```
void MathSet(){
    //遍历子集
    int u;//原集合
    // 遍历 u 的非空子集
    for (int s = u; s; s = (s - 1) & u) {
      // s 是 u 的一个非空子集
    }

}

//快速幂
long long binpow(long long a, long long b) {
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a;
    a = a * a;
    b >>= 1;
  }
  return res;
}

//高精度

void add(int a[], int b[], int c[]) {
  clear(c);
  // 高精度实现中，一般令数组的最大长度 LEN 比可能的输入大一些
  // 然后略去末尾的几次循环，这样一来可以省去不少边界情况的处理
  // 因为实际输入不会超过 1000 位，故在此循环到 LEN - 1 = 1003 已经足够
  for (int i = 0; i < LEN - 1; ++i) {
    // 将相应位上的数码相加
    c[i] += a[i] + b[i];
    if (c[i] >= 10) {
      // 进位
      c[i + 1] += 1;
      c[i] -= 10;
    }
  }
}


void sub(int a[], int b[], int c[]) {
  clear(c);

  for (int i = 0; i < LEN - 1; ++i) {
    // 逐位相减
    c[i] += a[i] - b[i];
    if (c[i] < 0) {
      // 借位
      c[i + 1] -= 1;
      c[i] += 10;
    }
  }
}

void mul(int a[], int b[], int c[]) {
  clear(c);

  for (int i = 0; i < LEN - 1; ++i) {
    // 这里直接计算结果中的从低到高第 i 位，且一并处理了进位
    // 第 i 次循环为 c[i] 加上了所有满足 p + q = i 的 a[p] 与 b[q] 的乘积之和
    // 这样做的效果和直接进行上图的运算最后求和是一样的，只是更加简短的一种实现方式
    for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];

    if (c[i] >= 10) {
      c[i + 1] += c[i] / 10;
      c[i] %= 10;
    }
  }
}

// 被除数 a 以下标 last_dg 为最低位，是否可以再减去除数 b 而保持非负
// len 是除数 b 的长度，避免反复计算
inline bool greater_eq(int a[], int b[], int last_dg, int len) {
  // 有可能被除数剩余的部分比除数长，这个情况下最多多出 1 位，故如此判断即可
  if (a[last_dg + len] != 0) return true;
  // 从高位到低位，逐位比较
  for (int i = len - 1; i >= 0; --i) {
    if (a[last_dg + i] > b[i]) return true;
    if (a[last_dg + i] < b[i]) return false;
  }
  // 相等的情形下也是可行的
  return true;
}

void div(int a[], int b[], int c[], int d[]) {
  clear(c);
  clear(d);

  int la, lb;
  for (la = LEN - 1; la > 0; --la)
    if (a[la - 1] != 0) break;
  for (lb = LEN - 1; lb > 0; --lb)
    if (b[lb - 1] != 0) break;
  if (lb == 0) {
    puts("> <");
    return;
  }  // 除数不能为零

  // c 是商
  // d 是被除数的剩余部分，算法结束后自然成为余数
  for (int i = 0; i < la; ++i) d[i] = a[i];
  for (int i = la - lb; i >= 0; --i) {
    // 计算商的第 i 位
    while (greater_eq(d, b, i, lb)) {
      // 若可以减，则减
      // 这一段是一个高精度减法
      for (int j = 0; j < lb; ++j) {
        d[i + j] -= b[j];
        if (d[i + j] < 0) {
          d[i + j + 1] -= 1;
          d[i + j] += 10;
        }
      }
      // 使商的这一位增加 1
      c[i] += 1;
      // 返回循环开头，重新检查
    }
  }
}
```

```
//筛素数
int n;
vector<char> is_prime(n + 1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
  if (is_prime[i]) {
    for (int j = i * i; j <= n; j += i) is_prime[j] = false;
  }
}

//求欧拉函数
void pre() {
  memset(is_prime, 1, sizeof(is_prime));
  int cnt = 0;
  is_prime[1] = 0;
  phi[1] = 1;
  for (int i = 2; i <= 5000000; i++) {
    if (is_prime[i]) {
      prime[++cnt] = i;
      phi[i] = i - 1;
    }
    for (int j = 1; j <= cnt && i * prime[j] <= 5000000; j++) {
      is_prime[i * prime[j]] = 0;
      if (i % prime[j])
        phi[i * prime[j]] = phi[i] * phi[prime[j]];
      else {
        phi[i * prime[j]] = phi[i] * prime[j];
        break;
      }
    }
  }
}

//求约数个数
void pre() {
  d[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!v[i]) v[i] = 1, p[++tot] = i, d[i] = 2, num[i] = 1;
    for (int j = 1; j <= tot && i <= n / p[j]; ++j) {
      v[p[j] * i] = 1;
      if (i % p[j] == 0) {
        num[i * p[j]] = num[i] + 1;
        d[i * p[j]] = d[i] / num[i * p[j]] * (num[i * p[j]] + 1);
        break;
      } else {
        num[i * p[j]] = 1;
        d[i * p[j]] = d[i] * 2;
      }
    }
  }
}

//求拓展GCD
int Exgcd(int a, int b, int &x, int &y) {
  if (!b) {
    x = 1;
    y = 0;
    return a;
  }
  int d = Exgcd(b, a % b, x, y);
  int t = x;
  x = y;
  y = t - (a / b) * y;
  return d;
}

//return F(i),F(i+1)
pair<int, int> fib(int n) {
  if (n == 0) return {0, 1};
  auto p = fib(n >> 1);
  int c = p.first * (2 * p.second - p.first);
  int d = p.first * p.first + p.second * p.second;
  if (n & 1)
    return {d, c + d};
  else
    return {c, d};
}

//FFT
#include <cstdio>
#include <cmath>
```

