Z &rhs) {
41         x = norm(x + rhs.x);
42         return *this;
43     }
44     Z &operator-=(const Z &rhs) {
45         x = norm(x - rhs.x);
46         return *this;
47     }
48     Z &operator/=(const Z &rhs) {
49         return *this *= rhs.inv();
50     }
51     friend Z operator*(const Z &lhs, const Z &rhs) {
52         Z res = lhs;
53         res *= rhs;
54         return res;
55     }
56     friend Z operator+(const Z &lhs, const Z &rhs) {
57         Z res = lhs;
58         res += rhs;
59         return res;
60     }
61     friend Z operator-(const Z &lhs, const Z &rhs) {
62         Z res = lhs;

```



```

63     res -= rhs;
64     return res;
65 }
66 friend Z operator/(const Z &lhs, const Z &rhs) {
67     Z res = lhs;
68     res /= rhs;
69     return res;
70 }
71 friend istream &operator>>(istream &is, Z &a) {
72     i64 v;
73     is >> v;
74     a = Z(v);
75     return is;
76 }
77 friend ostream &operator<<(ostream &os, const Z &a) {
78     return os << a.val();
79 }
80 };
81 vector<int> rev;
82 vector<Z> roots{0, 1};
83 void dft(vector<Z> &a) {
84     int n = a.size();
85     if (int(rev.size()) != n) {
86         int k = __builtin_ctz(n) - 1;
87         rev.resize(n);
88         for (int i = 0; i < n; i++) {
89             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
90         }
91     }
92     for (int i = 0; i < n; i++) {
93         if (rev[i] < i) {
94             swap(a[i], a[rev[i]]);
95         }
96     }
97     if (int(roots.size()) < n) {
98         int k = __builtin_ctz(roots.size());
99         roots.resize(n);
100        while ((1 << k) < n) {
101            Z e = power(Z(3), (mod - 1) >> (k + 1));
102            for (int i = 1 << (k - 1); i < (1 << k); i++) {
103                roots[i << 1] = roots[i];
104                roots[i << 1 | 1] = roots[i] * e;
105            }
106            k++;
107        }
108    }
109    for (int k = 1; k < n; k *= 2) {
110        for (int i = 0; i < n; i += 2 * k) {
111            for (int j = 0; j < k; j++) {
112                Z u = a[i + j], v = a[i + j + k] * roots[k + j];
113                a[i + j] = u + v, a[i + j + k] = u - v;
114            }
115        }
116    }
117 }

```

```

118 void idft(vector<Z> &a) {
119     int n = a.size();
120     reverse(a.begin() + 1, a.end());
121     dft(a);
122     Z inv = (1 - mod) / n;
123     for (int i = 0; i < n; i++) {
124         a[i] *= inv;
125     }
126 }
127 struct Poly {
128     vector<Z> a;
129     Poly() {}
130     Poly(const vector<Z> &a) : a(a) {}
131     Poly(const initializer_list<Z> &a) : a(a) {}
132     int size() const {
133         return a.size();
134     }
135     void resize(int n) {
136         a.resize(n);
137     }
138     Z operator[](int idx) const {
139         if (idx < size()) {
140             return a[idx];
141         } else {
142             return 0;
143         }
144     }
145     Z &operator[](int idx) {
146         return a[idx];
147     }
148     Poly mulxk(int k) const {
149         auto b = a;
150         b.insert(b.begin(), k, 0);
151         return Poly(b);
152     }
153     Poly modxk(int k) const {
154         k = min(k, size());
155         return Poly(vector<Z>(a.begin(), a.begin() + k));
156     }
157     Poly divxk(int k) const {
158         if (size() <= k) {
159             return Poly();
160         }
161         return Poly(vector<Z>(a.begin() + k, a.end()));
162     }
163     friend Poly operator+(const Poly &a, const Poly &b) {
164         vector<Z> res(max(a.size(), b.size()));
165         for (int i = 0; i < int(res.size()); i++) {
166             res[i] = a[i] + b[i];
167         }
168         return Poly(res);
169     }
170     friend Poly operator-(const Poly &a, const Poly &b) {
171         vector<Z> res(max(a.size(), b.size()));
172         for (int i = 0; i < int(res.size()); i++) {

```

