Pontificia Universidad Católica del Perú - FCI

XieXieLucas Notebook - Froz/Phibrain/Ands

November 9, 2017

Contents

1 Data Structures

1.1 Fenwick Tree

```
// Fenwick tree: O(log(n)) accumulated sum queries.

ll bitadd[N] ;
ll bitsub[N] ;
int n ;

void update( int idx, ll val1, ll val2 ){
	while( idx <=n ) {
		bitadd[idx] += val1 ;
		bitsub[idx] += val2 ;
		idx += idx & -idx ;
	}
}

void updaterange( int l , int r , ll val ){
	update( l , val , (l-1)*val ) ;
	update( r+1 , -val , -r*val) ;
}

ll get( int idx ){
	ll add = 0 , sub = 0, aux = idx ;
	while ( idx > 0 ){
		add += bitadd[idx] ;
		sub += bitsub[idx] ;
		idx -= idx & -idx ;
	}
	return aux*add - sub ;
}
```

1.2 Heavy Light Decomposition

```
//Heavy-Light Decomposition Tree for Commutative Operations
//Phibrain

inline ll ma(ll a, ll b){return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct ST{
    ll n;
    ll t[2*N];
    ll Op(ll &u, ll &v){ return ma(u,v); }
    inline void build(){
        RREP(i,n-1,1) t[i]=Op(t[i<<1], t[i<<1|1]);
    }
    inline void modify(ll p, ll val){
        for(t[p+=n] = val ; p >>= 1;) t[p] = Op(t[p<<1], t[p<<1|1]);
    }
    inline ll que(ll l, ll r){
        ll ansl=min, ansr=min;
        for(l += n, r += n; l < r; l >>= 1, r >>= 1){
            if(l&1) ansl = Op(ansl, t[l++]);
            if(r&1) ansr = Op(t[--r], ansr);
        }
        return Op(ansl, ansr);
    }
};
struct HLDES{
    ll n;
    ST st;
    vi adj[N];
    ll p[N],d[N],tsz[N],id[N],rt[N];
    ll gid;
    inline ll Op(ll val1, ll val2) {return ma(val1,val2);}
    inline ll make1(ll u,ll par,ll depth){
        p[u]=par; d[u]=depth; tsz[u]=1;
        for(auto v:adj[u])if(v!=p[u]) tsz[u]+=make1(v,u,depth+1);
        return tsz[u];
    }
```

```
    inline void make(){
        ll val=make1(0,-1,0);
    }
    inline void dfs(ll u, ll root){
        id[u]=gid++; rt[u]=root;
        ll w=0 , wsz=min;
        for(auto v: adj[u]) if(v!=p[u]){
            if(tsz[v]>wsz) {w=v; wsz=tsz[v];}
        }
        if(w) dfs(w,root);
        for(auto v:adj[u]) if(v!=p[u]) if(v!=w) dfs(v,v);
    }
    inline void upd(ll u, ll val){
        ll a=id[u];
        st.modify(a,val);
    }
    ll que(ll u, ll v){
        ll ans=0;// neutro?
        while(u!=-1){
            if(rt[u]==rt[v]){
                ll a=id[u], b=id[v];
                if(a>b) swap(a,b);
                ans=Op(ans,st.que(a,b+1));
                u=-1;
            }
            else{
                if(d[rt[u]]>d[rt[v]]) swap(u,v);
                ans=Op(ans,st.que(id[rt[v]],id[v]+1));
                v=p[rt[v]];
            }
        }
        return ans;
    }
    inline void build(){
        gid=0; st.n=n;
        make(); dfs(0,0);
        REP(i,0,n) st.t[i+n]=0;//val de cada t[i]
        st.build();
    }
};


//Heavy Light Decomposition General

struct HLDES{
    ll n;
    ST st1,st2;
    vi adj[N];
    vector<ii> ver[N];
    ll p[N],d[N],tsz[N],id[N],rt[N],ar[N],val[N],id1[N];
    ll gid,k;

    inline T Op(T &val1, T &val2){
      T ty;
      //Operacion del Heavy Light
      return ty;
    }

    inline ll make1(ll u,ll par,ll depth){
      p[u]=par; d[u]=depth; tsz[u]=1;
      for(auto v:adj[u])if(v!=p[u]) tsz[u]+=make1(v,u,depth+1);
      return tsz[u];
    }
    inline void make(){
      ll val=make1(0,-1,0);
    }
    inline void dfs(ll u, ll root){
      ar[gid]=val[u];
      id[u]=gid++; rt[u]=root;
      ll w=0LL , wsz=min;
      for(auto v: adj[u]) if(v!=p[u]){
        if(tsz[v]>wsz) {w=v; wsz=tsz[v];}
      }
      if(w) dfs(w,root);
      for(auto v:adj[u]) if(v!=p[u]) if(v!=w) dfs(v,v);
    }
    inline void solve(){
      ll ta;
      REP(i,0,n) ver[rt[i]].pb(mp(id[i],i));
      REP(i,0,n){
        if(ver[i].size()!=0){
          sort(all(ver[i]));
          ta=ver[i].size();
          ta=ver[i][ta-1].fst;
          for(auto j: ver[i]) id1[j.snd]=ta--;
        }
      }
    }
    inline ll LCA(ll u, ll v){
      while(rt[u]!=rt[v]){
```

```
      if(d[rt[u]]<d[rt[v]]) v=p[rt[v]];
      else u=p[rt[u]];
    }
    return d[u]>d[v]? v:u;
  }
  inline void upd(ll u, ll v, ll val){
    ll l, r, a, b;
    //Update del Heavy Light
  }
  inline ll que(ll u, ll v){
    //Query del HLD
  }
  inline void build(){
    REP(i,0,n) cin>>val[i];
    REP(i,0,n-1) {
      ll a,b; cin>>a>>b;
      a--;b--;
      adj[a].pb(b); adj[b].pb(a);
    }
    gid=0LL; k=0LL; st1.n=n; //st2.n=n;//st.made();
    make();
    dfs(0,0);
    REP(i,0,n) st1.ar[i]=ar[i];
    st1.build();
  }
} hld;
```

1.3 LCA Tree

```
const int MAX = 1e4;
const int LGMAX = 15;
//LCA construction in O(n*log(n)) with O(log(n)) queries.
struct LCATree{
        int n;
        vector<int> adj[MAX];
        int p[MAX][LGMAX]; // 2^j ancestor of node i
        int L[MAX];        // Depth of node i
        int q[MAX];        // (Queue used internally).

        LCATree(int N):n(N){}

        void dfs(int u, int h){
            L[u] = h;
            REP(i,0,sz(adj[u])){
                int v = adj[u][i];
                if (v != p[u][0]) {
                    p[v][0] = u;
                    dfs(v, h+1);
                }
            }
        }
        void buildlca(int r){
                REP(i,0,n) REP(pw,0,LGMAX) p[i][pw] = -1;
            dfs(r, 0);
                for (int pw = 1; (1<<pw) < n; pw++){
                REP(i,0,n) if (p[i][pw-1] != -1) p[i][pw] = p[p[i][pw-1]][
                    pw-1];
                }
        }
        int lca(int u, int v){
                if (L[u] < L[v]) swap(u,v);
                for (int pw = LGMAX-1; pw >= 0; pw--)
                        if (L[u] - (1<<pw) >= L[v])
                                u = p[u][pw];
                if (u == v) return u;
                for (int pw = LGMAX-1; pw >= 0; pw--){
                        if (p[u][pw] != p[v][pw]) {
                                u = p[u][pw];
                                v = p[v][pw];
                        }
                }
                return p[u][0];
        }
};

int main() {
        int n = 1e3;
    LCATree T(n);
    //Initialize n and the adj[] list
        T.buildlca(0); //Place the root instead of 0
        //Ready to answer queries
        return 0;
}
```

1.4 Lazy Propagation Segment Tree

```
// lazy propagation con propagacion y el update
//ejemplo de update en [l,r> la serie de fibonaci con a y b como primeros
    numeros (f[1]=a,f[2]=b)
//notar la forma de updatepro y proh;
//made preprocess y find el fib de posicion n con a y b como primeros
    numeros

inline ll ss(ll val) {return val%MOD;}

ll dpf[N];

inline void made(){
  dpf[1]=1 ; dpf[2]=1;
  REP(i,3,N) dpf[i]=ss(dpf[i-1]+dpf[i-2]);
}
inline ll find(ll a, ll b, ll n) {
  if(n<3) return n==1? a:b;
  return ss(a*dpf[n-2]+b*dpf[n-1]);
}

struct ST{
  ii lazy[4*N];
  ll tree[4*N], ar[N];
  ll n;
  inline void updatepro(ii laz,ll id, ll l,ll r){
    ll ta=r-l, sum=(find(laz.fst,laz.snd,ta+2)-laz.snd+MOD)%MOD;
    tree[id]=ss(tree[id]+sum);
    lazy[id].fst=ss(lazy[id].fst+laz.fst);
    lazy[id].snd=ss(lazy[id].snd+laz.snd);
  }
  inline void proh(ll id, ll l,ll r){
    ll mid=(l+r)>>1, ta=mid-l;
    updatepro(lazy[id],2*id,l,mid);
    ii laz;
    laz.fst=find(lazy[id].fst,lazy[id].snd,ta+1);
    laz.snd=find(lazy[id].fst,lazy[id].snd,ta+2);
    updatepro(laz,2*id+1,mid,r);
    lazy[id]={0LL,0LL};
  }
  inline void updateRange(ll x, ll y, ll a, ll b, ll id, ll l,ll r){
    if(x>=r || y<=l) return;
    if(x<=l && r<=y){
      ll ta=l-x; ii laz;
      laz.fst=find(a,b,ta+1); laz.snd=find(a,b,ta+2);
      updatepro(laz,id,l,r);
      return;
    }
    proh(id,l,r);ll mid=(l+r)>>1;
    updateRange(x,y,a,b,2*id,l,mid);
    updateRange(x,y,a,b,2*id+1,mid,r);
    tree[id]=ss(tree[2*id]+tree[2*id+1]);
  }
  inline ll getSum(ll x,ll y,ll id,ll l,ll r){
    if(x>=r || l>=y) return 0;
    if(x<=l && r<=y) return tree[id];
    proh(id,l,r);ll mid=(l+r)>>1;
    ll ez,ez1,ez2;
    ez1=getSum(x,y,2*id,l,mid);
    ez2=getSum(x,y,2*id+1,mid,r);ez=ss(ez1+ez2);
    return ez;
  }
  inline void build1( ll id, ll l, ll r){
    if (l > r) return ;
    if (r-l<2){tree[id] = ar[l];return;}
    ll mid = (l + r)>>1;
    build1(2*id, l,mid); build1(2*id+1, mid, r);
    tree[id] = ss(tree[id*2 ] + tree[id*2 + 1]);
  }
  inline void upd(ll x, ll y, ll a, ll b){
    updateRange(x,y,a,b,1,0,n);
  }
  inline void build(){
    build1(1,0,n);
  }
  inline ll que(ll x, ll y){
    return getSum(x,y,1,0,n);
  }
};
```