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <unordered_map>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;

const int N = 5000007;

const double PI = acos(-1);

int n, m;
int res, ans[N];
int limit = 1;//
int L;//二进制的位数
int R[N];

inline int read()
{
    register int x = 0, f = 1;
    register char ch = getchar();
    while(ch < '0' || ch > '9') {if(ch == '-')f = -1;ch = getchar();}
    while(ch >= '0' && ch <= '9') {x = x * 10 + ch - '0';ch = getchar();}
    return x * f;
}

struct Complex
{
    double x, y;
    Complex (double x = 0, double y = 0) : x(x), y(y) { }
}a[N], b[N];

Complex operator * (Complex J, Complex Q) {
    //模长相乘，幅度相加
    return Complex(J.x * Q.x - J.y * Q.y, J.x * Q.y + J.y * Q.x);
}
Complex operator - (Complex J, Complex Q) {
    return Complex(J.x - Q.x, J.y - Q.y);
}
Complex operator + (Complex J, Complex Q) {
    return Complex(J.x + Q.x, J.y + Q.y);
}

void FFT(Complex * A, int type)//FFT板子
{
    for(int i = 0; i < limit; ++ i)
        if(i < R[i])
            swap(A[i], A[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        Complex wn(cos(PI / mid), type * sin(PI / mid));
        for(int len = mid << 1, pos = 0; pos < limit; pos += len) {
            Complex w(1, 0);

            for(int k = 0; k < mid; ++ k, w = w * wn) {
                Complex x = A[pos + k];
                Complex y = w * A[pos + mid + k];
                A[pos + k] = x + y;
                A[pos + mid + k] = x - y;
            }
        }
    }
    if(type == 1) return ;
    for(int i = 0; i <= limit; ++ i)
        a[i].x /= limit, a[i].y /= limit;
}

int main()
{
    n = read(), m = read();
    for(int i = 0; i <= n; ++ i)
        a[i].x = read();
    for(int i = 0; i <= m; ++ i)
        a[i].y = read();//把b(x)放到a(x)的虚部上
    while(limit <= n + m)
        limit <<= 1, L ++ ;
    for(int i = 0; i < limit; ++ i)
        R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));
    FFT(a, 1);
    for(int i = 0; i <= limit; ++ i)
        a[i] = a[i] * a[i];//求出a(x)^2
    FFT(a, -1);
    for(int i = 0; i <= n + m; ++ i)
        printf("%d ", (int)(a[i].y / 2 + 0.5));
        //虚部取出来除2，注意要+0.5，否则精度会有问题,这里的x和y都是double
}



