Contents

1 DP

1.1 DSU on Tree

```
vector<int> *pvec[MAX];
vector<int> G[MAX];
int sz[MAX],color[MAX],color_counter[MAX];
pair<ll,int> Info[MAX];
pair<ll,int>dfs(int u,int p=-1,bool keep=false)
{
    int i,j,k,child,hchild=-1;
    for(i=0; i<G[u].size(); i++)
    {
        if(G[u][i]==p) continue;
        if(hchild==-1||sz[hchild]<sz[G[u][i]])
        {
            hchild=G[u][i];
        }
    }
    for(i=0; i<G[u].size(); i++)
    {
        if(G[u][i]==p||G[u][i]==hchild) continue;
        dfs(G[u][i],u,false);
    }
    if(hchild!=-1)
    {
        Info[u]=dfs(hchild,u,true);
        pvec[u]=pvec[hchild];
    }
    else
    {
        pvec[u]=new vector<int> ();
    }
    pvec[u]->push_back(u);
    color_counter[color[u]]++;
    if(color_counter[color[u]]>Info[u].second)
    {
        Info[u].second=color_counter[color[u]];
        Info[u].first=color[u];
    }
    else if(color_counter[color[u]]==Info[u].second)
    {
        Info[u].first=Info[u].first+color[u];
    }
    for(i=0; i<G[u].size(); i++)
    {
        if(G[u][i]==p||G[u][i]==hchild) continue;
        child=G[u][i];
        for(j=0; j<(*pvec[child]).size(); j++)
        {
            k=(*pvec[child])[j];
            pvec[u]->push_back(k);
            color_counter[color[k]]++;
            if(color_counter[color[k]]>Info[u].second)
            {
                Info[u].second=color_counter[color[k]];
                Info[u].first=color[k];
            }
          else if(color_counter[color[k]]==Info[u].second)
            {
                Info[u].first=Info[u].first+color[k];
            }
        }
    }
    if(!keep)
    {
        for(j=0; j<(*pvec[u]).size(); j++)
        {
            k=(*pvec[u])[j];
            color_counter[color[k]]--;
        }
    }
    return Info[u];
}
```

1.2 Divide and Conquer Optimization

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);
long long C(int i, int j);
// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;
    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
    best = min(best, {(k ? dp_before[k - 1] : 0) + C(k
    }
    dp_cur[mid] = best.first;
    int opt = best.second;
    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}   int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);
    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }
    return dp_before[n - 1];
}
```

1.3 Li Chao Tree

```
#include <bits/stdc++.h>
#include <vector>
#include<math.h>
#include<string.h>
using namespace std;
#define MAX 200005
#define MOD 1000000007
#define INF 10000000000
#define EPS 0.0000000001
#define FASTIO ios_base::sync_with_stdio(false);cin.ti
#include <ext/pb_ds/assoc_container.hpp>
```

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
class LiChaoTree
{
    long long L,R;
    bool minimize;
    int lines;
    struct Node
    {
        complex<long long> line;
        Node *children[2];
        Node(complex<long long> ln= {0,1000000000000000000})
        {
            line=ln;
            children[0]=0;
            children[1]=0;
        }
    } *root;
    long long dot(complex<long long> a, complex<long long> b)
    {
        return (conj(a) * b).real();
    }
    long long f(complex<long long> a, long long x)
    {
        return dot(a, {x, 1});
    }
    void clear(Node* &node)
    {
        if(node->children[0])
        {
            clear(node->children[0]);
        }
        if(node->children[1])
        {
            clear(node->children[1]);
        }
        delete node;
    }
    void add_line(complex<long long> nw, Node* &node, long long l, long long r)
    {
        if(node==0)
        {
            node=new Node(nw);
            return;
        }
        long long m = (l + r) / 2;
        bool lef = (f(nw, l) < f(node->line, l)&&minimize)||((!minimize)&&f(nw, l) > f(node->line, l));
        bool mid = (f(nw, m) < f(node->line, m)&&minimize)||((!minimize)&&f(nw, m) > f(node->line, m));
        if(mid)
        {
            swap(node->line, nw);
        }
        if(r - l == 1)
        {
            return;
        }
        else if(lef != mid)
        {
            add_line(nw, node->children[0], l, m);
        }
        else
        {
            add_line(nw, node->children[1], m, r);
        }
    }
    long long get(long long x, Node* &node, long long l, long long r)
    {
        long long m = (l + r) / 2;
        if(r - l == 1)
        {
            return f(node->line, x);
        }
        else if(x < m)
        {
            if(node->children[0]==0) return f(node->line, x);
            if(minimize) return min(f(node->line, x), get(x, node->children[0], l, m));
            else return max(f(node->line, x), get(x, node->children[0], l, m));
        }
        else
        {
            if(node->children[1]==0) return f(node->line, x);
            if(minimize) return min(f(node->line, x), get(x, node->children[1], m, r));
            else return max(f(node->line, x), get(x, node->children[1], m, r));
        }
    }
public:
    LiChaoTree(long long l=-1000000001,long long r=1000000001,bool mn=false)
    {
        L=l;
        R=r;
        root=0;
        minimize=mn;
        lines=0;
    }
    void AddLine(pair<long long,long long> ln)
    {
        add_line({ln.first,ln.second},root,L,R);
        lines++;
    }
    int number_of_lines()
    {
        return lines;
    }
    long long getOptimumValue(long long x)
    {
        return get(x,root,L,R);
    }
    ~LiChaoTree()
    {
        if(root!=0) clear(root);
    }
};
int main()
{
    return 0;
}
```

1.4 zero$_{m}atrix$

```
int zero_matrix(vector<vector<int>> a) {
    int n = a.size();
    int m = a[0].size();
    int ans = 0;
    vector<int> d(m, -1), d1(m), d2(m);
    stack<int> st;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (a[i][j] == 1)
                d[j] = i;
        }
        for (int j = 0; j < m; ++j) {
            while (!st.empty() && d[st.top()] <= d[j])
                st.pop();
            d1[j] = st.empty() ? -1 : st.top();
            st.push(j);
        }
        while (!st.empty())
            st.pop();
        for (int j = m - 1; j >= 0; --j) {
            while (!st.empty() && d[st.top()] <= d[j])
                st.pop();
            d2[j] = st.empty() ? m : st.top();
            st.push(j);
        }
        while (!st.empty())
            st.pop();
        for (int j = 0; j < m; ++j)
            ans = max(ans, (i - d[j]) * (d2[j] - d1[j] - 1));
    }
    return ans;
}
```

2 DS

2.1 BIT 2D

```
const int mx = 1002,my = 1002;
long long bit[4][mx][my];
void update( int x, int y, int val, int i ) {
  int y1;
  while( x<=mx ) {
    y1=y;
    while( y1<=my)
      bit[i][x][y1] += val, y1 += (y1&-y1);
    x += (x&-x);
  }
}
long long query( int x, int y, int i ) {
  long long ans=0; int y1;
  while( x>0 ) {
    y1 = y;
    while( y1>0 )
      ans += bit[i][x][y1], y1 -= (y1&-y1);
    x -= (x&-x);
  }
  return ans;
}
// add value k from (x1,y1) to (x2,y2) inclusive
void add( int x1, int y1, int x2, int y2, int k) {
  update(x1,y1,k,0);
  update(x1,y2+1,-k,0);
  update(x2+1,y1,-k,0);
  update(x2+1,y2+1,k,0);
  update(x1,y1,k*(1-y1),1);
  update(x1,y2+1,k*y2,1);
  update(x2+1,y1,k*(y1-1),1);
  update(x2+1,y2+1,-y2*k,1);
  update(x1,y1,k*(1-x1),2);
  update(x1,y2+1,k*(x1-1),2);
  update(x2+1,y1,k*x2,2);
  update(x2+1,y2+1,-x2*k,2);
```

```
    update(x1,y1,(x1-1)*(y1-1)*k,3);
    update(x1,y2+1,-y2*(x1-1)*k,3);
    update(x2+1,y1,-x2*(y1-1)*k,3);
    update(x2+1,y2+1,x2*y2*k,3);
}
// get value from (x1,y1) to (x2,y2) inclusive
long long get( int x1, int y1, int x2, int y2 ) {
    LL v1=query(x2,y2,0)*x2*y2 +
          query(x2,y2,1)*x2 +
          query(x2,y2,2)*y2 +
          query(x2,y2,3);
    LL v2=query(x2,y1-1,0)*x2*(y1-1) +
          query(x2,y1-1,1)*x2 +
          query(x2,y1-1,3) +
          query(x2,y1-1,2)*(y1-1);
    LL v3=query(x1-1,y2,0)*(x1-1)*y2 +
          query(x1-1,y2,2)*y2+
          query(x1-1,y2,1)*(x1-1) +
          query(x1-1,y2,3);
    LL v4=query(x1-1,y1-1,0)*(x1-1)*(y1-1) +
          query(x1-1,y1-1,1)*(x1-1) +
          query(x1-1,y1-1,2)*(y1-1) +
          query(x1-1,y1-1,3);
    LL ans=v1-v2-v3+v4;
    return ans;
}
```

2.2 CD - hellbent

```
vector <int> g[N]; int n, child[N], done[N];
void dfs_size(int u, int par) {
  child[u] = 1;
  for (int v: g[u]) {
    if (done[v] or v == par) continue;
    dfs_size(v, u); child[u] += child[v];
  }
}
int dfs_find_centroid(int u, int par, int sz) {
  for (int v: g[u]) {
    if (!done[v] and v != par and child[v] > sz) {
      return dfs_find_centroid(v,u,sz);
    }
  }
  return u;
}
void solve (int u) {/**problem specific things */}
void dfs_decompose(int u) {
  dfs_size(u, -1);
  int centroid=dfs_find_centroid(u,-1,child[u]/2);
  solve(centroid);
  done[centroid] = 1;
  for (int v : g[centroid]) {
    if (!done[v]) dfs_decompose(v);
  }
}
```

2.3 Hld - cpalgo

```
vector<int> parent, depth, heavy, head, pos;
int cur_pos;
int dfs(int v, vector<vector<int>> const& adj) {
    int size = 1;
    int max_c_size = 0;
    for (int c : adj[v]) {
        if (c != parent[v]) {
            parent[c] = v, depth[c] = depth[v] + 1;
            int c_size = dfs(c, adj);
            size += c_size;
            if (c_size > max_c_size)
                max_c_size = c_size, heavy[v] = c;}}
    return size;}
decompose(int v,int h,vector<vector<int>> const& adj){
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1)
        decompose(heavy[v], h, adj);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c, adj);}
}
void init(vector<vector<int>> const& adj) {
    int n = adj.size();
    parent = vector<int>(n);
    depth = vector<int>(n);
    heavy = vector<int>(n, -1);
    head = vector<int>(n);
    pos = vector<int>(n);
    cur_pos = 0;
    dfs(0, adj);
    decompose(0, 0, adj);
}
int query(int a, int b) {
  int res = 0;
  for (; head[a] != head[b]; b = parent[head[b]]) {
    if (depth[head[a]] > depth[head[b]])
      swap(a, b);
int cur_heavy_path_max =
segment_tree_query(pos[head[b]], pos[b]);
      res = max(res, cur_heavy_path_max);
  }
  if (depth[a] > depth[b])
      swap(a, b);
  int last_heavy_path_max =
  segment_tree_query(pos[a], pos[b]);
  res = max(res, last_heavy_path_max);
  return res;
}
```

