



Handbook USM

World Finals

Team: La CASneta

University: UTFSM

Contents

General: template, debugging tricks, optimization tricks, troubleshoot, hash

Graphs: bellman ford, eppstein, lca, lca rmq, hld, centroid decomposition, tarjan, tarjan scc, dominator tree, euler directed graph, undirected euler trail, 2sat, dinic, mfed, hopcroft karp, hungarian, min cost flow, general matching, global min cut, gomory hu, graph theory reference, fast chu liu, 4 cycle, 3 cycle, chromatic number, tortoise hare

Data structures: min deque, wavelet tree, succinct indexable dictionary, interval container, union find, union find rollback, sparse table, disjoint sparse table, range query constant, segment tree, segment tree with walk, segment tree lazy, iterative segment tree lazy, persistent segment tree, dynamic segment tree, query tree, bit, bit 2d, merge sort tree, line container, kd tree, mo, mo rollback, mo update, ordered set, treap, implicit treap, lct

Maths: equations, trigonometry, geometry, calculus, sums series, probability, mulmod, binary pow, binary pow 128 bits, binary pow mulmod, montgomery, number theory reference, extended gcd, crt, prime factors, counting divisors, discrete log, erathostenes sieve, euler phi, euler phi sieve, mobius sieve, miller rabin, pollard rho, fast prime factorization, mod sqrt, continued fractions, combinatorics reference, special numbers, factorial, tetration, gauss, xor basis, matrix, simplex, fft, fast ntt, fwht, poly shift, sieveking kung, lagrange point, berlekamp massey, linear recurrence, integrate

Geometry: point 2d, segment, halfplane, circle, line distance, segment distance, convex hull, halfplane intersection, point inside polygon, polygon area, nearest 2 points, minimum enclosing circle, convex hull 3d, order by angle, order by slope, lattice point, circle intersection, circle tangents, circle polygon intersection, polygon center, polygon cut, hull diameter

Strings: rolling hashing, kmp, z, aho corasick, suffix array, suffix automaton, manacher, trie, min rotation

Dynamic programming: lis, egg drop, d&c, knuth, li chao tree, li chao segment, fast knapsack

Combinatorial: weighted matroid intersection, simulated annealing

Template, Debugging Tricks, Optimization Tricks, Troubleshoot, Hash, Bellman Ford, Eppstein, LCA

General

Template

fa7b2c

Description: Just the starting template code

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

using namespace std;

using ll = long long;
using ld = long double;

const ll mod = 1e9 + 7;
const ll inf = 1e12;
const ld pi = acos(-1);

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    cout << fixed << setprecision(9);

    return 0;
}
```

Debugging Tricks

- `signal(SIGSEGV, [](int) { _Exit(0); })`; converts segfaults into Wrong Answers. Similarly catch `SIGABRT` (assertion failures) and `SIGFPE` (zero divisions).
- `feenableexcept(29)`; kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

Optimization Tricks

`_builtin_ia32_ldmxcsr(40896)`; disables denormals (which make floats 20x slower near their minimum value).

Bit hacks:

- `x & -x` is the least significant bit of `x`.
 - `for (int x = m; x;) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
 - `c = x&x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same popcount.
 - Subset sum over all bitmask in $O(2^K)$:
- ```
rep(b,0,K) rep(i,0,(1 << K))
 if (i & 1 << b) D[i] += D[i^(1 << b)];
```

## Pragmas:

- `#pragma GCC optimize ("Ofast")` — auto-vectorize loops, optimize floats.
- `#pragma GCC target ("avx2")` — can double vectorized performance (crashes on old machines).
- `#pragma GCC optimize ("trapv")` — kills program on integer overflows (slow).

## Troubleshoot

## Pre-submit:

- Write a few simple test cases if sample is not enough.
- Are time limits close? If so, generate max cases.
- Is the memory usage fine?
- Could anything overflow?
- Make sure to submit the right file.

## Wrong answer:

- Print your solution! Print debug output, as well.
- Are you clearing all data structures between test cases?
- Can your algorithm handle the whole range of input?
- Read the full problem statement again.
- Do you handle all corner cases correctly?
- Have you understood the problem correctly?
- Any uninitialized variables?
- Any overflows?
- Confusing N and M, i and j, etc.?
- Are you sure your algorithm works?
- What special cases have you not thought of?
- Are you sure the STL functions you use work as you think?
- Add some assertions, maybe resubmit.
- Create some testcases to run your algorithm on.
- Go through the algorithm for a simple case.
- Go through this list again.
- Explain your algorithm to a teammate.
- Ask the teammate to look at your code.
- Go for a small walk, e.g. to the toilet.
- Is your output format correct? (including whitespace)
- Are you using modular inverse?
- Rewrite your solution from the start or let a teammate do it.

## Runtime error:

- Have you tested all corner cases locally?
- Any uninitialized variables?
- Are you reading or writing outside the range of any vector?
- Any assertions that might fail?
- Any possible division by 0? (mod 0 for example)
- Any possible infinite recursion?
- Invalidated pointers or iterators?
- Are you using too much memory?
- Debug with resubmits (e.g. remapped signals, see Various).

## Time limit exceeded:

- Do you have any possible infinite loops?
- What is the complexity of your algorithm?
- Are you copying a lot of unnecessary data? (References)
- How big is the input and output? (consider `scanf`)
- Avoid vector, map, (use arrays/unordered\_map)
- What do your teammates think about your algorithm?

## Memory limit exceeded:

- What is the max amount of memory your algorithm should need?
- Are you clearing all data structures between test cases?

## Hash

If you want to check if your code is copied correctly, you can check the square at the top right of each code. If it matches the hash of this command, it is correctly copied.

```
cat template.cpp | tr -d '[:space:]' | md5sum | head -c 6
```

## Graphs

## Bellman Ford

# 28ac61

**Description:** Calculates shortest paths from `s` in a graph that might have negative edge weights.**Status:** Tested on CSES

```
struct BellmanFord {
 struct Edge { int from, to; ll weight; };
 int n, last_updated = -1; const ll INF = 1e18;
 vector<int> p; vector<ll> dist;
 BellmanFord(vector<Edge> &G, int s, int _n) {
 n = _n; dist.assign(n+1, INF);
 p.assign(n+1, -1); dist[s] = 0;
 for (int i = 1; i <= n; i++) {
 last_updated = -1;
 for (Edge &e : G)
 if (dist[e.from] + e.weight < dist[e.to]) {
 dist[e.to] = dist[e.from] + e.weight;
 p[e.to] = e.from; last_updated = e.to;
 }
 }
 }
 bool getCycle(vector<int> &cycle) {
 if (last_updated == -1) return false;
 for (int i = 0; i < n-1; i++)
 last_updated = p[last_updated];
 for (int x = last_updated ; x!=p[x]; x=p[x]) {
 cycle.push_back(x);
 if (x == last_updated and cycle.size() > 1) break;
 }
 reverse(cycle.begin(), cycle.end());
 return true;
 }
};
```

## Eppstein

# 52e499

**Description:** Solve k-shortest path problem  
**Status:** Tested on josupo.jp and CSES

```
struct Eppstein {
#define x first
#define y second
 using T = ll; const T INF = 1e15;
 using Edge = pair<int, T>;
 struct Node { int E[2] = {}, s{0}; Edge x; } T shortest;
 priority_queue<pair<T, int>> Q;
 vector<Node> P{1}; vector<int> h;
 Eppstein(vector<vector<Edge>> &G, int s, int t) {
 int n = G.size();
 vector<vector<Edge>> H(n);
 for(int i = 0; i < n; i++)
 for (Edge &e : G[i])
 H[e.x].push_back({i, e.y});
 vector<int> ord, par(n, -1);
 vector<T> d(n, -INF);
 Q.push({d[t], 0, t});
 while (!Q.empty()) {
 auto v = Q.top(); Q.pop();
 if (d[v.y] == v.x) {
 ord.push_back(v.y);
 for (Edge &e : H[v.y])
 if (v.x-e.y > d[e.x]) {
 Q.push({d[e.x] = v.x-e.y, e.x});
 par[e.x] = v.y;
 }
 }
 }
 if ((shortest = -d[s]) >= INF) return;
 h.resize(n);
 for (int v : ord) {
 int p = par[v];
 if (p+1) h[v] = h[p];
 for (Edge &e : G[v])
 if (d[e.x] > -INF) {
 T k = e.y - d[e.x] + d[v];
 if (k or e.x != p) h[v] = push(h[v], {e.x, k});
 else p = -1;
 }
 }
 }
 int push(int t, Edge x) {
 P.push(P[t]);
 if (!P[t].E[0].s or P[t].x.y >= x.y)
 swap(x, P[t].x);
 if (P[t].s) {
 int i = P[t].E[0], j = P[t].E[1];
 int d = P[i].s > P[j].s ? P[i].s : P[j].s;
 int k = push(d ? j : i, x);
 P[t].E[d] = k;
 }
 P[t].s++;
 return t;
 }
 int nextPath() {
 if (Q.empty()) return -1;
 auto v = Q.top(); Q.pop();
 for (int i : P[v.y].E) if (i)
 Q.push({v.x-P[i].x.y+P[v.y].x.y, i});
 int t = h[P[v.y].x.x];
 if (t) Q.push({v.x - P[t].x.y, t});
 return shortest - v.x;
 }
};
```

```
P[0].x.x = s;
Q.push({0, 0});
}
int push(int t, Edge x) {
 P.push_back(P[t]);
 if (!P[t].E[0].s or P[t].x.y >= x.y)
 swap(x, P[t].x);
 if (P[t].s) {
 int i = P[t].E[0], j = P[t].E[1];
 int d = P[i].s > P[j].s ? P[i].s : P[j].s;
 int k = push(d ? j : i, x);
 P[t].E[d] = k;
 }
 P[t].s++;
 return t;
}
int nextPath() {
 if (Q.empty()) return -1;
 auto v = Q.top(); Q.pop();
 for (int i : P[v.y].E) if (i)
 Q.push({v.x-P[i].x.y+P[v.y].x.y, i});
 int t = h[P[v.y].x.x];
 if (t) Q.push({v.x - P[t].x.y, t});
 return shortest - v.x;
}
```

## LCA

# 27c09a

**Description:** Computes lowest common ancestor, precomputed in  $O(V \log V)$ ,  $O(\log V)$  per query, uses binary lifting and works for directed and undirected graphs.**Status:** Tested

```
struct LCA {
 vector<vector<int>> T, parent;
 vector<int> depth, vis;
 int LOGN, V;
 // If WA, bigger logn?
 LCA(vector<vector<int>> &T, int logn = 20): T(T), LOGN(logn), vis(T.size()) {
 parent.assign(T.size()-1, vector<int>(LOGN, 0));
 depth.assign(T.size()+1, 0);
 // If is forest
 // for (int u = 0; u < T.size(); u++)
 // if (!vis[u]) dfs(u);
 }
 void dfs(int u = 0, int p = -1) {
 vis[u] = true;
 for (int v : T[u]) {
 if (p != v) {
 depth[v] = depth[u] + 1;
 parent[v][0] = u;
 for (int j = 1; j < LOGN; j++)
 parent[v][j] = parent[parent[v][j-1]][j-1];
 dfs(v, u);
 }
 }
 }
 int query(int u, int v) {
 if (depth[u] < depth[v]) swap(u, v);
 int k = depth[u]-depth[v];
 for (int j = LOGN - 1; j >= 0; j--) {
 if (k & (1 << j))
 u = parent[u][j];
 if (u == v)
 return u;
 for (int j = LOGN - 1; j >= 0; j--) {
 parent[v][j] = parent[parent[v][j-1]][j-1];
 dfs(v, u);
 }
 }
 }
}
```

```

 return parent[u][0];
}
};

Lca Rmq # ceb35b

```

**Description:** Computes lowest common ancestor, precomputed in  $O(V \log V)$ ,  $O(1)$  per query. Rooted in 0

**Status:** Tested on CSES

```

struct lca_rmq {
 using pll = pair<ll, ll>;
 vector<vector<int>> T;
 vector<pll> linear;
 vector<int> in, out, depth;
 ll t = 0, n;
 static pll _min(pll a, pll b) { return min(a, b); }
 sparse_table<pll, lca_rmq::_min> st;
 lca_rmq(){}
 lca_rmq(vector<vector<int>> &T, ll rooted = 0):
n(T.size()) {
 in = out = depth = vector<int>(n, -1);
 linear.resize(2*n);
 auto dfs = [&](int u, int d, auto&&dfs) -> void {
 linear[t] = {d, u}, in[u] = t++, depth[u] = d;
 for (int v : T[u]) {
 if (in[v] != -1) continue;
 dfs(v, d+1, dfs);
 linear[t++] = {d, u};
 }
 out[u] = t;
 };
 dfs(rooted, 0, dfs);
 st = sparse_table<pll, lca_rmq::_min>(linear);
}
ll query(int u, int v) {
 tie(u, v) = {in[u], in[v]};
 return st.query(min(u, v), max(u, v)).second;
}
};

```

HLD # a65b7f

**Author:** Javier Oliva

**Description:** Answer queries in  $O(\log(V) \cdot A \cdot B)$ , where  $A$  is the complexity of merging two chains and  $B$  is the complexity of the data structure query.

**Status:** Partially tested

```

template <class DS, class T, T merge(T, T), int IN_EDGES>
struct heavy_light {
 vector<int> parent, depth, heavy, head, pos_down;
 int n, cur_pos_down; DS ds_down;
 int dfs(int v, vector<vector<int>> adj) {
 const &adj) {
 int size = 1;
 int max_c_size = 0;
 for (int c : adj[v]) {
 if (c != parent[v]) {
 parent[c] = v, depth[c] = depth[v] + 1;
 int c_size = dfs(c, adj);
 size += c_size;
 if (c_size > max_c_size)
 max_c_size = c_size, heavy[v] = c;
 }
 }
 return size;
 }
 void decompose(int v, int h, vector<vector<int>> adj,
 vector<T> &a_down, vector<T> &values) {
 head[v] = h, pos_down[v] = cur_pos_down++;
 a_down[pos_down[v]] = values[v];
 if (heavy[v] != -1)
 decompose(heavy[v], h, adj, a_down, values);
 for (int c : adj[v]) {

```

```

 if (c != parent[v] && c != heavy[v])
 decompose(c, c, adj, a_down, values);
 }
 }
 heavy_light(vector<vector<int>> &adj, vector<T> &values) {
 n = adj.size();
 parent.resize(n);
 depth.resize(n);
 heavy.resize(n, -1);
 head.resize(n);
 pos_down.resize(n);
 vector<T> a_down(n);
 cur_pos_down = 0;
 dfs(0, adj);
 decompose(0, 0, adj, a_down, values);
 ds_down = DS(a_down);
 }
 void update(int a, int b, T x) {
 while (head[a] != head[b]) {
 if (depth[head[a]] < depth[head[b]])
 swap(a, b);
 ds_down.update(pos_down[head[a]], pos_down[a], x);
 a = parent[head[a]];
 }
 if (depth[a] < depth[b])
 swap(a, b);
 if (pos_down[b] + IN_EDGES > pos_down[a])
 return;
 ds_down.update(pos_down[b] + IN_EDGES, pos_down[a], x);
 }
 void update(int a, T x) { ds_down.update(pos_down[a], x); }
 T query(int a, int b) {
 T ans; bool has = 0;
 while (head[a] != head[b]) {
 if (depth[head[a]] < depth[head[b]])
 swap(a, b);
 ans = has ? merge(ans,
ds_down.query(pos_down[head[a]], pos_down[a])) :
ds_down.query(pos_down[head[a]], pos_down[a]);
 has = 1;
 a = parent[head[a]];
 }
 if (depth[a] < depth[b])
 swap(a, b);
 if (pos_down[b] + IN_EDGES > pos_down[a])
 return ans;
 return has ? merge(ans, ds_down.query(pos_down[b] +
IN_EDGES, pos_down[a])) : ds_down.query(pos_down[b] +
IN_EDGES, pos_down[a]);
 }
};

```

Centroid Decomposition # 898938

**Description:** centroid decomposition algorithm

**Status:** Partially tested

```

struct centroid_decomp {
 int n;
 vector<bool> vis;
 vector<int> subtr, parent;
 vector<vector<int>> adj;
 // Optional
 // Here you can insert the functions for traversing the
partition
 // -----
 void dfs(int v, int p = -1){
 for(int u: adj[v]){
 if(u != p && !vis[u]) dfs(u,v);
 }
 }
 // -----

```

```

 int calculate_subtree(int v, int p){
 subtr[v] = 1;
 for (int to : adj[v]){
 if (to == p || vis[to]) continue;
 subtr[v] += calculate_subtree(to,v);
 }
 return subtr[v];
 }
 int get_centroid(int v, int p, int sz){
 for (int to : adj[v]){
 if (to == p || vis[to]) continue;
 if (subtr[to]>2*sz) return
get_centroid(to,v,sz);
 }
 return v;
 }
 int build(int v = 0){
 int centroid =
get_centroid(v,v,calculate_subtree(v,v));
 // Optional
 // Here you can call functions for traversing the
partition
 // -----
 dfs(centroid);
 // -----
 vis[centroid] = 1;
 for (int to : adj[centroid]){
 if (vis[to]) continue;
 build(to);
 }
 // -----
 // Save parent centroid (Optional)
 parent[build(to)] = centroid;
 // -----
 }
 return centroid;
}
centroid_decomp(vector<vector<int>> &adj){
 adj = _adj;
 n = adj.size();
 subtr.resize(n,0);
 parent.resize(n,-1);
 vis.resize(n,0);
 build();
}

```

Connectivity Tarjan # 6a455e

**Description:** Finds bridges, articulation points, and biconnected components in an undirected graph using Tarjan's algorithm.

**Complexity:**  $O(V + E)$

**Status:** Tested on CSES, Codeforces, UVA

```

struct tarjan {
 struct edge { int u, v, comp, bridge; };
 int n, nbc, T;
 vector<vector<int>> adj;
 vector<edge> e;
 vector<int> disc, low, art, st;
 tarjan(int n) : n(n), nbc(0), T(0), adj(n), disc(n, -1),
low(n), art(n, 0) {}
 int add_edge(int u, int v) {
 int i = e.size();
 adj[u].push_back(i);
 adj[v].push_back(i);
 e.push_back({u, v, -1, false});
 return i;
 }
 void dfs(int u, int pe) {
 low[u] = disc[u] = T++;
 for (int i : adj[u]) {

```

```

 if (i == pe) continue;
 int v = e[i].u ^ e[i].v ^ u;
 if (disc[v] < 0) {
 st.push_back(i);
 dfs(v, i);
 if (low[v] > disc[u]) {
 e[i].bridge = true;
 }
 if (low[v] >= disc[u]) {
 art[u]++;
 while (st.back() != i) {
 e[st.back()].comp = nbc;
 st.pop_back();
 }
 e[st.back()].comp = nbc;
 st.pop_back();
 nbc++;
 }
 }
 low[u] = min(low[u], low[v]);
 } else if (disc[v] < disc[u]) {
 st.push_back(i);
 low[u] = min(low[u], disc[v]);
 }
 }
 void solve() {
 for (int i = 0; i < n; i++) {
 if (disc[i] < 0) dfs(i, -1), art[i]--;
 }
 }
 bool is_bridge(int i) { return e[i].bridge; }
 bool is_art(int v) { return art[v] > 0; }
 int component(int i) { return e[i].comp; }
};

```

Tarjan Scc # 1c8aed

**Description:** Finds SCC on a graph

**Complexity:**  $O(V + E)$

**Status:** Tested on CSES

```

struct tarjan {
 int n, timer;
 vector<bool> vis, ist;
 vector<int> tin, low, st, cmp;
 vector<vector<int>> scc;
 void dfs(int u, vector<vector<int>> &adj){
 vis[u] = ist[u] = 1;
 st.push_back(u); tin[u] = low[u] = timer++;
 for (auto v : adj[u]){
 if (!vis[v]){
 dfs(v,adj);
 low[u] = min(low[u],low[v]);
 } else if (ist[v]){
 low[u] = min(low[u],tin[v]);
 }
 }
 if (low[u] == tin[u]){
 scc.emplace_back(); int cur;
 do {
 cur = st.back(); st.pop_back();
 ist[cur] = 0; scc.back().push_back(cur);
 cmp[cur] = (int)scc.size()-1;
 } while (cur != u);
 }
 }
 tarjan(vector<vector<int>> &adj){
 n = adj.size(); tin.resize(n); timer = 0; cmp.resize(n);
 low.resize(n); vis.resize(n,0); ist.resize(n,0);
 for (int i = 0; i < n; i++) if (!vis[i]) dfs(i,adj);
 }
};

```

**Dominator Tree**

# 5a17fa

**Description:** Computes the immediate dominator of every node in a directed graph in  $O((|E|+|V|)\log(|V|))$ .

**Status:** Tested on CSES

```
struct dominator_tree {
 int n,l,s;
 vector<int> tin,par,vis_ord,dsu_mn,dsu_par, sdom,
 rdom,idom;
 vector<bool> vis;
 vector<vector<int>> up;
 vector<pair<int,int>> up_mn;
 void dfs(int u, int p, vector<vector<int>> & adj){
 if (vis[u]) return;
 vis[u] = 1;
 dep[u] = dep[p]+1;
 tin[u] = vis_ord.size();
 vis_ord.push_back(u);
 par[u] = p;
 for (int v : adj[u]) dfs(v,u,adj);
 }
 int find(int x){
 if (dsu_par[x] == -1) return x;
 if (dsu_par[dsu_par[x]] != -1){
 int res = find(dsu_par[x]);
 if (tin[res] < tin[dsu_mn[x]])
 dsu_mn[x] = res;
 dsu_par[x] = dsu_par[dsu_par[x]];
 }
 assert(dsu_mn[x] != INT32_MAX);
 return dsu_mn[x];
 }
 void get_sdom(vector<vector<int>> & rev){
 for (int i = vis_ord.size()-1; i >= 0; i--){
 int c = vis_ord[i];
 if (c == s) continue;
 for (int to : rev[c]){
 int res = find(to);
 if (sdom[c] == -1 || tin[res] <
 tin[sdom[c]])
 sdom[c] = res;
 }
 dsu_par[c] = par[c];
 dsu_mn[c] = sdom[c];
 if (sdom[c] == -1) cerr << c << endl;
 }
 }
 void init_up(){
 for (int i = 0; i < n; i++){
 if (!vis[i]) continue;
 up[i][0] = par[i];
 up_mn[i][0] = {tin[sdom[i]],i};
 }
 for (int i = 1; i < l; i++){
 for (int j = 0; j < n; j++) {
 if (!vis[j]) continue;
 int h = up[j][i-1];
 if (h == -1) {
 up[j][i] = -1;
 up_mn[j][i] = up_mn[j][i-1];
 } else {
 up[j][i] = up[h][i-1];
 up_mn[j][i] = min(up_mn[j][i-1], up_mn[h][i-1]);
 }
 }
 }
 int get_min(int v, int u){
 int dist = dep[u]-dep[v];
 pair<int,int> ret = up_mn[u][0];
 int w = u;
 for (int i = 0; i < l; i++)
 if (dist&(1<i)){
 ret = min(ret,up_mn[w][i]);
 }
 }
 }
}
```

**Dominator Tree, Euler Directed Graph, Undirected Euler Trail, 2Sat**

```
w = up[w][i];
}
assert(w == v);
return ret.second;
}
void get_idom(){
 for (int c : vis_ord){
 if (c == s) continue;
 rdom[c] = get_min(sdom[c],c);
 }
 for (int c : vis_ord)
 if (c != s)
 idom[c] = ((rdom[c] == c)? sdom[c]:idom[rdom[c]]);
}
dominator_tree(vector<vector<int>> & adj,
vector<vector<int>> & rev, int _s = 0, int logn = 20){
 n = adj.size(); l = logn; s = _s;
 tin.resize(n,INT32_MAX);
 par.resize(n,-1);
 vis.resize(n,0);
 dep.resize(n,0);
 dfs(s,-1,adj);
 dsu_par.resize(n,-1);
 dsu_mn.resize(n,INT32_MAX);
 sdom.resize(n,-1);
 rdom.resize(n,-1);
 idom.resize(n,-1);
 get_sdom(rev);
 up.assign(n,vector<int>(logn));
 up_mn.assign(n,vector<pair<int,int>>(logn));
 init_up();
 get_idom();
}
int find(int x){
 if (dsu_par[x] == -1) return x;
 if (dsu_par[dsu_par[x]] != -1){
 int res = find(dsu_par[x]);
 if (tin[res] < tin[dsu_mn[x]])
 dsu_mn[x] = res;
 dsu_par[x] = dsu_par[dsu_par[x]];
 }
 assert(dsu_mn[x] != INT32_MAX);
 return dsu_mn[x];
}
void get_sdom(vector<vector<int>> & rev){
 for (int i = vis_ord.size()-1; i >= 0; i--){
 int c = vis_ord[i];
 if (c == s) continue;
 for (int to : rev[c]){
 int res = find(to);
 if (sdom[c] == -1 || tin[res] <
 tin[sdom[c]])
 sdom[c] = res;
 }
 dsu_par[c] = par[c];
 dsu_mn[c] = sdom[c];
 if (sdom[c] == -1) cerr << c << endl;
 }
};
```

**Euler Directed Graph**

# 2c50f4

**Description:** Find Eulerian path in directed graphs using Hierholzer's algorithm

**Status:** Tested