```

173         res[i] = a[i] - b[i];
174     }
175     return Poly(res);
176 }
177 friend Poly operator*(Poly a, Poly b) {
178     if (a.size() == 0 || b.size() == 0) {
179         return Poly();
180     }
181     int sz = 1, tot = a.size() + b.size() - 1;
182     while (sz < tot) {
183         sz *= 2;
184     }
185     a.a.resize(sz);
186     b.a.resize(sz);
187     dft(a.a);
188     dft(b.a);
189     for (int i = 0; i < sz; i++) {
190         a.a[i] = a[i] * b[i];
191     }
192     idft(a.a);
193     a.resize(tot);
194     return a;
195 }
196 friend Poly operator*(Z a, Poly b) {
197     for (int i = 0; i < int(b.size()); i++) {
198         b[i] *= a;
199     }
200     return b;
201 }
202 friend Poly operator*(Poly a, Z b) {
203     for (int i = 0; i < int(a.size()); i++) {
204         a[i] *= b;
205     }
206     return a;
207 }
208 Poly &operator+=(Poly b) {
209     return (*this) = (*this) + b;
210 }
211 Poly &operator-=(Poly b) {
212     return (*this) = (*this) - b;
213 }
214 Poly &operator*=(Poly b) {
215     return (*this) = (*this) * b;
216 }
217 Poly deriv() const {
218     if (a.empty()) {
219         return Poly();
220     }
221     vector<Z> res(size() - 1);
222     for (int i = 0; i < size() - 1; i++) {
223         res[i] = (i + 1) * a[i + 1];
224     }
225     return Poly(res);
226 }
227 Poly integr() const {

```

```

228     vector<Z> res(size() + 1);
229     for (int i = 0; i < size(); i++) {
230         res[i + 1] = a[i] / (i + 1);
231     }
232     return Poly(res);
233 }
234 Poly inv(int m) const {
235     Poly x{a[0].inv()};
236     int k = 1;
237     while (k < m) {
238         k *= 2;
239         x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
240     }
241     return x.modxk(m);
242 }
243 Poly log(int m) const {
244     return (deriv() * inv(m)).integr().modxk(m);
245 }
246 Poly exp(int m) const {
247     Poly x{1};
248     int k = 1;
249     while (k < m) {
250         k *= 2;
251         x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
252     }
253     return x.modxk(m);
254 }
255 Poly pow(int k, int m) const {
256     int i = 0;
257     while (i < size() && a[i].val() == 0) {
258         i++;
259     }
260     if (i == size() || 1LL * i * k >= m) {
261         return Poly(vector<Z>(m));
262     }
263     Z v = a[i];
264     auto f = divxk(i) * v.inv();
265     return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
266 }
267 Poly sqrt(int m) const {
268     Poly x{1};
269     int k = 1;
270     while (k < m) {
271         k *= 2;
272         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((mod + 1) / 2);
273     }
274     return x.modxk(m);
275 }
276 Poly mulT(Poly b) const {
277     if (b.size() == 0) {
278         return Poly();
279     }
280     int n = b.size();
281     reverse(b.a.begin(), b.a.end());
282     return ((*this) * b).divxk(n - 1);

```

```

283     }
284 };
285
286 // vector
287 #define int long long
288 const int md = 998244353;
289
290 inline void add(int &x, int y) {
291     x += y;
292     if (x >= md) {
293         x -= md;
294     }
295 }
296
297 inline void sub(int &x, int y) {
298     x -= y;
299     if (x < 0) {
300         x += md;
301     }
302 }
303
304 inline int mul(int x, int y) {
305     return (long long) x * y % md;
306 }
307
308 inline int power(int x, int y) {
309     int res = 1;
310     for (; y; y >>= 1, x = mul(x, x)) {
311         if (y & 1) {
312             res = mul(res, x);
313         }
314     }
315     return res;
316 }
317
318 inline int inv(int a) {
319     a %= md;
320     if (a < 0) {
321         a += md;
322     }
323     int b = md, u = 0, v = 1;
324     while (a) {
325         int t = b / a;
326         b -= t * a;
327         swap(a, b);
328         u -= t * v;
329         swap(u, v);
330     }
331     if (u < 0) {
332         u += md;
333     }
334     return u;
335 }
336
337 namespace ntt {

```

