University of Copenhagen

3 little 3 late

Asbjørn Lind, Elias Rasmussen Lolck, Thor Vejen Eriksen

NWERC 2025

November 26, 2025

Setup

hash.sh

```
                                                                    5246ca
d41d8c# hashes a file, ignoring whitespaces and comments
d41d8c# use for verifying that code is copied correctly
5246ca cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |
       cut -c-6
```

vimrc

```
                                                                    39eb19
f112b5 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
2f1e84 ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space
       :]' \| md5sum \| cut -c-6
6ad224 vnoremap <silent> p "_dP
60b7c4 vnoremap <silent> <A-Down> :m '>+1<CR>gv=gv
39eb19 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
92f63c struct Fen {
04c831   vector <ll> v;
15fd8d   Fen(int s) : v(s, 0) { }
f76ea5   void update(int ind, ll val) {
4238a4     for (; ind < (int) v.size(); ind |= ind + 1) v[ind]
           += val;
222f2c   }
7b09a2   ll query(int ind) { // [0, ind), ind < 0 returns 0
37f317     ll res = 0;
cc7a2a     for (; ind > 0; ind &= ind - 1) res += v[ind - 1];
           // operation can be modified
552720     return res;
1c3977   }
348a7a   int lower_bound(ll sum) { // returns first i with
         query(i + 1) >= sum, n if not found
1f0b41     int ind = 0;
fe1e46     for (int p = 1 << 25; p; p >>= 1) // 1 << 25 can be
           lowered to ceil(log2(v.size()))
a63f8c       if (ind + p <= (int) v.size() && v[ind + p - 1] <
             sum)
a9f291         sum -= v[(ind += p) - 1];
15c383     return ind;
ac78de   }
1743e1 };
```

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
25e012 template <class T, bool VALS_IN_EDGES> class HLD {
e46734   private:
02923f   int N, R, tim = 0;   // n, root node, time
b57ab1   vector<vector<int>> adj;
a4ab58   vector<int> par, siz, depth, rt, pos;   // parent, size
         , depth, root, position arrays
6ee68b   MaxSegmentTree<T> segtree;              // Modify as
         needed
6ee68b
6ee68b   /** Compute the size of each subtree and set parent-
         child relationship
6ee68b    * Subtree of node v corresponds to segment [ pos[v],
         pos[v] + sz[v] ) */
442494   void dfs_sz(int v) {
3d8503     if (par[v] != -1) adj[v].erase(find(adj[v].begin(),
           adj[v].end(), par[v]));
a5ff86     for (int &u : adj[v]) {
b6cbb2       par[u] = v, depth[u] = depth[v] + 1;
a74c8d       dfs_sz(u);
b3252c       siz[v] += siz[u];
b7e81e       if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
d28e3c     }
78447f   }
78447f
78447f   /** Assign positions for nodes
78447f    * Path from v to the last vertex in ascending heavy
         path
         corresponds to [ pos[rt[v]], pos[v] ] */
8dac07   void dfs_hld(int v) {
196d7f     pos[v] = tim++;
a2082b     for (int u : adj[v]) {
c1174d       rt[u] = (u == adj[v][0] ? rt[v] : u);
2c5bc7       dfs_hld(u);
f882cf     }
219eb3   }
219eb3
219eb3   /** process all heavy path and combine their results
         */
dee5a3   template <class B> void process(int u, int v, B op) {
b399a2     for (; rt[u] != rt[v]; v = par[rt[v]]) {
53ea29       if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
d57a65       op(pos[rt[v]], pos[v]);
974c4a     }
e38566     if (depth[u] > depth[v]) swap(u, v);
a6df70     op(pos[u] + VALS_IN_EDGES, pos[v]);
de8934   }
0e823d   public:
bb15ce   HLD(vector<vector<int>> adj_, int _R)
64945f     : N(adj_.size()), R(_R), adj(adj_), par(N, -1),
           siz(N, 1), depth(N), rt(N),
825d07         pos(N), segtree(N)  // modify as needed
a141a0   {
266bf0     rt[R] = R;
3f8601     dfs_sz(R);
7ce78d     dfs_hld(R);
8ae2b7   }
68fdb7   T query_path(int u, int v) {
c00ebf     T res = 0;  // default value, modify depending on
           problem
           process(u, v, [&](int l, int r) {
27174d       res = max(res, segtree.range_max(l, r + 1));  //
           modify depending on problem
9c4a7c     });
9e8ba2     return res;
9807c7   }
5ddf8d   void modify_node(int u, T val) { segtree.set(pos[u],
         val); }
4fc289 };
```

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
4bbcdb struct Line { ll a, b; ll f(ll x) { return a * x + b; }
       };
7988a9 constexpr const Line LINF { 0, 1LL << 60 };
ffb13a struct LCT {
358a49   vector <Line> v; // coord-compression: modify v[x] ->
         v[conert(x)]
48d025   LCT(int size) { v.resize(size, LINF); }
8d520c   void insert(Line line, int l, int r) {
effece     if (l > r) return;
a07972     int mid = (l + r) >> 1;
318c53     if (line.f(mid) < v[mid].f(mid)) swap(line, v[mid]);
ec2a0e     if (line.f(l) < v[mid].f(l)) insert(line, l, mid -
           1);
665fcd     else insert(line, mid + 1, r);
cba366   }
212b60   Line query(int x, int l, int r) {
8c17fb     if (l > r) return LINF;
1f9b50     int mid = (l + r) >> 1;
3bd038     if (x == mid) return v[mid]; // faster on avg. - not
           necessary
ea215f     if (x < mid) return best_of(v[mid], query(x, l, mid
           - 1), x);
e40e21     return best_of(v[mid], query(x, mid + 1, r), x);
70ae78   }
2daa25   Line best_of(Line a, Line b, ll x) { return a.f(x) < b
         .f(x) ? a : b; }
f60397 };
```

Range Minimum Queries

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

```
                                                                    efbc6a
4fce64 template<class T>
14c70f struct RMQ {
b47928   vector<vector<T>> jmp;
275688   RMQ(const vector<T>& V) : jmp(1, V) {
016a6d     for (int pw = 1, k = 1; pw * 2 <= (int)V.size(); pw
           *= 2, ++k) {
ced242       jmp.emplace_back(V.size() - pw * 2 + 1);
5ca12e       for (int j = 0; j < (int)jmp[k].size(); j++)
a243f1         jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw
             ]);
59961f     }
d59b89   }
1e9295   T query(int a, int b) {
d52a69     assert(a < b); // or return inf if a == b
d44154     int dep = 31 - __builtin_clz(b - a);
df6f56     return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
da30bb   }
efbc6a };
```

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

```cpp
d41d8c  // #include <bits/extc++.h>
d41d8c
75f3c2  struct chash {
0d8969      const uint64_t C = ll(4e18 * acos(0) | 71);
16eb60      ll operator () (ll x) const { return __builtin_bswap64
            (x * C); }
cdd37e  };
cdd37e
c7be5a  template <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

```cpp
d41d8c  // #include "FenwickTree.h"
d41d8c
9a350e  struct 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) {
6c809          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

```cpp
72c11f  struct 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
746fa4  struct LineContainer : multiset<Line, less<>> {
746fa4      // (for doubles, use inf = 1/.0, div(a,b) = a/b)
```

```cpp
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

```cpp
bf28ea  struct 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:
3e798e  template <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:
a8e5c9  Node* 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
9954a1  Node* 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;
```

