Contents

1 Basic

1.1 .vimrc

```
syn on
se ai nu ru cul mouse=a
se cin et ts=2 sw=2 sts=2
so $VIMRUNTIME/mswin.vim
colo desert
se gfn=Monospace\ 14
noremap <buffer><F9> :! g++ -std=c++14 -O2 -Wall -Wshadow '%' -o '%<'<CR>
noremap <buffer><F5> :! './%<'<CR>
noremap <buffer><F6> :! './%<' < './%<.in'<CR>
noremap <buffer><F7> :! './%<' < './%<.in' > './%<.out'<CR>
```

1.2 Increase Stack Size

```
//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
}
```

2 Math

2.1 Euclidean's Algorithm

```
// a must be greater than b
pair< int, int > gcd( int a, int b ) {
  if ( b == 0 ) return { 1, 0 };
  pair< int, int > q = gcd( b, b % a );
  return { q.second, q.first - q.second * ( a / b ) };
}
```

2.2 Big Integer

```
const int base = 1000000000;
const int base_digits = 9;

class Bigint {
 public:
  vector< int > a;
  int sign;

  Bigint() : sign( 1 ) {}
  Bigint( long long v ) { *this = v; }
  Bigint( const string &s ) { read( s ); }
  void operator=( const Bigint &v ) {
    sign = v.sign;
    a = v.a;
  }
  void operator=( long long v ) {
    sign = 1;
    if ( v < 0 ) sign = -1, v = -v;
    for ( ; v > 0; v = v / base ) a.push_back( v % base );
  }
  Bigint operator+( const Bigint &v ) const {
    if ( sign == v.sign ) {
      Bigint res = v;
      for ( int i = 0, carry = 0; i < (int)max( a.size(), v.a.size() ) || carry; ++i ) {
        if ( i == (int)res.a.size() ) res.a.push_back( 0 );
        res.a[ i ] += carry + ( i < (int)a.size() ? a[ i ] : 0 );
        carry = res.a[ i ] >= base;
        if ( carry ) res.a[ i ] -= base;
      }
      return res;
    }
    return *this - ( -v );
  }
  Bigint operator-( const Bigint &v ) const {
    if ( sign == v.sign ) {
      if ( abs() >= v.abs() ) {
        Bigint res = *this;
      for ( int i = 0, carry = 0; i < (int)v.a.size()
           || carry; ++i ) {
        res.a[ i ] -= carry + ( i < (int)v.a.size() ?
             v.a[ i ] : 0 );
        carry = res.a[ i ] < 0;
        if ( carry ) res.a[ i ] += base;
      }
      res.trim();
      return res;
    }
    return -( v - *this );
  }
  return *this + ( -v );
}
void operator*=( int v ) {
  if ( v < 0 ) sign = -sign, v = -v;
  for ( int i = 0, carry = 0; i < (int)a.size() ||
      carry; ++i ) {
    if ( i == (int)a.size() ) a.push_back( 0 );
    long long cur = a[ i ] * (long long)v + carry;
    carry = (int)( cur / base );
    a[ i ] = (int)( cur % base );
  }
  trim();
}
Bigint operator*( int v ) const {
  Bigint res = *this;
  res *= v;
  return res;
}

friend pair< Bigint, Bigint > divmod( const Bigint &
    a1, const Bigint &b1 ) {
  int norm = base / ( b1.a.back() + 1 );
  Bigint a = a1.abs() * norm;
  Bigint b = b1.abs() * norm;
  Bigint q, r;
  q.a.resize( a.a.size() );

  for ( int i = a.a.size() - 1; i >= 0; i-- ) {
    r *= base;
    r += a.a[ i ];
    int s1 = r.a.size() <= b.a.size() ? 0 : r.a[ b.a.
        size() ];
    int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[
        b.a.size() - 1 ];
    int d = ( (long long)base * s1 + s2 ) / b.a.back
        ();
    r -= b * d;
    while ( r < 0 ) r += b, --d;
    q.a[ i ] = d;
  }

  q.sign = a1.sign * b1.sign;
  r.sign = a1.sign;
  q.trim();
  r.trim();
  return make_pair( q, r / norm );
}

Bigint operator/( const Bigint &v ) const { return
    divmod( *this, v ).first; }

Bigint operator%( const Bigint &v ) const { return
    divmod( *this, v ).second; }

void operator/=( int v ) {
  if ( v < 0 ) sign = -sign, v = -v;
  for ( int i = (int)a.size() - 1, rem = 0; i >= 0;
      --i ) {
    long long cur = a[ i ] + rem * (long long)base;
    a[ i ] = (int)( cur / v );
    rem = (int)( cur % v );
  }
  trim();
}
Bigint operator/( int v ) const {
  Bigint res = *this;
  res /= v;
  return res;
}
int operator%( int v ) const {
```

```
  if ( v < 0 ) v = -v;
  int m = 0;
  for ( int i = a.size() - 1; i >= 0; --i ) m = ( a[
      i ] + m * (long long)base ) % v;
  return m * sign;
}

void operator+=( const Bigint &v ) { *this = *this +
    v; }
void operator-=( const Bigint &v ) { *this = *this -
    v; }
void operator*=( const Bigint &v ) { *this = *this *
    v; }
void operator/=( const Bigint &v ) { *this = *this /
    v; }

bool operator<( const Bigint &v ) const {
  if ( sign != v.sign ) return sign < v.sign;
  if ( a.size() != v.a.size() ) return a.size() *
      sign < v.a.size() * v.sign;
  for ( int i = a.size() - 1; i >= 0; i-- )
    if ( a[ i ] != v.a[ i ] ) return a[ i ] * sign <
        v.a[ i ] * sign;
  return false;
}

bool operator>( const Bigint &v ) const { return v <
    *this; }
bool operator<=( const Bigint &v ) const { return !(
    v < *this ); }
bool operator>=( const Bigint &v ) const { return !(
    *this < v ); }
bool operator==( const Bigint &v ) const { return !(
    *this < v ) && !( v < *this ); }
bool operator!=( const Bigint &v ) const { return *
    this < v || v < *this; }

void trim() {
  while ( !a.empty() && !a.back() ) a.pop_back();
  if ( a.empty() ) sign = 1;
}
bool isZero() const { return a.empty() || ( a.size()
    == 1 && !a[ 0 ] ); }
Bigint operator-() const {
  Bigint res = *this;
  res.sign = -sign;
  return res;
}
Bigint abs() const {
  Bigint res = *this;
  res.sign *= res.sign;
  return res;
}
long long longValue() const {
  long long res = 0;
  for ( int i = a.size() - 1; i >= 0; i-- ) res = res
       * base + a[ i ];
  return res * sign;
}
friend Bigint gcd( const Bigint &a, const Bigint &b )
     { return b.isZero() ? a : gcd( b, a % b ); }
friend Bigint lcm( const Bigint &a, const Bigint &b )
     { return a / gcd( a, b ) * b; }
void read( const string &s ) {
  sign = 1;
  a.clear();
  int pos = 0;
  while ( pos < (int)s.size() && ( s[ pos ] == '-' ||
       s[ pos ] == '+' ) ) {
    if ( s[ pos ] == '-' ) sign = -sign;
    ++pos;
  }
  for ( int i = s.size() - 1; i >= pos; i -=
      base_digits ) {
    int x = 0;
    for ( int j = max( pos, i - base_digits + 1 ); j
        <= i; j++ ) x = x * 10 + s[ j ] - '0';
    a.push_back( x );
  }
  trim();
}
```

```
 friend istream &operator>>( istream &stream, Bigint &v ) {
  string s;
  stream >> s;
  v.read( s );
  return stream;
 }
 friend ostream &operator<<( ostream &stream, const Bigint &v ) {
  if ( v.sign == -1 ) stream << '-';
  stream << ( v.a.empty() ? 0 : v.a.back() );
  for ( int i = (int)v.a.size() - 2; i >= 0; --i )
   stream << setw( base_digits ) << setfill( '0' ) << v.a[ i ];
  return stream;
 }
 static vector< int > convert_base( const vector< int > &a, int old_digits, int new_digits ) {
  vector< long long > p( max( old_digits, new_digits ) + 1 );
  p[ 0 ] = 1;
  for ( int i = 1; i < (int)p.size(); i++ ) p[ i ] = p[ i - 1 ] * 10;
  vector< int > res;
  long long cur = 0;
  int cur_digits = 0;
  for ( int i = 0; i < (int)a.size(); i++ ) {
   cur += a[ i ] * p[ cur_digits ];
   cur_digits += old_digits;
   while ( cur_digits >= new_digits ) {
    res.push_back( int( cur % p[ new_digits ] ) );
    cur /= p[ new_digits ];
    cur_digits -= new_digits;
   }
  }
  res.push_back( (int)cur );
  while ( !res.empty() && !res.back() ) res.pop_back();
  return res;
 }
 typedef vector< long long > vll;
 static vll karatsubaMultiply( const vll &a, const vll &b ) {
  int n = a.size();
  vll res( n + n );
  if ( n <= 32 ) {
   for ( int i = 0; i < n; i++ )
    for ( int j = 0; j < n; j++ ) res[ i + j ] += a[ i ] * b[ j ];
   return res;
  }
  int k = n >> 1;
  vll a1( a.begin(), a.begin() + k );
  vll a2( a.begin() + k, a.end() );
  vll b1( b.begin(), b.begin() + k );
  vll b2( b.begin() + k, b.end() );

  vll a1b1 = karatsubaMultiply( a1, b1 );
  vll a2b2 = karatsubaMultiply( a2, b2 );

  for ( int i = 0; i < k; i++ ) a2[ i ] += a1[ i ];
  for ( int i = 0; i < k; i++ ) b2[ i ] += b1[ i ];

  vll r = karatsubaMultiply( a2, b2 );
  for ( int i = 0; i < (int)a1b1.size(); i++ ) r[ i ] -= a1b1[ i ];
  for ( int i = 0; i < (int)a2b2.size(); i++ ) r[ i ] -= a2b2[ i ];

  for ( int i = 0; i < (int)r.size(); i++ ) res[ i + k ] += r[ i ];
  for ( int i = 0; i < (int)a1b1.size(); i++ ) res[ i ] += a1b1[ i ];
  for ( int i = 0; i < (int)a2b2.size(); i++ ) res[ i + n ] += a2b2[ i ];
  return res;
 }
 Bigint operator*( const Bigint &v ) const {
  vector< int > a6 = convert_base( this->a, base_digits, 6 );
  vector< int > b6 = convert_base( v.a, base_digits, 6 );
  vll a( a6.begin(), a6.end() );
  vll b( b6.begin(), b6.end() );
  while ( a.size() < b.size() ) a.push_back( 0 );
  while ( b.size() < a.size() ) b.push_back( 0 );
  while ( a.size() & ( a.size() - 1 ) ) a.push_back( 0 ), b.push_back( 0 );
  vll c = karatsubaMultiply( a, b );
  Bigint res;
  res.sign = sign * v.sign;
  for ( int i = 0, carry = 0; i < (int)c.size(); i++ ) {
   long long cur = c[ i ] + carry;
   res.a.push_back( (int)( cur % 1000000 ) );
   carry = (int)( cur / 1000000 );
  }
  res.a = convert_base( res.a, 6, base_digits );
  res.trim();
  return res;
 }
};
```

2.3 FFT

```
// const int MAXN = 262144;
// (must be 2^k)
// before any usage, run pre_fft() first
//
// To implement poly. multiply:
//
// fft( n , a );
// fft( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// fft( n , c , 1 );
//
// then you have the result in c :: [cplx]
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
 for(int i=0; i<=MAXN; i++)
  omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
 int basic = MAXN / n;
 int theta = basic;
 for (int m = n; m >= 2; m >>= 1) {
  int mh = m >> 1;
  for (int i = 0; i < mh; i++) {
   cplx w = omega[inv ? MAXN-(i*theta%MAXN)
                     : i*theta%MAXN];
   for (int j = i; j < n; j += m) {
    int k = j + mh;
    cplx x = a[j] - a[k];
    a[j] += a[k];
    a[k] = w * x;
   }
  }
  theta = (theta * 2) % MAXN;
 }
 int i = 0;
 for (int j = 1; j < n - 1; j++) {
  for (int k = n >> 1; k > (i ^= k); k >>= 1);
  if (j < i) swap(a[i], a[j]);
 }
 if (inv)
  for (i = 0; i < n; i++)
   a[i] /= n;
}
```