2.4 Implicit Treap

```
#include<bits/stdc++.h>
#include<math.h>
#include<vector>
#include<stdlib.h>
using namespace std;
#define MAX 200005
#define MOD 998244353
#define NINF -1000000000000000000
template <class T>
class implicit_treap
{
    struct item
    {
        int prior, cnt;
        T value;
        bool rev;
        item *l,*r;
        item(T v)
        {
            value=v;
            rev=false;
            l=NULL;
            r=NULL;
            cnt=1;
            prior=rand();
        }
    } *root,*node;
    int cnt (item * it)
    {
        return it ? it->cnt : 0;
    }
    void upd_cnt (item * it)
    {
        if (it)
            it->cnt = cnt(it->l) + cnt(it->r) + 1;
    }
    void push (item * it)
    {
        if (it && it->rev)
        {
            it->rev = false;
            swap (it->l, it->r);
            if (it->l) it->l->rev ^= true;
            if (it->r) it->r->rev ^= true;
        }
    }
    void merge (item * & t, item * l, item * r)
    {
        push (l);
        push (r);
        if (!l || !r)
            t = l ? l : r;
        else if (l->prior > r->prior)
            merge (l->r, l->r, r), t = l;
        else
            merge (r->l, l, r->l), t = r;
        upd_cnt (t);
    }
    void split (item * t, item * & l, item * & r, int
    {
        if (!t)
            return void( l = r = 0 );
        push (t);
        int cur_key = add + cnt(t->l);
        if (key <= cur_key)
            split (t->l, l, t->l, key, add), r = t;
        else
            split (t->r, t->r, r, key, add + 1 + cnt(t-
l = t;
        upd_cnt (t);
    }
    void insert(item * &t,item * element,int key)
    {
        item *l,*r;
        split(t,l,r,key);
        merge(l,l,element);
        merge(t,l,r);
        l=NULL;
```

```
        r=NULL;
    }
    T elementAt(item * &t,int key)
    {
        push(t);
        T ans;
        if(cnt(t->l)==key) ans=t->value;
        else if(cnt(t->l)>key) ans=elementAt(t->l,key);
        else ans=elementAt(t->r,key-1-cnt(t->l));
        return ans;
    }
    void erase (item * & t, int key)
    {
        push(t);
        if(!t) return;
        if (key == cnt(t->l))
            merge (t, t->l, t->r);
        else if(key<cnt(t->l))
            erase(t->l,key);
        else
            erase(t->r,key-cnt(t->l)-1);
        upd_cnt(t);
    }
    void reverse (item * &t, int l, int r)
    {
        item *t1, *t2, *t3;
        split (t, t1, t2, l);
        split (t2, t2, t3, r-l+1);
        t2->rev ^= true;
        merge (t, t1, t2);
        merge (t, t, t3);
    }
    void cyclic_shift(item * &t,int L,int R)
    {
        if(L==R) return;
        item *l,*r,*m;
        split(t,t,l,L);
        split(l,l,m,R-L+1);
        split(l,l,r,R-L);
        merge(t,t,r);
        merge(t,t,l);
        merge(t,t,m);
        l=NULL;
        r=NULL;
        m=NULL;
    }
    void output (item * t,vector<T> &arr)
    {
        if (!t) return;
        push (t);
        output (t->l,arr);
        arr.push_back(t->value);
        output (t->r,arr);
    }
public:
    implicit_treap()
    {
        root=NULL;
    }
    void insert(T value,int position)
    {
        node=new item(value);
        insert(root,node,position);
    }
    void erase(int position)
    {
        erase(root,position);
    }
    void reverse(int l,int r)
    {
        reverse(root,l,r);
    }
    T elementAt(int position)
    {
        return elementAt(root,position);
    }
    void cyclic_shift(int L,int R)
    {
        cyclic_shift(root,L,R);
    }
    int size()
    {
        return cnt(root);
    }
    void output(vector<T> &arr)
    {
        output(root,arr);
    }
};
```

2.5 Mo Algorithm

```
#include<bits/stdc++.h>
using namespace std;
#define MOD 998244353
#define MAX 200005
#define MAX_BIT 50
#define PRECISION 0.00000000001
#define INF 2000000000
void remove(int idx); // TODO: remove value at idx from data structure
void add(int idx);   // TODO: add value at idx from data structure
int get_answer(); // TODO: extract the current answer of the data structure
int block_size;
struct Query {
    int l, r,k, idx;
    bool operator<(Query other) const
    {
        if(l/block_size!=other.l/block_size) return (l<other.l);
        return (l/block_size&1)? (r<other.r) : (r>other.r);
    }
};
vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}

int main()
{
    return 0;
}
```

2.6 Treap

```
#include<bits/stdc++.h>
#include<math.h>
#include<vector>
#include<stdlib.h>
using namespace std;
#define MAX 400005
#define MOD 998244353
#define INF 2000000000
template <class T>
class treap
{
    struct item
    {
        int prior, cnt;
        T key;
        item *l,*r;
        item(T v)
        {
            key=v;
            l=NULL;
            r=NULL;
            cnt=1;
            prior=rand();
        }
    }*root,*node;
    int cnt (item * it)
    {
        return it ? it->cnt : 0;
    }

    void upd_cnt (item * it)
    {
        if (it)
            it->cnt = cnt(it->l) + cnt(it->r) + 1;
    }

    void split (item * t, T key, item * & l, item * &
    {
        if (!t)
            l = r = NULL;
        else if (key < t->key)
            split (t->l, key, l, t->l), r = t;
        else
            split (t->r, key, t->r, r), l = t;
        upd_cnt(t);
```

```
}

void insert (item * & t, item * it)
{
    if (!t)
        t = it;
    else if (it->prior > t->prior)
        split (t, it->key, it->l, it->r), t = it;
    else
        insert (it->key < t->key ? t->l : t->r, it);
    upd_cnt(t);
}

void merge (item * & t, item * l, item * r)
{
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r), t = l;
    else
        merge (r->l, l, r->l), t = r;
    upd_cnt(t);
}

void erase (item * & t, T key)
{
    if (t->key == key)
        merge (t, t->l, t->r);
    else
        erase (key < t->key ? t->l : t->r, key);
    upd_cnt(t);
}

T elementAt(item * &t,int key)
{
    T ans;
    if(cnt(t->l)==key) ans=t->key;
    else if(cnt(t->l)>key) ans=elementAt(t->l,key);
    else ans=elementAt(t->r,key-1-cnt(t->l));
    upd_cnt(t);
    return ans;
}

item * unite (item * l, item * r)
{
    if (!l || !r) return l ? l : r;
    if (l->prior < r->prior) swap (l, r);
    item * lt, * rt;
    split (r, l->key, lt, rt);
    l->l = unite (l->l, lt);
    l->r = unite (l->r, rt);
    upd_cnt(l);
    upd_cnt(r);
    return l;
}
void heapify (item * t)
{
    if (!t) return;
    item * max = t;
    if (t->l != NULL && t->l->prior > max->prior)
        max = t->l;
    if (t->r != NULL && t->r->prior > max->prior)
        max = t->r;
    if (max != t)
    {
        swap (t->prior, max->prior);
        heapify (max);
    }
}

item * build (T * a, int n)
{
    if (n == 0) return NULL;
    int mid = n / 2;
    item * t = new item (a[mid], rand ());
    t->l = build (a, mid);
    t->r = build (a + mid + 1, n - mid - 1);
    heapify (t);
    return t;
}
void output (item * t,vector<T> &arr)
{
    if (!t) return;
    output (t->l,arr);
    arr.push_back(t->key);
    output (t->r,arr);
}
public:
    treap()
    {
        root=NULL;
    }
    treap(T *a,int n)
    {
        build(a,n);
    }
    void insert(T value)
    {
        node=new item(value);
        insert(root,node);
    }
    void erase(T value)
    {
        erase(root,value);
    }
    T elementAt(int position)
    {
        return elementAt(root,position);
    }
    int size()
    {
        return cnt(root);
    }
    void output(vector<T> &arr)
    {
        output(root,arr);
    }
    int range_query(T l,T r) //(l,r]
    {
        item *previous,*next,*current;
        split(root,l,previous,current);
        split(current,r,current,next);
        int ans=cnt(current);
        merge(root,previous,current);
        merge(root,root,next);
        previous=NULL;
        current=NULL;
        next=NULL;
        return ans;
    }
};
```

2.7 sparse table 2d

```
int st[K][K][N][N]; int lg[N];
void pre() {
  lg[1] = 0;
  for (int i=2; i<N; i++) lg[i] = lg[i/2]+1;
}
int query(int l1, int r1, int l2, int r2) {
  int xx = lg[l2-l1+1], yy = lg[r2-r1+1];
  return max(max(st[xx][yy][l1][r1],
          st[xx][yy][l2-(1<<xx)+1][r1]),
        max(st[xx][yy][l1][r2-(1<<yy)+1],
        st[xx][yy][l2-(1<<xx)+1][r2-(1<<yy)+1]));
}
void build() {
  for (int x=0; x<K; x++) {
    for (int y=0; y<K; y++) {
      for (int i=1; i<=n; i++) {
        for (int j=1; j<=m; j++) {
          if (i+(1<<x)-1>n || j+(1<<y)-1>m)
            continue;
          if (!x&&!y) st[0][0][i][j]=flag[i][j];
          else if (x>0) st[x][y][i][j] =
max(st[x-1][y][i][j],st[x-1][y][i+(1<<(x-1))][j]);
          else if (y>0) st[x][y][i][j] =
max(st[x][y-1][i][j],st[x][y-1][i][j+(1<<(y-1))]);
        }
      }
    }
  }
}
```

3 Flow

3.1 Dinic's Algorithm

```
#include<bits/stdc++.h>
#include<vector>
using namespace std;
#define MAX 100
#define HUGE_FLOW 1000000000
#define BEGIN 1
#define DEFAULT_LEVEL 0
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u)
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
```

```cpp
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
int main()
{
    return 0;
}
```

3.2 Edmond's Blossom Algorithm

```cpp
/***Copied from https://codeforces.com/blog/entry/49402***/
/*
GETS:
V->number of vertices
E->number of edges
pair of vertices as edges (vertices are 1..V)

GIVES:
output of edmonds() is the maximum matching
match[i] is matched pair of i (-1 if there isn't a matched pair)
*/

#include <bits/stdc++.h>
using namespace std;
const int M=500;
struct struct_edge
{
    int v;
    struct_edge* n;
};
typedef struct_edge* edge;
struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v)
{
    top->v=v,top->n=adj[u],adj[u]=top++;
    top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v)
{
    static bool inp[M];
    memset(inp,0,sizeof(inp));
    while(1)
    {
        inp[u=base[u]]=true;
        if (u==root) break;
        u=father[match[u]];
    }
    while(1)
    {
        if (inp[v=base[v]]) return v;
        else v=father[match[v]];
    }
}
void mark_blossom(int lca,int u)
{
    while (base[u]!=lca)
    {
        int v=match[u];
        inb[base[u]]=inb[base[v]]=true;
        u=father[v];
        if (base[u]!=lca) father[u]=v;
    }
}
void blossom_contraction(int s,int u,int v)
{
    int lca=LCA(s,u,v);
    memset(inb,0,sizeof(inb));
    mark_blossom(lca,u);
    mark_blossom(lca,v);
    if (base[u]!=lca)
        father[u]=v;
    if (base[v]!=lca)
        father[v]=u;
    for (int u=0; u<V; u++)
        if (inb[base[u]])
        {
            base[u]=lca;
            if (!inq[u])
                inq[q[++qt]=u]=true;
        }
}
int find_augmenting_path(int s)
{
    memset(inq,0,sizeof(inq));
    memset(father,-1,sizeof(father));
    for (int i=0; i<V; i++) base[i]=i;
    inq[q[qh=qt=0]=s]=true;
    while (qh<=qt)
    {
        int u=q[qh++];
        for (edge e=adj[u]; e; e=e->n)
        {
            int v=e->v;
            if (base[u]!=base[v]&&match[u]!=v)
                if ((v==s)||(match[v]!=-1 && father[matc
                    blossom_contraction(s,u,v);
                else if (father[v]==-1)
                {
                    father[v]=u;
                    if (match[v]==-1)
                        return v;
                    else if (!inq[match[v]])
                        inq[q[++qt]=match[v]]=true;
                }
        }
    }
    return -1;
}
int augment_path(int s,int t)
{
    int u=t,v,w;
    while (u!=-1)
    {
        v=father[u];
        w=match[v];
        match[v]=u;
        match[u]=v;
        u=w;
    }
    return t!=-1;
}
int edmonds()
{
    int matchc=0;
    memset(match,-1,sizeof(match));
    for (int u=0; u<V; u++)
        if (match[u]==-1)
            matchc+=augment_path(u,find_augmenting_path
    return matchc;
```

```
}
int main()
{
    FILE *in=stdin;
    int u,v;
    fscanf(in,"%d",&V);
    while(fscanf(in,"%d %d",&u,&v)!=EOF)
    {

        if (!ed[u-1][v-1])
        {
            add_edge(u-1,v-1);
            ed[u-1][v-1]=ed[v-1][u-1]=true;
        }
    }
    printf("%d\n",2*edmonds());
    for (int i=0; i<V; i++)
        if (i<match[i])
            printf("%d %d\n",i+1,match[i]+1);
    return 0;
}
```

