# Rock Lee do Pagode Namora D+

University of Brasilia

## Contents

```
set ts=4 sw=4 sta nu rnu sc stl+=%F cindent
set bg=dark ruler timeoutlen=1000
imap {<CR> {<CR>}<Esc>O
nmap <F2> 0V$%d
nmap <C-down> :m+1<CR>
nmap <C-up> :m-2<CR>
nmap <C-a> ggVG
nmap <S-up> :m-2<CR>
nmap <S-down> :m+1<CR>
syntax on

vmap <C-c> "+y
set viminfo='20,\"1000
```

```
alias comp='g++ -std=c++17 -Wshadow -Wall -Wextra -Wformat=2 -Wconversion -fsanitize=address,undefined -fno-sanitize-recover -Wfatal-errors'
```

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

#define ff first
#define ss second
#define pb push_back

using namespace std;
using ll = long long;
using ii = pair<int, int>;

const int N = 100005;

int main() {

	return 0;
}
```

# 1 Data Structures

## 1.1 Fenwick Tree 2D

```cpp
vector<int> go[N];
vector<int> ft[N];

void prec_add(int x, int y) {
	for(; x < N; x += x & -x) {
		go[x].push_back(y);
	}
}
void init() {
	for(int i = 1; i < N; i++) {
		sort(go[i].begin(), go[i].end());
		go[i].resize(unique(go[i].begin(), go[i].end()) - go[i].begin());
		ft[i].assign(go[i].size() + 1, 0);
	}
}
void add(int x, int y, int val) {
	for(; x < N; x += x & -x) {
		int id = int(upper_bound(go[x].begin(), go[x].end(), y) - go[x].begin());
		for(; id < (int)ft[x].size(); id += id & -id)
			ft[x][id] += val;
	}
}
int sum(int x, int y) {
	int ans = 0;
	for(; x > 0; x -= x & -x) {
		int id = int(upper_bound(go[x].begin(), go[x].end(), y) - go[x].begin());
		for(; id > 0; id -= id & -id)
			ans += ft[x][id];
	}
	return ans;
}
```

## 1.2 Wavelet Tree

```cpp
template<typename T>
class wavelet { // 1-based!!
    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) { // it changes the argument array
		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]
};
```

## 1.3 Order Set

```cpp
#include <bits/extc++.h>

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
```

## 1.4 Hash table

```cpp
#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);
```

## 1.5 Convex Hull Trick Simple

```cpp
struct Line{
    ll m, b;
    inline ll eval(ll x) const{
        return x * m + b;
    }
};

// min => cht.back().m >= L.m
// max => cht.back().m <= L.m
void push_line(vector<Line> &cht, Line L){
  while((int)cht.size() >= 2){
    int sz = (int)cht.size();
    if((long double)(L.b-cht[sz-1].b)*(cht[sz-2].m-L.m)
   <= (long double)(L.b-cht[sz-2].b)*(cht[sz-1].m-L.m)){
      cht.pop_back();
    }
    else break;
  }
  cht.push_back(L);
}

// x increasing; pos = 0 in first call
ll linear_search(const vector<Line> &cht,ll x,int &pos){
```

```
    while(pos+1 < (int)cht.size()){
/*>>*/ if(cht[pos].eval(x) >= cht[pos+1].eval(x)) pos++;
        else break;
    }
    return cht[pos].eval(x);
}

ll binary_search(const vector<Line> &cht, ll x){
    int L = 0, R = (int)cht.size()-2;
    int bans = (int)cht.size()-1;
    while(L <= R){
        int mid = (L+R)/2;
        if(cht[mid].eval(x) >= cht[mid+1].eval(x)) // <<<
            L = mid + 1;
        else bans = mid, R = mid - 1;
    }
    return cht[bans].eval(x);
}
```

## 1.6 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 (long double)(x->b - y->b)*(z->m - y->m) >= (long
    ↪double)(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;
▷ }
};
```

## 1.7 Convex Hull Trick

```
/**
 * Author: Simon Lindholm
 * source:
    ↪https://github.com/kth-competitive-programming/kactl/blob/master/content/data-structures/LineContainer.h
 * License: CC0
 */

struct Line {
▷ mutable ll m, b, p;
▷ bool operator<(const Line& o) const { return m < o.m; }
▷ bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> { // CPP14 only
▷ // (for doubles, use inf = 1/.0, div(a,b) = a/b)
▷ const ll inf = LLONG_MAX;
▷ ll div(ll a, ll b) { // floored division
▷ ▷ return a / b - ((a ^ b) < 0 && a % b); }
▷ bool isect(iterator x, iterator y) {
▷ ▷ if (y == end()) { x->p = inf; return false; }
▷ ▷ if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
▷ ▷ else x->p = div(y->b - x->b, x->m - y->m);
▷ ▷ return x->p >= y->p;
▷ }
▷ void add(ll m, ll b) {
▷ ▷ auto z = insert({m, b, 0}), y = z++, x = y;
▷ ▷ while (isect(y, z)) z = erase(z);
▷ ▷ if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
▷ ▷ while ((y = x) != begin() && (--x)->p >= y->p)
▷ ▷ ▷ isect(x, erase(y));
▷ }
▷ ll query(ll x) {
▷ ▷ assert(!empty());
▷ ▷ auto l = *lower_bound(x);
▷ ▷ return l.m * x + l.b;
▷ }
};
```

## 1.8 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()); }
};
```

## 1.9 Sparse Table

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

## 1.10 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
▷ if(!u) return;
▷ 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);
}
void insert(int &u, int it){
▷ unlaze(u);
▷ if(!u) u = it;
▷ else if(Y[it] > Y[u]) split_val(u, X[it], L[it], R[it]), u = it;
▷ else insert(X[it] < X[u] ? L[u] : R[u], it);
▷ calc(u);
}
void erase(int &u, num key){
▷ unlaze(u);
▷ if(!u) return;
▷ if(X[u] == key) u = merge(L[u], R[u]);
▷ else erase(key < X[u] ? L[u] : R[u], key);
▷ calc(u);
}
int create_node(num key){
▷ X[en] = key;
▷ sz[en] = 1;
▷ L[en] = R[en] = 0;
▷ return en++;
}
int query(int u, int l, int r){//0 index
▷ unlaze(u);
▷ if(u! or r < 0 or l >= sz[u]) return identity_element;
▷ if(l <= 0 and r >= sz[u] - 1) return subt_data[u];
▷ int ans = query(L[u], l, r);
▷ if(l <= sz[ L[u] ] and sz[ L[u] ] <= r)
▷ ▷ ans = max(ans, st[u]);
▷ ans = max(ans, query(R[u], l-sz[L[u]]-1, r-sz[L[u]]-1));
```

```
▷ return ans;
}
```

## 1.11 ColorUpdate

```
// source:
    ↪https://github.com/tfg50/Competitive-Programming/tree/master/Biblioteca/Data%20Structures
template <class Info = int>
class ColorUpdate {
▷ set<Range> ranges;
public:
▷ struct Range {
▷ ▷ Range(int a = 0) : l(a) {}
▷ ▷ Range(int a, int b, Info c) : l(a), r(b), v(c) {}
▷ ▷ int l, r;
▷ ▷ Info v;
▷ ▷ bool operator<(const Range &b) const { return l < b.l; }
▷ };
▷ vector<Range> upd(int l, int r, Info v) {
▷ ▷ 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.emplace(cur.l, l, cur.v);
▷ ▷ ▷ ▷ ranges.emplace(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.emplace(cur.l, r, cur.v);
▷ ▷ ▷ ▷ ranges.emplace(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.emplace(l, r, v);
▷ ▷ return ans;
▷ }
};
```

## 1.12 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] = 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] ]
```

## 1.13 Iterative Segtree

```
T query(int l, int r){ // [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 in [0, n)
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]);
}
```

## 1.14 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));
    }
}
```

## 1.15 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;
}
```

# 2 Math

## 2.1 Extended Euclidean Algorithm

```
// a*x + b*y = gcd(a, b), <gcd, x, y>
tuple<int, int, int> gcd(int a, int b) {
▷ if(b == 0) return make_tuple(a, 1, 0);
▷ auto [q, w, e] = gcd(b, a % b);
```

```
    return make_tuple(q, e, w - e * (a / b));
}
```

## 2.2 Chinese Remainder Theorem