```
bool euler_directed_graph(vector<vector<int>> &adj,
vector<int> &path){
 path.resize(0);
 int n = adj.size();
 vector<int> indeg(n), outdeg(n);
 int v1 = -1, v2 = -1;
 for (int i = 0; i < n; i++){
 outdeg[i] = adj[i].size();
 for (int x : adj[i])
 indeg[x]++;
 }
 for (int i = 0; i < n; i++){
 int dif = outdeg[i] - indeg[i];
 if (dif == 1){
 if (v1 == -1) v1 = i;
 else return false;
 }
 else if (dif == -1){
 if (v2 == -1) v2 = i;
 else return false;
 }
 else if (dif != 0) return false;
 }
 if (v1 != v2 and (v1 == -1 or v2 == -1)) return false;
 int first = v1;
 if (v1 == -1) adj[v2].push_back(v1);
 else
 for (int i = 0; i < n; i++)
 if (outdeg[i] > 0)
 first = i;
 stack<int> st;
 st.push(first);
 vector<int> res;
```

```
while(!st.empty()){
 int v = st.top();
 if (outdeg[v] == 0){
 res.push_back(v);
 st.pop();
 }
 else{
 st.push(adj[v][0]);
 outdeg[v]--;
 swap(adj[v][0], adj[v][outdeg[v]]);
 }
}
for (int i = 0; i < n; i++)
 if (outdeg[i] != 0)
 return false;
reverse(res.begin(), res.end());
if (v1 != -1){
 for (int i = 0; i + 1 < res.size(); i++){
 if (res[i] == v2 && res[i + 1] == v1){
 vector<int> res2(res.begin() + i + 1, res.end());
 res2.insert(res2.end(), res.begin() + i + 1);
 res = res2;
 break;
 }
 }
}
path = res;
return true;
}
```

**Undirected Euler Trail**

# d56cbe

**Description:** Finds an Eulerian trail/circuit in an undirected graph using Hierholzer's algorithm. Handles self-loops and parallel edges.

- A trail exists iff 0 or 2 vertices have odd degree and the graph is connected.
- Uses XOR trick:  $v \wedge E[e].first \wedge E[e].second$  to traverse edges.

- vertices: vertex sequence of size  $m + 1$ .  
- edges: edge index sequence of size  $m$ .

**Complexity:**  $O(V + E)$

**Status:** Tested on [https://judge.yosupo.jp/problem/eulerian\\_trail\\_undirected](https://judge.yosupo.jp/problem/eulerian_trail_undirected)

```
struct undirected_euler_trail {
 ll n, m = 0;
 vector<vector<ll>> adj;
 vector<pair<ll, ll>> E;
 vector<ll> deg, vertices, edges;
 undirected_euler_trail(ll n) : n(n), adj(n), deg(n) {}

 void add_edge(ll u, ll v) {
 adj[u].push_back(m);
 if (u != v) adj[v].push_back(m);
 E.push_back({u, v});
 deg[u]++, deg[v]++, m++;
 }

 ll find_start() {
 ll s = -1, codd = 0;
 for (ll v = 0; v < n; v++) {
 if (deg[v]&1) codd++;
 else if (deg[v] > 0 && s < 0)
 s = v;
 }
 return (codd == 0 || codd == 2) ? s : -2;
 }

 bool solve() {
 if (m == 0) { vertices = {}; return true; }
```

```
ll s = find_start();
if (s == -2) return false;

vector<ll> ep(n);
edges.resize(m);
vector<bool> alive(m, true);
ll pp = 0, pq = m, v = s;

while (pp < pq) {
 if (ep[v] >= adj[v].size()) {
 if (pp == 0)
 return false;
 edges[-pq] = edges[-pp];
 v = E[edges[pq]].first ^ E[edges[pq]].second;
 } else {
 ll e = adj[v][ep[v]+];
 if (!alive[e])
 continue;
 alive[e] = false;
 edges[pp+] = e;
 v = E[e].first ^ E[e].second;
 }
}
vertices = {s};
for (ll e : edges) {
 s = E[e].first ^ E[e].second;
 vertices.push_back(s);
}
return true;
}
```

**2Sat**

# bc3d8c

**Description:** 2-SAT solver. Negated variables use  $\neg x$ .  $O(N + E)$ .

**Usage:** TwoSat ts(n); ts.either(0, -3); ts.setValue(2); ts.solve();

**Status:** KACTL based, not self tested

```
struct TwoSat {
 int N;
 vector<vector<int>> gr;
 vector<int> values;
 TwoSat(int n = 0) : N(n), gr(2*n) {}
 int addVar(){
 gr.emplace_back(); gr.emplace_back();
 return N++;
 }
 void either(int f, int j) {
 f = max(2*f, -1-2*f);
 j = max(2*j, -1-2*j);
 gr[f].push_back(j^1);
 gr[j].push_back(f^1);
 }
 void setValue(int x) { either(x, x); }
 void addMostOne(const vector<int>& li) {
 if ((int)li.size() <= 1) return;
 int cur = ~li[0];
 for (int i = 2; i < (int)li.size(); i++) {
 int next = addVar();
 either(cur, ~li[i]);
 either(~li[i], next);
 cur = ~next;
 }
 either(cur, ~li[1]);
 }
 vector<int> val, comp, z;
 int Time = 0;
 int dfs(int i) {
 int low = val[i] = ++Time, x; z.push_back(i);
 for (int e : gr[i]) if (!comp[e])
 low = min(low, val[e] ? val[e] : dfs(e));
 if (low == val[i]) do {
 x = z.back(); z.pop_back();
 }
```

```
comp[x] = low;
if (values[x>1] == -1)
 values[x>1] = x&1;
} while (x != i);
return val[i] = low;
}
```

```
bool solve() {
 values.assign(N, -1);
 val.assign(2*N, 0); comp = val;
 for (int i = 0; i < 2*N; i++) if (!comp[i]) dfs(i);
 for (int i = 0; i < N; i++)
 if (comp[2*i] == comp[2*i+1]) return false;
 return true;
};
```

### Fl Matching

# b2efb3

Author: Pablo Messina

Source: <https://github.com/PabloMessina/Competitive-Programming-Material>Description: Flow algorithm with complexity  $O(|E| \cdot |V|^2)$ 

Status: Not tested

```
struct Dinic {
 struct Edge { ll to, rev; ll f, c; };
 ll n, t_; vector<vector<Edge>> G;
 vector<ll> D, q, W;
 bool bfs(ll s, ll t) {
 W.assign(n, 0); D.assign(n, -1); D[s] = 0;
 ll f = 0, l = 0; q[l++] = s;
 while (f < l) {
 ll u = q[f++];
 for (const Edge &e : G[u]) if (D[e.to] == -1 && e.f < e.c)
 D[e.to] = D[u] + 1, q[l++] = e.to;
 }
 return D[t] != -1;
 }
 ll dfs(ll u, ll f) {
 if (u == t_) return f;
 for (ll &i = W[u]; i < (ll)G[u].size(); ++i)
 Edge &e = G[u][i]; ll v = e.to;
 if (e.c <= f || D[v] != D[u] + 1) continue;
 ll df = dfs(v, min(f, e.c - e.f));
 if (df > 0) { e.f += df, G[v][e.rev].f -= df; return
df; }
 }
 return 0;
 }
 Dinic(ll N) : n(N), G(N), D(N), q(N) {}
 void add_edge(ll u, ll v, ll cap) {
 G[u].push_back({v, (ll)G[v].size(), 0, cap});
 G[v].push_back({u, (ll)G[u].size() - 1, 0, 0}); // Use
cap instead of 0 if bidirectional
 }
 ll max_flow(ll s, ll t) {
 t_ = t; ll ans = 0;
 while (bfs(s, t)) while (ll dl = dfs(s, LLONG_MAX)) ans
+= dl;
 return ans;
 }
};
```

Description: Max flow but with lowerbound of flow for each edge. To check if a feasible solution, all edges of super-sink ( $N$ ) node should be saturated

Status: Tested

### Dinic, Mfed, Hopcroft Karp, Hungarian, Min Cost Flow

```
const ll inf = le15;
struct max_flow_edge_demands {
 Dinic mf;
 vector<ll> in, out;
 ll N;
 max_flow_edge_demands(ll N) : N(N), mf(N+2), in(N), out(N)
 {}
 void add_edge(ll u, ll v, ll cap, ll dem = 0) {
 mf.add_edge(u, v, cap - dem);
 out[u] += dem, in[v] += dem;
 }
 ll max_flow(ll s, ll t) {
 mf.add_edge(t, s, inf);
 for (ll i = 0; i < N; i++) {
 mf.add_edge(N, i, in[i]);
 mf.add_edge(i, N+1, out[i]);
 }
 return mf.max_flow(N, N+1);
 }
};
```

### Hopcroft Karp

# 2e26b3

Author: Abner Vidal

Description: Computes maximal matching in  $O(|E| \cdot \sqrt{|V|})$ , faster than Dinic.

Status: Tested on CSES

```
auto [pu, pi] = st.back(); st.pop_back();
int pv = g.edges[g.start[pu]+pi];
l[pu] = pv; r[pv] = pu; ptr[pu] = pi+1;
}
return true;
} else if (d[nxt] == d[u]+1) {
 st.push_back({u, i}); u = nxt; i = ptr[u];
 u_start = g.start[u]; u_end = g.start[u+1];
} else {
 d[u] = INF;
 if (st.empty()) return false;
 auto [pu, pi] = st.back(); st.pop_back();
 u = pu; i = pi+1; ptr[u] = i;
 u_start = g.start[u]; u_end = g.start[u+1];
}
}
int maximum_matching(int want) {
 int match = 0;
 for (int u = 1; u <= n && match < want; u++) {
 if (!l[u]) {
 for (int i = g.start[u]; i < g.start[u+1]; i++) {
 int v = g.edges[i];
 if (!r[v]) { l[u] = v; r[v] = u; match++; break; }
 }
 }
 }
 while (match < want && bfs()) {
 for (int u = 1; u <= n; u++) ptr[u] = 0;
 for (int u = 1; u <= n && match < want; u++)
 if (!l[u] && dfs(u)) match++;
 }
 return match;
}
```

### Hungarian

# 64d965

Description: Solves assignment problem in  $O(n^3)$ . If the matrix is rectangular in  $O(n^2m)$ , where  $n \leq m$ 

Status: Tested

```
void Hungarian(vector<vector<ll>> &A, vector<pair<int, int>>
&result, ll &c, const ll INF = le15) {
 int n = A.size() - 1, m = A[0].size() - 1;
 vector<ll> minv(m + 1), u(n + 1), v(m + 1);
 vector<int> p(m + 1), way(m + 1);
 vector<bool> used(m + 1);
 for (int i = 1; i <= n; ++i) {
 p[0] = i; int j0 = 0;
 for (int j = 0; j <= m; ++j)
 minv[j] = INF;
 for (int j = 0; j <= m; ++j)
 used[j] = false;
 do {
 used[j0] = true;
 int i0 = p[j0], j1;
 ll delta = INF;
 for (int j = 1; j <= m; ++j)
 if (!used[j]) {
 ll cur = A[i0][j] - u[i0] - v[j];
 if (cur < minv[j]) minv[j] = cur, way[j] = j0;
 if (minv[j] < delta) delta = minv[j], j1 = j;
 }
 for (int j = 0; j <= m; ++j) {
 if (used[j]) u[p[j]] += delta, v[j] -= delta;
 else minv[j] -= delta;
 }
 j0 = j1;
 } while (p[j0] != 0);
 do {
 int j1 = way[j0];
 p[j0] = p[j1];
 j0 = j1;
 }
```

```
} while(j0);
}
for (int i = 1; i <= m; ++i)
 result.push_back(make_pair(p[i], i));
C = -v[0];
}
```

### Min Cost Flow

# ebc841

Author: Abner Vidal

Description: Minimum cost flow with complexity  $O(F(|E|V|\log(|E|)))$ 

Status: Tested on CSES

```
template<class T1, class T2>
struct min_cost_flow {
 struct edge { int v; T1 c, f; T2 w; };
 vector<vector<int>> g;
 vector<T2> dist, pot;
 vector<T1> fl;
 vector<bool> vis;
 vector<int> par;
 vector<edge> e;
 min_cost_flow(int n) : g(n), dist(n), pot(n), fl(n), vis(n), par(n) {}
 void add_edge(int u, int v, T1 c, T2 w) {
 int k = e.size();
 e.push_back({v,c,w}); e.push_back({u,c,-w});
 g[u].push_back(k); g[v].push_back(k+1);
 }
 using pli = pair<T2,int>;
 void dijk(int s) {
 fill(dist.begin(),dist.end(),numeric_limits<T2>::max());
 fill(vis.begin(),vis.end(),0); fl[s] = 0;
 numeric_limits<T1>::max();
 priority_queue<pli,vector<pli>,greater<pli> q;
 q.push({0,s}); dist[s] = 0;
 while (!q.empty()) {
 pli p = q.top(); q.pop(); int x = p.second;
 if (vis[x]) continue; vis[x] = 1;
 for (int to : g[x]) {
 T2 d = p.first+e[to].w+pot[x].pot[e[to].v];
 if (!vis[e[to].v] && e[to].f < e[to].c && d < dist[e[to].v]) {
 dist[e[to].v] = d; fl[e[to].v] =
min(fl[x],e[to].c-e[to].f);
 par[e[to].v] = to; q.push({d,e[to].v});
 }
 }
 }
 }
 T2 calc_flow(int s, int t, T1 _f) {
 T2 cost = 0;
 while (1){
 dijk(s);
 if (!vis[t]) return -1;
 for (int i = 0; i < g.size(); i++)
 dist[i] += pot[i].pot[s];
 T1 mnf = min({_f,fl[t]});
 cost += dist[t]*mnf;
 int cur = t;
 while (cur != s){
 e[par[cur]].f += mnf;
 e[par[cur]^1].f -= mnf;
 cur = e[par[cur]^1].v;
 }
 dist.swap(pot);
 _f -= mnf;
 if (!_f) break;
 }
 return cost;
 }
};
```

**General Matching**

# 9bd007

**Description:** General graph matching via randomized Tutte matrix. Fails with probability  $\frac{N}{\text{mod}} \cdot O(N^3)$ .  
**Status:** KACTL based, not self tested

```
const ll MOD = 998244353;
ll pw(ll b, ll e, ll m = MOD) {
 ll r = 1; for (b %= m; e > 0; e >>= 1, b = b*b%m)
 if (e & 1) r = r*b%m; return r;
}
int matInv(vector<vector<ll>>& A) {
 int n = A.size(); vector<int> col(n);
 vector<vector<ll>> tmp(n, vector<ll>(n));
 for (int i = 0; i < n; i++) tmp[i][i] = 1, col[i] = i;
 for (int i = 0; i < n; i++) {
 int r = i, c = i;
 for (int j = i; j < n; j++) for (int k = i; k < n; k++)
 if (A[j][k] & r == j & c == k) goto found;
 return i;
 found:
 swap(A[i], A[r]); swap(tmp[i], tmp[r]);
 for (int j = 0; j < n; j++)
 swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
 swap(col[i], col[c]);
 ll v = pw(A[i][i], MOD - 2);
 for (int j = i+1; j < n; j++) {
 ll f = A[j][i] * v % MOD;
 A[j][i] = 0;
 for (int k = i+1; k < n; k++)
 A[j][k] = (A[j][k] - f*A[i][k]) % MOD;
 for (int k = 0; k < n; k++)
 tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % MOD;
 }
 for (int j = i+1; j < n; j++) A[i][j] = A[i][j]*v%MOD;
 for (int j = 0; j < n; j++) tmp[i][j] = tmp[i][j]*v%MOD;
 A[i][i] = 1;
 }
 for (int i = n-1; i > 0; --i) for (int j = 0; j < i; j++)
 {
 ll v = A[j][i];
 for (int k = 0; k < n; k++)
 tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % MOD;
 }
 for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
 A[col[i]][col[j]] = tmp[i][j]%MOD + (tmp[i][j]<0)*MOD;
 return n;
}
vector<pair<int,int>> generalMatching(int N,
 vector<pair<int,int>>& ed) {
 mt19937 rng(chrono::steady_clock::now()
 .time_since_epoch().count());
 vector<vector<ll>> mat(N, vector<ll>(N)), A;
 for (auto [a, b] : ed) {
 int r = rng() % MOD;
 mat[a][b] = r; mat[b][a] = (MOD - r) % MOD;
 }
 int r = matInv(A = mat), M = 2*N - r, fi, fj;
 if (M != N) do {
 mat.resize(M, vector<ll>(M));
 for (int i = 0; i < N; i++) {
 mat[i].resize(M);
 for (int j = N; j < M; j++) {
 int rv = rng() % MOD;
 mat[i][j] = rv; mat[j][i] = (MOD - rv) % MOD;
 }
 }
 } while (matInv(A = mat) != M);
 vector<int> has(M, 1);
 vector<pair<int,int>> ret;
 for (int it = 0; it < M/2; it++) {
 for (int i = 0; i < M; i++) if (has[i])
 for (int j = i+1; j < M; j++)
 if (A[i][j] && mat[i][j])
 { fi = i; fj = j; goto done; }
 }
 done:
 if (fi < N) ret.emplace_back(fi, fj);
 has[fi] = has[fj] = 0;
 for (int sw = 0; sw < 2; sw++) {
 ll a = pw(A[fi][fj], MOD - 2);
 for (int i = 0; i < M; i++)
 if (has[i] && A[i][fj]) {
 ll b = A[i][fj] * a % MOD;
 for (int j = 0; j < M; j++)
 A[i][j] = (A[i][j] - A[fi][j]*b) % MOD;
 }
 swap(fi, fj);
 }
 return ret;
}
```

**General Matching****General Matching, Global Min Cut, Gomory Hu, Graph Theory Reference, Fast Chu Liu, 4 Cycle**

```
break; done:
if (fj < N) ret.emplace_back(fi, fj);
has[fi] = has[fj] = 0;
for (int sw = 0; sw < 2; sw++) {
 ll a = pw(A[fi][fj], MOD - 2);
 for (int i = 0; i < M; i++)
 if (has[i] && A[i][fj]) {
 ll b = A[i][fj] * a % MOD;
 for (int j = 0; j < M; j++)
 A[i][j] = (A[i][j] - A[fi][j]*b) % MOD;
 }
 swap(fi, fj);
}
return ret;
```

**Global Min Cut**

# 2925a8

**Description:** Global minimum cut in undirected graph (Stoer-Wagner), adjacency matrix.  $O(V^3)$ .  
**Status:** KACTL based, not self tested

```
pair<int, vector<int>> globalMinCut(vector<vector<int>> mat) {
 pair<int, vector<int>> best = {INT_MAX, {}};
 int n = mat.size();
 vector<vector<int>> co(n);
 for (int i = 0; i < n; i++) co[i] = {i};
 for (int ph = 1; ph < n; ph++) {
 vector<int> w = mat[0];
 size_t s = 0, t = 0;
 for (int it = 0; it < n-ph; it++) {
 w[t] = INT_MIN;
 s = t;
 t = max_element(w.begin(), w.end()) - w.begin();
 for (int i = 0; i < n; i++) w[i] += mat[t][i];
 }
 best = min(best, {w[t] - mat[t][t], co[t]});
 co[s].insert(co[s].end(), co[t].begin(), co[t].end());
 for (int i = 0; i < n; i++) mat[s][i] += mat[t][i];
 for (int i = 0; i < n; i++) mat[t][s] = mat[s][i];
 mat[0][t] = INT_MIN;
 }
 return best;
}
```

**Gomory Hu**

# 3a134d

**Description:** Gomory-Hu tree (Gusfield's version). Returns tree edges  $[u, v, cap]$ . Max flow between any pair = min edge on tree path. Requires Dinic with `leftOfMinCut`: after `max_flow(s,t)`,  $D[v] \neq -1$  means  $v$  is on source side.  $O(V)$  flow computations.  
**Status:** KACTL based, not self tested

```
// Add to Dinic struct:
// bool leftOfMinCut(ll v) { return D[v] != -1; }
using Edge3 = array<ll, 3>;
vector<Edge3> gomoryHu(int N,
 vector<array<ll, 3>>& ed) {
 vector<Edge3> tree;
 vector<int> par(N, 0);
 for (int i = 1; i < N; i++) {
 Dinic D(N);
 for (auto& [a,b,c] : ed)
 D.add_edge(a, b, c);
 ll flow = D.max_flow(i, par[i]);
 tree.push_back({i, par[i], flow});
 for (int j = i+1; j < N; j++)
 if (par[j] == par[i] && D.D[j] != -1)
 par[j] = i;
 }
 return tree;
}
```

**Graph Theory Reference**

**Matrix-Tree theorem:** Create  $N \times N$  matrix  $\text{mat}$ . For each edge  $a \rightarrow b \in G$ :  $\text{mat}[a][b]--$ ,  $\text{mat}[b][b]++$  (and  $\text{mat}[b][a]--$ ,  $\text{mat}[a][a]++$  if undirected). Remove  $i$ -th row and column, take determinant: yields #directed spanning trees rooted at  $i$  (if undirected, remove any row/column).

**Erdős–Gallai theorem:** A simple graph with degrees  $d_1 \geq \dots \geq d_n$  exists iff  $d_1 + \dots + d_n$  is even and for every  $k = 1, \dots, n$ :

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

**Fast Chu Liu**

# lab942

**Description:** Minimum spanning arborescence (directed MST), optimized Edmonds/Chu-Liu algorithm using lazy skew heaps  
**Complexity:**  $O(E \log V)$ , returns  $(-1, \emptyset)$  if not reachable  
**Status:** Tested (Fastest Speedrun)

```
template <class T> struct fast_chu_liu {
 struct edge {
 int u, v;
 T len;
 };
 struct node {
 edge key;
 node *l = nullptr, *r = nullptr;
 T delta = 0;
 node(edge e) : key(e) {}
 void prop() {
 key.len += delta;
 if (l) l->delta += delta;
 if (r) r->delta += delta;
 delta = 0;
 }
 edge top() {
 prop();
 return key;
 }
 };
 int n;
 vector<edge> e;
 fast_chu_liu(int n) : n(n) {}
 void add_edge(int x, int y, T w) { e.push_back({x, y, w}); }
 node *merge(node *a, node *b) {
 if (!a || !b)
 return a ? a : b;
 a->prop(), b->prop();
 if (a->key.len > b->key.len)
 swap(a, b);
 swap(a->l, (a->r = merge(b, a->r)));
 return a;
 }
 void pop(node *&a) {
 a->prop();
 a = merge(a->l, a->r);
 }
 pair<T, vector<int>> solve(int root) {
 union_find_rb uf(n);
 vector<node*> heap(n, nullptr);
 for (edge &e : e)
 heap(ed.v) = merge(heap[ed.v], new node(ed));
 T ans = 0;
 vector<int> seen(n, -1), path(n), par(n);
 seen[root] = root;
 vector<edge> Q(n), in(n, {-1, -1, 0});
 deque<tuple<int, int, vector<edge>> cycs;
 for (int s = 0; s < n; s++) {
```

```
int u = s, qi = 0, w;
while (seen[u] < 0) {
 if (!heap[u])
 return {-1, {}};
 edge ed = heap[u]->top();
 heap[u]->delta -= ed.len, pop(heap[u]);
 Q[qi] = ed, path[qi++].u = u, seen[u] = s;
 ans += ed.len, u = uf.findSet(ed.u);
 if (seen[u] == s) {
 node *cyc = nullptr;
 int end = qi, t = (int)uf.ops.size();
 do
 cyc = merge(cyc, heap[w = path[--qi]]);
 while (uf.unionSet(u, w));
 u = uf.findSet(u), heap[u] = cyc, seen[u] = -1;
 cycs.push_front({u, t, {&Q[qi], &Q[qi]}});
 }
}
for (int i = 0; i < qi; i++)
 in.uf.findSet(Q[i].v) = Q[i];
}
```

```
for (auto &[u, t, comp] : cycs) {
 while ((int)uf.ops.size() > t)
 uf.rb();
 edge inEdge = in[u];
 for (auto &ed : comp)
 in.uf.findSet(ed.v) = ed;
 in.uf.findSet(inEdge.v) = inEdge;
 }
 for (int i = 0; i < n; i++)
 par[i] = in[i].u;
 return {ans, par};
}
}
4 Cycle
```

**Description:** Given a simple undirected graph with  $n$  nodes and  $m$  edges with no self loops or multiple edges find the number of 4 length cycles.  
Two cycles are different if their edge collections are different.  
Runs on  $O(m\sqrt{m})$  where  $m$  is the number of edges.  
**Status:** Tested on Codeforces

```
ll count_cycle_4(const vector<vector<ll>> g) {
 ll n = g.size(), w = 0;
 vector<ll> val(n, 0), deg(n);
 vector<vector<ll>> G(n);
 for (ll i = 0; i < n; i++) deg[i] = g[i].size();
 for (ll i = 0; i < n; i++) {
 for (auto e : g[i]) {
 if (e < i) continue;
 if (deg[i] <= deg[e]) G[i].push_back(e);
 else G[e].push_back(i);
 }
 }
 for (ll i = 0; i < n; i++) {
 for (auto u : g[i]) {
 for (auto v : G[u]) {
 if (deg[v] > deg[i] || (deg[v] == deg[i] && v > i))
 w += val[v]++;
 }
 }
 }
 for (auto u : g[i]) {
 for (auto v : G[u]) {
 val[v] = 0;
 }
 }
 return w;
}
```

## 3 Cycle

# a31479

## Tortoise Hare

# b935ee

**Description:** Given a simple undirected graph with  $n$  nodes and  $m$  edges with no self loops or multiple edges find the number of 3 length cycles.

Two cycles are different if their edge collections are different.  
Runs on  $O(m\sqrt{m})$  where  $m$  is the number of edges.

Status: Not tested

