胡写的板子 DA☆ZE

一些字符串处理/三个自动机

后缀自动机（SAM）

- 关于广义后缀自动机：
  - 给定n个字符串，询问每个字符串有多少子串（不包括空串）是所有n个字符串中至少k个字符串的子串（包括本身）。
  - 对于多串问题，普通SAM已经无法胜任。有各种应对这类多串问题的方法：

  (1) 直接建SAM，每次插入新串时将lst设为1即可，不会证明正确性。

  $ps:$ 这个必须每个串不相同，否则可能会炸

  (2) 和SA一样在多个串接在一起中间加入分隔符'#'，然后直接建SAM。

  (3) 离线做法：对所有串建出Trie，然后BFS遍历Trie，每个点在父亲的基础上$extend$即可（参考楼下代码）。时空复杂度$O(|A||T|)$。

  (4) 在线做法：模板如下，时间复杂度$O(|A||T|+G(T))$，空间复杂度$O(|A||T|)$，其中$|T|$是Trie的大小，$|A|$是字符集大小，$G(T)$是Trie上所有叶子深度之和。

$ $

- 结论：如果题目是以多个模板串形式给出的，建议使用在线做法（离线做法虽然复杂度优秀但需要先建出Trie所以大部分情况常数会较大）。如果是直接以Trie形式给出的，那么直接用离线算法就好了。

```
struct SAM{
    struct node{
        int ch[26];
        int len,fa;
        node(){memset(ch,0,sizeof(ch));len=0;}
    }dian[maxn<<1];

    int tot=1;
    /*------(3) trie 上跑 or 多个字符串不相同，开始用pos=add(c,1),之后pos=add(c,pos)------*/
    int add(int c,int last_p){
        int p=last_p;int np=++tot;
        //（1）直接加可以改成int p=las;int np=las=++tot;
        // 外界加一个int las=1;返回值改void，删掉return，int last_p

        dian[np].len=dian[p].len+1;
        for(;p&&!dian[p].ch[c];p=dian[p].fa)dian[p].ch[c]=np;
        if(!p)dian[np].fa=1;
        else
        {
            int q=dian[p].ch[c];
            if(dian[q].len==dian[p].len+1)dian[np].fa=q;
            else
            {
                int nq=++tot;dian[nq]=dian[q];
                dian[nq].len=dian[p].len+1;
                dian[q].fa=dian[np].fa=nq;
                for(;p&&dian[p].ch[c]==q;p=dian[p].fa)dian[p].ch[c]=nq;
            }
        }
        return np;
    }
```

```
    long long count_diff(){
        long long res=0;
        for(int i=1;i<=tot;i++)res+=dian[i].len-dian[dian[i].fa].len;
        return res;
    }//本质不同字符串数量

    //传奇操作，O(n)后缀bfs序--后缀链接fail的bfs序排名
    void gos(){
        for(int i=1;i<=tot;i++)bfnsum[dian[i].len]++;//len长有多少
        for(int i=1;i<=tot;i++)bfnsum[i]+=bfnsum[i-1];//前缀和
        for(int i=1;i<=tot;i++)bfn[bfnsum[dian[i].len]--]=i;//一个一个放回去
    }

}S_A_M;

//-------endpos集合算数-----
long long anslist[maxn];//长度为i的最大endpos集合数
int n;
struct SAM{
    struct node{
        int ch[26];
        int len,fa;
        long long cnttime;//endpos集合大小计数
        node(){memset(ch,0,sizeof(ch));len=0;cnttime=1;}
    }dian[maxn<<1];

    int tot=1;
    int las=1;
    /*------(3) trie 上跑 or 多个字符串不相同，开始用pos=add(c,1),之后pos=add(c,pos)------*/
    void add(int c){
        int p=las;int np=las=++tot;
        //（1）直接加可以改成int p=las;int np=las=++tot;
        // 外界加一个int las=1;返回值改void，删掉return，int last_p
        dian[np].len=dian[p].len+1;
        dian[np].cnttime=1;//正常开点就是1
        for(;p&&!dian[p].ch[c];p=dian[p].fa)dian[p].ch[c]=np;
        if(!p)dian[np].fa=1;
        else
        {
            int q=dian[p].ch[c];
            if(dian[q].len==dian[p].len+1)dian[np].fa=q;
            else
            {
                int nq=++tot;dian[nq]=dian[q];
                dian[nq].len=dian[p].len+1;
                dian[q].fa=dian[np].fa=nq;
                dian[nq].cnttime=0;//克隆的点不能重复计数
                for(;p&&dian[p].ch[c]==q;p=dian[p].fa)dian[p].ch[c]=nq;
            }
        }
    }

    long long count_diff(){
        long long res=0;
        for(int i=1;i<=tot;i++)res+=dian[i].len-dian[dian[i].fa].len;
        return res;
    }//本质不同字符串数量

    int bfn[maxn<<1],bfnsum[maxn<<1];

    void gos(){
        for(int i=1;i<=tot;i++)bfnsum[dian[i].len]++;
        for(int i=1;i<=tot;i++)bfnsum[i]+=bfnsum[i-1];
```

```
        for(int i=1;i<=tot;i++)bfn[bfnsum[dian[i].len]--]=i;//后缀树bfs序

        for(int i=tot;i>=1;i--)dian[dian[bfn[i]].fa].cnttime+=dian[bfn[i]].cnttime;//反向
倒推
    }

    void solve(){
        for(int i=1;i<=tot;i++){
            int rlen=dian[i].len;
            anslist[rlen]=max(anslist[rlen],dian[i].cnttime);
        }//性质，单调不升。一个显然的例子是长度为4的endpos集合大小肯定不会比3要大，否则3可以和4一样大
        for(int i=n;i>=1;i--)anslist[i]=max(anslist[i],anslist[i+1]);
        //反向更新，i被置为i+1~n的anslist的max
        for(int i=1;i<=n;i++)printf("%lld\n",anslist[i]);
    }

}S_A_M;

//-------------------(4)网上dalao代码---------------------//
int work(int p,int c){
    int nq=++nd,q=son[p][c]; mx[nq]=mx[p]+1;
    fa[nq]=fa[q]; fa[q]=nq;
    memcpy(son[nq],son[q],sizeof(son[q]));
    while (p && son[p][c]==q) son[p][c]=nq,p=fa[p];
    return nq;
}

int ext(int p,int c){
    if (son[p][c]){
        int q=son[p][c];
        if (mx[q]==mx[p]+1) return q; else return work(p,c);
    }else{
        int np=++nd; mx[np]=mx[p]+1;
        while (p && !son[p][c]) son[p][c]=np,p=fa[p];
        if (!p) fa[np]=1;
        else{
            int q=son[p][c];
            if (mx[q]==mx[p]+1) fa[np]=q; else fa[np]=work(p,c);
        }
        return np;
    }
}

//给定n个字符串，询问每个字符串有多少子串（不包括空串）是所有n个字符串中至少k个字符串的子串（包括本
身）。//
void solve(){
    int u; ll ans;
    rep(i,1,n){
        u=1;
        rep(j,0,len[i]){
            u=son[u][s[i][j]-'a']; int p=u;
            while (p && vis[p]!=i) tot[p]++,vis[p]=i,p=fa[p];
        }
    }
    radix();
    rep(i,2,nd) u=q[i],f[u]=f[fa[u]]+(tot[u]>=k ? mx[u]-mx[fa[u]] : 0);
    rep(i,1,n){
        u=1; ans=0;
        rep(j,0,len[i]) u=son[u][s[i][j]-'a'],ans+=f[u];
        printf("%lld ",ans);
    }
}
```

