viRUs

Team Reference Document

25/11/2015

Contents

1. Code Templates

1.1. Basic Configuration.

1.1.1. .bashrc.

```
xset r rate 150 100 // dd
set -o vi // f4
// 08
function check { // f6
----IFS= // 17
----s="" // 22
----cat $1 | while read l; do // 81
--------s="$s$(echo $l | sed 's/\s//g')\n" // d0
--------h=$(echo -ne "$s" | md5sum) // 27
--------echo "${h:0:2}  $l" // 1b
----done // aa
} // a7
# setxkbmap -option caps:escape dvorak is // 18
# setxkbmap en_US // 97
```

ProTip™: setxkbmap dvorak on qwerty: `o.yqtxmal ekrpat`

1.1.2. .vimrc.

```
set nocp et sw=4 ts=4 sts=4 si cindent hi=1000 nu ru noeb showcmd showmode // bb
syn on | colorscheme slate // e5
```

1.2. C++ Header.

A C++ header.

```cpp
#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 = ~(1<<31); // 2147483647 // 10
// b2
const double EPS = 1e-9; // d5
const double pi = acos(-1); // 67
typedef unsigned long long ull; // ff
typedef vector<vi> vvi; // 4b
typedef vector<vii> vvii; // 36
template <class T> T mod(T a, T b) { return (a % b + b) % b; } // d5
```

1.3. Java Template.

A Java template.

```java
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.

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

2.2. Segment Tree.

```cpp
#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 (a <= l && r <= b) return (lazy[i] = v) * (r - l + 1) + data[i];----// 06
--------int m = (l + r) / 2;-------------------------------------------------// cc
--------return data[i] = f(ru(a, b, v, l, m, 2*i+1),-------------------------// cc
------------------------   ru(a, b, v, m+1, r, 2*i+2));----------------------// 2b
----}------------------------------------------------------------------------// 0b
----void propagate(int l, int r, int i) {------------------------------------// a7
--------if (l > r || lazy[i] == INF) return;---------------------------------// 5f
--------data[i] += lazy[i] * (r - l + 1);------------------------------------// 44
--------if (l < r) {---------------------------------------------------------// 28
------------if (lazy[2*i+1] == INF) lazy[2*i+1] = lazy[i];-------------------// 4e
------------else lazy[2*i+1] += lazy[i];-------------------------------------// 1e
------------if (lazy[2*i+2] == INF) lazy[2*i+2] = lazy[i];-------------------// de
------------else lazy[2*i+2] += lazy[i];-------------------------------------// 74
--------}--------------------------------------------------------------------// 1f
--------lazy[i] = INF;-------------------------------------------------------// f8
----}------------------------------------------------------------------------// 41
};---------------------------------------------------------------------------// ae
```

2.2.1. *Persistent Segment Tree.*

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

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> struct matrix {-------------------------------------------// 0a
----int rows, cols, cnt; vector<T> data;-------------------------------------// a1
----inline T& at(int i, int j) { return data[i * cols + j]; }----------------// 5c
----matrix(int r, int c) : rows(r), cols(c), cnt(r * c) {--------------------// 56
--------data.assign(cnt, T(0)); }--------------------------------------------// e3
----matrix(const matrix& other) : rows(other.rows), cols(other.cols),--------// b5
--------cnt(other.cnt), data(other.data) { }---------------------------------// c1
----T& operator()(int i, int j) { return at(i, j); }-------------------------// 29
----matrix<T> operator +(const matrix& other) {------------------------------// 33
--------matrix<T> res(*this); rep(i,0,cnt) res.data[i] += other.data[i];------// f8
--------return res; }--------------------------------------------------------// 09
----matrix<T> operator -(const matrix& other) {------------------------------// 91
--------matrix<T> res(*this); rep(i,0,cnt) res.data[i] -= other.data[i];------// 7b
--------return res; }--------------------------------------------------------// 9a
----matrix<T> operator *(T other) {------------------------------------------// 99
--------matrix<T> res(*this); rep(i,0,cnt) res.data[i] *= other;-------------// 05
--------return res; }--------------------------------------------------------// 8c
----matrix<T> operator *(const matrix& other) {------------------------------// 31
--------matrix<T> res(rows, other.cols);-------------------------------------// 4c
--------rep(i,0,rows) rep(j,0,other.cols) rep(k,0,cols)----------------------// ae
----------------res(i, j) += at(i, k) * other.data[k * other.cols + j];------// 17
```

```
--------return res; }------------------------------------------------------// 65
----matrix<T> pow(int p) {-------------------------------------------------// 53
--------matrix<T> res(rows, cols), sq(*this);------------------------------// 87
--------rep(i,0,rows) res(i, i) = T(1);------------------------------------// 9d
--------while (p) {--------------------------------------------------------// 79
------------if (p & 1) res = res * sq;-------------------------------------// 62
------------p >>= 1;-------------------------------------------------------// 79
------------if (p) sq = sq * sq;-------------------------------------------// 35
--------} return res; }----------------------------------------------------// 22
----matrix<T> rref(T &det, int &rank) {------------------------------------// 2a
--------matrix<T> mat(*this); det = T(1), rank = max(rows, cols);----------// 7a
--------for (int r = 0, c = 0; c < cols; c++) {----------------------------// 8e
------------int k = r;-----------------------------------------------------// 5b
------------while (k < rows && eq<T>(mat(k, c), T(0))) k++;----------------// 3e
------------if (k >= rows) { rank--; continue; }---------------------------// 1a
------------if (k != r) {--------------------------------------------------// c4
----------------det *= T(-1);----------------------------------------------// 55
----------------rep(i,0,cols)----------------------------------------------// e1
--------------------swap(mat.at(k, i), mat.at(r, i));----------------------// 7d
------------} det *= mat(r, r);--------------------------------------------// b6
------------T d = mat(r,c);------------------------------------------------// 66
------------rep(i,0,cols) mat(r, i) /= d;----------------------------------// d1
------------rep(i,0,rows) {------------------------------------------------// f6
----------------T m = mat(i, c);-------------------------------------------// 05
----------------if (i != r && !eq<T>(m, T(0)))-----------------------------// 1a
--------------------rep(j,0,cols) mat(i, j) -= m * mat(r, j);--------------// 7b
------------} r++;---------------------------------------------------------// c5
--------} return mat; }---------------------------------------------------// b3
----matrix<T> transpose() {------------------------------------------------// 59
--------matrix<T> res(cols, rows);-----------------------------------------// 5b
--------rep(i,0,rows) rep(j,0,cols) res(j, i) = at(i, j);------------------// 92
--------return res; } };---------------------------------------------------// df
```

2.5. Cartesian Tree.

```
struct node {--------------------------------------------------------------// 36
----int x, y, sz;----------------------------------------------------------// e5
----node *l, *r;-----------------------------------------------------------// 4d
----node(int _x, int _y) : x(_x), y(_y), sz(1), l(NULL), r(NULL) { } };----// 19
int tsize(node* t) { return t ? t->sz : 0; }-------------------------------// 42
void augment(node *t) { t->sz = 1 + tsize(t->l) + tsize(t->r); }-----------// 1d
pair<node*,node*> split(node *t, int x) {----------------------------------// 1d
----if (!t) return make_pair((node*)NULL,(node*)NULL);---------------------// fd
----if (t->x < x) {--------------------------------------------------------// 0a
--------pair<node*,node*> res = split(t->r, x);----------------------------// b4
--------t->r = res.first; augment(t);--------------------------------------// 4d
--------return make_pair(t, res.second); }---------------------------------// e0
----pair<node*,node*> res = split(t->l, x);--------------------------------// b7
----t->l = res.second; augment(t);-----------------------------------------// 74
----return make_pair(res.first, t); }--------------------------------------// 46
node* merge(node *l, node *r) {--------------------------------------------// 3c
----if (!l) return r; if (!r) return l;------------------------------------// f0
----if (l->y > r->y) { l->r = merge(l->r, r); augment(l); return l; }------// be
----r->l = merge(l, r->l); augment(r); return r; }-------------------------// c0
node* find(node *t, int x) {-----------------------------------------------// b4
----while (t) {------------------------------------------------------------// 51
--------if (x < t->x) t = t->l;--------------------------------------------// 32
--------else if (t->x < x) t = t->r;---------------------------------------// da
--------else return t; }---------------------------------------------------// 0b
----return NULL; }---------------------------------------------------------// ae
node* insert(node *t, int x, int y) {--------------------------------------// 78
----if (find(t, x) != NULL) return t;--------------------------------------// 2f
----pair<node*,node*> res = split(t, x);-----------------------------------// ca
----return merge(res.first, merge(new node(x, y), res.second)); }----------// 0d
node* erase(node *t, int x) {----------------------------------------------// 4d
----if (!t) return NULL;---------------------------------------------------// 7b
----if (t->x < x) t->r = erase(t->r, x);-----------------------------------// 7c
----else if (x < t->x) t->l = erase(t->l, x);------------------------------// 48
----else { node *old = t; t = merge(t->l, t->r); delete old; }-------------// 22
----if (t) augment(t); return t; }-----------------------------------------// a3
int kth(node *t, int k) {--------------------------------------------------// b3
----if (k < tsize(t->l)) return kth(t->l, k);------------------------------// 64
----else if (k == tsize(t->l)) return t->x;--------------------------------// 97
----else return kth(t->r, k - tsize(t->l) - 1); }--------------------------// f0
```

2.6. 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
```

2.7. 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
```

2.8. 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
```

2.9. Convex Hull Trick.

