.*RU.*

Team Reference Document

20/11/2023

Contents

1. Code Templates

1.1. Basic Configuration.

1.2. C++ Header. A C++ header.

```
#pragma GCC optimize("Ofast","unroll-loops") --------------//c2
#pragma GCC target("avx2,fma") ----------------------------//ca
#include <bits/stdc++.h> ---------------------------------//82
using namespace std; -------------------------------------//04
#define rep(i,a,b) for (__typeof(a) i=(a); i<(b); ++i) ----//90
#define iter(it,c) for (__typeof((c).begin()) \ -----------//06
- it = (c).begin(); it != (c).end(); ++it) ----------------//f1
typedef pair<int, int> ii; -------------------------------//79
typedef vector<int> vi; ----------------------------------//2e
typedef vector<ii> vii; ----------------------------------//bf
typedef long long ll; ------------------------------------//3f
const int INF = ~(1<<31); --------------------------------//59
 ---------------------------------------------------------//c8
const double EPS = 1e-9; ---------------------------------//89
const double pi = acos(-1); ------------------------------//14
typedef unsigned long long ull; --------------------------//7b
typedef vector<vi> vvi; ----------------------------------//00
typedef vector<vii> vvii; --------------------------------//de
template <class T> T smod(T a, T b) { --------------------//66
- return (a % b + b) % b; } ------------------------------//ca
// ios::sync_with_stdio(false); --------------------------//be
```

2. Data Structures

2.1. Union-Find. An implementation of the Union-Find disjoint sets data structure.

```
struct union_find { --------------------------------------//42
- vi p; union_find(int n) : p(n, -1) { } -----------------//28
- int find(int x) { return p[x] < 0 ? x : p[x] = find(p[x]); }
- bool unite(int x, int y) { -----------------------------//6c
--- int xp = find(x), yp = find(y); ----------------------//64
```

```
--- if (xp == yp) return false; ---------------------------//0b
--- if (p[xp] > p[yp]) swap(xp,yp); -----------------------//78
--- p[xp] += p[yp], p[yp] = xp; ---------------------------//88
--- return true; } ----------------------------------------//1f
- int size(int x) { return -p[find(x)]; } }; --------------//b9
```

2.2. **Segment Tree.** An implementation of a Segment Tree.

```
#ifndef STNODE --------------------------------------------//3c
#define STNODE --------------------------------------------//69
struct node { ---------------------------------------------//89
- int l, r; -----------------------------------------------//bf
- ll x, lazy; ---------------------------------------------//b4
- node() {} -----------------------------------------------//5b
- node(int _l, int _r) : l(_l), r(_r), x(0), lazy(0) { } --//c9
- node(int _l, int _r, ll _x) : node(_l,_r) { x = _x; } ---//16
- node(node a, node b) : node(a.l,b.r) { x = a.x + b.x; } -//77
- void update(ll v) { x = v; } ----------------------------//13
- void range_update(ll v) { lazy = v; } -------------------//b5
- void apply() { x += lazy * (r - l + 1); lazy = 0; } -----//e6
- void push(node &u) { u.lazy += lazy; } }; ---------------//eb
#endif ----------------------------------------------------//fc
```

```
#ifndef STNODE --------------------------------------------//3c
#define STNODE --------------------------------------------//69
struct node { ---------------------------------------------//89
- int l, r; -----------------------------------------------//bf
- ll x, lazy; ---------------------------------------------//b4
- node() {} -----------------------------------------------//5b
- node(int _l, int _r) : l(_l), r(_r), x(INF), lazy(0) { } //7e
- node(int _l, int _r, ll _x) : node(_l,_r) { x = _x; } ---//65
- node(node a, node b) : node(a.l,b.r) { x = min(a.x, b.x); }
- void update(ll v) { x = v; } ----------------------------//0e
- void range_update(ll v) { lazy = v; } -------------------//61
- void apply() { x += lazy; lazy = 0; } -------------------//d8
- void push(node &u) { u.lazy += lazy; } }; ---------------//67
#endif ----------------------------------------------------//88
```

```
#include "segment_tree_node.cpp" --------------------------//8e
struct segment_tree { -------------------------------------//1e
- int n; --------------------------------------------------//ad
- vector<node> arr; ---------------------------------------//37
- segment_tree() { } --------------------------------------//ee
- segment_tree(const vector<ll> &a) : n(size(a)), arr(4*n) {
--- mk(a,0,0,n-1); } --------------------------------------//8c
- node mk(const vector<ll> &a, int i, int l, int r) { -----//e2
--- int m = (l+r)/2; --------------------------------------//d6
--- return arr[i] = l > r ? node(l,r) : -------------------//88
----- l == r ? node(l,r,a[l]) : ---------------------------//4c
----- node(mk(a,2*i+1,l,m),mk(a,2*i+2,m+1,r)); } ----------//49
- node update(int at, ll v, int i=0) { --------------------//37
--- propagate(i); -----------------------------------------//15
--- int hl = arr[i].l, hr = arr[i].r; ---------------------//35
--- if (at < hl || hr < at) return arr[i]; ----------------//b1
--- if (hl == at && at == hr) { --------------------------//bb
----- arr[i].update(v); return arr[i]; } ------------------//a4
--- return arr[i] = ---------------------------------------//20
----- node(update(at,v,2*i+1),update(at,v,2*i+2)); } ------//d0
- node query(int l, int r, int i=0) { ---------------------//10
--- propagate(i); -----------------------------------------//74
--- int hl = arr[i].l, hr = arr[i].r; ---------------------//5e
--- if (r < hl || hr < l) return node(hl,hr); -------------//1a
--- if (l <= hl && hr <= r) return arr[i]; ----------------//35
--- return node(query(l,r,2*i+1),query(l,r,2*i+2)); } -----//b6
- node range_update(int l, int r, ll v, int i=0) { --------//16
--- propagate(i); -----------------------------------------//d2
--- int hl = arr[i].l, hr = arr[i].r; ---------------------//6c
--- if (r < hl || hr < l) return arr[i]; ------------------//3c
--- if (l <= hl && hr <= r) -------------------------------//72
----- return arr[i].range_update(v), propagate(i), arr[i]; //f4
--- return arr[i] = node(range_update(l,r,v,2*i+1), -------//94
---------------------- range_update(l,r,v,2*i+2)); } ----//db
- void propagate(int i) { ---------------------------------//43
--- if (arr[i].l < arr[i].r) ------------------------------//ac
----- arr[i].push(arr[2*i+1]), arr[i].push(arr[2*i+2]); ---//a7
--- arr[i].apply(); } }; ----------------------------------//4a
```

2.2.1. *Persistent Segment Tree.*

```
int segcnt = 0; -------------------------------------------//cf
struct segment { ------------------------------------------//68
- int l, r, lid, rid, sum; --------------------------------//fc
} segs[2000000]; ------------------------------------------//dd
int build(int l, int r) { ---------------------------------//2b
- if (l > r) return -1; -----------------------------------//4e
- int id = segcnt++; --------------------------------------//a8
- segs[id].l = l; -----------------------------------------//90
- segs[id].r = r; -----------------------------------------//19
- if (l == r) segs[id].lid = -1, segs[id].rid = -1; -------//ee
- else { --------------------------------------------------//fe
--- int m = (l + r) / 2; ----------------------------------//14
--- segs[id].lid = build(l , m); --------------------------//e3
--- segs[id].rid = build(m + 1, r); } ---------------------//69
- segs[id].sum = 0; ---------------------------------------//21
- return id; } --------------------------------------------//c5
int update(int idx, int v, int id) { ----------------------//b8
- if (id == -1) return -1; --------------------------------//bb
- if (idx < segs[id].l || idx > segs[id].r) return id; ----//fb
- int nid = segcnt++; -------------------------------------//b3
- segs[nid].l = segs[id].l; -------------------------------//78
- segs[nid].r = segs[id].r; -------------------------------//ca
- segs[nid].lid = update(idx, v, segs[id].lid); -----------//92
- segs[nid].rid = update(idx, v, segs[id].rid); -----------//06
- segs[nid].sum = segs[id].sum + v; -----------------------//1a
- return nid; } -------------------------------------------//e6
int query(int id, int l, int r) { -------------------------//a2
- if (r < segs[id].l || segs[id].r < l) return 0; ---------//17
- if (l <= segs[id].l && segs[id].r <= r) return segs[id].sum;
- return query(segs[id].lid, l, r) ------------------------//5e
------ + query(segs[id].rid, l, r); } ---------------------//ce
```

2.3. **Fenwick Tree.** A Fenwick Tree is a data structure that represents an array of n numbers. It supports adjusting the i-th element in $O(\log n)$ time, and computing the sum of numbers in the range $i..j$ in $O(\log n)$ time. It only needs $O(n)$ space.

```
struct fenwick_tree { -------------------------------------//98
- int n; vi data; -----------------------------------------//d3
- fenwick_tree(int _n) : n(_n), data(vi(n)) { } -----------//db
- void update(int at, int by) { ---------------------------//76
--- while (at < n) data[at] += by, at |= at + 1; } --------//fb
- int query(int at) { -------------------------------------//71
--- int res = 0; ------------------------------------------//c3
--- while (at >= 0) res += data[at], at = (at & (at + 1)) - 1;
--- return res; } -----------------------------------------//e4
- int rsq(int a, int b) { return query(b) - query(a - 1); }//be
}; --------------------------------------------------------//57
struct fenwick_tree_sq { ----------------------------------//d4
- int n; fenwick_tree x1, x0; -----------------------------//18
- fenwick_tree_sq(int _n) : n(_n), x1(fenwick_tree(n)), ---//2e
--- x0(fenwick_tree(n)) { } -------------------------------//7c
- // insert f(y) = my + c if x <= y ------------------------//17
- void update(int x, int m, int c) { ----------------------//fc
--- x1.update(x, m); x0.update(x, c); } -------------------//d6
- int query(int x) { return x*x1.query(x) + x0.query(x); } //02
}; --------------------------------------------------------//ba
void range_update(fenwick_tree_sq &s, int a, int b, int k) {
- s.update(a, k, k * (1 - a)); s.update(b+1, -k, k * b); } //7b
int range_query(fenwick_tree_sq &s, int a, int b) { -------//83
- return s.query(b) - s.query(a-1); } ---------------------//31
```

2.4. **Matrix.** A Matrix class.

```
template <class K> bool eq(K a, K b) { return a == b; } ---//2a
template <> bool eq<double>(double a, double b) { ---------//f1
--- return abs(a - b) < EPS; } ----------------------------//14
template <class T> struct matrix { ------------------------//0c
- int rows, cols, cnt; vector<T> data; --------------------//b6
- inline T& at(int i, int j) { return data[i * cols + j]; }//53
- matrix(int r, int c) : rows(r), cols(c), cnt(r * c) { ---//f5
--- data.assign(cnt, T(0)); } -----------------------------//5b
- matrix(const matrix& other) : rows(other.rows), ---------//d8
--- cols(other.cols), cnt(other.cnt), data(other.data) { } //59
- T& operator()(int i, int j) { return at(i, j); } --------//db
- matrix<T> operator +(const matrix& other) { -------------//1f
--- matrix<T> res(*this); rep(i,0,cnt) --------------------//09
----- res.data[i] += other.data[i]; return res; } ---------//0d
- matrix<T> operator -(const matrix& other) { -------------//41
--- matrix<T> res(*this); rep(i,0,cnt) --------------------//9c
----- res.data[i] -= other.data[i]; return res; } ---------//b5
- matrix<T> operator *(T other) { -------------------------//5d
--- matrix<T> res(*this); ---------------------------------//72
--- rep(i,0,cnt) res.data[i] *= other; return res; } ------//7a
- matrix<T> operator *(const matrix& other) { -------------//98
--- matrix<T> res(rows, other.cols); ----------------------//96
--- rep(i,0,rows) rep(k,0,cols) rep(j,0,other.cols) -------//27
----- res(i, j) += at(i, k) * other.data[k * other.cols + j];
--- return res; } -----------------------------------------//11
- matrix<T> pow(ll p) { -----------------------------------//75
--- matrix<T> res(rows, cols), sq(*this); -----------------//82
--- rep(i,0,rows) res(i, i) = T(1); -----------------------//93
--- while (p) { -------------------------------------------//12
----- if (p & 1) res = res * sq; --------------------------//6e
----- p >>= 1; --------------------------------------------//8c
----- if (p) sq = sq * sq; --------------------------------//6a
```

```
--- } return res; } ----------------------------------------//81
- matrix<T> rref(T &det, int &rank) { -----------------------//0b
--- matrix<T> mat(*this); det = T(1), rank = 0; -------------//c9
--- for (int r = 0, c = 0; c < cols; c++) { -----------------//99
----- int k = r; --------------------------------------------//f0
----- rep(i,k+1,rows) if (abs(mat(i,c)) > abs(mat(k,c))) k = i;
----- if (k >= rows || eq<T>(mat(k, c), T(0))) continue; --//be
----- if (k != r) { -----------------------------------------//6a
------- det *= T(-1); ---------------------------------------//1b
------- rep(i,0,cols) swap(mat.at(k, i), mat.at(r, i)); ---//f8
----- } det *= mat(r, r); rank++; ---------------------------//0c
----- T d = mat(r,c); ---------------------------------------//af
----- rep(i,0,cols) mat(r, i) /= d; -------------------------//b8
----- rep(i,0,rows) { ---------------------------------------//dc
------- T m = mat(i, c); ------------------------------------//41
------- if (i != r && !eq<T>(m, T(0))) ----------------------//64
--------- rep(j,0,cols) mat(i, j) -= m * mat(r, j); -------//6f
----- } r++; ------------------------------------------------//9a
--- } return mat; } -----------------------------------------//6e
- matrix<T> transpose() { -----------------------------------//24
--- matrix<T> res(cols, rows); ------------------------------//b7
--- rep(i,0,rows) rep(j,0,cols) res(j, i) = at(i, j); -----//48
--- return res; } }; ----------------------------------------//60
```

2.5. **AVL Tree.** A fast, easily augmentable, balanced binary search tree.

```
#define AVL_MULTISET 0 --------------------------------------//b5
template <class T> ------------------------------------------//66
struct avl_tree { -------------------------------------------//b1
- struct node { ---------------------------------------------//db
--- T item; node *p, *l, *r; --------------------------------//5d
--- int size, height; ---------------------------------------//0d
--- node(const T &_item, node *_p = NULL) : item(_item), p(_p),
--- l(NULL), r(NULL), size(1), height(0) { } }; ------------//ad
- avl_tree() : root(NULL) { } -------------------------------//df
- node *root; -----------------------------------------------//15
- inline int sz(node *n) const { return n ? n->size : 0; } //6a
- inline int height(node *n) const { ------------------------//8c
--- return n ? n->height : -1; } ----------------------------//c6
- inline bool left_heavy(node *n) const { -------------------//6c
--- return n && height(n->l) > height(n->r); } -------------//33
- inline bool right_heavy(node *n) const { ------------------//c1
--- return n && height(n->r) > height(n->l); } -------------//4d
- inline bool too_heavy(node *n) const { --------------------//33
--- return n && abs(height(n->l) - height(n->r)) > 1; } ---//39
- void delete_tree(node *n) { if (n) { ----------------------//41
--- delete_tree(n->l), delete_tree(n->r); delete n; } } ---//97
- node*& parent_leg(node *n) { ------------------------------//1a
--- if (!n->p) return root; ---------------------------------//6e
--- if (n->p->l == n) return n->p->l; ----------------------//d3
--- if (n->p->r == n) return n->p->r; ----------------------//dc
--- assert(false); } ----------------------------------------//74
- void augment(node *n) { -----------------------------------//e6
--- if (!n) return; -----------------------------------------//44
--- n->size = 1 + sz(n->l) + sz(n->r); ---------------------//2e
--- n->height = 1 + max(height(n->l), height(n->r)); } ----//0a
- #define rotate(l, r) \ ----------------------------------//42
--- node *l = n->l; \ ---------------------------------------//30
--- l->p = n->p; \ ------------------------------------------//3d
--- parent_leg(n) = l; \ ------------------------------------//c7
--- n->l = l->r; \ ------------------------------------------//1e
--- if (l->r) l->r->p = n; \ --------------------------------//66
--- l->r = n, n->p = l; \ -----------------------------------//13
--- augment(n), augment(l) ----------------------------------//be
- void left_rotate(node *n) { rotate(r, l); } --------------//96
- void right_rotate(node *n) { rotate(l, r); } -------------//cf
- void fix(node *n) { ---------------------------------------//47
--- while (n) { augment(n); ---------------------------------//b0
----- if (too_heavy(n)) { -----------------------------------//d9
------- if (left_heavy(n) && right_heavy(n->l)) ------------//3c
--------- left_rotate(n->l); --------------------------------//5c
------- else if (right_heavy(n) && left_heavy(n->r)) ------//d7
--------- right_rotate(n->r); -------------------------------//2e
------- if (left_heavy(n)) right_rotate(n); ----------------//71
------- else left_rotate(n); --------------------------------//fb
------- n = n->p; } -----------------------------------------//e4
----- n = n->p; } } -----------------------------------------//93
- inline int size() const { return sz(root); } -------------//13
- node* find(const T &item) const { -------------------------//c1
--- node *cur = root; ---------------------------------------//84
--- while (cur) { -------------------------------------------//34
----- if (cur->item < item) cur = cur->r; ------------------//bf
----- else if (item < cur->item) cur = cur->l; -------------//ce
----- else break; } -----------------------------------------//aa
--- return cur; } -------------------------------------------//80
- node* insert(const T &item) { -----------------------------//2f
--- node *prev = NULL, **cur = &root; -----------------------//64
--- while (*cur) { ------------------------------------------//9a
----- prev = *cur; ------------------------------------------//78
----- if ((*cur)->item < item) cur = &((*cur)->r); --------//52
#if AVL_MULTISET --------------------------------------------//be
----- else cur = &((*cur)->l); ------------------------------//5a
#else -------------------------------------------------------//ce
----- else if (item < (*cur)->item) cur = &((*cur)->l); ---//63
----- else return *cur; -------------------------------------//8a
#endif ------------------------------------------------------//46
--- } -------------------------------------------------------//cc
--- node *n = new node(item, prev); -------------------------//1e
--- *cur = n, fix(n); return n; } ---------------------------//5b
- void erase(const T &item) { erase(find(item)); } ---------//ac
- void erase(node *n, bool free = true) { -------------------//23
--- if (!n) return; -----------------------------------------//42
--- if (!n->l && n->r) parent_leg(n) = n->r, n->r->p = n->p;
--- else if (n->l && !n->r) ---------------------------------//19
----- parent_leg(n) = n->l, n->l->p = n->p; ----------------//ab
--- else if (n->l && n->r) { --------------------------------//0c
----- node *s = successor(n); -------------------------------//12
----- erase(s, false); --------------------------------------//b0
----- s->p = n->p, s->l = n->l, s->r = n->r; ---------------//5e
----- if (n->l) n->l->p = s; --------------------------------//aa
----- if (n->r) n->r->p = s; --------------------------------//6c
----- parent_leg(n) = s, fix(s); ----------------------------//c7
----- return; -----------------------------------------------//0e
--- } else parent_leg(n) = NULL; ----------------------------//fc
--- fix(n->p), n->p = n->l = n->r = NULL; ------------------//a0
--- if (free) delete n; } -----------------------------------//f6
- node* successor(node *n) const { --------------------------//c0
--- if (!n) return NULL; ------------------------------------//07
--- if (n->r) return nth(0, n->r); --------------------------//6c
--- node *p = n->p; -----------------------------------------//ed
--- while (p && p->r == n) n = p, p = p->p; ----------------//54
--- return p; } ---------------------------------------------//15
- node* predecessor(node *n) const { ------------------------//12
--- if (!n) return NULL; ------------------------------------//c7
--- if (n->l) return nth(n->l->size-1, n->l); --------------//e1
--- node *p = n->p; -----------------------------------------//11
--- while (p && p->l == n) n = p, p = p->p; ----------------//ec
--- return p; } ---------------------------------------------//5e
- node* nth(int n, node *cur = NULL) const { ----------------//ab
--- if (!cur) cur = root; -----------------------------------//6d
--- while (cur) { -------------------------------------------//45
----- if (n < sz(cur->l)) cur = cur->l; --------------------//2e
----- else if (n > sz(cur->l)) ------------------------------//b4
------- n -= sz(cur->l) + 1, cur = cur->r; -----------------//28
----- else break; -------------------------------------------//c5
--- } return cur; } -----------------------------------------//2d
- int count_less(node *cur) { --------------------------------//f7
--- int sum = sz(cur->l); -----------------------------------//1f
--- while (cur) { -------------------------------------------//03
----- if (cur->p && cur->p->r == cur) sum += 1 + sz(cur->p->l);
----- cur = cur->p; -----------------------------------------//b8
--- } return sum; } -----------------------------------------//32
- void clear() { delete_tree(root), root = NULL; } }; -----//b8
```

Also a very simple wrapper over the AVL tree that implements a map interface.

```
#include "avl_tree.cpp" -------------------------------------//01
template <class K, class V> struct avl_map { ---------------//dc
- struct node { ---------------------------------------------//58
--- K key; V value; -----------------------------------------//78
--- node(K k, V v) : key(k), value(v) { } ------------------//89
--- bool operator <(const node &other) const { -------------//bb
----- return key < other.key; } }; --------------------------//4b
- avl_tree<node> tree; --------------------------------------//f9
- V& operator [](K key) { -----------------------------------//e6
--- typename avl_tree<node>::node *n = ----------------------//45
----- tree.find(node(key, V(0))); ---------------------------//d6
--- if (!n) n = tree.insert(node(key, V(0))); --------------//c8
--- return n->item.value; } }; ------------------------------//1f
```

2.6. **Dancing Links.** An implementation of Donald Knuth's Dancing Links data structure. A linked list supporting deletion and restoration of elements.

```
template <class T> ------------------------------------------//82
struct dancing_links { --------------------------------------//9e
- struct node { ---------------------------------------------//62
--- T item; -------------------------------------------------//dd
--- node *l, *r; --------------------------------------------//32
--- node(const T &_item, node *_l = NULL, node *_r = NULL) //6d
----- : item(_item), l(_l), r(_r) { -------------------------//6d
```

```
----- if (l) l->r = this; ---------------------------------//97
----- if (r) r->l = this; } }; ----------------------------//37
- node *front, *back; --------------------------------------//f7
- dancing_links() { front = back = NULL; } ----------------//cb
- node *push_back(const T &item) { -------------------------//4a
--- back = new node(item, back, NULL); --------------------//5c
--- if (!front) front = back; ------------------------------//7b
--- return back; } -----------------------------------------//55
- node *push_front(const T &item) { ------------------------//c0
--- front = new node(item, NULL, front); ------------------//a0
--- if (!back) back = front; -------------------------------//8b
--- return front; } ----------------------------------------//95
- void erase(node *n) { ------------------------------------//c3
--- if (!n->l) front = n->r; else n->l->r = n->r; ---------//38
--- if (!n->r) back = n->l; else n->r->l = n->l; } --------//8e
- void restore(node *n) { ----------------------------------//0e
--- if (!n->l) front = n; else n->l->r = n; ---------------//f4
--- if (!n->r) back = n; else n->r->l = n; } }; -----------//6d
```

2.7. **Misof Tree.** A simple tree data structure for inserting, erasing, and querying the nth largest element.

```
#define BITS 15 --------------------------------------------//7b
struct misof_tree { ----------------------------------------//fe
- int cnt[BITS][1<<BITS]; ----------------------------------//aa
- misof_tree() { memset(cnt, 0, sizeof(cnt)); } -----------//b0
- void insert(int x) { -------------------------------------//7f
--- for (int i = 0; i < BITS; cnt[i++][x]++, x >>= 1); } --//e2
- void erase(int x) { --------------------------------------//c8
--- for (int i = 0; i < BITS; cnt[i++][x]--, x >>= 1); } --//d4
- int nth(int n) { -----------------------------------------//c4
--- int res = 0; -------------------------------------------//cb
--- for (int i = BITS-1; i >= 0; i--) ---------------------//ba
----- if (cnt[i][res <<= 1] <= n) n -= cnt[i][res], res |= 1;
--- return res; } }; ---------------------------------------//89
```

2.8. **k-d Tree.** A k-dimensional tree supporting fast construction, adding points, and nearest neighbor queries.

```
#define INC(c) ((c) == K - 1 ? 0 : (c) + 1) ---------------//77
template <int K> struct kd_tree { --------------------------//93
- struct pt { -----------------------------------------------//99
--- double coord[K]; ---------------------------------------//31
--- pt() {} ------------------------------------------------//96
--- pt(double c[K]) { rep(i,0,K) coord[i] = c[i]; } -------//37
--- double dist(const pt &other) const { ------------------//16
----- double sum = 0.0; ------------------------------------//0c
----- rep(i,0,K) sum += pow(coord[i] - other.coord[i], 2.0);
----- return sqrt(sum); } }; -------------------------------//68
- struct cmp { ---------------------------------------------//8c
--- int c; -------------------------------------------------//fa
--- cmp(int _c) : c(_c) {} ---------------------------------//28
--- bool operator ()(const pt &a, const pt &b) const { ----//89
----- for (int i = 0, cc; i <= K; i++) { -------------------//bf
------- cc = i == 0 ? c : i - 1; ---------------------------//90
------- if (abs(a.coord[cc] - b.coord[cc]) > EPS) ---------//1b
--------- return a.coord[cc] < b.coord[cc]; ----------------//8b
----- } ----------------------------------------------------//70
----- return false; } }; -----------------------------------//66
- struct bb { -----------------------------------------------//56
--- pt from, to; -------------------------------------------//d7
--- bb(pt _from, pt _to) : from(_from), to(_to) {} --------//2e
--- double dist(const pt &p) { -----------------------------//ae
----- double sum = 0.0; ------------------------------------//28
----- rep(i,0,K) { -----------------------------------------//e4
------- if (p.coord[i] < from.coord[i]) -------------------//82
--------- sum += pow(from.coord[i] - p.coord[i], 2.0); ----//31
------- else if (p.coord[i] > to.coord[i]) ----------------//bd
--------- sum += pow(p.coord[i] - to.coord[i], 2.0); ------//c8
----- } ----------------------------------------------------//5f
----- return sqrt(sum); } ----------------------------------//ff
--- bb bound(double l, int c, bool left) { -----------------//54
----- pt nf(from.coord), nt(to.coord); ---------------------//e7
----- if (left) nt.coord[c] = min(nt.coord[c], l); --------//62
----- else nf.coord[c] = max(nf.coord[c], l); -------------//05
----- return bb(nf, nt); } }; ------------------------------//01
- struct node { ---------------------------------------------//0b
--- pt p; node *l, *r; --------------------------------------//dd
--- node(pt _p, node *_l, node *_r) ------------------------//21
----- : p(_p), l(_l), r(_r) {  } }; ------------------------//c8
- node *root; ----------------------------------------------//8f
- // kd_tree() : root(NULL) { } -----------------------------//d5
- kd_tree(vector<pt> pts) { ---------------------------------//49
--- root = construct(pts, 0, (int)size(pts) - 1, 0); } ----//0c
- node* construct(vector<pt> &pts, int from, int to, int c) {
--- if (from > to) return NULL; ----------------------------//ee
--- int mid = from + (to - from) / 2; ----------------------//82
--- nth_element(pts.begin() + from, pts.begin() + mid, ----//43
--------- pts.begin() + to + 1, cmp(c)); -------------------//b6
--- return new node(pts[mid], ------------------------------//68
----------- construct(pts, from, mid - 1, INC(c)), --------//c0
----------- construct(pts, mid + 1, to, INC(c))); } -------//15
- bool contains(const pt &p) { return _con(p, root, 0); } -//d2
- bool _con(const pt &p, node *n, int c) { -----------------//ab
--- if (!n) return false; ----------------------------------//59
--- if (cmp(c)(p, n->p)) return _con(p, n->l, INC(c)); ----//98
--- if (cmp(c)(n->p, p)) return _con(p, n->r, INC(c)); ----//fe
--- return true; } -----------------------------------------//73
- void insert(const pt &p) { _ins(p, root, 0); } ----------//4c
- void _ins(const pt &p, node* &n, int c) { ----------------//c4
--- if (!n) n = new node(p, NULL, NULL); -------------------//d4
--- else if (cmp(c)(p, n->p)) _ins(p, n->l, INC(c)); ------//f9
--- else if (cmp(c)(n->p, p)) _ins(p, n->r, INC(c)); } ----//1c
- void clear() { _clr(root); root = NULL; } ----------------//d2
- void _clr(node *n) { --------------------------------------//af
--- if (n) _clr(n->l), _clr(n->r), delete n; } ------------//94
- pair<pt, bool> nearest_neighbour(const pt &p, -----------//29
----- bool allow_same=true) { -------------------------------//8d
--- double mn = INFINITY, cs[K]; ---------------------------//7e
--- rep(i,0,K) cs[i] = -INFINITY; --------------------------//34
--- pt from(cs); --------------------------------------------//bc
--- rep(i,0,K) cs[i] = INFINITY; ---------------------------//1a
--- pt to(cs), resp; ----------------------------------------//aa
--- _nn(p, root, bb(from, to), mn, resp, 0, allow_same); --//8f
--- return make_pair(resp, !std::isinf(mn)); } ------------//bc
- void _nn(const pt &p, node *n, bb b, ---------------------//64
----- double &mn, pt &resp, int c, bool same) { -----------//5c
--- if (!n || b.dist(p) > mn) return; ----------------------//bc
--- bool l1 = true, l2 = false; ----------------------------//d8
--- if ((same || p.dist(n->p) > EPS) && p.dist(n->p) < mn) //b0
------- mn = p.dist(resp = n->p); --------------------------//05
--- node *n1 = n->l, *n2 = n->r; ---------------------------//73
--- rep(i,0,2) { --------------------------------------------//02
----- if (i == 1 || cmp(c)(n->p, p)) swap(n1,n2),swap(l1,l2);
----- _nn(p, n1, b.bound(n->p.coord[c], c, l1), mn, -------//b3
--------- resp, INC(c), same); } } }; ----------------------//1b
```