```

338 int base = 1, root = -1, max_base = -1;
339 vector<int> rev = {0, 1}, roots = {0, 1};
340
341 void init() {
342     int temp = md - 1;
343     max_base = 0;
344     while (temp % 2 == 0) {
345         temp >>= 1;
346         ++max_base;
347     }
348     root = 2;
349     while (true) {
350         if (power(root, 1 << max_base) == 1 && power(root, 1 << (max_base - 1)) != 1) {
351             break;
352         }
353         ++root;
354     }
355 }
356
357 void ensure_base(int nbase) {
358     if (max_base == -1) {
359         init();
360     }
361     if (nbase <= base) {
362         return;
363     }
364     assert(nbase <= max_base);
365     rev.resize(1 << nbase);
366     for (int i = 0; i < 1 << nbase; ++i) {
367         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (nbase - 1));
368     }
369     roots.resize(1 << nbase);
370     while (base < nbase) {
371         int z = power(root, 1 << (max_base - 1 - base));
372         for (int i = 1 << (base - 1); i < 1 << base; ++i) {
373             roots[i << 1] = roots[i];
374             roots[i << 1 | 1] = mul(roots[i], z);
375         }
376         ++base;
377     }
378 }
379
380 void dft(vector<int> &a) {
381     int n = a.size(), zeros = __builtin_ctz(n);
382     ensure_base(zeros);
383     int shift = base - zeros;
384     for (int i = 0; i < n; ++i) {
385         if (i < rev[i] >> shift) {
386             swap(a[i], a[rev[i] >> shift]);
387         }
388     }
389     for (int i = 1; i < n; i <= 1) {
390         for (int j = 0; j < n; j += i << 1) {
391             for (int k = 0; k < i; ++k) {
392                 int x = a[j + k], y = mul(a[j + k + i], roots[i + k]);

```

```

393         a[j + k] = (x + y) % md;
394         a[j + k + i] = (x + md - y) % md;
395     }
396 }
397 }
398 }
399
400 vector<int> multiply(vector<int> a, vector<int> b) {
401     int need = a.size() + b.size() - 1, nbase = 0;
402     while (1 << nbase < need) {
403         ++nbase;
404     }
405     ensure_base(nbase);
406     int sz = 1 << nbase;
407     a.resize(sz);
408     b.resize(sz);
409     bool equal = a == b;
410     dft(a);
411     if (equal) {
412         b = a;
413     } else {
414         dft(b);
415     }
416     int inv_sz = inv(sz);
417     for (int i = 0; i < sz; ++i) {
418         a[i] = mul(mul(a[i], b[i]), inv_sz);
419     }
420     reverse(a.begin() + 1, a.end());
421     dft(a);
422     a.resize(need);
423     return a;
424 }
425
426 vector<int> inverse(vector<int> a) {
427     int n = a.size(), m = (n + 1) >> 1;
428     if (n == 1) {
429         return vector<int>(1, inv(a[0]));
430     }
431     else
432     {
433         vector<int> b = inverse(vector<int>(a.begin(), a.begin() + m));
434         int need = n << 1, nbase = 0;
435         while (1 << nbase < need) {
436             ++nbase;
437         }
438         ensure_base(nbase);
439         int sz = 1 << nbase;
440         a.resize(sz);
441         b.resize(sz);
442         dft(a);
443         dft(b);
444         int inv_sz = inv(sz);
445         for (int i = 0; i < sz; ++i) {
446             a[i] = mul(mul(md + 2 - mul(a[i], b[i]), b[i]), inv_sz);
447         }

```