```
struct convex_hull_trick {---------------------------------------------------// 16
----vector<pair<double,double> > h;------------------------------------------// b4
----double intersect(int i) {------------------------------------------------// 9b
--------return (h[i+1].second-h[i].second)/(h[i].first-h[i+1].first); }------// b9
----void add(double m, double b) {-------------------------------------------// a4
--------h.push_back(make_pair(m,b));-----------------------------------------// f9
--------while (size(h) >= 3) {-----------------------------------------------// f6
------------int n = size(h);-------------------------------------------------// d8
------------if (intersect(n-3) < intersect(n-2)) break;----------------------// 07
------------swap(h[n-2], h[n-1]);--------------------------------------------// bf
------------h.pop_back(); } }------------------------------------------------// 4b
----double get_min(double x) {-----------------------------------------------// b0
--------int lo = 0, hi = size(h) - 2, res = -1;------------------------------// 5b
--------while (lo <= hi) {---------------------------------------------------// 24
------------int mid = lo + (hi - lo) / 2;------------------------------------// 5a
------------if (intersect(mid) <= x) res = mid, lo = mid + 1;----------------// 1d
------------else hi = mid - 1; }---------------------------------------------// b6
--------return h[res+1].first * x + h[res+1].second; } };--------------------// 84
```

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;-----------------------------------------// f1
----return dist;---------------------------------------------------------------// 78
}------------------------------------------------------------------------------// a9
```

3.1.3. *IDA* algorithm.*

```
int n, cur[100], pos;----------------------------------------------------------// 48
int calch() {------------------------------------------------------------------// 88
----int h = 0;-----------------------------------------------------------------// 4a
----rep(i,0,n) if (cur[i] != 0) h += abs(i - cur[i]);--------------------------// 9b
----return h;------------------------------------------------------------------// c5
}------------------------------------------------------------------------------// c8
int dfs(int d, int g, int prev) {----------------------------------------------// 12
----int h = calch();-----------------------------------------------------------// 5d
----if (g + h > d) return g + h;-----------------------------------------------// 15
----if (h == 0) return 0;------------------------------------------------------// ff
----int mn = INF;--------------------------------------------------------------// 7e
----rep(di,-2,3) {-------------------------------------------------------------// 0d
--------if (di == 0) continue;-------------------------------------------------// 0a
--------int nxt = pos + di;----------------------------------------------------// 76
--------if (nxt == prev) continue;---------------------------------------------// 39
--------if (0 <= nxt && nxt < n) {---------------------------------------------// 68
------------swap(cur[pos], cur[nxt]);------------------------------------------// 35
------------swap(pos,nxt);-----------------------------------------------------// 64
------------mn = min(mn, dfs(d, g+1, nxt));------------------------------------// 22
------------swap(pos,nxt);-----------------------------------------------------// 84
------------swap(cur[pos], cur[nxt]);------------------------------------------// 3b
--------}----------------------------------------------------------------------// 46
--------if (mn == 0) break;----------------------------------------------------// 8f
----}--------------------------------------------------------------------------// d3
----return mn;-----------------------------------------------------------------// da
}------------------------------------------------------------------------------// f8
int idastar() {----------------------------------------------------------------// 22
----rep(i,0,n) if (cur[i] == 0) pos = i;---------------------------------------// 6b
----int d = calch();-----------------------------------------------------------// 38
----while (true) {-------------------------------------------------------------// 18
--------int nd = dfs(d, 0, -1);------------------------------------------------// 42
--------if (nd == 0 || nd == INF) return d;------------------------------------// b5
--------d = nd;----------------------------------------------------------------// f7
----}--------------------------------------------------------------------------// f9
}------------------------------------------------------------------------------// 82
```

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
```

3.7. **Minimum Cost Maximum Flow.** Running time is $O(|V|^2|E|\log|V|)$. NOTE: Doesn't work on negative weights!

```
#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
```

A second implementation that is slower but works on negative weights.

```
struct flow_network {----------------------------------------------------------// 81
----struct mcmf_edge {---------------------------------------------------------// f6
--------int u, v;--------------------------------------------------------------// e1
--------ll w, c;---------------------------------------------------------------// b4
--------mcmf_edge* rev;--------------------------------------------------------// 9d
--------mcmf_edge(int _u, int _v, ll _w, ll _c, mcmf_edge* _rev = NULL) {------// ea
------------u = _u; v = _v; w = _w; c = _c; rev = _rev;------------------------// 83
--------}----------------------------------------------------------------------// 02
----};-------------------------------------------------------------------------// b9
----int n;---------------------------------------------------------------------// b4
----vector<pair<int, pair<ll, ll> > >* adj;------------------------------------// 72
----flow_network(int _n) {-----------------------------------------------------// 55
--------n = _n;----------------------------------------------------------------// fa
--------adj = new vector<pair<int, pair<ll, ll> > >[n];------------------------// bb
----}--------------------------------------------------------------------------// bd
----void add_edge(int u, int v, ll cost, ll cap) {-----------------------------// 79
--------adj[u].push_back(make_pair(v, make_pair(cap, cost)));------------------// c8
----}--------------------------------------------------------------------------// ed
----pair<ll,ll> min_cost_max_flow(int s, int t) {------------------------------// ea
--------vector<mcmf_edge*>* g = new vector<mcmf_edge*>[n];---------------------// ce
--------for (int i = 0; i < n; i++) {------------------------------------------// 57
------------for (int j = 0; j < size(adj[i]); j++) {---------------------------// 37
----------------mcmf_edge *cur = new mcmf_edge(i, adj[i][j].first,-------------// 21
------------------------------adj[i][j].second.first, adj[i][j].second.second),--// 56
------------------------*rev = new mcmf_edge(adj[i][j].first, i, 0,------------// 48
-----------------------------------adj[i][j].second.second, cur);--------------// b1
----------------cur->rev = rev;------------------------------------------------// ef
----------------g[i].push_back(cur);-------------------------------------------// 1d
----------------g[adj[i][j].first].push_back(rev);-----------------------------// 05
------------}------------------------------------------------------------------// ba
--------}----------------------------------------------------------------------// 83
--------ll flow = 0, cost = 0;-------------------------------------------------// 68
--------mcmf_edge** back = new mcmf_edge*[n];----------------------------------// e5
--------ll* dist = new ll[n];--------------------------------------------------// 50
--------while (true) {---------------------------------------------------------// 65
------------for (int i = 0; i < n; i++) back[i] = NULL, dist[i] = INF;---------// d0
------------dist[s] = 0;-------------------------------------------------------// 5e
------------for (int i = 0; i < n - 1; i++)------------------------------------// be
----------------for (int j = 0; j < n; j++)------------------------------------// 6e
--------------------if (dist[j] != INF)----------------------------------------// e3
------------------------for (int k = 0; k < size(g[j]); k++)-------------------// 85
----------------------------if (g[j][k]->w > 0 && dist[j] + g[j][k]->c <-------// 7f
------------------------------------dist[g[j][k]->v]) {------------------------// 6d
--------------------------------dist[g[j][k]->v] = dist[j] + g[j][k]->c;-------// cf
--------------------------------back[g[j][k]->v] = g[j][k];--------------------// 3d
----------------------------}--------------------------------------------------// f3
------------mcmf_edge* cure = back[t];-----------------------------------------// b4
------------if (cure == NULL) break;-------------------------------------------// ab
------------ll cap = INF;------------------------------------------------------// 7a
------------while (true) {-----------------------------------------------------// ad
----------------cap = min(cap, cure->w);---------------------------------------// c3
----------------if (cure->u == s) break;---------------------------------------// 82
----------------cure = back[cure->u];------------------------------------------// 45
------------}------------------------------------------------------------------// 91
------------assert(cap > 0 && cap < INF);--------------------------------------// ae
------------cure = back[t];----------------------------------------------------// b9
------------while (true) {-----------------------------------------------------// 2a
----------------cost += cap * cure->c;-----------------------------------------// f8
----------------cure->w -= cap;------------------------------------------------// d1
----------------cure->rev->w += cap;-------------------------------------------// cf
----------------if (cure->u == s) break;---------------------------------------// 8c
----------------cure = back[cure->u];------------------------------------------// 60
------------}------------------------------------------------------------------// 09
------------flow += cap;-------------------------------------------------------// f2
--------}----------------------------------------------------------------------// be
--------// instead of deleting g, we could also--------------------------------// e0
--------// use it to get info about the actual flow----------------------------// 6c
--------for (int i = 0; i < n; i++)--------------------------------------------// eb
------------for (int j = 0; j < size(g[i]); j++)-------------------------------// 82
----------------delete g[i][j];------------------------------------------------// 06
--------delete[] g;------------------------------------------------------------// 23
--------delete[] back;---------------------------------------------------------// 5a
--------delete[] dist;---------------------------------------------------------// b9
--------return make_pair(flow, cost);------------------------------------------// ec
----}--------------------------------------------------------------------------// ad
};-----------------------------------------------------------------------------// bf
```

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 $O(|V|^3|E|)$. NOTE: Not sure if it works correctly with disconnected graphs.

