University of Copenhagen

3 little 3 late

Asbjørn Lind, Elias Rasmussen Lolck, Thor Vejen Eriksen

NWERC 2025

November 25, 2025

Setup

hash.sh

5246ca
```
# hashes a file, ignoring whitespaces and comments
# use for verifying that code is copied correctly
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |
    cut -c-6
```

vimrc

39eb19
```
se ch=1 ic mouse=a sw=4 ts=4 nu rnu nuw=4 nowrap so=6
    siso=8 fdm=indent fdl=99 tm=100
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space
    :]' \| md5sum \| cut -c-6
vnoremap <silent> p "_dP
vnoremap <silent> <A-Down> :m '>+1<CR>gv=gv
vnoremap <silent> <A-Up> :m '<-2<CR>gv=gv
```

Data_structures

Fenwick tree

Description: Computes prefix sums and single element updates. Uses 0-indexing.
Usage: `Fen f(n); f.update(ind, val); f.query(ind); f.lower_bound(sum);`
Complexity: $\mathcal{O}(\log n)$ per update/query

1743e1
```
struct Fen {
  vector <ll> v;
  Fen(int s) : v(s, 0) { }
  void update(int ind, ll val) {
    for (; ind < (int) v.size(); ind |= ind + 1) v[ind]
    += val;
  }
  ll query(int ind) { // [0, ind), ind < 0 returns 0
    ll res = 0;
    for (; ind > 0; ind &= ind - 1) res += v[ind - 1];
    // operation can be modified
    return res;
  }
  int lower_bound(ll sum) { // returns first i with
    query(i + 1) >= sum, n if not found
    int ind = 0;
    for (int p = 1 << 25; p; p >>= 1) // 1 << 25 can be
    lowered to ceil(log2(v.size()))
      if (ind + p <= (int) v.size() && v[ind + p - 1] <
    sum)
        sum -= v[(ind += p) - 1];
    return ind;
  }
};
```

Heavy Light Decomposition

Description: Splits tree up in heavy and light paths, so a maximum of $\log N$ light paths are on path between two nodes, allows for efficient updates and queries on paths and subtrees at the same time. Code supports any commutative operations on path and queries. If order is important each heavy path should have two segtrees, and process should be updated to use the correct segtree operations.
Usage: Change `MaxSegmentTree<T>` depending on problem.
`HLD<T, VALS_IN_EDGES> hld(G,R); // T should be nodes used in segtree VALS_IN_EDGES should be true if the values are on the edge from the node to the parent. R should be the root of the tree.`
G should be an adjacency list for the tree
change `query_path / modify_node` to match the problem
Complexity: $\mathcal{O}(N + S)$ build time. N is the size of the tree and S is the time to build the Segtree, and $\mathcal{O}(\log N \cdot F)$ per path query where F is the time of the segtree queries.

4fc289
```
template <class T, bool VALS_IN_EDGES> class HLD {
  private:
  int N, R, tim = 0;  // n, root node, time
  vector<vector<int>> adj;
  vector<int> par, siz, depth, rt, pos;  // parent, size
  , depth, root, position arrays
  MaxSegmentTree<T> segtree;             // Modify as
  needed

  /** Compute the size of each subtree and set parent-
  child relationship
   * Subtree of node v corresponds to segment [ pos[v],
  pos[v] + sz[v] ) */
  void dfs_sz(int v) {
    if (par[v] != -1) adj[v].erase(find(adj[v].begin(),
  adj[v].end(), par[v]));
    for (int &u : adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfs_sz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }

  /** Assign positions for nodes
   * Path from v to the last vertex in ascending heavy
  path
       corresponds to [ pos[rt[v]], pos[v] ] */
  void dfs_hld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfs_hld(u);
    }
  }

  /** process all heavy path and combine their results
   */
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v]);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_IN_EDGES, pos[v]);
  }
  public:
  HLD(vector<vector<int>> adj_, int _R)
      : N(adj_.size()), R(_R), adj(adj_), par(N, -1),
  siz(N, 1), depth(N), rt(N),
        pos(N), segtree(N)  // modify as needed
  {
    rt[R] = R;
    dfs_sz(R);
    dfs_hld(R);
  }
  T query_path(int u, int v) {
    T res = 0;  // default value, modify depending on
  problem
    process(u, v, [&](int l, int r) {
      res = max(res, segtree.range_max(l, r + 1));  //
  modify depending on problem
    });
    return res;
  }
  void modify_node(int u, T val) { segtree.set(pos[u],
    val); }
};
```

Li-Chao tree

Description: Contianer of lines, online insertion/querying. Retrieve the line f with minimum $f(x)$ for a given x.
Usage: `LCT lct(n); lct.insert(line, 0, n - 1); lct.query(x, 0, n - 1);`
Complexity: $\mathcal{O}(\log n)$ per insertion/query

f60397
```
struct Line { ll a, b; ll f(ll x) { return a * x + b; }
    };
constexpr const Line LINF { 0, 1LL << 60 };
struct LCT {
  vector <Line> v; // coord-compression: modify v[x] ->
    v[conert(x)]
  LCT(int size) { v.resize(size, LINF); }
  void insert(Line line, int l, int r) {
    if (l > r) return;
    int mid = (l + r) >> 1;
    if (line.f(mid) < v[mid].f(mid)) swap(line, v[mid]);
    if (line.f(l) < v[mid].f(l)) insert(line, l, mid -
    1);
    else insert(line, mid + 1, r);
  }
  Line query(int x, int l, int r) {
    if (l > r) return LINF;
    int mid = (l + r) >> 1;
    if (x == mid) return v[mid]; // faster on avg. - not
    necessary
    if (x < mid) return best_of(v[mid], query(x, l, mid
    - 1), x);
    return best_of(v[mid], query(x, mid + 1, r), x);
  }
  Line best_of(Line a, Line b, ll x) { return a.f(x) < b
    .f(x) ? a : b; }
};
```

Range Minimum Queries

Description: [kactl] Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: RMQ rmq(values); rmq.query(inclusive, exclusive);
Complexity: $O(|V| \log |V| + Q)$

efbc6a
```
template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= (int)V.size(); pw
    *= 2, ++k) {
      jmp.emplace_back(V.size() - pw * 2 + 1);
      for (int j = 0; j < (int)jmp[k].size(); j++)
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw
    ]);
    }
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

Fast hash map

Description: 3x faster hash map, 1.5x more memory usage, similar API to std::unordered_map. Initial capacity, if provided, must be power of 2.
Usage: `hash_map <key_t, val_t> mp; mp[key] = val; mp.find(key); mp.begin(); mp.end(); mp.erase(key); mp.size();`
Complexity: $\mathcal{O}(1)$ per operation on average.

c7be5a

```
d41d8c// #include <bits/extc++.h>
d41d8c
75f3c2struct chash {
0d8969  const uint64_t C = ll(4e18 * acos(0)) | 71;
16eb60  ll operator () (ll x) const { return __builtin_bswap64
        (x * C); }
cdd37e};
cdd37e
c7be5atemplate <typename KEY_T, typename VAL_T> using hash_map
        = __gnu_pbds::gp_hash_table <KEY_T, VAL_T, chash>;
```

2D Fenwick Tree

Description: [kactl] Computes sums a[i,j] for all i¡I, j¡J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call fakeUpdate() before init()).
Complexity: $O(\log^2 N)$. (Use persistent segment trees for $O(\log N)$.)

1f913d

```
d41d8c// #include "FenwickTree.h"
d41d8c
9a350estruct FT2 {
d07a61  vector<vector<int>> ys; vector<FT> ft;
eab342  FT2(int limx) : ys(limx) {}
5192fd  void fakeUpdate(int x, int y) {
ab24a6    for (; x < (int)ys.size(); x |= x + 1) ys[x].
        push_back(y);
8debff  }
1a1e61  void init() {
0f7c18    for (auto& v : ys) sort(all(v)), ft.emplace_back(v.
        size());
7802af  }
622ba4  int ind(int x, int y) {
06c809    return (int)(lower_bound(all(ys[x]), y) - ys[x].
        begin()); }
600ce8  void update(int x, int y, ll dif) {
d98d54    for (; x < (int)ys.size(); x |= x + 1)
0f0032      ft[x].update(ind(x, y), dif);
9f67de  }
e35066  ll query(int x, int y) {
4291eb    ll sum = 0;
f9d14a    for (; x; x &= x - 1)
f0764d      sum += ft[x-1].query(ind(x-1, y));
89e0a0    return sum;
c86aec  }
1f913d};
```

Line Container

Description: [kactl] Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Complexity: $\mathcal{O}(\log N)$

8ec1c7

```
72c11fstruct Line {
14ce9c  mutable ll k, m, p;
0c4e40  bool operator<(const Line& o) const { return k < o.k;
        }
0dcc67  bool operator<(ll x) const { return p < x; }
7e3ecf};
7e3ecf
746fa4struct LineContainer : multiset<Line, less<>> {
746fa4  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
a3ffb4  static const ll inf = LLONG_MAX;
671986  ll div(ll a, ll b) { // floored division
fa88a2    return a / b - ((a ^ b) < 0 && a % b); }
1a98a7  bool isect(iterator x, iterator y) {
333497    if (y == end()) return x->p = inf, 0;
1202d3    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
d6d755    else x->p = div(y->m - x->m, x->k - y->k);
846095    return x->p >= y->p;
31f5a2  }
4fa010  void add(ll k, ll m) {
ebc1d3    auto z = insert({k, m, 0}), y = z++, x = y;
e189b8    while (isect(y, z)) z = erase(z);
56fc3e    if (x != begin() && isect(--x, y)) isect(x, y =
        erase(y));
6dc2b6    while ((y = x) != begin() && (--x)->p >= y->p)
3f513b      isect(x, erase(y));
4e2c33  }
809d2d  ll query(ll x) {
d8b625    assert(!empty());
143476    auto l = *lower_bound(x);
8818ad    return l.k * x + l.m;
5a0881  }
8ec1c7};
```

Persistent segment tree

Description: Zero-indexed, bounds are [l, r), operations can be modified. `update(...)` returns a pointer to a new tree with the applied update, all other trees remain unchanged. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
Usage: `Node* root = build(arr, 0, n); Node* another_root = update(root, ind, val, 0, n); query(some_root, l, r, 0, n).val; Node* empty_root = nullptr; Node* another_version = update(empty_root, ind, val, 0, n);`
Complexity: $\mathcal{O}(\log n)$ per update/query, $\mathcal{O}(n)$ per build

3237d5

```
bf28eastruct Node {
24f2c2  Node* l,* r;
1eddf6  int val; // i.e. data
9f97da  Node(int _v) : l(nullptr), r(nullptr), val(_v) { }
ad01ea  Node(Node* _l, Node* _r) : l(_l), r(_r), val(0) {
ad01ea    // i.e. merge two nodes:
6cb990    if (l) val += l->val;
bdea62    if (r) val += r->val;
97b9e8  }
089802};
089802
089802// slightly more memory, much faster:
3e798etemplate <typename... ARGS> Node* new_node(ARGS&&...
        args) {
196c33  static deque <Node> pool;
17bd12  pool.emplace_back(forward <ARGS> (args)...);
cc621a  return &pool.back();
b16dc2}
b16dc2// slightly less memory, much slower:
b16dc2// #define new_node(...) new Node(__VA_ARGS__)
b16dc2
b16dc2// optional:
a8e5c9Node* build(const vector <int>& a, int l, int r) {
085265  if (!(r - l - 1)) return new_node(a[l]);
c5e761  int mid = (l + r) >> 1;
80c83f  return new_node(build(a, l, mid), build(a, mid, r));
7b790d}
7b790d
7b790d// can be called with node == nullptr
9954a1Node* update(Node* node, int ind, int val, int l, int r)
        {
f8778c  if (!(r - l - 1)) return new_node(val); // i.e. point
        update
2b5823  int mid = (l + r) >> 1;
7c550e  Node* lf = node ? node->l : nullptr;
28db3c  Node* rg = node ? node->r : nullptr;
d13bbf  return new_node
496f9c    (ind < mid ? update(lf, ind, val, l, mid) : lf,
8e33d4    ind >= mid ? update(rg, ind, val, mid, r) : rg);
7d1cf8}
7d1cf8
ea439dNode query(Node* node, int tl, int tr, int l, int r) {
d3c68e  if (l >= tr || r <= tl || !node) return Node(0); // i.
        e. empty node
24ae6b  if (l >= tl && r <= tr) return *node;
27c8e9  int mid = (l + r) >> 1;
162e7e  Node lf = query(node->l, tl, tr, l, mid);
961e8a  Node rg = query(node->r, tl, tr, mid, r);
39468c  return Node(&lf, &rg);
3237d5}
```

Treap

Description: [kactl] A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Complexity: $O(\log N)$

1754b4

```
bf28eastruct Node {
09cf42  Node *l = 0, *r = 0;
6098a7  int val, y, c = 1;
1e3bd6  Node(int val) : val(val), y(rand()) {}
829930  void recalc();
daabb7};
daabb7
6c5593int cnt(Node* n) { return n ? n->c : 0; }
371cf9void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
371cf9
6b5795template<class F> void each(Node* n, F f) {
19c27d  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
cfbf7f}
cfbf7f
0d52f8pair<Node*, Node*> split(Node* n, int k) {
818a92  if (!n) return {};
38e9ec  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound
        (k)
ffd414    auto [L,R] = split(n->l, k);
d0f96d    n->l = R;
a93244    n->recalc();
2a2dae    return {L, n};
d87ec3  } else {
f6cf62    auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and
        just "k"
b25feb    n->r = L;
08a8e8    n->recalc();
2efe20    return {n, R};
163068  }
b242de}
b242de
27f149Node* merge(Node* l, Node* r) {
34dd9c  if (!l) return r;
917f04  if (!r) return l;
907de0  if (l->y > r->y) {
67d816    l->r = merge(l->r, r);
7199b3    return l->recalc(), l;
27ef3f  } else {
f27aa8    r->l = merge(l, r->l);
ffc207    return r->recalc(), r;
d588a0  }
a1f8a8}
a1f8a8
ba8befNode* ins(Node* t, Node* n, int pos) {
28b80c  auto [l,r] = split(t, pos);
6edc77  return merge(merge(l, n), r);
47352e}
47352e
```

```
47352e// Example application: move the range [l, r) to index k
43d58dvoid move(Node*& t, int l, int r, int k) {
dcf85b  Node *a, *b, *c;
b656e0  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
d864ac  if (k <= l) t = merge(ins(a, b, k), c);
5ef57f  else t = merge(a, ins(c, b, k - r));
1754b4}
```

Union Find with Rollback

Description: [kactl] Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: int t = uf.time(); ...; uf.rollback(t);
Complexity: $O(\log(N))$

b257a9

```
47a5e9struct RollbackUF {
09387e  vector<int> e; vector<pair<int, int>> st;
297ebb  RollbackUF(int n) : e(n, -1) {}
19d0f4  int size(int x) { return -e[find(x)]; }
e78bd7  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
1c6062  int time() { return st.size(); }
fdd411  void rollback(int t) {
809a58    for (int i = time(); i --> t;)
81fe5f      e[st[i].first] = st[i].second;
dc2c29    st.resize(t);
f824b7  }
cb8e6e  bool join(int a, int b) {
460ce9    a = find(a), b = find(b);
0787dc    if (a == b) return false;
02e7c7    if (e[a] > e[b]) swap(a, b);
2440c5    st.push_back({a, e[a]});
b52c51    st.push_back({b, e[b]});
124478    e[a] += e[b]; e[b] = a;
4379f7    return true;
515827  }
b257a9};
```

Wavelet tree

Description: Taken from `https://ideone.com/Tkters`. k-th smallest element in a range. Count number of elements less than or equal to k in a range. Count number of elements equal to k in a range.
Usage: `wavelet_tree wt(arr, arr+n, 1, 1000000000); wt.kth(l, r, k); wt.LTE(l, r, k); wt.count(l, r, k);`
Complexity: $\mathcal{O}(\log n)$ per query

364273

```
137ebfstruct wavelet_tree{
2f784e  #define vi vector<int>
6a3389  #define pb push_back
bd5515  int lo, hi;
441687  wavelet_tree *l, *r;
d7a498  vi b;
d7a498
d7a498  //nos are in range [x,y]
d7a498  //array indices are [from, to)
4907d3  wavelet_tree(int *from, int *to, int x, int y){
50c38b    lo = x, hi = y;
15e543    if(lo == hi or from >= to) return;
034eb1    int mid = (lo+hi)/2;
276c4a    auto f = [mid](int x){
4d4ca8      return x <= mid;
dc9b96    };
290aa3    b.reserve(to-from+1);
80c53a    b.pb(0);
55caf2    for(auto it = from; it != to; it++)
9e0a5f      b.pb(b.back() + f(*it));
9e0a5f    //see how lambda function is used here
f87134    auto pivot = stable_partition(from, to, f);
834105    l = new wavelet_tree(from, pivot, lo, mid);
765e4a    r = new wavelet_tree(pivot, to, mid+1, hi);
eea856  }
```