```cpp
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
ea439d  Node 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

```cpp
bf28ea  struct 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
6c5593  int cnt(Node* n) { return n ? n->c : 0; }
371cf9  void Node::recalc() { c = cnt(l) + cnt(r) + 1; }
371cf9
6b5795  template<class F> void each(Node* n, F f) {
19c27d      if (n) { each(n->l, f); f(n->val); each(n->r, f); }
cfbf7f  }
cfbf7f
0d52f8  pair<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
27f149  Node* 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
ba8bef  Node* ins(Node* t, Node* n, int pos) {
28b80c      auto [l,r] = split(t, pos);
6edc77      return merge(merge(l, n), r);
47352e  }
```

```
47352e // Example application: move the range [l, r) to index k
43d58d void 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

```
47a5e9 struct 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) {
460ec9     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

```
137ebf struct wavelet_tree{
2f784e   #define vi vector<int>
6a3389   #define pb push_back
bd5515   int lo, hi;
441e87   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   //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 ($\sim$x).
Usage: TwoSat ts(number of boolean variables) ts.implies(0, $\sim$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

```
d9d94e struct 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

```
14da59 bool 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
a5bc87 int 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
33e98d  struct 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: $\mathcal{O}(E + V)$

```
b04982  vector<int> val, comp, z, cont;
4df60f  int Time, ncomps;
29453f  template<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  }
ff80b2  template<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.

```
d26414  vector<int> lw, nm, pa, art;
561ea9  vector<pair<int, int>> brd;
c5abfe  int tt, ch, rt;
c5abfe
b41f22  void 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
d2205f  void 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: $\mathcal{O}(VE)$

```
f5e3e7  const ll inf = LLONG_MAX;
5567e9  struct Ed { int a, b, w, s() { return a < b ? a : -a;
          }};
2045f7  struct Node { ll dist = inf; int prev = -1; };
2045f7
019c78  void 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: $\mathcal{O}(E + V)$

```
fbe043  vector<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]) {
696d20            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<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;
41368c   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);
07adbc   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 $U = \max |cap|$.

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

```
                                                                      db429d
struct Dinic {
    struct Edge {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if you
            need flows
    };
    vector<int> lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, adj[b].size(), c, c});
        adj[b].push_back({a, adj[a].size() - 1, rcap, rcap})
            ;
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < (int)adj[v].size(); i++) {
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0; q[0] = s;
        for (int L = 0; L < 31; L++) do {
            lvl = ptr = vector<int>(q.size());
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) {
                int v = q[qi++];
                for (Edge e : adj[v])
                    if (!lvl[e.to] && e.c >> (30 - L))
                        q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
            }
            while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
        } while (lvl[t]);
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

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
// #include "../data-structures/UnionFindRollback.h"

struct Edge { int a, b; ll w; };
struct Node { /// lazy skew heap node
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ?: b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vector<int>> dmst(int n, int r, vector<Edge>& g
    ) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node
        {e});
    ll res = 0;
    vector<int> seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1,-1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    for (int s = 0; s < n; s++) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1,{}};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) { /// found cycle, contract
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {&Q[qi], &Q[end]}});
            }
        }
        for (int i = 0; i < qi; i++) in[uf.find(Q[i].b)] = Q
            [i];
    }

    for (auto& [u,t,comp] : cycs) { // restore sol (
        optional)
        uf.rollback(t);
        Edge inEdge = in[u];
        for (auto& e : comp) in[uf.find(e.b)] = e;
        in[uf.find(inEdge.b)] = inEdge;
    }
    for (int i = 0; i < n; i++) par[i] = in[i].a;
    return {res, par};
}
```

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
vector<int> edgeColoring(int N, vector<pair<int, int>>&
    eds) {
    vector<int> cc(N + 1), ret(eds.size()), fan(N), free(N
        ), loc;
    for (auto e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vector<int>> adj(N, vector<int>(ncols, -1));
    for (pair<int, int> e : eds) {
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) !=
            -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at
            ][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    for (int i = 0; i < (int)eds.size(); i++)
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret
            [i];
    return ret;
}
```

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
vector<int> eulerWalk(vector<vector<pair<int, int>>>& gr
    , int nedges, int src=0) {
    int n = gr.size();
    vector<int> D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = gr[x].
        size();
        if (it == end){ ret.push_back(x); s.pop_back();
        continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }}
    for (int x : D) if (x < 0 || ret.size() != nedges+1)
        return {};
    return {ret.rbegin(), ret.rend()};
}
```

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 $m[i][j] = \texttt{inf}$ if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, $\texttt{inf}$ if no path, or $\texttt{-inf}$ if the path goes through a negative-weight cycle.

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

```
                                                                      cf07b8
```

```
96441f const ll inf = 1LL << 62;
433b02 void floydWarshall(vector<vector<ll>>& m) {
aab24c   int n = m.size();
d21013   for (int i = 0; i < n; i++) m[i][i] = min(m[i][i], 0LL
         );
858ba6   for (int k = 0; k < n; k++)
104052     for (int i = 0; i < n; i++)
4bf791       for (int j = 0; j < n; j++)
b46e39         if (m[i][k] != inf && m[k][j] != inf) {
6cf776           auto newDist = max(m[i][k] + m[k][j], -inf);
80dc22           m[i][j] = min(m[i][j], newDist);
2cd540         }
ceef13   for (int k = 0; k < n; k++) if (m[k][k] < 0)
70fcf1     for (int i = 0; i < n; i++)
8c30d7       for (int j = 0; j < n; j++)
92c3f5         if (m[i][k] != inf && m[k][j] != inf) m[i][j] =
         -inf;
cf07b8 }
```

General Matching

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

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

```
                                                              7389c1
d41d8c // #include "../numerical/MatrixInverse-mod.h"
d41d8c
75fcdd vector<pair<int, int>> generalMatching(int N, vector<
       pair<int, int>>& ed) {
892b78   vector<vector<ll>> mat(N, vector<ll>(N)), A;
5789ef   for (auto pa : ed) {
30f40e     int a = pa.first, b = pa.second, r = rand() % mod;
37e4d9     mat[a][b] = r, mat[b][a] = (mod - r) % mod;
ccc1d2   }
ccc1d2
03ba4b   int r = matInv(A = mat), M = 2*N - r, fi, fj;
c57a0e   assert(r % 2 == 0);
c57a0e
e3ab96   if (M != N) do {
d0b33d     mat.resize(M, vector<ll>(M));
8bd063     for (int i = 0; i < N; i++) {
603144       mat[i].resize(M);
edc7da       for (int j = N; j < M; j++) {
bcfeff         int r = rand() % mod;
dc1b6c         mat[i][j] = r, mat[j][i] = (mod - r) % mod;
1eb54f       }
211d22     }
81dc1f   } while (matInv(A = mat) != M);
81dc1f
afa7f1   vector<int> has(M, 1); vector<pair<int, int>> ret;
aadd58   for (int it = 0; it < M / 2; it++) {
3496cc     for (int i = 0; i < M; i++) if (has[i])
be05a6       for (int j = i + 1; j < M; j++) if (A[i][j] && mat
       [i][j]) {
b7e188         fi = i; fj = j; goto done;
d251b9       } assert(0); done:
00ab32     if (fj < N) ret.emplace_back(fi, fj);
9e315f     has[fi] = has[fj] = 0;
b98121     for (int sw = 0; sw < 2; sw++) {
c3cac9       ll a = modpow(A[fi][fj], mod-2);
c9ac23       for (int i = 0; i < M; i++) if (has[i] && A[i][fj
       ]) {
41aca4         ll b = A[i][fj] * a % mod;
cf9147         for (int j = 0; j < M; j++) A[i][j] = (A[i][j] -
       A[fi][j] * b) % mod;
0795c8       }
b1a70a       swap(fi,fj);
d1b006     }
89343e   }
8f3c60   return ret;
7389c1 }
```

Global Minimum Cut

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

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