3.3 Hungarian Algorithm

```
#include <bits/stdc++.h>
#include <vector>
#include<math.h>
#include<string.h>
using namespace std;
#define MAX 300005
#define MOD 1000000007
#define GMAX 19
#define INF 20000000000000000
#define EPS 0.000000001
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(NULL)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
class HungarianAlgorithm
{
    int N,inf,n,max_match;
    int *lx,*ly,*xy,*yx,*slack,*slackx,*prev;
    int **cost;
    bool *S,*T;
    void init_labels()
    {
        for(int x=0;x<n;x++) lx[x]=0;
        for(int y=0;y<n;y++) ly[y]=0;
        for (int x = 0; x < n; x++)
            for (int y = 0; y < n; y++)
                lx[x] = max(lx[x], cost[x][y]);
    }
    void update_labels()
    {
        int x, y, delta = inf; //init delta as infinity
//augmenting path exists!
        for (y = 0; y < n; y++) //calculate delta using slack
            if (!T[y])
                delta = min(delta, slack[y]);
        for (x = 0; x < n; x++) //update X labels
            if (S[x]) lx[x] -= delta;
        for (y = 0; y < n; y++) //update Y labels
            if (T[y]) ly[y] += delta;
        for (y = 0; y < n; y++) //update slack array
            if (!T[y])
                slack[y] -= delta;
    }
    void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in the alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
    {
        S[x] = true; //add x to S
        prev[x] = prevx; //we need this when augmenting
        for (int y = 0; y < n; y++) //update slacks, because we add new vertex to S
            if (lx[x] + ly[y] - cost[x][y] < slack[y])
            {
                slack[y] = lx[x] + ly[y] - cost[x][y];
                slackx[y] = x;
            }
    }
    void augment() //main function of the algorithm
    {
        if (max_match == n) return; //check wether matching is already perfect
        int x, y, root; //just counters and root vertex
        int q[N], wr = 0, rd = 0; //q - queue for bfs, wr,rd - write and read
//pos in queue
        //memset(S, false, sizeof(S)); //init set S
        for(int i=0;i<n;i++) S[i]=false;
        //memset(T, false, sizeof(T)); //init set T
        for(int i=0;i<n;i++) T[i]=false;
        //memset(prev, -1, sizeof(prev)); //init set prev - for the alternating tree
        for(int i=0;i<n;i++) prev[i]=-1;
        for (x = 0; x < n; x++) //finding root of the tree
        {
            if (xy[x] == -1)
            {
                q[wr++] = root = x;
                prev[x] = -2;
                S[x] = true;
                break;
            }
        }
        for (y = 0; y < n; y++) //initializing slack array
        {
            slack[y] = lx[root] + ly[y] - cost[root][y];
            slackx[y] = root;
        }
        while (true) //main cycle
        {
            while (rd < wr) //building tree with bfs cycle
            {
                x = q[rd++]; //current vertex from X part
                for (y = 0; y < n; y++) //iterate through all edges in equality graph
                {
                    if (cost[x][y] == lx[x] + ly[y] && !T[y])
                    {
                        if (yx[y] == -1) break; //an exposed vertex in Y found, so
                        T[y] = true; //else just add y to T,
                        q[wr++] = yx[y]; //add vertex yx[y], which is matched
//with y, to the queue
                        add_to_tree(yx[y], x); //add edges (x,y), (y,yx[y]) to the tree
                    }
                }
                if (y < n) break; //augmenting path found!
            }
            if (y < n) break; //augmenting path found!
            update_labels(); //augmenting path not foun
            wr = rd = 0;
            for (y = 0; y < n; y++)
            {
                //in this cycle we add edges that were a
//result of improving the labeling, we add edge (slack
//and only if !T[y] && slack[y] == 0, also with this e
//(y, yx[y]) or augment the matching, if y was exposed
                if (!T[y] && slack[y] == 0)
                {
                    if (yx[y] == -1) //exposed vertex in
                    {
                        x = slackx[y];
                        break;
                    }
                    else
                    {
                        T[y] = true; //else just add y t
                        if (!S[yx[y]])
                        {
                            q[wr++] = yx[y]; //add vertex
//y, to the queue
                            add_to_tree(yx[y], slackx[y])
//yx[y]) to the tree
                        }
                    }
                }
            }
            if (y < n) break; //augmenting path found!
        }
        if (y < n) //we found augmenting path!
        {
            max_match++; //increment matching
//in this cycle we inverse edges along augmenting path
            for (int cx = x, cy = y, ty; cx != -2; cx =
            {
                ty = xy[cx];
                yx[cy] = cx;
                xy[cx] = cy;
            }
            augment(); //recall function, go to step 1
        }
    }//end of augment() function
public:
    HungarianAlgorithm(int vv,int inf=1000000000)
    {
        N=vv;
        n=N;
        max_match=0;
        this->inf=inf;
        lx=new int[N];
        ly=new int[N];//labels of X and Y parts
        xy=new int[N];//xy[x] - vertex that is matched
        yx=new int[N];//yx[y] - vertex that is matched
        slack=new int[N];//as in the algorithm descript
        slackx=new int[N];//slackx[y] such a vertex, th
        prev=new int[N];//array for memorizing alternat
        S=new bool[N];
        T=new bool[N];//sets S and T in algorithm
        cost=new int*[N];//cost matrix
```

```
        for(int i=0; i<N; i++)
        {
            cost[i]=new int[N];
        }
    }
    ~HungarianAlgorithm()
    {
        delete []lx;
        delete []ly;
        delete []xy;
        delete []yx;
        delete []slack;
        delete []slackx;
        delete []prev;
        delete []S;
        delete []T;
        int i;
        for(i=0; i<N; i++)
        {
            delete [](cost[i]);
        }
        delete []cost;
    }
    void setCost(int i,int j,int c)
    {
        cost[i][j]=c;
    }
    int* matching(bool first=true)
    {
        int *ans;
        ans=new int[N];
        for(int i=0;i<N;i++)
        {
            if(first) ans[i]=xy[i];
            else ans[i]=yx[i];
        }
        return ans;
    }
    int hungarian()
    {
        int ret = 0; //weight of the optimal matching
        max_match = 0; //number of vertices in current matching
        for(int x=0;x<n;x++) xy[x]=-1;
        for(int y=0;y<n;y++) yx[y]=-1;
        init_labels(); //step 0
        augment(); //steps 1-3
        for (int x = 0; x < n; x++) //forming answer there
            ret += cost[x][xy[x]];
        return ret;
    }
};
int main()
{
    int t,T=1;
    scanf("%d",&T);
    for(t=0;t<T;t++)
    {
        int n,i,j;
        scanf("%d",&n);
        HungarianAlgorithm h(n);
        int own[n],opposite[n];
        for(i=0;i<n;i++)
        {
            scanf("%d",&own[i]);
        }
        for(j=0;j<n;j++)
        {
            scanf("%d",&opposite[j]);
        }
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                int v;
                if(own[i]==opposite[j]) v=1;
                else if(own[i]>opposite[j]) v=2;
                else v=0;
                h.setCost(i,j,v);
            }
        }
        int ans=h.hungarian();
        printf("Case %d: %d\n",t+1,ans);
    }
    return 0;
}
```

3.4 Maximum Bipartite Matching

```
/** Source: https://iq.opengenus.org/hopcroft-karp-algorithm/ **/
#include <bits/stdc++.h>
#include <vector>
#include<math.h>
#include<string.h>
using namespace std;
#define MAX 300005
#define BEGIN 1
#define MOD 1000000007
#define INF INT_MAX
#define EPS 0.0000000001
#define CHAINS 18
#define NIL 0
#define NOT_VISITED 0
#define VISITING 1
#define VISITED 2
#define FASTIO ios_base::sync_with_stdio(false);cin.tie(NULL)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
// A class to represent Bipartite graph for
// Hopcroft Karp implementation
class BGraph
{
    // m and n are number of vertices on left
    // and right sides of Bipartite Graph
    int m, n;

    // adj[u] stores adjacents of left side
    // vertex 'u'. The value of u ranges from 1 to m.
    // 0 is used for dummy vertex
    std::list<int> *adj;

    // pointers for hopcroftKarp()
    int *pair_u, *pair_v, *dist;

public:
    BGraph(int m, int n);   // Constructor
    void addEdge(int u, int v); // To add edge

    // Returns true if there is an augmenting path
    bool bfs();

    // Adds augmenting path if there is one beginning
    // with u
    bool dfs(int u);

    // Returns size of maximum matching
    int hopcroftKarpAlgorithm();
};

// Returns size of maximum matching
int BGraph::hopcroftKarpAlgorithm()
{
    // pair_u[u] stores pair of u in matching on left
    // If u doesn't have any pair, then pair_u[u] is N
    pair_u = new int[m + 1];

    // pair_v[v] stores pair of v in matching on right
    // If v doesn't have any pair, then pair_u[v] is N
    pair_v = new int[n + 1];

    // dist[u] stores distance of left side vertices
    dist = new int[m + 1];

    // Initialize NIL as pair of all vertices
    for (int u = 0; u <= m; u++)
        pair_u[u] = NIL;
    for (int v = 0; v <= n; v++)
        pair_v[v] = NIL;

    // Initialize result
    int result = 0;

    // Keep updating the result while there is an
    // augmenting path possible.
    while (bfs())
    {
        // Find a free vertex to check for a matching
        for (int u = 1; u <= m; u++)

            // If current vertex is free and there is
            // an augmenting path from current vertex
            // then increment the result
            if (pair_u[u] == NIL && dfs(u))
                result++;
    }
    return result;
}

// Returns true if there is an augmenting path availab
bool BGraph::bfs()
{
    std::queue<int> q; //an integer queue for bfs

    // First layer of vertices (set distance as 0)
    for (int u = 1; u <= m; u++)
    {
        // If this is a free vertex, add it to queue
        if (pair_u[u] == NIL)
        {
            // u is not matched so distance is 0
            dist[u] = 0;
            q.push(u);
        }

        // Else set distance as infinite so that this v
```

```
        else
            dist[u] = INF;
    }

    // Initialize distance to NIL as infinite
    dist[NIL] = INF;

    // q is going to contain vertices of left side only
    while (!q.empty())
    {
        // dequeue a vertex
        int u = q.front();
        q.pop();

        // If this node is not NIL and can provide a shorter path to NIL then
        if (dist[u] < dist[NIL])
        {
            // Get all the adjacent vertices of the dequeued vertex u
            std::list<int>::iterator it;
            for (it = adj[u].begin(); it != adj[u].end(); ++it)
            {
                int v = *it;

                // If pair of v is not considered so far
                // i.e. (v, pair_v[v]) is not yet explored edge.
                if (dist[pair_v[v]] == INF)
                {
                    // Consider the pair and push it to queue
                    dist[pair_v[v]] = dist[u] + 1;
                    q.push(pair_v[v]);
                }
            }
        }
    }

    // If we could come back to NIL using alternating path of distinct
    // vertices then there is an augmenting path available
    return (dist[NIL] != INF);
}

// Returns true if there is an augmenting path beginning with free vertex u
bool BGraph::dfs(int u)
{
    if (u != NIL)
    {
        std::list<int>::iterator it;
        for (it = adj[u].begin(); it != adj[u].end(); ++it)
        {
            // Adjacent vertex of u
            int v = *it;

            // Follow the distances set by BFS search
            if (dist[pair_v[v]] == dist[u] + 1)
            {
                // If dfs for pair of v also returnn true then
                if (dfs(pair_v[v]) == true)
                {   // new matching possible, store the matching
                    pair_v[v] = u;
                    pair_u[u] = v;
                    return true;
                }
            }
        }

        // If there is no augmenting path beginning with u then.
            dist[u] = INF;
            return false;
    }
    return true;
}

// Constructor for initialization
BGraph::BGraph(int m, int n)
{
    this->m = m;
    this->n = n;
    adj = new std::list<int>[m + 1];
}

// function to add edge from u to v
void BGraph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add v to us list.
}
```

3.5 Minimum Cost Maximum Flow

```
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

4 Geo

4.1 Convex Hull

```
struct pt {double x, y;};
bool cmp(pt a, pt b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)<0;
}
bool ccw(pt a, pt b, pt c) {
  return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)>0;
}
vector<pt> a;
vector<pair<double,pair<double,double> > > pp;
void convex_hull(vector<pt>& a) {
    if (a.size() == 1)
        return;
    sort(a.begin(), a.end(), &cmp);
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2)) {
            while (up.size() >= 2 &&
        !cw(up[up.size()-2], up[up.size()-1], a[i]))
                up.pop_back();
            up.push_back(a[i]);
```

```
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2)) {
            while(down.size() >= 2 &&!
ccw(down[down.size()-2], down[down.size()-1], a[i]))
                down.pop_back();
            down.push_back(a[i]);
        }
    }
    a.clear();
for(int i=0;i<(int)up.size();i++)a.push_back(up[i]);
for(int i=down.size()-2;i>0;i--)a.push_back(down[i]);
}
```

4.2 Half Plane Intersection