AC自动机

```
struct AC{
    struct acnode{
        acnode *next[26];
        acnode *fail;
        int sum;
        acnode(){
            sum=0;
            for(int i=0;i<26;i++)next[i]=NULL;
            fail=NULL;
        }
    };
    acnode *root=new acnode;
    void insert(string s){
        int len=s.length();
        acnode *t=root;
        for(int i=0;i<len;i++){
            int cut=s[i]-'a';
            if(t->next[cut]==NULL){t->next[cut]=new acnode;}
            t=t->next[cut];
        }
        t->sum++;
    }

    void getfail(){
        queue <acnode*> q;
        acnode *t=root;
        for(int i=0;i<26;i++){
            if(t->next[i]!=NULL){
                t->next[i]->fail=root;
                q.push(t->next[i]);
            }
        }
        while(!q.empty()){
            t=q.front();
            q.pop();
            for(int i=0;i<26;i++){
                if(t->next[i]!=NULL){
                    acnode *p=t->fail;
                    while(p!=NULL){
                        if(p->next[i]!=NULL){
                            t->next[i]->fail=p->next[i];
                            break;
                        }
                        p=p->fail;
                    }
                    if(p==NULL){
                        t->next[i]->fail=root;
                    }
                    q.push(t->next[i]);
                }
            }
        }
    }
    int ans=0;
    void acfun(string s){
        acnode *p=root;
        int len=s.length();
        for(int i=0;i<len;i++){
            int cut=s[i]-'a';
```

```
            while(p->next[cut]==NULL && p!=NULL)p=p->fail;
            p=p->next[cut];
            if(p==NULL)p=root;

            acnode *t=p;
            while(t!=root){
                if(t->sum>=0){
                    ans+=t->sum;
                    t->sum=-1;
                }
                else break;
                t=t->fail;
            }
        }
    }

}A_C;
```

- lrj的AC自动机（感觉比我优美多了）：

```
待补充qwq，P215蓝book
```

回文树（PAM）/ 回文自动机

- len[i]表示编号为i的节点表示的回文串的长度（一个节点表一个回文串）
- next[i][c]表示编号为i的节点表示的回文串在两边添加字符以后变成的回文串的编号（和字典树类似）。
- fail[i]表示节点i失配以后跳转不等于自身的节点i表示的回串的最长后缀回文串（和AC自动机类似）。
- cnt[i]表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的）
- num[i]表示以节点i表示的最长回文串的最右端点为回文串结的回文串个数。
- last指向新添加一个字母后所形成的最长回文串表示的节点。
- S[i]表示第i次添加的字符（一开始设S[0] = -1（可以是任一个在串S中不会出现的字符））。
- p表示添加的节点个数。
- n表示添加的字符个数。

```
struct Palindromic_Tree {
    int next[MAXN][N] ;//next指针，next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail指针，失配后跳转到fail指针指向的节点
    int cnt[MAXN] ;
    int num[MAXN] ;
    int len[MAXN] ;//len[i]表示节点i表示的回文串的长度
    int S[MAXN] ;//存放添加的字符
    int last ;//指向上一个字符所在的节点，方便下一次add
    int n ;//字符数组指针
    int p ;//节点指针,p代表了节点个数，正好是（0 ~ p-1）

    int newnode ( int l ) {//新建节点
        for ( int i = 0 ; i < N ; ++ i ) next[p][i] = 0 ;
        cnt[p] = 0 ;
        num[p] = 0 ;
        len[p] = l ;
        return p ++ ;
    }

    void init () {//初始化
        p = 0 ;
        newnode (  0 ) ;
        newnode ( -1 ) ;//顺序别变啊...
        last = 0 ;
        n = 0 ;
```

```
        S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1 ;
    }

    int get_fail ( int x ) {//和KMP一样，失配后找一个尽量最长的
        while ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;
        return x ;
    }

    void add ( int c ) {
        c -= 'a' ;
        S[++ n] = c ;
        int cur = get_fail ( last ) ;//通过上一个回文串找这个回文串的匹配位置
        if ( !next[cur][c] ) {//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now = newnode ( len[cur] + 2 ) ;//新建节点
            fail[now] = next[get_fail ( fail[cur] )][c] ;//和AC自动机一样建立fail指针，以便失
配后跳转
            next[cur][c] = now ;
            num[now] = num[fail[now]] + 1 ;//看上去是本质不同的num
        }
        last = next[cur][c] ;
        cnt[last] ++ ;
    }

    void count () {
        for ( int i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;
        //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串！
    }

//你已经可以熟练操作这棵树了！
/*-------------------------------------------------------------------------------*/
//将串中本质不同的回文子串组成集合S，任取S中的两个元素(a，b)，问a是b的子串的对数
    bool vis[maxn];
    long long nxtr[maxn],fatr[maxn];
    void dfs(int x){
        if(x!=0&&x!=1)nxtr[x]=1;
        vector <int> A;
        for(int i=x;i>1;i=fail[i]){
            if(vis[i])break;
            vis[i]=1;
            A.push_back(i);
            fatr[x]++;
        }
        for(int i=0;i<N;i++){
            int y=next[x][i];
            if(y){
                dfs(y);
                nxtr[x]+=nxtr[y];
            }
        }
        for(int i=A.size()-1;i>=0;i--){
            vis[A[i]]=0;
        }//纯vis回调可能交叉线路爆炸...导致河流袭夺一样的事件
    }

    long long solve(){
        long long ans=0;
        memset(nxtr,0,sizeof(nxtr));
        memset(fatr,0,sizeof(fatr));
        dfs(0);dfs(1);
        for(int i=2;i<p;i++)ans+=(nxtr[i]*fatr[i]-1);
        return ans;
    }
```

```
/*-------------------------------------------------------------------------------*/
}P_A_M;
```

kmp（mp） 和 exkmp

kmp（mp）相关

- mp算法=kmp算法低配版，但是时间复杂度已经到了下限，并且足够使用
- lrj的kmp（实际上是mp，感觉我学了大几年都是mp算法）：

```
void getFail(char* P,int* f) {
    int m=strlen(P);
    f[0]=0;
    f[1]=0;
    for(int i=1; i<m; i++) {
        int j=f[i];
        while(j&&P[i]!=P[j])j=f[j];
        f[i+1]= P[i]==P[j] ? j+1 : 0;
    }
}
//这个fail是跳到哪里哦，举个例子：
//pos(i): 0 1 2 3 4 5 6 7 8 9 10 11
//p:      A B R A C A D A B R A  #
//fail:   0 0 0 0 1 0 1 0 1 2 3  4

void find(char* T,char* P,int* f) {
    int n=strlen(T),m=strlen(P);
    getFail(P,f);
    int j=0;
    for(int i=0; i<n; i++) {
        while(j&&P[j]!=T[i])j=f[j];
        if(P[j]==T[i])j++;
        if(j==m)printf("%d\n",i-m+1);//找到了!
    }
}
```

传奇exkmp

- exkmp,用来求一个串S和另一个串T的**所有后缀**的最长公共前缀~
- 俩数组，一个next（和上边那个不一样！这是开头长度，fail是结尾长度）是去匹配串（S）的，另一个是extend就是要的结果啦，是被匹配的串（T）的
- 直接贴代码，能看懂的...

```
int Next[maxn],extend[maxn];
void getNext(char str[]){//[0,len)
    int i=0,len=strlen(str);
    Next[0]=len;
    while(str[i]==str[i+1] && i+1<len)i++;
    Next[1]=i;
    int po=1;
    for(i=2;i<len;i++){
        if(Next[i-po]+i<Next[po]+po)Next[i]=Next[i-po];
        else {
            int j=Next[po]+po-i;
            if(j<0)j=0;
            while(i+j<len&&str[j]==str[j+i])j++;
            Next[i]=j;po=i;
        }
    }
}
```

```
void exkmp(char s1[],char s2[]){
    int i=0,len=strlen(s1),l2=strlen(s2);
    getNext(s2);
    while(s1[i]==s2[i] && i<len)i++;
    extend[0]=i;
    int po=0;
    for(i=1;i<len;i++){
        if(Next[i-po]+i<extend[po]+po)extend[i]=Next[i-po];
        else {
            int j=extend[po]+po-i;
            if(j<0)j=0;
            while(i+j<len&&s1[j+i]==s2[j])j++;
            extend[i]=j;po=i;
        }
    }
}
```

