Contents

1 Basic 1
1.1 vimrc 1
1.2 IncreaseStackSize 1
1.3 Pragma optimization 1
1.4 Debugger 1
1.5 Quick Random 2
1.6 IO Optimization 2

2 Data Structure 2
2.1 Bigint 2
2.2 Dark Magic 3
2.3 SkewHeap 3
2.4 Disjoint Set 3
2.5 Treap 4
2.6 SparseTable 4
2.7 Linear Basis 4

3 Graph 4
3.1 BCC Edge 4
3.2 BCC Vertex 5
3.3 Bipartite Matching 5
3.4 Minimum Cost Maximum Flow 5
3.5 General Graph Matching 6
3.6 MaximumFlow 6
3.7 Kuhn Munkres 6
3.8 2-SAT 7
3.9 Lowbit Decomposition 7
3.10 MaxClique 8
3.11 Min-Cut 8

4 Math 8
4.1 Prime Table 8
4.2 $\lfloor \frac{n}{i} \rfloor$ Enumeration 9
4.3 ax+by=gcd 9
4.4 Pollard Rho 9
4.5 Pi Count (Linear Sieve) 9
4.6 Range Sieve 9
4.7 Miller Rabin 9
4.8 Inverse Element 9
4.9 Euler Phi Function 10
4.10 Gauss Elimination 10
4.11 Fast Fourier Transform 10
4.12 Chinese Remainder 10
4.13 Berlekamp Massey 11
4.14 NTT 11
4.15 FWT 11
4.16 DiscreteLog 11
4.17 Quadratic residue 11

5 Geometry 12
5.1 Point Class 12
5.2 Circle Class 12
5.3 Line Class 12
5.4 Triangle Circumcentre 13
5.5 2D Convex Hull 13
5.6 2D Farthest Pair 13
5.7 2D Cosest Pair 13
5.8 SimulateAnnealing 13
5.9 Ternary Search on Integer 14
5.10 Minimum Covering Circle 14
5.11 KDTree (Nearest Point) 14

6 Stringology 14
6.1 Hash 14
6.2 Suffix Array 15
6.3 Aho-Corasick Algorithm 15
6.4 Suffix Automaton 15
6.5 KMP 16
6.6 Z value 16
6.7 Lexicographically Smallest Rotation 16
6.8 BWT 16

7 Misc 16
7.1 Degree Sequence Validity 16
7.1.1 Erdős–Gallai Theorem 16
7.1.2 Havel–Hakimi algorithm 16
7.2 MaximumEmptyRect 17
7.3 DP-opt Condition 17
7.3.1 totally monotone (concave/convex) 17
7.3.2 monge condition (concave/convex) 17
7.4 Convex 1D/1D DP 17
7.5 Josephus Problem 17

1 Basic

1.1 vimrc

```
se nu rnu bs=2 ru mouse=a cin et ts=4 sw=4 sts=4
syn on
filetype indent on
inoremap {<CR> {<CR>}<Esc>O
```

1.2 IncreaseStackSize

```
//stack resize(change esp to rsp if 64-bit system)
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
// craziest way
static void run_stack_sz(void(*func)(),size_t stsize){
  char *stack, *send;
  stack=(char *)malloc(stsize);
  send=stack+stsize-16;
  send=(char *)((uintptr_t)send/16*16);
  asm volatile(
    "mov %%rsp, (%0)\n"
    "mov %0, %%rsp\n"
    :
    : "r" (send));
  func();
  asm volatile(
    "mov (%0), %%rsp\n"
    :
    : "r" (send));
  free(stack);
}
```

1.3 Pragma optimization

```
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

1.4 Debugger

```
#! /usr/bin/env python3
import subprocess as sp
os_name = __import__('platform').system()
cmd,prefix = [],""
if os_name == 'Windows':
  cmd=["cmd", "/C"]
else:
  cmd = ["bash", "-c"]
  prefix = "./"
def GetTestData(exe):
  myout=sp.check_output(cmd+["%s%s"%(prefix, exe)])
  return myout.decode("utf8")
def Judge(a,b,testdata):
  f = open("test.in", "w+")
  f.write(testdata)
  f.close()
  c=sp.check_output(cmd+["%s%s<test.in"%(prefix, a)])
  d=sp.check_output(cmd+["%s%s<test.in"%(prefix, b)])
  if not c == d:
    print("answer: %s"%c.decode("utf8"),end="")
    print("output: %s"%d.decode("utf8"),end="")
    print("WA!")
    return False
  return True
if __name__ == '__main__':
  cnt = 0
  isOK = True
  while isOK:
    cnt += 1
    print(cnt)
    isOK=Judge("sol", "mysol", GetTestData("gen"))
```

1.5 Quick Random

```
template<class T,T x1,T x2,T x3,int y1,int y2,int y3>
struct PRNG {
  using S = typename std::make_signed<T>::type;
  T s;
  PRNG(T _s = 0) : s(_s) {}
  T next() {
    T z = (s += x1);
    z = (z ^ (z >> y1)) * x2;
    z = (z ^ (z >> y2)) * x3;
    return z ^ (z >> y3);
  }
  T next(T n) { return next() % n; }
  S next(S l, S r){return l+next(r-l+1);}
  T operator()() { return next(); }
  T operator()(T n) { return next(n); }
  S operator()(S l, S r) { return next(l, r); }
  static T gen(T s) { return PRNG(s)(); }
  template<class U>
  void shuffle(U first,U last){
    size_t n=last-first;
    for(size_t i=0;i<n;i++)
      swap(first[i],first[next(i+1)]);
  }
};
using R32=PRNG<uint32_t,0x9E3779B1,0x85EBCA6B,
0xC2B2AE35,16,13,16>;
R32 r32;
using R64=PRNG<uint64_t,0x9E3779B97F4A7C15,
0xBF58476D1CE4E5B9,0x94D049BB133111EB,30,27,31>;
R64 r64;
```

1.6 IO Optimization

```
static inline int gc() {
  static char buf[ 1 << 20 ], *p = buf, *end = buf;
  if ( p == end ) {
    end = buf + fread( buf, 1, 1 << 20, stdin );
    if ( end == buf ) return EOF;
    p = buf;
  }
  return *p++;
}
template < typename T >
static inline bool gn( T &_ ) {
  register int c = gc(); register T __ = 1; _ = 0;
  while(('0'>c||c>'9') && c!=EOF && c!='-') c = gc();
  if(c == '-') { __ = -1; c = gc(); }
  if(c == EOF) return false;
  while('0'<=c&&c<='9') _ = _ * 10 + c - '0', c = gc();
  _ *= __;
  return true;
}
template < typename T, typename ...Args >
static inline bool gn( T &x, Args &...args )
{ return gn(x) && gn(args...); }
```

2 Data Structure

2.1 Bigint