```
eea856
eea856  //kth smallest element in [l, r]
6a485a  int kth(int l, int r, int k){
161294    if(l > r) return 0;
000e05    if(lo == hi) return lo;
515897    int inLeft = b[r] - b[l-1];
1c793f    int lb = b[l-1]; //amt of nos in first (l-1) nos
          that go in left
5207bc    int rb = b[r]; //amt of nos in first (r) nos that go
          in left
491f0c    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
ba11bf    return this->r->kth(l-lb, r-rb, k-inLeft);
408cd0  }
408cd0
408cd0  //count of nos in [l, r] Less than or equal to k
d6b496  int LTE(int l, int r, int k) {
56eb2f    if(l > r or k < lo) return 0;
5c546e    if(hi <= k) return r - l + 1;
b5a26e    int lb = b[l-1], rb = b[r];
9638eb    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb
          , r-rb, k);
b8e885  }
b8e885
b8e885  //count of nos in [l, r] equal to k
59067a  int count(int l, int r, int k) {
431d4b    if(l > r or k < lo or k > hi) return 0;
49fc8e    if(lo == hi) return r - l + 1;
1dcf86    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
6c2de0    if(k <= mid) return this->l->count(lb+1, rb, k);
d7dcf8    return this->r->count(l-lb, r-rb, k);
de1518  }
c5a5e8  ~wavelet_tree(){
d00d14    delete l;
80917d    delete r;
98e8a4  }
364273};
```

Graph

2SAT

Description: [kactl] Classic 2sat. Negated variables are represented by bit-inversions (`~x`).
Usage: TwoSat ts(number of boolean variables) ts.implies(0, ~3); // Var 0 is true implies Var 3 is false ts.setValue(2); // Var 2 is true ts.solve(); // Returns true iff solvable ts.values[0..N-1] holds the assigned values of the vars
Complexity: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of implications.

687afd

```
d9d94estruct TwoSat {
257c73  int N;
acaed1  vector<vector<int>> gr;
e3b414  vector<int> values; // 0 = false, 1 = true
e3b414
1db182  TwoSat(int n = 0) : N(n), gr(2*n) {}
1db182
456e83  int addVar() { // (optional)
980100    gr.emplace_back();
dbc033    gr.emplace_back();
89ea35    return N++;
7cd843  }
7cd843
6884ef  void implies(int f, int j) {
675b93    f = max(2*f, -1-2*f);
fd1f51    j = max(2*j, -1-2*j);
25d911    gr[f].push_back(j);
44876d    gr[j^1].push_back(f^1);
586863  }
```

```
d49b70  void setValue(int x) { implies(~x, x); }
d49b70
ac3612  vector<int> val, comp, z;
21be16  int time = 0;
da8762  int dfs(int i) {
e1f921    int low = val[i] = ++time, x; z.push_back(i);
91f364    for(int e : gr[i]) if (!comp[e])
088468      low = min(low, val[e] ?: dfs(e));
ef3d1d    if (low == val[i]) do {
a40d63      x = z.back(); z.pop_back();
84ea57      comp[x] = low;
342697      if (values[x>>1] == -1)
b29446        values[x>>1] = x&1;
70a8c0    } while (x != i);
8e9386    return val[i] = low;
d347bc  }
d347bc
f87746  bool solve() {
e3fee0    values.assign(N, -1);
5af767    val.assign(2*N, 0); comp = val;
fa7f60    for (int i = 0; i < 2 * N; i++) if (!comp[i]) dfs(i)
          ;
fe9261    for (int i = 0; i < N; i++) if (comp[2*i] == comp[2*
          i+1]) return 0;
e73e36    return 1;
de6a95  }
687afd};
```

DFS matching

Description: [kactl] Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: vector¡int¿ btoa(m, -1); dfsMatching(g, btoa);
Complexity: $\mathcal{O}(VE)$

6ffaed

```
14da59bool find(int j, vector<vector<int>>& g, vector<int>&
          btoa, vector<int>& vis) {
f96d52  if (btoa[j] == -1) return 1;
fdd1e6  vis[j] = 1; int di = btoa[j];
9e1dc8  for (int e : g[di])
819d84    if (!vis[e] && find(e, g, btoa, vis)) {
8c5b10      btoa[e] = di;
288309      return 1;
7152d2    }
787ed6  return 0;
7004b6}
7004b6
a5bc87int dfsMatching(vector<vector<int>>& g, vector<int>&
          btoa) {
6bfc1b  vector<int> vis;
26cf3b  for (int i = 0; i < (int)g.size(); i++) {
220e30    vis.assign(btoa.size(), 0);
4d977a    for (int j : g[i])
7305e1      if (find(j, g, btoa, vis)) {
0c039d        btoa[j] = i;
04ba9c        break;
48b242      }
6a722f  }
1fa635  return btoa.size() - (int)count(btoa.begin(), btoa.end
          (), -1);
6ffaed}
```

Lowest Common Ancestor

Description: [kactl] Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Complexity: $O(N \log N + Q)$

88b441

```
d41d8c// #include "../data-structures/RMQ.h"
d41d8c
33e98dstruct LCA {
818206  int T = 0;
27f863  vector<int> time, path, ret;
b6da25  RMQ<int> rmq;
b6da25
c9cdfd  LCA(vector<vector<int>>& C) : time(C.size()), rmq((dfs
          (C,0,-1), ret)) {}
bfce37  void dfs(vector<vector<int>>& C, int v, int par) {
cd8a38    time[v] = T++;
4602c1    for (int y : C[v]) if (y != par) {
514920      path.push_back(v), ret.push_back(time[v]);
744afb      dfs(C, y, v);
223aa8    }
c9c425  }
c9c425
5f0389  int lca(int a, int b) {
71b1cb    if (a == b) return a;
651e6e    tie(a, b) = minmax(time[a], time[b]);
e36be8    return path[rmq.query(a, b)];
c2f2e7  }
88b441};
```

Strongly Connected Components

Description: [kactl] Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: scc(graph, [&](vi& v) ...) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.
Complexity: $\mathcal{O}(E + V)$

```
b4e965
b04982vector<int> val, comp, z, cont;
4df60fint Time, ncomps;
29453ftemplate<class G, class F> int dfs(int j, G& g, F& f) {
1185da  int low = val[j] = ++Time, x; z.push_back(j);
952f7a  for (auto e : g[j]) if (comp[e] < 0)
887fdf    low = min(low, val[e] ?: dfs(e,g,f));
887fdf
ac52b9  if (low == val[j]) {
4a98e4    do {
e84f5d      x = z.back(); z.pop_back();
956c36      comp[x] = ncomps;
b2c14e      cont.push_back(x);
c0f991    } while (x != j);
d2742b    f(cont); cont.clear();
4b9f39    ncomps++;
a7f82f  }
495602  return val[j] = low;
9dea3d}
ff80b2template<class G, class F> void scc(G& g, F f) {
1bcd05  int n = g.size();
727cbc  val.assign(n, 0); comp.assign(n, -1);
b42fc9  Time = ncomps = 0;
2d2858  for (int i = 0; i < n; i++) if (comp[i] < 0) dfs(i, g,
          f);
b4e965}
```

Articulation points and Bridges

Description: Finds articulation point and bridges in an undirected graph
Usage: `cutpoints(G)`
G should be an undirected unweighted adjacencylist. art[i] is 1 if node i is an articulation point brd contains a list of edges that are bridges (The edges are not necessarily given with the correct orientation)
Complexity: $\mathcal{O}(N + E)$, where N is the number of nodes, and E is the number of edges.

```
b1c04a
d26414vector<int> lw, nm, pa, art;
561ea9vector<pair<int, int>> brd;
c5abfeint tt, ch, rt;
c5abfe
b41f22void f(int u, const vector<vector<int>> &G) {
0d52be  lw[u] = nm[u] = tt++;
97ca8e  for(int v : G[u]) {
7fc934    if(!nm[v]) {
be65cf      ch += (pa[v] = u) == rt;
0ce899      f(v, G);
d3a414      art[u] = lw[v] >= nm[u];
1132c9      if(lw[v] > nm[u]) brd.emplace_back(u, v);
4ee09e      lw[u] = min(lw[u], lw[v]);
fb6793    }
a90199    else if(v != pa[u]) lw[u] = min(lw[u], nm[v]);
b19853  }
115d70}
115d70
d2205fvoid cutpoints(const vector<vector<int>> &G) {
ab5749  int n = G.size();
878ea5  art.assign(n, 0);
1648f8  lw.assign(n, 0);
3c9b13  nm.assign(n, 0);
87809e  pa.assign(n, -1);
495a7f  brd.clear();
d2822a  tt = 1;
4ff71c  for(int i = 0; i < n; ++i)
6968b0    if(!nm[i]) {
ea7e84      rt = i, ch = 0;
83fbbb      f(i, G);
e35ad9      art[rt] = ch > 1;
339ea8    }
b1c04a}
```

Bellman Ford

Description: [kactl] Calculates shortest paths from "s" in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < \sim 2^{63}$.
Complexity: $\mathcal{O}(VE)$

```
71a596
f5e3e7const ll inf = LLONG_MAX;
5567e9struct Ed { int a, b, w, s() { return a < b ? a : -a;
        }};
2045f7struct Node { ll dist = inf; int prev = -1; };
2045f7
019c78void bellmanFord(vector<Node>& nodes, vector<Ed>& eds,
        int s) {
ec0b61  nodes[s].dist = 0;
1ecaa3  sort(eds.begin(), eds.end(), [](Ed a, Ed b) { return a
        .s() < b.s(); });
1ecaa3
111794  int lim = nodes.size() / 2 + 2; // /3+100 with
        shuffled vertices
503e7b  for (int i = 0; i < lim; i++) for (Ed ed : eds) {
214c1c    Node cur = nodes[ed.a], &dest = nodes[ed.b];
be15e9    if (abs(cur.dist) == inf) continue;
2bf0c3    ll d = cur.dist + ed.w;
82f784    if (d < dest.dist) {
bf8441      dest.prev = ed.a;
e56662      dest.dist = (i < lim-1 ? d : -inf);
1dc21c    }
39b23a  }
9061e4  for (int i = 0; i < lim; i++) for (Ed e : eds) {
bcdab2    if (nodes[e.a].dist == -inf)
40d057      nodes[e.b].dist = -inf;
6e8b4c  }
71a596}
```

Biconnected Components

Description: Finds all biconnected components in an undirected graph, and construct the block-cut tree of the graph. In a block-cut tree each node correspond to a biconnected component or an articualtion point. Checking if there is a path from u to v not going through w in the graph then correspond to checking if there exists a path from id[u] to id[v] in the block-cut tree not going through id[w].
Usage: `G should be an adjaceny list with edges both ways vector<bool> art; vector<int> id; auto T = bcomps(G, art, id) then T is the block-cut tree.`
Complexity: $\mathcal{O}(E + V)$

```
af555c
fbe043vector<vector<int>> bcomps(vector<vector<int>> &G,
        vector<bool> &art, vector<int> &id) {
72d6f2  int n = G.size();
1e9122  vector<vector<int>> cmps, t;
77b695  vector<int> s, nm(n, 0), lw(n, 0);
1c234b  art.resize(n);
aa0037  id.resize(n);
a3e9c0  int tt = 0;
f45022  auto f = [&](auto &&self, int u, int p) -> void {
0b98b4    lw[u] = nm[u] = ++tt;
4592c3    s.push_back(u);
83b8dc    for(int v : G[u]) {
481639      if(v == p) continue;
3cd4fd      if(nm[v]) lw[u] = min(lw[u], nm[v]);
da1a15      else {
c915c2        self(self, v, u);
53f70c        lw[u] = min(lw[u], lw[v]);
725a94        if(lw[v] >= nm[u]) {
696d29          art[u] = (nm[u] > 1 || nm[v] > 2);
a2a70a          cmps.push_back({u});
49c8ce          while(cmps.back().back() != v) {
2a6b39            cmps.back().push_back(s.back());
daedb8            s.pop_back();
220568          }
395ea3        }
348215      }
5c9444    }
d415a6  };
552f68  for(int i = 0; i < n; ++i) if(!nm[i]) {
8fba5a    tt = 0;
5a7d40    f(f, i, -1);
69b077  }
888bf5  int ni = 0;
68519d  for(int i = 0; i < n; ++i) if(art[i]) {
a26b19    id[i] = ni++;
8b59bc    t.push_back({});
0fc78e  }
2122e6  for(auto &c : cmps) {
2254bb    int u = ni++;
036368    t.push_back({});
7ff747    for(int v : c) {
14ea4c      if(!art[v]) id[v] = u;
26fa22      else {
78a18f        t[u].push_back(id[v]);
544ab7        t[id[v]].push_back(u);
d50f49      }
d55ab1    }
dc536a  }
263eef  return t;
af555c}
```

Binary Lifting

Description: [kactl] Calculate power of two jumps in a tree. Assumes root node points to itself
Usage: treeJump(parent list); // To get jump table jmp(jump table, v, k); // Get k'th ancestor of v lca(jumpt table, depth list, a, b); // Get lowest common ancestor of a and b

Complexity: construction $O(N \log N)$, queries $O(\log N)$

```
aec6cd
0ec025 vector<vector<int>> treeJump(vector<int>& P){
dcf724   int on = 1, d = 1;
801d15   while(on < (int)P.size()) on *= 2, d++;
0dd875   vector<vector<int>> jmp(d, P);
9a891e   for (int i = 1; i < d; i++)
a0a9ef     for (int j = 0; j < (int)P.size(); j++)
d91f9f       jmp[i][j] = jmp[i-1][jmp[i-1][j]];
005456   return jmp;
2ff4c2 }
2ff4c2
85b061 int jmp(vector<vector<int>>& tbl, int nod, int steps){
ca8806   for (int i = 0; i < (int)tbl.size(); i++)
51bc0c     if(steps&(1<<i)) nod = tbl[i][nod];
09c31e   return nod;
7f4e63 }
7f4e63
5c366c int lca(vector<vector<int>>& tbl, vector<int>& depth,
         int a, int b) {
f395df   if (depth[a] < depth[b]) swap(a, b);
8c5c81   a = jmp(tbl, a, depth[a] - depth[b]);
b71ad8   if (a == b) return a;
41358b   for (int i = tbl.size() - 1; ~i; i--) {
759916     int c = tbl[i][a], d = tbl[i][b];
803269     if (c != d) a = c, b = d;
92e5e6   }
eb1ca2   return tbl[0][a];
aec6cd }
```

Centroid decomposition

Description: Computes a centroid decomposition and invokes the given callback in top-down depth-first order. Takes an adjacency list. See comment in case of disconnected graphs.
Usage: `centroid_decomposition(adj, [] (int centroid) { ... }, optional_root);`
Complexity: $\mathcal{O}(n \log n)$ and exactly one callback invocation per vertex

```
f06581
5c9f0c void centroid_decomposition(const std::vector <std::
         vector <int>>& g, std::function <void (int)>& callback
         , int root = 0) {
70e3f7   const int n = g.size();
45a964   std::vector <bool> vis(n, false);
47a2cd   std::vector <int> sub(n);
84f4f8   auto size = [&] (auto&& self, int v, int p = -1) ->
          int {
864e90     sub[v] = 1;
a9f1b2     for (int x : g[v]) if (!vis[x] && x != p) sub[v] +=
          self(self, x, v);
68e984     return sub[v];
6fc26d   };
837008   auto cen = [&] (auto&& self, int ts, int v, int p =
          -1) -> int {
facdd1     for (int x : g[v])
6fbf87       if (!vis[x] && x != p && sub[x] >= ts)
7e9b79         return self(self, ts, x, v);
71c226     return v;
3015ef   };
7e9cd5   auto dfs = [&] (auto&& self, int v) -> void {
2ee12b     int c = cen(cen, size(size, v) >> 1, v);
7dfc26     callback(c);
9217b9     vis[c] = true;
5ce597     for (int x : g[c]) if (!vis[x]) self(self, x);
528d37   };
5a52a5   dfs(dfs, root);
5a52a5   // if g is disconnected, do this instead
5a52a5   // for (int v = 0; v < n; v++) if (!vis[v]) dfs(dfs, v
         );
f06581 }
```

Compress Tree

Description: [kactl] Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S| - 1$) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.
Usage: li = the subset of nodes.
Complexity: $O(|S| \log |S|)$

```
739860
d41d8c // #include "LCA.h"
d41d8c
ffa2cb vector<pair<int, int>> compressTree(LCA& lca, vector<int
         > li) {
1c459d   static vector<int> rev; rev.resize(lca.time.size());
93ff63   vector<int> &T = lca.time;
05a1fa   auto cmp = [&](int a, int b) { return T[a] < T[b]; };
b3a68e   sort(li.begin(), li.end(), cmp);
606467   int m = li.size()-1;
861a50   for (int i = 0; i < m; i++) {
92c897     int a = li[i], b = li[i+1];
8368bc     li.push_back(lca.lca(a, b));
25c364   }
b46935   sort(li.begin(), li.end(), cmp);
c3d1b5   li.erase(unique(li.begin(), li.end()), li.end());
d5bbd4   for (int i = 0; i < m + 1; i++) rev[li[i]] = i;
a71dbd   vector<pair<int, int>> ret = {pair<int, int>(0, li[0])
          };
c66945   for (int i = 0; i < m; i++) {
47af2d     int a = li[i], b = li[i+1];
177378     ret.emplace_back(rev[lca.lca(a, b)], b);
a57581   }
d166c7   return ret;
739860 }
```

Critical nodes

Description: Finds necessary nodes in a directed graph between two cities u, v. That is nodes that appears on every path between u and v
Usage: `critical(G)`
G should be an directed unweighted adjacencylist. returns a list with the indices of the critical nodes. Returns an empty list if u and v are not in the same component. Additionally pt will contain a path from u to v.
Complexity: $\mathcal{O}(N + E)$, where N is the number of nodes, and E is the number of edges.