2.4 NTT

```
typedef long long LL;
// Remember coefficient are mod P
/* p=a*2^n+1
```

```
   n    2^n         p          a    root
   5    32          97         3    5
   6    64          193        3    5
   7    128         257        2    3
   8    256         257        1    3
   9    512         7681       15   17
   10   1024        12289      12   11
   11   2048        12289      6    11
   12   4096        12289      3    11
   13   8192        40961      5    3
   14   16384       65537      4    3
   15   32768       65537      2    3
   16   65536       65537      1    3
   17   131072      786433     6    10
   18   262144      786433     3    10 (605028353,
       2308, 3)
   19   524288      5767169    11   3
   20   1048576     7340033    7    3
   21   2097152     23068673   11   3
   22   4194304     104857601  25   3
   23   8388608     167772161  20   3
   24   16777216    167772161  10   3
   25   33554432    167772161  5    3 (1107296257, 33,
       10)
   26   67108864    469762049  7    3
   27   134217728   2013265921 15   31 */
// (must be 2^k)
// To implement poly. multiply:
// NTT<P, root, MAXN> ntt;
// ntt( n , a ); // or ntt.tran( n , a );
// ntt( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// ntt( n , c , 1 );
//
// then you have the result in c :: [LL]
template<LL P, LL root, int MAXN>
struct NTT{
  static LL bigmod(LL a, LL b) {
    LL res = 1;
    for (LL bs = a; b; b >>= 1, bs = (bs * bs) % P) {
      if(b&1) res=(res*bs)%P;
    }
    return res;
  }
  static LL inv(LL a, LL b) {
    if(a==1)return 1;
    return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
  }
  LL omega[MAXN+1];
  NTT() {
    omega[0] = 1;
    LL r = bigmod(root, (P-1)/MAXN);
    for (int i=1; i<=MAXN; i++)
      omega[i] = (omega[i-1]*r)%P;
  }
  // n must be 2^k
  void tran(int n, LL a[], bool inv_ntt=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
      int mh = m >> 1;
      for (int i = 0; i < mh; i++) {
        LL w = omega[i*theta%MAXN];
        for (int j = i; j < n; j += m) {
          int k = j + mh;
          LL x = a[j] - a[k];
          if (x < 0) x += P;
          a[j] += a[k];
          if (a[j] > P) a[j] -= P;
          a[k] = (w * x) % P;
        }
      }
      theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
      for (int k = n >> 1; k > (i ^= k); k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
    if (inv_ntt) {
      LL ni = inv(n,P);
      reverse( a+1 , a+n );
      for (i = 0; i < n; i++)
        a[i] = (a[i] * ni) % P;
    }
  }
  void operator()(int n, LL a[], bool inv_ntt=false) {
    tran(n, a, inv_ntt);
  }
};
const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

2.5 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n−2] if
// you want to use magic.
bool witness(LL a,LL n,LL u,int t){
  LL x=mypow(a,u,n);
  for(int i=0;i<t;i++) {
    LL nx=mul(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(LL n,int s=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n == 2;
  LL u=n-1; int t=0;
  // n-1 = u*2^t
  while(!(u&1)) u>>=1, t++;
  while(s--){
    LL a=randll()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

2.6 Chinese Remainder

```
int pfn;
// number of distinct prime factors
int pf[MAXN]; // prime factor powers
int rem[MAXN]; // corresponding remainder
int pm[MAXN];
inline void generate_primes() {
  int i,j;
  pnum=1;
  prime[0]=2;
  for(i=3;i<MAXVAL;i+=2) {
    if(nprime[i]) continue;
    prime[pnum++]=i;
    for(j=i*i;j<MAXVAL;j+=i) nprime[j]=1;
  }
}
inline int inverse(int x,int p) {
  int q,tmp,a=x,b=p;
  int a0=1,a1=0,b0=0,b1=1;
  while(b) {
    q=a/b; tmp=b; b=a-b*q; a=tmp;
    tmp=b0; b0=a0-b0*q; a0=tmp;
    tmp=b1; b1=a1-b1*q; a1=tmp;
  }
  return a0;
}
inline void decompose_mod() {
  int i,p,t=mod;
  pfn=0;
  for(i=0;i<pnum&&prime[i]<=t;i++) {
    p=prime[i];
    if(t%p==0) {
      pf[pfn]=1;
      while(t%p==0) {
        t/=p;
        pf[pfn]*=p;
      }
      pfn++;
    }
  }
  if(t>1) pf[pfn++]=t;
}
inline int chinese_remainder() {
  int i,m,s=0;
  for(i=0;i<pfn;i++) {
    m=mod/pf[i];
    pm[i]=(LL)m*inverse(m,pf[i])%mod;
    s=(s+(LL)pm[i]*rem[i])%mod;
  }
  return s;
}
```

2.7 Pollard's rho

```
// does not work when n is prime
LL f(LL x, LL mod){
  return add(mul(x,x,mod),1,mod);
}
LL pollard_rho(LL n) {
  if(!(n&1)) return 2;
  while(true){
    LL y=2, x=rand()%(n-1)+1, res=1;
    for(int sz=2; res==1; sz*=2) {
      for(int i=0; i<sz && res<=1; i++) {
        x = f(x, n);
        res = __gcd(abs(x-y), n);
      }
      y = x;
    }
    if (res!=0 && res!=n) return res;
  }
}
```

2.8 Roots of Polynomial

```
const double eps = 1e-12;
const double inf = 1e+12;
double a[ 10 ], x[ 10 ];
int n;
int sign( double x ){
  return (x < -eps)?(-1):(x>eps);
}
double f(double a[], int n, double x){
  double tmp=1,sum=0;
  for(int i=0;i<=n;i++){
    sum=sum+a[i]*tmp;
    tmp=tmp*x;
  }
  return sum;
}
double binary(double l,double r,double a[],int n){
  int sl=sign(f(a,n,l)),sr=sign(f(a,n,r));
  if(sl==0) return l;
  if(sr==0) return r;
  if(sl*sr>0) return inf;
  while(r-l>eps){
    double mid=(l+r)/2;
    int ss=sign(f(a,n,mid));
    if(ss==0) return mid;
    if(ss*sl>0) l=mid; else r=mid;
  }
  return l;
}
void solve(int n,double a[],double x[],int &nx){
  if(n==1){
    x[1]=-a[0]/a[1];
    nx=1;
    return;
  }
  double da[10], dx[10];
  int ndx;
  for(int i=n;i>=1;i--) da[i-1]=a[i]*i;
  solve(n-1,da,dx,ndx);
  nx=0;
  if(ndx==0){
    double tmp=binary(-inf,inf,a,n);
    if (tmp<inf) x[++nx]=tmp;
    return;
  }
  double tmp;
  tmp=binary(-inf,dx[1],a,n);
  if(tmp<inf) x[++nx]=tmp;
  for(int i=1;i<=ndx-1;i++){
    tmp=binary(dx[i],dx[i+1],a,n);
    if(tmp<inf) x[++nx]=tmp;
  }
  tmp=binary(dx[ndx],inf,a,n);
  if(tmp<inf) x[++nx]=tmp;
}
int main() {
  scanf("%d",&n);
  for(int i=n;i>=0;i--) scanf("%lf",&a[i]);
  int nx;
  solve(n,a,x,nx);
  for(int i=1;i<=nx;i++) printf("%.6f\n",x[i]);
}
```

2.9 Simplex

```
const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
double x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[MAXN][MAXM], double b[MAXN],
               double c[MAXM], int n, int m){
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1;
    d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j)
        if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
        for (int j = 0; j <= m; ++j) if (j != s)
          d[i][j] += d[r][j] * d[i][s];
        d[i][s] *= d[r][s];
      }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; ++j)
      if (s < 0 || ix[s] > ix[j]) {
        if (d[n + 1][j] > eps ||
            (d[n + 1][j] > -eps && d[n][j] > eps))
          s = j;
      }
    if (s < 0) break;
    for (int i = 0; i < n; ++i) if (d[i][s] < -eps) {
      if (r < 0 ||
          (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s])
              < -eps ||
          (dd < eps && ix[r + m] > ix[i + m]))
        r = i;
    }
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for(int i=0; i<m; i++) x[i] = 0;
  for (int i = m; i < n + m; ++i) { // the missing
      enumerated x[i] = 0
    if (ix[i] < m - 1){
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i-m][m];
    }
  }
  return ans;
}
```

3 Data Structure

3.1 Disjoint Set

```
class DisjointSet {
 public:
  static const int N = 1e5 + 10;
  int p[ N ];
  void Init( int x ) {
    for ( int i = 1; i <= x; ++i ) p[ i ] = i;
  }
  int Find( int x ) { return x == p[ x ] ? x : p[ x ] =
       Find( p[ x ] ); }
  void Union( int x, int y ) { p[ Find( x ) ] = Find( y
       ); }
};
```

3.2 Disjoint Set with Undo

```
struct DisjointSet{
  // save() is like recursive
  // undo() is like return
  int n, fa[ N ], sz[ N ];
  vector< pair<int*,int> > h;
  vector<int> sp;
  void init( int tn ){
    n=tn;
    for( int i = 0 ; i < n ; i ++ ){
      fa[ i ]=i;
      sz[ i ]=1;
    }
    sp.clear(); h.clear();
  }
  void assign( int *k, int v ){
    h.PB( {k, *k} );
    *k = v;
  }
  void save(){ sp.PB(SZ(h)); }
  void undo(){
    assert(!sp.empty());
    int last=sp.back(); sp.pop_back();
    while( SZ(h)!=last ){
      auto x=h.back(); h.pop_back();
      *x.first = x.second;
    }
  }
  int f( int x ){
    while( fa[ x ] != x ) x = fa[ x ];
    return x;
  }
  void uni( int x , int y ){
    x = f( x ); y = f( y );
    if( x == y ) return;
    if( sz[ x ] < sz[ y ] ) swap( x, y );
    assign( &sz[ x ] , sz[ x ] + sz[ y ] );
    assign( &fa[ y ] , x);
  }
}djs;
```

3.3 Segement Tree with Lazy Tag

```
#define L( X ) ( X << 1 )
#define R( X ) ( ( X << 1 ) + 1 )
#define mid ( ( l + r ) >> 1 )

class SegmentTree {
 public:
  static const int N = 1e5 + 10;
  int arr[ N ], st[ N << 2 ], lazy[ N << 2 ];

  inline void Pull( int now ) { st[ now ] = max( st[ L(
       now ) ], st[ R( now ) ] ); }
  inline void Push( int now, int l, int r ) {
    if ( lazy[ now ] != 0 ) {
      if ( l != r ) {
        st[ L( now ) ] += lazy[ now ];
        st[ R( now ) ] += lazy[ now ];
        lazy[ L( now ) ] += lazy[ now ];
        lazy[ R( now ) ] += lazy[ now ];
      }
      lazy[ now ] = 0;
    }
  }
  void Build( int now, int l, int r ) {
    if ( l == r ) {
      st[ now ] = arr[ l ];
      return;
    }
    Build( L( now ), l, mid );
    Build( R( now ), mid + 1, r );
    Pull( now );
  }
  void Update( int ql, int qr, int value, int now, int
      l, int r ) {
    if ( ql > qr || l > qr || r < ql ) return;
    Push( now, l, r );
    if ( l == ql && qr == r ) {
      st[ now ] += value;
      lazy[ now ] += value;
      return;
    }
    if ( qr <= mid )
      Update( ql, qr, value, L( now ), l, mid );
    else if ( mid < ql )
      Update( ql, qr, value, R( now ), mid + 1, r );
    else {
      Update( ql, mid, value, L( now ), l, mid );
      Update( mid + 1, qr, value, R( now ), mid + 1, r
          );
    }
    Pull( now );
  }
  int Query( int ql, int qr, int now, int l, int r ) {
    if ( ql > qr || l > qr || r < ql ) return 0;
    Push( now, l, r );
    if ( l == ql && qr == r ) return st[ now ];
    if ( qr <= mid )
      return Query( ql, qr, L( now ), l, mid );
    else if ( mid < ql )
      return Query( ql, qr, R( now ), mid + 1, r );
    else {
      int left = Query( ql, mid, L( now ), l, mid );
      int right = Query( mid + 1, qr, R( now ), mid +
          1, r );
      int ans = max( left, right );
      return ans;
    }
  }
};
```

3.4 Copy on Write Segement Tree

```
// tested with ASC 29 B
#define mid ( ( l + r ) >> 1 )
class Node {
 public:
  int value, l, r, who;
  Node() {}
  Node( int _v ) : value( _v ) { l = r = who = 0; }
};
class SegmentTree {
 public:
  static const int N = 1e9;
  vector< Node > st;

  inline void Pull( int now ) {
    int lchild = st[ now ].l;
    int rchild = st[ now ].r;
    if ( lchild != 0 ) {
      st[ now ].value = st[ lchild ].value;
      st[ now ].who = st[ lchild ].who;
    }
    if ( rchild != 0 && st[ rchild ].value > st[ now ].
        value ) {
      st[ now ].value = st[ rchild ].value;
      st[ now ].who = st[ rchild ].who;
    }
  }
  void Build() {
    st.push_back( Node() );  // Null Node
    st.push_back( Node( 0 ) );
  }
  void Update( int ql, int qr, int value, int who, int
       now = 1, int l = 1, int r = N ) {
    if ( ql > qr or qr < l or ql > r ) return;
    if ( l == ql && qr == r ) {
      st[ now ].value = value;
      st[ now ].who = who;
      return;
    }
    if ( qr <= mid ) {
      if ( st[ now ].l == 0 ) {
        st[ now ].l = st.size();
        st.push_back( Node( 0 ) );
      }
      Update( ql, qr, value, who, st[ now ].l, l, mid )
          ;
    }
    else if ( mid < ql ) {
      if ( st[ now ].r == 0 ) {
        st[ now ].r = st.size();
        st.push_back( Node( 0 ) );
      }
      Update( ql, qr, value, who, st[ now ].r, mid + 1,
           r );
    }
    else {
      if ( st[ now ].l == 0 ) {
        st[ now ].l = st.size();
        st.push_back( Node( 0 ) );
      }
      if ( st[ now ].r == 0 ) {
        st[ now ].r = st.size();
        st.push_back( Node( 0 ) );
      }
      Update( ql, mid, value, who, st[ now ].l, l, mid
          );
      Update( mid + 1, qr, value, who, st[ now ].r, mid
           + 1, r );
    }
    Pull( now );
  }
  pair< int, int > Query( int ql, int qr, int now = 1,
       int l = 1, int r = N ) {
    if ( ql > qr or qr < l or ql > r ) return { 0, 0 };
    if ( l == ql && qr == r ) {
      return { st[ now ].value, st[ now ].who };
    }
    if ( qr <= mid ) {
      if ( st[ now ].l == 0 ) return { 0, 0 };
      return Query( ql, qr, st[ now ].l, l, mid );
    }
    else if ( mid < ql ) {
      if ( st[ now ].r == 0 ) return { 0, 0 };
      return Query( ql, qr, st[ now ].r, mid + 1, r );
    }
    else {
      pair< int, int > lchild = { 0, 0 };
      if ( st[ now ].l != 0 ) lchild = Query( ql, mid,
          st[ now ].l, l, mid );
      pair< int, int > rchild = { 0, 0 };
      if ( st[ now ].r != 0 ) rchild = Query( mid + 1,
          qr, st[ now ].r, mid + 1, r );
      pair< int, int > ans = { 0, 0 };
      if ( lchild.first > ans.first ) {
        ans.first = lchild.first;
        ans.second = lchild.second;
      }
      if ( rchild.first > ans.first ) {
        ans.first = rchild.first;
        ans.second = rchild.second;
      }
      return ans;
    }
  }
};
```

3.5 Persistent Segement Tree

