viRUs

Team Reference Document

29/07/2015

Contents

1. Code Templates

1.1. Basic Configuration.

Vim and (Caps Lock = Escape) configuration.

```
o.yqtxmal ekrpat  # setxkbmap dvorak for dvorak on qwerty
setxkbmap -option caps:escape
set -o vi
xset r rate 150 100
cat > ~/.vimrc
set nocp et sw=4 ts=4 sts=4 si cindent hi=1000 nu ru noeb showcmd showmode
syn on | colorscheme slate
```

1.2. C++ Header.

A C++ header.

```
#include <bits/stdc++.h>-------------------------------------------------------// 84
using namespace std;-----------------------------------------------------------// 16
template <class T> int size(const T &x) { return x.size(); }-------------------// 5f
#define rep(i,a,b) for (__typeof(a) i=(a); i<(b); ++i)-------------------------// 6c
#define iter(it,c) for (__typeof((c).begin()) it = (c).begin(); it != (c).end(); ++it)
typedef pair<int, int> ii;-----------------------------------------------------// 44
typedef vector<int> vi;--------------------------------------------------------// 9d
typedef vector<ii> vii;--------------------------------------------------------// eb
typedef long long ll;----------------------------------------------------------// 47
const int INF = 2147483647;----------------------------------------------------// db
-------------------------------------------------------------------------------// d8
const double EPS = 1e-9;-------------------------------------------------------// 18
const double pi = acos(-1);----------------------------------------------------// ec
typedef unsigned long long ull;------------------------------------------------// 30
typedef vector<vi> vvi;--------------------------------------------------------// 92
typedef vector<vii> vvii;------------------------------------------------------// fc
template <class T> T mod(T a, T b) { return (a % b + b) % b; }-----------------// 3f
```

1.3. Java Template.

A Java template.

```
import java.util.*;------------------------------------------------------------// 37
import java.math.*;------------------------------------------------------------// 89
import java.io.*;--------------------------------------------------------------// 28
-------------------------------------------------------------------------------// a3
public class Main {------------------------------------------------------------// 17
----public static void main(String[] args) throws Exception {------------------// 02
--------Scanner in = new Scanner(System.in);-----------------------------------// ef
--------PrintWriter out = new PrintWriter(System.out, false);------------------// 62
--------// code----------------------------------------------------------------// e6
--------out.flush();-----------------------------------------------------------// 56
----}--------------------------------------------------------------------------// 79
}------------------------------------------------------------------------------// 00
```

2. Data Structures

2.1. Union-Find.

```
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]); }---------------// ba
----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]) p[xp] += p[yp], p[yp] = xp;-------------------------// 3a
--------else p[yp] += p[xp], p[xp] = yp;---------------------------------------// 3e
--------return true; }---------------------------------------------------------// 0b
----int size(int x) { return -p[find(x)]; } };---------------------------------// 28
```

2.2. Segment Tree.

```
#ifdef SEG_MIN-----------------------------------------------------------------// 03
const int ID = INF;------------------------------------------------------------// 56
int f(int a, int b) { return min(a, b); }--------------------------------------// 4f
#else--------------------------------------------------------------------------// 0e
const int ID = 0;--------------------------------------------------------------// 3e
int f(int a, int b) { return a + b; }------------------------------------------// dd
#endif-------------------------------------------------------------------------// 16
struct segment_tree {----------------------------------------------------------// ab
----int n; vi data, lazy;------------------------------------------------------// dd
----segment_tree() {}----------------------------------------------------------// 93
----segment_tree(const vi &arr) : n(size(arr)), data(4*n), lazy(4*n,INF) {-----// f1
--------mk(arr, 0, n-1, 0); }--------------------------------------------------// e9
----int mk(const vi &arr, int l, int r, int i) {-------------------------------// 12
--------if (l == r) return data[i] = arr[l];-----------------------------------// 5b
--------int m = (l + r) / 2;---------------------------------------------------// de
--------return data[i] = f(mk(arr, l, m, 2*i+1), mk(arr, m+1, r, 2*i+2)); }----// 0a
----int query(int a, int b) { return q(a, b, 0, n-1, 0); }---------------------// f6
----int q(int a, int b, int l, int r, int i) {---------------------------------// 22
--------propagate(l, r, i);----------------------------------------------------// 12
--------if (r < a || b < l) return ID;-----------------------------------------// c7
--------if (a <= l && r <= b) return data[i];----------------------------------// ce
--------int m = (l + r) / 2;---------------------------------------------------// 7a
--------return f(q(a, b, l, m, 2*i+1), q(a, b, m+1, r, 2*i+2)); }--------------// 5c
----void update(int i, int v) { u(i, v, 0, n-1, 0); }--------------------------// 90
----int u(int i, int v, int l, int r, int j) {---------------------------------// 02
--------propagate(l, r, j);----------------------------------------------------// ae
--------if (r < i || i < l) return data[j];------------------------------------// 92
--------if (l == i && r == i) return data[j] = v;------------------------------// 4a
--------int m = (l + r) / 2;---------------------------------------------------// cb
--------return data[j] = f(u(i, v, l, m, 2*j+1), u(i, v, m+1, r, 2*j+2)); }----// 34
----void range_update(int a, int b, int v) { ru(a, b, v, 0, n-1, 0); }---------// 71
----int ru(int a, int b, int v, int l, int r, int i) {-------------------------// e0
--------propagate(l, r, i);----------------------------------------------------// 19
--------if (l > r) return ID;--------------------------------------------------// cc
--------if (r < a || b < l) return data[i];------------------------------------// d9
--------if (l == r) return data[i] += v;---------------------------------------// 5f
--------if (a <= l && r <= b) return (lazy[i] = v) * (r - l + 1) + data[i];----// 76
--------int m = (l + r) / 2;---------------------------------------------------// e7
--------return data[i] = f(ru(a, b, v, l, m, 2*i+1),---------------------------// 0e
------------------------   ru(a, b, v, m+1, r, 2*i+2));------------------------// f2
----}--------------------------------------------------------------------------// 47
----void propagate(int l, int r, int i) {--------------------------------------// b5
--------if (l > r || lazy[i] == INF) return;-----------------------------------// 83
--------data[i] += lazy[i] * (r - l + 1);--------------------------------------// 99
--------if (l < r) {-----------------------------------------------------------// dd
```

```
------------if (lazy[2*i+1] == INF) lazy[2*i+1] = lazy[i];---------------------// ee
------------else lazy[2*i+1] += lazy[i];---------------------------------------// 72
------------if (lazy[2*i+2] == INF) lazy[2*i+2] = lazy[i];---------------------// dd
------------else lazy[2*i+2] += lazy[i];---------------------------------------// a4
--------}----------------------------------------------------------------------// e9
--------lazy[i] = INF;---------------------------------------------------------// c4
----}--------------------------------------------------------------------------// 2c
};-----------------------------------------------------------------------------// 17
```

2.3. **Fenwick Tree.**

```
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;-------------// 37
--------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) { x1.update(x, m); x0.update(x, c); }-----// 45
----int query(int x) { return x*x1.query(x) + x0.query(x); }-------------------// 73
};-----------------------------------------------------------------------------// 13
void range_update(fenwick_tree_sq &s, int a, int b, int k) {-------------------// 89
----s.update(a, k, k * (1 - a)); s.update(b+1, -k, k * b); }-------------------// 7f
int range_query(fenwick_tree_sq &s, int a, int b) {----------------------------// 15
----return s.query(b) - s.query(a-1); }----------------------------------------// f3
```

2.4. **Matrix.**

```
template <class K> bool eq(K a, K b) { return a == b; }------------------------// 2a
template <> bool eq<double>(double a, double b) { return abs(a - b) < EPS; }---// a7
template <class T>-------------------------------------------------------------// 53
class matrix {-----------------------------------------------------------------// 85
public:------------------------------------------------------------------------// be
----int rows, cols;------------------------------------------------------------// d3
----matrix(int r, int c) : rows(r), cols(c), cnt(r * c) {----------------------// 34
--------data.assign(cnt, T(0)); }----------------------------------------------// d0
----matrix(const matrix& other) : rows(other.rows), cols(other.cols),----------// fe
--------cnt(other.cnt), data(other.data) { }-----------------------------------// ed
----T& operator()(int i, int j) { return at(i, j); }---------------------------// e0
----void operator +=(const matrix& other) {------------------------------------// c9
--------rep(i,0,cnt) data[i] += other.data[i]; }-------------------------------// 2e
----void operator -=(const matrix& other) {------------------------------------// f2
--------rep(i,0,cnt) data[i] -= other.data[i]; }-------------------------------// 52
----void operator *=(T other) {------------------------------------------------// 14
--------rep(i,0,cnt) data[i] *= other; }---------------------------------------// dd
----matrix<T> operator +(const matrix& other) {--------------------------------// cb
--------matrix<T> res(*this); res += other; return res; }----------------------// d5
----matrix<T> operator -(const matrix& other) {--------------------------------// a0
--------matrix<T> res(*this); res -= other; return res; }----------------------// f5
----matrix<T> operator *(T other) {--------------------------------------------// 60
--------matrix<T> res(*this); res *= other; return res; }----------------------// 73
----matrix<T> operator *(const matrix& other) {--------------------------------// 0a
--------matrix<T> res(rows, other.cols);---------------------------------------// d7
--------rep(i,0,rows) rep(j,0,other.cols) rep(k,0,cols)------------------------// 4b
----------------res(i, j) += at(i, k) * other.data[k * other.cols + j];--------// 08
--------return res; }----------------------------------------------------------// 58
----matrix<T> transpose() {----------------------------------------------------// 3a
--------matrix<T> res(cols, rows);---------------------------------------------// fe
--------rep(i,0,rows) rep(j,0,cols) res(j, i) = at(i, j);----------------------// 2b
--------return res; }----------------------------------------------------------// 23
----matrix<T> pow(int p) {-----------------------------------------------------// da
--------matrix<T> res(rows, cols), sq(*this);----------------------------------// e6
--------rep(i,0,rows) res(i, i) = T(1);----------------------------------------// 09
--------while (p) {------------------------------------------------------------// ea
------------if (p & 1) res = res * sq;-----------------------------------------// 66
------------p >>= 1;-----------------------------------------------------------// 17
------------if (p) sq = sq * sq;-----------------------------------------------// 85
--------} return res; }--------------------------------------------------------// 18
----matrix<T> rref(T &det) {---------------------------------------------------// bd
--------matrix<T> mat(*this); det = T(1);--------------------------------------// 6b
--------for (int r = 0, c = 0; c < cols; c++) {--------------------------------// 33
------------int k = r;---------------------------------------------------------// 42
------------while (k < rows && eq<T>(mat(k, c), T(0))) k++;--------------------// 59
------------if (k >= rows) continue;-------------------------------------------// aa
------------if (k != r) {------------------------------------------------------// fd
----------------det *= T(-1);--------------------------------------------------// 06
----------------rep(i,0,cols)--------------------------------------------------// 2f
--------------------swap(mat.at(k, i), mat.at(r, i));--------------------------// 01
------------} det *= mat(r, r);------------------------------------------------// 35
------------T d = mat(r,c);----------------------------------------------------// 31
------------rep(i,0,cols) mat(r, i) /= d;--------------------------------------// 9e
------------rep(i,0,rows) {----------------------------------------------------// d3
----------------T m = mat(i, c);-----------------------------------------------// 0f
----------------if (i != r && !eq<T>(m, T(0)))---------------------------------// ba
--------------------rep(j,0,cols) mat(i, j) -= m * mat(r, j);------------------// 30
------------} r++;-------------------------------------------------------------// 48
--------} return mat; }-------------------------------------------------------// f6
private:-----------------------------------------------------------------------// a6
----int cnt;-------------------------------------------------------------------// 99
----vector<T> data;------------------------------------------------------------// 7a
----inline T& at(int i, int j) { return data[i * cols + j]; }------------------// b6
};-----------------------------------------------------------------------------// b3
```