SA!（后缀数组）

- 我终于看懂了倍增构造+简单使用...
- 但是代码莫得看懂咋写的...主要玩仨数组,sa,rank,**height**
- 简单来讲：rank数组rank[i]就是从i开始那个后缀（i~len）的字典序排名，sa数组sa[i]是排名为i的后缀开头位置是哪个
- pdf：后缀数组是“排第几的是谁？”，名次数组是“你排第几？”。容易看出，后缀数组和名次数组为互逆运算。
- 一份O(nlogn)的倍增构造代码模板：

```
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int cmp(int *r,int a,int b,int l){
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m){
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++)ws[i]=0;
    for(i=0;i<n;i++)ws[x[i]=r[i]]++;
    for(i=1;i<m;i++)ws[i]+=ws[i-1];
    for(i=n-1;i>=0;i--)sa[--ws[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p){
        for(p=0,i=n-j;i<n;i++)y[p++]=i;
        for(i=0;i<n;i++)if(sa[i]>=j)y[p++]=sa[i]-j;
        for(i=0;i<n;i++)wv[i]=x[y[i]];
        for(i=0;i<m;i++)ws[i]=0;
        for(i=0;i<n;i++)ws[wv[i]]++;
        for(i=1;i<m;i++)ws[i]+=ws[i-1];
        for(i=n-1;i>=0;i--)sa[--ws[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j) ? p-1 : p++;
    }
    return;
}
```

- important：height数组
- 定义 height[i]=suffix(sa[i-1])和 suffix(sa[i])的最长公共前缀
- **所以是height[2]开始的...**

```
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n){
    int i,j,k=0;
    for(i=1;i<=n;i++)rank[sa[i]]=i;
    for(i=0;i<n;height[rank[i++]]=k)
        for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
    return;
}
```

- 简单应用：（时刻记得height是rank相邻两个suffix（后缀）的值，height分组判定特别好用！）
  - 1.最长公共前缀：给定一个字符串，询问某两个后缀的最长公共前缀。
    - 直接RMQ对height找区间最小值。
  - 2.可重叠最长重复子串：给定一个字符串，求最长重复子串，这两个子串可以重叠。
    - 就是height最大值...
  - 3.不可重叠最长重复子串（pku1743）：给定一个字符串，求最长重复子串，这两个子串不能重叠。
    - 设k为长然后二分k按height分组（height >= k放一块，height < k的分开）判定：对于每组后缀，只须判断每个后缀的 sa 值的最大值和最小值之差是否不小于k。如果有一组满足，则说明存在，否则不存在。整个做法的时间复杂度为O(nlogn)。

一些数学

逆元 与 exgcd

- (如果a,n互质可以直接费马小定理$fast_pow(a,n-2)$)
- 费马小定理
  $A^{P-1} \equiv 1 \equiv A^0 (mod\ P)$
  $A^{P-2} \equiv \frac{1}{A} (mod\ P)$
- 欧拉定理：$a^{\phi(x)} \equiv 1 \pmod{x}$ **(x,a为正整数，且x,a互质)**
- 裴蜀定理：对任意两个整数a、b设d是它们的最大公约数。那么关于未知数x和y的线性丢番图方程（称为裴蜀等式）：$ax+by=m$
  有整数解（x,y）当且仅当m是d的倍数。裴蜀等式有解时必然有无穷多个解。

```
//ax + by = d，且|x|+|y|最小，其中d=gcd(a,b)
//即使a，b在int范围内，x和y也有可能超过int范围
void exgcd(ll a, ll b, ll &d, ll &x, ll &y)
{
    if (!b){ d = a; x = 1; y = 0;}
    else{ exgcd(b, a % b, d, y, x); y -= x * (a / b);}
}

//计算模n下a的逆。如果不存在逆，返回-1
//ax=1(mod n)
ll inv(ll a, ll n)
{
    ll d, x, y;
    exgcd(a, n, d, x, y);
    return d == 1 ? (x + n) % n : -1;
}
```

$BSGS$ 拔山盖世算法! ~ $A^x \equiv B\ (mod\ P)$

- 求解x $A^x \equiv B\ (mod\ P)$
- 小规模请直接暴力扫$0 \sim P-1$ 否则错！
- （$0 \sim P-1$）原因：
- 费马小定理：$A^x \equiv A (mod\ P) \rightarrow A^{x-1} \equiv 1 \equiv A^0 (mod\ P)$
- ps：原因好像不对

- $O(\sqrt{n}+logn)$(分块 + map)

```
typedef long long ll;
map <ll,ll> mp;
ll qp(ll a,ll k,ll p){
    ll res=1;
    ll base=a;
    while(k){
        if(k&1)res=res*base%p;
        base=base*base%p;
        k>>=1;
    }
    return res;
}

//-1 -> no solution
//小规模直接暴力 0 ~ P-1（费马小定理）
ll bsgs(ll A,ll B,ll P){
    if(A%P==0)return -1;
    mp.clear();

    ll m=ceil(sqrt(P));
    //这里最初是sqrt（p），但是如下文所述可以自己改...
    // int m = ceil(pow(p, 1.0 / 3)); 可修改，同思想可以换m大小范围（应该是），（处理大步找小步）
or （处理小步找大步）
    // 大步是pow(2/3)，所以小步的查找范围更大，但是大步枚举的次数更少

    ll tmp=B%P;
    for(ll i=0;i<=m;i++){
        mp[tmp]=i;//自己手打的hash可能不知道快多少！
        tmp=tmp*A%P;
    }
    ll t=qp(A,m,P);
    tmp=1;
    for(ll i=1;i<=m;i++){
        tmp=tmp*t%P;
        if(mp.count(tmp)){
            return ((i*m%P-mp[tmp])%P+P)%P;
        }
    }
    return -1;
}
```

comet oj瞎推$O(n)$计算所有差平方和

$$
\begin{aligned}
&\sum(w_i-w_x)^2\\
=&\sum(w_i^2-2w_iw_x+w_x^2)\\
=&\sum(w_i^2)-2w_x\sum w_i+n*w_x^2
\end{aligned}
$$

FFT（快速傅里叶变换）

- **FFT只能处理n为2的整数次幂的多项式**，不够整数次幂的全补上0 $\rightarrow 0*x^{m-1}+0*x^m+...$,长度要是$2^n$
- 加速俩多项式相乘(**卷积/大数乘法算**$a*10^0+b*10^1+c*10^2+...$,(数组也就默认了10^n，乘完进位输出就好了))
  $O(n^2)\rightarrow O(nlogn)$

DFT（离散傅里叶变换）

- 复数（complex）一个特性
- $(a_1,\theta_1)*(a_2,\theta_2)=(a_1a_2,\theta_1+\theta_2)$ 模长相乘，极角相加
- 复数域坐标系下单位圆上的点平分：

$$\omega_n^k = \cos\frac{k}{n}2\pi + i\sin\frac{k}{n}2\pi$$

$$(\omega_n^1)^k = \omega_n^k$$

- 为了不去算x_i^n这里的$O(n)$我们选ω_n^i加速DFT进行分治成为了FFT...
- ω叫**单位根**了,特定性质:

1.$\omega_n^k = \omega_{2n}^{2k}$ $(\omega_n^k = \cos\frac{k}{n}2\pi + i\sin\frac{k}{n}2\pi = \cos\frac{2k}{2n}2\pi + i\sin\frac{2k}{2n}2\pi = \omega_{2n}^{2k})$

2.$\omega_n^{k+\frac{n}{2}} = -\omega_n^k$ $(\omega_n^{\frac{n}{2}} = \cos\frac{\frac{n}{2}}{n}2\pi + i\sin\frac{\frac{n}{2}}{n}2\pi = cos\pi + isin\pi = -1, e^{ix} = cosx + isinx, e^{i\pi} + 1 = 0)$

3.$\omega_n^0 = \omega_n^n = 1 = 1 + 0i$

FFT推导

- 将原本的DFT**递归分治**处理，推导如下:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1}x^{n-1}$$

按下标的奇偶性分割A(x)

$$A(x) = (a_0 + a_2 x^2 + \ldots + a_{n-2}x^{n-2}) + (a_1 x + a_3 x^3 + \ldots + a_{n-1}x^{n-1})$$

$$= (a_0 + a_2 x^2 + \ldots + a_{n-2}x^{n-2}) + x(a_1 + a_3 x^2 + \ldots + a_{n-2}x^{n-1})$$

