

100% É Pouco, Pagode Importa D+

University of Brasilia

November 8, 2018

Contents

1 Data Structures

1.1 Merge Sort Tree	2
1.2 Wavelet Tree	2
1.3 Order Set	2
1.4 Hash table	3
1.5 Convex Hull Trick	3
1.6 Min queue	3
1.7 Sparse Table	3
1.8 Treap	3
1.9 ColorUpdate	4
1.10 Heavy Light Decomposition	4
1.11 Iterative Segtree	4
1.12 LiChao's Segtree	5
1.13 Palindromic tree	5

2 Math

2.1 Euclides Extendido	5
2.2 Prefix inverse	5
2.3 Pollard Rho	5
2.4 Miller Rabin	6
2.5 Totiente	6
2.6 Mobius Function	6
2.7 Mulmod TOP	6
2.8 Determinant	6
2.9 FFT	6
2.10 NTT	7

3 Graphs

3.1 Dinic	7
3.2 Min Cost Max Flow	8
3.3 Small to Large	8
3.4 Junior e Falta de Ideias	9
3.5 Kosaraju	9
3.6 Tarjan	10
3.7 Max Clique	10
3.8 Dominator Tree	10
3.9 Min Cost Matching	11

4 Strings

4.1 Aho Corasick	11
4.2 Suffix Array	11
4.3 Z Algorithm	12
4.4 Prefix function/KMP	12
4.5 Min rotation	12
4.6 All palindrome	12
4.7 Palindromic Tree	12
4.8 Suffix Automaton	13
4.9 Suffix Tree	13

5 Geometry

5.1 2D basics	14
5.2 Nearest Points	15
5.3 Convex Hull	16
5.4 Check point inside polygon	16
5.5 Check point inside polygon without lower/up- per hull	16
5.6 Minkowski sum	16

6 Miscellaneous

6.1 LIS	17
6.2 DSU rollback	17
6.3 Buildings	18
6.4 Rand	18
6.5 Klondike	18
6.6 Hilbert Order	19
6.7 Modular Factorial	19
6.8 Enumerating all submasks of a bitmask	19
6.9 Slope Trick	19
6.10 Fast IO	19
6.11 Big int	20
6.12 Knapsack Bounded with Cost	23
6.13 Burnside's Lemma	23
6.14 Wilson's Theorem	23
6.15 Fibonacci	23
6.16 Edge coloring	23
6.17 Notes	23

```
set ts=4 sw=4 sta nu rnu sc stl+=%F cindent
imap {<CR> {<CR>}<Esc>0
nmap <F2> 0V$%d
nmap <C-down> :m+1<CR>
nmap <C-up> :m-2<CR>
vmap <C-c> "+y
nmap <C-a> ggVG
syntax on
alias cmp='g++ -Wall -Wformat=2 -Wshadow -Wconversion -
fsanitize=address -fsanitize=undefined -fno-sanitize-
recover -std=c++11'
```

Data Structures

Merge Sort Tree

```
struct MergeTree{
    int n;
    vector<vector<int>> st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){
            st[p].push_back(v[L]);
            return;
        }
        int mid = (L+R)/2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p].resize(R-L+1);
        merge(st[2*p].begin(), st[2*p].end(),
              st[2*p+1].begin(), st[2*p+1].end(),
              st[p].begin());
    }

    int query(int p, int L, int R, int i, int j, int x)
    const{
        if(L > j || R < i) return 0;
        if(L >= i && R <= j){
            int id = lower_bound(st[p].begin(), st[p].end(),
                                  x) - st[p].begin();
            return int(st[p].size()) - id;
        }
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j, x) +
               query(2*p+1, mid+1, R, i, j, x);
    }
};

public:
    MergeTree(int sz, const int v[]): n(sz), st(4*sz){
        build(1, 1, n, v);
    }

    //number of elements >= x on segment [i, j]
    int query(int i, int j, int x) const{
        if(i > j) swap(i, j);
        return query(1, 1, n, i, j, x);
    }
};
```

Wavelet Tree

```
template<typename T>
class wavelet{
    T L, R;
    vector<int> l;
    vector<T> sum; // <<
    wavelet *lef, *rig;
```

```
int r(int i) const{ return i - l[i]; }

public:
    template<typename ITER>
    wavelet(ITER bg, ITER en){
        lef = rig = nullptr;
        L = *bg, R = *bg;

        for(auto it = bg; it != en; it++){
            L = min(L, *it), R = max(R, *it);
        } if(L == R) return;

        T mid = L + (R - L)/2;
        l.reserve(std::distance(bg, en) + 1);
        sum.reserve(std::distance(bg, en) + 1);
        l.push_back(0), sum.push_back(0);
        for(auto it = bg; it != en; it++){
            l.push_back(l.back() + (*it <= mid)),
            sum.push_back(sum.back() + *it);
        }

        auto tmp = stable_partition(bg, en, [mid](T x){
            return x <= mid;
        });

        if(bg != tmp) lef = new wavelet(bg, tmp);
        if(tmp != en) rig = new wavelet(tmp, en);
    }

    ~wavelet(){
        delete lef;
        delete rig;
    }

    // 1 index, first is 1st
    T kth(int i, int j, int k) const{
        if(L >= R) return L;
        int c = l[j] - l[i-1];
        if(c >= k) return lef->kth(l[i-1]+1, l[j], k);
        else return rig->kth(r(i-1)+1, r(j), k - c);
    }

    // # elements > x on [i, j]
    int cnt(int i, int j, T x) const{
        if(L > x) return j - i + 1;
        if(R <= x || L == R) return 0;
        int ans = 0;
        if(lef) ans += lef->cnt(l[i-1]+1, l[j], x);
        if(rig) ans += rig->cnt(r(i-1)+1, r(j), x);
        return ans;
    }

    // sum of elements <= k on [i, j]
    T sumk(int i, int j, T k){
        if(L == R) return R <= k ? L * (j - i + 1) : 0;
        if(R <= k) return sum[j] - sum[i-1];
        int ans = 0;
        if(lef) ans += lef->sumk(l[i-1]+1, l[j], k);
        if(rig) ans += rig->sumk(r(i-1)+1, r(j), k);
        return ans;
    }

    // swap (i, i+1) just need to update "array" l[i]
};
```

Order Set

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

```
#include <ext/pb_ds/detail/standard_policies.hpp>
```

```
using namespace __gnu_pbds; // or pb_ds;
```

```
template<typename T, typename B = null_type>
using oset = tree<T, B, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// find_by_order / order_of_key
```

Hash table

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

```
using namespace __gnu_pbds;
```

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

```
gp_hash_table<long long, int, custom_hash> table;
unordered_map<long long, int, custom_hash> uhash;
uhash.reserve(1 << 15);
uhash.max_load_factor(0.25);
```

Convex Hull Trick

```
const ll is_query = -(1LL<<62);
struct Line{
    ll m, b;
    mutable function<const Line*> succ;
    bool operator<(const Line& rhs) const{
        if(rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if(!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct Cht : public multiset<Line>{ // maintain max
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*
            (y->m - x->m);
    }
    void insert_line(ll m, ll b){
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*
            next(y); };
        if(bad(y)){ erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(
```

```
        next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y)
        );
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

Min queue

```
template<typename T>
class minQ{
    deque<tuple<T, int, int> > p;
    T delta;
    int sz;
public:
    minQ() : delta(0), sz(0) {}
    inline int size() const{ return sz; }
    inline void add(T x){ delta += x; }
    inline void push(T x, int id){
        x -= delta, sz++;
        int t = 1;
        while(p.size() > 0 && get<0>(p.back()) >= x)
            t += get<1>(p.back()), p.pop_back();
        p.emplace_back(x, t, id);
    }
    inline void pop(){
        get<1>(p.front())--, sz--;
        if(!get<1>(p.front())) p.pop_front();
    }
    T getmin() const{ return get<0>(p.front())+delta; }
    int getid() const{ return get<2>(p.front()); }
};
```

Sparse Table

```
const int N = 100005;

int v[N], n;
int dn[N][20];
int fn(int i, int j){
    if(j == 0) return v[i];
    if(!dn[i][j]) return dn[i][j];
    return dn[i][j] = min(fn(i, j-1), fn(i + (1 << (j-1)
        ), j-1));
}

int lg(int x){ return 31 - __builtin_clz(x); }

int getmn(int l, int r){ // [l, r]
    int lz = lg(r - l + 1);
    return min(fn(l, lz), fn(r - (1 << lz) + 1, lz));
}
```

Treap

```
// source: https://github.com/victorsenam/caderno/blob/
// master/code/treap.cpp
//const int N = ; typedef int num;
num X[N]; int en = 1, Y[N], sz[N], L[N], R[N];
void calc (int u) { // update node given children info
    sz[u] = sz[L[u]] + 1 + sz[R[u]];
    // code here, no recursion
}
void unlaze (int u) {
    if(!u) return;
    // code here, no recursion
}
```