```
                                                              1ae302
998236 pair<int, vector<int>> globalMinCut(vector<vector<int>>
       mat) {
cc2329   pair<int, vector<int>> best = {INT_MAX, {}};
6e6907   int n = mat.size();
078db0   vector<vector<int>> co(n);
30bded   for (int i = 0; i < n; i++) co[i] = {i};
b13f78   for (int ph = 1; ph < n; ph++) {
24ca9b     vector<int> w = mat[0];
e13dd0     size_t s = 0, t = 0;
0d930e     for (int it = 0; it < n - ph; it++) { // O(V^2) -> O
       (E log V) with prio. queue
5ba239       w[t] = INT_MIN;
37cd7c       s = t, t = max_element(w.begin(), w.end()) - w.
       begin();
42d91b       for (int i = 0; i < n; i++) w[i] += mat[t][i];
147091     }
679d40     best = min(best, {w[t] - mat[t][t], co[t]});
b7fbc7     co[s].insert(co[s].end(), co[t].begin(), co[t].end()
       );
64e78c     for (int i = 0; i < n; i++) mat[s][i] += mat[t][i];
c07778     for (int i = 0; i < n; i++) mat[i][s] = mat[s][i];
2efbe7     mat[0][t] = INT_MIN;
074af6   }
5ca6d4   return best;
1ae302 }
```

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

```
                                                              291aa9
d41d8c // #include "PushRelabel.h"
d41d8c
2d0038 typedef array<ll, 3> Edge;
55d44c vector<Edge> gomoryHu(int N, vector<Edge> ed) {
ec4f34   vector<Edge> tree;
cf2bc7   vector<int> par(N);
155edc   for (int i = 1; i < N; i++) {
c1ec86     PushRelabel D(N); // Dinic also works
4aeb96     for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
afd7b4     tree.push_back({i, par[i], D.calc(i, par[i])});
daa146     for (int j = i+1; j < N; j++)
7e46f4       if (par[j] == par[i] && D.leftOfMinCut(j)) par[j]
       = i;
0a52f0   }
b63797   return tree;
291aa9 }
```

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 $N \le M$.

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

```
                                                              bdc2be
0d4430 pair<int, vector<int>> hungarian(const vector<vector<int
       >> &a) {
49a369   if (a.empty()) return {0, {}};
```

```
04780a   int n = a.size() + 1, m = a[0].size() + 1;
7a22a6   vector<int> u(n), v(m), p(m), ans(n - 1);
6c1c96   for (int i = 1; i < n; i++) {
067ab1     p[0] = i;
b664ef     int j0 = 0; // add "dummy" worker 0
5a10a8     vector<int> dist(m, INT_MAX), pre(m, -1);
182e7a     vector<bool> done(m + 1);
565e8b     do { // dijkstra
66c443       done[j0] = true;
3458f3       int i0 = p[j0], j1, delta = INT_MAX;
71555e       for (int j = 1; j < m; j++) if (!done[j]) {
2bb1c6         auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
8ada1c         if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
6fd6fb         if (dist[j] < delta) delta = dist[j], j1 = j;
c0194f       }
7f2af5       for (int j = 0; j < m; j++) {
9bf35f         if (done[j]) u[p[j]] += delta, v[j] -= delta;
3bf594         else dist[j] -= delta;
6690b0       }
5abc0e       j0 = j1;
df2e64     } while (p[j0]);
7ad364     while (j0) { // update alternating path
b8e757       int j1 = pre[j0];
5c226f       p[j0] = p[j1], j0 = j1;
528e93     }
eea634   }
           for (int j = 1; j < m; j++) if (p[j]) ans[p[j] - 1] =
       j - 1;
202184   return {-v[0], ans}; // min cost
bdc2be }
```

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 $O(\log N)$.

```
                                                              0fb462
bf28ea struct Node { // Splay tree. Root's pp contains tree's
       parent.
0dc895   Node *p = 0, *pp = 0, *c[2];
038f31   bool flip = 0;
210611   Node() { c[0] = c[1] = 0; fix(); }
a4e156   void fix() {
5b7890     if (c[0]) c[0]->p = this;
577fff     if (c[1]) c[1]->p = this;
577fff     // (+ update sum of subtree elements etc. if wanted)
4268f1   }
34cb58   void pushFlip() {
1b908c     if (!flip) return;
a0ef26     flip = 0; swap(c[0], c[1]);
da653a     if (c[0]) c[0]->flip ^= 1;
168072     if (c[1]) c[1]->flip ^= 1;
d94cfc   }
829eb8   int up() { return p ? p->c[1] == this : -1; }
b374bb   void rot(int i, int b) {
f8bc45     int h = i ^ b;
042831     Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ?
       y : x;
679f6a     if ((y->p = p)) p->c[up()] = y;
59c9a7     c[i] = z->c[i ^ 1];
9fc417     if (b < 2) {
0ef3d2       x->c[h] = y->c[h ^ 1];
17a30e       y->c[h ^ 1] = x;
653614     }
3eddae     z->c[i ^ 1] = this;
395960     fix(); x->fix(); y->fix();
03a4e1     if (p) p->fix();
8a07c8     swap(pp, y->pp);
966070   }
```

```
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
d41d8c ///   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;
913d3a 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 });
9e2427        }
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
2276e  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());
91921a    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. $\mathrm{modLog}(a,1,m)$ 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. $\mathrm{modsum}(\text{to}, c, k, m) = \sum_{i=0}^{\text{to}-1} (ki+c)\%m$. 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
f47760  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;
102494    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: $\mathcal{O}(N^3)$

$$\text{-- bd5cec}$$

```
e36c74 double 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 $N = 2^{22}$)

$$\text{-- 3dd197}$$

```
bccabc typedef complex<double> C;
b05ddb typedef vector<double> vd;
760a36 void 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 }
bf0709 vd 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 $N \log_2 N \cdot mod < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, mod)$.

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

$$\text{-- b82773}$$

```
d41d8c // #include "FastFourierTransform.h"
d41d8c
192b04 typedef vector<ll> vl;
1dbf8b template<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: $\mathcal{O}(N \log N)$

$$\text{-- 25c175}$$

```
ac2a38 void 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
ae85b6          /// 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 }
cef5d7 vi 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: $\mathcal{O}(\log((b - a)/\epsilon))$

$$\text{-- 31d45b}$$

```
d41d8c /// It is important for r to be precise, otherwise we
          don't necessarily maintain the inequality a < x1 < x2
          < b.
eb1b64 double 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.

$$\text{-- 8eeeaf}$$

```
9eb631 typedef array<double, 2> P;
9eb631
710806 template<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 $S[0\ldots \geq n-1]$ and $tr[0\ldots n-1]$. 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 $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ 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 $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ 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
28e96e   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];
f44d51   }
f44d51
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 $2^ab + 1$, 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
b5e822  const 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.
7458ca  typedef vector<ll> vl;
0ca385  void 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  }
bbaf00  vl 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),
c0b0e4e      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
ae03ae  typedef vector<double> vd;
28ccec  vd 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 x²-3x+2 = 0`