2.5. **AVL Tree.**

```
#define AVL_MULTISET 0-------------------------------------------------------// b5
-----------------------------------------------------------------------------// 61
```

```
template <class T>----// 22
class avl_tree {----// ff
public:----// f6
----struct node {----// 45
--------T item; node *p, *l, *r;----// a6
--------int size, height;----// 33
--------node(const T &_item, node *_p = NULL) : item(_item), p(_p),----// 4f
--------l(NULL), r(NULL), size(1), height(0) { } };----// 0d
----avl_tree() : root(NULL) { }----// 5d
----node *root;----// 91
----node* find(const T &item) const {----// 65
--------node *cur = root;----// b4
--------while (cur) {----// 8b
------------if (cur->item < item) cur = cur->r;----// 71
------------else if (item < cur->item) cur = cur->l;----// cd
------------else break; }----// 4f
--------return cur; }----// 84
----node* insert(const T &item) {----// 4e
--------node *prev = NULL, **cur = &root;----// 60
--------while (*cur) {----// aa
------------prev = *cur;----// f0
------------if ((*cur)->item < item) cur = &((*cur)->r);----// 1b
#if AVL_MULTISET----// 0a
------------else cur = &((*cur)->l);----// eb
#else----// ff
------------else if (item < (*cur)->item) cur = &((*cur)->l);----// 54
------------else return *cur;----// 54
#endif----// af
--------}----// ec
--------node *n = new node(item, prev);----// eb
--------*cur = n, fix(n); return n; }----// 29
----void erase(const T &item) { erase(find(item)); }----// 67
----void erase(node *n, bool free = true) {----// 58
--------if (!n) return;----// 96
--------if (!n->l && n->r) parent_leg(n) = n->r, n->r->p = n->p;----// 12
--------else if (n->l && !n->r) parent_leg(n) = n->l, n->l->p = n->p;----// 6b
--------else if (n->l && n->r) {----// 6c
------------node *s = successor(n);----// e5
------------erase(s, false);----// 0a
------------s->p = n->p, s->l = n->l, s->r = n->r;----// 5a
------------if (n->l) n->l->p = s;----// 0b
------------if (n->r) n->r->p = s;----// ed
------------parent_leg(n) = s, fix(s);----// 82
------------return;----// e5
--------} else parent_leg(n) = NULL;----// de
--------fix(n->p), n->p = n->l = n->r = NULL;----// 43
--------if (free) delete n; }----// 23
----node* successor(node *n) const {----// 23
--------if (!n) return NULL;----// 37
--------if (n->r) return nth(0, n->r);----// 23
--------node *p = n->p;----// a7
--------while (p && p->r == n) n = p, p = p->p;----// 9e
--------return p; }----// c7
----node* predecessor(node *n) const {----// b4
--------if (!n) return NULL;----// dd
--------if (n->l) return nth(n->l->size-1, n->l);----// 10
--------node *p = n->p;----// ea
--------while (p && p->l == n) n = p, p = p->p;----// 6d
--------return p; }----// e7
----inline int size() const { return sz(root); }----// ef
----void clear() { delete_tree(root), root = NULL; }----// 84
----node* nth(int n, node *cur = NULL) const {----// e4
--------if (!cur) cur = root;----// e5
--------while (cur) {----// 29
------------if (n < sz(cur->l)) cur = cur->l;----// 75
------------else if (n > sz(cur->l)) n -= sz(cur->l) + 1, cur = cur->r;----// cd
------------else break;----// c0
--------} return cur; }----// ed
----int count_less(node *cur) {----// ec
--------int sum = sz(cur->l);----// bf
--------while (cur) {----// 6f
------------if (cur->p && cur->p->r == cur) sum += 1 + sz(cur->p->l);----// 5c
------------cur = cur->p;----// eb
--------} return sum; }----// a0
private:----// d5
----inline int sz(node *n) const { return n ? n->size : 0; }----// 3f
----inline int height(node *n) const { return n ? n->height : -1; }----// a6
----inline bool left_heavy(node *n) const {----// a0
--------return n && height(n->l) > height(n->r); }----// a8
----inline bool right_heavy(node *n) const {----// 27
--------return n && height(n->r) > height(n->l); }----// c8
----inline bool too_heavy(node *n) const {----// 0b
--------return n && abs(height(n->l) - height(n->r)) > 1; }----// f8
----void delete_tree(node *n) {----// fd
--------if (n) { delete_tree(n->l), delete_tree(n->r); delete n; } }----// ef
----node*& parent_leg(node *n) {----// 6a
--------if (!n->p) return root;----// ac
--------if (n->p->l == n) return n->p->l;----// 83
--------if (n->p->r == n) return n->p->r;----// cc
--------assert(false); }----// 20
----void augment(node *n) {----// 72
--------if (!n) return;----// 0e
--------n->size = 1 + sz(n->l) + sz(n->r);----// 93
--------n->height = 1 + max(height(n->l), height(n->r)); }----// 41
----#define rotate(l, r) \----// 62
--------node *l = n->l; \----// 7a
--------l->p = n->p; \----// 2b
--------parent_leg(n) = l; \----// fc
--------n->l = l->r; \----// e8
--------if (l->r) l->r->p = n; \----// d3
--------l->r = n, n->p = l; \----// eb
```

```
--------augment(n), augment(l) // 81
----void left_rotate(node *n) { rotate(r, l); } // 45
----void right_rotate(node *n) { rotate(l, r); } // ca
----void fix(node *n) { // 0d
--------while (n) { augment(n); // 69
------------if (too_heavy(n)) { // 4c
----------------if (left_heavy(n) && right_heavy(n->l)) left_rotate(n->l); // a9
----------------else if (right_heavy(n) && left_heavy(n->r)) // b9
--------------------right_rotate(n->r); // 08
----------------if (left_heavy(n)) right_rotate(n); // 93
----------------else left_rotate(n); // d5
----------------n = n->p; } // 28
------------n = n->p; } } }; // a2
```

2.6. Heap.

```
#define RESIZE // d0
#define SWP(x,y) tmp = x, x = y, y = tmp // fb
struct default_int_cmp { // 8d
----default_int_cmp() { } // 35
----bool operator ()(const int &a, const int &b) { return a < b; } }; // e9
template <class Compare = default_int_cmp> // 30
class heap { // 05
private: // 39
----int len, count, *q, *loc, tmp; // 0a
----Compare _cmp; // 98
----inline bool cmp(int i, int j) { return _cmp(q[i], q[j]); } // a0
----inline void swp(int i, int j) { // 1c
--------SWP(q[i], q[j]), SWP(loc[q[i]], loc[q[j]]); } // 67
----void swim(int i) { // 33
--------while (i > 0) { // 1a
------------int p = (i - 1) / 2; // 77
------------if (!cmp(i, p)) break; // a9
------------swp(i, p), i = p; } } // 93
----void sink(int i) { // ce
--------while (true) { // 3c
------------int l = 2*i + 1, r = l + 1; // b4
------------if (l >= count) break; // d5
------------int m = r >= count || cmp(l, r) ? l : r; // cc
------------if (!cmp(m, i)) break; // 42
------------swp(m, i), i = m; } } // 1d
public: // cd
----heap(int init_len = 128) : count(0), len(init_len), _cmp(Compare()) { // 17
--------q = new int[len], loc = new int[len]; // f8
--------memset(loc, 255, len << 2); } // f7
----~heap() { delete[] q; delete[] loc; } // 09
----void push(int n, bool fix = true) { // b7
--------if (len == count || n >= len) { // 0f
#ifdef RESIZE // a9
------------int newlen = 2 * len; // 22
------------while (n >= newlen) newlen *= 2; // 2f
------------int *newq = new int[newlen], *newloc = new int[newlen]; // e3
------------rep(i,0,len) newq[i] = q[i], newloc[i] = loc[i]; // d7
------------memset(newloc + len, 255, (newlen - len) << 2); // 3e
------------delete[] q, delete[] loc; // 76
------------loc = newloc, q = newq, len = newlen; // 9e
#else // 8e
------------assert(false); // 23
#endif // 2a
--------} // 37
--------assert(loc[n] == -1); // 4e
--------loc[n] = count, q[count++] = n; // cf
--------if (fix) swim(count-1); } // b9
----void pop(bool fix = true) { // 4f
--------assert(count > 0); // 4d
--------loc[q[0]] = -1, q[0] = q[--count], loc[q[0]] = 0; // 66
--------if (fix) sink(0); // bb
----} // bc
----int top() { assert(count > 0); return q[0]; } // 1f
----void heapify() { for (int i = count - 1; i > 0; i--) // d5
--------if (cmp(i, (i - 1) / 2)) swp(i, (i - 1) / 2); } // 43
----void update_key(int n) { // 62
--------assert(loc[n] != -1), swim(loc[n]), sink(loc[n]); } // ca
----bool empty() { return count == 0; } // 7e
----int size() { return count; } // 5f
----void clear() { count = 0, memset(loc, 255, len << 2); } }; // de
```

2.7. Dancing Links.

```
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; // 81
--------} // 2d
----}; // d3
----node *front, *back; // aa
----dancing_links() { front = back = NULL; } // 72
----node *push_back(const T &item) { // 83
--------back = new node(item, back, NULL); // c4
--------if (!front) front = back; // d2
--------return back; // c0
----} // a9
----node *push_front(const T &item) { // 4a
--------front = new node(item, NULL, front); // 47
--------if (!back) back = front; // 10
--------return front; // cf
----} // b6
----void erase(node *n) { // a0
--------if (!n->l) front = n->r; else n->l->r = n->r; // ab
```

```
--------if (!n->r) back = n->l; else n->r->l = n->l; // 1b
----} // 7b
----void restore(node *n) { // 82
--------if (!n->l) front = n; else n->l->r = n; // a5
--------if (!n->r) back = n; else n->r->l = n; // 9d
----} // eb
}; // 5e
```

2.8. Misof Tree.

```
#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) { for (int i = 0; i < BITS; cnt[i++][x]++, x >>= 1); } // 5a
----void erase(int x) { for (int i = 0; i < BITS; cnt[i++][x]--, x >>= 1); } // 49
----int nth(int n) { // 8a
--------int res = 0; // a4
--------for (int i = BITS-1; i >= 0; i--) // 99
------------if (cnt[i][res <<= 1] <= n) n -= cnt[i][res], res |= 1; // f4
--------return res; // 3a
----} // b5
}; // 0a
```

2.9. k-d Tree.

```
#define INC(c) ((c) == K - 1 ? 0 : (c) + 1) // 77
template <int K> // cd
class kd_tree { // 7e
public: // c7
----struct pt { // 78
--------double coord[K]; // d6
--------pt() {} // c1
--------pt(double c[K]) { rep(i,0,K) coord[i] = c[i]; } // 15
--------double dist(const pt &other) const { // a5
------------double sum = 0.0; // 6c
------------rep(i,0,K) sum += pow(coord[i] - other.coord[i], 2.0); // 5e
------------return sqrt(sum); } }; // ba
----struct cmp { // de
--------int c; // a9
--------cmp(int _c) : c(_c) {} // a0
--------bool operator ()(const pt &a, const pt &b) { // 00
------------for (int i = 0, cc; i <= K; i++) { // a7
----------------cc = i == 0 ? c : i - 1; // 36
----------------if (abs(a.coord[cc] - b.coord[cc]) > EPS) // 54
--------------------return a.coord[cc] < b.coord[cc]; // f4
------------} // 72
------------return false; } }; // b9
----struct bb { // 2d
--------pt from, to; // 66
--------bb(pt _from, pt _to) : from(_from), to(_to) {} // 93
--------double dist(const pt &p) { // f4
------------double sum = 0.0; // 16
------------rep(i,0,K) { // fc
----------------if (p.coord[i] < from.coord[i]) // a9
--------------------sum += pow(from.coord[i] - p.coord[i], 2.0); // ed
----------------else if (p.coord[i] > to.coord[i]) // d7
--------------------sum += pow(p.coord[i] - to.coord[i], 2.0); // 7a
------------} // 21
------------return sqrt(sum); } // 53
--------bb bound(double l, int c, bool left) { // 9c
------------pt nf(from.coord), nt(to.coord); // 39
------------if (left) nt.coord[c] = min(nt.coord[c], l); // 2a
------------else nf.coord[c] = max(nf.coord[c], l); // fc
------------return bb(nf, nt); } }; // d7
----struct node { // 04
--------pt p; node *l, *r; // cf
--------node(pt _p, node *_l, node *_r) : p(_p), l(_l), r(_r) {  } }; // cb
----node *root; // 16
----// kd_tree() : root(NULL) { } // 66
----kd_tree(vector<pt> pts) { root = construct(pts, 0, size(pts) - 1, 0); } // 35
----node* construct(vector<pt> &pts, int from, int to, int c) { // 4f
--------if (from > to) return NULL; // 87
--------int mid = from + (to - from) / 2; // ac
--------nth_element(pts.begin() + from, pts.begin() + mid, // c0
--------------------pts.begin() + to + 1, cmp(c)); // d3
--------return new node(pts[mid], construct(pts, from, mid - 1, INC(c)), // 36
------------------------construct(pts, mid + 1, to, INC(c))); } // 97
----bool contains(const pt &p) { return _con(p, root, 0); } // fd
----bool _con(const pt &p, node *n, int c) { // 82
--------if (!n) return false; // d7
--------if (cmp(c)(p, n->p)) return _con(p, n->l, INC(c)); // 46
--------if (cmp(c)(n->p, p)) return _con(p, n->r, INC(c)); // 1c
--------return true; } // 58
----void insert(const pt &p) { _ins(p, root, 0); } // 1e
----void _ins(const pt &p, node* &n, int c) { // 80
--------if (!n) n = new node(p, NULL, NULL); // 3b
--------else if (cmp(c)(p, n->p)) _ins(p, n->l, INC(c)); // cb
--------else if (cmp(c)(n->p, p)) _ins(p, n->r, INC(c)); } // 2b
----void clear() { _clr(root); root = NULL; } // 56
----void _clr(node *n) { if (n) _clr(n->l), _clr(n->r), delete n; } // 43
----pt nearest_neighbour(const pt &p, bool allow_same=true) { // f1
--------assert(root); // c0
--------double mn = INFINITY, cs[K]; // 66
--------rep(i,0,K) cs[i] = -INFINITY; // e4
--------pt from(cs); // d3
--------rep(i,0,K) cs[i] = INFINITY; // c9
--------pt to(cs); // 4e
--------return _nn(p, root, bb(from, to), mn, 0, allow_same).first; // ae
----} // a0
----pair<pt, bool> _nn( // cd
------------const pt &p, node *n, bb b, double &mn, int c, bool same) { // 65
--------if (!n || b.dist(p) > mn) return make_pair(pt(), false); // 6f
--------bool found = same || p.dist(n->p) > EPS, l1 = true, l2 = false; // a1
--------pt resp = n->p; // b7
```

```
--------if (found) mn = min(mn, p.dist(resp));-----------------------------------// 4d
--------node *n1 = n->l, *n2 = n->r;---------------------------------------------// 49
--------rep(i,0,2) {-------------------------------------------------------------// 07
------------if (i == 1 || cmp(c)(n->p, p)) swap(n1, n2), swap(l1, l2);-----------// 2b
------------pair<pt, bool> res =-------------------------------------------------// 6e
----------------_nn(p, n1, b.bound(n->p.coord[c], c, l1), mn, INC(c), same);---// 6a
------------if (res.second && (!found || p.dist(res.first) < p.dist(resp)))----// f6
----------------resp = res.first, found = true;----------------------------------// 37
--------}------------------------------------------------------------------------// 57
--------return make_pair(resp, found); } };--------------------------------------// 05
```

2.10. **Sqrt Decomposition.**

```
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, arr.begin()+min(i+K, cnt))));----// f0
}--------------------------------------------------------------------------------// 03
int split(int at) {--------------------------------------------------------------// 71
----int i = 0;-------------------------------------------------------------------// 8a
----while (i < size(T) && at >= size(T[i].arr))----------------------------------// 6c
--------at -= size(T[i].arr), i++;-----------------------------------------------// 9a
----if (i >= size(T)) return size(T);--------------------------------------------// 83
----if (at == 0) return i;-------------------------------------------------------// 49
----T.insert(T.begin() + i + 1, segment(vi(T[i].arr.begin() + at, T[i].arr.end())));
----T[i] = segment(vi(T[i].arr.begin(), T[i].arr.begin() + at));-----------------// af
----return i + 1;----------------------------------------------------------------// ac
}--------------------------------------------------------------------------------// ea
void insert(int at, int v) {-----------------------------------------------------// 5f
----vi arr; arr.push_back(v);----------------------------------------------------// 6a
----T.insert(T.begin() + split(at), segment(arr));-------------------------------// 67
}--------------------------------------------------------------------------------// cc
void erase(int at) {-------------------------------------------------------------// be
----int i = split(at); split(at + 1);--------------------------------------------// da
----T.erase(T.begin() + i);------------------------------------------------------// 6b
}--------------------------------------------------------------------------------// 4b
```