```
ll count_cycle_3(const vector<vector<ll>>& g) {
 ll n = g.size(), w = 0;
 vector<ll> val(n, 0), deg(n);
 vector<vector<ll>> G(n);

 for (ll i = 0; i < n; i++) deg[i] = g[i].size();
 for (ll i = 0; i < n; i++) {
 for (auto e : g[i]) {
 if (e < i) continue;
 if (deg[e] <= deg[e]) G[i].push_back(e);
 else G[e].push_back(i);
 }
 }

 for (ll i = 0; i < n; i++) {
 for (auto u : G[i]) val[u] = i;
 for (auto e : g[i]) {
 if (e < i) continue;
 // x = deg[i] + deg[e] + 3;
 for (auto v : G[e])
 if (val == i) {
 w++;
 // w+=x + deg[v] - 2;
 // For number of 3-cycles with or without an extra
 // edge
 }
 }
 }
 return w;
}
```

## Chromatic Number

# 363c8f

**Description:** Computes the chromatic number of a graph in  $O(2^{\lfloor n \rfloor} n)$

Status: Tested

```
int chromatic(vector<int> & adj){
 int n = adj.size();
 if (n == 0) return 0;
 ll mod = 998244353, m = (1<<n);
 vector<ll> ind(m), s(m);
 for (int i = 0; i < m; i++)
 s[i] = ((n - __builtin_popcount(i))&1?-1:1);
 ind[0] = 1;
 for (int i = 1; i < m; i++) {
 int ctz = __builtin_ctz(i);
 ind[i] = (ind[i - (1<<ctz)]+ind[(i - (1<<ctz))&adj[ctz]])%mod;
 }
 for (int j = 1; j < n; j++) {
 ll sum = 0;
 for (int i = 0; i < m; i++) {
 ll cur = (s[i]*ind[i])%mod;
 s[i] = cur;
 sum = (sum+cur)%mod;
 }
 if (sum != 0) return j;
 }
 return n;
}
```

## Data Structures

## Min Deque

# 36a398

**Description:** Get the minimum element of the deque, runs on  $O(1)$  per operation amortized. Can be modified to get the maximum just changing the function

Status: Not tested

```
template<typename T>
struct min_deque {
 min_stack<T> l, r, t;
 void rebalance() {
 bool f = false;
 if (l.empty()) {f = true; l.swap(r);}
 int sz = r.size() / 2;
 while (sz--) {t.push(r.top()); r.pop();}
 while (!r.empty()) {l.push(r.top()); r.pop();}
 if (f) l.swap(r);
 }
 int getmin() {
 if (l.empty()) return r.getmin();
 if (r.empty()) return l.getmin();
 return min(l.getmin(), r.getmin());
 }
 bool empty() {return l.empty() and r.empty();}
 int size() {return l.size() + r.size();}
 void push_front(int x) {l.push(x);}
 void push_back(int x) {r.push(x);}
 void pop_front() {if (l.empty()) rebalance(); l.pop();}
 void pop_back() {if (r.empty()) rebalance(); r.pop();}
 int front() {if (l.empty()) rebalance(); return l.top();}
 int back() {if (r.empty()) rebalance(); return r.top();}
 void swap(min_deque &x) {l.swap(x.l); r.swap(x.r);}
};
```

## Wavelet Tree

# 96d72f

**Description:** Succint data structure to store strings in compressed space. It generalizes rank and select operations. Runs on  $O(\log \sigma)$ , where  $\sigma$  is the size of the alphabet

Status: Tested

```
template<typename T, int MAXLOG>
struct wavelet_tree {
 struct node {
 succinct_indexable_dictionary sid;
 node *ch[2];
 node() = default;
 node(size_t len) : sid(len + 1), ch{nullptr} {}
 };
 node *root;
 node *build(vector<T> &v, vector<T> &rbuf, int bit, int
 l, int r) {
```

```
 if (l >= r or bit == -1) return nullptr;
 node *node = new node(r-l);
 ll mu = 0, right = 0;
 for (ll k = l; k < r; k++) {
 if ((v[k] >> bit) & 1)
 rbuf[right++] = v[k], _node->sid.set(k - l);
 else v[l + left++] = v[k];
 }
 for (int k = 0; k < right; k++) v[l + left + k] =
 rbuf[k];
 _node->sid.build();
 _node->ch[0] = build(v, rbuf, bit - 1, l, l + left);
 _node->ch[1] = build(v, rbuf, bit - 1, l + left, r);
 return _node;
 }

 wavelet_tree() = default;
 wavelet_tree(vector<T> &v) {
 vector<T> rbuf(v.size());
 root = build(v, rbuf, MAXLOG - 1, 0, v.size());
 }
}
```

```
int rank(node *t, int l, int r, const T&x, int level) {
 if (l >= r or t == nullptr) return 0;
 if (level == -1) return r-l;
 bool f = (x >> level) & 1;
 l = t->sid.rank(f, l), r = t->sid.rank(f, r);
 return rank(t->ch[f], l, r, x, level-1);
}
int rank(const T &x, int r) { return rank(root, 0, r, x,
MAXLOG-1); }
T kth(node *t, int l, int r, int k, int level) {
 if (l >= r || t == nullptr) return 0;
 int cnt = t->sid.rank(false, r) - t->sid.rank(false, l);
 bool f = cnt <= k;
 l = t->sid.rank(f, l), r = t->sid.rank(f, r);
 if (f) return kth(t->ch[f], l, r, k - cnt, level - 1) |
(T(1) << level);
 return kth(t->ch[f], l, r, k, level - 1);
}
// k-th(0-indexed) smallest number in v[l,r]
T kth_smallest(int l, int r, int k) {
 return kth(root, l, r+1, k, MAXLOG - 1);
}
// k-th(0-indexed) largest number in v[l,r]
T kth_largest(int l, int r, int k) {
 return kth(l, r, r - l - k);
}
int range_freq(node *t, int l, int r, T upper, int level) {
 if (t == nullptr || l >= r) return 0;
 bool f = ((upper >> level) & 1);
 int ret = 0;
 if (f) ret += t->sid.rank(false, r) - t->sid.rank(false,
l);
 l = t->sid.rank(f, l), r = t->sid.rank(f, r);
 return range_freq(t->ch[f], l, r, upper, level - 1) +
ret;
}
// count i s.t. (l <= i < r) && (v[i] < upper)
int range_freq(int l, int r, T upper) {
 return range_freq(root, l, r, upper, MAXLOG - 1);
}
```

## Succinct Indexable Dictionary

# 597f5a

**Description:** Space-efficient bit vector with rank support  
Status: Tested

```
struct succinct_indexable_dictionary {
 size_t len, blocks;
 vector<unsigned> bit, sum;
 succinct_indexable_dictionary() = default;
 succinct_indexable_dictionary(size_t len) : len(len),
blocks((len + 31)>>5) {
 bit.assign(blocks, 0U);
 }
}
```

```
 sum.assign(blocks, 0U);
}
void set(int k) { bit[k >> 5] |= 1U << (k & 31); }
void build() {
 sum[0] = 0U;
 for (int i = 1; i < blocks; i++)
 sum[i] = sum[i - 1] + __builtin_popcount(bit[i - 1]);
}
bool operator[](int k) { return (bool)((bit[k >> 5] >>
(31)) & 1); }
int rank(int k) {
 return (sum[k >> 5] + __builtin_popcount(bit[k >> 5] &
((1U << (k & 31)) - 1)));
}
int rank(bool val, int k) { return (val ? rank(k) : k -
rank(k)); }
};
```

## Interval Container

# 021bcf

**Description:** Add/remove intervals from a set of disjoint intervals  $[L, R]$ . Merges overlapping intervals on add.  $O(\log N)$ .

**Status:** KACTL based, not self tested

```
using pii = pair<int,int>;
set<pii>::iterator addInterval(set<pii>& is,
 int L, int R) {
 if (L == R) return is.end();
 auto it = is.lower_bound({L, R}), before = it;
 while (it != is.end() && it->first <= R) {
 R = max(R, it->second);
 before = it = is.erase(it);
 }
 if (it != is.begin() && (it->second >= L) {
 L = min(L, it->first);
 R = max(R, it->second);
 is.insert(it, {L, R});
 }
 return is.insert(before, {L, R});
}
void removeInterval(set<pii>& is, int L, int R) {
 if (L == R) return;
 auto it = addInterval(is, L, R);
 auto r2 = it->second;
 if (it->first == L) is.erase(it);
 else (int&)it->second = L;
 if (R != r2) is.emplace(R, r2);
}
```

## Union Find

# 8e1991

**Description:** Finds sets and merges elements, complexity  $O(\alpha(n))$ .  $\alpha(10^{600}) \approx 4$

Status: Highly tested

```
struct union_find {
 vector<int> e;
 union_find(int n) { e.assign(n, -1); }
 int findSet(int x) {
 return (e[x] < 0 ? x : e[x] = findSet(e[x]));
 }
 bool sameSet (int x, int y) { return findSet(x) ==
findSet(y); }
 int size (int x) { return -e[findSet(x)]; }
 bool unionSet (int x, int y) {
 x = findSet(x), y = findSet(y);
 if (x == y) return 0;
 if (e[x] > e[y]) swap(x, y);
 e[x] += e[y], e[y] = x;
 return 1;
 }
}
```

**Union Find Rollback**

# 47ec45

**Description:** Same utility of normal union-find, but you can rollback the last union. Runs on  $O(\alpha(n))$

**Status:** Highly tested

```
struct op{
 int v,u;
 int v_value,u_value;
 op(int _v,int _v_value,int _u,int _u_value):
 v(_v),v_value(_v_value),u(_u),u_value(_u_value) {}
};

struct union_find_rb {
 vector<int> e;
 stack<op> ops;
 int comps;
 union_find_rb(){}
 union_find_rb(int n): comps(n) {e.assign(n, -1);}
 int findSet (int x) {
 return (e[x] < 0 ? x : findSet(e[x]));
 }
 bool sameSet (int x, int y) { return findSet(x) == findSet(y); }
 int size (int x) { return -e[findSet(x)]; }
 void unionSet (int x, int y) {
 x = findSet(x), y = findSet(y);
 if (x == y) return 0;
 if (e[x] > e[y]) swap(x, y);
 ops.push(op(x,e[x],y,e[y]));
 comps--;
 e[x] += e[y], e[y] = x;
 return 1;
 }
 void rb(){
 if(ops.empty()) return;
 op last = ops.top(); ops.pop();
 e[last.v] = last.v_value;
 e[last.u] = last.u_value;
 comps++;
 }
};
```

**Sparse Table**

# 015caa

**Description:** Range queries, precomputed  $O(n \log n)$ , query  $O(1)$ .

- Only supports queries of idempotent functions (min, max, gcd)
- Does not allow updates

**Status:** Highly tested

```
template <typename T, T m_(T, T)>
struct sparse_table {
 int n;
 vector<vector<T>> table;
 sparse_table() {}
 sparse_table(vector<T> &arr) {
 n = arr.size();
 int k = log2_floor(n) + 1;
 table.assign(n, vector<T>(k));
 for (int i = 0; i < n; i++)
 table[i][0] = arr[i];
 for (int j = 1; j < k; j++)
 for (int i = 0; i + (1 << j) <= n; i++)
 table[i][j] = m_(table[i][j-1], table[i+(1<<(j-1))][j-1]);
 }
 T query(int l, int r) {
 int k = log2_floor(r - l + 1);
 return m_(table[l][k], table[r-(1 << k)+1][k]);
 }
 int log2_floor(int n) { return n ? __builtin_clzll(1 - __builtin_clzll(n) : -1; }
};
```

**Disjoint Sparse Table**

# d60d01

**Description:** Range queries, precomputed  $O(n \log n)$ , query  $O(1)$ .

- Only needs associativity
- Does not allow updates

**Status:** Tested on CSES/Codeforces

```
template <typename T, T (*m_)(T, T)>
struct disjoint_sparse_table{
 static const int MAXLOG = 20;
 vector<T> dp[MAXLOG]; vector<bool> has[MAXLOG];
 disjoint_sparse_table() {}
 disjoint_sparse_table(vector<T> &ar){
 int i, c, h, l, n = (int)ar.size();
 for (h = 0; h <= __lg(n); h++){
 dp[h].resize(n + 1), has[h].resize(n+1);
 for (c = l = 1 << h; c < n + l; c += (l << 1)){
 for (i = c + 1; i <= min(n, c + l); i++) {
 if (!has[h][i-1]) dp[h][i] = ar[i-1];
 else dp[h][i] = m_(dp[h][i - 1], ar[i-1]);
 has[h][i] = 1;
 }
 for (i = min(n, c) - 1; i >= c - l; i--) {
 if (!has[h][i+1]) dp[h][i] = ar[i];
 else dp[h][i] = m_(ar[i], dp[h][i+1]);
 has[h][i] = 1;
 }
 }
 }
 T query(int l, int r){
 int h = __lg(l ^ (r + 1));
 if (!has[h][l]) return dp[h][r+1];
 if (!has[h][r+1]) return dp[h][l];
 return m_(dp[h][l], dp[h][r + 1]);
 }
 }
};
```

**Range Query Constant**

# 0328b2

**Description:** Range queries, build  $O(n)$ , query  $O(1)$ , positions  $[0, n - 1]$ , merge must be overlapping friendly, for example min,max,gcd and lcm

**Status:** Tested on CSES

```
template<class T, T merge(T,T)>
struct range_query {
 int n,sz,ld,bl; vector<int> q;
 vector<vector<T>> st; vector<T> arr;
 range_query(vector<T> &a){
 n = a.size(); sz = log2(n);
 bl = (n-sz)/sz; ld = log2(bl); arr = a;
 st.assign(bl, vector<T>(ld+1)); q.resize(n);
 for (int i = 0; i < n; i++){
 st[i/sz][0] = ((i%sz == 0)?a[i]:merge(st[i/sz][0],a[i]));
 int last = a[i]; q[i] = ((i > 0)?q[i-1]:0);
 while (q[i] != 0 && merge(a[i]-last,__builtin_ctz(q[i])),last) == last)
 q[i] ^= (1<<__builtin_ctz(q[i]));
 q[i] = (q[i]<<1); q[i] &= (1<<sz)-1;
 }
 for (int i = 1; i <= ld; i++)
 for (int j = 0; j < bl-(1<<(i-1)); j++)
 st[j][i] = merge(st[j][i-1],st[j+(1<<(i-1))][i-1]);
 }
 T spquery(int l, int r){
 int lsz = log2(r-l+1);
 return merge(st[l][lsz],st[r-(1<<lsz)+1][lsz]);
 }
 T query(int l, int r){
 if (l/sz == r/sz)
 return arr[r-(31-__builtin_clz(q[r]&((1<<(r-1))))-1)]);
 T res; bool hasR = 0;
 int ql = (l-sz+1)/sz, qr = (r/sz)-1;
 if (ql <= qr) res = spquery(ql,qr), hasR = 1;
 int pl = ql*sz-1, pr = (qr+1)*sz;
 if (l <= pl){
 int vl = arr[pl-(31-__builtin_clz(q[pl]&((1<<(pl-1))))-1))];
 res = (hasR?merge(res,vl):vl), hasR = 1;
 }
 if (pr <= r){
 int vr = arr[r-(31-__builtin_clz(q[r]&((1<<(r-pr+1))))-1)]);
 res = (hasR?merge(res,vr):vr), hasR = 1;
 }
 return res;
 }
};
```

```
l+1))-1))));}
T res; bool hasR = 0;
int ql = (l-sz-1)/sz, qr = (r/sz)-1;
if (ql <= qr) res = spquery(ql,qr), hasR = 1;
int pl = ql*sz-1, pr = (qr+1)*sz;
if (l <= pl){
 int vl = arr[pl-(31-__builtin_clz(q[pl]&((1<<(pl-1))))-1))];
 res = (hasR?merge(res,vl):vl), hasR = 1;
}
if (pr <= r){
 int vr = arr[r-(31-__builtin_clz(q[r]&((1<<(r-pr+1))))-1))];
 res = (hasR?merge(res,vr):vr), hasR = 1;
}
return res;
};
```

**Segment Tree**

# 913fb5

**Description:** Range queries, build  $O(n)$ , query and update  $O(\log n)$ , positions  $[0, n - 1]$

**Status:** Highly tested

```
template<class T, T m_(T, T)> struct segment_tree{
 int n; vector<T> ST;
 segment_tree(){}
 segment_tree(vector<T> &a){
 n = a.size(); ST.resize(n << 1);
 for (int i=n;i<(n<<1);i++)ST[i]=a[i-n];
 for (int i=n-1;i>0;i--)ST[i]=m_(ST[i<<1],ST[i<<1|1]);
 }
 void update(int pos, T val){ // replace with val
 ST[pos += n] = val;
 for (pos >= 1; pos > 0; pos >>= 1)
 ST[pos] = m_(ST[pos<<1], ST[pos<<1|1]);
 }
 T query(int l, int r){ // [l, r]
 T ansL,ansR; bool hasL = 0, hasR = 0;
 for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
 if (l & 1)
 ansL=(hasL?m_(ansL,ST[l++]):ST[l++]),hasL=1;
 if (r & 1)
 ansR=(hasR?m_(ST[-r],ansR):ST[-r]),hasR=1;
 if (!hasL) return ansR; if (!hasR) return ansL;
 return m_(ansL,ansR);
 }
 }
};
```

**Segment Tree With Walk**

# 026166

**Description:** Extends segment\_tree with walk operation. Returns first index  $i \geq l$  where  $ST[i] \geq k$ , or  $-1$  if none exists. Walk  $O(\log n)$ .

**Status:** Partially tested

```
template<class T, T m_(T, T)> struct segment_tree_with_walk : segment_tree<T,m_> {
 int walk(int l, T k) {
 vector<int> L, R;
 int r = this->n - 1;
 for (l += this->n - 1; l < r; l >>= 1, r >>= 1) {
 if (l & 1) L.push_back(l++);
 if (r & 1) R.push_back(--r);
 }
 reverse(R.begin(), R.end());
 L.insert(L.end(), R.begin(), R.end());
 for (int v : L) {
 if (this->ST[v] >= k) {
 while (v < this->n) v = (this->ST[v << 1] >= k) ? (v << 1) : (v << 1 | 1);
 }
 }
 }
};
```

```
return v - this->n;
}
}
return -1;
}
};

Segment Tree Lazy
```

# 0fe589

**Description:** Segment tree but with range updates, build  $O(n)$ , query and update  $O(\log n)$

**Status:** Highly tested

```
template<
 class T1, // answer value stored on nodes
 class T2, // lazy update value stored on nodes
 T1 merge(T1, T1),
 void pushUp(T2&, /*parent*/, T2& /*child*/, int, int, int,
 int), // push update value from a node to another. parent -> child
 void applyUp(T2&, T1&, int, int) // apply the update value of a node to its answer value. upd -> ans
>
struct segment_tree_lazy{
 vector<T1> ST; vector<T2> lazy; vector<bool> upd;
 int n;
 void build(int i, int l, int r, vector<T1>&values){
 if (l == r){
 ST[i] = values[l];
 return;
 }
 build(i << 1, l, (l + r) >> 1, values);
 build(i << 1 | 1, (l + r) / 2 + 1, r, values);
 ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
 }
 segment_tree_lazy() {}
 segment_tree_lazy(vector<T1>&values){
 n = values.size(); ST.resize(n << 2 | 3);
 lazy.resize(n << 2 | 3); upd.resize(n << 2 | 3, false);
 build(1, 0, n - 1, values);
 }
 void push(int i, int l, int r){
 if (upd[i]){
 applyUp(lazy[i], ST[i], l, r);
 if (l != r){
 pushUp(lazy[i], lazy[i << 1], l, r, l, (l + r) / 2 + 1, r);
 }
 upd[i] = false;
 lazy[i] = T2();
 }
 }
 void update(int i, int l, int r, int a, int b, T2 &u){
 if (l >= a and r <= b){
 pushUp(u, lazy[i], a, b, l, r);
 upd[i] = true;
 }
 push(i, l, r);
 if (l > b or r < a) return;
 if (l >= a and r <= b) return;
 update(i << 1, l, (l + r) >> 1, a, b, u);
 update(i << 1 | 1, (l + r) / 2 + 1, r, a, b, u);
 ST[i] = merge(ST[i << 1], ST[i << 1 | 1]);
 }
 void update(int a, int b, T2 u){
 if (a > b){
 update(0, b, u);
 update(a, n - 1, u);
 return;
 }
 update(1, 0, n - 1, a, b, u);
 }
};
```

```

 }
T1 query(int i, int l, int r, int a, int b){
 push(i, l, r);
 if (a <= l and r <= b)
 return ST[i];
 int mid = (l + r) >> 1;
 if (mid < a)
 return query(i << 1 | 1, mid + 1, r, a, b);
 if (mid > b)
 return query(i << 1, l, mid, a, b);
 return merge(query(i << 1, l, mid, a, b), query(i << 1 |
1, mid + 1, r, a, b));
}
T1 query(int a, int b){
 if (a > b) return merge(query(a, n - 1), query(0, b));
 return query(1, 0, n - 1, a, b);
}
}

Iterative Segment Tree Lazy # 8fa047

```

**Description:** Segment tree with lazy propagation, but now iterative. Build is  $O(n)$ , queries and updates are  $O(\log n)$

**Status:** Tested

```

template<class T1, class T2> T1 merge(T1&, T1&), void
applyUpd(T1&, T2&, int), void pushUpd(T2&, T2&, int)>
struct segment_tree_lazy {
 int n; vector<T1> st; vector<T2> lazy;
 segment_tree_lazy(vector<T1> & a){
 n = a.size(); h = sizeof(int)*8-*builtin_clz(n);
 st.resize(n<<1); lazy.resize(n);
 for (int i = 0; i < n; i++) st[n+i] = a[i];
 for (int i = n-1; i > 0; i--) st[i] =
merge(st[i<<1],st[i<<1|1]);
 }
 void calc(int pos, int sz){
 st[pos] = merge(st[pos<<1],st[pos<<1|1]);
 if (lazy[pos].upd()) applyUpd(st[pos],lazy[pos],sz);
 }
 void apply(int pos, T2 val, int sz){
 applyUpd(st[pos],val,sz);
 if (pos < n) pushUpd(val,lazy[pos],sz);
 }
 void push(int l, int r){
 int s = h, sz = 1<<(h-1);
 for (l += n, r += n-1; s > 0; s--, sz >>= 1)
 for (int i = l>s; i <= r>s; i++) if
(lazy[i].upd()){
 apply(i<<1, lazy[i],sz);
 apply(i<<1|1, lazy[i],sz);
 lazy[i] = T2();
 }
 }
 void update(int l, int r, T2 val){
 push(l,l+1); push(r,r+1);
 bool cl = 0, cr = 0; int sz = 1;
 for (l += n, r += n+1; l < r; l >>= 1, r >>= 1, sz
<<= 1){
 if (cl) calc(l-1,sz);
 if (cr) calc(r,sz);
 if (l&1) apply(l++,val,sz), cl = 1;
 if (r&1) apply(--r,val,sz), cr = 1;
 }
 for (-l; r > 0; l >>= 1, r >>= 1, sz <= 1){
 if (cl) calc(l,sz);
 if (cr && (!cl || l != r)) calc(r,sz);
 }
 }
 T1 query(int l, int r){
 push(l,l+1); push(r,r+1);
 T1 ansL, ansR; bool hasL = 0,hasR = 0;
 for (l += n, r += n+1; l < r; l >>= 1, r >>= 1){
 if (l&1) ansL = (hasL?merge(ansL,st[l++]):st[l+
++]), hasL = 1;

```

## Iterative Segment Tree Lazy, Persistent Segment Tree, Dynamic Segment Tree, Query Tree, BIT

```

 if (r&1) ansR = (hasR?merge(st[--r],ansR):st[--r]),
 hasR = 1;
 }
 if (!hasL) return ansR; if (!hasR) return ansL;
 return merge(ansL,ansR);
 }
 /* example of use */
 struct lz {
 ll val;
 lz(){ val = 0; }
 bool upd(){ return (val != 0); }
 };
 ll merge(ll & a, ll & b){
 return a+b;
 }
 void applyUpd(ll & v, lz & u, int sz){
 v += u.val*((ll)sz);
 }
 void pushUpd(lz & u1, lz & u2, int sz){
 u2.val += u1.val;
 }
}

Persistent Segment Tree # 887787

```

**Description:** Segment tree but saves a new version of the tree for each update, build  $O(n)$ , query  $O(\log n)$ , new nodes for each update  $O(\log n)$

**Status:** Tested on CSES