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

```
                                              b00bfe
d41d8c  // #include "Polynomial.h"
d41d8c
64af29  vector<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
213314  struct 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
943c93  typedef double T; // long double, Rational, double + mod
                <P>...
4a7fa3  typedef vector<T> vd;
19471c  typedef vector<vd> vvd;
19471c
6296c1  const 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
004b50  struct LPSolver {
34f6a6    int m, n;
a8b98c    vi N, B;
a50829    vvd D;
a50829
e8814c    LPSolver(const vvd& A, const vd& b, const vd& c) :
            m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
09ecbe      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
a00ca8      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] =
                b[i];}
eab15d      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
03bb56      N[n] = -1; D[m+1][n] = 1;
4c20cd    }
dcadf8
dcadf8    void pivot(int r, int s) {
d2dadf      T *a = D[r].data(), inv = 1 / a[s];
72cb06      rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
93b9bd        T *b = D[i].data(), inv2 = b[s] * inv;
a86c76        rep(j,0,n+2) b[j] -= a[j] * inv2;
c1f31d        b[s] = a[s] * inv2;
ee22d8      }
df792b      rep(j,0,n+2) if (j != s) D[r][j] *= inv;
d3cb55      rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
9e2376      D[r][s] = inv;
6bf9c5      swap(B[r], N[s]);
b3404b    }
193de8
193de8    bool simplex(int phase) {
ede257      int x = m + phase - 1;
f695c2      for (;;) {
0aa9db        int s = -1;
8b65cd        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
96f50e        if (D[x][s] >= -eps) return true;
e72781        int r = -1;
fcd18c        rep(i,0,m) {
a7d0e5          if (D[i][s] <= eps) continue;
f65882          if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
01fd61                        < MP(D[r][n+1] / D[r][s], B[r])) r
8af3f7            = 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

```
ae03ae typedef vector<double> vd;
1784ea const double eps = 1e-12;
1784ea
dbdd92 int 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
f9498c rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
f9498c // ... then at the end:
3b9d4d x.assign(m, undefined);
45bf44 rep(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];
08e495 fail:; }
```

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

```
9831fe typedef bitset<1000> bs;
9831fe
1dc5af int 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

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

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

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

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

```
943c93 typedef double T;
b20c01 vector<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

```
a5630e vector <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
b345c0 vector <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

```
aa956c array<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;
```

```
50aaeb      while (L>=1 && R+1<n && s[L-1] == s[R+1])
8d40fb        p[z][i]++, L--, R++;
bf129e      if (R>r) l=L, r=R;
bc88f0    }
15f35b    return p;
61383b }
```

Minimum rotation

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

```
                                                                          4bb91c
5fa8d6 int minRotation(string s) {
cfc6e5    int a=0, N=s.size(); s += s;
62f43d    for (int b = 0; b < N; b++) for (int k = 0; k < N; k
           ++) {
8fbae9      if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
           break;}
bcec3c      if (s[a+k] > s[b+k]) { a = b; break; }
f7d0b4    }
6cb331    return a;
4bb91c }
```

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.

```
                                                                          2e25f9
c5aa9e struct RH {
64eb2a    int MD, n, b, ib; // b is base, ib inverse base mod MD
3b195e    vector<int> p, ip, hs;
011265    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
74c3ce      p[0] = ip[0] = 1;
d28127      hs[0] = s[0];
5bb806      for(int i = 1; i < n; ++i){
3f448a        p[i] = (ll) p[i - 1] * b % MD;
4870cc        ip[i] = (ll) ip[i - 1] * ib % MD;
66aa32        hs[i] = ((ll) s[i] * p[i] + hs[i - 1]) % MD; // s[
           i] can be changed to some hash function
adef78      }
1e7e6b    }
16c258    int hash(int l, int r){
d9aae2      return (ll) (hs[r] - (l ? hs[l - 1] : 0) + MD) * ip[
           l] % MD;
1379de    }
2e25f9 };
```

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

```
                                                                          3d234e
10747d struct SA {
31fdad    struct State {
fad143      int length;
7e049f      int link;
ec43e2      int next[26];
209696      int cnt;
0a95ea      bool is_clone;
dafc14      int first_pos;
0fbc43      State(int _length, int _link) :
578718      length(_length),
8f88e0      link(_link),
05402c      cnt(0),
```

```
c214c3      is_clone(false),
c445b2      first_pos(-1)
df1390      {
24aab0        memset(next, -1, sizeof(next));
c13076      }
575a7c    };
c5435a    std::vector<State> states;
0c2d55    int size;
dadfdf    int last;
26a9fe    bool did_init_count;
7c701c    int str_len;
339b92    bool did_init_css;
edd2c0    SA() :
247d2e    states(1, State(0, -1)),
27dd74    size(1),
f6f1cc    last(0),
b25e35    did_init_count(false),
5b001a    str_len(0),
1d383e    did_init_css(false)
18e6a6    { }
ca6810    void push(char c) {
525d03      str_len++;
8f2dae      did_init_count = false;
4a4bd8      did_init_css = false;
26359b      int cur = size;
d5aba5      states.resize(++size, State(states[last].length + 1,
           -1));
01ccfe      states[cur].first_pos = states[cur].length - 1;
106f4e      int p = last;
5f2312      while (p != -1 && states[p].next[c - 'a'] == -1) {
67b05d        states[p].next[c - 'a'] = cur;
73ba4b        p = states[p].link;
0db291      }
a55669      if (p == -1) {
0cd45a        states[cur].link = 0;
577086      } else {
c98ad9        int q = states[p].next[c - 'a'];
6024e1        if (states[p].length + 1 == states[q].length) {
1de958          states[cur].link = q;
930e14        } else {
aed05d          int clone = size;
afbe23          states.resize(++size, State(states[p].length +
           1, states[q].link));
4443c2          states[clone].is_clone = true;
af2be1          memcpy(states[clone].next, states[q].next,
           sizeof(State::next));
61ac3d          states[clone].first_pos = states[q].first_pos;
13bea7          while (p != -1 && states[p].next[c - 'a'] == q)
           {
627f1c            states[p].next[c - 'a'] = clone;
411652            p = states[p].link;
20432b          }
34a7da          states[q].link = states[cur].link = clone;
98914e        }
0461f9      }
591347      last = cur;
301567    }
d0cce2    bool exists(const std::string& pattern) {
0ffabb      int node = 0;
13e5cf      int index = 0;
192e18      while (index < (int) pattern.length() && states[node
           ].next[pattern[index] - 'a'] != -1) {
efffe7        node = states[node].next[pattern[index] - 'a'];
cbf0e9        index++;
709389      }
356eef      return index == (int) pattern.size();
4db848    }
0ff9b8    int count(const std::string& pattern) {
66e217      if (!did_init_count) {
13d2c1        did_init_count = true;
702df7        for (int i = 1; i < size; i++) {
57b2d4          states[i].cnt = !states[i].is_clone;
```

```
24878a        }
9c6d77        std::vector<std::vector<int>> of_length(str_len
           + 1);
d9c5db        for (int i = 0; i < size; i++) {
c408de          of_length[states[i].length].push_back(i);
9d793e        }
e08272        for (int l = str_len; l >= 0; l--) {
e9fd3e          for (int node : of_length[l]) {
ff7da1            if (states[node].link != -1) {
fa5d99              states[states[node].link].cnt += states[node
           ].cnt;
c92599            }
9f0d9a          }
418535        }
ce47a0      }
c62dc8      int node = 0;
1a6274      int index = 0;
d32f26      while (index < (int) pattern.length() && states[node
           ].next[pattern[index] - 'a'] != -1) {
6d8dce        node = states[node].next[pattern[index] - 'a'];
1ad0b3        index++;
edf68d      }
72ab54      return index == (int) pattern.size() ? states[node].
           cnt : 0;
f7682f    }
f397ab    int first_occ(const std::string& pattern) {
53dacd      int node = 0;
6bbd47      int index = 0;
442e13      while (index < (int) pattern.length() && states[node
           ].next[pattern[index] - 'a'] != -1) {
652cc2        node = states[node].next[pattern[index] - 'a'];
8e968d        index++;
ef6d88      }
a59113      return index == (int) pattern.size() ? states[node].
           first_pos - (int) pattern.size() + 1 : -1;
a65c30    }
9afeb2    size_t count_substrings() {
a7f74b      static std::vector<size_t> dp;
9e504d      if (!did_init_css) {
9a3afa        did_init_css = true;
fce801        dp = std::vector<size_t> (size, 0);
75426a        auto dfs = [&] (auto&& self, int node) -> size_t {
673f0b          if (node == -1) {
0b0f06            return 0;
9fa531          }
99b459          if (dp[node]) {
ac9ba2            return dp[node];
519c50          }
983e54          dp[node] = 1;
1d020f          for (int i = 0; i < 26; i++) {
2e5625            dp[node] += self(self, states[node].next[i]);
515699          }
02606f          return dp[node];
b1fb1b        };
a3a17c        dfs(dfs, 0);
d8b4f0      }
8b5414      return dp[0] - 1;
e1c0a8    }
db005c };
db005c
db005c // usage example: Repeating Substring submission on cses
           .fi
2f5768 int main() {
109b3e    std::ios::sync_with_stdio(0); std::cin.tie(0);
c0bcd4    std::string s; std::cin >> s;
c9c93c    int n; std::cin >> n;
0c8f98    SA sa;
3b67c6    for (char c : s) {
5bd287      sa.push(c);
27d539    }
c64da9    sa.count("");
66d2ad    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