2.11. **Monotonic Queue.**

```
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(); }-------------------------------------------// d2
};-------------------------------------------------------------------------------// 74
struct min_queue {---------------------------------------------------------------// b4
----min_stack inp, outp;---------------------------------------------------------// 3d
----void push(int x) { inp.push(x); }--------------------------------------------// 6b
----void fix() {-----------------------------------------------------------------// 5d
--------if (outp.empty()) while (!inp.empty())-----------------------------------// 3b
------------outp.push(inp.top()), inp.pop();-------------------------------------// 8e
----}----------------------------------------------------------------------------// 3f
----int top() { fix(); return outp.top(); }--------------------------------------// dc
----int mn() {-------------------------------------------------------------------// 39
--------if (inp.empty()) return outp.mn();---------------------------------------// 01
--------if (outp.empty()) return inp.mn();---------------------------------------// 90
--------return min(inp.mn(), outp.mn()); }---------------------------------------// 97
----void pop() { fix(); outp.pop(); }--------------------------------------------// 4f
----bool empty() { return inp.empty() && outp.empty(); }-------------------------// 65
};-------------------------------------------------------------------------------// 60
```

3. Graphs

3.1. Single-Source Shortest Paths.

3.1.1. *Dijkstra's algorithm.*

```
int *dist, *dad;-----------------------------------------------------------------// 46
struct cmp {---------------------------------------------------------------------// a5
----bool operator()(int a, int b) {----------------------------------------------// bb
--------return dist[a] != dist[b] ? dist[a] < dist[b] : a < b; }-----------------// e6
};-------------------------------------------------------------------------------// 41
pair<int*, int*> dijkstra(int n, int s, vii *adj) {------------------------------// 53
----dist = new int[n];-----------------------------------------------------------// 84
----dad = new int[n];------------------------------------------------------------// 05
----rep(i,0,n) dist[i] = INF, dad[i] = -1;---------------------------------------// 80
----set<int, cmp> pq;------------------------------------------------------------// 98
----dist[s] = 0, pq.insert(s);---------------------------------------------------// 1f
----while (!pq.empty()) {--------------------------------------------------------// 47
--------int cur = *pq.begin(); pq.erase(pq.begin());-----------------------------// 58
--------rep(i,0,size(adj[cur])) {------------------------------------------------// a6
------------int nxt = adj[cur][i].first,-----------------------------------------// a4
----------------ndist = dist[cur] + adj[cur][i].second;--------------------------// 3a
------------if (ndist < dist[nxt]) pq.erase(nxt),--------------------------------// 2d
----------------dist[nxt] = ndist, dad[nxt] = cur, pq.insert(nxt);---------------// eb
--------}------------------------------------------------------------------------// d2
----}----------------------------------------------------------------------------// df
----return pair<int*, int*>(dist, dad);------------------------------------------// e3
}--------------------------------------------------------------------------------// 9b
```

3.1.2. *Bellman-Ford algorithm.*

```
int* bellman_ford(int n, int s, vii* adj, bool& has_negative_cycle) {----------// cf
----has_negative_cycle = false;--------------------------------------------------// 47
----int* dist = new int[n];------------------------------------------------------// 7f
----rep(i,0,n) dist[i] = i == s ? 0 : INF;---------------------------------------// df
----rep(i,0,n-1) rep(j,0,n) if (dist[j] != INF)----------------------------------// 4d
--------rep(k,0,size(adj[j]))----------------------------------------------------// 88
------------dist[adj[j][k].first] = min(dist[adj[j][k].first],-----------------// e1
--------------------dist[j] + adj[j][k].second);-------------------------------// 18
----rep(j,0,n) rep(k,0,size(adj[j]))---------------------------------------------// f8
--------if (dist[j] + adj[j][k].second < dist[adj[j][k].first])----------------// 37
------------has_negative_cycle = true;-------------------------------------------// f0
----return dist;-----------------------------------------------------------------// 78
}--------------------------------------------------------------------------------// a9
```

3.2. **Strongly Connected Components.**

3.2.1. *Kosaraju's algorithm.*

```
#include "../data-structures/union_find.cpp"------------------------------------// 5e
---------------------------------------------------------------------------------// 11
vector<bool> visited;------------------------------------------------------------// 66
vi order;------------------------------------------------------------------------// 9b
---------------------------------------------------------------------------------// a5
void scc_dfs(const vvi &adj, int u) {--------------------------------------------// a1
----int v; visited[u] = true;----------------------------------------------------// e3
----rep(i,0,size(adj[u]))--------------------------------------------------------// 2d
--------if (!visited[v = adj[u][i]]) scc_dfs(adj, v);----------------------------// a2
----order.push_back(u);----------------------------------------------------------// 02
}--------------------------------------------------------------------------------// 53
---------------------------------------------------------------------------------// 63
pair<union_find, vi> scc(const vvi &adj) {---------------------------------------// c2
----int n = size(adj), u, v;-----------------------------------------------------// f8
----order.clear();---------------------------------------------------------------// 20
----union_find uf(n);------------------------------------------------------------// a8
----vi dag;----------------------------------------------------------------------// 61
----vvi rev(n);------------------------------------------------------------------// c5
----rep(i,0,n) rep(j,0,size(adj[i])) rev[adj[i][j]].push_back(i);---------------// 7e
----visited.resize(n), fill(visited.begin(), visited.end(), false);-------------// 80
----rep(i,0,n) if (!visited[i]) scc_dfs(rev, i);---------------------------------// 4e
----fill(visited.begin(), visited.end(), false);---------------------------------// 59
----stack<int> S;----------------------------------------------------------------// bb
----for (int i = n-1; i >= 0; i--) {---------------------------------------------// 96
--------if (visited[order[i]]) continue;-----------------------------------------// db
--------S.push(order[i]), dag.push_back(order[i]);-------------------------------// 68
--------while (!S.empty()) {-----------------------------------------------------// 9e
------------visited[u = S.top()] = true, S.pop(), uf.unite(u, order[i]);--------// b3
------------rep(j,0,size(adj[u])) if (!visited[v = adj[u][j]]) S.push(v);------// 1b
--------}------------------------------------------------------------------------// 61
----}----------------------------------------------------------------------------// 57
----return pair<union_find, vi>(uf, dag);----------------------------------------// 2b
}--------------------------------------------------------------------------------// 92
```

3.3. **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
```

3.4. **Euler Path.**

```
#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);------------------------// c1
----}----------------------------------------------------------------------------// ed
----if ((start == -1) != (end == -1) || (c != 2 && c != 0)) return ii(-1,-1);--// 54
----if (start == -1) start = end = any;------------------------------------------// 5e
----return ii(start, end);-------------------------------------------------------// a2
}--------------------------------------------------------------------------------// eb
bool euler_path() {--------------------------------------------------------------// b4
----ii se = start_end();---------------------------------------------------------// 8a
----int cur = se.first, at = m + 1;----------------------------------------------// b6
----if (cur == -1) return false;-------------------------------------------------// ac
----stack<int> s;----------------------------------------------------------------// 1c
----while (true) {---------------------------------------------------------------// b3
--------if (outdeg[cur] == 0) {--------------------------------------------------// 0d
------------res[--at] = cur;-----------------------------------------------------// bd
------------if (s.empty()) break;------------------------------------------------// c6
------------cur = s.top(); s.pop();----------------------------------------------// 06
```

```
        } else s.push(cur), cur = adj[cur][--outdeg[cur]]; // 9e
    } // a4
    return at == 0; // ac
} // 22
```

3.5. Bipartite Matching.

3.5.1. *Alternating Paths algorithm.*

```
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 || alternating_path(owner[right])) { // f6
            owner[right] = left; return 1; // f2
        } } // 88
    return 0; } // 41
```

3.5.2. *Hopcroft-Karp algorithm.* 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; } // 89
    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]; // 79
            } // b8
        } // 0d
        return dist(-1) != INF; // 43
    } // 2c
    bool dfs(int v) { // 26
        if(v != -1) { // d8
            iter(u, adj[v]) // 99
                if(dist(R[*u]) == dist(v) + 1) // 74
                    if(dfs(R[*u])) { // 40
                        R[*u] = v, L[v] = *u; // 47
                        return true; // a2
                    } // 17
            dist(v) = INF; // 62
            return false; // 3c
        } // 3d
        return true; // ae
    } // 0f
    void add_edge(int i, int j) { adj[i].push_back(j); } // 92
    int maximum_matching() { // a2
        int matching = 0; // 71
        memset(L, -1, sizeof(int) * N); // 72
        memset(R, -1, sizeof(int) * M); // bf
        while(bfs()) rep(i,0,N) // 3e
            matching += L[i] == -1 && dfs(i); // 1d
        return matching; // ec
    } // 8b
}; // b7
```

3.6. Maximum Flow.

3.6.1. *Dinic's algorithm.* An implementation of Dinic's algorithm that runs in $O(|V|^2|E|)$.

```
#define MAXV 2000 // ba
int q[MAXV], d[MAXV]; // e6
struct flow_network { // 12
    struct edge { // 1e
        int v, cap, nxt; // ab
        edge() { } // 38
        edge(int _v, int _cap, int _nxt) : v(_v), cap(_cap), nxt(_nxt) { } // bc
    }; // 6e
    int n, ecnt, *head, *curh; // 46
    vector<edge> e, e_store; // 1f
    flow_network(int _n, int m = -1) : n(_n), ecnt(0) { // d3
        e.reserve(2 * (m == -1 ? n : m)); // 24
        head = new int[n], curh = new int[n]; // 6b
        memset(head, -1, n * sizeof(int)); // 56
    } // 77
    void destroy() { delete[] head; delete[] curh; } // f6
    void reset() { e = e_store; } // 87
    void add_edge(int u, int v, int uv, int vu = 0) { // cd
        e.push_back(edge(v, uv, head[u])); head[u] = ecnt++; // c9
        e.push_back(edge(u, vu, head[v])); head[v] = ecnt++; // 89
    } // 14
    int augment(int v, int t, int f) { // 3f
        if (v == t) return f; // 6d
        for (int &i = curh[v], ret; i != -1; i = e[i].nxt) // f9
            if (e[i].cap > 0 && d[e[i].v] + 1 == d[v]) // cc
                if ((ret = augment(e[i].v, t, min(f, e[i].cap))) > 0) // 1f
                    return (e[i].cap -= ret, e[i^1].cap += ret, ret); // ac
        return 0; // 19
    } // fd
    int max_flow(int s, int t, bool res = true) { // 31
        if(s == t) return 0; // 9d
        e_store = e; // 57
        int f = 0, x, l, r; // 0e
```

```
--------while (true) {----------------------------------------------------------// b5
------------memset(d, -1, n * sizeof(int));-------------------------------------// a8
------------l = r = 0, d[q[r++] = t] = 0;---------------------------------------// 0e
------------while (l < r)-------------------------------------------------------// 7a
----------------for (int v = q[l++], i = head[v]; i != -1; i = e[i].nxt)-------// a2
--------------------if (e[i^1].cap > 0 && d[e[i].v] == -1)---------------------// 29
------------------------d[q[r++] = e[i].v] = d[v]+1;---------------------------// 28
------------if (d[s] == -1) break;----------------------------------------------// a0
------------memcpy(curh, head, n * sizeof(int));--------------------------------// 10
------------while ((x = augment(s, t, INF)) != 0) f += x;----------------------// a6
--------}-----------------------------------------------------------------------// 96
--------if (res) reset();-------------------------------------------------------// 21
--------return f;---------------------------------------------------------------// b6
----}---------------------------------------------------------------------------// 1b
};------------------------------------------------------------------------------// 3b
```

3.6.2. *Edmonds Karp's algorithm.* An implementation of Edmonds Karp's algorithm that runs in $O(|V||E|^2)$.

```
#define MAXV 2000-------------------------------------------------------------// ba
int q[MAXV], d[MAXV], p[MAXV];--------------------------------------------------// 7b
struct flow_network {-----------------------------------------------------------// 5e
----struct edge {---------------------------------------------------------------// fc
--------int v, cap, nxt;--------------------------------------------------------// cb
--------edge(int _v, int _cap, int _nxt) : v(_v), cap(_cap), nxt(_nxt) {  }----// 7a
----};--------------------------------------------------------------------------// 31
----int n, ecnt, *head;---------------------------------------------------------// 39
----vector<edge> e, e_store;----------------------------------------------------// ea
----flow_network(int _n, int m = -1) : n(_n), ecnt(0) {------------------------// 34
--------e.reserve(2 * (m == -1 ? n : m));---------------------------------------// 92
--------memset(head = new int[n], -1, n << 2);----------------------------------// 58
----}---------------------------------------------------------------------------// 3a
----void destroy() { delete[] head; }-------------------------------------------// d5
----void reset() { e = e_store; }-----------------------------------------------// 1b
----void add_edge(int u, int v, int uv, int vu=0) {-----------------------------// 7c
--------e.push_back(edge(v, uv, head[u])); head[u] = ecnt++;--------------------// 4c
--------e.push_back(edge(u, vu, head[v])); head[v] = ecnt++;--------------------// bc
----}---------------------------------------------------------------------------// ef
----int max_flow(int s, int t, bool res = true) {-------------------------------// 12
--------if (s == t) return 0;---------------------------------------------------// d6
--------e_store = e;------------------------------------------------------------// 9e
--------int f = 0, l, r, v;-----------------------------------------------------// 6f
--------while (true) {----------------------------------------------------------// 42
------------memset(d, -1, n << 2);----------------------------------------------// 3b
------------memset(p, -1, n << 2);----------------------------------------------// 92
------------l = r = 0, d[q[r++] = s] = 0;---------------------------------------// 5f
------------while (l < r)-------------------------------------------------------// 2c
----------------for (int u = q[l++], i = head[u]; i != -1; i = e[i].nxt)-------// c6
--------------------if (e[i].cap > 0 &&-----------------------------------------// 8a
----------------------------(d[v = e[i].v] == -1 || d[u] + 1 < d[v]))----------// 2f
------------------------d[v] = d[u] + 1, p[q[r++] = v] = i;---------------------// d5
------------if (p[t] == -1) break;----------------------------------------------// 4f
------------int x = INF, at = p[t];---------------------------------------------// b1
------------while (at != -1) x = min(x, e[at].cap), at = p[e[at^1].v];---------// 8a
------------at = p[t], f += x;--------------------------------------------------// 2d
------------while (at != -1)----------------------------------------------------// cd
----------------e[at].cap -= x, e[at^1].cap += x, at = p[e[at^1].v];-----------// 2e
--------}-----------------------------------------------------------------------// 47
--------if (res) reset();-------------------------------------------------------// 3b
--------return f;---------------------------------------------------------------// bc
----}---------------------------------------------------------------------------// 05
};------------------------------------------------------------------------------// 75
```

