University of Copenhagen

# 3 little 3 late

Asbjørn Lind, Elias Rasmussen Lolck, Thor Vejen Eriksen

NWERC 2024

November 22, 2024

# Setup

## INFO.tex

### How to submit/debug

Remember:

- Fast input.
- Unsure of time limit? Generate simple max cases!
- Check memory limits.
- Check overflow!
- Turbo mega check the right file gets submitted.
- Compile (and run test cases) at least once with `dc`; strongly consider resolving warnings.
- Overflow?
- Make sure you are reading e.g. $n$ and $m$ in the right order.
- Do not have uninitialized variables!
- If WA/RE: print code. Take a quick walk. Maybe even rewrite everything. RE can mean MLE. Invalidated pointers/iterators?

### During test session

- `setxkbmap dk/us`.
- That bashrc/vimrc works.
- Printing.
- Sending clarification.
- `cppreference`.
- CLI submission if it exists.
- Whitespace sensitivity in submissions.
- Return non-zero from main.
- Printing to `stderr` during otherwise correct submission.
- Source code size limit (if not stated by jury).
- Get MLE and check if it shows as RE.
- Check compile time limit.
- `__int128`.
- Check available binaries (yoinked from kactl): `echo $PATH | tr ':' ' ' | xargs ls | grep -v / | sort | uniq | tr ' n' ' '`

## bashrc.sh

4b42a4

```
64a45f setxkbmap -option caps:escape
64a45f # fast:
92e6e6 xset r rate 200 120
92e6e6 # normal:
efd146 xset r rate 500 35
efd146 # debug compile (C++):
3b0f04 dc() {
53a367   bsnm=$(basename "$1" .cpp)
53a367   # EUC uses -std=gnu++20
8f434d   command="g++ ${bsnm}.cpp -o $bsnm -Wshadow -Wall -g -
         fsanitize=address,undefined -D_GLIBCXX_DEBUG -std=gnu
         ++20 -Wfatal-errors"
98f084   echo $command
26aae8   $command
1b7e88 }
4b42a4 set -o vi
```

## 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
```

## KACTL template

bd2055

```
d41d8c // in addition to template.h, kactl uses:
7ab427 #define rep(i,a,b) for(int i = a; i < (b); ++i)
793c8d #define sz(x) (int)(x).size()
c1f7c4 typedef pair<int,int> pii;
bd2055 typedef vector<int> vi;
```

## template

d74cfd

```
d41d8c // #include <bits/stdc++.h>
ca417d using namespace std;
fbb4e1 typedef long long ll;
22323a #define all(x) (x).begin(), (x).end()
22323a
251bca int main() {
04007e   ios::sync_with_stdio(0); cin.tie(0);
d74cfd }
```

## vimrc

88d5be

```
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
5c5f32 vnoremap <silent> <A-Down> :m '>+1<CR>gv=gv
7c854e vnoremap <silent> <A-Up> :m '<-2<CR>gv=gv
88d5be vnoremap <silent> p "_dP
```

# Data_structures

## Disjoint Set Union

**Description**: Classic DSU using path compression and union by rank. `unite` returns true iff `u` and `v` were disjoint.
**Usage**: `Dsu d(n); d.unite(a, b); d.find(a);`
**Complexity**: `find()`, `unite()` are amortized $\mathcal{O}(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. Basically $\mathcal{O}(1)$.

5168ab

```
e9a6d7 struct Dsu {
20b72f   vector <int> p, rank;
b8d9ca   Dsu(int n) {
743598     p.resize(n); rank.resize(n, 0);
53d18c     iota(p.begin(), p.end(), 0);
b27a44   }
aef80c   int find(int x) {
f5ffdc     return p[x] == x ? x : p[x] = find(p[x]);
a1cfa1   }
fecab9   bool unite(int u, int v) {
675018     if ((u = find(u)) == (v = find(v))) return false;
de3de6     if (rank[u] < rank[v]) swap(u, v);
859abb     p[v] = u;
0e6393     rank[u] += rank[u] == rank[v];
49561c     return true;
052f6f   }
5168ab };
```

## 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); lct.query(x, 0, n);`
**Complexity**: $\mathcal{O}(\log n)$ per insertion/query

ba9fe6

```
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)]
f584d6   LCT(int size) { v.resize(size + 1, LINF); }
d9141d   void insert(Line line, int l, int r) {
eaba83     if (l > r) return;
fb3606     int mid = (l + r) >> 1;
fa90d9     if (line.f(mid) < v[mid].f(mid)) swap(line, v[mid]);
9e933f     if (line.f(l) < v[mid].f(l)) insert(line, l, mid -
         1);
17cdcd     else insert(line, mid + 1, r);
3dae75   }
995afc   Line query(int x, int l, int r) {
d3e628     if (l > r) return LINF;
3e2cdb     int mid = (l + r) >> 1;
5b4a90     if (x == mid) return v[mid]; // faster on avg. - not
          necessary
9c3466     if (x < mid) return best_of(v[mid], query(x, l, mid
         - 1), x);
0065c4     return best_of(v[mid], query(x, mid + 1, r), x);
9d9761   }
c445dc   Line best_of(Line a, Line b, ll x) { return a.f(x) < b
         .f(x) ? a : b; }
ba9fe6 };
```

## Rollback Union Find

**Description**: Yoinked from 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**: $\mathcal{O}(\log n)$.

de4ad0