```
#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(bool));-------------------------------------// c9
```

```
--------d[q[r++] = s] = 1;-----------------------------------------------------// dd
--------while (l < r) {--------------------------------------------------------// 45
------------same[v = q[l++]] = true;-------------------------------------------// c5
------------for (int i = g.head[v]; i != -1; i = g.e[i].nxt)-------------------// 66
----------------if (g.e[i].cap > 0 && d[g.e[i].v] == 0)------------------------// 21
--------------------d[q[r++] = g.e[i].v] = 1;----------------------------------// dd
--------}----------------------------------------------------------------------// 44
--------rep(i,s+1,n)-----------------------------------------------------------// 71
------------if (par[i].first == par[s].first && same[i]) par[i].first = s;-----// 97
--------g.reset();-------------------------------------------------------------// d8
----}--------------------------------------------------------------------------// 93
----rep(i,0,n) {---------------------------------------------------------------// 14
--------int mn = INF, cur = i;-------------------------------------------------// 59
--------while (true) {---------------------------------------------------------// b8
------------cap[cur][i] = mn;--------------------------------------------------// 8d
------------if (cur == 0) break;-----------------------------------------------// fb
------------mn = min(mn, par[cur].second), cur = par[cur].first;---------------// 4d
--------}----------------------------------------------------------------------// aa
----}--------------------------------------------------------------------------// 90
----return make_pair(par, cap);------------------------------------------------// 62
}------------------------------------------------------------------------------// b3
int compute_max_flow(int s, int t, const pair<vii, vvi> &gh) {-----------------// 93
----if (s == t) return 0;------------------------------------------------------// 33
----int cur = INF, at = s;-----------------------------------------------------// e7
----while (gh.second[at][t] == -1)---------------------------------------------// 42
--------cur = min(cur, gh.first[at].second), at = gh.first[at].first;----------// 8d
----return min(cur, gh.second[at][t]);-----------------------------------------// 54
}------------------------------------------------------------------------------// 46
```

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 adj; segment_tree values;----------------------------------------------// 13
----HLD(int _n) : n(_n), sz(n, 1), head(n), parent(n, -1), loc(n), adj(n) {----// 1c
--------vi tmp(n, ID); values = segment_tree(tmp); }---------------------------// f0
----void add_edge(int u, int v) { adj[u].push_back(v), adj[v].push_back(u); }--// 77
----void update_cost(int u, int v, int c) {------------------------------------// 7b
--------if (parent[v] == u) swap(u, v); assert(parent[u] == v);----------------// db
--------values.update(loc[u], c); }--------------------------------------------// 50
----int csz(int u) {-----------------------------------------------------------// 7c
--------rep(i,0,size(adj[u])) if (adj[u][i] != parent[u])----------------------// a5
------------sz[u] += csz(adj[parent[adj[u][i]] = u][i]);-----------------------// c2
--------return sz[u]; }--------------------------------------------------------// 75
----void part(int u) {---------------------------------------------------------// c3
--------head[u] = curhead; loc[u] = curloc++;----------------------------------// 63
--------int best = -1;---------------------------------------------------------// 27
--------rep(i,0,size(adj[u]))--------------------------------------------------// 49
------------if (adj[u][i] != parent[u] && (best == -1 || sz[adj[u][i]] > sz[best]))
----------------best = adj[u][i];----------------------------------------------// 26
--------if (best != -1) part(best);--------------------------------------------// c4
--------rep(i,0,size(adj[u]))--------------------------------------------------// 92
------------if (adj[u][i] != parent[u] && adj[u][i] != best)-------------------// e8
----------------part(curhead = adj[u][i]); }-----------------------------------// 88
----void build(int r = 0) { curloc = 0, csz(curhead = r), part(r); }-----------// 78
----int lca(int u, int v) {----------------------------------------------------// 74
--------vi uat, vat; int res = -1;---------------------------------------------// 43
--------while (u != -1) uat.push_back(u), u = parent[head[u]];-----------------// 51
--------while (v != -1) vat.push_back(v), v = parent[head[v]];-----------------// 6d
--------u = size(uat) - 1, v = size(vat) - 1;----------------------------------// 8a
--------while (u >= 0 && v >= 0 && head[uat[u]] == head[vat[v]])---------------// ae
------------res = (loc[uat[u]] < loc[vat[v]] ? uat[u] : vat[v]), u--, v--;-----// a2
--------return res; }----------------------------------------------------------// 91
----int query_upto(int u, int v) { int res = ID;-------------------------------// 72
--------while (head[u] != head[v])---------------------------------------------// 69
------------res = f(res, values.query(loc[head[u]], loc[u])),------------------// a4
------------u = parent[head[u]];-----------------------------------------------// 8c
--------return f(res, values.query(loc[v] + 1, loc[u])); }---------------------// ea
----int query(int u, int v) { int l = lca(u, v);-------------------------------// 53
--------return f(query_upto(u, l), query_upto(v, l)); } };---------------------// 5b
```

3.10. **Centroid Decomposition.**

```
#define MAXV 100100------------------------------------------------------------// 86
#define LGMAXV 20--------------------------------------------------------------// aa
int jmp[MAXV][LGMAXV],---------------------------------------------------------// 6d
----path[MAXV][LGMAXV],--------------------------------------------------------// 9d
----sz[MAXV], seph[MAXV],------------------------------------------------------// cf
----shortest[MAXV];------------------------------------------------------------// 6b
struct centroid_decomposition {------------------------------------------------// 99
----int n; vvi adj;------------------------------------------------------------// e9
----centroid_decomposition(int _n) : n(_n), adj(n) { }-------------------------// 46
----void add_edge(int a, int b) { adj[a].push_back(b), adj[b].push_back(a); }--// bc
----int dfs(int u, int p) {----------------------------------------------------// 8f
--------sz[u] = 1;-------------------------------------------------------------// c8
--------rep(i,0,size(adj[u])) if (adj[u][i] != p) sz[u] += dfs(adj[u][i], u);--// 78
--------return sz[u]; }--------------------------------------------------------// f4
----void makepaths(int sep, int u, int p, int len) {---------------------------// 84
--------jmp[u][seph[sep]] = sep, path[u][seph[sep]] = len;---------------------// d9
--------int bad = -1;----------------------------------------------------------// af
--------rep(i,0,size(adj[u])) {------------------------------------------------// f4
------------if (adj[u][i] == p) bad = i;---------------------------------------// cf
------------else makepaths(sep, adj[u][i], u, len + 1);------------------------// f2
--------}----------------------------------------------------------------------// 8a
--------if (p == sep) swap(adj[u][bad], adj[u].back()), adj[u].pop_back(); }---// 07
----void separate(int h=0, int u=0) {------------------------------------------// 03
--------dfs(u,-1); int sep = u;------------------------------------------------// b5
--------down: iter(nxt,adj[sep])-----------------------------------------------// 04
------------if (sz[*nxt] < sz[sep] && sz[*nxt] > sz[u]/2) {--------------------// db
----------------sep = *nxt; goto down; }---------------------------------------// 1a
--------seph[sep] = h, makepaths(sep, sep, -1, 0);-----------------------------// ed
--------rep(i,0,size(adj[sep])) separate(h+1, adj[sep][i]); }------------------// 90
```

```
    void paint(int u) {                                              // bd
        rep(h,0,seph[u]+1)                                           // c5
            shortest[jmp[u][h]] = min(shortest[jmp[u][h]], path[u][h]); } // 11
    int closest(int u) {                                             // 91
        int mn = INF/2;                                              // fe
        rep(h,0,seph[u]+1) mn = min(mn, path[u][h] + shortest[jmp[u][h]]); // 3e
        return mn; } };                                              // 13
```

