Header

```
/*-------property of the half blood prince-----*/

#include <bits/stdc++.h>
#include <dirent.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#define MIN(X,Y) X<Y?X:Y
#define MAX(X,Y) X>Y?X:Y
#define ISNUM(a) ('0'<=(a) && (a)<='9')
#define ISCAP(a) ('A'<=(a) && (a)<='Z')
#define ISSML(a) ('a'<=(a) && (a)<='z')
#define ISALP(a) (ISCAP(a) || ISSML(a))
#define MXX 10000000000
#define MNN -MXX
#define ISVALID(X,Y,N,M) ((X)>=1 && (X)<=(N) && (Y)>=1 && (Y)<=(M))
#define LLI long long int
#define VI vector<int>
#define VLLI vector<long long int>
#define MII map<int,int>
#define SI set<int>
#define PB push_back
#define MSI map<string,int>
#define PII pair<int,int>
#define PLLI pair<LLI,LLI>
#define PDD pair<double,double>
#define FREP(i,I,N) for(int (i)=(int)(I);(i)<=(int)(N);(i)++)
#define eps 0.0000000001
#define RFREP(i,N,I) for(int (i)=(int)(N);(i)>=(int)(I);(i)--)
#define SORTV(VEC) sort(VEC.begin(),VEC.end())
#define SORTVCMP(VEC,cmp) sort(VEC.begin(),VEC.end(),cmp)
#define REVV(VEC) reverse(VEC.begin(),VEC.end())
#define INRANGED(val,l,r) (((l)<(val) || fabs((val)-(l))<eps) && ((val)<(r) || fabs((val)-(r))<eps))
#define INRANGEI(val,l,r) ((val)>=(l) && (val)<=(r))
#define MSET(a,b) memset(a,b,sizeof(a))
using namespace std;
using namespace __gnu_pbds;

//int dx[]={1,0,-1,0};int dy[]={0,1,0,-1}; //4 Direction
//int dx[]={1,1,0,-1,-1,-1,0,1};int dy[]={0,1,1,1,0,-1,-1,-1};//8 direction
//int dx[]={2,1,-1,-2,-2,-1,1,2};int dy[]={1,2,2,1,-1,-2,-2,-1};//Knight Direction
//int dx[]={2,1,-1,-2,-1,1};int dy[]={0,1,1,0,-1,-1}; //Hexagonal Direction


//typedef tree < int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update > ordered_set;
```

Trie

```
class TrieNode {
public:
    // Initialize your data structure here.
    TrieNode() {
        value = 0;
        parent = NULL; //need parent for delete operation
        for (int i=0;i<26;i++){
            children[i] = NULL;
        }
        freq = 0;
    }
    int value;
    int freq;
    int childofparent; //0 theke 25 er moddhe kichu ekta
    TrieNode* children[26];
    TrieNode* parent;
};

class Trie {
public:
    Trie() {
        root = new TrieNode();
        count = 0;
    }

    // Inserts a word into the trie.
    int insert(string s) {
        TrieNode *p = root;
        int len = s.size();
        for (int i=0;i<len;i++){
            int idx = s[i] - 'a';
            if (! p->children[idx]){
                p->children[idx] = new TrieNode();
            }
            p->children[idx]->parent = p;
            p->children[idx]->childofparent = idx;
            p = p->children[idx];
        }
        count++;
        p->value = count;
        p->freq++;
        return p->freq;
    }

    // Returns if the word is in the trie.