```
class BigInt{
  private:
    using lld = int_fast64_t;
    #define PRINTF_ARG PRIdFAST64
    #define LOG_BASE_STR "9"
    static constexpr lld BASE = 1000000000;
    static constexpr int LOG_BASE = 9;
    vector<lld> dig;
    bool neg;
    inline int len() const { return (int) dig.size(); }
    inline int cmp_minus(const BigInt& a) const {
      if(len() == 0 && a.len() == 0) return 0;
      if(neg ^ a.neg)return (int)a.neg*2 - 1;
      if(len()!=a.len())
        return neg?a.len()-len():len()-a.len();
      for(int i=len()-1;i>=0;i--) if(dig[i]!=a.dig[i])
        return neg?a.dig[i]-dig[i]:dig[i]-a.dig[i];
      return 0;
    }
    inline void trim(){
      while(!dig.empty()&&!dig.back())dig.pop_back();
      if(dig.empty()) neg = false;
    }
  public:
    BigInt(): dig(vector<lld>()), neg(false){}
    BigInt(lld a): dig(vector<lld>()){
      neg = a<0; dig.push_back(abs(a));
      trim();
    }
    BigInt(const string& a): dig(vector<lld>()){
      assert(!a.empty()); neg = (a[0]=='-');
      for(int i=((int)a.size())-1;i>=neg;i-=LOG_BASE){
        lld cur = 0;
        for(int j=min(LOG_BASE-1,i-neg);j>=0;j--)
          cur = cur*10+a[i-j]-'0';
        dig.push_back(cur);
      } trim();
    }
    inline bool operator<(const BigInt& a)const
      {return cmp_minus(a)<0;}
    inline bool operator<=(const BigInt& a)const
      {return cmp_minus(a)<=0;}
    inline bool operator==(const BigInt& a)const
      {return cmp_minus(a)==0;}
    inline bool operator!=(const BigInt& a)const
      {return cmp_minus(a)!=0;}
    inline bool operator>(const BigInt& a)const
      {return cmp_minus(a)>0;}
    inline bool operator>=(const BigInt& a)const
      {return cmp_minus(a)>=0;}
    BigInt operator-() const {
      BigInt ret = *this;
      ret.neg ^= 1;
      return ret;
    }
    BigInt operator+(const BigInt& a) const {
      if(neg) return -(-(*this)+(-a));
      if(a.neg) return (*this)-(-a);
      int n = max(a.len(), len());
      BigInt ret; ret.dig.resize(n);
      lld pro = 0;
      for(int i=0;i<n;i++) {
        ret.dig[i] = pro;
        if(i < a.len()) ret.dig[i] += a.dig[i];
        if(i < len()) ret.dig[i] += dig[i];
        pro = 0;
        if(ret.dig[i] >= BASE) pro = ret.dig[i]/BASE;
        ret.dig[i] -= BASE*pro;
      }
      if(pro != 0) ret.dig.push_back(pro);
      return ret;
    }
    BigInt operator-(const BigInt& a) const {
      if(neg) return -(-(*this) - (-a));
      if(a.neg) return (*this) + (-a);
      int diff = cmp_minus(a);
      if(diff < 0) return -(a - (*this));
      if(diff == 0) return 0;
      BigInt ret; ret.dig.resize(len(), 0);
      for(int i=0;i<len();i++) {
        ret.dig[i] += dig[i];
        if(i < a.len()) ret.dig[i] -= a.dig[i];
        if(ret.dig[i] < 0){
          ret.dig[i] += BASE;
          ret.dig[i+1]--;
        }
      }
      ret.trim();
      return ret;
    }
    BigInt operator*(const BigInt& a) const {
      if(!len()||!a.len()) return 0;
      BigInt ret; ret.dig.resize(len()+a.len()+1);
      ret.neg = neg ^ a.neg;
      for(int i=0;i<len();i++)
        for(int j=0;j<a.len();j++){
          ret.dig[i+j] += dig[i] * a.dig[j];
          if(ret.dig[i+j] >= BASE) {
            lld x = ret.dig[i+j] / BASE;
            ret.dig[i+j+1] += x;
            ret.dig[i+j] -= x * BASE;
          }
        }
      ret.trim();
      return ret;
    }
    BigInt operator/(const BigInt& a) const {
      assert(a.len());
      if(len() < a.len()) return 0;
      BigInt ret; ret.dig.resize(len()-a.len()+1);
      ret.neg = a.neg;
      for(int i=len()-a.len();i>=0;i--){
        lld l = 0, r = BASE;
        while(r-l > 1){
          lld mid = (l+r)>>1;
          ret.dig[i] = mid;
          if(ret*a<=(neg?-(*this):(*this))) l = mid;
          else r = mid;
        }
        ret.dig[i] = l;
      }
      ret.neg ^= neg; ret.trim();
      return ret;
    }
    BigInt operator%(const BigInt& a) const {
      return (*this) - (*this) / a * a;
    }
    friend BigInt abs(BigInt a){
      a.neg = 1; return a;
    }
    friend void swap(BigInt& a, BigInt& b){
      swap(a.dig, b.dig); swap(a.neg, b.neg);
    }
    friend istream& operator>>(istream& ss, BigInt& a){
      string s; ss >> s; a = s;
      return ss;
    }
    friend ostream& operator<<(ostream& ss, BigInt& a){
      if(a.len() == 0) return ss << '0';
      if(a.neg) ss << '-';
      ss << a.dig.back();
      for(int i=a.len()-2;i>=0;i--)
        ss<<setw(LOG_BASE)<<setfill('0')<<a.dig[i];
      return ss;
    }
    inline void print() const {
      if(len() == 0){putchar('0');return;}
      if(neg) putchar('-');
      printf("%" PRINTF_ARG, dig.back());
      for(int i=len()-2;i>=0;i--)
        printf("%0" LOG_BASE_STR PRINTF_ARG, dig[i]);
    }
    #undef PRINTF_ARG
    #undef LOG_BASE_STR
};
```

2.2 Dark Magic

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using __gnu_pbds::pairing_heap_tag;
using __gnu_pbds::binary_heap_tag;
using __gnu_pbds::binomial_heap_tag;
using __gnu_pbds::rc_binomial_heap_tag;
using __gnu_pbds::thin_heap_tag;
template<typename T>
using pbds_heap=__gnu_pbds::prioity_queue<T,less<T>,\
                                     pairing_heap_tag>;
using __gnu_pbds::rb_tree_tag;
using __gnu_pbds::ov_tree_tag;
using __gnu_pbds::splay_tree_tag;
template<typename T>
using ordered_set = __gnu_pbds::tree<T,\
__gnu_pbds::null_type,less<T>,rb_tree_tag,\
__gnu_pbds::tree_order_statistics_node_update>;
template<typename A,typename B>
using hTable1=__gnu_pbds::cc_hash_table<A,B>;
template<typename A,typename B>
using hTable2=__gnu_pbds::gp_hash_table<A,B>;
int main(){
  ordered_set<int> ss;
  ss.insert(1); ss.insert(5);
  assert(*ss.find_by_order(0)==1);
  assert(ss.order_of_key(-1)==0);
  pbds_heap pq1, pq2;
  pq1.push(1); pq2.push(2);
  pq1.join(pq2);
  assert(pq2.size()==0);
  auto it = pq1.push(87);
  pq1.modify(it, 19);
  return 0;
}
```

2.3 SkewHeap

```
template < typename T, typename cmp = less< T > >
class SkewHeap{
private:
  struct SkewNode{
    T x;
    SkewNode *lc, *rc;
    SkewNode( T a = 0 ) : x( a ), lc( 0 ), rc( 0 ) {}
  } *root;
  cmp CMP_;
  size_t count;
  SkewNode* Merge( SkewNode* a, SkewNode* b ) {
    if ( !a or !b ) return a ? a : b;
    if ( CMP_( a->x, b->x ) ) swap( a, b );
    a -> rc = Merge( a->rc, b );
    swap( a -> lc, a->rc );
    return a;
  }
public:
  SkewHeap(): root( 0 ), count( 0 ) {}
  size_t size() { return count; }
  bool empty() { return count == 0; }
  T top() { return root->x; }
  void clear(){ root = 0; count = 0; }
  void push ( const T& x ) {
    SkewNode* a = new SkewNode( x );
    count += 1; root = Merge( root, a );
  }
  void join( SkewHeap& a ) {
    count += a.count; a.count = 0;
    root = Merge( root, a.root );
  }
  void pop() {
    count--; root = Merge( root->lc, root->rc );
  }
  friend void swap( SkewHeap& a, SkewHeap& b ) {
    swap( a.root, b.root ); swap( a.count, b.count );
  }
};
```

2.4 Disjoint Set

```
class DJS{
private:
  vector< int > fa, sz, sv;
  vector< pair< int*, int > > opt;
  inline void assign( int *k, int v ) {
    opt.emplace_back( k, *k );
    *k = v;
  }
public:
  inline void init( int n ) {
    fa.resize( n ); iota( fa.begin(), fa.end(), 0 );
    sz.resize( n ); fill( sz.begin(), sz.end(), 1 );
    opt.clear();
  }
  int query( int x ) {
    return ( fa[ x ] == x ) ? x : query( fa[ x ] );
  }
  inline void merge( int a, int b ) {
    int af = query( a ), bf = query( b );
    if( af == bf ) return;
    if( sz[ af ] < sz[ bf ] ) swap( af, bf );
    assign( &fa[ bf ], fa[ af ] );
    assign( &sz[ af ], sz[ af ] + sz[ bf ] );
  }
  inline void save() {sv.push_back( (int)opt.size() );}
  inline void undo() {
    int ls = sv.back(); sv.pop_back();
    while ( ( int ) opt.size() > ls ) {
      pair< int*, int > cur = opt.back();
      *cur.first = cur.second;
      opt.pop_back();
    }
  }
};
```

2.5 Treap

```
namespace Treap{
  #define sz( x ) ( ( x ) ? ( ( x )->size ) : 0 )
  struct node{
    int size;
    uint32_t pri;
    node *lc, *rc;
    node() : size(0), pri(rand()), lc( 0 ), rc( 0 ) {}
    void pull() {
      size = 1;
      if ( lc ) size += lc->size;
      if ( rc ) size += rc->size;
    }
  };
  node* merge( node* L, node* R ) {
    if ( not L or not R ) return L ? L : R;
    if ( L->pri > R->pri ) {
      L->rc = merge( L->rc, R );
      L->pull();
      return L;
    } else {
      R->lc = merge( L, R->lc );
      R->pull();
      return R;
    }
  }
  void split_by_size( node*rt,int k,node*&L,node*&R ) {
    if ( not rt ) L = R = nullptr;
    else if( sz( rt->lc ) + 1 <= k ) {
      L = rt;
      split_by_size( rt->rc,k-sz(rt->lc)-1,L->rc,R );
      L->pull();
    } else {
      R = rt;
      split_by_size( rt->lc, k, L, R->lc );
      R->pull();
    }
  }
  #undef sz
}
```

2.6 SparseTable

```
template < typename T, typename Cmp_ = less< T > >
class SparseTable {
private:
  vector< vector< T > > tbl;
  vector< int > lg;
  T cv( T a, T b ) {
    return Cmp_()( a, b ) ? a : b;
  }
public:
  void init( T arr[], int n ) {
    // 0-base
    lg.resize( n + 1 );
    lg[ 0 ] = -1;
    for( int i=1 ; i<=n ; ++i ) lg[i] = lg[i>>1] + 1;
    tbl.resize( lg[n] + 1 );
    tbl[ 0 ].resize( n );
    copy( arr, arr + n, tbl[ 0 ].begin() );
    for ( int i = 1 ; i <= lg[ n ] ; ++ i ) {
      int len = 1 << ( i - 1 ), sz = 1 << i;
      tbl[ i ].resize( n - sz + 1 );
      for ( int j = 0 ; j <= n - sz ; ++ j )
        tbl[i][j] = cv(tbl[i-1][j], tbl[i-1][j+len]);
    }
  }
  T query( int l, int r ) {
    // 0-base [l, r)
    int wh = lg[ r - l ], len = 1 << wh;
    return cv( tbl[ wh ][ l ], tbl[ wh ][ r - len ] );
  }
};
```

2.7 Linear Basis

```
struct LinearBasis {
private:
  int n, sz;
  vector< llu > B;
  inline llu two( int x ){ return ( ( llu ) 1 ) << x; }
public:
  void init( int n_ ) {
    n = n_; B.clear();
    B.resize( n ); sz = 0;
  }
  void insert( llu x ) {
    // add x into B
    for ( int i = n-1; i >= 0 ; --i ) if( two(i) & x ){
      if ( B[ i ] ) x ^= B[ i ];
      else {
        B[ i ] = x; sz++;
        for ( int j = i - 1 ; j >= 0 ; -- j )
          if( B[ j ] && ( two( j ) & B[ i ] ) )
            B[ i ] ^= B[ j ];
        for (int j = i + 1 ; j < n ; ++ j )
          if ( two( i ) & B[ j ] )
            B[ j ] ^= B[ i ];
        break;
      }
    }
  }
  inline int size() { return sz; }
  bool check( llu x ) {
    // is x in span(B) ?
    for ( int i = n-1 ; i >= 0 ; --i ) if( two(i) & x )
      if( B[ i ] ) x ^= B[ i ];
      else return false;
    return true;
  }
  llu kth_small(llu k) {
    /** 1-base would always > 0 **/
    /** should check it **/
    /* if we choose at least one element
       but size(B)(vectors in B)==N(original elements)
       then we can't get 0 */
    llu ret = 0;
    for ( int i = 0 ; i < n ; ++ i ) if( B[ i ] ) {
      if( k & 1 ) ret ^= B[ i ];
      k >>= 1;
    }
    return ret;
  }
} base;
```