3.11. **Least Common Ancestors, Binary Jumping.**

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

3.12. **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
```

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

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

4. Strings

4.1. **The Knuth-Morris-Pratt algorithm.**

```
int* compute_pi(const string &t) {                                   // a2
    int m = t.size();                                                // 8b
    int *pit = new int[m + 1];                                       // 8e
    if (0 <= m) pit[0] = 0;                                          // 42
    if (1 <= m) pit[1] = 0;                                          // 34
    rep(i,2,m+1) {                                                   // 0f
        for (int j = pit[i - 1]; ; j = pit[j]) {                     // b5
            if (t[j] == t[i - 1]) { pit[i] = j + 1; break; }         // 21
            if (j == 0) { pit[i] = 0; break; }                       // 95
        }                                                            // c9
    }                                                                // eb
    return pit; }                                                    // e8
int string_match(const string &s, const string &t) {                 // 9e
    int n = s.size(), m = t.size();                                  // 92
    int *pit = compute_pi(t);                                        // 72
    for (int i = 0, j = 0; i < n; ) {                                // 27
        if (s[i] == t[j]) {                                          // 73
            i++; j++;                                                // 7e
            if (j == m) {                                            // de
                return i - m;                                        // e9
                // or j = pit[j];                                    // ce
            }                                                        // 85
        }                                                            // 35
```

```
--------else if (j > 0) j = pit[j];--------------------------------------------// 43
--------else i++; }------------------------------------------------------------// b8
----delete[] pit; return -1; }-------------------------------------------------// e3
```

4.2. 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
```

4.3. 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
```

4.4. 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
```

4.5. **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
```

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

```
// TODO: Add longest common subsring----------------------------------------// 0e
const int MAXL = 100000;--------------------------------------------------------// 31
struct suffix_automaton {-------------------------------------------------------// e0
----vi  len, link, occur, cnt;--------------------------------------------------// 78
----vector<map<char,int> > next;------------------------------------------------// 90
----vector<bool> isclone;-------------------------------------------------------// 7b
----ll *occuratleast;-----------------------------------------------------------// f2
----int sz, last;---------------------------------------------------------------// 7d
----string s;-------------------------------------------------------------------// f2
----suffix_automaton() : len(MAXL*2), link(MAXL*2), occur(MAXL*2), next(MAXL*2),
----isclone(MAXL*2) { clear(); }------------------------------------------------// a3
----void clear(){ sz = 1; last = len[0] = 0; link[0] = -1; next[0].clear();----// aa
--------isclone[0] = false; }---------------------------------------------------// 26
----bool issubstr(string other){------------------------------------------------// 3b
--------for(int i = 0, cur = 0; i < size(other); ++i){-------------------------// 7f
------------if(cur == -1) return false; cur = next[cur][other[i]]; }-----------// 54
--------return true; }----------------------------------------------------------// 1a
----void extend(char c){ int cur = sz++; len[cur] = len[last] + 1;-------------// 1d
--------next[cur].clear(); isclone[cur] = false; int p = last;------------------// a9
--------for(; p != -1 && !next[p].count(c); p = link[p]){ next[p][c] = cur; }--// 6f
--------if(p == -1){ link[cur] = 0; }-------------------------------------------// 18
--------else{ int q = next[p][c];-----------------------------------------------// 34
------------if(len[p] + 1 == len[q]){ link[cur] = q; }--------------------------// 4d
------------else { int clone = sz++; isclone[clone] = true;---------------------// 57
----------------len[clone] = len[p] + 1;----------------------------------------// 8c
----------------link[clone] = link[q]; next[clone] = next[q];-------------------// 76
----------------for(; p != -1 && next[p].count(c) && next[p][c] == q; p = link[p]){
--------------------next[p][c] = clone; }---------------------------------------// 32
----------------link[q] = link[cur] = clone;------------------------------------// 73
------------} } last = cur; }---------------------------------------------------// b9
----void count(){---------------------------------------------------------------// e7
--------cnt=vi(sz, -1); stack<ii> S; S.push(ii(0,0));map<char,int>::iterator i;// 56
--------while(!S.empty()){------------------------------------------------------// 4c
------------ii cur = S.top(); S.pop();------------------------------------------// 67
------------if(cur.second){-----------------------------------------------------// 78
----------------for(i = next[cur.first].begin();i != next[cur.first].end();++i){
--------------------cnt[cur.first] += cnt[(*i).second]; } }--------------------// da
------------else if(cnt[cur.first] == -1){--------------------------------------// 99
----------------cnt[cur.first] = 1; S.push(ii(cur.first, 1));-------------------// bd
----------------for(i = next[cur.first].begin();i != next[cur.first].end();++i){
--------------------S.push(ii((*i).second, 0)); } } } }-------------------------// 61
----string lexicok(ll k){-------------------------------------------------------// 8b
--------int st = 0; string res; map<char,int>::iterator i;----------------------// cf
--------while(k){ for(i = next[st].begin(); i != next[st].end(); ++i){---------// 69
----------------if(k <= cnt[(*i).second]){ st = (*i).second; -----------------// ec
--------------------res.push_back((*i).first); k--; break;----------------------// 63
----------------} else { k -= cnt[(*i).second]; } } }---------------------------// ee
--------return res; }-----------------------------------------------------------// 0b
----void countoccur(){----------------------------------------------------------// ad
--------for(int i = 0; i < sz; ++i){ occur[i] = 1 - isclone[i]; }--------------// 1b
--------vii states(sz);---------------------------------------------------------// dc
--------for(int i = 0; i < sz; ++i){ states[i] = ii(len[i],i); }---------------// 97
--------sort(states.begin(), states.end());-------------------------------------// 8d
--------for(int i = size(states)-1; i >= 0; --i){ int v = states[i].second;----// a4
------------if(link[v] != -1) { occur[link[v]] += occur[v]; } } }--------------// cc
};------------------------------------------------------------------------------// 32
--------------------------------------------------------------------------------// 56
```

5. Mathematics

5.1. **Binomial Coefficients.** The binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the number of ways to choose k items out of a total of n items. Also contains an implementation of Lucas' theorem for computing the answer modulo a prime p.

```
int nck(int n, int k) {-------------------------------------------------------// f6
----if (n < k) return 0;------------------------------------------------------// 55
----k = min(k, n - k);--------------------------------------------------------// bd
----int res = 1;--------------------------------------------------------------// e6
----rep(i,1,k+1) res = res * (n - (k - i)) / i;-------------------------------// 4d
----return res;---------------------------------------------------------------// 1f
}-----------------------------------------------------------------------------// 6c
int nck(int n, int k, int p) {------------------------------------------------// cf
----int res = 1;--------------------------------------------------------------// 5c
----while (n || k) {----------------------------------------------------------// e2
--------res *= nck(n % p, k % p);---------------------------------------------// cc
--------res %= p, n /= p, k /= p;---------------------------------------------// 0a
----}-------------------------------------------------------------------------// d9
----return res;---------------------------------------------------------------// 30
}-----------------------------------------------------------------------------// 0a
```

5.2. **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
```

5.3. **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
```

5.4. **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
```

5.5. **Pollard's ρ algorithm.**

```
// public static int[] seeds = new int[] {2,3,5,7,11,13,1031};----------------// 1d
// public static BigInteger rho(BigInteger n, BigInteger seed) {--------------// 03
//---- int i = 0,-------------------------------------------------------------// 00
//-------- k = 2;-------------------------------------------------------------// 79
//---- BigInteger x = seed,---------------------------------------------------// cc
//----------------y = seed;---------------------------------------------------// 31
//---- while (i < 1000000) {--------------------------------------------------// 10
//-------- i++;---------------------------------------------------------------// 8c
//-------- x = (x.multiply(x).add(n).subtract(BigInteger.ONE)).mod(n);--------// 74
//-------- BigInteger d = y.subtract(x).abs().gcd(n);-------------------------// ce
//-------- if (!d.equals(BigInteger.ONE) && !d.equals(n)) {-------------------// b9
//------------ return d;------------------------------------------------------// 3b
//-------- } -----------------------------------------------------------------// 7c
//-------- if (i == k) {------------------------------------------------------// 2c
//------------ y = x;---------------------------------------------------------// 89
//------------ k = k*2;-------------------------------------------------------// 1d
//-------- }------------------------------------------------------------------// 10
//---- }----------------------------------------------------------------------// 96
//---- return BigInteger.ONE;-------------------------------------------------// 62
// }--------------------------------------------------------------------------// d7
```