由两者相似设:

$$A_1(x) = a_0 + a_2 x + a_4 x^2 \ldots + a_{n-2}x^{\frac{n}{2}-1}$$

$$A_2(x) = a_1 + a_3 x + a_5 x^2 \ldots + a_{n-1}x^{\frac{n}{2}-1}$$

$$A(x) = A_1(x^2) + xA_2(x^2)$$

再设$k < \frac{n}{2}$,将ω_n^k带入$A(x)$

$$A(\omega_n^k) = A_1((\omega_n^k)^2) + \omega_n^k A_2((\omega_n^k)^2)$$

$$= A_1(\omega_n^{2k}) + \omega_n^k A_2(\omega_n^{2k})$$

$$= A_1(\omega_{\frac{n}{2}}^k) + \omega_n^k A_2(\omega_{\frac{n}{2}}^k) \quad (\rightarrow \omega_n^k = \omega_{2n}^{2k})$$

再考虑带入$\omega_n^{k+\frac{n}{2}}$

$$A(\omega_n^{k+\frac{n}{2}}) = A_1((\omega_n^{k+\frac{n}{2}})^2) + \omega_n^{k+\frac{n}{2}} A_2((\omega_n^{k+\frac{n}{2}})^2)$$

$$= A_1(\omega_n^{2k+n}) + \omega_n^{k+\frac{n}{2}} A_2(\omega_n^{2k+n})$$

$$= A_1(\omega_n^{2k}\omega_n^n) - \omega_n^k A_2(\omega_n^{2k}\omega_n^n) \quad (\rightarrow \omega_n^{k+\frac{n}{2}} = -\omega_n^k)$$

$$= A_1(\omega_n^{2k}) - \omega_n^k A_2(\omega_n^{2k}) \quad (\rightarrow \omega_n^0 = \omega_n^n = 1 = 1 + 0i)$$

$$= A_1(\omega_{\frac{n}{2}}^k) - \omega_n^k A_2(\omega_{\frac{n}{2}}^k) \quad (\rightarrow \omega_n^k = \omega_{2n}^{2k})$$

发现仅有符号为+/-的差距，因此通过$A_1(\omega_{\frac{n}{2}}^k)$和$A_2(\omega_{\frac{n}{2}}^k)$的值可以得到$A(\omega_n^k)$和$A(\omega_n^{k+\frac{n}{2}})$的值（看好是A函数不是A1A2...$n/2 \rightarrow mid$)

IFFT（快速傅里叶逆变换）

- **一个多项式在分治的过程中乘上单位根的共轭复数，分治完的每一项除以n即为原多项式的每一项系数**
- 就是还是FFT函数就打个标记flag表示是该搞共轭（IFFT）还是原版的（FFT）

all代码

```
/*超朴素的递归版本FFT，容易爆。。*/
#include<complex>
#include<cmath>
#define cp complex<double>

const int maxn=6e4+50;
const double pi=acos(-1);
void FFT(cp *a,int n,int inv){//inv -> 取共轭 or not,a[]=>[0,n)
    if(n==1)return;
    int mid=n/2;
    static cp b[maxn];
    for(int i=0;i<=mid-1;i++)b[i]=a[i*2],b[i+mid]=a[i*2+1];//真实的拆分
    for(int i=0;i<=n-1;i++)a[i]=b[i];//真实再还原
    FFT(a,mid,inv);
    FFT(a+mid,mid,inv);
    for(int i=0;i<=mid-1;i++){
        cp x(cos(2*pi*i/n),inv*sin(2*pi*i/n));
        b[i]=a[i]+x*a[i+mid];
        b[i+mid]=a[i]-x*a[i+mid];
    }
    for(int i=0;i<=n-1;i++)a[i]=b[i];
}
cp ai[maxn],bi[maxn];

/*全新加入蝴蝶变化预处理的迭代版本FFT，求高精乘法*/
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
const int maxn=150000+50;
const double pi=acos(-1);

struct node{
    double x,y;
    node(double xx=0,double yy=0){x=xx;y=yy;}
    node operator * (const node &r)const{return node(x*r.x-y*r.y,x*r.y+y*r.x);}
    node operator + (const node &r)const{return node(x+r.x,y+r.y);}
    node operator - (const node &r)const{return node(x-r.x,y-r.y);}
}ai[maxn],bi[maxn];

int R[maxn];

void FFT(node *J,int len,double flag){
    for(int i=0;i<len;i++){
        if(i<R[i])swap(J[i],J[R[i]]);
    }
    for(int slen=1;slen<len;slen<<=1){
        node T(cos(pi/slen),flag*sin(pi/slen));
        for(int lcur=0;lcur<len;lcur+=(slen<<1)){
            node t(1,0);
            for(int i=0;i<slen;i++,t=t*T){
                node Nx=J[lcur+i],Ny=t*J[lcur+slen+i];
                J[lcur+i]=Nx+Ny;
                J[lcur+slen+i]=Nx-Ny;
            }
        }
    }
}
```

```
char s1[maxn],s2[maxn];
int ans[maxn];
int main()
{
    int n;
    scanf("%d",&n);
    scanf("%s",s1);
    scanf("%s",s2);
    int t=0;
    for(int i=n-1;i>=0;i--)ai[t++].x=s1[i]-'0';t=0;
    for(int i=n-1;i>=0;i--)bi[t++].x=s2[i]-'0';

    int lim=1;
    while(lim<n+n)lim<<=1;
    int bit=0;
    while((1<<bit)<lim)bit++;
    for(int i=0;i<lim;i++)R[i]=(R[i>>1]>>1)|((i&1)<<(bit-1));

    FFT(ai,lim,1);
    FFT(bi,lim,1);
    for(int i=0;i<lim;i++)ai[i]=ai[i]*bi[i];
    FFT(ai,lim,-1);

    for(int i=0;i<=lim;i++){
        ans[i]+=(int)(ai[i].x/lim+0.5);
        if(ans[i]>=10){
            ans[i+1]+=ans[i]/10;
            ans[i]%=10;
            if(i==lim)lim++;
        }
    }

    while(ans[lim]==0&&lim>=1)lim--;
    for(int i=lim;i>=0;i--)printf("%d",ans[i]);
    puts("");
    return 0;
}
```

NTT前置——群+数论の知识

群

- 实际上群出现好多次了。。。

定义（群）设G为某种元素组成的一个非空集合，若在G内定义一个称为乘法的运算“·”，满足以下条件：
（1）（封闭性）$\forall a,b\in G$有$a\cdot b\in G$；

（2）（结合性）$\forall a,b,c\in G$，有$a\cdot(b\cdot c)=(a\cdot b)\cdot c$；

（3）在G中有一个元素e，对G中任意元素g，有e·g=g·e=g，元素e称为**单位元**(好像也叫**幺元/零元**来着，$-g$叫**负元**)；

（4）对G中任一元素g都存在G中的一个元素g'，使得g·g'=g'·g=e，g称为可逆元，g'称为g的**逆元**，记作g^{-1}；

则称**G关于“·”形成一个群（Group）**，记作（G，·），通常在不混淆的情况下省略“·”，用G来表示一个群，a·b也简记为ab

- **WARNING**：运算均是抽象运算，并不只是我们熟知的数之间的乘法和加法。非空集合G中的元素形式多样，可以是数、集合、矩阵等研究对象。
- 额外扩展定义：
  - **Abel群或交换群**：群中的运算满足交换律的群

- **半群**：非空集合G满足条件（1）和（2），则称G为半群
- **含幺半群**：非空集合G满足条件（1），（2）和（3），则称G为含幺半群
- **阶**：群G中元素的个数称为G的阶，记作$|G|$，若$|G|<+\infty$，则称群G为有限群；若$|G|>+\infty$，则称群G为无限群。(对群大小的*阶*)
- **子群**扩展包：
  - 设H为群G的**一**个非空子集，若H对G的运算“·”也构成群，**则称H为群G的子群**，记作H≤G
  - **平凡子群**：由子群的定义知，H={e}和H=G都是群G的子群，称之为群G的平凡子群。
  - **真子群**：如果H不是群G的平凡子群，即$e \subset H \subset G$，则称H为群G的真子群
  - **循环群**：群G由**一**个元素a生成时，称为循环群（Cyclic Group），记为G=<a>。
    - **循环群特性**：（1）循环群是Abel群，（2）若G=<a>,则|G|=ord(a)。
  - **生成元**：群中元素可以由最小数目个群元的**乘积生成**，这组群元称为该群的生成元，生成元的数目为有限群的秩。
- **定义/ord/阶（元素）**：设G为群，$a \in G$，则使$a^n = e$成立的最小正整数n称为元素a的阶，记作**ord(a)**。如果不存在这样的正整数n，则称元素a为无限阶元。（类似数论中的阶）