1.5 Link Cut Tree

```
//Link cut tree

const int N = 1e5 + 2;

struct Node {
    Node *left, *right, *parent;
    bool revert;
```

```
    Node() : left(0), right(0), parent(0), revert(false) {}
    bool isRoot() {
        return parent == NULL ||
            (parent->left != this && parent->right != this);
    }
    void push() {
        if (revert) {
            revert = false;
            Node *t = left;
            left = right;
            right = t;
            if (left != NULL) left->revert = !left->revert;
            if (right != NULL) right->revert = !right->revert;
        }
    }
};

struct LinkCutTree{
    Node nos[N];

    LinkCutTree(){
        REP(i,0,N) nos[i] = Node();
    }

    void connect(Node *ch, Node *p, bool isLeftChild) {
        if (ch != NULL) ch->parent = p;
        if (isLeftChild) p->left = ch;
        else p->right = ch;
    }

    void rotate(Node *x){
        Node* p = x->parent;
        Node* g = p->parent;
        bool isRoot = p->isRoot();
        bool leftChild = x == p->left;

        connect(leftChild ? x->right : x->left, p, leftChild);
        connect(p, x, !leftChild);
        if (!isRoot) connect(x, g, p == g->left);
        else x->parent = g;
    }

    void splay(Node *x){
        while (!x->isRoot()) {
            Node *p = x->parent;
            Node *g = p->parent;
            if (!p->isRoot()) g->push();
            p->push();
            x->push();
            if (!p->isRoot()) {
                rotate((x == p->left) == (p == g->left) ? p : x);
            }
            rotate(x);
        }
        x->push();
    }

    Node *expose(Node *x) {
        Node *last = NULL, *y;
        for (y = x; y != NULL; y = y->parent) {
            splay(y);
            y->left = last;
            last = y;
        }
        splay(x);
        return last;
    }

    void makeRoot(Node *x) {
        expose(x);
        x->revert = !x->revert;
    }

    bool connected(Node *x, Node *y) {
        if (x == y) return true;
        expose(x);
        expose(y);
        return x->parent != NULL;
    }

    bool link(Node *x, Node *y) {
        if (connected(x, y)) return false;
        makeRoot(x);
        x->parent = y;
        return true;
    }

    bool cut(Node *x, Node *y) {
        makeRoot(x);
        expose(y);
```

```
        if (y->right != x || x->left != NULL || x->right != NULL)
            return false;
        y->right->parent = NULL;
        y->right = NULL;
        return true;
    }
};
```

1.6 Persistent Segment Tree

```
// Persistent segment tree implemented with pointers.
// Consider using a map<int, node*> which represents
// the segment tree at time t.
const int MAX = 1e6;
typedef int T;
T arr[MAX];
struct node {
        T val;
        node *l, *r;
        node(T val) : val(val), l(NULL), r(NULL) {}
        node(T val, node* l, node* r) : val(val), l(l), r(r) {}
};
// Identity element of Op()
const T OpId = 0;
// Associative query operation
T Op(T val1, T val2){
        return val1 + val2;
}
node* build(int a, int b) {
        if (a+1 == b) return new node(arr[a]);
        node* l = build(a, (a+b)/2);
        node* r = build((a+b)/2, b);
        return new node(Op(l->val, r->val), l, r);
}
// Branch and increment position p by val
node* update(node* u, int a, int b, int p, T val) {
        if (a > p || b <= p) return u;
        if (a+1 == b) return new node(Op(u->val, val));
        node* l = update(u->l, a, (a+b)/2, p, val);
        node* r = update(u->r, (a+b)/2, b, p, val);
        return new node(Op(l->val, r->val), l, r);
}
// Query t to get sum of values in range [i, j)
T query(node* u, int a, int b, int i, int j) {
        if (a >= j || b <= i) return OpId;
        if (a >= i && b <= j) return u->val;
        T q1 = query(u->l, a, (a+b)/2, i, j);
        T q2 = query(u->r, (a+b)/2, b, i, j);
        return Op(q1, q2);
}
map<int, node*> m;
node* st;
T val;
int n, p;
int main() {
        REP(i,0,n) arr[i] = 0; // Any starting values
        m.clear();
        st = build(0,n);
        m[0] = st;
        REP(i,0,n){
                // Modify position p with value val at time t
                st = update(st, 0, n, p, val);
                m[i] = st;
        }
        // Consider for example rectangular queries:
        // Sum of all nodes in [a,b]x[c,d] using one
        // coordinate as time and another as values
}
```

1.7 Segment Tree

```
// Iterative, fast, non-conmutative segment tree.
typedef int T;
const int MAX = 1e6;

// Identity element of the operation
const T OpId = 0;
// Associative internal operation
T Op(T& val1, T& val2){
    return val1 + val2;
}

// The user should fill t[n, 2*n)
T t[2*MAX];
int n;
```

```
void build(){
    for( int i = n-1 ; i > 0 ; i-- ) t[i] = Op(t[i<<1], t[i<<1|1]);
}

void modify( int p , T val ){
    for( t[p+=n] = val ; p >>= 1 ; ) t[p] = Op(t[p<<1], t[p<<1|1]);
}

T get( int l , int r ){ //[l,r)
    T ansl, ansr;
    ansl = ansr = OpId; //Initialize operation at Identity
    for( l += n, r += n ; l < r ; l >>= 1, r >>= 1 ){
            if(l&1) ansl = Op(ansl, t[l++]);
            if(r&1) ansr = Op(t[--r], ansr);
    }
    return Op(ansl, ansr);
}


int main(){
        // Read into t[n,2*n)
    build();
    // Answer queries
}
```

1.8 Wavelet Tree

```
/*
  Wavelet Tree Implementation
  Construction in O(nlogn)
  Queries in O(log(MAX))

  1 - based array!
*/



typedef vector<int> vi;

struct WT{
  int lo, hi;
  WT *l, *r; vi b;
  WT(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){
      return x <= mid;
    };
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++) b.pb(b.back() + f(*it));
    auto pivot = stable_partition(from, to, f);
    l = new WT(from, pivot, lo, mid);
    r = new WT(pivot, to, mid+1, hi);
  }
  //kth en [l,r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1]; //cantidad en los a primeros b[a]
    int lb = b[l-1];
    int rb = b[r];
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }

  //cantidad de numeros menoes a K en [l,r]
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }

  //cantidad de numeros en [l,r] iguales a k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
  ~WT(){
    delete l;
    delete r;
  }
};
```

2 Geometry

2.1 Convex Hull

```
//  INPUT:  a vector of input points, unordered.
//  OUTPUT: a vector of points in the convex hull,
//            counterclockwise, starting with bottom left

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
	T x, y;
	PT() {}
	PT(T x, T y) : x(x), y(y) {}
	bool operator<(const PT &rhs) const {
		return mp(y,x) < mp(rhs.y,rhs.x);
	}
	bool operator==(const PT &rhs) const {
		return mp(y,x) == mp(rhs.y,rhs.x);
	}
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) {
	return cross(a,b) + cross(b,c) + cross(c,a);
}

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS &&
                      (a.x-b.x)*(c.x-b.x) <= 0 &&
                      (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
	sort(pts.begin(), pts.end());
	pts.erase(unique(pts.begin(), pts.end()), pts.end());
	vector<PT> up, dn;
	for (int i = 0; i < pts.size(); i++) {
		while (up.size() > 1 &&
			area2(up[up.size()-2], up.back(), pts[i]) >= 0)
				up.pop_back();
		while (dn.size() > 1 &&
			area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0)
				dn.pop_back();
		up.push_back(pts[i]);
		dn.push_back(pts[i]);
	}
	pts = dn;
	for (int i = (int) up.size() - 2; i >= 1; i--)
		pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
	if (pts.size() <= 2) return;
	dn.clear();
	dn.push_back(pts[0]);
	dn.push_back(pts[1]);
	for (int i = 2; i < pts.size(); i++) {
		if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i]))
			dn.pop_back();
		dn.push_back(pts[i]);
	}
	if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
		dn[0] = dn.back();
		dn.pop_back();
	}
  pts = dn;
#endif
}
```

2.2 Delaunay Triangulation

```
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:    x[] = x-coordinates
//           y[] = y-coordinates
//
// OUTPUT:  triples = a vector containing m triples of indices
//                    corresponding to triangle vertices

#include<vector>
```

```
using namespace std;

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
        int n = x.size();
        vector<T> z(n);
        vector<triple> ret;

        for (int i = 0; i < n; i++)
            z[i] = x[i] * x[i] + y[i] * y[i];

        for (int i = 0; i < n-2; i++) {
            for (int j = i+1; j < n; j++) {
                for (int k = i+1; k < n; k++) {
                    if (j == k) continue;
                    double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-
                        z[i]);
                    double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-
                        z[i]);
                    double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-
                        y[i]);
                    bool flag = zn < 0;
                    for (int m = 0; flag && m < n; m++)
                        flag = flag && ((x[m]-x[i])*xn +
                                        (y[m]-y[i])*yn +
                                        (z[m]-z[i])*zn <= 0);
                    if (flag) ret.push_back(triple(i, j, k));
                }
            }
        }
        return ret;
}

int main(){
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

2.3 Geometry

```
// C++ routines for computational geometry.

const double INF = 1e100;
const double EPS = 1e-12;
const double PI = acos(-1);

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)  const { return PT(x*c, y*c ); }
  PT operator / (double c)  const { return PT(x/c, y/c ); }
};

double dot(PT p, PT q)  { return p.x*q.x+p.y*q.y; }
double norm(PT p) { return sqrt(dot(p,p)); }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

double angle(PT p){
    double res = acos(p.x / norm(p));
    if (p.y > 0) return res;
    else return 2*PI - res;
}
```

```
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// Whethes lines (a,b), (c,d) are parallel/collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
        if (LinesCollinear(a, b, c, d)) {
                if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
                        dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
                if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 &&
                         dot(c-b, d-b) > 0) return false;
                return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c
      ));
}

// determine if point is in a possibly non-convex polygon
// (by William Randolph Franklin); returns 1 for strictly
// interior points, 0 for strictly exterior points, and 0 or 1
// for the remaining points. Note that it is possible to
// convert this into an *exact* test using integer arithmetic
// by taking care of the division appropriately (making sure
// to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
```

```
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) *(q.y - p[i].y) /
            (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleInter(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// Area or centroid of a (possibly nonconvex) polygon,
// assuming the coordinates are listed in a clockwise or
// counterclockwise order. Note that the centroid is often
// known as the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// Whether or not a given (CW or CCW) polygon is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

```
// Computes the circumcenter of a Triangle PQR
PT circumcenter(PT p, PT q, PT r) {
	PT a = p-r, b = q-r;
	PT c = PT(dot(a, (p + r)) / 2, dot(b, (q + r)) / 2);
	return PT(dot(c, RotateCW90(PT(a.y, b.y))),
		dot(PT(a.x, b.x), RotateCW90(c))) / dot(a, RotateCW90(b));
}


//Check if a polygon is convex
bool isConvex(const vector<point> &P) {
	int sz = (int)P.size();
	if (sz <= 3) return false;
	bool isLeft = ccw(P[0], P[1], P[2]);
	for (int i = 1; i < sz-1; i++)
		if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
		    return false;
	return true;
}
```

2.4 Minkowski Sum

```
//Calcula suma de Minkowski en O(n + m)
//A y B deben estar en sentido antihorario
inline bool compare(PT a, PT b){
	// mas abajo, mas a la izquierda
	if(a.y < b.y) return 1;
	if(a.y == b.y) return a.x < b.x;
	return 0;
}

vector<PT> minkow_sum(const vector<PT>& a, const vector<PT>& b){
	vector< PT > out;
	out.clear();
	int lena = int(a.size());
	int lenb = int(b.size());
	int i = 0, j = 0;
	for(int q = 0; q < lena; ++q) if(compare(a[q], a[i])) i = q;
	for(int q = 0; q < lenb; ++q) if(compare(b[q], b[j])) j = q;
	ll pr;
	int nxti, nxtj;
	do{
		out.pb(a[i] + b[j]);
		nxti = (i + 1) % lena;
		nxtj = (j + 1) % lenb;
		pr = cross(a[nxti] - a[i], b[nxtj] - b[j]);
		if(pr > 0) i = nxti;
		else if(pr < 0) j = nxtj;
		else i = nxti, j = nxtj; // paralelas, subo en ambas
	}while((a[i] + b[j]) != out[0]);
	return out;
}
```