```
// x = vet[i].first (mod vet[i].second)
ll crt(const vector<pair<ll, ll>> &vet){
    ll ans = 0, lcm = 1;
    ll a, b, g, x, y;
    for(const auto &p : vet) {
        tie(a, b) = p;
        tie(g, x, y) = gcd(lcm, b);
        if((a - ans) % g != 0) return -1; // no solution
        ans = ans + x * ((a - ans) / g) % (b / g) * lcm;
        lcm = lcm * (b / g);
        ans = (ans % lcm + lcm) % lcm;
    }
    return ans;
}
```

## 2.3 Diophantine Solver

```
template<typename T>
T extgcd(T a, T b, T &x, T &y) {
  if (a == 0) {
    x = 0;
    y = 1;
    return b;
  }
  T p = b / a;
  T g = extgcd(b - p * a, a, y, x);
  x -= p * y;
  return g;
}

template<typename T>
bool diophantine(T a, T b, T c, T &x, T &y, T &g) {
  if (a == 0 && b == 0) {
    if (c == 0) {
      x = y = g = 0;
      return true;
    }
    return false;
  }
  if (a == 0) {
    if (c % b == 0) {
      x = 0;
      y = c / b;
      g = abs(b);
      return true;
    }
    return false;
  }
  if (b == 0) {
    if (c % a == 0) {
      x = c / a;
      y = 0;
      g = abs(a);
      return true;
    }
    return false;
  }
  g = extgcd(a, b, x, y);
  if (c % g != 0) {
    return false;
  }
  T dx = c / a;
  c -= dx * a;
  T dy = c / b;
  c -= dy * b;
  x = dx + mulmod(x, c / g, b);
  y = dy + mulmod(y, c / g, a);
  g = abs(g);
  return true;
}
```

## 2.4 Preffix inverse

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

## 2.5 Pollard Rho

```
ll rho(ll n){
    if(n % 2 == 0) return 2;
    ll d, c, x, y, prod;
    do{
        c = llrand(1, n - 1);
        x = llrand(1, n - 1);
        y = x;
        prod = 1;
        for(int i = 0; i < 40; i++) {
            x = add(mul(x, x, n), c, n);
            y = add(mul(y, y, n), c, n);
            y = add(mul(y, y, n), c, n);
            prod = mul(prod, abs(x - y), n) ?: prod;
        }
        d = __gcd(prod, n);
    } while(d == 1);
    return d;
}

ll pollard_rho(ll n){
    ll x, c, y, d, k;
    int i;
    do{
        i = 1;
        x = llrand(1, n-1), c = llrand(1, n-1);
        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;
}
```

## 2.6 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(2, n-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;
}
```

## 2.7 Primitive root

```
// a primitive root modulo n is any number g such that any c coprime to n is congruent to a power of g modulo n.
bool exists_root(ll n){
    if(n == 1 || n == 2 || n == 4) return true;
    if(n % 2 == 0) n /= 2;
    if(n % 2 == 0) return false;
    // test if n is a power of only one prime
    for(ll i = 3; i * i <= n; i += 2) if(n % i == 0){
        while(n % i == 0) n /= i;
        return n == 1;
    }
    return true;
}
ll primitive_root(ll n){
    if(n == 1 || n == 2 || n == 4) return n - 1;
    if(not exists_root(n)) return -1;
    ll x = phi(n);
    auto pr = factorize(x);
    auto check = [x, n, pr](ll m){
        for(ll p : pr) if(fexp(m, x / p, n) == 1)
            return false;
        return true;
    };
    for(ll m = 2; ; m++) if(__gcd(m, n) == 1)
        if(check(m)) return m;
}

// Let's denote R(n) as the set of primitive roots modulo n, p is prime
// g \in R(p) => (pow(g, p-1, p * p) == 1 ? g+p : g) \in R(pow(p, k)), for all k > 1
// g in R(pow(p, k)) => (g % 2 == 1 ? g : g + pow(p, k)) \in R(2*pow(p, k))
```

## 2.8 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)}
```

## 2.9 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;
}
```

## 2.10 Modular multiplication TOPPER

```
ll mulmod(ll a, ll b, ll mod) {
    ll q = ll((long double)a * (long double)b / (long double)mod);
    ll r = (a * b - mod * q) % mod;
    if(r < 0) r += mod;
    return r;
}
```

## 2.11 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / x yields the same value for l <= x <= r
}
for(int l, r = n; r > 0; r = l - 1) {
    int tmp = (n + r - 1) / r;
    l = (n + tmp - 1) / tmp;
    // (n+x-1) / x yields the same value for l <= x <= r
}
```

## 2.12 Matrix Determinant

```
int n;
long double a[n][n];

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

    return det;
}
```

## 2.13 Simplex Method

```
typedef long double dbl;
const dbl eps = 1e-6;
const int N = , M = ;

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
struct simplex {
    int X[N], Y[M];
    dbl A[M][N], b[M], c[N];
    dbl ans;
    int n, m;
    dbl sol[N];

    void pivot(int x, int y){
        swap(X[y], Y[x]);
        b[x] /= A[x][y];
        for(int i = 0; i < n; i++)
            if(i != y)
                A[x][i] /= A[x][y];
        A[x][y] = 1. / A[x][y];
        for(int i = 0; i < m; i++)
            if(i != x && abs(A[i][y]) > eps) {
                b[i] -= A[i][y] * b[x];
                for(int j = 0; j < n; j++) if(j != y)
                                A[i][j] -= A[i][y] * A[x][j];
                A[i][y] = -A[i][y] * A[x][y];
            }
        ans += c[y] * b[x];
        for(int i = 0; i < n; i++)
            if(i != y)
                c[i] -= c[y] * A[x][i];
        c[y] = -c[y] * A[x][y];
    }

    // maximiza sum(x[i] * c[i])
    // sujeito a
    // sum(a[i][j] * x[j]) <= b[i] para 0 <= i < m (Ax <= b)
    // x[i] >= 0 para 0 <= i < n (x >= 0)
    // (n variaveis, m restricoes)
    // guarda a resposta em ans e retorna o valor otimo
    dbl solve(int _n, int _m) {
        this->n = _n; this->m = _m;

            for(int i = 1; i < m; i++){
                int id = uniform_int_distribution<int>(0, i)(rng);
                swap(b[i], b[id]);
                for(int j = 0; j < n; j++)
                    swap(A[i][j], A[id][j]);
            }

        ans = 0.;
        for(int i = 0; i < n; i++) X[i] = i;
        for(int i = 0; i < m; i++) Y[i] = i + n;
        while(true) {
            int x = min_element(b, b + m) - b;
            if(b[x] >= -eps)
                break;
            int y = find_if(A[x], A[x] + n, [](dbl d) { return d < -eps; }) - A[x];
            if(y == n) throw 1; // no solution
            pivot(x, y);
        }
        while(true) {
            int y = max_element(c, c + n) - c;
            if(c[y] <= eps) break;
            int x = -1;
            dbl mn = 1. / 0.;
            for(int i = 0; i < m; i++)
                if(A[i][y] > eps && b[i] / A[i][y] < mn)
                    mn = b[i] / A[i][y], x = i;
            if(x == -1) throw 2; // unbounded
            pivot(x, y);
        }
        memset(sol, 0, sizeof(dbl) * n);
        for(int i = 0; i < m; i++)
            if(Y[i] < n)
                sol[Y[i]] = b[i];
        return ans;
    }
};
```

## 2.14 FFT

```
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, 0);
            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;
}
```

## 2.15 FFT Tourist

```
namespace fft {
  typedef double dbl;

  struct num {
    dbl x, y;
    num() { x = y = 0; }
    num(dbl x, dbl y) : x(x), y(y) {}
  };

  inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
  inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
  inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
  inline num conj(num a) { return num(a.x, -a.y); }

  int base = 1;
  vector<num> roots = {{0, 0}, {1, 0}};
  vector<int> rev = {0, 1};

  const dbl PI = acosl(-1.0);

  void ensure_base(int nbase) {
    if(nbase <= base) return;

    rev.resize(1 << nbase);
    for(int i = 0; i < (1 << nbase); i++) {
      rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
    }
    roots.resize(1 << nbase);

    while(base < nbase) {
      dbl angle = 2*PI / (1 << (base + 1));
      for(int i = 1 << (base - 1); i < (1 << base); i++) {
```