```
// tested with spoj MKTHNUM - K-th Number
#define mid ( ( l + r ) >> 1 )
class Node {
 public:
  int value, l, r;
  Node() { value = l = r = 0; }
};
class SegmentTree {
 public:
  static const int N = 1e5 + 10;
  int ver_size, st_size;
  vector< int > ver;
  vector< Node > st;

  SegmentTree() {
    ver_size = st_size = 0;
    ver.resize( N );
    st.resize( 70 * N );
    ver[ ver_size++ ] = 1;
    st[ 0 ] = st[ 1 ] = Node();
    st_size = 2;
  }
  void AddVersion() {
    ver[ ver_size++ ] = st_size++;
    st[ ver[ ver_size - 1 ] ] = st[ ver[ ver_size - 2 ]
         ];
  }
  inline void Pull( int now ) {
    int lchild = st[ now ].l, rchild = st[ now ].r;
    st[ now ].value = st[ lchild ].value + st[ rchild
        ].value;
  }
  void Build( int now = 1, int l = 1, int r = N ) {
    if ( l == r ) return;
    st[ now ].l = st_size++;
    st[ now ].r = st_size++;
    Build( st[ now ].l, l, mid );
    Build( st[ now ].r, mid + 1, r );
    Pull( now );
  }
  void Update( int prv_now, int now, int pos, int l =
      1, int r = N ) {
    if ( l == r ) {
      st[ now ].value += 1;
      return;
    }
    if ( pos <= mid ) {
      st[ now ].l = st_size++;
      st[ st[ now ].l ] = st[ st[ prv_now ].l ];
      Update( st[ prv_now ].l, st[ now ].l, pos, l, mid
          );
    }
    else {
      st[ now ].r = st_size++;
      st[ st[ now ].r ] = st[ st[ prv_now ].r ];
      Update( st[ prv_now ].r, st[ now ].r, pos, mid +
          1, r );
    }
    Pull( now );
  }
  pair< int, bool > Query( int prv_now, int now, int k,
       int l = 1, int r = N ) {
    int prv_value = st[ prv_now ].value, now_value = st
        [ now ].value;
    if ( l == r && now_value - prv_value == k )
      return make_pair( l, true );
    else if ( now_value - prv_value < k )
      return make_pair( now_value - prv_value, false );
    pair< int, bool > child = Query( st[ prv_now ].l,
        st[ now ].l, k, l, mid );
    if ( child.second == false ) {
      k -= st[ st[ now ].l ].value - st[ st[ prv_now ].
          l ].value;
      child = Query( st[ prv_now ].r, st[ now ].r, k,
          mid + 1, r );
    }
    return child;
  }
};
```

3.6 Rope

```
#include<ext/rope>
using namespace __gnu_cxx;
// inserts c before p.
iterator insert(const iterator& p, charT c) :
// inserts n copies of c before p.
iterator insert(const iterator& p, size_t n, charT c) :
// inserts the character c before the ith element.
void insert(size_t i, charT c) :
// erases the element pointed to by p.
void erase(const iterator& p) :
// erases the range [f, l).
void erase(const iterator& f, const iterator& l) :
// Appends a C string.
void append(const charT* s) :
void replace(const iterator& f, const iterator& l,
    const rope& x)
void replace(const iterator& f, const iterator& l,
    const charT* s)
void replace(const iterator& f1, const iterator& l1,
    const charT* f2, const charT* l2)
void replace(const iterator& f1, const iterator& l1,
    const iterator& f2, const iterator& l2)
void replace(const iterator& p, const rope& x)
void replace(size_t i, size_t n, const rope& x)
void replace(size_t i, size_t n, charT c)
void replace(size_t i, size_t n, const charT* f, const
    charT* l)
void replace(size_t i, size_t n, const iterator& f,
    const iterator& l)
rope substr(iterator f, iterator l) const
rope substr(const_iterator f, const_iterator l) const
rope substr(size_t i, size_t n = 1) const
```

3.7 pb_ds

```
/***************PB_DS priority_queue****************/
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
typedef priority_queue<T,less<T>,pairing_heap_tag> PQ;
typedef PQ::point_iterator PQit;
point_iterator push(const_reference key)
void modify(point_iterator it, const_reference key)
void erase(point_iterator it)
T top()
void pop()
point_iterator begin()
point_iterator end()
void join(priority_queue &other)
template<class Pred> void split(Pred prd,
    priority_queue &other) //Other will contain only
    values v for which prd(v) is true. When calling
    this method, other's policies must be equivalent to
     this object's policies.
template<class Pred> size_type erase_if(Pred prd) //
    Erases any value satisfying prd; returns the number
     of value erased.
//1. push will return a point_iterator, which can be
    saved in a vector and modify or erase afterward.
//2. using begin() and end() can traverse all elements
    in the priority_queue.
//3. after join, other will be cleared.
//4. for optimizing Dijkstra, use pairing_heap
//5. binary_heap_tag is better that std::priority_queue
//6. pairing_heap_tag is better than binomial_heap_tag
    and rc_binomial_heap_tag
//7. when using only push, pop and join, use
    binary_heap_tag
//8. when using modify, use pairing_heap_tag or
    thin_heap_tag
/*****************PB_DS tree**********************/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
typedef tree<K, T, less<K>, rb_tree_tag, Node_Update>
    TREE;
//similar to std::map
//when T = __gnu_pbds::null_type, become std::set
//when Node_Update = tree_order_statistics_node_update,
     TREE become a ordered TREE with two new functions:
//1. iterator find_by_order(size_type order) return the
     smallest order-th element(e.x. when order = 0,
    return the smallest element), when order > TREE.
    size(), return end()
//2. size_type order_of_key(const_reference key) return
     number of elements smaller than key
void join(tree &other) //other和*this的值域不能相交
void split(const_reference key, tree &other) // 清空
    other，然後把*this當中所有大於key的元素移到other
//自定義Node_Update ： 查詢子段和的map<int, int>，需要紀
    🄵子樹的mapped_value的和。
template<class Node_CItr, class Node_Itr, class Cmp_Fn,
     class _Alloc>
struct my_nd_upd {
  virtual Node_CItr node_begin () const = 0;
  virtual Node_CItr node_end () const = 0;
  typedef int metadata_type ; //額外信息，這邊用int
  inline void operator()(Node_Itr it,Node_CItr end_it){
    Node_Itr l=it.get_l_child(), r=it.get_r_child();
    int left = 0 , right = 0;
    if(l != end_it) left = l.get_metadata();
    if(r != end_it) right = r.get_metadata();
    const_cast<metadata_type&>(it.get_metadata())=
      left+right+(*it)->second;
  }
  //operator()功能是將節點it的信息更新，end_it表空節點
  //it是Node_Itr，*之後變成iterator，再取->second變節點
      的mapped_value
  inline int prefix_sum (int x) {
    int ans = 0;
    Node_CItr it = node_begin();
    while(it!=node_end()){
      Node_CItr l = it.get_l_child() , r = it.
          get_r_child();
      if(Cmp_Fn()(x , (*it)->first)) it = l;
      else {
        ans += (*it)->second;
        if(l != node_end ()) ans += l.get_metadata();
        it = r;
      }
    }
    return ans;
  }
  inline int interval_sum(int l ,int r)
  {return prefix_sum(r)-prefix_sum(l-1);}
};
tree<int, int, less<int>, rb_tree_tag, my_nd_upd> T;
printf("%d\n", T.interval_sum(a, b));
/*****************PB_DS hash*********************/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
__gnu_pbds::cc_hash_table<Key, Mapped>
__gnu_pbds::gp_hash_table<Key, Mapped>
//支援find和operator[]
/*****************PB_DS trie********************/
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
typedef trie<string, null_type,
    trie_string_access_traits<>, pat_trie_tag,
             trie_prefix_search_node_update> pref_trie;
```

```
pref_trie.insert(const string &str);
auto range = pref_trie.prefix_range(const string &str);
for(auto it = range.first; it != range.second; ++it)
  cout << *it << '\n';
```

	push	pop	modify	erase	join
std::priority_queue	$\lg(n)$	$\lg(n)$	$n\lg(n)$	$n\lg(n)$	$n\lg(n)$
pairing_heap_tag	1	$\lg(n)$	$\lg(n)$	$\lg(n)$	1
binary_heap_tag	$\lg(n)$	$\lg(n)$	n	n	n
binomial_heap_tag	1	$\lg(n)$	$\lg(n)$	$\lg(n)$	$\lg(n)$
rc_binomial_heap_tag	1	$\lg(n)$	$\lg(n)$	$\lg(n)$	$\lg(n)$
thin_heap_tag	1	$\lg(n)$	$\lg(n)$[ps]	$\lg(n)$	n

ps: 1 if increased_key only else $\lg(n)$

3.8 Link-Cut Tree

```
const int MXN = 100005;
const int MEM = 100005;
struct Splay {
  static Splay nil, mem[MEM], *pmem;
  Splay *ch[2], *f;
  int val, rev, size;
  Splay () : val(-1), rev(0), size(0)
  { f = ch[0] = ch[1] = &nil; }
  Splay (int _val) : val(_val), rev(0), size(1)
  { f = ch[0] = ch[1] = &nil; }
  bool isr()
  { return f->ch[0] != this && f->ch[1] != this; }
  int dir()
  { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d){
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push(){
    if( !rev ) return;
    swap(ch[0], ch[1]);
    if (ch[0] != &nil) ch[0]->rev ^= 1;
    if (ch[1] != &nil) ch[1]->rev ^= 1;
    rev=0;
  }
  void pull(){
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::
    mem;
Splay *nil = &Splay::nil;
void rotate(Splay *x){
  Splay *p = x->f;
  int d = x->dir();
  if (!p->isr()) p->f->setCh(x, p->dir());
  else x->f = p->f;
  p->setCh(x->ch[!d], d);
  x->setCh(p, !d);
  p->pull(); x->pull();
}
vector<Splay*> splayVec;
void splay(Splay *x){
  splayVec.clear();
  for (Splay *q=x;; q=q->f){
    splayVec.push_back(q);
    if (q->isr()) break;
  }
  reverse(begin(splayVec), end(splayVec));
  for (auto it : splayVec) it->push();
  while (!x->isr()) {
    if (x->f->isr()) rotate(x);
    else if (x->dir()==x->f->dir())
      rotate(x->f),rotate(x);
    else rotate(x),rotate(x);
  }
}
Splay* access(Splay *x){
  Splay *q = nil;
  for (;x!=nil;x=x->f){
    splay(x);
    x->setCh(q, 1);
    q = x;
  }
  return q;
}
void evert(Splay *x){
  access(x);
  splay(x);
  x->rev ^= 1;
  x->push(); x->pull();
}
void link(Splay *x, Splay *y){
  evert(x);
  access(x);
  splay(x);
  evert(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y){
  evert(x);
  access(y);
  splay(y);
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}
int N, Q;
Splay *vt[MXN];
Splay* root(Splay *x) {
  access(x);
  while(x->ch[0] != nil) x = x->ch[0];
  splay(x);
  return x;
}
bool con(Splay *x, Splay *y) {
  return root(x) == root(y);
}
int main(int argc, char** argv){
  scanf("%d%d", &N, &Q);
  for (int i=1; i<=N; i++)
    vt[i] = new (Splay::pmem++) Splay(i);
    // link(vt[u], vt[v]);
    // cut(vt[u], vt[v]);
    // con(vt[u], vt[v]);
  }
}
```

3.9 Treap

```
struct Treap{
  int sz , val , pri , tag;
  Treap *l , *r;
  Treap( int _val ){
    val = _val; sz = 1;
    pri = rand(); l = r = NULL; tag = 0;
  }
};
void push( Treap * a ){
  if( a->tag ){
    Treap *swp = a->l; a->l = a->r; a->r = swp;
    int swp2;
    if( a->l ) a->l->tag ^= 1;
    if( a->r ) a->r->tag ^= 1;
    a->tag = 0;
  }
}
int Size( Treap * a ){ return a ? a->sz : 0; }
void pull( Treap * a ){
  a->sz = Size( a->l ) + Size( a->r ) + 1;
}
Treap* merge( Treap *a , Treap *b ){
  if( !a || !b ) return a ? a : b;
  if( a->pri > b->pri ){
    push( a );
    a->r = merge( a->r , b );
    pull( a );
    return a;
  }else{
    push( b );
    b->l = merge( a , b->l );
    pull( b );
    return b;
  }
}
```

```
void split( Treap *t , int k , Treap*&a , Treap*&b ){
  if( !t ){ a = b = NULL; return; }
  push( t );
  if( Size( t->l ) + 1 <= k ){
    a = t;
    split( t->r , k - Size( t->l ) - 1 , a->r , b );
    pull( a );
  }else{
    b = t;
    split( t->l , k , a , b->l );
    pull( b );
  }
}
```

4 Graph

4.1 Dijkstra's Algorithm

