ACM-ICPC

D-Tesla 模板

2018

dengtesla

2018/7/20

目录

1 数论

1.1 线性筛打质数表，质因数分解

```
#define MAXIMUM 26
int prime[1000000];
bool isprime[1000000];

void get_prime(int listsize)
{
    int primesize=1;
    memset(isprime,1,sizeof(isprime));
    isprime[1] = false;
    for(int i=2;i<=listsize;i++)
    {
        if(isprime[i]) prime[primesize++]=i;
        for(int j=1;i*prime[j]<=listsize&&j<=primesize;j++)
        {
            isprime[i*prime[j]] = false;
            if(i%prime[j]==0) break;
        }
    }
}

struct p
{
    int value;
    int time;
}p[MAXIMUM];

void prime_factorization(long long n)
{
    memset(p,0,sizeof(p));
    long long psize=0;
    long long a = n;
    for(int t = 1;1LL*prime[t]*prime[t]<=n;t++)
    {
        if(a%prime[t]==0) p[++psize].value = prime[t];
        while(a%prime[t]==0)
```

```
        {
            p[psize].time += 1;
            a = a / prime[t];
        }
        if(a<=90000)
        if(isprime[a])
        {
            p[++psize].value = a;
            p[psize].time += 1;
            a = 1;
            break;
        }
        if(a==1) break;
    }
    if(a!=1)
    {
        p[++psize].value = a;
        p[psize].time = 1;
    }
}
```

1.2 莫比乌斯函数，欧拉函数

```
const int MAX = 101000;


int mu[MAX+10];
bool isprime[MAX+10];
int prime[MAX+10];

void get_mobius(int n)
{
    mu[1] = 1;
    int tot = 0;
    memset(isprime,1,sizeof(isprime));
    for(int i=2;i<=n;i++)
    {
        if(isprime[i])
        {
            prime[++tot] = i;
            mu[i] = -1;
        }
```

```
        for(int j=1;prime[j]*i<=n;j++)
        {
            isprime[prime[j]*i] = 0;
            if(i%prime[j]==0)
            {
                mu[prime[j]*i] = 0;
                break;
            }
            mu[prime[j]*i]=-mu[i];
        }
    }
}

int euler[MAX+10];

void get_euler(int n)
{
    euler[1] = 1;
    int tot = 0;
    memset(isprime,1,sizeof(isprime));
    for(int i=2;i<=n;i++)
    {
        if(isprime[i])
        {
            prime[++tot] = i;
            euler[i] = i-1;
        }
        for(int j=1;prime[j]*i<=n;j++)
        {
            isprime[prime[j]*i] = 0;
            if(i%prime[j]==0)
            {
                euler[prime[j]*i] = prime[j]*euler[i];
                break;
            }
            euler[prime[j]*i] = euler[i]*(prime[j]-1);
        }
    }
}
```

1.3 Miller-Rabin 素性测试

```
typedef long long LL;
LL iprime[6] = {2, 3, 5, 233, 331};
LL qmul(LL x, LL y, LL mod) { // 乘法防止溢出， 如果p * p 不爆 LL 的话可以直接乘； O(1)乘法或者转化成二进制加法
    return (x * y - (long long)(x / (long double)mod * y + 1e-3) *mod + mod) % mod;
}
LL qpow(LL a, LL n, LL mod) {
    LL ret = 1;
    while(n) {
        if(n & 1) ret = qmul(ret, a, mod);
        a = qmul(a, a, mod);
        n >>= 1;
    }
    return ret;
}
bool Miller_Rabin(LL p) {
    if(p < 2) return 0;
    if(p != 2 && p % 2 == 0) return 0;
    LL s = p - 1;
    while(! (s & 1)) s >>= 1;
    for(int i = 0; i < 5; ++i) {
        if(p == iprime[i]) return 1;
        LL t = s, m = qpow(iprime[i], s, p);
        while(t != p - 1 && m != 1 && m != p - 1) {
            m = qmul(m, m, p);
            t <<= 1;
        }
        if(m != p - 1 && !(t & 1)) return 0;
    }
    return 1;
}
```

