Contents

1 Basic

1.1 Default Code

```cpp
#include <bits/stdc++.h>
using namespace std;
using lld = int64_t;
using llu = uint64_t;
using llf = long double;
using PII = pair<int,int>;
using PIL = pair<int,lld>;
using PLI = pair<lld,int>;
using PLL = pair<lld,lld>;
template<typename T>
using maxHeap = priority_queue<T,vector<T>,less<T>>;
template<typename T>
using minHeap = priority_queue<T,vector<T>,greater<T>>;
#define FF first
#define SS second
#define SZ(x) (int)((x).size())
#define ALL(x) begin(x), end(x)
#define PB push_back
#define WC(x) while((x)--)
template<typename Iter>
ostream& _out(ostream &s, Iter b, Iter e) {
  s<<"[";
  for(auto it=b;it!=e;it++)s<<(it==b?"":" ")<<*it;
  s<<"]";
  return s;
}
template<typename A, typename B>
ostream& operator<<(ostream &s,const pair<A,B>&p)
{return s<<"("<<p.FF<<","<<p.SS<<")";}
template<typename T>
ostream& operator<<(ostream &s,const vector<T>&c)
{return _out(s,ALL(c));}
bool debug = 0;
#define DUMP(x) if(debug)cerr<<__PRETTY_FUNCTION__<<":"
    <<__LINE__<<" - "<<(#x)<<"="<<(x)<<'\n'
template<typename T>
void DEBUG(const T& x){if(debug)cerr<<x;}
template<typename T,typename...Args>
void DEBUG(const T&head,const Args&...tail){
  if(debug){cerr<<head;DEBUG(tail...);}
}
int main(int argc,char* argv[]){
  if(argc>1&&string(argv[1])=="-D")debug=1;
  if(!debug){ios_base::sync_with_stdio(0);cin.tie(0);}
  return 0;
}
```

1.2 IncreaseStackSize

```cpp
//stack resize(change esp to rsp if 64-bit system)
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
// craziest way
static void run_with_stack_size(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

```cpp
#pragma GCC optimize("Ofast,no-stack-protector,no-math-
    errno,unroll-loops")
```

```
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm
    ,mmx,avx,tune=native")
```

1.4 Debugger

```
#! /usr/bin/env python3
import subprocess, platform
os_name = platform.system()
cmd,prefix = [],""
if os_name == 'Windows':
  cmd=["cmd", "/C"]
else:
  cmd = ["bash", "-c"]
  prefix = "./"
def GetTestData(exe):
  myout=subprocess.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=subprocess.check_output(cmd+["%s%s < test.in"%(
      prefix, a)])
  d=subprocess.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("1234.exe", "test.exe", GetTestData("
        gen.exe"))
```

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,0
    xC2B2AE35,16,13,16>;
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) {
    if ((end = buf + fread(buf, 1, 1 << 20, stdin)) ==
        buf) return EOF;
    p = buf;
  }
  return *p++;
}
template<typename T>
static inline bool gn(T &_){
  register int c = gc(); register T __ = 1; _ = 0;
  while(!isdigit(c) and c!=EOF and c!='-') c = gc();
  if(c == '-') { __ = -1; c = gc(); }
  if(c == EOF) return false;
  while(isdigit(c)) _ = _ * 10 + c - '0', c = gc();
  _ *= __;
  return true;
}
template <typename T, typename ...Args>
static inline bool gn(T &x, Args& ...args){return gn(x)
     and 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 and 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() and dig.back()==0) 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()==0 or a.len()==0) 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, const
      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 unordered_map

```
#include <ext/pb_ds/assoc_container.hpp>
using __gnu_pbds::cc_hash_table;
using __gnu_pbds::gp_hash_table;
template<typename A, typename B> using hTable1 =
    cc_hash_table<A,B>;
template<typename A, typename B> using hTable2 =
    gp_hash_table<A,B>;
```

2.3 extc_balance_tree

```
#include <ext/pb_ds/assoc_container.hpp>
using __gnu_pbds::tree;
using __gnu_pbds::rb_tree_tag;
using __gnu_pbds::ov_tree_tag;
using __gnu_pbds::splay_tree_tag;
using __gnu_pbds::null_type;
using __gnu_pbds::tree_order_statistics_node_update;
template<typename T>
using ordered_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
template<typename A, B>
using ordered_map = tree<A, B, less<A>, rb_tree_tag,
    tree_order_statistics_node_update>;
int main(){
  ordered_set<int> ss;
  ordered_map<int,int> mm;
  ss.insert(1);
  ss.insert(5);
  assert(*ss.find_by_order(0)==1);
  assert(ss.order_of_key(-1)==0);
  assert(ss.order_of_key(87)==2);
  return 0;
}
```

2.4 extc_heap

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using __gnu_pbds::priority_queue;
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;

int main(){
  priority_queue<int,less<int>,pairing_heap_tag> pq1,
      pq2;
  pq1.push(1);
  pq2.push(2);
  pq1.join(pq2);
  assert(pq2.size()==0);
  auto it = pq1.push(87);
  pq1.modify(it, 19);
  while(!pq1.empty()){
    pq1.top();
    pq1.pop();
  }
  return 0;
}
```