```

void split_val(int u, num x, int &l, int &r) { // l gets
    <= x, r gets > x
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(X[u] <= x) { split_val(R[u], x, l, r); R[u] = l;
        l = u; }
    else { split_val(L[u], x, l, r); L[u] = r; r = u; }
    calc(u);
}
void split_sz(int u, int s, int &l, int &r) { // l gets
    first s, r gets remaining
    unlaze(u); if(!u) return (void) (l = r = 0);
    if(sz[L[u]] < s) { split_sz(R[u], s - sz[L[u]] - 1,
        l, r); R[u] = l; l = u; }
    else { split_sz(L[u], s, l, r); L[u] = r; r = u; }
    calc(u);
}
int merge(int l, int r) { // els on l <= els on r
    unlaze(l); unlaze(r); if(!l || !r) return l + r; int
        u;
    if(Y[l] > Y[r]) { R[l] = merge(R[l], r); u = l; }
    else { L[r] = merge(l, L[r]); u = r; }
    calc(u); return u;
}
void init(int n=N-1) { // XXX call before using other
    funcs
    for(int i = en = 1; i <= n; i++) { Y[i] = i; sz[i] =
        1; L[i] = R[i] = 0; }
    random_shuffle(Y + 1, Y + n + 1);
}

```

ColorUpdate

// source: <https://github.com/tfg50/Competitive-Programming/tree/master/Biblioteca/Data%20Structures>

```

#include <set>
#include <vector>

template <class Info = int>
class ColorUpdate {
public:
    struct Range {
        Range(int l = 0) { this->l = l; }
        Range(int l, int r, Info v) {
            this->l = l;
            this->r = r;
            this->v = v;
        }
        int l, r;
        Info v;

        bool operator < (const Range &b) const { return l
            < b.l; }
    };

    std::vector<Range> upd(int l, int r, Info v) {
        std::vector<Range> ans;
        if(l >= r) return ans;
        auto it = ranges.lower_bound(l);
        if(it != ranges.begin()) {
            it--;
            if(it->r > l) {
                auto cur = *it;
                ranges.erase(it);
                ranges.insert(Range(cur.l, l, cur.v));
                ranges.insert(Range(l, cur.r, cur.v));
            }
        }
    }
}

```

```

it = ranges.lower_bound(r);
if(it != ranges.begin()) {
    it--;
    if(it->r > r) {
        auto cur = *it;
        ranges.erase(it);
        ranges.insert(Range(cur.l, r, cur.v));
        ranges.insert(Range(r, cur.r, cur.v));
    }
}
for(it = ranges.lower_bound(l); it != ranges.end
    () && it->l < r; it++) {
    ans.push_back(*it);
}
ranges.erase(ranges.lower_bound(l), ranges.
    lower_bound(r));
ranges.insert(Range(l, r, v));
return ans;
}

private:
    std::set<Range> ranges;
};

```

Heavy Light Decomposition

```

void dfs_sz(int u){
    sz[u] = 1;

    for(auto &v : g[u]) if(v == p[u]){
        swap(v, g[u].back());
        g[u].pop_back();
        break;
    }

    for(auto &v : g[u]){
        p[v] = u;
        dfs_sz(v);
        sz[u] += sz[v];
        if(sz[v] > sz[g[u][0]])
            swap(v, g[u][0]);
    }
}

// nxt[u] = start of path with u
// set nxt[root] beforehand
void dfs_hld(int u){
    in[u] = t++;
    rin[in[u]] = u;
    for(auto v : g[u]){
        nxt[v] = (v == g[u][0] ? nxt[u] : v);
        dfs_hld(v);
    }
    out[u] = t;
}

// subtree of u => [ in[u], out[u] )
// path from nxt[u] to u => [ in[ nxt[u] ], in[u] ]

```

Iterative Segtree

```

T query(int l, int r, int &pos){ // [l, r]
    T rl, rr;
    for(l += n, r += n+1; l < r; l >>= 1, r >>= 1){
        if(l & 1) rl = merge(rl, st[l++]);
        if(r & 1) rr = merge(st[--r], rr);
    }
    return merge(rl, rr);
}

```

```
// initially save v[i] in st[n+i] for all i
void build(){
    for(int p = n-1; p > 0; p--){
        st[p] = merge(st[2*p], st[2*p+1]);
    }

    void update(int p, T val){
        st[p += n] = val;
        while(p >= 1) st[p] = merge(st[2*p], st[2*p+1]);
    }
}
```

LiChao's Segtree

```
void add_line(line nw, int v = 1, int l = 0, int r =
    maxn) { // [l, r)
    int m = (l + r) / 2;
    bool lef = nw.eval(l) < st[v].eval(l);
    bool mid = nw.eval(m) < st[v].eval(m);
    if(mid) swap(st[v], nw);
    if(r - l == 1) {
        return;
    } else if(lef != mid) {
        add_line(nw, 2 * v, l, m);
    } else {
        add_line(nw, 2 * v + 1, m, r);
    }
}

int get(int x, int v = 1, int l = 0, int r = maxn) {
    int m = (l + r) / 2;
    if(r - l == 1) {
        return st[v].eval(x);
    } else if(x < m) {
        return min(st[v].eval(x), get(x, 2*v, l, m));
    } else {
        return min(st[v].eval(x), get(x, 2*v+1, m, r));
    }
}
```

Palindromic tree

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 3e5 + 1, sigma = 26;
int len[maxn], link[maxn], to[maxn][sigma];
int slink[maxn], diff[maxn], series_ans[maxn];
int sz, last, n;
char s[maxn];

void init()
{
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}

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

void add_letter(char c)
{
    s[n++] = c - 'a';
    last = get_link(last);
```

```
if(!to[last][c])
{
    len[sz] = len[last] + 2;
    link[sz] = to[get_link(link[last])][c];
    diff[sz] = len[sz] - len[link[sz]];
    if(diff[sz] == diff[link[sz]])
        slink[sz] = slink[link[sz]];
    else
        slink[sz] = link[sz];
    to[last][c] = sz++;
}
last = to[last][c];
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    init();
    string s;
    cin >> s;
    int n = s.size();
    int ans[n + 1];
    memset(ans, 63, sizeof(ans));
    ans[0] = 0;
    for(int i = 1; i <= n; i++)
    {
        add_letter(s[i - 1]);
        for(int v = last; len[v] > 0; v = slink[v])
        {
            series_ans[v] = ans[i - (len[slink[v]] + diff
                [v])];
            if(diff[v] == diff[link[v]])
                series_ans[v] = min(series_ans[v],
                    series_ans[link[v]]);
            ans[i] = min(ans[i], series_ans[v] + 1);
        }
        cout << ans[i] << "\n";
    }
    return 0;
}
```

Math

Euclides Extendido

```
// a*x + b*y = gcd(a, b), <gcd, x, y>
tuple<int, int, int> euclidesExt(int a, int b) {
    if(b == 0) return make_tuple(a, 1, 0);
    int q, w, e;
    tie(q, w, e) = euclidesExt(b, a % b);
    return make_tuple(q, e, w - e * (a / b));
}
```

Prefix inverse

```
inv[1] = 1;
for(int i = 2; i < p; i++)
    inv[i] = (p - (p/i) * inv[p%i] % p) % p;
```

Pollard Rho

```
ll rho(ll n){
    if(n % 2 == 0) return 2;

    ll d, c, x, y;
    do{
        c = llrand() % n, x = llrand() % n, y = x;
        do{
```

```

        x = add(mul(x, x, n), c, n);
        y = add(mul(y, y, n), c, n);
        y = add(mul(y, y, n), c, n);
        d = __gcd(abs(x - y), n);
    }while(d == 1);
}while(d == n);
return d;
}

ll pollard_rho(ll n){
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand() % n, c = llrand() % n;
        y = x, k = 4;
        do{
            if(++i == k) y = x, k *= 2;
            x = add(mul(x, x, n), c, n);
            d = __gcd(abs(x - y), n);
        }while(d == 1);
    }while(d == n);
    return d;
}

void factorize(ll val, map<ll, int> &fac){
    if(rabin(val)) fac[ val ]++;
    else{
        ll d = pollard_rho(val);
        factorize(d, fac);
        factorize(val / d, fac);
    }
}

map<ll, int> factor(ll val){
    map<ll, int> fac;
    if(val > 1) factorize(val, fac);
    return fac;
}

```

Miller Rabin

```

bool rabin(ll n){
    if(n <= 1) return 0;
    if(n <= 3) return 1;
    ll s = 0, d = n - 1;
    while(d % 2 == 0) d /= 2, s++;
    for(int k = 0; k < 64; k++){
        ll a = (llrand() % (n - 3)) + 2;
        ll x = fexp(a, d, n);
        if(x != 1 && x != n-1){
            for(int r = 1; r < s; r++){
                x = mul(x, x, n);
                if(x == 1) return 0;
                if(x == n-1) break;
            }
            if(x != n-1) return 0;
        }
    }
    return 1;
}

