1 数学

1.1 快速求逆元

使用条件：$x \in [0, mod)$ 并且 x 与 mod 互质

```
LL inv(LL a, LL p) {
    LL d, x, y;
    exgcd(a, p, d, x, y);
    return d == 1 ? (x + p) % p : -1;
}
```

1.2 扩展欧几里德算法

返回结果：$ax + by = gcd(a, b)$

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(!b) return x = 1, y = 0, a;
    else {
        LL d = exgcd(b, a % b, x, y);
        LL t = x;  x = y;
        y = t - a / b * y;
        return d;
    }
}
```

1.3 中国剩余定理

返回结果：

$$x \equiv r_i (mod \;\; p_i) \;\; (0 \le i < n)$$

```
LL china(int n, int *a, int *m) {
    LL M = 1, d, x = 0, y;
    for(int i = 0; i < n; i++)
        M *= m[i];
    for(int i = 0; i < n; i++) {
        LL w = M / m[i];
        d = exgcd(m[i], w, d, y);
        y = (y % M + M) % M;
        x = (x + y * w % M * a[i]) % M;
    }
    while(x < 0)x += M;
    return x;
}
```

1.4 中国剩余定理 2

```
//merge Ax=B and ax=b to A'x=B'
void merge(LL &A,LL &B,LL a,LL b){
    LL x,y;
    sol(A,-a,b-B,x,y);
    A=lcm(A,a);
    B=(a*y+b)%A;
    B=(B+A)%A;
}
```

1.5 卢卡斯定理

```
LL Lucas(LL n, LL m, LL p) {
    LL ans = 1;
    while(n && m) {
        LL a = n % p, b = m % p;
        if(a < b) return 0;
        ans = (ans * C(a, b, p)) % p;
        n /= p; m /= p;
    }
    return ans % p;
}
```

1.6 小步大步

返回结果：$a^x = b \;\; (mod \;\; p)$　　使用条件：p 为质数

```
LL BSGS(LL a, LL b, LL p) {
    LL m = sqrt(p) + .5, v = inv(pw(a, m, p), p), e = 1;
    map<LL, LL> hash; hash[1] = 0;
```

```
    for(int i = 1; i < m; i++) e = e * a % p, hash[e] = i;
    for(int i = 0; i <= m; i++)
        if(hash.count(b)) return i * m + hash[b];
        else b = b * v % p;
    return -1;
}
```

1.7 Miller Rabin 素数测试

```
const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool check(long long n, int base) {
    long long n2 = n - 1, res; int s = 0;
    while(n2 % 2 == 0) n2 >>= 1, s++;
    res = pw(base, n2, n);
    if((res == 1) || (res == n - 1)) return 1;
    while(s--) {
        res = mul(res, res, n);
        if(res == n - 1) return 1;
    }
    return 0; // n is not a strong pseudo prime
}
bool isprime(const long long &n) {
    if(n == 2) return true;
    if(n < 2 || n % 2 == 0) return false;
    for(int i = 0; i < 12 && BASE[i] < n; i++)
        if(!check(n, BASE[i])) return false;
    return true;
}
```

1.8 Pollard Rho 大数分解

```
LL prho(LL n, LL c) {
    LL i = 1, k = 2, x = rand() % (n - 1) + 1, y = x;
    while(1) {
        i++; x = (x * x % n + c) % n;
        LL d = __gcd((y - x + n) % n, n);
        if(d > 1 && d < n)return d;
        if(y == x)return n;
        if(i == k)y = x, k <<= 1;
    }
}
void factor(LL n, vector<LL>&fat) {
    if(n == 1)return;
    if(isprime(n)) {fat.push_back(n); return;}
    LL p = n;
    while(p >= n)p = prho(p, rand() % (n - 1) + 1);
    factor(p, fat);factor(n / p, fat);
}
```

1.9 快速数论变换 (zky)

返回结果: $c_i = \sum_{0\le j\le i} a_j \cdot b_{i-j}(mod)\ \ (0 \le i < n)$

```
/*{(mod,G)}={(81788929,7),(101711873,3),(167772161,3)
      ,(377487361,7),(998244353,3),(1224736769,3)
      ,(1300234241,3),(1484783617,5)}*/
int mo = 998244353, G = 3;
void NTT(int a[], int n, int f) {
    for(register int i = 0; i < n; i++)
        if(i < rev[i])
            swap(a[i], a[rev[i]]);
    for (register int i = 2; i <= n; i <<= 1) {
        static int exp[maxn];
        exp[0] = 1;
        exp[1] = pw(G, (mo - 1) / i);
        if(f == -1)exp[1] = pw(exp[1], mo - 2);
        for(register int k = 2; k < (i >> 1); k++)
            exp[k] = 1LL * exp[k - 1] * exp[1] % mo;
        for(register int j = 0; j < n; j += i) {
            for(register int k = 0; k < (i >> 1); k++) {
                register int &pA = a[j + k], &pB = a[j + k + (i >> 1)];
                register int A = pA, B = 1LL * pB * exp[k] % mo;
                pA = (A + B) % mo;
                pB = (A - B + mo) % mo;
            }
```