3.7. **Minimum Cost Maximum Flow.** Running time is $O(|V|^2|E|\log|V|)$.

```
#define MAXV 2000-------------------------------------------------------------// ba
int d[MAXV], p[MAXV], pot[MAXV];------------------------------------------------// 80
struct cmp {--------------------------------------------------------------------// d1
----bool operator ()(int i, int j) {--------------------------------------------// 8a
--------return d[i] == d[j] ? i < j : d[i] < d[j];-----------------------------// 89
----}---------------------------------------------------------------------------// df
};------------------------------------------------------------------------------// cf
struct flow_network {-----------------------------------------------------------// eb
----struct edge {---------------------------------------------------------------// 9a
--------int v, cap, cost, nxt;--------------------------------------------------// ad
--------edge(int _v, int _cap, int _cost, int _nxt)-----------------------------// ec
------------: v(_v), cap(_cap), cost(_cost), nxt(_nxt) {  }---------------------// c4
----};--------------------------------------------------------------------------// ad
----int n, ecnt, *head;---------------------------------------------------------// 46
----vector<edge> e, e_store;----------------------------------------------------// 4b
----flow_network(int _n, int m = -1) : n(_n), ecnt(0) {------------------------// dd
--------e.reserve(2 * (m == -1 ? n : m));---------------------------------------// e6
--------memset(head = new int[n], -1, n << 2);----------------------------------// 6c
----}---------------------------------------------------------------------------// f3
----void destroy() { delete[] head; }-------------------------------------------// ac
----void reset() { e = e_store; }-----------------------------------------------// 88
----void add_edge(int u, int v, int cost, int uv, int vu=0) {-------------------// b4
--------e.push_back(edge(v, uv, cost, head[u])); head[u] = ecnt++;--------------// 43
--------e.push_back(edge(u, vu, -cost, head[v])); head[v] = ecnt++;-------------// 53
----}---------------------------------------------------------------------------// 16
----ii min_cost_max_flow(int s, int t, bool res = true) {-----------------------// 6d
--------if (s == t) return ii(0, 0);--------------------------------------------// 34
--------e_store = e;------------------------------------------------------------// 70
--------memset(pot, 0, n << 2);-------------------------------------------------// 24
--------int f = 0, c = 0, v;----------------------------------------------------// d4
--------while (true) {----------------------------------------------------------// 29
------------memset(d, -1, n << 2);----------------------------------------------// fd
------------memset(p, -1, n << 2);----------------------------------------------// b7
------------set<int, cmp> q;----------------------------------------------------// d8
------------q.insert(s); d[s] = 0;----------------------------------------------// 1d
------------while (!q.empty()) {------------------------------------------------// 04
----------------int u = *q.begin();---------------------------------------------// dd
----------------q.erase(q.begin());---------------------------------------------// 20
----------------for (int i = head[u]; i != -1; i = e[i].nxt) {------------------// 02
```

```
--------------------if (e[i].cap == 0) continue;-------------------------------// 1c
--------------------int cd = d[u] + e[i].cost + pot[u] - pot[v = e[i].v];------// 1d
--------------------if (d[v] == -1 || cd < d[v]) {-----------------------------// d2
------------------------if (q.find(v) != q.end()) q.erase(q.find(v));----------// e2
------------------------d[v] = cd; p[v] = i;-----------------------------------// f7
------------------------q.insert(v);-------------------------------------------// 74
--------------------}----------------------------------------------------------// 6c
----------------}--------------------------------------------------------------// 1b
------------}------------------------------------------------------------------// da
------------if (p[t] == -1) break;---------------------------------------------// 09
------------int x = INF, at = p[t];--------------------------------------------// e8
------------while (at != -1) x = min(x, e[at].cap), at = p[e[at^1].v];---------// 32
------------at = p[t], f += x;-------------------------------------------------// 43
------------while (at != -1)---------------------------------------------------// 53
----------------e[at].cap -= x, e[at^1].cap += x, at = p[e[at^1].v];-----------// 95
------------c += x * (d[t] + pot[t] - pot[s]);---------------------------------// 44
------------rep(i,0,n) if (p[i] != -1) pot[i] += d[i];-------------------------// 86
--------}----------------------------------------------------------------------// 4e
--------if (res) reset();------------------------------------------------------// d7
--------return ii(f, c);-------------------------------------------------------// 9f
----}--------------------------------------------------------------------------// 4c
};-----------------------------------------------------------------------------// ec
```

3.8. All Pairs Maximum Flow.

3.8.1. *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 $O(|V|^3|E|)$.

```
#include "dinic.cpp"-----------------------------------------------------------// 58
-------------------------------------------------------------------------------// 25
bool same[MAXV];---------------------------------------------------------------// 59
pair<vii, vvi> construct_gh_tree(flow_network &g) {----------------------------// 77
----int n = g.n, v;------------------------------------------------------------// 5d
----vii par(n, ii(0, 0)); vvi cap(n, vi(n, -1));-------------------------------// 49
----rep(s,1,n) {---------------------------------------------------------------// 9e
--------int l = 0, r = 0;------------------------------------------------------// 08
--------par[s].second = g.max_flow(s, par[s].first, false);--------------------// 54
--------memset(d, 0, n * sizeof(int));-----------------------------------------// c8
--------memset(same, 0, n * sizeof(int));--------------------------------------// b7
--------d[q[r++] = s] = 1;-----------------------------------------------------// cb
--------while (l < r) {--------------------------------------------------------// d4
------------same[v = q[l++]] = true;-------------------------------------------// f6
------------for (int i = g.head[v]; i != -1; i = g.e[i].nxt)-------------------// da
----------------if (g.e[i].cap > 0 && d[g.e[i].v] == 0)------------------------// 9a
--------------------d[q[r++] = g.e[i].v] = 1;----------------------------------// 1f
--------}----------------------------------------------------------------------// d4
--------rep(i,s+1,n)-----------------------------------------------------------// 35
------------if (par[i].first == par[s].first && same[i]) par[i].first = s;-----// 93
--------g.reset();-------------------------------------------------------------// 87
----}--------------------------------------------------------------------------// 22
----rep(i,0,n) {---------------------------------------------------------------// 34
--------int mn = INF, cur = i;-------------------------------------------------// ac
--------while (true) {---------------------------------------------------------// c9
------------cap[cur][i] = mn;--------------------------------------------------// 3b
------------if (cur == 0) break;-----------------------------------------------// 37
------------mn = min(mn, par[cur].second), cur = par[cur].first;---------------// e8
--------}----------------------------------------------------------------------// de
----}--------------------------------------------------------------------------// 99
----return make_pair(par, cap);------------------------------------------------// 75
}------------------------------------------------------------------------------// f6
int compute_max_flow(int s, int t, const pair<vii, vvi> &gh) {-----------------// 2a
----if (s == t) return 0;------------------------------------------------------// 7a
----int cur = INF, at = s;-----------------------------------------------------// 57
----while (gh.second[at][t] == -1)---------------------------------------------// e0
--------cur = min(cur, gh.first[at].second), at = gh.first[at].first;----------// 00
----return min(cur, gh.second[at][t]);-----------------------------------------// 09
}------------------------------------------------------------------------------// 07
```

3.9. Heavy-Light Decomposition.

```
#include "../data-structures/segment_tree.cpp"---------------------------------// 16
struct HLD {-------------------------------------------------------------------// 25
----int n, curhead, curloc;----------------------------------------------------// d9
----vi sz, head, parent, loc;--------------------------------------------------// 81
----vvi below; segment_tree values;--------------------------------------------// 96
----HLD(int _n) : n(_n), sz(n, 1), head(n), parent(n, -1), loc(n), below(n) {--// 4f
--------vi tmp(n, ID); values = segment_tree(tmp); }---------------------------// a7
----void add_edge(int u, int v) { below[parent[v] = u].push_back(v); }---------// f8
----void update_cost(int u, int v, int c) {------------------------------------// 12
--------if (parent[v] == u) swap(u, v); assert(parent[u] == v);----------------// 9f
--------values.update(loc[u], c); }--------------------------------------------// 9a
----void csz(int u) { rep(i,0,size(below[u]))----------------------------------// 61
------------csz(below[u][i]), sz[u] += sz[below[u][i]]; }----------------------// e7
----void part(int u) {---------------------------------------------------------// da
--------head[u] = curhead; loc[u] = curloc++;----------------------------------// cc
--------int best = -1;---------------------------------------------------------// 55
--------rep(i,0,size(below[u]))------------------------------------------------// 0f
------------if (best == -1 || sz[below[u][i]] > sz[best]) best = below[u][i];--// 7d
--------if (best != -1) part(best);--------------------------------------------// 26
--------rep(i,0,size(below[u]))------------------------------------------------// 44
------------if (below[u][i] != best) part(curhead = below[u][i]); }------------// 84
----void build() { int u = curloc = 0;-----------------------------------------// c4
--------while (parent[u] != -1) u++;-------------------------------------------// d4
--------csz(u); part(curhead = u); }-------------------------------------------// 38
----int lca(int u, int v) {----------------------------------------------------// 81
--------vi uat, vat; int res = -1;---------------------------------------------// e9
--------while (u != -1) uat.push_back(u), u = parent[head[u]];-----------------// 5c
--------while (v != -1) vat.push_back(v), v = parent[head[v]];-----------------// 1f
--------u = size(uat) - 1, v = size(vat) - 1;----------------------------------// a4
--------while (u >= 0 && v >= 0 && head[uat[u]] == head[vat[v]])---------------// fe
------------res = (loc[uat[u]] < loc[vat[v]] ? uat[u] : vat[v]), u--, v--;-----// 13
--------return res; }----------------------------------------------------------// 3b
----int query_upto(int u, int v) { int res = ID;-------------------------------// bf
--------while (head[u] != head[v])---------------------------------------------// 81
```

```
------------res = f(res, values.query(loc[head[u]], loc[u]));-------------------// 66
------------u = parent[head[u]];------------------------------------------------// 72
--------return f(res, values.query(loc[v] + 1, loc[u])); }----------------------// d7
----int query(int u, int v) { int l = lca(u, v);--------------------------------// fb
--------return f(query_upto(u, l), query_upto(v, l)); } };----------------------// 21
```

3.10. **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);--------------------------------------------------// 6e
----}---------------------------------------------------------------------------// 6b
----void query(int x, int y) {--------------------------------------------------// d3
--------queries[x].push_back(ii(y, size(answers)));-----------------------------// a0
--------queries[y].push_back(ii(x, size(answers)));-----------------------------// 14
--------answers.push_back(-1);--------------------------------------------------// ca
----}---------------------------------------------------------------------------// 6b
----void process(int u) {-------------------------------------------------------// 85
--------ancestor[u] = u;--------------------------------------------------------// 1a
--------rep(i,0,size(adj[u])) {-------------------------------------------------// ce
------------int v = adj[u][i];--------------------------------------------------// dd
------------process(v);---------------------------------------------------------// e8
------------uf.unite(u,v);------------------------------------------------------// 55
------------ancestor[uf.find(u)] = u;-------------------------------------------// 1d
--------}-----------------------------------------------------------------------// 57
--------colored[u] = true;------------------------------------------------------// b9
--------rep(i,0,size(queries[u])) {---------------------------------------------// d7
------------int v = queries[u][i].first;----------------------------------------// 89
------------if (colored[v]) {---------------------------------------------------// cb
----------------answers[queries[u][i].second] = ancestor[uf.find(v)];-----------// 63
------------}-------------------------------------------------------------------// d0
--------}-----------------------------------------------------------------------// 40
----}---------------------------------------------------------------------------// a9
};------------------------------------------------------------------------------// 1e
```

4. Strings

4.1. **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) { return a.nr < b.nr; }---------// 77
struct suffix_array {-----------------------------------------------------------// 87
----string s; int n; vvi P; vector<entry> L; vi idx;----------------------------// b6
----suffix_array(string _s) : s(_s), n(size(s)) {-------------------------------// a3
--------L = vector<entry>(n), P.push_back(vi(n)), idx = vi(n);------------------// 12
--------rep(i,0,n) P[0][i] = s[i];----------------------------------------------// 5c
--------for (int stp = 1, cnt = 1; cnt >> 1 < n; stp++, cnt <<= 1) {------------// 86
------------P.push_back(vi(n));-------------------------------------------------// 53
------------rep(i,0,n)----------------------------------------------------------// 6f
----------------L[L[i].p = i].nr = ii(P[stp - 1][i],----------------------------// e2
------------------------i + cnt < n ? P[stp - 1][i + cnt] : -1);----------------// 43
------------sort(L.begin(), L.end());-------------------------------------------// 5f
------------rep(i,0,n)----------------------------------------------------------// a8
----------------P[stp][L[i].p] = i > 0 &&---------------------------------------// 3a
--------------------L[i].nr == L[i - 1].nr ? P[stp][L[i - 1].p] : i;------------// 55
--------}-----------------------------------------------------------------------// 8b
--------rep(i,0,n) idx[P[size(P) - 1][i]] = i;----------------------------------// 17
----}---------------------------------------------------------------------------// d9
----int lcp(int x, int y) {-----------------------------------------------------// 71
--------int res = 0;------------------------------------------------------------// d6
--------if (x == y) return n - x;-----------------------------------------------// bc
--------for (int k = size(P) - 1; k >= 0 && x < n && y < n; k--)----------------// fe
------------if (P[k][x] == P[k][y]) x += 1 << k, y += 1 << k, res += 1 << k;---// b7
--------return res;-------------------------------------------------------------// bc
----}---------------------------------------------------------------------------// f1
};------------------------------------------------------------------------------// f6
```