3 Graph

3.1 BCC Edge

```
class BCC{
private:
  vector< int > low, dfn;
  int cnt;
  vector< bool > bridge;
  vector< vector< PII > > G;
  void dfs( int w, int f ) {
    dfn[ w ] = cnt++;
    low[ w ] = dfn[ w ];
    for ( auto [ u, t ] : G[ w ] ) {
      if ( u == f ) continue;
      if ( dfn[ u ] != 0 ) {
        low[ w ] = min( low[ w ], dfn[ u ] );
      }else{
        dfs( u, w );
        low[ w ] = min( low[ w ], low[ u ] );
        if ( low[ u ] > dfn[ w ] ) bridge[ t ] = true;
      }
    }
  }
public:
  void init( int n, int m ) {
    G.resize( n );
    fill( G.begin(), G.end(), vector< PII >() );
    bridge.clear(); bridge.resize( m );
    low.clear(); low.resize( n );
    dfn.clear(); dfn.resize( n );
    cnt = 0;
  }
  void add_edge( int u, int v ) {
    // should check for multiple edge
    G[ u ].emplace_back( v, cnt );
    G[ v ].emplace_back( u, cnt ++ );
  }
  void solve(){ cnt = 1; dfs( 0, 0 ); }
  // the id will be same as insert order, 0-base
  bool is_bridge( int x ) { return bridge[ x ]; }
} bcc;
```

3.2 BCC Vertex

```
class BCC{
  private:
    int n, ecnt;
    vector< vector< pair< int, int > > > G;
    vector< int > low, dfn, id;
    vector< bool > vis, ap;
    void dfs( int u, int f, int d ) {
      int child = 0;
      dfn[ u ] = low[ u ] = d; vis[ u ] = true;
      for ( auto e : G[ u ] ) if ( e.first != f ) {
        if ( vis[ e.first ] ) {
          low[ u ] = min( low[ u ], dfn[ e.first ] );
        } else {
          dfs( e.first, u, d + 1 ); child ++;
          low[ u ] = min( low[ u ], low[ e.first ] );
          if ( low[ e.first ] >= d ) ap[ u ] = true;
        }
      }
      if ( u == f and child <= 1 ) ap[ u ] = false;
    }
    void mark( int u, int idd ) {
      // really??????????
      if ( ap[ u ] ) return;
      for ( auto e : G[ u ] )
        if( id[ e.second ] != -1 ) {
          id[ e.second ] = idd;
          mark( e.first, idd );
        }
    }
  public:
    void init( int n_ ) {
      ecnt = 0, n = n_;
      G.clear(); G.resize( n );
      low.resize( n ); dfn.resize( n );
      ap.clear(); ap.resize( n );
      vis.clear(); vis.resize( n );
    }
    void add_edge( int u, int v ) {
      G[ u ].emplace_back( v, ecnt );
      G[ v ].emplace_back( u, ecnt ++ );
    }
    void solve() {
      for ( int i = 0 ; i < n ; ++ i )
        if ( not vis[ i ] ) dfs( i, i, 0 );
      id.resize( ecnt );
      fill( id.begin(), id.end(), -1 );
      ecnt = 0;
      for ( int i = 0 ; i < n ; ++ i )
        if ( ap[ i ] ) for ( auto e : G[ i ] )
          if( id[ e.second ] != -1 ) {
            id[ e.second ] = ecnt;
            mark( e.first, ecnt ++ );
          }
    }
    int get_id( int x ) { return id[ x ]; }
    int count() { return ecnt; }
    bool is_ap( int u ) { return ap[ u ]; }
} bcc;
```

3.3 Bipartite Matching

```
class BipartiteMatching{
  private:
    vector<int> X[N], Y[N];
    int fX[N], fY[N], n;
    bitset<N> walked;
    bool dfs(int x){
      for(auto i:X[x]){
        if(walked[i])continue;
        walked[i]=1;
        if(fY[i]==-1||dfs(fY[i])){
          fY[i]=x;fX[x]=i;
          return 1;
        }
      }
      return 0;
    }
  public:
    void init(int _n){
      n=_n;
      for(int i=0;i<n;i++){
        X[i].clear();
        Y[i].clear();
        fX[i]=fY[i]=-1;
      }
      walked.reset();
    }
    void add_edge(int x, int y){
      X[x].push_back(y);
      Y[y].push_back(y);
    }
    int solve(){
      int cnt = 0;
      for(int i=0;i<n;i++){
        walked.reset();
        if(dfs(i)) cnt++;
      }
      // return how many pair matched
      return cnt;
    }
};
```

3.4 Minimum Cost Maximum Flow

```
class MiniCostMaxiFlow{
  using CapT = int;
  using WeiT = int64_t;
  using PCW = pair<CapT,WeiT>;
  static constexpr CapT INF_CAP = 1 << 30;
  static constexpr WeiT INF_WEI = 1LL<<60;
private:
  struct Edge{
    int to, back;
    WeiT wei;
    CapT cap;
    Edge() {}
    Edge(int a,int b,WeiT c,CapT d):
      to(a),back(b),wei(c),cap(d)
    {}
  };
  int ori, edd;
  vector<vector<Edge>> G;
  vector<int> fa, wh;
  vector<bool> inq;
  vector<WeiT> dis;
  PCW SPFA(){
    fill(inq.begin(),inq.end(),false);
    fill(dis.begin(),dis.end(),INF_WEI);
    queue<int> qq; qq.push(ori);
    dis[ori]=0;
    while(!qq.empty()){
      int u=qq.front();qq.pop();
      inq[u] = 0;
      for(int i=0;i<SZ(G[u]);++i){
        Edge e=G[u][i];
        int v=e.to;
        WeiT d=e.wei;
        if(e.cap<=0||dis[v]<=dis[u]+d)
          continue;
        dis[v]=dis[u]+d;
        fa[v]=u,wh[v]=i;
        if(inq[v]) continue;
        qq.push(v);
        inq[v]=1;
      }
    }
    if(dis[edd]==INF_WEI)
      return {-1,-1};
    CapT mw=INF_CAP;
    for(int i=edd;i!=ori;i=fa[i])
      mw=min(mw,G[fa[i]][wh[i]].cap);
    for (int i=edd;i!=ori;i=fa[i]){
      auto &eg=G[fa[i]][wh[i]];
      eg.cap-=mw;
      G[eg.to][eg.back].cap+=mw;
    }
    return {mw,dis[edd]};
  }
public:
  void init(int a,int b,int n){
    ori=a,edd=b;
    G.clear();G.resize(n);
    fa.resize(n);wh.resize(n);
    inq.resize(n); dis.resize(n);
  }
  void add_edge(int st,int ed,WeiT w,CapT c){
    G[st].emplace_back(ed,SZ(G[ed]),w,c);
    G[ed].emplace_back(st,SZ(G[st])-1,-w,0);
  }
  PCW solve(){
    /* might modify to
    cc += ret.first * ret.second
    or
    ww += ret.first * ret.second
    */
    CapT cc=0; WeiT ww=0;
    while(true){
      PCW ret=SPFA();
      if(ret.first==-1) break;
      cc+=ret.first;
      ww+=ret.second;
    }
    return {cc,ww};
  }
} mcmf;
```

3.5 General Graph 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;
```

3.6 MaximumFlow

```
class Dinic{
private:
  using CapT = int64_t;
  struct Edge{
    int to, rev;
    CapT cap;
  };
  int n, st, ed;
  vector<vector<Edge>> G;
  vector<int> lv;
  bool BFS(){
    fill(lv.begin(), lv.end(), -1);
    queue<int> bfs;
    bfs.push(st);
    lv[st] = 0;
    while(!bfs.empty()){
      int u = bfs.front(); bfs.pop();
      for(auto e: G[u]){
        if(e.cap <= 0 or lv[e.to]!=-1) continue;
        lv[e.to] = lv[u] + 1;
        bfs.push(e.to);
      }
    }
    return (lv[ed]!=-1);
  }
  CapT DFS(int u, CapT f){
    if(u == ed) return f;
    CapT ret = 0;
    for(auto& e: G[u]){
      if(e.cap <= 0 or lv[e.to]!=lv[u]+1) continue;
      CapT nf = DFS(e.to, min(f, e.cap));
      ret += nf; e.cap -= nf; f -= nf;
      G[e.to][e.rev].cap += nf;
      if(f == 0) return ret;
    }
    if(ret == 0) lv[u] = -1;
    return ret;
  }
public:
  void init(int n_, int st_, int ed_){
    n = n_, st = st_, ed = ed_;
    G.resize(n); lv.resize(n);
    fill(G.begin(), G.end(), vector<Edge>());
  }
  void add_edge(int u, int v, CapT c){
    G[u].push_back({v, (int)G[v].size(), c});
    G[v].push_back({u, ((int)G[u].size())-1, 0});
  }
  CapT max_flow(){
    CapT ret = 0;
    while(BFS()){
      CapT f = DFS(st, numeric_limits<CapT>::max());
      ret += f;
      if(f == 0) break;
    }
    return ret;
  }
} flow;
```

3.7 Kuhn Munkres