```
      roots[i << 1] = roots[i];
      dbl angle_i = angle * (2 * i + 1 - (1 << base));
      roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
    }
    base++;
  }
}

void fft(vector<num> &a, int n = -1) {
  if(n == -1) {
    n = a.size();
  }
  assert((n & (n-1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for(int i = 0; i < n; i++) {
    if(i < (rev[i] >> shift)) {
      swap(a[i], a[rev[i] >> shift]);
    }
  }
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        num z = a[i+j+k] * roots[j+k];
        a[i+j+k] = a[i+j] - z;
        a[i+j] = a[i+j] + z;
      }
    }
  }
}

vector<num> fa, fb;
vector<int> multiply(vector<int> &a, vector<int> &b) {
  int need = a.size() + b.size() - 1;
  int nbase = 0;
  while((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  if(sz > (int) fa.size()) {
    fa.resize(sz);
  }
  for(int i = 0; i < sz; i++) {
    int x = (i < (int) a.size() ? a[i] : 0);
    int y = (i < (int) b.size() ? b[i] : 0);
    fa[i] = num(x, y);
  }
  fft(fa, sz);
  num r(0, -0.25 / sz);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
    if(i != j) {
      fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
    }
    fa[i] = z;
  }
  fft(fa, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    res[i] = fa[i].x + 0.5;
  }
  return res;
}

vector<int> multiply_mod(vector<int> &a, vector<int> &b, int m,
  ↪int eq = 0) {
  int need = a.size() + b.size() - 1;
  int nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  if (sz > (int) fa.size()) {
    fa.resize(sz);
  }
  for (int i = 0; i < (int) a.size(); i++) {
    int x = (a[i] % m + m) % m;
    fa[i] = num(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
  fft(fa, sz);
  if (sz > (int) fb.size()) {
    fb.resize(sz);
  }
  if (eq) {
    copy(fa.begin(), fa.begin() + sz, fb.begin());
  } else {
    for (int i = 0; i < (int) b.size(); i++) {
      int x = (b[i] % m + m) % m;
      fb[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
    fft(fb, sz);
  }
  dbl ratio = 0.25 / sz;
  num r2(0, -1);
  num r3(ratio, 0);
  num r4(0, -ratio);
  num r5(0, 1);
  for (int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    num a1 = (fa[i] + conj(fa[j]));
    num a2 = (fa[i] - conj(fa[j])) * r2;
    num b1 = (fb[i] + conj(fb[j])) * r3;
    num b2 = (fb[i] - conj(fb[j])) * r4;
    if (i != j) {
      num c1 = (fa[j] + conj(fa[i]));
      num c2 = (fa[j] - conj(fa[i])) * r2;
      num d1 = (fb[j] + conj(fb[i])) * r3;
      num d2 = (fb[j] - conj(fb[i])) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz);
  fft(fb, sz);
  vector<int> res(need);
  for (int i = 0; i < need; i++) {
    long long aa = fa[i].x + 0.5;
    long long bb = fb[i].x + 0.5;
    long long cc = fa[i].y + 0.5;
    res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
  }
  return res;
 }

 vector<int> square_mod(vector<int> &a, int m) {
  return multiply_mod(a, a, m, 1);
 }
}
```

## 2.16 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 * 1ll * 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] * 1ll * 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 * 1ll * wlen % mod);
▷ ▷ ▷ }
▷ ▷ }
▷ }
▷ if (invert) {
▷ ▷ int nrev = reverse (n, mod);
▷ ▷ for (int i=0; i<n; ++i)
▷ ▷ ▷ a[i] = int (a[i] * 1ll * nrev % mod);
▷ }
}
```

## 2.17 Gauss

```
// Solves systems of linear equations.
// To use, build a matrix of coefficients and call run(mat, R, C).
  ↪If the i-th variable is free, row[i] will be -1, otherwise
  ↪it's value will be ans[i].

namespace Gauss {
  const int MAXC = 1001;
  int row[MAXC];
  double ans[MAXC];

  void run(double mat[][MAXC], int R, int C) {
    REP(i, C) row[i] = -1;

    int r = 0;
    REP(c, C) {
      int k = r;
      FOR(i, r, R) if(fabs(mat[i][c]) > fabs(mat[k][c])) k = i;
      if(fabs(mat[k][c]) < eps) continue;

      REP(j, C+1) swap(mat[r][j], mat[k][j]);
      REP(i, R) if (i != r) {
        double w = mat[i][c] / mat[r][c];
        REP(j, C+1) mat[i][j] -= mat[r][j] * w;
      }
      row[c] = r++;
    }

    REP(i, C) {
      int r = row[i];
```

```
      ans[i] = r == -1 ? 0 : mat[r][C] / mat[r][i];
    }
  }
}
```

### 2.18 Gauss Xor

```
const ll MAX = 1e9;
const int LOG_MAX = 64 - __builtin_clzll((ll)MAX);

struct Gauss {
    array<ll, LOG_MAX> vet;
    int size;
    Gauss() : size(0) {
        fill(vet.begin(), vet.end(), 0);
    }
    Gauss(vector<ll> vals) : size(0) {
        fill(vet.begin(), vet.end(), 0);
        for(ll val : vals) add(val);
    }
    bool add(ll val) {
        for(int i = 0; i < LOG_MAX; i++) if(val & (1LL << i)) {
            if(vet[i] == 0) {
                vet[i] = val;
                size++;
                return true;
            }
            val ^= vet[i];
        }
        return false;
    }
};
```

### 2.19 Simpson

```
inline double simpson(double fl,double fr,double fmid,double l,double r) {
    return (fl + fr + 4.0 * fmid) * (r - l) / 6.0;
}
double rsimpson(double slr,double fl,double fr,double fmid,double l,double r) {
    double mid = (l+r)*0.5;
    double fml = f((l+mid)*0.5), fmr = f((mid+r)*0.5);
    double slm = simpson(fl, fmid, fml, l, mid);
    double smr = simpson(fmid, fr, fmr, mid, r);
    if(fabs(slr-slm-smr) < eps and r - l < delta) return slr;
    return rsimpson(slm,fl,fmid,fml,l,mid) + rsimpson(smr,fmid,fr,fmr,mid,r);
}
double integrate(double l,double r) {
    double mid = (l+r)*0.5;
    double fl = f(l), fr = f(r), fmid = f(mid);
    return rsimpson(simpson(fl,fr,fmid,l,r),fl,fr,fmid,l,r);
}
```

### 2.20 Matrix

```
template <const size_t n, const size_t m, class T = modBase<>>
struct Matrix {
  T v[n][m];

  Matrix(int d = 0) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        v[i][j] = T(0);
      }
      if (i < m) {
        v[i][i] = T(d);
      }
    }
  }

  template <size_t mm>
  Matrix<n, mm, T> operator*(Matrix<m, mm, T> &o) {
    Matrix<n, mm, T> ans;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < mm; j++) {
        for (int k = 0; k < m; k++) {
          ans.v[i][j] = ans.v[i][j] + v[i][k] * o.v[k][j];
        }
      }
    }
    return ans;
  }
};
```

# 3 Graphs

### 3.1 Bipartite Matching

```
// O(V * E)
int match[N];
int vis[N], pass;
vector<int> g[N];

bool dfs(int u) {
    vis[u] = pass;

    for(int v : g[u]) if(vis[v] != pass) {
        vis[v] = pass;
        if(match[v] == -1 or dfs(match[v])) {
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int max_maching() {
    memset(match, -1, sizeof match);
    int max_matching_size = 0;
    for(int u : vertices_on_side_A) {
        pass++;
        if(dfs(i)) max_matching_size++;
    }
    return max_matching_size;
}
```

### 3.2 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; // v.cap - v.flow < lim
        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(auto& ed : g[u]) {
            auto v = edge[ed];
            if(v.flow >= v.cap || vis[v.to] == pass)
                continue; // v.cap - v.flow < lim
            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;
    //for(lim = (1LL << 62); lim >= 1; lim /= 2)
    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;
}
```

### 3.3 Push relabel

```
// Push relabel in O(V^2 E^0.5) with gap heuristic
// It's quite fast
template<typename flow_t = long long>
struct PushRelabel {
    struct Edge { int to, rev; flow_t f, c; };
    vector<vector<Edge> > g;
    vector<flow_t> ec;
    vector<Edge*> cur;
    vector<vector<int> > hs;
    vector<int> H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
    void add_edge(int s, int t, flow_t cap, flow_t rcap=0) {
        if (s == t) return;
        Edge a = {t, (int)g[t].size(), 0, cap};
        Edge b = {s, (int)g[s].size(), 0, rcap};
        g[s].push_back(a);
        g[t].push_back(b);
    }
    void add_flow(Edge& e, flow_t f) {
```