1.4 快速幂，矩阵快速幂

```
int poww(int a,int b){
    int ans=1,base=a;
    while(b!=0){
        if(b&1!=0)
            ans*=base;
        base*=base;
        b>>=1;
   }
    return ans;
}


///矩阵快速幂

const int N=10;
int tmp[N][N];
void multi(int a[][N],int b[][N],int n)
{
    memset(tmp,0,sizeof tmp);
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
        for(int k=0;k<n;k++)
        tmp[i][j]+=a[i][k]*b[k][j];
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
        a[i][j]=tmp[i][j];
}
int res[N][N];
void Pow(int a[][N],int n)
{
    memset(res,0,sizeof res);//n 是幂, N 是矩阵大小
    for(int i=0;i<n;i++) res[i][i]=1;
    while(n)
    {
        if(n&1)
            multi(res,a,N);//res=res*a;复制直接在 multi 里面实现了;
        multi(a,a,N);//a=a*a
        n>>=1;
    }
}
```

1.5 卢卡斯、大组合数取模

```
LL PowMod(LL a,LL b,LL MOD){
    LL ret=1;
    while(b){
        if(b&1) ret=(ret*a)%MOD;
        a=(a*a)%MOD;
        b>>=1;
    }
    return ret;
}
LL fac[100005];
LL Get_Fact(LL p){
    fac[0]=1;
    for(LL i=1;i<=p;i++)
        fac[i]=(fac[i-1]*i)%p;  //预处理阶乘
}
LL Lucas(LL n,LL m,LL p){
    LL ret=1;
    while(n&&m){
        LL a=n%p,b=m%p;
        if(a<b) return 0;
        ret=(ret*fac[a]*PowMod(fac[b]*fac[a-b]%p,p-2,p))%p;
        n/=p;
        m/=p;
    }
    return ret;
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        LL n,m,p;
        scanf("%I64d%I64d%I64d",&n,&m,&p);
        Get_Fact(p);

        printf("%I64d\n",Lucas(n,m,p));
    }
    return 0;
}
卢卡斯定理
O(logp(n)*p)
```

1.6 中国剩余定理（不互质情况）

```
using namespace std;
const int maxn=100005;
const int inf=0x7fffffff;
typedef long long ll;
void ex_gcd(ll a,ll b,ll &d,ll &x,ll &y)//扩展欧几里得
{
    if(!b) {d=a;x=1;y=0;}
    else{
        ex_gcd(b,a%b,d,y,x);
        y-=x*(a/b);
    }
}
ll ex_crt(ll *m,ll *r,int n)
{
    ll M=m[1],R=r[1],x,y,d;
    for(int i=2;i<=n;i++){
        ex_gcd(M,m[i],d,x,y);
        if((r[i]-R)%d) return -1;
        x=(r[i]-R)/d*x%(m[i]/d);
        R+=x*M;
        M=M/d*m[i];
        R%=M;
    }
    return R>0?R:R+M;
}
int main()
{
    int t,n;
    scanf("%d",&t);
    for(int cas=1;cas<=t;cas++){
        scanf("%d",&n);
        ll m[maxn],r[maxn];//m 除数，r 余数
        for(int i=1;i<=n;i++) scanf("%lld",&m[i]);
        for(int i=1;i<=n;i++) scanf("%lld",&r[i]);
        printf("Case %d: %I64d\n",cas,ex_crt(m,r,n));
    }
    return 0;
}
```

关于原理的推导：

$$\begin{cases} x = a_1 \pmod{n_1} \\ x = a_2 \pmod{n_2} \end{cases} \quad \textbf{(0)}$$

$$\begin{aligned} x &= n_1k_1 + a_1 \\ x &= n_2k_2 + a_2 \end{aligned} \quad \textbf{(1)}$$

$$n_1k_1 + a_1 = n_2k_2 + a_2$$
$$n_1k_1 = (a_2 - a_1) + n_2k_2$$
$$n_1k_1 = (a_2 - a_1) \pmod{n_2}$$