```

template<class T, T _m(T, T)>
struct persistent_segment_tree {
 vector<T> ST;
 vector<int> L, R;
 int n, rt;
 persistent_segment_tree(int n): ST(1, T()), L(1, 0), R(1,
0), n(n), rt(0) {}
 int new_node(T v, int l = 0, int r = 0) {
 int ks = ST.size();
 ST.push_back(v); L.push_back(l); R.push_back(r);
 return ks;
 }
 int update(int k, int l, int r, int p, T v) {
 int ks = new_node(ST[k], L[k], R[k]);
 if (l == r) {
 ST[ks] = v; return ks;
 }
 int m = (l + r) / 2, ps;
 if (p <= m) {
 ps = update(L[ks], l, m, p, v);
 L[ks] = ps;
 } else {
 ps = update(R[ks], m + 1, r, p, v);
 R[ks] = ps;
 }
 ST[ks] = _m(ST[L[ks]], ST[R[ks]]);
 return ks;
 }
 T query(int k, int l, int r, int a, int b) {
 if (l >= a and r <= b)
 return ST[k];
 int m = (l + r) / 2;
 if (b <= m)
 return query(L[k], l, m, a, b) :
init(l, mid);
 else if (a > mid)
 return query(R[i], mid + 1, r, a, b) :
init(mid + 1, r);
 if (L[i] == 0) L[i] = addNode(l, mid);
 if (R[i] == 0) R[i] = addNode(mid + 1, r);
 return merge(query(L[i], l, mid, a, b), query(R[i], mid
+ 1, r, a, b));
 }
 T query(MAXi a, MAXi b) {
 return query(0, 0, n - 1, a, b);
 }
 void update(int i, MAXi l, MAXi r, MAXi p, T v) {
 if (l == r){
 ST[i] = v; return;
 }
 MAXi mid = (l + r) / 2LL;
 if (p <= mid)
 update(L[i] != 0 ? L[i] : L[i] = addNode(l, mid), l,
mid, p, v);
 else
 update(R[i] != 0 ? R[i] : R[i] = addNode(mid + 1, r),
mid + 1, r, p, v);
 ST[i] = merge(
 L[i] != 0 ? ST[L[i]] : init(l, mid),
 R[i] != 0 ? ST[R[i]] : init(mid + 1, r)
);
 }
 void update(int p, T v) {
 update(0, 0, n - 1, p, v);
 }
 int update(int p, T v) {
 return update(rt, p, v);
 }
}

Persistent Segment Tree # 887787

```

```

 T query(int k, int a, int b) {
 return query(k, 0, n - 1, a, b);
 }
};

Dynamic Segment Tree # df5ff1

```

**Description:** The nodes are created when the same value is updated, enabling queries in intervals like  $[0, 10^{12}]$ . Overall complexity is  $O(Q \log n)$

**Status:** Not tested

```

template<class T, class MAXi, class Maxi, int sz>
T merge(T, T), // Data type of the nodes
MAXi n, int n_count; // Data type of the ranges (ll or
i128)
T merge(T, T), // Merge
T init(MAXi, MAXi) // Default value for [a, b]
>
struct dynamic_segment_tree {
 vector<T> ST; vector<int> L, R;
 MAXi n; int n_count;
 dynamic_segment_tree (MAXi n, int r) :
n(n), n_count(1), L(1), R(1), ST(1){
 ST.reserve(r);
 L.reserve(r);
 R.reserve(r);
 ST[0] = init(0, n - 1);
}
int addNode(MAXi l, MAXi r){
 L.push_back(0);
 R.push_back(0);
 ST.push_back(init(l, r));
 return n_count++;
}
T query(int i, MAXi l, MAXi r, MAXi a, MAXi b) {
 if (a <= l and r <= b)
 return ST[i];
 MAXi mid = ((l + r) >> 1LL);
 if (b <= mid)
 return query(L[i], l, mid, a, b) :
init(l, mid);
 else if (a > mid)
 return query(R[i], mid + 1, r, a, b) :
init(mid + 1, r);
 if (L[i] == 0) L[i] = addNode(l, mid);
 if (R[i] == 0) R[i] = addNode(mid + 1, r);
 return merge(query(L[i], l, mid, a, b), query(R[i], mid
+ 1, r, a, b));
}
T query(MAXi a, MAXi b) {
 return query(0, 0, n - 1, a, b);
}
void update(int i, MAXi l, MAXi r, MAXi p, T v) {
 if (l == r){
 ST[i] = v; return;
 }
 MAXi mid = (l + r) / 2LL;
 if (p <= mid)
 update(L[i] != 0 ? L[i] : L[i] = addNode(l, mid), l,
mid, p, v);
 else
 update(R[i] != 0 ? R[i] : R[i] = addNode(mid + 1, r),
mid + 1, r, p, v);
 ST[i] = merge(
 L[i] != 0 ? ST[L[i]] : init(l, mid),
 R[i] != 0 ? ST[R[i]] : init(mid + 1, r)
);
}
void update(int p, T v) {
 update(0, 0, n - 1, p, v);
}
int update(int p, T v) {
 return update(rt, p, v);
}

Dynamic Segment Tree # df5ff1

```

## Query Tree

# ec393d

**Description:** Allow us to solve dynamic connectivity offline

**Status:** Tested

```

struct query{
 int v,u;
 bool status;
 query(int _v,int _u) : v(_v),u(_u) {};
};
struct query_tree{
 vector<vector<query>> tree;
 int size;
 // rollback structure
 UnionFindRB uf;
 query_tree(int _size,int n) : size(_size) {uf =
UnionFindRB(n); tree.resize(4*_size + 4);}
 void addTree(int v,int l,int r,int ul,int ur,query& q){
 if(ul > ur) return;
 if(l == ul && ur == r){tree[v].push_back(q); return; }
 int mid = (l + r) / 2;
 addTree(2*v,l,mid,ul,min(ur,mid),q);
 addTree(2*v + 1,mid + 1,r,max(ul,mid + 1),ur,q);
 }
 void add(query q,int l,int r){addTree(1,0,size -
1,l,r,q);}
 void dfs(int v,int l,int r,vector<int> &ans){
 // change in data structure
 for(query &q: tree[v]) q.status = uf.unionSet(q.v,q.u);
 if(l == r) ans[l] = uf.comps;
 else{
 int mid = (l + r) / 2;
 dfs(2*v,l,mid,ans);
 dfs(2*v + 1,mid + 1,r,ans);
 }
 // rollback in data structure
 for(query q: tree[v]) if(q.status) uf.rb();
 }
 vector<int> getAns(){
 vector<int> ans(size);
 dfs(1,0,size - 1,ans);
 return ans;
 }
};

BIT # b586d7

```

**Description:** Range queries and updates, precomputed  $O(n \log n)$ , query/update  $O(\log n)$

- Sum operation can be changed to xor

**Status:** Tested

```

struct fenwick_tree {
 vector<ll> bit; int n;
 fenwick_tree(int n): n(n) { bit.assign(n, 0); }
 fenwick_tree(vector<ll> &a): fenwick_tree(a.size()) {
 for (size_t i = 0; i < a.size(); i++)
 add(i, a[i]);
 }
 ll sum(int r) {
 ll ret = 0;
 for (; r >= 0; r = (r & (r + 1)) - 1)
 ret += bit[r];
 return ret;
 }
 ll sum(int l, int r) {
 return sum(r) - sum(l - 1);
 }
 void add(int idx, ll delta) {
 for (; idx < n; idx = idx | (idx + 1))
 bit[idx] += delta;
 }
};

BIT # b586d7

```

**Bit 2D**

# a9206c

**Description:** 2D Range queries, Runs on  $O(\log n \cdot \log m)$  per operation  
**Status:** Tested

```
struct fenwick_tree_2d {
 int N, M;
 vector<vector<ll>> BIT;
 fenwick_tree_2d(int N, int M) : N(N), M(M) {
 BIT.assign(N + 1, vector<ll>(M + 1, 0));
 }
 void update(int x, int y, ll v) {
 for (int i = x; i <= N; i += (i & -i))
 for (int j = y; j <= M; j += (j & -j))
 BIT[i][j] += v;
 }
 ll sum(int x, int y) {
 ll s = 0;
 for (int i = x; i > 0; i -= (i & -i))
 for (int j = y; j > 0; j -= (j & -j))
 s += BIT[i][j];
 return s;
 }
 ll query(int x1, int y1, int x2, int y2) {
 return sum(x2, y2) - sum(x2, y1 - 1) - sum(x1 - 1, y2) +
 sum(x1 - 1, y1 - 1);
 }
};
```

**Merge Sort Tree**

# 4d309e

**Description:** Iterative merge sort tree. Each node stores sorted elements in its range.

- Query: Count elements  $>$  k in range  $[i, j]$ .  $O(\log^2 n)$  per operation.
- Build:  $O(n \log n)$

**Status:** Distinct Values Queries (CSES)

```
template <typename T>
struct merge_sort_tree {
 int N; vector<vector<T>> ST;
 merge_sort_tree() {}
 merge_sort_tree(vector<T> &vs) {
 N = vs.size(); ST.resize(2 * N);
 for (int i = 0; i < N; i++) ST[i + N] = {vs[i]};
 for (int i = N - 1; i > 0; i--)
 merge(ST[i < 1].begin(), ST[i < 1].end(), ST[i < 1]);
 1).begin(), ST[i < 1].end(), back_inserter(ST[i]));
 }
 int query(int i, int j, int k) {
 int res = 0;
 for (int i += N, j += N + 1; i < j; i >= 1, j >= 1) {
 if (i & 1) res += ST[i].size() -
 (upper_bound(ST[i].begin(), ST[i].end(), k) -
 ST[i].begin()), i++;
 if (j & 1) res += ST[j].size() -
 (upper_bound(ST[j].begin(), ST[j].end(), k) -
 ST[j].begin()));
 }
 return res;
 }
};
```

**Line Container**

# c89a68

**Description:** Line container for convex hull trick optimization in DP  
**Status:** Tested

```
const ll INF = 1e15;
struct Line {
 mutable ll a, b, c;
}
```

**Bit 2D, Merge Sort Tree, Line Container, Kd Tree, MO, Mo Rollback**

```
bool operator<(Line r) const {
 return a < r.a;
}
bool operator<(ll x) const {
 return c < x;
}
// dynamically insert `a*x + b` lines and query for maximum
// at any x all operations have complexity O(log N)
struct LineContainer: multiset < Line, less > {
 ll div(ll a, ll b) {
 return a / b - ((a ^ b) < 0 && a % b);
 }
 bool isect(iterator x, iterator y) {
 if (y == end()) return x -> c = INF, 0;
 if (x -> a == y -> a) x -> c = x -> b > y -> b ? INF : -
 INF;
 else x -> c = div(y -> b - x -> b, x -> a - y -> a);
 return x -> c >= y -> c;
 }
 void add(ll a, ll b) {
 // a *= -1, b *= -1 // for min
 auto z = insert({
 a,
 b,
 0
 }), y = z++, x = y;
 while (isect(y, z)) z = erase(z);
 if (x != begin() && isect(--x, y)) isect(x, y) =
 erase(y);
 while ((y = x) != begin() && (--x) -> c >= y -> c)
 isect(x, erase(y));
 }
 ll query(ll x) {
 if (empty()) return -INF; // INF for min
 auto l = *lower_bound(x);
 return l.a * x + l.b;
 // return -l.a * x - l.b; // for min
 }
};
```

**Kd Tree**

# c0b721

**Description:** given an array of  $n$  points, it answers the minimum manhattan distance from the query point in  $O(3^d \log(n))$

**Status:** Tested on CSES

```
template<class T, int d>
struct pt {
 array<T, d> p;
 bool operator!=(const pt &o) const { return p != o.p; }
 T dist(const pt &o) const {
 T res = 0;
 for (int i = 0; i < d; i++)
 res += abs(p[i] - o.p[i]);
 return res;
 }
};
template<class pt, class T, int d>
struct kd_tree {
 struct node {
 int ax; T val; pt p;
 node *izq, *der;
 array<T, d> mn, mx;
 };
 node *root;
 T mn_dist(const pt &q, const array<T, d> &mnb, const
array<T, d> &mxb) {
 T dist = 0;
 for (int i = 0; i < d; i++) {
 if (q.p[i] < mnb[i]) dist += mnb[i] - q.p[i];
 else if (q.p[i] > mxb[i]) dist += q.p[i] - mxb[i];
 }
 }
};
```

```
 return dist;
}
node *build_tree(vector<pt> &a, int l, int r) {
 if (l > r) return nullptr;
 node *cur = new node();
 array<T, d> minb, maxb;
 for (int i = 0; i < d; i++) {
 minb[i] = a[l].p[i];
 maxb[i] = a[l].p[i];
 }
 for (int j = l + 1; j <= r; j++) {
 for (int i = 0; i < d; i++) {
 if (a[j].p[i] < minb[i]) minb[i] =
 a[j].p[i];
 if (a[j].p[i] > maxb[i]) maxb[i] =
 a[j].p[i];
 }
 }
 cur->mn = minb; cur->mx = maxb; int ax = 0;
 T mxld = 0;
 for (int i = 0; i < d; i++) {
 T ld = maxb[i] - minb[i];
 if (ld > mxld) {
 mxld = ld;
 ax = i;
 }
 }
 cur->ax = ax; int mid = (l+r)/2;
 nth_element(a.begin() + l, a.begin() + mid,
 a.begin() + r + 1, [ax](const pt &x, const pt &y) {
 return x.p[ax] < y.p[ax];
 });
 cur->p = a[mid]; cur->val = a[mid].p[ax];
 cur->izq = build_tree(a, l, mid - 1);
 cur->der = build_tree(a, mid + 1, r);
 return cur;
}
kd_tree(vector<pt> &a) {
 root = build_tree(a, 0, a.size() - 1);
}
void query(node *cur, const pt &q, T &ans) {
 if (cur == nullptr) return;
 if (q != cur->p) {
 T d_here = q.dist(cur->p);
 if (d_here < ans)
 ans = d_here;
 } else {
 ans = min(ans, (T)0);
 }
 vector<pair<T, node *>> ch;
 if (cur->izq) {
 T d_izq = mn_dist(q, cur->izq->mn, cur->izq-
 >mx);
 ch.push_back({d_izq, cur->izq});
 }
 if (cur->der) {
 T d_der = mn_dist(q, cur->der->mn, cur->der-
 >mx);
 ch.push_back({d_der, cur->der});
 }
 sort(ch.begin(), ch.end(), [](){const auto &a, const
auto &b {
 return a.first < b.first;
 }});
 for (auto &d_val, to : ch)
 if (d_val < ans) query(to, q, ans);
}
T query(const pt &q) {
 T ans = numeric_limits<T>::max();
 query(root, q, ans);
 return ans;
}
```

**MO**

# 2c642f

**Description:** Allow us to answer queries offline with sqrt decomposition  
**Status:** Tested

```
struct query {
 ll l, r, i;
};
template<class T, typename U, class T2>
struct mo_algorithm {
 vector<U> ans;
 template<typename... Args>
 mo_algorithm(vector<T> &v, vector<query> &queries, Args...
args) {
 T2 ds(args...);
 ll block_sz = sqrt(v.size());
 ans.assign(queries.size(), -1);
 sort(queries.begin(), queries.end(), [&](auto &a, auto
&b) {
 return a.l/block_sz != b.l/block_sz ? a.l/block_sz <
b.l/block_sz : a.r < b.r;
 });
 ll l = 0, r = -1;
 for (query q : queries) {
 while (l > q.l) ds.add(v[--l]);
 while (r < q.r) ds.add(v[+r]);
 while (l < q.l) ds.remove(v[l++]);
 while (r > q.r) ds.remove(v[r--]);
 ans[q.i] = ds.answer(q);
 }
 }
};
```

**Mo Rollback**

# 69da82

**Description:** Allow us to answer queries offline with sqrt decomposition  
**Status:** Tested

```
struct query {
 ll l, r, i;
};
template<typename T, typename U, class T2>
struct rollback_mo_algorithm {
 vector<U> ans;
 template<typename... Args>
 rollback_mo_algorithm(vector<T> &v, vector<query> &queries, Args...
args) {
 ll block_sz = sqrt(v.size());
 ans.resize(queries.size());
 T2 ds(args...);
 sort(queries.begin(), queries.end(), [&](auto &a, auto
&b) {
 return a.l/block_sz != b.l/block_sz ? a.l/block_sz <
b.l/block_sz : a.r < b.r;
 });
 ll t = 0, last, border, last_block = -1, block;
 for (ll i = 0; i < queries.size(); i++) {
 block = queries[i].l / block_sz;
 if (block == queries[i].r / block_sz) {
 T2 tmp(args...);
 for (ll k = queries[i].l; k <= queries[i].r; k++)
 tmp.add(v[k]);
 ans[queries[i].i] = tmp.answer(queries[i]);
 continue;
 }
 if (last_block != block) {
 ds = T2(args...);
 border = block_sz * (block + 1);
 last = border;
 }
 last_block = block;
 }
 }
};
```

```

 for(ll k = last + 1; k <= queries[i].r; k++)
 ds.add(v[k]);
 t = ds.snapshot();
 for(ll k = queries[i].l; k <= border; k++)
 ds.add(v[k]);
 ans[queries[i].i] = ds.answer(queries[i]);
 ds.rollback(t);
 last = queries[i].r;
}
};

Mo Update # 7b9b48

```

**Description:** Allow us to answer queries and updates offline with block decomposition in  $O(q \cdot n^{\frac{2}{3}})$

**Status:** Not tested

```

struct query {
 int l,r,t,i;
};
template<class T, typename U, class T2>
struct mo_update {
 vector<U> ans;
 mo_update(vector<T> &v, vector<query> &queries,
 vector<pair<int,T>> &updates, T2 &ds) {
 ds.init(v,updates);
 int block_sz = cbtrt(2*v.size()*v.size());
 ans.assign(queries.size(), 1);
 sort(queries.begin(), queries.end(), [&](auto a, auto b) {
 if (a.l/block_sz == b.l/block_sz){
 if (a.r/block_sz == b.r/block_sz) return a.t < b.t;
 return a.r/block_sz < b.r/block_sz;
 }
 return a.l/block_sz < b.l/block_sz;
 });
 int l = 0, r = -1;
 for (query q : queries) {
 while (ds.t < q.t) ds.update(l,r);
 while (ds.t > q.t) ds.rollback(l,r);
 while (l > q.l) ds.add(-l);
 while (r < q.r) ds.add(r);
 while (l < q.l) ds.remove(l--);
 while (r > q.r) ds.remove(r--);
 ans[q.i] = ds.answer(q);
 }
 }
};

P 1 1 Trees Ordered Set # 862f9a

```

**Description:** Built-in version of C++ of select and rank operations in sets

**Status:** Tested

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using oset = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;

```

Treap # c949ed

**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.

**Status:** Tested

```
const ll INF = 1e15;
struct treap {
 static mt19937_64 MT;
 node *root;
 vector<node*> v;
 int sz() { return v.size(); }
 void add(int k, ll v) { v.push_back({k, v}); }
 void update() {
 sort(v.begin(), v.end());
 root = merge(v.begin(), v.end());
 }
 node* find(int k) {
 auto it = lower_bound(v.begin(), v.end(), {k, -1});
 if (it == v.end() || it->k != k) return NULL;
 return it->r;
 }
 void add(int k, ll v) {
 auto it = lower_bound(v.begin(), v.end(), {k, -1});
 if (it == v.end() || it->k != k) v.push_back({k, v});
 else it->r = merge(it, it+1);
 }
 void update() {
 sort(v.begin(), v.end());
 root = merge(v.begin(), v.end());
 }
 node* merge(node* L, node* R) {
 if (!L) return R;
 if (!R) return L;
 if (L->k < R->k) {
 L->r = merge(L->r, R);
 return L;
 } else {
 R->l = merge(L, R->l);
 return R;
 }
 }
 void remove(int k) {
 auto it = lower_bound(v.begin(), v.end(), {k, -1});
 if (it == v.end() || it->k != k) return;
 if (it == v.begin() && it->r == NULL) v.erase(it);
 else if (it == v.end() || it->r == NULL) v.erase(it-1);
 else {
 it->r = merge(it+1, it->r);
 v.erase(it);
 }
 }
 int rank(int k) {
 auto it = lower_bound(v.begin(), v.end(), {k, -1});
 if (it == v.end() || it->k != k) return 0;
 return it - v.begin();
 }
 void print() {
 cout << "Treap: ";
 for (auto n : v) cout << n.k << " ";
 cout << endl;
 }
};

Treap ans;

```

```

struct node {
 node *l, *r;
 ll key, priority, value, max_value;
 max_value = value = v;
};

ll value(node* T) { return T ? T->value : -INF; }
ll max_value(node* T) { return T ? T->max_value : -INF; }
void update(node* T) {
 T->max_value = max({T->value, max_value(T->l),
 max_value(T->r)});
}

node *root;
void merge(node* &T, node* T1, node* T2) {
 if (T1 == NULL) { T = T2; return; }
 if (T2 == NULL) { T = T1; return; }
 if (T1->priority > T2->priority)
 merge(T1->r, T1->l, T2), T = T1;
 else merge(T2->l, T1, T2->r), T = T2;
 return update(T);
}

void merge(node* &T, node* T1, node* T2, node* T3) {
 merge(T, T1, T2); merge(T, T, T3);
}

void split(node* T, ll x, node* &T1, node* &T2) {
 if (T == NULL) { T1 = T2 = NULL; return; }
 if (T->key <= x) { split(T->r, x, T->l, T2); T1 = T;
 else { split(T->l, x, T1, T->r); T2 = T; }
 return update(T);
 }

 void split(node* T, ll x, ll y, node* &T1, node* &T2,
 node* &T3) {
 split(T, x-1, T1, T2); split(T2, y, T3, T3);
 }

 bool search(node* T, ll x) {
 if (T == NULL) return false; if (T->key == x) return true;
 if (x < T->key) return search(T->l, x);
 return search(T->r, x);
 }

 void insert(node* &T, node* n) {
 if (T == NULL) { T = n; return; }
 if (n->priority > T->priority) {
 split(T, n->key, n->l, n->r), T = n;
 } else if (n->key < T->key) insert(T->l, n);
 else insert(T->r, n);
 return update(T);
 }

 void erase(node* &T, ll x) {
 if (T == NULL) return;
 if (T->key == x) { merge(T->l, T->r); }
 else if (x < T->key) erase(T->l, x);
 else erase(T->r, x);
 return update(T);
 }

 bool set(node* T, ll k, ll v) {
 if (T == NULL) return false;
 bool found;
 if (T->key == k) T->value = k, found = true;
 else if (k < T->key) found = set(T->l, k, v);
 else found = set(T->r, k, v);
 if (found) update(T); return found;
 }

 node* find(node* T, ll k) {
 if (T == NULL) return NULL;
 if (T->key == k) return T;
 if (k < T->key) return find(T->l, k);
 return find(T->r, k);
 }

 treap() {root = NULL;}
 treap(ll x) {root = new node(x);}
 treap &merge(treap &O) {merge(root, O.root, O.root); return *this; }
 treap split(ll x) {treap ans; split(root, x, root, ans.root); return ans; }
};

Mo Update, Ordered Set, Treap, Implicit Treap # 7b9b48

```

```

bool search(ll x) {return search(root, x); }
void insert(ll x) {if(search(root, x)) return; return
insert(root, new node(x));}
void erase(ll x) {return erase(root, x); }
void set(ll k, ll v) {if(set(root, k, v)) return;
insert(root, new node(k, v));}
ll operator[](ll k) {
 node* n = find(root, k);
 if(n == NULL) n = new node(k), insert(root, n); return
n->value;
}
ll query(ll a, ll b) {
 node *T1, *T2, *T3; split(root, a, b, T1, T2, T3);
 ll ans = max_value(T2); merge(root, T1, T2, T3);
 return ans;
}
mt19937_64
treap::Mt(chrono::system_clock::now().time_since_epoch().count)
Implicit Treap # c24c46

```

**Description:** Acts like an array, allowing us to reverse, sum, remove, slice, order of, and remove in  $O(\log n)$

**Status:** Tested

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct implicit_treap {
 static mt19937_64 MT;
 struct node{
 node *l = nullptr, *r = nullptr, *par = nullptr;
 ll sz = 1, p = MT(), v, sumv, lzsum = 0;
 bool lzflip = false;
 node(ll x=0) {sumv=v=x; }
 } *root = nullptr;
 gp_hash_table<int, node*> mp; // Mapping values to nodes
 for_order_of
 implicit_treap(){root=nullptr;}
 implicit_treap(ll x){root=new node(x);}
 ll sz(node* T) { return T?T->sz:0; }
 ll sumv(node* T) { return T?T->sumv:0; }
 void upd(node* T){
 if (T->l) T->l->par = T;
 if (T->r) T->r->par = T;
 T->sumv=T->v+sumv(T->l)+sumv(T->r);
 T->sz=lzsum+sz(T->l)+sz(T->r);
 }
 void push(node* T){
 if (T->lzsum){
 T->v+=T->lzsum;
 T->sumv+=T->sz*T->lzsum;
 if (T->l) T->l->lzsum+=T->lzsum;
 if (T->r) T->r->lzsum+=T->lzsum;
 T->lzsum=0;
 }
 if (T->lzflip){
 swap(T->l, T->r);
 if (T->l) T->l->lzflip^=1;
 if (T->r) T->r->lzflip^=1;
 T->lzflip=0;
 }
 }
 void merge(node*& T, node* L, node*& R){
 if (!L||!R){T=L;R;return;}
 push(L);push(R);
 if (L->r>R->l) merge(L->r, L->l, R), T=L;
 else merge(R->l, L, R->r), T=R;
 upd(T);
 }
 void split(node* T, int k, node*& L, node*& R){
 if (!T){L=R=NULL;return;}
 push(T);
 int key=sz(T->l);
 if (key<=k) split(T->l, k, L, R), L=T;
 else split(T->r, k-key-1, R, L), R=T;
 }
 void set(node* T, int k, ll x){
 push(T);
 if (key==k) set(T->l, k, x);
 else set(T->r, k-key-1, x);
 upd(T);
 }
 void update(int i, int j, ll x){
 node *T1,*T2,*T3;
 split(root,i-1,T1), split(T2,j-i,T2), split(T2,j-i,T2,T3);
 merge(root,T1,T2);
 }
 void flip(int i, int j){
 node *T1,*T2,*T3;
 split(root,i-1,T1), split(T2,j-i,T2), T2->lzflip^=1;
 merge(T2,T2,T3), merge(root,T1,T2);
 }
 void insert(int i, ll x){
 node *T1,*T2;
 split(root,i-1,T1), split(T2,i,T2);
 merge(T1,T1,new node(x)), merge(root,T1,T2);
 }
 void push_back(ll x){auto nn = new node(x);
 merge(root,root, nn); mp[x] = nn;}
 int order_of(int x) {
 node* t = mp[x];
 vector<node*> path;
 for (node* cur = t; cur; cur = cur->par)
 path.push_back(cur);
 reverse(path.begin(), path.end());
 for (node* n : path) push(n), upd(n);
 int idx = sz(t->l);
 node* cur = t;
 while (cur->par) {
 if (cur == cur->par->r)
 idx += sz(cur->par->l) + 1;
 cur = cur->par;
 }
 }
};