```
47a5e9 struct RollbackUF {
32cc46   vi e; vector<pii> st;
66f6eb   RollbackUF(int n) : e(n, -1) {}
dfd9e1   int size(int x) { return -e[find(x)]; }
f73c5d   int find(int x) { return e[x] < 0 ? x : find(e[x]); }
821d77   int time() { return sz(st); }
154abb   void rollback(int t) {
d4a702     for (int i = time(); i --> t;)
96b4b9       e[st[i].first] = st[i].second;
93333b     st.resize(t);
e7fe82   }
3f4ca5   bool join(int a, int b) {
9dd20b     a = find(a), b = find(b);
081d43     if (a == b) return false;
40458e     if (e[a] > e[b]) swap(a, b);
3aaa7c     st.push_back({a, e[a]});
5f71eb     st.push_back({b, e[b]});
fa6967     e[a] += e[b]; e[b] = a;
21e12e     return true;
f0724e    }
de4ad0 };
```

## 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 };
```

## Fast hash map

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

c7be5a

```
d41d8c // #include <bits/extc++.h>
d41d8c
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>;
```

## 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
```

```
319c4cOuter root(new Inner(0, 0, MX_RC), 0, MX_RC);
1b943bvoid update(int r, int c, long long k) {
b78d10  root.update(r, c, k);
a445e8}
07e107long 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}
```

## Lazy segment tree

**Description**: Zero-indexed, bounds are [l, r), operations can be modified. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
**Usage**: `Lazy_segtree seg(n); seg.update(l, r, val); seg.query(l, r);`
**Complexity**: $\mathcal{O}(\log n)$ per update/query

69cb07

```
186ef1struct Lazy_segtree {
142517  typedef ll T; // change type here
7bd302  typedef ll LAZY_T; // change type here
b372d3  static constexpr T unit = 0; // change unit here
962f5f  static constexpr LAZY_T lazy_unit = 0; // change lazy unit here
7f83a4  T f(T l, T r) { return l + r; } // change operation here
c9312a  void push(int now, int l, int r) {
b8a6a8    if (w[now] == lazy_unit) return;
2624b4    v[now] += w[now] * (r - l); // operation can be modified
baf9bc    if (r - l - 1)
a7681c      w[now * 2 + 1].first += w[now],
693215      w[now * 2 + 2].first += w[now];
bcea3b    w[now] = lazy_unit;
6dfa7c  }
b801ca  int size;
c396cf  vector <T> v;
ce40fb  vector <LAZY_T> w;
37a544  Lazy_segtree(int s = 0) : size(s ? 1 << (32 - __builtin_clz(s)) : 0), v(size << 1, unit), w(size << 1, lazy_unit) { }
de9785  template <typename U> void update(int l, int r, U val) { update(l, r, val, 0, 0, size); }
677cbe  T query(int l, int r) { return query(l, r, 0, 0, size); }
9f1fbb  template <typename U> void update(int tl, int tr, U val, int now, int l, int r) {
a8f911    push(now, l, r);
0733ac    if (l >= tr || r <= tl) return;
7fba5d    if (l >= tl && r <= tr) {
7fba5d      // this does not *have* to accumulate, push is called before this:
95311e      w[now] += val; // operation can be modified
0745ca      push(now, l, r);
a90e6f      return;
92ee41    }
d9ba9d    int mid = (l + r) >> 1;
ef2051    update(tl, tr, val, now * 2 + 1, l, mid);
e67970    update(tl, tr, val, now * 2 + 2, mid, r);
d8795a    v[now] = f(v[now * 2 + 1], v[now * 2 + 2]);
162f13  }
99a91a  T query(int tl, int tr, int now, int l, int r) {
eef6b9    push(now, l, r);
15115d    if (l >= tr || r <= tl) return unit;
a2b812    if (l >= tl && r <= tr) return v[now];
a2df56    int mid = (l + r) >> 1;
2c75e7    return f(query(tl, tr, now * 2 + 1, l, mid), query(tl, tr, now * 2 + 2, mid, r));
293acc  }
5468d9  template <typename U> void build(const vector <U>& a) {
111070    for (int i = 0; i < (int) a.size(); i++) v[size - 1 + i] = a[i]; // operation can be modified
24e497    for (int i = size - 2; i >= 0; i--) v[i] = f(v[i * 2 + 1], v[i * 2 + 2]);
db93fb  }
69cb07};
```

## Matrix

**Description**: Yoinked from kactl. Basic operations on square matrices.
**Usage**: `Matrix<int, 3> A; A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}}; vector<int> vec = {1,2,3}; vec = (AN̂) * vec;`
**Complexity**: $\mathcal{O}(n^3)$ per multiplication, $\mathcal{O}(n^3 \log p)$ per exponentiation.

c43c7d

```
a9546ftemplate<class T, int N> struct Matrix {
3ef34d  typedef Matrix M;
e08add  array<array<T, N>, N> d{};
c6c78f  M operator*(const M& m) const {
1aac3d    M a;
5aa3ab    rep(i,0,N) rep(j,0,N)
b57a4b      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
5a335a    return a;
ae016f  }
db76e7  vector<T> operator*(const vector<T>& vec) const {
8c6915    vector<T> ret(N);
c690c4    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
85dfd1    return ret;
2f2bd4  }
d7ecae  M operator^(ll p) const {
8c6dab    assert(p >= 0);
75b035    M a, b(*this);
31b02f    rep(i,0,N) a.d[i][i] = 1;
6e9149    while (p) {
56e68d      if (p&1) a = a*b;
74866d      b = b*b;
12d234      p >>= 1;
1c0b4f    }
3182fe    return a;
c85f9b  }
c43c7d};
```

## Ordered Map

**Description**: `extc++.h` order statistics tree. `find_by_order` returns an iterator to the $k$th element (0-indexed), `order_of_key` returns the index of the element (0-indexed), i.e. the number of elements less than the argument.
**Usage**: `ordered_set <int> s; s.insert(1); s.insert(2); *s.find_by_order(0) = 3; s.erase(3); s.order_of_key(2);`
**Complexity**: Everything is $\mathcal{O}(\log n)$.

93d061