显然，要想有解，必有 gcd(n1,n2)|(a2-a1)。设 gcd(n1,n2)=d，$c = a_2 - a_1$，则有：

$$\frac{n_1}{d}k_1 = \frac{c}{d} \pmod{\frac{n_2}{d}}$$

$$k_1 = \frac{c}{d} * \left(\frac{n_1}{d}\right)^{-1} \pmod{\frac{n_2}{d}}$$

令 $K = \frac{c}{d} * \left(\frac{n_1}{d}\right)^{-1}$，则 $k_1 = y\frac{n_2}{d} + K$，将其带入(1)式得：

$$x = n_1\left(y\frac{n_2}{d} + K\right) + a_1$$

$$= \frac{n_1n_2}{d}y + n_1K + a_1$$

即：

$$x = n_1K + a_1 \pmod{\frac{n_1n_2}{d}}$$

$$x = a \pmod{n} \quad \textbf{(2)}$$

式(2)中：

$$a = n_1K + a_1, \quad n = \frac{n_1n_2}{d}$$

这样，成功的将**(0)**式的两个方程合并为式**(2)**的一个方程。
最终，合并 k 个方程的最小 x 的值为 a%n。

```
typedef __int64 int64;
int64 Mod;

int64 gcd(int64 a, int64 b)
{
    if(b==0)
        return a;
    return gcd(b,a%b);
}

int64 Extend_Euclid(int64 a, int64 b, int64&x, int64& y)
```

```
{
    if(b==0)
    {
        x=1,y=0;
        return a;
    }
    int64 d = Extend_Euclid(b,a%b,x,y);
    int64 t = x;
    x = y;
    y = t - a/b*y;
    return d;
}

//a 在模 n 乘法下的逆元，没有则返回-1
int64 inv(int64 a, int64 n)
{
    int64 x,y;
    int64 t = Extend_Euclid(a,n,x,y);
    if(t != 1)
        return -1;
    return (x%n+n)%n;
}

//将两个方程合并为一个
bool merge(int64 a1, int64 n1, int64 a2, int64 n2, int64& a3, int64& n3)
{
    int64 d = gcd(n1,n2);
    int64 c = a2-a1;
    if(c%d)
        return false;
    c = (c%n2+n2)%n2;
    c /= d;
    n1 /= d;
    n2 /= d;
    c *= inv(n1,n2);
    c %= n2;
    c *= n1*d;
    c += a1;
    n3 = n1*n2*d;
    a3 = (c%n3+n3)%n3;
    return true;
}

//求模线性方程组 x=ai(mod ni),ni 可以不互质
```

```
int64 China_Reminder2(int len, int64* a, int64* n)
{
    int64 a1=a[0],n1=n[0];
    int64 a2,n2;
    for(int i = 1; i < len; i++)
    {
        int64 aa,nn;
        a2 = a[i],n2=n[i];
        if(!merge(a1,n1,a2,n2,aa,nn))
            return -1;
        a1 = aa;
        n1 = nn;
    }
    Mod = n1;
    return (a1%n1+n1)%n1;
}
int64 a[1000],b[1000];
int main()
{
    int i;
    int k;
    while(scanf("%d",&k)!=EOF)
    {
        for(i = 0; i < k; i++)
            scanf("%I64d %I64d",&a[i],&b[i]);
        printf("%I64d\n",China_Reminder2(k,b,a));
    }
    return 0;
}
```

1.7 对前 n 个数分解质因数

```
#include<bits/stdc++.h>
using namespace std;
#define N 2000000
vector <pair<long long,int>> d[2000004];

void init()
{
    d[1].push_back({1,1});
    for(long long i=2;i<=N;i++)
    {
        if(d[i].empty())
        {
            for(long long j=i;j<=N;j+=i)
                d[j].push_back({i,1});

            long long w=i*i;
            while(w<=N)
            {
                for(long long j=w;j<=N;j+=w)
                    d[j][d[j].size()-1].second++;
                w*=i;
            }

        }
    }
}


int main()
{
    init();
    int t;
    while(cin >> t)
    for(int i=0;i<d[t].size();i++)
    {
        cout << "factor " << d[t][i].first << " is " << d[t][i].second << endl;
    }
}
```