```

Totiente

```

ll totiente(ll n){
    ll ans = n;
    for(ll i = 2; i*i <= n; i++){
        if(n % i == 0){
            ans = ans / i * (i - 1);

```

```

        while(n % i == 0) n /= i;
    }
}

if(n > 1) ans = ans / n * (n - 1);
return ans;
}

```

Mobius Function

```

memset(mu, 0, sizeof mu);
mu[1] = 1;
for(int i = 1; i < N; i++){
    for(int j = i + i; j < N; j += i)
        mu[j] -= mu[i];
}
// g(n) = sum{f(d)} => f(n) = sum{mu(d)*g(n/d)}

```

Mulmod TOP

```

constexpr uint64_t mod = (1ull<<61) - 1;
uint64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (
        uint32_t)b, h2 = b>>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&mod) + (l>>61) + (h << 3) + (m >>
        29) + (m << 35 >> 3) + 1;
    ret = (ret & mod) + (ret>>61);
    ret = (ret & mod) + (ret>>61);
    return ret-1;
}

```

Determinant

```

const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

```

```
cout << det;
```

FFT

```

struct base{
    double r, i;
    base(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    base operator*(base &o) const{
        return {r*o.r - i*o.i, r*o.i + o.r*i};
    }
    double real() const{ return r; }
    void operator*=(const base &o){

```

```

    (*this) = {r*o.r-i*o.i, r*o.i+o.r*i};
}
void operator+=(const base &o){r += o.r, i += o.i; }
void operator/=(const double &o){ r /= o, i /= o; }
void operator-=(const base &o){r -= o.r, i -= o.i; }
base operator+(const base &o){return {r+o.r,i+o.i};}
base operator-(const base &o){return {r-o.r,i-o.i};}
};

double PI = acos(-1);

void fft(vector<base> &a, bool inv){
    int n = (int)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 sz = 2; sz <= n; sz <= 1) {
        double ang = 2*PI/sz * (inv ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += sz){
            base w(1);
            for(int j = 0; j < sz/2; j++){
                base u = a[i+j], v = a[i+j+sz/2] * w;
                a[i+j] = u + v;
                a[i+j+sz/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv) for(int i = 0; i < n; i++) a[i] /= 1.0 * n;
}

void multiply(const vector<int> &a, const vector<int> &b
, vector<int> &res){
    vector<base> fa(a.begin(), a.end());
    vector<base> fb(b.begin(), b.end());
    size_t n = 1;
    while(n < a.size()) n <= 1;
    while(n < b.size()) n <= 1;
    n <= 1;
    fa.resize(n), fb.resize(n);

    fft(fa, false), fft(fb, false);
    for(size_t i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    res.resize(n);
    for(size_t i = 0; i < n; ++i)
        res[i] = int(fa[i].real() + 0.5);
}

```

NTT

```

const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> &a, bool invert) {
    int n = (int) 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 (wlen * 111 * 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 (a[i+j+len/2] * 1
                    11 * 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 (w * 111 * wlen % mod);
            }
        }
    }
    if (invert) {
        int nrev = reverse (n, mod);
        for (int i=0; i<n; ++i)
            a[i] = int (a[i] * 111 * nrev % mod);
    }
}

```

Graphs

Dinic

```

const int N = 100005;
const int E = 2000006;
vector<int> g[N];

int ne;
struct Edge{
    int from, to;
    ll flow, cap;
} edge[E];

int lvl[N], vis[N], pass, start = N-2, target = N-1;
int qu[N], qt, px[N];

ll run(int s, int sink, ll minE){
    if(s == sink) return minE;

    ll ans = 0;

    for(; px[s] < (int)g[s].size(); px[s]++){
        int e = g[s][ px[s] ];
        auto &v = edge[e], &rev = edge[e^1];
        if(lvl[v.to] != lvl[s]+1 || v.flow >= v.cap)
            continue;
        ll tmp = run(v.to, sink, min(minE, v.cap-v.flow));
        v.flow += tmp, rev.flow -= tmp;
        ans += tmp, minE -= tmp;
        if(minE == 0) break;
    }
    return ans;
}

bool bfs(int source, int sink){
    qt = 0;
    qu[qt++] = source;

```

```

    lvl[source] = 1;
    vis[source] = ++pass;

    for(int i = 0; i < qt; i++){
        int u = qu[i];
        px[u] = 0;
        if(u == sink) return true;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.flow >= v.cap || vis[v.to] == pass)
                continue;
            vis[v.to] = pass;
            lvl[v.to] = lvl[u]+1;
            qu[qt++] = v.to;
        }
    }
    return false;
}

ll flow(int source = start, int sink = target){
    ll ans = 0;
    while(bfs(source, sink))
        ans += run(source, sink, oo);
    return ans;
}

void addEdge(int u, int v, ll c = 1, ll rc = 0){
    edge[ne] = {u, v, 0, c};
    g[u].push_back(ne++);
    edge[ne] = {v, u, 0, rc};
    g[v].push_back(ne++);
}

void reset_flow(){
    for(int i = 0; i < ne; i++)
        edge[i].flow = 0;
}

Min Cost Max Flow

const ll oo = 1e18;
const int N = 505;
const int E = 30006;

vector<int> g[N];

int ne;

struct Edge{
    int from, to;
    ll cap, cost;
} edge[E];

int lvl[N], vis[N], pass, source, target, p[N], px[N];

ll d[N];

ll back(int s, ll minE){
    if(s == source) return minE;

    int e = p[s];

    ll f = back(edge[e].from, min(minE, edge[e].cap));
    edge[e].cap -= f;
    edge[e^1].cap += f;
    return f;
}

```

```

int dijkstra(){
    for(i, N) d[i] = oo;

    priority_queue<pair<ll, int> > q;

    d[source] = 0;

    q.emplace(0, source);

    while(!q.empty()){
        ll dis = -q.top().ff;
        int u = q.top().ss; q.pop();

        if(dis > d[u]) continue;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0) continue;
            if(d[u] + v.cost < d[v.to]){
                d[v.to] = d[u] + v.cost;
                p[v.to] = e;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
    return d[target] != oo;
}

pair<ll, ll> mincost(){
    ll ans = 0, mf = 0;
    while(dijkstra()){
        ll f = back(target, oo);
        mf += f;
        ans += f * d[target];
    }
    return {mf, ans};
}

void addEdge(int u, int v, ll c, ll cost){
    edge[ne] = {u, v, c, cost};
    g[u].pb(ne++);
}

Small to Large

void cnt_sz(int u, int p = -1){
    sz[u] = 1;

    for(int v : g[u]) if(v != p)
        cnt_sz(v, u), sz[u] += sz[v];
}

void add(int u, int p, int big = -1){
    // Update info about this vx in global answer

    for(int v : g[u]) if(v != p && v != big)
        add(v, u);
}

void dfs(int u, int p, int keep){

    int big = -1, mmx = -1;

    for(int v : g[u]) if(v != p && sz[v] > mmx)
        mmx = sz[v], big = v;

    for(int v : g[u]) if(v != p && v != big)

```



```

    dfs(v, u, 0);

    if(big != -1) dfs(big, u, 1);

    add(u, p, big);

    for(auto x : q[u]){
        // answer all queries for this vx
    }

    if(!keep){
        // Remove data from this subtree
    }
}

```

Junior e Falta de Ideias

```

#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
    subtree[v] = 1;
    for(pair<int,int> x : G[v]){
        if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.ff, v);
    }
    return subtree[v];
}

int centroid(int v, int p){
    for(pair<int,int> x : G[v]){
        if(x.ff == p || vis[x.ff]) continue;
        if(subtree[x.ff]*2 > treesize) return centroid(x.ff, v);
    }
    return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
    ans = min(ans, dist[k-d_atual] + custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            procurar_ans(x.ff, v, d_atual+1, custo+x.ss);
    }
}

void atualiza_distancia(int v, int p, int d_atual, ll custo){
    dist[d_atual] = min(dist[d_atual], custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            atualiza_distancia(x.ff, v, d_atual+1, custo+x.ss);
    }
}

```

```

void decomp(int v, int p){
    treesize = dfs(v, v);
    // if(treesize < k) return;
    int cent = centroid(v, v);
    vis[cent] = 1;

    for(int i = 1; i <= treesize; i++){
        dist[i] = 1e18;
    }

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff]){
            procurar_ans(x.ff, cent, 1, x.ss);
            atualiza_distancia(x.ff, cent, 1, x.ss);
        }
    }

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff])
            decomp(x.ff, cent);
    }
}

int main(){
    int n, i, a, b;

    scanf("%d%d", &n, &k);
    for(i = 2; i <= n; i++){
        scanf("%d%d", &a, &b);
        G[i].push_back(mp(a, b));
        G[a].push_back(mp(i, b));
    }
    ans = 1e18;
    decomp(1, -1);

    printf("%lld\n", ans == 1e18 ? -1 : ans);

    return 0;
}