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. **Primitive Root.**

```
#include "mod_pow.cpp"----------------------------------------------------------// c7
ll primitive_root(ll m) {-------------------------------------------------------// 8a
----vector<ll> div;-------------------------------------------------------------// f2
----for (ll i = 1; i*i <= m-1; i++) {-------------------------------------------// ca
--------if ((m-1) % i == 0) {---------------------------------------------------// 85
------------if (i < m) div.push_back(i);----------------------------------------// fd
------------if (m/i < m) div.push_back(m/i); } }--------------------------------// f2
----rep(x,2,m) {----------------------------------------------------------------// 57
--------bool ok = true;---------------------------------------------------------// 17
--------iter(it,div) if (mod_pow(x, *it, m) == 1) { ok = false; break; }--------// a4
--------if (ok) return x; }-----------------------------------------------------// 55
----return -1; }----------------------------------------------------------------// 15
```

5.9. **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
```

5.10. **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
```

5.11. **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
```

5.12. **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
```

5.13. **Number-Theoretic Transform.**

```
#include "../mathematics/primitive_root.cpp"------------------------------------// 8c
int mod = 998244353, g = primitive_root(mod),-----------------------------------// 9c
----ginv = mod_pow(g, mod-2, mod), inv2 = mod_pow(2, mod-2, mod);---------------// 75
#define MAXN (1<<22)------------------------------------------------------------// 94
struct Num {--------------------------------------------------------------------// c5
----int x;----------------------------------------------------------------------// 02
```

```
----Num(ll _x=0) { x = (_x%mod+mod)%mod; }------------------------------------// 1b
----Num operator +(const Num &b) { return x + b.x; }--------------------------// 08
----Num operator -(const Num &b) const { return x - b.x; }--------------------// 89
----Num operator *(const Num &b) const { return (ll)x * b.x; }----------------// e3
----Num operator /(const Num &b) const { return (ll)x * b.inv().x; }----------// 2a
----Num inv() const { return mod_pow((ll)x, mod-2, mod); }--------------------// d3
----Num pow(int p) const { return mod_pow((ll)x, p, mod); }-------------------// d5
} T1[MAXN], T2[MAXN];---------------------------------------------------------// d5
void ntt(Num x[], int n, bool inv = false) {----------------------------------// 24
----Num z = inv ? ginv : g;---------------------------------------------------// 00
----z = z.pow((mod - 1) / n);-------------------------------------------------// 45
----for (ll i = 0, j = 0; i < n; i++) {---------------------------------------// fe
--------if (i < j) swap(x[i], x[j]);------------------------------------------// eb
--------ll k = n>>1;----------------------------------------------------------// 02
--------while (1 <= k && k <= j) j -= k, k >>= 1;-----------------------------// d2
--------j += k; }-------------------------------------------------------------// c4
----for (int mx = 1, p = n/2; mx < n; mx <<= 1, p >>= 1) {--------------------// a2
--------Num wp = z.pow(p), w = 1;---------------------------------------------// 35
--------for (int k = 0; k < mx; k++, w = w*wp) {------------------------------// 42
------------for (int i = k; i < n; i += mx << 1) {----------------------------// 9a
----------------Num t = x[i + mx] * w;----------------------------------------// 22
----------------x[i + mx] = x[i] - t;-----------------------------------------// d0
----------------x[i] = x[i] + t; } } }----------------------------------------// 1e
----if (inv) {----------------------------------------------------------------// 6c
--------Num ni = Num(n).inv();------------------------------------------------// 18
--------rep(i,0,n) { x[i] = x[i] * ni; } } }----------------------------------// 72
void inv(Num x[], Num y[], int l) {-------------------------------------------// ee
----if (l == 1) { y[0] = x[0].inv(); return; }--------------------------------// 71
----inv(x, y, l>>1);----------------------------------------------------------// bd
----// NOTE: maybe l<<2 instead of l<<1---------------------------------------// 24
----rep(i,l>>1,l<<1) T1[i] = y[i] = 0;----------------------------------------// dd
----rep(i,0,l) T1[i] = x[i];--------------------------------------------------// f7
----ntt(T1, l<<1); ntt(y, l<<1);----------------------------------------------// a5
----rep(i,0,l<<1) y[i] = y[i]*2 - T1[i] * y[i] * y[i];------------------------// bc
----ntt(y, l<<1, true); }-----------------------------------------------------// 40
void sqrt(Num x[], Num y[], int l) {------------------------------------------// 44
----if (l == 1) { assert(x[0].x == 1); y[0] = 1; return; }--------------------// 64
----sqrt(x, y, l>>1);---------------------------------------------------------// a7
----inv(y, T2, l>>1);---------------------------------------------------------// 3b
----rep(i,l>>1,l<<1) T1[i] = T2[i] = 0;---------------------------------------// 1c
----rep(i,0,l) T1[i] = x[i];--------------------------------------------------// 08
----ntt(T2, l<<1); ntt(T1, l<<1);---------------------------------------------// 7e
----rep(i,0,l<<1) T2[i] = T1[i] * T2[i];--------------------------------------// 9f
----ntt(T2, l<<1, true);------------------------------------------------------// eb
----rep(i,0,l) y[i] = (y[i] + T2[i]) * inv2; }--------------------------------// b8
```

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

```
#define MAXN 5000-----------------------------------------------------------// f7
long double A[MAXN], B[MAXN], C[MAXN], D[MAXN], X[MAXN];----------------------// d8
void solve(int n) {-----------------------------------------------------------// 01
----C[0] /= B[0]; D[0] /= B[0];-----------------------------------------------// 94
----rep(i,1,n-1) C[i] /= B[i] - A[i]*C[i-1];----------------------------------// 6b
----rep(i,1,n) D[i] = (D[i] - A[i] * D[i-1]) / (B[i] - A[i] * C[i-1]);--------// 33
----X[n-1] = D[n-1];----------------------------------------------------------// c7
----for (int i = n-2; i>=0; i--) X[i] = D[i] - C[i] * X[i+1]; }---------------// ad
```

5.15. **Mertens Function.** Mertens function is $M(n) = \sum_{i=1}^{n} \mu(i)$. Let $L \approx (n \log\log n)^{2/3}$ and the algorithm runs in $O(n^{2/3})$. Can also be easily changed to compute the summatory Φ.

```
#define L 9000000-------------------------------------------------------------// 27
int mob[L], mer[L];-----------------------------------------------------------// f1
unordered_map<ll,ll> mem;-----------------------------------------------------// 30
ll M(ll n) {------------------------------------------------------------------// de
----if (n < L) return mer[n];-------------------------------------------------// 1c
----if (mem.find(n) != mem.end()) return mem[n];------------------------------// 79
----ll ans = 0, done = 1;-----------------------------------------------------// 48
----for (ll i = 2; i*i <= n; i++) ans += M(n/i), done = i;--------------------// 41
----for (ll i = 1; i*i <= n; i++) ans += mer[i] * (n/i - max(done, n/(i+1)));--// 43
----return mem[n] = 1 - ans; }------------------------------------------------// c2
void sieve() {----------------------------------------------------------------// b9
----for (int i = 1; i < L; i++) mer[i] = mob[i] = 1;--------------------------// f7
----for (int i = 2; i < L; i++) {---------------------------------------------// 8e
--------if (mer[i]) {---------------------------------------------------------// 8b
------------mob[i] = -1;------------------------------------------------------// e5
------------for (int j = i+i; j < L; j += i)----------------------------------// f0
----------------mer[j] = 0, mob[j] = (j/i)%i == 0 ? 0 : -mob[j/i];------------// 26
--------}---------------------------------------------------------------------// aa
--------mer[i] = mob[i] + mer[i-1]; } }---------------------------------------// 3b
```