    bool search(string key) {
        TrieNode *p = root;
        int len = key.size();
        for (int i=0;i<len;i++){
            int idx = key[i] - 'a';
            if (p->children[idx]){
                p = p->children[idx];
            }else{
                return false;
            }
        }
        if (p->value!=0){
```

```
            return true;
        }else{
            return false;
        }
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    bool startsWith(string prefix) {
        TrieNode *p = root;
        int len = prefix.size();
        for (int i=0;i<len;i++){
            int idx = prefix[i] - 'a';
            if (p->children[idx]){
                p = p->children[idx];
            }else{
                return false;
            }
        }
      return true;
    }

    void delete(string s){ //deletes a single instance of that word
        //add korte hobe
    }

private:
    TrieNode* root;
    int count;
};

// Your Trie object will be instantiated and called as such:
// Trie trie;
// trie.insert("somestring");
// trie.search("key");
```

parametric line

```
vector<PII>pts;
vector<PDD>ans;

int ansflag;

PDD conv(PII pt){
    PDD myp;
    myp.first = pt.first; myp.second = pt.second;
    return myp;
}

void init(){
    ansflag = 0;
    pts.clear();
    ans.clear();
}

struct line{
    PDD startpoint;
    PDD dir;
    line(PDD p1, PDD p2){
        startpoint = p1;
        dir.first = p2.first-p1.first;
        dir.second = p2.second-p1.second;
        //normalizedir();
        //cout<<"line parameters\n";
        //cout<<startpoint.first<<" "<<startpoint.second<<"\n";
        //cout<<dir.first<<" "<<dir.second<<"\n";
    }
    void normalizedir(){
        double sm = dir.first*dir.first+dir.second*dir.second;
        sm = sqrt(sm);
        dir.first/=sm; dir.second/=sm;
    }
    bool ispointonline(PDD myp){
        //cout<<"for point "<<myp.first<<" "<<myp.second<<"\n";
        if(fabs(dir.first)<eps){
            double t = (myp.second-startpoint.second)/dir.second;
            double xx = startpoint.first + t*dir.first;
            //cout<<xx<<" "<<myp.first<<"\n";
            //cout<<t<<"\n";
            if(fabs(xx-myp.first)<eps){
                if(INRANGED(t,0.0,1.0)){
                    return true;
                }
            }
        }
        else{
            double t = (myp.first-startpoint.first)/dir.first;
            double yy = startpoint.second + t*dir.second;
            //cout<<yy<<" "<<myp.second<<"\n";
            //cout<<t<<"\n";
            if(fabs(yy-myp.second)<eps){
                if(INRANGED(t,0.0,1.0)){
                    return true;
                }
            }
        }

        return false;
    }
    PDD linesolve(line line2){
        double hor = dir.first*line2.dir.second-line2.dir.first*dir.second;
        if(fabs(hor)<eps){
            return make_pair(-1000.00,-1000.00);
        }
        double lob = dir.second*(line2.startpoint.first-startpoint.first)-dir.first*(line2.startpoint.second-startpoint.second);
        double t2 = lob/hor;
        double t1;
```

```
        if(fabs(dir.first)<eps){
            t1 = line2.startpoint.second+(t2*line2.dir.second)-startpoint.second;
            t1 = t1/dir.second;
        }
        else{
            t1 = line2.startpoint.first+(t2*line2.dir.first)-startpoint.first;
            t1 = t1/dir.first;
        }
        return make_pair(t1,t2);
    }
    PDD getpoint(double t){
        PDD gopoint;
