Contents

1 Basic 1
- 1.1 Default Code . . . 1
- 1.2 IncreaseStackSize . . . 2
- 1.3 Pragma optimization . . . 2
- 1.4 Quick Random . . . 2
- 1.5 IO Optimization . . . 2

2 Data Structure 2
- 2.1 Bigint . . . 2
- 2.2 Fraction . . . 3
- 2.3 ScientificNotation . . . 4
- 2.4 unordered_map . . . 4
- 2.5 extc_balance_tree . . . 4
- 2.6 extc_heap . . . 4
- 2.7 PairingHeap . . . 5
- 2.8 Disjoint Set . . . 5
- 2.9 Treap . . . 5
- 2.10 SparseTable . . . 6
- 2.11 FenwickTree . . . 6

3 Graph 7
- 3.1 BCC Edge . . . 7
- 3.2 BCC Vertex . . . 7
- 3.3 Strongly Connected Components . . . 7
- 3.4 Articulation Point . . . 8
- 3.5 Bipartie Matching . . . 8
- 3.6 MinimumCostMaximumFlow . . . 8

4 Math 9
- 4.1 Prime Table . . . 9
- 4.2 ax+by=gcd . . . 9
- 4.3 Pollard Rho . . . 9
- 4.4 Linear Sieve . . . 9
- 4.5 NloglogN Sieve . . . 9
- 4.6 Range Sieve . . . 9
- 4.7 Miller Rabin . . . 10
- 4.8 Inverse Element . . . 10
- 4.9 Fast Fourier Transform . . . 10
- 4.10 NTT . . . 11

5 Geometry 11
- 5.1 Point Class . . . 11
- 5.2 2D Convex Hull . . . 12
- 5.3 SimulateAnnealing . . . 13

6 Stringology 13
- 6.1 Hash . . . 13
- 6.2 Suffix Array . . . 13
- 6.3 KMP . . . 14

1 Basic

1.1 Default Code

```cpp
#include <iostream>
#include <iomanip>
#include <string>
#include <algorithm>
#include <vector>
#include <queue>
#include <bitset>
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <random>
#include <utility>
#include <stack>
#include <sstream>
#include <functional>
#include <deque>
#include <cassert>
using namespace std;
/* include everything for Kotori~ <3 */

typedef long long lld;
typedef unsigned long long llu;
typedef long double llf;
typedef pair<int,int> PII;
typedef pair<int,lld> PIL;
typedef pair<lld,int> PLI;
typedef pair<lld,lld> PLL;
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 some types for Ruby! */

#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--)
/* make code shorter for Di~a~ */

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)); }
/* make output easier for Ainyan~n~ */

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...);}
}
/* Let's debug with Nico~Nico~Ni */

int main(int argc, char* argv[]){
  if(argc>1 and string(argv[1])=="-D") debug=1;
  if(!debug){ios_base::sync_with_stdio(0);cin.tie(0);}
  return 0;
}
```

1.2 IncreaseStackSize

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

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

1.3 Pragma optimization

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

1.4 Quick Random

```
// PRNG {{{
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, 0
    xBF58476D1CE4E5B9, 0x94D049BB133111EB, 30, 27, 31>;
R64 r64;
// }}}
```

1.5 IO Optimization

```
// I/O optimization start {{{
static inline int_fast32_t fastAtoi(const char *p,
    uint_fast32_t len) {
  uint_fast32_t res = 0;
  uint_fast8_t neg = *p == '-';
  if (neg) p++, len--;
  switch (len) {
    case 10: res += (*p++ & 15) * 1000000000;
    case 9: res += (*p++ & 15) * 100000000;
    case 8: res += (*p++ & 15) * 10000000;
    case 7: res += (*p++ & 15) * 1000000;
    case 6: res += (*p++ & 15) * 100000;
    case 5: res += (*p++ & 15) * 10000;
    case 4: res += (*p++ & 15) * 1000;
    case 3: res += (*p++ & 15) * 100;
    case 2: res += (*p++ & 15) * 10;
    case 1: res += (*p & 15);
  }
  return res * (neg ? -1 : 1);
}

static inline bool getRawChar(char *c) {
  static char buf[1 << 20], *p = buf, *end = buf;
  if (p == end) {
    if ((end = buf + fread(buf, 1, 1 << 20, stdin)) ==
        buf) return false;
    p = buf;
  }
  *c = *p++;
  return true;
}

static inline bool getInt(int32_t *x) {
  static char buf[12];
  uint_fast32_t i = 0;
  while (getRawChar(buf + i)) {
    if ((unsigned)(buf[i] - '0') > 10U && buf[i] != '-'
        ) {
      if (i) break;
      else continue;
    }
    i++;
  }
  if (!i) return false;
  *x = fastAtoi(buf, i);
  return true;
}
// I/O optimization end }}}
```