```
        }
    }
    if(f == -1) {
        int rv = pw(n, mo - 2) % mo;
        for(int i = 0; i < n; i++)
            a[i] = 1LL * a[i] * rv % mo;
    }
}
void mul(int m, int a[], int b[], int c[]) {
    int n = 1, len = 0;
    while(n < m)n <<= 1, len++;
    for (int i = 1; i < n; i++)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (len - 1));
    NTT(a, n, 1);
    NTT(b, n, 1);
    for(int i = 0; i < n; i++)
        c[i] = 1LL * a[i] * b[i] % mo;
    NTT(c, n, -1);
}
```

1.10 原根

```
vector<LL>fct;
bool check(LL x, LL g) {
    for(int i = 0; i < fct.size(); i++)
        if(pw(g, (x - 1) / fct[i], x) == 1)
            return 0;
    return 1;
}
LL findrt(LL x) {
    LL tmp = x - 1;
    for(int i = 2; i * i <= tmp; i++) {
        if(tmp % i == 0) {
            fct.push_back(i);
            while(tmp % i == 0)tmp /= i;
        }
    }
    if(tmp > 1) fct.push_back(tmp);
    // x is 1,2,4,p^n,2p^n
    // x has phi(phi(x)) primitive roots
    for(int i = 2; i < int(1e9); i++)
    if(check(x, i)) return i;
    return -1;
}
```

1.11 线性递推

//已知 $a_0, a_1, ..., a_{m-1}\\$
$a_n = c_0 * a_{n-m} + ... + c_{m-1} * a_{n-1}\\$
求 $a_n = v_0 * a_0 + v_1 * a_1 + ... + v_{m-1} * a_{m-1}\\$

```
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !!n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <<= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0);
        int b(!!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;
        } else {
            for(int i(0); i < m; i++) {
                for(int j(0), t(i + b); j < m; j++, t++) {
                    u[t] = (u[t] + v[i] * v[j]) % p;
                }
            }
            for(int i((m << 1) - 1); i >= m; i--) {
                for(int j(0), t(i - m); j < m; j++, t++) {
                    u[t] = (u[t] + c[j] * u[i]) % p;
                }
            }
        }
        copy(u, u + m, v);
    }
```

```
    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
    for(int i(m); i < 2 * m; i++) {
        a[i] = 0;
        for(int j(0); j < m; j++) {
            a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        b[j] = 0;
        for(int i(0); i < m; i++) {
            b[j] = (b[j] + v[i] * a[i + j]) % p;
        }
    }
    for(int j(0); j < m; j++) {
        a[j] = b[j];
    }
}
```

1.12 直线下整点个数

返回结果：$\sum_{0\le i<n}\lfloor\frac{a+b\cdot i}{m}\rfloor$ 使用条件：$n, m > 0$，$a, b \ge 0$ 时间复杂度：$\mathcal{O}(nlogn)$

```
LL solve(LL n, LL a, LL b, LL m) {
    if(b == 0)
        return n * (a / m);
    if(a >= m || b >= m)
        return n * (a / m) + (n - 1) * n / 2 * (b / m) + solve(n, a % m, b % m, m);
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

1.13 1e9+7 FFT

```
// double 精度对 10^9 + 7 取模最多可以做到 2^20
const int MOD = 1000003;
const double PI = acos(-1);
typedef complex<double> Complex;
const int N = 65536, L = 15, MASK = (1 << L) - 1;
Complex w[N];
void FFTInit() {
  for (int i = 0; i < N; ++i)
    w[i] = Complex(cos(2 * i * PI / N), sin(2 * i * PI / N));
}
void FFT(Complex p[], int n) {
  for (int i = 1, j = 0; i < n - 1; ++i) {
    for (int s = n; j ^= s >>= 1, ~j & s;);
    if (i < j) swap(p[i], p[j]);
  }
  for (int d = 0; (1 << d) < n; ++d) {
    int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
    for (int i = 0; i < n; i += m2) {
      for (int j = 0; j < m; ++j) {
        Complex &p1 = p[i + j + m], &p2 = p[i + j];
        Complex t = w[rm * j] * p1;
        p1 = p2 - t, p2 = p2 + t;
      } } }
}
Complex A[N], B[N], C[N], D[N];
void mul(int a[N], int b[N]) {
  for (int i = 0; i < N; ++i) {
    A[i] = Complex(a[i] >> L, a[i] & MASK);
    B[i] = Complex(b[i] >> L, b[i] & MASK);
  }
  FFT(A, N), FFT(B, N);
  for (int i = 0; i < N; ++i) {
    int j = (N - i) % N;
    Complex da = (A[i] - conj(A[j])) * Complex(0, -0.5),
        db = (A[i] + conj(A[j])) * Complex(0.5, 0),
        dc = (B[i] - conj(B[j])) * Complex(0, -0.5),
        dd = (B[i] + conj(B[j])) * Complex(0.5, 0);
    C[j] = da * dd + da * dc * Complex(0, 1);
    D[j] = db * dd + db * dc * Complex(0, 1);
  }
  FFT(C, N), FFT(D, N);
  for (int i = 0; i < N; ++i) {
    long long da = (long long)(C[i].imag() / N + 0.5) % MOD,
```

```
            db = (long long)(C[i].real() / N + 0.5) % MOD,
            dc = (long long)(D[i].imag() / N + 0.5) % MOD,
            dd = (long long)(D[i].real() / N + 0.5) % MOD;
    a[i] = ((dd << (L * 2)) + ((db + dc) << L) + da) % MOD;
  }
}
```