3 Graphs

3.1 2SAT

```
//2-SAT
//Conditions from 0 to 2*number of nodes, i and i^1 are reciprocal
//That means, ~0 is 1, ~1 is 0, ~2 is 3, ~3 is 2, etc
//When adding an edge, make sure to fix values
//For example, node from a to b (a,b >= 1)
//aa = (a-1)*2, bb = (b-1)*2, then a has "aa" as true and aa^1 as false
//Same to b
//To return to the main state, divide by 2 and sum 1

struct TwoSAT{
  int n;
  vector< vi> g, adj;
  vi d, low, scc, ans, lev;
  vector<bool> stacked, ok;
  stack<int> s;
  int ticks, current_scc;

  TwoSAT(int N):
    n(N),ticks(0), current_scc(0), g(N), adj(N), d(N), low(N), scc(N),
       ans(N), lev(N),
    stacked(N), ok(N){}

  void initialize(){
    REP(i,0,n){
      stacked[i] = false;
      d[i] = -1;
      scc[i] = -1;
      ok[i] = false;
```

```
      current_scc = ticks = 0;
    }
  }

  void addEdge(int a, int b){
    g[a].pb(b);
  }

  void tarjan(int u){
    d[u] = low[u] = ticks++;
    s.push(u);
    stacked[u] = true;
    const vector<int> &out = g[u];
    for (int k=0, m=out.size(); k<m; ++k){
      const int &v = out[k];
      if (d[v] == -1){
        tarjan(v);
        low[u] = min(low[u], low[v]);
      }else if (stacked[v]){
        low[u] = min(low[u], low[v]);
      }
    }
    if (d[u] == low[u]){
      int v;
      do{
        v = s.top();
        s.pop();
        stacked[v] = false;
        scc[v] = current_scc;
      }while (u != v);
      current_scc++;
    }
  }

  bool consistent(){
    for(int i = 0; i < n; i+=2){
      if(scc[i] == scc[i^1]){
        return false;
      }
    }
    return true;
  }

  void build(){
    REP(i,0,n){
      REP(j,0,sz(g[i])){
        int v = g[i][j];
        if(scc[i] != scc[v]){
          adj[i].pb(v); lev[v]++;
        }
      }
    }
  }

  void toposort(){
    queue<int> q;
    REP(i,0,current_scc){
      if(lev[i] == 0) q.push(i);
    }
    int x = 1;
    while(!q.empty()){
      int u = q.front(); q.pop();
      ans[u] = x ++;
      REP(i,0,sz(adj[u])){
        int v = adj[u][i];
        lev[v]--;
        if(lev[v] == 0) q.push(v);
      }
    }
  }

  void solve(){
    for(int i = 0; i<n; i+=2){
      if(ans[scc[i]] < ans[scc[i^1]]){
        ok[i] = false; ok[i^1] = true;
      }
      else{
        ok[i] = true; ok[i^1] = false;
      }
    }
  }

  bool go(){
    REP(i,0,n){
      if(scc[i] == -1) tarjan(i);
    }
    if(!consistent()) return false;
    else{
      build();
      toposort();
```

```
        solve();
        return true;
      }
    }

};

int main(){
   fastio;
   int n,m; cin >> n >> m;
   TwoSAT TS = TwoSAT(2*n);
   TS.initialize();

   //TO DO: ADD EDGES

   bool res = TS.go();
   if(!res) cout << "Impossible" << endl;
   else{
      for(int i = 0; i < 2*n; i+=2){
         int state = i/2 + 1;
         if(TS.ok[i]) //state is true
         else //state is false
      }
   return 0;
}
```

3.2 Biconnected Components

```
//Finds Biconnected Components

bool usd[1005];
int low[1005], d[1005], prev[1005], cnt;
vector <int> adj[1005];
stack <ii> S;

void Outcomp( int u , int v ){
        printf("New Component\n");
        ii e;
        do{
                e = S.top(); S.pop();
                cout << e.fst << " " << e.snd << endl;
        } while( e != mp( u , v ) );
}

void dfs( int u ){
        usd[u] = 1; cnt++;
        low[u] = d[u] = cnt;
        REP(i,0,sz(adj[u])){
                int v = adj[u][i];
                if( !usd[v] ){
                        S.push( mp( u , v ) );
                        prev[v] = u; dfs( v );
                        if( low[v] >= d[u] ) Outcomp( u , v );
                        low[u] = min( low[u] , low[v] );
                }
                else if( prev[u] != v and d[v] < d[u] ){
                        S.push( mp( u , v ) );
                        low[u] = min( low[u] , d[v] );
                }
        }
}

int main(){
        int n, m;
        cin >> n >> m;
        REP(i,0,m){
                int a , b;
                cin >> a >> b;
                adj[a].pb(b);
                adj[b].pb(a);
        }
        cnt = 0;
        memset(usd,0,sizeof(usd));
        memset(prev,-1,sizeof(prev));
        REP(i,0,n){
                if( !usd[i] ) dfs(i);
        }
        return 0;
}
```

3.3 Bridges and Articulation Points

```
//Finding bridges and articulation points

int low[N],id[N],parent[N];
bool art[N];
```

```
vi adj[N];
vi bridge[N];
int curr_id =0;
int root, rootchild;

void dfs(int u) {
	low[u] = id[u] = curr_id++;
	REP(j,0,sz(adj[u])) {
		int v = adj[u][j];
		if (id[v] == -1) {
			parent[v] = u;
			if (u == root) rootchild++;
			dfs(v);
			if (low[v] >= id[u]) art[u] = true;
			if (low[v] > id[u]){
				bridge[u].pb(v);
				bridge[v].pb(u); //store bridges in a sub
				    graph
			}
			low[u] = min(low[u], low[v]);
		}
		else if (v != parent[u]) low[u] = min(low[u], id[v]);
	}
}

//inside int main()
REP(i,0,n){
	if (id[i] == -1) {
		root = i; rootchild = 0; dfs(i);
		art[root] = (rootchild > 1);
	}
}
```

3.4 Eulerian Path

```
// Finds Eulerian Path (visits every edge exactly once)
// CYCLE exists iff all edges even degree, all edges in
// same connected component.
// PATH exists iff cycle exists and once edge removed
// [ Hamiltonian (all vertices) is NP complete ]
struct Edge;
typedef list<Edge>::iterator iter;
struct Edge
{
	int next_vertex;
	iter reverse_edge;
	Edge(int next_vertex) :next_vertex(next_vertex) { }
};
const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];      // adjacency list
vector<int> path;

void find_path(int v)
{
	while(adj[v].size() > 0)
	{
		int vn = adj[v].front().next_vertex;
		adj[vn].erase(adj[v].front().reverse_edge);
		adj[v].pop_front();
		find_path(vn);
	}
	path.push_back(v);
}
void add_edge(int a, int b)
{
	adj[a].push_front(Edge(b));
	iter ita = adj[a].begin();
	adj[b].push_front(Edge(a));
	iter itb = adj[b].begin();
	ita->reverse_edge = itb;
	itb->reverse_edge = ita;
}
```

3.5 Maximal Cliques

```
// Bron-Kerbosch algorithm for finding all the
// maximal cliques of a graph in O(3^(n/3))
// 3 ^ 13 = 1.6e6

// Call them using clique(0, (1LL << n) - 1, 0)
// n vertexs
ll adj[65];
// This algorithm finds all the maximal cliques containing an edge
// The cliques are found explicitly (the vertex of the cliques)
void clique(ll r, ll p, ll x) {
```

```cpp
    if (p == 0 && x == 0) {
        /* r is a maximal clique */
        /* Every 1 in r is a vertex of the clique
        Then, __builtin_popcountll(r) is the size of the clique*/
        return;
        }
    int pivot = -1;
    int menor = INF;
    for (int i = 0; i < n; i++) {
        if ( ((1LL << i) & p) || ((1LL << i) & x) ) {
            int x = __builtin_popcountll(p & (~(adj[i])));
            if (x < menor) {
                pivot = i;
                menor = x;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        if ((1LL << i) & p) {
            if (pivot != -1 && adj[pivot] & (1LL << i)) continue;
            clique(r | (1LL << i), p & adj[i], x & adj[i]);
            p = p ^ (1LL << i);
            x = x | (1LL << i);
        }
    }
}

// This one has the same idea, but is faster
// However, it only finds the size of the cliques
void clique2(int r, ll p, ll x){
    if(p == 0 && x == 0){
        // r is the size of the clique
    }
    if(p == 0) return;
    int u = __builtin__ctzll(p | x);
    ll c = p & ~ adj[u];
    while(c){
        int v = __builint_ctzll(c); //Number of trailing zeros
        clique(r + 1, p & adj[v], x & adj[v]);
        p ^= (1LL << v);
        x |= (1LL << v);
        c ^= (1LL << v);
    }
}
```

3.6 Tarjan Strongly Connected Components

```cpp
/* Complexity: O(E + V)
 Tarjan's algorithm for finding strongly connected
components.
 *d[i] = Discovery time of node i. (Initialize to -1)
 *low[i] = Lowest discovery time reachable from node
 i. (Doesn't need to be initialized)
 *scc[i] = Strongly connected component of node i. (Doesn't
 need to be initialized)
 *s = Stack used by the algorithm (Initialize to an empty
 stack)
 *stacked[i] = True if i was pushed into s. (Initialize to
 false)
 *ticks = Clock used for discovery times (Initialize to 0)
 *current_scc = ID of the current_scc being discovered
 (Initialize to 0)
*/

//DON'T FORGET TO INITIALIZE d[MAXN] TO -1 !!!!
vector<int> g[MAXN];
int d[MAXN], low[MAXN], scc[MAXN];
bool stacked[MAXN];
stack<int> s;
int ticks, current_scc;
void tarjan(int u){
  d[u] = low[u] = ticks++;
  s.push(u);
  stacked[u] = true;
  const vector<int> &out = g[u];
  for (int k=0, m=out.size(); k<m; ++k){
    const int &v = out[k];
    if (d[v] == -1){
      tarjan(v);
      low[u] = min(low[u], low[v]);
    }else if (stacked[v]){
      low[u] = min(low[u], low[v]);
    }
  }
  if (d[u] == low[u]){
    int v;
    do{
      v = s.top();
      s.pop();
      stacked[v] = false;
```

```
      scc[v] = current_scc;
    }while (u != v);
    current_scc++;
  }
}
```

4 Math

4.1 Chinese Remainder Theorem

```
/**
 * Chinese remainder theorem.
 * Find z such that z % x[i] = a[i] for all i.
 * */
long long crt(vector<long long> &a, vector<long long> &x) {
  long long z = 0;
  long long n = 1;
  for (int i = 0; i < x.size(); ++i)
    n *= x[i];

  for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i])) % n;
    tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
    z = (z + tmp) % n;
  }

  return (z + n) % n;
}
```

4.2 Cribas

```
// Criba en O(n)
// p[i] indica el valor del primo i-esimo
// A[i] indica que el menor factor primo de i
//                      es el primo A[i] - esimo
#define MAXN 100000

int A[MAXN + 1], p[MAXN + 1], pc = 0;

void sieve()
{
    for(int i=2; i<=MAXN; i++){
        if(!A[i]) p[A[i] = ++pc] = i;
        for(int j=1; j<=A[i] && i*p[j]<=MAXN; j++)
            A[i*p[j]] = j;
    }
}

//Criba para phi
int phi[MAX];

void CribaEuler(){
        REP(i,0, MAX) primo[i] = 1, phi[i] = 1;
        primo[0] = primo[1] = false;
        REP(i,2,MAX){
                if(primo[i]){
                        phi[i] = i - 1;
                        for(int j = i+i; j < MAX; j += i){
                                primo[j] = false;
                                int pot = 1, aux = j/i;
                                while( aux % i == 0 ){
                                        aux /= i, pot *= i;
                                }
                                phi[j] *= (i-1)*pot ;
                        }
                }
        }
}
```