//两数乘法FFT
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#define ll long long
using namespace std;

const ll N=5000010;
const double pi=acos(-1);
string s;
ll n,m,limit,c[N];
```

```
struct complex{
    double real,imag;
    complex(double X=0,double Y=0){real=X; imag=Y;}
}a[N],b[N];
inline complex operator +(complex a,complex b){return complex(a.real+b.real,a.imag+b.imag);}
inline complex operator -(complex a,complex b){return complex(a.real-b.real,a.imag-b.imag);}
inline complex operator *(complex a,complex b){return complex(a.real*b.real-a.imag*b.imag,a.real

void FFT(complex *a,ll op){
    for(ll i=0; i<limit; i++){
        if(i<c[i]) swap(a[i],a[c[i]]);
    }
    for(ll mid=1; mid<limit; mid<<=1){
        complex W(cos(pi/mid),op*sin(pi/mid));
        for(ll r=mid<<1,j=0; j<limit; j+=r){
            complex w(1,0);
            for(ll l=0; l<mid; l++,w=w*W){
                complex x=a[j+l],y=w*a[j+mid+l];
                a[j+l]=x+y; a[j+mid+l]=x-y;
            }
        }
    }
}

int main(){
    ios::sync_with_stdio(0);
    cin>>s; n=s.size()-1;
    for(ll i=0; i<=n; i++) a[n-i].real=s[i]-48;
    cin>>s; m=s.size()-1;
    for(ll i=0; i<=m; i++) b[m-i].real=s[i]-48;
    limit=1; ll l=0;
    while(limit<=n+m){
        limit<<=1;
        l++;
    }
    for(ll i=0; i<limit; i++) c[i]=(c[i>>1]>>1)|((i&1)<<(l-1));
    FFT(a,1); FFT(b,1);
    for(ll i=0; i<=limit; i++) a[i]=a[i]*b[i];
    FFT(a,-1);
    //上面都是FFT板子，不解释
    memset(c,0,sizeof(c));//c数组要清零，因为前面有赋值
    for(ll i=0; i<=n+m+1; i++) c[i]=a[i].real/limit+0.5;//模仿FFT输出时赋值
    for(ll i=0; i<=n+m; i++){//进位
        c[i+1]+=c[i]/10;
                c[i]%=10;
        }
        limit=n+m+1;
        while(c[limit]){//还有前面没有处理的
                c[limit+1]+=c[limit]/10;
                c[limit]%=10;
                limit++;
        }
    for(ll i=limit-1; i>=0; i--) cout<<c[i];//输出
    return 0;
}

//快速沃尔什变换
#include <cstdio>
#include <cstring>
#define ll long long
#define mod 998244353
#define maxn 1<<18

int n;
ll a[maxn],b[maxn],A[maxn],B[maxn];
void FWT_or(ll *f,int type)
{
        for(int mid=1;mid<n;mid<<=1)
        for(int block=mid<<1,j=0;j<n;j+=block)
        for(int i=j;i<j+mid;i++)
        f[i+mid]=(f[i+mid]+f[i]*type+mod)%mod;
}
void FWT_and(ll *f,int type)
{
        for(int mid=1;mid<n;mid<<=1)
        for(int block=mid<<1,j=0;j<n;j+=block)
        for(int i=j;i<j+mid;i++)
        f[i]=(f[i]+f[i+mid]*type+mod)%mod;
}
int inv_2=499122177;
void FWT_xor(ll *f,int type)
{
        for(int mid=1;mid<n;mid<<=1)
        for(int block=mid<<1,j=0;j<n;j+=block)
        for(int i=j;i<j+mid;i++)
        {
                ll x=f[i],y=f[i+mid];
                f[i]=(x+y)%mod*(type==1?1:inv_2)%mod;
                f[i+mid]=(x-y+mod)%mod*(type==1?1:inv_2)%mod;
        }
}
void work(void (*FWT)(ll *f,int type))//将函数作为参数传入
{
        for(int i=0;i<n;i++)a[i]=A[i],b[i]=B[i];
        FWT(a,1);FWT(b,1);
        for(int i=0;i<n;i++)a[i]=a[i]*b[i]%mod;
        FWT(a,-1);
        for(int i=0;i<n;i++)printf("%lld ",a[i]);
        printf("\n");
}

int main()
```

```
{
        scanf("%d",&n);n=1<<n;
        for(int i=0;i<n;i++)scanf("%lld",&A[i]),A[i]%=mod;
        for(int i=0;i<n;i++)scanf("%lld",&B[i]),B[i]%=mod;
        work(FWT_or);work(FWT_and);work(FWT_xor);
}
```