```
        Edge &back = g[e.to][e.rev];
        if (!ec[e.to] && f)
            hs[H[e.to]].push_back(e.to);
        e.f += f, ec[e.to] += f;
        back.f -= f, ec[back.to] -= f;
    }
    flow_t max_flow(int s, int t) {
        int v = g.size();
        H[s] = v; ec[t] = 1;
        vector<int> co(2 * v);
        co[0] = v-1;
        for(int i = 0; i < v; ++i) cur[i] = g[i].data();
        for(auto &e : g[s]) add_flow(e, e.c);

        if(hs[0].size())
        for (int hi = 0; hi >= 0;) {
            int u = hs[hi].back();
            hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == g[u].data() + g[u].size()) {
                    H[u] = 1e9;
                    for(auto &e:g[u])
                        if (e.c - e.f && H[u] > H[e.to]+1)
                            H[u] = H[e.to]+1, cur[u] = &e;
                    if (++co[H[u]], !--co[hi] && hi < v)
                        for(int i = 0; i < v; ++i)
                            if (hi < H[i] && H[i] < v){
                                --co[H[i]];
                                H[i] = v + 1;
                            }
                    hi = H[u];
                } else if (cur[u]->c - cur[u]->f && H[u] == H[cur[u]->to]+1)
                    add_flow(*cur[u], min(ec[u], cur[u]->c - cur[u]->f));
                else ++cur[u];
            while (hi >= 0 && hs[hi].empty()) --hi;
        }
        return -ec[s];
    }
};
```

### 3.4 Min Cost Max Flow

```
const ll oo = 1e18;
const int N = 422, E = 2 * 10006;

vector<int> g[N];
int ne;
struct Edge{
    int from, to; ll cap, cost;
} edge[E];
int start = N-1, target = N-2, p[N]; int inqueue[N];
ll d[N];
ll pot[N];
bool dijkstra(int source, int sink) {
    for(int i = 0; i < N; i++) d[i] = oo;
    d[source] = 0;
    priority_queue<pair<ll, int>> q;
    q.emplace(0, source);
    ll dt; int u;
    while(!q.empty()) {
        tie(dt, u) = q.top(); q.pop(); dt = -dt;
        if(dt > d[u]) continue;
        if(u == sink) return true;
        for(int e : g[u]) {
            auto v = edge[e];
            const ll cand = d[u] + v.cost + pot[u] - pot[v.to];
            if(v.cap > 0 and cand < d[v.to]) {
                p[v.to] = e;
                d[v.to] = cand;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
    return d[sink] < oo;
}

// <max flow, min cost>
pair<ll, ll> mincost(int source = start, int sink = target){
    ll ans = 0, mf = 0;
    while(dijkstra(source, sink)){
        ll f = oo;
        for(int u = sink; u != source; u = edge[ p[u] ].from)
            f = min(f, edge[ p[u] ].cap);
        mf += f;
        ans += f * (d[sink] - pot[source] + pot[sink]);
        for(int u = sink; u != source; u = edge[ p[u] ].from){
            edge[ p[u] ].cap -= f;
            edge[ p[u] ^ 1 ].cap += f;
        }
        for(int i = 0; i < N; i++) pot[i] = min(oo, pot[i] + d[i]);
    }
    return {mf, ans};
}
void addEdge(int u, int v, ll c, ll cost){
    assert(cost >= 0);
    edge[ne] = {u, v, c, cost};
    g[u].push_back(ne++);
    edge[ne] = {v, u, 0,-cost};
    g[v].push_back(ne++);
}
```

### 3.5 Blossom Algorithm for General Matching

```
const int MAXN = 2020 + 1;
// 1-based Vertex index
int vis[MAXN], par[MAXN], orig[MAXN], match[MAXN], aux[MAXN], t, N;
vector<int> conn[MAXN];
queue<int> Q;
void addEdge(int u, int v) {
    conn[u].push_back(v); conn[v].push_back(u);
}
void init(int n) {
    N = n; t = 0;
    for(int i=0; i<=n; ++i)
        conn[i].clear(), match[i] = aux[i] = par[i] = 0;
}
void augment(int u, int v) {
    int pv = v, nv;
    do {
        pv = par[v]; nv = match[pv];
        match[v] = pv; match[pv] = v;
        v = nv;
    } while(u != pv);
}
int lca(int v, int w) {
    ++t;
    while(true) {
        if(v) {
            if(aux[v] == t) return v; aux[v] = t;
            v = orig[par[match[v]]];
        }
        swap(v, w);
    }
}
void blossom(int v, int w, int a) {
    while(orig[v] != a) {
        par[v] = w; w = match[v];
        if(vis[w] == 1) Q.push(w), vis[w] = 0;
        orig[v] = orig[w] = a; v = par[w];
    }
}
bool bfs(int u) {
    fill(vis+1, vis+1+N, -1); iota(orig + 1, orig + N + 1, 1);
    Q = queue<int>(); Q.push(u); vis[u] = 0;
    while(!Q.empty()) {
        int v = Q.front(); Q.pop();
        for(int x: conn[v]) {
            if(vis[x] == -1) {
                par[x] = v; vis[x] = 1;
                if(!match[x]) return augment(u, x), true;
                Q.push(match[x]); vis[match[x]] = 0;
            }
            else if(vis[x] == 0 && orig[v] != orig[x]) {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a); blossom(v, x, a);
            }
        }
    }
    return false;
}
int Match() {
    int ans = 0;
    // find random matching (not necessary, constant improvement)
    vector<int> V(N-1); iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x: V) if(!match[x]){
        for(auto y: conn[x]) if(!match[y]) {
            match[x] = y, match[y] = x;
            ++ans; break;
        }
    }
    for(int i=1; i<=N; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
}
```

### 3.6 Blossom Algorithm for Weighted General Matching

```
// N^3 (but fast in practice)
static const int INF = INT_MAX;
static const int N = 514;
struct edge{
    int u,v,w; edge(){}
    edge(int ui,int vi,int wi)
        :u(ui),v(vi),w(wi){}
};
int n,n_x;
edge g[N*2][N*2];
int lab[N*2];
int match[N*2],slack[N*2],st[N*2],pa[N*2];
int flo_from[N*2][N+1],S[N*2],vis[N*2];
vector<int> flo[N*2];
queue<int> q;
int e_delta(const edge &e){
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
}
void update_slack(int u,int x){
    if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][x]))slack[x]=u;
}
void set_slack(int x){
```