```
struct KM{
  static constexpr lld INF = 1LL<<60;
  lld w[N][N], lx[N], ly[N], slack[N];
  int match[N], n, vx[N], vy[N], step_;
  void init(int n_){
    n=n_,step=0;
    memset(w,0,sizeof(w));
    memset(lx,0,sizeof(lx));
    memset(ly,0,sizeof(ly));
    memset(slack,0,sizeof(slack));
    memset(match,0,sizeof(match));
    memset(vx,0,sizeof(vx));
    memset(vy,0,sizeof(vy));
  }
  void add_edge(int u,int v,lld w){w[u][v]=w;}
  bool dfs(int x) {
    vx[x] = step_;
    for (int i = 0; i < n; ++i) {
      if (vy[i]==step_) continue;
      if (lx[x] + ly[i] > w[x][i]) {
        slack[i] = min(slack[i], lx[x] + ly[i] - w[x][i
            ]);
        continue;
      }
      vy[i] = step_;
      if (match[i] == -1 || dfs(match[i])) {
        match[i] = x;
        return true;
      }
    }
    return false;
  }
  lld solve() {
    fill_n(match, n, -1);
    fill_n(lx, n, -INF);
    fill_n(ly, n, 0);
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j)
        lx[i] = max(lx[i], w[i][j]);
    for (int i = 0; i < n; ++i) {
      fill_n(slack, n, INF);
      while (true) {
        step_++;
        if (dfs(i)) break;
        lld dlt = INF;
        for (int j = 0; j < n; ++j) if (vy[j] != step_)
          dlt = min(dlt, slack[j]);
        for (int j = 0; j < n; ++j) {
          if (vx[j]==step_) lx[j] -= dlt;
          if (vy[j]==step_) ly[j] += dlt;
          else slack[j] -= dlt;
        }
      }
    }
    lld res = 0;
    for (int i = 0; i < n; ++i) res += w[match[i]][i];
    return res;
  }
} km;
```

3.8 2-SAT

```
class TwoSat{
  private:
    int n;
    vector<vector<int>> rG,G,sccs;
    vector<int> ord,idx;
    vector<bool> vis,result;
    void dfs(int u){
      vis[u]=true;
      for(int v:G[u])
        if(!vis[v])
          dfs(v);
      ord.push_back(u);
    }
    void rdfs(int u){
      vis[u]=false;
      idx[u]=sccs.size()-1;
      sccs.back().push_back(u);
      for(int v:rG[u])
        if(vis[v])
          rdfs(v);
    }
  public:
    void init(int n_){
      n=n_;
      G.clear();
      G.resize(n);
      rG.clear();
      rG.resize(n);
      sccs.clear();
      ord.clear();
      idx.resize(n);
      result.resize(n);
    }
    void add_edge(int u,int v){
      G[u].push_back(v);
      rG[v].push_back(u);
    }
    void orr(int x,int y){
      if ((x^y)==1)return;
      add_edge(x^1,y);
      add_edge(y^1,x);
    }
    bool solve(){
      vis.clear();
      vis.resize(n);
      for(int i=0;i<n;++i)
        if(not vis[i])
          dfs(i);
      reverse(ord.begin(),ord.end());
      for (int u:ord){
        if(!vis[u])
          continue;
        sccs.push_back(vector<int>());
        rdfs(u);
      }
      for(int i=0;i<n;i+=2)
        if(idx[i]==idx[i+1])
          return false;
      vector<bool> c(sccs.size());
      for(size_t i=0;i<sccs.size();++i){
        for(size_t j=0;j<sccs[i].size();++j){
          result[sccs[i][j]]=c[i];
          c[idx[sccs[i][j]^1]]=!c[i];
        }
      }
      return true;
    }
    bool get(int x){return result[x];}
    inline int get_id(int x){return idx[x];}
    inline int count(){return sccs.size();}
} sat2;
```

3.9 Lowbit Decomposition

```
class LowbitDecomp{
private:
  int time_, chain_, LOG_N;
  vector< vector< int > > G, fa;
  vector< int > tl, tr, chain, chain_st;
  // chain_ : number of chain
  // tl, tr[ u ] : subtree interval in the seq. of u
  // chain_st[ u ] : head of the chain contains u
  // chian[ u ] : chain id of the chain u is on
  inline int lowbit( int x ) {
    return x & ( -x );
  }
  void predfs( int u, int f ) {
    chain[ u ] = 0;
    for ( int v : G[ u ] ) {
      if ( v == f ) continue;
      predfs( v, u );
      if( lowbit( chain[ u ] ) < lowbit( chain[ v ] ) )
        chain[ u ] = chain[ v ];
    }
    if ( not chain[ u ] )
      chain[ u ] = chain_ ++;
  }
  void dfschain( int u, int f ) {
    fa[ u ][ 0 ] = f;
    for ( int i = 1 ; i < LOG_N ; ++ i )
      fa[ u ][ i ] = fa[ fa[ u ][ i - 1 ] ][ i - 1 ];
    tl[ u ] = time_++;
    if ( not chain_st[ chain[ u ] ] )
      chain_st[ chain[ u ] ] = u;
    for ( int v : G[ u ] )
      if ( v != f and chain[ v ] == chain[ u ] )
        dfschain( v, u );
    for ( int v : G[ u ] )
      if ( v != f and chain[ v ] != chain[ u ] )
        dfschain( v, u );
    tr[ u ] = time_;
  }
  inline bool anc( int u, int v ) {
    return tl[ u ] <= tl[ v ] \
      and tr[ v ] <= tr[ u ];
  }
public:
  inline int lca( int u, int v ) {
    if ( anc( u, v ) ) return u;
    for ( int i = LOG_N - 1 ; i >= 0 ; -- i )
      if ( not anc( fa[ u ][ i ], v ) )
        u = fa[ u ][ i ];
    return fa[ u ][ 0 ];
  }
  void init( int n ) {
    n ++;
    for ( LOG_N = 0 ; ( 1 << LOG_N ) < n ; ++ LOG_N );
    fa.clear();
    fa.resize( n, vector< int >( LOG_N ) );
    G.clear(); G.resize( n );
    tl.clear(); tl.resize( n );
    tr.clear(); tr.resize( n );
    chain.clear(); chain.resize( n );
    chain_st.clear(); chain_st.resize( n );
  }
  void add_edge( int u , int v ) {
    // 1-base
    G[ u ].push_back( v );
    G[ v ].push_back( u );
  }
  void decompose(){
    chain_ = 1;
    predfs( 1, 1 );
    time_ = 0;
    dfschain( 1, 1 );
  }
  PII get_inter( int u ) { return {tl[ u ], tr[ u ]}; }
  vector< PII > get_path( int u , int v ){
    vector< PII > res;
    int g = lca( u, v );
    while ( chain[ u ] != chain[ g ] ) {
      int s = chain_st[ chain[ u ] ];
      res.emplace_back( tl[ s ], tl[ u ] + 1 );
      u = fa[ s ][ 0 ];
    }
    res.emplace_back( tl[ g ], tl[ u ] + 1 );
    while ( chain[ v ] != chain[ g ] ) {
      int s = chain_st[ chain[ v ] ];
      res.emplace_back( tl[ s ], tl[ v ] + 1 );
      v = fa[ s ][ 0 ];
    }
    res.emplace_back( tl[ g ] + 1, tl[ v ] + 1 );
    return res;
    /* res : list of intervals from u to v
     * ( note only nodes work, not edge )
     * usage :
     * vector< PII >& path = tree.get_path( u , v )
     * for( auto [ l, r ] : path ) {
     *   0-base [ l, r )
     * }
     */
  }
} tree;
```

3.10 MaxClique

```
#define N 111
struct MaxClique{ // 0-base
  typedef bitset< N > Int;
  Int linkto[ N ] , v[ N ];
  int n;
  void init( int _n ){
    n = _n;
    for( int i = 0 ; i < n ; i ++ ){
      linkto[ i ].reset();
      v[ i ].reset();
    }
  }
  void add_edge( int a , int b ){
    v[ a ][ b ] = v[ b ][ a ] = 1;
  }
  int popcount(const Int& val)
  { return val.count(); }
  int lowbit(const Int& val)
  { return val._Find_first(); }
  int ans , stk[ N ];
  int id[ N ] , di[ N ] , deg[ N ];
  Int cans;
  void maxclique(int elem_num, Int candi){
    if(elem_num > ans){
      ans = elem_num;
      cans.reset();
      for( int i = 0 ; i < elem_num ; i ++ )
        cans[ id[ stk[ i ] ] ] = 1;
    }
    int potential = elem_num + popcount(candi);
    if(potential <= ans) return;
    int pivot = lowbit(candi);
    Int smaller_candi = candi & (~linkto[pivot]);
    while(smaller_candi.count() && potential>ans){
      int next = lowbit(smaller_candi);
      candi[ next ] = !candi[ next ];
      smaller_candi[next] = !smaller_candi[next];
      potential --;
      if(next!=pivot
        &&!(smaller_candi&linkto[next]).count())
          continue;
      stk[elem_num] = next;
      maxclique(elem_num+1, candi&linkto[next]);
    }
  }
  int solve(){
    for( int i = 0 ; i < n ; i ++ ){
      id[ i ] = i;
      deg[ i ] = v[ i ].count();
    }
    sort( id , id + n , [&](int id1, int id2){
          return deg[id1] > deg[id2]; } );
    for( int i = 0 ; i < n ; i ++ )
      di[ id[ i ] ] = i;
    for( int i = 0 ; i < n ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        if( v[ i ][ j ] )
          linkto[ di[ i ] ][ di[ j ] ] = 1;
    Int cand; cand.reset();
    for( int i = 0 ; i < n ; i ++ )
      cand[ i ] = 1;
    ans = 1;
    cans.reset(); cans[ 0 ] = 1;
    maxclique(0, cand);
    return ans;
  }
} solver;
```

3.11 Min-Cut