数论部分

- **原根**：
  - 在gcd(a,m)=1时，定义 a对模m的指数 δm(a)为使 $a^d \equiv 1(\mod m)$ 成立的最小的正整数 d。可知 δm(a) **一**定小于等于 φ (m)，若δm(a) = φ(m)，则称**a是模m的原根。**
  - **原根**另**一**种定义：假设**一**个数g对于P来说是原根，那么$g^i \mod P$的结果两两不同,且有$1<g<P, 0<i<P$，那么g可以称为是P的**一**个原根
    - 原根例： 由于$3^1 \equiv 3(\mod 7)$，$3^2 \equiv 2(\mod 7)$，$3^3 \equiv 6(\mod 7)$，$3^4 \equiv 4(\mod 7)$，$3^5 \equiv 5(\mod 7)$，$3^6 \equiv 1(\mod 7)$，所以 3 是模 7 的**一**个原根。
  - 性质：如果g是P的原根，**那么g的（1...P-1）次幂mod P的结果一定互不相同。**
  - 【求原根方法】
    对数P−1分解质因数得到不同的质因子$p_1, p_2, p_3, \ldots, p_n$，对于任何$2 \le a \le P-1$，判定a是否为P的原根，只需要检验$a^{\frac{P-1}{p_1}}, a^{\frac{P-1}{p_2}}, \ldots, a^{\frac{P-1}{p_n}}$ 这n个数中，**是否存在一个数mod P为1**，若存在，则a不是P的原根，否则a是P的原根。
  - 原根**一**般都是很小的数(100以内)。

同余式的性质

- (1)同余式可以**逐项相加。**
  若$a_1 \equiv b_1 \pmod m, a_2 \equiv b_2 \pmod m, \ldots, a_n \equiv b_n \pmod m$
  则$a_1 + a_2 + \ldots + a_n \equiv b_1 + b_2 + \ldots + b_n \pmod m$
- (2)同余式**一**边的数可以**移到另一边，只要改变符号**就可以了。
  若$a + b \equiv c \pmod m$，则$a \equiv c - b \pmod m$
- (3)同余式的**每一边都可以增加或减去模的任意倍数。**
  若$a \equiv b \pmod m$，则$a \mp km \equiv b \pmod m$
- (4)同余式可以**逐项相乘。**
  若$a_1 \equiv b_1 \pmod m, a_2 \equiv b_2 \pmod m, \ldots, a_n \equiv b_n \pmod m$
  则$a_1 a_2 \ldots a_n \equiv b_1 b_2 \ldots b_n \pmod m$
- (5)我们可以将**性质(1)(4)推广**成以下的情形。
  $(1) \rightarrow$ 若$A \equiv B \pmod m, x_1 \equiv y_1 \pmod m, x_2 \equiv y_2 \pmod m, \ldots, x_k \equiv y_k \pmod m$
  则$\sum A x_1^{a_1} x_2^{a_2} \ldots x_k^{a_k} \equiv \sum B y_1^{a_1} y_2^{a_2} \ldots y_k^{a_k} \pmod m$
  $(4) \rightarrow$ 若$a_0 \equiv b_0 \pmod m, a_1 \equiv b_1 \pmod m, \ldots, a_n \equiv b_n \pmod m$
  则$a_0 x^n + a_1 x^{n-1} + \ldots + a_n \equiv (b_0 x^n + b_1 x^{n-1} + \ldots + b_n) \pmod m$

- (6)同余式**两边的数如有公约数**，此**公约数又和模互素**，那么就**可以把两边的数除以这个公约数。**
  若$a \equiv b \pmod{m}$，且$a = a_1 d, b = b_1 d, gcd(d, m) = 1$
  则$a_1 \equiv b_1 \pmod{m}$
- (7)同余式**两边的数和模**可以**同时乘上一个整数。**
  若$a \equiv b \pmod{m}$，
  则$ak \equiv bk \pmod{km}$
- (8)同余式**两边的数和模**可以**同时被它们任一公约数除。**
  若$a \equiv b \pmod{m}$，且$a = a_1 d, b = b_1 d, m = m_1 d$
  则$a_1 \equiv b_1 \pmod{m_1}$
- (9)如果同余式**对于模m成立**，那么它**对于m的任意约数相等的模d也成立。**
  若$a \equiv b \pmod{m}$，$m = m_1 d$
  则$a \equiv b \pmod{d}$
- (10)如果同余式**一边上的数和模能被某个数除尽**，则同余式的**另一边的数也能被这个数除尽。**
  若$a \equiv b \pmod{m}$，$k|a, k|m$
  则$k|b$
- (11)同余式**一边上的数与模的最大公约数，等于另一边上的数与模的最大公约数。**
  若$a \equiv b \pmod{m}$，
  则$gcd(a, m) = gcd(b, m)$
- 不大相同的简略版：

1.**反身性**：a≡a (mod m)；
2.**对称性**：若a≡b(mod m)，则b≡a (mod m)；
3.**传递性**：若a≡b(mod m)，b≡c(mod m)，则a≡c(mod m)；
4.**同余式相加**：若a≡b(mod m)，c≡d(mod m)，则a c≡b d(mod m)；
5.**同余式相乘**：若a≡b(mod m)，c≡d(mod m)，则ac≡bd(mod m)。
6.**线性运算**：如果a ≡ b (mod m)，c ≡ d (mod m)，那么
(1)a ± c ≡ b ± d (mod m)；
(2)a * c ≡ b * d (mod m)。
7.**除法**：若$ac \equiv bc \pmod{m}$，$c \neq 0$,则$a \equiv b \pmod{m/gcd(c, m)}$，其中gcd(c,m)表示c和m的最大公约数，特殊地$gcd(c, m) = 1$则$a \equiv b \pmod{m}$；
8.**幂运算**：如果$a \equiv b \pmod{m}$，那么$a^n \equiv b^n \pmod{m}$；
9.若$a \equiv b \pmod{m}$，$n = m$，则$a \equiv b \pmod{n}$；
10.若$a \equiv b \pmod{m_i}$，$(i = 1, 2...n)$则，$a \equiv b \pmod{[m_1, m_2, \ldots, m_n]}$其中$[m_1, m_2, \ldots m_n]$表示$m_1, m_2, \ldots m_n$的最小公倍数(lcm)。

NTT

- 迷之信息：其实这些模数的原根通常都是3
- 通常模数常见的有998244353,1004535809,469762049，这几个的原根都是3
- 大概就是...原根换单位根
- 要找到整数中模意义下的ω_n^k来加速DFT，这样没有精度误差、复数之类的东西了
  - 我们可以发现原根g拥有所有FFT所需的ω的性质，于是如果我们用$g^{\frac{P-1}{N}} \pmod{P}$来代替$\omega_n = e^{\frac{2\pi i}{N}}$，就能把复数对应成一个整数，在$\pmod{P}$意义下做快速变换了。
  - 可以证明：$\omega_n^k = g^{\frac{k \times (P-1)}{n}}$
- 单位根性质：
  1.$\omega_n^0, \ldots \omega_n^{n-1}$互不相同

  2.$\omega_n^0 = \omega_n^n = 1$

  3.$\omega_n^{k+\frac{n}{2}} = -\omega_n^k$，$\omega_{2n}^{2k} = \omega_n^k$

  4.$\omega_n^a \omega_n^b = \omega_n^{a+b}$

  5.$\sum_{i=1}^{n-1} \omega_n^i = 0 \quad (IDFT/IFFT$变化证明$)$
  - 证明原根具有相同性质：
    1.当$n \leq P - 1$时，根据定义可知$g_n^0, \ldots, g_n^{n-1}$互不相同，条件1成立。

2.根据欧拉定理：$g_n^0=1, g_n^n=g^{P-1}, g^{P-1}\equiv 1 \pmod P$。P为质数，这里的$\phi(P)=P-1$，条件2成立。

3.由上$g_n^n\equiv 1 \pmod P$，由**同余幂运算**，$g_n^{\frac{n}{2}} \pmod P$只能为1或-1，又因为$g_n^0,\dots,g_n^{n-1}$互不相同，得$g_n^{\frac{n}{2}}\equiv -1 \pmod P$，条件3成立。

4.带入显然幂运算指数相加+乘法分配律就是这里俩g_n乘一块...条件4成立。

5.由定义，$\sum_{i=1}^{n-1} g_n^i=\frac{g_n^1(1-g_n^1)}{1-g_n^1}=\frac{1-g_n^n}{1-g_n^1}=0$

- **所以只要替换$\omega_n^k=g^{\frac{k\times(P-1)}{n}}$就是NTT了！**
- 要求P是素数且N必须是$P-1$的因子，因为N是2^n，一般构造出P是$P=k*2^m+1$使$N|(P-1)$
- $1004535809=479*2^{21}+1 \quad g=3$
  $998244353=119*2^{23}+1 \quad g=3$

INTT（应该是逆变换吧...）

- 前边找的是$\omega_n^{-1},\omega_n^{-2},\dots,(\omega_n^{-1})^j$，是共轭复数。
- 那么这里就是$g_n^{-1},g_n^{-2}...$所以这是逆元喔
- **最后那里记得除以n变成乘以逆元**，第一份代码在NTT内完成了乘以逆元，注意n是长度（2^n）那个，不是简单的n...