```
  slack[x]=0;
  for(int u=1;u<=n;++u)
    if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
      update_slack(u,x);
}
void q_push(int x){
  if(x<=n)q.push(x);
  else for(size_t i=0;i<flo[x].size();i++)
    q_push(flo[x][i]);
}
void set_st(int x,int b){
  st[x]=b;
  if(x>n)for(size_t i=0;i<flo[x].size();++i)
    set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
  int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].begin();
  if(pr%2==1){
    reverse(flo[b].begin()+1,flo[b].end());
    return (int)flo[b].size()-pr;
  }else return pr;
}
void set_match(int u,int v){
  match[u]=g[u][v].v;
  if(u<=n) return;
  edge e=g[u][v];
  int xr=flo_from[u][e.u],pr=get_pr(u,xr);
  for(int i=0;i<pr;++i)set_match(flo[u][i],flo[u][i^1]);
  set_match(xr,v);
  rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end());
}
void augment(int u,int v){
  for(;;){
    int xnv=st[match[u]];
    set_match(u,v);
    if(!xnv)return;
    set_match(xnv,st[pa[xnv]]);
    u=st[pa[xnv]],v=xnv;
  }
}
int get_lca(int u,int v){
  static int t=0;
  for(++t;u||v;swap(u,v)){
    if(u==0)continue;
    if(vis[u]==t)return u;
    vis[u]=t;
    u=st[match[u]];
    if(u)u=st[pa[u]];
  }
  return 0;
}
void add_blossom(int u,int lca,int v){
  int b=n+1;
  while(b<=n_x&&st[b])++b;
  if(b>n_x)++n_x;
  lab[b]=0,S[b]=0;
  match[b]=match[lca];
  flo[b].clear();
  flo[b].push_back(lca);
  for(int x=u,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
  reverse(flo[b].begin()+1,flo[b].end());
  for(int x=v,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x]]),q_push(y);
  set_st(b,b);
  for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
  for(int x=1;x<=n;++x)flo_from[b][x]=0;
  for(size_t i=0;i<flo[b].size();++i){
    int xs=flo[b][i];
    for(int x=1;x<=n_x;++x)
      if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
        g[b][x]=g[xs][x],g[x][b]=g[x][xs];
    for(int x=1;x<=n;++x)
      if(flo_from[xs][x])flo_from[b][x]=xs;
  }
  set_slack(b);
}
void expand_blossom(int b){
  for(size_t i=0;i<flo[b].size();++i)
    set_st(flo[b][i],flo[b][i]);
  int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
  for(int i=0;i<pr;i+=2){
    int xs=flo[b][i],xns=flo[b][i+1];
    pa[xs]=g[xns][xs].u;
    S[xs]=1,S[xns]=0;
    slack[xs]=0,set_slack(xns);
    q_push(xns);
  }
  S[xr]=1,pa[xr]=pa[b];
  for(size_t i=pr+1;i<flo[b].size();++i){
    int xs=flo[b][i];
    S[xs]=-1,set_slack(xs);
  }
  st[b]=0;
}
bool on_found_edge(const edge &e){
  int u=st[e.u],v=st[e.v];
  if(S[v]==-1){
    pa[v]=e.u,S[v]=1;
    int nu=st[match[v]];
    slack[v]=slack[nu]=0;
    S[nu]=0,q_push(nu);
  }else if(S[v]==0){
    int lca=get_lca(u,v);
    if(!lca)return augment(u,v),augment(v,u),true;
    else add_blossom(u,lca,v);
  }
  return false;
}
bool matching(){
  memset(S+1,-1,sizeof(int)*n_x);
  memset(slack+1,0,sizeof(int)*n_x);
  q=queue<int>();
  for(int x=1;x<=n_x;++x)
    if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
  if(q.empty())return false;
  for(;;){
    while(q.size()){
      int u=q.front();q.pop();
      if(S[st[u]]==1)continue;
      for(int v=1;v<=n;++v)
        if(g[u][v].w>0&&st[u]!=st[v]){
          if(e_delta(g[u][v])==0){
            if(on_found_edge(g[u][v]))return true;
          }else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
    for(int x=1;x<=n_x;++x)
      if(st[x]==x&&slack[x]){
        if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
        else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x])/2);
      }
    for(int u=1;u<=n;++u){
      if(S[st[u]]==0){
        if(lab[u]<=d)return 0;
        lab[u]-=d;
      }else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b){
        if(S[st[b]]==0)lab[b]+=d*2;
        else if(S[st[b]]==1)lab[b]-=d*2;
      }
    q=queue<int>();
    for(int x=1;x<=n_x;++x)
      
      ↪if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta(g[slack[x]][x])==0
        if(on_found_edge(g[slack[x]][x]))return true;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(b);
  }
  return false;
}
pair<long long,int> solve(){
  memset(match+1,0,sizeof(int)*n);
  n_x=n;
  int n_matches=0;
  long long tot_weight=0;
  for(int u=0;u<=n;++u)st[u]=u,flo[u].clear();
  int w_max=0;
  for(int u=1;u<=n;++u)
    for(int v=1;v<=n;++v){
      flo_from[u][v]=(u==v?u:0);
      w_max=max(w_max,g[u][v].w);
    }
  for(int u=1;u<=n;++u)lab[u]=w_max;
  while(matching())++n_matches;
  for(int u=1;u<=n;++u)
    if(match[u]&&match[u]<u)
      tot_weight+=g[u][match[u]].w;
  return make_pair(tot_weight,n_matches);
}
void add_edge( int ui , int vi , int wi ){
  g[ui][vi].w = g[vi][ui].w = wi;
}
void init( int _n ){
  n = _n;
  for(int u=1;u<=n;++u)
    for(int v=1;v<=n;++v)
      g[u][v]=edge(u,v,0);
}
```

## 3.7 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*/ }
}
```

## 3.8 Kosaraju

```
vector<int> g[N], gt[N], S; int vis[N], cor[N];
void dfs(int u){
    vis[u] = 1; for(int v : g[u]) if(!vis[v]) dfs(v);
    S.push_back(u);
}
void dfst(int u, int e){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v]) dfst(v, e);
}
void kosaraju(){
    for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);
    for(int i = 1; i <= n; i++) for(int j : g[i])
        gt[j].push_back(i);
    int e = 0; reverse(S.begin(), S.end());
    for(int u : S) if(!cor[u]) dfst(u, ++e);
}
```

## 3.9 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(u == 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);
    }
}
```

## 3.10 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]);
}
```

## 3.11 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();
}
```

## 3.12 Min Cost Matching

```
// Min cost matching
// O(n^2 * m)
// n == nro de linhas
// m == nro de colunas
// 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];
```

# 4 Strings

## 4.1 Aho Corasick

```
int to[N][A];
int ne = 2, fail[N], term[N];
void add_string(const char *str, int id) {
    int p = 1;
    for(int i = 0; str[i]; i++) {
        int ch = str[i] - 'a';
        if(!to[p][ch]) to[p][ch] = ne++;
        p = to[p][ch];
    }
    term[p]++;
}
void init() {
    for(int i = 0; i < ne; i++) fail[i] = 1;
    queue<int> q; q.push(1);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int i = 0; i < A; i++){
            if(to[u][i]) {
                int v = to[u][i]; q.push(v);
                if(u != 1) {
                    fail[v] = to[ fail[u] ][i];
                    term[v] += term[ fail[v] ];
                }
            }
            else if(u != 1) to[u][i] = to[ fail[u] ][i];
            else to[u][i] = 1;
        }
    }
}
void clean() {
    memset(to, 0, ne * sizeof(to[0]));
    memset(fail, 0, ne * sizeof(fail[0]));
    memset(term, 0, ne * sizeof(term[0]));
    ne = 2;
}
```

## 4.2 Suffix Array

```
int lcp[N], c[N];

// Caractere final da string '\0' esta sendo considerado parte da
    ↪string s
void build_sa(char s[], int n, int a[]){
    const int A = 300; // Tamanho do alfabeto
    int c1[n], a1[n], h[n + A];
    memset(h, 0, sizeof h);

    for(int i = 0; i < n; i++) {
        c[i] = s[i];
        h[c[i] + 1]++;
    }

    partial_sum(h, h + A, h);
    for(int i = 0; i < n; i++)
        a[h[c[i]]++] = i;
    for(int i = 0; i < n; i++)
        h[c[i]]--;

    for(int L = 1; L < n; L <<= 1) {
        for(int i = 0; i < n; i++) {
            int j = (a[i] - L + n) % n;
            a1[h[c[j]]++] = j;
        }

        int cc = -1;
        for(int i = 0; i < n; i++) {
            if(i == 0 || c[a1[i]] != c[a1[i-1]] || c[(a1[i] + L) %
                ↪n] != c[(a1[i-1] + L) % n])
                h[++cc] = i;
            c1[a1[i]] = cc;
        }

        memcpy(a, a1, sizeof a1);
        memcpy(c, c1, sizeof c1);

        if(cc == n-1) break;
    }
}

void build_lcp(char s[], int n, int a[]){ // lcp[i] =
    ↪lcp(s[:a[i]], s[:a[i+1]])
    int k = 0;

    //memset(lcp, 0, sizeof lcp);
    for(int i = 0; i < n; i++){
        if(c[i] == n-1) continue;
        int j = a[c[i]+1];
        while(i+k < n && j+k < n && s[i+k] == s[j+k]) k++;
        lcp[c[i]] = k;
        if(k) k--;
    }
}

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

## 4.3 Adamant Suffix Tree

```
namespace sf {

const int inf = 1e9;
const int maxn = 200005;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], fpos[maxn], link[maxn];
int node, pos;
int sz = 1, n = 0;

int make_node(int _pos, int _len) {
  fpos[sz] = _pos;
  len[sz] = _len;
  return sz++;
}
void go_edge() {
  while (pos > len[to[node][s[n - pos]]]) {
    node = to[node][s[n - pos]];
    pos -= len[node];
  }
}
void add_letter(int c) {
  s[n++] = (char)c;
  pos++;
  int last = 0;
  while (pos > 0) {
    go_edge();
    int edge = s[n - pos];
    int &v = to[node][edge];
    int t = s[fpos[v] + pos - 1];
    if (v == 0) {
      v = make_node(n - pos, inf);
      link[last] = node;
      last = 0;
    } else if (t == c) {
      link[last] = node;
      return;
    } else {
      int u = make_node(fpos[v], pos - 1);
      to[u][c] = make_node(n - 1, inf);
      to[u][t] = v;
      fpos[v] += pos - 1;
      len[v] -= pos - 1;
      v = u;
      link[last] = u;
      last = u;
    }
    if (node == 0)
      pos--;
    else
      node = link[node];
  }
}
void add_string(char *str) {
  for (int i = 0; str[i]; i++) add_letter(str[i]);
  add_letter('$');
}
bool is_leaf(int u) { return len[u] > n; }
int get_len(int u) {
  if (!u) return 0;
  if (is_leaf(u)) return n - fpos[u];
  return len[u];
}
int leafs[maxn];
int calc_leafs(int u = 0) {
  leafs[u] = is_leaf(u);
  for (const auto &c : to[u]) leafs[u] += calc_leafs(c.second);
  return leafs[u];
}
}; // namespace sf

int main() { sf::len[0] = sf::inf; }
```