```
template< class T >
using MinHeap = priority_queue< T, vector< T >, greater
    < T > >;
vector< pair< int, int > > v[ N ];

vector< int > Dijkstra( int s ) {
  // n: number of nodes
  vector< int > d( n + 1, 1e9 );
  vector< bool > visit( n + 1, false );
  d[ s ] = 0;

  MinHeap< pair< int, int > > pq;
  pq.push( make_pair( d[ s ], s ) );
  while ( 1 ) {
    int now = -1;
    while ( !pq.empty() and visit[ now = pq.top().
        second ] ) pq.pop();
    if ( now == -1 or visit[ now ] ) break;
    visit[ now ] = true;
    for ( int i = 0; i < v[ now ].size(); ++i ) {
      int child = v[ now ][ i ].first;
      int w = v[ now ][ i ].second;
      if ( !visit[ child ] and ( d[ now ] + w ) < d[
          child ] ) {
        d[ child ] = d[ now ] + w;
        pq.push( make_pair( d[ child ], child ) );
      }
    }
  }
  return d;
}
```

4.2 Tarjan's Algorithm

```
// Build: O( V^2 ), Query: O( 1 )
// n: the number of nodes
int graph[ N ][ N ], lca[ N ][ N ];
vector< bool > visit( N, false );

void tarjan( int now ) {
  if ( visit[ now ] ) return;
  visit[ now ] = true;

  for ( int i = 1; i <= n; ++i )
    if ( visit[ i ] ) lca[ now ][ i ] = lca[ i ][ now ]
        = st.Find( i );

  for ( int i = 1; i <= n; ++i )
    if ( g[ now ][ i ] < 1e9 && !visit[ i ] ) {
      tarjan( i );
      st.Union( i, now );
    }
}
```

4.3 Jump Pointer Algorithm

```
// Build: O( VlogV ), Query: O( logV )
int tin[ N ], tout[ N ], ancestor[ N ][ 20 ];
vector< int > v[ N ];

void dfs( int now, int pnow ) {
  tin[ now ] = ++now_time;

  ancestor[ now ][ 0 ] = pnow;
  for ( int i = 1; i < 20; ++i )
    ancestor[ now ][ i ] = ancestor[ ancestor[ now ][ i
         - 1 ] ][ i - 1 ];

  for ( auto child : v[ now ] )
    if ( child != pnow ) dfs( child, now );

  tout[ now ] = ++now_time;
}
bool check_ancestor( int x, int y ) { return ( tin[ x ]
     <= tin[ y ] && tout[ x ] >= tout[ y ] ); }
int find_lca( int x, int y ) {
  if ( check_ancestor( x, y ) ) return x;
  if ( check_ancestor( y, x ) ) return y;

  for ( int i = 19; i >= 0; --i )
    if ( !check_ancestor( ancestor[ x ][ i ], y ) ) x =
         ancestor[ x ][ i ];
  return ancestor[ x ][ 0 ];
}
```

4.4 Maximum Clique

```
// max N = 64
typedef unsigned long long ll;
struct MaxClique{
  static const int N = 64;
  ll nb[ N ] , n , ans;
  void init( ll _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ) nb[ i ] = 0LLU;
  }
  void add_edge( ll _u , ll _v ){
    nb[ _u ] |= ( 1LLU << _v );
    nb[ _v ] |= ( 1LLU << _u );
  }
  void B( ll r , ll p , ll x , ll cnt , ll res ){
    if( cnt + res < ans ) return;
    if( p == 0LLU && x == 0LLU ){
      if( cnt > ans ) ans = cnt;
      return;
    }
    ll y = p | x; y &= -y;
    ll q = p & ( ~nb[ int( log2( y ) ) ] );
    while( q ){
      ll i = int( log2( q & (-q) ) );
      B( r | ( 1LLU << i ) , p & nb[ i ] , x & nb[ i ]
           , cnt + 1LLU , __builtin_popcountll( p & nb[
           i ] ) );
      q &= ~( 1LLU << i );
      p &= ~( 1LLU << i );
      x |= ( 1LLU << i );
    }
  }
  int solve(){
    ans = 0;
    ll _set = 0;
    if( n < 64 ) _set = ( 1LLU << n ) - 1;
    else{
      for( ll i = 0 ; i < n ; i ++ ) _set |= ( 1LLU <<
           i );
    }
    B( 0LLU , _set , 0LLU , 0LLU , n );
    return ans;
  }
} maxClique;
```

4.5 Heavy-Light Decomposition

```
#define SZ(c) (int)(c).size()
#define ALL(c) (c).begin(), (c).end()
#define REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
typedef tuple< int , int > tii;
const int MAXN = 100010;
const int LOG  = 19;
struct HLD{
  int n;
  vector<int> g[MAXN];
  int sz[MAXN], dep[MAXN];
  int ts, tid[MAXN], tdi[MAXN], tl[MAXN], tr[MAXN];
  //  ts : timestamp , useless after yutruli
  //  tid[ u ] : pos. of node u in the seq.
  //  tdi[ i ] : node at pos i of the seq.
  //  tl , tr[ u ] : subtree interval in the seq. of
      node u
  int mom[MAXN][LOG], head[MAXN];
  // head[ u ] : head of the chain contains u
  void dfssz(int u, int p){
    dep[u] = dep[p] + 1;
    mom[u][0] = p;
    sz[u] = 1;
    head[u] = u;
    for(int& v:g[u]) if(v != p){
      dep[v] = dep[u] + 1;
      dfssz(v, u);
      sz[u] += sz[v];
    }
  }
  void dfshl(int u){
    //printf("dfshl %d\n", u);
    ts++;
    tid[u] = tl[u] = tr[u] = ts;
    tdi[tid[u]] = u;
    sort(ALL(g[u]),
         [&](int a, int b){return sz[a] > sz[b];});
    bool flag = 1;
    for(int& v:g[u]) if(v != mom[u][0]){
      if(flag) head[v] = head[u], flag = 0;
      dfshl(v);
      tr[u] = tr[v];
    }
  }
  inline int lca(int a, int b){
    if(dep[a] > dep[b]) swap(a, b);
    //printf("lca %d %d\n", a, b);
    int diff = dep[b] - dep[a];
    REPD(k, LOG-1, 0) if(diff & (1<<k)){
      //printf("b %d\n", mom[b][k]);
      b = mom[b][k];
    }
    if(a == b) return a;
    REPD(k, LOG-1, 0) if(mom[a][k] != mom[b][k]){
      a = mom[a][k];
      b = mom[b][k];
    }
    return mom[a][0];
  }
  void init( int _n ){
    n = _n;
    REP( i , 1 , n ) g[ i ].clear();
  }
  void addEdge( int u , int v ){
    g[ u ].push_back( v );
    g[ v ].push_back( u );
  }
  void yutruli(){
    dfssz(1, 0);
    ts = 0;
    dfshl(1);
    REP(k, 1, LOG-1) REP(i, 1, n)
      mom[i][k] = mom[mom[i][k-1]][k-1];
  }
  vector< tii > getPath( int u , int v ){
    vector< tii > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
      res.push_back( tii(tid[ head[ v ] ] , tid[ v ]) )
          ;
      v = mom[ head[ v ] ][ 0 ];
    }
    res.push_back( tii( tid[ u ] , tid[ v ] ) );
    reverse( ALL( res ) );
    return res;
    /*
     * res : list of intervals from u to v
     * u must be ancestor of v
     * usage :
     * vector< tii >& path = tree.getPath( u , v )
     * for( tii tp : path ) {
     *   int l , r;tie( l , r ) = tp;
     *   upd( l , r );
     *   uu = tree.tdi[ l ] , vv = tree.tdi[ r ];
     *   uu ~> vv is a heavy path on tree
     * }
     */
  }
} tree;
```

4.6 Dominator Tree

```
const int MAXN = 100010;
struct DominatorTree{
#define REP(i,s,e) for(int i=(s);i<=(e);i++)
#define REPD(i,s,e) for(int i=(s);i>=(e);i--)
  int n , m , s;
  vector< int > g[ MAXN ] , pred[ MAXN ];
  vector< int > cov[ MAXN ];
  int dfn[ MAXN ] , nfd[ MAXN ] , ts;
  int par[ MAXN ];
  int sdom[ MAXN ] , idom[ MAXN ];
  int mom[ MAXN ] , mn[ MAXN ];
  inline bool cmp( int u , int v )
  { return dfn[ u ] < dfn[ v ]; }
  int eval( int u ){
    if( mom[ u ] == u ) return u;
    int res = eval( mom[ u ] );
    if(cmp( sdom[ mn[ mom[ u ] ] ] , sdom[ mn[ u ] ] ))
      mn[ u ] = mn[ mom[ u ] ];
    return mom[ u ] = res;
  }
  void init( int _n , int _m , int _s ){
    ts = 0; n = _n; m = _m; s = _s;
    REP( i, 1, n ) g[ i ].clear(), pred[ i ].clear();
  }
  void addEdge( int u , int v ){
    g[ u ].push_back( v );
    pred[ v ].push_back( u );
  }
  void dfs( int u ){
    ts++;
    dfn[ u ] = ts;
    nfd[ ts ] = u;
    for( int v : g[ u ] ) if( dfn[ v ] == 0 ){
      par[ v ] = u;
      dfs( v );
    }
  }
  void build(){
    REP( i , 1 , n ){
      dfn[ i ] = nfd[ i ] = 0;
      cov[ i ].clear();
      mom[ i ] = mn[ i ] = sdom[ i ] = i;
    }
    dfs( s );
    REPD( i , n , 2 ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      for( int v : pred[ u ] ) if( dfn[ v ] ){
        eval( v );
        if( cmp( sdom[ mn[ v ] ] , sdom[ u ] ) )
          sdom[ u ] = sdom[ mn[ v ] ];
      }
      cov[ sdom[ u ] ].push_back( u );
      mom[ u ] = par[ u ];
      for( int w : cov[ par[ u ] ] ){
        eval( w );
        if( cmp( sdom[ mn[ w ] ] , par[ u ] ) )
          idom[ w ] = mn[ w ];
        else idom[ w ] = par[ u ];
      }
      cov[ par[ u ] ].clear();
    }
    REP( i , 2 , n ){
      int u = nfd[ i ];
      if( u == 0 ) continue ;
      if( idom[ u ] != sdom[ u ] )
        idom[ u ] = idom[ idom[ u ] ];
    }
  }
} domT;
```

4.7 Number of Maximal Clique

```
// bool g[][] : adjacent array indexed from 1 to n
void dfs(int sz){
  int i, j, k, t, cnt, best = 0;
  if(ne[sz]==ce[sz]){ if (ce[sz]==0) ++ans; return; }
  for(t=0, i=1; i<=ne[sz]; ++i){
    for (cnt=0, j=ne[sz]+1; j<=ce[sz]; ++j)
    if (!g[lst[sz][i]][lst[sz][j]]) ++cnt;
    if (t==0 || cnt<best) t=i, best=cnt;
  } if (t && best<=0) return;
  for (k=ne[sz]+1; k<=ce[sz]; ++k) {
    if (t>0){ for (i=k; i<=ce[sz]; ++i)
        if (!g[lst[sz][t]][lst[sz][i]]) break;
      swap(lst[sz][k], lst[sz][i]);
    } i=lst[sz][k]; ne[sz+1]=ce[sz+1]=0;
    for (j=1; j<k; ++j)if (g[i][lst[sz][j]])
        lst[sz+1][++ne[sz+1]]=lst[sz][j];
    for (ce[sz+1]=ne[sz+1], j=k+1; j<=ce[sz]; ++j)
    if (g[i][lst[sz][j]]) lst[sz+1][++ce[sz+1]]=lst[sz
        ][j];
    dfs(sz+1); ++ne[sz]; --best;
    for (j=k+1, cnt=0; j<=ce[sz]; ++j) if (!g[i][lst[sz
        ][j]]) ++cnt;
    if (t==0 || cnt<best) t=k, best=cnt;
    if (t && best<=0) break;
}}
void work(){
  ne[0]=0; ce[0]=0;
  for(int i=1; i<=n; ++i) lst[0][++ce[0]]=i;
  ans=0; dfs(0);
}
```

4.8 Strongly Connected Component

```
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<MXN; i++){
      E[i].clear();
      rE[i].clear();
    }
  }
  void add_edge(int u, int v){
    E[u].PB(v);
    rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u])
      if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    for (auto v : rE[u])
      if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    FZ(vst);
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    FZ(vst);
    for (auto v : vec){
      if (!vst[v]){
        rDFS(v);
        nScc++;
      }
    }
  }
};
```

4.9 Dynamic MST