修改的楼上FFT变NTT代码

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
const int maxn=150000+50;
const long long P=998244353;
const long long g=3;
//这个变换只是借用P，g这样的数完成的多项式乘法，所以P，g构造使用就好没啥特殊，三模数NTT见
long long qp(long long a,long long k){
    long long res=1,base=a;
    while(k){
        if(k&1)res=res*base%P;
        base=base*base%P;
        k>>=1;
    }
    return res;
}

long long ai[maxn],bi[maxn];
int R[maxn];

void NTT(long long *J,int len,int flag){
    for(int i=0;i<len;i++){
        if(i<R[i])swap(J[i],J[R[i]]);
    }
    for(int slen=1;slen<len;slen<<=1){
        //node T(cos(pi/slen),flag*sin(pi/slen));
        long long T=qp(g,(P-1)/(slen<<1));
        for(int lcur=0;lcur<len;lcur+=(slen<<1)){
            long long t=1;
            for(int i=0;i<slen;i++,t=t*T%P){
                long long Nx=J[lcur+i],Ny=t*J[lcur+slen+i]%P;
                J[lcur+i]=(Nx+Ny)%P;
                J[lcur+slen+i]=(Nx-Ny+P)%P;
            }
        }
    }
```

```
    if(flag==1)return;
    long long inv=qp(len,P-2);reverse(J+1,J+len);
    for(int i=0;i<len;i++)J[i]=J[i]*inv%P;//不是很懂reverse
}

char s1[maxn],s2[maxn];
int ans[maxn];
int main()
{
    int n;
    scanf("%d",&n);
    scanf("%s",s1);
    scanf("%s",s2);
    int t=0;
    for(int i=n-1;i>=0;i--)ai[t++]=s1[i]-'0';t=0;
    for(int i=n-1;i>=0;i--)bi[t++]=s2[i]-'0';

    int lim=1;
    while(lim<n+n)lim<<=1;
    int bit=0;
    while((1<<bit)<lim)bit++;
    for(int i=0;i<lim;i++)R[i]=(R[i>>1]>>1)|((i&1)<<(bit-1));

    NTT(ai,lim,1);
    NTT(bi,lim,1);
    for(int i=0;i<lim;i++)ai[i]=ai[i]*bi[i];
    NTT(ai,lim,-1);

    for(int i=0;i<=lim;i++){
        ans[i]+=ai[i];
        if(ans[i]>=10){
            ans[i+1]+=ans[i]/10;
            ans[i]%=10;
            if(i==lim)lim++;
        }
    }

    while(ans[lim]==0&&lim>=1)lim--;
    for(int i=lim;i>=0;i--)printf("%d",ans[i]);
    puts("");
    return 0;
}

//我看的懂的自己写懂得NTT+INTT
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
const int maxn=150000+50;
const long long P=998244353;
const long long g=3;

long long qp(long long a,long long k){
    long long res=1,base=a;
    while(k){
        if(k&1)res=res*base%P;
        base=base*base%P;
        k>>=1;
    }
    return res;
}
```

```
long long ai[maxn],bi[maxn];
int R[maxn];

void NTT(long long *J,int len,int flag){
    for(int i=0;i<len;i++){
        if(i<R[i])swap(J[i],J[R[i]]);
    }
    for(int slen=1;slen<len;slen<<=1){
        //node T(cos(pi/slen),flag*sin(pi/slen));
        long long T=qp(flag==1 ? g : qp(g,P-2),(P-1)/(slen<<1));
        for(int lcur=0;lcur<len;lcur+=(slen<<1)){
            long long t=1;
            for(int i=0;i<slen;i++,t=t*T%P){
                long long Nx=J[lcur+i],Ny=t*J[lcur+slen+i]%P;
                J[lcur+i]=(Nx+Ny)%P;
                J[lcur+slen+i]=(Nx-Ny+P)%P;
            }
        }
    }
}

char s1[maxn],s2[maxn];
int ans[maxn];
int main()
{
    int n;
    scanf("%d",&n);
    scanf("%s",s1);
    scanf("%s",s2);
    int t=0;
    for(int i=n-1;i>=0;i--)ai[t++]=s1[i]-'0';t=0;
    for(int i=n-1;i>=0;i--)bi[t++]=s2[i]-'0';

    int lim=1;
    while(lim<n+n)lim<<=1;
    int bit=0;
    while((1<<bit)<lim)bit++;
    for(int i=0;i<lim;i++)R[i]=(R[i>>1]>>1)|((i&1)<<(bit-1));

    NTT(ai,lim,1);
    NTT(bi,lim,1);
    for(int i=0;i<lim;i++)ai[i]=ai[i]*bi[i];
    NTT(ai,lim,-1);

    long long invlim=qp(lim,P-2);
    for(int i=0;i<=lim;i++){
        ans[i]+=(ai[i]*invlim)%P;//除以n变乘逆元
        if(ans[i]>=10){
            ans[i+1]+=ans[i]/10;
            ans[i]%=10;
            if(i==lim)lim++;
        }
    }

    while(ans[lim]==0&&lim>=1)lim--;
    for(int i=lim;i>=0;i--)printf("%d",ans[i]);
    puts("");
    return 0;
}
```

多项式全套操作（加减乘除+求逆+牛顿迭代+带余除法+Ln+Exp+快速幂）

- ‘+’，‘-’自己瞎写吧....
- ‘*’，在上边那个NTT/FFT都行...

多项式除法 & 求逆

- 类比同余系下数字的逆元,只要求多项式的逆元,就可以把除法转换成乘法了。
- 倍增求逆：...我放弃...

MTT & 分治FFT

BM

FWT

莫比乌斯反演/狄利克雷卷积

杜教筛

二次剩余

Lucas定理 & EXLucas

CRT & EXCRT

- 处理同余方程一片，求x

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
.
.
.$$x \equiv a_n \pmod{m_n}$$

- CRT(中国剩余定理)：处理楼上那个问题，前置是那堆**m俩俩互质**
- 处理方法是搞**所有m的lcm**，
  然后对一个式子算$\frac{lcm}{m_i}t_i \equiv 1 \pmod{m_i}$这里$t_i$用**exgcd**算得
  这个式子两边同时乘上a_i，左边就是**一个x的解值**啦，类似：$x_i \equiv a_i \frac{lcm}{m_i}t_i \equiv a_i \pmod{m_i}$
  然后把所有这样得到的x**加和**就好啦!(因为其他的lcm同余全是0)
- 也就是：$x = \sum_{i=1}^{k} a_i \frac{lcm}{m_i}t_i, (t_i \rightarrow exgcd)$
- 板子crt：**看清是哪个余数哪个模数**

```
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
typedef long long ll;
const int maxn=10+10;
ll a[maxn],b[maxn];
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    ll ret=exgcd(b,a%b,y,x);
    y-=(a/b)*x;
    return ret;
}
ll mul(ll a,ll b,ll mod)
{
    ll res=0;
    while(b>0)
```