4.2. **Aho-Corasick Algorithm.**

```
struct aho_corasick {-----------------------------------------------------------// 78
----struct out_node {-----------------------------------------------------------// 3e
--------string keyword; out_node *next;-----------------------------------------// f0
--------out_node(string k, out_node *n) : keyword(k), next(n) { }---------------// 26
----};--------------------------------------------------------------------------// b9
----struct go_node {------------------------------------------------------------// 40
--------map<char, go_node*> next;-----------------------------------------------// 6b
--------out_node *out; go_node *fail;-------------------------------------------// 3e
--------go_node() { out = NULL; fail = NULL; }----------------------------------// 0f
----};--------------------------------------------------------------------------// c0
----go_node *go;----------------------------------------------------------------// b8
----aho_corasick(vector<string> keywords) {-------------------------------------// 4b
--------go = new go_node();-----------------------------------------------------// 77
--------iter(k, keywords) {-----------------------------------------------------// f2
------------go_node *cur = go;--------------------------------------------------// a2
------------iter(c, *k)---------------------------------------------------------// 6e
----------------cur = cur->next.find(*c) != cur->next.end() ? cur->next[*c] :--// 97
--------------------(cur->next[*c] = new go_node());----------------------------// af
------------cur->out = new out_node(*k, cur->out);------------------------------// 3f
--------}-----------------------------------------------------------------------// eb
--------queue<go_node*> q;------------------------------------------------------// 2c
--------iter(a, go->next) q.push(a->second);------------------------------------// db
--------while (!q.empty()) {----------------------------------------------------// 07
------------go_node *r = q.front(); q.pop();------------------------------------// e0
------------iter(a, r->next) {--------------------------------------------------// 18
----------------go_node *s = a->second;-----------------------------------------// 55
----------------q.push(s);------------------------------------------------------// b5
----------------go_node *st = r->fail;------------------------------------------// 53
```

```
----------------while (st && st->next.find(a->first) == st->next.end())--------// 0e
--------------------st = st->fail;-----------------------------------------------// b3
----------------if (!st) st = go;------------------------------------------------// 0b
----------------s->fail = st->next[a->first];------------------------------------// c1
----------------if (s->fail) {---------------------------------------------------// 98
--------------------if (!s->out) s->out = s->fail->out;--------------------------// ad
--------------------else {-------------------------------------------------------// 5b
------------------------out_node* out = s->out;----------------------------------// b8
------------------------while (out->next) out = out->next;-----------------------// b4
------------------------out->next = s->fail->out;--------------------------------// 62
--------------------}------------------------------------------------------------// a6
----------------}----------------------------------------------------------------// 81
------------}--------------------------------------------------------------------// 55
--------}------------------------------------------------------------------------// bf
----}----------------------------------------------------------------------------// de
----vector<string> search(string s) {--------------------------------------------// c4
--------vector<string> res;------------------------------------------------------// 79
--------go_node *cur = go;-------------------------------------------------------// 85
--------iter(c, s) {-------------------------------------------------------------// 57
------------while (cur && cur->next.find(*c) == cur->next.end())-----------------// df
----------------cur = cur->fail;-------------------------------------------------// b1
------------if (!cur) cur = go;--------------------------------------------------// 92
------------cur = cur->next[*c];-------------------------------------------------// 97
------------if (!cur) cur = go;--------------------------------------------------// 01
------------for (out_node *out = cur->out; out; out = out->next)-----------------// d7
----------------res.push_back(out->keyword);-------------------------------------// 7c
--------}------------------------------------------------------------------------// 56
--------return res;--------------------------------------------------------------// 6b
----}----------------------------------------------------------------------------// 3e
};-------------------------------------------------------------------------------// de
```

4.3. **The Z algorithm.**

```
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;--------------------------------------------------------------------// 78
}--------------------------------------------------------------------------------// 16
```

4.4. **Eertree.**

```
#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;
--------if (!st[p].to[c-BASE]) {---------------------------------------------------// 82
------------int q = last = sz++;---------------------------------------------------// 42
------------st[p].to[c-BASE] = q;--------------------------------------------------// fc
------------st[q].len = st[p].len + 2;---------------------------------------------// c5
------------do { p = st[p].link;---------------------------------------------------// 04
------------} while (p != -1 && (n < st[p].len + 2 || c != s[n - st[p].len - 2]));
------------if (p == -1) st[q].link = 1;-------------------------------------------// 77
------------else st[q].link = st[p].to[c-BASE];------------------------------------// 6a
------------return 1; }------------------------------------------------------------// 29
--------last = st[p].to[c-BASE];---------------------------------------------------// 42
--------return 0; } };-------------------------------------------------------------// ec
```

5. Mathematics

5.1. Big Integer.

```
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) : sign(other.sign), data(other.data) { }---------------// 3b
----int sign;----------------------------------------------------------------------// 26
----vector<unsigned int> data;-----------------------------------------------------// 19
----static const int dcnt = 9;-----------------------------------------------------// 12
----static const unsigned int radix = 1000000000U;---------------------------------// f0
----int size() const { return data.size(); }---------------------------------------// 29
----void init(string n) {----------------------------------------------------------// 13
--------intx res; res.data.clear();------------------------------------------------// 4e
--------if (n.empty()) n = "0";----------------------------------------------------// 99
--------if (n[0] == '-') res.sign = -1, n = n.substr(1);---------------------------// 3b
--------for (int i = n.size() - 1; i >= 0; i -= intx::dcnt) {----------------------// e7
------------unsigned int digit = 0;------------------------------------------------// 98
------------for (int j = intx::dcnt - 1; j >= 0; j--) {----------------------------// 72
----------------int idx = i - j;---------------------------------------------------// cd
----------------if (idx < 0) continue;---------------------------------------------// 52
----------------digit = digit * 10 + (n[idx] - '0');-------------------------------// 1f
------------}----------------------------------------------------------------------// c0
------------res.data.push_back(digit);---------------------------------------------// 07
```

```
--------}---------------------------------------------------------------// fb
--------data = res.data;------------------------------------------------// 7d
--------normalize(res.sign);--------------------------------------------// 76
----}-------------------------------------------------------------------// 6e
----intx& normalize(int nsign) {----------------------------------------// 3b
--------if (data.empty()) data.push_back(0);----------------------------// fa
--------for (int i = data.size() - 1; i > 0 && data[i] == 0; i--)------// 27
------------data.erase(data.begin() + i);-------------------------------// 67
--------sign = data.size() == 1 && data[0] == 0 ? 1 : nsign;-----------// ff
--------return *this;---------------------------------------------------// 40
----}-------------------------------------------------------------------// ac
----friend ostream& operator <<(ostream& outs, const intx& n) {--------// 0d
--------if (n.sign < 0) outs << '-';------------------------------------// c0
--------bool first = true;----------------------------------------------// 33
--------for (int i = n.size() - 1; i >= 0; i--) {-----------------------// 63
------------if (first) outs << n.data[i], first = false;----------------// 33
------------else {------------------------------------------------------// 1f
----------------unsigned int cur = n.data[i];---------------------------// 0f
----------------stringstream ss; ss << cur;-----------------------------// 8c
----------------string s = ss.str();------------------------------------// 64
----------------int len = s.size();-------------------------------------// 0d
----------------while (len < intx::dcnt) outs << '0', len++;-----------// 0a
----------------outs << s;----------------------------------------------// 97
------------}-----------------------------------------------------------// f7
--------}---------------------------------------------------------------// e9
--------return outs;----------------------------------------------------// cf
----}-------------------------------------------------------------------// b9
----string to_string() const { stringstream ss; ss << *this; return ss.str(); }// fc
----bool operator <(const intx& b) const {------------------------------// 21
--------if (sign != b.sign) return sign < b.sign;-----------------------// cf
--------if (size() != b.size())-----------------------------------------// 4d
------------return sign == 1 ? size() < b.size() : size() > b.size();--// 4d
--------for (int i = size() - 1; i >= 0; i--) if (data[i] != b.data[i])--------// 35
----------------return sign == 1 ? data[i] < b.data[i] : data[i] > b.data[i];--// 27
--------return false;---------------------------------------------------// ca
----}-------------------------------------------------------------------// 32
----intx operator -() const { intx res(*this); res.sign *= -1; return res; }--// 9d
----friend intx abs(const intx &n) { return n < 0 ? -n : n; }-----------// 02
----intx operator +(const intx& b) const {------------------------------// f8
--------if (sign > 0 && b.sign < 0) return *this - (-b);----------------// 36
--------if (sign < 0 && b.sign > 0) return b - (-*this);----------------// 70
--------if (sign < 0 && b.sign < 0) return -((-*this) + (-b));----------// 59
--------intx c; c.data.clear();-----------------------------------------// 18
--------unsigned long long carry = 0;-----------------------------------// 5c
--------for (int i = 0; i < size() || i < b.size() || carry; i++) {----// e3
------------carry += (i < size() ? data[i] : 0ULL) +--------------------// 91
----------------(i < b.size() ? b.data[i] : 0ULL);----------------------// 0c
------------c.data.push_back(carry % intx::radix);----------------------// 86
------------carry /= intx::radix;---------------------------------------// fd
--------}---------------------------------------------------------------// 50
--------return c.normalize(sign);---------------------------------------// 20
```

```
----}-------------------------------------------------------------------// 70
----intx operator -(const intx& b) const {------------------------------// 53
--------if (sign > 0 && b.sign < 0) return *this + (-b);----------------// 8f
--------if (sign < 0 && b.sign > 0) return -(-*this + b);---------------// 1b
--------if (sign < 0 && b.sign < 0) return (-b) - (-*this);-------------// a1
--------if (*this < b) return -(b - *this);-----------------------------// 36
--------intx c; c.data.clear();-----------------------------------------// 6b
--------long long borrow = 0;-------------------------------------------// f8
--------rep(i,0,size()) {-----------------------------------------------// a7
------------borrow = data[i] - borrow - (i < b.size() ? b.data[i] : 0ULL);----// a5
------------c.data.push_back(borrow < 0 ? intx::radix + borrow : borrow);------// 9b
------------borrow = borrow < 0 ? 1 : 0;--------------------------------// fb
--------}---------------------------------------------------------------// dd
--------return c.normalize(sign);---------------------------------------// 5c
----}-------------------------------------------------------------------// 5e
----intx operator *(const intx& b) const {------------------------------// b3
--------intx c; c.data.assign(size() + b.size() + 1, 0);----------------// 3a
--------rep(i,0,size()) {-----------------------------------------------// 0f
------------long long carry = 0;----------------------------------------// 15
------------for (int j = 0; j < b.size() || carry; j++) {--------------// 95
----------------if (j < b.size()) carry += (long long)data[i] * b.data[j];----// 6d
----------------carry += c.data[i + j];---------------------------------// c6
----------------c.data[i + j] = carry % intx::radix;--------------------// a8
----------------carry /= intx::radix;-----------------------------------// dc
------------}-----------------------------------------------------------// e3
--------}---------------------------------------------------------------// f0
--------return c.normalize(sign * b.sign);------------------------------// 09
----}-------------------------------------------------------------------// a7
----friend pair<intx,intx> divmod(const intx& n, const intx& d) {------// 40
--------assert(!(d.size() == 1 && d.data[0] == 0));--------------------// 42
--------intx q, r; q.data.assign(n.size(), 0);-------------------------// 5e
--------for (int i = n.size() - 1; i >= 0; i--) {-----------------------// 52
------------r.data.insert(r.data.begin(), 0);---------------------------// cb
------------r = r + n.data[i];------------------------------------------// ea
------------long long k = 0;--------------------------------------------// dd
------------if (d.size() < r.size())------------------------------------// 4d
----------------k = (long long)intx::radix * r.data[d.size()];---------// d2
------------if (d.size() - 1 < r.size()) k += r.data[d.size() - 1];----// af
------------k /= d.data.back();-----------------------------------------// 85
------------r = r - abs(d) * k;-----------------------------------------// 3b
------------// if (r < 0) for (ll t = 1LL << 62; t >= 1; t >>= 1) {-----// 0e
------------//---- intx dd = abs(d) * t;--------------------------------// 9d
------------//---- while (r + dd < 0) r = r + dd, k -= t; }-------------// a1
------------while (r < 0) r = r + abs(d), k--;--------------------------// cb
------------q.data[i] = k;----------------------------------------------// 1a
--------}---------------------------------------------------------------// 3c
--------return pair<intx, intx>(q.normalize(n.sign * d.sign), r);------// 9e
----}-------------------------------------------------------------------// a7
----intx operator /(const intx& d) const {------------------------------// 22
--------return divmod(*this,d).first; }---------------------------------// c3
----intx operator %(const intx& d) const {------------------------------// 32
```

```
--------return divmod(*this,d).second * sign; }--------------------------------// 0c
};-----------------------------------------------------------------------------// 64
```

5.1.1. *Fast Multiplication.*

```
#include "intx.cpp"-----------------------------------------------------------// 83
#include "fft.cpp"------------------------------------------------------------// 13
------------------------------------------------------------------------------// e0
intx fastmul(const intx &an, const intx &bn) {--------------------------------// ab
----string as = an.to_string(), bs = bn.to_string();--------------------------// 32
----int n = size(as), m = size(bs), l = 1,------------------------------------// dc
--------len = 5, radix = 100000,----------------------------------------------// 4f
--------*a = new int[n], alen = 0,--------------------------------------------// b8
--------*b = new int[m], blen = 0;--------------------------------------------// 0a
----memset(a, 0, n << 2);-----------------------------------------------------// 1d
----memset(b, 0, m << 2);-----------------------------------------------------// 01
----for (int i = n - 1; i >= 0; i -= len, alen++)-----------------------------// 6e
--------for (int j = min(len - 1, i); j >= 0; j--)----------------------------// 43
------------a[alen] = a[alen] * 10 + as[i - j] - '0';-------------------------// 14
----for (int i = m - 1; i >= 0; i -= len, blen++)-----------------------------// b6
--------for (int j = min(len - 1, i); j >= 0; j--)----------------------------// ae
------------b[blen] = b[blen] * 10 + bs[i - j] - '0';-------------------------// 9b
----while (l < 2*max(alen,blen)) l <<= 1;-------------------------------------// 51
----cpx *A = new cpx[l], *B = new cpx[l];-------------------------------------// 0d
----rep(i,0,l) A[i] = cpx(i < alen ? a[i] : 0, 0);----------------------------// ff
----rep(i,0,l) B[i] = cpx(i < blen ? b[i] : 0, 0);----------------------------// 7f
----fft(A, l); fft(B, l);-----------------------------------------------------// 77
----rep(i,0,l) A[i] *= B[i];--------------------------------------------------// 1c
----fft(A, l, true);----------------------------------------------------------// ec
----ull *data = new ull[l];---------------------------------------------------// f1
----rep(i,0,l) data[i] = (ull)(round(real(A[i])));----------------------------// e2
----rep(i,0,l-1)--------------------------------------------------------------// c8
--------if (data[i] >= (unsigned int)(radix)) {-------------------------------// 03
------------data[i+1] += data[i] / radix;-------------------------------------// 48
------------data[i] %= radix;-------------------------------------------------// 94
--------}---------------------------------------------------------------------// 47
----int stop = l-1;-----------------------------------------------------------// 92
----while (stop > 0 && data[stop] == 0) stop--;-------------------------------// 5b
----stringstream ss;----------------------------------------------------------// a6
----ss << data[stop];---------------------------------------------------------// f3
----for (int i = stop - 1; i >= 0; i--)---------------------------------------// 7b
--------ss << setfill('0') << setw(len) << data[i];---------------------------// 41
----delete[] A; delete[] B;---------------------------------------------------// dd
----delete[] a; delete[] b;---------------------------------------------------// 77
----delete[] data;------------------------------------------------------------// 5e
----return intx(ss.str());----------------------------------------------------// 88
}-----------------------------------------------------------------------------// d8
```