2.5 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(nullptr), rc(nullptr){}
  } *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;
  }
  void clear(SkewNode*& a){
    if(!a) return;
    clear(a->lc); clear(a->rc);
    delete a; a = nullptr;
  }
public:
  SkewHeap(): root(nullptr), count(0){}
  bool empty(){return count==0;}
  size_t size(){return count;}
  T top(){return root->x;}
  void clear(){clear(root);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 -= 1;
    SkewNode* rt = Merge(root->lc, root->rc);
    delete root; root = rt;
  }
  friend void swap(SkewHeap& a, SkewHeap& b){
    swap(a.root, b.root);
  }
};
```

2.6 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){
    if(fa[x] == x) return x;
    return 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();
    }
  }
};
```

2.7 Treap

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

2.8 SparseTable

```
template<typename T, typename Cmp_=less<T>>
class SparseTable{
private:
  vector<vector<T>> table;
  vector<int> lg;
  T cmp_(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, lg[1] = 0;
```

```
    for(int i=2;i<=n;i++) lg[i] = lg[i>>1]+1;
    table.resize(lg[n]+1);
    table[0].resize(n);
    for(int i=0;i<n;i++) table[0][i] = arr[i];
    for(int i=1;i<=lg[n];i++){
      int len = 1<<(i-1), sz = 1<<i;
      table[i].resize(n-sz+1);
      for(int j=0;j<=n-sz;j++){
        table[i][j] = cmp_(table[i-1][j], table[i-1][j+
            len]);
      }
    }
  }
  T query(int l, int r){
    // 0-base [l, r)
    int wh = lg[r-l], len=1<<wh;
    return cmp_(table[wh][l], table[wh][r-len]);
  }
};
```

2.9 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] and 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> bcc;
  vector<vector<PII>> G;
  void dfs(int w, int f){
    dfn[w] = cnt++;
    low[w] = dfn[w];
    for(auto i: G[w]){
      int u = i.FF, t = i.SS;
      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]) bcc[t] = true;
      }
    }
  }
public:
  void init(int n, int m){
    G.resize(n);
    fill(G.begin(), G.end(), vector<PII>());
    bcc.clear(); bcc.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].PB({v, cnt});
    G[v].PB({u, cnt});
    cnt++;
  }
  void solve(){cnt = 1;dfs(0, 0);}
  // the id will be same as insert order, 0-base
  bool is_bcc(int x){return bcc[x];}
} bcc;
```

3.2 BCC Vertex

```
class BCC{
  private:
    int n, ecnt;
    vector< vector< pair< int, int > > > G;
    vector< int > low, id;
    vector< bool > vis, ap;
    void dfs( int u, int f, int dfn ) {
      low[ u ] = dfn; vis[ u ] = true;
      for ( auto e : G[ u ] ) if ( e.first != f ) {
        if ( vis[ e.first ] ) {
          low[ u ] = min( low[ u ], low[ e.first ] );
        } else {
          dfs( e.first, u, dfn + 1 );
          if ( low[ e.first ] >= dfn ) ap[ u ] = true;
        }
      }
    }
    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.clear();
      low.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 Strongly Connected Components

```
class SCC{
private:
  int n, num_;
  vector<vector<int>> G, rG;
  vector<int> ord, num;
  bool vis[N];
  void dfs(int u){
    if(vis[u]) return;
    vis[u]=1;
    for(auto v: G[u]) dfs(v);
    ord.PB(u);
  }
  void rdfs(int u){
    if(vis[u]) return;
    num[u] = num_;
    vis[u] = 1;
    for(auto v: rG[u]) rdfs(v);
  }
public:
  inline void init(int n_){
    n=n_, num_=0;
    G.resize(n); rG.resize(n);
    num.resize(n);
    for(int i=0;i<n;i++) G[i].clear();
    for(int i=0;i<n;i++) rG[i].clear();
  }
  inline void add_edge(int st, int ed){
    G[st].PB(ed);
    rG[ed].PB(st);
  }
  void solve(){
    memset(vis, 0, sizeof(vis));
    for(int i=0;i<n;i++){
      if(!vis[i]) dfs(i);
    }
    reverse(ALL(ord));
    memset(vis, 0, sizeof(vis));
    for(auto i: ord){
      if(!vis[i]){
        rdfs(i);
        num_++;
      }
    }
  }
  inline int get_id(int x){return num[x];}
  inline int count(){return num_;}
} scc;
```