2 Data Structure

2.1 Bigint

```
struct Bigint{
  static const int LEN = 60;
  static const int BIGMOD = 10000;

  int s;
  int vl, v[LEN];
  //  vector<int> v;
  Bigint() : s(1) { vl = 0; }
  Bigint(long long a) {
    s = 1; vl = 0;
    if (a < 0) { s = -1; a = -a; }
    while (a) {
      push_back(a % BIGMOD);
      a /= BIGMOD;
    }
  }
  Bigint(string str) {
    s = 1; vl = 0;
    int stPos = 0, num = 0;
    if (!str.empty() && str[0] == '-') {
      stPos = 1;
      s = -1;
    }
    for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
      num += (str[i] - '0') * q;
      if ((q *= 10) >= BIGMOD) {
        push_back(num);
        num = 0; q = 1;
      }
    }
    if (num) push_back(num);
    n();
  }

  int len() const {
    return vl;
    //    return SZ(v);
  }
  bool empty() const { return len() == 0; }
  void push_back(int x) {
```

```
  v[vl++] = x;
  //    v.PB(x);
}
void pop_back() {
  vl--;
  //    v.pop_back();
}
int back() const {
  return v[vl-1];
  //    return v.back();
}
void n() {
  while (!empty() && !back()) pop_back();
}
void resize(int nl) {
  vl = nl;
  fill(v, v+vl, 0);
  //    v.resize(nl);
  //    fill(ALL(v), 0);
}

void print() const {
  if (empty()) { putchar('0'); return; }
  if (s == -1) putchar('-');
  printf("%d", back());
  for (int i=len()-2; i>=0; i--) printf("%.4d",v[i]);
}
friend std::ostream& operator << (std::ostream& out,
    const Bigint &a) {
  if (a.empty()) { out << "0"; return out; }
  if (a.s == -1) out << "-";
  out << a.back();
  for (int i=a.len()-2; i>=0; i--) {
    char str[10];
    snprintf(str, 5, "%.4d", a.v[i]);
    out << str;
  }
  return out;
}

int cp3(const Bigint &b)const {
  if (s != b.s) return s - b.s;
  if (s == -1) return -(-*this).cp3(-b);
  if (len() != b.len()) return len()-b.len();//int
  for (int i=len()-1; i>=0; i--)
    if (v[i]!=b.v[i]) return v[i]-b.v[i];
  return 0;
}

bool operator < (const Bigint &b)const{ return cp3(b)
    <0; }
bool operator <= (const Bigint &b)const{ return cp3(b
    )<=0; }
bool operator == (const Bigint &b)const{ return cp3(b
    )==0; }
bool operator != (const Bigint &b)const{ return cp3(b
    )!=0; }
bool operator > (const Bigint &b)const{ return cp3(b)
    >0; }
bool operator >= (const Bigint &b)const{ return cp3(b
    )>=0; }

Bigint operator - () const {
  Bigint r = (*this);
  r.s = -r.s;
  return r;
}
Bigint operator + (const Bigint &b) const {
  if (s == -1) return -(-(*this)+(-b));
  if (b.s == -1) return (*this)-(-b);
  Bigint r;
  int nl = max(len(), b.len());
  r.resize(nl + 1);
  for (int i=0; i<nl; i++) {
    if (i < len()) r.v[i] += v[i];
    if (i < b.len()) r.v[i] += b.v[i];
    if(r.v[i] >= BIGMOD) {
      r.v[i+1] += r.v[i] / BIGMOD;
      r.v[i] %= BIGMOD;
    }
  }
  r.n();
  return r;
}
Bigint operator - (const Bigint &b) const {
  if (s == -1) return -(-(*this)-(-b));
  if (b.s == -1) return (*this)+(-b);
  if ((*this) < b) return -(b-(*this));
  Bigint r;
  r.resize(len());
  for (int i=0; i<len(); i++) {
    r.v[i] += v[i];
    if (i < b.len()) r.v[i] -= b.v[i];
    if (r.v[i] < 0) {
      r.v[i] += BIGMOD;
      r.v[i+1]--;
    }
  }
  r.n();
  return r;
}
Bigint operator * (const Bigint &b) {
  Bigint r;
  r.resize(len() + b.len() + 1);
  r.s = s * b.s;
  for (int i=0; i<len(); i++) {
    for (int j=0; j<b.len(); j++) {
      r.v[i+j] += v[i] * b.v[j];
      if(r.v[i+j] >= BIGMOD) {
        r.v[i+j+1] += r.v[i+j] / BIGMOD;
        r.v[i+j] %= BIGMOD;
      }
    }
  }
  r.n();
  return r;
}
Bigint operator / (const Bigint &b) {
  Bigint r;
  r.resize(max(1, len()-b.len()+1));
  int oriS = s;
  Bigint b2 = b; // b2 = abs(b)
  s = b2.s = r.s = 1;
  for (int i=r.len()-1; i>=0; i--) {
    int d=0, u=BIGMOD-1;
    while(d<u) {
      int m = (d+u+1)>>1;
      r.v[i] = m;
      if((r*b2) > (*this)) u = m-1;
      else d = m;
    }
    r.v[i] = d;
  }
  s = oriS;
  r.s = s * b.s;
  r.n();
  return r;
}
Bigint operator % (const Bigint &b) {
  return (*this)-(*this)/b*b;
}
};
```