        gopoint.first = startpoint.first+t*dir.first;
        gopoint.second = startpoint.second+t*dir.second;
        return gopoint;
    }
    bool isparallel(line line2){
        return (dir.first==line2.dir.first && dir.second==line2.dir.second);
    }
    void print(){
        cout<<"line start : "<<startpoint.first<<" "<<startpoint.second<<"\n";
        cout<<"line end : "<<startpoint.first+dir.first<<" "<<startpoint.second+dir.second<<"\n";
    }
};
```

Half Plane Intersect

```
#include <bits/stdc++.h>
#define ll              long long
//#define pi              pair <int,int>
#define pl              pair <ll,ll>
#define ps              pair <string,string>
#define vi          vector <int>
#define vl              vector <ll>
#define vpi             vector <pi>
#define vpl             vector <pl>
#define st              string
#define vs              vector <st>
#define f(i,a,b)        for(ll i=(a);i<(b);i++)
#define fd(i,a,b)       for(ll i=(a);i>(b);i--)
#define Max(a,b)        ((a)>(b)?(a):(b))
#define Min(a,b)        ((a)<(b)?(a):(b))
#define x               first
#define y               second
#define si(a)           scanf("%d",&a)
#define sii(a,b)        scanf("%d %d",&a,&b)
#define siii(a,b,c)     scanf("%d %d %d",&a,&b,&c)
#define sl(a)           scanf("%lld",&a)
#define sll(a,b)        scanf("%lld %lld",&a,&b)
#define slll(a,b,c)     scanf("%lld %lld %lld",&a,&b,&c);
#define pf              printf
#define pfi(n)          printf("%d\n",n)
#define pfl(n)          printf("%lld\n",n)
#define pfls(n)         printf("%lld ",n)
#define pfci(n,ans)     printf("Case %lld: %d\n",n,ans)
#define pfcl(n,ans)     printf("Case %lld: %lld\n",n,ans)
#define pfcd(n,ans)     printf("Case %lld: %lf\n",n,ans)
#define pb              push_back
#define all(v)          v.begin(),v.end()
#define mem(a,v)        memset(a,v,sizeof(a))
#define INF 1e18
#define MAX 30007
#define MOD 1000000007
#define LG  16
#define mid ((l+r)/2)
using namespace std;

typedef pair<int,int> pii;
const double pi = 4 * atan(1);
const double eps = 1e-12;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }
inline double getDistance (double x, double y) { return sqrt(x * x + y * y); }
inline double torad(double deg) { return deg / 180 * pi; }

struct Point {
    double x, y;
    Point (double x = 0, double y = 0): x(x), y(y) {}
    void read () { scanf("%lf%lf", &x, &y); }
    void write () { printf("%lf %lf", x, y); }

    bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }
    bool operator != (const Point& u) const { return !(*this == u); }
    bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }
    bool operator > (const Point& u) const { return u < *this; }
    bool operator <= (const Point& u) const { return *this < u || *this == u; }
    bool operator >= (const Point& u) const { return *this > u || *this == u; }
    Point operator + (const Point& u) { return Point(x + u.x, y + u.y); }
    Point operator - (const Point& u) { return Point(x - u.x, y - u.y); }
    Point operator * (const double u) { return Point(x * u, y * u); }
    Point operator / (const double u) { return Point(x / u, y / u); }
    double operator * (const Point& u) { return x*u.y - y*u.x; }
};
typedef Point Vector;
typedef vector<Point> Polygon;

struct DirLine {
    Point p;
    Vector v;
    double ang;
    DirLine () {}
    DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator < (const DirLine& u) const { return ang < u.ang; }
};