2.9. **Sqrt Decomposition.** Design principle that supports many operations in amortized $\sqrt{n}$ per operation.

```
struct segment { -------------------------------------------//b2
- vi arr; --------------------------------------------------//8c
- segment(vi _arr) : arr(_arr) { } }; ----------------------//11
vector&lt;segment&gt; T; -----------------------------------------//a1
int K; ------------------------------------------------------//dc
void rebuild() { -------------------------------------------//17
- int cnt = 0; ----------------------------------------------//14
- rep(i,0,size(T)) -----------------------------------------//b1
--- cnt += size(T[i].arr); ---------------------------------//d1
- K = static_cast<int>(ceil(sqrt(cnt)) + 1e-9); ----------//4c
- vi arr(cnt); ---------------------------------------------//14
- for (int i = 0, at = 0; i < size(T); i++) ---------------//79
--- rep(j,0,size(T[i].arr)) --------------------------------//a4
----- arr[at++] = T[i].arr[j]; -----------------------------//f7
- T.clear(); ------------------------------------------------//4c
- for (int i = 0; i < cnt; i += K) -------------------------//79
--- T.push_back(segment(vi(arr.begin()+i, -----------------//13
------------------------- arr.begin()+min(i+K, cnt)))); } //d5
int split(int at) { -----------------------------------------//13
- int i = 0; ------------------------------------------------//b5
- while (i < size(T) && at >= size(T[i].arr)) -------------//ea
--- at -= size(T[i].arr), i++; -----------------------------//e8
- if (i >= size(T)) return size(T); ------------------------//df
- if (at == 0) return i; -----------------------------------//42
- T.insert(T.begin() + i + 1, ------------------------------//bc
----- segment(vi(T[i].arr.begin() + at, T[i].arr.end()))); //34
- T[i] = segment(vi(T[i].arr.begin(), T[i].arr.begin() + at));
- return i + 1; } ------------------------------------------//87
void insert(int at, int v) { -------------------------------//9a
- vi arr; arr.push_back(v); --------------------------------//f3
- T.insert(T.begin() + split(at), segment(arr)); } --------//e7
void erase(int at) { ---------------------------------------//06
- int i = split(at); split(at + 1); ------------------------//ec
- T.erase(T.begin() + i); } --------------------------------//a9
```

2.10. **Monotonic Queue.** A queue that supports querying for the minimum element. Useful for sliding window algorithms.

```
struct min_stack { -----------------------------------------//d8
- stack<int> S, M; -----------------------------------------//fe
- void push(int x) { ---------------------------------------//20
--- S.push(x); ---------------------------------------------//e2
--- M.push(M.empty() ? x : min(M.top(), x)); } ------------//92
- int top() { return S.top(); } ----------------------------//f1
```

```
- int mn() { return M.top(); } ----------------------------//02
- void pop() { S.pop(); M.pop(); } ------------------------//fd
- bool empty() { return S.empty(); } }; -------------------//ed
struct min_queue { ----------------------------------------//90
- min_stack inp, outp; ------------------------------------//ed
- void push(int x) { inp.push(x); } -----------------------//b3
- void fix() { --------------------------------------------//0a
--- if (outp.empty()) while (!inp.empty()) ----------------//76
----- outp.push(inp.top()), inp.pop(); } ------------------//67
- int top() { fix(); return outp.top(); } -----------------//c0
- int mn() { ----------------------------------------------//79
--- if (inp.empty()) return outp.mn(); --------------------//d2
--- if (outp.empty()) return inp.mn(); --------------------//6e
--- return min(inp.mn(), outp.mn()); } --------------------//c3
- void pop() { fix(); outp.pop(); } -----------------------//61
- bool empty() { return inp.empty() && outp.empty(); } }; -//89
```

2.11. **Convex Hull Trick.** If converting to integers, look out for division by 0 and $\pm\infty$.

```
struct convex_hull_trick { --------------------------------//16
- vector<pair<double,double> > h; -------------------------//b4
- double intersect(int i) { -------------------------------//9b
--- return (h[i+1].second-h[i].second) / ------------------//43
----- (h[i].first-h[i+1].first); } ------------------------//2e
- void add(double m, double b) { --------------------------//c4
--- h.push_back(make_pair(m,b)); --------------------------//67
--- while (size(h) >= 3) { --------------------------------//85
----- int n = size(h); ------------------------------------//b0
----- if (intersect(n-3) < intersect(n-2)) break; ---------//b3
----- swap(h[n-2], h[n-1]); -------------------------------//1c
----- h.pop_back(); } } -----------------------------------//1f
- double get_min(double x) { ------------------------------//ad
--- int lo = 0, hi = (int)size(h) - 2, res = -1; ----------//ed
--- while (lo <= hi) { ------------------------------------//c3
----- int mid = lo + (hi - lo) / 2; -----------------------//c9
----- if (intersect(mid) <= x) res = mid, lo = mid + 1; ---//2d
----- else hi = mid - 1; } --------------------------------//cb
--- return h[res+1].first * x + h[res+1].second; } }; -----//1b
```

And dynamic variant:

```
const ll is_query = -(1LL<<62); ---------------------------//49
struct Line { ---------------------------------------------//f1
- ll m, b; ------------------------------------------------//28
- mutable function<const Line*()> succ; -------------------//44
- bool operator<(const Line& rhs) const { -----------------//28
--- if (rhs.b != is_query) return m < rhs.m; --------------//1e
--- const Line* s = succ(); -------------------------------//90
--- if (!s) return 0; -------------------------------------//c5
--- ll x = rhs.m; -----------------------------------------//ce
--- return b - s->b < (s->m - m) * x; } }; ----------------//67
// will maintain upper hull for maximum -------------------//d4
struct HullDynamic : public multiset<Line> { --------------//90
- bool bad(iterator y) { ----------------------------------//a9
--- auto z = next(y); -------------------------------------//39
--- if (y == begin()) { -----------------------------------//ad
----- if (z == end()) return 0; ---------------------------//ed
----- return y->m == z->m && y->b <= z->b; } --------------//57
--- auto x = prev(y); -------------------------------------//42
--- if (z == end()) return y->m == x->m && y->b <= x->b; --//20
--- return (x->b - y->b)*(z->m - y->m) >= -----------------//97
--------- (y->b - z->b)*(y->m - x->m); } ------------------//1f
- void insert_line(ll m, ll b) { --------------------------//7b
--- auto y = insert({ m, b }); ----------------------------//24
--- y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
--- if (bad(y)) { erase(y); return; } ---------------------//ab
--- while (next(y) != end() && bad(next(y))) erase(next(y));
--- while (y != begin() && bad(prev(y))) erase(prev(y)); } //8e
- ll eval(ll x) { -----------------------------------------//1e
--- auto l = *lower_bound((Line) { x, is_query }); --------//ef
--- return l.m * x + l.b; } }; ----------------------------//08
```

2.12. **Sparse Table.**

```
struct sparse_table { vvi m; ------------------------------//ed
- sparse_table(vi arr) { ----------------------------------//cd
--- m.push_back(arr); -------------------------------------//cb
--- for (int k = 0; (1<<(++k)) <= size(arr); ) { ----------//19
----- m.push_back(vi(size(arr)-(1<<k)+1)); ----------------//8e
----- rep(i,0,size(arr)-(1<<k)+1) -------------------------//fd
------- m[k][i] = min(m[k-1][i], m[k-1][i+(1<<(k-1))]); } }//05
- int query(int l, int r) { -------------------------------//e1
--- int k = 0; while (1<<(k+1) <= r-l+1) k++; -------------//fa
--- return min(m[k][l], m[k][r-(1<<k)+1]); } }; -----------//70
```

2.13. **Splay Tree.** Supports the following queries

1 L R Output Maximum value in range [L,R]
2 L R Reverse the array [L,R]
3 L R v add v in range [L,R]
4 pos removes entry from pos
5 pos v - insert an element after position v

```
typedef ll T; ---------------------------------------------//6d
const int N = 5e5+50; // >= Node + Query ------------------//84
T ar[N]; // Initial Array ---------------------------------//7c
struct Node { ---------------------------------------------//b1
--- Node *ch[2],*pre; -------------------------------------//8e
--- T val, mx, sum, add; ----------------------------------//65
--- bool rev, flip; ---------------------------------------//94
--- int size; ---------------------------------------------//7b
--- Node() {size=0;val=mx=-1e9;add=0;} --------------------//b5
--- void addIt(T ad) { ------------------------------------//7c
------- add+=ad; mx+=ad; ----------------------------------//c0
------- sum+=size*ad; val+=ad; } --------------------------//33
--- void revIt() {rev^=1;} --------------------------------//71
--- void upd() { -----------------------------------------//96
------- size=ch[0]->size+ch[1]->size+1; -------------------//97
------- mx=max(val,max(ch[0]->mx,ch[1]->mx)); -------------//91
------- sum= ch[0]->sum + ch[1]->sum + val; } -------------//0f
--- void pushdown(); --------------------------------------//bf
} Tnull,*null=&Tnull; -------------------------------------//9d
void Node::pushdown(){ ------------------------------------//8e
--- if (add!=0){ ------------------------------------------//c1
------- for (int i=0;i<2;++i) -----------------------------//4e
----------- if (ch[i]!=null) ch[i]->addIt(add); -----------//91
------- add = 0; } ----------------------------------------//e0
--- if (rev){ ---------------------------------------------//a1
------- swap(ch[0],ch[1]); --------------------------------//6a
------- for (int i=0;i<2;i++) -----------------------------//c6
----------- if (ch[i]!=null) ch[i]->revIt(); --------------//18
------- flip ^= 1; rev = 0;} } ----------------------------//c0
struct Splay{ ---------------------------------------------//51
--- Node nodePool[N],*cur,*root; --------------------------//ee
--- Splay() { cur=nodePool; root=null; } ------------------//85
--- void clear() { cur=nodePool; root=null; } -------------//18
--- Node* newNode(T v,Node* f) { --------------------------//6e
------- cur->ch[0]=cur->ch[1]=null; -----------------------//30
------- cur->size=1; cur->val=v; --------------------------//73
------- cur->mx=v; cur->sum = 0; --------------------------//04
------- cur->add=0; cur->pre=f; ---------------------------//e1
------- cur->rev = cur->flip=0; ---------------------------//21
------- return cur++; } -----------------------------------//6e
--- Node* build(int l,int r,Node* f) { --------------------//58
------- if(l>r) return null; ------------------------------//d2
------- int m=(l+r)>>1; -----------------------------------//67
------- Node* t=newNode(ar[m],f); -------------------------//c0
------- t->ch[0]=build(l,m-1,t); --------------------------//cc
------- t->ch[1]=build(m+1,r,t); --------------------------//15
------- t->upd(); -----------------------------------------//ec
------- return t; } ---------------------------------------//e2
--- void rotate(Node* x,int c) { --------------------------//6d
------- Node* y=x->pre; -----------------------------------//d1
------- y->pushdown(); x->pushdown(); ---------------------//1c
------- y->ch[!c]=x->ch[c]; -------------------------------//de
------- if (x->ch[c]!=null) x->ch[c]->pre=y; --------------//b3
------- x->pre=y->pre; ------------------------------------//22
------- if (y->pre!=null) { -------------------------------//8a
----------- if (y->pre->ch[0]==y) y->pre->ch[0]=x; --------//da
----------- else y->pre->ch[1]=x; } -----------------------//2e
------- x->ch[c]=y; y->pre=x; y->upd(); -------------------//fd
------- if (y==root) root=x; } ----------------------------//56
--- void splay(Node* x,Node* f) { -------------------------//92
------- x->pushdown(); ------------------------------------//5e
------- while (x->pre!=f) { -------------------------------//d0
----------- if (x->pre->pre==f) { -------------------------//75
--------------- if (x->pre->ch[0]==x) rotate(x,1); --------//1f
--------------- else rotate(x,0); -------------------------//99
----------- } else { --------------------------------------//74
--------------- Node *y=x->pre,*z=y->pre; -----------------//1f
--------------- if (z->ch[0]==y) { ------------------------//b7
------------------- if (y->ch[0]==x) rotate(y,1),rotate(x,1);
------------------- else rotate(x,0),rotate(x,1); ---------//9f
--------------- } else { ----------------------------------//43
------------------- if (y->ch[1]==x) rotate(y,0),rotate(x,0);
------------------- else rotate(x,1),rotate(x,0); } } } ---//ed
------- x->upd(); } ---------------------------------------//a0
--- void select(int k,Node* f) { --------------------------//08
------- int tmp; Node* x=root; ----------------------------//d0
------- x->pushdown(); k++; -------------------------------//d8
------- for(;;){ ------------------------------------------//69
----------- x->pushdown(); --------------------------------//26
----------- tmp=x->ch[0]->size; ---------------------------//a9
----------- if (k==tmp+1) break; --------------------------//60
```

```
----------- if (k<=tmp) x=x->ch[0]; ------------------------//cd
----------- else { -----------------------------------------//a7
--------------- k-=tmp+1; ----------------------------------//ca
--------------- x=x->ch[1]; } } ----------------------------//af
------- splay(x,f); } --------------------------------------//2f
--- Node*&get(int l, int r) { ------------------------------//53
------- select(l-1,null); select(r+1,root); ----------------//91
------- return root->ch[1]->ch[0]; } -----------------------//ee
--- void reverse(int l,int r) { ----------------------------//96
------- Node* o=get(l,r); ----------------------------------//6d
------- o->rev^=1; -----------------------------------------//1e
------- splay(o,null); } -----------------------------------//cf
--- void del(int p) { --------------------------------------//1d
----------- select(p-1,null); select(p+1,root); ------------//a8
----------- root->ch[1]->ch[0] = null; ---------------------//f0
----------- splay(root->ch[1],null); } ---------------------//6f
--- void split(int l,int r,Node*&s1) { ---------------------//62
------- Node* tmp=get(l,r); --------------------------------//a1
------- root->ch[1]->ch[0]=null; ---------------------------//8d
------- root->ch[1]->upd(); --------------------------------//06
------- root->upd(); ---------------------------------------//04
------- s1=tmp; } ------------------------------------------//7c
--- void cut(int l,int r) { --------------------------------//a5
------- Node* tmp; -----------------------------------------//11
------- split(l,r,tmp); ------------------------------------//ad
------- select(root->size-2,null); -------------------------//29
------- root->ch[1]->ch[0]=tmp; ----------------------------//28
------- tmp->pre=root->ch[1]; ------------------------------//31
------- root->ch[1]->upd(); --------------------------------//7a
------- root->upd(); } -------------------------------------//72
--- void init(int n) { -------------------------------------//11
------- clear(); -------------------------------------------//66
------- root=newNode(0,null); ------------------------------//ee
------- root->ch[1]=newNode(n+1,root); ---------------------//23
------- root->ch[1]->ch[0]=build(1,n,root->ch[1]); ---------//d6
------- splay(root->ch[1]->ch[0],null); } ------------------//c6
--- void insertPos(int pos,T v) { --------------------------//38
------- select(pos,null); select(pos+1,root); --------------//54
------- root->ch[1]->ch[0] = newNode(v,root->ch[1]); -------//6e
------- splay(root->ch[1]->ch[0],null); } ------------------//5e
--- void insertRange(int pos,Node *s) { --------------------//71
------- select(pos,null); select(pos+1,root); --------------//ec
------- root->ch[1]->ch[0] = s; ----------------------------//c6
------- s->pre = root->ch[1]; ------------------------------//2f
------- root->ch[1]->upd(); --------------------------------//3b
------- root->upd(); } -------------------------------------//aa
--- T query(int l,int r) { ---------------------------------//56
------- Node *o = get(l,r); --------------------------------//32
------- return o->mx; } ------------------------------------//ec
--- void addRange(int l,int r,T v) { -----------------------//62
------- Node *o = get(l,r); --------------------------------//d2
------- o->add+=v; o->val+=v; ------------------------------//af
------- o->sum+=o->size * v; -------------------------------//28
------- splay(o,null); } -----------------------------------//37
--- void output(int l,int r) { -----------------------------//a6
------- for (int i=l;i<=r;i++){ ----------------------------//ad
----------- select(i,null); --------------------------------//e8
----------- cout << root->val << ' '; ----------------------//78
------- }; cout << endl; } } st; ---------------------------//b6
```

3. Graphs

3.1. Single-Source Shortest Paths.

3.1.1. *Dijkstra's algorithm.* An implementation of Dijkstra's algorithm. It runs in $\Theta(|E|\log|V|)$ time.

```
int *dist, *dad; -------------------------------------------//46
struct cmp { -----------------------------------------------//a5
- bool operator()(int a, int b) const { --------------------//6a
--- return dist[a] != dist[b] ? dist[a] < dist[b] : a < b; }
}; ---------------------------------------------------------//c1
pair<int*, int*> dijkstra(int n, int s, vii *adj) { --------//37
- dist = new int[n]; ---------------------------------------//c3
- dad = new int[n]; ----------------------------------------//c9
- rep(i,0,n) dist[i] = INF, dad[i] = -1; -------------------//8c
- set<int, cmp> pq; ----------------------------------------//32
- dist[s] = 0, pq.insert(s); -------------------------------//eb
- while (!pq.empty()) { ------------------------------------//10
--- int cur = *pq.begin(); pq.erase(pq.begin()); ----------//7f
--- rep(i,0,size(adj[cur])) { ------------------------------//5e
----- int nxt = adj[cur][i].first, -------------------------//b0
------- ndist = dist[cur] + adj[cur][i].second; ------------//1d
----- if (ndist < dist[nxt]) pq.erase(nxt), ----------------//2c
------- dist[nxt] = ndist, dad[nxt] = cur, pq.insert(nxt); //77
--- } } ----------------------------------------------------//45
- return pair<int*, int*>(dist, dad); } --------------------//44
```

3.1.2. *Bellman-Ford algorithm.* The Bellman-Ford algorithm solves the single-source shortest paths problem in $O(|V||E|)$ time. It is slower than Dijkstra's algorithm, but it works on graphs with negative edges and has the ability to detect negative cycles, neither of which Dijkstra's algorithm can do.

```
int* bellman_ford(int n, int s, vii* adj, bool& ncycle) { -//07
- ncycle = false; ------------------------------------------//00
- int* dist = new int[n]; ----------------------------------//62
- rep(i,0,n) dist[i] = i == s ? 0 : INF; -------------------//a6
- rep(i,0,n-1) rep(j,0,n) if (dist[j] != INF) --------------//f1
--- rep(k,0,size(adj[j])) ----------------------------------//20
----- dist[adj[j][k].first] = min(dist[adj[j][k].first], -//c2
--------- dist[j] + adj[j][k].second); ---------------------//2a
- rep(j,0,n) rep(k,0,size(adj[j])) { -----------------------//4c
----- if (dist[j] == INF) continue; ------------------------//30
----- if (dist[j] + adj[j][k].second < dist[adj[j][k].first])
------- ncycle = true; -------------------------------------//e4
--- } ------------------------------------------------------//8f
- return dist; } -------------------------------------------//91
```

3.1.3. *IDA* algorithm.*

```
int n, cur[100], pos; --------------------------------------//48
int calch() { ----------------------------------------------//88
- int h = 0; -----------------------------------------------//4a
- rep(i,0,n) if (cur[i] != 0) h += abs(i - cur[i]); -------//9b
- return h; } ----------------------------------------------//f8
int dfs(int d, int g, int prev) { --------------------------//e5
- int h = calch(); -----------------------------------------//ef
- if (g + h > d) return g + h; -----------------------------//39
- if (h == 0) return 0; ------------------------------------//f6
- int mn = INF; --------------------------------------------//44
- rep(di,-2,3) { -------------------------------------------//61
--- if (di == 0) continue; ---------------------------------//ab
--- int nxt = pos + di; ------------------------------------//45
--- if (nxt == prev) continue; -----------------------------//fc
--- if (0 <= nxt && nxt < n) { -----------------------------//82
----- swap(cur[pos], cur[nxt]); ----------------------------//9c
----- swap(pos,nxt); ---------------------------------------//af
----- mn = min(mn, dfs(d, g+1, nxt)); ----------------------//63
----- swap(pos,nxt); ---------------------------------------//8c
----- swap(cur[pos], cur[nxt]); } --------------------------//e1
--- if (mn == 0) break; } ----------------------------------//5a
- return mn; } ---------------------------------------------//89
int idastar() { --------------------------------------------//49
- rep(i,0,n) if (cur[i] == 0) pos = i; ---------------------//0a
- int d = calch(); -----------------------------------------//57
- while (true) { -------------------------------------------//de
--- int nd = dfs(d, 0, -1); --------------------------------//2a
--- if (nd == 0 || nd == INF) return d; --------------------//bd
--- d = nd; } } --------------------------------------------//7a
```

3.2. All-Pairs Shortest Paths.

3.2.1. *Floyd-Warshall algorithm.* The Floyd-Warshall algorithm solves the all-pairs shortest paths problem in $O(|V|^3)$ time.

```
void floyd_warshall(int** arr, int n) { --------------------//21
- rep(k,0,n) rep(i,0,n) rep(j,0,n) -------------------------//af
--- if (arr[i][k] != INF && arr[k][j] != INF) --------------//84
----- arr[i][j] = min(arr[i][j], arr[i][k] + arr[k][j]); --//39
- // Check negative cycles -----------------------------------//ee
- rep(i,0,n) rep(j,0,n) rep(k,0,n) -------------------------//2e
--- if (arr[i][k] != INF && arr[k][k] < 0 && arr[k][j]!=INF)
----- arr[i][j] = -INF; } ----------------------------------//eb
```

3.3. Strongly Connected Components.

3.3.1. *Kosaraju's algorithm.* Kosarajus's algorithm finds strongly connected components of a directed graph in $O(|V|+|E|)$ time. Returns a Union-Find of the SCCs, as well as a topological ordering of the SCCs. Note that the ordering specifies a random element from each SCC, not the UF parents!

```
#include "../data-structures/union_find.cpp" --------------//5e
vector<bool> visited; --------------------------------------//ab
vi order; --------------------------------------------------//b0
void scc_dfs(const vvi &adj, int u) { ----------------------//f8
- int v; visited[u] = true; --------------------------------//82
- rep(i,0,size(adj[u])) ------------------------------------//59
--- if (!visited[v = adj[u][i]]) scc_dfs(adj, v); ---------//c8
- order.push_back(u); } ------------------------------------//c9
pair<union_find, vi> scc(const vvi &adj) { -----------------//59
- int n = size(adj), u, v; ---------------------------------//3e
- order.clear(); -------------------------------------------//09
- union_find uf(n); vi dag; vvi rev(n); --------------------//bf
- rep(i,0,n) rep(j,0,size(adj[i])) rev[adj[i][j]].push_back(i);
- visited.resize(n); ---------------------------------------//60
```

```
- fill(visited.begin(), visited.end(), false); ------------//96
- rep(i,0,n) if (!visited[i]) scc_dfs(rev, i); ------------//35
- fill(visited.begin(), visited.end(), false); ------------//17
- stack<int> S; -------------------------------------------//e3
- for (int i = n-1; i >= 0; i--) { ----------------------//ee
--- if (visited[order[i]]) continue; ----------------------//99
--- S.push(order[i]), dag.push_back(order[i]); ------------//91
--- while (!S.empty()) { ----------------------------------//9e
----- visited[u = S.top()] = true, S.pop(); ---------------//5b
----- uf.unite(u, order[i]); ------------------------------//81
----- rep(j,0,size(adj[u])) -------------------------------//c5
------- if (!visited[v = adj[u][j]]) S.push(v); } } -------//d0
- return pair<union_find, vi>(uf, dag); } -----------------//04
```

3.4. **Cut Points and Bridges.**

```
#define MAXN 5000 -----------------------------------------//f7
int low[MAXN], num[MAXN], curnum; -------------------------//d7
void dfs(const vvi &adj, vi &cp, vii &bri, int u, int p) { //22
- low[u] = num[u] = curnum++; -----------------------------//a3
- int cnt = 0; bool found = false; ------------------------//97
- rep(i,0,size(adj[u])) { ---------------------------------//ae
--- int v = adj[u][i]; ------------------------------------//56
--- if (num[v] == -1) { -----------------------------------//3b
----- dfs(adj, cp, bri, v, u); ----------------------------//ba
----- low[u] = min(low[u], low[v]); -----------------------//be
----- cnt++; ----------------------------------------------//e0
----- found = found || low[v] >= num[u]; ------------------//30
----- if (low[v] > num[u]) bri.push_back(ii(u, v)); -------//bf
--- } else if (p != v) low[u] = min(low[u], num[v]); } ----//76
- if (found && (p != -1 || cnt > 1)) cp.push_back(u); } ---//3e
pair<vi,vii> cut_points_and_bridges(const vvi &adj) { -----//76
- int n = size(adj); --------------------------------------//c8
- vi cp; vii bri; -----------------------------------------//fb
- memset(num, -1, n << 2); --------------------------------//45
- curnum = 0; ---------------------------------------------//07
- rep(i,0,n) if (num[i] == -1) dfs(adj, cp, bri, i, -1); --//7e
- return make_pair(cp, bri); } ----------------------------//4c
```

3.5. **Minimum Spanning Tree.**

3.5.1. *Kruskal's algorithm.*

```
#include "../data-structures/union_find.cpp" --------------//5e
vector<pair<int, ii> > mst(int n, -------------------------//42
--- vector<pair<int, ii> > edges) { -----------------------//64
- union_find uf(n); ---------------------------------------//96
- sort(edges.begin(), edges.end()); -----------------------//c3
- vector<pair<int, ii> > res; -----------------------------//8c
- rep(i,0,size(edges)) ------------------------------------//b0
--- if (uf.find(edges[i].second.first) != -----------------//2d
------- uf.find(edges[i].second.second)) { ----------------//e8
----- res.push_back(edges[i]); ----------------------------//1d
----- uf.unite(edges[i].second.first, ---------------------//33
------------- edges[i].second.second); } ------------------//65
- return res; } -------------------------------------------//d0
```

3.6. **Topological Sort.**

3.6.1. *Modified Depth-First Search.*

```
void tsort_dfs(int cur, char* color, const vvi& adj, ------//d5
--- stack<int>& res, bool& cyc) { -------------------------//b8
- color[cur] = 1; -----------------------------------------//b7
- rep(i,0,size(adj[cur])) { -------------------------------//70
--- int nxt = adj[cur][i]; --------------------------------//c7
--- if (color[nxt] == 0) ----------------------------------//97
----- tsort_dfs(nxt, color, adj, res, cyc); ---------------//5c
--- else if (color[nxt] == 1) -----------------------------//75
----- cyc = true; -----------------------------------------//ae
--- if (cyc) return; } ------------------------------------//5c
- color[cur] = 2; -----------------------------------------//91
- res.push(cur); } ----------------------------------------//a0
vi tsort(int n, vvi adj, bool& cyc) { ---------------------//9a
- cyc = false; --------------------------------------------//a1
- stack<int> S; -------------------------------------------//64
- vi res; -------------------------------------------------//a1
- char* color = new char[n]; ------------------------------//5d
- memset(color, 0, n); ------------------------------------//5c
- rep(i,0,n) { --------------------------------------------//a6
--- if (!color[i]) { --------------------------------------//1a
----- tsort_dfs(i, color, adj, S, cyc); -------------------//c1
----- if (cyc) return res; } } ----------------------------//6b
- while (!S.empty()) res.push_back(S.top()), S.pop(); -----//bf
- return res; } -------------------------------------------//60
```