2.2 Fraction

```
/************************
 n為分子，d為分母
 若分數為0則n=0,d=1
 若為負數則負號加在分子
 必定約到最簡分數
************************/
#ifndef SUNMOON_FRACTION
#define SUNMOON_FRACTION
#include<algorithm>
template<typename T>
struct fraction{
  T n,d;
  fraction(const T &_n=0,const T &_d=1):n(_n),d(_d){
    T t=std::__gcd(n,d);
    n/=t,d/=t;
    if(d<0)n=-n,d=-d;
  }
  fraction operator-()const{
    return fraction(-n,d);
  }
  fraction operator+(const fraction &b)const{
    return fraction(n*b.d+b.n*d,d*b.d);
  }
  fraction operator-(const fraction &b)const{
```

```
    return fraction(n*b.d-b.n*d,d*b.d);
  }
  fraction operator*(const fraction &b)const{
    return fraction(n*b.n,d*b.d);
  }
  fraction operator/(const fraction &b)const{
    return fraction(n*b.d,d*b.n);
  }
  fraction operator+=(const fraction &b){
    return *this=fraction(n*b.d+b.n*d,d*b.d);
  }
  fraction operator-=(const fraction &b){
    return *this=fraction(n*b.d-b.n*d,d*b.d);
  }
  fraction operator*=(const fraction &b){
    return *this=fraction(n*b.n,d*b.d);
  }
  fraction operator/=(const fraction &b){
    return *this=fraction(n*b.d,d*b.n);
  }
  bool operator <(const fraction &b)const{
    return n*b.d<b.n*d;
  }
  bool operator >(const fraction &b)const{
    return n*b.d>b.n*d;
  }
  bool operator ==(const fraction &b)const{
    return n*b.d==b.n*d;
  }
  bool operator <=(const fraction &b)const{
    return n*b.d<=b.n*d;
  }
  bool operator >=(const fraction &b)const{
    return n*b.d>=b.n*d;
  }
};
#endif
```

2.3 ScientificNotation

```
#include <cmath>
#include <cstdio>
#include <iostream>
#include <algorithm>

struct SciFi{
    typedef double base_t;
  base_t x; int p;
  SciFi(){x=0;p=0;}
  SciFi(base_t k){
    p = floor(log10(k));
    x = k / pow((base_t)10, p);
  }
  SciFi(base_t a, int b){
    x=a;p=b;
  }
  SciFi operator=(base_t k){
    p = floor(log10(k));
    x = k / pow((base_t)10, p);
    return *this;
  }
  SciFi operator*(SciFi k)const{
    int nP = p+k.p;
    base_t nX = x*k.x;
    int tp = floor(log10(nX));
    return SciFi(nX/pow((base_t)10, tp), nP+tp);
  }
  SciFi operator*=(SciFi k){
    p+=k.p;
    x*=k.x;
    int tp = floor(log10(x));
    p+=tp;
    x/=pow((base_t)10, tp);
    return *this;
  }
  SciFi operator+(SciFi k)const{
    int newP = std::min(k.p, p);
    base_t x1 = x*pow((base_t)10, p-newP);
    base_t x2 = k.x*pow((base_t)10, k.p-newP);
    x1+=x2;
    int tp = floor(log10(x1));
    newP+=tp;
    x1 /= pow((base_t)10, tp);
    return SciFi(x1, newP);
  }
  SciFi operator+=(SciFi k){
    int newP = std::min(k.p, p);
    base_t x1 = x*pow((base_t)10, p-newP);
    base_t x2 = k.x*pow((base_t)10, k.p-newP);
    x1+=x2;
    int tp = floor(log10(x1));
    newP+=tp;
    x1 /= pow((base_t)10, tp);
    x=x1;p=newP;
    return *this;
  }
  bool operator<(SciFi a)const{
    if(p == a.p) return x<a.x;
    return p<a.p;
  }
  bool operator>(SciFi a)const{
    if(p == a.p) return x>a.x;
    return p>a.p;
  }
  bool operator==(SciFi a)const{
    return p==a.p and x==a.x;
  }
};

int main(){
  double a; scanf("%lf",&a);
  SciFi aa=a, x;
  x = aa*SciFi(2);
  printf("%.2lfe%c%03d\n", x.x, "+-"[x.p<0], abs(x.p));
  return 0;
}
```

2.4 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>;
```

2.5 extc_balance_tree

```
#include <functional>
#include <ext/pb_ds/assoc_container.hpp>
using std::less;
using std::greater;
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;
}
```

2.6 extc_heap

```
#include <functional>
#include <ext/pb_ds/assoc_container.hpp>
```

```cpp
#include <ext/pb_ds/priority_queue.hpp>
using std::less;
using std::greater;
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;
}
```

2.7 PairingHeap

```cpp
#include <vector>
using std::vector;

template<class __type, class __cmp=less<__type>>
class pairingHeap{
  private:
      struct pairingNode{
          __type val;
            vector<pairingNode*> child;
            pairingNode(){
                val = 0;
                child.clear();
            }
            pairingNode(int x): val(x){
                child.clear();
            }
      };
    pairingNode* root;
    int count;
    __cmp _cp;
    void remove(pairingNode* cur){
        if(cur==nullptr) return;
        for(auto i: cur->child) remove(i);
        delete cur;
    }
  public:
    pairingHeap(){root=nullptr;count=0;}
    inline bool empty(){return count==0;}
    inline __type top(){return root->val;}
    inline int size(){return count;}
    inline void clear(){remove(root);root=nullptr;count
        =0;}
    inline void push(__type a){
      count++;
      auto mynode = new pairingNode(a);
      if(root==nullptr) root = mynode;
      else{
        if(_cp(root->val, mynode->val)) swap(root,
            mynode);
        root->child.push_back(mynode);
      }
    }
    inline void pop(){
      count--;
      queue<pairingNode*> que;
      for(auto i:root->child) que.push(i);
      delete root;
      while(que.size() > 1){
        auto tp1=que.front();que.pop();
        auto tp2=que.front();que.pop();
        if(_cp(tp1->val, tp2->val)) swap(tp1, tp2);
        tp1->child.push_back(tp2);
        que.push(tp1);
      }
      if(que.empty()) root=nullptr;
      else root = que.front();
    }
    inline void join(pairingHeap<__type, __cmp>& pq2){
        if(_cp(root->val, pq2.root->val)) swap(root,
            pq2.root);
        root->child.push_back(pq2.root);
      count += pq2.count;
        pq2.root = nullptr;
        pq2.count = 0;
    }
};