1.8 将根式转换为连分数形式

```
#include<bits/stdc++.h>
using namespace std;

vector<int> a;
vector<int> b;
vector<int> c;

void get_fractions(int n)
{
    a.clear();
    b.clear();
    c.clear();
    int AA;
    AA = floor(sqrt(n));
    int c0 = n-AA*AA;
    int a0 = (sqrt(n)+AA)/(n-AA*AA);
    int b0 = a0*(n-AA*AA)-AA;
    c.push_back(c0);
    a.push_back(a0);
    b.push_back(b0);
    int i=0;
    do
    {
        int ccc = (n - b[i]*b[i])/c[i];
        int aaa = (sqrt(n)+b[i])/ccc;
        int bbb = aaa*ccc-b[i];
        if(a[0]==aaa&&b[0]==bbb&&c[0]==ccc)
            break;
        c.push_back(ccc);
        a.push_back(aaa);
        b.push_back(bbb);
        i++;
    }while(1);
    printf("[%d;(",AA);
    vector<int>::iterator it;
    for(it=a.begin();it!=a.end();it++)
        cout<<*it<<(it==a.end()-1?")]":",");
    //for(auto& x : a) cout << x;
    cout << endl;
}
int main()
```

```
{
    int n;
    while(cin >> n)
        get_fractions(n);
    return 0;
}
```

1.9 Meissel-Lehmer 算法（求 1e11 内质数个数）

```
//Meisell-Lehmer
//G++ 218ms 43252k
#include<cstdio>
#include<cmath>
using namespace std;
#define LL long long
const int N = 5e6 + 2;
bool np[N];
int prime[N], pi[N];
int getprime()
{
    int cnt = 0;
    np[0] = np[1] = true;
    pi[0] = pi[1] = 0;
    for(int i = 2; i < N; ++i)
    {
        if(!np[i]) prime[++cnt] = i;
        pi[i] = cnt;
        for(int j = 1; j <= cnt && i * prime[j] < N; ++j)
        {
            np[i * prime[j]] = true;
            if(i % prime[j] == 0)   break;
        }
    }
    return cnt;
}
const int M = 7;
const int PM = 2 * 3 * 5 * 7 * 11 * 13 * 17;
int phi[PM + 1][M + 1], sz[M + 1];
```

```
void init()
{
    getprime();
    sz[0] = 1;
    for(int i = 0; i <= PM; ++i)  phi[i][0] = i;
    for(int i = 1; i <= M; ++i)
    {
        sz[i] = prime[i] * sz[i - 1];
        for(int j = 1; j <= PM; ++j) phi[j][i] = phi[j][i - 1] - phi[j /
prime[i]][i - 1];
    }
}
int sqrt2(LL x)
{
    LL r = (LL)sqrt(x - 0.1);
    while(r * r <= x)   ++r;
    return int(r - 1);
}
int sqrt3(LL x)
{
    LL r = (LL)cbrt(x - 0.1);
    while(r * r * r <= x)   ++r;
    return int(r - 1);
}
LL getphi(LL x, int s)
{
    if(s == 0)  return x;
    if(s <= M)  return phi[x % sz[s]][s] + (x / sz[s]) * phi[sz[s]][s];
    if(x <= prime[s]*prime[s])   return pi[x] - s + 1;
    if(x <= prime[s]*prime[s]*prime[s] && x < N)
    {
        int s2x = pi[sqrt2(x)];
        LL ans = pi[x] - (s2x + s - 2) * (s2x - s + 1) / 2;
        for(int i = s + 1; i <= s2x; ++i) ans += pi[x / prime[i]];
        return ans;
    }
    return getphi(x, s - 1) - getphi(x / prime[s], s - 1);
}
LL getpi(LL x)
{
    if(x < N)   return pi[x];
    LL ans = getphi(x, pi[sqrt3(x)]) + pi[sqrt3(x)] - 1;
    for(int i = pi[sqrt3(x)] + 1, ed = pi[sqrt2(x)]; i <= ed; ++i) ans -= getpi(x
/ prime[i]) - i + 1;
```

```
    return ans;
}
LL lehmer_pi(LL x)
{
    if(x < N)   return pi[x];
    int a = (int)lehmer_pi(sqrt2(sqrt2(x)));
    int b = (int)lehmer_pi(sqrt2(x));
    int c = (int)lehmer_pi(sqrt3(x));
    LL sum = getphi(x, a) +(LL)(b + a - 2) * (b - a + 1) / 2;
    for (int i = a + 1; i <= b; i++)
    {
        LL w = x / prime[i];
        sum -= lehmer_pi(w);
        if (i > c) continue;
        LL lim = lehmer_pi(sqrt2(w));
        for (int j = i; j <= lim; j++) sum -= lehmer_pi(w / prime[j]) - (j
- 1);
    }
    return sum;
}
int main()
{
    init();
    LL n;
    while(~scanf("%lld",&n))
    {
        printf("%lld\n",lehmer_pi(n));
    }
    return 0;
}
```