```

448         reverse(a.begin() + 1, a.end());
449         dft(a);
450         a.resize(n);
451         return a;
452     }
453 }
454 }
455
456 using ntt::multiply;
457 using ntt::inverse;
458
459 vector<int>& operator += (vector<int> &a, const vector<int> &b) {
460     if (a.size() < b.size()) {
461         a.resize(b.size());
462     }
463     for (int i = 0; i < b.size(); ++i) {
464         add(a[i], b[i]);
465     }
466     return a;
467 }
468
469 vector<int> operator + (const vector<int> &a, const vector<int> &b) {
470     vector<int> c = a;
471     return c += b;
472 }
473
474 vector<int>& operator -= (vector<int> &a, const vector<int> &b) {
475     if (a.size() < b.size()) {
476         a.resize(b.size());
477     }
478     for (int i = 0; i < b.size(); ++i) {
479         sub(a[i], b[i]);
480     }
481     return a;
482 }
483
484 vector<int> operator - (const vector<int> &a, const vector<int> &b) {
485     vector<int> c = a;
486     return c -= b;
487 }
488
489 vector<int>& operator *= (vector<int> &a, const vector<int> &b) {
490     if (min(a.size(), b.size()) < 128) {
491         vector<int> c = a;
492         a.assign(a.size() + b.size() - 1, 0);
493         for (int i = 0; i < c.size(); ++i) {
494             for (int j = 0; j < b.size(); ++j) {
495                 add(a[i + j], mul(c[i], b[j]));
496             }
497         }
498     } else {
499         a = multiply(a, b);
500     }
501     return a;
502 }

```



```

503
504 vector<int> operator * (const vector<int> &a, const vector<int> &b) {
505     vector<int> c = a;
506     return c *= b;
507 }
508
509 vector<int>& operator /= (vector<int> &a, const vector<int> &b) {
510     int n = a.size(), m = b.size();
511     if (n < m) {
512         a.clear();
513     } else {
514         vector<int> c = b;
515         reverse(a.begin(), a.end());
516         reverse(c.begin(), c.end());
517         c.resize(n - m + 1);
518         a *= inverse(c);
519         a.erase(a.begin() + n - m + 1, a.end());
520         reverse(a.begin(), a.end());
521     }
522     return a;
523 }
524
525 vector<int> operator / (const vector<int> &a, const vector<int> &b) {
526     vector<int> c = a;
527     return c /= b;
528 }
529
530 vector<int>& operator %= (vector<int> &a, const vector<int> &b) {
531     int n = a.size(), m = b.size();
532     if (n >= m) {
533         vector<int> c = (a / b) * b;
534         a.resize(m - 1);
535         for (int i = 0; i < m - 1; ++i) {
536             sub(a[i], c[i]);
537         }
538     }
539     return a;
540 }
541
542 vector<int> operator % (const vector<int> &a, const vector<int> &b) {
543     vector<int> c = a;
544     return c %= b;
545 }
546
547 vector<int> derivative(const vector<int> &a) {
548     int n = a.size();
549     vector<int> b(n - 1);
550     for (int i = 1; i < n; ++i) {
551         b[i - 1] = mul(a[i], i);
552     }
553     return b;
554 }
555
556 vector<int> primitive(const vector<int> &a) {
557     int n = a.size();

```