4.3 Euler Totient

```
//Euler Totient
//Finds all k <= n where gcd(k,n) == 1

int phi(int n){
        int res = n;
        for(int p = 2; p*p <= n; p++){
                if(n % p == 0){
                        while(n % p == 0)
                                n/=p;
                        res -= res/p;
                }
        }
```

```
        if (n > 1) res -= res/n;
        return res;
}
```

4.4 Inverso Modular

```
/** Inverso Modular **/
#define MAX 100
#define MOD 1000000009

long long inverso[MAX];

void inv(){
        inverso[1] = 1;
        REP(i,2,MAX) inverso[i] = ( (MOD-MOD/i) * inverso[MOD%i] ) % MOD;
}
```

4.5 Miller Rabin

```
//Miller-Rabin primality test

ll pow(ll a, ll b, ll c){
    ll ans = 1;
    while(b){
        if(b&1) ans = (1LL*ans*a)%c;
        a = (1LL*a*a)%c;
        b >>=1;
    }
    return ans;
}

bool miller(ll p, ll it = 10){
    if(p<2) return 0;
    if(p!=2 && (p&1) == 0) return 0;
    ll s=p-1;
    while((s&1) == 0) s>>=1;
    while(it--){
        ll a = rand()%(p-1)+1, temp = s;
        ll mod = pow(a,temp,p);
        while(temp!= p-1 && mod!=1 && mod!=p-1){
            mod = (1LL*mod*mod)%p;
            temp<<=1;
        }
        if(mod!=p-1 && (temp&1) == 0) return 0;
    }
    return 1;
}
```

4.6 Number Theory

```
// Encuentra el menor positivo de la forma ax+ b , my+ n
// ( x,y enteros no necesariamente postivos)
// Si son positivos, hallar los coeficientes y sumar lo
// que falta para que de positivo

// d: mcd(a,b) ( d > 0 )
// x,y enteros tales que a*x + b*y = d
// las demas soluciones son ( x + (b/d)t , y - (a/d)t )
void gcdextend(ll a, ll b, ll &x, ll &y, ll &d){
    if( b == 0){
       if(a>0) x = 1, y = 0 , d = a ;
       else x = -1 , y = 0 , d = -a ;
       return ;
    }
    gcdextend(b,a%b,x,y,d) ;
    ll x1 = y , y1 = x - (a/b)*y ;
    x = x1 , y = y1 ;
}


// menor positivo que es u modulo modPos
inline ll ADDTOPOSITIVE(ll u, ll modPOS){
    if(modPOS < 0) modPOS = -modPOS ;
    if(u >= 0) return u%modPOS;
    u = -u ;
    if(u%modPOS == 0) return 0;
    return modPOS*((u/modPOS)+1) - u ;
}

// Encuentra el menor positivo que es
// de la forma ax + b , my + n
// los demas son de la forma ans + ((a/d)*m)*t
// retorna -1 si no hay solucion ( mcd(a,m) no divide a n - b )
inline ll FINDmenorLCS(ll a,ll b,ll m,ll n){
```

```
    //Cuidado con el caso a = 0 , m = 0 ,
        //porque es solo verificar b = n

    a = abs(a) ; m = abs(m) ;
    if( a == 0 ) {
       swap(a,m);
       swap(b,n) ;
    }
    if( m == 0 ){
       if( (n-b) % a == 0) return n ;
       else return -1 ;
    }
    ll x , y , d ;
    gcdextend(a,m,x,y,d) ;
    if((n-b)%d != 0) return -1 ;
    ll temp = a*x*((n-b)/d) + b ;
    temp = ADDTOPOSITIVE(temp,a*(m/d)) ;
    return temp ;
}

//Finds partition of n (number of ways to obtain n as a sum of positive
    numbers)

int partition(int n) {
  int[] dp = new int[n + 1];
  dp[0] = 1;
  for (int i = 1; i <= n; i++) {
    for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1) {
     dp[i] += dp[i - (3 * j * j - j) / 2] * r;
     if (i - (3 * j * j + j) / 2 >= 0) {
       dp[i] += dp[i - (3 * j * j + j) / 2] * r;
     }
    }
  }
  return dp[n];
}
```

4.7 Pollard Rho

```
#define MAXL (50000>>5)+1
#define GET(x) (mark[x>>5]>>(x&31)&1)
#define SET(x) (mark[x>>5] |= 1<<(x&31))
int mark[MAXL];
int P[50000], Pt = 0;
void sieve() {
    register int i, j, k;
    SET(1);
    int n = 46340;
    for (i = 2; i <= n; i++) {
       if (!GET(i)) {
          for (k = n/i, j = i*k; k >= i; k--, j -= i)
             SET(j);
          P[Pt++] = i;
       }
    }
}
ll mul(unsigned ll a, unsigned ll b, unsigned ll mod) {
    ll ret = 0;
    for (a %= mod, b %= mod; b != 0; b >>= 1, a <<= 1, a = a >= mod ? a -
        mod : a) {
       if (b&1) {
          ret += a;
          if (ret >= mod)   ret -= mod;
       }
    }
    return ret;
}
void exgcd(ll x, ll y, ll &g, ll &a, ll &b) {
    if (y == 0)
       g = x, a = 1, b = 0;
    else
       exgcd(y, x%y, g, b, a), b -= (x/y) * a;
}

ll inverse(ll x, ll p) {
    ll g, b, r;
    exgcd(x, p, g, r, b);
    if (g < 0) r = -r;
    return (r%p + p)%p;
}

ll mpow(ll x, ll y, ll mod) { // mod < 2^32
    ll ret = 1;
    while (y) {
       if (y&1)
          ret = (ret * x)%mod;
       y >>= 1, x = (x * x)%mod;
```

```
    }
    return ret % mod;
}

ll mpow2(ll x, ll y, ll mod) {
    ll ret = 1;
    while (y) {
        if (y&1)
            ret = mul(ret, x, mod);
        y >>= 1, x = mul(x, x, mod);
    }
    return ret % mod;
}

int isPrime(ll p) { // implements by miller-babin
    if (p < 2 || !(p&1))     return 0;
    if (p == 2)                  return 1;
    ll q = p-1, a, t;
    int k = 0, b = 0;
    while (!(q&1))    q >>= 1, k++;
    for (int it = 0; it < 2; it++) {
        a = rand()%(p-4) + 2;
        t = mpow2(a, q, p);
        b = (t == 1) || (t == p-1);
        for (int i = 1; i < k && !b; i++) {
            t = mul(t, t, p);
            if (t == p-1)
                b = 1;
        }
        if (b == 0)
            return 0;
    }
    return 1;
}

ll pollard_rho(ll n, ll c) {
    ll x = 2, y = 2, i = 1, k = 2, d;
    while (true) {
        x = (mul(x, x, n) + c);
        if (x >= n)    x -= n;
        d = __gcd(x - y, n);
        if (d > 1) return d;
        if (++i == k) y = x, k <<= 1;
    }
    return n;
}

void factorize(int n, vector<ll> &f) {
    for (int i = 0; i < Pt && P[i]*P[i] <= n; i++) {
        if (n%P[i] == 0) {
                while (n%P[i] == 0)
                        f.push_back(P[i]), n /= P[i];
        }
    }
    if (n != 1) f.push_back(n);
}

void llfactorize(ll n, vector<ll> &f) {
    if (n == 1)
        return ;
    if (n < 1e+9) {
        factorize(n, f);
        return ;
    }
    if (isPrime(n)) {
        f.push_back(n);
        return ;
    }

    ll d = n;
    for (int i = 2; d == n; i++)
        d = pollard_rho(n, i);
    llfactorize(d, f);
    llfactorize(n/d, f);
}

vector<ll> f;
map<ll, int> r;

int main() {
    sieve();
    ll n;
    scanf("%lld", &n);
    llfactorize(n, f);
    for (auto &x : f) r[x]++;
    ll last;
    for (auto it = r.begin(); it != r.end(); it++) {
        if (it != r.begin()) printf(" ");
        last = it -> first;
        printf("%lld", last);
```

```
        if (it->second > 1){
            for( int i = 0 ; i < it->second - 1 ; ++i ) printf(" %lld",last
                ) ;
        }
    }
    return 0;
}
```

4.8 Simplex Method

```
// Two-phase simplex algorithm for solving linear programs:
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c as
// arguments. Then, call Solve(x).

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

	LPSolver(const VVD &A, const VD &b, const VD &c) :
		m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
		for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
			D[i][j] = A[i][j];
		for (int i = 0; i < m; i++)
			B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];
		for (int j = 0; j < n; j++)
			N[j] = j; D[m][j] = -c[j];
			N[n] = -1; D[m+1][n] = 1;
	}

	void Pivot(int r, int s) {
		for (int i = 0; i < m+2; i++) if (i != r)
			for (int j = 0; j < n+2; j++) if (j != s)
				D[i][j] -= D[r][j] * D[i][s] / D[r][s];
		for (int j = 0; j < n+2; j++) if (j != s)
			D[r][j] /= D[r][s];
		for (int i = 0; i < m+2; i++) if (i != r)
			D[i][s] /= -D[r][s];
		D[r][s] = 1.0 / D[r][s];
		swap(B[r], N[s]);
	}

	bool Simplex(int phase) {
		int x = phase == 1 ? m+1 : m;
		while (true) {
			int s = -1;
			for (int j = 0; j <= n; j++) {
				if (phase == 2 && N[j] == -1) continue;
				if (s == -1 || D[x][j] < D[x][s] ||
					D[x][j] == D[x][s] && N[j] < N[s]) s
					    = j;
			}
			if (D[x][s] >= -EPS) return true;
			int r = -1;
			for (int i = 0; i < m; i++) {
				if (D[i][s] <= 0) continue;
				if (r == -1 ||
					D[i][n+1] / D[i][s] < D[r][n+1] / D[r
					    ][s] ||
				 D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s
				     ] &&
					B[i] < B[r]) r = i;
	}
			if (r == -1) return false;
			Pivot(r, s);
		}
	}

	DOUBLE Solve(VD &x) {
		int r = 0;
		for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r =
		    i;
```

```
            if (D[r][n+1] <= -EPS) {
                Pivot(r, n);
                if (!Simplex(1) || D[m+1][n+1] < -EPS)
                    return -numeric_limits<DOUBLE>::infinity();
                for (int i = 0; i < m; i++) if (B[i] == -1) {
                    int s = -1;
                    for (int j = 0; j <= n; j++)
                        if (s == -1 || D[i][j] < D[i][s] ||
                            D[i][j] == D[i][s] && N[j] < N
                                [s]) s = j;
                    Pivot(i, s);
                }
            }
            if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
            x = VD(n);
            for (int i = 0; i < m; i++) if (B[i] < n)
                x[B[i]] = D[i][n+1];
            return D[m][n+1];
        }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: "<< value << endl;
  cerr << "SOLUTION:";
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

4.9 Teorema de Lucas

```
ll comb[105][105];

//Devuleve la comb(n,k) % m para n,k grandes y m pequeno

ll lucas( ll n , ll k , ll m ){
    //Se puede precalcular la combinatoria afuera
    REP(i,0,52) REP(j,0,52){
        if( j == 0 ) comb[i][0] = 1;
        else if( j > i ) comb[i][j] = 0;
        else comb[i][j] = ( comb[i-1][j] + comb[i-1][j-1] ) % m;
    }

    ll ans = 1, x, y;

    while( n ){
        x = n % m, y = k % m;
        ans = ( ans * comb[x][y] ) % m;
        n /= m, k /= m;
    }
    return ans;
}
```

5 Misc

5.1 Centroid Decomposition

```
#define N 100002

inline ll ma(ll a, ll b){ return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct CD{
    vector< int > graph[N];
```