```
91980e
a59858 vector<int> pt, nx, s1, s2;
a59858
36e303 int f1(int u, int tg, const vector<vector<int>> &G, int
         d = 0) {
a21a3e   if(s1[u]) return 0;
44f377   pt.push_back(u);
cc8ea6   nx[u] = d;
b51f99   s1[u] = 1;
11c4dc   if(u == tg) return 1;
244e8b   for (int v : G[u]) if(f1(v, tg, G, d + 1)) return 1;
cffa2b   pt.pop_back();
9417b3   return nx[u] = 0;
da5e4b }
da5e4b
3c4ca0 int f2(int u, const vector<vector<int>> &G) {
294863   int a = 0;
8a5926   if(s2[u]) return 0;
513f05   s2[u] = 1;
b1247d   for(int v : G[u]) a = max(a, nx[v] ? nx[v] : f2(v, G))
          ;
2882c7   return a;
547daf }
547daf
```

```
ae9591 vector<int> critical(const vector<vector<int>> &G, int u
         , int v) {
940fbe   int n = G.size();
cc34cc   nx.assign(n, 0);
07ddbc   s1.assign(n, 0);
d5a0bd   s2.assign(n, 0);
e57be4   f1(u, v, G);
b9995e   vector<int> art;
be3255   for(int i = 0, j = 0; i < (int)pt.size(); j = max(j,
          f2(pt[i++], G)))
cbd451     if(i == j) art.push_back(pt[i]);
c43572   return art;
91980e }
```

Critical nodes on minimal path

Description: Finds minimal-route necessary nodes in a directed weighted graph between two cities u, v. That is nodes that appears on every minimum-length path between u and v
Usage: `critical(G)`
G should be an directed unweighted adjacencylist. returns a list with the indices of the critical nodes. Returns an empty list if u and v are not in the same component.
Complexity: $\mathcal{O}(N + E)$, where N is the number of nodes, and E is the number of edges.

```
7bc9ff
b6af35 vector<int> critical_minimal(const vector<vector<pair<
         int, int>>> &G, int u, int v) {
648082   int n = G.size();
ac5881   priority_queue<array<ll, 3>> pq;
748fc3   queue<int> q;
cfc332   vector<ll> di(n, -1);
dc7775   vector<int> dg(n, 0), art;
8e166f   set<int> am;
418677   vector<vector<int>> ig(n);
48a8b2   pq.push({0, u, u});
396711   while(pq.size()){
1aa873     auto [d, x, p] = pq.top();
89c670     pq.pop();
651303     if(~di[x]){
fa9295       if(-d == di[x]) ig[x].push_back(p);
fa9c13       continue;
300cdd     }
f91cce     di[x] = -d;
4a1a32     if(x != p) ig[x].push_back(p);
9f73d6     for(auto y : G[x]) pq.push({d - y.second, y.first, x
          });
f8bf2f   }
99fa85   if(!~di[v]) return {};
95ed89   for(int i = 0; i < n; ++i) for(auto x : ig[i]) dg[x
          ]++;
9f23b3   for(int i = 0; i < n; ++i) if(!dg[i]) q.push(i);
c09d27   while(q.size()) {
4d9fe2     auto x = q.front();
ca785d     q.pop();
05ddc8     if(x == v) continue;
594d0a     for(auto y : ig[x]) if(!--dg[y]) q.push(y);
03d5c7   }
0c167a   q.push(v);
7a831c   while(q.size()) {
c3c1ae     auto x = q.front();
10f8ac     q.pop();
3eb8f1     am.erase(x);
20c905     if(!am.size()) art.push_back(x);
84de4c     for(auto y : ig[x]) {
963abd       am.insert(y);
cd9eb4       if(!--dg[y]) q.push(y);
31c43c     }
2aaf9e   }
884d9c   return art;
7bc9ff }
```

Dinic

Description: [kactl] Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$.

Complexity: $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

db429d

```
14df72struct Dinic {
9230ca  struct Edge {
ca825e    int to, rev;
eceace    ll c, oc;
299dbe    ll flow() { return max(oc - c, 0LL); } // if you
           need flows
9d5927  };
fcd83d  vector<int> lvl, ptr, q;
acb492  vector<vector<Edge>> adj;
0445a0  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
47fa48  void addEdge(int a, int b, ll c, ll rcap = 0) {
ae98a8    adj[a].push_back({b, adj[b].size(), c, c});
3d0468    adj[b].push_back({a, adj[a].size() - 1, rcap, rcap})
          ;
c717e4  }
2b9c8c  ll dfs(int v, int t, ll f) {
cd940a    if (v == t || !f) return f;
4f0943    for (int& i = ptr[v]; i < (int)adj[v].size(); i++) {
5956b9      Edge& e = adj[v][i];
0dc357      if (lvl[e.to] == lvl[v] + 1)
4897b7        if (ll p = dfs(e.to, t, min(f, e.c))) {
573fa0          e.c -= p, adj[e.to][e.rev].c += p;
818785          return p;
bccb92        }
41b170    }
adb0f1    return 0;
79fda3  }
0a956a  ll calc(int s, int t) {
67cda4    ll flow = 0; q[0] = s;
0fa931    for (int L = 0; L < 31; L++) do {
f0e6b0      lvl = ptr = vector<int>(q.size());
024194      int qi = 0, qe = lvl[s] = 1;
1ace63      while (qi < qe && !lvl[t]) {
ef60bd        int v = q[qi++];
a5c460        for (Edge e : adj[v])
2ced44          if (!lvl[e.to] && e.c >> (30 - L))
48346c            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
426a65      }
015733      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
f6a4b9    } while (lvl[t]);
6e6677    return flow;
23ce03  }
79aff9  bool leftOfMinCut(int a) { return lvl[a] != 0; }
db429d};
```

Directed Minimum Spanning Tree (int Directed Graph)

Description: [kactl] Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.

Complexity: $\mathcal{O}(E\log V)$

27e676

```
d41d8c// #include "../data-structures/UnionFindRollback.h"
d41d8c
030131struct Edge { int a, b; ll w; };
7519f2struct Node { /// lazy skew heap node
45a8d0  Edge key;
348382  Node *l, *r;
59f245  ll delta;
958c51  void prop() {
c4174f    key.w += delta;
9353bd    if (l) l->delta += delta;
69a899    if (r) r->delta += delta;
cfc93b    delta = 0;
31f792  }
61e0cf  Edge top() { prop(); return key; }
67708e};
d59b55Node *merge(Node *a, Node *b) {
6b68b8  if (!a || !b) return a ?: b;
839210  a->prop(), b->prop();
7c5d9a  if (a->key.w > b->key.w) swap(a, b);
c76878  swap(a->l, (a->r = merge(b, a->r)));
046c62  return a;
5e360c}
821d19void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
821d19
6eb9a8pair<ll, vector<int>> dmst(int n, int r, vector<Edge>& g
       ) {
a0a15d  RollbackUF uf(n);
544201  vector<Node*> heap(n);
ee5419  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node
         {e});
490610  ll res = 0;
811d5d  vector<int> seen(n, -1), path(n), par(n);
bdf234  seen[r] = r;
a31f44  vector<Edge> Q(n), in(n, {-1,-1}), comp;
fbe5f9  deque<tuple<int, int, vector<Edge>>> cycs;
9d15a9  for (int s = 0; s < n; s++) {
42879d    int u = s, qi = 0, w;
c32d1d    while (seen[u] < 0) {
db0047      if (!heap[u]) return {-1,{}};
30f147      Edge e = heap[u]->top();
4fffc1      heap[u]->delta -= e.w, pop(heap[u]);
e10f5c      Q[qi] = e, path[qi++] = u, seen[u] = s;
10c4d1      res += e.w, u = uf.find(e.a);
ddeb26      if (seen[u] == s) { /// found cycle, contract
a470a9        Node* cyc = 0;
035938        int end = qi, time = uf.time();
59f8a2        do cyc = merge(cyc, heap[w = path[--qi]]);
233ca4        while (uf.join(u, w));
b9e8ef        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
600eb8        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
3dc6d7      }
0fad35    }
5f8489    for (int i = 0; i < qi; i++) in[uf.find(Q[i].b)] = Q
           [i];
b50d21  }
b50d21
2a32a4  for (auto& [u,t,comp] : cycs) { // restore sol (
         optional)
a4becb    uf.rollback(t);
7d0a6b    Edge inEdge = in[u];
397083    for (auto& e : comp) in[uf.find(e.b)] = e;
b568b8    in[uf.find(inEdge.b)] = inEdge;
8ba05b  }
cd9dc0  for (int i = 0; i < n; i++) par[i] = in[i].a;
1e59a5  return {res, par};
27e676}
```

Edge Coloring

Description: [kactl] Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

Complexity: $\mathcal{O}(NM)$

f465a3

```
3e791avector<int> edgeColoring(int N, vector<pair<int, int>>
        eds) {
fb404a  vector<int> cc(N + 1), ret(eds.size()), fan(N), free(N
        ), loc;
b665c8  for (auto e : eds) ++cc[e.first], ++cc[e.second];
6f74a5  int u, v, ncols = *max_element(all(cc)) + 1;
3b61b1  vector<vector<int>> adj(N, vector<int>(ncols, -1));
e6b161  for (pair<int, int> e : eds) {
e2b3b5    tie(u, v) = e;
f14049    fan[0] = v;
6c87b4    loc.assign(ncols, 0);
064af9    int at = u, end = u, d, c = free[u], ind = 0, i = 0;
1eae62    while (d = free[v], !loc[d] && (v = adj[u][d]) !=
          -1)
b2a2de      loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
5b3b2c    cc[loc[d]] = c;
e38b69    for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at
          ][cd])
ac4ca8      swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
dbee08    while (adj[fan[i]][d] != -1) {
fb930c      int left = fan[i], right = fan[++i], e = cc[i];
1c8a76      adj[u][e] = left;
aad73b      adj[left][e] = u;
61eb0d      adj[right][e] = -1;
444fd6      free[right] = e;
b6e824    }
c31c10    adj[u][d] = fan[i];
0eac72    adj[fan[i]][d] = u;
e8bfe2    for (int y : {fan[0], u, end})
52dcc8      for (int& z = free[y] = 0; adj[y][z] != -1; z++);
37a668  }
470b03  for (int i = 0; i < (int)eds.size(); i++)
45bafe    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret
          [i];
3f958d  return ret;
f465a3}
```

Euler Walk

Description: [kactl] Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.

Complexity: $\mathcal{O}(V+E)$

f237d8

```
7e2924vector<int> eulerWalk(vector<vector<pair<int, int>>>& gr
       , int nedges, int src=0) {
d91cd4  int n = gr.size();
90184b  vector<int> D(n), its(n), eu(nedges), ret, s = {src};
12987e  D[src]++; // to allow Euler paths, not just cycles
c5e021  while (!s.empty()) {
2ab8ef    int x = s.back(), y, e, &it = its[x], end = gr[x].
          size();
4894b0    if (it == end){ ret.push_back(x); s.pop_back();
          continue; }
6fb520    tie(y, e) = gr[x][it++];
a74b1f    if (!eu[e]) {
957036      D[x]--, D[y]++;
a1212f      eu[e] = 1; s.push_back(y);
58732d    }}
566a79  for (int x : D) if (x < 0 || ret.size() != nedges+1)
          return {};
fa8da4  return {ret.rbegin(), ret.rend()};
f237d8}
```

Floyd Warshall

Description: [kactl] Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] =$ `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.

Complexity: $\mathcal{O}(N^3)$

cf07b8

```cpp
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = m.size();
  for (int i = 0; i < n; i++) m[i][i] = min(m[i][i], 0LL);
  for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        if (m[i][k] != inf && m[k][j] != inf) {
          auto newDist = max(m[i][k] + m[k][j], -inf);
          m[i][j] = min(m[i][j], newDist);
        }
  for (int k = 0; k < n; k++) if (m[k][k] < 0)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

General Matching

Description: [kactl] Matching for general graphs. Fails with probability N/mod.

Complexity: $\mathcal{O}(N^3)$

```cpp
// #include "../numerical/MatrixInverse-mod.h"
vector<pair<int, int>> generalMatching(int N, vector<pair<int, int>>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (auto pa : ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

  int r = matInv(A = mat), M = 2*N - r, fi, fj;
  assert(r % 2 == 0);

  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 r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % 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;
    } assert(0); done:
    if (fj < N) ret.emplace_back(fi, fj);
    has[fi] = has[fj] = 0;
    for (int sw = 0; sw < 2; sw++) {
      ll a = modpow(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 Minimum Cut

Description: [kactl] Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.

Complexity: $\mathcal{O}(V^3)$

```cpp
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++) { // O(V^2) -> O(E log V) with prio. queue
      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[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

Gomory-Hu

Description: [kactl] Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.

Complexity: $O(V)$ Flow Computations

```cpp
// #include "PushRelabel.h"
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vector<int> par(N);
  for (int i = 1; i < N; i++) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    for (int j = i+1; j < N; j++)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

Hungarian

Description: [kactl] Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.

Complexity: $\mathcal{O}(N^2M)$

```cpp
pair<int, vector<int>> hungarian(const vector<vector<int>> &a) {
  if (a.empty()) return {0, {}};
  int n = a.size() + 1, m = a[0].size() + 1;
  vector<int> u(n), v(m), p(m), ans(n - 1);
  for (int i = 1; i < n; i++) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vector<int> dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      for (int j = 1; j < m; j++) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      for (int j = 0; j < m; j++) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  for (int j = 1; j < m; j++) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

Link Cut Tree

Description: [kactl] 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.

Complexity: All operations take amortized $O(\log N)$.

```cpp
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];
      y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
```

```
74bdc4  void splay() { /// Splay this up to the root. Always
         finishes without flip set.
e2fadb    for (pushFlip(); p; ) {
7a5c22      if (p->p) p->p->pushFlip();
6ffcea      p->pushFlip(); pushFlip();
3ef089      int c1 = up(), c2 = p->up();
7d338d      if (c2 == -1) p->rot(c1, 2);
652d9b      else p->p->rot(c2, c1 != c2);
1cf3c8    }
5a4303  }
d0ea9c  Node* first() { /// Return the min element of the
         subtree rooted at this, splayed to the top.
76d573    pushFlip();
ca32fb    return c[0] ? c[0]->first() : (splay(), this);
e95aca  }
225109};
225109
bea8de struct LinkCut {
6c77ba  vector<Node> node;
47ed13  LinkCut(int N) : node(N) {}
47ed13
391c16  void link(int u, int v) { // add an edge (u, v)
661716    assert(!connected(u, v));
14e70f    makeRoot(&node[u]);
aee6c0    node[u].pp = &node[v];
557426  }
d8c18d  void cut(int u, int v) { // remove an edge (u, v)
612611    Node *x = &node[u], *top = &node[v];
bdb8ca    makeRoot(top); x->splay();
37b1c0    assert(top == (x->pp ?: x->c[0]));
33e021    if (x->pp) x->pp = 0;
e75f7f    else {
dec201      x->c[0] = top->p = 0;
e4aaa1      x->fix();
83e12b    }
47de4b  }
1656f9  bool connected(int u, int v) { // are u, v in the same
         tree?
f905e2    Node* nu = access(&node[u])->first();
76020a    return nu == access(&node[v])->first();
6fb75d  }
399bef  void makeRoot(Node* u) { /// Move u to root of
         represented tree.
96cf2a    access(u);
27447c    u->splay();
826b3d    if(u->c[0]) {
4ee3da      u->c[0]->p = 0;
713d12      u->c[0]->flip ^= 1;
3ba226      u->c[0]->pp = u;
e81321      u->c[0] = 0;
248be7      u->fix();
9ee245    }
9af643  }
2bd857  Node* access(Node* u) { /// Move u to root aux tree.
         Return the root of the root aux tree.
b7e9da    u->splay();
3cc96b    while (Node* pp = u->pp) {
9c5fe2      pp->splay(); u->pp = 0;
9cf4aa      if (pp->c[1]) {
f9babf        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
9b8ee5      pp->c[1] = u; pp->fix(); u = pp;
d197c0    }
81e9a2    return u;
c39d6c  }
0fb462};
```

Maximal Cliques

Description: [kactl] Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

Complexity: $\mathcal{O}(3^{n/3})$, much faster for sparse graphs

```
d3d1a9
d41d8c /// Possible optimization: on the top-most
d41d8c /// recursion level, ignore 'cands', and go through
        nodes in order of increasing
d41d8c /// degree, where degrees go down as nodes are removed.
d41d8c /// (mostly irrelevant given MaximumClique)
d41d8c
753236 typedef bitset<128> B;
6454cc template<class F>
05d32c void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B
        R={}) {
d462aa  if (!P.any()) { if (!X.any()) f(R); return; }
abbe26  auto q = (P | X)._Find_first();
01a6f3  auto cands = P & ~eds[q];
e019ce  for (int i = 0; i < (int)eds.size()) if (cands[i]) {
c3d609    R[i] = 1;
a58ebf    cliques(eds, f, P & eds[i], X & eds[i], R);
791b2c    R[i] = P[i] = 0; X[i] = 1;
a9847c  }
d3d1a9}
```

Maximum Clique

Description: [kactl] Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph. Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

```
450d01
54ea03 typedef vector<bitset<200>> vb;
913d3d struct Maxclique {
2b09f0  double limit=0.025, pk=0;
93b51d  struct Vertex { int i, d=0; };
b929e8  typedef vector<Vertex> vv;
8ec016  vb e;
071744  vv V;
f35cfb  vector<vector<int>> C;
6887eb  vector<int> qmax, q, S, old;
dd6e7e  void init(vv& r) {
4b55bc    for (auto& v : r) v.d = 0;
60d689    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i
        ];
a3405d    sort(all(r), [](auto a, auto b) { return a.d > b.d;
        });
157f66    int mxD = r[0].d;
af5863    for (int i = 0; i < (int)r.size(); i++) r[i].d = min
        (i, mxD) + 1;
97ef25  }
ccbe9b  void expand(vv& R, int lev = 1) {
a66b83    S[lev] += S[lev - 1] - old[lev];
bd8155    old[lev] = S[lev - 1];
e8164f    while (R.size()) {
0120a4      if (q.size() + R.back().d <= qmax.size()) return;
bb22dc      q.push_back(R.back().i);
9d0473      vv T;
36b304      for(auto v:R) if (e[R.back().i][v.i]) T.push_back
        ({v.i});
364cc4      if (T.size()) {
13683b        if (S[lev]++ / ++pk < limit) init(T);
63bcf5        int j = 0, mxk = 1, mnk = max(qmax.size() - q.
        size() + 1, 1);
4acdd2        C[1].clear(), C[2].clear();
2f0793        for (auto v : T) {
a5dd38          int k = 1;
ae0ec7          auto f = [&](int i) { return e[v.i][i]; };
961987          while (any_of(all(C[k]), f)) k++;
0fa6d4          if (k > mxk) mxk = k, C[mxk + 1].clear();
42b97d          if (k < mnk) T[j++].i = v.i;
2f802c          C[k].push_back(v.i);
218470        }
9515ef        if (j > 0) T[j - 1].d = 0;
7715af        for (int k = mnk; k <= mxk; k++) for (int i : C[
        k])
b49ea4          T[j].i = i, T[j++].d = k;
3fd5d0        expand(T, lev + 1);
3c9524      } else if (q.size() > qmax.size()) qmax = q;
4cec15      q.pop_back(), R.pop_back();
b3e706    }
e9be7a  }
7e8a4e  vector<int> maxClique() { init(V), expand(V); return
         qmax; }
7e1788  Maxclique(vb conn) : e(conn), C(e.size()+1), S(C.size
        ()), old(S) {
728a95    for (int i = 0; i < (int)e.size(); i++) V.push_back
        ({i});
cca214  }
450d01};
```