```
b86749 vector <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
b8e08b typedef Point3D<double> P3;
b8e08b
6aa2ed struct 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
538b68 struct F { P3 q; int a, b, c; };
538b68
7d6924 vector<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

```
755634 struct 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 };
c1efa9 bool 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.
ccb19a pair<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 }
c11d8e Angle 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 }
89aa95 Angle 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
6269ec typedef Point<double> P;
888549 bool 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
7dc51e template<class P>
0406ad vector<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
6269ec typedef Point<double> P;
cf6463 #define arg(p, q) atan2(p.cross(q), p.dot(q))
cf0d22 double 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;
```

```
702f07      auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.
      dist2();
4c3d03      auto det = a * a - b;
3710c6      if (det <= 0) return arg(p, q) * r2;
15e178      auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(
      det));
1b08d3      if (t < 0 || 1 <= s) return arg(p, q) * r2;
a53ae4      P u = p + d * s, v = p + d * t;
f0b5ed      return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
6470ed    };
dabb77    auto sum = 0.0;
48e7de    rep(i,0,sz(ps))
96a7cf      sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
677d63    return sum;
a1ee63 }
```

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

```
d41d8c // #include "Point.h"
d41d8c
7dc51e template<class P>
e80549 vector<pair<P, P>> tangents(P c1, double r1, P c2,
      double r2) {
c7e310   P d = c2 - c1;
45b12a   double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr
      ;
c18727   if (d2 == 0 || h2 < 0)  return {};
f9fd85   vector<pair<P, P>> out;
0072fe   for (double sign : {-1, 1}) {
48be0b     P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
729d07     out.push_back({c1 + v * r1, c2 + v * r2});
41b560   }
2313ea   if (h2 == 0) out.pop_back();
054e70   return out;
b0153d }
```

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

```
d41d8c // #include "Point.h"
d41d8c
6269ec typedef Point<double> P;
5995a9 double ccRadius(const P& A, const P& B, const P& C) {
2d2b60   return (B-A).dist()*(C-B).dist()*(A-C).dist()/
d37107       abs((B-A).cross(C-A))/2;
032e3d }
990f04 P ccCenter(const P& A, const P& B, const P& C) {
d94b4d   P b = C-A, c = B-A;
fc3ed0   return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)
      /2;
1caa3a }
```

Closest pair of points

Description: Yoinked from kactl. Finds the closest pair of points.
Complexity: $\mathcal{O}(n \log n)$.
—————————————————————————————————————— ac41a6

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

```
2c0584 typedef Point<ll> P;
7549f9 pair<P, P> closest(vector<P> v) {
b02c53   assert(sz(v) > 1);
8f0c0e   set<P> S;
9e7fdf   sort(all(v), [](P a, P b) { return a.y < b.y; });
db620d   pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
2ac587   int j = 0;
14a5ea   for (P p : v) {
484ee7     P d{1 + (ll)sqrt(ret.first), 0};
0a3d44     while (v[j].y <= p.y - d.x) S.erase(v[j++]);
270154     auto lo = S.lower_bound(p - d), hi = S.upper_bound(p
      + d);
e75de8     for (; lo != hi; ++lo)
4128f5       ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
afb942     S.insert(p);
a4382b   }
65a931   return ret.second;
ac41a6 }
```

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

```
d41d8c // #include "Point.h"
d41d8c
2c0584 typedef Point<ll> P;
af1648 vector<P> convexHull(vector<P> pts) {
bf096e   if (sz(pts) <= 1) return pts;
086de3   sort(all(pts));
3e3497   vector<P> h(sz(pts)+1);
cc9643   int s = 0, t = 0;
8b7a3b   for (int it = 2; it--; s = --t, reverse(all(pts)))
2fd8c4     for (P p : pts) {
e7eb7c       while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0)
      t--;
f4a7b9       h[t++] = p;
56ca78     }
b08f4b   return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
      h[1])};
310954 }
```

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

```
d41d8c // #include "Point.h"
d41d8c // #include "3d_hull.h"
d41d8c
6abbcc template<class P, class F>
b5fdca void delaunay(vector<P>& ps, F trifun) {
6b1956   if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2])
      < 0);
0c9f52     trifun(0,1+d,2-d); }
d1e435   vector<P3> p3;
3ff622   for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
263f28   if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3
      [t.a]).
cf39a1     cross(p3[t.c]-p3[t.a]).dot(P3(0,0,1)) < 0)
c20439     trifun(t.a, t.c, t.b);
c0e7bc }
```