```
/* Dynamic MST O( Q lg^2 Q )
 (qx[i], qy[i])->chg weight of edge No.qx[i] to qy[i]
 delete an edge: (i, \infty)
 add an edge: change from \infty to specific value
 */
const int SZ=M+3*MXQ;
int a[N],*tz;
int find(int xx){
  int root=xx; while(a[root]) root=a[root];
  int next; while((next=a[xx])){a[xx]=root; xx=next; }
  return root;
}
bool cmp(int aa,int bb){ return tz[aa]<tz[bb]; }
int kx[N],ky[N],kt, vd[N],id[M], app[M];
bool extra[M];
void solve(int *qx,int *qy,int Q,int n,int *x,int *y,
    int *z,int m1,long long ans){
  if(Q==1){
    for(int i=1;i<=n;i++) a[i]=0;
    z[ qx[0] ]=qy[0]; tz = z;
    for(int i=0;i<m1;i++) id[i]=i;
    sort(id,id+m1,cmp); int ri,rj;
    for(int i=0;i<m1;i++){
      ri=find(x[id[i]]); rj=find(y[id[i]]);
      if(ri!=rj){ ans+=z[id[i]]; a[ri]=rj; }
    }
    printf("%lld\n",ans);
    return;
  }
  int ri,rj;
  //contract
  kt=0;
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<Q;i++){
    ri=find(x[qx[i]]); rj=find(y[qx[i]]); if(ri!=rj) a[
        ri]=rj;
  }
  int tm=0;
  for(int i=0;i<m1;i++) extra[i]=true;
  for(int i=0;i<Q;i++) extra[ qx[i] ]=false;
  for(int i=0;i<m1;i++) if(extra[i]) id[tm++]=i;
  tz=z; sort(id,id+tm,cmp);
  for(int i=0;i<tm;i++){
    ri=find(x[id[i]]); rj=find(y[id[i]]);
    if(ri!=rj){
      a[ri]=rj; ans += z[id[i]];
      kx[kt]=x[id[i]]; ky[kt]=y[id[i]]; kt++;
    }
  }
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<kt;i++) a[ find(kx[i]) ]=find(ky[i]);
  int n2=0;
  for(int i=1;i<=n;i++) if(a[i]==0)
  vd[i]=++n2;
  for(int i=1;i<=n;i++) if(a[i])
  vd[i]=vd[find(i)];
  int m2=0, *Nx=x+m1, *Ny=y+m1, *Nz=z+m1;
  for(int i=0;i<m1;i++) app[i]=-1;
  for(int i=0;i<Q;i++) if(app[qx[i]]==-1){
    Nx[m2]=vd[ x[ qx[i] ] ]; Ny[m2]=vd[ y[ qx[i] ] ];
        Nz[m2]=z[ qx[i] ];
    app[qx[i]]=m2; m2++;
  }
  for(int i=0;i<Q;i++){ z[ qx[i] ]=qy[i]; qx[i]=app[qx[
      i]]; }
  for(int i=1;i<=n2;i++) a[i]=0;
  for(int i=0;i<tm;i++){
    ri=find(vd[ x[id[i]] ]);  rj=find(vd[ y[id[i]] ]);
    if(ri!=rj){
      a[ri]=rj; Nx[m2]=vd[ x[id[i]] ];
      Ny[m2]=vd[ y[id[i]] ]; Nz[m2]=z[id[i]]; m2++;
    }
  }
  int mid=Q/2;
  solve(qx,qy,mid,n2,Nx,Ny,Nz,m2,ans);
  solve(qx+mid,qy+mid,Q-mid,n2,Nx,Ny,Nz,m2,ans);
}
int x[SZ],y[SZ],z[SZ],qx[MXQ],qy[MXQ],n,m,Q;
void init(){
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++) scanf("%d%d%d",x+i,y+i,z+i);
  scanf("%d",&Q);
  for(int i=0;i<Q;i++){ scanf("%d%d",qx+i,qy+i); qx[i
      ]--; }
}
void work(){ if(Q) solve(qx,qy,Q,n,x,y,z,m,0); }
int main(){init(); work(); }
```

4.10 General Matching

```
const int N = 514, E = (2e5) * 2;
struct Graph{
  int to[E],bro[E],head[N],e;
  int lnk[N],vis[N],stp,n;
  void init( int _n ){
    stp = 0; e = 1; n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      lnk[i] = vis[i] = 0;
  }
  void add_edge(int u,int v){
    to[e]=v,bro[e]=head[u],head[u]=e++;
    to[e]=u,bro[e]=head[v],head[v]=e++;
  }
  bool dfs(int x){
    vis[x]=stp;
    for(int i=head[x];i;i=bro[i]){
      int v=to[i];
      if(!lnk[v]){
        lnk[x]=v,lnk[v]=x;
        return true;
      }else if(vis[lnk[v]]<stp){
        int w=lnk[v];
        lnk[x]=v,lnk[v]=x,lnk[w]=0;
        if(dfs(w)){
          return true;
        }
        lnk[w]=v,lnk[v]=w,lnk[x]=0;
      }
    }
    return false;
  }
  int solve(){
    int ans = 0;
    for(int i=1;i<=n;i++)
      if(!lnk[i]){
        stp++; ans += dfs(i);
      }
    return ans;
  }
} graph;
```

4.11 Minimum General Weighted Matching

```
struct Graph {
  // Minimum General Weighted Matching (Perfect Match)
  static const int MXN = 105;
  int n, edge[MXN][MXN];
  int match[MXN],dis[MXN],onstk[MXN];
  vector<int> stk;
  void init(int _n) {
    n = _n;
    for( int i = 0 ; i < n ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        edge[ i ][ j ] = 0;
  }
  void add_edge(int u, int v, int w)
  { edge[u][v] = edge[v][u] = w; }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.PB(u);
    onstk[u] = 1;
    for (int v=0; v<n; v++){
      if (u != v && match[u] != v && !onstk[v]){
        int m = match[v];
        if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1;
          stk.PB(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
    stk.pop_back();
    return false;
  }
  int solve() {
    // find a match
    for (int i=0; i<n; i+=2){
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for( int i = 0 ; i < n ; i ++ )
        onstk[ i ] = dis[ i ] = 0;
      for (int i=0; i<n; i++){
        stk.clear();
        if (!onstk[i] && SPFA(i)){
          found = 1;
          while (SZ(stk)>=2){
            int u = stk.back(); stk.pop_back();
            int v = stk.back(); stk.pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for (int i=0; i<n; i++)
      ret += edge[i][match[i]];
    ret /= 2;
    return ret;
  }
}graph;
```

4.12 Minimum Steiner Tree

```
// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree{
#define V 33
#define T 8
#define INF 1023456789
  int n , dst[V][V] , dp[1 << T][V] , tdst[V];
  void init( int _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ){
      for( int j = 0 ; j < n ; j ++ )
        dst[ i ][ j ] = INF;
      dst[ i ][ i ] = 0;
    }
  }
  void add_edge( int ui , int vi , int wi ){
    dst[ ui ][ vi ] = min( dst[ ui ][ vi ] , wi );
    dst[ vi ][ ui ] = min( dst[ vi ][ ui ] , wi );
  }
  void shortest_path(){
    for( int k = 0 ; k < n ; k ++ )
      for( int i = 0 ; i < n ; i ++ )
        for( int j = 0 ; j < n ; j ++ )
          dst[ i ][ j ] = min( dst[ i ][ j ],
                dst[ i ][ k ] + dst[ k ][ j ] );
  }
  int solve( const vector<int>& ter ){
    int t = (int)ter.size();
    for( int i = 0 ; i < ( 1 << t ) ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        dp[ i ][ j ] = INF;
    for( int i = 0 ; i < n ; i ++ )
      dp[ 0 ][ i ] = 0;
    for( int msk = 1 ; msk < ( 1 << t ) ; msk ++ ){
      if( msk == ( msk & (-msk) ) ){
        int who = __lg( msk );
        for( int i = 0 ; i < n ; i ++ )
          dp[ msk ][ i ] = dst[ ter[ who ] ][ i ];
        continue;
      }
      for( int i = 0 ; i < n ; i ++ )
        for( int submsk = ( msk - 1 ) & msk ; submsk ;
                 submsk = ( submsk - 1 ) & msk )
            dp[ msk ][ i ] = min( dp[ msk ][ i ],
                            dp[ submsk ][ i ] +
                            dp[ msk ^ submsk ][ i ] );
      for( int i = 0 ; i < n ; i ++ ){
        tdst[ i ] = INF;
        for( int j = 0 ; j < n ; j ++ )
          tdst[ i ] = min( tdst[ i ],
                     dp[ msk ][ j ] + dst[ j ][ i ] );
      }
      for( int i = 0 ; i < n ; i ++ )
        dp[ msk ][ i ] = tdst[ i ];
    }
    int ans = INF;
    for( int i = 0 ; i < n ; i ++ )
      ans = min( ans , dp[ ( 1 << t ) - 1 ][ i ] );
    return ans;
  }
} solver;
```

4.13 BCC based on Vertex

```
struct BccVertex {
  int n,nScc,step,dfn[MXN],low[MXN];
  vector<int> E[MXN],sccv[MXN];
  int top,stk[MXN];
  void init(int _n) {
    n = _n;
    nScc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].PB(v);
    E[v].PB(u);
  }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    stk[top++] = u;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        DFS(v,u);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) {
          int z;
          sccv[nScc].clear();
          do {
            z = stk[--top];
            sccv[nScc].PB(z);
          } while (z != v);
          sccv[nScc].PB(u);
          nScc++;
        }
      } else {
        low[u] = min(low[u],dfn[v]);
      }
    }
  }
  vector<vector<int>> solve() {
    vector<vector<int>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        DFS(i,i);
      }
    }
    REP(i,nScc) res.PB(sccv[i]);
    return res;
  }
}graph;
```

4.14 BCC based on Edge

```
vector<int> g[N];
int dep[N], low[N], stmp;
void go(int now, int prt){
  dep[now]=low[now]=++stmp;
  for( int son : g[now] ){
    if( son == prt ) continue;
    if( dep[son] ){
      low[now]=min(low[now], dep[son]);
      return;
    }
    go(son, now);
    low[now]=min(low[now], low[son]);
    if(dep[now] >= low[son])
      djs.uni(now, son);
    else
      // (now, son) is a bridge
  }
}
void solve(){
  stmp = 0;
  for( int i = 1 ; i <= n ; i ++ )
    dep[ i ] = 0;
  for( int i = 1 ; i <= n ; i ++ )
    if( dep[i] == 0 )
      go( i , i );
}
```

5 Flow

5.1 Bipartite Matching

```
struct BipartiteMatching {  // O( ( V + E ) * sqrt( V ) )
  vector< int > G[ N ];     // N = total number of nodes = n + m
  int n, m, match[ N ], dist[ N ];
  // n: number of nodes on left side, nodes are numbered 1 to n
  // m: number of nodes on right side, nodes are numbered n+1 to n+m
  // G = NIL[0] u G1[G[1---n]] u G2[G[n+1---n+m]]
  bool BFS() {
    queue< int > Q;
    for ( int i = 1; i <= n; i++ ) {
      if ( match[ i ] == 0 ) {
        dist[ i ] = 0;
        Q.push( i );
      }
      else
        dist[ i ] = INF;
    }
    dist[ 0 ] = INF;
    while ( !Q.empty() ) {
      int u = Q.front();
      Q.pop();
      if ( dist[ u ] < dist[ 0 ] )
        for ( int v : G[ u ] )
          if ( dist[ match[ v ] ] == INF ) {
            dist[ match[ v ] ] = dist[ u ] + 1;
            Q.push( match[ v ] );
          }
    }
    return ( dist[ 0 ] != INF );
  }
  bool DFS( int u ) {
    if ( u != 0 ) {
      for ( int v : G[ u ] )
        if ( dist[ match[ v ] ] == dist[ u ] + 1 && DFS
            ( match[ v ] ) ) {
          match[ v ] = u;
          match[ u ] = v;
          return true;
        }
      dist[ u ] = INF;
      return false;
    }
    return true;
  }
  int Max_Match() {
    int matching = 0;
    fill_n( match, n + m + 1, 0 );
    while ( BFS() )
      for ( int i = 1; i <= n; i++ )
        if ( match[ i ] == 0 && DFS( i ) ) matching++;
    return matching;
  }
  void AddEdge( int u, int v ) { G[ u ].push_back( n +
      v ); }
  void DFS2( int u ) {
    dist[ u ] = 1;
    for ( int v : G[ u ] )
      if ( v != match[ u ] ) {
        dist[ v ] = 1;
        if ( match[ v ] != 0 ) DFS2( match[ v ] );
      }
  }
  void Min_Vertex_Cover( vector< int > &lrtn, vector<
      int > &rrtn ) {
    // after calling Max_Match
    fill_n( dist + 1, n + m, 0 );
    for ( int i = 1; i <= n; i++ )
      if ( match[ i ] == 0 ) DFS2( i );
    for ( int i = 1; i <= n; i++ )
      if ( dist[ i ] == 0 ) lrtn.push_back( i );
    for ( int i = n + 1; i <= n + m; i++ )
      if ( dist[ i ] == 1 ) rrtn.push_back( i - n );
  }
} ob;
```

5.2 MaxFlow (ISAP)

```
// O( V^2 * E ) V up to 2w
#define SZ( c ) ( (int)( c ).size() )
class MaxFlow {
 public:
  static const int MAXV = 5e3 + 10;
  static const int INF = 1e18;
  struct Edge {
    int v, c, r;
    Edge( int _v, int _c, int _r ) : v( _v ), c( _c ),
        r( _r ) {}
  };
  int s, t;
  vector< Edge > G[ MAXV * 2 ];
  int iter[ MAXV * 2 ], d[ MAXV * 2 ], gap[ MAXV * 2 ],
      tot;
  void Init( int x ) {
    tot = x + 2;
    s = x + 1, t = x + 2;
    for ( int i = 0; i <= tot; i++ ) {
      G[ i ].clear();
      iter[ i ] = d[ i ] = gap[ i ] = 0;
    }
  }
  void AddEdge( int u, int v, int c ) {
    G[ u ].push_back( Edge( v, c, SZ( G[ v ] ) ) );
    G[ v ].push_back( Edge( u, 0, SZ( G[ u ] ) - 1 ) );
  }
  int DFS( int p, int flow ) {
    if ( p == t ) return flow;
    for ( int &i = iter[ p ]; i < SZ( G[ p ] ); i++ ) {
      Edge &e = G[ p ][ i ];
      if ( e.c > 0 && d[ p ] == d[ e.v ] + 1 ) {
        int f = DFS( e.v, min( flow, e.c ) );
        if ( f ) {
          e.c -= f;
          G[ e.v ][ e.r ].c += f;
          return f;
        }
      }
    }
    if ( ( --gap[ d[ p ] ] ) == 0 )
      d[ s ] = tot;
    else {
      d[ p ]++;
      iter[ p ] = 0;
      ++gap[ d[ p ] ];
    }
    return 0;
  }
  int Solve() {
    int res = 0;
    gap[ 0 ] = tot;
    for ( res = 0; d[ s ] < tot; res += DFS( s, INF ) )
      ;
    return res;
  }
};
```

5.3 MinCostMaxFlow