Min Cost Max Flow

Description: [kactl] Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only. Status: Tested on kattis:mincostmaxflow, stress-tested against another implementation

Complexity: $O(FE\log(V))$ where F is max flow. $O(VE)$ for setpi.

```
df859b
d41d8c // #include <bits/extc++.h> /// include-line, keep-
        include
d41d8c
9f43ac const ll INF = numeric_limits<ll>::max() / 4;
9f43ac
49eea0 struct MCMF {
1681cd  struct edge {
d4edf5    int from, to, rev;
00467c    ll cap, cost, flow;
2b1b2e  };
3ecc0d  int N;
1d58ff  vector<vector<edge>> ed;
9c51a0  vector<int> seen;
66096d  vector<ll> dist, pi;
ffc1c4  vector<edge*> par;
ffc1c4
bf4b99  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N),
         par(N) {}
bf4b99
f3fa50  void addEdge(int from, int to, ll cap, ll cost) {
f4f64b    if (from == to) return;
71e990    ed[from].push_back(edge{ from,to,ed[to].size(),cap,
         cost,0 });
e34760    ed[to].push_back(edge{ to,from,ed[from].size()-1,0,-
         cost,0 });
affb5d  }
affb5d
62902a  void path(int s) {
da4e0c    fill(all(seen), 0);
ec82c7    fill(all(dist), INF);
bf2f86    dist[s] = 0; ll di;
bf2f86
cbc205    __gnu_pbds::priority_queue<pair<ll, int>> q;
9dfccd    vector<decltype(q)::point_iterator> its(N);
608ecc    q.push({ 0, s });
608ecc
385ba0    while (!q.empty()) {
586f36      s = q.top().second; q.pop();
cd41e0      seen[s] = 1; di = dist[s] + pi[s];
990236      for (edge& e : ed[s]) if (!seen[e.to]) {
1f5d62        ll val = di - pi[e.to] + e.cost;
ec1e5b        if (e.cap - e.flow > 0 && val < dist[e.to]) {
634f61          dist[e.to] = val;
651516          par[e.to] = &e;
495a10          if (its[e.to] == q.end())
5e4657            its[e.to] = q.push({ -dist[e.to], e.to });
```

```
c257fc            else
941e5f              q.modify(its[e.to], { -dist[e.to], e.to });
9e2d27          }
72722c        }
26e34c      }
6b2528      for (int i = 0; i < N; i++) pi[i] = min(pi[i] + dist
          [i], INF);
919505    }
919505
8c7573    pair<ll, ll> maxflow(int s, int t) {
687d12      ll totflow = 0, totcost = 0;
068f6b      while (path(s), seen[t]) {
47f6b8        ll fl = INF;
925313        for (edge* x = par[t]; x; x = par[x->from])
3ba9d1          fl = min(fl, x->cap - x->flow);
3ba9d1
ff13d6        totflow += fl;
8ebc00        for (edge* x = par[t]; x; x = par[x->from]) {
5c4cb0          x->flow += fl;
c3a97a          ed[x->to][x->rev].flow -= fl;
c23229        }
1ff3a7      }
c128d1      for (int i = 0; i < N; i++) for(edge& e : ed[i])
          totcost += e.cost * e.flow;
4260b7      return {totflow, totcost/2};
b565e3    }
b565e3
b565e3    // If some costs can be negative, call this before
          maxflow:
b58b45    void setpi(int s) { // (otherwise, leave this out)
be8bf1      fill(all(pi), INF); pi[s] = 0;
335398      int it = N, ch = 1; ll v;
7907da      while (ch-- && it--)
76aa50        for (int i = 0; i < N; i++) if (pi[i] != INF)
de4ea5          for (edge& e : ed[i]) if (e.cap)
a3038c            if ((v = pi[i] + e.cost) < pi[e.to])
f1444d              pi[e.to] = v, ch = 1;
2b882c      assert(it >= 0); // negative cost cycle
40527f    }
df859b};
```

Push Relabel

Description: [kactl] Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.
Complexity: $O(V^2\sqrt{E})$

fa2f25

```
49faef  struct PushRelabel {
a8847d    struct Edge {
815784      int dest, back;
4b2438      ll f, c;
d82272    };
e68988    vector<vector<Edge>> g;
74f8e7    vector<ll> ec;
cf3254    vector<Edge*> cur;
6ffc7b    vector<vector<int>> hs; vector<int> H;
0776ec    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n
          ) {}
0776ec
8cc70c    void addEdge(int s, int t, ll cap, ll rcap=0) {
dd6ab2      if (s == t) return;
f24a4e      g[s].push_back({t, g[t].size(), 0, cap});
cfee23      g[t].push_back({s, g[s].size()-1, 0, rcap});
3c5845    }
3c5845
6108aa    void addFlow(Edge& e, ll f) {
4a496c      Edge &back = g[e.dest][e.back];
9962c0      if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest
          );
10b818      e.f += f; e.c -= f; ec[e.dest] += f;
938f2c      back.f -= f; back.c += f; ec[back.dest] -= f;
7bedff    }
49eca0    ll calc(int s, int t) {
12bff3      int v = g.size(); H[s] = v; ec[t] = 1;
2fe358      vector<int> co(2*v); co[0] = v-1;
2e7f4b      for (int i = 0; i < v; i++) cur[i] = g[i].data();
5b7892      for (Edge& e : g[s]) addFlow(e, e.c);
5b7892
fc0451      for (int hi = 0;;) {
492d91        while (hs[hi].empty()) if (!hi--) return -ec[s];
d0ec4f        int u = hs[hi].back(); hs[hi].pop_back();
3f702f        while (ec[u] > 0)  // discharge u
a59281          if (cur[u] == g[u].data() + g[u].size()) {
d0256a            H[u] = 1e9;
f416d3            for (Edge& e : g[u]) if (e.c && H[u] > H[e.
          dest]+1)
2c841f              H[u] = H[e.dest]+1, cur[u] = &e;
efc38c            if (++co[H[u]], !--co[hi] && hi < v)
71af09              for (int i = 0; i < v; i++) if (hi < H[i] &&
           H[i] < v)
22809a                --co[H[i]], H[i] = v + 1;
ea6458            hi = H[u];
de403e          } else if (cur[u]->c && H[u] == H[cur[u]->dest
          ]+1)
8808f3            addFlow(*cur[u], min(ec[u], cur[u]->c));
8385b6          else ++cur[u];
0b1a98      }
27d38f    }
6fe658    bool leftOfMinCut(int a) { return H[a] >= g.size(); }
fa2f25};
```

Topological Sort

Description: [kactl] Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Complexity: $O(|V| + |E|)$

9eae37

```
92d6c9  vector<int> topoSort(const vector<vector<int>>& gr) {
75cd8a    vector<int> indeg(gr.size()), q;
31e012    for (auto& li : gr) for (int x : li) indeg[x]++;
088354    for (int i = 0; i < (int)gr.size(); i++) if (indeg[i]
          == 0) q.push_back(i);
6a033a    for (int j = 0; j < (int)q.size(); j++) for (int x :
           gr[q[j]])
1f2c0b      if (--indeg[x] == 0) q.push_back(x);
cd7706    return q;
9eae37  }
```

Combinatorial

Permutation to Int

Description: [kactl] Given a permutation, returns the number of lexicographically strictly smaller permutations.
Complexity: $\mathcal{O}(n)$, but returns a value that is $\mathcal{O}(n!)$

7016ba

```
9ab6e7  int permToInt(vector<int> v) {
a6407c    int use = 0, i = 0, r = 0;
5878fd    for(int x : v) {
ba160a      r = r * ++i + __builtin_popcount(use & -(1<<x));
27b952      use |= 1 << x;
5d9fcb    }
4a7d46    return r;
7016ba  }
```

Multinomial

Description: [kactl] Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$.

7f8833

```
8c3101  ll multinomial(vector<int> v) {
93a8d1    ll c = 1, m = v.size() ? v[0] : 1;
5019e7    for (int i = 1; i < (int)v.size(); i++)
fad3cf      for (int j = 0; j < v[i]; j++)
3daa43        c = c * ++m / (j+1);
99415d    return c;
7f8833  }
```

Number_theory

Chinese Remainder Theorem

Description: [kactl] Chinese Remainder Theorem. `crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Complexity: $\log(n)$

04d93a

```
d41d8c  // #include "euclid.h"
d41d8c
24a218  ll crt(ll a, ll m, ll b, ll n) {
6cb862    if (n > m) swap(a, b), swap(m, n);
8f59af    ll x, y, g = euclid(m, n, x, y);
7424cf    assert((a - b) % g == 0); // else no solution
eaeb2a    x = (b - a) % n * x % n / g * m + a;
000521    return x < 0 ? x + m*n/g : x;
04d93a  }
```

Continued Fractions

Description: [kactl] Given N and a real number $x \ge 0$, finds the closest rational approximation p/q with $p, q \le N$. It will obey $|p/q - x| \le 1/qN$. For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.
Complexity: $O(\log N)$

bc259b

```
b6384a  pair<ll, ll> approximate(ld x, ll N) {
3b433c    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; ld y
            = x;
68a164    for (;;) {
e33cf1      ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q :
           inf),
774b33          a = (ll)floor(y), b = min(a, lim),
c80ba9          NP = b*P + LP, NQ = b*Q + LQ;
cdc284      if (a > b) {
cdc284        // If b > a/2, we have a semi-convergent that
          gives us a
cdc284        // better approximation; if b = a/2, we *may* have
           one.
cdc284        // Return {P, Q} here for a more canonical
          approximation.
fa4e9d        return (abs(x - (ld)NP / (ld)NQ) < abs(x - (ld)P /
           (ld)Q)) ?
484e1c            make_pair(NP, NQ) : make_pair(P, Q);
8d0b91      }
ebbbcb      if (abs(y = 1/(y - (ld)a)) > 3*N) {
f20204        return {NP, NQ};
385cde      }
8f9b12      LP = P; P = NP;
ae398c      LQ = Q; Q = NQ;
ae2560    }
bc259b  }
```

Euclid Extended

Description: [kactl] Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of $a \pmod b$.

33ba8f

```
c2276e ll euclid(ll a, ll b, ll &x, ll &y) {
fda33f   if (!b) return x = 1, y = 0, a;
d3dcdb   ll d = euclid(b, a % b, y, x);
05ab91   return y -= a/b * x, d;
33ba8f }
```

Factor

Description: [kactl] Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -¿ {11, 19, 11}).

Complexity: $O(n^{1/4})$, less for numbers with small factors.

cece17

```
d41d8c // #include "ModMulLL.h"
d41d8c // #include "MillerRabin.h"
d41d8c
7eb30f ull pollard(ull n) {
56776d   ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
12dccb   auto f = [&](ull x) { return modmul(x, x, n) + i; };
6b05a1   while (t++ % 40 || __gcd(prd, n) == 1) {
47de4d     if (x == y) x = ++i, y = f(x);
a82195     if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd =
           q;
5dc2c5     x = f(x), y = f(f(y));
0b4d32   }
8bfe46   return __gcd(prd, n);
cd2ac3 }
c3787b vector<ull> factor(ull n) {
6303f2   if (n == 1) return {};
74d420   if (isPrime(n)) return {n};
09e534   ull x = pollard(n);
0d9093   auto l = factor(x), r = factor(n / x);
66e11d   l.insert(l.end(), r.begin(), r.end());
91921d   return l;
cece17 }
```

Miller Rabin

Description: [kactl] Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.

Complexity: 7 times the complexity of $a^b \mod c$.

573e3b

```
f4cf5b typedef unsigned long long ull;
92e1d3 ull modmul(ull a, ull b, ull M) {
00ac89   ll ret = a * b - M * ull(1.L / M * a * b);
21b1bc   return ret + M * (ret < 0) - M * (ret >= (ll)M);
a9c350 }
438153 ull modpow(ull b, ull e, ull mod) {
c04010   ull ans = 1;
aea873   for (; e; b = modmul(b, b, mod), e /= 2)
f5aa70     if (e & 1) ans = modmul(ans, b, mod);
6d3d5f   return ans;
bbbd8f }
c27895 bool isPrime(ull n) {
6816d1   if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
80c10d   ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
           1795265022},
dce1c7       s = __builtin_ctzll(n-1), d = n >> s;
c7af76   for (ull a : A) {   // ^ count trailing zeroes
a3e8c4     ull p = modpow(a%n, d, n), i = s;
5892ce     while (p != 1 && p != n - 1 && a % n && i--)
cabf50       p = modmul(p, p, n);
4b8881     if (p != n-1 && i != s) return 0;
197b25   }
0e9a77   return 1;
573e3b }
```

Mod Inverse

Description: [kactl] Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.

24f722

```
d41d8c // const ll mod = 1000000007, LIM = 200000; ///include-
          line
66d058 ll* inv = new ll[LIM] - 1; inv[1] = 1;
24f722 for (int i = 2; i < LIM; i++) inv[i] = mod - (mod / i) *
         inv[mod % i] % mod;
```

Mod Logarithm

Description: [kactl] Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a. Time: $O(\sqrt{m})$

394cb2

```
0c88ae ll modLog(ll a, ll b, ll m) {
b2527f   ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
bb2a05   unordered_map<ll, ll> A;
390a19   while (j <= n && (e = f = e * a % m) != b % m)
2605ad     A[e * b % m] = j++;
2d9fb0   if (e == b % m) return j;
36aeb4   if (__gcd(m, e) == __gcd(m, b))
022b1e     for (int i = 2; i < n + 2; i++) if (A.count(e = e *
           f % m))
b756dc       return n * i - A[e];
f9fdb3   return -1;
394cb2 }
```

Mod Square Root

Description: [kactl] Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution). Comlexity: $O(\log^2 p)$ worst case, $O(\log p)$ for most p

1336e8

```
150a47 ll modpow(ll b, ll e, ll mod) {
cc2a06   ll ans = 1;
4873c0   for (; e; b = b * b % mod, e /= 2)
dc653a     if (e & 1) ans = ans * b % mod;
16c649   return ans;
ade764 }
ade764
c7807b ll sqrt(ll a, ll p) {
ff5189   a %= p; if (a < 0) a += p;
46f839   if (a == 0) return 0;
d3f67a   assert(modpow(a, (p-1)/2, p) == 1); // else no
          solution
d9b5ee   if (p % 4 == 3) return modpow(a, (p+1)/4, p);
d9b5ee   // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8
          == 5
841741   ll s = p - 1, n = 2;
ab100e   int r = 0, m;
7efe33   while (s % 2 == 0)
40e0d2     ++r, s /= 2;
40e0d2   /// find a non-square mod p
f13233   while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
c5bb84   ll x = modpow(a, (s + 1) / 2, p);
e5065e   ll b = modpow(a, s, p), g = modpow(n, s, p);
47f61c   for (;; r = m) {
581152     ll t = b;
8c31f5     for (m = 0; m < r && t != 1; ++m)
e334b0       t = t * t % p;
17a2e7     if (m == 0) return x;
d43153     ll gs = modpow(g, 1LL << (r - m - 1), p);
dd2f15     g = gs * gs % p;
d455e6     x = x * gs % p;
843562     b = b * g % p;
c3367c   }
1336e8 }
```

Mod Sums of Progressions

Description: [kactl] Sums of mod'ed arithmetic progressions. `modsum(to, c, k, m)` $= \sum_{i=0}^{\text{to}-1} (ki + c)\%m$. `divsum` is similar but for floored division.

Complexity: $\log(m)$, with a large constant.

5c5bc5

```
f4cf5b typedef unsigned long long ull;
6bd037 ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
6bd037 /// ^ written in a weird way to deal with overflows
          correctly
6bd037
c2a3c4 ull divsum(ull to, ull c, ull k, ull m) {
df3a05   ull res = k / m * sumsq(to) + c / m * to;
45fcd1   k %= m; c %= m;
d4b74d   if (!k) return res;
da4668   ull to2 = (to * k + c) / m;
c692ff   return res + (to - 1) * to2 - divsum(to2, m-1 - c, m,
           k);
4a574e }
4a574e
8eb039 ll modsum(ull to, ll c, ll k, ll m) {
290fd2   c = ((c % m) + m) % m;
148f40   k = ((k % m) + m) % m;
f535c2   return to * c + k * sumsq(to) - m * divsum(to, c, k, m
           );
5c5bc5 }
```

Phi Function

Description: [kactl] *Euler's* ϕ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with n.

d892a2

```
fd7760 const int LIM = 5000000;
b4bbf9 int phi[LIM];
b4bbf9
e30f2f void calculatePhi() {
4860ef   for (int i = 0; i < LIM; i++) phi[i] = i&1 ? i : i/2;
bfb9a1   for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
b4629f     for (int j = i; j < LIM; j += i) phi[j] -= phi[j] /
           i;
d892a2 }
```

Numerical

Berlekamp Massey

Description: [kactl] Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.

Usage: `BerlekampMassey({ 0, 1, 1, 3, 5, 11 })` yields { 1, 2 }.