3.7. **Euler Path.** Finds an euler path (or circuit) in a directed graph, or reports that none exist.

```
#define MAXV 1000 -----------------------------------------//2f
#define MAXE 5000 -----------------------------------------//87
vi adj[MAXV]; ---------------------------------------------//ff
int n, m, indeg[MAXV], outdeg[MAXV], res[MAXE + 1]; -------//49
ii start_end() { ------------------------------------------//30
- int start = -1, end = -1, any = 0, c = 0; ---------------//74
- rep(i,0,n) { --------------------------------------------//20
--- if (outdeg[i] > 0) any = i; ---------------------------//63
--- if (indeg[i] + 1 == outdeg[i]) start = i, c++; --------//5a
--- else if (indeg[i] == outdeg[i] + 1) end = i, c++; -----//13
--- else if (indeg[i] != outdeg[i]) return ii(-1,-1); } ---//ba
- if ((start == -1) != (end == -1) || (c != 2 && c != 0)) -//89
--- return ii(-1,-1); -------------------------------------//9c
- if (start == -1) start = end = any; ---------------------//4c
- return ii(start, end); } --------------------------------//bb
bool euler_path() { ---------------------------------------//4d
- ii se = start_end(); ------------------------------------//11
- int cur = se.first, at = m + 1; -------------------------//ca
- if (cur == -1) return false; ----------------------------//eb
- stack<int> s; -------------------------------------------//6c
- while (true) { ------------------------------------------//73
--- if (outdeg[cur] == 0) { -------------------------------//3f
----- res[--at] = cur; ------------------------------------//5e
----- if (s.empty()) break; -------------------------------//c5
----- cur = s.top(); s.pop(); -----------------------------//17
--- } else s.push(cur), cur = adj[cur][--outdeg[cur]]; } --//77
- return at == 0; } ---------------------------------------//32
```

And an undirected version, which finds a cycle.

```
multiset<int> adj[1010]; ----------------------------------//8c
list<int> L; ----------------------------------------------//9f
list<int>::iterator euler(int at, int to, -----------------//80
--- list<int>::iterator it) { -----------------------------//b4
- if (at == to) return it; --------------------------------//b8
- L.insert(it, at), --it; ---------------------------------//ef
- while (!adj[at].empty()) { ------------------------------//d0
--- int nxt = *adj[at].begin(); ---------------------------//a9
--- adj[at].erase(adj[at].find(nxt)); ---------------------//56
--- adj[nxt].erase(adj[nxt].find(at)); --------------------//b7
--- if (to == -1) { ---------------------------------------//7b
----- it = euler(nxt, at, it); ----------------------------//be
----- L.insert(it, at); -----------------------------------//82
----- --it; -----------------------------------------------//36
--- } else { ----------------------------------------------//c9
----- it = euler(nxt, to, it); ----------------------------//d7
----- to = -1; } } ----------------------------------------//15
- return it; } --------------------------------------------//73
// euler(0,-1,L.begin()) ----------------------------------//fd
```

3.8. **Bipartite Matching.**

3.8.1. *Alternating Paths algorithm.* The alternating paths algorithm solves bipartite matching in $O(mn^2)$ time, where m, n are the number of vertices on the left and right side of the bipartite graph, respectively.

```
vi* adj; --------------------------------------------------//cc
bool* done; -----------------------------------------------//b1
int* owner; -----------------------------------------------//26
int alternating_path(int left) { -------------------------//da
- if (done[left]) return 0; -------------------------------//08
- done[left] = true; --------------------------------------//f2
- rep(i,0,size(adj[left])) { ------------------------------//1b
--- int right = adj[left][i]; -----------------------------//46
--- if (owner[right] == -1 || -----------------------------//b6
------- alternating_path(owner[right])) { -----------------//82
----- owner[right] = left; return 1; } } ------------------//9b
- return 0; } ---------------------------------------------//7c
```

3.8.2. *Hopcroft-Karp algorithm.* An implementation of Hopcroft-Karp algorithm for bipartite matching. Running time is $O(|E|\sqrt{|V|})$.

```
#define MAXN 5000 -----------------------------------------//f7
int dist[MAXN+1], q[MAXN+1]; ------------------------------//b8
#define dist(v) dist[v == -1 ? MAXN : v] ------------------//0f
struct bipartite_graph { ----------------------------------//2b
- int N, M, *L, *R; vi *adj; ------------------------------//fc
- bipartite_graph(int _N, int _M) : N(_N), M(_M), ---------//8d
--- L(new int[N]), R(new int[M]), adj(new vi[N]) {} -------//cd
- ~bipartite_graph() { delete[] adj; delete[] L; delete[] R; }
- bool bfs() { --------------------------------------------//f5
--- int l = 0, r = 0; -------------------------------------//37
--- rep(v,0,N) if(L[v] == -1) dist(v) = 0, q[r++] = v; ----//f9
----- else dist(v) = INF; ---------------------------------//aa
--- dist(-1) = INF; ---------------------------------------//f2
--- while(l < r) { ----------------------------------------//ba
----- int v = q[l++]; -------------------------------------//50
----- if(dist(v) < dist(-1)) { ----------------------------//f1
------- iter(u, adj[v]) if(dist(R[*u]) == INF) ------------//9b
```

```
--------- dist(R[*u]) = dist(v) + 1, q[r++] = R[*u]; } } --//78
--- return dist(-1) != INF; } ------------------------------//e3
- bool dfs(int v) { ----------------------------------------//7d
--- if(v != -1) { ------------------------------------------//3e
----- iter(u, adj[v]) --------------------------------------//af
------- if(dist(R[*u]) == dist(v) + 1) ---------------------//21
--------- if(dfs(R[*u])) { ---------------------------------//cd
----------- R[*u] = v, L[v] = *u; --------------------------//0f
----------- return true; } ---------------------------------//b7
----- dist(v) = INF; ---------------------------------------//dd
----- return false; } --------------------------------------//40
--- return true; } -----------------------------------------//4a
- void add_edge(int i, int j) { adj[i].push_back(j); } ----//69
- int maximum_matching() { ---------------------------------//9a
--- int matching = 0; --------------------------------------//f3
--- memset(L, -1, sizeof(int) * N); ------------------------//c3
--- memset(R, -1, sizeof(int) * M); ------------------------//bd
--- while(bfs()) rep(i,0,N) --------------------------------//db
----- matching += L[i] == -1 && dfs(i); --------------------//27
--- return matching; } }; ----------------------------------//e1
```

3.8.3. *Minimum Vertex Cover in Bipartite Graphs.*

```
#include "hopcroft_karp.cpp" -------------------------------//05
vector<bool> alt; ------------------------------------------//cc
void dfs(bipartite_graph &g, int at) { ---------------------//14
- alt[at] = true; ------------------------------------------//df
- iter(it,g.adj[at]) { -------------------------------------//9f
--- alt[*it + g.N] = true; ---------------------------------//68
--- if (g.R[*it] != -1 && !alt[g.R[*it]]) dfs(g, g.R[*it]); } }
vi mvc_bipartite(bipartite_graph &g) { ---------------------//b1
- vi res; g.maximum_matching(); ----------------------------//fd
- alt.assign(g.N + g.M,false); -----------------------------//14
- rep(i,0,g.N) if (g.L[i] == -1) dfs(g, i); ----------------//ff
- rep(i,0,g.N) if (!alt[i]) res.push_back(i); --------------//66
- rep(i,0,g.M) if (alt[g.N + i]) res.push_back(g.N + i); --//30
- return res; } --------------------------------------------//c4
```

3.9. **Maximum Flow.**

3.9.1. *Dinic's algorithm.* An implementation of Dinic's algorithm that runs in $O(|V|^2|E|)$. It computes the maximum flow of a flow network.

```
#define MAXV 2000 ------------------------------------------//ba
int q[MAXV], d[MAXV]; --------------------------------------//e6
struct flow_network { --------------------------------------//12
- struct edge { int v, nxt, cap; ---------------------------//63
--- edge(int _v, int _cap, int _nxt) -----------------------//d4
----- : v(_v), nxt(_nxt), cap(_cap) { } }; -----------------//e9
- int n, *head, *curh; vector<edge> e, e_store; ------------//e8
- flow_network(int _n) : n(_n) { ---------------------------//54
--- curh = new int[n]; -------------------------------------//8c
--- memset(head = new int[n], -1, n*sizeof(int)); } -------//c6
- void reset() { e = e_store; } ----------------------------//37
- void add_edge(int u, int v, int uv, int vu=0) { ---------//e4
--- e.push_back(edge(v,uv,head[u])); head[u]=(int)size(e)-1;
--- e.push_back(edge(u,vu,head[v])); head[v]=(int)size(e)-1;}
- int augment(int v, int t, int f) { -----------------------//98
--- if (v == t) return f; ----------------------------------//6d
--- for (int &i = curh[v], ret; i != -1; i = e[i].nxt) ----//1e
----- if (e[i].cap > 0 && d[e[i].v] + 1 == d[v]) ----------//96
------- if ((ret = augment(e[i].v, t, min(f, e[i].cap))) > 0)
--------- return (e[i].cap -= ret, e[i^1].cap += ret, ret);//3c
--- return 0; } --------------------------------------------//bd
- int max_flow(int s, int t, bool res=true) { --------------//0a
--- e_store = e; -------------------------------------------//8b
--- int l, r, f = 0, x; ------------------------------------//46
--- while (true) { -----------------------------------------//27
----- memset(d, -1, n*sizeof(int)); ------------------------//59
----- l = r = 0, d[q[r++] = t] = 0; ------------------------//3d
----- while (l < r) ----------------------------------------//6f
------- for (int v = q[l++], i = head[v]; i != -1; i=e[i].nxt)
--------- if (e[i^1].cap > 0 && d[e[i].v] == -1) ----------//d1
----------- d[q[r++] = e[i].v] = d[v]+1; -------------------//5c
----- if (d[s] == -1) break; -------------------------------//d9
----- memcpy(curh, head, n * sizeof(int)); -----------------//ab
----- while ((x = augment(s, t, INF)) != 0) f += x; } -----//82
--- if (res) reset(); --------------------------------------//13
--- return f; } }; -----------------------------------------//b3
```

3.9.2. *Edmonds Karp's algorithm.* An implementation of Edmonds Karp's algorithm that runs in $O(|V||E|^2)$. It computes the maximum flow of a flow network.

```
#define MAXV 2000 ------------------------------------------//ba
int q[MAXV], p[MAXV], d[MAXV]; -----------------------------//22
struct flow_network { --------------------------------------//cf
- struct edge { int v, nxt, cap; ---------------------------//95
--- edge(int _v, int _cap, int _nxt) -----------------------//52
----- : v(_v), nxt(_nxt), cap(_cap) { } }; -----------------//60
- int n, *head; vector<edge> e, e_store; -------------------//ea
- flow_network(int _n) : n(_n) { ---------------------------//ea
--- memset(head = new int[n], -1, n*sizeof(int)); } -------//07
- void reset() { e = e_store; } ----------------------------//4e
- void add_edge(int u, int v, int uv, int vu=0) { ---------//19
--- e.push_back(edge(v,uv,head[u])); head[u]=(int)size(e)-1;
--- e.push_back(edge(u,vu,head[v])); head[v]=(int)size(e)-1;}
- int max_flow(int s, int t, bool res=true) { --------------//bf
--- e_store = e; -------------------------------------------//c0
--- int l, r, v, f = 0; ------------------------------------//96
--- while (true) { -----------------------------------------//8f
----- memset(d, -1, n*sizeof(int)); ------------------------//5b
----- memset(p, -1, n*sizeof(int)); ------------------------//0a
----- l = r = 0, d[q[r++] = s] = 0; ------------------------//ec
----- while (l < r) ----------------------------------------//26
------- for (int u = q[l++], i = head[u]; i != -1; i=e[i].nxt)
--------- if (e[i].cap > 0 && ------------------------------//f4
------------- (d[v = e[i].v] == -1 || d[u] + 1 < d[v])) ---//fb
----------- d[v] = d[u] + 1, p[q[r++] = v] = i; ------------//e1
----- if (p[t] == -1) break; -------------------------------//6d
----- int at = p[t], x = INF; ------------------------------//13
----- while (at != -1) -------------------------------------//27
------- x = min(x, e[at].cap), at = p[e[at^1].v]; ---------//f3
----- at = p[t], f += x; -----------------------------------//03
----- while (at != -1) -------------------------------------//09
------- e[at].cap -= x, e[at^1].cap += x, at = p[e[at^1].v]; }
--- if (res) reset(); --------------------------------------//6c
--- return f; } }; -----------------------------------------//04
```

3.10. **Minimum Cost Maximum Flow.** An implementation of Edmonds Karp's algorithm, modified to find shortest path to augment each time (instead of just any path). It computes the maximum flow of a flow network, and when there are multiple maximum flows, finds the maximum flow with minimum cost. Running time is $O(|V|^2|E|\log|V|)$.

```
#define MAXV 2000 ------------------------------------------//ba
int d[MAXV], p[MAXV], pot[MAXV]; ---------------------------//80
struct cmp { bool operator()(int i, int j) const { --------//30
--- return d[i] == d[j] ? i < j : d[i] < d[j]; } }; -------//dc
struct flow_network { --------------------------------------//5b
- struct edge { int v, nxt, cap, cost; ---------------------//e5
--- edge(int _v, int _cap, int _cost, int _nxt) ------------//ac
----- : v(_v), nxt(_nxt), cap(_cap), cost(_cost) { } }; ---//70
- int n; vi head; vector<edge> e, e_store; -----------------//9e
- flow_network(int _n) : n(_n), head(n,-1) { } -------------//36
- void reset() { e = e_store; } ----------------------------//2e
- void add_edge(int u, int v, int cost, int uv, int vu=0) {//21
--- e.push_back(edge(v, uv, cost, head[u])); ---------------//85
--- head[u] = (int)size(e)-1; ------------------------------//d8
--- e.push_back(edge(u, vu, -cost, head[v])); --------------//e9
--- head[v] = (int)size(e)-1; } ----------------------------//5b
- ii min_cost_max_flow(int s, int t, bool res=true) { -----//31
--- e_store = e; -------------------------------------------//2b
--- memset(pot, 0, n*sizeof(int)); -------------------------//2f
--- rep(it,0,n-1) rep(i,0,size(e)) if (e[i].cap > 0) ------//59
----- pot[e[i].v] = ----------------------------------------//9b
------- min(pot[e[i].v], pot[e[i^1].v] + e[i].cost); ------//9a
--- int v, f = 0, c = 0; -----------------------------------//13
--- while (true) { -----------------------------------------//04
----- memset(d, -1, n*sizeof(int)); ------------------------//3c
----- memset(p, -1, n*sizeof(int)); ------------------------//49
----- set<int, cmp> q; -------------------------------------//af
----- d[s] = 0; q.insert(s); -------------------------------//6d
----- while (!q.empty()) { ---------------------------------//e1
------- int u = *q.begin(); --------------------------------//98
------- q.erase(q.begin()); --------------------------------//78
------- for (int i = head[u]; i != -1; i = e[i].nxt) { ----//88
--------- if (e[i].cap == 0) continue; ---------------------//ec
--------- int cd = d[u] + e[i].cost + pot[u] - pot[v = e[i].v];
--------- if (d[v] == -1 || cd < d[v]) { -------------------//89
----------- q.erase(v); ------------------------------------//2b
----------- d[v] = cd; p[v] = i; ---------------------------//ea
----------- q.insert(v); } } } -----------------------------//d2
----- if (p[t] == -1) break; -------------------------------//e5
----- int at = p[t], x = INF; ------------------------------//0b
----- while (at != -1) -------------------------------------//82
------- x = min(x, e[at].cap), at = p[e[at^1].v]; ---------//1c
----- at = p[t], f += x; -----------------------------------//84
----- while (at != -1) -------------------------------------//c9
------- e[at].cap -= x, e[at^1].cap += x, at = p[e[at^1].v];
----- c += x * (d[t] + pot[t] - pot[s]); -------------------//d7
----- rep(i,0,n) if (p[i] != -1) pot[i] += d[i]; } --------//99
```

```
--- if (res) reset(); ------------------------------------//9c
--- return ii(f, c); } }; --------------------------------//92
```

3.11. All Pairs Maximum Flow.

3.11.1. *Gomory-Hu Tree.* An implementation of the Gomory-Hu Tree. The spanning tree is constructed using Gusfield's algorithm in $O(|V|^2)$ plus $|V|-1$ times the time it takes to calculate the maximum flow. If Dinic's algorithm is used to calculate the max flow, the running time is $O(|V|^3|E|)$. NOTE: Not sure if it works correctly with disconnected graphs.

```
#include "dinic.cpp" -------------------------------------//58
bool same[MAXV]; -----------------------------------------//35
pair<vii, vvi> construct_gh_tree(flow_network &g) { -------//2f
- int n = g.n, v; ----------------------------------------//40
- vii par(n, ii(0, 0)); vvi cap(n, vi(n, -1)); -----------//03
- rep(s,1,n) { -------------------------------------------//03
--- int l = 0, r = 0; ------------------------------------//50
--- par[s].second = g.max_flow(s, par[s].first, false); ---//12
--- memset(d, 0, n * sizeof(int)); -----------------------//a1
--- memset(same, 0, n * sizeof(bool)); -------------------//61
--- d[q[r++] = s] = 1; -----------------------------------//d9
--- while (l < r) { --------------------------------------//4b
----- same[v = q[l++]] = true; ---------------------------//3b
----- for (int i = g.head[v]; i != -1; i = g.e[i].nxt) ----//55
------- if (g.e[i].cap > 0 && d[g.e[i].v] == 0) -----------//d4
--------- d[q[r++] = g.e[i].v] = 1; } --------------------//a7
--- rep(i,s+1,n) -----------------------------------------//3f
----- if (par[i].first == par[s].first && same[i]) --------//2f
------- par[i].first = s; --------------------------------//fb
--- g.reset(); } -----------------------------------------//43
- rep(i,0,n) { -------------------------------------------//d3
--- int mn = INF, cur = i; -------------------------------//10
--- while (true) { ---------------------------------------//42
----- cap[cur][i] = mn; ----------------------------------//48
----- if (cur == 0) break; -------------------------------//b7
----- mn = min(mn, par[cur].second), cur = par[cur].first; } }
- return make_pair(par, cap); } --------------------------//d9
int compute_max_flow(int s, int t, const pair<vii, vvi> &gh) {
- int cur = INF, at = s; ---------------------------------//af
- while (gh.second[at][t] == -1) -------------------------//59
--- cur = min(cur, gh.first[at].second), -----------------//b2
--- at = gh.first[at].first; -----------------------------//04
- return min(cur, gh.second[at][t]); } -------------------//aa
```

3.12. Heavy-Light Decomposition.

```
#include "../data-structures/segment_tree.cpp" ------------//16
const int ID = 0; ----------------------------------------//fa
int f(int a, int b) { return a + b; } --------------------//e6
struct HLD { ---------------------------------------------//e3
- int n, curhead, curloc; --------------------------------//1c
- vi sz, head, parent, loc; ------------------------------//b6
- vvi adj; segment_tree values; --------------------------//e3
- HLD(int _n) : n(_n), sz(n, 1), head(n), ----------------//1a
--------------- parent(n, -1), loc(n), adj(n) { ----------//d0
--- vector<ll> tmp(n, ID); values = segment_tree(tmp); } --//0d
- void add_edge(int u, int v) { --------------------------//c2
--- adj[u].push_back(v); adj[v].push_back(u); } ----------//7f
- void update_cost(int u, int v, int c) { ----------------//55
--- if (parent[v] == u) swap(u, v); assert(parent[u] == v);//53
--- values.update(loc[u], c); } --------------------------//3b
- int csz(int u) { ---------------------------------------//4f
--- rep(i,0,size(adj[u])) if (adj[u][i] != parent[u]) -----//42
----- sz[u] += csz(adj[parent[adj[u][i]] = u][i]); --------//f2
--- return sz[u]; } --------------------------------------//4d
- void part(int u) { -------------------------------------//33
--- head[u] = curhead; loc[u] = curloc++; ----------------//b5
--- int best = -1; ---------------------------------------//de
--- rep(i,0,size(adj[u])) --------------------------------//5b
----- if (adj[u][i] != parent[u] && ----------------------//dd
--------- (best == -1 || sz[adj[u][i]] > sz[best])) -------//50
------- best = adj[u][i]; --------------------------------//7d
--- if (best != -1) part(best); --------------------------//56
--- rep(i,0,size(adj[u])) --------------------------------//b6
----- if (adj[u][i] != parent[u] && adj[u][i] != best) ----//b4
------- part(curhead = adj[u][i]); } ---------------------//af
- void build(int r = 0) { --------------------------------//f6
--- curloc = 0, csz(curhead = r), part(r); } -------------//86
- int lca(int u, int v) { --------------------------------//7c
--- vi uat, vat; int res = -1; ---------------------------//2c
--- while (u != -1) uat.push_back(u), u = parent[head[u]]; //c0
--- while (v != -1) vat.push_back(v), v = parent[head[v]]; //48
--- u = (int)size(uat) - 1, v = (int)size(vat) - 1; -------//9e
--- while (u >= 0 && v >= 0 && head[uat[u]] == head[vat[v]])
----- res = (loc[uat[u]] < loc[vat[v]] ? uat[u] : vat[v]), //be
----- u--, v--; ------------------------------------------//3b
--- return res; } ----------------------------------------//7a
- int query_upto(int u, int v) { int res = ID; -----------//ab
--- while (head[u] != head[v]) ---------------------------//c6
----- res = f(res, values.query(loc[head[u]], loc[u]).x), -//67
----- u = parent[head[u]]; -------------------------------//db
--- return f(res, values.query(loc[v] + 1, loc[u]).x); } --//7e
- int query(int u, int v) { int l = lca(u, v); -----------//8a
--- return f(query_upto(u, l), query_upto(v, l)); } }; ----//65
```

3.13. Centroid Decomposition.

```
#define MAXV 100100 --------------------------------------//86
#define LGMAXV 20 ----------------------------------------//aa
int jmp[MAXV][LGMAXV], -----------------------------------//6d
- path[MAXV][LGMAXV], ------------------------------------//9d
- sz[MAXV], seph[MAXV], ----------------------------------//cf
- shortest[MAXV]; ----------------------------------------//6b
struct centroid_decomposition { --------------------------//99
- int n; vvi adj; ----------------------------------------//e9
- centroid_decomposition(int _n) : n(_n), adj(n) { } -----//46
- void add_edge(int a, int b) { --------------------------//84
--- adj[a].push_back(b); adj[b].push_back(a); } ----------//65
- int dfs(int u, int p) { --------------------------------//dd
--- sz[u] = 1; -------------------------------------------//bf
--- rep(i,0,size(adj[u])) --------------------------------//ef
----- if (adj[u][i] != p) sz[u] += dfs(adj[u][i], u); -----//9d
--- return sz[u]; } --------------------------------------//bb
- void makepaths(int sep, int u, int p, int len) { -------//fe
--- jmp[u][seph[sep]] = sep, path[u][seph[sep]] = len; ----//19
--- int bad = -1; ----------------------------------------//f6
--- rep(i,0,size(adj[u])) { ------------------------------//c5
----- if (adj[u][i] == p) bad = i; -----------------------//38
----- else makepaths(sep, adj[u][i], u, len + 1); ---------//93
--- } ----------------------------------------------------//b9
--- if (p == sep) ----------------------------------------//a0
----- swap(adj[u][bad], adj[u].back()), adj[u].pop_back(); }
- void separate(int h=0, int u=0) { ----------------------//6e
--- dfs(u,-1); int sep = u; ------------------------------//29
--- down: iter(nxt,adj[sep]) -----------------------------//c2
----- if (sz[*nxt] < sz[sep] && sz[*nxt] > sz[u]/2) { -----//09
------- sep = *nxt; goto down; } -------------------------//5d
--- seph[sep] = h, makepaths(sep, sep, -1, 0); -----------//5d
--- rep(i,0,size(adj[sep])) separate(h+1, adj[sep][i]); } -//7c
- void paint(int u) { ------------------------------------//f1
--- rep(h,0,seph[u]+1) -----------------------------------//da
----- shortest[jmp[u][h]] = min(shortest[jmp[u][h]], ------//77
---------------------------- path[u][h]); } --------------//b2
- int closest(int u) { -----------------------------------//ec
--- int mn = INF/2; --------------------------------------//1f
--- rep(h,0,seph[u]+1) -----------------------------------//80
----- mn = min(mn, path[u][h] + shortest[jmp[u][h]]); -----//5c
--- return mn; } }; --------------------------------------//82
```

3.14. Least Common Ancestors, Binary Jumping.

```
struct node { -------------------------------------------//36
- node *p, *jmp[20]; -------------------------------------//24
- int depth; ---------------------------------------------//10
- node(node *_p = NULL) : p(_p) { ------------------------//78
--- depth = p ? 1 + p->depth : 0; ------------------------//3b
--- memset(jmp, 0, sizeof(jmp)); -------------------------//64
--- jmp[0] = p; ------------------------------------------//64
--- for (int i = 1; (1<<i) <= depth; i++) ----------------//a8
----- jmp[i] = jmp[i-1]->jmp[i-1]; } }; ------------------//3b
node* st[100000]; ----------------------------------------//65
node* lca(node *a, node *b) { ----------------------------//29
- if (!a || !b) return NULL; -----------------------------//cd
- if (a->depth < b->depth) swap(a,b); --------------------//fe
- for (int j = 19; j >= 0; j--) --------------------------//b3
--- while (a->depth - (1<<j) >= b->depth) a = a->jmp[j]; --//c0
- if (a == b) return a; ----------------------------------//08
- for (int j = 19; j >= 0; j--) --------------------------//11
--- while (a->depth >= (1<<j) && a->jmp[j] != b->jmp[j]) --//f0
----- a = a->jmp[j], b = b->jmp[j]; ----------------------//d0
- return a->p; } -----------------------------------------//c5
```

3.15. Tarjan's Off-line Lowest Common Ancestors Algorithm.

```
#include "../data-structures/union_find.cpp" -------------//5e
struct tarjan_olca { -------------------------------------//87
- int *ancestor; -----------------------------------------//39
- vi *adj, answers; --------------------------------------//dd
- vii *queries; ------------------------------------------//66
- bool *colored; -----------------------------------------//97
- union_find uf; -----------------------------------------//70
- tarjan_olca(int n, vi *_adj) : adj(_adj), uf(n) { ------//78
--- colored = new bool[n]; -------------------------------//8d
```

```
--- ancestor = new int[n]; ------------------------------//f2
--- queries = new vii[n]; -------------------------------//3e
--- memset(colored, 0, n); } ----------------------------//78
- void query(int x, int y) { ----------------------------//29
--- queries[x].push_back(ii(y, size(answers))); ---------//5e
--- queries[y].push_back(ii(x, size(answers))); ---------//07
--- answers.push_back(-1); } ----------------------------//74
- void process(int u) { ---------------------------------//38
--- ancestor[u] = u; ------------------------------------//a8
--- rep(i,0,size(adj[u])) { -----------------------------//24
----- int v = adj[u][i]; --------------------------------//2d
----- process(v); ---------------------------------------//0f
----- uf.unite(u,v); ------------------------------------//14
----- ancestor[uf.find(u)] = u; } -----------------------//f7
--- colored[u] = true; ----------------------------------//cf
--- rep(i,0,size(queries[u])) { -------------------------//28
----- int v = queries[u][i].first; ----------------------//2d
----- if (colored[v]) { ---------------------------------//23
------- answers[queries[u][i].second] = ancestor[uf.find(v)];
----- } } } }; ------------------------------------------//0b
```

3.16. **Minimum Mean Weight Cycle.** Given a strongly connected directed graph, finds the cycle of minimum mean weight. If you have a graph that is not strongly connected, run this on each strongly connected component.

```
double min_mean_cycle(vector<vector<pair<int,double>>> adj){
- int n = size(adj); double mn = INFINITY; --------------//dc
- vector<vector<double> > arr(n+1, vector<double>(n, mn)); //ce
- arr[0][0] = 0; ----------------------------------------//59
- rep(k,1,n+1) rep(j,0,n) iter(it,adj[j]) ---------------//b3
--- arr[k][it->first] = min(arr[k][it->first], ----------//d2
-------------------------- it->second + arr[k-1][j]); ---//9a
- rep(k,0,n) { ------------------------------------------//d3
--- double mx = -INFINITY; ------------------------------//b4
--- rep(i,0,n) mx = max(mx, (arr[n][i]-arr[k][i])/(n-k)); -//bc
--- mn = min(mn, mx); } ---------------------------------//2b
- return mn; } ------------------------------------------//cf
```