double getDot (Vector a, Vector b) { return a.x * b.x + a.y * b.y; }
double getCross (Vector a, Vector b) { return a.x * b.y - a.y * b.x; }
```

```
double getLength (Vector a) { return sqrt(getDot(a, a)); }
double getPLength (Vector a) { return getDot(a, a); }
double getAngle (Vector u) { return atan2(u.y, u.x); }
double getAngle (Vector a, Vector b) { return acos(getDot(a, b) / getLength(a) / getLength(b)); }
Vector rotate (Vector a, double rad) { return Vector(a.x*cos(rad)-a.y*sin(rad), a.x*sin(rad)+a.y*cos(rad)); }
Vector getNormal (Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {
    if (dcmp(getCross(v, w)) == 0) return false;
    Vector u = p - q;
    double k = getCross(w, u) / getCross(v, w);
    o = p + v * k;
    return true;
}

bool onLeft(DirLine l, Point p) { return dcmp(l.v * (p-l.p)) >= 0; }

int halfPlaneIntersection(DirLine* li, int n, Point* poly) {
    sort(li, li + n);

    int first, last;
    Point* p = new Point[n];
    DirLine* q = new DirLine[n];
    q[first=last=0] = li[0];

    for (int i = 1; i < n; i++) {
        while (first < last && !onLeft(li[i], p[last-1])) last--;
        while (first < last && !onLeft(li[i], p[first])) first++;
        q[++last] = li[i];

        if (dcmp(q[last].v * q[last-1].v) == 0) {
            last--;
            if (onLeft(q[last], li[i].p)) q[last] = li[i];
        }

        if (first < last)
            getIntersection(q[last-1].p, q[last-1].v, q[last].p, q[last].v, p[last-1]);
    }

    while (first < last && !onLeft(q[first], p[last-1])) last--;
    if (last - first <= 1) { delete [] p; delete [] q; return 0; }
    getIntersection(q[last].p, q[last].v, q[first].p, q[first].v, p[last]);

    int m = 0;
    for (int i = first; i <= last; i++) poly[m++] = p[i];
    delete [] p; delete [] q;
    return m;
}

Point pts[MAX],  poly[MAX];
DirLine dir[MAX], tmp[MAX];
Vector nrm[MAX];

int get(double sh,int n){
          f(i,0,n){
                    tmp[i]=dir[i];
                    tmp[i].p=tmp[i].p+nrm[i]*sh;
          }
          return halfPlaneIntersection(tmp, n, poly);
}
int main(){
          while(1){
                    int N;
                    cin>>N;
                    if(!N) break;
                    f(i,0,N){
                              pts[i].read();
                    }
                    //reverse(pts,pts+N);
                    f(i,0,N){
                              dir[i]=DirLine(pts[i],pts[(i+1)%N]-pts[i]);
                              nrm[i]=getNormal(pts[(i+1)%N]-pts[i]);
                              nrm[i]=nrm[i]/getLength(nrm[i]);
                    }
                    double l=0, r=100000.0;
                    f(i,0,100){
                              if(!get(mid,N)) r=mid;
                              else l=mid;
                    }
                    //cout<<mid<<endl;
                    pf("%.8lf\n",mid);
          }
}
```

Dinik

```
#include <iostream>
#include <cstdio>
#include <cstring>

using namespace std;

#define si(a) scanf("%d",&a)
#define f first
#define s second
#define mp(a,b) make_pair(a,b)
#define INF 2000000000

const int MAX_E=60003;
const int MAX_V=5003;
int ver[MAX_E],cap[MAX_E],nx[MAX_E],last[MAX_V],ds[MAX_V],st[MAX_V],now[MAX_V],edge_count,S,T;

inline void reset()
{
          memset(nx,-1,sizeof(nx));
          memset(last,-1,sizeof(last));
          edge_count=0;
```

```
}
inline void addedge(const int v,const int w,const int capacity,const int reverse_capacity)
{