```
#define MAX 200005
#define MOD 1009
#define SMOD 998244353
#define ROOT 318
#define GMAX 19
#define INF 1000000000000000000
#define EPS 0.000000001
#define NIL 0
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
class HalfPlaneIntersection{
    static double eps, inf;
public:
    struct Point{
        double x, y;
explicit Point(double x=0,double y=0):x(x), y(y) {}
friend Point operator+(const Point&p,const Point&q){
    return Point(p.x + q.x, p.y + q.y);
}
friend Point operator-(const Point&p,const Point&q){
    return Point(p.x - q.x, p.y - q.y);
}
friend Point operator*(const Point&p,double&k){
    return Point(p.x * k, p.y * k);
}
friend double cross(const Point& p, const Point& q){
    return p.x * q.y - p.y * q.x;
}
};
// Basic half-plane struct.
    struct Halfplane
    {// 'p' is a passing point of the line and
        //'pq' is the direction vector of the line.
        Point p, pq;  double angle;
        Halfplane() {}
        Halfplane(const Point&a,const Point&b):p(a)
        ,pq(b-a){
            angle = atan2l(pq.y, pq.x);
        }
// Check if point 'r' is outside this half-plane.
// Every half-plane allows the region to the LEFT
// of its line.
        bool out(const Point& r){
            return cross(pq, r - p) < -eps;
        }
//Comparator for sorting. If the angle of both half-
//planes is equal, the leftmost one should go first.
        bool operator < (const Halfplane& e) const{
            if (fabsl(angle - e.angle) < eps)
                return cross(pq, e.p - p) < 0;
            return angle < e.angle;
        }// We use equal comparator for std::unique
        //to easily remove parallel half-planes.
        bool operator == (const Halfplane& e) const{
            return fabsl(angle - e.angle) < eps;
        }// Intersection point of the lines of two
//half-planes. It is assumed they're never parallel.
        friend Point inter(const Halfplane& s,
                           const Halfplane& t){
double alpha=cross((t.p-s.p),t.pq)/cross(s.pq,t.pq);
return s.p + (s.pq * alpha);
        }
    };
vector<Point>hp_intersect(vector<Halfplane>&H)
    {
        Point box[4] = // Bounding box in CCW order
        {
            Point(inf, inf),
            Point(-inf, inf),
            Point(-inf, -inf),
            Point(inf, -inf)
        };
for(int i=0;i<4;i++){//Add bounding box half-planes.
            Halfplane aux(box[i], box[(i+1) % 4]);
            H.push_back(aux);
        }
        // Sort and remove duplicates
        sort(H.begin(), H.end());
        H.erase(unique(H.begin(),H.end()), H.end());
        deque<Halfplane> dq;
        int len = 0;
        for(int i = 0; i < int(H.size()); i++){
// Remove from the back of the deque while last
// half-plane is redundant
while(len>1&&H[i].out(inter(dq[len-1], dq[len-2]))){
                dq.pop_back();
                --len;
            }// Remove from the front of the deque
        //while first half-plane is redundant
    while (len > 1 && H[i].out(inter(dq[0],dq[1]))){
                dq.pop_front();
                --len;
            } // Add new half-plane
            dq.push_back(H[i]); ++len;
        } // Final cleanup: Check half-planes at the
        //front against the back and vice-versa
while(len>2&&dq[0].out(inter(dq[len-1],dq[len-2]))){
            dq.pop_back(); --len;
        }
while(len > 2 && dq[len-1].out(inter(dq[0],dq[1]))){
            dq.pop_front(); --len;
        }
        // Report empty intersection if necessary
        if (len < 3) return vector<Point>();
// Reconstruct the convex polygon from
//the remaining half-planes.
        vector<Point> ret(len);
        for(int i = 0; i+1 < len; i++){
            ret[i] = inter(dq[i], dq[i+1]);
        }
        ret.back() = inter(dq[len-1], dq[0]);
        return ret;
    }
};
double HalfPlaneIntersection::eps=1e-9;
double HalfPlaneIntersection::inf=1e9;
    vector<HalfPlaneIntersection::Halfplane> V;
    vector<HalfPlaneIntersection::Point> P;
    for(i=0; i<n; i++){
        int c;
        scanf("%d",&c);
        HalfPlaneIntersection::Halfplane h;
        HalfPlaneIntersection::Point p;
        for(j=0; j<c; j++){
            scanf("%lf %lf",&p.x,&p.y);
            P.push_back(p);
        }
        for(j=0; j<c; j++){
h=HalfPlaneIntersection::Halfplane(P[j],P[(j+1)%c]);
            V.push_back(h);
        }
        P.clear();
    }
    P=HalfPlaneIntersection::hp_intersect(V);
    double ans=0;
    n=P.size();
    for(i=0; i<n; i++){
ans=ans+P[i].x*P[(i+1)%n].y-P[i].y*P[(i+1)%n].x;
    }
    ans=ans/2;
```

4.3 Line Segment Intersection

```
struct pt {
    double x, y;
    bool operator<(const pt& p) const
    {
  return x<p.x-EPS||(abs(x-p.x)<EPS && y < p.y- EPS);
    }
};
struct line {
    double a, b, c;
    line() {}
    line(pt p, pt q){
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }
    void norm(){
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS) a /= z, b /= z, c /= z;
    }
    double dist(pt p){ return a * p.x + b * p.y + c;}
};
double det(double a, double b, double c, double d){
    return a * d - b * c;
}
inline bool betw(double l, double r, double x){
    return min(l, r) <= x + EPS && x <= max(l,r)+EPS;
}
bool intersect_1d(double a, double b, dbl c, dbl d)
```

```
{
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}bool
intersect(pt a,pt b,pt c,pt d,pt& left,pt& right){
    if (!intersect_1d(a.x, b.x, c.x, d.x) ||
      !intersect_1d(a.y, b.y, c.y, d.y))return false;
    line m(a, b);  line n(c, d);
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
    if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
            return false;
        if (b < a) swap(a, b);
        if (d < c) swap(c, d);
        left = max(a, c); right = min(b, d);
        return true;
    } else {
left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
return betw(a.x,b.x,left.x)&&betw(a.y,b.y,left.y) &&
    betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y);
    }
}
```

4.4 Minimum Perimeter Triangle

```
#define MAX 300005
#define MOD 1000000007
#define SMOD 998244353
#define INF 6000000000000000000
#define EPS 0.0000000001
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
struct pt{
    double x, y;
    int id;
};
struct cmp_x{
    bool operator()(const pt & a, const pt &b)const{
        return a.x < b.x || (a.x == b.x && a.y<b.y);
    }
};
struct cmp_y{
    bool operator()(const pt& a, const pt & b)const{
        return a.y < b.y;
    }
};
int n; vector<pt> a; double mindist;
pair<int,pair<int, int> > best_pair;
void upd_ans(const pt & a, const pt &b,const pt&c){
    double distC = sqrt((a.x-b.x)*(a.x-b.x)
            +(a.y-b.y)*(a.y-b.y));
    double distA = sqrt((c.x - b.x)*(c.x - b.x)
            + (c.y - b.y)*(c.y - b.y));
    double distB = sqrt((a.x - c.x)*(a.x - c.x)
            + (a.y - c.y)*(a.y - c.y));
    if (distA + distB + distC < mindist){
    mindist = distA + distB + distC;
    best_pair=make_pair(a.id,make_pair(b.id,c.id));
    }
}
vector<pt> t;
void rec(int l, int r){
    if (r - l <= 3 &&r - l >=2){
        for (int i = l; i < r; ++i){
            for (int j = i + 1; j < r; ++j){
                for(int k=j+1;k<r;k++){
                    upd_ans(a[i],a[j],a[k]);
                }
            }
        }
        sort(a.begin() + l, a.begin() + r, cmp_y());
        return;
    }
    int m = (l + r) >> 1; int midx = a[m].x;
    rec(l, m); rec(m, r);
    merge(a.begin() + l, a.begin() + m, a.begin()
    + m, a.begin() + r, t.begin(), cmp_y());
    copy(t.begin(), t.begin() + r - l, a.begin()+l);
    int tsz = 0;
    for (int i = l; i < r; ++i){
        if (abs(a[i].x - midx) < mindist/2){
            for (int j = tsz - 1; j >= 0
            && a[i].y - t[j].y < mindist/2; --j){
                if(i+1<r) upd_ans(a[i],a[i+1],t[j]);
                if(j>0) upd_ans(a[i], t[j-1], t[j]);
            }
            t[tsz++] = a[i];
        }
    }
```

4.5 Minkowski

```
#define MAX 300005
#define BEGIN 1
#define CHAINS 18
#define NOT_VISITED 0
#define VISITING 1
#define VISITED 2
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
struct pt{
    long long x, y;
    pt() {}
    pt(long long _x, long long _y):x(_x), y(_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);
    }
    long long cross(const pt & p) const{
        return x * p.y - y * p.x;
    }
    long long dot(const pt & p) const{
        return x * p.x + y * p.y;
    }
    long long cross(const pt & a, const pt&b)const{
        return (a - *this).cross(b - *this);
    }
    long long dot(const pt & a, const pt & b) const{
        return (a - *this).dot(b - *this);
    }
    long long sqrLen() const{
        return this->dot(*this);
    }
};
class pointLocationInPolygon{
    bool lexComp(const pt & l, const pt & r){
        return l.x < r.x || (l.x == r.x && l.y<r.y);
    }
    int sgn(long long val){
        return val > 0 ? 1 : (val == 0 ? 0 : -1);
    }
    vector<pt> seq; int n; pt translate;
    bool pointInTriangle(pt a, pt b, pt c,pt point){
        long long s1 = abs(a.cross(b, c));
        long long s2 = abs(point.cross(a, b)) +
    abs(point.cross(b, c)) + abs(point.cross(c, a));
        return s1 == s2;
    }
public:
    pointLocationInPolygon(){}
    pointLocationInPolygon(vector<pt> & points){
        prepare(points);
    }
    void prepare(vector<pt> & points){
        seq.clear();
        n = points.size();
        int pos = 0;
        for(int i = 1; i < n; i++){
            if(lexComp(points[i], points[pos]))
                pos = i;
        }
        translate.x=points[pos].x;
        translate.y=points[pos].y;
        rotate(points.begin(), points.begin() +
         pos, points.end());
        n--;
        seq.resize(n);
        for(int i = 0; i < n; i++)
            seq[i] = points[i + 1] - points[0];
    }
    bool pointInConvexPolygon(pt point){
        point.x-=translate.x;
        point.y-=translate.y;
        if(seq[0].cross(point) != 0 && sgn(seq[0].
        cross(point))!=sgn(seq[0].cross(seq[n-1])))
            return false;
        if(seq[n-1].cross(point)!=0&&sgn(seq[n-1]
        .cross(point))!=sgn(seq[n-1].cross(seq[0])))
            return false;
        if(seq[0].cross(point) == 0)
            return seq[0].sqrLen()>= point.sqrLen();
        int l = 0, r = n - 1;
        while(r - l > 1){
            int mid = (l + r)/2; int pos = mid;
            if(seq[pos].cross(point) >= 0)l = mid;
            else r = mid;
        }
        int pos = l;
        return pointInTriangle(seq[pos],
        seq[pos+1],pt(0,0),point);
    }
```

```
    ~pointLocationInPolygon(){
        seq.clear();
    }
};
class Minkowski{
    static void reorder_polygon(vector<pt> & P){
        size_t pos = 0;
        for(size_t i = 1; i < P.size(); i++){
            if(P[i].y < P[pos].y ||
        (P[i].y == P[pos].y && P[i].x < P[pos].x))
                pos = i;
        }
        rotate(P.begin(), P.begin() + pos, P.end());
    }
public:
    vector<pt> minkowski(vector<pt> P,vector<pt>Q){
        // the first vertex must be the lowest
        reorder_polygon(P);
        reorder_polygon(Q);
        // we must ensure cyclic indexing
        P.push_back(P[0]);
        P.push_back(P[1]);
        Q.push_back(Q[0]);
        Q.push_back(Q[1]);
        // main part
        vector<pt> result;
        size_t i = 0, j = 0;
        while(i < P.size() - 2 || j < Q.size() - 2){
            result.push_back(P[i] + Q[j]);
auto cross = (P[i + 1] - P[i]).cross(Q[j+1]-Q[j]);
            if(cross >= 0) ++i;
            if(cross <= 0) ++j;
        }
        return result;
    }
};
```

4.6 Pair of Intersecting Segments