3.17. **Minimum Arborescence.** Given a weighted directed graph, finds a subset of edges of minimum total weight so that there is a unique path from the root r to each vertex. Returns a vector of size n, where the ith element is the edge for the ith vertex. The answer for the root is undefined!

```
#include "../data-structures/union_find.cpp" -------------//5e
struct arborescence { -----------------------------------//fa
- int n; union_find uf; ---------------------------------//70
- vector<vector<pair<ii,int> > > adj; -------------------//b7
- arborescence(int _n) : n(_n), uf(n), adj(n) { } -------//45
- void add_edge(int a, int b, int c) { ------------------//68
--- adj[b].push_back(make_pair(ii(a,b),c)); } -----------//8b
- vii find_min(int r) { ---------------------------------//88
--- vi vis(n,-1), mn(n,INF); vii par(n); ----------------//74
--- rep(i,0,n) { ----------------------------------------//10
----- if (uf.find(i) != i) continue; --------------------//9c
----- int at = i; ---------------------------------------//67
----- while (at != r && vis[at] == -1) { ----------------//57
------- vis[at] = i; ------------------------------------//21
------- iter(it,adj[at]) if (it->second < mn[at] && -------//4a
----------- uf.find(it->first.first) != at) -------------//b9
--------- mn[at] = it->second, par[at] = it->first; -------//aa
------- if (par[at] == ii(0,0)) return vii(); -----------//a9
------- at = uf.find(par[at].first); } ------------------//8a
----- if (at == r || vis[at] != i) continue; ------------//4e
----- union_find tmp = uf; vi seq; ----------------------//ec
----- do { seq.push_back(at); at = uf.find(par[at].first); //0b
----- } while (at != seq.front()); ----------------------//bc
----- iter(it,seq) uf.unite(*it,seq[0]); ----------------//a5
----- int c = uf.find(seq[0]); --------------------------//21
----- vector<pair<ii,int> > nw; -------------------------//4a
----- iter(it,seq) iter(jt,adj[*it]) --------------------//2b
------- nw.push_back(make_pair(jt->first, ---------------//c0
------------- jt->second - mn[*it])); -------------------//ea
----- adj[c] = nw; --------------------------------------//c2
----- vii rest = find_min(r); ---------------------------//40
----- if (size(rest) == 0) return rest; -----------------//1d
----- ii use = rest[c]; ---------------------------------//cc
----- rest[at = tmp.find(use.second)] = use; ------------//63
----- iter(it,seq) if (*it != at) -----------------------//19
------- rest[*it] = par[*it]; ---------------------------//05
----- return rest; } ------------------------------------//d6
--- return par; } }; ------------------------------------//25
```

3.18. **Blossom algorithm.** Finds a maximum matching in an arbitrary graph in $O(|V|^4)$ time. Be vary of loop edges.

```
#define MAXV 300 -----------------------------------------//3c
bool marked[MAXV], emarked[MAXV][MAXV]; -----------------//3a
int S[MAXV]; --------------------------------------------//f4
vi find_augmenting_path(const vector<vi> &adj,const vi &m){//38
- int n = size(adj), s = 0; -----------------------------//cd
- vi par(n,-1), height(n), root(n,-1), q, a, b; ---------//ba
- memset(marked,0,sizeof(marked)); ----------------------//35
- memset(emarked,0,sizeof(emarked)); --------------------//31
- rep(i,0,n) if (m[i] >= 0) emarked[i][m[i]] = true; -----//c3
----------- else root[i] = i, S[s++] = i; ---------------//c6
- while (s) { -------------------------------------------//0b
--- int v = S[--s]; -------------------------------------//d8
--- iter(wt,adj[v]) { -----------------------------------//c2
----- int w = *wt; --------------------------------------//70
----- if (emarked[v][w]) continue; ----------------------//18
----- if (root[w] == -1) { ------------------------------//77
------- int x = S[s++] = m[w]; --------------------------//e5
------- par[w]=v, root[w]=root[v], height[w]=height[v]+1; -//fd
------- par[x]=w, root[x]=root[w], height[x]=height[w]+1; -//ae
----- } else if (height[w] % 2 == 0) { ------------------//55
------- if (root[v] != root[w]) { -----------------------//75
--------- while (v != -1) q.push_back(v), v = par[v]; -----//9f
--------- reverse(q.begin(), q.end()); ------------------//2f
--------- while (w != -1) q.push_back(w), w = par[w]; -----//8f
--------- return q; -------------------------------------//51
------- } else { ----------------------------------------//e5
--------- int c = v; ------------------------------------//e1
--------- while (c != -1) a.push_back(c), c = par[c]; -----//5c
--------- c = w; ----------------------------------------//5f
--------- while (c != -1) b.push_back(c), c = par[c]; -----//bf
--------- while (!a.empty()&&!b.empty()&&a.back()==b.back())
----------- c = a.back(), a.pop_back(), b.pop_back(); -----//df
--------- memset(marked,0,sizeof(marked)); --------------//74
--------- fill(par.begin(), par.end(), 0); --------------//39
--------- iter(it,a) par[*it] = 1; iter(it,b) par[*it] = 1;//19
--------- par[c] = s = 1; -------------------------------//42
--------- rep(i,0,n) root[par[i] = par[i] ? 0 : s++] = i; -//90
--------- vector<vi> adj2(s); ---------------------------//2c
--------- rep(i,0,n) iter(it,adj[i]) { ------------------//85
----------- if (par[*it] == 0) continue; ----------------//c1
----------- if (par[i] == 0) { --------------------------//7f
------------- if (!marked[par[*it]]) { ------------------//5a
--------------- adj2[par[i]].push_back(par[*it]); --------//e6
--------------- adj2[par[*it]].push_back(par[i]); --------//35
--------------- marked[par[*it]] = true; } --------------//67
----------- } else adj2[par[i]].push_back(par[*it]); } ---//bd
--------- vi m2(s, -1); ---------------------------------//c2
--------- if (m[c] != -1) m2[m2[par[m[c]]] = 0] = par[m[c]];
--------- rep(i,0,n) if(par[i]!=0&&m[i]!=-1&&par[m[i]]!=0) //61
----------- m2[par[i]] = par[m[i]]; ---------------------//3c
--------- vi p = find_augmenting_path(adj2, m2); ---------//09
--------- int t = 0; ------------------------------------//53
--------- while (t < size(p) && p[t]) t++; --------------//b8
--------- if (t == size(p)) { ---------------------------//d8
----------- rep(i,0,size(p)) p[i] = root[p[i]]; ----------//8d
----------- return p; } ---------------------------------//21
--------- if (!p[0] || (m[c] != -1 && p[t+1] != par[m[c]]))//ee
----------- reverse(p.begin(), p.end()), t=(int)size(p)-t-1;
--------- rep(i,0,t) q.push_back(root[p[i]]); -----------//10
--------- iter(it,adj[root[p[t-1]]]) { ------------------//95
----------- if (par[*it] != (s = 0)) continue; ----------//e9
----------- a.push_back(c), reverse(a.begin(), a.end()); --//42
----------- iter(jt,b) a.push_back(*jt); ----------------//52
----------- while (a[s] != *it) s++; --------------------//a6
----------- if((height[*it]&1)^(s<(int)size(a)-(int)size(b)))
------------- reverse(a.begin(),a.end()), s=(int)size(a)-s-1;
----------- while(a[s]!=c)q.push_back(a[s]),s=(s+1)%size(a);
----------- q.push_back(c); -----------------------------//79
----------- rep(i,t+1,size(p)) q.push_back(root[p[i]]); ---//1a
----------- return q; } } } -----------------------------//1a
----- emarked[v][w] = emarked[w][v] = true; } -----------//82
--- marked[v] = true; } return q; } ---------------------//95
vii max_matching(const vector<vi> &adj) { ---------------//40
- vi m(size(adj), -1), ap; vii res, es; -----------------//2d
- rep(i,0,size(adj)) iter(it,adj[i]) es.emplace_back(i,*it);
- random_shuffle(es.begin(), es.end()); -----------------//9e
- iter(it,es) if (m[it->first] == -1 && m[it->second] == -1)
--- m[it->first] = it->second, m[it->second] = it->first; -//1c
- do { ap = find_augmenting_path(adj, m); ---------------//64
----- rep(i,0,size(ap)) m[m[ap[i^1]] = ap[i]] = ap[i^1]; -//62
- } while (!ap.empty()); --------------------------------//27
- rep(i,0,size(m)) if (i < m[i]) res.emplace_back(i, m[i]);//8c
- return res; } -----------------------------------------//90
```

3.19. **Maximum Density Subgraph.** Given (weighted) undirected graph G. Binary search density. If g is current density, construct flow network: (S,u,m), $(u,T,m+2g-d_u)$, $(u,v,1)$, where m is a large constant (larger than sum of edge weights). Run floating-point max-flow. If minimum cut has empty S-component, then maximum density is smaller than g, otherwise it's larger. Distance between valid densities is at least $1/(n(n-1))$. Edge case when density is 0. This also works for weighted graphs by replacing d_u by the weighted degree, and doing more iterations (if weights are not integers).

3.20. **Maximum-Weight Closure.** Given a vertex-weighted directed graph G. Turn the graph into a flow network, adding weight ∞ to each edge. Add vertices S, T. For each vertex v of weight w, add edge (S,v,w) if $w \geq 0$, or edge $(v,T,-w)$ if $w < 0$. Sum of positive weights minus minimum $S-T$ cut is the answer. Vertices reachable from S are in the closure. The maximum-weight closure is the same as the complement of the minimum-weight closure on the graph with edges reversed.

3.21. **Maximum Weighted Independent Set in a Bipartite Graph.** This is the same as the minimum weighted vertex cover. Solve this by constructing a flow network with edges $(S,u,w(u))$ for $u \in L$, $(v,T,w(v))$ for $v \in R$ and (u,v,∞) for $(u,v) \in E$. The minimum S,T-cut is the answer. Vertices adjacent to a cut edge are in the vertex cover.

3.22. **Synchronizing word problem.** A DFA has a synchronizing word (an input sequence that moves all states to the same state) iff. each pair of states has a synchronizing word. That can be checked using reverse DFS over pairs of states. Finding the shortest synchronizing word is NP-complete.

3.23. **Max flow with lower bounds on edges.** Change edge $(u,v,l \leq f \leq c)$ to $(u,v,f \leq c-l)$. Add edge (t,s,∞). Create super-nodes S, T. Let $M(u) = \sum_v l(v,u) - \sum_v l(u,v)$. If $M(u) < 0$, add edge $(u,T,-M(u))$, else add edge $(S,u,M(u))$. Max flow from S to T. If all edges from S are saturated, then we have a feasible flow. Continue running max flow from s to t in original graph.

3.24. **Tutte matrix for general matching.** Create an $n \times n$ matrix A. For each edge (i,j), $i<j$, let $A_{ij} = x_{ij}$ and $A_{ji} = -x_{ij}$. All other entries are 0. The determinant of A is zero iff. the graph has a perfect matching. A randomized algorithm uses the Schwartz–Zippel lemma to check if it is zero.

4. Strings

4.1. **The Knuth-Morris-Pratt algorithm.** An implementation of the Knuth-Morris-Pratt algorithm. Runs in $O(n+m)$ time, where n and m are the lengths of the string and the pattern.

```
vi kmppi(const string &p) { ------------------------------//a3
--- int m = p.size(), i = 0, j = -1; ---------------------//c6
--- vi b(m + 1, -1); -------------------------------------//73
--- while(i < m) { ---------------------------------------//cd
------- while(j >= 0 && p[i] != p[j]) j = b[j]; ----------//e6
------- b[++i] = ++j; } ----------------------------------//9d
--- return b; } ------------------------------------------//d4
vi kmp(const string &s, const string &p) { ---------------//43
--- int n = s.size(), m = p.size(), i = 0, j = 0; --------//a2
--- vi b = kmppi(p), a = vi(); ---------------------------//57
--- while(i < n) { ---------------------------------------//c7
------- while(j >= 0 && s[i] != p[j]) j = b[j]; ----------//c1
------- ++i; ++j; ----------------------------------------//02
------- if(j == m) { -------------------------------------//1d
----------- a.push_back(i - j); --------------------------//5c
----------- j = b[j]; ------------------------------------//3a
------- } ------------------------------------------------//b4
--- } ----------------------------------------------------//24
--- return a; } ------------------------------------------//3e
```

4.2. **The Z algorithm.** Given a string S, $Z_i(S)$ is the longest substring of S starting at i that is also a prefix of S. The Z algorithm computes these Z values in $O(n)$ time, where $n = |S|$. Z values can, for example, be used to find all occurrences of a pattern P in a string T in linear time. This is accomplished by computing Z values of $S = PT$, and looking for all i such that $Z_i \geq |P|$.

```
int* z_values(const string &s) { -------------------------//4d
- int n = size(s); ---------------------------------------//97
- int* z = new int[n]; -----------------------------------//c4
- int l = 0, r = 0; --------------------------------------//1c
- z[0] = n; ----------------------------------------------//98
- rep(i,1,n) { -------------------------------------------//b2
--- z[i] = 0; --------------------------------------------//4c
--- if (i > r) { -----------------------------------------//6d
----- l = r = i; -----------------------------------------//24
----- while (r < n && s[r - l] == s[r]) r++; -------------//68
----- z[i] = r - l; r--; ---------------------------------//07
--- } else if (z[i - l] < r - i + 1) z[i] = z[i - l]; ----//6f
--- else { -----------------------------------------------//a8
----- l = i; ---------------------------------------------//55
----- while (r < n && s[r - l] == s[r]) r++; -------------//2c
----- z[i] = r - l; r--; } } -----------------------------//13
- return z; } --------------------------------------------//d0
```

4.3. **Trie.** A Trie class.

```
template <class T> ---------------------------------------//82
struct trie { --------------------------------------------//4a
- struct node { ------------------------------------------//39
--- map<T, node*> children; ------------------------------//82
--- int prefixes, words; ---------------------------------//ff
--- node() { prefixes = words = 0; } }; ------------------//16
- node* root; --------------------------------------------//97
- trie() : root(new node()) {  } -------------------------//d2
- template <class I> -------------------------------------//2f
- void insert(I begin, I end) { --------------------------//3b
--- node* cur = root; ------------------------------------//ae
--- while (true) { ---------------------------------------//03
----- cur->prefixes++; -----------------------------------//6c
----- if (begin == end) { cur->words++; break; } ---------//df
----- else { ---------------------------------------------//51
------- T head = *begin; ---------------------------------//8f
------- typename map<T, node*>::const_iterator it; -------//ff
------- it = cur->children.find(head); -------------------//57
------- if (it == cur->children.end()) { -----------------//f7
--------- pair<T, node*> nw(head, new node()); -----------//66
--------- it = cur->children.insert(nw).first; -----------//c5
------- } begin++, cur = it->second; } } } ---------------//68
- template<class I> --------------------------------------//51
- int countMatches(I begin, I end) { ---------------------//84
--- node* cur = root; ------------------------------------//88
--- while (true) { ---------------------------------------//5b
----- if (begin == end) return cur->words; ---------------//61
----- else { ---------------------------------------------//c1
------- T head = *begin; ---------------------------------//75
------- typename map<T, node*>::const_iterator it; -------//00
------- it = cur->children.find(head); -------------------//c6
------- if (it == cur->children.end()) return 0; ---------//06
------- begin++, cur = it->second; } } } -----------------//85
- template<class I> --------------------------------------//e7
- int countPrefixes(I begin, I end) { --------------------//7d
--- node* cur = root; ------------------------------------//c6
--- while (true) { ---------------------------------------//ac
----- if (begin == end) return cur->prefixes; ------------//33
----- else { ---------------------------------------------//85
------- T head = *begin; ---------------------------------//0e
------- typename map<T, node*>::const_iterator it; -------//6e
------- it = cur->children.find(head); -------------------//40
------- if (it == cur->children.end()) return 0; ---------//18
------- begin++, cur = it->second; } } } }; --------------//7a
```

4.4. **Suffix Array.** An $O(n \log^2 n)$ construction of a Suffix Tree.

```
struct entry { ii nr; int p; }; --------------------------//f9
bool operator <(const entry &a, const entry &b) { --------//58
- return a.nr < b.nr; } ----------------------------------//61
struct suffix_array { ------------------------------------//e7
- string s; int n; vvi P; vector<entry> L; vi idx; -------//30
- suffix_array(string _s) : s(_s), n(size(s)) { ----------//ea
--- L = vector<entry>(n), P.push_back(vi(n)), idx = vi(n); //99
--- rep(i,0,n) P[0][i] = s[i]; ---------------------------//5c
--- for (int stp = 1, cnt = 1; cnt >> 1 < n; stp++, cnt <<= 1){
----- P.push_back(vi(n)); --------------------------------//76
----- rep(i,0,n) -----------------------------------------//f6
------- L[L[i].p = i].nr = ii(P[stp - 1][i], -------------//f0
----------- i + cnt < n ? P[stp - 1][i + cnt] : -1); -----//27
----- sort(L.begin(), L.end()); --------------------------//3e
----- rep(i,0,n) -----------------------------------------//ad
------- P[stp][L[i].p] = i > 0 && ------------------------//bd
--------- L[i].nr == L[i - 1].nr ? P[stp][L[i - 1].p] : i; }
--- rep(i,0,n) idx[P[(int)size(P) - 1][i]] = i; } --------//cf
- int lcp(int x, int y) { --------------------------------//ec
--- int res = 0; -----------------------------------------//0e
--- if (x == y) return n - x; ----------------------------//0c
--- for (int k = (int)size(P)-1; k >= 0 && x<n && y<n; k--)//7d
----- if (P[k][x] == P[k][y]) ----------------------------//14
------- x += 1 << k, y += 1 << k, res += 1 << k; ---------//c0
--- return res; } }; -------------------------------------//be
```

4.5. **Aho-Corasick Algorithm.** An implementation of the Aho-Corasick algorithm. Constructs a state machine from a set of keywords which can be used to search a string for any of the keywords.

```
struct aho_corasick { ------------------------------------//78
- struct out_node { --------------------------------------//3e
--- string keyword; out_node *next; ----------------------//f0
--- out_node(string k, out_node *n) ----------------------//20
----- : keyword(k), next(n) { } }; -----------------------//3f
- struct go_node { ---------------------------------------//7a
--- map<char, go_node*> next; ----------------------------//44
```

```
--- out_node *out; go_node *fail; -------------------------//9c
--- go_node() { out = NULL; fail = NULL; } }; -------------//39
- go_node *go; ---------------------------------------------//b8
- aho_corasick(vector<string> keywords) { -----------------//e5
--- go = new go_node(); -----------------------------------//59
--- iter(k, keywords) { -----------------------------------//18
----- go_node *cur = go; ----------------------------------//8f
----- iter(c, *k) -----------------------------------------//62
------- cur = cur->next.find(*c) != cur->next.end() ? -----//c4
--------- cur->next[*c] : (cur->next[*c] = new go_node()); //f9
----- cur->out = new out_node(*k, cur->out); } ------------//d6
--- queue<go_node*> q; ------------------------------------//9a
--- iter(a, go->next) q.push(a->second); ------------------//8f
--- while (!q.empty()) { ----------------------------------//d1
----- go_node *r = q.front(); q.pop(); --------------------//f0
----- iter(a, r->next) { ----------------------------------//a9
------- go_node *s = a->second; ---------------------------//ac
------- q.push(s); ----------------------------------------//35
------- go_node *st = r->fail; ----------------------------//44
------- while (st && st->next.find(a->first) == -----------//91
----------- st->next.end()) st = st->fail; ----------------//2b
------- if (!st) st = go; ---------------------------------//33
------- s->fail = st->next[a->first]; ---------------------//ad
------- if (s->fail) { ------------------------------------//36
--------- if (!s->out) s->out = s->fail->out; -------------//02
--------- else { ------------------------------------------//cc
----------- out_node* out = s->out; -----------------------//70
----------- while (out->next) out = out->next; ------------//7f
----------- out->next = s->fail->out; } } } } } -----------//dc
- vector<string> search(string s) { -----------------------//34
--- vector<string> res; -----------------------------------//43
--- go_node *cur = go; ------------------------------------//4c
--- iter(c, s) { ------------------------------------------//75
----- while (cur && cur->next.find(*c) == cur->next.end()) //95
------- cur = cur->fail; ----------------------------------//c0
----- if (!cur) cur = go; ---------------------------------//1f
----- cur = cur->next[*c]; --------------------------------//63
----- if (!cur) cur = go; ---------------------------------//d1
----- for (out_node *out = cur->out; out; out = out->next) //aa
------- res.push_back(out->keyword); } --------------------//ec
--- return res; } }; --------------------------------------//87
```

4.6. eerTree. Constructs an eerTree in $O(n)$, one character at a time.

```
#define MAXN 100100 -------------------------------------------//29
#define SIGMA 26 ----------------------------------------------//e2
#define BASE 'a' ----------------------------------------------//a1
char *s = new char[MAXN]; -------------------------------------//db
struct state { ------------------------------------------------//33
- int len, link, to[SIGMA]; -----------------------------------//24
} *st = new state[MAXN+2]; ------------------------------------//57
struct eertree { ----------------------------------------------//78
- int last, sz, n; --------------------------------------------//ba
- eertree() : last(1), sz(2), n(0) { --------------------------//83
--- st[0].len = st[0].link = -1; ------------------------------//3f
--- st[1].len = st[1].link = 0; } -----------------------------//34
- int extend() { ----------------------------------------------//c2
--- char c = s[n++]; int p = last; ----------------------------//25
--- while (n - st[p].len - 2 < 0 || c != s[n - st[p].len - 2])
----- p = st[p].link; -----------------------------------------//b0
--- if (!st[p].to[c-BASE]) { ----------------------------------//f4
----- int q = last = sz++; ------------------------------------//ff
----- st[p].to[c-BASE] = q; -----------------------------------//b9
----- st[q].len = st[p].len + 2; ------------------------------//c3
----- do { p = st[p].link; ------------------------------------//80
----- } while (p != -1 && (n < st[p].len + 2 || --------------//74
------------- c != s[n - st[p].len - 2])); -------------------//93
----- if (p == -1) st[q].link = 1; ----------------------------//e8
----- else st[q].link = st[p].to[c-BASE]; ---------------------//bf
----- return 1; } ---------------------------------------------//0a
--- last = st[p].to[c-BASE]; ----------------------------------//63
--- return 0; } }; --------------------------------------------//b6
```

4.7. Suffix Automaton. Minimum automata that accepts all suffixes of a string with $O(n)$ construction. The automata itself is a DAG therefore suitable for DP, examples are counting unique substrings, occurrences of substrings and suffix.

```
// TODO: Add longest common subsring ------------------------//0e
const int MAXL = 100000; ------------------------------------//31
struct suffix_automaton { -----------------------------------//e0
- vi  len, link, occur, cnt; --------------------------------//78
- vector<map<char,int> > next; ------------------------------//90
- vector<bool> isclone; -------------------------------------//7b
- ll *occuratleast; -----------------------------------------//f2
- int sz, last; ---------------------------------------------//7d
- string s; -------------------------------------------------//f2
- suffix_automaton() : len(MAXL*2), link(MAXL*2), ----------//36
--- occur(MAXL*2), next(MAXL*2), isclone(MAXL*2) { clear(); }
- void clear() { sz = 1; last = len[0] = 0; link[0] = -1; -//91
--------------- next[0].clear(); isclone[0] = false; } ---//21
- bool issubstr(string other){ ------------------------------//46
--- for(int i = 0, cur = 0; i < size(other); ++i){ --------//2e
----- if(cur == -1) return false; cur = next[cur][other[i]]; }
--- return true; } ------------------------------------------//3e
- void extend(char c){ int cur = sz++; len[cur] = len[last]+1;
--- next[cur].clear(); isclone[cur] = false; int p = last; //3d
--- for(; p != -1 && !next[p].count(c); p = link[p]) ------//10
----- next[p][c] = cur; -------------------------------------//41
--- if(p == -1){ link[cur] = 0; } ---------------------------//40
--- else{ int q = next[p][c]; -------------------------------//67
----- if(len[p] + 1 == len[q]){ link[cur] = q; } ----------//d2
----- else { int clone = sz++; isclone[clone] = true; -----//56
------- len[clone] = len[p] + 1; ----------------------------//71
------- link[clone] = link[q]; next[clone] = next[q]; -----//6d
------- for(; p != -1 && next[p].count(c) && next[p][c] == q;
------------ p = link[p]){ ----------------------------------//8c
--------- next[p][c] = clone; } -----------------------------//70
------- link[q] = link[cur] = clone; ------------------------//16
----- } } last = cur; } -------------------------------------//0f
- void count(){ ---------------------------------------------//ef
--- cnt=vi(sz, -1); stack<ii> S; S.push(ii(0,0)); ---------//8a
--- map<char,int>::iterator i; ------------------------------//81
--- while(!S.empty()){ --------------------------------------//20
----- ii cur = S.top(); S.pop(); ----------------------------//09
----- if(cur.second){ ---------------------------------------//bb
------- for(i = next[cur.first].begin(); --------------------//e2
----------- i != next[cur.first].end();++i){ ----------------//32
--------- cnt[cur.first] += cnt[(*i).second]; } } ---------//f1
----- else if(cnt[cur.first] == -1){ ------------------------//8f
------- cnt[cur.first] = 1; S.push(ii(cur.first, 1)); -----//9e
------- for(i = next[cur.first].begin(); --------------------//7e
----------- i != next[cur.first].end();++i){ ----------------//4c
--------- S.push(ii((*i).second, 0)); } } } } -------------//55
- string lexicok(ll k){ -------------------------------------//ef
--- int st = 0; string res; map<char,int>::iterator i; ----//7f
--- while(k){ -----------------------------------------------//68
----- for(i = next[st].begin(); i != next[st].end(); ++i){ //7e
------- if(k <= cnt[(*i).second]){ st = (*i).second; ------//ed
--------- res.push_back((*i).first); k--; break; ----------//61
------- } else { k -= cnt[(*i).second]; } } } -------------//7d
--- return res; } -------------------------------------------//32
- void countoccur(){ ----------------------------------------//a6
--- for(int i = 0; i < sz; ++i){ occur[i] = 1 - isclone[i]; }
--- vii states(sz); -----------------------------------------//23
--- for(int i = 0; i < sz; ++i){ states[i] = ii(len[i],i); }
--- sort(states.begin(), states.end()); ---------------------//25
--- for(int i = (int)size(states)-1; i >= 0; --i){ --------//d3
----- int v = states[i].second; -----------------------------//3d
----- if(link[v] != -1) { occur[link[v]] += occur[v]; }}}};//97
```

4.8. Hashing. Modulus should be a large prime. Can also use multiple instances with different moduli to minimize chance of collision.

```
struct hasher { int b = 311, m; vi h, p; -------------------//61
- hasher(string s, int _m) ----------------------------------//1a
--- : m(_m), h(size(s)+1), p(size(s)+1) { -----------------//9d
--- p[0] = 1; h[0] = 0; -------------------------------------//0d
--- rep(i,0,size(s)) p[i+1] = (ll)p[i] * b % m; -----------//17
--- rep(i,0,size(s)) h[i+1] = ((ll)h[i] * b + s[i]) % m; } //7c
- int hash(int l, int r) { ----------------------------------//f2
--- return (h[r+1] + m - (ll)h[l] * p[r-l+1] % m) % m; } };//6e
```

5. Mathematics

5.1. Fraction. A fraction (rational number) class. Note that numbers are stored in lowest common terms.

```
template <class T> struct fraction { -----------------------//27
- T gcd(T a, T b) { return b == T(0) ? a : gcd(b, a % b); }//fe
- T n, d; ---------------------------------------------------//6a
- fraction(T n_=T(0), T d_=T(1)) { -------------------------//be
--- assert(d_ != 0); ----------------------------------------//41
--- n = n_, d = d_; -----------------------------------------//d7
--- if (d < T(0)) n = -n, d = -d; ---------------------------//ac
--- T g = gcd(abs(n), abs(d)); ------------------------------//bb
--- n /= g, d /= g; } ---------------------------------------//55
- fraction(const fraction<T>& other) ------------------------//e3
--- : n(other.n), d(other.d) { } ---------------------------//fa
- fraction<T> operator +(const fraction<T>& other) const { //d9
--- return fraction<T>(n * other.d + other.n * d, ---------//bd
-------------------- d * other.d);} -------------------------//99
- fraction<T> operator -(const fraction<T>& other) const { //ae
```

```
--- return fraction<T>(n * other.d - other.n * d, ---------//4a
---------------------- d * other.d); } --------------------//8c
- fraction<T> operator *(const fraction<T>& other) const { //ea
--- return fraction<T>(n * other.n, d * other.d); } -------//65
- fraction<T> operator /(const fraction<T>& other) const { //52
--- return fraction<T>(n * other.d, d * other.n); } -------//af
- bool operator <(const fraction<T>& other) const { -------//f6
--- return n * other.d < other.n * d; } -------------------//d9
- bool operator <=(const fraction<T>& other) const { ------//77
--- return !(other < *this); } ----------------------------//bc
- bool operator >(const fraction<T>& other) const { -------//2c
--- return other < *this; } -------------------------------//04
- bool operator >=(const fraction<T>& other) const { ------//db
--- return !(*this < other); } ----------------------------//89
- bool operator ==(const fraction<T>& other) const { ------//c9
--- return n == other.n && d == other.d; } ----------------//02
- bool operator !=(const fraction<T>& other) const { ------//a4
--- return !(*this == other); } }; ------------------------//12
```