```

558     vector<int> b(n + 1), invs(n + 1);
559     for (int i = 1; i <= n; ++i) {
560         invs[i] = i == 1 ? 1 : mul(md - md / i, invs[md % i]);
561         b[i] = mul(a[i - 1], invs[i]);
562     }
563     return b;
564 }
565
566 vector<int> logarithm(const vector<int> &a) {
567     vector<int> b = primitive(derivative(a) * inverse(a));
568     b.resize(a.size());
569     return b;
570 }
571
572 vector<int> exponent(const vector<int> &a) {
573     vector<int> b(1, 1);
574     while (b.size() < a.size()) {
575         vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
576         add(c[0], 1);
577         vector<int> old_b = b;
578         b.resize(b.size() << 1);
579         c -= logarithm(b);
580         c *= old_b;
581         for (int i = b.size() >> 1; i < b.size(); ++i) {
582             b[i] = c[i];
583         }
584     }
585     b.resize(a.size());
586     return b;
587 }
588
589 vector<int> power(vector<int> a, int m) {
590     int n = a.size(), p = -1;
591     vector<int> b(n);
592     for (int i = 0; i < n; ++i) {
593         if (a[i]) {
594             p = i;
595             break;
596         }
597     }
598     if (p == -1) {
599         b[0] = !m;
600         return b;
601     }
602     if ((long long) m * p >= n) {
603         return b;
604     }
605     int mu = power(a[p], m), di = inv(a[p]);
606     vector<int> c(n - m * p);
607     for (int i = 0; i < n - m * p; ++i) {
608         c[i] = mul(a[i + p], di);
609     }
610     c = logarithm(c);
611     for (int i = 0; i < n - m * p; ++i) {
612         c[i] = mul(c[i], m);

```

```

613     }
614     c = exponent(c);
615     for (int i = 0; i < n - m * p; ++i) {
616         b[i + m * p] = mul(c[i], mu);
617     }
618     return b;
619 }
620
621 vector<int> sqrt(const vector<int> &a) {
622     vector<int> b(1, 1);
623     while (b.size() < a.size()) {
624         vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
625         vector<int> old_b = b;
626         b.resize(b.size() << 1);
627         c *= inverse(b);
628         for (int i = b.size() >> 1; i < b.size(); ++i) {
629             b[i] = mul(c[i], (md + 1) >> 1);
630         }
631     }
632     b.resize(a.size());
633     return b;
634 }
635
636 vector<int> multiply_all(int l, int r, vector<vector<int>> &all) {
637     if (l > r) {
638         return vector<int>();
639     } else if (l == r) {
640         return all[l];
641     } else {
642         int y = (l + r) >> 1;
643         return multiply_all(l, y, all) * multiply_all(y + 1, r, all);
644     }
645 }
646
647 vector<int> evaluate(const vector<int> &f, const vector<int> &x) {
648     int n = x.size();
649     if (!n) {
650         return vector<int>();
651     }
652     vector<vector<int>> up(n * 2);
653     for (int i = 0; i < n; ++i) {
654         up[i + n] = vector<int>{(md - x[i]) % md, 1};
655     }
656     for (int i = n - 1; i; --i) {
657         up[i] = up[i << 1] * up[i << 1 | 1];
658     }
659     vector<vector<int>> down(n * 2);
660     down[1] = f % up[1];
661     for (int i = 2; i < n * 2; ++i) {
662         down[i] = down[i >> 1] % up[i];
663     }
664     vector<int> y(n);
665     for (int i = 0; i < n; ++i) {
666         y[i] = down[i + n][0];
667     }

```

```

668     return y;
669 }
670
671 vector<int> interpolate(const vector<int> &x, const vector<int> &y) {
672     int n = x.size();
673     vector<vector<int>> up(n * 2);
674     for (int i = 0; i < n; ++i) {
675         up[i + n] = vector<int>{(md - x[i]) % md, 1};
676     }
677     for (int i = n - 1; i; --i) {
678         up[i] = up[i << 1] * up[i << 1 | 1];
679     }
680     vector<int> a = evaluate(derivative(up[1]), x);
681     for (int i = 0; i < n; ++i) {
682         a[i] = mul(y[i], inv(a[i]));
683     }
684     vector<vector<int>> down(n * 2);
685     for (int i = 0; i < n; ++i) {
686         down[i + n] = vector<int>(1, a[i]);
687     }
688     for (int i = n - 1; i; --i) {
689         down[i] = down[i << 1] * up[i << 1 | 1] + down[i << 1 | 1] * up[i << 1];
690     }
691     return down[1];
692 }

```