int main(){
  pairingHeap<int> pq1, pq2;
  for(int i=0;i<1e5;i++) pq1.push(i);
  for(int i=1e5;i<2e5;i++) pq2.push(i);
  pq1.join(pq2);
  while(!pq1.empty()){
    // cout<<pq1.top()<<" ";
    pq1.pop();
  }
  return 0;
}
```

2.8 Disjoint Set

```cpp
class DJS{
  private:
    vector<int> fa, sz, sv;
    vector<pair<int*, int>> opt;
    inline void assign(int *k, int v){
      opt.PB({k, *k});
      *k = v;
    }
  public:
    inline void init(int n){
      fa.resize(n);
      sz.resize(n);
      for(int i=0;i<n;i++){
        fa[i] = i;
        sz[i] = 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.PB(SZ(opt));}
    inline void undo(){
      int ls = sv.back(); sv.pop_back();
      while(SZ(opt) > ls){
        pair<int*, int> cur=opt.back();
        *cur.FF = cur.SS;
        opt.pop_back();
      }
    }
};
```

2.9 Treap

```cpp
#include <cstdlib>

class Treap{
  private:
    const int MEM = 500000 + 5;
    unsigned seed;
    inline unsigned myrand(){
      static unsigned seed = time(NULL);
      seed = seed*seed*127 + seed*227 + 2147483587;
      seed ^= seed*97;
      seed /= 7123;
      return seed;
    }
    struct node{
      node *lc, *rc;
```

```
    int pri, size, val;
    node(){}
    node(int x):
      lc(nullptr),
      rc(nullptr),
      pri(myrand()),
      size(1),
      val(x)
      {}
    inline void pull(){
      size = 1;
      if(lc) size += lc->size;
      if(rc) size += rc->size;
    }
  } *root, pool[MEM];
  int mem_;
  inline node* new_node(int x){
    static int mem_ = 0;
    assert(mem_ < MEM);
    pool[mem_]=node(x);
    return &pool[mem_++];
  }
  inline int sz(node* x){return x?x->size:0;}
  node* merge(node *a, node *b){
    if(!a or !b) return a?a:b;
    if(a->pri > b->pri){
      a->rc = merge(a->rc, b);
      a->pull();
      return a;
    }else{
      b->lc = merge(a, b->lc);
      b->pull();
      return b;
    }
  }
  void split(Treap* t, int k, Treap*& a, Treap*& b){
    if(!t) a=b=nullptr;
    else if(sz(t->lc) < k){
      a = t;
      split(t->rc, k - sz(t->lc) - 1 , a->rc, b);
      a->pull();
    }else{
      b = t;
      split(t->lc, k, a, b->lc);
      b->pull();
    }
  }
  int oOk(node* rr, int x){
    if(rr==NULL)return 0;
    if((rr->val) < x)return gSize(rr->l)+oOk(rr->r, x
        )+1;
    else return oOk(rr->l, x);
  }
 public:
  Treap(){root=nullptr;seed=time(NULL);mem_=0;}
  void do_something_at(int l, int r){
    // 1-base [l, r]
    split(root, l-1, tl, root);
    split(root, r-l+1, root, tr);
    root = merge(tl, merge(root, tr));
  }
  void insert(int x){
    node *a, *b;
    split(root, x, a, b);
    root = merge(merge(a, new node(x)), b);
    root->size = gSize(root->l)+gSize(root->r)+1;
  }
  void remove(int x){
    //need debug may contain bugs
    node *a, *b, *c, *d;
    split(root, x, a, b);
    a->size = gSize(a->l)+gSize(a->r);
    split(a, x-1, c, d);
    root = merge(b, c);
    root->size = gSize(root->l)+gSize(root->r);
    delete d;
  }
  int order_of_key(int x){return oOk(root,x);}
};

int main(){

  return 0;
}
```

2.10 SparseTable

```
#include <algorithm>
using std::min;

const int N = 1<<20;
const int LOG_N = 21;

class SparseTable{
  private:
    int table[N][LOG_N];
  public:
    void build(int n, int arr[]){
      // [1, n]
      for(int i=1;i<=n;i++) table[i][0] = arr[i];
      for(int j=1;(1<<j)<=n;j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
          table[i][j] = min(table[i][j-1], table[i
              +(1<<(j-1))][j-1]);
        }
      }
    }
    int query(int l, int r){
      // 1-base [l, r]
      int k = 31-__builtin_clz(r-l+1);
      return min(table[l][k], table[r-(1<<k)+1][k]);
    }
};
```

2.11 FenwickTree