5.2. **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.

```
int nck(int n, int k) {-------------------------------------------------------// f6
----if (n - k < k) k = n - k;-------------------------------------------------// 18
----int res = 1;--------------------------------------------------------------// cb
----rep(i,1,k+1) res = res * (n - (k - i)) / i;-------------------------------// 16
----return res;---------------------------------------------------------------// 6d
}-----------------------------------------------------------------------------// 3d
```

5.3. **Euclidean algorithm.**

```
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }------------------// d9
```

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

```
int egcd(int a, int b, int& x, int& y) {--------------------------------------// 85
----if (b == 0) { x = 1; y = 0; return a; }-----------------------------------// 7b
----else {--------------------------------------------------------------------// 00
--------int d = egcd(b, a % b, x, y);-----------------------------------------// 34
--------x -= a / b * y;-------------------------------------------------------// 4a
--------swap(x, y);-----------------------------------------------------------// 26
--------return d;-------------------------------------------------------------// db
----}-------------------------------------------------------------------------// 9e
}-----------------------------------------------------------------------------// 40
```

5.4. **Trial Division Primality Testing.**

```
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
----int s = static_cast<int>(sqrt(static_cast<double>(n)));-------------------// 64
----for (int i = 5; i <= s; i += 6)-------------------------------------------// 6c
--------if (n % i == 0 || n % (i + 2) == 0) return false;---------------------// e9
----return true; }------------------------------------------------------------// 43
```

5.5. **Miller-Rabin 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
```

5.6. Sieve of Eratosthenes.

```
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) if (prime[i]) primes.push_back((i << 1) + 3);------------// 29
----delete[] prime; // can be used for O(1) lookup-----------------------------// 36
----return primes; }-----------------------------------------------------------// 72
```

5.7. Modular Multiplicative Inverse.

```
#include "egcd.cpp"------------------------------------------------------------// 55
-------------------------------------------------------------------------------// e8
int mod_inv(int a, int m) {----------------------------------------------------// 49
----int x, y, d = egcd(a, m, x, y);--------------------------------------------// 3e
----if (d != 1) return -1;-----------------------------------------------------// 20
----return x < 0 ? x + m : x;--------------------------------------------------// 3c
}------------------------------------------------------------------------------// 69
```

5.8. Chinese Remainder Theorem.

```
#include "egcd.cpp"------------------------------------------------------------// 55
int crt(const vi& as, const vi& ns) {------------------------------------------// c3
----int cnt = size(as), N = 1, x = 0, r, s, l;---------------------------------// 55
----rep(i,0,cnt) N *= ns[i];---------------------------------------------------// b1
----rep(i,0,cnt) egcd(ns[i], l = N/ns[i], r, s), x += as[i] * s * l;-----------// 21
----return mod(x, N); }--------------------------------------------------------// b2
```

5.9. Linear Congruence Solver.

A function that returns all solutions to $ax \equiv b \pmod{n}$, modulo n.

```
#include "egcd.cpp"------------------------------------------------------------// 55
vi linear_congruence(int a, int b, int n) {------------------------------------// c8
----int x, y, d = egcd(a, n, x, y);--------------------------------------------// 7a
----vi res;--------------------------------------------------------------------// f5
----if (b % d != 0) return res;------------------------------------------------// 30
----int x0 = mod(b / d * x, n);------------------------------------------------// 48
----rep(k,0,d) res.push_back(mod(x0 + k * n / d, n));--------------------------// 7e
----return res;----------------------------------------------------------------// fe
}------------------------------------------------------------------------------// c0
```

5.10. Numeric Integration.

```
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);-----------------// 0c
}------------------------------------------------------------------------------// 4b
```

5.11. 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;----------------------------------------------------------------// 11
----}--------------------------------------------------------------------------// d0
----for (int mx = 1; mx < n; mx <<= 1) {---------------------------------------// 15
--------cpx wp = exp(cpx(0, (inv ? -1 : 1) * pi / mx)), w = 1;-----------------// 79
--------for (int m = 0; m < mx; m++, w *= wp) {--------------------------------// dc
------------for (int i = m; i < n; i += mx << 1) {-----------------------------// 6a
----------------cpx t = x[i + mx] * w;-----------------------------------------// 12
----------------x[i + mx] = x[i] - t;------------------------------------------// 73
----------------x[i] += t;-----------------------------------------------------// 0e
------------}------------------------------------------------------------------// 14
--------}----------------------------------------------------------------------// a4
----}--------------------------------------------------------------------------// bf
----if (inv) rep(i,0,n) x[i] /= cpx(n);----------------------------------------// 16
}------------------------------------------------------------------------------// 1c
void czt(cpx *x, int n, bool inv=false) {--------------------------------------// c5
----int len = 2*n+1;-----------------------------------------------------------// bc
----while (len & (len - 1)) len &= len - 1;------------------------------------// 65
----len <<= 1;-----------------------------------------------------------------// 21
----cpx w = exp(-2.0L * pi / n * cpx(0,1)),------------------------------------// 45
--------*c = new cpx[n], *a = new cpx[len],------------------------------------// 4e
--------*b = new cpx[len];-----------------------------------------------------// 30
----rep(i,0,n) c[i] = pow(w, (inv ? -1.0 : 1.0)*i*i/2);------------------------// 9e
----rep(i,0,n) a[i] = x[i] * c[i], b[i] = 1.0L/c[i];---------------------------// e9
----rep(i,0,n-1) b[len - n + i + 1] = 1.0L/c[n-i-1];---------------------------// 9f
----fft(a, len); fft(b, len);--------------------------------------------------// 63
----rep(i,0,len) a[i] *= b[i];-------------------------------------------------// 58
----fft(a, len, true);---------------------------------------------------------// 2d
----rep(i,0,n) {---------------------------------------------------------------// ff
--------x[i] = c[i] * a[i];----------------------------------------------------// 77
--------if (inv) x[i] /= cpx(n);-----------------------------------------------// b1
----}--------------------------------------------------------------------------// 27
----delete[] a;----------------------------------------------------------------// 0a
----delete[] b;----------------------------------------------------------------// 5c
----delete[] c;----------------------------------------------------------------// f8
}------------------------------------------------------------------------------// c6
```

5.12. Formulas.

- Number of ways to choose k objects from a total of n objects where order matters and each item can only be chosen once: $P_k^n = \frac{n!}{(n-k)!}$
- Number of ways to choose k objects from a total of n objects where order matters and each item can be chosen multiple times: n^k
- Number of permutations of n objects, where there are n_1 objects of type 1, n_2 objects of type 2, …, n_k objects of type k: $\binom{n}{n_1,n_2,\ldots,n_k} = \frac{n!}{n_1!\times n_2!\times\cdots\times n_k!}$
- Number of ways to choose k objects from a total of n objects where order does not matter and each item can only be chosen once: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{n-k} = \prod_{i=1}^{k}\frac{n-(k-i)}{i} = \frac{n!}{k!(n-k)!}, \binom{n}{0} = 1, \binom{0}{k} = 0$
- Number of ways to choose k objects from a total of n objects where order does not matter and each item can be chosen multiple times: $f_k^n = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!}$
- Number of integer solutions to $x_1 + x_2 + \cdots + x_n = k$ where $x_i \geq 0$: f_k^n
- Number of subsets of a set with n elements: 2^n
- $|A \cup B| = |A| + |B| - |A \cap B|$
- $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$
- Number of ways to walk from the lower-left corner to the upper-right corner of an $n \times m$ grid by walking only up and to the right: $\binom{n+m}{m}$
- Number of strings with n sets of brackets such that the brackets are balanced: $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$
- Number of triangulations of a convex polygon with n points, number of rooted binary trees with $n+1$ vertices, number of paths across an $n \times n$ lattice which do not rise above the main diagonal: C_n
- Number of permutations of n objects with exactly k ascending sequences or *runs*: $\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-k-1}\right\rangle = k\left\langle{n-1 \atop k}\right\rangle + (n-k+1)\left\langle{n-1 \atop k-1}\right\rangle = \sum_{i=0}^{k}(-1)^i\binom{n+1}{i}(k+1-i)^n, \left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$
- Number of permutations of n objects with exactly k cycles: $\left[{n \atop k}\right] = \left[{n-1 \atop k-1}\right] + (n-1)\left[{n-1 \atop k}\right]$
- Number of ways to partition n objects into k sets: $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}, \left\{{n \atop 0}\right\} = \left\{{n \atop n}\right\} = 1$
- Number of permutations of length n that have no fixed points (derangements): $D_0 = 1, D_1 = 0, D_n = (n-1)(D_{n-1} + D_{n-2})$
- Number of permutations of length n that have exactly k fixed points: $\binom{n}{k} D_{n-k}$
- **Jacobi symbol:** $\left(\frac{a}{b}\right) = a^{(b-1)/2} \pmod{b}$
- **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 containing i lattice points and having b lattice points on the boundary has area $i + \frac{b}{2} - 1$.
- **Divisor sigma:** The sum of divisors of n to the xth power is $\sigma_x(n) = \prod_{i=0}^{r}\frac{p_i^{(a_i+1)x}-1}{p_i^x - 1}$ where $n = \prod_{i=0}^{r} p_i^{a_i}$ is the prime factorization.
- **Divisor count:** A special case of the above is $\sigma_0(n) = \prod_{i=0}^{r}(a_i + 1)$.
- **Euler's totient:** The number of integers less than n that are comprime 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, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.
- The number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.
- $\gcd(2^a - 1, 2^b - 1) = 2^{\gcd(a,b)} - 1$

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

6. Geometry

6.1. Primitives.

```
#include <complex>-------------------------------------------------------------// 8e
#define P(p) const point &p----------------------------------------------------// b8
#define L(p0, p1) P(p0), P(p1)------------------------------------------------// 30
#define C(p0, r) P(p0), double r----------------------------------------------// 08
#define PP(pp) pair<point,point> &pp-------------------------------------------// a1
typedef complex<double> point;-------------------------------------------------// 9e
double dot(P(a), P(b)) { return real(conj(a) * b); }---------------------------// 4a
double cross(P(a), P(b)) { return imag(conj(a) * b); }-------------------------// f3
point rotate(P(p), double radians = pi / 2, P(about) = point(0,0)) { ----------// 0b
----return (p - about) * exp(point(0, radians)) + about; }---------------------// f5
point reflect(P(p), L(about1, about2)) {---------------------------------------// 45
----point z = p - about1, w = about2 - about1;---------------------------------// 74
----return conj(z / w) * w + about1; }-----------------------------------------// d1
point proj(P(u), P(v)) { return dot(u, v) / dot(u, u) * u; }-------------------// 98
point normalize(P(p), double k = 1.0) { ---------------------------------------// a9
----return abs(p) == 0 ? point(0,0) : p / abs(p) * k; } //TODO: TEST-----------// 1c
bool parallel(L(a, b), L(p, q)) { return abs(cross(b - a, q - p)) < EPS; }-----// 74
double ccw(P(a), P(b), P(c)) { return cross(b - a, c - b); }-------------------// ab
bool collinear(P(a), P(b), P(c)) { return abs(ccw(a, b, c)) < EPS; }-----------// 95
bool collinear(L(a, b), L(p, q)) {---------------------------------------------// de
----return abs(ccw(a, b, p)) < EPS && abs(ccw(a, b, q)) < EPS;  }--------------// 27
double angle(P(a), P(b), P(c)) {-----------------------------------------------// 93
----return acos(dot(b - a, c - b) / abs(b - a) / abs(c - b)); }----------------// a2
double signed_angle(P(a), P(b), P(c)) {----------------------------------------// 46
----return asin(cross(b - a, c - b) / abs(b - a) / abs(c - b)); }--------------// 80
double angle(P(p)) { return atan2(imag(p), real(p)); }-------------------------// c0
point perp(P(p)) { return point(-imag(p), real(p)); }--------------------------// 3c
double progress(P(p), L(a, b)) {-----------------------------------------------// c7
----if (abs(real(a) - real(b)) < EPS)------------------------------------------// 7d
--------return (imag(p) - imag(a)) / (imag(b) - imag(a));----------------------// b7
----else return (real(p) - real(a)) / (real(b) - real(a)); }-------------------// 6c
bool intersect(L(a, b), L(p, q), point &res, bool segment = false) {-----------// b4
----// NOTE: check for parallel/collinear lines before calling this function---// 88
----point r = b - a, s = q - p;------------------------------------------------// 54
----double c = cross(r, s), t = cross(p - a, s) / c, u = cross(p - a, r) / c;--// 29
----if (segment && (t < 0-EPS || t > 1+EPS || u < 0-EPS || u > 1+EPS))---------// 30
--------return false;----------------------------------------------------------// c0
----res = a + t * r;-----------------------------------------------------------// 88
----return true;---------------------------------------------------------------// 03
}------------------------------------------------------------------------------// 92
point closest_point(L(a, b), P(c), bool segment = false) {---------------------// 06
----if (segment) {-------------------------------------------------------------// 90
--------if (dot(b - a, c - b) > 0) return b;-----------------------------------// 93
--------if (dot(a - b, c - a) > 0) return a;-----------------------------------// bb
----}--------------------------------------------------------------------------// d5
----double t = dot(c - a, b - a) / norm(b - a);--------------------------------// 61
----return a + t * (b - a);----------------------------------------------------// 4f
}------------------------------------------------------------------------------// 19
```

```
double line_segment_distance(L(a,b), L(c,d)) {----------------------------------// f6
----double x = INFINITY;--------------------------------------------------------// 8c
----if (abs(a - b) < EPS && abs(c - d) < EPS) x = abs(a - c);-------------------// 5f
----else if (abs(a - b) < EPS) x = abs(a - closest_point(c, d, a, true));------// 97
----else if (abs(c - d) < EPS) x = abs(c - closest_point(a, b, c, true));------// 68
----else if ((ccw(a, b, c) < 0) != (ccw(a, b, d) < 0) &&----------------------// fa
------------ (ccw(c, d, a) < 0) != (ccw(c, d, b) < 0)) x = 0;-------------------// bb
----else {----------------------------------------------------------------------// 5b
--------x = min(x, abs(a - closest_point(c,d, a, true)));----------------------// 07
--------x = min(x, abs(b - closest_point(c,d, b, true)));----------------------// 75
--------x = min(x, abs(c - closest_point(a,b, c, true)));----------------------// 48
--------x = min(x, abs(d - closest_point(a,b, d, true)));----------------------// 75
----}---------------------------------------------------------------------------// 60
----return x;-------------------------------------------------------------------// 57
}-------------------------------------------------------------------------------// 8e
int intersect(C(A, rA), C(B, rB), point & res1, point & res2) {-----------------// ca
----double d = abs(B - A);------------------------------------------------------// 06
----if ((rA + rB) <  (d - EPS) || d < abs(rA - rB) - EPS) return 0;------------// 5d
----double a = (rA*rA - rB*rB + d*d) / 2 / d, h = sqrt(rA*rA - a*a);-----------// 5e
----point v = normalize(B - A, a), u = normalize(rotate(B-A), h);--------------// da
----res1 = A + v + u, res2 = A + v - u;-----------------------------------------// c2
----if (abs(u) < EPS) return 1; return 2;---------------------------------------// 95
}-------------------------------------------------------------------------------// 4e
int intersect(L(A, B), C(O, r), point & res1, point & res2) {-------------------// e4
---- double h = abs(O - closest_point(A, B, O));--------------------------------// f4
---- if(r < h - EPS) return 0;--------------------------------------------------// 89
---- point H = proj(O - A, B - A) + A, v = normalize((B - A), sqrt(r*r - h*h));// a9
---- res1 = H + v; res2 = H - v;------------------------------------------------// ab
---- if(abs(v) < EPS) return 1; return 2;---------------------------------------// f7
 }------------------------------------------------------------------------------// b8
int tangent(P(A), C(O, r), point & res1, point & res2) {------------------------// 15
----point v = O - A; double d = abs(v);-----------------------------------------// 2c
----if (d < r - EPS) return 0;--------------------------------------------------// 14
----double alpha = asin(r / d), L = sqrt(d*d - r*r);---------------------------// 45
----v = normalize(v, L);--------------------------------------------------------// 10
----res1 = A + rotate(v, alpha); res2 = A + rotate(v, -alpha);-----------------// 56
----if (abs(r - d) < EPS || abs(v) < EPS) return 1;----------------------------// 1d
----return 2;-------------------------------------------------------------------// 97
}-------------------------------------------------------------------------------// 46
void tangent_outer(point A, double rA, point B, double rB, PP(P), PP(Q)) {-----// 61
----if (rA - rB > EPS) { swap(rA, rB); swap(A, B); }---------------------------// 2a
----double theta = asin((rB - rA)/abs(A - B));---------------------------------// 0a
----point v = rotate(B - A, theta + pi/2), u = rotate(B - A, -(theta + pi/2));-// e3
----u = normalize(u, rA);-------------------------------------------------------// 30
----P.first = A + normalize(v, rA); P.second = B + normalize(v, rB);-----------// 08
----Q.first = A + normalize(u, rA); Q.second = B + normalize(u, rB);-----------// 2a
}-------------------------------------------------------------------------------// 2d
```

