Contents

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 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<<'\n';}
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[0])=="-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

```
#include<algorithm>
#include<iostream>
#include<sstream>
#include<iomanip>
#include<vector>
#include<string>
#include<cmath>
using namespace std;
template<typename T>
inline string to_string(const T& x){
   stringstream ss;
   return ss<<x,ss.str();
}
typedef long long LL;
struct bigN:vector<LL>{
   const static int base=1000000000,width=log10(base);
   bool negative;
   bigN(const_iterator a,const_iterator b):vector<LL>(a
       ,b){}
   bigN(string s){
      if(s.empty())return;
      if(s[0]=='-')negative=1,s=s.substr(1);
      else negative=0;
      for(int i=int(s.size())-1;i>=0;i-=width){
         LL t=0;
         for(int j=max(0,i-width+1);j<=i;++j)
            t=t*10+s[j]-'0';
         push_back(t);
      }
      trim();
   }
   template<typename T>
      bigN(const T &x):bigN(to_string(x)){}
   bigN():negative(0){}
   void trim(){
      while(size()&&!back())pop_back();
      if(empty())negative=0;
   }
   void carry(int _base=base){
      for(size_t i=0;i<size();++i){
         if(at(i)>=0&&at(i)<_base)continue;
         if(i+1u==size())push_back(0);
         int r=at(i)%_base;
         if(r<0)r+=_base;
         at(i+1)+=(at(i)-r)/_base;
         at(i)=r;
      }
   }
   int abscmp(const bigN &b)const{
      if(size()>b.size())return 1;
      if(size()<b.size())return -1;
      for(int i=int(size())-1;i>=0;--i){
         if(at(i)>b[i])return 1;
         if(at(i)<b[i])return -1;
      }
      return 0;
   }
   int cmp(const bigN &b)const{
      if(negative!=b.negative)return negative?-1:1;
      return negative?-abscmp(b):abscmp(b);
   }
   bool operator<(const bigN&b)const{return cmp(b)<0;}
   bool operator>(const bigN&b)const{return cmp(b)>0;}
   bool operator<=(const bigN&b)const{return cmp(b)
       <=0;}
   bool operator>=(const bigN&b)const{return cmp(b)
       >=0;}
   bool operator==(const bigN&b)const{return !cmp(b);}
   bool operator!=(const bigN&b)const{return cmp(b)
       !=0;}
   bigN abs()const{
      bigN res=*this;
      return res.negative=0, res;
   }
   bigN operator-()const{
      bigN res=*this;
      return res.negative=!negative, res;
   }
   bigN operator+(const bigN &b)const{
      if(negative)return -(-(*this)+(-b));
      if(b.negative)return *this-(-b);
      bigN res=*this;
      if(b.size()>size())res.resize(b.size());
      for(size_t i=0;i<b.size();++i)res[i]+=b[i];
      return res.carry(),res.trim(),res;
   }
   bigN operator-(const bigN &b)const{
      if(negative)return -(-(*this)-(-b));
      if(b.negative)return *this+(-b);
      if(abscmp(b)<0)return -(b-(*this));
      bigN res=*this;
      if(b.size()>size())res.resize(b.size());
      for(size_t i=0;i<b.size();++i)res[i]-=b[i];
      return res.carry(),res.trim(),res;
   }
   //operator* using karatsuba
   bigN convert_base(int old_width,int new_width)const{
      vector<long long> p(max(old_width,new_width)+1,1)
          ;
      for(size_t i=1;i<p.size();++i)p[i]=p[i-1]*10;
      bigN ans;
      long long cur=0;
      int cur_id=0;
      for(size_t i=0;i<size();++i){
         cur+=at(i)*p[cur_id];
         cur_id+=old_width;
         while(cur_id>=new_width){
            ans.push_back(cur%p[new_width]);
            cur/=p[new_width];
            cur_id-=new_width;
         }
      }
      return ans.push_back(cur),ans.trim(),ans;
   }
   bigN karatsuba(const bigN &b)const{
      bigN res;res.resize(size()*2);
      if(size()<=32){
         for(size_t i=0;i<size();++i)
            for(size_t j=0;j<size();++j)
               res[i+j]+=at(i)*b[j];
         return res;
      }
      size_t k=size()/2;
      bigN a1(begin(),begin()+k);
      bigN a2(begin()+k,end());
      bigN b1(b.begin(),b.begin()+k);
      bigN b2(b.begin()+k,b.end());

      bigN a1b1=a1.karatsuba(b1);
      bigN a2b2=a2.karatsuba(b2);

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

      bigN r=a2.karatsuba(b2);
```

```
        for(size_t i=0;i<a1b1.size();++i)r[i]-=a1b1[i];
        for(size_t i=0;i<a2b2.size();++i)r[i]-=a2b2[i];

        for(size_t i=0;i<r.size();++i)res[i+k]+=r[i];
        for(size_t i=0;i<a1b1.size();++i)res[i]+=a1b1[i];
        for(size_t i=0;i<a2b2.size();++i)res[i+size()]+=
            a2b2[i];
        return res;
    }
    bigN operator*(const bigN &b)const{
        const static int mul_base=1000000,mul_width=log10
            (mul_base);
        bigN A=convert_base(width,mul_width);
        bigN B=b.convert_base(width,mul_width);
        int n=max(A.size(),B.size());
        while(n&(n-1))++n;
        A.resize(n),B.resize(n);
        bigN res=A.karatsuba(B);
        res.negative=negative!=b.negative;
        res.carry(mul_base);
        res=res.convert_base(mul_width,width);
        return res.trim(),res;
    }
    bigN operator*(long long b)const{
        bigN res=*this;
        if(b<0)res.negative=!negative,b=-b;
        for(size_t i=0,is=0;i<res.size()||is;++i){
            if(i==res.size())res.push_back(0);
            long long a=res[i]*b+is;
            is=a/base;
            res[i]=a%base;
        }
        return res.trim(),res;
    }
    bigN operator/(const bigN &b)const{
        int norm=base/(b.back()+1);
        bigN x=abs()*norm;
        bigN y=b.abs()*norm;
        bigN q,r;
        q.resize(x.size());
        for(int i=int(x.size())-1;i>=0;--i){
            r=r*base+x[i];
            int s1=r.size()<=y.size()?0:r[y.size()];
            int s2=r.size()<y.size()?0:r[y.size()-1];
            int d=(LL(base)*s1+s2)/y.back();
            r=r-y*d;
            while(r.negative)r=r+y,--d;
            q[i]=d;
        }
        q.negative=negative!=b.negative;
        return q.trim(),q;
    }
    bigN operator%(const bigN &b)const{
        return *this-(*this/b)*b;
    }
    friend istream& operator>>(istream &ss,bigN &b){
        string s;
        return ss>>s, b=s, ss;
    }
    friend ostream& operator<<(ostream &ss,const bigN &b
        ){
        if(b.negative)ss<<'-';
        ss<<(b.empty()?0:b.back());
        for(int i=int(b.size())-2;i>=0;--i)
            ss<<setw(width)<<setfill('0')<<b[i];
        return ss;
    }
    template<typename T>
        operator T(){
            stringstream ss;
            ss<<*this;
            T res;
            return ss>>res,res;
        }
};
```