```
#define MAX 100009
#define MAX_NODES 100005
struct pt {
    double x, y;
};
struct seg {
    pt p, q; int id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y+(q.y-p.y)*(x-p.x)/(q.x - p.x);
    }
};
bool intersect1d(double l1, dbl r1, dbl l2, dbl r2){
    if (l1 > r1) swap(l1, r1);
    if (l2 > r2) swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
  double s =(b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b){
    return intersect1d(a.p.x,a.q.x,b.p.x,b.q.x) &&
            intersect1d(a.p.y,a.q.y,b.p.y,b.q.y) &&
            vec(a.p,a.q,b.p)*vec(a.p,a.q,b.q) <= 0 &&
            vec(b.p, b.q, a.p)*vec(b.p,b.q,a.q) <= 0;
}
bool operator<(const seg& a, const seg& b){
    double x=max(min(a.p.x,a.q.x),min(b.p.x,b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x; int tp, id;
    event() {}
event(double x, int tp, int id):x(x),tp(tp),id(id){}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS) return x < e.x;
        return tp > e.tp;
    }
};
set<seg> s; vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size(); vector<event> e;
    for (int i = 0; i < n; ++i) {
    e.push_back(event(min(a[i].p.x,a[i].q.x),+1,i));
    e.push_back(event(max(a[i].p.x,a[i].q.x),-1,i));
    }
    sort(e.begin(), e.end()); s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt=
            s.lower_bound(a[id]),prv=prev(nxt);
            if(nxt!=s.end()&&intersect(*nxt,a[id]))
                return make_pair(nxt->id, id);
            if(prv!=s.end()&&intersect(*prv,a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id])
            , prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() &&
            intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);
}
```

4.7 Vertical Decomposition

```
#define MAX 300005
#define MOD 1000000007
#define GMAX 19
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
typedef double dbl;
const dbl eps = 1e-9;
inline bool eq(dbl x, dbl y){
    return fabs(x - y) < eps;
}
inline bool lt(dbl x, dbl y){
    return x < y - eps;
}
inline bool gt(dbl x, dbl y){
    return x > y + eps;
}
inline bool le(dbl x, dbl y){
    return x < y + eps;
}
inline bool ge(dbl x, dbl y){
    return x > y - eps;
}
struct pt{
    dbl x, y;
    inline pt operator - (const pt & p)const{
        return pt{x - p.x, y - p.y};
    }
    inline pt operator + (const pt & p)const{
        return pt{x + p.x, y + p.y};
    }
    inline pt operator * (dbl a)const{
        return pt{x * a, y * a};
    }
    inline dbl cross(const pt & p)const{
        return x * p.y - y * p.x;
    }
    inline dbl dot(const pt & p)const{
        return x * p.x + y * p.y;
    }
    inline bool operator == (const pt & p)const{
        return eq(x, p.x) && eq(y, p.y);
    }
};
struct Line{
    pt p[2];
    Line(){} Line(pt a, pt b):p{a, b}{}
    pt vec()const{
        return p[1] - p[0];
    }
    pt& operator [](size_t i){
        return p[i];
    }
};
inline bool lexComp(const pt & l, const pt & r){
    if(fabs(l.x - r.x) > eps){
        return l.x < r.x;
    }
    else return l.y < r.y;
}
vector<pt> interSegSeg(Line l1, Line l2){
    if(eq(l1.vec().cross(l2.vec()), 0)){
        if(!eq(l1.vec().cross(l2[0] - l1[0]), 0))
            return {};
        if(!lexComp(l1[0], l1[1]))
            swap(l1[0], l1[1]);
        if(!lexComp(l2[0], l2[1]))
            swap(l2[0], l2[1]);
        pt l = lexComp(l1[0], l2[0]) ? l2[0] :l1[0];
```

```
        pt r = lexComp(l1[1], l2[1]) ? l1[1] :l2[1];
        if(l == r)
            return {l};
        else
  return lexComp(l,r)?vector<pt>{l,r}:vector<pt>();
    }
    else{
        dbl s = (l2[0] - l1[0]).cross(l2.vec()) /
         l1.vec().cross(l2.vec());
        pt inter = l1[0] + l1.vec() * s;
        if(ge(s, 0) && le(s, 1) && le((l2[0] -
        inter).dot(l2[1] - inter), 0))
            return {inter};
        else
            return {};
    }
}
char get_segtype(Line segment,pt other_point){
    if(eq(segment[0].x, segment[1].x))
        return 0;
    if(!lexComp(segment[0], segment[1]))
        swap(segment[0], segment[1]);
    return (segment[1]-segment[0]).cross(other_point
     - segment[0]) > 0 ? 1 : -1;
}
dbl union_area(vector<tuple<pt, pt,pt> > triangles){
    vector<Line> segments(3 * triangles.size());
    vector<char> segtype(segments.size());
    for(size_t i = 0; i < triangles.size(); i++){
        pt a, b, c; tie(a, b, c) = triangles[i];
  segments[3*i]=lexComp(a,b)?Line(a,b):Line(b,a);
  segtype[3 * i] = get_segtype(segments[3 * i], c);
  segments[3*i+1]=lexComp(b,c)?Line(b,c):Line(c,b);
  segtype[3*i+1]=get_segtype(segments[3*i+1],a);
  segments[3*i+2]=lexComp(c,a)?Line(c,a):Line(a,c);
  segtype[3*i+2]=get_segtype(segments[3*i+2],b);
    }
  vector<dbl>k(segments.size()),b(segments.size());
    for(size_t i = 0; i < segments.size(); i++){
        if(segtype[i]){
            k[i]=(segments[i][1].y-segments[i][0].y)
              /(segments[i][1].x-segments[i][0].x);
        b[i]=segments[i][0].y-k[i]*segments[i][0].x;
        }
    }
    dbl ans = 0;
    for(size_t i = 0; i < segments.size(); i++){
        if(!segtype[i]) continue;
        dbl l = segments[i][0].x,r=segments[i][1].x;
        vector<pair<dbl, int> > evts;
        for(size_t j = 0; j < segments.size(); j++){
            if(!segtype[j] || i == j)
                continue;
  dbl l1 = segments[j][0].x, r1 = segments[j][1].x;
            if(ge(l1, r) || ge(l, r1))
                continue;
  dbl common_l = max(l, l1), common_r = min(r, r1);
  auto pts = interSegSeg(segments[i], segments[j]);
            if(pts.empty()){
                dbl yl1 = k[j] * common_l + b[j];
                dbl yl = k[i] * common_l + b[i];
              if(lt(yl1, yl) == (segtype[i] == 1)){
                int evt_type = -segtype[i] * segtype[j];
                evts.emplace_back(common_l, evt_type);
                evts.emplace_back(common_r, -evt_type);
                  }
                }
                else if(pts.size() == 1u){
 dbl yl=k[i]*common_l+b[i],yl1=k[j]*common_l+b[j];
 int evt_type = -segtype[i] * segtype[j];
 if(lt(yl1, yl) == (segtype[i] == 1)){
     evts.emplace_back(common_l, evt_type);
     evts.emplace_back(pts[0].x, -evt_type);
 }
 yl =k[i]*common_r+b[i],yl1=k[j]*common_r+b[j];
 if(lt(yl1, yl) == (segtype[i] == 1)){
     evts.emplace_back(pts[0].x, evt_type);
     evts.emplace_back(common_r, -evt_type);
 }
                }
                else{
                    if(segtype[j] != segtype[i] || j>i){
                        evts.emplace_back(common_l, -2);
                        evts.emplace_back(common_r, 2);
                    }
                }
            }
            evts.emplace_back(l, 0);
            sort(evts.begin(), evts.end());
            size_t j = 0; int balance = 0;
            while(j < evts.size()){
                size_t ptr = j;
                while(ptr < evts.size() &&
                eq(evts[j].first, evts[ptr].first)){
                    balance += evts[ptr].second;
                    ++ptr;
                }
                if(!balance && !eq(evts[j].first, r)){
dbl next_x = ptr == evts.size() ?r:evts[ptr].first;
                    ans -= segtype[i] * (k[i] * (next_x
+evts[j].first)+2*b[i])*(next_x-evts[j].first);
                }
                j = ptr;
            }
    }
    return ans/2;
}
```

4.8 common tangent

```
struct pt {
    double x, y;
    pt operator- (pt p) {
        pt res = { x-p.x, y-p.y };
        return res;
    }
};
struct circle : pt {
    double r;
};
struct line {
    double a, b, c;
};
const double EPS = 1E-9;
double sqr (double a) {
    return a * a;
}
void tangents (pt c, double r1, double r2,
          vector<line> & ans) {
    double r = r2 - r1;
    double z = sqr(c.x) + sqr(c.y);
    double d = z - sqr(r);
    if (d < -EPS) return;
    d = sqrt (abs (d));
    line l; l.a = (c.x * r + c.y * d) / z;
    l.b = (c.y * r - c.x * d) / z; l.c = r1;
    ans.push_back (l);
}
vector<line> tangents (circle a, circle b) {
    vector<line> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b-a, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].c -= ans[i].a * a.x + ans[i].b * a.y;
    return ans;
}
```

5 Graph

5.1 Articulation Vertex

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

5.2 Strongly Connected Components

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...

    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }

    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);

            // ... processing next component ...

            component.clear();
        }
}
```

6 Math

6.1 Discrete Root

```
#define MAX 100000
int prime[MAX+1],Phi[MAX+1];
void sieve(){
    int i,j;
    for(i=2; i*i<=MAX; i++){
        if(prime[i]) continue;
        for(j=i; j*i<=MAX; j++){
            if(prime[i*j]==0) prime[i*j]=i;
        }
    }
}
void PhiWithSieve(){
    int i;
    for(i=2; i<=MAX; i++){
        if(prime[i]==0){
            Phi[i]=i-1;
        }
        else if((i/prime[i])%prime[i]==0){
            Phi[i]=Phi[i/prime[i]]*prime[i];
        }
        else{
            Phi[i]=Phi[i/prime[i]]*(prime[i]-1);
        }
    }
}
int gcd(int a,int b){
    if(b==0) return a;
    else return gcd(b,a%b);
}
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p), --b;
        else
            a = int (a * 1ll * a % p), b >>= 1;
    return res;
}
int PrimitiveRoot(int p){
    vector<int>fact;
    int phi=Phi[p];
    int n=phi;
    while(n>1){
        if(prime[n]==0){
            fact.push_back(n);
            n=1;
        }
        else{
            int f=prime[n];
            while(n%f==0){
                n=n/f;
            }
            fact.push_back(f);
        }
    }
    int res;
    for(res=p-1; res>1; --res){
        for(n=0; n<fact.size(); n++){
            if(powmod(res,phi/fact[n],p)==1){
                break;
            }
        }
        if(n>=fact.size()) return res;
    }
    return -1;
}
int DiscreteLog(int a, int b, int m) {
    a %= m, b %= m;
    int n = sqrt(m) + 1;
    map<int, int> vals;
    for (int p = 1; p <= n; ++p)
        vals[powmod(a,(int) (1ll* p * n) %m , m)]=p;
    for (int q = 0; q <= n; ++q) {
        int cur = (powmod(a, q, m) * 1ll * b) % m;
        if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
            return ans;
        }
    }
    return -1;
}
vector<int> DiscreteRoot(int n,int a,int k){
    int g = PrimitiveRoot(n);
    vector<int> ans;
    int any_ans = DiscreteLog(powmod(g,k,n),a,n);
    if (any_ans == -1){
        return ans;
    }
    int delta = (n-1) / gcd(k, n-1);
    for(intcur=any_ans%delta;cur<n-1;cur+=delta)
        ans.push_back(powmod(g, cur, n));
    sort(ans.begin(), ans.end());
    return ans;
}
```