Dynamic Convex Hull

Description: Supports building a convex hull one point at a time. Viewing the convex hull along the way.
—————————————————————————————————————— 431bba

```
be520b struct point {
0196fa   ll x, y;
f2e821   point(ll x=0, ll y=0): x(x), y(y) {}
0293d7   point operator-(const point &p) const { return point
      (x-p.x, y-p.y); }
5dae65   point operator*(const ll k) const { return point(k*x
      , k*y); }
f50d29   ll cross(const point &p) const { return x*p.y - p.x*
      y; }
9d44db   bool operator<(const point &p) const { return x < p.
      x || x == p.x && y < p.y; }
77f7cb };
77f7cb
2ce416 bool above(set<point> &hull, point p, ll scale = 1) {
b5ac08   auto it = hull.lower_bound(point((p.x+scale-1)/scale
      , 0));
75d58b   if (it == hull.end()) return true;
b7dcd8   if (p.y <= it->y*scale) return false;
fb2eae   if (it == hull.begin()) return true;
8a5eb9   auto jt = it--;
a7a017   return (p-*it*scale).cross(*jt-*it) < 0;
ecae32 }
ecae32
2b34b3 void add(set<point> &hull, point p) {
de0486   if (!above(hull, p)) return;
0a152b   auto pit = hull.insert(p).first;
3ba588   while (pit != hull.begin()) {
2b6ffc     auto it = prev(pit);
9de99b     if (it->y <= p.y || (it != hull.begin() && (*it
      -*prev(it)).cross(*pit-*it) >= 0))
65eae8       hull.erase(it);
d03c84     else
87aefe       break;
f787d7   }
2f06a3   auto it = next(pit);
78b06b   while (it != hull.end()) {
d7d62c     if (next(it) != hull.end() && (*it-p).cross(*
      next(it)-*it) >= 0)
b4dd19       hull.erase(it++);
6f504f     else
ae162a       break;
7a0510   }
431bba }
```

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

```
d41d8c // #include "Point.h"
d41d8c
2c0584 typedef Point<ll> P;
28b700 array<P, 2> hullDiameter(vector<P> S) {
9bdd0c   int n = sz(S), j = n < 2 ? 0 : 1;
12ea1a   pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
5c70ae   rep(i,0,j)
e5ff70     for (;; j = (j + 1) % n) {
26329e       res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j
      ]}});
e7f091       if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i])
      >= 0)
49f898         break;
cf85e0     }
d9bfba   return res.second;
c571b8 }
```

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
// #include "Point.h"
// #include "On_segment.h"
// #include "Segment_distance.h"

template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict
;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q)
 > 0;
  }
  return cnt;
}
```

KD-tree

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

```
                                                    bac5b0
// #include "Point.h"

typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a
 point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child
(not
      // best performance with many duplicates in the
 middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
```

```
                                                    bac5b0
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})
) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point
 itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt)
;
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared
 distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

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
// #include "Point.h"

#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(
j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n)
 < 0
template <class P> int extrVertex(vector<P>& poly, P dir
) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo
) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
```

```
                                                    7cf45b
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly))
    {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

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
// #include "Point.h"

template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

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
// #include "Point.h"

template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

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

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq)
    );
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.
    dist2();
}
```

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: $\mathcal{O}(n \log n)$.

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

typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
  vi id(sz(ps));
  iota(all(id), 0);
  vector<array<int, 3>> edges;
  rep(k,0,4) {
    sort(all(id), [&](int i, int j) {
      return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
    map<int, int> sweep;
    for (int i : id) {
      for (auto it = sweep.lower_bound(-ps[i].y);
           it != sweep.end(); sweep.erase(it++)) {
        int j = it->second;
        P d = ps[i] - ps[j];
        if (d.y > d.x) break;
        edges.push_back({d.y + d.x, i, j});
      }
      sweep[-ps[i].y] = i;
    }
    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x
      , p.y);
  }
  return edges;
}
```

Minimum enclosing circle

Description: Yoinked from kactl. Computes the minimum circle that encloses a set of points.

Complexity: $\mathcal{O}(n)$.

```
// #include "circumcircle.h"

pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

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>.

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

template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

2D Point

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

```
template <class T> int sgn(T x) { return (x > 0) - (x <
    0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,
    p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,
    p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*
    this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist
    ()=1
  P perp() const { return P(-y, x); } // rotates +90
    degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the
    origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

3D Point

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

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(
    z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z)
    ; }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z)
    ; }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x
    );
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-
    pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval
    [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z);
    }
  P unit() const { return *this/(T)dist(); } //makes
    dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around
    axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.
    unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

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: $\mathcal{O}(\log n)$.

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

typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true)
    {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p
    );
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p
    ) <= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

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!

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

template<class T>
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
6269ec typedef Point<double> P;
fa2dc3 P 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
6269ec typedef Point<double> P;
b4b253 vector<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
6269ec typedef Point<double> P;
940b75 double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
51eb9c double 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       }
97ea86       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
f9cf71 template<class V, class L>
8b5f1f double 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
6269ec typedef Point<double> P;
789af4 double 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
dae11d template<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
7dc51e template<class P>
fad9c9 int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
fad9c9
bb2891 template<class P>
059ae5 int 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
c5faf9 double 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.

```
// #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: $\mathcal{O}(N \log N)$

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

```
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: $\mathcal{O}(E() \cdot log_{1/u}(T))$.

```
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.

```
// 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.

```
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.

```
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: $\mathcal{O}(N \max(w_i))$

```
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 $[0, 2b)$.

```
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

```
f7d7f8 set<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
b05726 void 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

```
24b8d1 template<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 $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. 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;}
1823b8 int i = from;
dbde22 while (i&3 && i < to) F // for alignment, if needed
4379e1 while (i + 4 <= to) { F F F F }
520e76 while (i < to) F
```

Xor basis

Description: Basis of vectors in Z_2^d

61b70d

```
bf37aa struct 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 };
```

Last_minute

Implicit 2D segment tree

Description: Classic implicit 2D segment tree taken from my solution to IOI game 2013. It is in rough shape, but it works. Designed to be [inclusive, exclusive). It is old and looks shady, only rely slightly on it, maybe even just make a new one if you need one.
Usage: See usage example at the bottom.
Complexity: $\mathcal{O}(\log^2 n)$ per operation *I think*.

ae92aa

```
299b05 constexpr const int MX_RC = 1 << 30;
299b05
a3032e struct Inner {
493223   long long val;
140d19   int lv, rv;
4cb72f   Inner* lc,* rc;
f24e1f   Inner(long long _val, int _l, int _r) :
3cdb99   val(_val), lv(_l), rv(_r), lc(nullptr), rc(nullptr)
ab764d   { }
60af3c   ~Inner() {
2e9793     delete(lc);
02da3e     delete(rc);
b8d074   }
00e411   void update(int ind, long long nev, int l = 0, int r =
         MX_RC) {
ca7a61     if (!(r - l - 1)) {
226ff1       assert(lv == l && rv == r);
bac672       assert(ind == l);
a41337       val = nev;
b0b081       return;
78f219     }
23eba4     int mid = (l + r) >> 1;
286913     if (ind < mid) {
```