```
int sub[N],p[N];
//sub[i]: size del nodo i luego de descomponer el tree
//p[i]: padre del nodo i luego de descomponer el tree
//notar que el padre del centroid es -2
// el tree esó 1 0 base
//para inicializar addEddge(a,b);
//para construir el centroid tree, solo llamar init(root); root:
    root del tree
void addEdge(int &a, int &b){
        graph[a].pb(b);
        graph[b].pb(a);
}
inline void dfs(int cur, int parent){
        sub[cur] = 1;
        for(int i = 0; i < sz(graph[cur]); ++i){
                int to = graph[cur][i];
                if(to != parent && p[to] == -1){
                        dfs(to, cur);
                        sub[cur] += sub[to];
                }
        }
}
inline void decompose(int cur, int parent, int sb, int prevc){
        for(int i = 0; i < sz(graph[cur]); ++i){
                int to = graph[cur][i];
                if(to != parent && p[to] == -1 && (2 * sub[to] > sb)
                    ){
                        decompose(to, cur, sb, prevc);
                        return;
                }
        }
        p[cur] = prevc;
        for(int i = 0; i < sz(graph[cur]); ++i){
                int to = graph[cur][i];
                if(p[to] == -1){
                        dfs(to, - 1);
                        decompose(to, cur, sub[to], cur);
                }
        }
}
inline void init(int start){
        for(int i = 0; i < N; ++i) p[i] = -1;
        dfs(start, - 1);
        decompose(start, -1, sub[start], -2);
}
};
int cnt=1;
vi adj[N];
int d[N];
inline void make(int &u, int x, int depth){
        d[u]=depth;
        for(auto v :adj[u]) if(v!=x) make(v,u,depth+1);
}
int main() {
        fastio;
        int n; cin>>n;
        CD cd; //cd.n=n;
        REP(i,0,n-1) {
                ll a,b; cin>>a>>b;
                cd.addEdge(a,b);
        }
        cd.init(1);
        int pa, root;
        REP(i,1,n+1) {
                pa=cd.p[i];
                if(pa==-2) root=i;
                if(pa!=-2) {
                        adj[i].pb(pa);
                        adj[pa].pb(i);
                }
        }
        make(root,0,1);
        char is;map<int, string> m;int k=1,flag=1;
        for(is='A'; is<='Z'; is++) m[k++]=is;
        REP(i,1,n+1) if(d[i]>26) flag=0;
        if(flag==0) cout<<"Impossible!"<<endl;
        if(flag==1) {
                REP(i,1,n+1) cout<<m[d[i]]<<endl;
        }
        return 0;
}
```

5.2 Closest Pair

```
//Closest Pair Algorithm with Sweep
//Complexity: O(nlogn)

#define MAX_N 100000
```

```
#define px second
#define py first
typedef pair<long long, long long> point;

int N;
point P[MAX_N];
set<point> box;

bool compare_x(point a, point b){ return a.px<b.px; }

inline double dist(point a, point b){
	return sqrt((a.px-b.px)*(a.px-b.px)+(a.py-b.py)*(a.py-b.py));
}
double closest_pair(){
	if(N<=1) return -1;
	sort(P,P+N,compare_x);
	double ret = dist(P[0],P[1]);
	box.insert(P[0]);
	set<point> :: iterator it;
	for(int i = 1,left = 0;i<N;++i){
		while(left<i && P[i].px-P[left].px>ret) box.erase(P[left
		    ++]);
		for(it = box.lower_bound(make_pair(P[i].py-ret,P[i].px-ret)
		    );
		it!=box.end() && P[i].py+ret>=(*it).py;++it)
		ret = min(ret, dist(P[i],*it));
		box.insert(P[i]);
	}
	return ret;
}
```

5.3 Convex Hull Trick

```
 // Simple Hull
struct HullSimple { // Upper envelope for Maximum.
   // Special case: strictly increasing slope in insertions,
   // increasing value in queries.
   deque<pair<ll, ll> > dq;
   ld cross(pair<ll, ll> l1, pair<ll, ll> l2){
      return (ld)(l2.snd - l1.snd) / (ld)(l1.fst - l2.fst);
   }
   void insert_line(ll m, ll b){
      pair<ll,ll> line = mp(m,b);
      while (sz(dq) > 1 && cross(line, dq[sz(dq)-1]) <=
            cross(dq[sz(dq)-1], dq[sz(dq)-2])) dq.pop_back();
      dq.pb(mp(m,b));
   }

   ll eval(pair<ll, ll> line, ll x){
      return line.fst * x + line.snd;
   }

   ll eval(ll x){
      while (sz(dq) > 1 && eval(dq[0], x) < eval(dq[1],x))
         dq.pop_front();
      return eval(dq[0],x);
   }
};

// Dynamic Hull
// Compile with g++ -std=c++11 file.cpp -o file
typedef long double ld;
const ll is_query = -(1LL<<62);
struct Line {
   ll m, b;
   mutable function<const Line*()> succ;
   bool operator<(const Line& rhs) const {
      if (rhs.b != is_query) return m < rhs.m;
      const Line* s = succ();
      if (!s) return 0;
      ll x = rhs.m;
      return b - s->b < (s->m - m) * x;
   }
};
// Upper envelope for Maximum
struct HullDynamic : public multiset<Line> {
    bool bad(iterator y) {
      auto z = next(y);
      if (y == begin()) {
         if (z == end()) return 0;
         return y->m == z->m && y->b <= z->b;
      }
      auto x = prev(y);
      if (z == end()) return y->m == x->m && y->b <= x->b;
      return (x->b - y->b)*(z->m - y->m) >=
                              (y->b - z->b)*(y->m - x->m);
   }
   void insert_line(ll m, ll b) {
```

```
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0: &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

5.4 Dates

```
//
// Time - Leap years
//

// A[i] has the accumulated number of days from months previous to i
const int A
    [13] = { 0, 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 };
// same as A, but for a leap year
const int B
    [13] = { 0, 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
// returns number of leap years up to, and including, y
int leap_years(int y) { return y / 4 - y / 100 + y / 400; }
bool is_leap(int y) { return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0)
    ; }
// number of days in blocks of years
const int p400 = 400*365 + leap_years(400);
const int p100 = 100*365 + leap_years(100);
const int p4  = 4*365 + 1;
const int p1  = 365;
int date_to_days(int d, int m, int y)
{
  return (y - 1) * 365 + leap_years(y - 1) + (is_leap(y) ? B[m] : A[m]) +
      d;
}
void days_to_date(int days, int &d, int &m, int &y)
{
  bool top100; // are we in the top 100 years of a 400 block?
  bool top4;   // are we in the top 4 years of a 100 block?
  bool top1;   // are we in the top year of a 4 block?

  y = 1;
  top100 = top4 = top1 = false;

  y += ((days-1) / p400) * 400;
  d = (days-1) % p400 + 1;

  if (d > p100*3) top100 = true, d -= 3*p100, y += 300;
  else y += ((d-1) / p100) * 100, d = (d-1) % p100 + 1;

  if (d > p4*24) top4 = true, d -= 24*p4, y += 24*4;
  else y += ((d-1) / p4) * 4, d = (d-1) % p4 + 1;

  if (d > p1*3) top1 = true, d -= p1*3, y += 3;
  else y += (d-1) / p1, d = (d-1) % p1 + 1;

  const int *ac = top1 && (!top4 || top100) ? B : A;
  for (m = 1; m < 12; ++m) if (d <= ac[m + 1]) break;
  d -= ac[m];
}
```

5.5 Divide and Conquer Trick

```
// Divide and Conquer DP optimization.
// Problem: dp[i][j] = min{k>j} (func(j,k) + dp[i-1][k]).
// (That is, split n objects into k buckets with cost
// func per bucket). Necessary condition: argmin(dp[i][j]) <=
// argmin(dp[i][j+1]) (this is "opt")
// Naive complexity: O(kn^2)
// Improved complexity: O(knlog(n))
// Consider checking if opt[i+1][j] <= opt[i][j] <= opt[i][j+1]
// and using a knuth-like O(n^2) loop

const ll INF = 1e18;
int n, k;

ll c[8100];
ll s[8100];
ll dp[810][8100];

ll func(int i, int j){ return (s[j] - s[i])*(j-i); }

void go(int i, int l, int r, int optl, int optr){
```

```
        if (l >= r) return;
        int m = (l+r)/2;
        int opt = n;
        dp[i][m] = INF;
        for(int u = optr; u>= optl; u--){
                ll curr = dp[i-1][u] + func(m,u);
                if(curr < dp[i][m]){
                        dp[i][m] = curr;
                        opt = u;
                }
        }
        go(i,l,m,optl, opt);
        go(i,m+1,r,opt,optr);
}

int main(){
        fastio;
        cin >> n >> k;
        REP(i,0,n) cin >> c[i];
        s[0] = 0;
        REP(i,0,n+1) s[i] = s[i-1] + c[i-1];
        REP(i,1,k+1) dp[i][n] = INF;
        REP(i,0,n) dp[0][i] = INF;
        dp[0][n] = 0;
        REP(i,1,k+1) go(i,0,n,0,n);
        cout << dp[k][0] << endl;
        return 0;
}

//Divide and Conquer Trick by Ands

void compute(int cnt, int l, int r, int optl, int optr){
        if(l > r) return ;
        int mid = ( l + r ) >> 1 ;
        int opt = -1 ;
        ll value = 1e18 ;
        int last = cnt^1 ;
        for(int idx = optl ; idx <= min(mid-1,optr); ++idx){
                ll tmp = dp[last][idx] + C[idx][mid] ;
                if(tmp < value){
                        value = tmp ;
                        opt = idx ;
                }
        }
        dp[cnt&1][mid] = value ;
        compute(cnt, l, mid-1, optl, opt);
        compute(cnt, mid+1, r, opt, optr);
}


int main(){
        //casos base
        for(int cnt = 2; cnt <= m ; ++cnt) compute(cnt&1, 0, n-1, 0, n-1) ;
}
```

5.6 Fractions

```
struct Frac{
    int num, den;
    Frac(){
        num = 0; den = 1;
    }
    Frac(int a, int b): num(a), den(b){}
    Frac(int a):num(a), den(1){}

    void normalize(){
        if(num == 0){
            den = 1;
        }
        if(den < 0){
            den = -den;
            num = -num;
        }
    }

    Frac fix(int a, int b){
        if(!a) return Frac(0,1);
        if(!b) return Frac(oo,1);
        int foo = gcd(abs(a),abs(b));
        Frac ret = Frac(a/foo, b/foo);
        ret.normalize();
        return ret;
    }

    Frac operator + (const Frac& other){
        int num2 = num*other.den + den*other.num, den2 = den*other.den;
        return fix(num2,den2);
    }
```

```
    Frac operator - (const Frac& other){
        int num2 = num*other.den - den*other.num, den2 = den*other.den;
        return fix(num2,den2);
    }

    Frac operator * (int c){
        int num2 = num*c, den2 = den;
        return fix(num2,den2);
    }

    Frac operator * (const Frac& other){
        int num2 = num*other.num, den2 = den*other.den;
        return fix(num2,den2);
    }

    Frac operator / (int c){
        int num2 = num, den2 = den * c;
        return fix(num2,den2);
    }

    Frac operator / (const Frac& other){
        int num2 = num*other.den, den2 = den*other.num;
        return fix(num2,den2);
    }

    bool operator < (const Frac& other) const{
        if(num * other.den < other.num*den) return true;
        return false;
    }

    bool operator == (const Frac& other) const{
        if(num == other.num && den == other.den) return true;
        return false;
    }
};
```

5.7 Longest Increasing Subsequence

```
// Simple O( nlogn ) Longest Increasing Subsequence
// Answer is stored in array b[N]

int LIS( vi &a ){
    int b[N];
    int sz = 0;
    REP(i,0,a.size()){
        int j = lower_bound( b , b + sz , a[ i ] ) - b;
        // (lower) a < b < c
        // (upper) a <= b <= c
        b[ j ] = a[ i ];
        if( j == sz ) sz++;
    }
    return sz;
}
```