Complexity: $\mathcal{O}(N^2)$

96548b

```
d41d8c // #include "../number-theory/ModPow.h"
d41d8c
c102ae vector<ll> berlekampMassey(vector<ll> s) {
4a819a   int n = sz(s), L = 0, m = 0;
102d94   vector<ll> C(n), B(n), T;
b21e6e   C[0] = B[0] = 1;
b21e6e
b7979b   ll b = 1;
241c0c   rep(i,0,n) { ++m;
e8466a     ll d = s[i] % mod;
```

```
7e74b0    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
f1ebd1    if (!d) continue;
b3b877    T = C; ll coef = d * modpow(b, mod-2) % mod;
b5778a    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
a5ab84    if (2 * L > i) continue;
2475e2    L = i + 1 - L; B = T; b = d; m = 0;
3dc38b  }
3dc38b
deac77  C.resize(L + 1); C.erase(C.begin());
5fed96  for (ll& x : C) x = (mod - x) % mod;
3f3762  return C;
96548b}
```

Determinant of matrix

Description: [kactl] Calculates the determinant of a matrix. Note: destroys the matrix.

Complexity: $\mathcal{O}(N^3)$

bd5cec

```
e36c74double det(vector<vector<double>>& a) {
590c12  int n = sz(a); double res = 1;
d90a91  rep(i,0,n) {
4bd724    int b = i;
309239    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b =
      j;
c6c8fd    if (i != b) swap(a[i], a[b]), res *= -1;
658965    res *= a[i][i];
390833    if (res == 0) return 0;
15fcb2    rep(j,i+1,n) {
356eb5      double v = a[j][i] / a[i][i];
979baa      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
ebf330    }
aa3042  }
7feeff  return res;
bd5cec}
```

Fast Fourier Transform

Description: [kactl] `fft(a)` computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. 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). Otherwise, use NTT/FFTMod.

Complexity: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

3dd197

```
bccabctypedef complex<double> C;
b05ddbtypedef vector<double> vd;
760a36void fft(vector<C>& a) {
547c8a  int n = sz(a), L = 31 - __builtin_clz(n);
1ec777  static vector<complex<long double>> R(2, 1);
1e9f4b  static vector<C> rt(2, 1);  // (^ 10% faster if double
      )
beb684  for (static int k = 2; k < n; k *= 2) {
af116f    R.resize(n); rt.resize(n);
69a3c0    auto x = polar(1.0L, acos(-1.0L) / k);
148d3c    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i
      /2];
42ea68  }
d8b6b6  vi rev(n);
394b0e  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
8afdf7  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
14a253  for (int k = 1; k < n; k *= 2)
9f2153    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
9f2153      // C z = rt[j+k] * a[i+j+k]; // (25% faster if
      hand-rolled)  /// include-line
71bb8d      auto x = (double *)&rt[j+k], y = (double *)&a[i+j+
      k];       /// exclude-line
f0fec3      C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);
                /// exclude-line
ab793c      a[i + j + k] = a[i + j] - z;
939962      a[i + j] += z;
a3c605    }
de1acd}
bf0709vd conv(const vd& a, const vd& b) {
368356  if (a.empty() || b.empty()) return {};
cc42f4  vd res(sz(a) + sz(b) - 1);
819e9e  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
95ab64  vector<C> in(n), out(n);
1f7947  copy(all(a), begin(in));
6e8e10  rep(i,0,sz(b)) in[i].imag(b[i]);
dc6bfc  fft(in);
0ff507  for (C& x : in) x *= x;
a1edd0  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
d6e709  fft(out);
399c53  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
0ac860  return res;
3dd197}
```

Fast Fourier Transform under modulo

Description: [kactl] Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.

Complexity: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (twice as slow as NTT or FFT)

b82773

```
d41d8c// #include "FastFourierTransform.h"
d41d8c
192b04typedef vector<ll> vl;
1dbf8btemplate<int M> vl convMod(const vl &a, const vl &b) {
ffecc4  if (a.empty() || b.empty()) return {};
9094f2  vl res(sz(a) + sz(b) - 1);
2c46a2  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(
      M));
21d40b  vector<C> L(n), R(n), outs(n), outl(n);
ff2f33  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] %
      cut);
f13a07  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] %
      cut);
f8a1f3  fft(L), fft(R);
747bd0  rep(i,0,n) {
153b79    int j = -i & (n - 1);
a18b88    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
1a97e3    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1
      i;
455f55  }
67d701  fft(outl), fft(outs);
086d2a  rep(i,0,sz(res)) {
8bdaab    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])
      +.5);
9ac06e    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
0af53f    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
26b37c  }
94c360  return res;
b82773}
```

Fast Subset Transform

Description: [kactl] Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of a must be a power of two.

Complexity: $\mathcal{O}(N \log N)$

25c175

```
ac2a38void FST(vi& a, bool inv) {
99f61d  for (int n = sz(a), step = 1; step < n; step *= 2) {
fb24ab    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+step
      ) {
9824d9      int &u = a[j], &v = a[j + step]; tie(u, v) =
ae85b6        inv ? pii(v - u, u) : pii(v, u + v); // AND
ae85b6        // inv ? pii(v, u - v) : pii(u + v, u); // OR
      /// include-line
ae85b6        // pii(u + v, u - v);                    // XOR
      /// include-line
535601    }
462b78  }
462b78  // if (inv) for (int& x : a) x /= sz(a); // XOR only
      /// include-line
a727eb}
cef5d7vi conv(vi a, vi b) {
73474b  FST(a, 0); FST(b, 0);
df4270  rep(i,0,sz(a)) a[i] *= b[i];
a35d7f  FST(a, 1); return a;
25c175}
```

Golden section search

Description: [kactl] Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is eps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.

Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`

Complexity: $\mathcal{O}(\log((b-a)/\epsilon))$

31d45b

```
d41d8c/// It is important for r to be precise, otherwise we
      don't necessarily maintain the inequality a < x1 < x2
      < b.
eb1b64double gss(double a, double b, double (*f)(double)) {
6c8388  double r = (sqrt(5)-1)/2, eps = 1e-7;
2a17ea  double x1 = b - r*(b-a), x2 = a + r*(b-a);
f89d5b  double f1 = f(x1), f2 = f(x2);
40bd12  while (b-a > eps)
0713d5    if (f1 < f2) { //change to > to find maximum
012afe      b = x2; x2 = x1; f2 = f1;
4ed154      x1 = b - r*(b-a); f1 = f(x1);
c73cf7    } else {
62bf16      a = x1; x1 = x2; f1 = f2;
0fa28d      x2 = a + r*(b-a); f2 = f(x2);
821619    }
39c67b  return a;
31d45b}
```

Hill climbing

Description: [kactl] Poor man's optimization for unimodal functions.

8eeeaf

```
9eb631typedef array<double, 2> P;
9eb631
710806template<class F> pair<double, P> hillClimb(P start, F f
      ) {
18b365  pair<double, P> cur(f(start), start);
68a8ed  for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
1a21bb    rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
d5ba69      P p = cur.second;
aaa103      p[0] += dx*jmp;
bd427b      p[1] += dy*jmp;
64a5cc      cur = min(cur, make_pair(f(p), p));
93215a    }
523260  }
34f652  return cur;
8eeeaf}
```

Integer determinant

Description: [kactl] Calculatres determinant using modular arithmetics.
Complexity: $\mathcal{O}(N^3)$

```
3313dc
0311cc const ll mod = 12345;
ea0b38 ll det(vector<vector<ll>>& a) {
aeac6f   int n = sz(a); ll ans = 1;
c9d9cd   rep(i,0,n) {
cab51f     rep(j,i+1,n) {
4f621e       while (a[j][i] != 0) { // gcd step
155e04         ll t = a[i][i] / a[j][i];
f94a75         if (t) rep(k,i,n)
618162           a[i][k] = (a[i][k] - a[j][k] * t) % mod;
4d6748         swap(a[i], a[j]);
cbbac3         ans *= -1;
3e9488       }
7effce     }
7173b1     ans = ans * a[i][i] % mod;
c4c228     if (!ans) return 0;
666fb0   }
cd2f86   return (ans + mod) % mod;
3313dc }
```

Integrate

Description: [kactl] Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

```
4756fc
044d82 template<class F>
751e63 double quad(double a, double b, F f, const int n = 1000)
       {
840c14   double h = (b - a) / 2 / n, v = f(a) + f(b);
b84885   rep(i,1,n*2)
e9333e     v += f(a + i*h) * (i&1 ? 4 : 2);
df3a8f   return v * h / 3;
4756fc }
```

Adaptive integrate

Description: [kactl] Fast integration using an adaptive Simpson's rule.
Usage: `double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});`

```
92dd79
0705cd typedef double d;
459b90 #define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) /
       6
459b90
f429e0 template <class F>
e701f0 d rec(F& f, d a, d b, d eps, d S) {
eda167   d c = (a + b) / 2;
bdc489   d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
b97adb   if (abs(T - S) <= 15 * eps || b - a < 1e-10)
3f5868     return T + (T - S) / 15;
4d1eec   return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps /
         2, S2);
a81d9a }
e8c244 template<class F>
248534 d quad(d a, d b, F f, d eps = 1e-8) {
868afd   return rec(f, a, b, eps, S(a, b));
92dd79 }
```

Linear recurrence

Description: [kactl] Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \dots \geq n-1]$ and $tr[0 \dots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({ 0, 1 }, { 1, 1 }, k) // k'th Fibonacci number`
Complexity: $\mathcal{O}(n^2 \log k)$

```
03b92e
166499 const ll mod = 5; /** exclude-line */
166499
cfe688 typedef vector<ll> Poly;
28d968 ll linearRec(Poly S, Poly tr, ll k) {
9a5aa3   int n = sz(tr);
9a5aa3
d76ed5   auto combine = [&](Poly a, Poly b) {
b28dcf     Poly res(n * 2 + 1);
2c9a7f     rep(i,0,n+1) rep(j,0,n+1)
6a6759       res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
511d7b     for (int i = 2 * n; i > n; --i) rep(j,0,n)
a92240       res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j])
           % mod;
1fa6e9     res.resize(n + 1);
56b081     return res;
88cd0e   };
88cd0e
5db532   Poly pol(n + 1), e(pol);
b92c68   pol[0] = e[1] = 1;
b92c68
ac9c4b   for (++k; k; k /= 2) {
cf96d4     if (k % 2) pol = combine(pol, e);
e31603     e = combine(e, e);
08992c   }
08992c
dfd443   ll res = 0;
d5c608   rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
7e7da0   return res;
03b92e }
```

Matrix Inversion under modulo

Description: [kactl] Invert matrix A modulo a prime. Returns rank; result is stored in A unless singular (rank ¡ n). For prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Complexity: $\mathcal{O}(n^3)$

```
0b7b13
d41d8c // #include "../number-theory/ModPow.h"
d41d8c
7025f3 int matInv(vector<vector<ll>>& A) {
8d1bdf   int n = sz(A); vi col(n);
ff2cbf   vector<vector<ll>> tmp(n, vector<ll>(n));
ebd124   rep(i,0,n) tmp[i][i] = 1, col[i] = i;
ebd124
4c70b5   rep(i,0,n) {
196537     int r = i, c = i;
163e60     rep(j,i,n) rep(k,i,n) if (A[j][k]) {
843bfc       r = j; c = k; goto found;
670a88     }
43b703     return i;
79369e found:
6f7f47     A[i].swap(A[r]); tmp[i].swap(tmp[r]);
013e1f     rep(j,0,n)
994d92       swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c])
           ;
f483b9     swap(col[i], col[c]);
a33b6a     ll v = modpow(A[i][i], mod - 2);
221dbc     rep(j,i+1,n) {
4dc1d6       ll f = A[j][i] * v % mod;
820a75       A[j][i] = 0;
191b80       rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod
           ;
2034cf       rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) %
           mod;
3af408     }
402ef6     rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
6e1d6e     rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
7099c7     A[i][i] = 1;
b5fe9f   }
b5fe9f
9c015a   for (int i = n-1; i > 0; --i) rep(j,0,i) {
8a334f     ll v = A[j][i];
fb9283     rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) %
           mod;
597dbe   }
597dbe
765b04   rep(i,0,n) rep(j,0,n)
2446cb     A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] <
           0)*mod;
1914c7   return n;
0b7b13 }
```

Matrix inversion

Description: [kactl] Invert matrix A. Returns rank; result is stored in A unless singular (rank ¡ n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1})$ (mod p^k) where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Complexity: $\mathcal{O}(n^3)$

```
ebfff6
4b565b int matInv(vector<vector<double>>& A) {
e91afd   int n = sz(A); vi col(n);
2e69f1   vector<vector<double>> tmp(n, vector<double>(n));
9a9a66   rep(i,0,n) tmp[i][i] = 1, col[i] = i;
9a9a66
8ece41   rep(i,0,n) {
a71041     int r = i, c = i;
3ff7a0     rep(j,i,n) rep(k,i,n)
c8b6a2       if (fabs(A[j][k]) > fabs(A[r][c]))
6b4e10         r = j, c = k;
baa3bb     if (fabs(A[r][c]) < 1e-12) return i;
7482dd     A[i].swap(A[r]); tmp[i].swap(tmp[r]);
c4816d     rep(j,0,n)
6e2f7f       swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c])
           ;
6ce940     swap(col[i], col[c]);
59c017     double v = A[i][i];
e17078     rep(j,i+1,n) {
1c2a5d       double f = A[j][i] / v;
3cc4a2       A[j][i] = 0;
9da1ac       rep(k,i+1,n) A[j][k] -= f*A[i][k];
293c3d       rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
4b5802     }
f7a458     rep(j,i+1,n) A[i][j] /= v;
678f7a     rep(j,0,n) tmp[i][j] /= v;
bbea47     A[i][i] = 1;
cd352a   }
cd352a
cd352a   /// forget A at this point, just eliminate tmp
         backward
28ee96   for (int i = n-1; i > 0; --i) rep(j,0,i) {
973479     double v = A[j][i];
b3722c     rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
fd4d51   }
fd4d51
09764f   rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
898124   return n;
ebfff6 }
```

Number Theoretic Transform

Description: [kactl] `ntt(a)` computes $\hat{f}(k) = \sum_x a[x] g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, `reverse(start+1, end)`, NTT back. Inputs must be in $[0, mod)$.

Complexity: $\mathcal{O}(N \log N)$

ced03d

```
d41d8c// #include "../number-theory/ModPow.h"
d41d8c
b5e822const ll mod = (119 << 23) + 1, root = 62; // =
       998244353
b5e822// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479
       << 21
b5e822// and 483 << 21 (same root). The last two are > 10^9.
7458catypedef vector<ll> vl;
0ca385void ntt(vl &a) {
c96375  int n = sz(a), L = 31 - __builtin_clz(n);
7bd0b3  static vl rt(2, 1);
668758  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
4c5a31    rt.resize(n);
1759b1    ll z[] = {1, modpow(root, mod >> s)};
2921d8    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
5faa22  }
3ee1db  vi rev(n);
78dccf  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
158770  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
225017  for (int k = 1; k < n; k *= 2)
61bd17    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
64cbc8      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i +
       j];
cba978      a[i + j + k] = ai - z + (z > ai ? mod : 0);
4b5040      ai += (ai + z >= mod ? z - mod : z);
35d5bf    }
29a029}
bbaf00vl conv(const vl &a, const vl &b) {
4001b0  if (a.empty() || b.empty()) return {};
ac0aeb  int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
cb0e4e      n = 1 << B;
10d0fe  int inv = modpow(n, mod - 2);
5e3527  vl L(a), R(b), out(n);
8e31ec  L.resize(n), R.resize(n);
6415db  ntt(L), ntt(R);
c16165  rep(i,0,n)
1c4346    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv %
       mod;
4af30c  ntt(out);
70c6bc  return {out.begin(), out.begin() + s};
ced03d}
```

Number Theoretic Transform

Description: [kactl] Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.

Complexity: $\mathcal{O}(n^2)$

08bf48

```
ae03aetypedef vector<double> vd;
28ccecvd interpolate(vd x, vd y, int n) {
a3ca7f  vd res(n), temp(n);
01cf0e  rep(k,0,n-1) rep(i,k+1,n)
1590be    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
ca948d  double last = 0; temp[0] = 1;
58fd2d  rep(k,0,n) rep(i,0,n) {
9c95bc    res[i] += y[k] * temp[i];
e58978    swap(last, temp[i]);
eb1dd0    temp[i] -= last * x[k];
8c43d1  }
d408ff  return res;
08bf48}
```

Polynomial roots

Description: [kactl] Finds the real roots to a polynomial.

Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`

Complexity: $\mathcal{O}(n^2 \log(1/\epsilon))$

b00bfe

```
d41d8c// #include "Polynomial.h"
d41d8c
64af29vector<double> polyRoots(Poly p, double xmin, double
       xmax) {
a63eaa  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
343f7f  vector<double> ret;
2acf4e  Poly der = p;
8409d9  der.diff();
105e2f  auto dr = polyRoots(der, xmin, xmax);
31d1fe  dr.push_back(xmin-1);
324645  dr.push_back(xmax+1);
5604f0  sort(all(dr));
50119c  rep(i,0,sz(dr)-1) {
d045cc    double l = dr[i], h = dr[i+1];
2748c8    bool sign = p(l) > 0;
ea5d57    if (sign ^ (p(h) > 0)) {
cc4926      rep(it,0,60) { // while (h - l > 1e-8)
40bd6f        double m = (l + h) / 2, f = p(m);
145fe6        if ((f <= 0) ^ sign) l = m;
8da3ef        else h = m;
4f1379      }
f5991f      ret.push_back((l + h) / 2);
1c9b1d    }
d5f24e  }
a514b7  return ret;
b00bfe}
```

Polynomial structure

Description: [kactl]

c9b7b0

```
213314struct Poly {
640a33  vector<double> a;
aea975  double operator()(double x) const {
b40030    double val = 0;
1b799c    for (int i = sz(a); i--;) (val *= x) += a[i];
3743d7    return val;
f7a37b  }
187735  void diff() {
462d92    rep(i,1,sz(a)) a[i-1] = i*a[i];
1e1024    a.pop_back();
d447a3  }
cd4862  void divroot(double x0) {
3236c3    double b = a.back(), c; a.back() = 0;
06b4f8    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+
       b, b=c;
071796    a.pop_back();
43bc43  }
c9b7b0};
```

Simplex

Description: [kactl] Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.

Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}}; vd b = {1,1,-4}, c = {-1,-1}, x; T val = LPSolver(A, b, c).solve(x);`

Complexity: $\mathcal{O}(NM\#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

aa8530

```
943c93typedef double T; // long double, Rational, double + mod
       <P>...
4a7fa3typedef vector<T> vd;
19471ctypedef vector<vd> vvd;
19471c
6296c1const T eps = 1e-8, inf = 1/.0;
20f308#define MP make_pair
80a946#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s
       ])) s=j
80a946
004b50struct LPSolver {
34f6a6  int m, n;
a8b98c  vi N, B;
a50829  vvd D;
a50829
e8814c  LPSolver(const vvd& A, const vd& b, const vd& c) :
09ecbe    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
a00ca8      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
eab15d      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] =
       b[i];}
03bb56      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
4c20cd      N[n] = -1; D[m+1][n] = 1;
dcadf8    }
dcadf8
d2dadf  void pivot(int r, int s) {
72cb06    T *a = D[r].data(), inv = 1 / a[s];
93b9bd    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
a86c76      T *b = D[i].data(), inv2 = b[s] * inv;
c1f31d      rep(j,0,n+2) b[j] -= a[j] * inv2;
ee22d8      b[s] = a[s] * inv2;
df792b    }
d3cb55    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
9e2376    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
6bf9c5    D[r][s] = inv;
b3404b    swap(B[r], N[s]);
193de8  }
193de8
ede257  bool simplex(int phase) {
f695c2    int x = m + phase - 1;
0aa9db    for (;;) {
8b65cd      int s = -1;
96f50e      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
e72781      if (D[x][s] >= -eps) return true;
fcd18c      int r = -1;
a7d0e5      rep(i,0,m) {
f65882        if (D[i][s] <= eps) continue;
01fd61        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
8af3f7                       < MP(D[r][n+1] / D[r][s], B[r])) r
       = i;
170720      }
23b7a6      if (r == -1) return false;
100fe3      pivot(r, s);
d81c2f    }
62b7d3  }
62b7d3
48ae53  T solve(vd &x) {
b0718e    int r = 0;
cc8cd8    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
dc34d7    if (D[r][n+1] < -eps) {
fbfb80      pivot(r, n);
09ceea      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf
       ;
6b2bed      rep(i,0,m) if (B[i] == -1) {
9aa881        int s = 0;
db9144        rep(j,1,n+1) ltj(D[i]);
d11ba5        pivot(i, s);
213eb8      }
36d5c1    }
```

```
e286bf    bool ok = simplex(1); x = vd(n);
002972    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
8dddea    return ok ? D[m][n+1] : inf;
bc3870  }
aa8530};
```

Solve linear

Description: [kactl] Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Complexity: $\mathcal{O}(n^2m)$

44c9ab

```
ae03aetypedef vector<double> vd;
1784eaconst double eps = 1e-12;
1784ea
dbdd92int solveLinear(vector<vd>& A, vd& b, vd& x) {
2cfbc7  int n = sz(A), m = sz(x), rank = 0, br, bc;
61ac86  if (n) assert(sz(A[0]) == m);
274909  vi col(m); iota(all(col), 0);
274909
27c9a7  rep(i,0,n) {
cdb1df    double v, bv = 0;
9bbd0f    rep(r,i,n) rep(c,i,m)
889ccc      if ((v = fabs(A[r][c])) > bv)
4cafdf        br = r, bc = c, bv = v;
236408    if (bv <= eps) {
008896      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
b9eea0      break;
e8dea5    }
e256ad    swap(A[i], A[br]);
f84bc6    swap(b[i], b[br]);
b1eb75    swap(col[i], col[bc]);
0bea42    rep(j,0,n) swap(A[j][i], A[j][bc]);
bc2598    bv = 1/A[i][i];
292cf7    rep(j,i+1,n) {
416953      double fac = A[j][i] * bv;
f8d04b      b[j] -= fac * b[i];
fe2cdd      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
34df26    }
cc5189    rank++;
66cd8f  }
66cd8f
5f0090  x.assign(m, 0);
21a20d  for (int i = rank; i--;) {
5fa421    b[i] /= A[i][i];
9d7b80    x[col[i]] = b[i];
a0bd4f    rep(j,0,i) b[j] -= A[j][i] * b[i];
55ec26  }
ec3430  return rank; // (multiple solutions if rank < m)
44c9ab}
```

Solve linear 2

Description: [kactl] To get all uniquely determined values of x back from SolveLinear, make the below changes.

08e495

```
d41d8c// #include "SolveLinear.h"
d41d8c
f9498crep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
f9498c// ... then at the end:
3b9d4dx.assign(m, undefined);
45bf44rep(i,0,rank) {
22b426  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
46800e  x[col[i]] = b[i] / A[i][i];
08e495fail:; }
```

Solve linear (binary)

Description: [kactl] Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Complexity: $\mathcal{O}(n^2m)$

fa2d7a

```
9831fetypedef bitset<1000> bs;
9831fe
1dc5afint solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
cdaa0f  int n = sz(A), rank = 0, br;
d90d1b  assert(m <= sz(x));
b3f2a0  vi col(m); iota(all(col), 0);
ede46d  rep(i,0,n) {
1de653    for (br=i; br<n; ++br) if (A[br].any()) break;
af7a74    if (br == n) {
f718ae      rep(j,i,n) if(b[j]) return -1;
4a27f9      break;
84b30e    }
bb0b8a    int bc = (int)A[br]._Find_next(i-1);
95e130    swap(A[i], A[br]);
94782d    swap(b[i], b[br]);
df32d9    swap(col[i], col[bc]);
31f207    rep(j,0,n) if (A[j][i] != A[j][bc]) {
8c102f      A[j].flip(i); A[j].flip(bc);
bf5e08    }
e5befe    rep(j,i+1,n) if (A[j][i]) {
dcae48      b[j] ^= b[i];
7a6a34      A[j] ^= A[i];
0837c3    }
c27cd3    rank++;
4de1ff  }
4de1ff
d4948b  x = bs();
3e622f  for (int i = rank; i--;) {
6b7244    if (!b[i]) continue;
c2244c    x[col[i]] = 1;
17ba9a    rep(j,0,i) b[j] ^= A[j][i];
fe12f5  }
df4d62  return rank; // (multiple solutions if rank < m)
fa2d7a}
```

Tridiagonal

Description: [kactl] $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1,-1,-1,\ldots,-1,1\}, \{0,c_1,c_2,\ldots,c_n\}, \{b_1,b_2,\ldots,b_n,0\}, \{a_0,d_1,d_2,\ldots,d_n,a_{n+1}\}).$$

Fails if the solution is not unique. If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Complexity: $\mathcal{O}(N)$

8f9fa8

```
943c93typedef double T;
b20c01vector<T> tridiagonal(vector<T> diag, const vector<T>&
       super,
f819b9    const vector<T>& sub, vector<T> b) {
52eb69  int n = sz(b); vi tr(n);
399c67  rep(i,0,n-1) {
a25828    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i
       ] == 0
5648ab      b[i+1] -= b[i] * diag[i+1] / super[i];
0189fd      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
5606e8      diag[i+1] = sub[i]; tr[++i] = 1;
ad967f    } else {
e9d89b      diag[i+1] -= super[i]*sub[i]/diag[i];
13335c      b[i+1] -= b[i]*sub[i]/diag[i];
25f2e7    }
7da0d1  }
db774b  for (int i = n; i--;) {
ff86e5    if (tr[i]) {
1481b0      swap(b[i], b[i-1]);
c73d58      diag[i-1] = diag[i];
6bd4e6      b[i] /= super[i-1];
9a7f8a    } else {
2fb613      b[i] /= diag[i];
a82648      if (i) b[i-1] -= b[i]*super[i-1];
94ec57    }
4f78c5  }
b1f2c9  return b;
8f9fa8}
```

Strings

KMP

Description: [kactl] pi[x] computes the length of the longest prefix of s that ends at x, other than s[0...x] itself. Ex. `abacaba -> 0010123`.
Complexity: $\mathcal{O}(N)$

f2828c

```
a5630evector <int> pi(const string& s) {
21bd98  vector <int> p(s.size());
57bd54  for (int i = 1; i < (int) s.size(); i++) {
db0f96    int g = p[i-1];
80a190    while (g && s[i] != s[g]) g = p[g-1];
e7b6fa    p[i] = g + (s[i] == s[g]);
4d0cc7  }
e07336  return p;
807129}
807129
b345c0vector <int> match(const string& s, const string& pat) {
db18ca  vector <int> p = pi(pat + '\0' + s), res;
81432c  for (int i = (int) p.size() - (int) s.size(); i < (int
       ) p.size(); i++)
f9107d    if (p[i] == (int) pat.size()) res.push_back(i - 2 *
       (int) pat.size());
dfc5f5  return res;
f2828c}
```

Manacher

Description: [kactl] For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest add (half rounded down).
Complexity: $\mathcal{O}(N)$

61383b

```
aa956carray<vector <int>, 2> manacher(const string& s) {
7d3176  int n = s.size();
92fdcc  array<vector <int>,2> p = {vector <int> (n+1), vector
       <int> (n)};
9a7ffd  for (int z = 0; z < 2; z++) for (int i=0,l=0,r=0; i <
       n; i++) {
6371de    int t = r-i+!z;
102697    if (i<r) p[z][i] = min(t, p[z][l+t]);
a6ed96    int L = i-p[z][i], R = i+p[z][i]-!z;
```

```
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

Minimum rotation

Description: [kactl] Finds the lexicographically smallest roration of a string.
Complexity: $\mathcal{O}(N)$

```
int minRotation(string s) {
  int a=0, N=s.size(); 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; }
  }
  return a;
}
```

Rolling Hash

Description: RH prepare string s, and hash gives the hash of the substring [l, r] inclusive. ib is pow(b, -1, MD), MD should be prime
Complexity: $\mathcal{O}(n)$ preprocessing, $\mathcal{O}(1)$ hash.

```
struct RH {
  int MD, n, b, ib; // b is base, ib inverse base mod MD
  vector<int> p, ip, hs;
  RH(string s, int _b = 69, int _ib = 579710149, int _MD
      = 1e9 + 7) : MD(_MD), n((int)s.size()), b(_b), ib(_ib
    ), p(n), ip(n), hs(n) { // _b = 63, _ib = 698412843,
    MD = 1e9 + 207
    p[0] = ip[0] = 1;
    hs[0] = s[0];
    for(int i = 1; i < n; ++i){
      p[i] = (ll) p[i - 1] * b % MD;
      ip[i] = (ll) ip[i - 1] * ib % MD;
      hs[i] = ((ll) s[i] * p[i] + hs[i - 1]) % MD; // s[
    i] can be changed to some hash function
    }
  }
  int hash(int l, int r){
    return (ll) (hs[r] - (l ? hs[l - 1] : 0) + MD) * ip[
    l] % MD;
  }
};
```

Suffix automaton

Description: Standard suffix automaton. Does what you'd expect.
Usage: See example main function below. This was thrown in last minute from a working cses solution.
Complexity: $\mathcal{O}(\log n)$ per update/query

```
struct SA {
  struct State {
    int length;
    int link;
    int next[26];
    int cnt;
    bool is_clone;
    int first_pos;
    State(int _length, int _link) :
    length(_length),
    link(_link),
    cnt(0),
    is_clone(false),
    first_pos(-1)
    {
      memset(next, -1, sizeof(next));
    }
  };
  std::vector <State> states;
  int size;
  int last;
  bool did_init_count;
  int str_len;
  bool did_init_css;
  SA() :
  states(1, State(0, -1)),
  size(1),
  last(0),
  did_init_count(false),
  str_len(0),
  did_init_css(false)
  { }
  void push(char c) {
    str_len++;
    did_init_count = false;
    did_init_css = false;
    int cur = size;
    states.resize(++size, State(states[last].length + 1,
    -1));
    states[cur].first_pos = states[cur].length - 1;
    int p = last;
    while (p != -1 && states[p].next[c - 'a'] == -1) {
      states[p].next[c - 'a'] = cur;
      p = states[p].link;
    }
    if (p == -1) {
      states[cur].link = 0;
    } else {
      int q = states[p].next[c - 'a'];
      if (states[p].length + 1 == states[q].length) {
        states[cur].link = q;
      } else {
        int clone = size;
        states.resize(++size, State(states[p].length +
    1, states[q].link));
        states[clone].is_clone = true;
        memcpy(states[clone].next, states[q].next,
    sizeof(State::next));
        states[clone].first_pos = states[q].first_pos;
        while (p != -1 && states[p].next[c - 'a'] == q)
    {
          states[p].next[c - 'a'] = clone;
          p = states[p].link;
        }
        states[q].link = states[cur].link = clone;
      }
    }
    last = cur;
  }
  bool exists(const std::string& pattern) {
    int node = 0;
    int index = 0;
    while (index < (int) pattern.length() && states[node
    ].next[pattern[index] - 'a'] != -1) {
      node = states[node].next[pattern[index] - 'a'];
      index++;
    }
    return index == (int) pattern.size();
  }
  int count(const std::string& pattern) {
    if (!did_init_count) {
      did_init_count = true;
      for (int i = 1; i < size; i++) {
        states[i].cnt = !states[i].is_clone;
      }
    }
    std::vector <std::vector <int>> of_length(str_len
    + 1);
    for (int i = 0; i < size; i++) {
      of_length[states[i].length].push_back(i);
    }
    for (int l = str_len; l >= 0; l--) {
      for (int node : of_length[l]) {
        if (states[node].link != -1) {
          states[states[node].link].cnt += states[node
    ].cnt;
        }
      }
    }
    int node = 0;
    int index = 0;
    while (index < (int) pattern.length() && states[node
    ].next[pattern[index] - 'a'] != -1) {
      node = states[node].next[pattern[index] - 'a'];
      index++;
    }
    return index == (int) pattern.size() ? states[node].
    cnt : 0;
  }
  int first_occ(const std::string& pattern) {
    int node = 0;
    int index = 0;
    while (index < (int) pattern.length() && states[node
    ].next[pattern[index] - 'a'] != -1) {
      node = states[node].next[pattern[index] - 'a'];
      index++;
    }
    return index == (int) pattern.size() ? states[node].
    first_pos - (int) pattern.size() + 1 : -1;
  }
  size_t count_substrings() {
    static std::vector <size_t> dp;
    if (!did_init_css) {
      did_init_css = true;
      dp = std::vector <size_t> (size, 0);
      auto dfs = [&] (auto&& self, int node) -> size_t {
        if (node == -1) {
          return 0;
        }
        if (dp[node]) {
          return dp[node];
        }
        dp[node] = 1;
        for (int i = 0; i < 26; i++) {
          dp[node] += self(self, states[node].next[i]);
        }
        return dp[node];
      };
      dfs(dfs, 0);
    }
    return dp[0] - 1;
  }
};

// usage example: Repeating Substring submission on cses
    .fi
int main() {
  std::ios::sync_with_stdio(0); std::cin.tie(0);
  std::string s; std::cin >> s;
  int n; std::cin >> n;
  SA sa;
  for (char c : s) {
    sa.push(c);
  }
  sa.count("");
  int len = -1;
```

```
bb09b1  int ind = -1;
af0b43  for (int i = 1; i < sa.size; i++) {
f4d141    if (sa.states[i].cnt > 1) {
eb5645      if (len < sa.states[i].length) {
961e2f        len = sa.states[i].length;
becb1e        ind = sa.states[i].first_pos - len + 1;
5af6dc      }
3b9795    }
0f2256  }
f0ebc0  if (len == -1) {
de5034    std::cout << "-1\n";
c8c5ae    return 0;
a99b6e  }
f38c31  for (int i = 0; i < len; i++) {
0d86ab    std::cout << s[i + ind];
42f1ff  }
228fb9  std::cout << "\n";
3d234e}
```

Z-function

Description: [kactl] z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. Ex. abacaba -> 0010301.
Complexity: $\mathcal{O}(N)$

```
d0fcad
b86749vector <int> Z(const string& S) {
63e1e3  vector <int> z(S.size());
749eac  int l = -1, r = -1;
ec3aad  for (int i = 1; i < (int) S.size(); i++) {
391986    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
26d12f    while (i + z[i] < (int) S.size() && S[i + z[i]] == S
          [z[i]])
036fc5      z[i]++;
5dfcb4    if (i + z[i] > r)
765e28      l = i, r = i + z[i];
2e06c4  }
d9efc2  return z;
d0fcad}
```

Geometry

3D convex hull

Description: Yoinked from kactl. Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.
Complexity: $\mathcal{O}(n^2)$.

