

Contents

1 Basic	1
1.1 Default Code	1
1.2 IncreaseStackSize	2
1.3 Pragma optimization	2
2 Data Structure	2
2.1 BigInt	2
2.2 Fraction	3
2.3 ScientificNotation	3
2.4 unordered_map	4
2.5 extc_balance_tree	4
2.6 extc_heap	4
2.7 PairingHeap	4
2.8 Disjoint Set	5
2.9 Treap	5
2.10 SparseTable	5
2.11 FenwickTree	5
3 Graph	6
3.1 BCC Edge	6
3.2 BCC Vertex	6
3.3 Strongly Connected Components	6
3.4 Articulation Point	7
3.5 Bipartite Matching	7
3.6 MinimumCostMaximumFlow	7
4 Math	8
4.1 ax+by=gcd	8
4.2 Pollard Rho	8
4.3 Linear Sieve	8
4.4 NloglogN Sieve	8
4.5 Miller Rabin	9
4.6 Inverse Element	9
4.7 Fast Fourier Transform	9
4.8 NTT	9
5 Geometry	10
5.1 Point Class	10
5.2 2D Convex Hull	11
5.3 SimulateAnnealing	11
6 Stringology	12
6.1 Hash	12
6.2 Suffix Array	12
6.3 KMP	12

1 Basic

1.1 Default Code

```

#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 lldf;
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")
#pragma GCC target("avx,tune=native")
// or #pragma GCC target ("sse4")
```

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 -(-(*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>
#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

```

#include <vector>
using std::vector;

template<typename T>
class pairingHeap{
private:
    struct pairingNode{
        T val;
        vector<pairingNode*> child;
    };
    pairingNode* root;
    size_t count=0;
public:
    pairingHeap(){root=NULL;count=0;}
    inline bool empty(){return count==0;}
    inline T top(){return root->val;}
    inline size_t size(){return count;}
    inline void push(T a){
        count++;
        if(root==NULL){
            root = new pairingNode;
            root->val=a;
        }else{
            auto temp = new pairingNode;
            temp->val=a;
            if(root->val>temp->val)
                root->child.push_back(temp);
            else{
                temp->child.push_back(root);
                swap(temp,root);
            }
        }
    }
    inline void join(pairingHeap& a){
        count+=a.size();
        auto temp = a.root;
        if(root->val>temp->val){
            root->child.push_back(temp);
        }else{
            temp->child.push_back(root);
            swap(temp,root);
        }
        a.root=nullptr;
        a.count=0;
    }
    inline void pop(){
        count--;
        queue<pairingNode*> QQ;
        for(auto i:root->child) QQ.push(i);
        delete root;
        while(QQ.size()>1){
            pairingNode* tp1=QQ.front();QQ.pop();
            pairingNode* tp2=QQ.front();QQ.pop();
            if(tp1->val>tp2->val){
                tp1->child.push_back(tp2);
                QQ.push(tp1);
            }else{
                tp2->child.push_back(tp1);
                QQ.push(tp2);
            }
        }
        if(QQ.empty()) root=NULL;
        else root = QQ.front();
    }
};

```

```
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

```
class DJS{
private:
    int arr[N];
public:
    int query(int x){
        while(arr[x]!=x) x=arr[x];
        return x;
    }
    int merge(int a, int b){
        arr[query(a)]=query(b);
    }
};
```

2.9 Treap