1.14 自适应辛普森

```
double area(const double &left, const double &right) {
    double mid = (left + right) / 2;
    return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
}

double simpson(const double &left, const double &right,
               const double &eps, const double &area_sum) {
    double mid = (left + right) / 2;
    double area_left = area(left, mid);
    double area_right = area(mid, right);
    double area_total = area_left + area_right;
    if (std::abs(area_total - area_sum) < 15 * eps) {
        return area_total + (area_total - area_sum) / 15;
    }
    return simpson(left, mid, eps / 2, area_left)
         + simpson(mid, right, eps / 2, area_right);
}

double simpson(const double &left, const double &right, const double &eps) {
    return simpson(left, right, eps, area(left, right));
}
```

1.15 多项式求根

```
const double eps=1e-12;
double a[10][10];
typedef vector<double> vd;
int sgn(double x) { return x < -eps ? -1 : x > eps; }
double mypow(double x,int num){
  double ans=1.0;
  for(int i=1;i<=num;++i)ans*=x;
  return ans;
}
double f(int n,double x){
  double ans=0;
  for(int i=n;i>=0;--i)ans+=a[n][i]*mypow(x,i);
  return ans;
}
double getRoot(int n,double l,double r){
  if(sgn(f(n,l))==0)return l;
  if(sgn(f(n,r))==0)return r;
  double temp;
  if(sgn(f(n,l))>0)temp=-1;else temp=1;
  double m;
  for(int i=1;i<=10000;++i){
    m=(l+r)/2;
    double mid=f(n,m);
    if(sgn(mid)==0){
      return m;
    }
    if(mid*temp<0)l=m;else r=m;
  }
  return (l+r)/2;
}
vd did(int n){
  vd ret;
  if(n==1){
    ret.push_back(-1e10);
    ret.push_back(-a[n][0]/a[n][1]);
    ret.push_back(1e10);
    return ret;
  }
  vd mid=did(n-1);
  ret.push_back(-1e10);
  for(int i=0;i+1<mid.size();++i){
    int t1=sgn(f(n,mid[i])),t2=sgn(f(n,mid[i+1]));
```

```
    if(t1*t2>0)continue;
    ret.push_back(getRoot(n,mid[i],mid[i+1]));
  }
  ret.push_back(1e10);
  return ret;
}
int main(){
  int n; scanf("%d",&n);
  for(int i=n;i>=0;--i){
    scanf("%lf",&a[n][i]);
  }
  for(int i=n-1;i>=0;--i)
    for(int j=0;j<=i;++j)a[i][j]=a[i+1][j+1]*(j+1);
  vd ans=did(n);
  sort(ans.begin(),ans.end());
  for(int i=1;i+1<ans.size();++i)printf("%.10f\n",ans[i]);
  return 0;
}
```

1.16 魔幻多项式

多项式求逆

原理： 令 $G(x) = x * A - 1$（其中 A 是一个多项式系数），根据牛顿迭代法有：

$$\begin{aligned} F_{t+1}(x) &\equiv F_t(x) - \frac{F_t(x) * A(x) - 1}{A(x)} \\ &\equiv 2F_t(x) - F_t(x)^2 * A(x) \pmod{x^{2t}} \end{aligned}$$

注意事项：

1. $F(x)$ 的常数项系数必然不为 0，否则没有逆元；
2. 复杂度是 $O(n \log n)$ 但是常数比较大（10^5 大概需要 0.3 秒左右）；
3. 传入的两个数组必须不同，但传入的次数界没有必要是 2 的次幂；

```
void getInv(int *a, int *b, int n) {
  static int tmp[MAXN];
  b[0] = fpm(a[0], MOD - 2, MOD);
  for (int c = 2, M = 1; c < (n << 1); c <<= 1) {
    for (; M <= 3 * (c - 1); M <<= 1);
    meminit(b, c, M);
    meminit(tmp, c, M);
    memcopy(tmp, a, 0, c);
    DFT(tmp, M, 0);
    DFT(b, M, 0);
    for (int i = 0; i < M; i++) {
      b[i] = 1ll * b[i] * (2ll - 1ll * tmp[i] * b[i] % MOD + MOD) % MOD;
    }
    DFT(b, M, 1);
    meminit(b, c, M);
    }
}
```