```

Kosaraju

```

vector<int> g[N], gt[N], S;

int vis[N], cor[N], tempo = 1;

void dfs(int u){
    vis[u] = 1;
    for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}

int e;

void dfst(int u){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v);
}

int main(){

    for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);

    e = 0;
    reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u])
        e++, dfst(u);

    return 0;
}

```

Tarjan

```
int cnt = 0, root;
void dfs(int u, int p = -1){
    low[u] = num[u] = ++t;
    for(int v : g[u]){
        if(!num[v]){
            dfs(v, u);
            if(v == root) cnt++;
            if(low[v] >= num[u]) u PONTO DE ARTICULACAO;
            if(low[v] > num[u]) ARESTA u->v PONTE;
            low[u] = min(low[u], low[v]);
        }
        else if(v != p) low[u] = min(low[u], num[v]);
    }
}
```

root PONTO DE ARTICULACAO <=> cnt > 1

```
void tarjanSCC(int u){
    low[u] = num[u] = cnt++;
    vis[u] = 1;
    S.push_back(u);
    for(int v : g[u]){
        if(!num[v]) tarjanSCC(v);
        if(vis[v]) low[u] = min(low[u], low[v]);
    }
    if(low[u] == num[u]){
        ssc[u] = ++ssc_cnt; int v;
        do{
            v = S.back(); S.pop_back(); vis[v] = 0;
            ssc[v] = ssc_cnt;
        }while(u != v);
    }
}
```

Max Clique

long long adj[N], dp[N];

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        int x;
        scanf("%d",&x);
        if(x || i == j)
            adj[i] |= 1LL << j;
    }
}
```

```
int resto = n - n/2;
int C = n/2;
for(int i = 1; i < (1 << resto); i++){
    int x = i;
    for(int j = 0; j < resto; j++){
        if(i & (1 << j))
            x &= adj[j + C] >> C;
    }
    if(x == i){
        dp[i] = __builtin_popcount(i);
    }
}
```

```
for(int i = 1; i < (1 << resto); i++){
    for(int j = 0; j < resto; j++){
        if(i & (1 << j))
            dp[i] = max(dp[i], dp[i ^ (1 << j)]);
    }
}
```

```
int maxCliq = 0;
for(int i = 0; i < (1 << C); i++){
```

```
    int x = i, y = (1 << resto) - 1;
    for(int j = 0; j < C; j++){
        if(i & (1 << j))
            x &= adj[j] & ((1 << C) - 1), y &= adj[j] >> C;
    }
    if(x != i) continue;
    maxCliq = max(maxCliq, __builtin_popcount(i) + dp[y]);
}
```

Dominator Tree

```
vector<int> g[N], gt[N], T[N];
vector<int> S;
int dsu[N], label[N];
int sdom[N], idom[N], dfs_time, id[N];
```

```
vector<int> bucket[N];
vector<int> down[N];
```

```
void prep(int u){
    S.push_back(u);
    id[u] = ++dfs_time;
    label[u] = sdom[u] = dsu[u] = u;

    for(int v : g[u]){
        if(!id[v])
            prep(v), down[u].push_back(v);
        gt[v].push_back(u);
    }
}
```

```
int fnd(int u, int flag = 0){
    if(u == dsu[u]) return u;
    int v = fnd(dsu[u], 1), b = label[ dsu[u] ];
    if(id[ sdom[b] ] < id[ sdom[ label[u] ] ])
        label[u] = b;
    dsu[u] = v;
    return flag ? v : label[u];
}
```

```
void build_dominator_tree(int root, int sz){
    // memset(id, 0, sizeof(int) * (sz + 1));
    // for(int i = 0; i <= sz; i++) T[i].clear();
    prep(root);
```

reverse(S.begin(), S.end());

```
int w;
for(int u : S){
    for(int v : gt[u]){
        w = fnd(v);
        if(id[ sdom[w] ] < id[ sdom[u] ])
            sdom[u] = sdom[w];
    }
    gt[u].clear();

    if(u != root) bucket[ sdom[u] ].push_back(u);

    for(int v : bucket[u]){
        w = fnd(v);
        if(sdom[w] == sdom[v]) idom[v] = sdom[v];
        else idom[v] = w;
    }
    bucket[u].clear();

    for(int v : down[u]) dsu[v] = u;
```

```

        down[u].clear();
    }

    reverse(S.begin(), S.end());

    for(int u : S) if(u != root){
        if(idom[u] != sdom[u]) idom[u] = idom[ idom[u] ];
        T[ idom[u] ].push_back(u);
    }

    S.clear();
}

```

Min Cost Matching

```

// Min cost matching
// O(n^2 * m)
// n == nro de linhas
// m == nro de columnas
// n <= m | flow == n
// a[i][j] = custo pra conectar i a j
vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
for(int i = 1; i <= n; ++i){
    p[0] = i;
    int j0 = 0;
    vector<int> minv(m + 1, oo);
    vector<char> used(m + 1, false);
    do{
        used[j0] = true;
        int i0 = p[j0], delta = oo, j1;
        for(int j = 1; j <= m; ++j)
            if(!used[j]){
                int cur = a[i0][j] - u[i0] - v[j];
                if(cur < minv[j])
                    minv[j] = cur, way[j] = j0;
                if(minv[j] < delta)
                    delta = minv[j], j1 = j;
            }
        for(int j = 0; j <= m; ++j)
            if(used[j])
                u[p[j]] += delta, v[j] -= delta;
            else
                minv[j] -= delta;
        j0 = j1;
    }while(p[j0] != 0);

    do{
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    }while(j0);
}

```

```

// match[i] = coluna escolhida para linha i
vector<int> match(n + 1);
for(int j = 1; j <= m; ++j)
    match[p[j]] = j;

```

```
int cost = -v[0];
```

Strings

Aho Corasick

```

map<char, int> to[N];
int ne = 1, term[N], fail[N];

void add_string(char *str){
    int p = 0;

```

```

    for(int i = 0; str[i]; i++){
        if(!to[p][ str[i] ]) to[p][ str[i] ] = ne++;
        p = to[p][ str[i] ];
    }
    term[p] = 1;
}

```

```

int go(int s, char c){
    while(s && !to[s].count(c)) s = fail[s];
    if(to[s].count(c)) return to[s][c];
    return s;
}

```

```

void init(){

    queue<int> q;
    q.push(0);

    int u, v; char c;
    while(!q.empty()){
        u = q.front(); q.pop();

        for(auto w : to[u]){
            tie(c, v) = w;
            q.push(v);
            if(u){
                fail[v] = go(fail[u], c);
                term[v] |= term[ fail[v] ];
            }
        }
    }
}

```

Suffix Array

```

char s[N];
int n, sa[N], tsa[N], lcp[N], r[N], nr[N], c[N];

void sort(int k, int mx){
    mx += 2;
    memset(c, 0, sizeof(int) * mx);
    for(int i = 0; i < n; i++) c[i + k < n ? r[i+k]+2 : 1]++;
    partial_sum(c, c+mx, c);
    int t;
    for(int i = 0; i < n; i++)
        t = sa[i]+k < n ? r[ sa[i]+k ]+1 : 0,
        tsa[ c[t]++ ] = sa[i];
    memcpy(sa, tsa, sizeof(int) * n);
}

```

```

void build_sa(){

    for(int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];

    int t = 300, a, b;
    for(int sz = 1; sz < n; sz *= 2){
        sort(sz, t), sort(0, t);
        t = nr[ sa[0] ] = 0;
        for(int i = 1; i < n; i++){
            a = sa[i]+sz < n ? r[ sa[i]+sz ] : -1;
            b = sa[i-1]+sz < n ? r[ sa[i-1]+sz ] : -1;
            nr[ sa[i] ] = r[ sa[i] ] == r[ sa[i-1] ] && a == b ? t : ++t;
        }
        if(t == n-1) break;
    }
}

```

```

    memcpy(r, nr, sizeof(int) * n);
}
}

void build_lcp(){ // lcp[i] = lcp(s[:i], s[:i+1])
    int k = 0;
    for(int i = 0; i < n; i++) r[ sa[i] ] = i;

    for(int i = 0; i < n; i++){
        if(r[i] == n-1) k = 0;
        else{
            int j = sa[r[i]+1];
            while(i+k < n && j+k < n && s[i+k] == s[j+k])
                k++;
        }
        lcp[r[i]] = k;
        if(k) k--;
    }
}

int comp_lcp(int i, int j){
    if(i == j) return n - i;
    if(r[i] > r[j]) swap(i, j);
    return min(lcp[k] for k in [r[i], r[j]-1]);
}

```

Z Algorithm