6.2 Fast Fourier Transform

```
#define MOD 1000000007
#define MAX 200005
#define PMAX 55
#define PRECISION 0.000001
#define INF 2000000000
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd>& a, bool invert){
    int n = a.size();
    for(int i = 1, j = 0; i < n; i++){
        int bit = n>>1;
        for(; j&bit; bit>>=1){
            j^=bit;
        }
        j ^= bit;
        if(i < j) swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1){
        double ang = 2*PI/len*(invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += len){
            cd w(1);
            for(int j = 0; j < len/2; j++){
                cd u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if(invert){
        for(cd &x: a) x /= n;
    }
}
vector<int> multiply(vector<int>&a,vector<int>&b){
    vector<cd> fa(a.begin(), a.end());
    vector<cd> fb(b.begin(), b.end()); int n = 1;
    while(n < a.size()+b.size()) n <<= 1;
    fa.resize(n); fb.resize(n);
```

```
    fft(fa, false);
    fft(fb, false);
    for(int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);
    vector<int> result(n);
    for(int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
//Number Theoretic Transformation
ll gcd(ll a,ll b){
    if(b==0) return a;
    else return gcd(b,a%b);
}
ll egcd(ll a, ll b, ll & x, ll & y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }
    ll x1, y1; ll d = egcd(b % a, a, x1, y1);
    x = y1 - (b / a) * x1; y = x1;
    return d;
}
ll ModuloInverse(ll a,ll n){
    ll x,y; x=gcd(a,n);
    a=a/x; n=n/x;
    ll res = egcd(a,n,x,y); x=(x%n+n)%n;
    return x;
}
const int mod = 998244353;
const int root = 15311432;
const int root_1 = 469870224;
const int root_pw = 1 << 23;
void fft(vector<int> & a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        int wlen = invert ? root_1 : root;
        for (int i = len; i < root_pw; i <<= 1)
            wlen = (int)(1LL * wlen * wlen % mod);
        for (int i = 0; i < n; i += len) {
            int w = 1;
            for (int j = 0; j < len / 2; j++) {
int u=a[i+j],v=(int)(1LL*a[i+j+len/2]*w%mod);
a[i+j] = u + v < mod ? u + v : u + v - mod;
a[i+j+len/2] = u - v >= 0 ? u - v : u - v + mod;
w = (int)(1LL * w * wlen % mod);
            }
        }
    }
    if (invert) {
        int n_1 = (int) ModuloInverse(n, mod);
        for (int & x : a)
            x = (int)(1LL * x * n_1 % mod);
    }
}
vector<int> multiply(vector<int>&a,vector<int> &b){
    vector<int> fa(a.begin(), a.end());
    vector<int> fb(b.begin(), b.end());
    int n = 1;
    while(n < a.size()+b.size())
        n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa, false); fft(fb, false);
    for(int i = 0; i < n; i++)
        fa[i] = (int) (1LL*fa[i]*fb[i]%mod);
    fft(fa, true);
    vector<int> result(n);
    for(int i = 0; i < n; i++)
        result[i] = fa[i];
    return result;
}
```

6.3 Polynomial Algebra

```
namespace algebra {
  const int inf = 1e9; const int magic = 500;
// threshold for sizes to run the naive algo
  namespace fft {
    const int maxn = 1 << 18;
    typedef double ftype;
    typedef complex<ftype> point;
    point w[maxn];
    const ftype pi = acos(-1);
    bool initiated = 0;
    void init() {
      if(!initiated) {
        for(int i = 1; i < maxn; i *= 2) {
          for(int j = 0; j < i; j++) {
            w[i + j] = polar(ftype(1), pi * j / i);
          }
        }
        initiated = 1;
      }
    }
    template<typename T>
    void fft(T *in, point *out, int n, int k = 1) {
      if(n == 1) {
        *out = *in;
      } else {
        n /= 2;
        fft(in, out, n, 2 * k);
        fft(in + k, out + n, n, 2 * k);
        for(int i = 0; i < n; i++) {
          auto t = out[i + n] * w[i + n];
          out[i + n] = out[i] - t;
          out[i] += t;
        }
      }
    }
    template<typename T>
    void mul_slow(vector<T> &a, const vector<T>&b){
      vector<T> res(a.size() + b.size() - 1);
      for(size_t i = 0; i < a.size(); i++) {
        for(size_t j = 0; j < b.size(); j++) {
          res[i + j] += a[i] * b[j];
        }
      }
      a = res;
    }
    template<typename T>
    void mul(vector<T> &a, const vector<T> &b) {
      if(min(a.size(), b.size()) < magic) {
        mul_slow(a, b);
        return;
      }
      init();
      static const int shift= 15,mask=(1<<shift)-1;
      size_t n = a.size() + b.size() - 1;
      while(__builtin_popcount(n) != 1) {
        n++;
      }
      a.resize(n);
      static point A[maxn], B[maxn];
      static point C[maxn], D[maxn];
      for(size_t i = 0; i < n; i++) {
        A[i] = point(a[i] & mask, a[i] >> shift);
        if(i < b.size()) {
          B[i] = point(b[i] & mask, b[i] >> shift);
        } else {
          B[i] = 0;
        }
      }
      fft(A, C, n); fft(B, D, n);
      for(size_t i = 0; i < n; i++) {
        point c0 = C[i] + conj(C[(n - i) % n]);
        point c1 = C[i] - conj(C[(n - i) % n]);
        point d0 = D[i] + conj(D[(n - i) % n]);
        point d1 = D[i] - conj(D[(n - i) % n]);
        A[i] = c0 * d0 - point(0, 1) * c1 * d1;
        B[i] = c0 * d1 + d0 * c1;
      }
      fft(A, C, n); fft(B, D, n);
      reverse(C + 1, C + n);
      reverse(D + 1, D + n);
      int t = 4 * n;
      for(size_t i = 0; i < n; i++) {
        int64_t A0 = llround(real(C[i]) / t);
        T A1 = llround(imag(D[i]) / t);
        T A2 = llround(imag(C[i]) / t);
        a[i] = A0 + (A1 << shift) + (A2 << 2*shift);
      }
      return;
    }
  }
  template<typename T>
  T bpow(T x, size_t n) {
    return n?n%2?x*bpow(x,n-1):bpow(x*x,n/2):T(1);
  }
  template<typename T>
  T bpow(T x, size_t n, T m) {
    return n?n%2?x*bpow(x,n-1,m)%m:
           bpow(x*x%m,n/2,m):T(1);
  }
  template<typename T>
  T gcd(const T &a, const T &b) {
    return b == T(0) ? a : gcd(b, a % b);
  }
  template<typename T>
  T nCr(T n, int r) { // runs in O(r)
    T res(1);
    for(int i = 0; i < r; i++) {
```

```
    res *= (n - T(i));
    res /= (i + 1);
  }
  return res;
}
template<int m>
struct modular {
  int64_t r;
  modular() : r(0) {}
  modular(int64_t rr) : r(rr) {
  if(abs(r) >= m) r %= m; if(r < 0) r += m;}
  modular inv() const {return bpow(*this, m - 2);}
  modular operator*(modular&t){return(r*t.r)%m;}
modular operator/(modular&t){return*this*t.inv();}
  modular operator+=(modular&t){r+=t.r;
  if(r>=m) r-=m; return *this;}
  modular operator -= ( modular &t) {
      r -= t.r; if(r < 0) r += m; return *this;}
  modular operator + ( modular &t) {
      return modular(*this) += t;}
  modular operator - ( modular &t) {
      return modular(*this) -= t;}
  modular operator *= ( modular &t) {
      return *this = *this * t;}
  modular operator /= ( modular &t) {
      return *this = *this / t;}
  bool operator==(modular&t){return r==t.r;}
  bool operator!=(modular&t){return r != t.r;}
  operator int64_t() const {return r;}
};
template<int T>
istream& operator>> (istream &in, modular<T> &x){
  return in >> x.r;
}
template<typename T>
struct poly {
  vector<T> a;
  void normalize() { // get rid of leading zeroes
    while(!a.empty() && a.back() == T(0)) {
      a.pop_back();
    }
  }
  poly(){}
  poly(T a0) : a{a0}{normalize();}
  poly(vector<T> t) : a(t){normalize();}
  poly operator += (const poly &t) {
    a.resize(max(a.size(), t.a.size()));
    for(size_t i = 0; i < t.a.size(); i++) {
      a[i] += t.a[i];
    }
    normalize();
    return *this;
  }
  poly operator -= (const poly &t) {
    a.resize(max(a.size(), t.a.size()));
    for(size_t i = 0; i < t.a.size(); i++) {
      a[i] -= t.a[i];
    }
    normalize();
    return *this;
  }
  poly operator+(poly &t){return poly(*this)+=t;}
  poly operator-(poly &t){return poly(*this)-=t;}
poly mod_xk(size_t k){
// get same polynomial mod x^k
  k = min(k, a.size());
  return vector<T>(begin(a), begin(a) + k);
}
poly mul_xk(size_t k) const { // multiply by x^k
  poly res(*this);
  res.a.insert(begin(res.a), k, 0);
  return res;
}
poly div_xk(size_t k) const {
// divide by x^k, dropping coefficients
  k = min(k, a.size());
  return vector<T>(begin(a) + k, end(a));
}
poly substr(size_t l, size_t r) const {
// return mod_xk(r).div_xk(l)
  l = min(l, a.size());
  r = min(r, a.size());
  return vector<T>(begin(a) + l, begin(a) + r);
}
poly inv(size_t n) const {
// get inverse series mod x^n
  assert(!is_zero());
  poly ans = a[0].inv();
  size_t a = 1;
  while(a < n) {
    poly C =(ans*mod_xk(2*a)).substr(a, 2 * a);
    ans -= (ans * C).mod_xk(a).mul_xk(a);
    a *= 2;
  }
  return ans.mod_xk(n);
}
poly operator *= (const poly &t)
  {fft::mul(a, t.a); normalize(); return *this;}
poly operator * (const poly &t) const
  {return poly(*this) *= t;}
poly reverse(size_t n, bool rev = 0) const {
// reverses and leaves only n terms
  poly res(*this);
  if(rev) { // If rev = 1 then tail goes to head
    res.a.resize(max(n, res.a.size()));
  }
  std::reverse(res.a.begin(), res.a.end());
  return res.mod_xk(n);
}
pair<poly, poly> divmod_slow(const poly &b)
const { // when divisor or quotient is small
  vector<T> A(a);
  vector<T> res;
  while(A.size() >= b.a.size()) {
    res.push_back(A.back() / b.a.back());
    if(res.back() != T(0)) {
      for(size_t i = 0; i < b.a.size(); i++) {
        A[A.size() - i - 1] -= res.back() *
                  b.a[b.a.size() - i - 1];
      }
    }
    A.pop_back();
  }
  std::reverse(begin(res), end(res));
  return {res, A};
}
pair<poly, poly> divmod(const poly &b) const
{ // returns quotient and remainder of a mod b
  if(deg() < b.deg()) {
    return {poly{0}, *this};
  }
  int d = deg() - b.deg();
  if(min(d, b.deg()) < magic) {
    return divmod_slow(b);
  }
  poly D = (reverse(d + 1) * b.reverse(d + 1).
  inv(d + 1)).mod_xk(d + 1).reverse(d + 1, 1);
  return {D, *this - D * b};
}
poly operator/(poly&t){return divmod(t).first;}
poly operator%(poly&t){return divmod(t).second;}
poly operator *= ( T &x) {
  for(auto &it: a) {
    it *= x;
  }
  normalize();
  return *this;
}
poly operator /= (const T &x) {
  for(auto &it: a) {
    it /= x;
  }
  normalize();
  return *this;
}
poly operator *(T &x){return poly(*this) *= x;}
poly operator /(T &x){return poly(*this) /= x;}
void print() const {
  for(auto it: a) {
    cout << it << ' ';
  }
  cout << endl;
}
T eval(T x){ // evaluates in single point x
  T res(0);
  for(int i = int(a.size()) - 1; i >= 0; i--) {
    res *= x;
    res += a[i];
  }
  return res;
}
T& lead() { // leading coefficient
  return a.back();
}
int deg() const { // degree
  return a.empty() ? -inf : a.size() - 1;
}
bool is_zero() const { // is polynomial zero
  return a.empty();
}
T operator [](int idx) const {
  return idx>=(int)a.size()||idx<0?T(0):a[idx];
}
T& coef(size_t idx) {
// mutable reference at coefficient
  return a[idx];
}
bool operator == ( poly &t) {return a == t.a;}
```

```
  bool operator != ( poly &t) {return a != t.a;}
  poly deriv() { // calculate derivative
    vector<T> res;
    for(int i = 1; i <= deg(); i++) {
      res.push_back(T(i) * a[i]);
    }
    return res;
  }
  poly integr() { // calculate integral with C = 0
    vector<T> res = {0};
    for(int i = 0; i <= deg(); i++) {
      res.push_back(a[i] / T(i + 1));
    }
    return res;
  }
  size_t leading_xk() const {
  // Let p(x) = x^k * t(x), return k
    if(is_zero()) {
      return inf;
    }
    int res = 0;
    while(a[res] == T(0)) {
      res++;
    }
    return res;
  }
  poly log(size_t n){//calculate log p(x) mod x^n
    assert(a[0] == T(1));
    return (deriv().mod_xk(n)*inv(n))
            .integr().mod_xk(n);
  }
  poly exp(size_t n){//calculate exp p(x) mod x^n
    if(is_zero()) {
      return T(1);
    }
    assert(a[0] == T(0));
    poly ans = T(1); size_t a = 1;
    while(a < n) {
      poly C=ans.log(2*a).div_xk(a)-substr(a,2*a);
      ans -= (ans * C).mod_xk(a).mul_xk(a);
      a *= 2;
    }
    return ans.mod_xk(n);
  }
  poly pow_slow(size_t k,size_t n){//if k is small
    return k?k%2?(*this*pow_slow(k-1,n)).mod_xk(n)
  :(*this* *this).mod_xk(n).pow_slow(k/2,n):T(1);
  }
  poly pow(size_t k,size_t n){
  //calculate p^k(n) mod x^n
    if(is_zero()) return *this;
    if(k < magic) return pow_slow(k, n);
    int i = leading_xk();
    T j = a[i];
    poly t = div_xk(i) / j;
    return bpow(j,k)*(t.log(n)*T(k))
            .exp(n).mul_xk(i*k).mod_xk(n);
  }
  poly mulx(T x){
  //component-wise multiplication with x^k
    T cur = 1; poly res(*this);
    for(int i = 0; i <= deg(); i++) {
      res.coef(i) *= cur;
      cur *= x;
    }
    return res;
  }
  poly mulx_sq(T x){
  //component-wise multiplication with x^{k^2}
    T cur = x; T total = 1; T xx = x * x;
    poly res(*this);
    for(int i = 0; i <= deg(); i++) {
      res.coef(i) *= total;
      total *= cur;
      cur *= xx;
    }
    return res;
  }
  vector<T> chirpz_even(T z, int n) {
  // P(1), P(z^2), P(z^4), ..., P(z^2(n-1))
    int m = deg();
    if(is_zero()) {
      return vector<T>(n, 0);
    }
    vector<T> vv(m + n);
    T zi = z.inv(); T zz = zi * zi;
    T cur = zi; T total = 1;
    for(int i = 0; i <= max(n - 1, m); i++) {
      if(i <= m) {vv[m - i] = total;}
      if(i < n) {vv[m + i] = total;}
      total *= cur;
      cur *= zz;
    }
  poly w=(mulx_sq(z)*vv).substr(m,m+n).mulx_sq(z);
    vector<T> res(n);
    for(int i = 0; i < n; i++) {
      res[i] = w[i];
    }
    return res;
  }
  vector<T> chirpz(T z, int n) {
  // P(1), P(z), P(z^2), ..., P(z^(n-1))
    auto even = chirpz_even(z, (n + 1) / 2);
    auto odd = mulx(z).chirpz_even(z, n / 2);
    vector<T> ans(n);
    for(int i = 0; i < n / 2; i++) {
      ans[2 * i] = even[i];
      ans[2 * i + 1] = odd[i];
    }
    if(n % 2 == 1) {
      ans[n - 1] = even.back();
    }
    return ans;
  }
  template<typename iter>
  vector<T> eval(vector<poly>&tree,int v,iter l
  ,iter r) { // auxiliary evaluation function
    if(r - l == 1) {
      return {eval(*l)};
    } else {
      auto m = l + (r - l) / 2;
      auto A = (*this % tree[2 * v]).eval(tree,
              2 * v, l, m);
      auto B = (*this % tree[2 * v + 1]).eval
                (tree, 2 * v + 1, m, r);
      A.insert(end(A), begin(B), end(B));
      return A;
    }
  }
  vector<T> eval(vector<T> x) {
    // evaluate polynomial in (x1, ..., xn)
    int n = x.size();
    if(is_zero()) {
      return vector<T>(n, T(0));
    }
    vector<poly> tree(4 * n);
    build(tree, 1, begin(x), end(x));
    return eval(tree, 1, begin(x), end(x));
  }
  template<typename iter>
  poly inter(vector<poly>&tree,int v,iter l,iter
r,iter ly,iter ry){//axiliary interpolation function
    if(r - l == 1) {
      return {*ly / a[0]};
    } else {
      auto m = l + (r - l) / 2;
      auto my = ly + (ry - ly) / 2;
      auto A = (*this % tree[2 * v]).
              inter(tree, 2 * v, l, m, ly, my);
      auto B = (*this % tree[2 * v + 1]).
            inter(tree, 2 * v + 1, m, r, my, ry);
      return A * tree[2 * v + 1] + B * tree[2*v];
    }
  }
};
template<typename T>
poly<T> operator * (const T& a,const poly<T>&b){
  return b * a;
}
template<typename T>
poly<T> xk(int k) { // return x^k
  return poly<T>{1}.mul_xk(k);
}
template<typename T>
T resultant(poly<T>a,poly<T>b){
  //computes resultant of a and b
  if(b.is_zero()) {
    return 0;
  } else if(b.deg() == 0) {
    return bpow(b.lead(), a.deg());
  } else {
    int pw = a.deg();
    a %= b; pw -= a.deg();
    T mul=bpow(b.lead(),pw)*T((b.deg()&a.deg()&1)
    ?-1:1);  T ans= resultant(b, a);
    return ans * mul;
  }
}
template<typename iter>
poly<typename iter::value_type> kmul(iter L,
                              iter R) { // computes
//(x-a1)(x-a2)...(x-an) without building tree
  if(R - L == 1) {
  return vector<typename iter::value_type>{-*L,1};
  } else {
    iter M = L + (R - L) / 2;
    return kmul(L, M) * kmul(M, R);
  }
}
```

```cpp
  template<typename T, typename iter>
  poly<T> build(vector<poly<T>> &res, int v, iter L,
                              iter R){//
// builds evaluation tree for (x-a1)(x-a2)...(x-an)
    if(R - L == 1) {
      return res[v] = vector<T>{-*L, 1};
    } else {
      iter M = L + (R - L) / 2;
      return res[v] = build(res, 2 * v, L, M)
              * build(res, 2 * v + 1, M, R);
    }
  }
  template<typename T>
  poly<T> inter(vector<T>x,vector<T>y){//interpo-
  //lates minimum polynomial from (xi, yi) pairs
    int n = x.size();
    vector<poly<T>> tree(4 * n);
    return build(tree, 1, begin(x), end(x)).deriv()
.inter(tree, 1, begin(x), end(x), begin(y), end(y));
  }
};
using namespace algebra;
const int mod = 1e9 + 7;
typedef modular<mod> base;
typedef poly<base> polyn;
using namespace algebra;
signed main() {
  int n = 100000;
  polyn a; vector<base> x;
  for(int i = 0; i <= n; i++) {
    a.a.push_back(1 + rand() % 100);
    x.push_back(1 + rand() % (2 * n));
  }
  sort(begin(x), end(x));
  x.erase(unique(begin(x), end(x)), end(x));
  auto b = a.eval(x);
  cout << clock() / double(CLOCKS_PER_SEC) << endl;
  auto c = inter(x, b);
  polyn md = kmul(begin(x), end(x));
  cout << clock() / double(CLOCKS_PER_SEC) << endl;
  assert(c == a % md);
}
```