2 字符串

2.1 KMP 字符串匹配

```
const int MAX = 200;
int nextt[MAX];

void getNext(string t)
{
    int j, k;
    memset(nextt,0,sizeof(nextt));
    j = 0; k = -1; nextt[0] = -1;
    while(j < t.length())
    {
        if(k == -1 || t[j] == t[k])
            nextt[++j] = ++k;
        else
            k = nextt[k];
    }
}
/*
返回模式串 t 在主串 s 中首次出现的位置
返回的位置是从 0 开始的。
*/
int KMP_Index(string t,string s)
{
    int i = 0, j = 0;
    getNext(t);

    while(i < s.length() && j < t.length())
    {
        if(j == -1 || s[i] == t[j])
        {
            i++; j++;
        }
        else
            j = nextt[j];
    }
    if(j == t.length())
        return i - t.length();
    else
        return -1;
```

```
}
/*
返回模式串 t 在主串 s 中出现的次数
*/
int KMP_Count(string t,string s)
{
    int ans = 0;
    int i, j = 0;

    if(s.length() == 1 && t.length() == 1)
    {
        if(s[0] == t[0])
            return 1;
        else
            return 0;
    }
    getNext(t);
    for(i = 0; i < s.length(); i++)
    {
        while(j > 0 && s[i] != t[j])
            j = nextt[j];
        if(s[i] == t[j])
            j++;
        if(j == t.length())
        {
            ans++;
            j = nextt[j];
        }
    }
    return ans;
}
```

3 数据结构

3.1 并查集

```
#define MAX 10000;

struct UF
{
  int ranking;
  int parent;
}UF[MAX];

void init(int n)
{
  for(int i=0;i<=n;i++)
  {
    UF[i].parent=i;
    UF[i].ranking=0;
  }
}

int get_parent(int x)
{
   if(UF[x].parent==x) return x;
   return get_parent(UF[x].parent);
}

void Union(int a,int b)
{
  a=get_parent(a);
  b=get_parent(b);
  if(UF[a].rank>UF[b].rank) UF[b].parent = UF[a].parent;
  else
  {
    UF[a].parent = UF[b].parent;
    if(UF[a].rank==UF[b].rank) UF[a].rank++;
  }
}
```

3.2 线段树

```
const int maxn = 100007;

struct Tree
{
    int l,r,sum;
    int vis;
}t[maxn<<2];

void push_up(int step)
{
    t[step].sum = t[step*2].sum + t[step*2+1].sum;
}

void push_down(int step)
{
    if(!t[step].vis) return;
    t[step*2].vis += t[step].vis;
    t[step*2+1].vis += t[step].vis;
    t[step*2].sum += t[step].vis*(t[step*2].r-t[step*2].l+1);
    t[step*2+1].sum += t[step].vis*(t[step*2+1].r-t[step*2+1].l+1);
    t[step].vis = 0;
}


void build(int l,int r,int step)
{
    t[step].l = l,t[step].r = r,t[step].sum = t[step].vis = 0;
    if(l==r) return;
    int mid = (l+r)/2;
    build(l,mid,step*2);
    build(mid+1,r,step*2+1);
}

void update(int l,int r,int val,int step)
{
    if(l==t[step].l&&r==t[step].r)
    {
        t[step].vis += val;
        t[step].sum += (r-l+1)*val;
        return;
    }
```

```
    int mid = (t[step].l+t[step].r)/2;
    push_down(step);
    if(r<=mid) update(l,r,val,step*2);
    else if(l>mid) update(l,r,val,step*2+1);
    else update(l,mid,val,step*2),update(mid+1,r,val,step*2+1);
    push_up(step);
}

int query(int l,int r,int step)
{
    if(l==t[step].l&&r==t[step].r)
        return t[step].sum;
    int mid = (t[step].l+t[step].r)/2;
    push_down(step);
    if(r<=mid) return query(l,r,step*2);
    else if(l>mid) return query(l,r,step*2+1);
    else return query(l,mid,step*2)+query(mid+1,r,step*2+1);
}
```