```

vector<int> z_algo(const string &s) {
    int n = s.size(), L = 0, R = 0;
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= R) z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
            z[i]++;
        if(i+z[i]-1 > R) L = i, R = i + z[i] - 1;
    }
    return z;
}

```

Prefix function/KMP

```

vector<int> prefix_function(const string &s){
    int n = s.size();
    vector<int> b(n+1);
    b[0] = -1;
    int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}

void kmp(const string &t, const string &p){
    vector<int> b = prefix_function(p);
    int n = t.size(), m = p.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
        j++;
        if(j == m){
            //patern of p found on t
            j = b[j];
        }
    }
}

```

Min rotation

```

int min_rotation(int *s, int N) {
    REP(i, N) s[N+i] = s[i];

    int a = 0;
    REP(b, N) REP(i, N) {
        if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1);
            break; }
        if (s[a+i] > s[b+i]) { a = b; break; }
    }
    return a;
}

```

All palindrome

```

void manacher(char *s, int N, int *rad) {
    static char t[2*MAX];
    int m = 2*N - 1;

    REP(i, m) t[i] = -1;
    REP(i, N) t[2*i] = s[i];

    int x = 0;
    FOR(i, 1, m) {
        int &r = rad[i] = 0;
        if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
        while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r+1]) ++r;
        if (i+r >= x+rad[x]) x = i;
    }

    REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
    REP(i, m) rad[i] /= 2;
}

```

Palindromic Tree

```

const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root with len 0
int suff; // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while(true){
        curlen = tree[cur].len;
        if (pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }
}

```

```

num++;
suff = num;
tree[num].len = tree[cur].len + 2;
tree[cur].next[let] = num;

if (tree[num].len == 1){
    tree[num].sufflink = 2;
    tree[num].num = 1;
    return true;
}

while (true){
    cur = tree[cur].sufflink;
    curlen = tree[cur].len;
    if(pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[
        pos]){
        tree[num].sufflink = tree[cur].next[let];
        break;
    }
}

tree[num].num = 1 + tree[tree[num].sufflink].num;

return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
    }

    return 0;
}

```

Suffix Automaton

```

map<char, int> to[2*N];
int link[2*N], len[2*N], last = 0, sz = 1;

void add_letter(char c){
    int p = last;
    last = sz++;
    len[last] = len[p] + 1;
    for(; !to[p][c]; p = link[p]) to[p][c] = last;
    if(to[p][c] == last){
        link[last] = 0;
        return;
    }
    int u = to[p][c];
    if(len[u] == len[p]+1){
        link[last] = u;
        return;
    }
    int c1 = sz++;
    to[c1] = to[u];
    link[c1] = link[u];
    len[c1] = len[p]+1;
    link[last] = link[u] = c1;
}

```

```

for(; to[p][c] == u; p = link[p]) to[p][c] = c1;
}

```

Suffix Tree

```

namespace sf {
// const int NS = ; const int N = * 2;
int cn, cd, ns, en = 1, lst;
string S[NS]; int si = -1;
vector<int> sufn[N]; // sufn[si][i] no do sufixo S[si][i
...]
struct node {
    int l, r, si, p, suf;
    map<char, int> adj;
    node() : l(0), r(-1), suf(0), p(0) {}
    node(int L, int R, int S, int P) : l(L), r(R), si(S)
        , p(P) {}
    inline int len() { return r - l + 1; }
    inline int operator[](int i) { return S[si][l + i]; }
    inline int& operator()(char c) { return adj[c]; }
} t[N];
inline int new_node(int L, int R, int S, int P) { t[en]
    = node(L, R, S, P); return en++; }
void add_string(string s) {
    s += '$'; S[++si] = s; sufn[si].resize(s.size() + 1)
        ; cn = cd = 0;
    int i = 0; const int n = s.size();
    for(int j = 0; j < n; j++)
        for(; i <= j; i++) {
            if(cd == t[cn].len() && t[cn](s[j])) { cn = t
                [cn](s[j]); cd = 0; }
            if(cd < t[cn].len() && t[cn][cd] == s[j]) {
                cd++;
                if(j < s.size() - 1) break;
            }
            else {
                if(i) t[lst].suf = cn;
                for(; i <= j; i++) { sufn[si][i] = cn;
                    cn = t[cn].suf; }
            }
        }
    else if(cd == t[cn].len()) {
        sufn[si][i] = en;
        if(i) t[lst].suf = en; lst = en;
        t[cn](s[j]) = new_node(j, n - 1, si, cn);
        cn = t[cn].suf; cd = t[cn].len();
    }
    else {
        int mid = new_node(t[cn].l, t[cn].l + cd -
            1, t[cn].si, t[cn].p);
        t[t[cn].p](t[cn][0]) = mid;
        if(ns) t[ns].suf = mid;
        if(i) t[lst].suf = en; lst = en;
        sufn[si][i] = en;
        t[mid](s[j]) = new_node(j, n - 1, si, mid)
            ;
        t[mid](t[cn][cd]) = cn;
        t[cn].p = mid; t[cn].l += cd; cn = t[mid].
            p;
        int g = cn? j - cd : i + 1; cn = t[cn].suf
            ;
        while(g < j && g + t[t[cn](S[si][g])].len
            () <= j) {
            cn = t[cn](S[si][g]); g += t[cn].len();
        }
        if(g == j) { ns = 0; t[mid].suf = cn; cd =
            t[cn].len(); }
        else { ns = mid; cn = t[cn](S[si][g]); cd
            = j - g; }
    }
}
}

```

```

    }
}
};

```

Geometry

2D basics

```

typedef double coord;
double eps = 1e-7;
bool eq(coord a, coord b){ return abs(a - b) <= eps; }

struct vec{
    coord x, y; int id;
    vec(coord a = 0, coord b = 0) : x(a), y(b) {}
    vec operator+(const vec &o) const{
        return {x + o.x, y + o.y};
    }
    vec operator-(const vec &o) const{
        return {x - o.x, y - o.y};
    }
    vec operator*(coord t) const{
        return {x * t, y * t};
    }
    vec operator/(coord t) const{
        return {x / t, y / t};
    }
    coord operator*(const vec &o) const{ // cos
        return x * o.x + y * o.y;
    }
    coord operator^(const vec &o) const{ // sin
        return x * o.y - y * o.x;
    }
    bool operator==(const vec &o) const{
        return eq(x, o.x) && eq(y, o.y);
    }
    bool operator<(const vec &o) const{
        if(!eq(x, o.x)) return x < o.x;
        return y < o.y;
    }
    coord cross(const vec &a, const vec &b) const{
        return (a-(*this)) ^ (b-(*this));
    }
    int ccw(const vec &a, const vec &b) const{
        coord tmp = cross(a, b);
        return (tmp > eps) - (tmp < -eps);
    }
    coord dot(const vec &a, const vec &b) const{
        return (a-(*this)) * (b-(*this));
    }
    coord len() const{
        return sqrt(x * x + y * y); // <
    }
    double angle(const vec &a, const vec &b) const{
        return atan2(cross(a, b), dot(a, b));
    }
    double tan(const vec &a, const vec &b) const{
        return cross(a, b) / dot(a, b);
    }
    vec unit() const{
        return operator/(len());
    }
    int quad() const{
        if(x > 0 && y >=0) return 0;
        if(x <=0 && y > 0) return 1;
        if(x < 0 && y <=0) return 2;
        return 3;
    }
}

```

```

bool comp(const vec &a, const vec &b) const{
    return (a - *this).comp(b - *this);
}

bool comp(vec b){
    if(quad() != b.quad()) return quad() < b.quad();
    if(!eq(operator^(b), 0)) return operator^(b) > 0;
    return (*this) * (*this) < b * b;
}

template<class T>
void sort_by_angle(T first, T last) const{
    std::sort(first, last, [=](const vec &a, const
        vec &b){
            return comp(a, b);
        });
}

vec rot90() const{ return {-y, x}; }
vec rot(double a) const{
    return {cos(a)*x -sin(a)*y, sin(a)*x +cos(a)*y};
}

struct line{
    coord a, b, c; vec n;
    line(vec q, vec w){ // q.cross(w, (x, y)) = 0
        a = -(w.y-q.y);
        b = w.x-q.x;
        c = -(a * q.x + b * q.y);
        n = {a, b};
    }
    coord dist(const vec &o) const{
        return abs(eval(o)) / n.len();
    }
    bool contains(const vec &o) const{
        return eq(a * o.x + b * o.y + c, 0);
    }
    coord dist(const line &o) const{
        if(!parallel(o)) return 0;
        if(!eq(o.a * b, o.b * a)) return 0;
        if(!eq(a, 0))
            return abs(c - o.c * a / o.a) / n.len();
        if(!eq(b, 0))
            return abs(c - o.c * b / o.b) / n.len();
        return abs(c - o.c);
    }
    bool parallel(const line &o) const{
        return eq(n ^ o.n, 0);
    }
    bool operator==(const line &o) const{
        if(!eq(a*o.b, b*o.a)) return false;
        if(!eq(a*o.c, c*o.a)) return false;
        if(!eq(c*o.b, b*o.c)) return false;
        return true;
    }
    bool intersect(const line &o) const{
        return !parallel(o) || *this == o;
    }
    vec inter(const line &o) const{
        if(parallel(o)){
            if(*this == o){ }
            else{ /* dont intersect */ }
        }

        auto tmp = n ^ o.n;
        return {(o.c*b -c*o.b)/tmp, (o.a*c -a*o.c)/tmp};
    }
    vec at_x(coord x) const{
        return {x, (-c-a*x)/b};
    }
}

```