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

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

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

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

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

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

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

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

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

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

5.19. **Formulas.**

- Number of permutations of n objects, where there are n_1 objects of type 1, n_2 objects of type 2, $\ldots$, 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 be chosen multiple times: $f^n_k = \binom{n+k-1}{k} = \frac{(n+k-1)!}{k!(n-1)!}$
- Number of integer solutions to $x_1 + x_2 + \cdots + x_n = k$ where $x_i \geq 0$: f^n_k
- 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 trees on n labeled vertices: n^{n-2}
- 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: $\binom{n}{k}D_{n-k}$
- Number of trees on n labeled vertices: n^{n-2}
- **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.
- **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 $G = (L \cup R, E)$, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover. Let U be the set of unmatched vertices in L, and Z be the set of vertices that are either in U or are connected to U by an alternating path. Then $K = (L \setminus Z) \cup (R \cap Z)$ is the minimum vertex cover.
- A minumum Steiner tree for n vertices requires at most $n - 2$ additional Steiner vertices.
- The number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.
- $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$
- **Wilson's theorem:** $(n-1)! \equiv -1 \pmod n$ iff. n is prime
- **Lagrange polynomial** through points $(x_0, y_0), \ldots, (x_k, y_k)$ is $L(x) = \sum_{j=0}^{k} y_j \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{x-x_m}{x_j-x_m}$
- $\sum_{k=0}^{m}(-1)^k\binom{n}{k} = (-1)^m\binom{n-1}{m}$
- $2^{\omega(n)} = O(\sqrt{n})$, where $\omega(n)$ is the number of distinct prime factors
- $\sum_{i=1}^{n} 2^{\omega(i)} = O(n \log n)$
- **Hook length formula:** If λ is a Young diagram and $h_\lambda(i, j)$ is the hook-length of cell (i, j), then then the number of Young tableux $d_\lambda = n!/\prod h_\lambda(i, j)$.
- **Möbius inversion formula:** If $f(n) = \sum_{d|n} g(d)$, then $g(n) = \sum_{d|n} \mu(d)f(n/d)$

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

6. Geometry

6.1. **Primitives.**

```
#define P(p) const point &p----------------------------------------------------// 2e
#define L(p0, p1) P(p0), P(p1)-------------------------------------------------// cf
#define C(p0, r) P(p0), double r-----------------------------------------------// f1
#define PP(pp) pair<point,point> &pp-------------------------------------------// e5
typedef complex<double> point;-------------------------------------------------// 6a
double dot(P(a), P(b)) { return real(conj(a) * b); }---------------------------// d2
double cross(P(a), P(b)) { return imag(conj(a) * b); }-------------------------// 8a
point rotate(P(p), double radians = pi / 2, P(about) = point(0,0)) {-----------// 23
----return (p - about) * exp(point(0, radians)) + about; }---------------------// 25
point reflect(P(p), L(about1, about2)) {---------------------------------------// 50
----point z = p - about1, w = about2 - about1;---------------------------------// 8b
----return conj(z / w) * w + about1; }-----------------------------------------// 83
point proj(P(u), P(v)) { return dot(u, v) / dot(u, u) * u; }-------------------// e7
point normalize(P(p), double k = 1.0) {----------------------------------------// 5f
----return abs(p) == 0 ? point(0,0) : p / abs(p) * k; }------------------------// 4a
double ccw(P(a), P(b), P(c)) { return cross(b - a, c - b); }-------------------// 27
bool collinear(P(a), P(b), P(c)) { return abs(ccw(a, b, c)) < EPS; }-----------// b3
double angle(P(a), P(b), P(c)) {-----------------------------------------------// 61
----return acos(dot(b - a, c - b) / abs(b - a) / abs(c - b)); }----------------// c7
double signed_angle(P(a), P(b), P(c)) {----------------------------------------// 4a
----return asin(cross(b - a, c - b) / abs(b - a) / abs(c - b)); }--------------// 40
double angle(P(p)) { return atan2(imag(p), real(p)); }-------------------------// e6
point perp(P(p)) { return point(-imag(p), real(p)); }--------------------------// d9
double progress(P(p), L(a, b)) {-----------------------------------------------// b3
----if (abs(real(a) - real(b)) < EPS)------------------------------------------// 5e
--------return (imag(p) - imag(a)) / (imag(b) - imag(a));----------------------// 5e
----else return (real(p) - real(a)) / (real(b) - real(a)); }-------------------// 31
-------------------------------------------------------------------------------// 53
-------------------------------------------------------------------------------// 46
```

6.2. **Lines.**

```
#include "primitives.cpp"------------------------------------------------------// e0
-------------------------------------------------------------------------------// 85
bool collinear(L(a, b), L(p, q)) {---------------------------------------------// 2f
----return abs(ccw(a, b, p)) < EPS && abs(ccw(a, b, q)) < EPS;  }--------------// 3e
bool parallel(L(a, b), L(p, q)) { return abs(cross(b - a, q - p)) < EPS; }-----// 8d
point closest_point(L(a, b), P(c), bool segment = false) {---------------------// f2
----if (segment) {-------------------------------------------------------------// f4
```

```
--------if (dot(b - a, c - b) > 0) return b;-----------------------------------// 88
--------if (dot(a - b, c - a) > 0) return a;-----------------------------------// 75
----}--------------------------------------------------------------------------// ce
----double t = dot(c - a, b - a) / norm(b - a);--------------------------------// 62
----return a + t * (b - a);----------------------------------------------------// 6e
}------------------------------------------------------------------------------// 8c
double line_segment_distance(L(a,b), L(c,d)) {---------------------------------// f3
----double x = INFINITY;-------------------------------------------------------// 64
----if (abs(a - b) < EPS && abs(c - d) < EPS) x = abs(a - c);-----------------// a5
----else if (abs(a - b) < EPS) x = abs(a - closest_point(c, d, a, true));-----// 23
----else if (abs(c - d) < EPS) x = abs(c - closest_point(a, b, c, true));-----// 53
----else if ((ccw(a, b, c) < 0) != (ccw(a, b, d) < 0) &&----------------------// 6d
------------ (ccw(c, d, a) < 0) != (ccw(c, d, b) < 0)) x = 0;-----------------// bf
----else {---------------------------------------------------------------------// e1
--------x = min(x, abs(a - closest_point(c,d, a, true)));---------------------// 29
--------x = min(x, abs(b - closest_point(c,d, b, true)));---------------------// fe
--------x = min(x, abs(c - closest_point(a,b, c, true)));---------------------// 81
--------x = min(x, abs(d - closest_point(a,b, d, true)));---------------------// e4
----}--------------------------------------------------------------------------// c5
----return x;------------------------------------------------------------------// b7
}------------------------------------------------------------------------------// 27
bool intersect(L(a, b), L(p, q), point &res, bool segment = false) {-----------// d2
----// NOTE: check for parallel/collinear lines before calling this function---// 1b
----point r = b - a, s = q - p;------------------------------------------------// 34
----double c = cross(r, s), t = cross(p - a, s) / c, u = cross(p - a, r) / c;--// 0b
----if (segment && (t < 0-EPS || t > 1+EPS || u < 0-EPS || u > 1+EPS))---------// e4
--------return false;----------------------------------------------------------// e3
----res = a + t * r;-----------------------------------------------------------// 47
----return true;---------------------------------------------------------------// 05
}------------------------------------------------------------------------------// 44
-------------------------------------------------------------------------------// cc
```