5.8 Matrix Structure

```
const int MN = 111;
const int mod = 10000;

struct matrix {
  int r, c;
  int m[MN][MN];

  matrix (int _r, int _c) : r (_r), c (_c) {
    memset(m, 0, sizeof m);
  }

  void print() {
    for (int i = 0; i < r; ++i) {
      for (int j = 0; j < c; ++j)
        cout << m[i][j] << " ";
      cout << endl;
    }
  }

  int x[MN][MN];
  matrix & operator *= (const matrix &o) {
    memset(x, 0, sizeof x);
    for (int i = 0; i < r; ++i)
      for (int k = 0; k < c; ++k)
        if (m[i][k] != 0)
          for (int j = 0; j < c; ++j) {
            x[i][j] = (x[i][j] + ((m[i][k] * o.m[k][j]) % mod) ) % mod;
          }
    memcpy(m, x, sizeof(m));
```

```
    return *this;
  }
};

void matrix_pow(matrix b, long long e, matrix &res) {
  memset(res.m, 0, sizeof res.m);
  for (int i = 0; i < b.r; ++i)
    res.m[i][i] = 1;

  if (e == 0) return;
  while (true) {
    if (e & 1) res *= b;
    if ((e >>= 1) == 0) break;
    b *= b;
  }
}
```

5.9 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef
tree<
        int,
        null_type,
        less<int>,
        rb_tree_tag,
        tree_order_statistics_node_update
>ordered_set;
// ordered_set
// X.find_by_order(k) returns an iterator to the k-th largest element (
    counting from zero)
// X.order_of_key(v) returns the number of items in a set that are
    strictly smaller than v
int main() {
        int N;
        ordered_set Y;
        Y.insert(5);
        trace (*Y.find_by_order(0));
}
```

5.10 Parallel Binary Search

```
//Cada query esta en (low[i], high[i]]
//Tocheck tiene los valores acutales a verificar
//en el bsearch

//Solved puede tener 1, -1
//1: el unico valor posible cumple
//-1: no hay respuesta

int low[MAXN];
int high[MAXN];
char solved[MAXN];
vector< int > tocheck[MAXN];

int main(){
        // Leer n, m
        // Leer a[i], b[i] (i en [1, m])
        // Leer q: queries
        // Leer x[i], y[i], z[i] (i en [0, q])

        for(int i = 0; i < q; ++i)
        low[i] = 0, high[i] = m;

        bool done = 0;
        DSU uf(n); // DSU structure
        int curvis;
        while(!done){
                done = 1;
                for(int i = 0; i < q; ++i){
                        int mid = (low[i] + high[i]) >> 1;
                        tocheck[mid].pb(i);
                }
                uf.clear(n);
                int last = -1;
                for(int value = 0; value <= m; ++value){
                        if(tocheck[value].empty()) continue;
                        for(int i = last + 1; i <= value; ++i)
                                uf.join(a[i], b[i]);
                        last = value;
                        while(!tocheck[value].empty()){
                                int id = tocheck[value].back();
                                tocheck[value].pop_back();
                                int u = x[id], v = y[id];
                                int visited = z[id];
```

```
                    if(low[id] + 1 == high[id]) solved[id] = 1;
                    if(uf.connected(u, v)) curvis = uf.size(u);
                    else curvis = uf.size(u) + uf.size(v);
                    if(curvis >= visited) high[id] = value;
                    else low[id] = value;
                    if(low[id] == high[id]) solved[id] = -1;
                }
            }
            for(int i = 0; i < q; ++i)
                if(solved[i] == 0) done = 0;
        }
        for(int i = 0; i < q; ++i)
            if(solved[i] == -1) cout << -1 << endl;
            else cout << high[i] << endl;
}
```

5.11 Unordered Map

```
unordered_map<int,int> mp;
mp.reserve(1024); // power of 2 is better
mp.max_load_factor(0.25); // 0.75 used in java
```

6 Network Flows

6.1 Bipartite Matching

```
// O(V*E) maximum bipartite matching
int p[MAX];           // Parent of right-node v in the matching
int vis[MAX];  // Whether left-node u has been visited
vi adj[MAX];   // Standard adjacency list

int match(int u) {
        if (vis[u]) return 0;
        vis[u] = 1;
        REP(i,0,adj[u].size()){
                int v = adj[u][i];
                if (p[v] == -1 || match(p[v])) {
                        p[v] = u;
                        return 1;
                }
        }
    return 0;
}

int main(){
        // build adj here with n left nodes
    // and V total nodes
        int n = 1000000;
    int V = 2000000;
        int maxMatch = 0;
        REP(i,0,V) p[i] = -1;
        REP(u,0,n){
                REP(i,0,n) vis[i] = 0;
                maxMatch += match(u);
        }
        printf("Found %d matchings\n", maxMatch)
}
```

6.2 Dinic Flow

```
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source and sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain actual flow values, look at edges with capacity > 0
//       (zero capacity edges are residual edges).

typedef long long LL;

struct Edge {
  int u, v;
  LL cap, flow;
  Edge() {}
  Edge(int u, int v, LL cap): u(u), v(v), cap(cap), flow(0) {}
};
```

```
struct Dinic {
  int N;
  vector<Edge> E;
  vector<vector<int>> g;
  vector<int> d, pt;

  Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge(int u, int v, LL cap) {
    if (u != v) {
      E.emplace_back(Edge(u, v, cap));
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(Edge(v, u, 0));
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS(int S, int T) {
    queue<int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while(!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k: g[u]) {
        Edge &e = E[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    }
    return d[T] != N + 1;
  }

  LL DFS(int u, int T, LL flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      Edge &e = E[g[u][i]];
      Edge &oe = E[g[u][i]^1];
      if (d[e.v] == d[e.u] + 1) {
        LL amt = e.cap - e.flow;
        if (flow != -1 && amt > flow) amt = flow;
        if (LL pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    }
    return 0;
  }

  LL MaxFlow(int S, int T) {
    LL total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (LL flow = DFS(S, T))
        total += flow;
    }
    return total;
  }
};
```

6.3 Edmonds Blossom

```
// Maximum general matching (not necessarily bipartite)
// Make sure to set N in main()
// Claimed O(N^4) running time

int N; // the number of vertices in the graph
typedef vector<int> vi;
typedef vector< vector<int> > vvi;
vi match;
vi vis;

void couple(int n, int m) { match[n]=m; match[m]=n; }

// True if augmenting path or a blossom (if blossom is non-empty).
// the dfs returns true from the moment the stem of the flower is
// reached and thus the base of the blossom is an unmatched node.
// blossom should be empty when dfs is called and
// contains the nodes of the blossom when a blossom is found.
bool dfs(int n, vvi &conn, vi &blossom) {
  vis[n]=0;
  REP(i, 0, N) if(conn[n][i]) {
    if(vis[i]==-1) {
```

```
      vis[i]=1;
      if(match[i]==-1 || dfs(match[i], conn, blossom)) {
                        couple(n,i);
                        return true;
                }
    }
    if(vis[i]==0 || SZ(blossom)) { // found flower
      blossom.pb(i); blossom.pb(n);
      if(n==blossom[0]) { match[n]=-1; return true; }
      return false;
    }
  }
  return false;
}

// search for an augmenting path.
// if a blossom is found build a new graph (newconn) where the
// (free) blossom is shrunken to a single node and recurse.
// if a augmenting path is found it has already been augmented
// except if the augmented path ended on the shrunken blossom.
// in this case the matching should be updated along the
// appropriate direction of the blossom.
bool augment(vvi &conn) {
        REP(m, 0, N) if(match[m]==-1) {
                vi blossom;
                vis=vi(N,-1);
                if(!dfs(m, conn, blossom)) continue;
                if(SZ(blossom)==0) return true; // augmenting path found

// blossom is found so build shrunken graph
                int base=blossom[0], S=SZ(blossom);
                vvi newconn=conn;
                REP(i, 1, S-1) REP(j, 0, N)
                        newconn[base][j]=newconn[j][base]|=conn[blossom[i]][
                            j];
                REP(i, 1, S-1) REP(j, 0, N)
                        newconn[blossom[i]][j]=newconn[j][blossom[i]]=0;
                newconn[base][base]=0; // is now the new graph
                if(!augment(newconn)) return false;
                int n=match[base];

// if n!=-1 the augmenting path ended on this blossom
    if(n!=-1) REP(i, 0, S) if(conn[blossom[i]][n]) {
      couple(blossom[i], n);
      if(i&1) for(int j=i+1; j<S; j+=2)
                                couple(blossom[j],blossom[j+1]);
      else for(int j=0; j<i; j+=2)
                                couple(blossom[j],blossom[j+1]);
      break;
    }
    return true;
  }
  return false;
}

// conn is the NxN adjacency matrix
// returns the number of edges in a max matching.
int edmonds(vvi &conn) {
  int res=0;
  match=vi(N,-1);
  while(augment(conn)) res++;
  return res;
}

/****************************************************/
set<pair<int,int> > used;
int main(){
  int n;
  cin >> n;
  N = n;
  vvi conn;
  vi tmp;
  tmp.assign(n,0);
  REP(i, 0, n) conn.push_back(tmp);
  int u, v;
  while(cin >> u >> v){
    u--; v--;
    if(u > v) swap(u,v);
    if(used.count(make_pair(u,v))) continue;
    used.insert(make_pair(u,v));
    conn[u][v] = conn[v][u] = 1;
  }
  int res = edmonds(conn);
  cout<<res*2<<endl;
  REP(i, 0, n) {
    if(match[i] > i){
      cout<<i+1<<" "<<match[i] + 1<<endl;
    }
  }
  return 0;
```

```
}
```

6.4 Min Cost Max Flow

```
const int MAXN = 5010;

const ll INF = 1e15;
struct edge { int dest;ll origcap, cap; ll cost; int rev; };

struct MinCostMaxFlow {

    vector<edge> adj[MAXN];
    ll dis[MAXN], cost;
    int source, target, iter;
    ll cap;
    edge* pre[MAXN];
    int queued[MAXN];
    MinCostMaxFlow (){}
    void AddEdge(int from, int to, ll cap, ll cost) {
        adj[from].push_back(edge {to, cap, cap, cost, (int)adj[to].size()})
            ;
        adj[to].push_back(edge {from,0, 0, -cost, (int)adj[from].size()
             - 1});
    }

    bool spfa() {
        REP(i,0,MAXN) queued[i] = 0;
        fill(dis, dis + MAXN, INF);
        queue<int> q;
        pre[source] = pre[target] = 0;
        dis[source] = 0;
        q.emplace(source);
        queued[source] = 1;
        while (!q.empty()) {
            int x = q.front();

            ll d = dis[x];
            q.pop();
            queued[x] = 0;
            for (auto& e : adj[x]) {
                int y = e.dest;
                ll w = d + e.cost;
                if (e.cap < 1 || dis[y] <= w) continue;
                dis[y] = w;
                pre[y] = &e;
                if(!queued[y]){
                        q.push(y);
                        queued[y] = 1;
                }
            }
        }
        edge* e = pre[target];

        if (!e) return 0;
        while (e) {
            edge& rev = adj[e->dest][e->rev];
            e->cap -= cap;
            rev.cap += cap;
            cost += cap * e->cost;
            e = pre[rev.dest];
        }
        return 1;
    }

    pair<ll,ll> GetMaxFlow(int S, int T) {
        cap = 1, source = S, target = T, cost = 0;
        while(spfa()) {}
        ll totflow = 0;
        for(auto e: adj[source]){
            totflow += (e.origcap - e.cap);
        }
        return make_pair(totflow, cost);
    }
};
```

6.5 Push Relabel Max Flow