```
d41d8c// #include <bits/extc++.h>
d41d8c// if judge does not have extc++.h, use:
d41d8c// #include <ext/pb_ds/assoc_container.hpp>
d41d8c// #include <ext/pb_ds/tree_policy.hpp>
0d73c5using namespace __gnu_pbds;
647225template <typename T> using ordered_set = tree <T,
    null_type, less <T>, rb_tree_tag,
    tree_order_statistics_node_update>;
d8888dtemplate <typename T, typename U> using ordered_map =
    tree <T, U, less <T>, rb_tree_tag,
    tree_order_statistics_node_update>;
d8888d
d8888d// yeet from kactl:
b3a888void example() {
ce24bb  ordered_set <int> t, t2; t.insert(8);
26f820  auto it = t.insert(10).first;
4f4555  assert(it == t.lower_bound(9));
aa0d3a  assert(t.order_of_key(10) == 1);
de8f5b  assert(t.order_of_key(11) == 2);
080225  assert(*t.find_by_order(0) == 8);
86be9f  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
93d061}
```

## Persistent segment tree

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

3237d5

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

## Segment tree

**Description**: Zero-indexed, bounds are [l, r), operations can be modified. $\mathcal{O}(\log n)$ find_first and the like can be implemented by checking bounds, then checking left tree, then right tree, recursively.
**Usage**: `Segtree seg(n); seg.update(ind, val); seg.query(l, r);`
**Complexity**: $\mathcal{O}(\log n)$ per update/query.

```cpp
struct Segtree {
  typedef ll T; // change type here
  static constexpr T unit = 0; // change unit here
  T f(T l, T r) { return l + r; } // change operation here
  int size;
  vector <T> v;
  Segtree(int s = 0) : size(s ? 1 << (32 - __builtin_clz(s)) : 0), v(size << 1, unit) { }
  void update(int ind, T val) { update(ind, val, 0, 0, size); }
  T query(int l, int r) { return query(l, r, 0, 0, size); }
  void update(int ind, T val, int now, int l, int r) {
    if (!(r - l - 1)) { v[now] = val; return; } // operation can be modified
    int mid = (l + r) >> 1;
    if (ind < mid) update(ind, val, now * 2 + 1, l, mid);
    else update(ind, val, now * 2 + 2, mid, r);
    v[now] = f(v[now * 2 + 1], v[now * 2 + 2]);
  }
  T query(int tl, int tr, int now, int l, int r) {
    if (l >= tr || r <= tl) return unit;
    if (l >= tl && r <= tr) return v[now];
    int mid = (l + r) >> 1;
    return f(query(tl, tr, now * 2 + 1, l, mid), query(tl, tr, now * 2 + 2, mid, r));
  }
  template <typename U> void build(const vector <U>& a) {
    for (int i = 0; i < (int) a.size(); i++) v[size - 1 + i] = a[i]; // operation can be modified
    for (int i = size - 2; i >= 0; i--) v[i] = f(v[i * 2 + 1], v[i * 2 + 2]);
  }
};
```

## Sparse table

**Description**: Yoinked from kactl. Classic sparse table, implemented with range minimum queries, can be modified.
**Usage**: `Sparse s(vec); s.query(a, b);`
**Complexity**: $\mathcal{O}(\mid V \mid \log \mid V \mid +Q)$.

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

## Treap

**Description**: Yoinked from 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**: $\mathcal{O}(\log n)$ operations.

```cpp
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto pa = split(n->l, k);
    n->l = pa.second;
    n->recalc();
    return {pa.first, n};
  } else {
    auto pa = split(n->r, k - cnt(n->l) - 1); // and just "k"
    n->r = pa.first;
    n->recalc();
    return {n, pa.second};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    l->recalc();
    return l;
  } else {
    r->l = merge(l, r->l);
    r->recalc();
    return r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto pa = split(t, pos);
  return merge(merge(pa.first, n), pa.second);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

## 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

```cpp
struct wavelet_tree{
  #define vi vector<int>
  #define pb push_back
  int lo, hi;
  wavelet_tree *l, *r;
  vi b;

  //nos are in range [x,y]
  //array indices are [from, to)
  wavelet_tree(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){
      return x <= mid;
    };
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++)
      b.pb(b.back() + f(*it));
    //see how lambda function is used here
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid+1, hi);
  }

  //kth smallest element in [l, r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1];
    int lb = b[l-1]; //amt of nos in first (l-1) nos that go in left
    int rb = b[r]; //amt of nos in first (r) nos that go in left
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }

  //count of nos in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }

  //count of nos in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
  ~wavelet_tree(){
    delete l;
    delete r;
  }
};
```

# 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;
56ac78     }
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
d41d8c // #include "Point.h"
d41d8c // #include "On_segment.h"
d41d8c // #include "Segment_distance.h"
d41d8c
7dc51e template<class P>
8cfa07 bool inPolygon(vector<P> &p, P a, bool strict = true) {
68a46b   int cnt = 0, n = sz(p);
49a14b   rep(i,0,n) {
1c161f     P q = p[(i + 1) % n];
ca77bc     if (onSegment(p[i], q, a)) return !strict;
ca77bc     //or: if (segDist(p[i], q, a) <= eps) return !strict
          ;
8d185a     cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q)
           > 0;
ae1a12   }
3f2423   return cnt;
2bf504 }
```

## KD-tree

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