```
#include <vector>
using std::vector;

template<typename T>
class BIT{
    #define ALL(x) begin(x), end(x)
  private:
    vector<T> arr;
    int n;
    inline int lowbit(int x){return x & (-x);}
    T query(int x){
      T ret = 0;
      while(x > 0){
        ret += arr[x];
        x -= lowbit(x);
      }
      return ret;
    }
  public:
    void init(int n_){
      n = n_;
      arr.resize(n);
      fill(ALL(arr), 0);
    }
    void modify(int pos, T v){
      while(pos < n){
        arr[pos] += v;
        pos += lowbit(pos);
      }
    }
    T query(int l, int r){
      // 1-base (l, r]
      return query(r) - query(l);
        }
    #undef ALL
};

template<typename T>
class BIT{
    #define ALL(x) begin(x), end(x)
  private:
        vector<T> arr;
        int n;
        inline int lowbit(int x){return x & (-x);}
        void add(int s, int v){
      while(s){
        arr[s]+=v;
        s-=lowbit(s);
      }
    }
  public:
    void init(int n_){
```

```
            n = n_;
            arr.resize(n);
            fill(ALL(arr), 0);
    }
    void add(int l, int r, T v){
            //1-base (l, r]
      add(l, -v);
      add(r, v);
    }
    T query(int x){
      T r=0;
      while(x<size){
        r+=arr[x];
        x+=lowbit(x);
      }
      return r;
        }
    #undef ALL
};
```

3 Graph

3.1 BCC Edge

```
class BCC{
  private:
    int low[N], dfn[N], cnt;
    bool bcc[N];
    vector<PII> G[N];
    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){
      for(int i=0;i<n;i++) G[i].clear();
      fill(bcc, bcc+m, false);
      cnt = 0;
    }
    void add_edge(int u, int v){
      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

```
struct BccVertex {
  int n,nBcc,step,root,dfn[MXN],low[MXN];
  vector<int> E[MXN], ap;
  vector<pii> bcc[MXN];
  int top;
  pii stk[MXN];
  void init(int _n) {
    n = _n;
    nBcc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v) {
    E[u].PB(v);
    E[v].PB(u);
  }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    int son = 0;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        son++;
        stk[top++] = {u,v};
        DFS(v,u);
        if (low[v] >= dfn[u]) {
          if(v != root) ap.PB(v);
          do {
            assert(top > 0);
            bcc[nBcc].PB(stk[--top]);
          } while (stk[top] != pii(u,v));
          nBcc++;
        }
        low[u] = min(low[u], low[v]);
      } else {
        if (dfn[v] < dfn[u]) stk[top++] = pii(u,v);
        low[u] = min(low[u],dfn[v]);
      }
    }
    if (u == root && son > 1) ap.PB(u);
  }
  // return the edges of each bcc;
  vector<vector<pii>> solve() {
    vector<vector<pii>> res;
    for (int i=0; i<n; i++) {
      dfn[i] = low[i] = -1;
    }
    ap.clear();
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) {
        top = 0;
        root = i;
        DFS(i,i);
      }
    }
    REP(i,nBcc) res.PB(bcc[i]);
    return res;
  }
}
```

3.3 Strongly Connected Components

```
class SCC{
  private:
    int n, num_;
    vector<int> G[N], rG[N], 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;
      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 Articulation Point

```
#include <bits/stdc++.h>
using namespace std;
#define N 1000000+5

class AP{
  private:
    vector<int> graph[N];
    bitset<N> visited, result;
    int low[N], lv[N];
    void dfs(int x, int f, int cnt){
      low[x]=cnt;
      lv[x]=cnt;
      visited[x]=1;
      int child=0;
      for(auto i:graph[x]){
        if(i!=f){
          if(visited[i]){
            low[x] = min(low[x], low[i]);
          }else{
            child++;
            dfs(i,x,cnt+1);
            low[x] = min(low[x], low[i]);
            if(low[i] >= lv[x]) result[x]=1;
          }
        }
      }
      if(lv[x]==1 && child <= 1)
        result[x]=0;
    }
  public:
    void init(int sz){
      for(int i=0;i<sz;i++) graph[i].clear();
      visited.reset(); result.reset();
    }
    void AddEdge(int u, int v){
      graph[u].push_back(v);
      graph[v].push_back(u);
    }
    void solve(){
      dfs(1, 1, 1);
    }
    bool isAP(int x){
      return result[x];
    }
} ap;

int main(){
  int n,m;cin>>n>>m;
  ap.init(n+2);
  for(int i=0;i<m;i++){
    int st,ed;cin>>st>>ed;
    ap.AddEdge(st, ed);
  }
  ap.solve();
  for(int i=1;i<=n;i++) if(ap.isAP(i)) cout<<i<<'\n';
  return 0;
}
```

3.5 Bipartie Matching

```
#include <bits/stdc++.h>
using namespace std;
#define N 500

class BipartieMatching{
  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 AddEdge(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;
    }
};
```

3.6 MinimumCostMaximumFlow

```
class MiniCostMaxiFlow{
  typedef int CapT;
  typedef lld WeiT;
  typedef pair<CapT, WeiT> PCW;
  const CapT INF_CAP = 1<<30;
  const WeiT INF_WEI = 1LL<<60;
  const int MAXV = N;
  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, V;
    vector<Edge> G[MAXV];
    int fa[MAXV], wh[MAXV];
    bool inq[MAXV];
    WeiT dis[MAXV];
    PCW SPFA(){
      for(int i=0;i<V;i++) inq[i]=0;
      for(int i=0;i<V;i++) dis[i]=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 and dis[v] > dis[u]+d){
            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=MAXV){
      V=n;
      ori = a;
      edd = b;
      for(int i=0;i<n;i++) G[i].clear();
    }
    void addEdge(int st, int ed, WeiT w, CapT c){
      G[st].PB(Edge(ed, SZ(G[ed]), w, c));
      G[ed].PB(Edge(st, SZ(G[st])-1, -w, 0));
    }
    PCW solve(){
      CapT cc=0; WeiT ww=0;
      while(true){
        PCW ret = SPFA();
        if(ret.FF==-1) break;
        cc += ret.FF;
        ww += ret.SS;
      }
      return {cc, ww};
    }
} mcmf;
```

4 Math

4.1 Prime Table