3.3 zkw 线段树

3.4 主席树

3.5 树状数组

4 计算几何

4.1 基础模板

```
struct spot ///存储点，也可指代向量
{
    double x;
    double y;
    double z;
};

spot cross(const spot &a,const spot &b) ///计算向量a和向量b的叉乘(有顺序)
{
    spot w;
    w.x = a.y*b.z-b.y*a.z;
    w.y = a.z*b.x-a.x*b.z;
    w.z = a.x*b.y-a.y*b.x;
    return w;
}

double dot(const spot &a,const spot &b) ///计算向量a和向量b的点乘积
{
    return a.x*b.x+a.y*b.y+a.z*b.z;
}

double norm(const spot &a) ///计算向量a的模长
{
    return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
}
```

5 杂项

5.1 罗马-数字转换

```
#include<bits/stdc++.h>
using namespace std;
char str[10]="IVXLCDM";
int num[10]={1,5,10,50,100,500,1000};


int roman_to_num(char s[])
{
    int len=strlen(s);
    int cnt=0,a[20];
    for(int i=0;i<len;i++)
    {
        int f=0,t;
        if(s[i]==s[i+1]&&i!=len-1)
        {
            if(s[i]==s[i+2]&&i!=len-2)
            {
                f=2;
            }
            else
                f=1;
        }
        for(int k=0;k<7;k++)
            if(s[i]==str[k])
        {
            t=k;
            break;
        }
        a[cnt++]=num[t]*(f+1);
        i+=f;


    }
    int sum=0;
    for(int i=0;i<cnt;i++)
        sum+=a[i];
```

```
    for(int i=0;i<cnt-1;i++)
    {
        if(a[i]<a[i+1])sum-=2*a[i];
    }
    return sum;
}


string num_to_roman(int num)
{
    char* digit[10] = {"","I","II","III","IV","V","VI","VII","VIII","IX"};
        char* ten[10] =
{"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};
        char* hundreds[10] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC",
"DCCC", "CM"};
        char* thousand[7] = {"","M","MM","MMM","MMMM","MMMMM","MMMMMM"};
        string ans;

        ans = string(thousand[num/1000]) + string(hundreds[num%1000/100])
+ string(ten[num%100/10]) + string(digit[num%10]);
        return ans;
}


int main()
{
    char a[101];
    scanf("%s",a);
    cout << roman_to_num(a) << endl;
    cout << num_to_roman(roman_to_num(a)) << endl;
}
```

5.2 Bm 算法求解线性递推

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (long long i=a;i<n;i++)
#define per(i,a,n) for (long long i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((long long)(x).size())
typedef vector<long long> VI;
typedef long long ll;
typedef pair<long long,long long> PII;
const ll mod=1e9+7;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0);
for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head

long long _,n;
namespace linear_seq
{
    const long long N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<long long> Md;
    void mul(ll *a,ll *b,long long k)
    {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k)
            _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (long long i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md))
_c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    long long solve(ll n,VI a,VI b)
    { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
//        printf("%d\n",SZ(b));
        ll ans=0,pnt=0;
        long long k=SZ(a);
        assert(SZ(a)==SZ(b));
```

```
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (long long p=pnt;p>=0;p--)
        {
            mul(res,res,k);
            if ((n>>p)&1)
            {
                for (long long i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md))
res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s)
    {
        VI C(1,1),B(1,1);
        long long L=0,m=1,b=1;
        rep(n,0,SZ(s))
        {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n)
            {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            }
            else
            {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
```

```
        }
        return C;
    }
    long long gao(VI a,ll n)
    {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};


int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        int n;
        cin >> n;

printf("%I64d\n",linear_seq::gao(VI{2,24,96,416,1536,5504,18944,64000,21
2992,702464},n-1));
    }
}
```

5.3 FFT 求多项式乘法