## 4.4 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;
}
```

## 4.5 Prefix function/KMP

```
vector<int> preffix_function(const string &s){
▹ int n = s.size(); vector<int> b(n+1);
```

```cpp
    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 = preffix_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];
        }
    }
}
```

## 4.6 Min rotation

```cpp
// remember std::rotate
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;
}
```

## 4.7 Manacher

```cpp
// rad[2 * i] = largest palindrome cetered at char i
// rad[2 * i + 1] = largest palindrome cetered between chars i and
    ↪i+i
void manacher(char *s, int n, int *rad) {
    static char t[2*MAX];
    int m = 2 * n - 1;

    for(int i = 0; i < m; i++) t[i] = -1;
    for(int i = 0; i < n; i++) t[2 * i] = s[i];

    int x = 0;
    rad[0] = 0; // <
    for(int i = 1; i < m; i++) {
        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 and i + r + 1 < m and
            t[i - r - 1] == t[i + r + 1]) ++r;
        if(i + r >= x + rad[x]) x = i;
    }

    for(int i = 0; i < m; i++) {
        if(i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
    }
    // for(int i = 0; i < m; i++) rad[i] /= 2;
}
```

## 4.8 Suffix Automaton

```cpp
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;
}
```

# 5 Geometry

## 5.1 2D basics

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

struct vec{
    cod x, y; int id;
    vec(cod a = 0, cod 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*(cod t) const{
        return {x * t, y * t};
    }
    vec operator/(cod t) const{
        return {x / t, y / t};
    }
    cod operator*(const vec &o) const{ // cos
        return x * o.x + y * o.y;
    }
    cod 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;
    }
    cod cross(const vec &a, const vec &b) const{
        return (a-(*this)) ^ (b-(*this));
    }
    int ccw(const vec &a, const vec &b) const{
        cod tmp = cross(a, b);
        return (tmp > eps) - (tmp < -eps);
    }
    cod dot(const vec &a, const vec &b) const{
        return (a-(*this)) * (b-(*this));
    }
    cod 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};
    }
    vec proj(const vec &b) const{ // proj of *this onto b
        cod k = operator*(b) / (b * b);
        return b * k;
    }
    // proj of (*this) onto the plane orthogonal to b
    vec rejection(vec b) const{
        return (*this) - proj(b);
    }
};

struct line{
    cod 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};
    }
    cod 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);
    }
    cod 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(cod x) const{
        return {x, (-c-a*x)/b};
    }
    vec at_y(cod y) const{
        return {(-c-b*y)/a, y};
    }
    cod 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;
    }
    cod len() const{
        return (p-q).len();
    }
    cod 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;
    }
    cod 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)));
    }
    cod 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;
    }
    cod 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);
    }
    cod 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)));
    }
    cod 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);
    }
    cod 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(cod a, cod b, cod c){
    cod s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
line mediatrix(const vec &a, const vec &b) {
    auto tmp = (b - a) * 2;
    return line(tmp.x, tmp.y, a * a - b * b);
}
struct circle {
    vec c; cod r;
    circle() : c(0, 0), r(0) {}
    circle(const vec o) : c(o), r(0) {}
    circle(const vec &a, const vec &b) {
        c = (a + b) * 0.5; r = (a - c).len();
    }
    circle(const vec &a, const vec &b, const vec &cc) {
        c = mediatrix(a, b).inter(mediatrix(b, cc));
        r = (a - c).len();
    }
    bool inside(const vec &a) const {
        return (a - c).len() <= r;
    }
};
circle min_circle_cover(vector<vec> v) {
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = (int)v.size();
    for(int i = 0; i < n; i++) if(!ans.inside(v[i])) {
        ans = circle(v[i]);
        for(int j = 0; j < i; j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0; k<j; k++)if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

## 5.2 Circle line intersection

```
// intersection of line a * x + b * y + c = 0
// and circle centered at the origin with radius r
double r, a, b, c; // given as input
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
if(c*c > r*r*(a*a+b*b)+EPS)
   puts("no points");
else if(abs(c*c - r*r*(a*a+b*b)) < EPS){
   puts("1 point");
   cout << x0 << ' ' << y0 << '\n';
}
else {
   double d = r*r - c*c/(a*a+b*b);
   double mult = sqrt (d / (a*a+b*b));
   double ax, ay, bx, by;
   ax = x0 + b * mult;
   bx = x0 - b * mult;
   ay = y0 - a * mult;
   by = y0 + a * mult;
   puts ("2 points");
   cout<<ax<<' '<<ay<<'\n'<<bx<<' '<<by<<'\n';
}
```

## 5.3 Half plane intersection

```
const double eps = 1e-8;
typedef pair<long double, long double> pi;
bool z(long double x){ return fabs(x) < eps; }
struct line{
    long double a, b, c;
    bool operator<(const line &l)const{
```

```
        bool flag1 = pi(a, b) > pi(0, 0);
        bool flag2 = pi(l.a, l.b) > pi(0, 0);
        if(flag1 != flag2) return flag1 > flag2;
        long double t = ccw(pi(0, 0), pi(a, b), pi(l.a, l.b));
        return z(t) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : t > 0;
    }
    pi slope(){ return pi(a, b); }
};
pi cross(line a, line b){
    long double det = a.a * b.b - b.a * a.b;
    return pi((a.c * b.b - a.b * b.c) / det, (a.a * b.c - a.c * b.a) / det);
}
bool bad(line a, line b, line c){
    if(ccw(pi(0, 0), a.slope(), b.slope()) <= 0) return false;
    pi crs = cross(a, b);
    return crs.first * c.a + crs.second * c.b >= c.c;
}
bool solve(vector<line> v, vector<pi> &solution){ // ax + by <= c;
    sort(v.begin(), v.end());
    deque<line> dq;
    for(auto &i : v){
        if(!dq.empty() && z(ccw(pi(0, 0), dq.back().slope(), i.slope()))) continue;
        while(dq.size() >= 2 && bad(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
        while(dq.size() >= 2 && bad(i, dq[0], dq[1])) dq.pop_front();
        dq.push_back(i);
    }
    while(dq.size() > 2 && bad(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
    while(dq.size() > 2 && bad(dq.back(), dq[0], dq[1])) dq.pop_front();
    vector<pi> tmp;
    for(int i=0; i<dq.size(); i++){
        line cur = dq[i], nxt = dq[(i+1)%dq.size()];
        if(ccw(pi(0, 0), cur.slope(), nxt.slope()) <= eps) return false;
        tmp.push_back(cross(cur, nxt));
    }
    solution = tmp;
    return true;
}
```

## 5.4 Detect empty Half plane intersection

```
// abs(point a) = absolute value of a
// ccw(a, b, c) = a.ccw(b, c)
pair<bool, point> half_inter(vector<pair<point,point> > &vet){
    random_shuffle(all(vet));
    point p;
    rep(i,0,sz(vet)) if(ccw(vet[i].x,vet[i].y,p) != 1){
        point dir = (vet[i].y - vet[i].x) / abs(vet[i].y - vet[i].x);
        point l = vet[i].x - dir*1e15;
        point r = vet[i].x + dir*1e15;
        if(r < l) swap(l, r);
        rep(j, 0, i){
            if(ccw(point(), vet[i].x-vet[i].y, vet[j].x-vet[j].y) == 0){
                if(ccw(vet[j].x, vet[j].y, p) == 1)
                    continue;
                return mp(false, point());
            }
            if(ccw(vet[j].x, vet[j].y, l) != 1)
                l = max(l, line_intersect(vet[i].x,vet[i].y,vet[j].x,vet[j].y));
            if(ccw(vet[j].x, vet[j].y, r) != 1)
                r = min(r, line_intersect(vet[i].x,vet[i].y,vet[j].x,vet[j].y));
            if(!(l < r)) return mp(false, point());
        }
        p = r;
    }
    return mp(true, p);
}
```

## 5.5 Circle Circle intersection

Assume that the first circle is centered at the origin and second at $(x2, y2)$. Find circle line intersection of first circle and line $Ax + By + C = 0$, where $A = -2x_2$, $B = -2y_2$, $C = x_2^2 + y_2^2 + r_1^2 - r_2^2$.

Be aware of corner case with two circles centered at the same point.

## 5.6 Tangents of two circles