Implicit Treap # c24c46

```

```

 }
 return idx;
 }
};

mt19937_64
implicit_treap::MT(chrono::system_clock::now().time_since_epoch());
LCT # e6d607

```

**Description:** Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree. Runs on  $O(\log n)$  per operation.

**Status:** Tested

---

```

struct Node { // Splay tree. Root's pp contains tree's parent.
 Node *p = 0, *pp = 0, *c[2];
 bool flip = 0;
 Node() { c[0] = c[1] = 0; fix(); }
 void fix() {
 if (c[0]) c[0]->p = this;
 if (c[1]) c[1]->p = this;
 // (+ update sum of subtree elements etc. if wanted)
 }
 void pushFlip() {
 if (!flip) return;
 flip = 0; swap(c[0], c[1]);
 if (c[0]) c[0]->flip ^= 1;
 if (c[1]) c[1]->flip ^= 1;
 }
 int up() { return p ? p->c[1] == this : -1; }
 void rot(int i, int b) {
 int h = i ^ b;
 Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y :
x;
 if ((y->p = p)) p->c[up()] = y;
 c[i] = z->c[i ^ 1];
 if (b < 2) {
 x->c[h] = y->c[h ^ 1];
 z->c[h ^ 1] = b ? x : this;
 }
 y->c[i ^ 1] = b ? this : x;
 fix(); x->fix(); y->fix();
 if (p) p->fix(); swap(pp, y->pp);
 }
 void splay() { /// Splay this up to the root. Always
finishes without flip set.
 for (pushFlip(); p;) {
 if (p->p) p->p->pushFlip();
 p->pushFlip(); pushFlip();
 int c1 = up(), c2 = p->up();
 if (c2 == -1) p->rot(c1, 2);
 else p->p->rot(c2, c1 != c2);
 }
 Node *first() { /// Return the min element of the subtree
rooted at this, splayed to the top.
 pushFlip();
 return c[0] ? c[0]->first() : (splay(), this);
 }
};

struct link_cut {
 vector<Node*> node;
 link_cut(int N) : node(N) {}
 void link(int u, int v) { // add an edge (u, v)
 makeRoot(&node[u]);
 node[u].pp = &node[v];
 }
 void cut(int u, int v) { // remove an edge (u, v)
 Node *x = &node[u], *top = &node[v];
 makeRoot(top);
 x->splay();
 assert(top == (x->pp ?: x->c[0]));
 }
};

```

## LCT, Equations, Trigonometry, Geometry, Calculus, Sums Series, Probability

```

if (x->pp) x->pp = 0;
else {
 x->c[0] = top->p = 0;
 x->fix();
}
.count();
bool connected(int u, int v) {
 Node *nu = access(&node[u])->first();
 return nu == access(&node[v])->first();
}
void makeRoot(Node *u) {
 access(u); u->splay();
 if (u->c[0]) {
 u->c[0]->p = 0; u->c[0]->flip ^= 1;
 u->c[0]->pp = u; u->c[0] = 0;
 u->fix();
 }
}
Node *access(Node *u) {
 u->splay();
 while (Node *pp = u->pp) {
 pp->splay(); u->pp = 0;
 if (pp->c[1]) {
 pp->c[1]->p = 0;
 pp->c[1]->pp = pp;
 }
 pp->c[1] = u;
 pp->fix(); u = pp;
 }
 return u;
}

```

## Maths

## Equations

Quadratic:

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \text{ extremum at } x = -\frac{b}{2a}$$

Linear system 2x2:

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} \Rightarrow \begin{cases} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{cases}$$

Cramer's rule: Given  $Ax = b$ , then  $x_i = \det(A_{i'}) / \det(A)$  where  $A_{i'}$  is  $A$  with  $i$ -th column replaced by  $b$ .Recurrences: If  $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$  and  $r_1, \dots, r_k$  are distinct roots of  $x^k - c_1 x^{k-1} - \dots - c_k$ , then:

$$a_n = d_1 r_1^n + \dots + d_k r_k^n$$

Non-distinct root  $r$  becomes polynomial factor, e.g.  $a_n = (d_1 + d_2)r^n$ .

## Trigonometry

Sum identities:

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$

$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

Sum to product:

$$\sin v + \sin w = 2 \sin\left(\frac{v+w}{2}\right) \cos\left(\frac{v-w}{2}\right)$$

$$\cos v + \cos w = 2 \cos\left(\frac{v+w}{2}\right) \cos\left(\frac{v-w}{2}\right)$$

Linear combination:

$$a \cos x + b \sin x = r \cos(x - \varphi)$$

where  $r = \sqrt{a^2 + b^2}$ ,  $\varphi = \text{atan2}(b, a)$ .

## Geometry

Triangles — sides  $a, b, c$ , semiperimeter  $p = (a+b+c)/2$ 

$$A = \sqrt{p(p-a)(p-b)(p-c)}$$

$$R = \frac{abc}{4A}, \quad r = \frac{A}{p}$$

$$m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}, \quad s_a = \sqrt{bc \left[ 1 - \left( \frac{a}{b+c} \right)^2 \right]}$$

Law of sines:

$$\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$$

Law of cosines:  $a^2 = b^2 + c^2 - 2bc \cos \alpha$ 

Law of tangents:

$$\frac{a+b}{a-b} = \frac{\tan((\alpha+\beta)/2)}{\tan((\alpha-\beta)/2)}$$

Quadrilaterals — sides  $a, b, c, d$ , diag  $e, f$ , diag angle  $\theta$ ,  $F = b^2 + d^2 - a^2 - c^2$ 

$$4A = 2ef \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

Cyclic: opposite angles sum  $180^\circ$ ,  $ef = ac + bd$ ,  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

Spherical coordinates:

$$x = r \sin \theta \cos \varphi, \quad y = r \sin \theta \sin \varphi, \quad z = r \cos \theta$$

$$r = \sqrt{x^2 + y^2 + z^2}, \quad \theta = \arccos(z/r), \quad \varphi = \text{atan2}(y, x)$$

## Calculus

Derivatives:

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}}, \quad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x, \quad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

Integrals:

$$\int \tan ax \, dx = -\frac{\ln|\cos ax|}{a}$$

$$\int x \sin ax \, dx = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} \, dx = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x)$$

$$\int x e^{ax} \, dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x) \, dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x) \, dx$$

## Sums Series

Geometric sum:

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, \quad c \neq 1$$

Power sums:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{n(2n+1)(n+1)}{6}$$

$$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=1}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

Taylor series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots \quad (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \dots \quad (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

## Probability

Expected value and variance:

$$\mu = \mathbb{E}(X) = \sum_x x p_X(x), \quad \sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2$$

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent  $X, Y$ :  $V(aX + bY) = a^2 V(X) + b^2 V(Y)$ .Binomial  $\text{Bin}(n, p)$ :  $n$  independent yes/no trials, each with probability  $p$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad \mu = np, \quad \sigma^2 = np(1-p)$$

First success  $\text{Fs}(p)$ : Trials until first success.

$$p(k) = p(1-p)^{k-1}, \quad \mu = \frac{1}{p}, \quad \sigma^2 = \frac{1-p}{p^2}$$

Poisson  $\text{Po}(\lambda)$ : Events in fixed time, average rate  $\lambda$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad \mu = \lambda, \quad \sigma^2 = \lambda$$

Uniform  $\text{U}(a, b)$ :

$$f(x) = \frac{1}{b-a} \text{ for } a < x < b, \quad \mu = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential Exp( $\lambda$ ):

$$f(x) = \lambda e^{-\lambda x} \text{ for } x \geq 0, \quad \mu = \frac{1}{\lambda}, \quad \sigma^2 = \frac{1}{\lambda^2}$$

Normal  $\mathcal{N}(\mu, \sigma^2)$ :

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$aX_1 + bX_2 + c \sim \mathcal{N}(a\mu_1 + b\mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

Markov chains: Transition  $P = (p_{ij})$ ,  $P^{(n)} = P^n P^{(0)}$ .

Stationary:  $\pi = \pi P$ . If irreducible:  $\pi_i = 1/\mathbb{E}(T_i)$ . Connected undirected non-bipartite graph with uniform transitions:  $\pi_i \propto \deg(i)$ . Ergodic iff irreducible and aperiodic.

$$\text{Absorption: } a_{ij} = p_{ij} + \sum_{k \in G} a_{ik} p_{kj}, \quad t_i = 1 + \sum_{k \in G} p_{ki} t_k.$$

## Mulmod # c29f90

Description: Multiply two number fast, secure for number lesser than  $2^{57}$

Status: Partially tested

```
ll mulmod(ll a, ll b, ll m) {
 ll r=a*b-(ll)(ld)a*b/m+.5)*m;
 return r<0?r+m:r;
}
```

## Binary Pow # 6c139d

Description: Exponentiation by squares, computes  $a^b \bmod m$  in  $O(\log b)$

Status: Highly tested

```
ll binpow(ll a, ll b, ll m) {
 a %= m;
 ll res = 1;
 while (b > 0) {
 if (b & 1)
 res = (res * a) % m;
 a = (a * a) % m;
 b >>= 1;
 }
 return res;
}
```

## Binary Pow 128 Bits # 1cf714

Description: Binary pow in 128 bits, computes  $a^b \bmod m$  in  $O(\log b)$

Status: Highly tested

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpow(u64 a, u64 b, u64 m) {
 a %= m;
 u64 res = 1;
 while (b > 0) {
 if (b & 1)
 res = (u128)res * a % m;
 a = (u128)a * a % m;
 b >>= 1;
 }
 return res;
}
```

## Binary Pow Mulmod

Divisors # c5b02f

Description: Binary pow using mulmod, mulmod can be changed.

Status: Partially tested

```
ll binpow(ll b, ll e, ll m){
 if(!e) return 1;
 ll q=binpow(b,e/2,m);q=mulmod(q,q,m);
 return e&1?mulmod(b,q,m):q;
}
```

## Montgomery

# 2810ef

Description: Optimizes multiplication with large modules, useful for fast\_ntt.

Status: Tested on codeforces.

```
using ull = uint64_t;
struct montgomery {
 ull n, nr;
 constexpr montgomery(ull n) : n(n), nr(1) {
 for (int i = 0; i < 6; i++)
 nr *= 2 - n * nr;
 }
 ull reduce(__uint128_t x) const {
 ull q = __uint128_t(x) * nr;
 ull m = ((__uint128_t) q * n) >> 64;
 ull res = (x >> 64) + n - m;
 if (res >= n)
 res -= n;
 return res;
 }
 ull multiply(ull x, ull y) const {
 return reduce((__uint128_t) x * y);
 }
 ull transform(ull x) const {
 return (__uint128_t(x) << 64) % n;
 }
};
```

## Number Theory Reference

Bézout's identity: For  $a \neq 0, b \neq 0$ ,  $d = \gcd(a, b)$  is the smallest positive integer with integer solutions to  $ax + by = d$ . All solutions:

$$\left( x + \frac{kb}{\gcd(a,b)}, \quad y - \frac{ka}{\gcd(a,b)} \right), \quad k \in \mathbb{Z}$$

Pythagorean triples: Uniquely generated by:

$$a = k(m^2 - n^2), \quad b = k(2mn), \quad c = k(m^2 + n^2)$$

with  $m > n > 0, k > 0, \gcd(m, n) = 1$ , and  $m, n$  not both odd.

Möbius function:

$$\mu(n) = \begin{cases} 0 & \text{if } n \text{ is not squarefree} \\ 1 & \text{if } n \text{ has even number of prime factors} \\ -1 & \text{if } n \text{ has odd number of prime factors} \end{cases}$$

Möbius inversion:

$$g(n) = \sum_{d|n} f(d) \leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

$$\sum_{d|n} \mu(d) = [n=1] \quad (\text{very useful})$$

$$g(n) = \sum_{n|d} f(d) \leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$$

## Log, Erathostenes Sieve

# aa7220

$$g(n) = \sum_{1 \leq m \leq n} f\left(\left\lfloor \frac{n}{m} \right\rfloor\right) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m) g\left(\left\lfloor \frac{n}{m} \right\rfloor\right)$$

Reference primes: 970592641 (31-bit), 31443539979727 (45-bit), 3006703054056749 (52-bit).  $p = 962592769: 2^{21} | p - 1$ . There are 78498 primes below  $10^6$ .

Primitive roots exist mod any prime power  $p^a$ , except  $p = 2, a > 2$ .

Estimates:  $\sum_{d|n} d = O(n \log \log n)$ .

#divisors of  $n$ : 100 for  $n < 5 \cdot 10^4$ , 500 for  $n < 10^7$ , 2000 for  $n < 10^{10}$ , 200000 for  $n < 10^{19}$ .

## Extended Gcd

# 493f93

Description: Extended euclidean algorithm

Status: Not stress-tested, but it gave AC in problems

```
struct GCD_type { ll x, y, d; };
GCD_type ex_GCD(ll a, ll b){
 if (b == 0) return {1, 0, a};
 GCD_type pom = ex_GCD(b, a % b);
 return {pom.y, pom.x - a / b * pom.y, pom.d};
}
```

## CRT

# 725c32

Description: Chinese remainder theorem

Status: Stress-tested

```
using lll = __int128_t;
ll crt(vector<ll> a, vector<ll> m) {
 int n = a.size();
 for (int i = 0; i < n; i++) {
 a[i] %= m[i];
 a[i] = a[i] < 0 ? a[i] + m[i] : a[i];
 }
 ll ans = a[0];
 ll M = m[0];
 for (int i = 1; i < n; i++) {
 auto pom = ex_GCD(M, m[i]);
 ll x1 = pom.x;
 ll d = pom.d;
 if ((a[i] - ans) % d != 0)
 return -1;
 ans = ans + x1 * (a[i] - ans) / d % (m[i] / d) * M;
 M = M * m[i] / d;
 ans %= M;
 ans = ans < 0 ? ans + M : ans;
 M = M / (lll) __gcd((lll)M, (lll)m[i]) * m[i];
 }
 return ans;
}
```

## Prime Factors

# d58001

Description: Get all prime factors of  $n$ , runs on  $O(\sqrt{n})$

Status: Highly tested

```
vector<ll> primeFactors(ll n) {
 vector<ll> factors;
 for (ll i = 2; (i*i) <= n; i++) {
 while (n % i == 0) {
 factors.push_back(i);
 n /= i;
 }
 if (n > 1) factors.push_back(n);
 }
 return factors;
}
```

## Counting Divisors

# aa7220

Description: Counting divisors in  $O(n^{\frac{1}{2}})$

Status: Tested on codeforces

```
const int MX_P = 1e6 + 1;
eratosthenes_sieve sieve(MX_P);
ll counting_divisors(ll n) {
 ll ret = 1;
 for (ll p : sieve.primes) {
 if (p*p > n) break;
 ll count = 1;
 while (n % p == 0)
 n /= p, count++;
 ret *= count;
 }
 ll isqrt = sqrtl(n);
 if (miller_rabin(n)) ret *= 2;
 else if (isqrt*isqrt == n and miller_rabin(isqrt)) ret *= 3;
 else if (n != 1) ret *= 4;
 return ret;
}
```

## Discrete Log

# 8796c5

Author: Marcos Kolodny (MarcosK)

Description: Returns  $x$  such that  $a^x = b \pmod m$  or  $-1$  if nonexistent

Status: Tested on josupo.jp

```
ll discrete_log(ll a,ll b,ll m) {
 a%=m;
 b%=m;
 if(b == 1) return 0;
 ll cnt=0, tmp=1;
 for(ll g=__gcd(a,m);g!=1;g=__gcd(a,m)) {
 if(b%g) return -1;
 m/g, b/g;
 tmp = (__int128)tmp*a/g%m;
 ++cnt;
 if(b == tmp) return cnt;
 }
 map<ll, ll> w;
 ll s = ceil(sqrtl(m)), base = b;
 for (ll i = 0; i < s; i++) {
 w[base] = i, base = (__int128)base*a%m;
 base=binpow(a,s,m);
 ll key=tmp;
 for (ll i = 1; i < s+2; i++) {
 key=(_int128)base*key%m;
 if(w.count(key)) return i*s-w[key]+cnt;
 }
 }
 return -1;
}
```

## Erathostenes Sieve

# 3efae9

Description: Get all prime numbers up to  $n$ , runs on  $O(n \log(\log(n)))$

Status: Highly tested

```
struct eratosthenes_sieve {
 vector<ll> primes;
 vector<bool> isPrime;
 eratosthenes_sieve(ll n) {
 isPrime.resize(n + 1, true);
 isPrime[0] = isPrime[1] = false;
 for (ll i = 2; i <= n; i++) {
 if (isPrime[i]) {
 primes.push_back(i);
 for (ll j = i*i; j <= n; j += i)
 isPrime[j] = false;
 }
 }
 }
}
```

```

 }
};

Euler Phi # ac936a

```

Description: Return  $\varphi(n)$ , runs on  $O(\sqrt{n})$

Status: Not tested

```

ll phi(ll n) {
 ll result = n;
 for (ll i = 2; i*i <= n; i++) {
 if (n % i == 0) {
 while (n % i == 0) n /= i;
 result -= result / i;
 }
 }
 if (n > 1)
 result -= result / n;
 return result;
}

```

**Euler Phi Sieve** # 837c81

Description: Return  $\varphi(n)$  for all positive  $n$ , runs on  $O(n \log(\log(n)))$

Status: Not tested

```

struct euler_phi {
 vector<int> phi;
 euler_phi(int n) {
 phi.resize(n + 1);
 for (int i = 1; i <= n; i++)
 phi[i] = i;
 for (int i = 2; i <= n; i++)
 if (phi[i] == i)
 for (int j = i; j <= n; j += i)
 phi[j] = phi[j] / i * (i - 1);
 }
};

```

**Mobius Sieve** # c26c90

Description: Compute the first values for the mobius function in  $O(n \log(n))$

Status: Tested in codeforces.

```

vector<int> mobius_sieve(int mxn){
 vector<int> mob(mxn+1);
 mob[1] = 1;
 for (int i = 2; i <= mxn; i++){
 mob[i] = -1;
 for (int j = 2*i; j <= mxn; j += i)
 mob[j] = -mob[i];
 }
 return mob;
}

```

**Miller Rabin** # d3ad25

Description: Detect if a number is prime or not in  $O(\log^2(n))$ , needs 128 bits binary pow

Status: Highly tested

```

bool miller_rabin(uint64_t n) {
 if (n <= 1) return false;
 auto check = [] (uint64_t n, uint64_t a, uint64_t d,
 uint64_t s) {
 uint64_t x = binpow(a, d, n); // Usar binpow de 128bits
 if (x == 1 || x == n-1) return true;
 for (int m = 0; m < r && t != 1; ++m) t = t*t%p;
 if (t == 1) return false;
 };
}

```

```

 for (uint64_t r = 1; r < s; r++) {
 x = (_uint128_t)x*x % n;
 if (x == n-1) return true;
 }
 return true;
};

uint64_t r = 0, d = n - 1;
while ((d & 1) == 0) d >>= 1, r++;
for (int x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
 if (x == n) return true;
 if (check(n, x, d, r)) return false;
}
return true;
}

```

**Pollard Rho** # 90cc17

Description: Finds a non-trivial factor of a composite number  $n$  using Pollard's Rho algorithm.

Status: Partially tested

```

ll pollard_rho(ll n) {
 if((n&1)==1) return 2;
 ll x = 2, y=2, d=1;
 ll c = rand()%n+1;
 while(d==1) {
 x=(mulmod(x,x,n)+c)%n;
 y=(mulmod(y,y,n)+c)%n;
 y=(mulmod(y,y,n)+c)%n;
 if(x>=y)d=_gcd(x-y,n);
 else d=_gcd(y-x,n);
 }
 return d==n? pollard_rho(n):d;
}

```

**Fast Prime Factorization** # ec94d3

Description: Factorize the number using Pollard's Rho and Miller Rabin  $O(n^{1/3})$

Status: Partially tested

```

void fact(ll n, map<ll, int> &f) { //0 (lg n)^3
 if(n==1) return;
 if(nd_miller_rabin(n)) { f[n]++;
 return; }
 ll q=pollard_rho(n); fact(q,f); fact(n/q,f);
}

```

**Mod Sqrt** # 5f3464

Description: Tonelli-Shanks. Finds  $x$  s.t.  $x^2 \equiv a \pmod{p}$  ( $-x$  gives the other).  $O(\log^2 p)$  worst case.

Status: KACTL based, not self tested

```

ll modpow(ll b, ll e, ll m) {
 ll r = 1; for (b %= m; e > 0; e >= 1, b = b*b%m)
 if (e & 1) r = r*b%m; return r;
}

ll modSqrt(ll a, ll p) {
 a %= p; if (a < 0) a += p;
 if (a == 0) return 0;
 assert(modpow(a, (p-1)/2, p) == 1);
 if (p % 4 == 3) return modpow(a, (p+1)/4, p);
 ll s = p-1, m;
 while (s % 2 == 0) ++r, s /= 2;
 while (modpow(n, (p-1)/2, p) != p-1) ++n;
 ll x = modpow(a, (s+1)/2, p);
 ll b = modpow(a, s, p), g = modpow(n, s, p);
 for (; r == m) {
 ll t = b;
 for (m = 0; m < r && t != 1; ++m) t = t*t%p;
 if (m == 0) return x;
 }
}

```

## Numbers

```

 ll gs = modpow(g, 1LL << (r-m-1), p);
 g = gs*gs%p;
 x = x*gs%p;
 b = b*g%p;
}
}

```

**Continued Fractions** # b1d934

Description: Given  $N$  and real  $x \geq 0$ , finds closest rational  $p/q$  with  $p, q \leq N$ . Obey  $|p/q - x| \leq 1/(qN)$ .  $O(\log N)$ .