3.4 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;
    }
};
```

3.5 MinimumCostMaximumFlow

```
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(){
    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;
```

3.6 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

```
// 2-SAT solver based on Kosaraju's algorithm.
// Variables are 0-based. Positive variables are stored
     in vertices 2n, corresponding negative variables
    in 2n+1
// TODO: This is quite slow (3x-4x slower than Gabow's
    algorithm)
struct TwoSat {
  int n;
  vector<vector<int> > adj, radj, scc;
  vector<int> sid, vis, val;
  stack<int> stk;
  int scnt;
  // n: number of variables, including negations
  TwoSat(int n): n(n), adj(n), radj(n), sid(n), vis(n),
       val(n, -1) {}
  // adds an implication
  void impl(int x, int y) { adj[x].push_back(y); radj[y
      ].push_back(x); }
  // adds a disjunction
```

```
  void vee(int x, int y) { impl(x^1, y); impl(y^1, x);
      }
  // forces variables to be equal
  void eq(int x, int y) { impl(x, y); impl(y, x); impl(
      x^1, y^1); impl(y^1, x^1); }
  // forces variable to be true
  void tru(int x) { impl(x^1, x); }
  void dfs1(int x) {
    if (vis[x]++) return;
    for (int i = 0; i < adj[x].size(); i++) {
      dfs1(adj[x][i]);
    }
    stk.push(x);
  }
  void dfs2(int x) {
    if (!vis[x]) return; vis[x] = 0;
    sid[x] = scnt; scc.back().push_back(x);
    for (int i = 0; i < radj[x].size(); i++) {
      dfs2(radj[x][i]);
    }
  }
  // returns true if satisfiable, false otherwise
  // on completion, val[x] is the assigned value of
      variable x
  // note, val[x] = 0 implies val[x^1] = 1
  bool two_sat() {
    scnt = 0;
    for (int i = 0; i < n; i++) {
      dfs1(i);
    }
    while (!stk.empty()) {
      int v = stk.top(); stk.pop();
      if (vis[v]) {
        scc.push_back(vector<int>());
        dfs2(v);
        scnt++;
      }
    }
    for (int i = 0; i < n; i += 2) {
      if (sid[i] == sid[i+1]) return false;
    }
    vector<int> must(scnt);
    for (int i = 0; i < scnt; i++) {
      for (int j = 0; j < scc[i].size(); j++) {
        val[scc[i][j]] = must[i];
        must[sid[scc[i][j]^1]] = !must[i];
      }
    }
    return true;
  }
};
```

3.9 HeavyLightDecomp

```
#define REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
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 prt[MAXN][LOG], head[MAXN];
  // head[ u ] : head of the chain contains u
  void dfssz(int u, int p){
    dep[u] = dep[p] + 1;
    prt[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){
    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 != prt[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);
    int diff = dep[b] - dep[a];
    REPD(k, LOG-1, 0) if(diff & (1<<k)){
      b = prt[b][k];
    }
    if(a == b) return a;
    REPD(k, LOG-1, 0) if(prt[a][k] != prt[b][k]){
      a = prt[a][k]; b = prt[b][k];
    }
    return prt[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)
      prt[i][k] = prt[prt[i][k-1]][k-1];
  }
  vector< PII > getPath( int u , int v ){
    vector< PII > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
      res.push_back( PII(tid[ head[ v ] ] , tid[ v ]) )
          ;
      v = prt[ head[ v ] ][ 0 ];
    }
    res.push_back( PII( 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< PII >& path = tree.getPath( u , v )
     * for( PII 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;
```

3.10 MaxClique

```
struct MaxClique {
    int n, deg[maxn], ans;
    bitset<maxn> adj[maxn];
    vector<pair<int, int>> edge;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) adj[i].reset();
        for (int i = 0; i < n; ++i) deg[i] = 0;
        edge.clear();
    }
    void add_edge(int a, int b) {
        edge.emplace_back(a, b);
        ++deg[a]; ++deg[b];
    }
    int solve() {
        vector<int> ord;
        for (int i = 0; i < n; ++i) ord.push_back(i);
        sort(ord.begin(), ord.end(), [&](const int &a,
            const int &b) { return deg[a] < deg[b]; });
        vector<int> id(n);
        for (int i = 0; i < n; ++i) id[ord[i]] = i;
        for (auto e : edge) {
            int u = id[e.first], v = id[e.second];
            adj[u][v] = adj[v][u] = true;
```