	ver[edge_count]=w; cap[edge_count]=capacity; nx[edge_count]=last[v]; last[v]=edge_count++;
	ver[edge_count]=v; cap[edge_count]=reverse_capacity; nx[edge_count]=last[w]; last[w]=edge_count++;
}
inline bool bfs()
{
	memset(ds,-1,sizeof(ds));
	int a,b;
	a=b=0;
	st[0]=T;
	ds[T]=0;
	while (a<=b)
	{
		int v=st[a++];
		for (int w=last[v];w>=0;w=nx[w])
		{
			if (cap[w^1]>0 && ds[ver[w]]==-1)
			{
				st[++b]=ver[w];
				ds[ver[w]]=ds[v]+1;
			}
		}
	}
	return ds[S]>=0;
}
int dfs(int v,int cur)
{
	if (v==T) return cur;
	for (int &w=now[v];w>=0;w=nx[w])
	{
		if (cap[w]>0 && ds[ver[w]]==ds[v]-1)
		{
			int d=dfs(ver[w],min(cur,cap[w]));
			if (d)
			{
				cap[w]-=d;
				cap[w^1]+=d;
				return d;
			}
		}
	}
	return 0;
}
inline long long flow()
{
	long long res=0;
	while (bfs())
	{
		for (int i=0;i<MAX_V;i++) now[i]=last[i];
		while (1)
		{
			int tf=dfs(S,INF);
			res+=tf;
			if (!tf) break;
		}
	}
	return res;
}

int main()
{
    //freopen("input.txt","r",stdin);
    int n,m,i;
    si(n);si(m);
    reset();
    S=0;
    T=n-1;
    for(i=0;i<m;i++){
        int u,v,w;
        si(u);si(v);si(w);
        u--;v--;
        addedge(u,v,w,w);
    }
    cout<<flow()<<endl;
    return 0;
}
```

Sample Div Conq DP

```
/*-------property of the half blood prince-----*/

int dp[803][4003];
int familiarval[4003][4003];
//int dp_partition[5003][5003];

void init(){
    MSET(familiarval,0);
}

void precalcor(int n){
    FREP(j,2,n)familiarval[1][j]+=familiarval[1][j-1];
    FREP(i,2,n)familiarval[i][1]+=familiarval[i-1][1];

    FREP(i,2,n){
        FREP(j,2,n){
            familiarval[i][j]+=(familiarval[i-1][j]+familiarval[i][j-1]);
            familiarval[i][j]-=(familiarval[i-1][j-1]);
        }
    }
}

int cost(int l, int r){
    int val1 = familiarval[r][r];
    int val2 = familiarval[l-1][r];
```

```
    int val3 = familiarval[r][l-1];
    int val4 = familiarval[l-1][l-1];
    return (val1+val4)-(val2+val3);
}

void div_conq(int seg, int l, int r, int ans1, int ans2){
    //finding dp[g][(l+r)/2] when i know that dp_partition lies between ans1 and ans2
    if(l>r)return;

    int mid = l+(r-l)/2;
    dp[seg][mid] = 2000000000;
    int parti = -1;
    FREP(k,ans1,min(mid,ans2)){
        int val = dp[seg-1][k]+cost(k+1,mid);
        if(val<dp[seg][mid]){
            dp[seg][mid] = val;
            //dp_partition[seg][mid] = k;
            parti = k;
        }
    }
    div_conq(seg,l,mid-1,ans1,parti);
    div_conq(seg,mid+1,r,parti,ans2);
}

void solve(int gg, int n){
    //MSET(dp_partition,-1);
    FREP(i,0,n)dp[1][i] = cost(1,i);
    FREP(g,2,gg)div_conq(g,1,n,1,n);
}

void printdp(int k, int n){
    FREP(i,1,k)FREP(j,1,n)printf("dp[%d][%d] = %d\n",i,j,dp[i][j]);
}

void printfamiliar(int n){
    FREP(i,1,n){
        FREP(j,1,n){
            printf("%d ",familiarval[i][j]);
        }
        printf("\n");
    }
}

char buffer[10000];

int main(){
    //int t;
    //scanf("%d",&t);
    //init();
    int K,N;
    scanf("%d%d\n",&N,&K);
    for(int i=1; i<=N; i++) {
        gets(buffer);
        for(int j=1; j<=N; j++)
            familiarval[i][j] = (buffer[2*(j - 1)] - '0');
    }
    precalcor(N);
    solve(K,N);
    printf("%d\n",dp[K][N]/2);
    //printfamiliar(N);
    //printf("\n");
    //printdp(K,N);

    return 0;
}
```

Segmented Sieve

```
int arr[SIZE];

///Returns number of primes between segment [a,b]
int segmentedSieve ( int a, int b ) {
    if ( a == 1 ) a++;

    int sqrtn = sqrt ( b );

    memset ( arr, 0, sizeof arr ); ///Make all index of arr 0.