```
// solve first for same circle(and infinitely many tangents)
// Find up to four tangents of two circles
void tangents(pt c, double r1, double r2, vector<line> & ans){
    double r = r2 - r1;
    double z = c.x * c.x + c.y * c.y;
    double d = z - r * 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;
    pt aux = a.center - b.center;
    for(int i = -1; i <= 1; i += 2)
        for(int j = -1; j <= 1; j += 2)
            tangents(aux, 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;
}
```

## 5.7 Convex Hull

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

    vector<vec> L, U;
    for(auto p : P){
        // BE CAREFUL WITH OVERFLOW!
        // MAX VALUE (2*A)^2, where 0 <= abs(p.x), abs(p.y) <= A
        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;
}
```

## 5.8 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);
}
```

## 5.9 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;
}
```

## 5.10 Minkowski sum

```
vector<vec> msum(vector<vec>& a, 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;
}
```

## 5.11 Geo Notes

### 5.11.1 Center of mass

**System of points(2D/3D):** Mass weighted average of points.
**Frame(2D/3D):** Get middle point of each segment solve as previously.
**Triangle:** Average of vertices.
**2D Polygon:** Compute **signed** area and center of mass of triangle $((0,0), p_i, p_{i+1})$. Then solve as system of points.
**Polyhedron surface:** Solve each face as a 2D polygon(be aware of (0, 0)) then replace each face with its center of mass and solve as system of points.
**Tetrahedron(Triangular pyramid):** As triangles, its the average of points.
**Polyhedron:** Can be done as 2D polygon, but with tetrahedralization intead of triangulation.

### 5.11.2 Pick's Theorem

Given a polygon without self-intersections and all its vertices on integer coordinates in some 2D grid. Let $A$ be its area, $I$ the number of points with integer coordinates stricly inside the polygon and $B$ the number of points with integer coordinates in the border of the polygon. The following formula holds: $A = I + \frac{B}{2} - 1$.

# 6 Miscellaneous

## 6.1 Cute 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();
```

## 6.2 Cute 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();
```

## 6.3 Efficient recursive lambda

```
template<class Fun>
class y_combinator_result {
  Fun fun_;
public:
  template<class T>
  explicit y_combinator_result(T &&fun):
      fun_(std::forward<T>(fun)) {}

  template<class ...Args>
  decltype(auto) operator()(Args &&...args) {
    return fun_(std::ref(*this), std::forward<Args>(args)...);
  }
};

template<class Fun>
decltype(auto) y_combinator(Fun &&fun) {
  return
      y_combinator_result<std::decay_t<Fun>>(std::forward<Fun>(fun));
}

// auto gcd = y_combinator([](auto gcd, int a, int b) -> int {
// return b == 0 ? a : gcd(b, a % b);
// });
```

## 6.4 Bitsets

```
#define private public
#include <bitset>
#undef private
#include <bits/stdc++.h>

using namespace std;

#define tab _M_w
using biti = typename
    remove_reference<decltype(bitset<404>().tab[0])>::type;
const int SIZE = 8 * sizeof(biti);
const int LOG = __builtin_ctz(SIZE);

template<size_t Nw>
int find_prev(const bitset<Nw> &x, int v) {
    int start = v >> LOG;
    int first_bits = v & (SIZE - 1);
    if(first_bits) {
        biti curr = x.tab[start];
        curr = curr << (SIZE - first_bits) >> (SIZE - first_bits);
        if(curr)
            return start << LOG | (SIZE - __builtin_clzl(curr) - 1);
    }
    for(int i = start - 1; i >= 0; i--) {
        biti curr = x.tab[i];
        if(curr) {
            return (i << LOG) | (SIZE - __builtin_clzl(curr) - 1);
        }
    }
    return -1;
}
// s._Find_first(); s._Find_next(k); find_prev(s, k+1);
// _Unchecked_set/_Unchecked_reset/_Unchecked_flip
```

## 6.5 Buildings

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

## 6.6 Rand

```
#include <random>
#include <chrono>
cout << RAND_MAX << endl;
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
shuffle(p.begin(), p.end(), rng);
uniform_int_distribution<int>(a,b)(rng);
```

## 6.7 Klondike

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

int v[305], dp[305][305], 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;
}
```

## 6.8 Hilbert Order

```
// maybe use B = n / sqrt(q) before this
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;
}
```

## 6.9 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) % p;
    return (f_exp(tmp, p - 2, p) * factmod(n, p)) % p;
}
```

## 6.10 Enumeration 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){

}

// loop through all supermasks of a given bitmask
for(int super = mask; super < (1 << n); super = (super + 1) |
    ↪mask) {

}
```

## 6.11 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();
▷ }
}
```

## 6.12 LCA <O(nlgn), O(1)>

```
int start[N], dfs_time;
int tour[2*N], id[2*N];

void dfs(int u){
    start[u] = dfs_time;
    id[dfs_time] = u;
    tour[dfs_time++] = start[u];
    for(int v : g[u]){
        dfs(v);
        id[dfs_time] = u;
        tour[dfs_time++] = start[u];
    }
}

int LCA(int u, int v){
    if(start[u] > start[v]) swap(u, v);
    return id[min(tour[k]for k in [start[u],start[v]])];
}
```

## 6.13 Buffered reader

```
// source:
    ↪https://github.com/ngthanhtrung23/ACM_Notebook_new/blob/master/buffered_reader.h
int INP,AM,REACHEOF;
#define BUFSIZE (1<<12)
char BUF[BUFSIZE+1], *inp=BUF;
#define GETCHAR(INP) { \
    if(!*inp && !REACHEOF) { \
        memset(BUF,0,sizeof BUF);\
        int inpzzz = fread(BUF,1,BUFSIZE,stdin);\
        if (inpzzz != BUFSIZE) REACHEOF = true;\
        inp=BUF; \
    } \
    INP=*inp++; \
}
#define DIG(a) (((a)>='0')&&((a)<='9'))
#define GN(j) { \
    AM=0;\
    GETCHAR(INP); while(!DIG(INP) && INP!='-') GETCHAR(INP);\
    if (INP=='-') {AM=1;GETCHAR(INP);} \
    j=INP-'0'; GETCHAR(INP); \
    while(DIG(INP)){j=10*j+(INP-'0');GETCHAR(INP);} \
    if (AM) j=-j;\
}
```

## 6.14 Modular summation

```
//calcula (sum(0 <= i <= n) P(i)) % mod,
//onde P(i) eh uma PA modular (com outro modulo)
namespace sum_pa_mod{
▷ ll calc(ll a, ll b, ll n, ll mod){
▷ ▷ assert(a&&b);
▷ ▷ if(a >= b){
▷ ▷ ▷ ll ret = ((n*(n+1)/2)%mod)*(a/b);
▷ ▷ ▷ if(a%b) ret = (ret + calc(a%b,b,n,mod))%mod;
▷ ▷ ▷ else ret = (ret+n+1)%mod;
▷ ▷ ▷ return ret;
▷ ▷ }
▷ ▷ return ((n+1)*(((n*a)/b+1)%mod) - calc(b,a,(n*a)/b,mod) + mod +
    ↪n/b + 1)%mod;
▷ }

▷ //P(i) = a*i mod m
▷ ll solve(ll a, ll n, ll m, ll mod){
▷ ▷ a = (a%m + m)%m;
▷ ▷ if(!a) return 0;
▷ ▷ ll ret = (n*(n+1)/2)%mod;
▷ ▷ ret = (ret*a)%mod;
▷ ▷ ll g = __gcd(a,m);
▷ ▷ ret -= m*(calc(a/g,m/g,n,mod)-n-1);
▷ ▷ return (ret%mod + mod)%mod;

▷ }
▷ //P(i) = a + r*i mod m
▷ ll solve(ll a, ll r, ll n, ll m, ll mod){
▷ ▷ a = (a%m + m)%m;
▷ ▷ r = (r%m + m)%m;
▷ ▷ if(!r) return (a*(n+1))%mod;
▷ ▷ if(!a) return solve(r, n, m, mod);
▷ ▷ ll g, x, y;
▷ ▷ g = gcdExtended(r, m, x, y);
▷ ▷ x = (x%m + m)%m;
▷ ▷ ll d = a - (a/g)*g;
▷ ▷ a -= d;
▷ ▷ x = (x*(a/g))%m;
▷ ▷ return (solve(r, n+x, m, mod) - solve(r, x-1, m, mod) + mod +
    ↪d*(n+1))%mod;
▷ }
};
```