```
        }
        bitset<maxn> r, p;
        for (int i = 0; i < n; ++i) p[i] = true;
        ans = 0;
        dfs(r, p);
        return ans;
    }
    void dfs(bitset<maxn> r, bitset<maxn> p) {
        if (p.count() == 0) return ans = max(ans, (int)
            r.count()), void();
        if ((r | p).count() <= ans) return;
        int now = p._Find_first();
        bitset<maxn> cur = p & ~adj[now];
        for (now = cur._Find_first(); now < n; now =
            cur._Find_next(now)) {
            r[now] = true;
            dfs(r, p & adj[now]);
            r[now] = false;
            p[now] = false;
        }
    }
};
```

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

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

4.3 ax+by=gcd

```
// By Adrien1018 (not knowing how to use.
// ax+ny = 1, ax+ny == ax == 1 (mod n)
tuple<int, int, int> extended_gcd(int a, int b) {
  if (!b) return make_tuple(a, 1, 0);
  int d, x, y;
  tie(d, x, y) = extended_gcd(b, a % b);
  return make_tuple(d, y, x - (a / b) * y);
}
```

4.4 Pollard Rho

```
// does not work when n is prime
lld modit(lld x,lld mod) {
  if(x>=mod) x-=mod;
  //if(x<0) x+=mod;
  return x;
}
lld mult(lld x,lld y,lld mod) {
  lld s=0,m=x%mod;
  while(y) {
    if(y&1) s=modit(s+m,mod);
    y>>=1;
    m=modit(m+m,mod);
  }
  return s;
}
lld f(lld x,lld mod) {
  return modit(mult(x,x,mod)+1,mod);
}
lld pollard_rho(lld n) {
  if(!(n&1)) return 2;
  while (true) {
    lld 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;
  }
}
```

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 NloglogN Sieve

```
void Sieve(int n){
  for(int i=2;i<=n;i++){
    if(notprime[i]) continue;
    primes.push_back(i);
    for(int j=i*i;j<=n;j+=i) notprime[i]=true;
  }
}
```

4.7 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;
  }
}
```

4.8 Miller Rabin

```
lld modu(lld a, lld m){
  while(a >= m) a -= m;
  return a;
}
lld mul(lld a, lld b, lld m){
  if(a < b) swap(a, b);
  lld ret = 0;
  while(b){
    if(b & 1) ret = modu(ret+a, m);
    a = modu(a+a, m);
    b >>= 1;
  }
  return ret;
}
lld qPow(lld a, lld k, lld m){
  lld ret = 1;
  a %= m;
  while(k){
    if(k & 1) ret = mul(ret, a, m);
    a = mul(a, a, m);
    k >>= 1;
  }
  return modu(ret, m);
}
bool witness(lld a, lld s, int t, lld n){
  lld b = qPow(a, s, n);
  if(b == 0) return false;
  while(t--){
    lld bb = mul(b, b, n);
    if(bb == 1 and b != 1 and b != n-1) return true;
    b = bb;
  }
  return b != 1;
}
bool miller_rabin(lld n){
  if(n < 2) return false;
  if(!(n & 1)) return (n==2);
  lld x = n-1; int t = 0;
  while(!(x&1)) x >>= 1, t++;
  lld sprp[] =
      {2,325,9375,28178,450775,9780504,1795265022};
  for(int i=0;i<7;i++){
    if(witness(sprp[i]%n, x, t, n)) return false;
  }
  return true;
}
```

4.9 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;
}
```

4.10 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;
      }
    }
  }
}
```

4.11 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;
}
```

4.12 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;
}
```

4.13 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)
*/
```

4.14 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];
}
```

4.15 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;
```

4.16 DiscreteLog

```cpp
// Baby-step Giant-step Algorithm
// a x + by = g
void exgcd(long long  x, long long y, long long &g,
    long long &a, long long &b) {
  if (y == 0)
    g = x, a = 1, b = 0;
  else
    exgcd(y, x%y, g, b, a), b -= (x/y) * a;
}
long long inverse(long long x, long long p) {
  long long g, b, r;
  exgcd(x, p, g, r, b);
  if (g < 0)  r = -r;
  return (r%p + p)%p;
}
long long BSGS(long long P, long long B, long long N) {
  // find B^L = N mod P
  unordered_map<long long, int> R;
  long long sq = (long long) sqrt(P);
  long long t = 1, f;
  for (int i = 0; i < sq; i++) {
    if (t == N)
      return i;
    if (!R.count(t))
      R[t] = i;
    t = (t * B) % P;
  }

  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;
}
```

5 Geometry

5.1 Point Class

```cpp
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

```cpp
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

```cpp
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 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';
*/
```

6.4 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;
    }
  }
}
```

6.5 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);
}
```

6.6 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 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;
}
```

7.2 DP-opt Condition

7.2.1 totally monotone (concave/convex)

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

7.2.2 monge condition (concave/convex)

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

7.3 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&lt;segment&gt; 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);
  }
}
```