```
#include <cstdlib>

class Treap{
private:
    struct node{
        node* l;
        node* r;
        int pri, size, val;
        node(){l=NULL; r=NULL; pri=rand(); size=0;}
        node(int x){l=NULL; r=NULL; pri=rand(); size=1; val=x;}
        ~node(){if(l) delete l; if(r) delete r; l=NULL; r=NULL;}
    };
    node* root;
    inline int gSize(node* x){return (x==NULL)?0:(x->size);}
    node* merge(node* x, node* y){
        if(x==NULL || y==NULL) return x?x:y;
        else if(x->pri > y->pri){
            x->r = merge(x->r, y);
            x->size = gSize(x->l)+gSize(x->r)+1;
            return x;
        }else{
            y->l = merge(x, y->l);
            y->size = gSize(y->l)+gSize(y->r)+1;
            return y;
        }
    }
    void split(node* rr, int x, node*& l, node*& r){
        if(rr==NULL) r=l=NULL;
        else if(rr->val <= x){
            l=rr;
            split(rr->r, x, l->r, r);
            l->size = gSize(l->r)+gSize(l->l)+1;
        }else{
            r=rr;
            split(rr->l, x, l, r->l);
            r->size = gSize(r->r)+gSize(r->l)+1;
        }
    }
    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=NULL;}
    ~Treap(){delete root; root=NULL;}
    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){
        // l-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

```

struct BccEdge {
    static const int MXN = 100005;
    struct Edge { int v, eid; };
    int n, m, step, par[MXN], dfn[MXN], low[MXN];
    vector<Edge> E[MXN];
    DisjointSet djs;
    void init(int n) {
        n = n; m = 0;
        for (int i=0; i<n; i++) E[i].clear();
        djs.init(n);
    }
    void add_edge(int u, int v) {
        E[u].PB({v, m});
        E[v].PB({u, m});
        m++;
    }
    void DFS(int u, int f, int f_eid) {
        par[u] = f;
        dfn[u] = low[u] = step++;
        for (auto it:E[u]) {
            if (it.eid == f_eid) continue;
            int v = it.v;
            if (dfn[v] == -1) {
                DFS(v, u, it.eid);
                low[u] = min(low[u], low[v]);
            } else {
                low[u] = min(low[u], dfn[v]);
            }
        }
    }
    void solve() {
        step = 0;
        memset(dfn, -1, sizeof(int)*n);
        for (int i=0; i<n; i++) {
            if (dfn[i] == -1) DFS(i, i, -1);

```

```

        }
        djs.init(n);
        for (int i=0; i<n; i++) {
            if (low[i] < dfn[i]) djs.uni(i, par[i]);
        }
    }
};

```

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 Bipartite Matching

```

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

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 AddEdge(int x, int y){
        X[x].push_back(y);
        Y[y].push_back(x);
    }
    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 $ax+by=\gcd$

```

// By Adrien1018 (not knowing how to use.
//  $ax+ny = 1, ax+ny == ax \pmod n$ 
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)};
}

```

4.2 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;
    }
}

```

4.3 Linear Sieve

```

const int N = 200000000;
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;
        }
    }
}

```

4.4 NloglogN Sieve

```

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

void Sieve(int n){
    // reverse true 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[j]=true;
        }
    }
}

```

4.5 Miller Rabin

```

template<typename T>
inline T pow(T a, T b, T mod){ //  $a^b \pmod{mod}$ 
    T ret=1;
    while(b){
        if(b&1) ret=(ret*a)%mod;
        b>>=1;
        a = (a*a)%mod;
    }
    return ret%mod;
}
int sprp[3]={2,7,61}; // for int range
int llsp[7]={2,325,9375,28178,450775,9780504,1795265022}; //
// at least unsigned long long

```



```

template<typename T>
inline bool isprime(T n, int *sprp, int num) {
    if (n==2) return 1;
    if (n<2 || n%2==0) return 0;
    int t=0;
    T u=n-1;
    for (; u%2==0; ++t) u>>=1;
    for (int i=0; i<num; ++i) {
        T a=sprp[i]%n;
        if (a==0 || a==1 || a==n-1) continue;
        T x=pow(a, u, n);
        if (x==1 || x==n-1) continue;
        for (int j=0; j<t; ++j) {
            x=(x*x)%n;
            if (x==1) return 0;
            if (x==n-1) break;
        }
        if (x==n-1) continue;
        return 0;
    }
    return 1;
}

```

4.6 Inverse Element

```

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

```

4.7 Fast Fourier Transform

```

// 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 = acos(-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;
}

```

4.8 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.c) < 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;  
}
```