Status: KACTL based, not self tested

```

pair<ll,ll> approximate(ld x, ll N) {
 ll LP = 0, LQ = 1, P = 1, Q = 0;
 ll inf = LLONG_MAX;
 ld y = x;
 for (;;) {
 ll lim = min(P ? (N-LP)/P : inf,
 Q ? (N-LQ)/Q : inf);
 ll a = (ll)floor(y), b = min(a, lim);
 ll NP = b*P+LP, NQ = b*Q+LQ;
 if (a > b) {
 return {abs(x-(ld)NP/(ld)NQ) <
 abs(x-(ld)P/(ld)Q));
 }
 if (abs(y - 1/(y-(ld)a)) > 3*N)
 return {NP, NQ};
 LP = P; P = NP;
 LQ = Q; Q = NQ;
 }
}

```

**Combinatorics Reference**

**Derangements:** Permutations where no element is in its original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

**Burnside's lemma:** Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals:

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

where  $X^g$  are elements fixed by  $g$ . For rotational symmetry with  $G = \mathbb{Z}_n$ :

$$g(n) = \frac{1}{n} \sum_{k \mid n} f(k)\varphi(n/k)$$

**Partition function:** Ways to write  $n$  as sum of positive integers (order irrelevant).

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n-k(3k-1)/2)$$

$$p(n) \sim \frac{0.145}{n} \cdot e^{2.56\sqrt{n}}$$

| $n$    | 0 | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9  | 20  | 50  | 100 |
|--------|---|---|---|---|---|---|----|----|----|----|-----|-----|-----|
| $p(n)$ | 1 | 1 | 2 | 3 | 5 | 7 | 11 | 15 | 22 | 30 | 627 | 2e5 | 2e8 |

**Lucas' theorem:** For prime  $p$ , write  $n = \sum n_i p^i$ ,  $m = \sum m_i p^i$ :

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

Multinomial:  $\binom{n}{k_1, k_2, \dots, k_m} = \frac{n!}{k_1! k_2! \dots k_m!}$

## Special Numbers

**Bernoulli numbers:** EGF:  $B(t) = t/(e^t - 1)$ .

| $B_0$ | $B_1$          | $B_2$         | $B_3$ | $B_4$           | $B_5$ | $B_6$          | $\dots$ |
|-------|----------------|---------------|-------|-----------------|-------|----------------|---------|
| 1     | $-\frac{1}{2}$ | $\frac{1}{6}$ | 0     | $-\frac{1}{30}$ | 0     | $\frac{1}{42}$ |         |

Sums of powers:

$$\sum_{i=1}^n i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula:

$$\sum_{i=m}^{\infty} f(i) \approx \int_m^{\infty} f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f''(m)}{720} + O(f^{(5)}(m))$$

**Stirling 1st kind:** Permutations on  $n$  items with  $k$  cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

$$\sum_{k=0}^n c(n, k)x^k = x(x+1)\cdots(x+n-1)$$

**Eulerian numbers:** Permutations of  $S_n$  with exactly  $k$  ascents.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

**Stirling 2nd kind:** Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), \quad S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

**Bell numbers:** Total partitions of  $n$  distinct elements.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, \dots$

For  $p$  prime:  $B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$

**Labeled unrooted trees:** #on  $n$  vertices:  $n^{n-2}$ , #on  $k$  trees of size  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$ . #with degrees  $d_i$ :  $\frac{(n-2)!}{(d_1-1) \cdots (d_n-1)!}$

**Catalan numbers:**

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- Sub-diagonal monotone paths in  $n \times n$  grid
- Strings with  $n$  pairs of correctly nested parentheses
- Binary trees with  $n+1$  leaves (0 or 2 children)
- Ordered trees with  $n+1$  vertices
- Triangulations of convex polygon with  $n+2$  sides
- Permutations of  $[n]$  with no 3-term increasing subseq

## Factorial, Tetration, Gauss, Xor Basis, Matrix, Simplex, FFT

## Factorial

# 7e9435

**Description:** Calculate factorials from 1 to  $n$  and their factorial, runs on  $O(n)$   
**Status:** Highly tested

```
struct Factorial {
 vector<ll> f, finv, inv; ll mod;
 Factorial(ll n, ll mod): mod(mod) {
 f.assign(n+1, 1); inv.assign(n+1, 1); finv.assign(n+1, 1);
 for (ll i = 2; i <= n; ++i)
 inv[i] = (mod-i) * inv[mod % i] % mod;
 for (ll i = 1; i <= n; ++i) {
 f[i] = (f[i-1] * i) % mod;
 finv[i] = (finv[i-1] * inv[i]) % mod;
 }
 }
};
```

## Tetration

# b9b3df

**Description:** Calculate  $b^a \bmod m$   
**Status:** Tested on josupo.jp

```
map<ll, ll> memophi;
ll tetratoin(ll a, ll b, ll m) {
 if (m == 1) return 0;
 if (a == 0) return (b+1) % 2 % m;
 if (a == 1 or b == 0) return 1;
 if (b == 1) return a % m;
 if (a == 2 and b == 2) return 4 % m;
 if (a == 2 and b == 3) return 16 % m;
 if (a == 3 and b == 2) return 27 % m;
 if (memophi.find(m) == memophi.end())
 memophi[m] = phi(m);
 ll tot = memophi[m];
 ll n = tetratoin(a, b-1, tot);
 return binpow(a, (n < tot ? n + tot : n), m);
}
```

## Algebra

# 576480

**Description:** Matrix elimination, runs on  $O(n^3)$   
**Status:** Not tested

```
const double EPS = 1e-18;
const int INF = 2; // it doesn't actually have to be infinity or a big number
const int MOD = 1e9+7;
int gauss(vector<vector<ll>> a, vector<ll> &ans) {
 int n = (int) a.size();
 int m = (int) a[0].size() - 1;

 vector<int> where(m, -1);
 for (int col = 0, row = 0; col < m && row < n; ++col) {
 int sel = row;
 for (int i = row; i < n; ++i)
 if (abs(a[i][col]) > abs(a[sel][col]))
 sel = i;
 if (!a[sel][col])
 continue;
 for (int i = col; i <= m; ++i)
 swap(a[sel][i], a[row][i]);
 where[col] = row;

 for (int i = 0; i < n; ++i)
 if (i != row) {
 ll c = (a[i][col] * binpow(a[row][col], MOD - 2,
 MOD)) % MOD;
 for (int j = col; j <= m; ++j)
 a[i][j] = ((a[i][j] - a[row][j] * c) % MOD + MOD)
 % MOD;
 }
 }
}
```

## Gauss

# 576480

```
 }
 ++row;
}

ans.assign(m, 0);
for (int i = 0; i < m; ++i)
 if (where[i] != -1)
 ans[i] = (a[where[i]][m] * binpow(a[where[i]][i], MOD - 2, MOD)) % MOD;

for (int i = 0; i < m; ++i)
 if (where[i] == -1)
 return INF;
 return 1;
}
```

## Xor Basis

# bc3ab1

**Description:** Calculate the xor basis of a set of numbers.

**Complexity:**

- Insertion:  $O(d)$
- Query if a number can be formed:  $O(d)$
- Get maximum xor subset:  $O(d)$
- Merging two bases:  $O(d^2)$

**Status:** Partially tested

```
struct xor_basis {
 int d, sz;
 vector<long long> basis;
 xor_basis(int _d, d, sz(0), basis(_d, 0) {};
 bool insert(long long mask) {
 for (int i = d - 1; i >= 0; --i) {
 if ((mask & (1LL << i)) == 0) continue;
 if (!basis[i]) {
 basis[i] = mask;
 ++sz;
 return true;
 }
 mask ^= basis[i];
 }
 return false;
 }
 bool can_make(long long x){
 for (int i = d - 1; i >= 0; --i) {
 if ((x & (1LL << i)) == 0) continue;
 if (!basis[i]) return false;
 x ^= basis[i];
 }
 return x == 0;
 }
 long long get_max(){
 long long res = 0;
 for (int i = d - 1; i >= 0; --i)
 if ((res ^ basis[i]) > res) res ^= basis[i];
 return res;
 }
}
```

## Matrix

# a95ea4

**Author:** Carlos Lagos

**Description:** Matrix operations including multiplication and binary exponentiation

**Status:** Tested

```
template<class T>
vector<vector<T>> multWithoutMOD(vector<vector<T>> &a,
vector<vector<T>> &b){
 int n = a.size(), m = b[0].size(), l = a[0].size();
 vector<vector<T>> ans(n, vector<T>(m, 0));
 for (int i = 0; i < n; i++)
 for (int j = 0; j < m; j++)
 for (int k = 0; k < l; k++)
 ans[i][j] += a[i][k]*b[k][j];
 return ans;
}
```

```
vector<vector<T>> ans(n, vector<T>(m, 0));
for (int i = 0; i < n; i++)
 for (int j = 0; j < m; j++)
 for (int k = 0; k < l; k++)
 ans[i][j] += a[i][k]*b[k][j];
return ans;
}
```

```
template<class T>
vector<vector<T>> mult(vector<vector<T>> a,
vector<vector<T>> b, ll mod){
 int n = a.size(), m = b[0].size(), l = a[0].size();
 vector<vector<T>> ans(n, vector<T>(m, 0));
 for (int i = 0; i < n; i++)
 for (int j = 0; j < m; j++)
 for (int k = 0; k < l; k++)
 T temp = ((ll)a[i][k]*b[k][j]) % mod;
 ans[i][j] = (ans[i][j] + temp) % mod;
 return ans;
}
```

```
vector<vector<ll>> binpow(vector<vector<ll>> v, ll n, long long mod){
 ll dim = v.size(); vector<vector<ll>> ans(dim, vector<ll>(dim, 0));
 ans(dim, vector<ll>(dim, 0));
 for (ll i = 0; i < dim; i++) ans[i][i] = 1;
 while (n){
 if (n & 1) ans = mult(ans, v, mod);
 v = mult(v, v, mod);
 n = n >> 1;
 }
 return ans;
}
```

## Simplex

# 9f5446

**Description:** Optimizes linear function, based of linear restrictions, in  $O(n^2)$

**Status:** Not tested

```
template<class T> struct Simplex {
 T ans;
 vector<vector<T>> a;
 vector<T> b,c,d;
 void pivot(int ii, int jj){
 d[ii] = c[jj];
 T s1 = a[ii][jj];
 for (int i = 0; i < a[0].size(); i++)
 a[ii][i] /= s1;
 b[ii] /= s1;
 for (int i = 0; i < d.size(); i++){
 if (i == ii || a[i][jj] == 0) continue;
 T s2 = a[i][jj];
 for (int j = 0; j < a[0].size(); j++)
 a[i][j] -= s2*a[ii][j];
 b[i] -= s2*b[ii];
 }
 }
 bool next_point(){
 int idx = -1; T mx;
 for (int i = 0; i < (int)a[0].size(); i++){
 T z = 0;
 for (int j = 0; j < (int)d.size(); j++)
 z += a[j][i]*d[j];
 if (idx == -1 || mx < c[i]-z)
 mx = c[i]-z, idx = i;
 }
 if (mx > 0){
 int idx2 = -1; T mn;
 for (int i = 0; i < d.size(); i++)
 if (d[i] > 0)
 mn = d[i], idx2 = i;
 if (mn > 0)
 pivot(idx, idx2);
 }
 }
}
```

```
for (int i = 0; i < (int)b.size(); i++){
 if (a[i][idx] == 0 || b[i]/a[i][idx] <= 0)
 continue;
 if (idx2 == -1 || mn > b[i]/a[i][idx])
 mn = b[i]/a[i][idx], idx2 = i;
}
if (idx2 == -1) return 0; // unbounded
pivot(idx2, idx);
return 1;
}
return 0;
}
Simplex(vector<vector<T>> &a, vector<T> &b,
vector<T> &c) : a(a), b(b), c(c){
 d.resize(b.size(), 0);
 while (next_point());
 ans = 0;
 for (int i = 0; i < b.size(); i++)
 ans += b[i]*d[i];
}
```

## Polynomials

# 00a05a

**Description:** FFT(a) computes  $\widehat{f(k)} = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2, runs on  $O(N \log N)$ , where  $N = |A| + |B|$

- For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back.

- Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2(N) < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs).

**Status:** Highly tested (josupo.jp)

```
struct FFT {
 const long double PI = acos(-1);
 typedef long double d; // to double if too slow
 void fft(vector<complex<d>> &a) {
 int n = a.size(), L = 31 - __builtin_clz(n);
 vector<complex<d>> R(2, 1), rt(2, 1);
 for (int k = 2; k < n; k *= 2) {
 R.resize(n); rt.resize(n);
 auto x = polar(1.0L, PI / k);
 for (int i = k; i < 2*k; ++i) rt[i] = R[i] = i & 1 ?
 R[i / 2] * x : R[i / 2];
 vector<int> rev(n);
 for (int i = 0; i < n; ++i) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
 for (int i = 0; i < n; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
 for (int k = 1; k < n; k *= 2)
 for (int i = 0; i < n; i += 2 * k)
 for (int j = 0; j < k; ++j) {
 auto x = (d*)Sqrt[j + k], y = (d*)&a[i + j + k];
 complex<d> z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
 a[i + j + k] = a[i + j] - z, a[i + j] += z;
 }
 vector<int> conv(vector<d> &a, vector<d> &b) {
 if (a.empty() || b.empty()) return {};
 vector<d> res(a.size() + b.size() - 1);
 int B = 32 - __builtin_clz(res.size()), n = 1 << B;
 vector<complex<d>> in(n), out(n);
 copy(a.begin(), a.end(), in.begin());
 for (int i = 0; i < b.size(); ++i) in[i].imag(b[i]);
 fft(in); for (auto &x : in) x *= x;
 for (int i = 0; i < n; ++i) out[i] = in[-i & (n - 1)] -
 conj(in[i]);
 fft(out); for (int i = 0; i < res.size(); ++i) res[i] =
 imag(out[i]) / (4 * n);
 }
 }
 }
}
```

```

vector<int> resint(res.size());
for (int i = 0; i < res.size(); i++) resint[i] =
round(res[i]);
return resint;
}

vector<ll> convMod(vector<ll> &a, vector<ll> &b, ll mod) {
if (a.empty() || b.empty()) return {};
vector<ll> res(a.size() + b.size() - 1);
int B = 32 - __builtin_clz(res.size()), n = 1 << B;
ll cut = (ll)sqrt(mod);
vector<complex<d>> L(n), R(n), outs(n), outl(n);
for (int i = 0; i < (int)a.size(); i++) L[i] =
complex<d>(a[i]/cut, ai[i]*cut);
for (int i = 0; i < (int)b.size(); i++) R[i] =
complex<d>(b[i]/cut, bi[i]*cut);
fft(L, fft(R));
for (int i = 0; i < n; i++) {
int j = -i & (n-1);
outl[j] = (L[i] + conj(L[j])) * R[i] / ((d)2.0 * n);
outs[j] = (L[i] - conj(L[j])) * R[i] / ((d)2.0 * n) /
complex<d>(0, 1);
}
fft(outl, fft(outs));
for (int i = 0; i < (int)res.size(); i++) {
ll av = (ll)(real(outl[i])+.5), cv = (ll)
(imag(outs[i])+.5);
ll bv = (ll)(imag(outl[i])+.5) + (ll)
(real(outs[i])+.5);
res[i] = ((av % mod * cut + bv) % mod * cut + cv) %
mod;
}
vector<ll> resll(res.size());
for (int i = 0; i < (int)res.size(); i++) resll[i] =
(ll)round(res[i]);
return resll;
}
};

Fast Ntt # d579d9

```

**Description:** Same utility as FFT but with some magic primes, runs in  $O(n \log n)$  with better constant

Possible primes and their roots:

- 998244353, 3
- 9223372036737335297, 3
- 2013265921, 31

requires binpow(be careful with overflow), and montgomery

**Status:** Tested on codeforces, every prime above

```

using ull = uint64_t;
template<ull mod, ull gen>
struct fast_ntt {
 fast_ntt() {}
 vector<int> bit_sort(int n) {
 int h = -1;
 vector<int> rev(n, 0);
 int skip = __lg(n) - 1;
 for (int i = 1; i < n; ++i) {
 if (!(i & (i - 1)))
 ++h;
 rev[i] = rev[i ^ (1 << h)] | (1 << (skip - h));
 }
 return rev;
 }
 void ntt(vector<ull> &a, vector<int> &rev, montgomery&
red, ull inv_n, ull inv_root, bool invert) {
 int n = (int)a.size();
 for (int i = 0; i < n; ++i)
 if (i < rev[i])
 swap(a[i], a[rev[i]]);
 ull w = invert ? inv_root : root;
 vector<ull> W(n >> 1, red.transform(1));
 for (int i = 1; i < (n >> 1); ++i)
 W[i] = red.multiply(W[i-1], w);
 }
};

Fast Ntt # d579d9

```

```

int lim = __lg(n);
for (int i = 0; i < lim; ++i)
 for (int j = 0; j < n; ++j)
 if (!(j & (1 << i))) {
 ull t = red.multiply(a[j ^ (1 << i)],
W[(j & ((1 << i) - 1)) * (n >> (i + 1))]);
 a[j ^ (1 << i)] = a[j] >= t ? a[j] - t :
a[j] + mod - t;
 a[j] = a[j] + t < mod ? a[j] + t : a[j] +
t - mod;
 }
 if (invert)
 for (int i = 0; i < n; i++)
 a[i] = red.multiply(a[i], inv_n);
}

vector<ull> conv(vector<ull> a, vector<ull> b) {
 montgomery red(mod);
 for (auto& x : a)
 x = red.transform(x);
 for (auto& x : b)
 x = red.transform(x);
 int n = 1;
 while (n < a.size() || n < b.size())
 n <<= 1;
 n <<= 1;
 a.resize(n);
 b.resize(n);
 ull inv_n = red.transform(bin_pow(n, mod-2, mod));
 ull root = red.transform(bin_pow(gen, (mod-1)/n,
mod));
 ull inv_root =
red.transform(bin_pow(red.reduce(root), mod-2, mod));
 auto rev = bit_sort(n);
 ntt(a, rev, red, inv_n, root, inv_root, false);
 ntt(b, rev, red, inv_n, root, inv_root, false);
 for (int i = 0; i < n; i++)
 a[i] = red.multiply(a[i], b[i]);
 ntt(a, rev, red, inv_n, root, inv_root, true);
 for (auto& x : a)
 x = red.reduce(x);
 return a;
}
};

Fwht # fe4251

```

**Author:**

**Description:** Computes XOR, OR and AND convolution of two arrays in  $O(n \log n)$ .

op = 0 -> OR, op = 1 -> AND, op = 2 -> XOR, mxn and n must be powers of two.

**Status:** Tested on CSES

```

const int mod = 1e9+7, mxn = (1<<20), op = 2;
const int inv2 = (mod+1) >> 1;
struct fwht {
 int P1[mxn], P2[mxn];
 void wt(int *a, int n){
 if (n == 0) return;
 int m = n/2;
 wt(a+m, m);
 for (int i = 0; i < m; i++){
 int x = a[i], y = a[i+m];
 if (op == 0) a[i] = x, a[i+m] = (x+y)%mod;
 if (op == 1) a[i] = (x+y)%mod, a[i+m] = y;
 if (op == 2) a[i] = (x+y)%mod, a[i+m] = (x-y+mod)%mod;
 }
 }
 void iwt(int* a, int n){
 if (n == 0) return;
 int m = n/2;
 iwt(a, m);
 for (int i = 0; i < m; i++){
 int x = a[i], y = a[i+m];
 if (op == 0) a[i] = x, a[i+m] = (y-x+mod)%mod;
 if (op == 1) a[i] = (x+y)%mod, a[i+m] = y;
 if (op == 2) a[i] = (x+y)%mod, a[i+m] = (x-y+mod)%mod;
 }
 }
};

Fwht # fe4251

```

```

if (op == 0) a[i] = x, a[i+m] = (y-x+mod)%mod;
if (op == 1) a[i] = (x-y+mod)%mod, a[i+m] = y;
if (op == 2) a[i] = 1LL*(x+y)*inv2%mod, a[i+m] =
1LL*(x-y+mod)*inv2%mod;
}

vector<int> conv(int n, vector<int> A, vector<int> B) {
 A.resize(n); B.resize(n);
 for (int i = 0; i < n; i++) P1[i] = A[i];
 for (int i = 0; i < n; i++) P2[i] = B[i];
 wt(P1, n); wt(P2, n);
 for (int i = 0; i < n; i++) P1[i] = 1LL*P1[i]*P2[i]%mod;
 iwt(P1, n);
 return vector<int> (P1, P1 + n);
}
};

Poly Shift # 981779

```

**Description:** Solves  $f(x + c) = \sum_0^{n-1} b_i \cdot x^i$

**Status:** Tested

```

vector<ll> polyShift(vector<ll> &a, ll shift) {
 // change for any mod for ntt
 const ll mod = 998244353;
 NTT<998244353, 3> ntt;
 int n = a.size() - 1;
 Factorial f(n, mod);
 vector<ll> x(n+1), y(n+1);
 ll cur = 1;
 for (int i = 0; i <= n; i++) {
 x[i] = cur * f.finv[i] % mod;
 cur = cur * shift % mod;
 y[i] = a[n - i] * f.f[n-i] % mod;
 }
 vector<ll> tmp = ntt.conv(x, y, res(n+1));
 for (int i = 0; i <= n; i++)
 res[i] = tmp[n-i] * f.finv[i] % mod;
 return res;
}
};

Poly Shift # 981779

```

**Description:** Computes the first n elements of the inverse series of a polynomial, requires fast\_ntt, and good modulus.

**Status:** Tested on yosupo

```

using ull = uint64_t;
const ull mod = 998244353, pr = 3;
vector<ull> sieveking_kung(vector<ull> a, int n) {
 vector<ull> ans = {bin_pow(a[0], mod-2, mod)};
 fast_ntt<mod, pr> pol;
 for (int i = 1; i < n; i <<= 1) {
 vector<ull>
a_trunc(a.begin(), a.begin() + min((int)a.size(), 2*i));
 vector<ull> res = pol.conv(ans, a_trunc);
 res = pol.conv(res, ans); res.resize(2*i);
 for (int j = 0; j < 2 * i; ++j)
 res[j] = ((j < (int)ans.size()) ? 2*ans[j]:0ULL) + mod *
res[j] % mod;
 ans = res;
 }
 ans.resize(n);
 return ans;
}
};

Sieveking Kung # 4fe9d8

```

**Description:** Using points evaluated at 0,1..n from f(n), interpolates f(x).

Time complexity O(k), where k is the amount of points

**Status:** Tested on codeforces

```

ll lagrange_point(vector<ll> &p, ll x, ll mod){
 int n = p.size();
 if (x < n) return p[x];
 vector<ll> pref(n+1), suff(n+1);
 for (int i = 1; i <= n; i++) pref[i] = (pref[i-1]*(x-
(i-1)))%mod;
 for (int i = n-1; i >= 0; i--) suff[i] = (suff[i+1]*(x-
i))%mod;
 vector<ll> ifact(n);
 ll fact = 1;
 for (ll i = 1; i < n; i++) fact = (fact*i)%mod;
 ifact[0] = 1;
 for (ll i = 1; i < n-2; i++) ifact[i] = (ifact[i-1] * (mod-
i))%mod;
 ll res = 0;
 for (int i = 1; i <= n; i++){
 ll val = (pref[i-1]*suff[i])%mod;
 ll temp = ((n-i)&1)?mod-1:1;
 val = (val*temp)%mod;
 val = (val*ifact[i-1])%mod;
 val = (val*ifact[n-i])%mod;
 val = (val*p[i-1])%mod;
 res = (res+val)%mod;
 }
 if (res < 0) res += mod;
 return res;
}
};

Berlekamp Massey # e57731

```

**Description:** Recovers n-order linear recurrence from first 2n terms. Output size  $\leq n$ .  $O(N^2)$ .

**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) returns {1, 2}

**Status:** KACTL based, not self tested

```

const ll mod = 998244353;
ll mdpow(ll b, ll e, ll m = mod) {
 ll r = 1;
 for (b %= m; e > 0; e >>= 1, b = b*b%m)
 if (e & 1) r = r*b%m;
 return r;
}
vector<ll> berlekampMassey(vector<ll> s) {
 int n = s.size(), L = 0, m = 0;
 vector<ll> C(n), B(n), T;
 C[0] = B[0] = 1;
 ll b = 1;
 for (int i = 0; i < n; i++) { +m;
 ll d = s[i] % mod;
 for (int j = 1; j <= L; j++)
 d = (d + C[j] * s[i-j]) % mod;
 if (!d) continue;
 T = C; ll coef = d * mdpow(b, mod-2) % mod;
 for (int j = m; j < n; j++)
 C[j] = (C[j] - coef * B[j-m]) % mod;
 if (2*L > i) continue;
 L = i+1-L; B = T; b = d; m = 0;
 }
 C.resize(L+1); C.erase(C.begin());
 for (ll &x : C) x = (mod - x) % mod;
 return C;
}
};

Linear Recurrence # 7f4f80

```

**Description:** k-th term of n-order linear recurrence  $S[i] = \sum_j S[i - j - 1] \text{tr}[j]$ , given  $S[0..n-1]$  and  $\text{tr}[0..n-1]$ . Faster than matrix exponentiation.  $O(n^2 \log k)$ .