6.4 all comb

```cpp
vector<int> ans;
void gen(int n, int k, int idx, bool rev) {
    if (k > n || k < 0) return;
    if (!n) {
        for (int i = 0; i < idx; ++i) {
            if (ans[i])
                cout << i + 1;
        }
        cout << "\n";
        return;
    }
    ans[idx] = rev;
    gen(n - 1, k - rev, idx + 1, false);
    ans[idx] = !rev;
    gen(n - 1, k - !rev, idx + 1, true);
}
void all_combinations(int n, int k) {
    ans.resize(n);
    gen(n, k, 0, false);
}
```

6.5 gauss elimination

```cpp
/*  n rows/equations, m+1 columns, m variables
    calculates determinant, rank and ans[] ->value
    for variables
    returns {0, 1, INF} -> number of solutions */
const double EPS = 1e-9;
#define MAX 105
#define INF 100000000
int where[MAX], Rank;
double Det;
int gauss(double a[MAX][MAX],
          double ans[MAX], int n, int m) {
  Det = 1.0, Rank = 0;
  memset(where, -1, sizeof(where));
  for(int col=0,row = 0;col<m&&row < n; ++col) {
    int sel = row;
    for(int i = row+1; i < n; ++i)
      if(fabs(a[i][col])>fabs(a[sel][col])) sel=i;
    if(fabs(a[sel][col])<EPS) {Det=0.0; continue;}
  for(int j=0;j<=m;++j)swap(a[sel][j],a[row][j]);
    if(row != sel) Det = -Det;
    Det *= a[row][col];
    where[col] = row;
    double s = (1.0 / a[row][col]);
    for(int j = 0; j <= m; ++j) a[row][j] *= s;
    for(int i = 0; i < n; ++i) if (i != row &&
          fabs(a[i][col]) > EPS) {
      double t = a[i][col];
      for(int j = 0; j <= m; ++j)
        a[i][j] -= a[row][j] * t;
    }
    ++row, ++Rank;
  }
  for(int i = 0; i < m; ++i)
    ans[i] = (where[i] == -1) ?0.0:a[where[i]][m];
  for(int i = Rank; i < n; ++i)
    if(fabs(a[i][m]) > EPS) return 0;
  for(int i = 0; i < m; ++i)
    if(where[i] == -1) return INF;
  return 1;
}
// calculates gauss modulo a prime
long long Det;
long long bigmod(long long x,
                 long long pow, long long mod) {
   long long ret = 1;
   while(pow > 0) {
      if(pow & 1) ret = (ret * x) % mod;
      x = (x * x) % mod;
      pow >>= 1;
   }
   return ret;
}
#define INVERSE(a, m) bigmod(a, m-2, m)
int gauss(long long a[MAX][MAX],
  long long ans[MAX],int n,int m,long long mod){
 Det = 1, Rank = 0;
 memset(where, -1, sizeof(where));
 for(int col = 0, row = 0; col<m&&row<n;++col){
  int sel = row;
  for(int i = row+1; i < n; ++i)
     if(abs(a[i][col]) > abs(a[sel][col])) sel=i;
  if(!a[sel][col]) { Det = 0; continue; }
  for(int j=0;j<=m;++j) swap(a[sel][j],a[row][j]);
  if(row != sel) Det = -Det;
  Det = (Det * a[row][col]) % mod;
  where[col] = row;
                 // inverse of a[row][col]
  long long s = INVERSE(a[row][col], mod);
  for(int j = 0; j <= m; ++j)
    a[row][j] = (a[row][j] * s) % mod;
  for(int i = 0; i < n; ++i) if (i != row &&
                            a[i][col] > 0) {
     long long t = a[i][col];
     for(int j = 0; j <= m; ++j) a[i][j] =
      (a[i][j] - (a[row][j]*t) % mod + mod)%mod;
  }
  ++row, ++Rank;
 }
 for(int i = 0; i < m; ++i)
  ans[i] = (where[i] == -1) ? 0 : a[where[i]][m];
 for(int i = Rank; i < n; ++i)
  if(a[i][m]) return 0;
 for(int i = 0; i < m; ++i)
  if(where[i] == -1) return INF;
 return 1;
}
// calculates 32 times faster for modulo 2
int Det; // number of variables (must be defined)
int gauss(vector < bitset<MAX> > &a,
          bitset<MAX> &ans, int n, int m) {
  Det = 1, Rank = 0;
   memset(where, -1, sizeof(where));
   for(int col=0,row=0; col < m && row < n;++col){
      int sel = row;
      for(int i = row; i < n; ++i)
        if(a[i][col]) { sel = i; break; }
      if(!a[sel][col]) { Det = 0; continue; }
      swap(a[sel], a[row]);
      if(row != sel) Det = -Det;
      Det &= a[row][col];
      where[col] = row;
      for(int i = 0; i < n; ++i)
        if (i != row&&a[i][col] > 0)a[i]^=a[row];
      ++row, ++Rank;
   }
   for(int i = 0; i < m; ++i)
      ans[i] = (where[i] == -1)?0:a[where[i]][m];
   for(int i = Rank;i<n;++i)if(a[i][m]) return 0;
   for(int i = 0; i < m; ++i)
    if(where[i] == -1) return INF;
   return 1;
}
```

6.6 linear sieve

```cpp
vector<int> lp(N+1);
vector<int> pr;
for (int i=2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;pr.push_back(i);}
for (int j=0; j < (int)pr.size()
&& pr[j] <= lp[i] && i*pr[j] <= N; ++j)
    lp[i * pr[j]] = pr[j];
}
```

7 String

7.1 Aho Corasick

```cpp
const int K = 26;

struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

7.2 Hashing

```cpp
const int MAX = 3000009;
ll mods[2] = {1000000007, 1000000009};
//Some back-up primes: 1072857881, 1066517951, 1040160883
ll bases[2] = {137, 281};
ll pwbase[3][MAX];
void Preprocess(){
    pwbase[0][0] = pwbase[1][0] = 1;
    for(ll i = 0; i < 2; i++){
        for(ll j = 1; j < MAX; j++){
            pwbase[i][j] = (pwbase[i][j - 1] * bases[i]) % mods[i];
        }
    }
}

ll fmod(ll a, ll b, int md=mods[0]) {
  unsigned long long x = (long long) a * b;
  unsigned xh = (unsigned) (x >> 32), xl = (unsigned) x, d, m;
  asm(
    "divl %4; \n\t"
    : "=a" (d), "=d" (m)
    : "d" (xh), "a" (xl), "r" (md)
  );
  return m;
}

struct Hashing{
    vector<vector<ll>> hsh;