```
    {
        if(b&1) res=(res+a)%mod;
        a=(a+a)%mod;
        b>>=1;
    }
    return res;
}
int k;
ll crt(){
    ll x,y,res=0,lcm=1,tmp;
    for(int i=1;i<=k;i++)lcm*=b[i];
    for(int i=1;i<=k;i++){
        tmp=lcm/b[i];
        exgcd(b[i],tmp,x,y);
        y=(y%b[i]+b[i])%b[i];
        res=(res+mul(mul(y,tmp,lcm),(a[i]%lcm+lcm)%lcm,lcm))%lcm;
    }//a可能负数
    return (lcm+res)%lcm;
}
int main()
{
    scanf("%d",&k);
    for(int i=1;i<=k;i++)scanf("%lld",&a[i]);//余数a
    for(int i=1;i<=k;i++)scanf("%lld",&b[i]);//模数b
    printf("%lld\n",crt());
    return 0;
}
```

- exCRT(扩展版本)：这次**不用m俩俩互质**啦！（指不定啥关系）
- 板子excrt：
  处理那堆m不互质的，return -1就是无解，一个一个exgcd往上怼就好了hhhhhh
- 解决ing：**只考虑两个**怎么处理?
  可以得到：$x = a_1 + k_1 * m_1; x = a_2 + k_2 * m_2$
  所以 $a_1 + k_1 * m_1 = a_2 + k_2 * m_2$
  $k_2 * m_2 - k_1 * m_1 = a_1 - a_2 \quad \rightarrow exgcd解k，反推x$
  注意细节上**爆longlong和模负数、(+mod)%mod**之类的事情，二合一再继续上边的操作
  看清是哪个余数哪个模数

```
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
typedef long long ll;
//太沙雕的出题人可以考虑上边用下边
//using ll=__int128;
const int maxn=100+10;
ll a[maxn],r[maxn];
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(b==0){
        x=1,y=0;
        return a;
    }
    ll ret=exgcd(b,a%b,y,x);
    y-=(a/b)*x;
    return ret;
}
int n;
ll mul(ll a,ll b,ll mod)
{
    ll res=0;
    while(b>0)
```

```
    {
        if(b&1) res=(res+a)%mod;
        a=(a+a)%mod;
        b>>=1;
    }
    return res;
}
ll excrt(){
    ll M=a[1],R=r[1],x,y,d;//模a,余数r
    for(int i=2;i<=n;i++){
        d=exgcd(M,a[i],x,y);
        ll tmp=((r[i]-R%a[i]+a[i])%a[i]);
        if(tmp%d) return -1;
        //x=(R-r[i])/d*x%a[i];
        tmp/=d;
        x=mul(x,tmp,a[i]);
        R+=M*x;
        M=a[i]/d*M;
        R=(R%M+M)%M;
    }
    return (R%M+M)%M;
}
int main()
{
    ll m;
    scanf("%d%lld",&n,&m);
    for(int i=1;i<=n;i++)scanf("%lld%lld",&a[i],&r[i]);//模a,余数r
    long long ans=excrt();
    if(ans==-1)puts("he was definitely lying");
    else if(ans>m)puts("he was probably lying");
    else printf("%lld\n",ans);
    return 0;
}
```

上边那个**容易炸long long*long long**
所以写了**快速乘取模mul**
但是日常还是炸，建议__**int128**
或者下边的python3/java不会qwq

```
def egcd(a, b):
    """扩展欧几里得"""
    if 0 == b:
        return 1, 0, a
    x, y, q = egcd(b, a % b)
    x, y = y, (x - a // b * y)
    return x, y, q


s = input()
n, m = int(s.split()[0]), int(s.split()[1])
# print(n, m)
M = 0
R = 0
x = 0
y = 0
d = 0
ok = 1
for i in range(0, n):
    if i == 0:
        s = input()
        M, R = int(s.split()[0]), int(s.split()[1])
    else:
```

```
        s = input()
        ui, ri = int(s.split()[0]), int(s.split()[1])
        x, y, d = egcd(M, ui)
        if (R - ri) % d != 0:
            ok = 0
        x = (R - ri) // d * x % ui
        R -= M * x
        M = ui // d * M
        R = R % M

if ok == 1:
    if R > m:
        print("he was probably lying")
    else:
        print(R)
else:
    print("he was definitely lying")
```

拉格朗日插值法vs快速插值法

因式定理 & 余数定理

- **因式定理**:
  “设f(x)为一多项式，则x-a为f(x)的因式”等价于f(a)=0。
  (大概就是感觉能拆个(x-a)出来变成(x-a)*f'(x)=f(x)然后(x-a)=0)
- 推广:
  “ax-b为f(x)的因式”等价于$f(\frac{b}{a})=0$。
- **余数定理**:
  当一个多项式f(x)除以(x – a)时，所得的余数等于 f(a)
  整系数多项式f(x)除以(x-a)商为q(x)，余式为r，则$f(x)=(x-a)q(x)+r.$
  如果多项式r=0，那么多项式f(x)必定含有因式(x-a)。反过来，如果f(x)含有因式(x-a)，那么，r=0。
- 余式定理的推论
  当一个多项式 f(x) 除以 (mx – n) 时，所得的余数等于 f(n/m)

拉格朗日插值法

- 解决的问题:给你 n 个不同的点值(x_i,y_i),让你求出一个低于 n 次的多项式满足经过上述 n 个点。【就是点值转系数表示...】
- FFT/NTT用了单位根/原根，这里可不是Ixxx...（逆xxx..）
- 拉格朗日的想法:
  1.对每个(x_i,y_i)构造$f_i(x_i)=y_i,f_i(x_k)=0\ [k\neq i]$,然后把他们加起来就好啦~
  2.先构造第二块都是零的：$G=\prod_{k=1}^{n}[k\neq i](x-x_k)$
  3.再来构造一波第一个$f(x_i)=y_i$：把x_i带入G可得到$G(x_i)=\prod_{k=1}^{n}[k\neq i](x_i-x_k)$就是常数了
  4.综合2&3得到可以设$f_i(x)=\frac{y_iG}{G(x_i)},(满足\ f_i(x_i)=y_i,f_i(x_k)=0[k\neq i])$
  5.再通过4来综合$O(n^2)$时间转换：$F=\sum_{i=1}^{n}(y_i\prod_{k=1,k\neq i}^{n}\frac{x-x_k}{x_i-x_k})$

```
//给定n个点，请你确定这个多项式，并将k代入求值
//求出的值对998244353取模
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
const int maxn=2000+5;
const long long mod=998244353;
struct node{
    long long x,y;
}nds[maxn];
long long qp(long long a,long long k){
```

```
    long long res=1,base=a;
    while(k){
        if(k&1)res=res*base%mod;
        base=base*base%mod;
        k>>=1;
    }
    return res;
}
int main()
{
    int n;
    long long k;
    scanf("%d%lld",&n,&k);
    long long ans=0;
    for(int i=1;i<=n;i++){
        scanf("%lld%lld",&nds[i].x,&nds[i].y);
    }
    for(int i=1;i<=n;i++){
        long long tans=nds[i].y;
        for(int j=1;j<=n;j++){
            if(i==j)continue;
            tans=((tans*(k-nds[j].x+mod)%mod)*qp(nds[i].x-nds[j].x,mod-2)+mod)%mod;
        }
        ans=(ans+tans+mod)%mod;
    }
    printf("%lld\n",ans);
    return 0;
}
```

简单的高斯消元板板

- 复杂度$O(n^3)$

```
const double eps=1e-8;
double A[maxn][maxn];
double B[maxn];
double ans[maxn];
//A*ans=B
void Gauss(){
    int num;
    for(int i=1;i<n;i++){//i<n!!!! 没有 =
        num=i;
        for(int j=i+1;j<=n;j++){
            if(abs(A[j][i])>abs(A[num][i]))num=j;
        }//暴力找最小

        for(int j=1;j<=n;j++)swap(A[i][j],A[num][j]);
        swap(B[i],B[num]);//暴力交换

        for(int j=i+1;j<=n;j++){
            if(abs(A[j][i])<=eps)continue;
            double t=A[j][i]/A[i][i];// A[j][i] !!!
            for(int k=1;k<=n;k++){
                A[j][k]-=A[i][k]*t;
            }
            B[j]-=B[i]*t;
        }//暴力消到0/上三角矩阵
    }

    for(int i=n;i>=1;i--){
        ans[i]=B[i]/A[i][i];
        for(int j=i;j>=1;j--){
```

```
            B[j]-=A[j][i]*ans[i];
        }
    }//反向向上得到解
    return;
}
```

模意义下的高斯消元板板（和最前边的逆元联动）

- 注意得有逆元可取...