```
// 1000000000 < primes < 2147483647
1002939109, 1020288887, 1028798297, 1038684299,
1041211027, 1051762951, 1058585963, 1063020809,
1094763083, 1106384353, 1120154459, 1140593173,
1147930723, 1172520109, 1183835981, 1187659051,
1241251303, 1247184097, 1255940849, 1272759031,
1287027493, 1288511629, 1294632499, 1312650799,
1314753281, 1320080669, 1321970357, 1333133947,
1337684419, 1353508067, 1358715989, 1364961029,
1366046831, 1376536367, 1381705499, 1410637769,
1411311571, 1422795043, 1437499801, 1495803851,
1511764363, 1526710979, 1538018089, 1542373769,
1545326953, 1549429633, 1556212739, 1575971759,
1586465261, 1608336427, 1609783001, 1620728569,
1643267081, 1652401603, 1656717203, 1660920671,
1666858577, 1669260361, 1670240317, 1678791131,
1685583143, 1725964619, 1734856421, 1743134179,
1761537223, 1774260193, 1778872889, 1781930609,
1803000149, 1814256623, 1834876331, 1839154463,
1840044389, 1843241713, 1856039431, 1868564531,
1868732623, 1884198443, 1884616807, 1885059541,
1909942399, 1914471137, 1923951707, 1925453197,
1937719153, 1954649041, 1958915237, 1970709803,
1979612177, 1980446837, 1989761941, 2007826547,
2008033571, 2011186739, 2039465081, 2039728567,
2093735719, 2116097521, 2123852629, 2140170259

// 2147483647 < primes < 4000000000

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,
3384716479, 3386991323
```

4.2 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);
}
// ax+by = gcd (by Eddy1021
PII gcd(int a, int b){
  if(b == 0) return {1, 0};
  PII q = gcd(b, a % b);
  return {q.second, q.first - q.second * (a / b)};
}
```

4.3 Pollard Rho

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

4.4 Linear Sieve

```
const int N = 20000000;
bool sieve[N];

void linear_sieve(){
  vector<int> prime;
  for (int i=2; i<N; i++){
    if (!sieve[i]) prime.push_back(i);
    for (int j=0; i*prime[j]<N; j++)
    {
      sieve[i*prime[j]] = true;
      if (i % prime[j] == 0) break;
    }
  }
}
```

4.5 NloglogN Sieve

```
bool notprime[N];
vector<int> primes;

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

4.6 Range Sieve

```cpp
#include <algorithm>
typedef long long lld;
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,lld);

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

```cpp
// n < 4,759,123,141        3 :  2, 7, 61
// n < 1,122,004,669,633    4 :  2, 13, 23, 1662803
// n < 3,474,749,660,383          6 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
#include <type_traits>
#include <cstdint>
typedef long long lld;

lld mul(lld a, lld b, lld m){
  // return (a*b)%m;
  lld ret = 0;
  while(b){
    if(b&1) ret = (ret+a)%m;
    a = (a+a)%m;
    b>>=1;
  }
  return ret;
}

lld qPow(lld a, lld n, lld m){
  lld ret = 1;
  while(n){
    if(n&1) ret = ret*a%m;
    n >>= 1;
    a = a*a%m;
  }
  return ret%m;
}
bool witness(lld a,lld n,lld u,int t){
  lld x=qPow(a,u,n);
  for(int i=0;i<t;i++) {
    lld nx=mul(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(lld n,int s=100) {
  // iterate s times of witness on n
  // return 1 if prime, 0 otherwise
  if(n<2) return 0;
  if(!(n&1)) return n == 2;
  lld u=n-1; int t=0;
  // n-1 = u*2^t
  while(!(u&1)) u>>=1, t++;
  while(s--){
    lld a=r64()%(n-1)+1;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

4.8 Inverse Element

```cpp
// x's inverse mod k
// if k is prime
long long GetInv(long long x, long long k){
  return qPow(x, k-2);
}

inline long long Euler(long long x){
  long long r=1;
  for(long long 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;
}
// x's inverse mod k
// if k is not prime
long long GetInv(long long x, long long k){
  return qPow(x, Euler(k)-1);
}
// or extended_gcd(x, k).second
// 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 Fast Fourier Transform