```
const int maxn = 500 + 5;
int w[maxn][maxn], g[maxn];
bool v[maxn], del[maxn];

void add_edge(int x, int y, int c) {
    w[x][y] += c;
    w[y][x] += c;
}

pair<int, int> phase(int n) {
    memset(v, false, sizeof(v));
    memset(g, 0, sizeof(g));
    int s = -1, t = -1;
    while (true) {
        int c = -1;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            if (c == -1 || g[i] > g[c]) c = i;
        }
        if (c == -1) break;
        v[c] = true;
        s = t, t = c;
        for (int i = 0; i < n; ++i) {
            if (del[i] || v[i]) continue;
            g[i] += w[c][i];
        }
    }
    return make_pair(s, t);
}

int mincut(int n) {
    int cut = 1e9;
    memset(del, false, sizeof(del));
    for (int i = 0; i < n - 1; ++i) {
        int s, t; tie(s, t) = phase(n);
        del[t] = true;
        cut = min(cut, g[t]);
        for (int j = 0; j < n; ++j) {
            w[s][j] += w[t][j];
            w[j][s] += w[j][t];
        }
    }
    return cut;
}
```

4 Math

4.1 Prime Table

1002939109, 1020288887, 1028798297, 1038684299,
1041211027, 1051762951, 1058585963, 1063020809,
1147930723, 1172520109, 1183835981, 1187659051,
1241251303, 1247184097, 1255940849, 1272759031,

$1287027493, 1288511629, 1294632499, 1312650799, 1868732623, 1884198443, 1884616807, 1885059541, 1909942399, 1914471137, 1923951707, 1925453197, 1979612177, 1980446837, 1989761941, 2007826547, 2008033571, 2011186739, 2039465081, 2039728567, 2093735719, 2116097521, 2123852629, 2140170259, 3148478261, 3153064147, 3176351071, 3187523093, 3196772239, 3201312913, 3203063977, 3204840059, 3210224309, 3213032591, 3217689851, 3218469083, 3219857533, 3231880427, 3235951699, 3273767923, 3276188869, 3277183181, 3282463507, 3285553889, 3319309027, 3327005333, 3327574903, 3341387953, 3373293941, 3380077549, 3380892997, 3381118801$

4.2 $\lfloor \frac{n}{i} \rfloor$ Enumeration

$T_0 = 1, T_{i+1} = \lfloor \frac{n}{\lfloor \frac{n}{T_i+1} \rfloor} \rfloor$

4.3 ax+by=gcd

```
// ax+ny = 1, ax+ny == ax == 1 (mod n)
void exgcd(lld x,lld y,lld &g,lld &a,lld &b) {
  if (y == 0) g=x,a=1,b=0;
  else
    exgcd(y,x%y,g,b,a),b-=(x/y)*a;
}
```

4.4 Pollard Rho

```
// does not work when n is prime
// return any non-trivial factor
llu pollard_rho(llu n){
  static auto f=[](llu x,llu k,llu m){
    return add(k,mul(x,x,m),m);
  };
  if (!(n&1)) return 2;
  mt19937 rnd(120821011);
  while(true){
    llu y=2,yy=y,x=rnd()%n,t=1;
    for(llu sz=2;t==1;sz<<=1) {
      for(llu i=0;i<sz;++i){
        if(t!=1)break;
        yy=f(yy,x,n);
        t=gcd(yy>y?yy-y:y-yy,n);
      }
      y=yy;
    }
    if(t!=1&&t!=n) return t;
  }
}
```

4.5 Pi Count (Linear Sieve)

```
static constexpr int N = 1000000 + 5;
lld pi[N];
vector<int> primes;
bool sieved[N];
lld cube_root(lld x){
  lld s=cbrt(x-static_cast<long double>(0.1));
  while(s*s*s <= x) ++s;
  return s-1;
}
lld square_root(lld x){
  lld s=sqrt(x-static_cast<long double>(0.1));
  while(s*s <= x) ++s;
  return s-1;
}
void init(){
  primes.reserve(N);
  primes.push_back(1);
  for(int i=2;i<N;i++) {
    if(!sieved[i]) primes.push_back(i);
    pi[i] = !sieved[i] + pi[i-1];
    for(int p: primes) if(p > 1) {
      if(p * i >= N) break;
      sieved[p * i] = true;
      if(p % i == 0) break;
    }
  }
}
lld phi(lld m, lld n) {
  static constexpr int MM = 80000, NN = 500;
  static lld val[MM][NN];
  if(m<MM&&n<NN&&val[m][n])return val[m][n]-1;
  if(n == 0) return m;
  if(primes[n] >= m) return 1;
  lld ret = phi(m,n-1)-phi(m/primes[n],n-1);
  if(m<MM&&n<NN) val[m][n] = ret+1;
  return ret;
}
lld pi_count(lld);
lld P2(lld m, lld n) {
  lld sm = square_root(m), ret = 0;
  for(lld i = n+1;primes[i]<=sm;i++)
    ret+=pi_count(m/primes[i])-pi_count(primes[i])+1;
  return ret;
}
lld pi_count(lld m) {
  if(m < N) return pi[m];
  lld n = pi_count(cube_root(m));
  return phi(m, n) + n - 1 - P2(m, n);
}
```

4.6 Range Sieve

```
const int MAX_SQRT_B = 50000;
const int MAX_L = 200000 + 5;

bool is_prime_small[MAX_SQRT_B];
bool is_prime[MAX_L];

void sieve(lld l, lld r){
  // [l, r)
  for(lld i=2;i*i<r;i++) is_prime_small[i] = true;
  for(lld i=l;i<r;i++) is_prime[i-l] = true;
  if(l==1) is_prime[0] = false;
  for(lld i=2;i*i<r;i++){
    if(!is_prime_small[i]) continue;
    for(lld j=i*i;j*j<r;j+=i) is_prime_small[j]=false;
    for(lld j=std::max(2LL, (l+i-1)/i)*i;j<r;j+=i)
      is_prime[j-l]=false;
  }
}
```

4.7 Miller Rabin

```
bool isprime(llu x){
  static llu magic[]={2,325,9375,28178,\
                      450775,9780504,1795265022};
  static auto witn=[](llu a,llu u,llu n,int t){
    a = mpow(a,u,n);
    if (!a)return 0;
    while(t--){
      llu a2=mul(a,a,n);
      if(a2==1 && a!=1 && a!=n-1)
        return 1;
      a = a2;
    }
    return a!=1;
  };
  if(x<2)return 0;
  if(!(x&1))return x==2;
  llu x1=x-1;int t=0;
  while(!(x1&1))x1>>=1,t++;
  for(llu m:magic)
    if(witn(m,x1,x,t))
      return 0;
  return 1;
}
```

4.8 Inverse Element

```
// x's inverse mod k
long long GetInv(long long x, long long k){
  // k is prime: euler_(k)=k-1
  return qPow(x, euler_phi(k)-1);
}
// if you need [1, x] (most use: [1, k-1]
void solve(int x, long long k){
  inv[1] = 1;
  for(int i=2;i<x;i++)
    inv[i] = ((long long)(k - k/i) * inv[k % i]) % k;
}
```

4.9 Euler Phi Function

```
/*
   extended euler:
   a^b mod p
   if gcd(a, p)==1: a^(b%phi(p))
   elif b < phi(p): a^b mod p
   else a^(b%phi(p) + phi(p))
 */
lld euler_phi(int x){
  lld r=1;
  for(int i=2;i*i<=x;++i){
    if(x%i==0){
      x/=i;
      r*=(i-1);
      while(x%i==0){
        x/=i;
        r*=i;
      }
    }
  }
  if(x>1) r*=x-1;
  return r;
}

vector<int> primes;
bool notprime[N];
lld phi[N];
void euler_sieve(int n){
  for(int i=2;i<n;i++){
    if(!notprime[i]){
      primes.push_back(i);
      phi[i] = i-1;
    }
    for(auto j: primes){
      if(i*j >= n) break;
      notprime[i*j] = true;
      phi[i*j] = phi[i] * phi[j];
      if(i % j == 0){
        phi[i*j] = phi[i] * j;
        break;
      }
    }
  }
}
```

4.10 Gauss Elimination

```
typedef long double llf;
const int N = 300;
const llf EPS = 1e-8;

// make m[i][i] = x, m[i][j] = 0
// v is for solving equation:
// for(int i=0;i<n;i++) ans[pos[i]] = val[i]/mtx[i][pos
    [i]];
// for(int i=0;i<n;i++) cout << ans[i] << '\n';
bool Gauss(llf m[N][N], llf v[N], int n, int pos[N]){
  for(int i=0;i<n;i++){
    int x=-1, y=-1; llf e = 0;
    for(int j=i;j<n;j++) for(int k=i;k<n;k++){
      if(fabs(m[j][pos[k]])>e){
        e = fabs(m[j][pos[k]]);
        x = j, y = k;
      }
    }
    if(x==-1 or y==-1) return false;
    swap(m[x], m[i]);
    swap(v[x], v[i]);
    swap(pos[y], pos[i]);
    for(int j=i+1;j<n;j++){
      llf xi = m[j][pos[i]]/m[i][pos[i]];
      for(int k=0;k<n;k++) m[j][pos[k]] -= xi*m[i][pos[
          k]];
      v[j] -= xi*v[i];
    }
  }
  for(int i=n-1;i>=0;i--){
    for(int j=i-1;j>=0;j--){
      llf xi = m[j][pos[i]]/m[i][pos[i]];
      for(int k=0;k<n;k++) m[j][pos[k]] -= xi*m[i][pos[
          k]];
      v[j] -= xi*v[i];
    }
  }
  return true;
}
```

4.11 Fast Fourier Transform

```
/*
   polynomial multiply:
   DFT(a, len); DFT(b, len);
   for(int i=0;i<len;i++) c[i] = a[i]*b[i];
   iDFT(c, len);
   (len must be 2^k and >= 2*(max(a, b)))
   Hand written Cplx would be 2x faster
 */
Cplx omega[2][N];
void init_omega(int n) {
  static constexpr llf PI=acos(-1);
  const llf arg=(PI+PI)/n;
  for(int i=0;i<n;++i)
    omega[0][i]={cos(arg*i),sin(arg*i)};
  for(int i=0;i<n;++i)
    omega[1][i]=conj(omega[0][i]);
}
void tran(Cplx arr[],int n,Cplx omg[]) {
  for(int i=0,j=0;i<n;++i){
    if(i>j)swap(arr[i],arr[j]);
    for(int l=n>>1;(j^=l)<l;l>>=1);
  }
  for (int l=2;l<=n;l<<=1){
    int m=l>>1;
    for(auto p=arr;p!=arr+n;p+=l){
      for(int i=0;i<m;++i){
        Cplx t=omg[n/l*i]*p[m+i];
        p[m+i]=p[i]-t;
        p[i]+=t;
      }
    }
  }
}
void DFT(Cplx arr[],int n){
  tran(arr,n,omega[0]);
}
void iDFT(Cplx arr[],int n){
  tran(arr,n,omega[1]);
  for(int i=0;i<n;++i)arr[i]/=n;
}
```

4.12 Chinese Remainder