6.3. **Circles.**

```
#include "primitives.cpp"----------------------------------------------------// e0
int intersect(C(A, rA), C(B, rB), point & res1, point & res2) {----------------// 3b
----double d = abs(B - A);-----------------------------------------------------// 5c
----if ((rA + rB) <  (d - EPS) || d < abs(rA - rB) - EPS) return 0;------------// 39
----double a = (rA*rA - rB*rB + d*d) / 2 / d, h = sqrt(rA*rA - a*a);-----------// 9b
----point v = normalize(B - A, a), u = normalize(rotate(B-A), h);--------------// 79
----res1 = A + v + u, res2 = A + v - u;----------------------------------------// 24
----if (abs(u) < EPS) return 1; return 2;--------------------------------------// 82
}------------------------------------------------------------------------------// bb
-------------------------------------------------------------------------------// ce
int intersect(L(A, B), C(O, r), point & res1, point & res2) {------------------// 4d
---- double h = abs(O - closest_point(A, B, O));-------------------------------// d0
---- if(r < h - EPS) return 0;-------------------------------------------------// af
---- point H = proj(O - A, B - A) + A, v = normalize((B - A), sqrt(r*r - h*h));// 1a
---- res1 = H + v; res2 = H - v;-----------------------------------------------// fc
---- if(abs(v) < EPS) return 1; return 2;--------------------------------------// d1
}------------------------------------------------------------------------------// f8
-------------------------------------------------------------------------------// 74
int tangent(P(A), C(O, r), point & res1, point & res2) {-----------------------// ec
----point v = O - A; double d = abs(v);----------------------------------------// 62
----if (d < r - EPS) return 0;-------------------------------------------------// fc
----double alpha = asin(r / d), L = sqrt(d*d - r*r);---------------------------// f0
----v = normalize(v, L);-------------------------------------------------------// 0f
----res1 = A + rotate(v, alpha); res2 = A + rotate(v, -alpha);-----------------// 9e
----if (abs(r - d) < EPS || abs(v) < EPS) return 1;----------------------------// ee
----return 2;------------------------------------------------------------------// 4f
}------------------------------------------------------------------------------// 3c
-------------------------------------------------------------------------------// 29
void tangent_outer(point A, double rA, point B, double rB, PP(P), PP(Q)) {-----// 9e
----if (rA - rB > EPS) { swap(rA, rB); swap(A, B); }---------------------------// 9c
----double theta = asin((rB - rA)/abs(A - B));---------------------------------// 68
----point v = rotate(B - A, theta + pi/2), u = rotate(B - A, -(theta + pi/2));// d7
----u = normalize(u, rA);------------------------------------------------------// 87
----P.first = A + normalize(v, rA); P.second = B + normalize(v, rB);-----------// 20
----Q.first = A + normalize(u, rA); Q.second = B + normalize(u, rB);-----------// 52
}------------------------------------------------------------------------------// 63
```

6.4. **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
```

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

```
#include "polygon.cpp"-----------------------------------------------------// 58
#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
```

6.6. **Line Segment Intersection.**

```
#include "primitives.cpp"--------------------------------------------------// e0
#include "lines.cpp"-------------------------------------------------------// 54
bool line_segment_intersect(L(a, b), L(c, d), point &A, point &B) {-----------// 34
----if (abs(a - b) < EPS && abs(c - d) < EPS) {-----------------------------// 74
--------A = B = a; return abs(a - d) < EPS; }-----------------------------// 37
----else if (abs(a - b) < EPS) {-------------------------------------------// 07
--------A = B = a; double p = progress(a, c,d);---------------------------// 6c
--------return 0.0 <= p && p <= 1.0-----------------------------------------// 4c
------------&& (abs(a - c) + abs(d - a) - abs(d - c)) < EPS; }---------------// d2
----else if (abs(c - d) < EPS) {-------------------------------------------// a0
--------A = B = c; double p = progress(c, a,b);---------------------------// c0
--------return 0.0 <= p && p <= 1.0-----------------------------------------// fb
------------&& (abs(c - a) + abs(b - c) - abs(b - a)) < EPS; }---------------// 15
----else if (collinear(a,b, c,d)) {----------------------------------------// 49
--------double ap = progress(a, c,d), bp = progress(b, c,d);-----------------// 34
--------if (ap > bp) swap(ap, bp);----------------------------------------// e0
--------if (bp < 0.0 || ap > 1.0) return false;---------------------------// 14
--------A = c + max(ap, 0.0) * (d - c);-----------------------------------// 14
--------B = c + min(bp, 1.0) * (d - c);-----------------------------------// 26
--------return true; }------------------------------------------------------// bd
----else if (parallel(a,b, c,d)) return false;--------------------------------// fc
----else if (intersect(a,b, c,d, A, true)) {----------------------------------// 78
--------B = A; return true; }-----------------------------------------------// 7d
----return false;----------------------------------------------------------// 4b
}-------------------------------------------------------------------------------// 05
--------------------------------------------------------------------------------// 83
```

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

```
double gc_distance(double pLat, double pLong,----------------------------------// 7b
----------------  double qLat, double qLong, double r) {------------------------// a4
----pLat *= pi / 180; pLong *= pi / 180;--------------------------------------// ee
----qLat *= pi / 180; qLong *= pi / 180;--------------------------------------// 75
----return r * acos(cos(pLat) * cos(qLat) * cos(pLong - qLong) +-----------------// e3
--------------------sin(pLat) * sin(qLat));-----------------------------------// 1e
--------------------------------------------------------------------------------// 60
}-------------------------------------------------------------------------------// 3f
```

6.8. **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
```

6.9. **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
```

6.10. **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
```

6.11. 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
```

6.12. Rotating Calipers.

```
#include "primitives.cpp"----------------------------------------------------// e0
struct caliper {---------------------------------------------------------------// 8e
----ii pt;---------------------------------------------------------------------// 05
----double angle;--------------------------------------------------------------// d4
----caliper(ii _pt, double _angle) : pt(_pt), angle(_angle) { }----------------// 35
----double angle_to(ii pt2) {--------------------------------------------------// 8b
--------double x = angle - atan2(pt2.second - pt.second, pt2.first - pt.first);// 1e
--------while (x >= pi) x -= 2*pi;---------------------------------------------// 4a
--------while (x <= -pi) x += 2*pi;--------------------------------------------// a3
--------return x; }------------------------------------------------------------// 7d
----void rotate(double by) {---------------------------------------------------// 57
--------angle -= by;-----------------------------------------------------------// 5d
```

```
--------while (angle < 0) angle += 2*pi;--------------------------------------// 03
----}-------------------------------------------------------------------------// 20
----void move_to(ii pt2) { pt = pt2; }----------------------------------------// 37
----double dist(const caliper &other) {---------------------------------------// 68
--------point a(pt.first,pt.second),------------------------------------------// d7
------------  b = a + exp(point(0,angle)) * 10.0,-----------------------------// 2e
------------  c(other.pt.first, other.pt.second);-----------------------------// 71
--------return abs(c - closest_point(a, b, c));-------------------------------// 58
----} };----------------------------------------------------------------------// 4b
------------------------------------------------------------------------------// c5
// int h = convex_hull(pts);--------------------------------------------------// 9c
// double mx = 0;-------------------------------------------------------------// f1
// if (h > 1) {---------------------------------------------------------------// 26
//---- int a = 0,-------------------------------------------------------------// e6
//-------- b = 0;-------------------------------------------------------------// df
//---- rep(i,0,h) {-----------------------------------------------------------// 1d
//-------- if (hull[i].first < hull[a].first)---------------------------------// ac
//------------ a = i;---------------------------------------------------------// b1
//-------- if (hull[i].first > hull[b].first)---------------------------------// 02
//------------ b = i;---------------------------------------------------------// 84
//---- }----------------------------------------------------------------------// 1e
//---- caliper A(hull[a], pi/2), B(hull[b], 3*pi/2);--------------------------// 60
//---- double done = 0;-------------------------------------------------------// 3c
//---- while (true) {---------------------------------------------------------// 31
//-------- mx = max(mx, abs(point(hull[a].first,hull[a].second)---------------// e3
//-------------------  - point(hull[b].first, hull[b].second)));--------------// 24
//-------- double tha = A.angle_to(hull[(a+1)%h]),----------------------------// 57
//---------------- thb = B.angle_to(hull[(b+1)%h]);---------------------------// f1
//-------- if (tha <= thb) {--------------------------------------------------// 91
//------------ A.rotate(tha);-------------------------------------------------// c9
//------------ B.rotate(tha);-------------------------------------------------// f4
//------------ a = (a+1) % h;-------------------------------------------------// d4
//------------ A.move_to(hull[a]);--------------------------------------------// b3
//-------- } else {-----------------------------------------------------------// 56
//------------ A.rotate(thb);-------------------------------------------------// 56
//------------ B.rotate(thb);-------------------------------------------------// 38
//------------ b = (b+1) % h;-------------------------------------------------// 96
//------------ B.move_to(hull[b]);--------------------------------------------// 38
//-------- }------------------------------------------------------------------// bc
//-------- done += min(tha, thb);---------------------------------------------// d2
//-------- if (done > pi) {---------------------------------------------------// c2
//------------ break;---------------------------------------------------------// e8
//-------- }------------------------------------------------------------------// 37
//---- }----------------------------------------------------------------------// ac
// }--------------------------------------------------------------------------// 9c
```

6.13. **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.
- **Euler's formula:** $V - E + F = 2$
- Side lengths a, b, c can form a triangle iff. $a + b > c$, $b + c > a$ and $a + c > b$.
- Sum of internal angles of a regular convex n-gon is $(n - 2)\pi$.

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. ***n*th 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
```