## 6.15 Edge coloring CPP

```
const int MX = 300;
int C[MX][MX] = {}, G[MX][MX] = {};

void solve(vector<pii> &E, int N){
    int X[MX] = {}, a, b;

    auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };
    auto color = [&](int u, int v, int c){
        int p = G[u][v];
        G[u][v] = G[v][u] = c;
        C[u][c] = v; C[v][c] = u;
        C[u][p] = C[v][p] = 0;
        if( p ) X[u] = X[v] = p;
        else update(u), update(v);
        return p; };
    auto flip = [&](int u, int c1, int c2){
        int p = C[u][c1], q = C[u][c2];
        swap(C[u][c1], C[u][c2]);
        if( p ) G[u][p] = G[p][u] = c2;
        if( !C[u][c1] ) X[u] = c1;
        if( !C[u][c2] ) X[u] = c2;
        return p; };

    for(int i = 1; i <= N; i++) X[i] = 1;
    for(int t = 0; t < E.size(); t++){
        int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c
            ↪= c0, d;
        vector<pii> L;
        int vst[MX] = {};
        while(!G[u][v0]){
            L.emplace_back(v, d = X[v]);
            if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c =
                ↪color(u, L[a].first, c);
            else
                ↪if(!C[u][d])for(a=(int)L.size()-1;a>=0;a--)color(u,L[a].
            else if( vst[d] ) break;
            else vst[d] = 1, v = C[u][d];
        }
        if( !G[u][v0] ){
            for(;v; v = flip(v, c, d), swap(c, d));
            if(C[u][c0]){
                for(a = (int)L.size()-2; a >= 0 && L[a].second != c;
                    ↪a--);
                for(; a >= 0; a--) color(u, L[a].first, L[a].second);
            } else t--;
        }
    }
}
```

### 6.16 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 its given by:

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

### 6.17 Wilson's Theorem

$(n-1)! = -1 \mod n \iff n$ is prime

### 6.18 Fibonacci

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

### 6.19 Lucas's Theorem

For non-negative integers $m$ and $n$ and a prime $p$, the following congruence holds:

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p$$

where $m_i$ is the i-th digit of $m$ in base $p$. $\binom{a}{b} = 0$ if $a < b$.

### 6.20 Kirchhoff's Theorem

Laplacian matrix is $L = D - A$, where $D$ is a diagonal matrix with vertex degrees on the diagonals and $A$ is adjacency matrix.

The number of spanning trees is any cofactor of L. i-th cofactor is determinant of the matrix gotten by removing i-th row and column of L.

#### 6.20.1 Multigraphs

In $D[i][i]$ all loops are excluded. $A[i][j]$ = number of edges from $i$ to $j$.

#### 6.20.2 Directed multigraphs

$D[i][i]$ = indegree of i minus the number of loops at i. $A[i][j]$ = number of edges from $i$ to $j$.

The number of oriented spanning trees rooted at a vertex i is the determinant of the matrix gotten by removing the ith row and column of L.

### 6.21 Matroid

Let $X$ set of objects, $I \subseteq 2^X$ set of independents sets such that:

1. $\emptyset \in I$
2. $A \in I, B \subseteq A \implies B \in I$
3. Exchange axiom, $A \in I, B \in I, |B| > |A| \implies \exists x \in B \setminus A : A \cup \{x\} \in I$
4. $A \subseteq X$ and $I$ and $I'$ are maximal independent subsets of A then $|I| = |I'|$

Then $(X, I)$ is a matroid. The combinatorial optimization problem associated with it is: Given a weight $w(e) \geq 0$ $\forall e \in X$, find an independet subset that has the largest possible total weight.

### 6.22 Matroid intersection

```
// Input two matroids (X, I_a) and (X, I_b)
// output set I of maximum size, I \in I_a and I \in I_b
set<> I;
while(1){
    for(e_i : X \ I)
        if(I + e_i \in I_a and I + e_i \in I_b)
            I = I + e_i;
    set<> A, T; queue<> Q;
    for(x : X) label[x] = MARK1;
    for(e_i : X \ I){
        if(I + e_i \in I_a)
            Q.push(e_i), label[e_i] = MARK2;
        else{
            for(x such that I - x + e_i \in I_a)
                A[x].push(e_i);
        }
        if(I + e_i \in I_b)
            T = T + {e_i}
        else{
            for(x such that I - x + e_i \in I_b)
                A[e_i].push(x);
        }
    }
    if(T.empty()) break;
    bool found = false;
    while(!Q.empty() and !found){
        auto e = Q.front(); Q.pop();
        for(x : A[e]) if(label[x] == MARK1){
            label[x] = e; Q.push(x);
            if(x \in T){
                found = true; put = 1;
                while(label[x] != MARK2){
                    I = put ? (I + x) : (I - x);
                    put = 1 - put;
                }
                I = I + x;
                break;
            }
        }
    }
    if(!found) break;
}
return I;
```

Where path(e) = [e] if label[e] = MARK2, path(label[e]) + [e] otherwise.

#### 6.22.1 Matroid Union

Given $k$ matroids over the same set of objects $(X, I_1), (X, I_2), \ldots, (X, I_k)$ find $A_1 \in I_1$, $A_2 \in I_2$, $\ldots$, $A_k \in I_k$ such that $i \neq j, A_i \cap A_j = \emptyset$ and $|\bigcup_{i=1}^{k} A_i|$ is maximum. Matroid union can be reduced to matroid intersection as follows.

Let $X' = X \times \{1, 2, \ldots, k\}$, ie, $k$ copies of each element of $X$ with different colors. $M1 = (X', Q)$ where $B \in Q \iff \forall\ 1 \leq i \leq k$, $\{x \mid (x, i) \in B\} \in I_i$, ie, for each color, $B$ is independent. $M2 = (X', W)$ where $B \in W \iff i \neq j \implies \neg((x, i) \in B \wedge (x, j) \in B)$, ie, each element is picked by at most one color.

Intersection of $M1$ and $M2$ is the answer for the combinatorial problem of matroid union.

### 6.23 Notes

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

**Small to large**

**Trick in statement** If $k$ sets are given you should note that the amount of different set sizes is $O(\sqrt{s})$ where $s$ is total size of those sets. And no more than $\sqrt{s}$ sets have size greater than $\sqrt{s}$. For example, a path to the root in Aho-Corasick through suffix links will have at most $O(\sqrt{s})$ vertices.

**gcd on subsegment**, we have at most $\log(a_i)$ different values in $\{\gcd(a_j, a_{j+1}, ..., a_i)$ for $j < i\}$.

**From static set to expandable**. To insert, create a new set with the new element. While there are two sets with same size, merge them. There will be at most $\log(n)$ disjoints sets.

**Matrix exponentiation optimization**. Save binary power of $A_{nxn}$ and answer $q$ queries $b = A^m x$ in $O((n^3 +$

$qn^2)log(m))$.

**Ternary search on integers into binary search**, comparing f(mid) and f(mid+1), binary search on derivative

**Dynamic offline set** For each element we will wind segment of time $[a, b]$ such that element is present in the set during this whole segment. Now we can come up with recursive procedure which handles $[l, r]$ time segment considering that all elements such that $[l, r] \subset [a, b]$ are already included into the set. Now, keeping this invariant we recursively go into $[l, m]$ and $[m + 1, r]$ subsegments. Finally when we come into segment of length 1.

$a > b \implies a \mod b < \frac{a}{2}$

**Convex Hull**. The expected number of points in the convex hull of a random set of points is $O(log(n))$. The number of points in a convex hull with points coordinates limited by $L$ is $O(L^{2/3})$.

**Tree path query**. Sometimes the linear query is fast enough. Just do adamant's hld sorting subtrees by their size and remap vertices indexes.

**Range query** offline can be solved by a sweep, ordering queries by R.

Maximal number of divisors of any n-digit number. 7 4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680, 161280, 245760, 368640, 552960, 860160, 1290240, 1966080, 2764800, 4128768, 6193152, 8957952, 13271040, 19660800, 28311552, 41287680, 59719680, 88473600, 127401984, 181665792, 264241152, 382205952, 530841600