```
246e24       if (lc) {
3a66b2         if (lc->lv != l || lc->rv != mid) {
926f8a           Inner* tmp = lc;
c8fd20           lc = new Inner(0, l, mid);
6536fd           (tmp->lv < ((l + mid) >> 1) ? lc->lc : lc->rc)
       = tmp;
94bb73         }
e88f6e         lc->update(ind, nev, l, mid);
a69813       } else lc = new Inner(nev, ind, ind + 1);
1d67a7     } else {
a18480       if (rc) {
849ed9         if (rc->lv != mid || rc->rv != r) {
3d82c2           Inner* tmp = rc;
08e48b           rc = new Inner(0, mid, r);
3cf492           (tmp->lv < ((mid + r) >> 1) ? rc->lc : rc->rc)
       = tmp;
18683f         }
1ddbfc         rc->update(ind, nev, mid, r);
637a18       } else rc = new Inner(nev, ind, ind + 1);
1ea254     }
97c33a     val = std::gcd(lc ? lc->val : 0, rc ? rc->val : 0);
45be42   }
66c546   long long query(int tl, int tr, int l = 0, int r =
       MX_RC) {
a00435     if (l >= tr || r <= tl) return 0;
edccb1     if (!(rv - lv - 1)) {
81886f       if (lv >= tr || rv <= tl) return 0;
6228a6       return val;
c4aa5d     }
0dbaae     assert(l == lv && r == rv);
791073     if (l >= tl && r <= tr) return val;
88a336     int mid = (l + r) >> 1;
b766e2     return std::gcd(lc ? lc->query(tl, tr, l, mid) : 0,
       rc ? rc->query(tl, tr, mid, r) : 0);
3c130a   }
f0c650   void fill(Inner* source) {
a568f5     val = source->val;
13392a     if (!(lv - rv - 1)) return;
221c61     if (source->lc) {
e7c4fa       lc = new Inner(source->lc->val, source->lc->lv,
       source->lc->rv);
74f1f6       lc->fill(source->lc);
071c5b     }
ad50a0     if (source->rc) {
9adebe       rc = new Inner(source->rc->val, source->rc->lv,
       source->rc->rv);
946ac9       rc->fill(source->rc);
b8bed9     }
c66f9e   }
ca99e3 };
ca99e3
fc64b2 struct Outer {
5d6d11   Inner* inner;
999186   int lv, rv;
9777b6   Outer* lc,* rc;
0d648e   Outer(Inner* _inner, int _l, int _r) :
b56d7c   inner(_inner), lv(_l), rv(_r), lc(nullptr), rc(nullptr
       )
6940a1   { }
262130   void update(int ind_outer, int ind_inner, long long
       nev, int l = 0, int r = MX_RC) {
a44e79     if (!(r - l - 1)) {
42e19d       assert(lv == l && rv == r);
5de54d       assert(ind_outer == l);
084529       assert(inner);
01581a       inner->update(ind_inner, nev);
66ce83       return;
922db4     }
4a146c     int mid = (l + r) >> 1;
ad897f     if (ind_outer < mid) {
033f38       if (lc) {
8382cb         if (lc->lv != l || lc->rv != mid) {
```

```
90c8a1            Outer* tmp = lc;
68043c            lc = new Outer(new Inner(0, 0, MX_RC), l, mid)
        ;
bb30e9            lc->inner->fill(tmp->inner);
dd2110            (tmp->lv < ((l + mid) >> 1) ? lc->lc : lc->rc)
        = tmp;
238e44          }
1b68a4          lc->update(ind_outer, ind_inner, nev, l, mid);
d00de8        } else {
ec9cc5          lc = new Outer(new Inner(0, 0, MX_RC), ind_outer
        , ind_outer + 1);
634434          lc->inner->update(ind_inner, nev);
d50020        }
b10d2d      } else {
dea3a0        if (rc) {
8dd98c          if (rc->lv != mid || rc->rv != r) {
4ffdfc            Outer* tmp = rc;
30bea6            rc = new Outer(new Inner(0, 0, MX_RC), mid, r)
        ;
92bf85            rc->inner->fill(tmp->inner);
9fa89c            (tmp->lv < ((mid + r) >> 1) ? rc->lc : rc->rc)
        = tmp;
d7a050          }
f1f0a4          rc->update(ind_outer, ind_inner, nev, mid, r);
ce85a4        } else {
2c95cc          rc = new Outer(new Inner(nev, 0, MX_RC),
        ind_outer, ind_outer + 1);
28a30e          rc->inner->update(ind_inner, nev);
814060        }
5306be      }
58d7c9      inner->update(ind_inner, std::gcd(
481d69      lc ? lc->inner->query(ind_inner, ind_inner + 1) : 0,
5841e8      rc ? rc->inner->query(ind_inner, ind_inner + 1) : 0)
        );
f96d2a    }
3aa830    long long query(int tl_outer, int tr_outer, int
        tl_inner, int tr_inner, int l = 0, int r = MX_RC) {
066056      if (l >= tr_outer || r <= tl_outer) return 0;
3c5cd3      if (!(rv - lv - 1)) {
d45d84        if (lv >= tr_outer || rv <= tl_outer) return 0;
9a1950        return inner->query(tl_inner, tr_inner);
818e67      }
a36529      assert(l == lv && r == rv);
248d9f      if (l >= tl_outer && r <= tr_outer)
d023a8        return inner->query(tl_inner, tr_inner);
091cda      int mid = (l + r) >> 1;
555dd1      return std::gcd(
5b0f33      lc ? lc->query(tl_outer, tr_outer, tl_inner,
        tr_inner, l, mid) : 0,
1dbd44      rc ? rc->query(tl_outer, tr_outer, tl_inner,
        tr_inner, mid, r) : 0);
aae6cb    }
82e377  };
82e377
82e377  // this is how it has been used in the solution to IOI
        game 2013
319c4c  Outer root(new Inner(0, 0, MX_RC), 0, MX_RC);
1b943b  void update(int r, int c, long long k) {
b78d10    root.update(r, c, k);
a445e8  }
07e107  long long calculate(int r_l, int c_l, int r_r, int c_r)
        {
2f2876    return root.query(r_l, r_r + 1, c_l, c_r + 1);
ae92aa  }
```

(FROM CHAT-GPT) Lazy Treap

Description: Yoinked from CHAT-GPT. Treap supporting entire tree updates/queries + split/merge. Tested a bit.

Complexity: $\mathcal{O}(\log n)$ or better on average.

```
                                                        079e94
04e57f  namespace gpt {
```

```
04e57f  struct Node {
bd90c7    int val;              // Value of the node
82a88d    int priority;         // Priority for treap balancing
936861    int size;             // Size of the subtree rooted at
f06b6e  this node
470da1    int subtree_min;      // Minimum value in the subtree
2cfef7    int lazy_add;         // Lazy propagation value
23326b    std::shared_ptr <Node> left;      // Left child
a12a7d    std::shared_ptr <Node> right;     // Right child
a12a7d
a7d60f    Node(int _val) : val(_val), priority(std::rand()),
        size(1), subtree_min(_val),
df6847            lazy_add(0), left(nullptr), right(nullptr)
        {}
cf00b1  };
cf00b1
927367  int Size(std::shared_ptr <Node> node) {
be2e7c    return node ? node->size : 0;
57ac95  }
57ac95
d96914  int SubtreeMin(std::shared_ptr <Node> node) {
70f2ce    return node ? node->subtree_min : std::
        numeric_limits<int>::max();
c6ade3  }
c6ade3
b21091  void Propagate(std::shared_ptr <Node> node) {
47147f    if (node && node->lazy_add != 0) {
3bea38      node->val += node->lazy_add;
3bea38
a81cb7      if (node->left) {
2ed017        node->left->lazy_add += node->lazy_add;
2d2865        node->left->subtree_min += node->lazy_add;
1cb442      }
949e73      if (node->right) {
809440        node->right->lazy_add += node->lazy_add;
60b193        node->right->subtree_min += node->lazy_add;
b2dcee      }
b2dcee
733db5      node->lazy_add = 0;
733db5
733db5      // Update subtree_min after propagation
3f1c9f      node->subtree_min = std::min(node->val, std::min(
        SubtreeMin(node->left), SubtreeMin(node->right)));
ac1fec    }
96381e  }
96381e
92dc9a  void Update(std::shared_ptr <Node> node) {
569774    if (node) {
46d481      node->size = 1 + Size(node->left) + Size(node->
        right);
71da08      node->subtree_min = std::min(node->val, std::min(
        SubtreeMin(node->left), SubtreeMin(node->right)));
255fb2    }
135393  }
135393
a66610  void Split(std::shared_ptr <Node> node, int index, std
        ::shared_ptr <Node>& left, std::shared_ptr <Node>&
        right) {
a35ff8    if (!node) {
7a1b78      left = right = nullptr;
5664ac      return;
96b76e    }
376919    Propagate(node);
737679    int curr_index = Size(node->left);
812dd6    if (index <= curr_index) {
d5e7f5      Split(node->left, index, left, node->left);
4ce725      right = node;
94fe0b    } else {
c0c52d      Split(node->right, index - curr_index - 1, node->
        right, right);
6e5040      left = node;
941006    }
67af59    Update(node);
```