5.2. **Big Integer.** A big integer class.

```
struct intx { ---------------------------------------------//cf
- intx() { normalize(1); } --------------------------------//6c
- intx(string n) { init(n); } -----------------------------//b9
- intx(int n) { stringstream ss; ss << n; init(ss.str()); }//36
- intx(const intx& other) ---------------------------------//a6
--- : sign(other.sign), data(other.data) { } --------------//3d
- int sign; -----------------------------------------------//de
- vector<unsigned int> data; ------------------------------//e7
- static const int dcnt = 9; ------------------------------//1a
- static const unsigned int radix = 1000000000U; ----------//5d
- int size() const { return data.size(); } ----------------//54
- void init(string n) { -----------------------------------//b4
--- intx res; res.data.clear(); ---------------------------//29
--- if (n.empty()) n = "0"; -------------------------------//fc
--- if (n[0] == '-') res.sign = -1, n = n.substr(1); ------//8a
--- for (int i = n.size() - 1; i >= 0; i -= intx::dcnt) { -//b8
----- unsigned int digit = 0; -----------------------------//91
----- for (int j = intx::dcnt - 1; j >= 0; j--) { ---------//b1
------- int idx = i - j; ----------------------------------//08
------- if (idx < 0) continue; ----------------------------//03
------- digit = digit * 10 + (n[idx] - '0'); } ------------//c8
----- res.data.push_back(digit); } ------------------------//6a
--- data = res.data; --------------------------------------//70
--- normalize(res.sign); } --------------------------------//4e
- intx& normalize(int nsign) { ----------------------------//65
--- if (data.empty()) data.push_back(0); ------------------//97
--- for (int i = data.size() - 1; i > 0 && data[i] == 0; i--)
----- data.erase(data.begin() + i); -----------------------//26
--- sign = data.size() == 1 && data[0] == 0 ? 1 : nsign; --//dc
--- return *this; } ---------------------------------------//b5
- friend ostream& operator <<(ostream& outs, const intx& n) {
--- if (n.sign < 0) outs << '-'; --------------------------//3e
--- bool first = true; ------------------------------------//cb
--- for (int i = n.size() - 1; i >= 0; i--) { -------------//7a
----- if (first) outs << n.data[i], first = false; --------//29
----- else { ----------------------------------------------//b3
------- unsigned int cur = n.data[i]; ---------------------//f8
------- stringstream ss; ss << cur; -----------------------//85
------- string s = ss.str(); ------------------------------//47
------- int len = s.size(); -------------------------------//34
------- while (len < intx::dcnt) outs << '0', len++; ------//c6
------- outs << s; } } ------------------------------------//93
--- return outs; } ----------------------------------------//0f
- string to_string() const { ------------------------------//38
--- stringstream ss; ss << *this; return ss.str(); } ------//51
- bool operator <(const intx& b) const { ------------------//24
--- if (sign != b.sign) return sign < b.sign; -------------//20
--- if (size() != b.size()) -------------------------------//ca
----- return sign == 1 ? size() < b.size() : size() > b.size();
--- for (int i = size() - 1; i >= 0; i--) -----------------//73
----- if (data[i] != b.data[i]) ---------------------------//14
------- return sign == 1 ? data[i] < b.data[i] ------------//2a
---------------------- : data[i] > b.data[i]; ------------//0c
--- return false; } ---------------------------------------//ba
- intx operator -() const { -------------------------------//bc
--- intx res(*this); res.sign *= -1; return res; } --------//19
- friend intx abs(const intx &n) { return n < 0 ? -n : n; }//61
- intx operator +(const intx& b) const { ------------------//cc
--- if (sign > 0 && b.sign < 0) return *this - (-b); ------//46
--- if (sign < 0 && b.sign > 0) return b - (-*this); ------//d7
--- if (sign < 0 && b.sign < 0) return -((-*this) + (-b)); //ae
--- intx c; c.data.clear(); -------------------------------//51
--- unsigned long long carry = 0; -------------------------//35
--- for (int i = 0; i < size() || i < b.size() || carry; i++) {
----- carry += (i < size() ? data[i] : 0ULL) + ------------//f0
------- (i < b.size() ? b.data[i] : 0ULL); ----------------//b6
----- c.data.push_back(carry % intx::radix); --------------//39
----- carry /= intx::radix; } -----------------------------//51
--- return c.normalize(sign); } ---------------------------//95
- intx operator -(const intx& b) const { ------------------//35
--- if (sign > 0 && b.sign < 0) return *this + (-b); ------//b4
--- if (sign < 0 && b.sign > 0) return -(-*this + b); -----//59
--- if (sign < 0 && b.sign < 0) return (-b) - (-*this); ---//84
--- if (*this < b) return -(b - *this); -------------------//7f
--- intx c; c.data.clear(); -------------------------------//46
--- long long borrow = 0; ---------------------------------//05
--- rep(i,0,size()) { -------------------------------------//9f
----- borrow = data[i] - borrow ---------------------------//a4
--------------------- - (i < b.size() ? b.data[i] : 0ULL);//aa
----- c.data.push_back(borrow < 0 ? intx::radix + borrow --//13
------------------------------- : borrow); ---------------//d1
----- borrow = borrow < 0 ? 1 : 0; } ----------------------//1b
--- return c.normalize(sign); } ---------------------------//8a
- intx operator *(const intx& b) const { ------------------//c3
--- intx c; c.data.assign(size() + b.size() + 1, 0); ------//7d
--- rep(i,0,size()) { -------------------------------------//c0
----- long long carry = 0; --------------------------------//f6
----- for (int j = 0; j < b.size() || carry; j++) { -------//c8
------- if (j < b.size()) ---------------------------------//bc
--------- carry += (long long)data[i] * b.data[j]; --------//37
------- carry += c.data[i + j]; ---------------------------//5c
------- c.data[i + j] = carry % intx::radix; --------------//cd
------- carry /= intx::radix; } } -------------------------//ef
--- return c.normalize(sign * b.sign); } ------------------//ca
- friend pair<intx,intx> divmod(const intx& n, const intx& d) {
--- assert(!(d.size() == 1 && d.data[0] == 0)); -----------//67
--- intx q, r; q.data.assign(n.size(), 0); ----------------//e2
--- for (int i = n.size() - 1; i >= 0; i--) { -------------//76
----- r.data.insert(r.data.begin(), 0); -------------------//2a
----- r = r + n.data[i]; ----------------------------------//58
----- long long k = 0; ------------------------------------//6a
----- if (d.size() < r.size()) ----------------------------//01
------- k = (long long)intx::radix * r.data[d.size()]; ----//0d
----- if (d.size() - 1 < r.size()) k += r.data[d.size() - 1];
----- k /= d.data.back(); ---------------------------------//61
----- r = r - abs(d) * k; ---------------------------------//e4
----- // if (r < 0) for (ll t = 1LL << 62; t >= 1; t >>= 1) {
----- //     intx dd = abs(d) * t; ------------------------//3b
----- //     while (r + dd < 0) r = r + dd, k -= t; } -----//bb
----- while (r < 0) r = r + abs(d), k--; ------------------//b2
----- q.data[i] = k; } ------------------------------------//eb
--- return pair<intx, intx>(q.normalize(n.sign * d.sign), r); }
- intx operator /(const intx& d) const { ------------------//20
--- return divmod(*this,d).first; } -----------------------//c2
- intx operator %(const intx& d) const { ------------------//d9
--- return divmod(*this,d).second * sign; } }; ------------//28
```

5.2.1. *Fast Multiplication.* Fast multiplication for the big integer using Fast Fourier Transform.

```
#include "intx.cpp" ---------------------------------------//83
#include "fft.cpp" ----------------------------------------//13
intx fastmul(const intx &an, const intx &bn) { ------------//03
- string as = an.to_string(), bs = bn.to_string(); --------//fe
- int n = size(as), m = size(bs), l = 1, ------------------//a6
--- len = 5, radix = 100000, ------------------------------//b5
--- *a = new int[n], alen = 0, ----------------------------//4b
--- *b = new int[m], blen = 0; ----------------------------//c3
- memset(a, 0, n << 2); -----------------------------------//1d
- memset(b, 0, m << 2); -----------------------------------//d1
- for (int i = n - 1; i >= 0; i -= len, alen++) -----------//22
--- for (int j = min(len - 1, i); j >= 0; j--) ------------//3e
----- a[alen] = a[alen] * 10 + as[i - j] - '0'; -----------//31
- for (int i = m - 1; i >= 0; i -= len, blen++) -----------//f3
--- for (int j = min(len - 1, i); j >= 0; j--) ------------//a4
----- b[blen] = b[blen] * 10 + bs[i - j] - '0'; -----------//36
- while (l < 2*max(alen,blen)) l <<= 1; -------------------//8e
- cpx *A = new cpx[l], *B = new cpx[l]; -------------------//7d
- rep(i,0,l) A[i] = cpx(i < alen ? a[i] : 0, 0); ----------//01
- rep(i,0,l) B[i] = cpx(i < blen ? b[i] : 0, 0); ----------//d1
- fft(A, l); fft(B, l); -----------------------------------//77
- rep(i,0,l) A[i] *= B[i]; --------------------------------//78
- fft(A, l, true); ----------------------------------------//4b
- ull *data = new ull[l]; ---------------------------------//ab
- rep(i,0,l) data[i] = (ull)(round(real(A[i]))); ----------//f4
- rep(i,0,l-1) ---------------------------------------------//a0
--- if (data[i] >= (unsigned int)(radix)) { ---------------//8f
----- data[i+1] += data[i] / radix; -----------------------//b1
----- data[i] %= radix; } ---------------------------------//7d
```

```
- int stop = l-1; -----------------------------------------//f5
- while (stop > 0 && data[stop] == 0) stop--; -------------//36
- stringstream ss; ----------------------------------------//75
- ss << data[stop]; ---------------------------------------//e9
- for (int i = stop - 1; i >= 0; i--) ---------------------//99
--- ss << setfill('0') << setw(len) << data[i]; -----------//8d
- delete[] A; delete[] B; ---------------------------------//ad
- delete[] a; delete[] b; ---------------------------------//5b
- delete[] data; ------------------------------------------//1e
- return intx(ss.str()); } --------------------------------//cf
```

5.3. **Binomial Coefficients.** The binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the number of ways to choose k items out of a total of n items. Also contains an implementation of Lucas' theorem for computing the answer modulo a prime p. Use modular multiplicative inverse if needed, and be very careful of overflows.

```
int nck(int n, int k) { -----------------------------------//f6
- if (n < k) return 0; ------------------------------------//55
- k = min(k, n - k); --------------------------------------//bd
- int res = 1; --------------------------------------------//e6
- rep(i,1,k+1) res = res * (n - (k - i)) / i; -------------//4d
- return res; } -------------------------------------------//0e
int nck(int n, int k, int p) { ----------------------------//94
- int res = 1; --------------------------------------------//30
- while (n || k) { ----------------------------------------//84
--- res = nck(n % p, k % p) % p * res % p; ----------------//33
--- n /= p, k /= p; } -------------------------------------//bf
- return res; } -------------------------------------------//f4
```

5.4. **Euclidean algorithm.** The Euclidean algorithm computes the greatest common divisor of two integers a, b.

```
ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); } -//39
```

The extended Euclidean algorithm computes the greatest common divisor d of two integers a, b and also finds two integers x, y such that $a \times x + b \times y = d$.

```
ll egcd(ll a, ll b, ll& x, ll& y) { -----------------------//e0
- if (b == 0) { x = 1; y = 0; return a; } -----------------//8b
- ll d = egcd(b, a % b, x, y); ----------------------------//6a
- x -= a / b * y; swap(x, y); return d; } -----------------//95
```

5.5. **Trial Division Primality Testing.** An optimized trial division to check whether an integer is prime.

```
bool is_prime(int n) { ------------------------------------//6c
- if (n < 2) return false; --------------------------------//c9
- if (n < 4) return true; ---------------------------------//d9
- if (n % 2 == 0 || n % 3 == 0) return false; -------------//0f
- if (n < 25) return true; --------------------------------//ef
- for (int i = 5; i*i <= n; i += 6) -----------------------//38
--- if (n % i == 0 || n % (i + 2) == 0) return false; -----//69
- return true; } ------------------------------------------//b1
```

5.6. **Miller-Rabin Primality Test.** The Miller-Rabin probabilistic primality test.

```
#include "mod_pow.cpp" ------------------------------------//c7
bool is_probable_prime(ll n, int k) { ---------------------//be
- if (~n & 1) return n == 2; ------------------------------//d1
- if (n <= 3) return n == 3; ------------------------------//39
- int s = 0; ll d = n - 1; --------------------------------//37
- while (~d & 1) d >>= 1, s++; ----------------------------//35
- while (k--) { -------------------------------------------//c8
--- ll a = (n - 3) * rand() / RAND_MAX + 2; ---------------//06
--- ll x = mod_pow(a, d, n); ------------------------------//64
--- if (x == 1 || x == n - 1) continue; -------------------//9b
--- bool ok = false; --------------------------------------//03
--- rep(i,0,s-1) { ----------------------------------------//13
----- x = (x * x) % n; ------------------------------------//90
----- if (x == 1) return false; ---------------------------//5c
----- if (x == n - 1) { ok = true; break; } ---------------//a1
--- } -----------------------------------------------------//3a
--- if (!ok) return false; --------------------------------//37
- } return true; } ----------------------------------------//fe
```

5.7. **Pollard's ρ algorithm.**

```
ll rho(ll n) { --------------------------------------------//c1
--- vector<ll> seed = {2, 3, 4, 5, 7, 11, 13, 1031}; ------//e3
--- for(ll s : seed) { ------------------------------------//b6
------- ll x = s, y = x, d = 1; ---------------------------//cf
------- while(d == 1) { -----------------------------------//ec
---------- x = ((x * x + 1) % n + n) % n; -----------------//bf
---------- y = ((y * y + 1) % n + n) % n; -----------------//5b
---------- y = ((y * y + 1) % n + n) % n; -----------------//2a
---------- d = gcd(abs(x - y), n); } ----------------------//b4
------- if(d == n) continue; ------------------------------//d0
------- return d; } ---------------------------------------//98
--- return -1; } ------------------------------------------//54
```

5.8. **Sieve of Eratosthenes.** An optimized implementation of Eratosthenes' Sieve.

```
vi prime_sieve(int n) { -----------------------------------//40
- int mx = (n - 3) >> 1, sq, v, i = -1; -------------------//27
- vi primes; ----------------------------------------------//8f
- bool* prime = new bool[mx + 1]; -------------------------//ef
- memset(prime, 1, mx + 1); -------------------------------//28
- if (n >= 2) primes.push_back(2); ------------------------//f4
- while (++i <= mx) if (prime[i]) { -----------------------//73
--- primes.push_back(v = (i << 1) + 3); -------------------//be
--- if ((sq = i * ((i << 1) + 6) + 3) > mx) break; --------//2d
--- for (int j = sq; j <= mx; j += v) prime[j] = false; } -//2e
- while (++i <= mx) ---------------------------------------//52
--- if (prime[i]) primes.push_back((i << 1) + 3); ---------//ff
- delete[] prime; // can be used for O(1) lookup ----------//ae
- return primes; } ----------------------------------------//a8
```

5.9. **Divisor Sieve.** A O(n) prime sieve. Computes the smallest divisor of any number up to n.

```
vi divisor_sieve(int n) { ---------------------------------//7f
- vi mnd(n+1, 2), ps; -------------------------------------//ca
- if (n >= 2) ps.push_back(2); ----------------------------//79
- mnd[0] = 0; ---------------------------------------------//3d
- for (int k = 1; k <= n; k += 2) mnd[k] = k; -------------//b1
- for (int k = 3; k <= n; k += 2) { -----------------------//d9
--- if (mnd[k] == k) ps.push_back(k); ---------------------//7c
--- rep(i,1,size(ps)) -------------------------------------//3d
----- if (ps[i] > mnd[k] || ps[i]*k > n) break; -----------//6f
----- else mnd[ps[i]*k] = ps[i]; } ------------------------//06
- return ps; } --------------------------------------------//06
```

5.10. **Modular Exponentiation.** A function to perform fast modular exponentiation.

```
template <class T> ----------------------------------------//82
T mod_pow(T b, T e, T m) { --------------------------------//aa
- T res = T(1); -------------------------------------------//85
- while (e) { ---------------------------------------------//b7
--- if (e & T(1)) res = smod(res * b, m); -----------------//6d
--- b = smod(b * b, m), e >>= T(1); } ---------------------//12
- return res; } -------------------------------------------//86
```

5.11. **Modular Multiplicative Inverse.** A function to find a modular multiplicative inverse. Alternatively use `mod_pow(a,m-2,m)` when m is prime.

```
#include "egcd.cpp" ---------------------------------------//55
ll mod_inv(ll a, ll m) { ----------------------------------//0a
- ll x, y, d = egcd(a, m, x, y); --------------------------//db
- return d == 1 ? smod(x,m) : -1; } -----------------------//7a
```

A sieve version:

```
vi inv_sieve(int n, int p) { ------------------------------//40
- vi inv(n,1); --------------------------------------------//d7
- rep(i,2,n) inv[i] = (p - (ll)(p/i) * inv[p%i] % p) % p; -//fe
- return inv; } -------------------------------------------//14
```

5.12. **Primitive Root.**

```
#include "mod_pow.cpp" ------------------------------------//c7
ll primitive_root(ll m) { ---------------------------------//8a
- vector<ll> div; -----------------------------------------//f2
- for (ll i = 1; i*i <= m-1; i++) { -----------------------//ca
--- if ((m-1) % i == 0) { ---------------------------------//85
----- if (i < m) div.push_back(i); ------------------------//fd
----- if (m/i < m) div.push_back(m/i); } } ----------------//f2
- rep(x,2,m) { --------------------------------------------//57
--- bool ok = true; ---------------------------------------//17
--- iter(it,div) if (mod_pow<ll>(x, *it, m) == 1) { -------//48
----- ok = false; break; } --------------------------------//e5
--- if (ok) return x; } -----------------------------------//00
- return -1; } --------------------------------------------//a8
```

5.13. **Chinese Remainder Theorem.** An implementation of the Chinese Remainder Theorem.

```
#include "egcd.cpp" ---------------------------------------//55
ll crt(vector<ll> &as, vector<ll> &ns) { ------------------//72
- ll cnt = size(as), N = 1, x = 0, r, s, l; ---------------//ce
- rep(i,0,cnt) N *= ns[i]; --------------------------------//6a
- rep(i,0,cnt) egcd(ns[i], l = N/ns[i], r, s), x += as[i]*s*l;
- return smod(x, N); } ------------------------------------//80
pair<ll,ll> gcrt(vector<ll> &as, vector<ll> &ns) { --------//30
- map<ll,pair<ll,ll> > ms; --------------------------------//79
- rep(at,0,size(as)) { ------------------------------------//45
--- ll n = ns[at]; ----------------------------------------//48
--- for (ll i = 2; i*i <= n; i = i == 2 ? 3 : i + 2) { ----//d5
----- ll cur = 1; -----------------------------------------//88
----- while (n % i == 0) n /= i, cur *= i; ----------------//38
----- if (cur > 1 && cur > ms[i].first) -------------------//97
```

```
------- ms[i] = make_pair(cur, as[at] % cur); } -----------//af
--- if (n > 1 && n > ms[n].first) -------------------------//0d
----- ms[n] = make_pair(n, as[at] % n); } -----------------//6f
- vector<ll> as2, ns2; ll n = 1; --------------------------//cc
- iter(it,ms) { -------------------------------------------//6e
--- as2.push_back(it->second.second); ---------------------//f8
--- ns2.push_back(it->second.first); ----------------------//2b
--- n *= it->second.first; } ------------------------------//ba
- ll x = crt(as2,ns2); ------------------------------------//57
- rep(i,0,size(as)) if (smod(x,ns[i]) != smod(as[i],ns[i]))//d6
----- return ii(0,0); -------------------------------------//e6
- return make_pair(x,n); } --------------------------------//e1
```

5.14. **Linear Congruence Solver.** Given $ax \equiv b \pmod{n}$, returns (t, m) such that all solutions are given by $x \equiv t \pmod{m}$. No solutions iff $(0, 0)$ is returned.

```
#include "egcd.cpp" ---------------------------------------//55
pair<ll,ll> linear_congruence(ll a, ll b, ll n) { ---------//62
- ll x, y, d = egcd(smod(a,n), n, x, y); ------------------//17
- if ((b = smod(b,n)) % d != 0) return ii(0,0); -----------//5a
- return make_pair(smod(b / d * x, n),n/d); } -------------//3d
```

5.15. **Berlekamp-Massey algorithm.** Given a sequence of integers in some field, finds a linear recurrence of minimum order that generates the sequence in $O(n^2)$.

```
template<class K> bool eq(K a, K b) { return a == b; } ----//2a
template<> bool eq<long double>(long double a,long double b){
--- return abs(a - b) < EPS; } ----------------------------//0c
template <class Num> --------------------------------------//0d
vector<Num> berlekamp_massey(vector<Num> s) { -------------//da
- int m = 1, L = 0; bool sw; ------------------------------//da
- vector<Num> C = {1}, B = {1}, T, res; Num b = 1, a; -----//af
- rep(i,0,s.size()) { -------------------------------------//16
--- Num d = s[i]; -----------------------------------------//2a
--- rep(j,1,L+1) d = d + C[j] * s[i-j]; -------------------//c3
--- if (eq(d,Num(0))) { m++; continue; } ------------------//bf
--- if ((sw = 2*L <= i)) C.resize((L = i+1-L)+1), T = C; --//39
--- a = d / b; for (int j = m; j < C.size(); j++) ---------//2e
----- C[j] = C[j] - a * B[j-m]; ---------------------------//5f
--- m++; if (sw) B = T, b = d, m = 1; } -------------------//d6
- for (int i = 1; i <= L; i++) res.push_back(-C[i]); ------//bd
- return res; } -------------------------------------------//74
```

5.16. **Tonelli-Shanks algorithm.** Given prime p and integer $1 \leq n < p$, returns the square root r of n modulo p. There is also another solution given by $-r$ modulo p.

```
#include "mod_pow.cpp" ------------------------------------//c7
ll leg(ll a, ll p) { --------------------------------------//65
- if (a % p == 0) return 0; -------------------------------//ad
- if (p == 2) return 1; -----------------------------------//e3
- return mod_pow(a, (p-1)/2, p) == 1 ? 1 : -1; } ----------//1a
ll tonelli_shanks(ll n, ll p) { ---------------------------//34
- assert(leg(n,p) == 1); ----------------------------------//25
- if (p == 2) return 1; -----------------------------------//84
- ll s = 0, q = p-1, z = 2; -------------------------------//fb
- while (~q & 1) s++, q >>= 1; ----------------------------//8f
- if (s == 1) return mod_pow(n, (p+1)/4, p); --------------//c5
- while (leg(z,p) != -1) z++; -----------------------------//80
- ll c = mod_pow(z, q, p), --------------------------------//59
--- r = mod_pow(n, (q+1)/2, p), ---------------------------//0c
--- t = mod_pow(n, q, p), ---------------------------------//51
--- m = s; ------------------------------------------------//18
- while (t != 1) { ----------------------------------------//77
--- ll i = 1, ts = (ll)t*t % p; ---------------------------//05
--- while (ts != 1) i++, ts = ((ll)ts * ts) % p; ----------//f0
--- ll b = mod_pow(c, 1LL<<(m-i-1), p); -------------------//ac
--- r = (ll)r * b % p; ------------------------------------//be
--- t = (ll)t * b % p * b % p; ----------------------------//61
--- c = (ll)b * b % p; ------------------------------------//8f
--- m = i; } ----------------------------------------------//65
- return r; } ---------------------------------------------//59
```

5.17. **Numeric Integration.** Numeric integration using Simpson's rule.

```
double integrate(double (*f)(double), double a, double b, -//76
--- double delta = 1e-6) { --------------------------------//c0
- if (abs(a - b) < delta) ---------------------------------//38
--- return (b-a)/8 * --------------------------------------//56
----- (f(a) + 3*f((2*a+b)/3) + 3*f((a+2*b)/3) + f(b)); ----//e1
- return integrate(f, a, ----------------------------------//64
----- (a+b)/2, delta) + integrate(f, (a+b)/2, b, delta); } //a3
```

5.18. **Linear Recurrence Relation.** Computes the n-th term satisfying the linear recurrence relation with initial terms `init` and coefficients `c` in $O(k^2 \log n)$.

```
ll tmp[10000]; --------------------------------------------//b0
void mul(vector<ll> &a, vector<ll> &b, --------------------//6c
-------- const vector<ll> &c, ll mod) { -------------------//d1
- memset(tmp,0,sizeof(tmp)); ------------------------------//67
- rep(i,0,a.size()) rep(j,0,b.size()) ---------------------//93
--- tmp[i+j] = (tmp[i+j] + a[i] * b[j]) % mod; ------------//e8
- for (int i=(int)(a.size()+b.size())-2; i>=c.size(); i--) //bd
--- rep(j,0,c.size()) -------------------------------------//18
----- tmp[i-j-1] = (tmp[i-j-1] + tmp[i]*c[j]) % mod; ------//cc
- rep(i,0,a.size()) a[i] = i < c.size() ? tmp[i] : 0; } ---//a4
ll nth_term(const vector<ll> &init, const vector<ll> &c, --//e1
----------- ll n, ll mod) { -------------------------------//1d
- if (n < init.size()) return init[n]; --------------------//b3
- int l = max(2, (int)c.size()); --------------------------//95
- vector<ll> x(l), t(l); x[1]=t[0]=1; ---------------------//1c
- while (n) { if (n & 1) mul(t, x, c, mod); ---------------//e1
--- mul(x, x, c, mod); n >>= 1; } -------------------------//f9
- ll res = 0; ---------------------------------------------//5e
- rep(i,0,c.size()) res = (res + init[i] * t[i]) % mod; ---//b8
- return res; } -------------------------------------------//7c
```

5.19. **Fast Fourier Transform.** The Cooley-Tukey algorithm for quickly computing the discrete Fourier transform. The `fft` function only supports powers of twos. The `czt` function implements the Chirp Z-transform and supports any size, but is slightly slower.

```
#include <complex> ----------------------------------------//8e
typedef complex<long double> cpx; -------------------------//25
// NOTE: n must be a power of two -------------------------//14
void fft(cpx *x, int n, bool inv=false) { -----------------//36
- for (int i = 0, j = 0; i < n; i++) { --------------------//f9
--- if (i < j) swap(x[i], x[j]); --------------------------//44
--- int m = n>>1; -----------------------------------------//9c
--- while (1 <= m && m <= j) j -= m, m >>= 1; -------------//fe
--- j += m; } ---------------------------------------------//83
- for (int mx = 1; mx < n; mx <<= 1) { --------------------//16
--- cpx wp = exp(cpx(0, (inv ? -1 : 1) * pi / mx)), w = 1; //5c
--- for (int m = 0; m < mx; m++, w *= wp) { ---------------//82
----- for (int i = m; i < n; i += mx << 1) { --------------//23
------- cpx t = x[i + mx] * w; ----------------------------//44
------- x[i + mx] = x[i] - t; -----------------------------//da
------- x[i] += t; } } } ----------------------------------//57
- if (inv) rep(i,0,n) x[i] /= cpx(n); } -------------------//50
void czt(cpx *x, int n, bool inv=false) { -----------------//0d
- int len = 2*n+1; ----------------------------------------//c5
- while (len & (len - 1)) len &= len - 1; -----------------//1b
- len <<= 1; ----------------------------------------------//d4
- cpx w = exp(-2.0L * pi / n * cpx(0,1)), -----------------//d5
--- *c = new cpx[n], *a = new cpx[len], -------------------//09
--- *b = new cpx[len]; ------------------------------------//78
- rep(i,0,n) c[i] = pow(w, (inv ? -1.0 : 1.0)*i*i/2); -----//da
- rep(i,0,n) a[i] = x[i] * c[i], b[i] = 1.0L/c[i]; --------//67
- rep(i,0,n-1) b[len - n + i + 1] = 1.0L/c[n-i-1]; --------//4c
- fft(a, len); fft(b, len); -------------------------------//1d
- rep(i,0,len) a[i] *= b[i]; ------------------------------//a6
- fft(a, len, true); --------------------------------------//96
- rep(i,0,n) { --------------------------------------------//29
--- x[i] = c[i] * a[i]; -----------------------------------//43
--- if (inv) x[i] /= cpx(n); } ----------------------------//ed
- delete[] a; ---------------------------------------------//f7
- delete[] b; ---------------------------------------------//94
- delete[] c; } -------------------------------------------//2c
```