```
const long long md=1e9+7;
long long A[maxn][maxn];
long long B[maxn];
//double ans[maxn];
long long p[maxn];//这个p就是ans啦
//A*ans=B
int k;
using ll=long long;
void exgcd(ll a, ll b, ll &d, ll &x, ll &y)
{
    if (!b){ d = a; x = 1; y = 0;}
    else{ exgcd(b, a % b, d, y, x); y -= x * (a / b);}
}

//ax=1(mod n)
ll inv(ll a, ll n)
{
    ll d, x, y;
    exgcd(a, n, d, x, y);
    return d == 1 ? (x + n) % n : -1;
}//前方联动inv
void Gauss(){
    int num;
    int n=k;
    for(int i=1;i<n;i++){//i<n!!!! 没有 =
        num=i;
        for(int j=i+1;j<=n;j++){
            if(abs(A[j][i])>abs(A[num][i]))num=j;
        }//暴力找最小

        for(int j=1;j<=n;j++)swap(A[i][j],A[num][j]);
        swap(B[i],B[num]);//暴力交换

        for(int j=i+1;j<=n;j++){
            //if(abs(A[j][i])<=eps)continue;
            if(A[j][i]==0)continue;
            //double t=A[j][i]/A[i][i];// A[j][i] !!!
            long long t=(A[j][i]*inv(A[i][i],md)%md+md)%md;
            for(int k=1;k<=n;k++){
                A[j][k]=(A[j][k]-A[i][k]*t%md+md)%md;
            }
            B[j]=(B[j]-B[i]*t%md+md)%md;
        }//暴力消到0/上三角矩阵
    }

    for(int i=n;i>=1;i--){
        //ans[i]=B[i]/A[i][i];
        p[i]=(B[i]*inv(A[i][i],md)%md+md)%md;
        for(int j=i;j>=1;j--){
            B[j]=(B[j]-A[j][i]*p[i]%md+md)%md;
        }
    }//反向向上得到解
```

```
    return;
}
```

一些数据结构

线段树各种瞎写

```
//这里是最大/最小连续字段和，查询返回node
struct ST{
    #define ls i<<1
    #define rs i<<1|1

    struct node{
        int l,r;
        long long sum;
        long long lmxs,rmxs;//,maxsum;
        long long lmis,rmis;//,minsum;
    }T[maxn<<2],ans;

    void pushup(int i){
        T[i].sum=T[ls].sum+T[rs].sum;

        //T[i].maxsum=max(T[ls].rmxs+T[rs].lmxs,max(T[ls].maxsum,T[rs].maxsum));
        T[i].lmxs=max(T[ls].sum+T[rs].lmxs,T[ls].lmxs);
        T[i].rmxs=max(T[rs].sum+T[ls].rmxs,T[rs].rmxs);

        //T[i].minsum=min(T[ls].rmis+T[rs].lmis,min(T[ls].minsum,T[rs].minsum));
        T[i].lmis=min(T[ls].sum+T[rs].lmis,T[ls].lmis);
        T[i].rmis=min(T[rs].sum+T[ls].rmis,T[rs].rmis);
    }

    void build(int i,int l,int r){
        T[i].l=l;T[i].r=r;
        if(l==r){
            T[i].lmis=T[i].rmis=b[l];//T[i].minsum=b[l];
            T[i].lmxs=T[i].rmxs=b[l];//T[i].maxsum=b[l];
            T[i].sum=b[l];
            return;
        }
        int mid=(l+r)>>1;
        build(ls,l,mid);
        build(rs,mid+1,r);
        pushup(i);
    }

    node query_max(int i,int nl,int nr){
        if(nl<=T[i].l&&T[i].r<=nr)return T[i];
        int mid=(T[i].l+T[i].r)>>1;
        if(nr<=mid)return query_max(ls,nl,nr);
        if(mid<nl)return query_max(rs,nl,nr);
        node lo=query_max(ls,nl,nr),ro=query_max(rs,nl,nr),ans;

        ans.sum=lo.sum+ro.sum;
        //ans.maxsum=max(lo.rmxs+ro.lmxs,max(lo.maxsum,ro.maxsum));
        ans.lmxs=max(lo.sum+ro.lmxs,lo.lmxs);
        ans.rmxs=max(ro.sum+lo.rmxs,ro.rmxs);
        return ans;
    }

    node query_min(int i,int nl,int nr){
        if(nl<=T[i].l&&T[i].r<=nr)return T[i];
```

```
        int mid=(T[i].l+T[i].r)>>1;
        if(nr<=mid)return query_min(ls,nl,nr);
        if(mid<nl)return query_min(rs,nl,nr);
        node lo=query_min(ls,nl,nr),ro=query_min(rs,nl,nr),ans;

        ans.sum=lo.sum+ro.sum;
        //ans.minsum=min(lo.rmis+ro.lmis,min(lo.minsum,ro.minsum));
        ans.lmis=min(lo.sum+ro.lmis,lo.lmis);
        ans.rmis=min(ro.sum+lo.rmis,ro.rmis);
        return ans;
    }
}seg;
```

一些日常的dp

LIS (最长上升子序列) $O(nlogn)$

```
//最简谱的LIS
int d[maxn];
int lis(int c[],int lenc){
    d[1]=c[1];
    int len=1;
    for(int i=2;i<=lenc;i++){
        if(c[i]>d[len]){
            d[++len]=c[i];
        }
        else {
            int pos=lower_bound(d+1,d+1+len,c[i])-d;
            d[pos]=c[i];
        }
    }
    return len;
}

//能够记录pre/father/from前驱的，vis顺手还标记了一下
//这里的d记录了下标而不是数值，所以二分自己手写吧。。。
//里边的二分是[ll,rr)这样的，正好ll+1是要修改的，注意ll=0是起始
int d[maxn];
bool vis[maxn];
int from[maxn];
int lis_where(int c[],int lenc){
    memset(vis,0,sizeof(vis));
    memset(from,0,sizeof(from));

    int u=1;
    while(c[u]==-1)u++;
    d[1]=u;
    //可能的修改d[1]=1;
    int len=1;

    for(int i=u+1;i<=lenc;i++){
        if(c[i]==-1)continue;
        //这个是那道题会删除数组的数，所以我置换成-1表示删掉了。
        //如果没有删除而且还有可能有负数存在的话，请务必修改这里！
        //（上边那个while和u的存在也是这个原因）

        if(c[i]>c[d[len]]){
            d[++len]=i;
            from[i]=d[len-1];
        }
        else {
```

```
            //int pos=lower_bound(d+1,d+1+len,c[i])-d;
            int ll=0,rr=len+1,mid;
            while(ll+1<rr){//[ll,rr)
                mid=(ll+rr)>>1;
                if(c[d[mid]]<c[i])ll=mid;
                else rr=mid;
            }
            d[ll+1]=i;
            from[i]=d[ll];
        }
    }

    int cur=d[len];
    while(cur>0)vis[cur]=1,cur=from[cur];
    //如果不用标记似乎可以去掉哟
    return len;
}
```

没思路的时候看看这些沙雕想法

- 不管出题人怎么神仙也不能跳过的步骤应该就不会超时
- 如果我是出题人，那么我想考的点应该是？
- 打表找规律猜公式瞎搞
- 如果数据莫名要不要试试打表啊~可能答案没数据范围那么大
- 没事打个暴力吧，搞不好发现能奇怪技巧（拉格朗日插值啥的）加速暴力时间就是正解哇
- 如果二分的话，考虑一下左边搞一搞，右边检测可能就复杂度降下来了呢$O(2^n) \rightarrow O(2*2^{\frac{n}{2}})$
- 没法做？经过若干次尝试和总结后...会发现哒！
- 热爱生活，别怕烦躁哟da☆ze~

卡--------卡----------

卡常的方法分为以下几类：

0.硬件优化：浮点加速器，CPU等等。

1.编译优化：编译开关，inline，register 之类的。

2.封装优化：把代码的封装拆开/循环展开，封装得过死会使效率降低(stl)。

3.编写优化：细节优化，手写栈之类。

4.时空互易： 时间换空间以达到时空均衡(常数层面上)。