6.2. Polygon.

```
#include "primitives.cpp"-------------------------------------------------------// e0
typedef vector<point> polygon;--------------------------------------------------// b3
double polygon_area_signed(polygon p) {-----------------------------------------// 31
----double area = 0; int cnt = size(p);-----------------------------------------// a2
----rep(i,1,cnt-1) area += cross(p[i] - p[0], p[i + 1] - p[0]);----------------// 51
----return area / 2; }----------------------------------------------------------// 66
double polygon_area(polygon p) { return abs(polygon_area_signed(p)); }---------// a4
#define CHK(f,a,b,c) (f(a) < f(b) && f(b) <= f(c) && ccw(a,c,b) < 0)----------// 8f
int point_in_polygon(polygon p, point q) {--------------------------------------// 5d
----int n = size(p); bool in = false; double d;---------------------------------// 69
----for (int i = 0, j = n - 1; i < n; j = i++)----------------------------------// f3
--------if (collinear(p[i], q, p[j]) &&-----------------------------------------// 9d
------------0 <= (d = progress(q, p[i], p[j])) && d <= 1)-----------------------// 4b
------------return 0;-----------------------------------------------------------// b3
----for (int i = 0, j = n - 1; i < n; j = i++)----------------------------------// 67
--------if (CHK(real, p[i], q, p[j]) || CHK(real, p[j], q, p[i]))--------------// b4
------------in = !in;-----------------------------------------------------------// ff
----return in ? -1 : 1; }-------------------------------------------------------// ba
// pair<polygon, polygon> cut_polygon(const polygon &poly, point a, point b) {-// 0d
//---- polygon left, right;-----------------------------------------------------// 0a
//---- point it(-100, -100);----------------------------------------------------// 5b
//---- for (int i = 0, cnt = poly.size(); i < cnt; i++) {-----------------------// 70
//-------- int j = i == cnt-1 ? 0 : i + 1;--------------------------------------// 02
//-------- point p = poly[i], q = poly[j];--------------------------------------// 44
//-------- if (ccw(a, b, p) <= 0) left.push_back(p);---------------------------// 8d
//-------- if (ccw(a, b, p) >= 0) right.push_back(p);--------------------------// 43
//-------- // myintersect = intersect where-------------------------------------// ba
//-------- // (a,b) is a line, (p,q) is a line segment--------------------------// 7e
//-------- if (myintersect(a, b, p, q, it))-------------------------------------// 6f
//------------ left.push_back(it), right.push_back(it);-------------------------// 8a
//---- }------------------------------------------------------------------------// e0
//---- return pair<polygon, polygon>(left, right);------------------------------// 3d
// }----------------------------------------------------------------------------// 07
```

6.3. Convex Hull.

```
#include "polygon.cpp"----------------------------------------------------------// 58
#define MAXN 1000-----------------------------------------------------------------// 09
point hull[MAXN];---------------------------------------------------------------// 43
bool cmp(const point &a, const point &b) {--------------------------------------// 32
----return abs(real(a) - real(b)) > EPS ?---------------------------------------// 44
--------real(a) < real(b) : imag(a) < imag(b); }--------------------------------// 40
int convex_hull(polygon p) {----------------------------------------------------// cd
----int n = size(p), l = 0;-----------------------------------------------------// 67
----sort(p.begin(), p.end(), cmp);----------------------------------------------// 3d
----rep(i,0,n) {----------------------------------------------------------------// e4
--------if (i > 0 && p[i] == p[i - 1]) continue;--------------------------------// c7
--------while (l >= 2 && ccw(hull[l - 2], hull[l - 1], p[i]) >= 0) l--;--------// 62
--------hull[l++] = p[i];-------------------------------------------------------// bd
----}---------------------------------------------------------------------------// d2
----int r = l;------------------------------------------------------------------// 30
----for (int i = n - 2; i >= 0; i--) {------------------------------------------// 59
--------if (p[i] == p[i + 1]) continue;-----------------------------------------// af
--------while (r - l >= 1 && ccw(hull[r - 2], hull[r - 1], p[i]) >= 0) r--;----// 4d
```

```
--------hull[r++] = p[i];-----------------------------------------------------// f5
----}-------------------------------------------------------------------------// f6
----return l == 1 ? 1 : r - 1;------------------------------------------------// a6
}-----------------------------------------------------------------------------// 6d
```

6.4. **Line Segment Intersection.**

```
#include "primitives.cpp"-----------------------------------------------------// e0
bool line_segment_intersect(L(a, b), L(c, d), point &A, point &B) {-----------// 6c
----if (abs(a - b) < EPS && abs(c - d) < EPS) {-------------------------------// db
--------A = B = a; return abs(a - d) < EPS; }---------------------------------// ee
----else if (abs(a - b) < EPS) {----------------------------------------------// 03
--------A = B = a; double p = progress(a, c,d);-------------------------------// c9
--------return 0.0 <= p && p <= 1.0-------------------------------------------// 8a
------------&& (abs(a - c) + abs(d - a) - abs(d - c)) < EPS; }----------------// 27
----else if (abs(c - d) < EPS) {----------------------------------------------// 26
--------A = B = c; double p = progress(c, a,b);-------------------------------// d9
--------return 0.0 <= p && p <= 1.0-------------------------------------------// 8e
------------&& (abs(c - a) + abs(b - c) - abs(b - a)) < EPS; }----------------// 4f
----else if (collinear(a,b, c,d)) {-------------------------------------------// bc
--------double ap = progress(a, c,d), bp = progress(b, c,d);------------------// a7
--------if (ap > bp) swap(ap, bp);--------------------------------------------// b1
--------if (bp < 0.0 || ap > 1.0) return false;-------------------------------// 0c
--------A = c + max(ap, 0.0) * (d - c);---------------------------------------// f6
--------B = c + min(bp, 1.0) * (d - c);---------------------------------------// 5c
--------return true; }--------------------------------------------------------// ab
----else if (parallel(a,b, c,d)) return false;--------------------------------// ca
----else if (intersect(a,b, c,d, A, true)) {----------------------------------// 10
--------B = A; return true; }-------------------------------------------------// bf
----return false;-------------------------------------------------------------// b7
}-----------------------------------------------------------------------------// 8b
------------------------------------------------------------------------------// e6
```

6.5. **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) +--------------// e3
--------------------sin(pLat) * sin(qLat));-----------------------------------// 1e
------------------------------------------------------------------------------// 60
}-----------------------------------------------------------------------------// 3f
```

6.6. **Triangle Circumcenter.** Returns the unique point that is the same distance from all three points. It is also the center of the unique circle that goes through all three points.

```
#include "primitives.cpp"-----------------------------------------------------// e0
point circumcenter(point a, point b, point c) {-------------------------------// 76
----b -= a, c -= a;-----------------------------------------------------------// 41
----return a + perp(b * norm(c) - c * norm(b)) / 2.0 / cross(b, c);-----------// 7a
}-----------------------------------------------------------------------------// c3
```

6.7. **Closest Pair of Points.**

```
#include "primitives.cpp"-----------------------------------------------------// e0
------------------------------------------------------------------------------// 85
struct cmpx { bool operator ()(const point &a, const point &b) {--------------// 01
--------return abs(real(a) - real(b)) > EPS ?---------------------------------// e9
------------real(a) < real(b) : imag(a) < imag(b); } };-----------------------// 53
struct cmpy { bool operator ()(const point &a, const point &b) {--------------// 6f
----return abs(imag(a) - imag(b)) > EPS ?-------------------------------------// 0b
------------imag(a) < imag(b) : real(a) < real(b); } };-----------------------// a4
double closest_pair(vector<point> pts) {--------------------------------------// f1
----sort(pts.begin(), pts.end(), cmpx());-------------------------------------// 0c
----set<point, cmpy> cur;-----------------------------------------------------// bd
----set<point, cmpy>::const_iterator it, jt;----------------------------------// a6
----double mn = INFINITY;-----------------------------------------------------// f9
----for (int i = 0, l = 0; i < size(pts); i++) {------------------------------// ac
--------while (real(pts[i]) - real(pts[l]) > mn) cur.erase(pts[l++]);--------// 8b
--------it = cur.lower_bound(point(-INFINITY, imag(pts[i]) - mn));------------// fc
--------jt = cur.upper_bound(point(INFINITY, imag(pts[i]) + mn));-------------// 39
--------while (it != jt) mn = min(mn, abs(*it - pts[i])), it++;---------------// 09
--------cur.insert(pts[i]); }-------------------------------------------------// 82
----return mn; }--------------------------------------------------------------// 4c
```

6.8. **3D 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) : x(_x), y(_y), z(_z) {}---------// fc
----point3d operator+(P(p)) const {-------------------------------------------// 17
--------return point3d(x + p.x, y + p.y, z + p.z); }--------------------------// 8e
----point3d operator-(P(p)) const {-------------------------------------------// fb
--------return point3d(x - p.x, y - p.y, z - p.z); }--------------------------// 83
----point3d operator-() const {-----------------------------------------------// 89
--------return point3d(-x, -y, -z); }-----------------------------------------// d4
----point3d operator*(double k) const {---------------------------------------// 4d
--------return point3d(x * k, y * k, z * k); }--------------------------------// fd
----point3d operator/(double k) const {---------------------------------------// 95
--------return point3d(x / k, y / k, z / k); }--------------------------------// 58
----double operator%(P(p)) const {--------------------------------------------// d1
--------return x * p.x + y * p.y + z * p.z; }---------------------------------// 09
----point3d operator*(P(p)) const {-------------------------------------------// 4f
--------return point3d(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x); }--------// ed
----double length() const {---------------------------------------------------// 3e
--------return sqrt(*this % *this); }-----------------------------------------// 05
----double distTo(P(p)) const {-----------------------------------------------// dd
--------return (*this - p).length(); }----------------------------------------// 57
----double distTo(P(A), P(B)) const {-----------------------------------------// bd
--------// A and B must be two different points-------------------------------// 4e
--------return ((*this - A) * (*this - B)).length() / A.distTo(B); }----------// 6e
```

```
----point3d normalize(double k = 1) const {---------------------------------// db
--------// length() must not return 0---------------------------------------// 3c
--------return (*this) * (k / length()); }----------------------------------// d4
----point3d getProjection(P(A), P(B)) const {-------------------------------// 86
--------point3d v = B - A;--------------------------------------------------// 64
--------return A + v.normalize((v % (*this - A)) / v.length()); }-----------// 53
----point3d rotate(P(normal)) const {---------------------------------------// 55
--------// normal must have length 1 and be orthogonal to the vector--------// eb
----   return (*this) * normal; }-------------------------------------------// 5c
----point3d rotate(double alpha, P(normal)) const {-------------------------// 21
--------return (*this) * cos(alpha) + rotate(normal) * sin(alpha); }--------// 82
----point3d rotatePoint(P(O), P(axe), double alpha) const{------------------// 7a
--------point3d Z = axe.normalize(axe % (*this - O));-----------------------// ba
--------return O + Z + (*this - O - Z).rotate(alpha, O); }------------------// 38
----bool isZero() const {---------------------------------------------------// 64
--------return abs(x) < EPS && abs(y) < EPS && abs(z) < EPS; }--------------// 15
----bool isOnLine(L(A, B)) const {------------------------------------------// 30
--------return ((A - *this) * (B - *this)).isZero(); }----------------------// 58
----bool isInSegment(L(A, B)) const {---------------------------------------// f1
--------return isOnLine(A, B) && ((A - *this) % (B - *this)) < EPS; }-------// d9
----bool isInSegmentStrictly(L(A, B)) const {-------------------------------// 0e
--------return isOnLine(A, B) && ((A - *this) % (B - *this)) < -EPS; }------// ba
----double getAngle() const {-----------------------------------------------// 0f
--------return atan2(y, x); }-----------------------------------------------// 40
----double getAngle(P(u)) const {-------------------------------------------// d5
--------return atan2((*this * u).length(), *this % u); }--------------------// 79
----bool isOnPlane(PL(A, B, C)) const {-------------------------------------// 8e
--------return abs((A - *this) * (B - *this) % (C - *this)) < EPS; } };-----// 74
int line_line_intersect(L(A, B), L(C, D), point3d &O){----------------------// dc
----if (abs((B - A) * (C - A) % (D - A)) > EPS) return 0;-------------------// 6a
----if (((A - B) * (C - D)).length() < EPS)---------------------------------// 79
--------return A.isOnLine(C, D) ? 2 : 0;------------------------------------// 09
----point3d normal = ((A - B) * (C - B)).normalize();-----------------------// bc
----double s1 = (C - A) * (D - A) % normal;---------------------------------// 68
----O = A + ((B - A) / (s1 + ((D - B) * (C - B) % normal))) * s1;-----------// 56
----return 1; }-------------------------------------------------------------// a7
int line_plane_intersect(L(A, B), PL(C, D, E), point3d & O) {---------------// 09
----double V1 = (C - A) * (D - A) % (E - A);--------------------------------// c1
----double V2 = (D - B) * (C - B) % (E - B);--------------------------------// 29
----if (abs(V1 + V2) < EPS)-------------------------------------------------// 81
--------return A.isOnPlane(C, D, E) ? 2 : 0;--------------------------------// d5
----O = A + ((B - A) / (V1 + V2)) * V1;-------------------------------------// 38
----return 1; }-------------------------------------------------------------// ce
bool plane_plane_intersect(P(A), P(nA), P(B), P(nB), point3d &P, point3d &Q) {-// 5a
----point3d n = nA * nB;----------------------------------------------------// 49
----if (n.isZero()) return false;-------------------------------------------// 03
----point3d v = n * nA;-----------------------------------------------------// d7
----P = A + (n * nA) * ((B - A) % nB / (v % nB));---------------------------// 1a
----Q = P + n;--------------------------------------------------------------// 9c
----return true; }----------------------------------------------------------// 1a
```