5.20. **Number-Theoretic Transform.** Other possible moduli: 2 113 929 217 (2^{25}), 2 013 265 920 268 435 457 (2^{28}, with $g = 5$).

```
#include "../mathematics/primitive_root.cpp" --------------//8c
int mod = 998244353, g = primitive_root(mod), -------------//9c
- ginv = mod_pow<ll>(g, mod-2, mod), ----------------------//7e
- inv2 = mod_pow<ll>(2, mod-2, mod); ----------------------//5b
#define MAXN (1<<22) --------------------------------------//29
struct Num { ----------------------------------------------//bf
- int x; --------------------------------------------------//5b
- Num(ll _x=0) { x = (_x%mod+mod)%mod; } ------------------//6f
- Num operator +(const Num &b) { return x + b.x; } --------//55
- Num operator -(const Num &b) const { return x - b.x; } --//c5
- Num operator *(const Num &b) const { return (ll)x * b.x; }
- Num operator /(const Num &b) const { --------------------//5e
--- return (ll)x * b.inv().x; } ---------------------------//f1
- Num inv() const { return mod_pow<ll>((ll)x, mod-2, mod); }
- Num pow(int p) const { return mod_pow<ll>((ll)x, p, mod); }
} T1[MAXN], T2[MAXN]; -------------------------------------//47
void ntt(Num x[], int n, bool inv = false) { --------------//d6
- Num z = inv ? ginv : g; ---------------------------------//22
- z = z.pow((mod - 1) / n); -------------------------------//6b
- for (ll i = 0, j = 0; i < n; i++) { ---------------------//8e
--- if (i < j) swap(x[i], x[j]); --------------------------//0c
--- ll k = n>>1; ------------------------------------------//e1
```

```
--- while (1 <= k && k <= j) j -= k, k >>= 1; -------------//dd
--- j += k; } --------------------------------------------//ee
- for (int mx = 1, p = n/2; mx < n; mx <<= 1, p >>= 1) { --//23
--- Num wp = z.pow(p), w = 1; ----------------------------//af
--- for (int k = 0; k < mx; k++, w = w*wp) { -------------//2b
----- for (int i = k; i < n; i += mx << 1) { -------------//32
------- Num t = x[i + mx] * w; ---------------------------//82
------- x[i + mx] = x[i] - t; ----------------------------//67
------- x[i] = x[i] + t; } } } ---------------------------//b9
- if (inv) { ---------------------------------------------//64
--- Num ni = Num(n).inv(); -------------------------------//91
--- rep(i,0,n) { x[i] = x[i] * ni; } } } -----------------//7f
void inv(Num x[], Num y[], int l) { ----------------------//1e
- if (l == 1) { y[0] = x[0].inv(); return; } -------------//5b
- inv(x, y, l>>1); ---------------------------------------//7e
- // NOTE: maybe l<<2 instead of l<<1 --------------------//e6
- rep(i,l>>1,l<<1) T1[i] = y[i] = 0; ---------------------//2b
- rep(i,0,l) T1[i] = x[i]; -------------------------------//60
- ntt(T1, l<<1); ntt(y, l<<1); ---------------------------//4c
- rep(i,0,l<<1) y[i] = y[i]*2 - T1[i] * y[i] * y[i]; -----//14
- ntt(y, l<<1, true); } ----------------------------------//18
void sqrt(Num x[], Num y[], int l) { ---------------------//9f
- if (l == 1) { assert(x[0].x == 1); y[0] = 1; return; } --//5d
- sqrt(x, y, l>>1); --------------------------------------//7b
- inv(y, T2, l>>1); --------------------------------------//50
- rep(i,l>>1,l<<1) T1[i] = T2[i] = 0; --------------------//56
- rep(i,0,l) T1[i] = x[i]; -------------------------------//e6
- ntt(T2, l<<1); ntt(T1, l<<1); --------------------------//25
- rep(i,0,l<<1) T2[i] = T1[i] * T2[i]; -------------------//6b
- ntt(T2, l<<1, true); -----------------------------------//9d
- rep(i,0,l) y[i] = (y[i] + T2[i]) * inv2; } -------------//9d
```

5.21. **Fast Hadamard Transform.** Computes the Hadamard transform of the given array. Can be used to compute the XOR-convolution of arrays, exactly like with FFT. For AND-convolution, use $(x+y, y)$ and $(x-y, y)$. For OR-convolution, use $(x, x+y)$ and $(x, -x+y)$. **Note**: Size of array must be a power of 2.

```
void fht(vi &arr, bool inv=false, int l=0, int r=-1) { ----//f7
- if (r == -1) { fht(arr,inv,0,size(arr)); return; } ------//e5
- if (l+1 == r) return; ----------------------------------//3c
- int k = (r-l)/2; ---------------------------------------//8f
- if (!inv) fht(arr, inv, l, l+k), fht(arr, inv, l+k, r); -//ef
- rep(i,l,l+k) { int x = arr[i], y = arr[i+k]; ------------//93
--- if (!inv) arr[i] = x-y, arr[i+k] = x+y; ---------------//81
--- else arr[i] = (x+y)/2, arr[i+k] = (-x+y)/2; } ---------//38
- if (inv) fht(arr, inv, l, l+k), fht(arr, inv, l+k, r); } //db
```

5.22. **Tridiagonal Matrix Algorithm.** Solves a tridiagonal system of linear equations $a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$ where $a_1 = c_n = 0$. Beware of numerical instability.

```
#define MAXN 5000 -----------------------------------------//f7
long double A[MAXN], B[MAXN], C[MAXN], D[MAXN], X[MAXN]; --//d8
void solve(int n) { --------------------------------------//01
- C[0] /= B[0]; D[0] /= B[0]; ----------------------------//94
- rep(i,1,n-1) C[i] /= B[i] - A[i]*C[i-1]; ---------------//6b
- rep(i,1,n) ---------------------------------------------//52
--- D[i] = (D[i] - A[i] * D[i-1]) / (B[i] - A[i] * C[i-1]);//d4
- X[n-1] = D[n-1]; ---------------------------------------//d7
- for (int i = n-2; i>=0; i--) ---------------------------//65
--- X[i] = D[i] - C[i] * X[i+1]; } -----------------------//6c
```

5.23. **Mertens Function.** Mertens function is $M(n) = \sum_{i=1}^{n} \mu(i)$. Let $L \approx (n \log\log n)^{2/3}$ and the algorithm runs in $O(n^{2/3})$.

```
#define L 9000000 -----------------------------------------//27
int mob[L], mer[L]; --------------------------------------//f1
unordered_map<ll,ll> mem; --------------------------------//30
ll M(ll n) { ---------------------------------------------//de
- if (n < L) return mer[n]; ------------------------------//1c
- if (mem.find(n) != mem.end()) return mem[n]; -----------//79
- ll ans = 0, done = 1; ----------------------------------//48
- for (ll i = 2; i*i <= n; i++) ans += M(n/i), done = i; --//41
- for (ll i = 1; i*i <= n; i++) --------------------------//35
--- ans += mer[i] * (n/i - max(done, n/(i+1))); -----------//94
- return mem[n] = 1 - ans; } -----------------------------//5c
void sieve() { -------------------------------------------//94
- for (int i = 1; i < L; i++) mer[i] = mob[i] = 1; --------//a8
- for (int i = 2; i < L; i++) { --------------------------//94
--- if (mer[i]) { ----------------------------------------//33
----- mob[i] = -1; ---------------------------------------//3c
----- for (int j = i+i; j < L; j += i) -------------------//58
------- mer[j] = 0, mob[j] = (j/i)%i == 0 ? 0 : -mob[j/i]; }
--- mer[i] = mob[i] + mer[i-1]; } } ----------------------//70
```

5.24. **Summatory Phi.** The summatory phi function $\Phi(n) = \sum_{i=1}^{n} \phi(i)$. Let $L \approx (n \log\log n)^{2/3}$ and the algorithm runs in $O(n^{2/3})$.

```
#define N 10000000 ----------------------------------------//e8
ll sp[N]; ------------------------------------------------//90
unordered_map<ll,ll> mem; --------------------------------//54
ll sumphi(ll n) { ----------------------------------------//3a
- if (n < N) return sp[n]; -------------------------------//de
- if (mem.find(n) != mem.end()) return mem[n]; -----------//4c
- ll ans = 0, done = 1; ----------------------------------//b2
- for (ll i = 2; i*i <= n; i++) ans += sumphi(n/i), done = i;
- for (ll i = 1; i*i <= n; i++) --------------------------//5a
--- ans += sp[i] * (n/i - max(done, n/(i+1))); ------------//b0
- return mem[n] = n*(n+1)/2 - ans; } ---------------------//fa
void sieve() { -------------------------------------------//55
- for (int i = 1; i < N; i++) sp[i] = i; -----------------//61
- for (int i = 2; i < N; i++) { --------------------------//f4
--- if (sp[i] == i) { ------------------------------------//e3
----- sp[i] = i-1; ---------------------------------------//d9
----- for (int j = i+i; j < N; j += i) sp[j] -= sp[j] / i; }
--- sp[i] += sp[i-1]; } } --------------------------------//f3
```

5.25. **Prime π.** Returns $\pi(\lfloor n/k \rfloor)$ for all $1 \le k \le n$, where $\pi(n)$ is the number of primes $\le n$. Can also be modified to accumulate any multiplicative function over the primes.

```
#include "prime_sieve.cpp" --------------------------------//3d
unordered_map<ll,ll> primepi(ll n) { ---------------------//73
#define f(n) (1) ------------------------------------------//34
#define F(n) (n) ------------------------------------------//99
- ll st = 1, *dp[3], k = 0; ------------------------------//a7
- while (st*st < n) st++; --------------------------------//bd
- vi ps = prime_sieve(st); -------------------------------//ae
- ps.push_back(st+1); ------------------------------------//21
- rep(i,0,3) dp[i] = new ll[2*st]; -----------------------//5a
- ll *pre = new ll[(int)size(ps)-1]; ---------------------//79
- rep(i,0,(int)size(ps)-1) -------------------------------//fd
--- pre[i] = f(ps[i]) + (i == 0 ? f(1) : pre[i-1]); -------//3e
#define L(i) ((i)<st?(i)+1:n/(2*st-(i))) -----------------//f6
#define I(l) ((l)<st?(l)-1:2*st-n/(l)) -------------------//8e
- rep(i,0,2*st) { ----------------------------------------//3a
--- ll cur = L(i); ---------------------------------------//97
--- while ((ll)ps[k]*ps[k] <= cur) k++; ------------------//21
--- dp[2][i] = k, dp[1][i] = F(L(i)), dp[0][i] = 0; } -----//a4
- for (int j = 0, start = 0; start < 2*st; j++) { ---------//2b
--- rep(i,start,2*st) { ----------------------------------//a3
----- if (j >= dp[2][i]) { start++; continue; } ----------//00
----- ll s = j == 0 ? f(1) : pre[j-1]; -------------------//19
----- int l = I(L(i)/ps[j]); -----------------------------//6d
----- dp[j&1][i] = dp[~j&1][i] ---------------------------//ed
------- - f(ps[j]) * (dp[~min(j,(int)dp[2][l])&1][l] - s); //c3
--- } } --------------------------------------------------//59
- unordered_map<ll,ll> res; ------------------------------//96
- rep(i,0,2*st) res[L(i)] = dp[~dp[2][i]&1][i]-f(1); ------//5a
- delete[] pre; rep(i,0,3) delete[] dp[i]; ---------------//c1
- return res; } ------------------------------------------//69
```

5.26. **Josephus problem.** Last man standing out of n if every kth is killed. Zero-based, and does not kill 0 on first pass.

```
int J(int n, int k) { ------------------------------------//27
- if (n == 1) return 0; ----------------------------------//e8
- if (k == 1) return n-1; --------------------------------//21
- if (n < k) return (J(n-1,k)+k)%n; ----------------------//31
- int np = n - n/k; --------------------------------------//b4
- return k*((J(np,k)+np-n%k%np)%np) / (k-1); } -----------//dd
```

5.27. **Number of Integer Points under Line.** Count the number of integer solutions to $Ax + By \le C$, $0 \le x \le n$, $0 \le y$. In other words, evaluate the sum $\sum_{x=0}^{n} \left\lfloor \frac{C-Ax}{B} + 1 \right\rfloor$. To count all solutions, let $n = \left\lfloor \frac{c}{a} \right\rfloor$. In any case, it must hold that $C - nA \ge 0$. Be very careful about overflows.

```
ll floor_sum(ll n, ll a, ll b, ll c) { -------------------//db
- if (c == 0) return 1; ----------------------------------//42
- if (c < 0) return 0; -----------------------------------//1c
- if (a % b == 0) return (n+1)*(c/b+1)-n*(n+1)/2*a/b; -----//88
- if (a >= b) return floor_sum(n,a%b,b,c)-a/b*n*(n+1)/2; --//bb
- ll t = (c-a*n+b)/b; ------------------------------------//c6
- return floor_sum((c-b*t)/b,b,a,c-b*t)+t*(n+1); } --------//9b
```

5.28. **Numbers and Sequences.** Some random prime numbers: 1031, 32771, 1048583, 33554467, 1073741827, 34359738421, 1099511627791, 35184372088891, 1125899906842679, 36028797018963971.

More random prime numbers: $10^3 + \{-9, -3, 9, 13\}$, $10^6 + \{-17, 3, 33\}$, $10^9 + \{7, 9, 21, 33, 87\}$.

Some maximal divisor counts:

840	32
720 720	240
735 134 400	1 344
963 761 198 400	6 720
866 421 317 361 600	26 880
897 612 484 786 617 600	103 680

5.29. Game Theory.

- Useful identity: $\oplus_{x=0}^{a-1} x = [0, a-1, 1, a][a\%4]$
- **Nim**: Winning position if $n_1 \oplus \cdots \oplus n_k = 0$
- **Misére Nim**: Winning position if some $n_i > 1$ and $n_1 \oplus \cdots \oplus n_k = 0$, or all $n_i \leq 1$ and $n_1 \oplus \cdots \oplus n_k = 1$

6. Geometry

6.1. Primitives.
Geometry primitives.

```
#define P(p) const point &p ------------------------------//2e
#define L(p0, p1) P(p0), P(p1) ---------------------------//cf
#define C(p0, r) P(p0), double r -------------------------//f1
#define PP(pp) pair<point,point> &pp ---------------------//e5
typedef complex<double> point; ---------------------------//6a
double dot(P(a), P(b)) { return real(conj(a) * b); } ------//d2
double cross(P(a), P(b)) { return imag(conj(a) * b); } ----//8a
point rotate(P(p), double radians = pi / 2, ---------------//98
------------ P(about) = point(0,0)) { ---------------------//19
- return (p - about) * exp(point(0, radians)) + about; } --//9b
point reflect(P(p), L(about1, about2)) { ------------------//f7
- point z = p - about1, w = about2 - about1; --------------//3f
- return conj(z / w) * w + about1; } ----------------------//b3
point proj(P(u), P(v)) { return dot(u, v) / dot(u, u) * u; }
point normalize(P(p), double k = 1.0) { -------------------//05
- return abs(p) == 0 ? point(0,0) : p / abs(p) * k; } -----//f7
double ccw(P(a), P(b)) { return cross(a, b); } ------------//4c
double ccw(P(a), P(b), P(c)) { return cross(b - a, c - b); }
bool collinear(P(a), P(b), P(c)) { ------------------------//52
- return abs(ccw(a, b, c)) < EPS; } -----------------------//7f
double angle(P(a), P(b), P(c)) { --------------------------//ee
- return acos(dot(b - a, c - b) / abs(b - a) / abs(c - b)); }
double signed_angle(P(a), P(b), P(c)) { -------------------//99
- return asin(cross(b - a, c - b) / abs(b - a) / abs(c - b)); }
double angle(P(p)) { return atan2(imag(p), real(p)); } ----//68
point perp(P(p)) { return point(-imag(p), real(p)); } -----//b3
double progress(P(p), L(a, b)) { --------------------------//2f
- if (abs(real(a) - real(b)) < EPS) -----------------------//cd
--- return (imag(p) - imag(a)) / (imag(b) - imag(a)); -----//09
- else return (real(p) - real(a)) / (real(b) - real(a)); } //6b
```

6.2. Lines.
Line related functions.

```
#include "primitives.cpp" ---------------------------------//e0
bool collinear(L(a, b), L(p, q)) { ------------------------//7c
- return abs(ccw(a, b, p)) < EPS && abs(ccw(a, b, q)) < EPS; }
bool parallel(L(a, b), L(p, q)) { -------------------------//58
- return abs(cross(b - a, q - p)) < EPS; } ----------------//9c
point closest_point(L(a, b), P(c), bool segment = false) { //c7
- if (segment) { ------------------------------------------//2d
--- if (dot(b - a, c - b) > 0) return b; ------------------//dd
--- if (dot(a - b, c - a) > 0) return a; ------------------//69
- } -------------------------------------------------------//a3
- double t = dot(c - a, b - a) / norm(b - a); -------------//c3
- return a + t * (b - a); } -------------------------------//f3
double line_segment_distance(L(a,b), L(c,d)) { ------------//17
- double x = INFINITY; ------------------------------------//cf
- if (abs(a - b) < EPS && abs(c - d) < EPS) x = abs(a - c);//eb
- else if (abs(a - b) < EPS) ------------------------------//cd
--- x = abs(a - closest_point(c, d, a, true)); ------------//81
- else if (abs(c - d) < EPS) ------------------------------//b9
--- x = abs(c - closest_point(a, b, c, true)); ------------//b0
- else if ((ccw(a, b, c) < 0) != (ccw(a, b, d) < 0) && ----//48
------ (ccw(c, d, a) < 0) != (ccw(c, d, b) < 0)) x = 0; ---//0f
- else { --------------------------------------------------//2c
--- x = min(x, abs(a - closest_point(c,d, a, true))); -----//0e
--- x = min(x, abs(b - closest_point(c,d, b, true))); -----//f1
--- x = min(x, abs(c - closest_point(a,b, c, true))); -----//72
--- x = min(x, abs(d - closest_point(a,b, d, true))); -----//ff
- } -------------------------------------------------------//8b
- return x; } ---------------------------------------------//b6
bool intersect(L(a,b), L(p,q), point &res, ----------------//00
--- bool lseg=false, bool rseg=false) { -------------------//e2
- // NOTE: check parallel/collinear before ----------------//7a
- point r = b - a, s = q - p; -----------------------------//5c
- double c = cross(r, s), ----------------------------------//de
------- t = cross(p - a, s) / c, u = cross(p - a, r) / c; //ee
- if (lseg && (t < 0-EPS || t > 1+EPS)) return false; ----//7a
- if (rseg && (u < 0-EPS || u > 1+EPS)) return false; ----//8a
- res = a + t * r; return true; } -------------------------//72
```

6.3. Circles.
Circle related functions.

```
#include "lines.cpp" --------------------------------------//d3
int intersect(C(A, rA), C(B, rB), point &r1, point &r2) { -//41
- double d = abs(B - A); ----------------------------------//5c
- if ((rA + rB) <  (d - EPS) || d < abs(rA - rB) - EPS) ---//4e
--- return 0; ---------------------------------------------//27
- double a = (rA*rA - rB*rB + d*d) / 2 / d, ---------------//1d
-------- h = sqrt(rA*rA - a*a); ---------------------------//e0
- point v = normalize(B - A, a), --------------------------//81
------- u = normalize(rotate(B-A), h); --------------------//83
- r1 = A + v + u, r2 = A + v - u; -------------------------//12
- return 1 + (abs(u) >= EPS); } ---------------------------//28
int intersect(L(A, B), C(O, r), point &r1, point &r2) { ---//cc
- point H = proj(B-A, O-A) + A; double h = abs(H-O); ------//b1
- if (r < h - EPS) return 0; ------------------------------//fe
- point v = normalize(B-A, sqrt(r*r - h*h)); --------------//77
- r1 = H + v, r2 = H - v; ---------------------------------//ce
- return 1 + (abs(v) > EPS); } ----------------------------//a4
int tangent(P(A), C(O, r), point &r1, point &r2) { --------//51
- point v = O - A; double d = abs(v); ---------------------//30
- if (d < r - EPS) return 0; ------------------------------//fc
- double alpha = asin(r / d), L = sqrt(d*d - r*r); --------//93
- v = normalize(v, L); ------------------------------------//01
- r1 = A + rotate(v, alpha), r2 = A + rotate(v, -alpha); --//10
- return 1 + (abs(v) > EPS); } ----------------------------//0c
void tangent_outer(C(A,rA), C(B,rB), PP(P), PP(Q)) { ------//d5
- // if (rA - rB > EPS) { swap(rA, rB); swap(A, B); } -----//e9
- double theta = asin((rB - rA)/abs(A - B)); --------------//1d
- point v = rotate(B - A, theta + pi/2), ------------------//28
------- u = rotate(B - A, -(theta + pi/2)); ---------------//11
- u = normalize(u, rA); -----------------------------------//66
- P.first = A + normalize(v, rA); -------------------------//e5
- P.second = B + normalize(v, rB); ------------------------//73
- Q.first = A + normalize(u, rA); -------------------------//aa
- Q.second = B + normalize(u, rB); } ----------------------//65
void tangent_inner(C(A,rA), C(B,rB), PP(P), PP(Q)) { ------//57
- point ip = (rA*B + rB*A)/(rA+rB); -----------------------//9d
- assert(tangent(ip, A, rA, P.first, Q.first) == 2); ------//0b
- assert(tangent(ip, B, rB, P.second, Q.second) == 2); } --//e7
pair<point,double> circumcircle(point a, point b, point c) {
- b -= a, c -= a; -----------------------------------------//e3
- point p = perp(b*norm(c)-c*norm(b))/2.0/cross(b, c); ----//4d
- return make_pair(a+p,abs(p)); } -------------------------//32
```

6.4. Polygon.
Polygon primitives.

```
#include "lines.cpp" --------------------------------------//d3
typedef vector<point> polygon; ---------------------------//1e
double polygon_area_signed(polygon p) { -------------------//85
- double area = 0; int cnt = size(p); ---------------------//36
- rep(i,1,cnt-1) area += cross(p[i] - p[0], p[i + 1] - p[0]);
- return area / 2; } --------------------------------------//f2
double polygon_area(polygon p) { --------------------------//70
- return abs(polygon_area_signed(p)); } -------------------//4e
#define CHK(f,a,b,c) \ -----------------------------------//ef
--- (f(a) < f(b) && f(b) <= f(c) && ccw(a,c,b) < 0) -------//a9
int point_in_polygon(polygon p, point q) { ----------------//4a
- int n = size(p); bool in = false; double d; ------------//b8
- for (int i = 0, j = n - 1; i < n; j = i++) -------------//cf
--- if (collinear(p[i], q, p[j]) && ----------------------//80
----- 0 <= (d = progress(q, p[i], p[j])) && d <= 1) ------//4c
----- return 0; ------------------------------------------//ae
- for (int i = 0, j = n - 1; i < n; j = i++) -------------//07
--- if (CHK(real, p[i], q, p[j]) || CHK(real, p[j], q, p[i]))
----- in = !in; ------------------------------------------//2b
- return in ? -1 : 1; } ----------------------------------//92
pair<polygon, polygon> cut_polygon(const polygon &poly, ---//68
-------------------------------- point a, point b) { ----//b7
- polygon left, right; point it; -------------------------//53
- for (int i = 0, cnt = poly.size(); i < cnt; i++) { ------//f4
--- point p = poly[i], q = poly[i == cnt-1 ? 0 : i + 1]; --//80
--- if (ccw(a, b, p) < EPS) left.push_back(p); -----------//01
--- if (ccw(a, b, p) > -EPS) right.push_back(p); ---------//1a
--- if (intersect(a, b, p, q, it, false, true)) ----------//ad
----- left.push_back(it), right.push_back(it); } ---------//bc
- return {left,right}; } ---------------------------------//3a
```

6.5. Convex Hull.
An algorithm that finds the Convex Hull of a set of points. NOTE: Doesn't work on some weird edge cases. (A small case that included three collinear lines would return the same point on both the upper and lower hull.)

```
#include "polygon.cpp" -----------------------------------//58
bool cmp(const point &a, const point &b) { ---------------//e2
- return abs(real(a) - real(b)) > EPS ? ------------------//23
--- real(a) < real(b) : imag(a) < imag(b); } -------------//25
polygon convexHull(polygon pts) { ------------------------//22
- sort(pts.begin(), pts.end(), cmp); ---------------------//13
- polygon hull; ------------------------------------------//7d
- hull.reserve(pts.size() + 1); --------------------------//5a
- for (int phase = 0; phase < 2; ++phase) { --------------//9d
--- auto start = hull.size(); ----------------------------//5a
--- for (auto& p : pts) { --------------------------------//68
```

```
----- while (hull.size() >= start+2 && --------------------//8d
----------- ccw(p, hull.back(), hull[hull.size()-2]) <= 0)//b3
------- hull.pop_back(); ----------------------------------//04
----- hull.push_back(p); ----------------------------------//ec
--- } -----------------------------------------------------//a3
--- hull.pop_back(); --------------------------------------//db
--- reverse(pts.begin(), pts.end()); ----------------------//b5
- } -------------------------------------------------------//df
- if (hull.size() == 2 && hull[0] == hull[1]) hull.pop_back();
- return hull; } ------------------------------------------//8a
```

6.6. **Line Segment Intersection.** Computes the intersection between two line segments.

```
#include "lines.cpp" --------------------------------------//d3
bool line_segment_intersect(L(a, b), L(c, d), point &A, ---//bf
------------------------------------------- point &B) { -//5f
- if (abs(a - b) < EPS && abs(c - d) < EPS) { ------------//4f
--- A = B = a; return abs(a - d) < EPS; } ----------------//cf
- else if (abs(a - b) < EPS) { ----------------------------//8d
--- A = B = a; double p = progress(a, c,d); ---------------//e0
--- return 0.0 <= p && p <= 1.0 ---------------------------//94
----- && (abs(a - c) + abs(d - a) - abs(d - c)) < EPS; } --//53
- else if (abs(c - d) < EPS) { ----------------------------//83
--- A = B = c; double p = progress(c, a,b); ---------------//8a
--- return 0.0 <= p && p <= 1.0 ---------------------------//35
----- && (abs(c - a) + abs(b - c) - abs(b - a)) < EPS; } --//28
- else if (collinear(a,b, c,d)) { -------------------------//e6
--- double ap = progress(a, c,d), bp = progress(b, c,d); --//b8
--- if (ap > bp) swap(ap, bp); ----------------------------//a5
--- if (bp < 0.0 || ap > 1.0) return false; ---------------//11
--- A = c + max(ap, 0.0) * (d - c); -----------------------//09
--- B = c + min(bp, 1.0) * (d - c); -----------------------//78
--- return true; } ----------------------------------------//65
- else if (parallel(a,b, c,d)) return false; --------------//c1
- else if (intersect(a,b, c,d, A, true,true)) { -----------//e8
--- B = A; return true; } ---------------------------------//b0
- return false; } -----------------------------------------//fa
```

6.7. **Great-Circle Distance.** Computes the distance between two points (given as latitude/longitude coordinates) on a sphere of radius r.

```
double gc_distance(double pLat, double pLong, -------------//7b
-------- double qLat, double qLong, double r) { -----------//a4
- pLat *= pi / 180; pLong *= pi / 180; --------------------//ee
- qLat *= pi / 180; qLong *= pi / 180; --------------------//75
- return r * acos(cos(pLat) * cos(qLat) * cos(pLong - qLong) +
-------- sin(pLat) * sin(qLat)); } ------------------------//e5
```

6.8. **Smallest Enclosing Circle.** Computes the smallest enclosing circle using Welzl's algorithm in expected $O(n)$ time.

```
#include "circles.cpp" ------------------------------------//37
vector<point> wP, wR; -------------------------------------//a1
pair<point,double> welzl() { ------------------------------//19
- if (wP.empty() || wR.size() == 3) { ---------------------//96
--- if (wR.empty()) return make_pair(point(), 0); ---------//db
--- if (wR.size() == 1) return make_pair(wR[0], 0); -------//57
--- if (wR.size() == 2) return make_pair((wR[0]+wR[1])/2.0,//7a
---------------------------------------- abs(wR[0]-wR[1])/2);
--- if (abs(cross(wR[1]-wR[0], wR[2]-wR[0])) < EPS) { -----//bc
----- point res; double mx = -INFINITY, d; ----------------//57
----- rep(i,0,3) rep(j,i+1,3) -----------------------------//cb
------- if ((d = abs(wR[i] - wR[j])) > mx) ----------------//2c
--------- mx = d, res = (wR[i] + wR[j]) / 2.0; ------------//99
----- return make_pair(res, mx/2.0); } --------------------//2d
--- return circumcircle(wR[0], wR[1], wR[2]); } -----------//ba
- swap(wP[rand() % wP.size()], wP.back()); ----------------//fd
- point res = wP.back(); wP.pop_back(); -------------------//6e
- pair<point,double> D = welzl(); -------------------------//a3
- if (abs(res - D.first) > D.second + EPS) { --------------//e9
--- wR.push_back(res); D = welzl(); wR.pop_back(); --------//3e
- } wP.push_back(res); return D; } ------------------------//d7
```