7.7. **Simulated Annealing.** An example use of Simulated Annealing to find a permutation of length n that maximizes $\sum_{i=1}^{n-1} |p_i - p_{i+1}|$.

```
double curtime() { return static_cast<double>(clock()) / CLOCKS_PER_SEC; }-----// 9d
int simulated_annealing(int n, double seconds) {------------------------------// 54
----default_random_engine rng;------------------------------------------------// 67
----uniform_real_distribution<double> randfloat(0.0, 1.0);--------------------// ed
----uniform_int_distribution<int> randint(0, n - 2);--------------------------// bb
------------------------------------------------------------------------------// 88
----// random initial solution------------------------------------------------// 22
----vi sol(n);----------------------------------------------------------------// 33
----rep(i,0,n) sol[i] = i + 1;------------------------------------------------// ee
----random_shuffle(sol.begin(), sol.end());-----------------------------------// 1e
------------------------------------------------------------------------------// 5b
----// initialize score------------------------------------------------------// 11
----int score = 0;------------------------------------------------------------// 4d
----rep(i,1,n) score += abs(sol[i] - sol[i-1]);-------------------------------// 74
------------------------------------------------------------------------------// 25
----int iters = 0;------------------------------------------------------------// 4d
----double T0 = 100.0, T1 = 0.001,--------------------------------------------// f4
--------   progress = 0, temp = T0,-------------------------------------------// 8b
--------   starttime = curtime();---------------------------------------------// a2
----while (true) {------------------------------------------------------------// db
--------if (!(iters & ((1 << 4) - 1))) {--------------------------------------// e8
------------progress = (curtime() - starttime) / seconds;---------------------// a0
------------temp = T0 * pow(T1 / T0, progress);-------------------------------// 12
------------if (progress > 1.0) break;----------------------------------------// 48
--------}---------------------------------------------------------------------// 1a
------------------------------------------------------------------------------// 88
--------// random mutation---------------------------------------------------// 84
--------int a = randint(rng);-------------------------------------------------// f7
------------------------------------------------------------------------------// 02
--------// compute delta for mutation----------------------------------------// 4e
--------int delta = 0;--------------------------------------------------------// 10
--------if (a > 0) delta += abs(sol[a+1] - sol[a-1]) - abs(sol[a] - sol[a-1]);-// 21
--------if (a+2 < n) delta += abs(sol[a] - sol[a+2]) - abs(sol[a+1] - sol[a+2]);
------------------------------------------------------------------------------// 22
--------// maybe apply mutation----------------------------------------------// 4d
--------if (delta >= 0 || randfloat(rng) < exp(delta / temp)) {---------------// a6
------------swap(sol[a], sol[a+1]);-------------------------------------------// ce
------------score += delta;---------------------------------------------------// 64
------------// if (score >= target) return;----------------------------------// a6
--------}---------------------------------------------------------------------// 85
--------iters++;--------------------------------------------------------------// 3c
----}-------------------------------------------------------------------------// ec
----return score;-------------------------------------------------------------// d0
}-----------------------------------------------------------------------------// ec
```

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)?
- 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?
- 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) {------------------------------------------------------// 73
----int y = x & -x, z = x + y;------------------------------------------// 12
----return z | ((x ^ z) >> 2) / y;--------------------------------------// 97
}-----------------------------------------------------------------------// 14
```

9. Misc

9.1. Debugging Tips.

- Stack overflow? Recursive DFS on tree that is actually a long path?
- Floating-point numbers
  - Getting `NaN`? Make sure `acos` etc. are not getting values out of their range (perhaps `1+eps`).
  - Rounding negative numbers?
  - Outputting in scientific notation?
- Wrong Answer?
  - Integer overflow?
  - Think very carefully about boundaries of all input parameters
  - Try out possible edge cases:
    * $n = 0, n = -1, n = 1, n = 2^{31} - 1$ or $n = -2^{31}$
    * List is empty, or contains a single element
    * n is even, n is odd
    * Graph is empty, or contains a single vertex
    * Graph is a multigraph (loops or multiple edges)
    * Polygon is concave or non-simple
  - Are multiple test cases being handled correctly? Try repeating the same test case many times.
  - Is initial condition wrong for small cases?

9.2. Solution Ideas.

- Dynamic Programming
  - Drop a parameter, recover from others
  - Swap answer and a parameter
  - Parsing CFGs: CYK Algorithm
  - Optimizations
    * Convex hull optimization
      · $\mathrm{dp}[i] = \min_{j<i}\{\mathrm{dp}[j] + b[j] \times a[i]\}$
      · $b[j] \geq b[j+1]$
      · optionally $a[i] \leq a[i+1]$
      · $O(n^2)$ to $O(n)$
    * Divide and conquer optimization
      · $\mathrm{dp}[i][j] = \min_{k<j}\{\mathrm{dp}[i-1][k] + C[k][j]\}$
      · $A[i][j] \leq A[i][j+1]$
      · $O(kn^2)$ to $O(kn \log n)$
      · sufficient: $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$, $a \leq b \leq c \leq d$ (QI)
    * Knuth optimization
      · $\mathrm{dp}[i][j] = \min_{i<k<j}\{\mathrm{dp}[i][k] + \mathrm{dp}[k][j] + C[i][j]\}$
      · $A[i][j-1] \leq A[i][j] \leq A[i+1][j]$
      · $O(n^3)$ to $O(n^2)$
      · sufficient: QI and $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$
- Greedy
- Randomized
- Optimizations
  - Use bitset (/64)
  - Switch order of loops (cache locality)
- Process queries offline
  - Mo's algorithm
- Square-root decomposition
- Precomputation
- Efficient simulation
  - Mo's algorithm
  - Sqrt decomposition
  - Store 2^k jump pointers
- Data structure techniques
  - Sqrt buckets
  - Store 2^k jump pointers
  - 2^k merging trick
- Counting
  - Inclusion-exclusion principle
  - Generating functions
- Graphs
  - Can we model the problem as a graph?
  - Can we use any properties of the graph?
  - Strongly connected components
  - Cycles (or odd cycles)
  - Bipartite (no odd cycles)
    * Bipartite matching
    * Hall's marriage theorem
    * Stable Marriage
  - Cut vertex/bridge
  - Biconnected components
  - Degrees of vertices (odd/even)
  - Trees
    * Heavy-light decomposition
    * Centroid decomposition

    * Least common ancestor
  - Eulerian path/circuit
  - Chinese postman problem
  - Topological sort
  - (Min-Cost) Max Flow
  - Min Cut
    * Maximum Density Subgraph
  - Huffman Coding
  - Min-Cost Arborescence
  - Steiner Tree
  - Kirchoff's matrix tree theorem
  - Prüfer sequences
- Mathematics
  - Is the function multiplicative?
  - Look for a pattern
  - Permutations
    * Consider the cycles of the permutation
  - Functions
    * Sum of piecewise-linear functions is a piecewise-linear function
    * Sum of convex (concave) functions is convex (concave)
  - Modular arithmetic
    * Chinese Remainder Theorem
    * Linear Congruence
  - Sieve
  - System of linear equations
  - Values to big to represent?
    * Compute using the logarithm
    * Divide everything by some large value
- Logic
  - 2-SAT
  - XOR-SAT (Gauss elimination or Bipartite matching)
- Meet in the middle
- Only work with the smaller half ($\log(n)$)
- Strings
  - Trie (maybe over something weird, like bits)
  - Suffix array
  - Suffix automaton (+DP?)
  - Aho-Corasick
  - eerTree
  - Work with $S + S$
- Hashing
- Euler tour, tree to array
- Segment trees
  - Lazy propagation
  - Persistent
  - Implicit
  - Segment tree of X
- Geometry
  - Minkowski sum (of convex sets)
  - Rotating calipers
  - Sweep line (horizontally or vertically?)
  - Sweep angle
  - Convex hull
- Fix a parameter (possibly the answer).
- Are there few distinct values?
- Binary search
- Sliding Window (+ Monotonic Queue)
- Computing a Convolution? Fast Fourier Transform
- Exact Cover (+ Algorithm X)
- Cycle-Finding
- What is the smallest set of values that identify the solution? The cycle structure of the permutation? The powers of primes in the factorization?
- Look at the complement problem
  - Minimize something instead of maximizing
- Immediately enforce necessary conditions. (All values greater than 0? Initialize them all to 1)
- Add large constant to negative numbers to make them positive
- Counting/Bucket sort

Practice Contest Checklist.

- How many operations per second?
- What is the stack size?
- How to use printf/scanf with long long/long double?
- Is $\_\_int128$ available?
- Does MLE give RTE or MLE as a verdict? What about stack overflow?
- What is $RAND\_MAX$?