```
// O( V^2 * F )
class MinCostMaxFlow {
 public:
  static const int MAXV = 2000;
  static const int INF = 1e9;
  struct Edge {
    int v, cap, w, rev;
    Edge() {}
    Edge( int t2, int t3, int t4, int t5 ) : v( t2 ),
        cap( t3 ), w( t4 ), rev( t5 ) {}
  };
  int V, s, t;
  vector< Edge > g[ MAXV ];
  void Init( int n ) {
    V = n + 4;             // total number of nodes
    s = n + 1, t = n + 4;  // s = source, t = sink
    for ( int i = 1; i <= V; i++ ) g[ i ].clear();
  }
  // cap: capacity, w: cost
  void AddEdge( int a, int b, int cap, int w ) {
    g[ a ].push_back( Edge( b, cap, w, (int)g[ b ].size
        () ) );
    g[ b ].push_back( Edge( a, 0, -w, (int)g[ a ].size
        () - 1 ) );
  }
  int d[ MAXV ], id[ MAXV ], mom[ MAXV ];
  bool inqu[ MAXV ];
  int qu[ 2000000 ], ql, qr;
  // the size of qu should be much large than MAXV
  int MncMxf() {
    int INF = INF;
    int mxf = 0, mnc = 0;
    while ( 1 ) {
      fill( d + 1, d + 1 + V, INF );
      fill( inqu + 1, inqu + 1 + V, 0 );
      fill( mom + 1, mom + 1 + V, -1 );
      mom[ s ] = s;
      d[ s ] = 0;
      ql = 1, qr = 0;
      qu[ ++qr ] = s;
      inqu[ s ] = 1;
      while ( ql <= qr ) {
        int u = qu[ ql++ ];
        inqu[ u ] = 0;
        for ( int i = 0; i < (int)g[ u ].size(); i++ )
            {
          Edge &e = g[ u ][ i ];
          int v = e.v;
          if ( e.cap > 0 && d[ v ] > d[ u ] + e.w ) {
            d[ v ] = d[ u ] + e.w;
            mom[ v ] = u;
            id[ v ] = i;
            if ( !inqu[ v ] ) qu[ ++qr ] = v, inqu[ v ]
                 = 1;
          }
        }
      }
      if ( mom[ t ] == -1 ) break;
      int df = INF;
      for ( int u = t; u != s; u = mom[ u ] ) df = min(
           df, g[ mom[ u ] ][ id[ u ] ].cap );
      for ( int u = t; u != s; u = mom[ u ] ) {
        Edge &e = g[ mom[ u ] ][ id[ u ] ];
        e.cap -= df;
        g[ e.v ][ e.rev ].cap += df;
      }
      mxf += df;
      mnc += df * d[ t ];
    }
    return mnc;
  }
};
```

5.4 BoundedMaxFlow

```
// node from 0 ~ size - 1
class Graph {
 public:
  Graph( const int &size )
      : size_( size + 2 ),
        source_( size ),
        sink_( size + 1 ),
        edges_( size_ ),
        capacity_( size_, vector< int >( size_, 0 ) ),
        lower_bound_( size_, vector< int >( size_, 0 )
            ),
        lower_bound_sum_( size_, 0 ) {}
  void AddEdge( int from, int to, int lower_bound, int
      capacity ) {
    edges_[ from ].push_back( to );
    edges_[ to ].push_back( from );

    capacity_[ from ][ to ] += capacity - lower_bound;
    lower_bound_[ from ][ to ] += lower_bound;

    lower_bound_sum_[ from ] += lower_bound;
    lower_bound_sum_[ to ] -= lower_bound;
  }
  int MaxFlow() {
    int expected_source = 0, expected_sink = 0;
    for ( int i = 0; i < source_; ++i )
      if ( lower_bound_sum_[ i ] > 0 ) {
        capacity_[ i ][ sink_ ] = lower_bound_sum_[ i
            ];
        edges_[ i ].push_back( sink_ );
        edges_[ sink_ ].push_back( i );
        expected_sink += lower_bound_sum_[ i ];
      }
      else if ( lower_bound_sum_[ i ] < 0 ) {
        capacity_[ source_ ][ i ] = -lower_bound_sum_[
            i ];
        edges_[ source_ ].push_back( i );
        expected_source -= lower_bound_sum_[ i ];
      }
    int Flow = 0;
    while ( BFS( source_, sink_ ) )
      for ( auto &from : edges_[ sink_ ] ) {
        if ( from_[ from ] == -1 ) continue;

        from_[ sink_ ] = from;
        int current_Flow = numeric_limits< int >::max()
            ;
        for ( int i = sink_; i != source_; i = from_[ i
             ] )
          current_Flow = min( current_Flow, capacity_[
              from_[ i ] ][ i ] );
        if ( not current_Flow ) continue;
        for ( int i = sink_; i != source_; i = from_[ i
             ] ) {
          capacity_[ from_[ i ] ][ i ] -= current_Flow;
          capacity_[ i ][ from_[ i ] ] += current_Flow;
        }
        Flow += current_Flow;
      }
    if ( Flow != expected_source ) return -1;
    return Flow;
  }
  int Flow( int from, int to ) { return lower_bound_[
      from ][ to ] + capacity_[ to ][ from ]; }

 private:
  bool BFS( int source, int sink ) {
    queue< int > Q;
    Q.push( source );
    from_ = vector< int >( size_, -1 );
    from_[ source ] = source;

    while ( !Q.empty() ) {
      int node = Q.front();
      Q.pop();
      if ( node == sink ) continue;
      for ( auto &neighbour : edges_[ node ] )
        if ( from_[ neighbour ] == -1 && capacity_[
            node ][ neighbour ] > 0 ) {
          from_[ neighbour ] = node;
          Q.push( neighbour );
        }
    }
    return from_[ sink ] != -1;
  }
  int size_, source_, sink_;
  vector< vector< int > > edges_;
  vector< vector< int > > capacity_;
  vector< vector< int > > lower_bound_;
  vector< int > lower_bound_sum_;
  vector< int > from_;
};
```

5.5 Dinic

```
struct Dinic{
  static const int MXN = 10000;
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      }
    }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
  }
}flow;
```

5.6 DMST

```
/*
 * Edmond's algoirthm for Directed MST
 * runs in O(VE)
 */
const int MAXV = 10010;
const int MAXE = 10010;
const int INF  = 2147483647;
struct Edge{
  int u, v, c;
  Edge(){}
  Edge(int x, int y, int z) :
    u(x), v(y), c(z){}
};
int V, E, root;
Edge edges[MAXE];
inline int newV(){
  V++;
  return V;
}
inline void addEdge(int u, int v, int c){
  E++;
  edges[E] = Edge(u, v, c);
}
bool con[MAXV];
int mnInW[MAXV], prv[MAXV], cyc[MAXV], vis[MAXV];
inline int DMST(){
  fill(con, con+V+1, 0);
  int r1 = 0, r2 = 0;
  while(1){
    fill(mnInW, mnInW+V+1, INF);
    fill(prv, prv+V+1, -1);
    REP(i, 1, E){
      int u=edges[i].u, v=edges[i].v, c=edges[i].c;
      if(u != v && v != root && c < mnInW[v])
        mnInW[v] = c, prv[v] = u;
    }
    fill(vis, vis+V+1, -1);
    fill(cyc, cyc+V+1, -1);
    r1 = 0;
    bool jf = 0;
    REP(i, 1, V){
      if(con[i]) continue ;
      if(prv[i] == -1 && i != root) return -1;
      if(prv[i] > 0) r1 += mnInW[i];
      int s;
      for(s = i; s != -1 && vis[s] == -1; s = prv[s])
        vis[s] = i;
      if(s > 0 && vis[s] == i){
        // get a cycle
        jf = 1;
        int v = s;
        do{
          cyc[v] = s, con[v] = 1;
          r2 += mnInW[v];
          v = prv[v];
        }while(v != s);
        con[s] = 0;
      }
    }
    if(!jf) break ;
    REP(i, 1, E){
      int &u = edges[i].u;
      int &v = edges[i].v;
      if(cyc[v] > 0) edges[i].c -= mnInW[edges[i].v];
      if(cyc[u] > 0) edges[i].u = cyc[edges[i].u];
      if(cyc[v] > 0) edges[i].v = cyc[edges[i].v];
      if(u == v) edges[i--] = edges[E--];
    }
  }
  return r1+r2;
}
```

5.7 SW min-cut

```
// global min cut
struct SW{ // O(V^3)
  static const int MXN = 514;
  int n,vst[MXN],del[MXN];
  int edge[MXN][MXN],wei[MXN];
  void init(int _n){
    n = _n;
    FZ(edge);
    FZ(del);
  }
  void add_edge(int u, int v, int w){
    edge[u][v] += w;
    edge[v][u] += w;
  }
  void search(int &s, int &t){
    FZ(vst); FZ(wei);
    s = t = -1;
    while (true){
      int mx=-1, cur=0;
      for (int i=0; i<n; i++)
        if (!del[i] && !vst[i] && mx<wei[i])
          cur = i, mx = wei[i];
      if (mx == -1) break;
      vst[cur] = 1;
      s = t;
      t = cur;
      for (int i=0; i<n; i++)
        if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
    }
  }
  int solve(){
    int res = 2147483647;
    for (int i=0,x,y; i<n-1; i++){
      search(x,y);
      res = min(res,wei[y]);
      del[y] = 1;
      for (int j=0; j<n; j++)
        edge[x][j] = (edge[j][x] += edge[y][j]);
    }
    return res;
  }
}graph;
```

5.8 Theorem

```
/*
Lucas’  Theorem:
  For non-negative integer n,m and prime P,
  C(m,n) mod P = C(m/M,n/M) * C(m%M,n%M) mod P
  = mult_i ( C(m_i,n_i) )
  where m_i is the i-th digit of m in base P.

Pick’ s Theorem
  A = i + b/2 - 1

Kirchhoff's theorem
  A_{ii} = deg(i), A_{ij} = (i,j) \in E ? -1 : 0
  Deleting any one row, one column, and cal the det(A)
*/
```

6 Geometry

6.1 Half Plane Intersection

6.2 Intersection of 2 Lines

```
#define N 100010
#define EPS 1e-8
#define SIDE 10000000
struct PO{ double x , y ; } p[ N ], o ;
struct LI{
  PO a, b;
  double angle;
  void in( double x1 , double y1 , double x2 , double
      y2 ){
    a.x = x1 ; a.y = y1 ; b.x = x2 ; b.y = y2;
  }
}li[ N ] , deq[ N ];
int n , m , cnt;
inline int dc( double x ){
  if ( x > EPS ) return 1;
  else if ( x < -EPS ) return -1;
  return 0;
}
inline PO operator-( PO a, PO b ){
  PO c;
  c.x = a.x - b.x ; c.y = a.y - b.y;
  return c;
}
inline double cross( PO a , PO b , PO c ){
  return ( b.x - a.x ) * ( c.y - a.y ) - ( b.y - a.y )
      * ( c.x - a.x );
}
inline bool cmp( const LI &a , const LI &b ){
  if( dc( a.angle - b.angle ) == 0 ) return dc( cross(
      a.a , a.b , b.a ) ) < 0;
  return a.angle > b.angle;
}
inline PO getpoint( LI &a , LI &b ){
  double k1 = cross( a.a , b.b , b.a );
  double k2 = cross( a.b , b.a , b.b );
  PO tmp = a.b - a.a , ans;
  ans.x = a.a.x + tmp.x * k1 / ( k1 + k2 );
  ans.y = a.a.y + tmp.y * k1 / ( k1 + k2 );
  return ans;
}
inline void getcut(){
  sort( li + 1 , li + 1 + n , cmp ); m = 1;
  for( int i = 2 ; i <= n ; i ++ )
    if( dc( li[ i ].angle - li[ m ].angle ) != 0 )
      li[ ++ m ] = li[ i ];
  deq[ 1 ] = li[ 1 ]; deq[ 2 ] = li[ 2 ];
  int bot = 1 , top = 2;
  for( int i = 3 ; i <= m ; i ++ ){
    while( bot < top && dc( cross( li[ i ].a , li[ i ].
        b , getpoint( deq[ top ] , deq[ top - 1 ] ) ) )
         < 0 ) top -- ;
    while( bot < top && dc( cross( li[ i ].a , li[ i ].
        b , getpoint( deq[ bot ] , deq[ bot + 1 ] ) ) )
         < 0 ) bot ++ ;
    deq[ ++ top ] = li[ i ] ;
  }
  while( bot < top && dc( cross( deq[ bot ].a , deq[
      bot ].b , getpoint( deq[ top ] , deq[ top - 1 ] )
       ) ) < 0 ) top --;
  while( bot < top && dc( cross( deq[ top ].a , deq[
      top ].b , getpoint( deq[ bot ] , deq[ bot + 1 ] )
       ) ) < 0 ) bot ++;
  cnt = 0;
  if( bot == top ) return;
  for( int i = bot ; i < top ; i ++ ) p[ ++ cnt ] =
      getpoint( deq[ i ] , deq[ i + 1 ] );
  if( top - 1 > bot ) p[ ++ cnt ] = getpoint( deq[ bot
      ] , deq[ top ] );
}
double px[ N ] , py[ N ];
void read( int rm ) {
  for( int i = 1 ; i <= n ; i ++ ) px[ i + n ] = px[ i
      ] , py[ i + n ] = py[ i ];
  for( int i = 1 ; i <= n ; i ++ ){
    // half-plane from li[ i ].a -> li[ i ].b
    li[ i ].a.x = px[ i + rm + 1 ]; li[ i ].a.y = py[ i
         + rm + 1 ];
    li[ i ].b.x = px[ i ]; li[ i ].b.y = py[ i ];
    li[ i ].angle = atan2( li[ i ].b.y - li[ i ].a.y ,
        li[ i ].b.x - li[ i ].a.x ) ;
  }
}
inline double getarea( int rm ){
  read( rm ); getcut();
  double res = 0.0;
  p[ cnt + 1 ] = p[ 1 ];
  for( int i = 1 ; i <= cnt ; i ++ ) res += cross( o ,
      p[ i ] , p[ i + 1 ] ) ;
  if( res < 0.0 ) res *= -1.0;
  return res;
}
```

6.3 Intersection of 2 Segments