```
// Fast O(|V|^3) flow, works for n ~ 5000 with no problem
// Actual flow values in edges with cap > 0 (0 cap = residual)

typedef long long LL;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
```

```
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) :
       N(N), G(N), excess(N), dist(N), active(N), count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

      void Enqueue(int v) {
              if (!active[v] && excess[v] > 0) {
                      active[v] = true; Q.push(v);
              }
      }

  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }

  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }

  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
        dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
  }

      void Discharge(int v) {
      for (int i = 0; excess[v] > 0 && i < G[v].size(); i++)
              Push(G[v][i]);
      if (excess[v] > 0) {
              if (count[dist[v]] == 1) Gap(dist[v]);
              else Relabel(v);
              }
      }

  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }

    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }

    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
    return totflow;
  }
};
```

7 Strings

7.1 Aho Corasick + Compression

```
// Aho Corasick automaton. O(n) in size of Trie.
// Allows searching for a dictionary of patterns in a string.
// Consider using DP[u, pos], for instance.

const int MAXN = 500000; // Sum of words*length
const int SZA = 26;           // Alphabet size

map<int,int> adj[MAXN]; // Trie
int isEnd[MAXN];      // Example: How many words end at node u
int gid;                            // Id of last node set
int f[MAXN];          // Aho Corasick failure function

void init(int id){
	isEnd[id] = 0;
	adj[id].clear();
}

void add(string s){
	int u = 0;    // Current node
	REP(p,0,sz(s)){
		int id = s[p] - 'a';
		if (!adj[u].count(id)){
			adj[u][id] = ++gid;   // Lazy initialization
			init(gid);
		}
		u = adj[u][id];
	}
	isEnd[u]++;
}

void build(){
   // BFS-DP Aho Corasick construction
	queue<int> q;
	f[0] = 0;
	REPIT(it, adj[0]){
		int u = it->snd;
		q.push(u);
		f[u] = 0;
	}
	while (!q.empty()){
		int e = q.front();
		q.pop();
		REPIT(it, adj[e]){
			int i = it->fst;
			int u = it->snd;
			q.push(u);
			int v = f[e];
			while (v && !adj[v].count(i)) v = f[v];
			f[u] = (adj[v].count(i) ? adj[v][i] : 0);
			// Aggregate necessary information here
			// In general, S[u] += S[f[u]]
			isEnd[u] += isEnd[f[u]];
		}
	}
}

// Search string s for all strings in trie
ll search(string s){
	ll ans = 0;
	int u = 0;
	REP(p,0,sz(s)){
		int id = s[p] - 'a';
		while (u && !adj[u].count(id)) u = f[u];
		if (adj[u].count(id)) u = adj[u][id];
		ans += isEnd[u];
	}
	return ans;
}

int main(){
	gid = 0;
	init(0);
	// Ready for add(s), build(), search(t)
	return 0;
}
```

7.2 Aho Corasick

```
// ----------------------------aho corasick--------------------------
// cantidad de repeticiones de cada string sobre un text en O(M+N)

#define N 100000 // ñtamao del text
#define M 1005 //ñtamao de cada string a buscar
```

```
ll n;
char text[N];// string donde buscar
char buf[N]; // string a buscar
ll cnt[M]; // cnt[i]: cantidad de ocurrencias del string i

ll root, nodes;
// nodes: cantidad de nodos en el trie,
//root: que nodo del trie estoy
struct trieNode{
    bool seen;
    ll matchFail,fail;
    vi matches;
    map< char, ll > next;
    trieNode(){}
    trieNode(bool seen, ll &matchFail, ll &fail, vi & matches, map<char,
        ll> & next):
    seen(seen), matchFail(matchFail),fail(fail),matches(matches), next(
        next){}
} trie[N];
// antes de insertar, notar que root=0 y nodes=1
inline void insert(char * s, ll wordId){ //
    //wordId: id del string
    ll x = root, ta=strlen(s);
    REP(i,0,ta){
        ll &nxt = trie[x].next[ s[i] ];
        if (!nxt) nxt = ++nodes;
        x = nxt;
    }
    trie[x].matches.push_back(wordId);
}

inline ll find(ll x, char ch){
    while (x && !trie[x].next.count(ch)) x = trie[x].fail;
    return x ? trie[x].next[ch] : root;
}

inline void bfs(){
    trie[root].fail = 0;
    queue< ll > q;
    q.push(root);
    while(q.empty()){
        ll u = q.front(),v; q.pop();
        char ch;
        for (auto &it: trie[u].next){
            ch = it.fst, v = it.snd;
            ll f = find(trie[u].fail, ch);
            trie[v].fail = f;
            trie[v].matchFail = trie[f].matches.empty() ? trie[f].matchFail
                 : f;
            q.push(v);
        }
    }
}
inline void search(){
    ll x = root;
    ll ta=strlen(text);
    REP(i,0,ta){
        x = find(x, text[i]);
        for (ll t = x; t && !trie[t].seen; t = trie[t].matchFail){
            trie[t].seen = true;
            REP(j,0, sz(trie[t].matches)) cnt[trie[t].matches[j]] ++;
        }
    }
}

int main(){
    root = ++nodes;//inicializacion
    scanf( "%s", &text );
    scanf( "%d", &n );
    REP(i,0, n){
        scanf( "%s", &buf );
        insert(buf, i);
    }
    bfs(); search();
    REP(i,0,n) printf( "%s\n", cnt[i]>0 ? "Y" : "N" );

    return 0;
}
```

7.3 Knuth Morris Pratt

```
// KMP algorithm for finding a pattern in a string in O(n+m).

const int MAX = 1000000;

int b[MAX]; // Fail function
char p[MAX]; // Pattern string
```

```cpp
char t[MAX]; // Text string

int n; // Text string length
int m; // Pattern string length

void kmpPreprocess(){
    int i=0, j=-1;
    b[i]=j;
    while (i<m){
        while (j>=0 && p[i]!=p[j]) j=b[j];
        i++; j++;
        b[i]=j;
    }
}

void report(int x){
        cout << "Found on: " << x << endl;
}

void kmpSearch(){
    int i=0, j=0;
    while (i<n){
        while (j>=0 && t[i]!=p[j]) j=b[j];
        i++; j++;
        if (j==m){
            report(i-j);
            j=b[j];
        }
    }
}
```

7.4 Manacher Algorithm

```cpp
// Manacher's algorithm for finding all palindromes
// in a string in O(n).

int n;
char s[200200];
char aux[100100];
int p[200200];

int main(){
        scanf("%s%n", aux, &n);
        s[0] = '^';
        s[1] = '#';
        REP(i,0,n){
                s[2*i+2] = aux[i];
                s[2*i+3] = '#';
        }
        s[2*n+2] = '\0';
        int c = 0, r = 0;
        REP(i,0,2*n+2){
                if (i > r) p[i] = 0;
                else p[i] = min(r-i, p[2*c-i]);
                while (s[i+p[i]+1] == s[i-p[i]-1]) p[i]++;
                if (i + p[i] > r){
                        c = i;
                        r = i + p[i];
                }
        }

        printf("%s\n", s);
        REP(i,0,2*n+2) {
                printf("%d", p[i]);
        }
        printf("\n");
        return 0;
}
```

7.5 Palindromic Tree

```cpp
// adamant's palindromic tree online O(n*log(|E|)) construction
// Tutorial: http://adilet.org/blog/25-09-14/
// Add/Delete operation can be supported in O(logn) by doing
// check(link[v]), v = slink[v] in get_link
// (periodicity -> same initial char)
const int maxn = 5e5, sigma = 26, INF = 1e9;
int s[maxn], len[maxn], link[maxn], to[maxn][sigma];
int n, last, sz;
// All these optional (palindromic factoring)
int d[maxn], slink[maxn], dpe[maxn], dpo[maxn];
int anse[maxn], anso[maxn], prve[maxn], prvo[maxn];

void init(){ // Call with n=0
        s[n++] = -1;
        link[0] = 1;
```

```
        len[1] = -1;
        sz = 2;
        anse[0] = 0;
        anso[0] = INF;
}

int get_link(int v){
        while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
        return v;
}

ii getmin(int v, int* ans, int* dp, int* prv){
        dp[v] = ans[n - (len[slink[v]] + d[v]) - 1];
        int best = n - (len[slink[v]] + d[v]) - 1;
        if (d[v] == d[link[v]]){
                if (dp[v] > dp[link[v]]){
                        dp[v] = dp[link[v]];
                        best = prv[n-1-d[v]];
                }
        }
        return mp(dp[v] + 1, best);
}

void add_letter(int c){
        s[n++] = c;
        last = get_link(last);
        if(!to[last][c]) {
                len [sz] = len[last] + 2;
                link[sz] = to[get_link(link[last])][c];
                d[sz] = len[sz] - len[link[sz]];
                if (d[sz] == d[link[sz]]) slink[sz] = slink[link[sz]];
                else slink[sz] = link[sz];
                to[last][c] = sz++;
        }
        last = to[last][c];

        anse[n-1] = INF;
        for (int v = last; len[v] > 0; v = slink[v]){
                ii acte = getmin(v, anso, dpe, prve);
                if (act.fst < anse[n-1]){
                        anse[n-1] = act.fst;
                        prve[n-1] = act.snd;
                }
        }
        anso[n-1] = INF;
        for (int v = last; len[v] > 0; v = slink[v]){
                ii act = getmin(v, anse, dpo, prvo);
                if (act.fst <= anso[n-1]){
                        anso[n-1] = act.fst;
                        prvo[n-1] = act.snd;
                }
        }
}
```

7.6 Suffix Array

```
// ----------------------------Suffix array--------------------------
// construccion en nlog^2(n)
//usa lcp(x,y)=mi[lcp(x,x+1),lcp(x+1,x+2)....lcp(y-1,y)]
//construye el lcp(x,y) con sparce table, notar que los indices son 0 base
//s=ababa
//s1[0]=ababa,s1[1]=baba,s1[2]=aba, s1[3]=ba,s1[4]=a, s1[5]='$'
//s2={$,a,aba,ababa,ba,baba}={5,4,2,0,3,1}=r
//r[i] lista de los sufijos ordenados en 0 base
//indice de s1={ababa,baba,aba,ba,a,$}={3,5,2,4,1,0}=p
//p[i] posicion del i substring en el suffix array (s1) en 0 base

#define N 100010
#define M 20
inline ll ma(ll a, ll b){ return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct SA{
  //asignar s:string(char), n ñtamao del string
  ll n,t;
  ll p[N],r[N],h[N];
  char s[N];
  ll rmq[M][N];
  ll flog2[N];
  inline void fix_index(ll b, ll e){
    ll lastpk, pk, d;
    lastpk = p[r[b]+t];
    d = b;
    REP(i,b,e){
      if (((pk = p[r[i]+t]) != lastpk) && (b > lastpk || pk >= e)){
        lastpk = pk;
        d = i;
```