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("%.21fe%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;

class SparseTable{
  private:
    int table[30][N];
  public:
    void init(int n, int arr[]){
      for(int i=0;i<n;i++)
        table[0][i]=arr[i];
      for(int i=1;(1<<i)<=n;i++)
        for(int j=0;(1<<i)+j<=N;j++)
          table[i][j]=min(table[i-1][j],table[i-1][j
              +(1<<(i-1))]);
    }
    void query(int l, int r){
      //0-base [l, r]
      int k = 31-__builtin_clz(r-l);
      return min(minSTable[k][l], minSTable[k][r-(1<<k)
          +1]);
    }
};
```

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

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

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 <iostream>
#include <vector>
#include <bitset>
#include <algorithm>
using namespace std;
#define PB push_back
const int N = 1000 + 5;

vector<int> X[N], Y[N];
bitset<N> visited;
int arr[N], matchX[N], matchY[N];
bool DFS(int);

int main(){
    int n; cin>>n;
    // read graph... for(int i=0;i<n;i++)
  fill(matchX, matchX+n+1, -1);
  fill(matchY, matchY+n+1, -1);
  for(int i=1;i<=n;i++){
    visited.reset();
    DFS(i);
  }
  return 0;
}

bool DFS(int x){
  for(auto y:X[x]){
    if(visited[y]) continue;
    visited[y]=1;
    if(matchY[y]==-1||DFS(matchY[y])){
      matchY[y]=x;
      matchX[x]=y;
      return 1;
    }
  }
  return 0;
}
```

4 Math

4.1 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)};
}
```

4.2 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 = 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;
    }
  }
}
```

4.4 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;
    }
  }
}
```

4.5 Miller Rabin

```
template<typename T>
inline T pow(T a,T b,T mod){// a^b mod 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 llsprp
    [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 or a==1 or a==n-1)continue;
    T x=pow(a,u,n);
    if(x==1 or 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 = 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;
}
```

4.8 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 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;
}
```

5.2 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 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();
  }
}
```

6.2 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;
}
```