Fast Fourier Transformation

Step-1 FFT

$$A(x) = a_0 x^0 + a_1 x^1 + \ldots + a_{n-1} x^{n-1}.$$

Divide it into two polynomials, one with even coefficients(0) and the other with the odd coefficients(1):

$$A_0(x) = a_0 x^0 + a_2 x^1 + \ldots + a_{n-2} x^{n/2-1},$$

$$A_1(x) = a_1 x^0 + a_3 x^1 + \ldots + a_{n-1} x^{n/2-1}.$$

Finally we can write original polynomial as combination of Ao and A1:

$$A(x) = A_0(x^2) + x A_1(x^2). \qquad (1)$$

Now writing th original polynomial A(x) in point form for n-th root of unity:

$$y_k = y_k^0 + w_n^k y_k^1, \qquad k = 0 \ldots n/2-1.$$

$$y_{k+n/2} = A(w_n^{k+n/2}) = A_0(w_n^{2k+n}) + w_n^{k+n/2} A_1(w_n^{2k+n}) = A_0(w_n^{2k} w_n^n) + w_n^k w_n^{n/2} A_1(w_n^{2k} w_n^n) =$$

$$= A_0(w_n^{2k}) - w_n^k A_1(w_n^{2k}) = y_k^0 - w_n^k y_k^1.$$

Finally our **point form** will be:

$$y_k = y_k^0 + w_n^k y_k^1, \qquad k = 0 \ldots n/2-1,$$

$$y_{k+n/2} = y_k^0 - w_n^k y_k^1, \qquad k = 0 \ldots n/2-1.$$

FFT()

```
vector<complex<double> > fft(vector<complex<double> > a){
    int n = (int)a.size();
    if(n <= 1)
        return a;
    //Dividing a0 and a1 as even and odd polynomial of degree n/2
    vector<complex<double> > a0(n/2), a1(n/2);
    for(int i = 0;i<n/2;i++){
        a0[i] = a[2*i];
        a1[i] = a[2*i + 1];
    }
    //Divide step
    //Recursively calling FFT on polynolmial of degree n/2
    a0 = fft(a0);
    a1 = fft(a1);
    double ang = 2*PI/n;
    //defining w1 and wn
    complex<double> w(1) , wn(cos(ang),sin(ang));
    for(int i = 0;i<n/2;i++){
        //for powers of k<= n/2
        a[i] = a0[i] + w*a1[i];
        //powers of k > n/2
        a[i + n/2] = a0[i] - w*a1[i];
        //Updating value of wk
        w *= wn;
    }
    return a;
}
```

Time complexity: O(nlogn)

Step 2: Convolution

Multiply()

```
void multiply(vector<int> a, vector<int> b){
    vector<complex<double> > fa(a.begin(),a.end()), fb(b.
begin(),b.end());
    int n = 1;
    //resizing value of n as power of 2
    while(n < max(a.size(),b.size()))
        n <<= 1;
    n <<= 1;
    //cout<<n<<endl;
    fa.resize(n);
    fb.resize(n);
    //Calling FFT on polynomial A and B
    //fa and fb denotes the point form
    fa = fft(fa);
    fb = fft(fb);

    //Convolution step
    for(int i = 0;i<n;i++){
        fa[i] = fa[i] * fb[i];
    }

    //Converitng fa from coefficient back to point form
    fa = inv_fft(fa);
    return;
}
```

Time Complexity: O(n) (exluding the time for calculating FFT())

Step 3: Inverse FFT

So, suppose we are given a vector (y0,y1,y2, …yn-1).Now we need to restore it back to coefficient form.

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

Then the vector {a0,a1,a2,…an-1}can be found by multiplying the vector {y0, y1,y2,…yn-1}by an inverse matrix:

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

Thus we get the formula:

$$a_k = \frac{1}{n} \sum_{j=0}^{n-1} y_j w_n^{-kj}.$$

Comparing it with the formula for y_k:

$$y_k = \sum_{j=0}^{n-1} a_j w_n^{kj},$$

Inverse_FFT()

```
vector<complex<double> > inv_fft(vector<complex<double> > y){
    int n = y.size();
    if(n <= 1)
        return y;
    vector<complex<double> > y0(n/2), y1(n/2);
    for(int i = 0;i<n/2;i++){
        y0[i] = y[2*i];
        y1[i] = y[2*i + 1];
    }
    y0 = inv_fft(y0);
    y1 = inv_fft(y1);
    double ang = 2 * PI/n * -1;
    complex<double> w(1), wn(cos(ang), sin(ang));
    for(int i = 0;i<n/2;i++){
        y[i] = y0[i] + w*y1[i];
        y[i + n/2] = y0[i] - w*y1[i];
```

```
            y[i] /= 2;
            y[i + n/2] /= 2;
            w *= wn;
        }
    //each element of the result is divided by 2
    //assuming that the division by 2 will take place at
each level of recursion
    //then eventually just turns out that all the element
s are divided into n.

    return y;
}
```

Time Complexity: O(nlogn)

Ques: Very Fast Multiplicaiton

http://www.spoj.com/problems/VFMUL/

```
#include<iostream>
#include<vector>
#include<complex>
#include<algorithm>

using namespace std;

#define PI 3.14159265358979323846

```

```
vector<complex<double> > fft(vector<complex<double> > a){
    //for(int i = 0;i<a.size();i++)cout<<a[i]<<" ";cout<<
endl;
    int n = (int)a.size();
    if(n <= 1)
        return a;
    vector<complex<double> > a0(n/2), a1(n/2);
    for(int i = 0;i<n/2;i++){
        a0[i] = a[2*i];
        a1[i] = a[2*i + 1];
        //cout<<n<<" "<<a0[i]<<" "<<a1[i]<<endl;
    }
    a0 = fft(a0);
    a1 = fft(a1);
    double ang = 2*PI/n;
    complex<double> w(1) , wn(cos(ang),sin(ang));
    for(int i = 0;i<n/2;i++){
        a[i] = a0[i] + w*a1[i];
        a[i + n/2] = a0[i] - w*a1[i];
        w *= wn;
        //cout<<a[i]<<"   "<<a[i+n/2]<<endl;
    }
    return a;
}

vector<complex<double> > inv_fft(vector<complex<double> >
 y){
    int n = y.size();
    if(n <= 1)
        return y;
    vector<complex<double> > y0(n/2), y1(n/2);
    for(int i = 0;i<n/2;i++){
        y0[i] = y[2*i];
        y1[i] = y[2*i + 1];
    }
    y0 = inv_fft(y0);
    y1 = inv_fft(y1);
    double ang = 2 * PI/n * -1;
    complex<double> w(1), wn(cos(ang), sin(ang));
    for(int i = 0;i<n/2;i++){
        y[i] = y0[i] + w*y1[i];
        y[i + n/2] = y0[i] - w*y1[i];
        y[i] /= 2;
        y[i + n/2] /= 2;
        w *= wn;
    }
    return y;
}

void multiply(vector<int> a, vector<int> b){
    vector<complex<double> > fa(a.begin(),a.end()), fb(b.
begin(),b.end());
    int n = 1;
    while(n < max(a.size(),b.size()))
        n <<= 1;
    n <<= 1;
```

```cpp
    //cout<<n<<endl;
    fa.resize(n);
    fb.resize(n);
    fa = fft(fa);
    fb = fft(fb);
    for(int i = 0;i<n;i++){
        fa[i] = fa[i] * fb[i];
        //cout<<fa[i]<<endl;
    }
    fa = inv_fft(fa);
    vector<int> res(n);
    for(int i = 0;i<n;i++){
        res[i] = int(fa[i].real() + 0.5);
        //cout<<res[i]<<endl;
    }
    int carry = 0;
    for(int i = 0;i<n;i++){
        res[i] = res[i] + carry;
        carry = res[i] / 10;
        res[i] %= 10;
    }
    bool flag = 0;
    for(int i = n-1;i>=0;i--){if(res[i] || flag){printf("%d",res[i]);flag = 1;}}
    if(!flag)printf("0");
    printf("\n");
    return;
}
```

```cpp

int main(){
    int n;
    scanf("%d",&n);
    while(n--){
        string x,y;
        vector<int> a,b;
        cin>>x>>y;
        for(int i = 0;i<x.length();i++){
            a.push_back(int(x[i] - '0'));
        }
        for(int i = 0;i<y.length();i++){
            b.push_back(int(y[i] - '0'));
        }
        reverse(a.begin(),a.end());
        reverse(b.begin(),b.end());
        multiply(a,b);
    }
    return 0;
}
```