```

    }
    vec at_y(coord y) const{
        return {(-c-b*y)/a, y};
    }
    coord eval(const vec &o) const{
        return a * o.x + b * o.y + c;
    }
};

struct segment{
    vec p, q;
    segment(vec a = vec(), vec b = vec()): p(a), q(b) {}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(o, q) >= -eps && q.dot(o, p) >= -eps;
    }
    coord len() const{
        return (p-q).len();
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return min((o-q).len(), (o-p).len());
    }
    bool contains(const vec &o) const{
        return eq(p.cross(q, o), 0) && onstrip(o);
    }
    bool intersect(const segment &o) const{
        if(contains(o.p)) return true;
        if(contains(o.q)) return true;
        if(o.contains(q)) return true;
        if(o.contains(p)) return true;
        return p.ccw(q, o.p) * p.ccw(q, o.q) == -1
        && o.p.ccw(o.q, q) * o.p.ccw(o.q, p) == -1;
    }
    bool intersect(const line &o) const{
        return o.eval(p) * o.eval(q) <= 0;
    }
    coord dist(const segment &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || onstrip(o.q)
            || o.onstrip(p) || o.onstrip(q))
                return line(p, q).dist(line(o.p, o.q));
        }
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
            min(o.dist(p), o.dist(q)));
    }
    coord dist(const line &o) const{
        if(line(p, q).parallel(o))
            return line(p, q).dist(o);
        else if(intersect(o)) return 0;
        return min(o.dist(p), o.dist(q));
    }
};

struct hray{
    vec p, q;
    hray(vec a = vec(), vec b = vec()): p(a), q(b){}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(q, o) >= -eps;
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return (o-p).len();
    }
    bool intersect(const segment &o) const{
        if(!o.intersect(line(p,q))) return false;

```

```

        if(line(o.p, o.q).parallel(line(p,q)))
            return contains(o.p) || contains(o.q);
        return contains(line(p,q).inter(line(o.p,o.q)));
    }
    bool contains(const vec &o) const{
        return eq(line(p, q).eval(o), 0) && onstrip(o);
    }
    coord dist(const segment &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || onstrip(o.q))
                return line(p, q).dist(line(o.p, o.q));
            return o.dist(p);
        }
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
            o.dist(p));
    }
    bool intersect(const hray &o) const{
        if(!line(p, q).parallel(line(o.p, o.q)))
            return false;
        auto pt = line(p, q).inter(line(o.p, o.q));
        return contains(pt) && o.contains(pt); // <<
    }
    bool intersect(const line &o) const{
        if(line(p, q).parallel(o)) return line(p, q) == o;
        if(o.contains(p) || o.contains(q)) return true;
        return (o.eval(p) >= -eps)^(o.eval(p)<o.eval(q));
        return contains(o.inter(line(p, q)));
    }
    coord dist(const line &o) const{
        if(line(p,q).parallel(o))
            return line(p,q).dist(o);
        else if(intersect(o)) return 0;
        return o.dist(p);
    }
    coord dist(const hray &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || o.onstrip(p))
                return line(p,q).dist(line(o.p, o.q));
            return (p-o.p).len();
        }
        else if(intersect(o)) return 0;
        return min(dist(o.p), o.dist(p));
    }
};

double heron(coord a, coord b, coord c){
    coord s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}

```

Nearest Points

```

struct pt {
    int x, y, id;
};

inline bool cmp_x (const pt &a, const pt &b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

inline bool cmp_y (const pt &a, const pt &b) {
    return a.y < b.y;
}

pt a[MAXN];

double mindist;

```

```

int ansa, ansb;

inline void upd_ans (const pt & a, const pt & b) {
    double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)
        *(a.y-b.y) + .0);
    if (dist < mindist)
        mindist = dist, ansa = a.id, ansb = b.id;
}

void rec (int l, int r) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans (a[i], a[j]);
        sort (a+l, a+r+1, &cmp_y);
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec (l, m), rec (m+1, r);
    static pt t[MAXN];
    merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
    copy (t, t+r-1+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (abs (a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y <
                mindist; --j)
                upd_ans (a[i], t[j]);
            t[tsz++] = a[i];
        }
}

sort (a, a+n, &cmp_x);
mindist = 1E20;
rec (0, n-1);

```

Convex Hull

```

vector<vec> monotone_chain_ch(vector<vec> P){
    sort(P.begin(), P.end());

    vector<vec> L, U;
    for(auto p : P){
        while(L.size() >= 2 && L[L.size() - 2].cross(L.
            back(), p) < 0)
            L.pop_back();

        L.push_back(p);
    }

    reverse(P.begin(), P.end());
    for(auto p : P){
        while(U.size() >= 2 && U[U.size() - 2].cross(U.
            back(), p) < 0)
            U.pop_back();

        U.push_back(p);
    }

    L.pop_back(), U.pop_back();

    L.reserve(L.size() + U.size());
    L.insert(L.end(), U.begin(), U.end());

    return L;
}

```

```

}

```

Check point inside polygon

```

bool below(const vector<vec> &vet, vec p){
    auto it = lower_bound(vet.begin(), vet.end(), p);
    if(it == vet.end()) return false;
    if(it == vet.begin()) return *it == p;
    return prev(it)->cross(*it, p) <= 0;
}

bool above(const vector<vec> &vet, vec p){
    auto it = lower_bound(vet.begin(), vet.end(), p);
    if(it == vet.end()) return false;
    if(it == vet.begin()) return *it == p;
    return prev(it)->cross(*it, p) >= 0;
}

// lowerhull, upperhull and point, borders included
bool inside_poly(const vector<vec> &lo, const vector<vec>
    > &hi, vec p){
    return below(hi, p) && above(lo, p);
}

```

Check point inside polygon without lower/upper hull

```

// borders included
// must not have 3 colinear consecutive points
bool inside_poly(const vector<vec> &v, vec p){
    if(v[0].ccw(v[1], p) < 0) return false;
    if(v[0].ccw(v.back(), p) > 0) return 0;
    if(v[0].ccw(v.back(), p) == 0)
        return v[0].dot(p, v.back()) >= 0
            && v.back().dot(p, v[0]) >= 0;

    int L = 1, R = (int)v.size() - 1, ans = 1;

    while(L <= R){
        int mid = (L+R)/2;
        if(v[0].ccw(v[mid], p) >= 0) ans = mid, L = mid
            +1;
        else R = mid-1;
    }

    return v[ans].ccw(v[(ans+1)%v.size()], p) >= 0;
}

```

Minkowski sum

```

vector<vec> mk(const vector<vec> &a, const vector<vec> &b){
    int i = 0, j = 0;
    for(int k = 0; k < (int)a.size(); k++){
        if(a[k] < a[i])
            i = k;
        for(int k = 0; k < (int)b.size(); k++){
            if(b[k] < b[j])
                j = k;
        }

        vector<vec> c;
        c.reserve(a.size() + b.size());
        for(int k = 0; k < int(a.size()+b.size()); k++){
            vec pt{a[i] + b[j]};
            if((int)c.size() >= 2
                && c[c.size()-2].ccw(c.back(), pt) == 0)
                c.pop_back();
            c.push_back(pt);
            int q = i+1, w = j+1;
            if(q == int(a.size())) q = 0;
            if(w == int(b.size())) w = 0;
            if(c.back().ccw(a[i]+b[w], a[q]+b[j]) < 0) i = q;
        }
    }
}

```



```

        else j = w;
    }
    c.shrink_to_fit();

    return c;
}

```

Miscellaneous

LIS

```

multiset<int> S;
for(int i = 0; i < n; i++){
    auto it = S.upper_bound(a[i]); // low for inc
    if(it != S.end()) S.erase(it);
    S.insert(a[i]);
}
ans = S.size();

```

DSU rollback

```

#include <bits/stdc++.h>

using namespace std;

struct DSU{
    vector<int> sz, p, change;
    vector<tuple<int, int, int>> modifications;
    vector<size_t> saves;
    bool bipartite;