**Usage:** linearRec({0,1}, {1,1}, k) returns k-th Fibonacci

**Status:** KACTL based, not self tested

```

const ll mod = 998244353;
using Poly = vector<ll>;
ll linearRec(Poly S, Poly tr, ll k) {
 ...
}
};

Linear Recurrence # 7f4f80

```

## Integrate, Point 2D, Segment, Halfplane, Circle, Line Distance, Segment Distance, Convex Hull, Halfplane Intersection

```

int n = tr.size();
auto combine = [&](Poly a, Poly b) {
 Poly res(n*2+1);
 for (int i = 0; i <= n; i++)
 for (int j = 0; j <= n; j++)
 res[i+j] = (res[i+j] + a[i]*b[j]) % mod;
 for (int i = 2*n; i > n; --i)
 for (int j = 0; j < n; j++)
 res[i-1-j] = (res[i-1-j] + res[i]*tr[j]) % mod;
 res.resize(n+1);
 return res;
};

Poly pol(n+1), e(pol);
pol[0] = e[1] = 1;
for (++k; k /= 2) {
 if (k % 2) pol = combine(pol, e);
 e = combine(e, e);
}

ll res = 0;
for (int i = 0; i < n; i++)
 res = (res + pol[i+1] * S[i]) % mod;
return res;
}

```

## Integrate # eebbd4

Description: Adaptive Simpson's rule for numerical integration. Error proportional to  $h^4$ . Status: KACTL based, not self tested

```

template<class F>
ld simpsonRec(F& f, ld a, ld b,
 ld eps, ld S) {
 ld c = (a+b)/2;
 ld S1 = (f(a)+4*f((a+c)/2)+f(c))*(c-a)/6;
 ld S2 = (f(c)+4*f((c+b)/2)+f(b))*(b-c)/6;
 ld T = S1+S2;
 if (abs(T-S) <= 15*eps || b-a < le-10)
 return T + (T-S)/15;
 return simpsonRec(f, a, c, eps/2, S1)
 + simpsonRec(f, c, b, eps/2, S2);
}

template<class F>
ld integrate(ld a, ld b, F f,
 ld eps = le-8) {
 ld S = (f(a)+4*f((a+b)/2)+f(b))*(b-a)/6;
 return simpsonRec(f, a, b, eps, S);
}

```

## Geometry

## Point 2D # 421c69

Description: Just a 2D Point Status: Tested

```

template <typename T> struct Point2D {
 T x, y;
 Point2D() {};
 Point2D(T x_, T y_) : x(x_), y(y_) {};
 Point2D<T> &operator=(Point2D<T> t) {
 x = t.x, y = t.y;
 return *this;
 }
 Point2D<T> &operator+=(Point2D<T> t) {
 x += t.x, y += t.y;
 return *this;
 }
 Point2D<T> operator-(Point2D<T> t) { return {x - t.x, y - t.y}; }
 Point2D<T> operator+(Point2D<T> t) { return {x + t.x, y + t.y}; }
}

```

```

Point2D<T> operator*(T t) { return {x * t, y * t}; }
Point2D<T> operator/(T t) { return {x / t, y / t}; }
Point2D<T> &operator+=(Point2D<T> t) {
 x += t.x, y += t.y;
 return *this;
}
Point2D<T> &operator-=(Point2D<T> t) {
 x -= t.x, y -= t.y;
 return *this;
}
Point2D<T> &operator/=(Point2D<T> t) {
 x /= t.x, y /= t.y;
 return *this;
}
T operator|(Point2D<T> b) { return x * b.x + y * b.y; }
T operator^(Point2D<T> b) { return x * b.y - y * b.x; }
T cross(Point2D<T> a, Point2D<T> b) { return (a - *this) ^ (b - *this); }
T norm() { return (*this) | (*this); }
T sqdist(Point2D<T> b) { return ((*this) - b).norm(); }
double abs() { return sqrt(norm()); }
double proj(Point2D<T> b) { return (*this) | b / b.abs(); }
double angle(Point2D<T> b) {
 return acos((*this) | b) / this->abs() / b.abs();
}
Point2D<T> rotate(T a) {
 return {cosl(a) * x - sinl(a) * y, sinl(a) * x + cosl(a) * y};
}
template <typename T> Point2D<T> operator*(T a, Point2D<T> b) { return b * a; }

```

## Segment # 02f807

Description: Segment implementation Status: Partially tested

```

template<typename T>
struct Segment {
 Point2D<T> P;
 Point2D<T> Q;
 const T INF = numeric_limits<T>::max();
 Segment(Point2D<T> P, Point2D<T> Q) : P(P), Q(Q) {}
 int sign(T x, T eps = 0) { return x > eps ? 1 : x < -eps ? -1 : 0; }
 bool contain(Point2D<T> p, T eps = 0) {
 return ((P - p)|(Q - p)) <= (T)0 and abs(((Q - P)^(p - P))) <= eps;
 }
 bool intersect(Segment<T> b) {
 if (this->contain(b.P) or this->contain(b.Q) or
 b.contain(P) or b.contain(Q))
 return true;
 // change < 0 or <= depending the problem
 return sign(((b.P) - P)^(Q - P))*sign(((b.Q) - P)^(Q - P)) < 0 and
 sign(((P - b.Q)^(b.Q - b.P))*sign(((Q - b.P)^(b.Q - b.P))) < 0;
 }
 Point2D<T> intersection(Segment<T> b) {
 if (this->intersect(b))
 return (((b.Q - b.P)^(Q - b.P))*P + ((P - b.P)^(b.Q - b.P))*Q)/((P - Q)^(b.Q - b.P));
 return {INF, INF};
 }
}

```

Halfplane # 8f4c07

Description: Halfplane Status: Partially tested

```

struct Halfplane {
 Point2D<ld> p, pq;
 ld angle;
 Halfplane() {}
 Halfplane(Point2D<ld> &a, Point2D<ld> &b) : p(a), pq(b - a) {
 angle = atan2l(pq.y, pq.x);
 }
 bool out(Point2D<ld> r) { return ((pq) ^ (r - p)) < -eps; }
 bool operator<(Halfplane &e) { return angle < e.angle; }
 friend Point2D<ld> inter(Halfplane &s, Halfplane &t) {
 ld alpha = ((t.p - s.p) ^ t.pq) / ((s.pq) ^ (t.pq));
 return s.p + (s.pq * alpha);
 }
};

```

## Circle # 234d03

Description: 2D circle, constructors: empty, (center, radius), (a, b, c)  
circumcircle Status: Tested

```

template <typename T> struct circle {
 Point2D<T> c;
 T r;
 circle() : c(0), r(0) {}
 circle(Point2D<T> c_, T r_) : c(c_), r(r_) {}
 circle(Point2D<T> a, Point2D<T> b, Point2D<T> p) {
 Point2D<T> ab = b - a, ap = p - a;
 T d = 2 * (ab ^ ap);
 if (sign(d) == 0) {
 c = Point2D<T>();
 r = 0;
 return;
 }
 T s = ab.norm(), t = ap.norm();
 c = a + Point2D<T>(ap.y * s - ab.y * t, ab.x * t - ap.x * s) / d;
 r = c.sqdist(a);
 r = sqrtl(r);
 }
}

```

## Line Distance # d84fb4

Description: Signed distance from point  $p$  to line through  $a, b$ . Positive on left side seen from  $a$  to  $b$ . Uses Point2D.  
Status: KACTL based, not self tested

```

template<class T>
ld lineDist(Point2D<T> a, Point2D<T> b, Point2D<T> p) {
 return (ld)((b-a)^(p-a)) / (b-a).abs();
}

```

## Segment Distance # 6d1287

Description: Shortest distance from point  $p$  to segment  $s-e$ . Uses Point2D.  
Status: KACTL based, not self tested

```

using P = Point2D<ld>;
ld segDist(P s, P e, P p) {
 if (s.dist(e) < le-18) return (p-s).abs();
 auto d = (e-s).norm();
 auto t = min(d, max(0.0, (p-s)|(e-s)));
 return ((p-s)*d - (e-s)*t).abs() / d;
}

```

## Algorithms

## Convex Hull

# 5662dc

Description: Get the convex hull of the cloud of points, allow collinear points if is needed  
Status: Highly tested

```

template<typename T>
vector<Point2D<T>> convexHull(vector<Point2D<T>> cloud, bool ac = 0) {
 int n = cloud.size(), k = 0;
 sort(cloud.begin(), cloud.end(), [] (Point2D<T> &a, Point2D<T> &b) {
 return a.x < b.x or (a.x == b.x and a.y < b.y);
 });
 if (n <= 2) return cloud;
 bool allCollinear = true;
 for (int i = 2; i < n; ++i) {
 if ((cloud[i] - cloud[0]) ^ (cloud[i] - cloud[0])) != 0 {
 allCollinear = false; break;
 }
 }
 if (allCollinear) return ac ? cloud : vector<Point2D<T>>{cloud[0], cloud.back()};
 auto process = [&](int st, int end, int stp, int t, auto cmp) {
 for (int i = st; i != end; i += stp) {
 while (k >= t and cmp(ch[k-1], ch[k-2], cloud[i])) k--;
 ch[k++] = cloud[i];
 }
 };
 process(0, n, 1, 2, [&](auto a, auto b, auto c) {
 return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
 });
 process(n - 2, -1, -1, k + 1, [&](auto a, auto b, auto c) {
 return ((a - b) ^ (c - b)) < (ac ? 0 : 1);
 });
 ch.resize(k - 1);
 return ch;
}

```

## Halfplane Intersection # 82b7a7

Description: Halfplane intersection  
Status: Partially tested

```

const ld inf = le18;
const ld eps = le-9;
vector<Point2D<ld>> hp_intersect(vector<Halfplane> &H) {
 Point2D<ld> box[4] = {inf, inf}, {-inf, inf}, {-inf, -inf}, {inf, -inf};
 for (int i = 0; i < 4; i++) {
 Halfplane aux(box[i], box[(i + 1) % 4]);
 H.push_back(aux);
 }
 sort(H.begin(), H.end());
 deque<Halfplane> dq;
 int len = 0;
 for (int i = 0; i < H.size(); i++) {
 while (len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2])))
 dq.pop_back(), --len;
 while (len > 1 && H[i].out(inter(dq[0], dq[1])))
 dq.pop_front(), --len;
 if (len > 0 && fabsl(H[i].pq ^ dq[len - 1].pq) < eps) {
 if ((H[i].pq | dq[len - 1].pq) < 0.0)
 return vector<Point2D<ld>>();
 if (H[i].out(dq[len - 1].p))
 dq.pop_back(), --len;
 else
 dq.push_back(H[i]);
 }
 }
}

```

```

 continue;
 }
 dq.push_back(H[i]), ++len;
}
while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2])))
 dq.pop_back(), --len;
while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1])))
 dq.pop_front(), --len;
if (len < 3)
 return vector<Point2D<ld>>();
vector<Point2D<ld>> ret(len);
for (int i = 0; i + 1 < len; i++)
 ret[i] = inter(dq[i], dq[i + 1]);
ret.back() = inter(dq[len - 1], dq[0]);
return ret;
}

```

**Point Inside Polygon** # 12fb92

**Description:** Check if a point is inside, bounds or out of the polygon  
- 0: Bounds  
- 1: Inside  
- -1: Out  
**Status:** Tested

```

template<typename T>
int pointInsidePolygon(vector<Point2D<T>> &P, Point2D<T> q)
{
 int N = P.size(), cnt = 0;
 for(int i = 0; i < N; i++) {
 int j = (i == N-1 ? 0 : i+1);
 Segment<T> s(P[i], P[j]);
 if(s.contains(q)) return 0;
 if(P[i].x <= q.x & q.x < P[j].x & q.cross(P[i], P[j]) < 0) cnt++;
 else if(P[i].x <= q.x & q.x < P[j].x & q.cross(P[i], P[j]) < 0) cnt++;
 }
 return cnt&1 ? 1 : -1;
}

```

**Polygon Area** # 2f7cd4

**Description:** Get the area of a polygon  
- If you want the double of the area, just enable the flag  
**Status:** Highly tested

```

template<typename T>
T polygonArea(vector<Point2D<T>> P, bool x2 = 0) {
 T area = 0;
 for(int i = 0; i < P.size()-1; ++i)
 area += P[i]^*(P[i+1]);
 // Si el primer punto se repite, sacar:
 area += (P.back())*(P.front());
 return abs(area)/(x2 ? 1 : 2);
}

```

**Nearest 2 Points** # fe88e0

**Description:** Using divide and conquer, allow us to know what are the two nearest point between them  
**Status:** Tested on CSES

```

#define sq(x) ((x)*(x))
template <typename T>
pair<Point2D<T>, Point2D<T>>
nearestPoints(vector<Point2D<T>> &P, int l, int r) {
 const T INF = 1e10;
 if (r-l == 1) return {P[l], P[r]};
 if (l >= r) return {{INF, INF}, {-INF, -INF}};
}

```

**Point Inside Polygon, Polygon Area, Nearest 2 Points, Minimum Enclosing Circle, Convex Hull 3D**

```

int m = (l+r)/2;
pair<Point2D<T>, Point2D<T>> D1, D2, D;
D1 = nearestPoints(P, l, m);
D2 = nearestPoints(P, m+1, r);
D = (D1.first.sqdist(D1.second) <= D2.first.sqdist(D2.second) ? D1 : D2);

T d = D.first.sqdist(D.second), x_center = (P[m].x + P[m+1].x)/2;
vector<Point2D<T>> Pk;
for (int i = l; i <= r; i++)
 if (sq(Pk[i].x-x_center) <= d)
 Pk.push_back(Pk[i]);

sort(Pk.begin(), Pk.end(), [](const Point2D<T> p1, const Point2D<T> p2) {
 return p1.y != p2.y ? p1.y < p2.y : p1.x < p2.x;
});

for(int i = 0; i < Pk.size(); ++i) {
 for(int j = i-1; j >= 0; --j) {
 if(sq(Pk[i].y-Pk[j].y) > d) break;
 if(Pk[i].sqdist(Pk[j]) <= D.first.sqdist(D.second))
 D = {Pk[i],Pk[j]};
 }
 for(int j = i+1; j < Pk.size(); ++j) {
 if(sq(Pk[i].y-Pk[j].y) > d) break;
 if(Pk[i].sqdist(Pk[j]) <= D.first.sqdist(D.second))
 D = {Pk[i],Pk[j]};
 }
}

return D;
}

template <typename T>
pair<Point2D<T>, Point2D<T>>
nearestPoints(vector<Point2D<T>> &P) {
 sort(P.begin(), P.end(), [](const Point2D<T> &p1, const Point2D<T> &p2) {
 if (p1.x == p2.x) return p1.y < p2.y;
 return p1.x < p2.x;
 });

 return nearestPoints(P, 0, P.size()-1);
}

```

**Minimum Enclosing Circle** # 57bd558

**Description:** Minimum enclosing circle, Welzl algorithm  $O(n)$  expected

**Status:** Tested

```

circle<ld> minimum_enclosing_circle(vector<Point2D<ld>> p) {
 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
 shuffle(p.begin(), p.end(), rng);
 int n = p.size();
 if (n == 0) return circle<ld>();
 circle<ld> cur(p[0], 0);
 auto out = [&](int i) { return sign(sqrtl(cur.c.sqdist(p[i])) - cur.r) > 0; };
 for (int i = 1; i < n; i++) if (out(i)) {
 cur = circle<ld>(p[i], 0);
 for (int j = 0; j < i; j++) if (out(j)) {
 cur = circle<ld>((p[i] + p[j]) / 2,
 sqrtl(p[i].sqdist(p[j])) / 2);
 for (int k = 0; k < j; k++) if (out(k))
 cur = circle<ld>(p[i], p[j], p[k]);
 }
 }
 return cur;
}

```

**Convex Hull 3D** # c18592

**Description:** 3D Convex Hull using randomized incremental algorithm  
**Time:**  $O(n \log n)$  expected  
**Status:** Tested

```

template <typename T> struct convex_hull_3d {
 using pt = point3d<T>;
 const ld EPS = 1e-9;
 struct edge;
 struct face {
 int a, b, c;
 pt pa, pb, pc, q;
 edge *el, *e2, *e3;
 vector<int> points;
 int dead = 1e9;
 face(int a, int b, int c, pt pa, pt pb, pt pc)
 : a(a), b(b), c(c), pa(pa), pb(pb), pc(pc) {
 q = ((pb - pa)^*(pc - pa));
 el = e2 = e3 = nullptr;
 }
 };
 struct edge {
 edge *rev; face *f;
 };
 vector<pt> p;
 vector<face *> f;
 static void glue(face *F1, face *F2, edge *el, edge *e2)
 {
 el = new edge, e2 = new edge;
 el->rev = e2, e2->rev = el;
 el->f = F2, e2->f = F1;
 }
 void prepare() {
 int n = (int)p.size();
 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
 shuffle(p.begin(), p.end(), rng);
 vector<int> ve = {0};
 for (int i = 1; i < n; i++) {
 if (ve.size() == 1) {
 if ((p[ve[0]] - p[i]).abs() > EPS) ve.push_back(i);
 } else if (ve.size() == 2) {
 if ((p[ve[1]] - p[ve[0]]).cross(p[i] - p[ve[0]]) .abs() > EPS)
 ve.push_back(i);
 } else if (abs((p[i] - p[ve[0]]).dot((p[ve[1]] - p[ve[0]])) .cross(p[ve[2]] - p[ve[0]])) > EPS) {
 ve.push_back(i);
 break;
 }
 }
 assert((int)ve.size() == 4);
 vector<pt> ve2;
 for (int i : ve) ve2.push_back(p[i]);
 reverse(ve.begin(), ve.end());
 for (int i : ve) p.erase(p.begin() + i);
 p.insert(p.begin(), ve2.begin(), ve2.end());
 }
 convex_hull_3d(vector<pt> points) : p(move(points)) {
 int n = p.size();
 prepare();
 vector<face *> new_face(n, nullptr);
 vector<vector<face *>> conflict(n);
 auto add_face = [&](int a, int b, int c) {
 face *F = new face(a, b, c, p[a], p[b], p[c]);
 F->push_back(F);
 }
 }
}
```

```

 return F;
}

face *F1 = add_face(0, 1, 2);
face *F2 = add_face(0, 2, 1);
glue(F1, F2, F1->e1, F2->e3);
glue(F1, F2, F1->e2, F2->e2);
glue(F1, F2, F1->e3, F2->e1);

for (int i = 3; i < n; i++) {
 for (face *F : {F1, F2}) {
 T Q = (p[i] - p[F->a]).dot(F->q);
 if (Q > EPS) conflict[i].push_back(F);
 if (Q >= -EPS) F->points.push_back(i);
 }
}

for (int i = 3; i < n; i++) {
 for (face *F : conflict[i])
 F->dead = min(F->dead, i);
 int v = -1;
 for (face *F : conflict[i]) {
 if (F->dead != i)
 continue;
 int parr[3] = {F->a, F->b, F->c};
 edge *earr[3] = {F->e1, F->e2, F->e3};
 for (int j = 0; j < 3; j++) {
 int j2 = (j + 1)%3;
 if (earr[j]->f->dead > i) {
 face *Fn = new_face(parr[j]) = add_face(parr[j],
 parr[j2], i);
 set_union(F->points.begin(), F->points.end(),
 earr[j]->f->points.begin(), earr[j]->f->points.end(),
 back_inserter(Fn->points));
 Fn->points.erase(stable_partition(Fn->points.begin(), Fn-
 >points.end(),
 [&](int k) {
 return k > i and
 (p[k] - p[Fn-
 a]).dot(Fn->q) > EPS;
 }));
 Fn->points.end());
 for (int k : Fn->points)
 earr[j]->rev->f = Fn;
 Fn->el = earr[j];
 v = parr[j];
 }
 }
 if (v == -1) continue;
 while (new_face[v]->e2 == nullptr) {
 int u = new_face[v]->b;
 glue(new_face[v], new_face[u], new_face[v]->e2,
 new_face[u]->e3);
 v = u;
 }
 f.erase(remove_if(f.begin(), f.end(), [&](face *F) {
 return F->dead < n;
 }), f.end());
 }

 int num_faces() const { return f.size(); }
 T area() {
 T res = 0;
 for (face *F : f)
 res += F->q.abs() / 2.0;
 return res;
 }
 T volume() {
 T vol = 0;
 for (face *F : f)
 vol += F->pa.dot(F->q);
 }
}

```

```

 return abs(vol) / 6.0;
}
vector<array<int, 3>> faces() {
 vector<array<int, 3>> res;
 for (face *F : f)
 res.push_back({F->a, F->b, F->c});
 return res;
}

```

**Order By Angle**

# 63609f

**Description:** Sort the point depending their angle  
**Status:** Not tested

```

template <typename T>
int semiplane(Point2D<T> p) { return p.y > 0 or (p.y == 0 and p.x > 0); }

template <typename T>
void orderByAngle(vector<Point2D<T>> &p) {
 sort(p.begin(), p.end(), [](Point2D<T> &p1, Point2D<T> &p2) {
 int s1 = semiplane(p1), s2 = semiplane(p2);
 return s1 != s2 ? s1 > s2 : (p1.p2) > 0;
 });
}

```

**Order By Slope**

# 6aef32

**Description:** Sort the points depending the angle  
**Status:** Not tested

```

template <typename T>
void orderBySlope(vector<Point2D<T>> &p) {
 sort(p.begin(), p.end(), [](const Point2D<T> &p1, const Point2D<T> &p2) {
 Fraction<T> r1 = Fraction<T>(p1.x, p1.y), r2 =
 Fraction<T>(p2.x, p2.y);
 return r2 < r1;
 });
}

```

**Lattice Point**

# c29984

**Description:** How many integers point are inside or in the bounds of the polygon

- Only works for integers points

**Status:** Tested on CSES

```

pair<ll, ll> latticePoints(vector<Point2D<ll>> &p) {
 P.push_back(P.front());
 ll area = 0, bounds = 0;
 for(int i = 0; i < (int)P.size()-1; ++i) {
 area += P[i]^P[i+1];
 Point2D<ll> p = P[i+1]-P[i];
 bounds += abs(__gcd(p.x, p.y));
 }
 ll inside = (abs(area) - bounds + 2)/2;
 // Dejar el polígono como estaba antes
 P.pop_back();
 return {inside, bounds};
}

```

**Circle Intersection**

# b04fb1

**Description:** Computes the pair of points where two circles intersect. Returns false if no intersection. Uses Point2D.

**Status:** KACTL based, not self tested

```

using P = Point2D<ld>;
bool circleInter(P a, P b, ld r1, ld r2,
 pair<P*, P*>* out) {

```

```

 if (a.sqdist(b) < le-18) return false;
 P vec = b - a;
 ld d2 = vec.norm(), sum = r1+r2, dif = r1-r2,
 p = (d2 + r1*r1 - r2*r2) / (d2*d2),
 h2 = r1*r1 - p*p*d2;
 if (sum*sum < d2 || dif*dif > d2) return false;
 P per = {-vec.y, vec.x};
 P mid = a + vec*p;
 per = per * sqrt(fmax(0, h2) / d2);
 *out = {mid + per, mid - per};
 return true;
}

```

**Circle Tangents**

# b5a94c

**Description:** External tangents of two circles (negate  $r_2$  for internal). Returns 0, 1 or 2 tangents.  $\dots$ .first/.second = tangency points on circle 1/2. Set  $r_2 = 0$  for point tangents. Uses Point2D.  
**Status:** KACTL based, not self tested

```

using P = Point2D<ld>;
vector<pair<P, P>> tangents(P c1, ld r1,
 P c2, ld r2) {
 P d = c2 - c1;
 ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
 if (d2 < le-18 || h2 < 0) return {};
 vector<pair<P, P>> out;
 P perp = {-d.y, d.x};
 for (ld sign : {-1.0, 1.0}) {
 P v = (d*dr + perp*sqrt(h2)*sign) / d2;
 out.push_back({c1 + v*r1, c2 + v*r2});
 }
 if (h2 < le-18) out.pop_back();
 return out;
}

```

**Circle Polygon Intersection**

# 57ae89

**Description:** Area of intersection of a circle (center  $c$ , radius  $r$ ) with a CCW polygon.  $O(n)$ . Uses Point2D.

**Status:** KACTL based, not self tested

```

using P = Point2D<ld>;
ld circlePoly(P c, ld r, vector<P>& ps) {
 auto arg = [](P p, P q) {
 return atan2(p.q, p|q);
 };
 auto tri = [&](P p, P q) {
 auto r2 = r*r/2;
 P d = q - p;
 auto a = (d|p)/d.norm(), b = (p.norm() - r*r)/d.norm();
 auto det = a*a - b;
 if (det <= 0) return arg(p,q) * r2;
 auto s = max(0.0, -a*sqrt(det));
 auto t = min(1.0, -a+sqrt(det));
 if (t < 0 || 1 <= s) return arg(p,q) * r2;
 P u = p + d*s, v = q + d*(t-1);
 return arg(p,u)*r2 + (u^v)/2 + arg(v,q)*r2;
 };
 ld sum = 0;
 int n = ps.size();
 for (int i = 0; i < n; i++) {
 sum += tri(ps[i]-c, ps[(i+1)%n]-c);
 }
 return sum;
}

```

**Polygon Center**

# 744943

**Description:** Returns the centroid (center of mass) of a polygon.  $O(n)$ . Uses Point2D.

**Status:** KACTL based, not self tested

**Rolling Hashing, KMP, Z**

```

using P = Point2D<ld>;
P polygonCenter(vector<P>& v) {
 P res(0, 0); ld A = 0;
 int n = v.size();
 for (int i = 0, j = n-1; i < n; j = i++) {
 res = res + (v[i]+v[j]) * (v[j]-v[i]);
 A += v[j]*v[i];
 }
 return res / A / 3;
}

```

**Polygon Cut**

# 4da950

**Description:** Returns polygon with everything to the left of line  $s \rightarrow e$  cut away. Uses Point2D.

**Status:** KACTL based, not self tested

```

using P = Point2D<ld>;
vector<P> polygonCut(vector<P>& poly, P s, P e) {
 vector<P> res;
 int n = poly.size();
 for (int i = 0; i < n; i++) {
 P cur = poly[i], prev = i ? poly[i-1] : poly.back();
 auto a = s.cross(e, cur), b = s.cross(e, prev);
 if ((a < 0) != (b < 0))
 res.push_back(cur + (prev-cur) * (a / (a-b)));
 if (a < 0) res.push_back(cur);
 }
 return res;
}

```

**Hull Diameter**

# 38b049

**Description:** Returns the two points with max distance on a convex hull (CCW, no collinear points).  $O(n)$ . Uses Point2D.

**Status:** KACTL based, not self tested

```

template<class T>
array<Point2D<T>, 2> hullDiameter(vector<Point2D<T>& S) {
 int n = S.size(), j = n < 2 ? 0 : 1;
 pair<T, array<Point2D<T>, 2> res = {0, {S[0], S[0]}};
 for (int i = 0; i < j;) {
 for (; j = (j+1)%n) {
 res = max(res, {S[i].sqdist(S[j]), {S[i], S[j]}});
 if (((S[j+1]-S[i])^S[j])^(S[i+1]-S[i]) >= 0)
 i++, break;
 }
 return res.second;
 }
}

```

**Strings****Rolling Hashing**

# df98cc

**Description:** Rolling hash for fast substring comparison, single hash function

**Status:** Tested

```

template<class T>
struct rolling_hashing {
 int base, mod;
 vector<long long> p, H;
 int n;
 rolling_hashing(const T &s, int b, int m): base(b),
 mod(m), n(s.size()) {
 p.assign(n+1, 1);
 H.assign(n+1, 0);
 for (int i = 0; i < n; ++i) {
 H[i+1] = (H[i] * base + s[i]) % mod;
 p[i+1] = (p[i] * base) % mod;
 }
 }
}

```

```

int get(int l, int r) {
 int res = (H[r+1] - H[l] * p[r-l+1]) % mod;
 if (res < 0) res += mod;
 return res;
}

```

**KMP**

# 90eb22

**Description:** Find occurrences of a pattern within given text, runs in  $O(n+m)$ , where  $n$  and  $m$  are the lengths of the text and pattern respectively.