```
int ori( const PLL& o , const PLL& a , const PLL& b ){
  LL ret = ( a - o ) ^ ( b - o );
  return ret / max( 1ll , abs( ret ) );
}
// p1 == p2 || q1 == q2 need to be handled
bool banana( const PLL& p1 , const PLL& p2 ,
             const PLL& q1 , const PLL& q2 ){
  if( ( ( p2 - p1 ) ^ ( q2 - q1 ) ) == 0 ){ // parallel
    if( ori( p1 , p2 , q1 ) ) return false;
    return ( ( p1 - q1 ) * ( p2 - q1 ) ) <= 0 ||
           ( ( p1 - q2 ) * ( p2 - q2 ) ) <= 0 ||
           ( ( q1 - p1 ) * ( q2 - p1 ) ) <= 0 ||
           ( ( q1 - p2 ) * ( q2 - p2 ) ) <= 0;
  }
  return (ori( p1, p2, q1 ) * ori( p1, p2, q2 )<=0) &&
         (ori( q1, q2, p1 ) * ori( q1, q2, p2 )<=0);
}
```

6.4 Intersection of Circle and Segment

```
bool Inter( const Pt& p1 , const Pt& p2 , Circle& cc ){
  Pt dp = p2 - p1;
  double a = dp * dp;
  double b = 2 * ( dp * ( p1 - cc.O ) );
  double c = cc.O * cc.O + p1 * p1 - 2 * ( cc.O * p1 )
      - cc.R * cc.R;
  double bb4ac = b * b - 4 * a * c;
  return !( fabs( a ) < eps or bb4ac < 0 );
}
```

6.5 Intersection of Polygon and Circle

```
Pt ORI , info[ N ];
D r; int n;
// Divides into multiple triangle, and sum up
// oriented area
D area2(Pt pa, Pt pb){
  if( norm(pa) < norm(pb) ) swap(pa, pb);
  if( norm(pb) < eps ) return 0;
  D S, h, theta;
  D a = norm( pb ), b = norm( pa ), c = norm(pb - pa);
  D cosB = (pb * (pb - pa)) / a / c, B = acos(cosB);
  D cosC = (pa * pb) / a / b, C = acos(cosC);
  if(a > r){
    S = (C/2)*r*r;
    h = a*b*sin(C)/c;
    if (h < r && B < PI/2) S -= (acos(h/r)*r*r - h*sqrt
        (r*r-h*h));
  }else if(b > r){
    theta = PI - B - asin(sin(B)/r*a);
    S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
  }else S = .5*sin(C)*a*b;
  return S;
}
D area() {
  D S = 0;
  for(int i = 0; i < n; ++i)
    S += abs( area2(info[i], info[i + 1]) * sign( det(
        info[i], info[i + 1]));
  return fabs(S);
}
```

6.6 Intersection of 2 Circles

6.7 Circle Cover

```
#define N 1021
struct CircleCover{
  int C; Circle c[ N ];
  bool g[ N ][ N ], overlap[ N ][ N ];
  // Area[i] : area covered by at least i circles
  D Area[ N ];
  void init( int _C ){ C = _C; }
  bool CCinter( Circle& a , Circle& b , Pt& p1 , Pt& p2
       ){
    Pt o1 = a.O , o2 = b.O;
    D r1 = a.R , r2 = b.R;
    D d2 = ( o1 - o2 ) * ( o1 - o2 );
    D d = sqrt(d2);
    if( d > r1 + r2 ) return false;
    Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2))
        ;
    D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d)
        );
    Pt v = Pt( o1.Y-o2.Y , -o1.X + o2.X ) * A / (2*d2);
    p1 = u + v; p2 = u - v;
    return true;
  }
  struct Tevent {
    Pt p; D ang; int add;
    Tevent() {}
    Tevent(Pt _a, D _b, int _c): p(_a), ang(_b), add(_c
        ) {}
    bool operator<(const Tevent &a)const
    {return ang < a.ang;}
  }eve[ N * 2 ];
  // strict: x = 0, otherwise x = -1
  bool disjuct( Circle& a, Circle &b, int x ){
    return sign( norm( a.O - b.O ) - a.R - b.R ) > x;
  }
  bool contain( Circle& a, Circle &b, int x ){
    return sign( a.R - b.R - norm( a.O - b.O ) ) > x;
  }
  bool contain(int i, int j){ /* c[j] is non-strictly
      in c[i]. */
    return (sign(c[i].R - c[j].R) > 0 ||
            (sign(c[i].R - c[j].R) == 0 && i < j) ) &&
                contain(c[i], c[j], -1);
  }
  void solve(){
    for( int i = 0 ; i <= C + 1 ; i ++ )
      Area[ i ] = 0;
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        overlap[i][j] = contain(i, j);
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjuct(c[i], c[j], -1));
    for( int i = 0 ; i < C ; i ++ ){
      int E = 0, cnt = 1;
      for( int j = 0 ; j < C ; j ++ )
        if( j != i && overlap[j][i] )
          cnt ++;
      for( int j = 0 ; j < C ; j ++ )
        if( i != j && g[i][j] ){
          Pt aa, bb;
          CCinter(c[i], c[j], aa, bb);
          D A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X)
              ;
          D B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X)
              ;
          eve[E ++] = Tevent(bb, B, 1);
          eve[E ++] = Tevent(aa, A, -1);
          if(B > A) cnt ++;
        }
      if( E == 0 ) Area[ cnt ] += pi * c[i].R * c[i].R;
      else{
        sort( eve , eve + E );
        eve[E] = eve[0];
        for( int j = 0 ; j < E ; j ++ ){
          cnt += eve[j].add;
          Area[cnt] += (eve[j].p ^ eve[j + 1].p) * .5;
          D theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += 2. * pi;
          Area[cnt] += ( theta - sin(theta) ) * c[i].R
              * c[i].R * .5;
        }
      }
    }
  }
};
```

6.8 Tangent Line of 2 Circles

```
vector<Line> go( const Circle& c1 , const Circle& c2 ){
  vector<Line> ret;
  double d_sq = norm2( c1.O - c2.O );
  if( d_sq < eps ) return ret;
  double d = sqrt( d_sq );
  Pt v = ( c2.O - c1.O ) / d;
  for( int sign1 = 1 ; sign1 >= -1 ; sign1 -= 2 ){
    double c = ( c1.R - sign1 * c2.R ) / d;
    if( c * c > 1 ) continue;
    double h = sqrt( max( 0.0 , 1.0 - c * c ) );
    for( int sign2 = 1 ; sign2 >= -1 ; sign2 -= 2 ){
      Pt n;
      n.X = v.X * c - sign2 * h * v.Y;
      n.Y = v.Y * c + sign2 * h * v.X;
      Pt p1 = c1.O + n * c1.R;
      Pt p2 = c2.O + n * ( c2.R * sign1 );
      if( fabs( p1.X - p2.X ) < eps and
          fabs( p1.Y - p2.Y ) < eps )
        p2 = p1 + perp( c2.O - c1.O );
      ret.push_back( { p1 , p2 } );
    }
  }
  return ret;
}
```

6.9 KD Tree

```
const int MXN = 100005;
struct KDTree {
  struct Node {
    int x,y,x1,y1,x2,y2;
    int id,f;
    Node *L, *R;
  }tree[MXN];
  int n;
  Node *root;
  LL dis2(int x1, int y1, int x2, int y2) {
    LL dx = x1-x2;
    LL dy = y1-y2;
    return dx*dx+dy*dy;
  }
  static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
  static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
  void init(vector<pair<int,int>> ip) {
    n = ip.size();
    for (int i=0; i<n; i++) {
      tree[i].id = i;
      tree[i].x = ip[i].first;
      tree[i].y = ip[i].second;
    }
    root = build_tree(0, n-1, 0);
  }
  Node* build_tree(int L, int R, int dep) {
    if (L>R) return nullptr;
    int M = (L+R)/2;
    tree[M].f = dep%2;
    nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
        cmpy : cmpx);
    tree[M].x1 = tree[M].x2 = tree[M].x;
    tree[M].y1 = tree[M].y2 = tree[M].y;

    tree[M].L = build_tree(L, M-1, dep+1);
    if (tree[M].L) {
      tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
    }
    tree[M].R = build_tree(M+1, R, dep+1);
    if (tree[M].R) {
      tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
      tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
      tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
      tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
    }
    return tree+M;
  }
  int touch(Node* r, int x, int y, LL d2){
    LL dis = sqrt(d2)+1;
    if (x<r->x1-dis || x>r->x2+dis ||
        y<r->y1-dis || y>r->y2+dis)
      return 0;
    return 1;
  }
  void nearest(Node* r, int x, int y,
               int &mID, LL &md2){
    if (!r || !touch(r, x, y, md2)) return;
    LL d2 = dis2(r->x, r->y, x, y);
    if (d2 < md2 || (d2 == md2 && mID < r->id)) {
      mID = r->id;
      md2 = d2;
    }
    // search order depends on split dim
    if ((r->f == 0 && x < r->x) ||
        (r->f == 1 && y < r->y)) {
      nearest(r->L, x, y, mID, md2);
      nearest(r->R, x, y, mID, md2);
    } else {
      nearest(r->R, x, y, mID, md2);
      nearest(r->L, x, y, mID, md2);
    }
  }
  int query(int x, int y) {
    int id = 1029384756;
    LL d2 = 102938475612345678LL;
    nearest(root, x, y, id, d2);
    return id;
  }
}tree;
```

6.10 Lower Concave Hull

```
/****
  maintain a "concave hull" that support the following
  1. insertion of a line
  2. query of height(y) on specific x on the hull
 ****/
/* set as needed */
typedef long double LD;
const LD eps=1e-9;
const LD inf=1e19;
class Seg {
 public:
  LD m,c,x1,x2; // y=mx+c
  bool flag;
  Seg(
    LD _m,LD _c,LD _x1=-inf,LD _x2=inf,bool _flag=0)
    :m(_m),c(_c),x1(_x1),x2(_x2),flag(_flag) {}
  LD evaly(LD x) const {
    return m*x+c;
  }
  const bool operator<(LD x) const {
    return x2-eps<x;
  }
  const bool operator<(const Seg &b) const {
    if(flag||b.flag) return *this<b.x1;
    return m+eps<b.m;
  }
};
class LowerConcaveHull { // maintain a hull like: \__/
 public:
  set<Seg> hull;
  /* functions */
  LD xintersection(Seg a,Seg b) {
    return (a.c-b.c)/(b.m-a.m);
  }
  inline set<Seg>::iterator replace(set<Seg> &
      hull,set<Seg>::iterator it,Seg s) {
    hull.erase(it);
    return hull.insert(s).first;
  }
  void insert(Seg s) {
    // insert a line and update hull
    set<Seg>::iterator it=hull.find(s);
    // check for same slope
    if(it!=hull.end()) {
      if(it->c+eps>=s.c) return;
      hull.erase(it);
    }
    // check if below whole hull
    it=hull.lower_bound(s);
    if(it!=hull.end()&&
       s.evaly(it->x1)<=it->evaly(it->x1)+eps) return;
    // update right hull
    while(it!=hull.end()) {
      LD x=xintersection(s,*it);
      if(x>=it->x2-eps) hull.erase(it++);
      else {
        s.x2=x;
        it=replace(hull,it,Seg(it->m,it->c,x,it->x2));
        break;
      }
    }
    // update left hull
    while(it!=hull.begin()) {
      LD x=xintersection(s,*(--it));
      if(x<=it->x1+eps) hull.erase(it++);
      else {
        s.x1=x;
        it=replace(hull,it,Seg(it->m,it->c,it->x1,x));
        break;
      }
    }
    // insert s
    hull.insert(s);
  }
  void insert(LD m,LD c) { insert(Seg(m,c)); }
  LD query(LD x) { // return y @ given x
    set<Seg>::iterator it =
      hull.lower_bound(Seg(0.0,0.0,x,x,1));
    return it->evaly(x);
  }
};
```

6.11 Min Enclosing Circle