```
lld crt(lld ans[], lld pri[], int n){
  lld M = 1;
  for(int i=0;i<n;i++) M *= pri[i];
  lld ret = 0;
  for(int i=0;i<n;i++){
    lld inv = (gcd(M/pri[i], pri[i]).first + pri[i])%
        pri[i];
    ret += (ans[i]*(M/pri[i])%M * inv)%M;
    ret %= M;
  }
  return ret;
}
/*
Another:
x = a1 % m1
x = a2 % m2
g = gcd(m1, m2)
assert((a1-a2)%g==0)
[p, q] = exgcd(m2/g, m1/g)
return a2+m2*(p*(a1-a2)/g)
0 <= x < lcm(m1, m2)
*/
```

4.13 Berlekamp Massey

```
// x: 1-base, p[]: 0-base
template<size_t N>
vector<llf> BM(llf x[N],size_t n){
  size_t f[N]={0},t=0;llf d[N];
  vector<llf> p[N];
  for(size_t i=1,b=0;i<=n;++i) {
    for(size_t j=0;j<p[t].size();++j)
      d[i]+=x[i-j-1]*p[t][j];
    if(abs(d[i]-=x[i])<=EPS)continue;
    f[t]=i;if(!t){p[++t].resize(i);continue;}
    vector<llf> cur(i-f[b]-1);
    llf k=-d[i]/d[f[b]];cur.PB(-k);
    for(size_t j=0;j<p[b].size();j++)
      cur.PB(p[b][j]*k);
    if(cur.size()<p[t].size())cur.resize(p[t].size());
    for(size_t j=0;j<p[t].size();j++)cur[j]+=p[t][j];
    if(i-f[b]+p[b].size()>=p[t].size()) b=t;
    p[++t]=cur;
  }
  return p[t];
}
```

4.14 NTT

```
// Remember coefficient are mod P
/* p=a*2^n+1
   n    2^n         p          a    root
   16   65536       65537      1    3
   20   1048576     7340033    7    3 */
// (must be 2^k)
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 , 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;
    }
  }
};
const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

4.15 FWT

```
/* xor convolution:
 * x = (x0,x1) , y = (y0,y1)
 * z = ( x0y0 + x1y1 , x0y1 + x1y0 )
 * =>
 * x' = ( x0+x1 , x0-x1 ) , y' = ( y0+y1 , y0-y1 )
 * z' = ( ( x0+x1 )( y0+y1 ) , ( x0-x1 )( y0-y1 ) )
 * z = (1/2) * z''
 * or convolution:
 * x = (x0, x0+x1), inv = (x0, x1-x0) w/o final div
 * and convolution:
 * x = (x0+x1, x1), inv = (x0-x1, x1) w/o final div */
const LL MOD = 1e9+7;
inline void fwt( LL x[ MAXN ] , int N , bool inv=0 ) {
  for( int d = 1 ; d < N ; d <<= 1 ) {
    int d2 = d<<1;
    for( int s = 0 ; s < N ; s += d2 )
      for( int i = s , j = s+d ; i < s+d ; i++, j++ ){
        LL ta = x[ i ] , tb = x[ j ];
        x[ i ] = ta+tb;
        x[ j ] = ta-tb;
        if( x[ i ] >= MOD ) x[ i ] -= MOD;
        if( x[ j ] < 0 ) x[ j ] += MOD;
      }
  }
  if( inv )
    for( int i = 0 ; i < N ; i++ ) {
      x[ i ] *= inv( N, MOD );
      x[ i ] %= MOD;
    }
}
```

4.16 DiscreteLog

```
// Baby-step Giant-step Algorithm
lld BSGS(lld P, lld B, lld N) {
  // find B^L = N mod P
  unordered_map<lld, int> R;
  lld sq = (lld)sqrt(P);
  lld t = 1;
  for (int i = 0; i < sq; i++) {
    if (t == N)
      return i;
    if (!R.count(t))
      R[t] = i;
    t = (t * B) % P;
  }
  lld f = inverse(t, P);
  for(int i=0;i<=sq+1;i++) {
    if (R.count(N))
      return i * sq + R[N];
    N = (N * f) % P;
  }
  return -1;
}
```

4.17 Quadratic residue

```
struct Status{
  ll x,y;
};
ll w;
Status mult(const Status& a,const Status& b,ll mod){
  Status res;
  res.x=(a.x*b.x+a.y*b.y%mod*w)%mod;
  res.y=(a.x*b.y+a.y*b.x)%mod;
  return res;
}
inline Status qpow(Status _base,ll _pow,ll _mod){
  Status res;
  res.x=1,res.y=0;
  while(_pow>0){
    if(_pow&1) res=mult(res,_base,_mod);
    _base=mult(_base,_base,_mod);
    _pow>>=1;
  }
  return res;
}
inline ll check(ll x,ll p){
  return qpow_mod(x,(p-1)>>1,p);
}
inline ll get_root(ll n,ll p){
  if(p==2) return 1;
  if(check(n,p)==p-1) return -1;
  ll a;
  while(true){
    a=rand()%p;
    w=((a*a-n)%p+p)%p;
    if(check(w,p)==p-1) break;
  }
  Status res;
  res.x=a;
  res.y=1;
  res=qpow(res,(p+1)>>1,p);
  return res.x;
}
```

5 Geometry

5.1 Point Class

```
template<typename T>
struct Point{
  typedef long double llf;
  static constexpr llf EPS = 1e-8;
  T x, y;
  Point(T _=0, T __=0): x(_), y(__){}
  template<typename T2>
    Point(const Point<T2>& a): x(a.x), y(a.y){}
  inline llf theta() const {
    return atan2((llf)y, (llf)x);
  }
  inline llf dis() const {
    return hypot((llf)x, (llf)y);
  }
  inline llf dis(const Point& o) const {
    return hypot((llf)(x-o.x), (llf)(y-o.y));
  }
  Point operator-(const Point& o) const {
    return Point(x-o.x, y-o.y);
  }
  Point operator-=(const Point& o){
    x-=o.x, y-=o.y;
    return *this;
  }
  Point operator+(const Point& o) const {
    return Point(x+o.x, y+o.y);
  }
  Point operator+=(const Point& o){
    x+=o.x, y+=o.y;
    return *this;
  }
  Point operator*(const T& k) const {
    return Point(x*k, y*k);
  }
  Point operator*=(const T& k){
    x*=k, y*=k;
    return *this;
  }
  Point operator/(const T& k) const {
    return Point(x/k, y/k);
  }
  Point operator/=(const T& k){
    x/=k, y/=k;
    return *this;
  }
  Point operator-() const {
    return Point(-x, -y);
  }
  Point rot90() const {
    return Point(-y, x);
  }
  template<typename T2>
  bool in(const Circle<T2>& a) const {
    /* Add struct Circle at top */
    return a.o.dis(*this)+EPS <= a.r;
  }
  bool equal(const Point& o, true_type) const {
    return fabs(x-o.x) < EPS and fabs(y-o.y) < EPS;
  }
  bool equal(const Point& o, false_type) const {
    return tie(x, y) == tie(o.x, o.y);
  }
  bool operator==(const Point& o) const {
    return equal(o, is_floating_point<T>());
  }
  bool operator!=(const Point& o) const {
    return !(*this == o);
  }
  bool operator<(const Point& o) const {
    return theta() < o.theta();
    // sort like what pairs did
    // if(is_floating_point<T>()) return fabs(x-o.x)<
        EPS?y<o.y:x<o.x;
    // else return tie(x, y) < tie(o.x, o.y);
  }
  friend inline T cross(const Point& a, const Point& b)
      {
    return a.x*b.y - b.x*a.y;
  }
  friend inline T dot(const Point& a, const Point &b){
    return a.x*b.x + a.y*b.y;
  }
  friend ostream& operator<<(ostream& ss, const Point&
      o){
    ss<<"("<<o.x<<", "<<o.y<<")";
    return ss;
  }
};
```

5.2 Circle Class

```
template<typename T>
struct Circle{
  static constexpr llf EPS = 1e-8;
  Point<T> o;
  T r;
  vector<Point<llf>> operator&(const Circle& aa)const{
    // https://www.cnblogs.com/wangzming/p/8338142.html
    llf d=o.dis(aa.o);
    if(d > r+aa.r+EPS or d < fabs(r-aa.r)-EPS) return
        {};
    llf dt = (r*r - aa.r*aa.r)/d, d1 = (d+dt)/2;
    Point<llf> dir = (aa.o-o); dir /= d;
    Point<llf> pcrs = dir*d1 + o;
    dt=sqrt(max(0.0L, r*r - d1*d1)), dir=dir.rot90();
    return {pcrs + dir*dt, pcrs - dir*dt};
  }
};
```