```
                                                    bac5b0
d41d8c // #include "Point.h"
d41d8c
9a6170 typedef long long T;
d3d771 typedef Point<T> P;
3b6fe3 const T INF = numeric_limits<T>::max();
3b6fe3
632da2 bool on_x(const P& a, const P& b) { return a.x < b.x; }
624f75 bool on_y(const P& a, const P& b) { return a.y < b.y; }
624f75
319cda struct Node {
7cd9b0   P pt; // if this is a leaf, the single point in it
1149c5   T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
3f2a96   Node *first = 0, *second = 0;
3f2a96
edbce8   T distance(const P& p) { // min squared distance to a
          point
71ed74     T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
6963e4     T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
4a1b67     return (P(x,y) - p).dist2();
1460d4   }
1460d4
3f46ab   Node(vector<P>&& vp) : pt(vp[0]) {
```

```
ae3536    for (P p : vp) {
516c49      x0 = min(x0, p.x); x1 = max(x1, p.x);
28bf16      y0 = min(y0, p.y); y1 = max(y1, p.y);
2e9c2c    }
a1b63f    if (vp.size() > 1) {
a1b63f      // split on x if width >= height (not ideal...)
172b91      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
172b91      // divide by taking half the array for each child (not
172b91      // best performance with many duplicates in the middle)
21b567      int half = sz(vp)/2;
2f742c      first = new Node({vp.begin(), vp.begin() + half});
a66d3b      second = new Node({vp.begin() + half, vp.end()});
470fcd    }
0265cf  }
6fda19};
6fda19
ce4e50struct KDTree {
eee062  Node* root;
677e4a  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}
677e4a
7daf7f  pair<T, P> search(Node *node, const P& p) {
23e6bd    if (!node->first) {
23e6bd      // uncomment if we should not find the point itself:
23e6bd      // if (p == node->pt) return {INF, P()};
df1914      return make_pair((p - node->pt).dist2(), node->pt);
19dc67    }
19dc67
f3c18d    Node *f = node->first, *s = node->second;
c51266    T bfirst = f->distance(p), bsec = s->distance(p);
5cf03e    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
5cf03e
5cf03e    // search closest side first, other side if needed
fa9faa    auto best = search(f, p);
b7e192    if (bsec < best.first)
18c5d3      best = min(best, search(s, p));
891524    return best;
3771f7  }
3771f7
3771f7  // find nearest point to a point, and its squared distance
3771f7  // (requires an arbitrary operator< for Point)
5c5074  pair<T, P> nearest(const P& p) {
961132    return search(root, p);
60e74e  }
bac5b0};
```

## Line hull intersection

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

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

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

7cf45b

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

## Line line intersection

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

a01f81

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

## Line projection and reflection

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

b5562d

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

## Linear transformation

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

03a306

```
d41d8c// #include "Point.h"
d41d8c
6269ectypedef Point<double> P;
a0133aP linearTransformation(const P& p0, const P& p1,
f9bd62    const P& q0, const P& q1, const P& r) {
16967b  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
d52dff  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
03a306}
```

## Manhatten MST

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

df6f59

```
d41d8c// #include "Point.h"
d41d8c
bbe58ctypedef Point<int> P;
10752cvector<array<int, 3>> manhattanMST(vector<P> ps) {
82bb37  vi id(sz(ps));
129d92  iota(all(id), 0);
bded47  vector<array<int, 3>> edges;
4634f8  rep(k,0,4) {
55be09    sort(all(id), [&](int i, int j) {
f00400         return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
0a2d30    map<int, int> sweep;
6ada5f    for (int i : id) {
2327aa      for (auto it = sweep.lower_bound(-ps[i].y);
7348ca                it != sweep.end(); sweep.erase(it++)) {
931774        int j = it->second;
5297c6        P d = ps[i] - ps[j];
874f9c        if (d.y > d.x) break;
5f471a        edges.push_back({d.y + d.x, i, j});
28e949      }
5f0d0f      sweep[-ps[i].y] = i;
9ea743    }
9c2fdc    for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
666542  }
af3f66  return edges;
df6f59}
```

## Minimum enclosing circle

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

09dd0a

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

## Is on segment

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

c597e8

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

## 2D Point

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

47ec0a

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

## 3D Point

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

8058ae

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

## Is point in convex polygon

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

71446b

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

## Polygon area

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

f12300

```
d41d8c // #include "Point.h"
d41d8c
4fce64 template<class T>
df7c3f T polygonArea2(vector<Point<T>>& v) {
ab8862   T a = v.back().cross(v[0]);
0711d6   rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
b195d0   return a;
f12300 }
```

## Polygon center of mass

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

9706dc

```
d41d8c // #include "Point.h"
d41d8c
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     }
97ae86     sort(all(segs));
4e8cac     for (auto& s : segs) s.first = min(max(s.first, 0.0)
         , 1.0);
00b8ae     double sum = 0;
40a9a7     int cnt = segs[0].second;
317ef1     rep(j,1,sz(segs)) {
84ade9       if (!cnt) sum += segs[j].first - segs[j - 1].first
         ;
625398       cnt += segs[j].second;
d3398f     }
0e34c6     ret += A.cross(B) * sum;
6f2b4e   }
52ed80   return ret / 2;
3931c6 }
```

## Polyhedron volume

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

3058c3

```
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 ⇔ 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 ($\phi_1$) and f2 ($\phi_2$) from x axis and zenith angles (latitude) t1 ($\theta_1$) and t2 ($\theta_2$) 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.

f6bf6b

```
d41d8c // #include "Point.h"
d41d8c
7dc51e template<class P>
869862 double lineDist(const P& a, const P& b, const P& p) {
0aca9c   return (double)(b-a).cross(p-a)/(b-a).dist();
f6bf6b }
```

# Graphs

## Articulation points finding

**Description**: Yoinked from CP-algorithms. Standard articulation points finding algorithm.
**Complexity**: $\mathcal{O}(V + E)$.