```
#include <map>
#include <set>
#include <cmath>
#include <ctime>
#include <stack>
#include <queue>
#include <cstdio>
#include <cctype>
#include <bitset>
#include <string>
#include <vector>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <functional>
#define fuck(x) cout<<"["<<x<<"]";
#define FIN freopen("input.txt","r",stdin);
#define FOUT freopen("output.txt","w+",stdout);
//#pragma comment(linker, "/STACK:102400000,102400000")
using namespace std;
typedef long long LL;
typedef pair<int, int> PII;

const int MX = 3e5 + 5;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;

const double pi = acos(-1.0);
struct cp {
    double x, y;
    cp() {}
    cp (double x, double y): x(x), y(y) {}
    inline cp operator + (const cp &b) {
        return cp(x + b.x, y + b.y);
    }
    inline cp operator - (const cp &b) {
        return cp(x - b.x, y - b.y);
    }
    inline cp operator * (const cp &b) {
        return cp(x * b.x - y * b.y, x * b.y + y * b.x);
    }
```

```
} a[MX], b[MX];

int r[MX];
void fft(cp a[], int opt, int n) {
    for(int i = 0; i < n; i++) {
        if(i < r[i]) swap(a[i], a[r[i]]);
    }
    for(int i = 1; i < n; i <<= 1) {
        cp wn(cos(pi / i), opt * sin(pi / i));
        for(int p = i << 1, j = 0; j < n; j += p) {
            cp w(1, 0);
            for(int k = 0; k < i; k++, w = wn * w) {
                cp x = a[j + k], y = w * a[j + k + i];
                a[j + k] = x + y; a[j + k + i] = x - y;
            }
        }
    }
}
/*多项式 a，最高次为 n，多项式 b，最高次为 m
从 0 到 n 项的系数
卷积结果等于后来 a[].x
复杂度 O(nlogn)，最后的最高项为 n+m
*/
void solve(cp a[], cp b[], int n, int m) {
    int l = 0, nn, nm = n + m;
    for(nn = 1; nn <= nm; nn <<= 1) l++;
    for(int i = n + 1; i <= nn; i++) a[i] = cp(0, 0);
    for(int i = m + 1; i <= nn; i++) b[i] = cp(0, 0);
    n = nn; m = nm;

    for(int i = 0; i < n; i++) {
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    }
    fft(a, 1, n); fft(b, 1, n);
    for(int i = 0; i <= n; i++) {
        a[i] = a[i] * b[i];
    }
    fft(a, -1, n);
    for(int i = 0; i <= m; i++) {
        a[i].x /= n;
    }
}

int main() {
```

```
    int n, m; //FIN;
    scanf("%d%d", &n, &m);
    for(int i = 0; i <= n; i++) scanf("%lf", &a[i].x);
    for(int i = 0; i <= m; i++) scanf("%lf", &b[i].x);
    solve(a, b, n, m);
    for(int i = 0; i <= n + m; i++) {
        printf("%d%c", (int)(a[i].x + 0.5), i == n + m ? '\n' : ' ');
    }
}
```

5.4 拉格朗日插值

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;

double Lagrange(int N,vector<double>&X,vector<double>&Y,double x);

int main(){
  char a='n';
  do{
    cout<<"请输入差值次数 n 的值: "<<endl;
    int N;
    cin>>N;
    vector<double>X(N,0);
    vector<double>Y(N,0);
    cout<<"请输入插值点对应的值及函数值(Xi,Yi): "<<endl;
    for(int a=0;a<N;a++){
        cin>>X[a]>>Y[a];
    }
    cout<<"请输入要求值 x 的值: "<<endl;
    double x;
    cin>>x;
    double result=Lagrange(N,X,Y,x);
    cout<<"由拉格朗日插值法得出结果:  "<<result<<endl;
    cout<<"是否要继续?(y/n): ";
    cin>>a;
  }while(a=='y');
  return 0;
```

```
}

double Lagrange(int N,vector<double>&X,vector<double>&Y,double x){
  double result=0;
  for(int i=0;i<N;i++){
    double temp=Y[i];
    for(int j=0;j<N;j++){
    if(i!=j){
        temp = temp*(x-X[j]);
        temp = temp/(X[i]-X[j]);
   }
  }
  result += temp;
}
 return result;
};
```