    DSU(int n): sz(n+1, 1), p(n+1), change(n+1),
        bipartite(true){
        iota(p.begin(), p.end(), 0);
    }

    void add_edge(int u, int v){
        if(!bipartite) return;
        int must_change = get_colour(u) == get_colour(v);
        int a = rep(u), b = rep(v);
        if(sz[a] < sz[b]) swap(a, b);
        if(a != b){
            p[b] = a;
            modifications.emplace_back(b, change[b],
                bipartite);
            change[b] ^= must_change;
            sz[a] += sz[b];
        }
        else if(must_change){
            modifications.emplace_back(0, change[0],
                bipartite);
            bipartite = false;
        }
    }

    int rep(int u){
        return p[u] == u ? u : rep(p[u]);
    }

    int get_colour(int u){
        if(p[u] == u) return change[u];
        return change[u] ^ get_colour(p[u]);
    }

    void reset(){
        modifications.clear();
        saves.clear();
        iota(p.begin(), p.end(), 0);
        fill(sz.begin(), sz.end(), 1);

```

```

        fill(change.begin(), change.end(), 0);
        bipartite = true;
    }

    void rollback(){
        int u = get<0>(modifications.back());
        tie(ignore, change[u], bipartite) = modifications
            .back();
        sz[ p[u] ] -= sz[u];
        p[u] = u;
        modifications.pop_back();
    }

    void reload(){
        while(modifications.size() > saves.back())
            rollback();
        saves.pop_back();
    }

    void save(){
        saves.push_back(modifications.size());
    }
};

const int N = 100005;
const int B = 318;

int n, m, q;
int x[N], y[N], l[N], r[N], ans[N];

vector<int> qu[N];

int brute(int lef, int rig, DSU &s){
    s.save();
    for(int i = lef; i <= rig; i++)
        s.add_edge(x[i], y[i]);
    int ret = s.bipartite;
    s.reload();
    return ret;
}

int main(){
    scanf("%d %d %d", &n, &m, &q);

    for(int i = 1; i <= m; i++)
        scanf("%d %d", x+i, y+i);

    DSU s(n);
    for(int i = 0; i < q; i++){
        scanf("%d %d", l+i, r+i);
        if(r[i] - l[i] <= B + 10)
            ans[i] = brute(l[i], r[i], s);
        else qu[l[i] / B].push_back(i);
    }

    for(int i = 0; i <= m / B; i++){
        sort(qu[i].begin(), qu[i].end(), [](int a, int b){
            return r[a] < r[b];
        });
        s.reset();

        int R = (i+1)*B-1;

        for(int id : qu[i]){
            while(R < r[id]) ++R, s.add_edge(x[R], y[R]);
            s.save();

```

```

        for(int k = l[id]; k < (i+1)*B; k++)
            s.add_edge(x[k], y[k]);
        ans[id] = s.bipartite;
        s.reload();
    }
}

for(int i = 0; i < q; i++)
    printf("%s\n", ans[i] ? "Possible" : "Impossible");
}

```

Buildings

```

// count the number of circular arrays
// of size m, with elements on range
// [1, c**(x*x)]

```

```

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

#define debug(x) cerr << fixed << #x << " = " << x << endl;
#define ll long long

const int MOD = 1e9 + 7;
const int MAX = 1e5 + 5;
int dp[MAX];

inline int add(int a, int b) {
    a += b;
    if(a >= MOD) {
        a -= MOD;
    }
    return a;
}

inline int sub(int a, int b) {
    a -= b;
    if(0 > a) {
        a += MOD;
    }
    return a;
}

inline int mult(int a, int b) {
    return (1LL * a * b) % MOD;
}

int f_exp(int x, int exp) {
    if(exp == 0) {
        return 1;
    }
    else if(exp & 1) {
        return mult(x, f_exp(x, exp - 1));
    }
    return f_exp(mult(x, x), exp / 2);
}

inline int inv(int x) {
    return f_exp(x, MOD - 2);
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);

    int n, m, c;

```

```

    cin >> n >> m >> c;

    int x = f_exp(c, n * n);
    int ans = f_exp(x, m);
    for(int i = 1; i <= m; i++) {
        if(m % i == 0) {
            int y = f_exp(x, i);
            for(int j = 1; j < i; j++) {
                if(i % j == 0) {
                    y = sub(y, mult(j, dp[j]));
                }
            }
            dp[i] = mult(y, inv(i));
            ans = sub(ans, mult(i - 1, dp[i]));
        }
    }

    cout << ans << '\n';

    return 0;
}

```

Rand

```

cout << RAND_MAX << endl;
mt19937 rng(chrono::steady_clock::now().time_since_epoch
().count());
vector<int> permutation(N);

iota(permutation.begin(), permutation.end(), 0);

shuffle(permutation.begin(), permutation.end(), rng);

iota(permutation.begin(), permutation.end(), 0);

for(int i = 1; i < N; i++){
    swap(permutation[i], permutation[
        uniform_int_distribution<int>(0, i)(rng)]);
}

```

Klondike

```

// minimum number of moves to make
// all elements equal
// move: change a segment of equal value
// elements to any value

int v[305];
int dp[305][305];
int rec[305][305];

int f(int l, int r){
    if(r == l) return 1;
    if(r < l) return 0;
    if(dp[l][r] != -1) return dp[l][r];
    int ans = f(l+1, r) + 1;
    for(int i = l+1; i <= r; i++){
        if(v[i] == v[l])
            ans = min(ans, f(l, i - 1) + f(i+1, r));
    }

    return dp[l][r] = ans;
}

int main() {
    int n, m;
    memset(dp, -1, sizeof dp);
    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++){

```

```
scanf("%d",v+i);
if(i && v[i] == v[i-1]){
    i--;
    n--;
}
}
printf("%d\n",f(0, n-1) - 1);
// printf("%d\n",rec[0][n-1] );
// printf("%d\n",rec[1][n-1] );
// printf("%d\n",rec[2][n-3] );
}
```

Hilbert Order

```
// maybe use B = n / sqrt(q)
inline int64_t hilbertOrder(int x, int y, int pow = 21,
    int rotate = 0) {
    if(pow == 0) return 0;
    int hpow = 1 << (pow-1);
    int seg = (x < hpow) ? (
        (y < hpow) ? 0 : 3
    ) : (
        (y < hpow) ? 1 : 2
    );
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    int64_t subSquareSize = int64_t(1) << (2*pow - 2);
    int64_t ans = seg * subSquareSize;
    int64_t add = hilbertOrder(nx, ny, pow-1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize
        - add - 1);
    return ans;
}
```

Modular Factorial

```
// Compute (1*2*...*(p-1)*1*(p+1)*(p+2)*...*n) % p
// in O(p*lg(n))

int factmod(int n, int p){
    int ans = 1;
    while(n > 1){
        for(int i = 2; i <= n % p; i++){
            ans = (ans * i) % p;
            n /= p;
            if(n % 2) ans = p - ans;
        }
        return ans % p;
    }

    int fac_pow(int n, int p){
        int ans = 0;
        while(n) n /= p, ans += n;
        return ans;
    }

    int C(int n, int k, int p){
        if(fac_pow(n, p) > fac_pow(n-k, p) + fac_pow(k, p))
            return 0;
        int tmp = factmod(k, p) * factmod(n-k, p);
        return (f_exp(tmp, p-2, p) * factmod(n, p)) % p;
    }
}
```

Enumerating all submasks of a bitmask

```
// loop through all submask of a given bitmask
// it does not include mask 0
for(int sub = mask; sub; sub = (sub-1)&mask){
```

Slope Trick

///By woqjal25, contest: Codeforces Round #371 (Div. 1),
problem: (C) Sonya and Problem Wihtout a Legend,
Accepted, #

```
#include<stdio.h>
#include<queue>

int main()
{
    int n, t;
    long long ans = 0;
    std::priority_queue<int> Q;
    scanf("%d%d", &n, &t);
    Q.push(t);
    for(int i=1; i<n; i++)
    {
        scanf("%d", &t); t-=i;
        Q.push(t);
        if(Q.top() > t)
        {
            ans += Q.top() - t;
            Q.pop();
            Q.push(t);
        }
    }
    printf("%lld", ans);
    return 0;
}
```

Fast IO

```
#define pc(x) putchar_unlocked(x)
#define gc(x) getchar_unlocked(x)

inline void scan_int(int &x){
    register int c = gc();
    x = 0;
    int neg = 0;
    for(;; ((c < '0' || c > '9') && c != '-'); c = gc());
    if(c == '-'){
        neg = 1;
        c = gc();
    }
    for(; c >= '0' && c <= '9'; c = gc())
        x = (x << 1) + (x << 3) + c - '0';
    if(neg) x = -x;
}

inline void print_int(int n){
    int rev = 0, count = 0, neg;
    if(n == 0){
        pc('0');
        return;
    }
    if(n < 0) n = -n, neg = 1;
    while(n % 10 == 0) count++, n /= 10;
    for(rev = 0; n != 0; n /= 10)
        rev = (rev << 3) + (rev << 1) + n % 10;
    if(neg) pc('-');
    while(rev != 0) pc(rev % 10 + '0'), rev /= 10;
    while(count-->0) pc('0');
    pc('\n');
}
```