23f413

```
1a88fd int n; // number of nodes
2e71a7 vector<vector<int>> adj; // adjacency list of graph
2e71a7
553d0a vector<bool> visited;
00663b vector<int> tin, low;
901990 int timer;
901990
d987ec void dfs(int v, int p = -1) {
1a0c45   visited[v] = true;
e286e0   tin[v] = low[v] = timer++;
9157ae   int children=0;
b2fdfc   for (int to : adj[v]) {
ac6371     if (to == p) continue;
efefc6     if (visited[to]) {
bb0d96       low[v] = min(low[v], tin[to]);
2c6d5f     } else {
51b8d7       dfs(to, v);
2be87e       low[v] = min(low[v], low[to]);
bcef3d       if (low[to] >= tin[v] && p!=-1)
fc8020         IS_CUTPOINT(v);
9909e4       ++children;
beb204     }
59a553   }
37508f   if(p == -1 && children > 1)
939ca9     IS_CUTPOINT(v);
a16b47 }
a16b47
4d5f0e void find_cutpoints() {
5828db   timer = 0;
7c3d3d   visited.assign(n, false);
7e557a   tin.assign(n, -1);
3c48fc   low.assign(n, -1);
3c9834   for (int i = 0; i < n; ++i) {
44cd93     if (!visited[i])
006f9c       dfs (i);
aa8a61   }
23f413 }
```

## Bellman-Ford

**Description**: Yoinked from 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 $V^2 \max \mid w_i \mid < \sim 2^{63}$.
**Usage**: `bellmanFord(nodes, edges, s)`.
**Complexity**: $\mathcal{O}(VE)$.

830a8f

```
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;
15a23e   sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s();
         });
96d3f0   int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled
         vertices
9004e9   rep(i,0,lim) for (Ed ed : eds) {
c5c796     Node cur = nodes[ed.a], &dest = nodes[ed.b];
ed7594     if (abs(cur.dist) == inf) continue;
4a6344     ll d = cur.dist + ed.w;
167727     if (d < dest.dist) {
729010       dest.prev = ed.a;
68e296       dest.dist = (i < lim-1 ? d : -inf);
2e4a08     }
cab225   }
824bac   rep(i,0,lim) for (Ed e : eds) {
5e8ff4     if (nodes[e.a].dist == -inf)
6d95a8       nodes[e.b].dist = -inf;
8d0b1c   }
830a8f }
```

## Biconnected components

**Description**: Yoinked from kactl. Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.
**Usage**: `int eid = 0; ed.resize(n); for each edge (a, b) { ed[a].emplace(b, eid); ed[b].emplace(a, eid++); } bicomps([&] (const vi& edgelist) { ... });`
**Complexity**: $\mathcal{O}(E + V)$.

2965e5

```
16a1ed vi num, st;
5c7bd5 vector<vector<pii>> ed;
5c17a1 int Time;
bf2641 template<class F>
3e8eda int dfs(int at, int par, F& f) {
d1b332   int me = num[at] = ++Time, e, y, top = me;
95a358   for (auto pa : ed[at]) if (pa.second != par) {
e55cf3     tie(y, e) = pa;
e45b73     if (num[y]) {
fe0f3e       top = min(top, num[y]);
145ca4       if (num[y] < me)
01b6d5         st.push_back(e);
51d5dc     } else {
8aee96       int si = sz(st);
e478b0       int up = dfs(y, e, f);
4c0c04       top = min(top, up);
fb91dd       if (up == me) {
0aa7e5         st.push_back(e);
10c0ea         f(vi(st.begin() + si, st.end()));
7a2eb7         st.resize(si);
4c59fd       }
e01a87       else if (up < me) st.push_back(e);
47e7b7       else { /* e is a bridge */ }
7a2ccf     }
55ddf3   }
58e3ce   return top;
0b5c9f }
0b5c9f
2617cc template<class F> void bicomps(F f) {
b5c03f   num.assign(sz(ed), 0);
14c211   rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
2965e5 }
```

## Binary lifting with LCA

**Description**: Yoinked from kactl. Finds power of two jumps in a tree - and standard LCA. Assumes the root node points to itself!
**Usage**: `vector<vi> jmps = treeJump(parents); int l = lca(jmps, depth, a, b);`
**Complexity**: $\mathcal{O}(N \log N)$ construction. $\mathcal{O}(\log N)$ per query.

bfce85

```
750796 vector<vi> treeJump(vi& P){
d7f747   int on = 1, d = 1;
4e1485   while(on < sz(P)) on *= 2, d++;
40155b   vector<vi> jmp(d, P);
bcb753   rep(i,1,d) rep(j,0,sz(P))
35de77     jmp[i][j] = jmp[i-1][jmp[i-1][j]];
9cd4c2   return jmp;
6d3434 }
6d3434
d0c552 int jmp(vector<vi>& tbl, int nod, int steps){
68ef34   rep(i,0,sz(tbl))
fa7843     if(steps&(1<<i)) nod = tbl[i][nod];
5f4dea   return nod;
7ce14c }
7ce14c
48e3ef int lca(vector<vi>& tbl, vi& depth, int a, int b) {
dae62d   if (depth[a] < depth[b]) swap(a, b);
afb472   a = jmp(tbl, a, depth[a] - depth[b]);
74edff   if (a == b) return a;
ea1a60   for (int i = sz(tbl); i--;) {
67ff64     int c = tbl[i][a], d = tbl[i][b];
6533fb     if (c != d) a = c, b = d;
863967   }
b796a3   return tbl[0][a];
bfce85 }
```