5.3 Line Class

```
const Point<long double> INF_P(-1e20, 1e20);
const Point<long double> NOT_EXIST(1e20, 1e-20);
template<typename T>
struct Line{
  static constexpr long double EPS = 1e-8;
  // ax+by+c = 0
  T a, b, c;
  Line(): a(0), b(1), c(0){}
  Line(T _, T __, T ___): a(_), b(__), c(___){
    assert(fabs(a)>EPS or fabs(b)>EPS);
  }
  template<typename T2>
    Line(const Line<T2>& x): a(x.a), b(x.b), c(x.c){}
  typedef Point<long double> Pt;
  bool equal(const Line& o, true_type) const {
    return fabs(a-o.a) < EPS and fabs(b-o.b) < EPS and
        fabs(c-o.b) < EPS;
  }
  bool euqal(const Line& o, false_type) const {
    return a==o.a and b==o.b and c==o.c;
  }
  bool operator==(const Line& o) const {
    return euqal(o, is_floating_point<T>());
  }
  bool operator!=(const Line& o) const {
    return !(*this == o);
  }
  friend inline bool on_line__(const Point<T>& p, const
       Line& l, true_type){
    return fabs(l.a*p.x + l.b*p.y + l.c) < EPS;
  }
  friend inline bool on_line__(const Point<T>& p, const
       Line& l, false_type){
    return l.a*p.x + l.b*p.y + l.c == 0;
  }
  friend inline bool on_line(const Point<T>&p const
      Line& l){
    return on_line__(p, l, is_floating_point<T>());
  }
  friend inline bool is_parallel__(const Line& x, const
       Line& y, true_type){
    return fabs(x.a*y.b - x.b*y.a) < EPS;
  }
  friend inline bool is_parallel__(const Line& x, const
       Line& y, false_type){
    return x.a*y.b == x.b*y.a;
  }
  friend inline bool is_parallel(const Line& x, const
      Line& y){
    return is_parallel__(x, y, is_floating_point<T>());
  }
  friend inline Pt get_inter(const Line& x, const Line&
       y){
    typedef long double llf;
    if(x==y) return INF_P;
    if(is_parallel(x, y)) return NOT_EXIST;
    llf delta = x.a*y.b - x.b*y.a;
    llf delta_x = x.b*y.c - x.c*y.b;
    llf delta_y = x.c*y.a - x.a*y.c;
    return Pt(delta_x / delta, delta_y / delta);
  }
  friend ostream& operator<<(ostream& ss, const Line& o
      ){
    ss<<o.a<<"x+"<<o.b<<"y+"<<o.c<<"=0";
    return ss;
  }
};
template<typename T>
inline Line<T> get_line(const Point<T>& a, const Point<
    T>& b){
  return Line<T>(a.y-b.y, b.x-a.x, (b.y-a.y)*a.x-(b.x-a
      .x)*a.y);
}
```

5.4 Triangle Circumcentre

```
template<typename T>
Circle<llf> get_circum(const Point<T>& a, const Point<T
    >& b, const Point<T>& c){
  llf a1 = a.x-b.x;
  llf b1 = a.y-b.y;
  llf c1 = (a.x+b.x)/2 * a1 + (a.y+b.y)/2 * b1;
  llf a2 = a.x-c.x;
  llf b2 = a.y-c.y;
  llf c2 = (a.x+c.x)/2 * a2 + (a.y+c.y)/2 * b2;

  Circle<llf> cc;
  cc.o.x = (c1*b2-b1*c2)/(a1*b2-b1*a2);
  cc.o.y = (a1*c2-c1*a2)/(a1*b2-b1*a2);
  cc.r = hypot(cc.o.x-a.x, cc.o.y-a.y);
  return cc;
}
```

5.5 2D Convex Hull

```
template<typename T>
class ConvexHull_2D{
private:
  typedef Point<T> PT;
  vector<PT> dots;
  struct myhash{
    uint64_t operator()(const PT& a) const {
      uint64_t xx=0, yy=0;
      memcpy(&xx, &a.x, sizeof(a.x));
      memcpy(&yy, &a.y, sizeof(a.y));
      uint64_t ret = xx*17+yy*31;
      ret = (ret ^ (ret >> 16))*0x9E3779B1;
      ret = (ret ^ (ret >> 13))*0xC2B2AE35;
      ret = ret ^ xx;
      return (ret ^ (ret << 3)) * yy;
    }
  };
  unordered_set<PT, myhash> in_hull;
public:
  inline void init(){in_hull.clear();dots.clear();}
  void insert(const PT& x){dots.PB(x);}
  void solve(){
    sort(ALL(dots), [](const PT& a, const PT& b){
      return tie(a.x, a.y) < tie(b.x, b.y);
    });
    vector<PT> stk(SZ(dots)<<1);
    int top = 0;
    for(auto p: dots){
      while(top >= 2 and cross(p-stk[top-2], stk[top
          -1]-stk[top-2]) <= 0)
        top --;
      stk[top++] = p;
    }
    for(int i=SZ(dots)-2, t = top+1;i>=0;i--){
      while(top >= t and cross(dots[i]-stk[top-2], stk[
          top-1]-stk[top-2]) <= 0)
        top --;
      stk[top++] = dots[i];
    }
    stk.resize(top-1);
    swap(stk, dots);
    for(auto i: stk) in_hull.insert(i);
  }
  vector<PT> get(){return dots;}
  inline bool in_it(const PT& x){
    return in_hull.find(x)!=in_hull.end();
  }
};
```

5.6 2D Farthest Pair

```
// stk is from convex hull
n = (int)(stk.size());
int pos = 1, ans = 0; stk.push_back(arr[0]);
for(int i=0;i<n;i++){
  while(abs(cross(stk[i+1]-stk[i], stk[(pos+1)%n]-stk[i
      ]))\
    > abs(cross(stk[i+1]-stk[i], stk[pos]-stk[i]))) pos
        = (pos+1)%n;
  ans = max({ans, dis(stk[i], stk[pos]), dis(stk[i+1],
      stk[pos])});
}
```

5.7 2D Cosest Pair

```
struct Point{
  llf x, y;
  llf dis;
} arr[N];

inline llf get_dis(Point a, Point b){
  return hypot(a.x-b.x, a.y-b.y);
}

llf solve(){
  int cur = rand()%n;
  for(int i=0;i<n;i++) arr[i].dis = get_dis(arr[cur],
      arr[i]);
  sort(arr, arr+n, [](Point a, Point b){return a.dis <
      b.dis;});
  llf ans = 1e50;
  for(int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
      if(arr[j].dis - arr[i].dis > ans) break;
      ans = min(ans, get_dis(arr[i], arr[j]));
    }
  }
  return ans;
}
```

5.8 SimulateAnnealing

```
double getY(double);
int main(){
  int rr, ll;
  default_random_engine rEng(time(NULL));
  uniform_real_distribution<double> Range(-1,1);
  uniform_real_distribution<double> expR(0,1);
  auto Random=bind(Range,rEng), expRand=bind(expR,rEng)
      ;
  int step=0;
  double pace=rr-ll, mini=0.95; // need to search for
      it
  double x=max(min(Random()*pace+ll, rr), ll), y=getY(x
      );
  while(pace>=1e-7){
    double newX = max(min(x + Random()*pace, rr), ll);
    double newY = getY(newX);
    if(newY < y || expRand() < exp(-step))
      x=newX, y=newY;
    step++;
    pace*=mini;
  }
}
```

5.9 Ternary Search on Integer

```
int TernarySearch(int l, int r) {
  // (l, r]
  while (r - l > 1){
    int mid = (l + r)>>1;
    if (f(mid) > f(mid + 1)) r = mid;
    else l = mid;
  }
  return l+1;
}
```

5.10 Minimum Covering Circle

```
template<typename T>
Circle<llf> MinCircleCover(const vector<Point<T>>& pts)
    {
  random_shuffle(ALL(pts));
  Circle<llf> c = {pts[0], 0};
  int n = SZ(pts);
  for(int i=0;i<n;i++){
    if(pts[i].in(c)) continue;
    c = {pts[i], 0};
    for(int j=0;j<i;j++){
      if(pts[j].in(c)) continue;
      c.o = (pts[i] + pts[j]) / 2;
      c.r = pts[i].dis(c.o);
      for(int k=0;k<j;k++){
        if(pts[k].in(c)) continue;
        c = get_circum(pts[i], pts[j], pts[k]);
      }
    }
  }
  return c;
}
```

5.11 KDTree (Nearest Point)

```
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 Stringology

6.1 Hash

```
class Hash{
private:
  static const int N = 1000000;
  const int p = 127, q = 1208220623;
  int sz, prefix[N], power[N];
  inline int add(int x, int y){return x+y>=q?x+y-q:x+y
      ;}
  inline int sub(int x, int y){return x-y<0?x-y+q:x-y;}
  inline int mul(int x, int y){return 1LL*x*y%q;}
public:
  void init(const std::string &x){
    sz = x.size();
    prefix[0]=0;
    for(int i=1;i<=sz;i++) prefix[i]=add(mul(prefix[i
        -1], p), x[i-1]);
    power[0]=1;
    for(int i=1;i<=sz;i++) power[i]=mul(power[i-1], p);
  }
  int query(int l, int r){
    // 1-base (l, r]
    return sub(prefix[r], mul(prefix[l], power[r-l]));
  }
};
```

6.2 Suffix Array

```
//help by http://www.geeksforgeeks.org/suffix-array-set
    -2-a-nlognlogn-algorithm/
struct sfx{
  int index;
  int r,nr;
};
char str[N + 10];
int len;
vector<sfx> srs[N + 10];
int mapping[N + 10];
sfx sa[N + 10];
bool cmp(sfx a,sfx b){
  if(a.r==b.r){
    return a.nr<b.nr;
  }else{
    return a.r<b.r;
  }
}
void SA(){
  len = strlen(str);
  for(int i=0;i<len;i++){
    sa[i].index = i;
    sa[i].r=str[i];
    sa[i].nr=(i+1>=len)?0:str[i+1];
  }
  //sort(sa,sa+len,cmp);
  radixSort();
  for(int j=2;j<=len;j*=2){
    int cnt=1;
    int rr = sa[0].r;
    sa[0].r=cnt;
    mapping[sa[0].index]=0;
    for(int i=1;i<len;i++){
      if(sa[i].r == rr && sa[i].nr == sa[i-1].nr){
        rr=sa[i].r;
        sa[i].r=cnt;
      }else{
        rr=sa[i].r;
        sa[i].r=++cnt;
      }
      mapping[sa[i].index]=i;
    }
    for(int i=0;i<len;i++){
      int nn = sa[i].index+j;
      sa[i].nr = (nn>=len)?0:sa[mapping[nn]].r;
    }
    //sort(sa, sa+len, cmp);
    radixSort();
  }
}
void radixSort(){
  int m = 0;
  for(int i=0;i<len;i++){
    srs[sa[i].nr].PB(sa[i]);
    m=max(m,sa[i].nr);
  }
  int cnt=0;
  for(int i=0;i<=m;i++){
    if(srs[i].empty())continue;
    for(auto j:srs[i]){
      sa[cnt++] = j;
    }
    srs[i].clear();
  }
  m = 0;
  for(int i=0;i<len;i++){
    srs[sa[i].r].PB(sa[i]);
    m=max(m,sa[i].r);
  }
  cnt=0;
  for(int i=0;i<=m;i++){
    if(srs[i].empty())continue;
    for(auto j:srs[i]){
      sa[cnt++] = j;
    }
    srs[i].clear();
  }
}
```

6.3 Aho-Corasick Algorithm