```
inline void print_string(char *str){
    while(*str) pc(*str++);
    pc('\n');
}
```

Big int

```
/*
#####
##### THE BIG INT
#####
*/
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;
    /*<arpa>*/
    int size(){
        if(a.empty())return 0;
        int ans=(a.size()-1)*base_digits;
        int ca=a.back();
        while(ca)
            ans++,ca/=10;
        return ans;
    }
    bigint operator^(const bigint &v){
        bigint ans=1,a=*this,b=v;
        while(!b.isZero()){
            if(b%2)
                ans*=a;
            a*=a,b/=2;
        }
        return ans;
    }
    string to_string(){
        stringstream ss;
        ss << *this;
        string s;
        ss >> s;
        return s;
    }
    int sumof(){
        string s = to_string();
        int ans = 0;
        for(auto c : s) ans += c - '0';
        return ans;
    }
    /*</arpa>*/
    bigint() :
    sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;

```

```
    a.clear();
    if (v < 0)
        sign = -1, v = -v;
    for (; v > 0; v = v / base)
        a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;

        for (int i = 0, carry = 0; i < (int) max(a.size()
            , v.a.size()) || carry; ++i) {
            if (i == (int) res.a.size())
                res.a.push_back(0);
            res.a[i] += carry + (i < (int) a.size() ? a[i] :
                0);
            carry = res.a[i] >= base;
            if (carry)
                res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int) v.a.size()
                || carry; ++i) {
                res.a[i] -= carry + (i < (int) v.a.size() ? v
                    .a[i] : 0);
                carry = res.a[i] < 0;
                if (carry)
                    res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
    }

    void operator*=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() ||
        carry; ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / base);
        a[i] = (int) (cur % base);
        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"
            "(cur)", "c"(base));
    }
    trim();
    }

    bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
    }

```

```

void operator*=(long long v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) a.size() ||
        carry; ++i) {
        if (i == (int) a.size())
            a.push_back(0);
        long long cur = a[i] * (long long) v + carry;
        carry = (int) (cur / base);
        a[i] = (int) (cur % base);
        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"
            "(cur)", "c"(base));
    }
    trim();
}

bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1,
    const bigint &b1) {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());

    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.
            size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.
            a.size() - 1];
        int d = ((long long) base * s1 + s2) / b.a.back()
            ;
        r -= b * d;
        while (r < 0)
            r += b, --d;
        q.a[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}

bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = (int) a.size() - 1, rem = 0; i >= 0; --
        i) {
        long long cur = a[i] + rem * (long long) base;

        a[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
        m = (a[i] + m * (long long) base) % v;
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}

void operator-=(const bigint &v) {
    *this = *this - v;
}

void operator*=(const bigint &v) {
    *this = *this * v;
}

void operator/=(const bigint &v) {
    *this = *this / v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (a.size() != v.a.size())
        return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
        if (a[i] != v.a[i])
            return a[i] * sign < v.a[i] * sign;
    return false;
}

bool operator>(const bigint &v) const {
    return v < *this;
}

bool operator<=(const bigint &v) const {
    return !(v < *this);
}

bool operator>=(const bigint &v) const {
    return !(*this < v);
}

bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}

bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

void trim() {
    while (!a.empty() && !a.back())
        a.pop_back();
    if (a.empty())
        sign = 1;
}

```

```

bool isZero() const {
return a.empty() || (a.size() == 1 && !a[0]);
}

bigint operator-() const {
bigint res = *this;
res.sign = -sign;
return res;
}

bigint abs() const {
bigint res = *this;
res.sign *= res.sign;
return res;
}

long long longValue() const {
long long res = 0;
for (int i = a.size() - 1; i >= 0; i--)
    res = res * base + a[i];
return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b)
{
return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b)
{
return a / gcd(a, b) * b;
}

void read(const string &s) {
sign = 1;
a.clear();
int pos = 0;
while (pos < (int) s.size() && (s[pos] == '-' || s[
    pos] == '+')) {
    if (s[pos] == '-')
        sign = -sign;
    ++pos;
}
for (int i = s.size() - 1; i >= pos; i -=
    base_digits) {
    int x = 0;
    for (int j = max(pos, i - base_digits + 1); j <=
        i; j++)
        x = x * 10 + s[j] - '0';
    a.push_back(x);
}
trim();
}

friend istream& operator>>(istream &stream, bigint &
    v) {
string s;
stream >> s;
v.read(s);
return stream;
}

friend ostream& operator<<(ostream &stream, const
    bigint &v) {
if (v.sign == -1)
    stream << '-';
stream << (v.a.empty() ? 0 : v.a.back());
}

```

```

for (int i = (int) v.a.size() - 2; i >= 0; --i)
    stream << setw(base_digits) << setfill('0') << v.
        a[i];
return stream;
}

static vector<int> convert_base(const vector<int> &a
    , int old_digits, int new_digits) {
vector<long long> p(max(old_digits, new_digits) + 1)
    ;
p[0] = 1;
for (int i = 1; i < (int) p.size(); i++)
    p[i] = p[i - 1] * 10;
vector<int> res;
long long cur = 0;
int cur_digits = 0;
for (int i = 0; i < (int) a.size(); i++) {
    cur += a[i] * p[cur_digits];
    cur_digits += old_digits;
    while (cur_digits >= new_digits) {
        res.push_back((int)(cur % p[new_digits]));
        cur /= p[new_digits];
        cur_digits -= new_digits;
    }
}
res.push_back((int) cur);
while (!res.empty() && !res.back())
    res.pop_back();
return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll
    &b) {
int n = a.size();
vll res(n + n);
if (n <= 32) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[i + j] += a[i] * b[j];
    return res;
}

int k = n >> 1;
vll a1(a.begin(), a.begin() + k);
vll a2(a.begin() + k, a.end());
vll b1(b.begin(), b.begin() + k);
vll b2(b.begin() + k, b.end());

vll a1b1 = karatsubaMultiply(a1, b1);
vll a2b2 = karatsubaMultiply(a2, b2);

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

vll r = karatsubaMultiply(a2, b2);
for (int i = 0; i < (int) a1b1.size(); i++)
    r[i] -= a1b1[i];
for (int i = 0; i < (int) a2b2.size(); i++)
    r[i] -= a2b2[i];

for (int i = 0; i < (int) r.size(); i++)
    res[i + k] += r[i];
for (int i = 0; i < (int) a1b1.size(); i++)

```

```

    res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits,
        6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
        a.push_back(0);
    while (b.size() < a.size())
        b.push_back(0);
    while (a.size() & (a.size() - 1))
        a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i++)
    {
        long long cur = c[i] + carry;
        res.a.push_back((int) (cur % 10000000));
        carry = (int) (cur / 10000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
}
};
/*
##### THE BIG INT #####
#####
*/

```

Knapsack Bounded with Cost

```

// menor custo para conseguir peso ate M usando N tipos
// diferentes de elementos, sendo que o i-esimo elemento
// pode ser usado b[i] vezes, tem peso w[i] e custo c[i]
// O(N * M)

```

```

int b[N], w[N], c[N];
MinQueue Q[M];
int d[M] //d[i] = custo minimo para conseguir peso i

for(int i = 0; i <= M; i++) d[i] = i ? oo : 0;
for(int i = 0; i < N; i++){
    for(int j = 0; j < w[i]; j++){
        Q[j].clear();
        for(int j = 0; j <= M; j++){
            q = Q[j % w[i]];

```

```

        if(q.size() >= q) q.pop();
        q.add(c[i]);
        q.push(d[j]);
        d[j] = q.getmin();
    }
}

```

Burnside's Lemma

Let (G, \oplus) be a finite group that acts on a set X . It should hold that $e_g * x = x$ and $g_1 * (g_2 * x) = (g_1 \oplus g_2) * x, \forall x \in X, g_1, g_2 \in G$. For each $g \in G$ let $X^g = \{x \in X \mid g * x = x\}$. The number of orbits is given by:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

Wilson's Theorem

$(n-1)! \equiv -1 \pmod n \iff n$ is prime

Fibonacci

- $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$
- $F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$
- $GCD(F_n, F_m) = F_{GCD(n,m)}$
- $F_n = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$

Edge coloring

Data: A graph G

Result: A proper coloring c of the edges of G

Let $U := E(G)$;

while $U \neq \emptyset$ **do**

 Let (u,v) be any edge in U ;

 Let $F[1:k]$ be a maximal fan of u starting at $F[1]=v$;

 Let c be a color that is free on u and d be a color that is free on $F[k]$;

 Invert the cd_u path;

 Let $w \in V(G)$ be such that $w \in F, F' = [F[1]...w]$ is a fan and d is free on w ;

 Rotate F' and set $c(u,w)=d$;

$U := U - (u,v)$;

end

Notes

When we repeat something and each time we have probability p to succeed then the expected number of tries is $\frac{1}{p}$, till we succeed.