## Bridge finding

**Description**: Yoinked from CP-algorithms. Standard bridge finding algorithm.
**Complexity**: $\mathcal{O}(V + E)$.

a44485

```
1a88fd int n; // number of nodes
2e71a7 vector<vector<int>> adj; // adjacency list of graph
2e71a7
553d0a vector<bool> visited;
00663b vector<int> tin, low;
901990 int timer;
901990
d987ec void dfs(int v, int p = -1) {
1a0c45   visited[v] = true;
e286e0   tin[v] = low[v] = timer++;
e82afa   for (int to : adj[v]) {
4b3a29     if (to == p) continue;
57a119     if (visited[to]) {
440a56       low[v] = min(low[v], tin[to]);
f87d63     } else {
c6c172       dfs(to, v);
00a71d       low[v] = min(low[v], low[to]);
f54aa9       if (low[to] > tin[v])
08b207         IS_BRIDGE(v, to);
030e02     }
276ef5   }
8768b3 }
8768b3
190845 void find_bridges() {
12b6f3   timer = 0;
bbc736   visited.assign(n, false);
87eddd   tin.assign(n, -1);
864765   low.assign(n, -1);
bc4c5c   for (int i = 0; i < n; ++i) {
00e55b     if (!visited[i])
307b77       dfs(i);
6a780d   }
a44485 }
```

## DFS Bipartite Matching

**Description**: Yoinked from 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**: `vi btoa(m, -1); dfsMatching(g, btoa);`
**Complexity**: $\mathcal{O}(VE)$.

522b98

```
a47cc3 bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
98d83d   if (btoa[j] == -1) return 1;
6aa9ef   vis[j] = 1; int di = btoa[j];
d093f9   for (int e : g[di])
400b9b     if (!vis[e] && find(e, g, btoa, vis)) {
b1c950       btoa[e] = di;
107fe8       return 1;
cc0de1     }
bf43f0   return 0;
d13a81 }
1578f8 int dfsMatching(vector<vi>& g, vi& btoa) {
a6152c   vi vis;
49e964   rep(i,0,sz(g)) {
62eadd     vis.assign(sz(btoa), 0);
0eda2c     for (int j : g[i])
c468b2       if (find(j, g, btoa, vis)) {
407765         btoa[j] = i;
5b1f88         break;
5609e1       }
61061f   }
c95a04   return sz(btoa) - (int)count(all(btoa), -1);
522b98 }
```

## Dinic's Algorithm

**Description**: Yoinked from kactl. Finds the maximum flow from $s$ to $t$ in a directed graph. To obtain the actual flow values, look at all edges with capacity $> 0$ (zero capacity edges are residual edges).
**Usage**: `Dinic dinic(n); dinic.addEdge(a, b, c); dinic.maxFlow(s, t);`
**Complexity**: $\mathcal{O}(VE \log U)$ where $U = \max \mid \text{capacity} \mid$. $\mathcal{O}(\min(\sqrt{E}, V^{2/3})E)$ if $U = 1$; so $\mathcal{O}(\sqrt{V}E)$ for bipartite matching.

d7f0f1

```
14df72 struct Dinic {
9230ca   struct Edge {
ca825e     int to, rev;
eceace     ll c, oc;
299dbe     ll flow() { return max(oc - c, 0LL); } // if you
       need flows
9d5927   };
0aa82d   vi lvl, ptr, q;
31ed82   vector<vector<Edge>> adj;
fdd5b9   Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
4f21d6   void addEdge(int a, int b, ll c, ll rcap = 0) {
3c87cb     adj[a].push_back({b, sz(adj[b]), c, c});
95d74a     adj[b].push_back({a, sz(adj[a]) - 1, rcap, rcap});
a45d7e   }
0e705d   ll dfs(int v, int t, ll f) {
836e00     if (v == t || !f) return f;
2410c4     for (int& i = ptr[v]; i < sz(adj[v]); i++) {
d7080f       Edge& e = adj[v][i];
591b8b       if (lvl[e.to] == lvl[v] + 1)
fad0d4         if (ll p = dfs(e.to, t, min(f, e.c))) {
aedea1           e.c -= p, adj[e.to][e.rev].c += p;
02fe28           return p;
d3bb27         }
f4fbea     }
7da8aa     return 0;
72048c   }
e7b939   ll calc(int s, int t) {
2195f9     ll flow = 0; q[0] = s;
b15633     rep(L,0,31) do { // 'int L=30' maybe faster for
       random data
df0f60       lvl = ptr = vi(sz(q));
5d9371       int qi = 0, qe = lvl[s] = 1;
5702d8       while (qi < qe && !lvl[t]) {
a7da4e         int v = q[qi++];
8c4b36         for (Edge e : adj[v])
3c4dab           if (!lvl[e.to] && e.c >> (30 - L))
0d5640             q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
16dd6b       }
12fc53       while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
f2733d     } while (lvl[t]);
14d62b     return flow;
2b90e4   }
761cc4   bool leftOfMinCut(int a) { return lvl[a] != 0; }
d7f0f1 };
```

## MST in directed graphs

**Description**: Yoinked from kactl. Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
**Usage**: `pair <ll, vi> res = DMST(n, edges, root);`
**Complexity**: $\mathcal{O}(E \log V)$.