```
5b45fc
d41d8c// #include "Point_3D.h"
d41d8c
b8e08btypedef Point3D<double> P3;
b8e08b
6aa2edstruct PR {
cc2473  void ins(int x) { (a == -1 ? a : b) = x; }
e28e42  void rem(int x) { (a == x ? a : b) = -1; }
531490  int cnt() { return (a != -1) + (b != -1); }
5f78b5  int a, b;
9a9457};
9a9457
538b68struct F { P3 q; int a, b, c; };
538b68
7d6924vector<F> hull3d(const vector<P3>& A) {
1d7f45  assert(sz(A) >= 4);
39c3b5  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1,
        -1}));
39ded9#define E(x,y) E[f.x][f.y]
6eec88  vector<F> FS;
9469d2  auto mf = [&](int i, int j, int k, int l) {
47e4ee    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
60a935    if (q.dot(A[l]) > q.dot(A[i]))
d6434b      q = q * -1;
ed7472    F f{q, i, j, k};
dd2b5a    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
d2c39f    FS.push_back(f);
f13ccf  };
411dfe  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
489c42    mf(i, j, k, 6 - i - j - k);
489c42
42c30d  rep(i,4,sz(A)) {
b33224    rep(j,0,sz(FS)) {
77d954      F f = FS[j];
c1b7a2      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
d54d8c        E(a,b).rem(f.c);
6ed4b4        E(a,c).rem(f.b);
5384c9        E(b,c).rem(f.a);
2eb5b4        swap(FS[j--], FS.back());
3244b8        FS.pop_back();
40e2cb      }
66122d    }
47a0d8    int nw = sz(FS);
930bd5    rep(j,0,nw) {
5d88f4      F f = FS[j];
460e4f#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i
        , f.c);
cccf10      C(a, b, c); C(a, c, b); C(b, c, a);
9bd3f7    }
c8c803  }
29960f  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
3622d0    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
7f1cdc  return FS;
5b45fc};
```

Angle

Description: Yoinked from kactl. A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.
Usage: `vector <Angle> v = w[0], w[0].t360() ...; // sorted int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; } // sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i`

```
0f0602
755634struct Angle {
022c62  int x, y;
76ee53  int t;
d184d3  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
6c948b  Angle operator-(Angle b) const { return {x-b.x, y-b.y,
          t}; }
020235  int half() const {
b0dc15    assert(x || y);
9d5c24    return y < 0 || (y == 0 && x < 0);
39c79d  }
12afc7  Angle t90() const { return {-y, x, t + (half() && x >=
          0)}; }
05c9a0  Angle t180() const { return {-x, -y, t + half()}; }
3dd266  Angle t360() const { return {x, y, t + 1}; }
e258c0};
c1efa9bool operator<(Angle a, Angle b) {
c1efa9  // add a.dist2() and b.dist2() to also compare
        distances
a1f0ad  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
7d3b54          make_tuple(b.t, b.half(), a.x * (ll)b.y);
e78926}
e78926
e78926// Given two points, this calculates the smallest angle
        between
e78926// them, i.e., the angle that covers the defined line
        segment.
ccb19apair<Angle, Angle> segmentAngles(Angle a, Angle b) {
48d2ad  if (b < a) swap(a, b);
c0377f  return (b < a.t180() ?
4b88b6            make_pair(a, b) : make_pair(b, a.t360()));
eccd19}
c11d8eAngle operator+(Angle a, Angle b) { // point a + vector
        b
c7f4a3  Angle r(a.x + b.x, a.y + b.y, a.t);
7cc5c9  if (a.t180() < r) r.t--;
e12799  return r.t180() < a ? r.t360() : r;
3fb429}
89aa95Angle angleDiff(Angle a, Angle b) { // angle b - angle a
99d8df  int tu = b.t - a.t; a.t = b.t;
33f708  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b
        < a)};
0f0602}
```

Circle circle intersection

Description: Yoinked from kactl. Computes the pair of points at which two circles intersect. Returns false in case of no intersection.
Complexity: $\mathcal{O}(1)$.

```
84d6d3
d41d8c// #include "Point.h"
d41d8c
6269ectypedef Point<double> P;
888549bool circleInter(P a,P b,double r1,double r2,pair<P, P>*
        out) {
7e53c0  if (a == b) { assert(r1 != r2); return false; }
2e6973  P vec = b - a;
deb755  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
7b252e          p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p
        *p*d2;
6ad02a  if (sum*sum < d2 || dif*dif > d2) return false;
70d886  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2)
          / d2);
3dd318  *out = {mid + per, mid - per};
212ced  return true;
84d6d3}
```

Circle line intersection

Description: Yoinked from kactl. Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be `Point <double>`.

```
e0cfba
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
0406advector<P> circleLine(P c, double r, P a, P b) {
cddb51  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
e51742  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
64a27f  if (h2 < 0) return {};
3d9ab3  if (h2 == 0) return {p};
1be847  P h = ab.unit() * sqrt(h2);
3b1a3f  return {p - h, p + h};
e0cfba}
```

Circle polygon intersection

Description: Yoinked from kactl. Returns the area of the intersection of a circle with a ccw polygon.
Complexity: $\mathcal{O}(n)$.

```
a1ee63
d41d8c// #include "Point.h"
d41d8c
6269ectypedef Point<double> P;
cf6463#define arg(p, q) atan2(p.cross(q), p.dot(q))
cf0d22double circlePoly(P c, double r, vector<P> ps) {
419913  auto tri = [&](P p, P q) {
a6cf13    auto r2 = r * r / 2;
c0445a    P d = q - p;
```

```
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.
  dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(
  det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

Circle tangents

Description: Yoinked from kactl. Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

b0153d

```
// #include "Point.h"

template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2,
    double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr
  ;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

Circumcircle

Description: Yoinked from kactl. The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

1caa3a

```
// #include "Point.h"

typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)
  /2;
}
```

Closest pair of points

Description: Yoinked from kactl. Finds the closest pair of points.
Complexity: $\mathcal{O}(n \log n)$.

ac41a6

```
// #include "Point.h"

typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p
    + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

Convex hull

Description: Yoinked from kactl. Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Complexity: $\mathcal{O}(n \log n)$.

310954

```
// #include "Point.h"

typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0)
    t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
   h[1])};
}
```

Delaunay triangulation

Description: Yoinked from kactl. Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.
Complexity: $\mathcal{O}(n^2)$.

c0e7bc

```
// #include "Point.h"
// #include "3d_hull.h"

template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
  if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2])
  < 0);
    trifun(0,1+d,2-d); }
  vector<P3> p3;
  for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
  if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3
  [t.a]).
      cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
    trifun(t.a, t.c, t.b);
}
```

Dynamic Convex Hull

Description: Supports building a convex hull one point at a time. Viewing the convex hull along the way.

431bba

```
struct point {
    ll x, y;
    point(ll x=0, ll y=0): x(x), y(y) {}
    point operator-(const point &p) const { return point
  (x-p.x, y-p.y); }
    point operator*(const ll k) const { return point(k*x
  , k*y); }
    ll cross(const point &p) const { return x*p.y - p.x*
  y; }
    bool operator<(const point &p) const { return x < p.
  x || x == p.x && y < p.y; }
};

bool above(set<point> &hull, point p, ll scale = 1) {
    auto it = hull.lower_bound(point((p.x+scale-1)/scale
  , 0));
    if (it == hull.end()) return true;
    if (p.y <= it->y*scale) return false;
    if (it == hull.begin()) return true;
    auto jt = it--;
    return (p-*it*scale).cross(*jt-*it) < 0;
}

void add(set<point> &hull, point p) {
    if (!above(hull, p)) return;
    auto pit = hull.insert(p).first;
    while (pit != hull.begin()) {
        auto it = prev(pit);
        if (it->y <= p.y || (it != hull.begin() && (*it
  -*prev(it)).cross(*pit-*it) >= 0))
            hull.erase(it);
        else
            break;
    }
    auto it = next(pit);
    while (it != hull.end()) {
        if (next(it) != hull.end() && (*it-p).cross(*
  next(it)-*it) >= 0)
            hull.erase(it++);
        else
            break;
    }
```

Hull diameter

Description: Yoinked from kactl. Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Complexity: $\mathcal{O}(n)$.

c571b8

```
// #include "Point.h"

typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j
  ]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i])
   >= 0)
        break;
    }
  return res.second;
}
```

Inside polygon

Description: Yoinked from kactl. Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`
Complexity: $\mathcal{O}(n)$.

2bf504

```
d41d8c// #include "Point.h"
d41d8c// #include "On_segment.h"
d41d8c// #include "Segment_distance.h"
d41d8c
7dc51etemplate<class P>
8cfa07bool inPolygon(vector<P> &p, P a, bool strict = true) {
68a46b  int cnt = 0, n = sz(p);
49a14b  rep(i,0,n) {
1c161f    P q = p[(i + 1) % n];
ca77bc    if (onSegment(p[i], q, a)) return !strict;
ca77bc    //or: if (segDist(p[i], q, a) <= eps) return !strict
          ;
8d185a    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q)
          > 0;
ae1a12  }
3f2423  return cnt;
2bf504}
```

KD-tree

Description: Yoinked from kactl. 2D, can be extended to 3D. See comments for details.

bac5b0

```
d41d8c// #include "Point.h"
d41d8c
9a6170typedef long long T;
d3d771typedef Point<T> P;
3b6fe3const T INF = numeric_limits<T>::max();
3b6fe3
632da2bool on_x(const P& a, const P& b) { return a.x < b.x; }
624f75bool on_y(const P& a, const P& b) { return a.y < b.y; }
624f75
319cdastruct Node {
7cd9b0  P pt; // if this is a leaf, the single point in it
1149c5  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
3f2a96  Node *first = 0, *second = 0;
3f2a96
edbce8  T distance(const P& p) { // min squared distance to a
        point
71ed74    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
6963e4    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
4a1b67    return (P(x,y) - p).dist2();
1460d4  }
1460d4
3f46ab  Node(vector<P>&& vp) : pt(vp[0]) {
ae3536    for (P p : vp) {
516c49      x0 = min(x0, p.x); x1 = max(x1, p.x);
28bf16      y0 = min(y0, p.y); y1 = max(y1, p.y);
2e9c2c    }
a1b63f    if (vp.size() > 1) {
a1b63f      // split on x if width >= height (not ideal...)
172b91      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
172b91      // divide by taking half the array for each child
        (not
172b91      // best performance with many duplicates in the
        middle)
21b567      int half = sz(vp)/2;
2f742c      first = new Node({vp.begin(), vp.begin() + half});
a66d3b      second = new Node({vp.begin() + half, vp.end()});
470fcd    }
0265cf  }
6fda19};
6fda19
ce4e50struct KDTree {
eee062  Node* root;
677e4a  KDTree(const vector<P>& vp) : root(new Node({all(vp)})
        ) {}
677e4a
7daf7f  pair<T, P> search(Node *node, const P& p) {
23e6bd    if (!node->first) {
23e6bd      // uncomment if we should not find the point
        itself:
23e6bd      // if (p == node->pt) return {INF, P()};
df1914      return make_pair((p - node->pt).dist2(), node->pt)
          ;
19dc67    }
19dc67
f3c18d    Node *f = node->first, *s = node->second;
c51266    T bfirst = f->distance(p), bsec = s->distance(p);
5cf03e    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
5cf03e
5cf03e    // search closest side first, other side if needed
fa9faa    auto best = search(f, p);
b7e192    if (bsec < best.first)
18c5d3      best = min(best, search(s, p));
891524    return best;
3771f7  }
3771f7
3771f7  // find nearest point to a point, and its squared
        distance
3771f7  // (requires an arbitrary operator< for Point)
5c5074  pair<T, P> nearest(const P& p) {
961132    return search(root, p);
60e74e  }
bac5b0};
```

Line hull intersection

Description: Yoinked from kactl. Line-convex polygon intersection. The polygon must be ccw and have no collinear points. `lineHull(line, poly)` returns a pair describing the intersection of a line with the polygon:

- $(-1,-1)$ if no collision,
- $(i,-1)$ if touching the corner i,
- (i,i) if along side $(i,i+1)$,
- (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$.

In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.
Complexity: $\mathcal{O}(\log n)$.

7cf45b

```
d41d8c// #include "Point.h"
d41d8c
53058e#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(
       j)%n]))
d4b890#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n)
       < 0
8387c5template <class P> int extrVertex(vector<P>& poly, P dir
       ) {
6c658c  int n = sz(poly), lo = 0, hi = n;
b9df6a  if (extr(0)) return 0;
b3e410  while (lo + 1 < hi) {
407848    int m = (lo + hi) / 2;
1b27ac    if (extr(m)) return m;
604289    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
c739cd    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo
       ) = m;
efd609  }
743d4a  return lo;
ba41ca}
ba41ca
911b88#define cmpL(i) sgn(a.cross(poly[i], b))
26a22btemplate <class P>
d01376array<int, 2> lineHull(P a, P b, vector<P>& poly) {
d0d8a9  int endA = extrVertex(poly, (a - b).perp());
bc546b  int endB = extrVertex(poly, (b - a).perp());
ff77a0  if (cmpL(endA) < 0 || cmpL(endB) > 0)
07bb09    return {-1, -1};
a8a9c2  array<int, 2> res;
aa612e  rep(i,0,2) {
090d37    int lo = endB, hi = endA, n = sz(poly);
0ef38e    while ((lo + 1) % n != hi) {
71097d      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
d0c0d9      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
72e441    }
c0e123    res[i] = (lo + !cmpL(hi)) % n;
541f6a    swap(endA, endB);
d56a85  }
d847be  if (res[0] == res[1]) return {res[0], -1};
e14e7a  if (!cmpL(res[0]) && !cmpL(res[1]))
5b4ca0    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly))
          {
ab4398      case 0: return {res[0], res[0]};
e5b066      case 2: return {res[1], res[1]};
54f3d0    }
cba78e  return res;
7cf45b}
```

Line line intersection

Description: Yoinked from kactl. If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point¡ll¿ and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.
Usage: `auto res = lineInter(s1,e1,s2,e2); if (res.first == 1) cout << "intersection point at " << res.second << endl;`

a01f81

```
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
ebe700pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
662a43  auto d = (e1 - s1).cross(e2 - s2);
a6ba96  if (d == 0) // if parallel
47e53e    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
dfc20b  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
c4c8fb  return {1, (s1 * p + e1 * q) / d};
a01f81}
```

Line projection and reflection

Description: Yoinked from kactl. Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

b5562d

```
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
31a653P lineProj(P a, P b, P p, bool refl=false) {
3c6965  P v = b - a;
3d9bc7  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
b5562d}
```

Linear transformation

Description: Yoinked from kactl. Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

03a306

```
d41d8c// #include "Point.h"
```

```
d41d8c
6269ec typedef Point<double> P;
a0133a P linearTransformation(const P& p0, const P& p1,
f9bd62     const P& q0, const P& q1, const P& r) {
16967b   P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq)
         );
d52dff   return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.
          dist2();
03a306 }
```

Manhatten MST

Description: Yoinked from kactl. Given N points, returns up to $4N$ edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights $w(p,q) = |p.x - q.x| + |p.y - q.y|$. Edges are in the form $(distance, src, dst)$. Use a standard MST algorithm on the result to find the final MST.
Complexity: $\mathcal{O}(n \log n)$.

df6f59

```
d41d8c // #include "Point.h"
d41d8c
bbe58c typedef Point<int> P;
10752c vector<array<int, 3>> manhattanMST(vector<P> ps) {
82bb37   vi id(sz(ps));
129d92   iota(all(id), 0);
bded47   vector<array<int, 3>> edges;
4634f8   rep(k,0,4) {
55be09     sort(all(id), [&](int i, int j) {
f00400          return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
0a2d30     map<int, int> sweep;
6ada5f     for (int i : id) {
2327aa       for (auto it = sweep.lower_bound(-ps[i].y);
7348ca                 it != sweep.end(); sweep.erase(it++)) {
931774         int j = it->second;
5297c6         P d = ps[i] - ps[j];
874f9c         if (d.y > d.x) break;
5f471a         edges.push_back({d.y + d.x, i, j});
28e949       }
5f0d0f       sweep[-ps[i].y] = i;
9ea743     }
9c2fdc     for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x
           , p.y);
666542   }
af3f66   return edges;
df6f59 }
```

Minimum enclosing circle

Description: Yoinked from kactl. Computes the minimum circle that encloses a set of points.
Complexity: $\mathcal{O}(n)$.

09dd0a

```
d41d8c // #include "circumcircle.h"
d41d8c
a287af pair<P, double> mec(vector<P> ps) {
31fcb8   shuffle(all(ps), mt19937(time(0)));
76de0f   P o = ps[0];
56a5f0   double r = 0, EPS = 1 + 1e-8;
b5031b   rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
5e7038     o = ps[i], r = 0;
af79ee     rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
57d76d       o = (ps[i] + ps[j]) / 2;
da034d       r = (o - ps[i]).dist();
14cf15       rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
931d7a         o = ccCenter(ps[i], ps[j], ps[k]);
b9c1f4         r = (o - ps[i]).dist();
7cd516       }
03da47     }
bfac59   }
5ebee7   return {o, r};
09dd0a }
```

Is on segment

Description: Yoinked from kactl. Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using `Point <double>`.

c597e8

```
d41d8c // #include "Point.h"
d41d8c
5145ab template<class P> bool onSegment(P s, P e, P p) {
b95df6   return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
c597e8 }
```

2D Point

Description: Yoinked from kactl. Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.).

47ec0a

```
48b588 template <class T> int sgn(T x) { return (x > 0) - (x <
        0); }
fcf845 template<class T>
74299c struct Point {
f773fb   typedef Point P;
fa79fb   T x, y;
551774   explicit Point(T x=0, T y=0) : x(x), y(y) {}
1a0130   bool operator<(P p) const { return tie(x,y) < tie(p.x,
          p.y); }
3a27ca   bool operator==(P p) const { return tie(x,y)==tie(p.x,
          p.y); }
1dc17e   P operator+(P p) const { return P(x+p.x, y+p.y); }
189cbc   P operator-(P p) const { return P(x-p.x, y-p.y); }
268af3   P operator*(T d) const { return P(x*d, y*d); }
8cb755   P operator/(T d) const { return P(x/d, y/d); }
716d84   T dot(P p) const { return x*p.x + y*p.y; }
7ecfd2   T cross(P p) const { return x*p.y - y*p.x; }
520e7b   T cross(P a, P b) const { return (a-*this).cross(b-*
          this); }
e7b843   T dist2() const { return x*x + y*y; }
039a77   double dist() const { return sqrt((double)dist2()); }
039a77   // angle to x-axis in interval [-pi, pi]
cc70a2   double angle() const { return atan2(y, x); }
b02e9c   P unit() const { return *this/dist(); } // makes dist
          ()=1
e05505   P perp() const { return P(-y, x); } // rotates +90
          degrees
c0e5d2   P normal() const { return perp().unit(); }
c0e5d2   // returns point rotated 'a' radians ccw around the
          origin
91d8d5   P rotate(double a) const {
e458d5     return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
70601a   friend ostream& operator<<(ostream& os, P p) {
0e491f     return os << "(" << p.x << "," << p.y << ")"; }
47ec0a };
```

3D Point

Description: Yoinked from kactl. Class to handle points in 3D space. T can be e.g. double or long long. (Avoid int.).