```
72a424  }
72a424
477f6d  std::shared_ptr <Node> Merge(std::shared_ptr <Node>
        left, std::shared_ptr <Node> right) {
30f480    Propagate(left);
6a4737    Propagate(right);
c0d887    if (!left || !right) {
5679c7      return left ? left : right;
d222a5    }
ce868d    if (left->priority > right->priority) {
837b77      left->right = Merge(left->right, right);
ec1695      Update(left);
1307aa      return left;
5c4e61    } else {
edfe16      right->left = Merge(left, right->left);
2ac825      Update(right);
69733d      return right;
22b30b    }
4b207a  }
4b207a
5ca363  void AddValue(std::shared_ptr <Node> node, int value)
        {
07ffe3    if (!node) return;
98fec7    node->lazy_add += value;
666948    node->subtree_min += value;
73bfb0  }
73bfb0
0e2852  int GetMin(std::shared_ptr <Node> node) {
79ee6a    if (!node) return std::numeric_limits<int>::max();
cf5ae2    Propagate(node);
34e656    return node->subtree_min;
b325c2  }
b325c2
9fd74c  int GetMinIndex(std::shared_ptr <Node> node, int
        offset = 0) {
68c801    if (!node) return -1;
6c8c2c    Propagate(node);
371b68    int min_val = node->subtree_min;
371b68
5c0f9f    if (node->val == min_val) {
93b844      return offset + Size(node->left);
39df88    } else if (SubtreeMin(node->left) == min_val) {
dfe4e6      return GetMinIndex(node->left, offset);
94512f    } else {
31fd6e      return GetMinIndex(node->right, offset + Size(node
        ->left) + 1);
3be587    }
b66ce2  }
b66ce2
56e581  int GetValue(std::shared_ptr <Node> node, int index) {
235165    if (!node) return std::numeric_limits<int>::max();
        // Or throw an exception
856eaf    Propagate(node);
07f629    int curr_index = Size(node->left);
224972    if (index < curr_index) {
6a4181      return GetValue(node->left, index);
a8aa50    } else if (index == curr_index) {
a112e8      return node->val;
31cf82    } else {
5a000b      return GetValue(node->right, index - curr_index -
        1);
687c56    }
627bfc  }
627bfc
6967f8  void SetValue(std::shared_ptr <Node> node, int index,
        int value) {
31b57d    if (!node) return;  // Or throw an exception
aeeff0    Propagate(node);
2f644e    int curr_index = Size(node->left);
5bf1ab    if (index < curr_index) {
21b8f1      SetValue(node->left, index, value);
da67dd    } else if (index == curr_index) {
c08424      node->val = value;
```

```
      } else {
        SetValue(node->right, index - curr_index - 1,
value);
      }
      Update(node);
    }

    void InOrder(std::shared_ptr <Node> node) {
      if (!node) return;
      Propagate(node);
      InOrder(node->left);
      std::cout << node->val << " ";
      InOrder(node->right);
    }

    // Helper function to build a treap from an array
    std::shared_ptr <Node> Build(const int arr[], int l,
int r) {
      if (l > r) return nullptr;
      int m = (l + r) / 2;
      std::shared_ptr <Node> node = std::make_shared <Node
>(arr[m]);
      node->left = Build(arr, l, m - 1);
      node->right = Build(arr, m + 1, r);
      Update(node);
      return node;
    }

class Tree {

public:

    Tree() : m_data(nullptr) {
      this->m_init_random();
    }
    Tree(const std::vector <int>& data) {
      this->m_init_random();
      data.empty() ? this->m_data = nullptr :
      this->m_data = Build(data.data(), 0, data.size() -
1);
    }

    Tree split(int at) {
      assert(this->m_data && 0 <= at && at <= Size(this
->m_data));
      if (Size(this->m_data) == 0) {
        return Tree();
      }
      Tree result;
      std::shared_ptr <Node> left, right;
      Split(this->m_data, at, left, right);
      this->m_data = left;
      result.m_data = right;
      return result;
    }

    void merge(Tree& rhs) {
      this->m_data = Merge(this->m_data, rhs.m_data);
      rhs.m_data = nullptr;
    }

    void add_value(int value) {
      if (this->m_data) {
        AddValue(this->m_data, value);
      }
    }

    int get_min_index() {
      assert(this->m_data);
      return GetMinIndex(this->m_data);
    }

    int at(int index) {
```

```
      assert(this->m_data && 0 <= index && index < Size(
this->m_data));
      return GetValue(this->m_data, index);
    }

    void set(int index, int value) {
      assert(this->m_data && 0 <= index && index < Size(
this->m_data));
      SetValue(this->m_data, index, value);
    }

    int size() {
      return Size(this->m_data);
    }

private:

    std::shared_ptr <Node> m_data;

private:

    void m_init_random() {
      static bool initialized = false;
      if (!initialized) {
        std::srand(std::time(nullptr));
        initialized = true;
      }
    }
};
```

notes

math.tex

Equations

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

The extremum is given by $x = -b/2a$.

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

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where A'_i is A with the i'th column replaced by b.

Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

Trigonometry

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

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

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

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

Geometry

Triangles

Side lengths: a, b, c

Semiperimeter: $p = \frac{a + b + c}{2}$

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

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two): $s_a = \sqrt{bc \left[1 - \left(\frac{a}{b + c} \right)^2 \right]}$

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

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

Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

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

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac+bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

Spherical coordinates

$$
\begin{array}{ll}
x = r\sin\theta\cos\phi & r = \sqrt{x^2+y^2+z^2} \\
y = r\sin\theta\sin\phi & \theta = \mathrm{acos}(z/\sqrt{x^2+y^2+z^2}) \\
z = r\cos\theta & \phi = \mathrm{atan2}(y,x)
\end{array}
$$

Derivatives/Integrals

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

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

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

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

Sums

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

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

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

Series

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

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

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, \, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, \, (-\infty < x < \infty)$$

Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

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

For independent X and Y,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

Discrete distributions

0.134.1 Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n, p)$, $n = 1, 2, \ldots, 0 \leq p \leq 1$.

$$p(k) = \binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu = np, \, \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n, p)$ is approximately $\mathrm{Po}(np)$ for small p.

0.134.2 First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, \, k = 1, 2, \ldots$$

$$\mu = \frac{1}{p}, \, \sigma^2 = \frac{1-p}{p^2}$$

0.134.3 Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, \, k = 0, 1, 2, \ldots$$

$$\mu = \lambda, \, \sigma^2 = \lambda$$

Continuous distributions

0.135.1 Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \, \sigma^2 = \frac{(b-a)^2}{12}$$

0.135.2 Exponential distribution

The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \, \sigma^2 = \frac{1}{\lambda^2}$$

0.135.3 Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

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

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

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

Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n\mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k \to \infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik}p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki}t_k$.