```
    }
    p[r[i]]= d;
  }
}
//calculo de r y p
inline void suff_arr(){
  s[n++] = '$';
  ll bc[256];
  REP(i,0,256) bc[i]=0;
  REP(i,0,n) bc[(ll)s[i]]++;
  REP(i,1,256) bc[i] += bc[i-1];
  RREP(i,n-1,0) r[--bc[(ll)s[i]]] = i;
  RREP(i,n-1,0) p[i] = bc[(ll)s[i]];
  for (t = 1; t < n; t<<=1){
    for (ll i = 0, j = 1; i < n; i = j++){
      while (j < n && p[r[j]] == p[r[i]]) ++j;
      if (j-i > 1){
        sort(r+i, r+j, [&](const ll &i, const ll &j){return p[i+t] < p[j+
             t];});
        fix_index(i, j);
      }
    }
  }
}
//calcula h[i] en O(n) usando Kasai algorithm
inline void initlcp(){
  ll tam = 0, j;
  REP(i,0,n-1){
    j = r[p[i]-1];
    while(s[i+tam] == s[j+tam]) ++tam;
    h[p[i]-1] = tam;
    if (tam > 0) --tam;
  }
}
//construccion del RMQ para hallar lcp en un rango
inline void makelcp(){
  initlcp();
  REP(i,0,n-1) rmq[0][i] = h[i];
  ll lg = 0, pw = 1;
  do{
    REP(i,pw,pw*2) flog2[i] = lg;
    lg++; pw*=2;
    REP(i,0,n-1){
      if (i+pw/2 < n-1) rmq[lg][i] = mi(rmq[lg-1][i], rmq[lg-1][i+pw/2]);
      else rmq[lg][i] = rmq[lg-1][i];
    }
  } while(pw < n);
}
//calcula el lcp en [i,j] de s1(suffix array);
inline ll lcp(ll i, ll j){
  if (i == j) return n - r[i] - 1;
  ll lg = flog2[j-i], pw = (1<<lg);
  return mi(rmq[lg][i], rmq[lg][j-pw]);
}
//limpia y construye
inline void build(){
  memset(p,0,sizeof(p));
  memset(r,0,sizeof(r));
  memset(h,0,sizeof(h));
  memset(rmq,0,sizeof(rmq));
  memset(flog2,0,sizeof(flog2));
  suff_arr();
  makelcp();
}
};
int main(){
  //ejemplo, hallar la cantidad de diferentes substrings para t1 strings;
  ll t1; scanf("%lld", &t1);
  REP(ik,0,t1){
    SA sa; scanf("%s", &sa.s);
    ll ta=strlen(sa.s);
    sa.n=ta; sa.build();
    ll ans=0;
    REP(i,1,ta){
        ans+=sa.lcp(i,i+1);
    }
    ll xd=(ta*(ta+1)/2)-ans;
    printf("%lld\n",xd);
  }
  return 0;
}
```

7.7 Suffix Automaton

```
// O(n) Online suffix automaton construction
// len[u]: Max length of a string accepted by u
// link[u]: Suffix link of u
// Link edges give the suffix tree of reverse(s)
```

```
// Terminal nodes can be obtained by
//     traversing last's links

const int MAX = 1000000;
int len[MAX*2];
int link[MAX*2];
map<char,int> adj[MAX*2];
int sz, last;

// To reuse, clear adj[]
void sa_init() {
	sz = last = 0;
	len[0] = 0;
	link[0] = -1;
	sz++;
}

void sa_extend (char c) {
	int cur = sz++;
	len[cur] = len[last] + 1;
	int p;
	for (p=last; p!=-1 && !adj[p].count(c); p = link[p])
		adj[p][c] = cur;
	if (p == -1)
		link[cur] = 0;
	else {
		int q = adj[p][c];
		if (len[p] + 1 == len[q])
			link[cur] = q;
		else {
			int clone = sz++;
			len[clone] = len[p] + 1;
			adj[clone] = adj[q];
			link[clone] = link[q];
			for (; p != -1 && adj[p][c] == q; p = link[p])
				adj[p][c] = clone;
			link[q] = link[cur] = clone;
		}
	}
	last = cur;
}
```

7.8 Z-Algorithm

```
//Zfun(i) devuelve la longitud del maximo prefijo que empieza en i
vi Zfun(string s){
	vi Z(s.sz,0);
	int l = 0, r = 0;
	REP(i,1,sz(s)){
		if ( i<=r ) Z[i] = min(Z[i-l], r-i+1);
		while ( i+Z[i]<s.sz and s[i+Z[i]]==s[Z[i]] ) Z[i]++;
		if ( i+Z[i]-1>r ) l = i, r = i+Z[i]-1;
	}
	return Z;
}
```

8 Templates

8.1 Header Template

```
#include <bits/stdc++.h>
#include <sstream>
using namespace std;
#define fastio ios_base::sync_with_stdio(0);cin.tie(0);
#define trace(x) cerr << #x << ": " << x << '\n'
#define trace2(x,y) cerr << #x << ": " << x << " | " << #y << ": " << y
     << '\n';
#define trace3(x,y,z) cerr << #x << ": " << x << " | " << #y << ": " << y
     << " | " << #z << ": " << z << '\n';
#define all(v) (v).begin(),(v).end()
#define pb push_back
#define sz(v) ((int)v.size())
#define REP(i,x,y) for(int (i)=(x);(i)<(y);(i)++)
#define RREP(i,x,y) for(int (i)=(x);(i)>=(y);(i)--)
#define mp make_pair
#define fst first
#define snd second
typedef long long ll;
typedef pair<ll, ll> ii;

const int MOD = 1e9 + 7;
const int oo = 1e9;
const ll INF = 1e18;
const long double EPS = 1e-11;
```

8.2 Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -Wextra -Wno-sign-compare -O2 -g

all: %
%: %.cpp
	$(CXX) $(CXXFLAGS) -o $@ $@.cpp
```

8.3 Stack Size

```
 #include <sys/resource.h>

int main (int argc, char **argv){
    const rlim_t kStackSize = 64L * 1024L * 1024L; // min stack size = 64
        Mb
    struct rlimit rl;
    int result;

    result = getrlimit(RLIMIT_STACK, &rl);
    if (result == 0)
    {
        if (rl.rlim_cur < kStackSize)
        {
            rl.rlim_cur = kStackSize;
            result = setrlimit(RLIMIT_STACK, &rl);
            if (result != 0)
            {
                fprintf(stderr, "setrlimit returned result = %d\n", result)
                    ;
            }
        }
    }

    // ...

    return 0;
}
```

8.4 Vim Configuration (vimrc)

```
set number
set autoindent
set showmode
set backspace=indent,eol,start
set mouse=a
set ts=3
set shiftwidth=3
set pastetoggle=<F10>
colorscheme chroma
syntax on

nmap ,c <Esc>i<Home>//<Esc>
nmap ,d <Esc><Home>i<Del><Del><Esc>
```

9 Utils

9.1 MinXOR

```
/*
        Mininum XOR-Pair on an array in O(n)
        Trie-based Implementation
*/


#define INT_SIZE 32

struct TrieNode{
    int value;
    TrieNode * Child[2];
};

TrieNode * getNode(){
    TrieNode * newNode = new TrieNode;
    newNode->value = 0;
    newNode->Child[0] = newNode->Child[1] = NULL;
    return newNode;
}

void insert(TrieNode *root, int key){
    TrieNode *temp = root;

    for (int i = INT_SIZE-1; i >= 0; i--){
```

```
        bool current_bit = (key & (1<<i));

        if (temp->Child[current_bit] == NULL)
            temp->Child[current_bit] = getNode();

        temp = temp->Child[current_bit];
    }

    temp->value = key ;
}

int  minXORUtil(TrieNode * root, int key){
    TrieNode * temp = root;

    for (int i=INT_SIZE-1; i >= 0; i--){

        bool current_bit = ( key & ( 1<<i) );

        if (temp->Child[current_bit] != NULL)
            temp = temp->Child[current_bit];

        else if(temp->Child[1-current_bit] !=NULL)
            temp = temp->Child[1-current_bit];
    }
    return key ^ temp->value;
}

int minXOR(int arr[], int n){
    int min_xor = INT_MAX;

    TrieNode *root = getNode();
    insert(root, arr[0]);

    for (int i = 1 ; i < n; i++){
        min_xor = min(min_xor, minXORUtil(root, arr[i]));
        insert(root, arr[i]);
    }
    return min_xor;
}

int main(){
    int arr[] = {9, 5, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << minXOR(arr, n) << endl;
    return 0;
}
```

9.2 Offline Less K-Counting

```
//-------------------inversiones en un rango (offline)----------------
// ar[]: arreglo, queries=queri.pb(l,r,valor)
//assignar n,q; ez[i] respuesta para la querie i
//hacer read y make;

struct ST{
  ll n,q;
  vector<tri> querie;
  ll t[2*N],ar[N];
  ll poar[N],pok[N],ark[N],ez[N];
  vii v,v1;
  inline ll Op(ll &a,ll &b){ return a+b;}
  inline void build (){
    RREP(i,n-1,1) t[i]=Op(t[i<<1],t[i<<1|1]);
  }
  inline void modify (ll p, ll val){
    for(t[p+=n]=val;p>1;p>>=1) t[p>>1]=Op(t[p],t[p^1]);
  }
  inline ll que(ll l, ll r){
    ll res=0;
    for(l+=n,r+=n;l<r;l>>=1,r>>=1){
      if(l&1) res+=t[l++];
      if(r&1) res+=t[--r];
    }
    return res;
  }
  ll p1=0, p2=0,po=0;
  inline void read(){
    REP(i,0,n) v.push_back({ar[i],i});
    sort(all(v));
    REP(i,0,n) poar[p1++]=v[i].snd;
    REP(u,0,q){
      ll k=querie[u].itm3;
      ark[u]=k;
      v1.push_back({k,u});
    }
    sort(all(v1));
    REP(i,0,q) pok[p2++]=v1[i].snd;
  }
```

```
  inline void make(){
    REP(i,0,n) t[i+n]=0; build();
    REP(i,0,q){
      ll x=pok[i];
      // <k, <= k en l,r(despues del &&)
      //inversa , hacer t[i+n]=1;
      while(po<n && ar[poar[po]]<=ark[x]) modify(poar[po++],1);
      ez[x]=que(querie[x].itm1-1,querie[x].itm2);
    }
  }
}st;

int main(){fastio;
  ll n; cin>>n;
  st.n=n;
  REP(i,0,n) cin>>st.ar[i];
  ll q; cin>>q;
  st.q=q;
  REP(i,0,q){
    ll l,r,k; cin>>l>>r>>k;
    st.querie.push_back({l,{r,k}});
  }
  st.read(); st.make();
  REP(i,0,q) cout<<st.ez[i]<<endl;
  return 0;
}
```

9.3 Online Less K-Counting

```
/*-------------------inversiones en un rango (online)----------------
 construccion amortizada a nlog(n);
 cada querie en log^2(n);*/

struct T{
  vi v;
  T () {}
  T (vi v): v(v){}
};
struct ST{
  ll n,ans;
  T t[2*N];
  inline T Op(T &val1, T &val2 ){
    vi v;
    REP(i,0,val1.v.size()) v.pb(val1.v[i]);
    REP(i,0,val2.v.size()) v.pb(val2.v[i]);
    sort(all(v));
    T ty;
    ty.v=v;
    return ty;
  }
  inline ll Op1( T &val1,ll &k){
    ans=0;
    //usar upper_bound para valores mayores a k
    //usar quitar el val1.v.size() para valores menores o iguales a k
    // usar lower_bound para valores estrictamente menoes a k(sin el val1.
        v.size())
    ans+=val1.v.size()-(upper_bound(all (val1.v),k)-val1.v.begin());
    return ans;
  }
  inline void build(){
    RREP(i,n-1,1) t[i]=Op(t[i<<1],t[i<<1|1]);
  }
  inline ll que(ll l, ll r, ll k){
    ll ans=0;
    for(l+=n,r+=n;l<r;l>>=1,r>>=1){
      if(l&1) ans+=Op1(t[l++],k);
      if(r&1) ans+=Op1(t[--r],k);
    }
    return ans;
  }
}st;

int main(){fastio;
  ll n; cin>>n;
  st.n=n;
  REP(i,0,n) {
    ll x; cin>>x;
    st.t[i+n].v.push_back(x);
  }
  st.build();
  ll q,ans=0,l,r,k; cin>>q;
  REP(i,0,q){
    cin>>l>>r>>k;// queries 1 base
    ans=st.que(l-1,r,k);
    cout<<ans<<endl;
  }
  return 0;
}
```