8058ae

```
f10732 template<class T> struct Point3D {
144fa4   typedef Point3D P;
cac5b9   typedef const P& R;
521bb2   T x, y, z;
c7b7d0   explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(
          z) {}
9e2218   bool operator<(R p) const {
af5a46     return tie(x, y, z) < tie(p.x, p.y, p.z); }
16e4b3   bool operator==(R p) const {
fa5b42     return tie(x, y, z) == tie(p.x, p.y, p.z); }
141e02   P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z)
          ; }
825225   P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z)
          ; }
1ee29d   P operator*(T d) const { return P(x*d, y*d, z*d); }
660667   P operator/(T d) const { return P(x/d, y/d, z/d); }
d7cc17   T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
a9fb7d   P cross(R p) const {
b90dcd     return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x
           );
f914db   }
574fd0   T dist2() const { return x*x + y*y + z*z; }
f12431   double dist() const { return sqrt((double)dist2()); }
f12431   //Azimuthal angle (longitude) to x-axis in interval [-
          pi, pi]
c5f1d1   double phi() const { return atan2(y, x); }
c5f1d1   //Zenith angle (latitude) to the z-axis in interval
          [0, pi]
c1e43f   double theta() const { return atan2(sqrt(x*x+y*y),z);
          }
3396cd   P unit() const { return *this/(T)dist(); } //makes
          dist()=1
3396cd   //returns unit vector normal to *this and p
89ad86   P normal(P p) const { return cross(p).unit(); }
89ad86   //returns point rotated 'angle' radians ccw around
          axis
cfb921   P rotate(double angle, P axis) const {
6e0acf     double s = sin(angle), c = cos(angle); P u = axis.
          unit();
8303ee     return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
6c6b0d   }
8058ae };
```

Is point in convex polygon

Description: Yoinked from kactl. Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Complexity: $\mathcal{O}(\log n)$.

71446b

```
d41d8c // #include "Point.h"
d41d8c // #include "Side_of.h"
d41d8c // #include "On_segment.h"
d41d8c
2c0584 typedef Point<ll> P;
2c0584
912e4a bool inHull(const vector<P>& l, P p, bool strict = true)
        {
3f3f6c   int a = 1, b = sz(l) - 1, r = !strict;
7a3fc8   if (sz(l) < 3) return r && onSegment(l[0], l.back(), p
         );
b8cb94   if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
3c3a3b   if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p
         )<= -r)
bc80dd     return false;
709831   while (abs(a - b) > 1) {
e79ab6     int c = (a + b) / 2;
2a9b80     (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
e4f356   }
0b5229   return sgn(l[a].cross(l[b], p)) < r;
71446b }
```

Polygon area

Description: Yoinked from kactl. Returns *twice* the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

f12300

```
d41d8c // #include "Point.h"
d41d8c
4fce64 template<class T>
df7c3f T polygonArea2(vector<Point<T>>& v) {
```

```
ab8862  T a = v.back().cross(v[0]);
0711d6  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
b195d0  return a;
f12300}
```

Polygon center of mass

Description: Yoinked from kactl. Returns the center of mass for a polygon.
Complexity: $\mathcal{O}(n)$.

```
9706dc
d41d8c// #include "Point.h"
d41d8c
6269ectypedef Point<double> P;
fa2dc3P polygonCenter(const vector<P>& v) {
a6f845  P res(0, 0); double A = 0;
1dc006  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
082251    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
c6e9e9    A += v[j].cross(v[i]);
01751d  }
9d5722  return res / A / 3;
9706dc}
```

Polygon cut

Description: Yoinked from kactl. Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector <P> p = ...; p = polygonCut(p, P(0,0), P(1,0));`

```
f2b7d4
d41d8c// #include "Point.h"
d41d8c// #include "Line_intersection.h"
d41d8c
6269ectypedef Point<double> P;
b4b253vector<P> polygonCut(const vector<P>& poly, P s, P e) {
b83885  vector<P> res;
f6354c  rep(i,0,sz(poly)) {
3664ba    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
41eabb    bool side = s.cross(e, cur) < 0;
f87882    if (side != (s.cross(e, prev) < 0))
f7bea5      res.push_back(lineInter(s, e, cur, prev).second);
f5439d    if (side)
cf4e26      res.push_back(cur);
567ae4  }
75262c  return res;
f2b7d4}
```

Polygon union

Description: Yoinked from kactl. Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)
Complexity: $\mathcal{O}(n^2)$ where n is the total number of points.

```
3931c6
d41d8c// #include "Point.h"
d41d8c// #include "Side_of.h"
d41d8c
6269ectypedef Point<double> P;
940b75double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b
      .y; }
51eb9cdouble polyUnion(vector<vector<P>>& poly) {
9680ea  double ret = 0;
49c6ab  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
1ea114    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])
      ];
e9da64    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
aea249    rep(j,0,sz(poly)) if (i != j) {
03624d      rep(u,0,sz(poly[j])) {
0826f1        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[
      j])];
c62a46        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
ac826b        if (sc != sd) {
a48d6d          double sa = C.cross(D, A), sb = C.cross(D, B);
aeaa76          if (min(sc, sd) < 0)
13f2a7            segs.emplace_back(sa / (sa - sb), sgn(sc -
      sd));
ce5e1a        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-
      C))>0){
a4636e          segs.emplace_back(rat(C - A, B - A), 1);
d44814          segs.emplace_back(rat(D - A, B - A), -1);
67520d        }
c4b419      }
a1900f    }
97ae86    sort(all(segs));
4e8cac    for (auto& s : segs) s.first = min(max(s.first, 0.0)
      , 1.0);
00b8ae    double sum = 0;
40a9a7    int cnt = segs[0].second;
317ef1    rep(j,1,sz(segs)) {
84ade9      if (!cnt) sum += segs[j].first - segs[j - 1].first
      ;
625398        cnt += segs[j].second;
d3398f    }
0e34c6    ret += A.cross(B) * sum;
6f2b4e  }
52ed80  return ret / 2;
3931c6}
```

Polyhedron volume

Description: Yoinked from kactl. Magic formula for the volume of a polyhedron. Faces should point outwards.

```
3058c3
f9cf71template<class V, class L>
8b5f1fdouble signedPolyVolume(const V& p, const L& trilist) {
75c331  double v = 0;
828881  for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p
      [i.c]);
27c3d1  return v / 6;
3058c3}
```

Points line-segments distance

Description: Yoinked from kactl. Returns the shortest distance between point p and the line segment from point s to e.
Usage: `Point <double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

```
5c88f4
d41d8c// #include "Point.h"
d41d8c
6269ectypedef Point<double> P;
789af4double segDist(P& s, P& e, P& p) {
3139df  if (s==e) return (p-s).dist();
2506d7  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s
      )));
b95d89  return ((p-s)*d-(e-s)*t).dist()/d;
5c88f4}
```

Line segment line segment intersection

Description: Yoinked from kactl. If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point¡ll¿ and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.
Usage: `vector <P> inter = segInter(s1,e1,s2,e2); if (sz(inter)==1) cout << "segments intersect at " << inter[0] << endl;`

```
9d57f2
d41d8c// #include "Point.h"
d41d8c// #include "OnSegment.h"
d41d8c
dae11dtemplate<class P> vector<P> segInter(P a, P b, P c, P d)
      {
f4c95c  auto oa = c.cross(d, a), ob = c.cross(d, b),
5041fa       oc = a.cross(b, c), od = a.cross(b, d);
5041fa  // Checks if intersection is single non-endpoint point
      .
dec360  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
ab16eb    return {(a * ob - b * oa) / (ob - oa)};
43185b  set<P> s;
d73b7a  if (onSegment(c, d, a)) s.insert(a);
9f9c48  if (onSegment(c, d, b)) s.insert(b);
64d2c1  if (onSegment(a, b, c)) s.insert(c);
1dcb4f  if (onSegment(a, b, d)) s.insert(d);
c505dc  return {all(s)};
9d57f2}
```

Side of

Description: Yoinked from kactl. Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be `Point <T>` where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: `bool left = sideOf(p1,p2,q)==1;`

```
3af81c
d41d8c// #include "Point.h"
d41d8c
7dc51etemplate<class P>
fad9c9int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
fad9c9
bb2891template<class P>
059ae5int sideOf(const P& s, const P& e, const P& p, double
       eps) {
37dc17  auto a = (e-s).cross(p-s);
ea3543  double l = (e-s).dist()*eps;
765665  return (a > l) - (a < -l);
3af81c}
```

Spherical distance

Description: Yoinked from kactl. Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. $dx \cdot radius$ is then the difference between the two points in the x direction and d*radius is the total distance between the points.

```
611f07
c5faf9double sphericalDistance(double f1, double t1,
86b44b      double f2, double t2, double radius) {
2b5463  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
aa0db3  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
6da400  double dz = cos(t2) - cos(t1);
819384  double d = sqrt(dx*dx + dy*dy + dz*dz);
5b1067  return radius*2*asin(d/2);
611f07}
```

Line distance

Description: Yoinked from kactl. Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan.

P is supposed to be `Point <T>` or `Point3D <T>` where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

f6bf6b

```
// #include "Point.h"

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

Various

Longest increasing subsequence

Description: [kactl] Compute indices for the longest increasing subsequence.
Complexity: $\mathcal{O}(N \log N)$

155c66

```
template<class I> vector <int> lis(const vector<I>& S) {
  if (S.empty()) return {};
  vector <int> prev(S.size());
  typedef pair<I, int> p;
  vector<p> res;
  for (int i = 0; i < (int) S.size(); i++) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    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];
  return ans;
}
```

Sum over subsets

Description: Used to calculate the sum of values for all subsets of a given set or bitmask essentially calculates $S_x = \sum_{i|x=x} v_i$ for all x.
Usage: v should be a frequency array where the values correspond to how many elements have such mask `SOS(v) // returns vector with S_x`
Complexity: $\mathcal{O}(N \log N)$

881d6f

```
vector<int> SOS(vector<int> v) {
  int lg = 31 - __builtin_clz((int)v.size()), mx = 1 << lg;
  v.resize(mx);
  for(int i = 0; i < lg; ++i) for(int j = (1 << i); j < mx; ++j) if(j >> i & 1) v[j] += v[j - (1 << i)];
  return v;
}
```

Simmulated Annealing

Description: [cp-algorithms] A randomized approach to approximate a global optimum of a function (i.e TSP).
Usage: Fill in the state class: state() should be the initial state (initial guess) next() should create a neighbouring state, i.e. (For TSP swap two nodes in the order) E() should be the energy function, the thing that should be maximized. (For TSP the total distance)
Complexity: $\mathcal{O}(E() \cdot log_{1/u}(T))$.

fb4b5c

```
bool P(double E,double E_next,double T,mt19937 rng){
    double prob =  exp(-(E_next-E)/T);
    if(prob > 1) return true;
    else{
        bernoulli_distribution d(prob);
        return d(rng);
    }
}
class state {
    public:
    state() {
        // Generate the initial state
    }
    state next() {
        state s_next;
        // Modify s_next to a random neighboring state
        return s_next;
    }
    double E() {
        // Implement the energy function here
    };
};

pair<double, state> simAnneal() {
    state s = state();
    state best = s;
    double T = 10000; // Initial temperature
    double u = 0.995; // decay rate
    double E = s.E();
    double E_next;
    double E_best = E;
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    while (T > 1) {
        state next = s.next();
        E_next = next.E();
        if (P(E, E_next, T, rng)) {
            s = next;
            if (E_next < E_best) {
                best = s;
                E_best = E_next;
            }
            E = E_next;
        }
        T *= u;
    }
    return {E_best, best};
}
```

Bump allocator

Description: [kactl] When you need to dynamically allocate many objects and don't care about freeing them. `new X` otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

745db2

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
  static size_t i = sizeof buf;
  assert(s < i);
  return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

Bump allocator (STL)

Description: [kactl] See Bump allocator. This one is STL friendly.

bb66d4

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
  typedef T value_type;
  small() {}
  template<class U> small(const U&) {}
  T* allocate(size_t n) {
    buf_ind -= n * sizeof(T);
    buf_ind &= 0 - alignof(T);
    return (T*)(buf + buf_ind);
  }
  void deallocate(T*, size_t) {}
};
```

(very) fast input

Description: [kactl] Fast input. Desperation when facing TLE on big input tasks.

7b3c70

```
inline char gc() { // like getchar()
  static char buf[1 << 16];
  static size_t bc, be;
  if (bc >= be) {
    buf[0] = 0, bc = 0;
    be = fread(buf, 1, sizeof(buf), stdin);
  }
  return buf[bc++]; // returns 0 on EOF
}

int readInt() {
  int a, c;
  while ((a = gc()) < 40);
  if (a == '-') return -readInt();
  while ((c = gc()) >= 48) a = a * 10 + c - 480;
  return a - 48;
}
```

Fast knapsack

Description: [kactl] Given N non-negative integer weights w and a non-negative target t, computes the maximum S ¡= t such that S is the sum of some subset of the weights.
Complexity: $\mathcal{O}(N \max(w_i))$

7c4938

```
int knapsack(vector <int> w, int t) {
  int a = 0, b = 0, x;
  while (b < sz(w) && a + w[b] <= t) a += w[b++];
  if (b == sz(w)) return a;
  int m = *max_element(all(w));
  vi u, v(2*m, -1);
  v[a+m-t] = b;
  for (int i = b; i < (int) w.size(); i++) {
    u = v;
    for (int 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;
}
```

fast mod reduction

Description: [kactl] Compute $a\%b$ about 5 times faster than usual, where b is constant but not known at compile time. Returns a value congruent to $a \pmod b$ in the range $[0, 2b)$.

751a02

```
typedef unsigned long long ull;
struct FastMod {
  ull b, m;
  FastMod(ull b) : b(b), m(-1ULL / b) {}
  ull reduce(ull a) { // a % b + (0 or b)
```

```
c7e7c1     return a - (ull)((__uint128_t(m) * a) >> 64) * b;
03d237   }
751a02};
```

Interval container

Description: [kactl] Add and remove intervals [inclusive, exclusive). The maintained set has non-overlapping intervals at all times.
Complexity: Both operations are $\mathcal{O}(\log N)$ amortized.

```
f47dfb
f7d7f8set<pair <int, int>>::iterator addInterval(set<pair <int
       , int>>& is, int L, int R) {
c5c1db   if (L == R) return is.end();
82cedf   auto it = is.lower_bound({L, R}), before = it;
7c3bb5   while (it != is.end() && it->first <= R) {
81a0b4     R = max(R, it->second);
3a4dd8     before = it = is.erase(it);
a91ed2   }
b0b5fc   if (it != is.begin() && (--it)->second >= L) {
843a06     L = min(L, it->first);
795959     R = max(R, it->second);
5e5470     is.erase(it);
0f5234   }
29e9d4   return is.insert(before, {L,R});
16c3b2}
16c3b2
b05726void removeInterval(set<pair <int, int>>& is, int L, int
        R) {
324d6a   if (L == R) return;
5b2eae   auto it = addInterval(is, L, R);
1cdaff   auto r2 = it->second;
f1f136   if (it->first == L) is.erase(it);
312f69   else (int&)it->second = L;
bb3e12   if (R != r2) is.emplace(R, r2);
f47dfb}
```

Interval cover

Description: [kactl] Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Complexity: $\mathcal{O}(N \log N)$.

```
595f5d
24b8d1template<class T> vector <int> cover(pair<T, T> G,
        vector<pair<T, T>> I) {
df7cec   vector <int> S(I.size()), R;
313cfc   iota(S.begin(), S.end(), 0);
351c2c   sort(S.begin(), S.end(), [&](int a, int b) { return I[
        a] < I[b]; });
85d891   T cur = G.first;
0c3c11   int at = 0;
41fa20   while (cur < G.second) { // (A)
5f2202     pair<T, int> mx = make_pair(cur, -1);
6812fb     while (at < sz(I) && I[S[at]].first <= cur) {
436881       mx = max(mx, make_pair(I[S[at]].second, S[at]));
33b415       at++;
3f8e88     }
9bd97b     if (mx.second == -1) return {};
a6a3fe     cur = mx.first;
fc4c14     R.push_back(mx.second);
a285a0   }
45d172   return R;
595f5d}
```

Knuth DP

Description: [kactl] When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \leq f(a,d)$ and $f(a,c) + f(b,d) \leq f(a,d) + f(b,c)$ for all $a \leq b \leq c \leq d$. Consider also: Line container, monotone queues, ternary search.
Complexity: $\mathcal{O}(N^2)$

```
d41d8c
```

Manual loop unrolling

Description: [kactl] Manual loop unrolling.

```
520e76
5ec590#define F {...; ++i;}
1823b8int i = from;
dbde22while (i&3 && i < to) F // for alignment, if needed
4379e1while (i + 4 <= to) { F F F F }
520e76while (i < to) F
```

Xor basis

Description: Basis of vectors in Z_2^d

```
61b70d
bf37aastruct XB {
6ea8b3   vector<int> basis;
ae23d0   void ins(int mask) {
6f1850     for(auto &y : basis) {
24dad5       if(y < mask) swap(y, mask);
af22b6       mask = min(mask, mask ^ y);
241cda     }
5fc70a     if(mask) basis.push_back(mask); // if mask is 0
        value can already be represented by basis
3208a1   }
61b70d};
```