39e620

```
d41d8c // #include "../Data_structures/dsu_rollback.h"
d41d8c
030131 struct Edge { int a, b; ll w; };
7519f2 struct Node { /// lazy skew heap node
45a8d0   Edge key;
348382   Node *l, *r;
59f245   ll delta;
958c51   void prop() {
c4174f     key.w += delta;
9353bd     if (l) l->delta += delta;
69a899     if (r) r->delta += delta;
cfc93b     delta = 0;
31f792   }
61e0cf   Edge top() { prop(); return key; }
67708e };
d59b55 Node *merge(Node *a, Node *b) {
6b68b8   if (!a || !b) return a ?: b;
839210   a->prop(), b->prop();
7c5d9a   if (a->key.w > b->key.w) swap(a, b);
c76878   swap(a->l, (a->r = merge(b, a->r)));
046c62   return a;
5e360c }
821d19 void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
821d19
ef4c12 pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
4a59c3   RollbackUF uf(n);
a7352a   vector<Node*> heap(n);
a4c794   for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node
          {e});
5ec2e1   ll res = 0;
9ed102   vi seen(n, -1), path(n), par(n);
92e6e3   seen[r] = r;
c7b0b9   vector<Edge> Q(n), in(n, {-1,-1}), comp;
fc7b25   deque<tuple<int, int, vector<Edge>>> cycs;
360529   rep(s,0,n) {
96b18b     int u = s, qi = 0, w;
fae505     while (seen[u] < 0) {
2158f1       if (!heap[u]) return {-1,{}};
bcb3d2       Edge e = heap[u]->top();
cc1e56       heap[u]->delta -= e.w, pop(heap[u]);
d9d5a2       Q[qi] = e, path[qi++] = u, seen[u] = s;
fcb967       res += e.w, u = uf.find(e.a);
e7ed0a       if (seen[u] == s) { /// found cycle, contract
2e137f         Node* cyc = 0;
5167d4         int end = qi, time = uf.time();
618ecf         do cyc = merge(cyc, heap[w = path[--qi]]);
7cef71         while (uf.join(u, w));
3eb5cd         u = uf.find(u), heap[u] = cyc, seen[u] = -1;
3a9488         cycs.push_front({u, time, {&Q[qi], &Q[end]}});
ea74cd       }
db364e     }
93005f     rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
f2bc30   }
f2bc30
186c68   for (auto& [u,t,comp] : cycs) { // restore sol (
          optional)
55dced     uf.rollback(t);
6dda7b     Edge inEdge = in[u];
a32e6d     for (auto& e : comp) in[uf.find(e.b)] = e;
b092d0     in[uf.find(inEdge.b)] = inEdge;
c5d7d7   }
4f8a9a   rep(i,0,n) par[i] = in[i].a;
d28015   return {res, par};
39e620 }
```

## (D + 1)-edge coloring

**Description**: Yoinked from 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.)
**Usage**: `vi res = edgeColoring(N, eds);`
**Complexity**: $\mathcal{O}(NM)$.

e210e2

```
f41922 vi edgeColoring(int N, vector<pii> eds) {
aa3ad0   vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
04f5f4   for (pii e : eds) ++cc[e.first], ++cc[e.second];
a8572e   int u, v, ncols = *max_element(all(cc)) + 1;
d26648   vector<vi> adj(N, vi(ncols, -1));
fc7443   for (pii e : eds) {
e8084f     tie(u, v) = e;
1235a9     fan[0] = v;
2ddcc8     loc.assign(ncols, 0);
716e30     int at = u, end = u, d, c = free[u], ind = 0, i = 0;
96c76c     while (d = free[v], !loc[d] && (v = adj[u][d]) !=
        -1)
e45383       loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
9115a5     cc[loc[d]] = c;
5a2c0f     for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at
          ][cd])
8a99c9       swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
2827eb     while (adj[fan[i]][d] != -1) {
f3efaf       int left = fan[i], right = fan[++i], e = cc[i];
e98916       adj[u][e] = left;
90bb57       adj[left][e] = u;
4e1b6a       adj[right][e] = -1;
e7082c       free[right] = e;
657a28     }
a781ab     adj[u][d] = fan[i];
2eeb98     adj[fan[i]][d] = u;
783efe     for (int y : {fan[0], u, end})
2b36d8       for (int& z = free[y] = 0; adj[y][z] != -1; z++);
e9f8dc   }
967649   rep(i,0,sz(eds))
0c6ff6     for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret
          [i];
ce6fa1   return ret;
e210e2 }
```

## Edmonds-Karp

**Description**: Yoinked from kactl. Flow algorithm with guaranteed complexity $\mathcal{O}(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.
**Usage**: `edmondsKarp(graph, source, sink);`
**Complexity**: $\mathcal{O}(EV^2)$.

482fe0

```
676711 template<class T> T edmondsKarp(vector<unordered_map<int
         , T>>& graph, int source, int sink) {
dc891c   assert(source != sink);
16aa01   T flow = 0;
324dc1   vi par(sz(graph)), q = par;
324dc1
6b3baa   for (;;) {
b6886e     fill(all(par), -1);
8ed190     par[source] = 0;
f85f7e     int ptr = 1;
968ffa     q[0] = source;
968ffa
481db7     rep(i,0,ptr) {
4dfc15       int x = q[i];
0b66e7       for (auto e : graph[x]) {
47c24f         if (par[e.first] == -1 && e.second > 0) {
edc6f5           par[e.first] = x;
7bf6c2           q[ptr++] = e.first;
3c94b0           if (e.first == sink) goto out;
013016         }
e083c2       }
b22780     }
b14b2c     return flow;
a8b66f out:
f9f5c6     T inc = numeric_limits<T>::max();
ff74aa     for (int y = sink; y != source; y = par[y])
59bbb1       inc = min(inc, graph[par[y]][y]);
59bbb1
b7fadd     flow += inc;
874b49     for (int y = sink; y != source; y = par[y]) {
7342d3       int p = par[y];
39f2f7       if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
63483a       graph[y][p] += inc;
868f7e     }
98a343   }
482fe0 }
```

## Floyd-Warshall

**Description**: Yoinked from 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 $j$, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
**Usage**: `floydWarshall(m);`
**Complexity**: $\mathcal{O}(n^3)$.