6.9. **Closest Pair of Points.** A sweep line algorithm for computing the distance between the closest pair of points.

```
#include "primitives.cpp" ---------------------------------//e0
-----------------------------------------------------------//85
struct cmpx { bool operator ()(const point &a, ------------//5e
----------------------------- const point &b) const { ----//2f
--- return abs(real(a) - real(b)) > EPS ? -----------------//1e
----- real(a) < real(b) : imag(a) < imag(b); } }; ---------//22
struct cmpy { bool operator ()(const point &a, ------------//36
----------------------------- const point &b) const { ----//1a
- return abs(imag(a) - imag(b)) > EPS ? -------------------//09
----- imag(a) < imag(b) : real(a) < real(b); } }; ---------//0c
double closest_pair(vector<point> pts) { ------------------//68
- sort(pts.begin(), pts.end(), cmpx()); -------------------//19
- set<point, cmpy> cur; -----------------------------------//77
- set<point, cmpy>::const_iterator it, jt; ----------------//b1
- double mn = INFINITY; -----------------------------------//a5
- for (int i = 0, l = 0; i < size(pts); i++) { ------------//a5
--- while (real(pts[i]) - real(pts[l]) > mn) --------------//02
----- cur.erase(pts[l++]); --------------------------------//52
--- it = cur.lower_bound(point(-INFINITY, imag(pts[i]) - mn));
--- jt = cur.upper_bound(point(INFINITY, imag(pts[i]) + mn));
--- while (it != jt) mn = min(mn, abs(*it - pts[i])), it++;//1c
--- cur.insert(pts[i]); } ---------------------------------//f6
- return mn; } --------------------------------------------//45
```

6.10. **3D Primitives.** Three-dimensional geometry primitives.

```
#define P(p) const point3d &p -----------------------------//a7
#define L(p0, p1) P(p0), P(p1) ----------------------------//0f
#define PL(p0, p1, p2) P(p0), P(p1), P(p2) ----------------//67
struct point3d { ------------------------------------------//63
- double x, y, z; -----------------------------------------//e6
- point3d() : x(0), y(0), z(0) {} -------------------------//af
- point3d(double _x, double _y, double _z) ----------------//ab
--- : x(_x), y(_y), z(_z) {} ------------------------------//8a
- point3d operator+(P(p)) const { -------------------------//30
--- return point3d(x + p.x, y + p.y, z + p.z); } ----------//25
- point3d operator-(P(p)) const { -------------------------//2c
--- return point3d(x - p.x, y - p.y, z - p.z); } ----------//04
- point3d operator-() const { -----------------------------//30
--- return point3d(-x, -y, -z); } -------------------------//48
- point3d operator*(double k) const { ---------------------//56
--- return point3d(x * k, y * k, z * k); } ----------------//99
- point3d operator/(double k) const { ---------------------//d2
--- return point3d(x / k, y / k, z / k); } ----------------//75
- double operator%(P(p)) const { --------------------------//69
--- return x * p.x + y * p.y + z * p.z; } -----------------//b2
- point3d operator*(P(p)) const { -------------------------//50
--- return point3d(y*p.z - z*p.y, -------------------------//2b
------------------ z*p.x - x*p.z, x*p.y - y*p.x); } -------//26
- double length() const { ---------------------------------//25
--- return sqrt(*this % *this); } -------------------------//7c
- double distTo(P(p)) const { -----------------------------//c1
--- return (*this - p).length(); } ------------------------//5e
- double distTo(P(A), P(B)) const { -----------------------//dc
--- // A and B must be two different points ---------------//63
--- return ((*this - A) * (*this - B)).length() / A.distTo(B);}
- double signedDistTo(PL(A,B,C)) const { ------------------//ca
--- // A, B and C must not be collinear ------------------//ce
--- point3d N = (B-A)*(C-A); double D = A%N; --------------//1d
--- return ((*this)%N - D)/N.length(); } ------------------//5a
- point3d normalize(double k = 1) const { -----------------//28
--- // length() must not return 0 ------------------------//ec
--- return (*this) * (k / length()); } --------------------//44
- point3d getProjection(P(A), P(B)) const { ---------------//20
--- point3d v = B - A; ------------------------------------//27
--- return A + v.normalize((v % (*this - A)) / v.length()); }
- point3d rotate(P(normal)) const { -----------------------//a2
--- //normal must have length 1 and be orthogonal to the vector
--- return (*this) * normal; } ----------------------------//eb
- point3d rotate(double alpha, P(normal)) const { ---------//b4
--- return (*this) * cos(alpha) + rotate(normal) * sin(alpha);}
- point3d rotatePoint(P(O), P(axe), double alpha) const{ --//66
--- point3d Z = axe.normalize(axe % (*this - O)); ---------//f9
--- return O + Z + (*this - O - Z).rotate(alpha, O); } ----//87
- bool isZero() const { -----------------------------------//b3
--- return abs(x) < EPS && abs(y) < EPS && abs(z) < EPS; } //af
- bool isOnLine(L(A, B)) const { --------------------------//b5
--- return ((A - *this) * (B - *this)).isZero(); } --------//7a
- bool isInSegment(L(A, B)) const { -----------------------//da
--- return isOnLine(A, B) && ((A - *this) % (B - *this))<EPS;}
- bool isInSegmentStrictly(L(A, B)) const { ---------------//20
--- return isOnLine(A, B) && ((A - *this) % (B - *this))<-EPS;}
- double getAngle() const { -------------------------------//49
--- return atan2(y, x); } ---------------------------------//39
- double getAngle(P(u)) const { ---------------------------//68
--- return atan2((*this * u).length(), *this % u); } ------//0d
- bool isOnPlane(PL(A, B, C)) const { ---------------------//6b
--- return ------------------------------------------------//9a
----- abs((A - *this) * (B - *this) % (C - *this)) < EPS; } };
int line_line_intersect(L(A, B), L(C, D), point3d &O){ ----//c7
- if (abs((B - A) * (C - A) % (D - A)) > EPS) return 0; ---//2d
- if (((A - B) * (C - D)).length() < EPS) -----------------//16
--- return A.isOnLine(C, D) ? 2 : 0; ----------------------//30
- point3d normal = ((A - B) * (C - B)).normalize(); -------//2d
- double s1 = (C - A) * (D - A) % normal; -----------------//da
- O = A + ((B - A) / (s1 + ((D - B) * (C - B) % normal))) * s1;
- return 1; } ---------------------------------------------//2f
```

```
int line_plane_intersect(L(A, B), PL(C, D, E), point3d & O) {
- double V1 = (C - A) * (D - A) % (E - A); ----------------//3b
- double V2 = (D - B) * (C - B) % (E - B); ----------------//6d
- if (abs(V1 + V2) < EPS) ---------------------------------//48
--- return A.isOnPlane(C, D, E) ? 2 : 0; ------------------//39
- O = A + ((B - A) / (V1 + V2)) * V1; ---------------------//4c
- return 1; } ---------------------------------------------//fd
bool plane_plane_intersect(P(A), P(nA), P(B), P(nB), ------//f3
--- point3d &P, point3d &Q) { -----------------------------//a9
- point3d n = nA * nB; ------------------------------------//71
- if (n.isZero()) return false; ---------------------------//27
- point3d v = n * nA; -------------------------------------//60
- P = A + (n * nA) * ((B - A) % nB / (v % nB)); -----------//b4
- Q = P + n; ----------------------------------------------//63
- return true; } ------------------------------------------//80
double line_line_distance(L(A, B), L(C, D), point3d &E, ---//c8
------------------------------------------- point3d &F) { -//2e
- point3d w = (C-A), v = (B-A), u = (D-C), ----------------//98
-------- N = v*u, N1 = v*(u*v), N2 = u*(v*u); ------------//68
- if (w.isZero() || (v*w).isZero()) E = F = A; -----------//24
- else if (N.isZero()) E = A, -----------------------------//50
--- F = A + w - v * ((w%v)/(v%v)); ------------------------//7e
- else E = A + v*((w % N2)/(v%N2)), -----------------------//17
--- F = C + u*(((-w) % N1)/(u%N1)); -----------------------//d4
- return (F-E).length(); } --------------------------------//f4
```

6.11. **3D Convex Hull.**

```
#include "primitives3d.cpp" -------------------------------//9d
double mixed(P(a), P(b), P(c)) { return a % (b * c); } ----//fa
bool cmpy(const point3d& a, const point3d& b) { -----------//9c
- if (abs(a.y-b.y) > EPS) return a.y < b.y; ---------------//29
- if (abs(a.x-b.x) > EPS) return a.x < b.x; ---------------//ae
- return a.z < b.z; } -------------------------------------//01
point3d slp; ----------------------------------------------//37
bool cmpsl(const point3d& a, const point3d& b) { ----------//22
- point3d ad = a-slp, bd = b-slp; -------------------------//1a
- return atan2(ad.y, sqrt(ad.x*ad.x + ad.z*ad.z)) < -------//44
-------- atan2(bd.y, sqrt(bd.x*bd.x + bd.z*bd.z)); } ------//ba
set<vi> gift_wrap(vector<point3d> points) { ---------------//ac
- int n = points.size(), lowi = 0, lowj = 0; --------------//43
- set<vi> res; set<ii> vis; queue<ii> q; ------------------//6d
- if (n < 3) return res; ----------------------------------//2f
- rep(i,1,n) if (cmpy(points[i], points[lowi])) lowi = i; -//bc
- slp = points[lowi]; -------------------------------------//71
- if (lowj == lowi) lowj++; -------------------------------//ad
- rep(j,lowj+1,n) ------------------------------------------//42
--- if (j!=lowi && cmpsl(points[j], points[lowj])) lowj=j; //54
- q.push(ii(min(lowi,lowj), max(lowi,lowj))); -------------//f7
- while (!q.empty()) { ------------------------------------//b4
--- ii cur = q.front(); q.pop(); --------------------------//67
--- if (!vis.insert(cur).second) continue; ----------------//bd
--- int mni = 0, mxi = 0; ---------------------------------//a4
--- while (mni==cur.first || mni==cur.second) mni++,mxi++; //2d
--- rep(i,0,n) { ------------------------------------------//a8
----- if (i == cur.first || i == cur.second) continue; ----//cb
----- if (mixed(points[cur.second] - points[cur.first], ---//f2
----------- points[mni] - points[cur.first], --------------//32
----------- points[i] - points[cur.first]) < 0) mni = i; --//c7
----- if (mixed(points[cur.second] - points[cur.first], ---//37
----------- points[mxi] - points[cur.first], --------------//86
----------- points[i] - points[cur.first]) > 0) mxi = i; } //4f
--- vi a = {cur.first,cur.second}, b = {mni,mxi}; ---------//23
--- rep(i,0,2) { ------------------------------------------//e0
----- if (b[i] == -1) continue; ---------------------------//38
----- rep(j,0,2) q.push({min(b[i],a[j]), max(b[i],a[j])}); //96
----- vi v = {a[0], a[1], b[i]}; --------------------------//32
----- sort(v.begin(), v.end()); ---------------------------//b0
----- res.insert(v); } } return res; } --------------------//45
```

6.12. **Polygon Centroid.**

```
#include "polygon.cpp" ------------------------------------//58
point polygon_centroid(polygon p) { -----------------------//79
- double cx = 0.0, cy = 0.0; ------------------------------//d5
- double mnx = 0.0, mny = 0.0; ----------------------------//22
- int n = size(p); ----------------------------------------//2d
- rep(i,0,n) ----------------------------------------------//08
--- mnx = min(mnx, real(p[i])), ---------------------------//c6
--- mny = min(mny, imag(p[i])); ---------------------------//84
- rep(i,0,n) ----------------------------------------------//3f
--- p[i] = point(real(p[i]) - mnx, imag(p[i]) - mny); -----//49
- rep(i,0,n) { --------------------------------------------//3c
--- int j = (i + 1) % n; ----------------------------------//5b
--- cx += (real(p[i]) + real(p[j])) * cross(p[i], p[j]); --//4f
--- cy += (imag(p[i]) + imag(p[j])) * cross(p[i], p[j]); } //4a
- return point(cx, cy) / 6.0 / polygon_area_signed(p) -----//dd
-------- + point(mnx, mny); } -----------------------------//b5
```

6.13. **Rotating Calipers.**

```
#include "convex_hull.cpp" --------------------------------//6b
double maxDist2(const polygon& poly) { --------------------//58
- int n = static_cast<int>(poly.size()); ------------------//69
- double res = 0; -----------------------------------------//d7
- for (int i = 0, j = n < 2 ? 0 : 1; i < j; ++i) ----------//37
--- for (;; j = ((j + 1) % n)) { --------------------------//1b
----- res = max(res, abs(poly[i] - poly[j])); -------------//25
----- if (ccw(poly[i+1] - poly[i], ------------------------//2b
------- poly[((j + 1) % n)] - poly[j]) >= 0) break; -------//30
--- } -----------------------------------------------------//04
- return res; } -------------------------------------------//be
```

6.14. **Rectilinear Minimum Spanning Tree.** Given a set of n points in the plane, and the aim is to find a minimum spanning tree connecting these n points, assuming the Manhattan distance is used. The function `candidates` returns at most $4n$ edges that are a superset of the edges in a minimum spanning tree, and then one can use Kruskal's algorithm.

```
#define MAXN 100100 ---------------------------------------//29
struct RMST { ---------------------------------------------//71
- struct point { ------------------------------------------//be
--- int i; ll x, y; ---------------------------------------//a0
--- point() : i(-1) { } -----------------------------------//6e
--- ll d1() { return x + y; } -----------------------------//51
--- ll d2() { return x - y; } -----------------------------//0e
--- ll dist(point other) { --------------------------------//b6
----- return abs(x - other.x) + abs(y - other.y); } -------//c7
--- bool operator <(const point &other) const { -----------//e5
----- return y == other.y ? x > other.x : y < other.y; } --//88
- } best[MAXN], arr[MAXN], tmp[MAXN]; ---------------------//07
- int n; --------------------------------------------------//11
- RMST() : n(0) {} ----------------------------------------//1d
- void add_point(int x, int y) { --------------------------//13
--- arr[arr[n].i = n].x = x, arr[n++].y = y; } ------------//9d
- void rec(int l, int r) { --------------------------------//42
--- if (l >= r) return; -----------------------------------//ab
--- int m = (l+r)/2; --------------------------------------//55
--- rec(l,m), rec(m+1,r); ---------------------------------//61
--- point bst; --------------------------------------------//fa
--- for (int i = l, j = m+1, k = l; i <= m || j <= r; k++) {
----- if (j > r || (i <= m && arr[i].d1() < arr[j].d1())) {//c9
------- tmp[k] = arr[i++]; --------------------------------//4f
------- if (bst.i != -1 && (best[tmp[k].i].i == -1 --------//d0
----------------------- || best[tmp[k].i].d2() < bst.d2()))//72
--------- best[tmp[k].i] = bst; ---------------------------//a2
----- } else { --------------------------------------------//2b
------- tmp[k] = arr[j++]; --------------------------------//17
------- if (bst.i == -1 || bst.d2() < tmp[k].d2()) --------//bc
--------- bst = tmp[k]; } } -------------------------------//a5
--- rep(i,l,r+1) arr[i] = tmp[i]; } -----------------------//10
- vector<pair<ll,ii> > candidates() { ---------------------//65
--- vector<pair<ll, ii> > es; -----------------------------//a6
--- rep(p,0,2) { ------------------------------------------//6f
----- rep(q,0,2) { ----------------------------------------//32
------- sort(arr, arr+n); ---------------------------------//e6
------- rep(i,0,n) best[i].i = -1; ------------------------//a8
------- rec(0,n-1); ---------------------------------------//6a
------- rep(i,0,n) { --------------------------------------//34
--------- if(best[arr[i].i].i != -1) ----------------------//af
----------- es.push_back({arr[i].dist(best[arr[i].i]), ----//90
------------------------- {arr[i].i, best[arr[i].i].i}}); -//94
--------- swap(arr[i].x, arr[i].y); -----------------------//09
--------- arr[i].x *= -1, arr[i].y *= -1; } } -------------//74
----- rep(i,0,n) arr[i].x *= -1; } ------------------------//14
--- return es; } }; ---------------------------------------//84
```

6.15. **Line upper/lower envelope.** To find the upper/lower envelope of a collection of lines $a_i + b_i x$, plot the points (b_i, a_i), add the point $(0, \pm\infty)$ (depending on if upper/lower envelope is desired), and then find the convex hull.

6.16. **Formulas.** Let $a = (a_x, a_y)$ and $b = (b_x, b_y)$ be two-dimensional vectors.

- $a \cdot b = |a||b| \cos\theta$, where θ is the angle between a and b.
- $a \times b = |a||b| \sin\theta$, where θ is the signed angle between a and b.
- $a \times b$ is equal to the area of the parallelogram with two of its sides formed by a and b. Half of that is the area of the triangle formed by a and b.
- The line going through a and b is $Ax+By = C$ where $A = b_y - a_y$, $B = a_x - b_x$, $C = Aa_x + Ba_y$.
- Two lines $A_1x + B_1y = C_1$, $A_2x + B_2y = C_2$ are parallel iff. $D = A_1B_2 - A_2B_1$ is zero. Otherwise their unique intersection is $(B_2C_1 - B_1C_2, A_1C_2 - A_2C_1)/D$.

- **Euler's formula:** $V - E + F = 2$
- Side lengths a, b, c can form a triangle iff. $a + b > c$, $b + c > a$ and $a + c > b$.
- Sum of internal angles of a regular convex n-gon is $(n-2)\pi$.
- **Law of sines:** $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$
- **Law of cosines:** $b^2 = a^2 + c^2 - 2ac\cos B$
- Internal tangents of circles $(c_1, r_1), (c_2, r_2)$ intersect at $(c_1r_2 + c_2r_1)/(r_1 + r_2)$, external intersect at $(c_1r_2 - c_2r_1)/(r_1 + r_2)$.

7. Other Algorithms

7.1. 2SAT.
A fast 2SAT solver.

```
struct { vi adj; int val, num, lo; bool done; } V[2*1000+100];
struct TwoSat { ------------------------------------------//01
- int n, at = 0; vi S; -----------------------------------//3a
- TwoSat(int _n) : n(_n) { -------------------------------//d8
--- rep(i,0,2*n+1) ---------------------------------------//58
----- V[i].adj.clear(), ----------------------------------//77
----- V[i].val = V[i].num = -1, V[i].done = false; } -----//9a
- bool put(int x, int v) { -------------------------------//de
--- return (V[n+x].val &= v) != (V[n-x].val &= 1-v); } ---//26
- void add_or(int x, int y) { ----------------------------//85
--- V[n-x].adj.push_back(n+y), V[n-y].adj.push_back(n+x); }//66
- int dfs(int u) { ---------------------------------------//6d
--- int br = 2, res; -------------------------------------//74
--- S.push_back(u), V[u].num = V[u].lo = at++; ----------//d0
--- iter(v,V[u].adj) { -----------------------------------//31
----- if (V[*v].num == -1) { -----------------------------//99
------- if (!(res = dfs(*v))) return 0; ------------------//08
------- br |= res, V[u].lo = min(V[u].lo, V[*v].lo); -----//82
----- } else if (!V[*v].done) ----------------------------//46
------- V[u].lo = min(V[u].lo, V[*v].num); ---------------//d9
----- br |= !V[*v].val; } --------------------------------//0c
--- res = br - 3; ----------------------------------------//c7
--- if (V[u].num == V[u].lo) rep(i,res+1,2) { ------------//12
----- for (int j = (int)size(S)-1; ; j--) { --------------//3b
------- int v = S[j]; ------------------------------------//db
------- if (i) { -----------------------------------------//e4
--------- if (!put(v-n, res)) return 0; ------------------//8f
--------- V[v].done = true, S.pop_back(); ----------------//0f
------- } else res &= V[v].val; --------------------------//e4
------- if (v == u) break; } -----------------------------//d1
----- res &= 1; } ----------------------------------------//21
--- return br | !res; } ----------------------------------//66
- bool sat() { -------------------------------------------//da
--- rep(i,0,2*n+1) ---------------------------------------//cc
----- if (i != n && V[i].num == -1 && !dfs(i)) return false;
--- return true; } }; ------------------------------------//d7
```

7.2. DPLL Algorithm.
A SAT solver that can solve a random 1000-variable SAT instance within a second.

```
#define IDX(x) ((abs(x)-1)*2+((x)>0)) --------------------//ca
struct SAT { ---------------------------------------------//e3
- int n; -------------------------------------------------//6d
- vi cl, head, tail, val; --------------------------------//85
- vii log; vvi w, loc; -----------------------------------//ff
- SAT() : n(0) { } ---------------------------------------//f3
- int var() { return ++n; } ------------------------------//9a
- void clause(vi vars) { ---------------------------------//5e
--- set<int> seen; iter(it,vars) { -----------------------//66
----- if (seen.find(IDX(*it)^1) != seen.end()) return; ---//f9
----- seen.insert(IDX(*it)); } ---------------------------//4f
--- head.push_back(cl.size()); ---------------------------//1d
--- iter(it,seen) cl.push_back(*it); ---------------------//ad
--- tail.push_back((int)cl.size() - 2); } ----------------//21
- bool assume(int x) { -----------------------------------//58
--- if (val[x^1]) return false; --------------------------//07
--- if (val[x]) return true; -----------------------------//d6
--- val[x] = true; log.push_back(ii(-1, x)); -------------//9e
--- rep(i,0,w[x^1].size()) { -----------------------------//fd
----- int at = w[x^1][i], h = head[at], t = tail[at]; ----//9b
----- log.push_back(ii(at, h)); --------------------------//5c
----- if (cl[t+1] != (x^1)) swap(cl[t], cl[t+1]); --------//40
----- while (h < t && val[cl[h]^1]) h++; -----------------//0c
----- if ((head[at] = h) < t) { --------------------------//68
------- w[cl[h]].push_back(w[x^1][i]); -------------------//cd
------- swap(w[x^1][i--], w[x^1].back()); ----------------//2d
------- w[x^1].pop_back(); -------------------------------//61
------- swap(cl[head[at]++], cl[t+1]); -------------------//a9
----- } else if (!assume(cl[t])) return false; } ---------//3a
--- return true; } ---------------------------------------//f7
- bool bt() { --------------------------------------------//6e
--- int v = log.size(), x; ll b = -1; --------------------//09
--- rep(i,0,n) if (val[2*i] == val[2*i+1]) { -------------//66
----- ll s = 0, t = 0; -----------------------------------//02
----- rep(j,0,2) { iter(it,loc[2*i+j]) -------------------//c1
------- s+=1LL<<max(0,40-tail[*it]+head[*it]); swap(s,t); }//d4
----- if (max(s,t) >= b) b = max(s,t), x = 2*i + (t>=s); } //c1
--- if (b == -1 || (assume(x) && bt())) return true; -----//b6
--- while (log.size() != v) { ----------------------------//2a
----- int p = log.back().first, q = log.back().second; ---//11
----- if (p == -1) val[q] = false; else head[p] = q; -----//90
----- log.pop_back(); } ----------------------------------//c8
--- return assume(x^1) && bt(); } ------------------------//d3
- bool solve() { -----------------------------------------//b4
--- val.assign(2*n+1, false); ----------------------------//41
--- w.assign(2*n+1, vi()); loc.assign(2*n+1, vi()); ------//5b
--- rep(i,0,head.size()) { -------------------------------//18
----- if (head[i] == tail[i]+2) return false; ------------//51
----- rep(at,head[i],tail[i]+2) loc[cl[at]].push_back(i); }//f2
--- rep(i,0,head.size()) if (head[i] < tail[i]+1) rep(t,0,2)
----- w[cl[tail[i]+t]].push_back(i); ---------------------//20
--- rep(i,0,head.size()) if (head[i] == tail[i]+1) -------//0e
----- if (!assume(cl[head[i]])) return false; ------------//e3
--- return bt(); } ---------------------------------------//26
- bool get_value(int x) { return val[IDX(x)]; } }; -------//c2
```

7.3. Stable Marriage.
The Gale-Shapley algorithm for solving the stable marriage problem.

```
vi stable_marriage(int n, int** m, int** w) { ------------//e4
- queue<int> q; ------------------------------------------//f6
- vi at(n, 0), eng(n, -1), res(n, -1); vvi inv(n, vi(n)); -//c3
- rep(i,0,n) rep(j,0,n) inv[i][w[i][j]] = j; -------------//f1
- rep(i,0,n) q.push(i); ----------------------------------//d8
- while (!q.empty()) { -----------------------------------//68
--- int curm = q.front(); q.pop(); -----------------------//e2
--- for (int &i = at[curm]; i < n; i++) { ----------------//7e
----- int curw = m[curm][i]; -----------------------------//95
----- if (eng[curw] == -1) { } ---------------------------//f7
----- else if (inv[curw][curm] < inv[curw][eng[curw]]) ---//d6
------- q.push(eng[curw]); -------------------------------//2e
----- else continue; -------------------------------------//1d
----- res[eng[curw] = curm] = curw, ++i; break; } } ------//34
- return res; } ------------------------------------------//1f
```

7.4. Algorithm X.
An implementation of Knuth's Algorithm X, using dancing links. Solves the Exact Cover problem.

```
bool handle_solution(vi rows) { return false; } ----------//63
struct exact_cover { -------------------------------------//95
- struct node { ------------------------------------------//7e
--- node *l, *r, *u, *d, *p; -----------------------------//19
--- int row, col, size; ----------------------------------//ae
--- node(int _row, int _col) : row(_row), col(_col) { ----//c9
----- size = 0; l = r = u = d = p = NULL; } }; -----------//fe
- int rows, cols, *sol; ----------------------------------//b8
- bool **arr; --------------------------------------------//ea
- node *head; --------------------------------------------//ee
- exact_cover(int _rows, int _cols) ----------------------//fb
--- : rows(_rows), cols(_cols), head(NULL) { -------------//4e
--- arr = new bool*[rows]; -------------------------------//4a
--- sol = new int[rows]; ---------------------------------//14
--- rep(i,0,rows) ----------------------------------------//44
----- arr[i] = new bool[cols], memset(arr[i], 0, cols); } -//28
- void set_value(int row, int col, bool val = true) { ----//d7
--- arr[row][col] = val; } -------------------------------//a7
- void setup() { -----------------------------------------//ef
--- node ***ptr = new node**[rows + 1]; ------------------//9f
--- rep(i,0,rows+1) { ------------------------------------//ca
----- ptr[i] = new node*[cols]; --------------------------//09
----- rep(j,0,cols) --------------------------------------//42
------- if (i == rows || arr[i][j]) ptr[i][j] = new node(i,j);
------- else ptr[i][j] = NULL; } -------------------------//85
--- rep(i,0,rows+1) { ------------------------------------//58
----- rep(j,0,cols) { ------------------------------------//1d
------- if (!ptr[i][j]) continue; ------------------------//92
------- int ni = i + 1, nj = j + 1; ----------------------//50
------- while (true) { -----------------------------------//00
--------- if (ni == rows + 1) ni = 0; --------------------//f4
--------- if (ni == rows || arr[ni][j]) break; -----------//98
--------- ++ni; } ----------------------------------------//af
------- ptr[i][j]->d = ptr[ni][j]; -----------------------//41
------- ptr[ni][j]->u = ptr[i][j]; -----------------------//5c
------- while (true) { -----------------------------------//1c
--------- if (nj == cols) nj = 0; ------------------------//24
--------- if (i == rows || arr[i][nj]) break; ------------//fa
--------- ++nj; } ----------------------------------------//8b
------- ptr[i][j]->r = ptr[i][nj]; -----------------------//85
------- ptr[i][nj]->l = ptr[i][j]; } } -------------------//10
--- head = new node(rows, -1); ---------------------------//68
```

```
--- head->r = ptr[rows][0]; ------------------------------//54
--- ptr[rows][0]->l = head; ------------------------------//f3
--- head->l = ptr[rows][cols - 1]; -----------------------//fd
--- ptr[rows][cols - 1]->r = head; -----------------------//5a
--- rep(j,0,cols) { --------------------------------------//56
----- int cnt = -1; --------------------------------------//34
----- rep(i,0,rows+1) ------------------------------------//44
------- if (ptr[i][j]) cnt++, ptr[i][j]->p = ptr[rows][j]; //95
----- ptr[rows][j]->size = cnt; } ------------------------//a2
--- rep(i,0,rows+1) delete[] ptr[i]; ---------------------//f3
--- delete[] ptr; } --------------------------------------//c6
- #define COVER(c, i, j) \ --------------------------------//bf
--- c->r->l = c->l, c->l->r = c->r; \ --------------------//b2
--- for (node *i = c->d; i != c; i = i->d) \ -------------//d5
----- for (node *j = i->r; j != i; j = j->r) \ -----------//23
------- j->d->u = j->u, j->u->d = j->d, j->p->size--; ----//c3
- #define UNCOVER(c, i, j) \ ------------------------------//67
--- for (node *i = c->u; i != c; i = i->u) \ -------------//eb
----- for (node *j = i->l; j != i; j = j->l) \ -----------//d9
------- j->p->size++, j->d->u = j->u->d = j; \ -----------//0e
--- c->r->l = c->l->r = c; -------------------------------//21
- bool search(int k = 0) { -------------------------------//6f
--- if (head == head->r) { -------------------------------//6d
----- vi res(k); -----------------------------------------//ec
----- rep(i,0,k) res[i] = sol[i]; ------------------------//46
----- sort(res.begin(), res.end()); ----------------------//3d
----- return handle_solution(res); } ---------------------//68
--- node *c = head->r, *tmp = head->r; -------------------//2a
--- for ( ; tmp != head; tmp = tmp->r) -------------------//2f
----- if (tmp->size < c->size) c = tmp; ------------------//28
--- if (c == c->d) return false; -------------------------//3b
--- COVER(c, i, j); --------------------------------------//70
--- bool found = false; ----------------------------------//7f
--- for (node *r = c->d; !found && r != c; r = r->d) { ---//63
----- sol[k] = r->row; -----------------------------------//13
----- for (node *j = r->r; j != r; j = j->r) { -----------//71
------- COVER(j->p, a, b); } -----------------------------//96
----- found = search(k + 1); -----------------------------//1c
----- for (node *j = r->l; j != r; j = j->l) { -----------//1e
------- UNCOVER(j->p, a, b); } } -------------------------//2b
--- UNCOVER(c, i, j); ------------------------------------//48
--- return found; } }; -----------------------------------//5f
```

