**/\* Lazy Propagation \*/**

// update and query in range [x-y), sum in interval

**int** T[RANG], N[RANG];

**int** n;

**struct** LAZY\_PROP {

**int** V; **bool** B;

} lazy[RANG];

**void** shift(**int** id, **int** l, **int** r){

**if**(lazy[id].B){

lazy[2\*id].B = lazy[2\*id+1].B = 1;

lazy[2\*id].V = lazy[2\*id+1].V = lazy[id].V;

**int** mid = (l+r)/2;

T[2 \* id] = lazy[id].V\*(mid-l);

T[2\*id+1] = lazy[id].V\*(r-mid);

}

lazy[id].B = 0;

}

**void** build(**int** id = 1, **int** l = 0, **int** r = n){

**if**(r-l < 2){

lazy[id].B = 1; lazy[id].V = N[l];

T[id] = N[l];

**return**;

}

**int** mid = (l+r)/2;

build(2 \* id, l, mid);

build(2\*id+1, mid, r);

T[id] = T[2\*id] + T[2\*id+1];

}

**void** update(**int** x, **int** y, **int** v, **int** id = 1, **int** l = 0, **int** r = n){

**if**(x >= r || l >= y) **return**;

**if**(x <= l && r <= y){

lazy[id].B = 1; lazy[id].V = v;

T[id] = v\*(r-l);

**return**;

}

**int** mid = (l+r)/2;

shift(id, l, r);

update(x, y, v, 2 \* id, l, mid);

update(x, y, v, 2\*id+1, mid, r);

T[id] = T[2\*id] + T[2\*id+1];

}

**int** query(**int** x, **int** y, **int** id = 1, **int** l = 0, **int** r = n){

**if**(x >= r || l >= y) **return** 0;

**if**(x <= l && r <= y) **return** T[id];

shift(id, l, r);

**int** mid = (l+r)/2;

**int** p1 = query(x, y, 2 \* id, l, mid);

**int** p2 = query(x, y, 2\*id+1, mid, r);

**return** p1+p2;

}

**/\* Inverso de Factoriales \*/**

fact[0]=1;

**for**(**int** i=1; i<MN; i++)

fact[i] = (fact[i-1]\*(ll)i)%mod;

ifact[MN-1] = POW(fact[MN-1], mod-2);

**for**(**int** i=MN-2; i>=0; i--)

ifact[i] = (ifact[i+1]\*(ll)(i+1))%mod;

**/\* Algoritmo Shanka-Tonelli, devuelve x (mod p) tal que x^2 == a (mod p) \*/**

**long** **long** solve\_quadratic(**long** **long** a, **int** p){

**if**(a == 0) **return** 0;

**if**(p == 2) **return** a;

**if**(powMod(a,(p-1)/2, p) != 1) **return** -1;

**int** phi = p-1, n = 0, k = 0, q = 0;

**while**(phi%2 == 0) phi/=2, n ++;

k = phi;

**for**(**int** j = 2; j < p; j++)

**if**(powMod(j, (p-1)/2, p) == p-1){

q = j; **break**;

}

**long** **long** t = powMod(a, (k+1)/2, p);

**long** **long** r = powMod(a, k, p);

**while**(r != 1){

**int** i = 0, v = 1;

**while**(powMod(r, v, p) != 1){

v \*= 2; i++;

}

**long** **long** e = powMod(2, n-i-1, p);

**long** **long** u = powMod(q, k\*e, p);

t = (t\*u)%p; r = (r\*u\*u)%p;

}

**return** t;

}

**/\* Baker-Bird (2D pattern matching) \*/**

**const** **int** MAXN=2e3+10;

**char** P[MAXN][MAXN], T[MAXN][MAXN];

**typedef** **unsigned** **long** **long** ull;

ull hp[2][MAXN][MAXN], ht[2][MAXN][MAXN];

ull bas[2] = {1e9 + 7 , 1e9 + 11};

ull po[2][MAXN];

**int** np,mp,nt,mt;

**int** F[MAXN];

ull hash\_P(**int** k,**int** u){

**return** hp[u][k][np] - hp[u][k][0]\*po[u][np];

}

ull hash\_T(**int** k,**int** i,**int** f,**int** u){

**return** ht[u][k][f-1] - ht[u][k][i-1]\*po[u][f-i];

}

**int** main(){

scanf("%d%d%d%d",&np,&mp,&nt,&mt);

**for**(**int** i=1;i<=np;i++)

scanf("%s",P[i]+1);

**for**(**int** i=1;i<=nt;i++)

scanf("%s",T[i]+1);

po[0][0] = po[1][0] = 1;

**for**(**int** j = 0 ; j < 2 ;j++)

**for**(**int** i = 1 ; i <=nt ;i++)

po[j][i] = po[j][i-1]\*bas[j];

**for**(**int** i=1;i<=mp;i++)

hp[1][i][0] = hp[0][i][0] = 1;

**for**(**int** j = 0 ; j < 2 ;j++)

**for**(**int** k=1;k<=mp;k++)

**for**(**int** i =1 ; i <= np ;i++)

hp[j][k][i] = (hp[j][k][i-1]\*bas[j]) + P[i][k];

**for**(**int** i=0;i<mt;i++)

ht[1][i][0] = ht[0][i][0] = T[i][0];

**for**(**int** j = 0 ; j < 2 ;j++)

**for**(**int** k=1;k<=mt;k++)

**for**(**int** i =1 ; i <= nt ;i++)

ht[j][k][i] = (ht[j][k][i-1]\*bas[j]) + T[i][k];

F[1] = 0;

**int** k=0;

**for**(**int** i=2;i<=mp;i++){

**while**(k>0 && (hash\_P(k+1,0)!=hash\_P(i,0) || hash\_P(k+1,1)!=hash\_P(i,1)))

k=F[k];

**if**(hash\_P(k+1,0)==hash\_P(i,0) &&hash\_P(k+1,1)==hash\_P(i,1)) k++;

F[i]=k;

}

**int** cont=0;

**for**(**int** f=1;f<=nt-np+1;f++){

**int** k=0;

**for**(**int** i=1;i <= mt;i++){

**while**((k>0) && (hash\_P(k+1,0)!=hash\_T(i,f,f+np,0) ||

hash\_P(k+1,1)!=hash\_T(i,f,f+np,1)))

k = F[k];

**if**(hash\_P(k+1,0)==hash\_T(i,f,f+np,0) &&

hash\_P(k+1,1)==hash\_T(i,f,f+np,1)) k++;

**if**(k==mp)

cont++,k = F[k];//found

}

}

**cout** << cont << '\n';

**return** 0;

}