```
struct Mec{
  // return pair of center and r
  static const int N = 101010;
  int n;
  Pt p[ N ], cen;
  double r2;
  void init( int _n , Pt _p[] ){
    n = _n;
    memcpy( p , _p , sizeof(Pt) * n );
  }
  double sqr(double a){ return a*a; }
  Pt center(Pt p0, Pt p1, Pt p2) {
    Pt a = p1-p0;
    Pt b = p2-p0;
    double c1=norm2( a ) * 0.5;
    double c2=norm2( b ) * 0.5;
    double d = a ^ b;
    double x = p0.X + (c1 * b.Y - c2 * a.Y) / d;
    double y = p0.Y + (a.X * c2 - b.X * c1) / d;
    return Pt(x,y);
  }
  pair<Pt,double> solve(){
    random_shuffle(p,p+n);
    r2=0;
    for (int i=0; i<n; i++){
      if (norm2(cen-p[i]) <= r2) continue;
      cen = p[i];
      r2 = 0;
      for (int j=0; j<i; j++){
        if (norm2(cen-p[j]) <= r2) continue;
        cen=Pt((p[i].X+p[j].X)/2,(p[i].Y+p[j].Y)/2);
        r2 = norm2(cen-p[j]);
        for (int k=0; k<j; k++){
          if (norm2(cen-p[k]) <= r2) continue;
          cen = center(p[i],p[j],p[k]);
          r2 = norm2(cen-p[k]);
        }
      }
    }
    return {cen,sqrt(r2)};
  }
} mec;
```

6.12 Heart of Triangle

```
Pt inCenter( Pt &A,  Pt &B,  Pt &C) { // 内心
  double a = norm(B-C), b = norm(C-A), c = norm(A-B);
  return (A * a + B * b + C * c) / (a + b + c);
}
Pt circumCenter( Pt &a,  Pt &b,  Pt &c) { // 外心
  Pt bb = b - a, cc = c - a;
  double db=norm2(bb), dc=norm2(cc), d=2*(bb ^ cc);
  return a-Pt(bb.Y*dc-cc.Y*db, cc.X*db-bb.X*dc) / d;
}
Pt othroCenter( Pt &a,  Pt &b,  Pt &c) { // 垂心
  Pt ba = b - a, ca = c - a, bc = b - c;
  double Y = ba.Y * ca.Y * bc.Y,
    A = ca.X * ba.Y - ba.X * ca.Y,
    x0= (Y+ca.X*ba.Y*b.X-ba.X*ca.Y*c.X) / A,
    y0= -ba.X * (x0 - c.X) / ba.Y + ca.Y;
  return Pt(x0, y0);
}
```

7 String

7.1 Knuth-Morris-Pratt Algorithm

```
// test with CF 471 D
template< typename T >
vector< int > KMP( vector< T > target, vector< T >
    pattern ) {
  vector< int > match;
  if ( pattern.size() > target.size() ) return match;
  vector< int > failure_function( (int)target.size(),
      -1 );
  for ( int i = 1, j = failure_function[ 0 ] = -1; i <
      (int)pattern.size(); ++i ) {
    while ( j >= 0 and pattern[ j + 1 ] != pattern[ i ]
         )
      j = failure_function[ j ];
    if ( pattern[ j + 1 ] == pattern[ i ] ) j++;
    failure_function[ i ] = j;
    // KMP
    int pos = i, prv = failure_function[ pos ];
    while ( pos + 1 < (int)pattern.size() and pattern[
        pos + 1 ] == pattern[ prv + 1 ] ) {
      if ( failure_function[ pos ] == -1 ) break;
      pos = prv;
      prv = failure_function[ prv ];
    }
    failure_function[ i ] = prv;
  }
  for ( int i = 0, j = -1; i < (int)target.size(); ++i
      ) {
    while ( j >= 0 and pattern[ j + 1 ] != target[ i ]
         )
      j = failure_function[ j ];
    if ( pattern[ j + 1 ] == target[ i ] ) j++;
    if ( j == (int)pattern.size() - 1 ) {
      match.push_back( i - pattern.size() + 1 );
      j = failure_function[ j ];
    }
  }
  return match;
}
```

7.2 Z Value

```
void Z_value( string& s, vector< int >& z ) {
  z.resize( s.size() );
  int i, j, left, right, len = s.size();
  left = right = 0; z[ 0 ] = len;
  for ( i = 1; i < (int)s.size(); ++i ) {
    j = max( min( z[ i - left ], right - i ), 0 );
    for( ; i + j < len && s[ i + j ] == s[ j ]; ++j );
    z[ i ] = j;
    if( i + z[ i ] > right ) {
      left = i;
      right = i + z[ i ];
    }
  }
}
```

7.3 Z Value Palindrome

```
const int N = 1000 + 10;
int len, zv[ N * 2 ];
char op[ N * 2 ];
string ip;
int main(){
  cin >> ip; len = ip.size();
  int l2 = len * 2 + 1;
  for ( int i = 0; i < l2; ++i ) {
    if ( i & 1 ) op[ i ] = ip[ i / 2 ];
    else op[ i ] = '@';
  }
  int l = 0, r = 0;
  zv[ 0 ] = 1;
  for ( int i = 1; i < l2; ++i ) {
    if ( i > r ) {
      l = r = i;
      while ( l > 0 and r < l2 - 1 and op[ l - 1 ] ==
          op[ r + 1 ] )
        --l, ++r;
      zv[ i ] = r - l + 1;
    }
    else {
      int md = ( l + r ) / 2, j = md + md - i;
      zv[ i ] = zv[ j ];
      int q = zv[ i ] / 2, nr = i + q;
      if ( nr == r ) {
        l = i + i - r;
        while ( l > 0 and r < l2 - 1 and op[ l - 1 ] ==
             op[ r + 1 ] )
          --l, ++r;
        zv[ i ] = r - l + 1;
      }
      else if ( nr > r )
        zv[ i ] = ( r - i ) * 2 + 1;
    }
  }
}
```

7.4 Suffix Array

```
class SuffixArray {
  public:
    static const int N = 1e5 + 10;
    string st;
    int SA[ N ], RA[ N ], tempSA[ N ], tempRA[ N ],
        counting[ N ], n;
    int LCP[ N ], PLCP[ N ], Phi[ N ];

    void build( string& s ) { st = s + '.'; n = st.size
        (); constructSA(); constructLCP(); }
    void countingSort( int k ){
      int maxRange = max( 260, n );
      memset( counting, 0, sizeof counting );
      for ( int i = 0; i < n; i++ )
        counting[ i + k < n ? RA[ i + k ] : 0 ]++;

      int sum = 0;
      for ( int i = 0; i < maxRange; ++i ) {
        int temp = counting[ i ];
        counting[ i ] = sum;
        sum += temp;
      }

      for ( int i = 0 ; i < n; ++i )
        tempSA[ counting[ SA[ i ] + k < n ? RA[ SA[ i ]
             + k ] : 0 ]++ ] = SA[ i ];
      for ( int i = 0; i < n; ++i )
        SA[ i ] = tempSA[ i ];
    }
    void constructSA() {
      for ( int i = 0; i < n; ++i ) {
        SA[ i ] = i;
        RA[ i ] = st[ i ];
      }

      for ( int k = 1; k < n; k <<= 1 ) {
        countingSort( k ); countingSort( 0 );
        int rank;
        tempRA[ SA[ 0 ] ] = rank = 0;
        for ( int i = 1; i < n; ++i )
          tempRA[ SA[ i ] ] = ( RA[ SA[ i - 1 ] ] == RA
              [ SA[ i ] ] && RA[ SA[ i - 1 ] + k ] ==
              RA[ SA[ i ] + k ] ) ? rank : ++rank;
        for ( int i = 0; i < n; ++i ) RA[ i ] = tempRA[
             i ];
        if ( RA[ SA[ n - 1 ] ] == n - 1 ) break;
      }
    }
    void constructLCP() {
      Phi[ SA[ 0 ] ] = -1;
      for ( int i = 1; i < n; ++i )
        Phi[ SA[ i ] ] = SA[ i - 1 ];
      int L = 0;
      for ( int i = 0; i < n; ++i ) {
        if ( Phi[ i ] == -1 ) {
          PLCP[ i ] = 0;
          continue;
        }
        while ( st[ i + L ] == st[ Phi[ i ] + L ] ) L
            ++;
        PLCP[ i ] = L;
        L = max( L - 1, 0 );
      }
      for ( int i = 0; i < n; ++i ) LCP[ i ] = PLCP[ SA
          [ i ] ];
    }
};
```

7.5 Palindrome Tree

```
const int MAXN = 200010;
struct PalT{
  struct Node{
    int nxt[ 33 ] , len , fail;
    ll cnt;
  };
  int tot , lst;
  Node nd[ MAXN * 2 ];
  char* s;
  int newNode( int l , int _fail ){
    int res = ++tot;
    memset( nd[ res ].nxt , 0 , sizeof nd[ res ].nxt );
    nd[ res ].len = l;
    nd[ res ].cnt = 0;
    nd[ res ].fail = _fail;
    return res;
  }
  void push( int p ){
    int np = lst;
    int c = s[ p ] - 'a';
    while( p - nd[ np ].len - 1 < 0
        || s[ p ] != s[ p - nd[ np ].len - 1 ] )
      np = nd[ np ].fail;

    if( nd[ np ].nxt[ c ] ){
      nd[ nd[ np ].nxt[ c ] ].cnt++;
      lst = nd[ np ].nxt[ c ];
      return ;
    }
    int nq = newNode( nd[ np ].len + 2 , 0 );
    nd[ nq ].cnt++;
    nd[ np ].nxt[ c ] = nq;
    lst = nq;
    if( nd[ nq ].len == 1 ){
      nd[ nq ].fail = 2;
      return ;
    }
    int tf = nd[ np ].fail;
    while( p - nd[ tf ].len - 1 < 0
        || s[ p ] != s[ p - nd[ tf ].len - 1 ] )
      tf = nd[ tf ].fail;

    nd[ nq ].fail = nd[ tf ].nxt[ c ];
    return ;
  }
  void init( char* _s ){
    s = _s;
    tot = 0;
    newNode( -1 , 1 );
    newNode( 0 , 1 );
    lst = 2;
    for( int i = 0 ; s[ i ] ; i++ )
      push( i );
  }
  void yutruli(){
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
    REPD( i , tot , 1 )
      nd[ nd[ i ].fail ].cnt += nd[ i ].cnt;
    nd[ 1 ].cnt = nd[ 2 ].cnt = 0ll;
  }
} pA;
int main(){ pA.init( sa ); }
```

7.6 Suffix Automata

```
const int MAXM = 1000010;
struct SAM{
  int tot, root, lst, mom[MAXM], mx[MAXM];
  int acc[MAXM], nxt[MAXM][33];
  int newNode(){
    int res = ++tot;
    fill(nxt[res], nxt[res]+33, 0);
    mom[res] = mx[res] = acc[res] = 0;
    return res;
  }
  void init(){
    tot = 0;
    root = newNode();
    mom[root] = 0, mx[root] = 0;
    lst = root;
  }
  void push(int c){
    int p = lst;
    int np = newNode();
    mx[np] = mx[p]+1;
    for(; p && nxt[p][c] == 0; p = mom[p])
      nxt[p][c] = np;
    if(p == 0) mom[np] = root;
    else{
      int q = nxt[p][c];
      if(mx[p]+1 == mx[q]) mom[np] = q;
      else{
        int nq = newNode();
        mx[nq] = mx[p]+1;
        for(int i = 0; i < 33; i++)
          nxt[nq][i] = nxt[q][i];
        mom[nq] = mom[q];
        mom[q] = nq;
        mom[np] = nq;
        for(; p && nxt[p][c] == q; p = mom[p])
          nxt[p][c] = nq;
      }
    }
    lst = np;
  }
  void push(char *str){
    for(int i = 0; str[i]; i++)
      push(str[i]-'a'+1);
  }
} sam;
```

7.7 AC Automata

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

constexpr int sizz(2000 * 1000), sigma(62);
int nx[sizz][sigma], fl[sizz], spt;
void clear_ac() { spt = 1; }
int newnode() {
  fill_n(nx[spt], sigma, 0);
  return spt++;
}
int add(const char *s, int Z) {
  for(int i(0); s[i]; ++i) {
    int c(s[i] - 1);
    if (nx[Z][c] == 0) nx[Z][c] = newnode();
    Z = nx[Z][c];
  }
  return Z;
}
int q[sizz], qe; // save BFS order
void make_fl(int root) {
  fl[root] = qe = 0;
  q[qe++] = root;
  for(int qs = 0; qs < qe;) {
    int X = q[qs++];
    for(int c = 0; c < sigma; ++c) {
      if (nx[X][c]) {
        int Xc = nx[X][c], Z = fl[X];
        while (Z && nx[Z][c] == 0)
          Z = fl[Z];
        int d = Z ? nx[Z][c] : root;
        fl[Xc] = d;
        q[qe++] = Xc;
      }
    }
  }
}
int cnt[sizz];
void walk(const char *s, int root) {
  int Z = root;
  for(int i = 0; s[i]; ++i) {
    int c = s[i] - 1;
    while (Z && nx[Z][c] == 0)
      Z = fl[Z];
    Z = Z ? nx[Z][c] : root;
    ++cnt[Z];
  }
  for (int i = qe-1; i >= 0; --i)
    cnt[fl[q[i]]] += cnt[q[i]];
}
char mapper(char c) {
  if (islower(c)) return c - 'a' + 1;
  if (isupper(c)) return c - 'A' + 27;
  return c - '0' + 53;
}
char S[2048], M[1000514];
int main() {
  int n, op[1024];
  scanf("%s %d", M, &n);
  transform(M, M+strlen(M), M, mapper);
  clear_ac();
  int root = newnode();
  for(int i = 0; i < n; ++i) {
    scanf("%s", S);
    transform(S, S + strlen(S), S, mapper);
    op[i] = add(S, root);
  }
  make_fl(root);
  walk(M, root);
  for(int i = 0; i < n; ++i)
    printf("%d\n", cnt[op[i]]);
}
```

7.8 Smallest Rotation

```
string mcp(string s){
  int n = s.length();
  s += s;
  int i=0, j=1;
  while (i<n && j<n){
    int k = 0;
    while (k < n && s[i+k] == s[j+k]) k++;
    if (s[i+k] <= s[j+k]) j += k+1;
    else i += k+1;
    if (i == j) j++;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```