序列中两两之差

CF1398G

把题目化简之后，就是给一个序列 a_i 要能高效地得到 $a_i - a_j$ 构成的集合。

构造两个生成函数 $\sum x^{a_i}$ 和 $\sum x^{-a_i}$那么这两个多项式相乘得到的答案 $f(x)$ 中如果 x^i 的系数不为 0 则 i 可以被表示为序列中某两个数的差。

```
/*
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
*/
#include<bits/stdc++.h>
#define ALL(x) (x).begin(),(x).end()
#define ll long long
#define db double
#define ull unsigned long long
#define pii_ pair<int,int>
#define mp_ make_pair
#define pb push_back
#define fi first
#define se second
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define per(i,a,b) for(int i=(a);i>=(b);i--)
#define show1(a) cout<<#a<<" = "<<a<<endl
#define show2(a,b) cout<<#a<<" = "<<a<<"; "<<#b<<" = "<<b<<endl
using namespace std;
const ll INF = 1LL<<60;
const int inf = 1<<30;
const int maxn = 1e6+5;
const int M = 998244353;
inline void fastio() {ios::sync_with_stdio(false);cin.tie(0);cout.tie(0);}
ll qpow(ll a,ll b) {ll s=1;while(b){if(b&1)s=s*a%M;a=a*a%M;b>>=1;}return
s;}

int rev[maxn],f[maxn];
ll A[maxn],B[maxn];
void trans(ll a[],int lim,int type)
{
    rep(i,1,lim-1) if(i<rev[i]) swap(a[i],a[rev[i]]);
    for(int mid=1;mid<lim;mid<<=1){
        ll wn = qpow(3,(M-1)/mid/2);
        if(type==-1) wn = qpow(wn,M-2);
        for(int R=mid<<1,j=0;j<lim;j+=R){
            ll w = 1;
            for(int k=0;k<mid;k++,w=w*wn%M){
                ll x=a[j+k],y=w*a[j+mid+k]%M;
                a[j+k] = (x+y)%M;
                a[j+mid+k] = (x-y+M)%M;
```

```
            }
        }
    }
    if(type==-1){
        ll inv = qpow(lim,M-2);
        rep(i,0,lim) a[i] = a[i]*inv%M;
    }
}

int main()
{
    fastio();
    memset(f,-1,sizeof(f));
    int n,x,y;
    cin>>n>>x>>y;
    rep(i,0,n){
        int a;cin>>a;
        A[a] = 1;
        B[x-a] = 1;
    }
    int lim=1,l=0;
    while(lim<=(int)4e5) lim<<=1,l++;
    rep(i,1,lim-1) rev[i] = (rev[i>>1]>>1) | ((i&1)<<(l-1));
    trans(A,lim,1);trans(B,lim,1);
    rep(i,0,lim) A[i] = A[i] * B[i] %M;
    trans(A,lim,-1);
    rep(i,x+1,lim){
        if(2*(y + i-x) > (int)1e6) break;
        if(A[i] > 0) f[2*(y + i-x)] = 2*(y + i-x);
    }
    rep(i,4,1000000){ if(f[i]==-1) continue;
        for(int j = i+i;j<=1000000;j+=i){
            f[j] = max(f[j],f[i]);
        }
    }
    int q; cin>>q;
    while(q--){
        int l; cin>>l;
        cout<<f[l]<<" ";
    }
    return 0;
}
```

字符串匹配

先考虑如何用 FFT 做和 KMP 一样的事：

记 s 串为长度为 m 的模式串，t 串为长度为 n 的文本串，下标均从 0 开始。目标是找出所有 x 满足 $\forall i\in [0,m),t_{x-m+i+1} = s_i$。

上述条件可用一个式子来表示 $\sum _{i=0}^{m-1} (s_i - t_{x-m+i+1})^2 = 0$，展开后为 $\sum _{i=0}^{m-1} s_i^2 + t_{x-m+i+1}^2 - 2\cdot s_i\cdot t_{x-m+i+1}$。

两个平方项都可以通过前缀和得到。乘积项转换一下就会变成一个卷积的形式：把 s 串翻转一下得到 rs 串，于是乘积项就是 $\sum _{i=0}^{m-1} 2\cdot rs_{m-i-1} \cdot t_{x-m+i+1} = \sum _{i=0}^{m-1} rs_i \cdot t_{x-i}$，所以这里就可以用$\text{FFT}$处理的到。

最后的判断条件：卷积之后的多项式为 $f(x)$，在 t 串的 $x0$ 位置匹配当且仅当 $f(x) + pres[m-1] + pret[x] - pret[x-m-2] = 0$，总复杂度为 $O(nlogn)$。

虽然 FFt 多了一个 log 的复杂度，但有些匹配是 KMP 无法做但是 FFT 可以做。

|P4173

在正常匹配的基础上扩大了字符集的范围，多了一种 $*$ 字符