**Status:** Tested on CSES

```

// Memory-efficient string matching version
template<class T> struct KMP {
 T pattern; vector<int> lps;
 KMP(T &pat): pattern(pat) {
 lps.resize(pat.size(), 0);
 int len = 0, i = 1;
 while (i < pattern.size()) {
 if (pattern[i] == pattern[len])
 lps[i++] = ++len;
 else {
 if (len != 0) len = lps[len - 1];
 else lps[i++] = 0;
 }
 }
 }
 vector<int> search(T &text) {
 vector<int> matches;
 int i = 0, j = 0;
 while (i < text.size()) {
 if (pattern[j] == text[i]) {
 i++, j++;
 if (j == pattern.size()) {
 matches.push_back(i - j);
 j = lps[j - 1];
 }
 } else {
 if (j != 0) j = lps[j - 1];
 else i++;
 }
 }
 return matches;
 }
}

// Simple version
template<class T>
vector<int> prefix(T &s) {
 vector<int> P(s.size());
 P[0] = 0;
 for (int i = 1; i < s.size(); ++i) {
 P[i] = P[i - 1];
 while (P[i] > 0 && s[P[i]] != s[i])
 P[i] = P[P[i] - 1];
 if (s[P[i]] == s[i])
 ++P[i];
 }
 return P;
}

```

**Z**

# b71846

**Description:** Z-algorithm for string matching, finds all occurrences in  $O(n+m)$

**Status:** Tested

```

struct Z {
 int n, m;
 vector<int> z;
 Z(string s) {
 n = s.size();
 }
}

```

```

z.assign(n, 0);
int l = 0, r = 0;
for (int i = 1; i < n; i++) {
 if (i <= r)
 z[i] = min(r - i + 1, z[i - l]);
 while (i + z[i] < n && s[z[i]] == s[i + z[i]])
 ++z[i];
 if (i + z[i] - l > r)
 l = i, r = i + z[i] - 1;
}
}

Z(string p, string t) {
 string s = p + "#" + t;
 n = p.size();
 m = t.size();
 z.assign(n + m + 1, 0);
 int l = 0, r = 0;
 for (int i = 1; i < n + m + 1; i++) {
 if (i <= r)
 z[i] = min(r - i + 1, z[i - l]);
 while (i + z[i] < n + m + 1 && s[z[i]] == s[i + z[i]])
 ++z[i];
 if (i + z[i] - l > r)
 l = i, r = i + z[i] - 1;
 }
}

void p_in_t(vector<int>& ans) {
 for (int i = n + 1; i < n + m + 1; i++) {
 if (z[i] == n)
 ans.push_back(i - n - 1);
 }
}

```

**Aho Corasick** # 779b08

**Description:** Aho-Corasick algorithm for multiple pattern matching in text  
**Status:** Tested

```

struct AhoCorasick {
 enum {
 alpha = 26, first = 'a'
 }; // change this!
 struct Node {
 // (nmatches is optional)
 int back, next[alpha], start = -1, end = -1, nmatches = 0;
 Node(int v) {
 memset(next, v, sizeof(next));
 }
 };
 vector < Node > N;
 vi backp;
 void insert(string & s, int j) {
 assert(!s.empty());
 int n = 0;
 for (char c: s) {
 int & m = N[n].next[c - first];
 if (m == -1) {
 n = m = sz(N);
 N.emplace_back(-1);
 } else n = m;
 }
 if (N[n].end == -1) N[n].start = j;
 backp.push_back(N[n].end);
 N[n].end = j;
 N[n].nmatches++;
 }
 AhoCorasick(vector < string > & pat): N(1, -1) {
 rep(i, 0, sz(pat)) insert(pat[i], i);
 N[0].back = sz(N);
 N.emplace_back(0);
 }
 queue < int > q;
}

```

```

for (q.push(0); !q.empty(); q.pop()) {
 int n = q.front(), prev = N[n].back;
 rep(i, 0, alpha) {
 int & ed = N[n].next[i], y = N[prev].next[i];
 if (ed == -1) ed = y;
 else {
 N[ed].back = y;
 (N[ed].end == -1 ? N[ed].end : backp[N[ed].start]) = N[y].end;
 N[ed].nmatches += N[y].nmatches;
 q.push(ed);
 }
 }
}
vi find(string word) {
 int n = 0;
 vi res; // ll count = 0;
 for (char c: word) {
 n = N[n].next[c - first];
 res.push_back(N[n].end);
 // count += N[n].nmatches;
 }
 return res;
}
vector < vi > findAll(vector < string > & pat, string word) {
 vi r = find(word);
 vector < vi > res(sz(word));
 rep(i, 0, sz(word)) {
 int ind = r[i];
 while (ind != -1) {
 res[i - sz(pat[ind]) + 1].push_back(ind);
 ind = backp[ind];
 }
 }
 return res;
}

```

**Suffix Array** # 960aaaf

**Description:** Suffix array construction with LCP array using radix sort,  $O(n \log n)$   
**Status:** Tested

```

using ll = long long;
struct SA {
 int n;
 vector<int> C, R, R_, sa, sa_, lcp;
 inline int gr(int i) { return i < n ? R[i] : 0; }
 void csort(int maxv, int k) {
 C.assign(maxv + 1, 0);
 for (int i = 0; i < n; i++) C[gr(i + k)]++;
 for (int i = 1; i < maxv + 1; i++) C[i] += C[i - 1];
 for (int i = n - 1; i >= 0; i--) sa_[-C[gr(sa[i] + k)]] = sa[i];
 sa_.swap(sa_);
 }
 void getSA(vector<int>& s) {
 R = R_ = sa = sa_ = vector<int>(n);
 for (ll i = 0; i < n; i++) sa[i] = i;
 sort(sa.begin(), sa.end(), [&s](int i, int j) { return s[i] < s[j]; });
 int r = R[sa[0]] = 1;
 for (ll i = 1; i < n; i++) R[sa[i]] = (s[sa[i]] != s[sa[i - 1]]) ? ++r : r;
 for (int h = 1; h < n && r < n; h <= 1) {
 csort(r, h);
 csort(r, 0);
 r = R[sa[0]] = 1;
 for (int i = 1; i < n; i++) {
 if (R[sa[i]] != R[sa[i - 1]] || gr(sa[i] + h) != gr(sa[i - 1] + h)) r++;
 }
 }
 }
}

```

```

R_[sa[i]] = r;
R_.swap(R_);
}
void getLCP(vector<int> &s) {
 lcp.assign(n, 0);
 int k = 0;
 for (ll i = 0; i < n; i++) {
 int r = R[i] - 1;
 if (r == n - 1) {
 k = 0;
 continue;
 }
 int j = sa[r + 1];
 while (i + k < n && j + k < n && s[i + k] == s[j + k])
 k++;
 lcp[r] = k;
 if (k) k--;
 }
}
SA(vector<int> &s) {
 n = s.size();
 getSA();
 getLCP(s);
}

```

**Suffix Automaton** # d735bb

**Description:** Suffix automaton (DAWG) for string processing,  $O(n)$  construction  
**Status:** Tested

```

struct SuffixAutomaton {
 struct state {
 int len, link;
 int next[26];
 state(int _len = 0, int _link = -1) : len(_len), link(_link) {
 memset(next, -1, sizeof(next));
 }
 };
 vector<state> st;
 int last;
 SuffixAutomaton() {}
 SuffixAutomaton(const string &s) { init(s); }
 inline int State(int len = 0, int link = -1) {
 st.emplace_back(len, link);
 return st.size() - 1;
 }
 void init(const string &s) {
 st.reserve(2 * s.size());
 last = State();
 for (char c: s)
 extend(c);
 }
 void extend(char _c) {
 int c = _c - 'a', cur = State(st[last].len + 1), P = last;
 while ((P != -1) && (st[P].next[c] == -1)) {
 st[P].next[c] = cur;
 P = st[P].link;
 }
 if (P == -1)
 st[cur].link = 0;
 else {
 int Q = st[P].next[c];
 if (st[P].len + 1 == st[Q].len)
 st[cur].link = Q;
 else {
 int C = State(st[P].len + 1, st[Q].link);
 copy(st[Q].next, st[Q].next + 26, st[C].next);
 while ((P != -1) && (st[P].next[c] == Q)) {
 st[P].next[c] = C;

```

```

 P = st[P].link;
 }
 st[Q].link = st[cur].link = C;
 }
 }
 last = cur;
}

```

**Manacher** # 6abe01

**Description:** Find palindromes centered at  $i$  in  $O(n)$   
**Status:** Tested on CSES

```

template<class T>
struct manacher {
 vector<int> odd, even;
 T s; int n;
 manacher(T &s): s(s), n(s.size()) {
 odd.resize(n);
 even.resize(n);
 for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 1 : min(odd[l+r-i], r+i+1);
 while (0 <= i - k and i + k < n and s[i-k] == s[i+k]) k++;
 odd[i] = k--;
 if (i + k > r) l = i - k, r = i + k;
 }
 for (int i = 0, l = 0, r = -1; i < n; i++) {
 int k = (i > r) ? 0 : min(even[l+r-i+1], r - i + 1);
 while (0 <= i - k - 1 and i + k < n && s[i - k - 1] == s[i + k]) k++;
 even[i] = k--;
 if (i + k > r) l = i - k - 1, r = i + k;
 }
 }
 // Returns the longest palindrome centered at i
 pair<int, int> get(int i) {
 int o = 2 * odd[i] - 1; // Normally centered (Is odd size)
 int e = 2 * even[i]; // Centered to the right
 if (o >= e)
 return {i - odd[i] + 1, i + odd[i] - 1};
 return {i - even[i], i + even[i] - 1};
 }
};

```

**Trie** # c8f186

**Description:** Trie (prefix tree) for string storage and prefix queries  
**Status:** Tested

```

struct trie {
 vector <vector<ll>> tree;
 vector<ll> counts;
 trie() {
 tree.push_back(vector<ll>(26, -1));
 counts.push_back(0);
 }
 void insert(string &s) {
 ll i = 0, u = 0;
 while(i < s.size()){
 char c = s[i];
 if (tree[u][c - 'a'] == -1){
 ll pos = tree.size();
 tree.push_back(vector<ll>(26, -1));
 counts.push_back(0);
 tree[u][c - 'a'] = pos;
 }
 i++;
 u = tree[u][c - 'a'];
 }
 counts[u]++;
 }
}

```

## Min Rotation, LIS, Egg Drop, D&amp;C, Knuth, Li Chao Tree, Li Chao Segment

```

} ll search(string &s){
 ll i = 0, u = 0;
 ll count_ends = 0;
 while(i < s.size()) {
 char c = s[i];
 if (tree[u][c - 'a'] != -1) {
 //If you need to know how many trailing
 //characters it goes through, uncomment.
 //count_ends += counts[u];
 u = tree[u][c - 'a'];
 i++;
 } else break;
 }
 if(i == s.size()) count_ends += counts[u];
 return count_ends;
}

```

## Min Rotation # 184c43

Description: Finds the lexicographically smallest rotation of a string, runs on  $O(n)$ .  
Status: Tested on CSES

```

string minRotation(string &s) {
 int a = 0, N = s.size();
 string res = s; s += s;
 for (int b = 0; b < N; b++) {
 for (int k = 0; k < N; k++) {
 if (a + k == b || s[a + k] < s[b + k]) {
 b += max(0, k - 1); break;
 }
 if (s[a + k] > s[b + k]) {
 a = b; break;
 }
 }
 rotate(res.begin(), res.begin() + a, res.end());
 return res;
 }
}

```

## Dynamic Programming

## LIS # c94f42

Description: Longest increasing subsequence in  $O(n \log n)$ , allow us to recover the sequence  
Status: Highly tested

```

template <class I> vector<int> LIS(const vector<I> &S) {
 if (S.empty())
 return {};
 vector<int> prev(S.size());
 vector<pair<I, int>> res;
 for (int i = 0; i < S.size(); i++) {
 auto it = lower_bound(res.begin(), res.end(), pair<I,
 int>{S[i], i});
 if (it == res.end())
 res.emplace_back(), it = res.end() - 1;
 *it = {S[i], i};
 prev[i] = (it == res.begin()) ? 0 : (it - 1)->second;
 }
 int L = res.size(), cur = res.back().second;
 vector<int> ans(L);
 while (L--)
 ans[L] = cur, cur = prev[cur];
 /* Recover the sequence
 for (int i = 0; i+1 < ans.size(); i++)
 ans[i] = S[ans[i]];
 */
}

```

## Egg Drop # 93d38e

Description: Egg drop: min attempts for  $h$  floors and  $k$  eggs.  
 $O(kh \log h)$ .  
Status: Tested

```

const ll INF = 1e18;

ll egg_drop(int h, int k) {
 vector<vector<ll>> dp(h + 1, vector<ll>(k + 1, INF));
 for (int j = 0; j <= k; j++) dp[0][j] = 0;
 for (int j = 1; j <= k; j++) {
 for (int i = 1; i <= h; i++) {
 int lo = 1, hi = i;
 while (lo <= hi) {
 int mid = (lo + hi) / 2;
 ll br = dp[mid - 1][j - 1], sv = dp[i - mid][j];
 dp[i][j] = min(dp[i][j], 1 + max(br, sv));
 (br < sv) ? lo = mid + 1 : hi = mid - 1;
 }
 }
 }
 return dp[h][k];
}

// O(k*h) alternative: dp[t][k] = max floors with t tries
ll egg_drop_fast(int h, int k) {
 vector<ll> cur(k + 1, 0), nxt(k + 1);
 for (int t = 1; t <= h; t++) {
 for (int j = 1; j <= k; j++) {
 nxt[j] = cur[j - 1] + cur[j] + 1;
 if (nxt[j] >= h) return t;
 }
 swap(cur, nxt);
 }
 return h;
}

```

## D&amp;C # 05a4a8

Description: Divide and conquer DP optimization.  $O(kn^2)$  to  $O(kn \log n)$ .  
Status: Tested

```

const ll INF = 1e18;
int n;
vector<ll> dp, ndp;

ll cost(int l, int r); // define your cost function

void dac_solve(int l, int r, int optl, int opr) {
 if (l > r) return;
 int mid = (l + r) / 2, opt = optl;
 ndp[mid] = INF;
 for (int k = optl; k <= min(mid - 1, opr); k++) {
 ll val = dp[k] + cost(k + 1, mid);
 if (val < ndp[mid]) ndp[mid] = val, opt = k;
 }
 dac_solve(l, mid - 1, optl, opt);
 dac_solve(mid + 1, r, opt, opr);
}

ll dac_dp(int layers) {
 dp.assign(n + 1, INF); dp[0] = 0;
 for (int i = 0; i < layers; i++) {
 ndp.assign(n + 1, INF);
 dac_solve(1, n, 0, n - 1);
 swap(dp, ndp);
 }
}

```

## Knuth # b96d2e

Description: Knuth DP optimization.  $O(n^3)$  to  $O(n^2)$ .  
Status: Tested

```

const ll INF = 1e18;
int n;
vector<vector<ll>> dp;
vector<vector<int>> opt;

ll cost(int i, int j); // define your cost function

ll knuth_dp() {
 dp.assign(n, vector<ll>(n, 0));
 opt.assign(n, vector<int>(n));
 for (int i = 0; i < n; i++) opt[i][i] = i;

 for (int len = 2; len <= n; len++) {
 for (int i = 0; i + len <= n; i++) {
 int j = i + len - 1;
 dp[i][j] = INF;
 for (int k = opt[i][j - 1]; k <= opt[min(i + 1, j)]) {
 ll val = dp[i][k] + (k + 1 <= j ? dp[k + 1][j] : 0)
 + cost(i, j);
 if (val < dp[i][j]) dp[i][j] = val, opt[i][j] = k;
 }
 }
 }
 return dp[0][n - 1];
}

```

## Li Chao Tree # 90cf3d

Description: Dynamically insert lines of the form  $y = a*x + b$  and query for the minimum value at any  $x$  in a fixed interval  $[L, R]$ .  
Status: Partially tested

```

struct Line{
 ll a,b; // ax + b
 Line(ll _a, ll _b) : a(_a), b(_b) {}
 ll eval(ll x){ return a*x + b; }
};

struct li_chao{
 const ll INF = 1e18;
 ll L,R;
 vector<Line> st;
 li_chao(ll l,ll r){
 L = l, R = r+1;
 st = vector<Line>(4*(r - l + 1),Line(0,INF));
 };
 void add(Line nw, ll v, ll l, ll r){
 ll m = (l+r)/2;
 bool lc = nw.eval(l) < st[v].eval(l);
 bool mc = nw.eval(m) < st[v].eval(m);
 if(mc) swap(nw,st[v]);
 if(r - l == 1) return;
 if(lc != mc) add(nw,2*v,l,m);
 else add(nw,2*v + 1,m,r);
 }
 ll get(ll x, ll v, ll l, ll r){
 ll m = (l+r)/2;
 if(r - l == 1) return st[v].eval(x);
 if(x < m) return min(st[v].eval(x),get(x,2*v,l,m));
 return min(st[v].eval(x),get(x,2*v + 1,m,r));
 }
 void add(ll a, ll b){add(Line(a,b),1,L,R);}
};

ll get(ll x){return get(x,1,L,R);}

ll query_segment(ll x, int l, int r, int L, int R) {
 if (l > r || r < L || l > R)
 return INF;
 int m = (l + 1 == r ? l : (l + r) / 2);
 if (l >= L and r <= R) {
 ll ans = line.eval(x);
 if (l < r) {
 if (x <= m && left != nullptr)
 ans = min(ans, left->query_segment(x, l, m, L,
R));
 if (x > m && right != nullptr)
 ans = min(ans, right->query_segment(x, m + 1, r,
L, R));
 }
 return ans;
 }
 ll ans = inf;
 if (max(l, L) <= min(m, R)) {
 if (left == nullptr)
 left = new node({0, inf});
 left->add_segment(nw, l, m, L, R);
 } else {
 if (right == nullptr)
 right = new node(nw);
 else
 right->add_segment(nw, m + 1, r, L, R);
 }
 return;
}

if (max(l, L) <= min(m, R)) {
 if (left == nullptr)
 left = new node({0, inf});
 left->add_segment(nw, l, m, L, R);
}
if (max(m + 1, L) <= min(r, R)) {
 if (right == nullptr)
 right = new node({0, inf});
 right->add_segment(nw, m + 1, r, L, R);
}

ll query_segment(ll x, int l, int r, int L, int R) {
 if (l > r || r < L || l > R)
 return INF;
 int m = (l + 1 == r ? l : (l + r) / 2);
 if (l >= L and r <= R) {
 ll ans = line.eval(x);
 if (l < r) {
 if (x <= m && left != nullptr)
 ans = min(ans, left->query_segment(x, l, m, L,
R));
 if (x > m && right != nullptr)
 ans = min(ans, right->query_segment(x, m + 1, r,
L, R));
 }
 return ans;
 }
 ll ans = inf;
 if (max(l, L) <= min(m, R)) {
 if (left == nullptr)
 left = new node({0, inf});
 left->add_segment(nw, l, m, L, R);
 }
}

```

```

 left = new node({0, inf});
 ans = min(ans, left->query_segment(x, l, m, L, R));
}
if (max(m + 1, L) <= min(r, R)) {
 if (right == nullptr)
 right = new node({0, inf});
 ans = min(ans, right->query_segment(x, m + 1, r, L,
R));
}
return ans;
};

struct LiChaoTree {
 int L, R;
 node *root;
 LiChaoTree()
 : L(numeric_limits<int>::min() / 2),
R(numeric_limits<int>::max() / 2),
root(nullptr) {}
 LiChaoTree(int L, int R) : L(L), R(R) { root = new
node({0, inf}); }
 void add_line(Line line) { root->add_segment(line, L, R,
L, R); }
 // y = mx + b: x in [l, r]
 void add_segment(Line line, int l, int r) {
 root->add_segment(line, L, R, l, r);
 }
 ll query(ll x) { return root->query_segment(x, L, R, L,
R); }
 ll query_segment(ll x, int l, int r) {
 return root->query_segment(x, l, r, L, R);
 }
};

```

### Fast Knapsack # 133a7b

**Description:** Given  $N$  non-negative weights  $w$  and target  $t$ , computes  $\max S \leq t$  that is a subset sum.  $O(N \max(w_i))$ .

**Status:** KACTL based, not self tested

```

int knapsack(vector<int> w, int t) {
 int a = 0, b = 0, n = w.size(), x;
 while (b < n && a + w[b] <= t) a += w[b++];
 if (b == n) return a;
 int m = *max_element(w.begin(), w.end());
 vector<int> u, v(2*m, -1);
 v[a+m-t] = b;
 for (int i = b; i < n; i++) {
 u = v;
 for (x = 0; x < m; x++)
 v[x+w[i]] = max(v[x+w[i]], u[x]);
 for (x = 2*m; ~x > m;) {
 for (int j = max(0, u[x]); j < v[x]; j++)
 v[x-w[j]] = max(v[x-w[j]], j);
 }
 for (a = t; v[a+m-t] < 0; a--) ;
 }
 return a;
}

```

### Combinatorial

#### Weighted Matroid Intersection # 834d3c

**Description:** Maximum weight base in matroid intersection  $O(r^{2n}(O_1+O_2))$ , where  $r$  is the rank of the basis, and  $n$  is the size of the ground set,  $O_1$  is the cost of Oracle query in the first matroid, and  $O_2$  the cost of Oracle query in second matroid.

**Status:** Not tested

### Fast Knapsack, Weighted Matroid Intersection, Simulated Annealing

```

template<
 typename W_t, // weight type
 typename T, // element of the ground set type
 typename Orc1, // Oracle of the first matroid
 typename Orc2 // Oracle of the second matroid
>
vector<vector<T>> weighted_matroid_intersection(const
vector<T> & ground_set, const Orc1 & matroid1, const Orc2 &
matroid2){
 int n = ground_set.size();
 vector<vector<T>> res;
 vector<char> in_set(n, in_matroid1(n), in_matroid2(n));
 vector<pair<W_t,int>> dist(n);
 vector<pair<int,int>> edges;
 vector<int> par(n), l, r;
 l.reserve(n);
 r.reserve(n);
 while (1) {
 res.push_back({});
 for (int i = 0; i < n; i++)
 if (in_set[i])
 res.back().push_back(ground_set[i]);
 Orc1 m1 = matroid1;
 Orc2 m2 = matroid2;
 l.clear();
 r.clear();
 for (int i = 0; i < n; i++)
 if (in_set[i]) {
 m1.add(ground_set[i]);
 m2.add(ground_set[i]);
 l.push_back(i);
 } else {
 r.push_back(i);
 }
 fill(in_matroid1.begin(), in_matroid1.end(), 0);
 fill(in_matroid2.begin(), in_matroid2.end(), 0);
 for (int i : r) {
 in_matroid1[i] =
m1.independed_with(ground_set[i]);
 in_matroid2[i] =
m2.independed_with(ground_set[i]);
 }
 edges.clear();
 for (int i : l) {
 Orc1 m = matroid1;
 for (int j : l)
 if (i != j)
 m.add(ground_set[j]);

 for (int j : r)
 if (m.independed_with(ground_set[j]))
 edges.emplace_back(i, j);
 }
 Orc2 m = matroid2;
 for (int j : l)
 if (i != j)
 m.add(ground_set[j]);

 for (int j : r)
 if (m.independed_with(ground_set[j]))
 edges.emplace_back(j, i);
 }
}

```

```

 for (auto &[v, u] : edges) {
 if (dist[v].first == INF)
 continue;

 int coeff = in_set[v] ? -1 : 1;
 if (dist[u] > pair{dist[v].first + coeff *
ground_set[u].weight, dist[v].second + 1}) {
 par[u] = v;
 dist[u] = {dist[v].first + coeff *
ground_set[u].weight, dist[v].second + 1};
 any = 1;
 }
 if (!any)
 break;
 }
 int finish = -1;
 for (int v : r)
 if (in_matroid2[v] && dist[v].first != INF &&
(finish == -1 || dist[finish] > dist[v]))
 finish = v;
 if (finish == -1)
 break;

 for (; finish != -1; finish = par[finish])
 in_set[finish] ^= 1;
}
return res;
}

```

#### Simulated Annealing # 878c95

**Description:** Optimizes the function eval by exploring the neighbor solutions using simulated annealing.

- time\_limit: The maximum time to run the algorithm.
- t\_i: Initial temperature.
- t\_f: Final temperature.

The algorithm works for both minimization, to maximize you should change  $(\text{old\_val} - \text{new\_val})$  to  $(\text{new\_val} - \text{old\_val})$  and  $(\text{eval\_nxt} < \text{best})$  to  $(\text{eval\_nxt} > \text{best})$

$P(\text{old\_val}, \text{new\_val}, \text{temp})$  computes the probability of accepting a worse solution based on the current temperature.

Usage:

- T is the data type of the answer.
- repr is the data structure of the representation of the solution, it should contain the functions: init, eval, get\_neighbor and rollback.

Tips:

- Always write your intermediate solution to a file periodically (e.g., every 1 million iterations) to ensure partial scores if the process is interrupted.
- To choose a good  $t_i$  and  $\alpha$ , start with a high  $t_0$  and a low  $\alpha$ , then adjust based on how the temperature and energy function behave.

**Status:** Tested with sudoku.

```

template<class T, class repr>
struct simulated_annealing {
 T val;
 double get_time(){ return
chrono::duration<double>(chrono::high_resolution_clock::now().time_since_epoch()).count(); }
 double P(double old_val, double new_val, double temp)
{ return exp((old_val-new_val)/temp); }
 simulated_annealing(double time_limit, double t_i, double
t_f, repr & q){
 q.init();
 double st = get_time(); val = q.eval();
 while (1){
 double elapsed = get_time()-st;
 if (elapsed > time_limit) break;
 double ela_frac = elapsed/time_limit;
 double temp = t_i*pow(t_f/t_i,ela_frac);
 q.get_neighbor();
 T new_val = q.eval();
 if (new_val < val || ((double)rand())/RAND_MAX <
P(val,new_val,temp)){
 val = new_val;
 } else {
 q.rollback();
 }
 }
 }
}

```