6.9. **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) + point(mnx, mny); }---// a1
```

6.10. **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.

7. Other Algorithms

7.1. **2SAT.**

```
#include "../graph/scc.cpp"-------------------------------------------------// c3
----------------------------------------------------------------------------// 63
bool two_sat(int n, const vii& clauses, vi& all_truthy) {-------------------// f4
----all_truthy.clear();-----------------------------------------------------// 31
----vvi adj(2*n+1);---------------------------------------------------------// 7b
----rep(i,0,size(clauses)) {------------------------------------------------// 76
--------adj[-clauses[i].first + n].push_back(clauses[i].second + n);--------// eb
--------if (clauses[i].first != clauses[i].second)--------------------------// bc
------------adj[-clauses[i].second + n].push_back(clauses[i].first + n);----// f0
----}-----------------------------------------------------------------------// da
----pair<union_find, vi> res = scc(adj);------------------------------------// 00
----union_find scc = res.first;---------------------------------------------// 20
----vi dag = res.second;----------------------------------------------------// ed
----vi truth(2*n+1, -1);----------------------------------------------------// c7
----for (int i = 2*n; i >= 0; i--) {----------------------------------------// 50
--------int cur = order[i] - n, p = scc.find(cur + n), o = scc.find(-cur + n);-// 4f
--------if (cur == 0) continue;---------------------------------------------// cd
--------if (p == o) return false;-------------------------------------------// d0
--------if (truth[p] == -1) truth[p] = 1;-----------------------------------// d3
--------truth[cur + n] = truth[p];------------------------------------------// 50
--------truth[o] = 1 - truth[p];--------------------------------------------// 8c
--------if (truth[p] == 1) all_truthy.push_back(cur);-----------------------// 55
----}-----------------------------------------------------------------------// c3
----return true;------------------------------------------------------------// eb
}---------------------------------------------------------------------------// 6b
```

7.2. Stable Marriage.

```
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;-------------------------// a1
--------}---------------------------------------------------------------------// c4
----}-------------------------------------------------------------------------// 3d
----return res;---------------------------------------------------------------// 42
}-----------------------------------------------------------------------------// bf
```

7.3. Algorithm X.

```
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; }-----------------------------// c3
----};------------------------------------------------------------------------// c1
----int rows, cols, *sol;-----------------------------------------------------// 7b
----bool **arr;---------------------------------------------------------------// e6
----node *head;---------------------------------------------------------------// fe
----exact_cover(int _rows, int _cols) : rows(_rows), cols(_cols), head(NULL) {-// b6
--------arr = new bool*[rows];------------------------------------------------// cf
--------sol = new int[rows];--------------------------------------------------// 5f
--------rep(i,0,rows)---------------------------------------------------------// 9b
------------arr[i] = new bool[cols], memset(arr[i], 0, cols);-----------------// dd
----}-------------------------------------------------------------------------// 21
----void set_value(int row, int col, bool val = true) { arr[row][col] = val; }-// 9e
----void setup() {------------------------------------------------------------// a3
--------node ***ptr = new node**[rows + 1];-----------------------------------// bd
--------rep(i,0,rows+1) {-----------------------------------------------------// 76
------------ptr[i] = new node*[cols];-----------------------------------------// eb
------------rep(j,0,cols)-----------------------------------------------------// cd
----------------if (i == rows || arr[i][j]) ptr[i][j] = new node(i, j);-------// 16
----------------else ptr[i][j] = NULL;----------------------------------------// d2
--------}---------------------------------------------------------------------// ac
--------rep(i,0,rows+1) {-----------------------------------------------------// fc
------------rep(j,0,cols) {---------------------------------------------------// 51
----------------if (!ptr[i][j]) continue;-------------------------------------// f7
----------------int ni = i + 1, nj = j + 1;-----------------------------------// 7a
----------------while (true) {------------------------------------------------// fc
--------------------if (ni == rows + 1) ni = 0;-------------------------------// 4c
--------------------if (ni == rows || arr[ni][j]) break;----------------------// 8d
--------------------++ni;-----------------------------------------------------// 68
----------------}-------------------------------------------------------------// ad
----------------ptr[i][j]->d = ptr[ni][j];------------------------------------// 84
----------------ptr[ni][j]->u = ptr[i][j];------------------------------------// 66
----------------while (true) {------------------------------------------------// 7f
--------------------if (nj == cols) nj = 0;-----------------------------------// de
--------------------if (i == rows || arr[i][nj]) break;-----------------------// 4c
--------------------++nj;-----------------------------------------------------// c5
----------------}-------------------------------------------------------------// 72
----------------ptr[i][j]->r = ptr[i][nj];------------------------------------// 60
----------------ptr[i][nj]->l = ptr[i][j];------------------------------------// 82
------------}-----------------------------------------------------------------// 0b
--------}---------------------------------------------------------------------// 16
--------head = new node(rows, -1);--------------------------------------------// 66
--------head->r = ptr[rows][0];-----------------------------------------------// 3e
--------ptr[rows][0]->l = head;-----------------------------------------------// 8c
--------head->l = ptr[rows][cols - 1];----------------------------------------// 6a
--------ptr[rows][cols - 1]->r = head;----------------------------------------// c1
--------rep(j,0,cols) {-------------------------------------------------------// 92
------------int cnt = -1;-----------------------------------------------------// d4
------------rep(i,0,rows+1)---------------------------------------------------// bd
----------------if (ptr[i][j]) cnt++, ptr[i][j]->p = ptr[rows][j];------------// f3
------------ptr[rows][j]->size = cnt;-----------------------------------------// c2
--------}---------------------------------------------------------------------// b9
--------rep(i,0,rows+1) delete[] ptr[i];--------------------------------------// a5
--------delete[] ptr;---------------------------------------------------------// 72
----}-------------------------------------------------------------------------// 19
----#define COVER(c, i, j) \------------------------------------------------// 91
--------c->r->l = c->l, c->l->r = c->r; \-------------------------------------// 82
--------for (node *i = c->d; i != c; i = i->d) \------------------------------// 62
------------for (node *j = i->r; j != i; j = j->r) \--------------------------// 26
----------------j->d->u = j->u, j->u->d = j->d, j->p->size--;-----------------// c1
----#define UNCOVER(c, i, j) \----------------------------------------------// 89
--------for (node *i = c->u; i != c; i = i->u) \------------------------------// f0
------------for (node *j = i->l; j != i; j = j->l) \--------------------------// 7b
----------------j->p->size++, j->d->u = j->u->d = j; \------------------------// 65
--------c->r->l = c->l->r = c;------------------------------------------------// 0e
----bool search(int k = 0) {--------------------------------------------------// f9
--------if (head == head->r) {------------------------------------------------// 75
------------vi res(k);--------------------------------------------------------// 90
------------rep(i,0,k) res[i] = sol[i];---------------------------------------// 2a
------------sort(res.begin(), res.end());-------------------------------------// 63
------------return handle_solution(res);--------------------------------------// 11
--------}---------------------------------------------------------------------// 3d
--------node *c = head->r, *tmp = head->r;------------------------------------// a3
--------for ( ; tmp != head; tmp = tmp->r) if (tmp->size < c->size) c = tmp;---// 41
--------if (c == c->d) return false;------------------------------------------// 02
```

```
--------COVER(c, i, j);----------------------------------------------// f6
--------bool found = false;------------------------------------------// 8d
--------for (node *r = c->d; !found && r != c; r = r->d) {-----------// 78
------------sol[k] = r->row;-----------------------------------------// c0
------------for (node *j = r->r; j != r; j = j->r) { COVER(j->p, a, b); }-----// f9
------------found = search(k + 1);-----------------------------------// fb
------------for (node *j = r->l; j != r; j = j->l) { UNCOVER(j->p, a, b); }---// 87
--------}------------------------------------------------------------// 7c
--------UNCOVER(c, i, j);--------------------------------------------// a7
--------return found;------------------------------------------------// c0
----}----------------------------------------------------------------// d2
};-------------------------------------------------------------------// d7
```

7.4. nth Permutation.

```
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]], idx.erase(idx.begin() + fac[i]);-------// ee
----return per;------------------------------------------------------// ab
}--------------------------------------------------------------------// 37
```

7.5. Cycle-Finding.

```
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);----------------------------------------------// b4
}--------------------------------------------------------------------// 42
```

7.6. Dates.

```
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;---------------------------------------------------// e0
}--------------------------------------------------------------------// fa
void intToDate(int jd, int &y, int &m, int &d) {---------------------// a1
----int x, n, i, j;--------------------------------------------------// 00
----x = jd + 68569;--------------------------------------------------// 11
----n = 4 * x / 146097;----------------------------------------------// 2f
----x -= (146097 * n + 3) / 4;---------------------------------------// 58
----i = (4000 * (x + 1)) / 1461001;----------------------------------// 0d
----x -= 1461 * i / 4 - 31;------------------------------------------// 09
----j = 80 * x / 2447;-----------------------------------------------// 3d
----d = x - 2447 * j / 80;-------------------------------------------// eb
----x = j / 11;------------------------------------------------------// b7
----m = j + 2 - 12 * x;----------------------------------------------// 82
----y = 100 * (n - 49) + i + x;--------------------------------------// 70
}--------------------------------------------------------------------// af
```

8. Useful Information

8.1. Tips & Tricks.

- How fast does our algorithm have to be? Can we use brute-force?
- Does order matter?
- Is it better to look at the problem in another way? Maybe backwards?
- Are there subproblems that are recomputed? Can we cache them?
- Do we need to remember everything we compute, or just the last few iterations of computation?
- Does it help to sort the data?
- Can we speed up lookup by using a map (tree or hash) or an array?
- Can we binary search the answer?
- Can we add vertices/edges to the graph to make the problem easier? Can we turn the graph into some other kind of a graph (perhaps a DAG, or a flow network)?
- Make sure integers are not overflowing.
- Is it better to compute the answer modulo n? Perhaps we can compute the answer modulo $m_1, m_2, \ldots, m_k$, where $m_1, m_2, \ldots, m_k$ are pairwise coprime integers, and find the real answer using CRT?
- Are there any edge cases? When $n = 0, n = -1, n = 1, n = 2^{31} - 1$ or $n = -2^{31}$? When the list is empty, or contains a single element? When the graph is empty, or contains a single vertex? When the graph contains self-loops? When the polygon is concave or non-simple?
- Can we use exponentiation by squaring?

8.2. Fast Input Reading.

```
void readn(register int *n) {----------------------------------------// dc
----int sign = 1;----------------------------------------------------// 32
----register char c;-------------------------------------------------// a5
----*n = 0;----------------------------------------------------------// 35
----while((c = getc_unlocked(stdin)) != '\n') {----------------------// f3
--------switch(c) {--------------------------------------------------// 0c
------------case '-': sign = -1; break;------------------------------// 28
------------case ' ': goto hell;-------------------------------------// fd
------------case '\n': goto hell;------------------------------------// 79
------------default: *n *= 10; *n += c - '0'; break;-----------------// c0
--------}------------------------------------------------------------// 2d
----}----------------------------------------------------------------// c3
hell:----------------------------------------------------------------// ba
----*n *= sign;------------------------------------------------------// a0
}--------------------------------------------------------------------// 67
```

8.3. Bit Hacks.

- `n & -n` returns the first set bit in n.
- `n & (n - 1)` is 0 only if n is a power of two.
- `snoob(x)` returns the next integer that has the same amount of bits set as `x`. Useful for iterating through subsets of some specified size.

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