```
class AhoCorasick{
  private:
    static constexpr int Z = 26;
    struct node{
      node *nxt[ Z ], *fail;
      vector< int > data;
      node(): fail( nullptr ) {
        memset( nxt, 0, sizeof( nxt ) );
        data.clear();
      }
    } *rt;
    inline int Idx( char c ) { return c - 'a'; }
  public:
    void init() { rt = new node(); }
    void add( const string& s, int d ) {
      node* cur = rt;
      for ( auto c : s ) {
        if ( not cur->nxt[ Idx( c ) ] )
          cur->nxt[ Idx( c ) ] = new node();
        cur = cur->nxt[ Idx( c ) ];
      }
      cur->data.push_back( d );
    }
    void compile() {
      vector< node* > bfs;
      size_t ptr = 0;
      for ( int i = 0 ; i < Z ; ++ i ) {
        if ( not rt->nxt[ i ] )
          continue;
        rt->nxt[ i ]->fail = rt;
        bfs.push_back( rt->nxt[ i ] );
      }
      while ( ptr < bfs.size() ) {
        node* u = bfs[ ptr ++ ];
        for ( int i = 0 ; i < Z ; ++ i ) {
          if ( not u->nxt[ i ] )
            continue;
          node* u_f = u->fail;
          while ( u_f ) {
            if ( not u_f->nxt[ i ] ) {
              u_f = u_f->fail;
              continue;
            }
            u->nxt[ i ]->fail = u_f->nxt[ i ];
            break;
          }
          if ( not u_f ) u->nxt[ i ]->fail = rt;
          bfs.push_back( u->nxt[ i ] );
        }
      }
    }
    void match( const string& s, vector< int >& ret ) {
      node* u = rt;
      for ( auto c : s ) {
        while ( u != rt and not u->nxt[ Idx( c ) ] )
          u = u->fail;
        u = u->nxt[ Idx( c ) ];
        if ( not u ) u = rt;
        node* tmp = u;
        while ( tmp != rt ) {
          for ( auto d : tmp->data )
            ret.push_back( d );
          tmp = tmp->fail;
        }
      }
    }
} ac;
```

6.4 Suffix Automaton

```
struct Node{
  Node *green, *edge[26];
  int max_len;
  Node(const int _max_len)
    : green(NULL), max_len(_max_len){
    memset(edge,0,sizeof(edge));
  }
} *ROOT, *LAST;
void Extend(const int c) {
  Node *cursor = LAST;
  LAST = new Node((LAST->max_len) + 1);
  for(;cursor&&!cursor->edge[c]; cursor=cursor->green)
    cursor->edge[c] = LAST;
  if (!cursor)
    LAST->green = ROOT;
  else {
    Node *potential_green = cursor->edge[c];
    if((potential_green->max_len)==(cursor->max_len+1))
      LAST->green = potential_green;
    else {
//assert(potential_green->max_len>(cursor->max_len+1));
      Node *wish = new Node((cursor->max_len) + 1);
      for(;cursor && cursor->edge[c]==potential_green;
           cursor = cursor->green)
        cursor->edge[c] = wish;
      for (int i = 0; i < 26; i++)
        wish->edge[i] = potential_green->edge[i];
      wish->green = potential_green->green;
      potential_green->green = wish;
      LAST->green = wish;
    }
  }
}
char S[10000001], A[10000001];
int N;
int main(){
  scanf("%d%s", &N, S);
  ROOT = LAST = new Node(0);
  for (int i = 0; S[i]; i++)
    Extend(S[i] - 'a');
  while (N--){
    scanf("%s", A);
    Node *cursor = ROOT;
    bool ans = true;
    for (int i = 0; A[i]; i++){
      cursor = cursor->edge[A[i] - 'a'];
      if (!cursor) {
        ans = false;
        break;
      }
    }
    puts(ans ? "Yes" : "No");
  }
  return 0;
}
```

6.5 KMP

```
int F[N<<1];
void KMP(char s1[], char s2[], int n, int m){
  // make F[] for s1+'\0'+s2;
  char ss[N<<1];
  int len = n+m+1;
  for(int i=0;i<n;i++) ss[i] = s1[i];
  ss[n] = '\0';
  for(int i=0;i<m;i++) ss[i+1+n] = s2[i];
  F[0] = F[1] = 0;
  for(int i=1;i<len;i++){
    int j = F[i];
    while(j > 0 and ss[i]!=ss[j]) j = F[j];
    F[i+1] = (ss[i]==ss[j]?j+1:0);
  }
  // just find (F[len2+i] == len2)
  // i from 1 to len+1 for matching
}
/*
  [0, i]是個循環字串，且循環節為i-f[i]：
  if(f[i]>0 and i%(i-f[i])==0)
  cout << i << " " << i/(i-f[i]) << '\n';
*/
```

6.6 Z value

```
char s[MAXN];
int len,z[MAXN];
void Z_value() {
  int i,j,left,right;
  left=right=0; z[0]=len;
  for(i=1;i<len;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) {
      right=i+z[i];
      left=i;
    }
  }
}
```

6.7 Lexicographically 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);
}
```

6.8 BWT

```
struct BurrowsWheeler{
#define SIGMA 26
#define BASE 'a'
  vector<int> v[ SIGMA ];
  void BWT(char* ori, char* res){
    // make ori -> ori + ori
    // then build suffix array
  }
  void iBWT(char* ori, char* res){
    for( int i = 0 ; i < SIGMA ; i ++ )
      v[ i ].clear();
    int len = strlen( ori );
    for( int i = 0 ; i < len ; i ++ )
      v[ ori[i] - BASE ].push_back( i );
    vector<int> a;
    for( int i = 0 , ptr = 0 ; i < SIGMA ; i ++ )
      for( auto j : v[ i ] ){
        a.push_back( j );
        ori[ ptr ++ ] = BASE + i;
      }
    for( int i = 0 , ptr = 0 ; i < len ; i ++ ){
      res[ i ] = ori[ a[ ptr ] ];
      ptr = a[ ptr ];
    }
    res[ len ] = 0;
  }
} bwt;
```

7 Misc

7.1 Degree Sequence Validity

7.1.1 Erdős–Gallai Theorem

$d_1 \geq \cdots \geq d_n$ is a valid degree sequence iff
$\sum_{k=1}^{n} d_k \equiv 0 \mod 2$
$\wedge$
$\forall 1 \leq k \leq n, \sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$

7.1.2 Havel–Hakimi algorithm

find the vertex who has greatest degree unused, connect it with other greatest vertex.

7.2 MaximumEmptyRect

```
int max_empty_rect(int n, int m, bool blocked[N][N]){
  static int mxu[2][N], me=0,he=1,ans=0;
  for(int i=0;i<m;i++) mxu[he][i]=0;
  for(int i=0;i<n;i++){
    stack<PII,vector<PII>> stk;
    for(int j=0;j<m;++j){
      if(blocked[i][j]) mxu[me][j]=0;
      else mxu[me][j]=mxu[he][j]+1;
      int la = j;
      while(!stk.empty()&&stk.top().FF>mxu[me][j]){
        int x1 = i - stk.top().FF, x2 = i;
        int y1 = stk.top().SS, y2 = j;
        la = stk.top().SS; stk.pop();
        ans=max(ans,(x2-x1)*(y2-y1));
      }
      if(stk.empty()||stk.top().FF<mxu[me][j])
        stk.push({mxu[me][j],la});
    }
    while(!stk.empty()){
      int x1 = i - stk.top().FF, x2 = i;
      int y1 = stk.top().SS-1, y2 = m-1;
      stk.pop();
      ans=max(ans,(x2-x1)*(y2-y1));
    }
    swap(me,he);
  }
  return ans;
}
```

7.3 DP-opt Condition

7.3.1 totally monotone (concave/convex)

$\forall i < i', j < j', B[i][j] \leq B[i'][j] \implies B[i][j'] \leq B[i'][j']$
$\forall i < i', j < j', B[i][j] \geq B[i'][j] \implies B[i][j'] \geq B[i'][j']$

7.3.2 monge condition (concave/convex)

$\forall i < i', j < j', B[i][j] + B[i'][j'] \geq B[i][j'] + B[i'][j]$
$\forall i < i', j < j', B[i][j] + B[i'][j'] \leq B[i][j'] + B[i'][j]$

7.4 Convex 1D/1D DP

```
struct segment {
  int i, l, r;
  segment() {}
  segment(int a, int b, int c): i(a), l(b), r(c) {}
};

inline long long f(int l, int r) {
  return dp[l] + w(l + 1, r);
}

void solve() {
  dp[0] = 0ll;
  deque<segment> deq; deq.push_back(segment(0, 1, n));
  for (int i = 1; i <= n; ++i) {
    dp[i] = f(deq.front().i, i);
    while (deq.size() && deq.front().r < i + 1) deq.
        pop_front();
    deq.front().l = i + 1;
    segment seg = segment(i, i + 1, n);
    while (deq.size() && f(i, deq.back().l) < f(deq.
        back().i, deq.back().l)) deq.pop_back();
    if (deq.size()) {
      int d = 1048576, c = deq.back().l;
      while (d >>= 1) if (c + d <= deq.back().r) {
        if (f(i, c + d) > f(deq.back().i, c + d)) c +=
            d;
      }
      deq.back().r = c; seg.l = c + 1;
    }
    if (seg.l <= n) deq.push_back(seg);
  }
}
```

7.5 Josephus Problem

```
// n people kill m for each turn
int f(int n, int m) {
  int s = 0;
  for (int i = 2; i <= n; i++)
    s = (s + m) % i;
  return s;
}
// died at kth
int kth(int n, int m, int k){
  if (m == 1) return n-1;
  for (k = k*m+m-1; k >= n; k = k-n+(k-n)/(m-1));
  return k;
}
```