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

```
}
```

4.10 NTT

```
typedef long long LL;
// Remember coefficient are mod P
/* p=a*2^n+1
   n    2^n          p           a    root
   5    32           97          3    5
   6    64           193         3    5
   7    128          257         2    3
   8    256          257         1    3
   9    512          7681        15   17
   10   1024         12289       12   11
   11   2048         12289       6    11
   12   4096         12289       3    11
   13   8192         40961       5    3
   14   16384        65537       4    3
   15   32768        65537       2    3
   16   65536        65537       1    3
   17   131072       786433      6    10
   18   262144       786433      3    10 (605028353,
       2308, 3)
   19   524288       5767169     11   3
   20   1048576      7340033     7    3
   21   2097152      23068673    11   3
   22   4194304      104857601   25   3
   23   8388608      167772161   20   3
   24   16777216     167772161   10   3
   25   33554432     167772161   5    3 (1107296257, 33,
       10)
   26   67108864     469762049   7    3
   27   134217728    2013265921  15   31 */
// (must be 2^k)
// To implement poly. multiply:
// NTT<P, root, MAXN> ntt;
// ntt( n , a ); // or ntt.tran( n , a );
// ntt( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// ntt( n , c , 1 );
//
// then you have the result in c :: [LL]

template<LL P, LL root, int MAXN>
struct NTT{
  static LL bigmod(LL a, LL b) {
    LL res = 1;
    for (LL bs = a; b; b >>= 1, bs = (bs * bs) % P) {
      if(b&1) res=(res*bs)%P;
    }
    return res;
  }
  static LL inv(LL a, LL b) {
    if(a==1)return 1;
    return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
  }
  LL omega[MAXN+1];
  NTT() {
    omega[0] = 1;
    LL r = bigmod(root, (P-1)/MAXN);
    for (int i=1; i<=MAXN; i++)
      omega[i] = (omega[i-1]*r)%P;
  }
  // n must be 2^k
  void tran(int n, LL a[], bool inv_ntt=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
      int mh = m >> 1;
      for (int i = 0; i < mh; i++) {
        LL w = omega[i*theta%MAXN];
        for (int j = i; j < n; j += m) {
          int k = j + mh;
          LL x = a[j] - a[k];
          if (x < 0) x += P;
          a[j] += a[k];
          if (a[j] > P) a[j] -= P;
          a[k] = (w * x) % P;
        }
      }
      theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
      for (int k = n >> 1; k > (i ^= k); k >>= 1);
      if (j < i) swap(a[i], a[j]);
    }
    if (inv_ntt) {
      LL ni = inv(n,P);
      reverse( a+1 , a+n );
      for (i = 0; i < n; i++)
        a[i] = (a[i] * ni) % P;
    }
  }
  void operator()(int n, LL a[], bool inv_ntt=false) {
    tran(n, a, inv_ntt);
  }
};

const LL P=2013265921,root=31;
const int MAXN=4194304;
NTT<P, root, MAXN> ntt;
```

5 Geometry

5.1 Point Class

```
namespace Geometry{
  const long double EPS = 1e-8;
  const long double PI = acos((long double)-1);
  template<typename T>
  struct Point{
    typedef long double llf;
    T x, y;
    Point(): x(0), y(0){}
    Point(T _, T __): 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);
    }
    template<class = typename is_floating_point<T>::
        type>
    bool operator==(const Point& o) const {
      return fabs(x-o.x) < EPS and fabs(y-o.y) < EPS;
    }
    bool operator==(const Point& o) const {
      return x==o.x and y==o.y;
```

```
  }
  bool operator!=(const Point& o) const {
    return !(*this == o);
  }
  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;
  }
};
const Point<long double> INF_P(-1e20, 1e20);
const Point<long double> NOT_EXIST(1e20, 1e-20);
template<typename T>
struct Line{
  // 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;
  template<class = typename is_floating_point<T>::
      type>
  bool operator==(const Line& o) const {
    return fabs(a-o.a) < EPS and fabs(b-o.b) < EPS
        and fabs(c-o.b) < EPS;
  }
  bool operator==(const Line& o) const {
    return a==o.a and b==o.b and c==o.c;
  }
  bool operator!=(const Line& o) const {
    return !(*this == o);
  }
  template<class = typename is_floating_point<T>::
      type>
  friend inline bool on_line(const Point<T>& p, const
       Line& l){
    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){
    return l.a*p.x + l.b*p.y + l.c == 0;
  }
  template<class = typename is_floating_point<T>::
      type>
  friend inline bool is_parallel(const Line& x, const
       Line& y){
    return fabs(x.a*y.b - x.b*y.a) < EPS;
  }
  friend inline bool is_parallel(const Line& x, const
       Line& y){
    return x.a*y.b == x.b*y.a;
  }
  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);
}
template<typename T>
struct Segment{
  // p1.x < p2.x
  Line<T> base;
  Point<T> p1, p2;
  Segment(): base(Line<T>()), p1(Point<T>()), p2(
      Point<T>()){
    assert(on_line(p1, base) and on_line(p2, base));
  }
  Segment(Line<T> _, Point<T> __, Point<T> ___): base
      (_), p1(__), p2(___){
    assert(on_line(p1, base) and on_line(p2, base));
  }
  template<typename T2>
  Segment(const Segment<T2>& _): base(_.base), p1(_.
      p1), p2(_.p2) {}
  typedef Point<long double> Pt;
  friend bool on_segment(const Point<T>& p, const
      Segment& l){
    if(on_line(p, l.base))
      return (l.p1.x-p.x)*(p.x-l.p2.x)>=0 and (l.p1.y
          -p.y)*(p.y-l.p2.y)>=0;
    return false;
  }
  friend bool have_inter(const Segment& a, const
      Segment& b){
    if(is_parallel(a.base, b.base)){
      return on_segment(a.p1, b) or on_segment(a.p2,
          b) or on_segment(b.p1, a) or on_segment(b.
          p2, a);
    }
    Pt inter = get_inter(a.base, b.base);
    return on_segment(inter, a) and on_segment(inter,
        b);
  }
  friend inline Pt get_inter(const Segment& a, const
      Segment& b){
    if(!have_inter(a, b)){
      return NOT_EXIST;
    }else if(is_parallel(a.base, b.base)){
      if(a.p1 == b.p1){
        if(on_segment(a.p2, b) or on_segment(b.p2, a)
            ) return INF_P;
        else return a.p1;
      }else if(a.p1 == b.p2){
        if(on_segment(a.p2, b) or on_segment(b.p1, a)
            ) return INF_P;
        else return a.p1;
      }else if(a.p2 == b.p1){
        if(on_segment(a.p1, b) or on_segment(b.p2, a)
            ) return INF_P;
        else return a.p2;
      }else if(a.p2 == b.p2){
        if(on_segment(a.p1, b) or on_segment(b.p1, a)
            ) return INF_P;
        else return a.p2;
      }
      return INF_P;
    }
    return get_inter(a.base, b.base);
  }
  friend ostream& operator<<(ostream& ss, const
      Segment& o){
    ss<<o.base<<", "<<o.p1<<" ~ "<<o.p2;
    return ss;
  }
};
template<typename T>
inline Segment<T> get_segment(const Point<T>& a,
    const Point<T>& b){
  return Segment<T>(get_line(a, b), a, b);
}
};
```