    for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {
        int p = prime[i];
        int j = p * p;

        ///If j is smaller than a, then shift it inside of segment [a,b]
        if ( j < a ) j = ( ( a + p - 1 ) / p ) * p;

        for ( ; j <= b; j += p ) {
            arr[j-a] = 1; ///mark them as not prime
        }
    }

    int res = 0;
    for ( int i = a; i <= b; i++ ) {
        ///If it is not marked, then it is a prime
        if ( arr[i-a] == 0 ) res++;
    }
    return res;
}
```

Extended Euclidean

```
int ext_gcd ( int A, int B, int *X, int *Y ){
    int x2, y2, x1, y1, x, y, r2, r1, q, r;
    x2 = 1; y2 = 0;
    x1 = 0; y1 = 1;
```

```
    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, y2 = y1, x1 = x, y1 = y ) {
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    }
    *X = x2; *Y = y2;
    return r2;
}
```

FFT

```
#include <bits/stdc++.h>
#define cpx complex<double>
#define pi 3.14159265359
using namespace std;
vector<cpx> FFT(vector<int> vc){
        int n=vc.size();
        if(n==1){
                vector<cpx> tmp;
                tmp.push_back(cpx(1.0*vc[0],0.0));
                return tmp;
        }
        int half=n/2;
        vector<int> v0,v1;
        for(int i=0;i<half;i++){
                v0.push_back(vc[i<<1]);
                v1.push_back(vc[i<<1^1]);
        }
        vector<cpx> Y0=FFT(v0),Y1=FFT(v1),foo(n);
        cpx w(1,0),wn(cos(2*pi/n),sin(2*pi/n));
        for(int i=0;i<half;i++){
                foo[i]=Y0[i]+w*Y1[i];
                foo[i+half]=Y0[i]-w*Y1[i];
                w*=wn;
        }
        return foo;
}
vector<cpx> iFFT(vector<cpx> vc){
        int n=vc.size();
        if(n==1){
                return vc;
        }
        int half=n/2;
        vector<cpx> v0,v1;
        for(int i=0;i<half;i++){
                v0.push_back(vc[i<<1]);
                v1.push_back(vc[i<<1^1]);
        }
        vector<cpx> Y0=iFFT(v0),Y1=iFFT(v1),foo(n);
        cpx w(1,0),wn(cos(2*pi/n),sin(-2*pi/n));
        for(int i=0;i<half;i++){
                foo[i]=Y0[i]+w*Y1[i];
                foo[i+half]=Y0[i]-w*Y1[i];
                w*=wn;
        }
        return foo;
}
vector<cpx> comMul(vector<cpx> c1,vector<cpx> c2){
        int n1=c1.size(),n2=c2.size(),mx,mn;
        mx=max(n1,n2);
        mn=min(n1,n2);
        vector<cpx> res;
        for(int i=0;i<mn;i++) res.push_back(c1[i]*c2[i]);
        for(;mn<mx;mn++) res.push_back(cpx(0,0));
        return res;
}
int main() {
        vector<cpx> coef;
        vector<int> coef1,coef2;
        int n,m,tmp;
        scanf("%d %d",&n,&m);                           // n and m degree of two polynomials
        n++;
        m++;
        for(int i=n;i--;) {                             // enter n+1 coefficients as c0+c1*x+c2*x*x+....
                scanf("%d",&tmp);
                coef1.push_back(tmp);
        }
        tmp=1;
        while(tmp<=n) tmp<<=1;
        for(int i=n;i<tmp;i++) coef1.push_back(0);
        tmp<<=1;
        for(int i=tmp>>1;i<tmp;i++) coef1.push_back(0);
        for(int i=m;i--;) {                             // enter m+1 coefficients as c0+c1*x+c2*x*x+....
                scanf("%d",&tmp);
                coef2.push_back(tmp);
        }
        tmp=1;
        while(tmp<=m) tmp<<=1;
        for(int i=m;i<tmp;i++) coef2.push_back(0);
        tmp<<=1;
        for(int i=tmp>>1;i<tmp;i++) coef2.push_back(0);
        coef=iFFT(comMul(FFT(coef1),FFT(coef2)));
        for(int i=0;i<(n+m-1);i++){
                cout<<round(real(coef[i]))/tmp<<"  "; // coefficients of polynomial which is product of
        }                                                                                       // two polunomials
}
```