    Hashing(){}
    Hashing(string &_str) {
        hsh.push_back(vector<ll>(_str.size()+5, 0));
        hsh.push_back(vector<ll>(_str.size()+5, 0));
        Build1(_str);
    }

    void Build(const string &str){Build1(str);Build2(str);}
    void Build1(const string &str) {
        int j = 0;
        for(ll i = str.size() - 1; i >= 0; i--){
            hsh[j][i] = fmod(hsh[j][i + 1], bases[j], mods[j]) + str[i];
            if (hsh[j][i] > mods[j])
                hsh[j][i] -= mods[j];
        }
    }
    void Build2(const string &str) {
        int j = 1;
        for(ll i = str.size() - 1; i >= 0; i--){
            hsh[j][i] = fmod(hsh[j][i + 1], bases[j], mods[j]) + str[i];
            if (hsh[j][i] > mods[j])
                hsh[j][i] -= mods[j];
        }
    }

    pair<ll,ll> GetHash(ll i, ll j){
        assert(i <= j);
        ll tmp1 = (hsh[0][i] - fmod(hsh[0][j + 1] ,pwbase[0][j - i + 1]));
        ll tmp2 = (hsh[1][i] - fmod(hsh[1][j + 1] , pwbase[1][j - i + 1], mods[1]));
        if(tmp1 < 0) tmp1 += mods[0];
        if(tmp2 < 0) tmp2 += mods[1];
        return make_pair(tmp1, tmp2);
    }
    ll getSingleHash(ll i, ll j) {
        assert(i <= j);
        ll tmp1 = (hsh[0][i] - fmod(hsh[0][j + 1] ,pwbase[0][j - i + 1]));
        if(tmp1 < 0) tmp1 += mods[0];
        return tmp1;
    }
};
```

7.3 Manacher's Algorithm

```cpp
#include<bits/stdc++.h>
#include<vector>
using namespace std;
int main()
{
    int T,l;
    char s[MAX];
    gets(s);
    int n=strlen(s);
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i
        while (0 <= i - k && i + k < n && s[i - k] == s
        {
            k++;
        }
        d1[i] = k--;
        if (i + k > r)
        {
            l = i - k;
            r = i + k;
        }
    }
    vector<int> d2(n);
    for (int i = 0, l = 0, r = -1; i < n; i++)
    {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r
        while (0 <= i - k - 1 && i + k < n && s[i - k -
        {
            k++;
        }
        d2[i] = k--;
        if (i + k > r)
        {
            l = i - k - 1;
            r = i + k ;
        }
    }
    return 0;
}
```

7.4 Suffix Array

```cpp
vector<int> sort_cyclic_shifts(char *s) {
    int n = strlen(s);
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n),0);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i=1;i<alphabet;i++) cnt[i]+=cnt[i-1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]]) classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
```

```
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i<n;i++) cnt[c[pn[i]]]++;
      for(int i=1;i<classes;i++) cnt[i]+=cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            int ind=p[i] + (1 << h);
            if(ind>=n) ind=ind-n;
            pair<int, int> cur = {c[p[i]], c[ind]};
            ind=p[i-1] + (1 << h);
            if(ind>=n) ind=ind-n;
            pair<int, int> prev = {c[p[i-1]], c[ind]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
vector<int> suffix_array_construction(char *s) {
    int n=strlen(s);
    s[n]='#';
    vector<int> sorted_shifts=sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
vector<int> lcp_construction(char *s,
                    vector<int> const& p) {
    int n = strlen(s);
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;
    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}
int lcp(int i, int j) {
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (c[k][i] == c[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}
```

7.5 Suffix Automaton

```
class SuffixAutomaton{
  bool complete; int last; set<char> alphabet;
  struct state{
      // shortest_non_appearing_string -> snas
      // length_of_substrings -> los
      int len, link,endpos,first_pos,snas,height;
      ll substrings,los;
      bool is_clone;
      map<char, int> next; vector<int> inv_link;
      state(int leng=0,int li=0){
        len=leng; link=li; first_pos=-1;
        substrings=0; los=0;
        endpos=1; snas=0;
        is_clone=false; height=0;
      }
  };
  vector<state> st;
  void process(int node){
    map<char, int> ::iterator mit;
    st[node].substrings=1;st[node].snas=st.size();
    if((int) st[node].next.size()
        <(int) alphabet.size()) st[node].snas=1;
    for(mit=st[node].next.begin();
         mit!=st[node].next.end(); ++mit){
        if(st[mit->second].substrings==0)
          process(mit->second);
        st[node].height=max(st[node].height,
                1+st[mit->second].height);
        st[node].substrings=st[node].substrings
                +st[mit->second].substrings;
        st[node].los=st[node].los
  +st[mit->second].los+st[mit->second].substrings;
        st[node].snas=min(st[node].snas
        ,1+st[mit->second].snas);
    }
    if(st[node].link!=-1){
      st[st[node].link].inv_link.push_back(node);
    }
  }
  void set_suffix_links(int node){
    int i;
    for(i=0; i<st[node].inv_link.size(); i++){
      set_suffix_links(st[node].inv_link[i]);
      st[node].endpos=st[node].endpos+st[st[node].
                  inv_link[i]].endpos;
    }
  }
  void output_all_occurrences(int v,
  int P_length,vector<int> &pos){
    if (!st[v].is_clone)
        pos.push_back(st[v].first_pos - P_length+1);
    for (int u : st[v].inv_link)
        output_all_occurrences(u, P_length, pos);
  }
void kth_smallest(int node,int k,vector<char> &str){
  if(k==0) return;
  map<char, int> ::iterator mit;
  for(mit=st[node].next.begin();
    mit!=st[node].next.end(); ++mit){
    if(st[mit->second].substrings<k)
        k=k-st[mit->second].substrings;
    else{
        str.push_back(mit->first);
        kth_smallest(mit->second,k-1,str);
        return;
    }
  }
}
int find_occurrence_index(int node,int index,
                  vector<char> &str){
    if(index==str.size()) return node;
    if(!st[node].next.count(str[index])) return -1;
    else return find_occurrence_index(st[node].
            next[str[index]],index+1,str);
}
void klen_smallest(int node,int k,vector<char> &str)
{
  if(k==0) return;
  map<char, int> ::iterator mit;
  for(mit=st[node].next.begin();
    mit!=st[node].next.end(); ++mit){
      if(st[mit->second].height>=k-1){
          str.push_back(mit->first);
          klen_smallest(mit->second,k-1,str);
          return;
      }
  }
}
void minimum_non_existing_string(int node,
                  vector<char> &str){
  map<char, int> ::iterator mit;
  set<char>::iterator sit;
  for(mit=st[node].next.begin(),sit=
alphabet.begin();sit!=alphabet.end();++sit,++mit){
    if(mit==st[node].next.end()||mit->first!=(*sit))
    {
        str.push_back(*sit);
        return;
    }
    else if(st[node].snas==1+st[mit->second].snas)
    {
        str.push_back(*sit);
      minimum_non_existing_string(mit->second,str);
        return;
    }
  }
}
void find_substrings(int node,int index,vector<char>
&str,vector<pair<ll,ll> > &sub_info){
  sub_info.push_back(make_pair(st[node].substrings,
  st[node].los+st[node].substrings*index));
  if(index==str.size()) return;
  if(st[node].next.count(str[index]))
  {
    find_substrings(st[node].next[str[index]],
                    index+1,str,sub_info);
    return;
  }
  else
  {
    sub_info.push_back(make_pair(0,0));
  }
}
void check()
```

```
{
    if(!complete)
    {
        process(0);
        set_suffix_links(0);
        int i;
        complete=true;
    }
}
public:
SuffixAutomaton(set<char> &alpha)
{
  st.push_back(state(0,-1));
  last=0;
  complete=false;
  set<char>::iterator sit;
  for(sit=alpha.begin(); sit!=alpha.end(); sit++)
  {
    alphabet.insert(*sit);
  }
  st[0].endpos=0;
}
void sa_extend(char c){
  int cur = st.size();
  //printf("New node (%d,%c)\n",cur,c);
  st.push_back(state(st[last].len + 1));
  st[cur].first_pos=st[cur].len-1;
  int p = last;
  while (p != -1 && !st[p].next.count(c)){
      st[p].next[c] = cur;
      //printf("Set edge %d -> %d (%c)\n",p,cur,c);
      p = st[p].link;
  }
  if (p == -1){
      st[cur].link = 0;
      //printf("Set link %d -> %d\n",cur,0);
  }
  else{
      int q = st[p].next[c];
      if (st[p].len + 1 == st[q].len){
          st[cur].link = q;
          //printf("Set link %d -> %d\n",cur,q);
      }
      else{
          int clone = st.size();
//printf("Create clone node %d from %d\n",clone,q);
//printf("Set link %d -> %d\n",clone,st[q].link);
        st.push_back(state(st[p].len+1,st[q].link));
          st[clone].next = st[q].next;
          st[clone].is_clone=true;
          st[clone].endpos=0;
          st[clone].first_pos=st[q].first_pos;
          while (p != -1 && st[p].next[c] == q){
//printf("Change transition %d -> %d : %d -> %d (%c)
// \n",p,q,p,clone,c);
              st[p].next[c] = clone;
              p = st[p].link;
          }
//printf("Change link %d -> %d : %d -> %d\n",q,st[q]
// .link,q,clone);
//printf("Set link %d -> %d\n",cur,clone);
          st[q].link = st[cur].link = clone;
      }
  }
  last = cur;
  complete=false;
}
~SuffixAutomaton(){
    int i;
    for(i=0; i<st.size(); i++){
        st[i].next.clear();
        st[i].inv_link.clear();
    }
    st.clear();
    alphabet.clear();
}
  void kth_smallest(int k,vector<char> &str){
      check();
      kth_smallest(0,k,str);
  }
int FindFirstOccurrenceIndex(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind==0) return -1;
    else if(ind==-1) return st.size();
    else return st[ind].first_pos+1-(int)str.size();
}
void FindAllOccurrenceIndex(vector<char> &str,
        vector<int> &pos){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind!=-1)
        output_all_occurrences(ind,str.size(),pos);
}
int Occurrences(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
    if(ind==0) return 1;
    else if(ind==-1) return 0;
    else return st[ind].endpos;
}
void klen_smallest(int k,vector<char> &str){
    check();
    if(st[0].height>=k) klen_smallest(0,k,str);
}
void minimum_non_existing_string(vector<char> &str){
    check();
    int ind=find_occurrence_index(0,0,str);
  if(ind!=-1) minimum_non_existing_string(ind,str);
}
ll cyclic_occurrence(vector<char> &str){
  check();
  int i,j,len; ll ans=0;
  int n=str.size(); set<int> S;
  set<int>::iterator it;
  for(i=0,j=0,len=0; i<n*2-1; i++){
    //printf("%d->%c\n",i,str[i%n]);
    if(st[j].next.count(str[i%n])){
        len++;
        j=st[j].next[str[i%n]];
    }
    else{
      while(j!=-1&&(!st[j].next.count(str[i%n])))
          j=st[j].link;
      if(j!=-1){
          len=st[j].len+1;
          j=st[j].next[str[i%n]];
      }
      else{
        len=0;
        j=0;
      }
    }
    while(st[j].link!=-1&&st[st[j].link].len>=n){
       j=st[j].link; len=st[j].len;
    }
    if(len>=n) S.insert(j);
  }
  for(it=S.begin();it!=S.end();++it){
      ans=ans+st[*it].endpos;
  }
  return ans;
}
}; // main
vector<char> X;
  int i;
  set<char> alpha;
  for(i=0; i<26; i++){
      alpha.insert('a'+i);
  }
  SuffixAutomaton sa(alpha);
  char c;
  for(i=0;; i++){
      scanf("%c",&c);
      if(!('a'<=c&&c<='z')) break;
      sa.sa_extend(c);
  }
  int n,j;
  scanf("%d ",&n);
  for(j=0; j<n; j++){
      for(i=0;; i++){
          scanf("%c",&c);
          if(!('a'<=c&&c<='z')) break;
          X.push_back(c);
      }
      ll ans=sa.cyclic_occurrence(X);
      X.clear();
      printf("%I64d\n",ans);
  }
```