5.2 2D Convex Hull

```
#include <bits/stdc++.h>
using namespace std;
typedef long long lld;
typedef pair<lld, lld> PLL;

template<typename A, typename B>
pair<A, B> operator-(const pair<A, B>& a, const pair<A,
    B>& b){
  return {a.first-b.first, a.second-b.second};
```

```
}

class ConvexHull_2D{
  #define x first
  #define y second
  private:
    vector<PLL> dots, down, up;
    inline lld cross(PLL a, PLL b){
      return a.x*b.y-b.x*a.y;
    }
  public:
    void insert(PLL x){dots.push_back(x);}
    void solve(){
      down.clear();up.clear();
      sort(dots.begin(), dots.end());
      for(auto i: dots){
        while(up.size()>1){
          if(cross(i-up[up.size()-2], up.back()-up[up.
              size()-2]) <= 0) up.pop_back();
          else break;
        }
        up.push_back(i);
      }
      reverse(dots.begin(), dots.end());
      for(auto i: dots){
        while(down.size()>1){
          if(cross(i-down[down.size()-2], down.back()-
              down[down.size()-2]) <= 0) down.pop_back
              ();
          else break;
        }
        down.push_back(i);
      }
      dots.clear();
      dots.insert(dots.end(), down.begin(), down.end())
          ;
      dots.insert(dots.end(), up.begin(), up.end());
      sort(dots.begin(), dots.end());
      dots.resize(distance(dots.begin(), unique(dots.
          begin(), dots.end())));
      down.clear();up.clear();
    }
    vector<PLL> get(){
      return dots;
    }
    bool IsThis(PLL x){
      auto ret = lower_bound(dots.begin(), dots.end(),
          x);
      return *ret==x;
    }
    int count(){return dots.size();}
  #undef x
  #undef y
} cv;


int main(){
  ios_base::sync_with_stdio(0);cin.tie(0);
  int n; cin>>n;
  for(int i=0;i<n;i++){
    lld a,b;cin>>a>>b;
    cv.insert({a, b});
  }
  cv.solve();
  cout<<cv.count()<<'\n';
  return 0;
}
```

5.3 SimulateAnnealing

```
#include <random>
#include <functional>
#include <utility>
#include <algorithm>
using namespace std;

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);
  auto 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;
  }
}

double getY(double x){
    // get y using x
    return x;
}
```

6 Stringology

6.1 Hash

```
#include <string>
typedef long long lld;
const int N = 1000000;
class Hash{
    private:
        const lld p = 127, q = 1208220623;
        int sz;
        lld prefix[N], power[N];
    public:
        void init(const std::string &x){
            sz = x.size();
            prefix[0]=0;
            for(int i=1;i<=sz;i++) prefix[i]=((prefix[i
                -1]*p)%q+x[i-1])%q;
            power[0]=1;
            for(int i=1;i<=sz;i++) power[i]=(power[i
                -1]*p)%q;
        }
        lld query(int l, int r){
            // 1-base (l, r]
            return (prefix[r] - (prefix[l]*power[r-l])%
                q + q)%q;
        }
};
```

6.2 Suffix Array

```
//help by http://www.geeksforgeeks.org/suffix-array-set
    -2-a-nlognlogn-algorithm/
#include <bits/stdc++.h>
using namespace std;
#define PB push_back

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();
void radixSort();
```

```
int main(){
  gets(str);
  len = strlen(str);
  SA();
  for(int i=0;i<len;i++){
    printf("%d\n",sa[i].index);
  }
  return 0;
}

void SA(){
  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];
int match(const std::string& A, const std::string& B) {
  F[0] = -1, F[1] = 0;
  for (int i=1, j=0; i < B.size()-1; F[++i] = ++j) { //
       calculate failure function
    if (B[i] == B[j]) F[i] = F[j]; // optimization by
         Knuth, may not need this
    while (j != -1 && B[i] != B[j]) j = F[j];
  }
  for (int i=0, j=0; i-j+B.size() <= A.size(); i++, j
      ++) { // match
    while (j != -1 && A[i] != B[j]) j = F[j];
    if (j == B.size() - 1) return i - j; // match
        successfully at string B's end return result
  }
  return -1;
}
```