7.5. ***n*th Permutation.** A very fast algorithm for computing the nth permutation of the list $\{0, 1, \ldots, k-1\}$.

```
vector<int> nth_permutation(int cnt, int n) { ------------//78
- vector<int> idx(cnt), per(cnt), fac(cnt); --------------//9e
- rep(i,0,cnt) idx[i] = i; -------------------------------//bc
- rep(i,1,cnt+1) fac[i - 1] = n % i, n /= i; -------------//2b
- for (int i = cnt - 1; i >= 0; i--) ---------------------//f9
--- per[cnt - i - 1] = idx[fac[i]], ----------------------//a8
--- idx.erase(idx.begin() + fac[i]); ---------------------//39
- return per; } ------------------------------------------//a8
```

7.6. **Cycle-Finding.** An implementation of Floyd's Cycle-Finding algorithm.

```
ii find_cycle(int x0, int (*f)(int)) { -------------------//a5
- int t = f(x0), h = f(t), mu = 0, lam = 1; --------------//8d
- while (t != h) t = f(t), h = f(f(h)); ------------------//79
- h = x0; ------------------------------------------------//04
- while (t != h) t = f(t), h = f(h), mu++; ---------------//9d
- h = f(t); ----------------------------------------------//00
- while (t != h) h = f(h), lam++; ------------------------//5e
- return ii(mu, lam); } ----------------------------------//14
```

7.7. **Longest Increasing Subsequence.**

```
vi lis(vi arr) { -----------------------------------------//99
- if (arr.empty()) return vi(); --------------------------//3c
- vi seq, back(size(arr)), ans; --------------------------//0d
- rep(i,0,size(arr)) { -----------------------------------//10
--- int res = 0, lo = 1, hi = size(seq); -----------------//7d
--- while (lo <= hi) { -----------------------------------//54
----- int mid = (lo+hi)/2; -------------------------------//27
----- if (arr[seq[mid-1]] < arr[i]) res = mid, lo = mid + 1;
----- else hi = mid - 1; } -------------------------------//78
--- if (res < size(seq)) seq[res] = i; -------------------//cf
--- else seq.push_back(i); -------------------------------//10
--- back[i] = res == 0 ? -1 : seq[res-1]; } --------------//5b
- int at = seq.back(); -----------------------------------//25
- while (at != -1) ans.push_back(at), at = back[at]; -----//d3
- reverse(ans.begin(), ans.end()); -----------------------//4a
- return ans; } ------------------------------------------//70
```

7.8. **Dates.** Functions to simplify date calculations.

```
int intToDay(int jd) { return jd % 7; } ------------------//89
int dateToInt(int y, int m, int d) { ---------------------//96
- return 1461 * (y + 4800 + (m - 14) / 12) / 4 + ---------//a8
--- 367 * (m - 2 - (m - 14) / 12 * 12) / 12 - ------------//d1
--- 3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 + ---------//be
--- d - 32075; } -----------------------------------------//b6
void intToDate(int jd, int &y, int &m, int &d) { ---------//64
- int x, n, i, j; ----------------------------------------//e5
- x = jd + 68569; ----------------------------------------//97
- n = 4 * x / 146097; ------------------------------------//54
- x -= (146097 * n + 3) / 4; -----------------------------//dc
- i = (4000 * (x + 1)) / 1461001; ------------------------//ac
- x -= 1461 * i / 4 - 31; --------------------------------//33
- j = 80 * x / 2447; -------------------------------------//f8
- d = x - 2447 * j / 80; ---------------------------------//44
- x = j / 11; --------------------------------------------//24
- m = j + 2 - 12 * x; ------------------------------------//67
- y = 100 * (n - 49) + i + x; } --------------------------//d1
```

7.9. **Simplex.**

```
typedef long double DOUBLE; ------------------------------//c6
typedef vector<DOUBLE> VD; -------------------------------//c3
typedef vector<VD> VVD; ----------------------------------//ae
typedef vector<int> VI; ----------------------------------//51
const DOUBLE EPS = 1e-9; ---------------------------------//66
struct LPSolver { ----------------------------------------//65
 int m, n; -----------------------------------------------//1c
 VI B, N; ------------------------------------------------//a0
 VVD D; --------------------------------------------------//db
 LPSolver(const VVD &A, const VD &b, const VD &c) : -------//4f
- m(b.size()), n(c.size()), ------------------------------//53
- N(n + 1), B(m), D(m + 2, VD(n + 2)) { ------------------//d4
- for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) -//5e
--- D[i][j] = A[i][j]; -----------------------------------//4f
- for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1;//58
--- D[i][n + 1] = b[i]; } --------------------------------//44
- for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
- N[n] = -1; D[m + 1][n] = 1; } --------------------------//8d
 void Pivot(int r, int s) { ------------------------------//77
- double inv = 1.0 / D[r][s]; ----------------------------//22
- for (int i = 0; i < m + 2; i++) if (i != r) ------------//4c
-- for (int j = 0; j < n + 2; j++) if (j != s) -----------//9f
--- D[i][j] -= D[r][j] * D[i][s] * inv; ------------------//5b
- for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
- for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
- D[r][s] = inv; -----------------------------------------//28
- swap(B[r], N[s]); } ------------------------------------//a4
 bool Simplex(int phase) { -------------------------------//17
- int x = phase == 1 ? m + 1 : m; -------------------------//e9
- while (true) { ------------------------------------------//15
-- int s = -1; --------------------------------------------//59
-- for (int j = 0; j <= n; j++) { -------------------------//d1
--- if (phase == 2 && N[j] == -1) continue; ---------------//f2
--- if (s == -1 || D[x][j] < D[x][s] || -------------------//f8
------- D[x][j] == D[x][s] && N[j] < N[s]) s = j; } -------//ed
-- if (D[x][s] > -EPS) return true; -----------------------//35
-- int r = -1; --------------------------------------------//2a
-- for (int i = 0; i < m; i++) { --------------------------//d6
--- if (D[i][s] < EPS) continue; --------------------------//57
--- if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / --//d4
------- D[r][s] || (D[i][n + 1] / D[i][s]) == (D[r][n + 1] /
------- D[r][s]) && B[i] < B[r]) r = i; } -----------------//62
-- if (r == -1) return false; -----------------------------//e3
-- Pivot(r, s); } } ---------------------------------------//fe
 DOUBLE Solve(VD &x) { ------------------------------------//b2
- int r = 0; ----------------------------------------------//f8
- for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1])
--- r = i; ------------------------------------------------//b4
- if (D[r][n + 1] < -EPS) { -------------------------------//39
-- Pivot(r, n); -------------------------------------------//e1
-- if (!Simplex(1) || D[m + 1][n + 1] < -EPS) -------------//0e
---- return -numeric_limits<DOUBLE>::infinity(); ----------//49
-- for (int i = 0; i < m; i++) if (B[i] == -1) { ----------//85
--- int s = -1; -------------------------------------------//8d
--- for (int j = 0; j <= n; j++) --------------------------//9f
---- if (s == -1 || D[i][j] < D[i][s] || ------------------//90
-------- D[i][j] == D[i][s] && N[j] < N[s]) ---------------//c8
------ s = j; ---------------------------------------------//d4
--- Pivot(i, s); } } --------------------------------------//2f
- if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
- x = VD(n); ----------------------------------------------//87
- for (int i = 0; i < m; i++) if (B[i] < n) ---------------//e9
--- x[B[i]] = D[i][n + 1]; --------------------------------//bb
- return D[m][n + 1]; } }; --------------------------------//30
// Two-phase simplex algorithm for solving linear programs //c3
```

```
// of the form -------------------------------------------//21
//     maximize     c^T x ---------------------------------//1d
//     subject to   Ax <= b -------------------------------//6e
//                  x >= 0 --------------------------------//44
// INPUT: A -- an m x n matrix ----------------------------//23
//        b -- an m-dimensional vector --------------------//81
//        c -- an n-dimensional vector --------------------//e5
//        x -- a vector where the optimal solution will be //17
//             stored -------------------------------------//83
// OUTPUT: value of the optimal solution (infinity if -----//d5
//                   unbounded above, nan if infeasible) --//7d
// To use this code, create an LPSolver object with A, b, -//ea
// and c as arguments.  Then, call Solve(x). --------------//2a
// #include <iostream> ------------------------------------//56
// #include <iomanip> -------------------------------------//e6
// #include <vector> --------------------------------------//55
// #include <cmath> ---------------------------------------//a2
// #include <limits> --------------------------------------//ca
// using namespace std; -----------------------------------//21
// int main() { -------------------------------------------//27
//   const int m = 4; -------------------------------------//86
//   const int n = 3; -------------------------------------//b7
//   DOUBLE _A[m][n] = { ----------------------------------//8a
//     { 6, -1, 0 }, --------------------------------------//66
//     { -1, -5, 0 }, -------------------------------------//57
//     { 1, 5, 1 }, ---------------------------------------//6f
//     { -1, -5, -1 } -------------------------------------//0c
//   }; ---------------------------------------------------//06
//   DOUBLE _b[m] = { 10, -4, 5, -5 }; --------------------//80
//   DOUBLE _c[n] = { 1, -1, 0 }; -------------------------//c9
//   VVD A(m); --------------------------------------------//5f
//   VD b(_b, _b + m); ------------------------------------//14
//   VD c(_c, _c + n); ------------------------------------//78
//   for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);
//   LPSolver solver(A, b, c); ----------------------------//e5
//   VD x; ------------------------------------------------//c9
//   DOUBLE value = solver.Solve(x); ----------------------//c3
//   cerr << "VALUE: " << value << endl; // VALUE: 1.29032 //fc
//   cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1 -//3a
//   for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
//   cerr << endl; ----------------------------------------//5f
//   return 0; --------------------------------------------//61
// } ------------------------------------------------------//ab
```

7.10. **Fast Square Testing.** An optimized test for square integers.

```
long long M; -------------------------------------------//a7
void init_is_square() { --------------------------------//cd
- rep(i,0,64) M |= 1ULL << (63-(i*i)%64); } ------------//a6
inline bool is_square(ll x) { --------------------------//14
- if (x == 0) return true; // XXX ----------------------//e4
- if ((M << x) >= 0) return false; ---------------------//70
- int c = __builtin_ctz(x); ----------------------------//ce
- if (c & 1) return false; -----------------------------//8d
- x >>= c; ---------------------------------------------//19
- if ((x&7) - 1) return false; -------------------------//1f
- ll r = sqrt(x); --------------------------------------//19
- return r*r == x; } -----------------------------------//62
```

7.11. **128-bit Integer.** GCC has a 128-bit integer data type named `__int128`. Useful if doing multiplication of 64-bit integers, or something needing a little more than 64-bits to represent. There's also `__float128`.

7.12. **Bit Hacks.**

```
int snoob(int x) { -------------------------------------//73
- int y = x & -x, z = x + y; ---------------------------//12
- return z | ((x ^ z) >> 2) / y; } ---------------------//3d
```

Catalan	$C_0 = 1,\ C_n = \frac{1}{n+1}\binom{2n}{n} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \frac{4n-2}{n+1} C_{n-1}$	
Stirling 1st kind	$\left[{0 \atop 0}\right] = 1,\ \left[{n \atop 0}\right] = \left[{0 \atop n}\right] = 0,\ \left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$	#perms of n objs with exactly k cycles
Stirling 2nd kind	$\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1,\ \left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$	#ways to partition n objs into k nonempty sets
Euler	$\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1,\ \left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$	#perms of n objs with exactly k ascents
Euler 2nd Order	$\left\langle\!\left\langle{n \atop k}\right\rangle\!\right\rangle = (k+1)\left\langle\!\left\langle{n-1 \atop k}\right\rangle\!\right\rangle + (2n-k-1)\left\langle\!\left\langle{n-1 \atop k-1}\right\rangle\!\right\rangle$	#perms of $1,1,2,2,...,n,n$ with exactly k ascents
Bell	$B_1 = 1,\ B_n = \sum_{k=0}^{n-1} B_k\binom{n-1}{k} = \sum_{k=0}^{n}\left\{{n \atop k}\right\}$	#partitions of $1..n$ (Stirling 2nd, no limit on k)

#labeled rooted trees	n^{n-1}
#labeled unrooted trees	n^{n-2}
#forests of k rooted trees	$\frac{k}{n}\binom{n}{k}n^{n-k}$
$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$	$\sum_{i=1}^{n} i^3 = n^2(n+1)^2/4$
$!n = n\times!(n-1) + (-1)^n$	$!n = (n-1)(!(n-1)+!(n-2))$
$\sum_{i=1}^{n}\binom{n}{i}F_i = F_{2n}$	$\sum_i \binom{n-i}{i} = F_{n+1}$
$\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$	$x^k = \sum_{i=0}^{k} i!\left\{{k \atop i}\right\}\binom{x}{i} = \sum_{i=0}^{k}\left\langle{k \atop i}\right\rangle\binom{x+i}{k}$
$a \equiv b \pmod{x,y} \Rightarrow a \equiv b \pmod{\mathrm{lcm}(x,y)}$	$\sum_{d\mid n}\phi(d) = n$
$ac \equiv bc \pmod m \Rightarrow a \equiv b \pmod{\frac{m}{\gcd(c,m)}}$	$(\sum_{d\mid n}\sigma_0(d))^2 = \sum_{d\mid n}\sigma_0(d)^3$
p prime $\Leftrightarrow (p-1)! \equiv -1 \pmod p$	$\gcd(n^a-1, n^b-1) = n^{\gcd(a,b)} - 1$
$\sigma_x(n) = \prod_{i=0}^{r}\frac{p_i^{(a_i+1)x}-1}{p_i^x-1}$	$\sigma_0(n) = \prod_{i=0}^{r}(a_i+1)$
$\sum_{k=0}^{m}(-1)^k\binom{n}{k} = (-1)^m\binom{n-1}{m}$	
$2^{\omega(n)} = O(\sqrt{n})$	$\sum_{i=1}^{n} 2^{\omega(i)} = O(n\log n)$
$d = v_i t + \frac{1}{2}at^2$	$v_f^2 = v_i^2 + 2ad$
$v_f = v_i + at$	$d = \frac{v_i+v_f}{2}t$

7.13. **The Twelvefold Way.** Putting n balls into k boxes.

Balls Boxes	same same	distinct same	same distinct	distinct distinct	Remarks
-	$\mathrm{p}_k(n)$	$\sum_{i=0}^{k}\left\{{n \atop i}\right\}$	$\binom{n+k-1}{k-1}$	k^n	$\mathrm{p}_k(n)$: #partitions of n into $\leq k$ positive parts
size ≥ 1	$\mathrm{p}(n,k)$	$\left\{{n \atop k}\right\}$	$\binom{n-1}{k-1}$	$k!\left\{{n \atop k}\right\}$	$\mathrm{p}(n,k)$: #partitions of n into k positive parts
size ≤ 1	$[n \leq k]$	$[n \leq k]$	$\binom{k}{n}$	$n!\binom{k}{n}$	$[cond]$: 1 if $cond = true$, else 0

8. Useful Information

9. Misc

9.1. Debugging Tips.

- Stack overflow? Recursive DFS on tree that is actually a long path?
- Floating-point numbers
  - Getting `NaN`? Make sure `acos` etc. are not getting values out of their range (perhaps `1+eps`).
  - Rounding negative numbers?
  - Outputting in scientific notation?
- Wrong Answer?
  - Read the problem statement again!
  - Are multiple test cases being handled correctly? Try repeating the same test case many times.
  - Integer overflow?
  - Think very carefully about boundaries of all input parameters
  - Try out possible edge cases:
    - $n = 0, n = -1, n = 1, n = 2^{31} - 1$ or $n = -2^{31}$
    - List is empty, or contains a single element
    - n is even, n is odd
    - Graph is empty, or contains a single vertex
    - Graph is a multigraph (loops or multiple edges)
    - Polygon is concave or non-simple
  - Is initial condition wrong for small cases?
  - Are you sure the algorithm is correct?
  - Explain your solution to someone.
  - Are you using any functions that you don't completely understand? Maybe STL functions?
  - Maybe you (or someone else) should rewrite the solution?
  - Can the input line be empty?
- Run-Time Error?
  - Is it actually Memory Limit Exceeded?

9.2. Solution Ideas.

- Dynamic Programming
  - Parsing CFGs: CYK Algorithm
  - Drop a parameter, recover from others
  - Swap answer and a parameter
  - When grouping: try splitting in two
  - 2^k trick
  - When optimizing
    - Convex hull optimization
      - $\mathrm{dp}[i] = \min_{j<i}\{\mathrm{dp}[j] + b[j] \times a[i]\}$
      - $b[j] \geq b[j+1]$
      - optionally $a[i] \leq a[i+1]$
      - $O(n^2)$ to $O(n)$
    - Divide and conquer optimization
      - $\mathrm{dp}[i][j] = \min_{k<j}\{\mathrm{dp}[i-1][k] + C[k][j]\}$
      - $A[i][j] \leq A[i][j+1]$
      - $O(kn^2)$ to $O(kn \log n)$
      - sufficient: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$ (QI)
    - Knuth optimization
      - $\mathrm{dp}[i][j] = \min_{i<k<j}\{\mathrm{dp}[i][k] + \mathrm{dp}[k][j] + C[i][j]\}$
      - $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
      - $O(n^3)$ to $O(n^2)$
      - sufficient: QI and $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$
- Greedy
- Randomized
- Optimizations
  - Use bitset (/64)
  - Switch order of loops (cache locality)
- Process queries offline
  - Mo's algorithm
- Square-root decomposition
- Precomputation
- Efficient simulation
  - Mo's algorithm
  - Sqrt decomposition
  - Store 2^k jump pointers
- Data structure techniques
  - Sqrt buckets
  - Store 2^k jump pointers
  - 2^k merging trick
- Counting
  - Inclusion-exclusion principle
  - Generating functions
- Graphs
  - Can we model the problem as a graph?
  - Can we use any properties of the graph?
  - Strongly connected components
  - Cycles (or odd cycles)
  - Bipartite (no odd cycles)
    - Bipartite matching
    - Hall's marriage theorem
    - Stable Marriage
  - Cut vertex/bridge
  - Biconnected components
  - Degrees of vertices (odd/even)
  - Trees
    - Heavy-light decomposition
    - Centroid decomposition
    - Least common ancestor
    - Centers of the tree
  - Eulerian path/circuit
  - Chinese postman problem
  - Topological sort
  - (Min-Cost) Max Flow
  - Min Cut
    - Maximum Density Subgraph
  - Huffman Coding
  - Min-Cost Arborescence
  - Steiner Tree
  - Kirchoff's matrix tree theorem
  - Prüfer sequences
  - Lovász Toggle
  - Look at the DFS tree (which has no cross-edges)
  - Is the graph a DFA or NFA?
    - Is it the Synchronizing word problem?
- Mathematics
  - Is the function multiplicative?
  - Look for a pattern
  - Permutations
    - Consider the cycles of the permutation
  - Functions
    - Sum of piecewise-linear functions is a piecewise-linear function
    - Sum of convex (concave) functions is convex (concave)
  - Modular arithmetic
    - Chinese Remainder Theorem
    - Linear Congruence
  - Sieve
  - System of linear equations
  - Values too big to represent?
    - Compute using the logarithm
    - Divide everything by some large value
  - Linear programming
    - Is the dual problem easier to solve?
  - Can the problem be modeled as a different combinatorial problem? Does that simplify calculations?
- Logic
  - 2-SAT
  - XOR-SAT (Gauss elimination or Bipartite matching)
- Meet in the middle
- Only work with the smaller half ($\log(n)$)
- Strings
  - Trie (maybe over something weird, like bits)
  - Suffix array
  - Suffix automaton (+DP?)
  - Aho-Corasick
  - eerTree
  - Work with $S + S$
- Hashing
- Euler tour, tree to array
- Segment trees
  - Lazy propagation
  - Persistent
  - Implicit
  - Segment tree of X
- Geometry
  - Minkowski sum (of convex sets)
  - Rotating calipers
  - Sweep line (horizontally or vertically?)
  - Sweep angle
  - Convex hull
- Fix a parameter (possibly the answer).
- Are there few distinct values?
- Binary search
- Sliding Window (+ Monotonic Queue)
- Computing a Convolution? Fast Fourier Transform
- Computing a 2D Convolution? FFT on each row, and then on each column
- Exact Cover (+ Algorithm X)
- Cycle-Finding
- What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?
- Look at the complement problem

  - Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/Bucket sort

10. Formulas

- **Legendre symbol:** $\left(\frac{a}{b}\right) = a^{(b-1)/2} \pmod b$, b odd prime.
- **Heron's formula:** A triangle with side lengths a, b, c has area $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{a+b+c}{2}$.
- **Pick's theorem:** A polygon on an integer grid strictly containing i lattice points and having b lattice points on the boundary has area $i + \frac{b}{2} - 1$. (Nothing similar in higher dimensions)
- **Euler's totient:** The number of integers less than n that are coprime to n are $n\prod_{p|n}\left(1-\frac{1}{p}\right)$ where each p is a distinct prime factor of n.
- **König's theorem:** In any bipartite graph $G = (L \cup R, E)$, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover. Let U be the set of unmatched vertices in L, and Z be the set of vertices that are either in U or are connected to U by an alternating path. Then $K = (L \setminus Z) \cup (R \cap Z)$ is the minimum vertex cover.
- A minimum Steiner tree for n vertices requires at most $n-2$ additional Steiner vertices.
- The number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.
- **Lagrange polynomial** through points $(x_0, y_0), \ldots, (x_k, y_k)$ is $L(x) = \sum_{j=0}^{k} y_j \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$
- **Hook length formula:** If λ is a Young diagram and $h_\lambda(i, j)$ is the hook-length of cell (i, j), then then the number of Young tableux $d_\lambda = n!/\prod h_\lambda(i, j)$.
- **Möbius inversion formula:** If $f(n) = \sum_{d|n} g(d)$, then $g(n) = \sum_{d|n} \mu(d) f(n/d)$. If $f(n) = \sum_{m=1}^{n} g(\lfloor n/m \rfloor)$, then $g(n) = \sum_{m=1}^{n} \mu(m) f(\lfloor \frac{n}{m} \rfloor)$.
- #primitive pythagorean triples with hypotenuse $< n$ approx $n/(2\pi)$.
- **Frobenius Number:** largest number which can't be expressed as a linear combination of numbers $a_1, \ldots, a_n$ with non-negative coefficients. $g(a_1, a_2) = a_1 a_2 - a_1 - a_2$, $N(a_1, a_2) = (a_1 - 1)(a_2 - 1)/2$. $g(d \cdot a_1, d \cdot a_2, a_3) = d \cdot g(a_1, a_2, a_3) + a_3(d-1)$. An integer $x > (\max_i a_i)^2$ can be expressed in such a way iff. $x \mid \gcd(a_1, \ldots, a_n)$.

10.1. Physics.

- **Snell's law:** $\frac{\sin\theta_1}{v_1} = \frac{\sin\theta_2}{v_2}$

10.2. Markov Chains.

A Markov Chain can be represented as a weighted directed graph of states, where the weight of an edge represents the probability of transitioning over that edge in one timestep. Let $P^{(m)} = (p_{ij}^{(m)})$ be the probability matrix of transitioning from state i to state j in m timesteps, and note that $P^{(1)}$ is the adjacency matrix of the graph. **Chapman-Kolmogorov:** $p_{ij}^{(m+n)} = \sum_k p_{ik}^{(m)} p_{kj}^{(n)}$. It follows that $P^{(m+n)} = P^{(m)} P^{(n)}$ and $P^{(m)} = P^m$. If $p^{(0)}$ is the initial probability distribution (a vector), then $p^{(0)} P^{(m)}$ is the probability distribution after m timesteps.

The return times of a state i is $R_i = \{m \mid p_{ii}^{(m)} > 0\}$, and i is *aperiodic* if $\gcd(R_i) = 1$. A MC is aperiodic if any of its vertices is aperiodic. A MC is *irreducible* if the corresponding graph is strongly connected.

A distribution π is stationary if $\pi P = \pi$. If MC is irreducible then $\pi_i = 1/\mathbb{E}[T_i]$, where T_i is the expected time between two visits at i. π_j/π_i is the expected number of visits at j in between two consecutive visits at i. A MC is *ergodic* if $\lim_{m\to\infty} p^{(0)} P^m = \pi$. A MC is ergodic iff. it is irreducible and aperiodic.

A MC for a random walk in an undirected weighted graph (unweighted graph can be made weighted by adding 1-weights) has $p_{uv} = w_{uv}/\sum_x w_{ux}$. If the graph is connected, then $\pi_u = \sum_x w_{ux}/\sum_v \sum_x w_{vx}$. Such a random walk is aperiodic iff. the graph is not bipartite.

An *absorbing* MC is of the form $P = \begin{pmatrix} Q & R \\ 0 & I_r \end{pmatrix}$. Let $N = \sum_{m=0}^{\infty} Q^m = (I_t - Q)^{-1}$. Then, if starting in state i, the expected number of steps till absorption is the i-th entry in $N1$. If starting in state i, the probability of being absorbed in state j is the (i, j)-th entry of NR.

Many problems on MC can be formulated in terms of a system of recurrence relations, and then solved using Gaussian elimination.

10.3. Burnside's Lemma.

Let G be a finite group that acts on a set X. For each g in G let X^g denote the set of elements in X that are fixed by g. Then the number of orbits

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

$$Z(S_n) = \frac{1}{n} \sum_{l=1}^{n} a_l Z(S_{n-l})$$

10.4. Bézout's identity.

If (x, y) is any solution to $ax + by = d$ (e.g. found by the Extended Euclidean Algorithm), then all solutions are given by

$$\left(x + k\frac{b}{\gcd(a, b)}, y - k\frac{a}{\gcd(a, b)}\right)$$

10.5. Misc.

10.5.1. *Determinants and PM.*

$$det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}$$

$$\begin{aligned} pf(A) &= \frac{1}{2^n n!} \sum_{\sigma \in S_{2n}} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{\sigma(2i-1),\sigma(2i)} \\ &= \sum_{M \in \mathrm{PM}(n)} \operatorname{sgn}(M) \prod_{(i,j) \in M} a_{i,j} \end{aligned}$$

10.5.2. *BEST Theorem.*

Count directed Eulerian cycles. Number of OST given by Kirchoff's Theorem (remove r/c with root) $\#\mathrm{OST}(G, r) \cdot \prod_v (d_v - 1)!$

10.5.3. *Primitive Roots.*

Only exists when n is $2, 4, p^k, 2p^k$, where p odd prime. Assume n prime. Number of primitive roots $\phi(\phi(n))$ Let g be primitive root. All primitive roots are of the form g^k where $k, \phi(p)$ are coprime.
k-roots: $g^{i \cdot \phi(n)/k}$ for $0 \le i < k$

10.5.4. *Sum of primes.*

For any multiplicative f:

$$S(n, p) = S(n, p-1) - f(p) \cdot (S(n/p, p-1) - S(p-1, p-1))$$

10.5.5. *Floor.*

$$\lfloor \lfloor x/y \rfloor / z \rfloor = \lfloor x/(yz) \rfloor$$

$$x\%y = x - y \lfloor x/y \rfloor$$

Practice Contest Checklist

- How many operations per second? Compare to local machine.
- What is the stack size?
- How to use printf/scanf with long long/long double?
- Are `__int128` and `__float128` available?
- Does MLE give RTE or MLE as a verdict? What about stack overflow?
- What is `RAND_MAX`?
- How does the judge handle extra spaces (or missing newlines) in the output?
- Look at documentation for programming languages.
- Try different programming languages: C++, Java and Python.
- Try the submit script.
- Try local programs: i?python[23], factor.
- Try submitting with `assert(false)` and `assert(true)`.
- Return-value from `main`.
- Look for directory with sample test cases.
- Make sure printing works.
- Remove this page from the notebook.