531245

```
96441fconst ll inf = 1LL << 62;
433b02void floydWarshall(vector<vector<ll>>& m) {
b0c2bb  int n = sz(m);
2b4646  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
2794c2  rep(k,0,n) rep(i,0,n) rep(j,0,n)
7be85c    if (m[i][k] != inf && m[k][j] != inf) {
46581f      auto newDist = max(m[i][k] + m[k][j], -inf);
9a15b1      m[i][j] = min(m[i][j], newDist);
2682ca    }
7f7b97  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
54f5ea    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf
         ;
531245}
```

## General matching

**Description**: Yoinked from kactl. Matching for general graphs. Finds a maximum subset of edges such that each vertex is incident to at most one edge. Fails with probability $\frac{N}{\text{mod}}$.
**Usage**: `generalMatching(N, ed)`
**Complexity**: $\mathcal{O}(N^3)$.

cb1912

```
d41d8c// #include "../Maths/Matrix_inverse_mod.h"
d41d8c
d41d8c
376046vector<pii> generalMatching(int N, vector<pii>& ed) {
8d1892  vector<vector<ll>> mat(N, vector<ll>(N)), A;
ae1d83  for (pii pa : ed) {
d77802    int a = pa.first, b = pa.second, r = rand() % mod;
19e55d    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
614800  }
614800
5763d0  int r = matInv(A = mat), M = 2*N - r, fi, fj;
acf617  assert(r % 2 == 0);
acf617
9bc254  if (M != N) do {
480c1c    mat.resize(M, vector<ll>(M));
dfa134    rep(i,0,N) {
1593eb      mat[i].resize(M);
ea98c1      rep(j,N,M) {
d8fdfd        int r = rand() % mod;
60f83e        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
36a855      }
be41a1    }
c9966b  } while (matInv(A = mat) != M);
c9966b
50a07b  vi has(M, 1); vector<pii> ret;
17b324  rep(it,0,M/2) {
348eac    rep(i,0,M) if (has[i])
2b3a54      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
4934d8        fi = i; fj = j; goto done;
be61bb    } assert(0); done:
1a9b3a    if (fj < N) ret.emplace_back(fi, fj);
fcefbe    has[fi] = has[fj] = 0;
3d1959    rep(sw,0,2) {
b0634f      ll a = modpow(A[fi][fj], mod-2);
e24316      rep(i,0,M) if (has[i] && A[i][fj]) {
600ed4        ll b = A[i][fj] * a % mod;
d7826c        rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) %
         mod;
9debcf      }
07f84a      swap(fi,fj);
6c623e    }
f16c12  }
cd02c8  return ret;
cb1912}
```

## Global minimum cut

**Description**: Yoinked from kactl. Finds a global minimum cut in an undirected graph, as represented by an adjacency matrix.
**Usage**: `pair<int, vi> res = globalMinCut(mat);`
**Complexity**: $\mathcal{O}(V^3)$.

8b0e19

```
192f1dpair<int, vi> globalMinCut(vector<vi> mat) {
81f955  pair<int, vi> best = {INT_MAX, {}};
a4b19e  int n = sz(mat);
165100  vector<vi> co(n);
f640ab  rep(i,0,n) co[i] = {i};
a62b4e  rep(ph,1,n) {
bfa30c    vi w = mat[0];
6e33f2    size_t s = 0, t = 0;
76cb1b    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio.
         queue
c98135      w[t] = INT_MIN;
2c2cfb      s = t, t = max_element(all(w)) - w.begin();
9d976b      rep(i,0,n) w[i] += mat[t][i];
8c07c9    }
727626    best = min(best, {w[t] - mat[t][t], co[t]});
626622    co[s].insert(co[s].end(), all(co[t]));
d24f0e    rep(i,0,n) mat[s][i] += mat[t][i];
c7b746    rep(i,0,n) mat[i][s] = mat[s][i];
e25d4b    mat[0][t] = INT_MIN;
076888  }
6a68fc  return best;
8b0e19}
```

## Heavy-light decomposition

**Description**: Yoinked from kactl. Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log n$ light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. `VALS_EDGES` being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0. NOTE: below implementation uses kactl lazy segtree, this detail must be modified!
**Usage**: `HLD <false> hld(adj); hld.query_path(u, v); ...`
**Complexity**: $\mathcal{O}(\log n)$ segtree operations per operation.

6f34db

```
d41d8c// #include "...kactl_segtree..."
d41d8c
303396template <bool VALS_EDGES> struct HLD {
931c5a  int N, tim = 0;
878e29  vector<vi> adj;
644a8a  vi par, siz, depth, rt, pos;
6b55a4  Node *tree;
d266b7  HLD(vector<vi> adj_)
9b9a5f    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
         depth(N),
ef2f12      rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0);
         dfsHld(0); }
f1f501  void dfsSz(int v) {
6937fc    if (par[v] != -1) adj[v].erase(find(all(adj[v]), par
         [v]));
c2274a    for (int& u : adj[v]) {
20e816      par[u] = v, depth[u] = depth[v] + 1;
f64490      dfsSz(u);
7b9912      siz[v] += siz[u];
ef818f      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
b0fa49    }
9ba8db  }
b0240c  void dfsHld(int v) {
925ec3    pos[v] = tim++;
6a30a7    for (int u : adj[v]) {
039f8a      rt[u] = (u == adj[v][0] ? rt[v] : u);
2698ee      dfsHld(u);
39b629    }
39d559  }
9dbc9b  template <class B> void process(int u, int v, B op) {
ddfb6b    for (; rt[u] != rt[v]; v = par[rt[v]]) {
f6dd0a      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
87a197      op(pos[rt[v]], pos[v] + 1);
fa17fe    }
f837ff    if (depth[u] > depth[v]) swap(u, v);
3bc5a1    op(pos[u] + VALS_EDGES, pos[v] + 1);
0d5603  }
178671  void modifyPath(int u, int v, int val) {
99a5b1    process(u, v, [&](int l, int r) { tree->add(l, r,
         val); });
79ce98  }
fb7383  int queryPath(int u, int v) { // Modify depending on
         problem
c2f5e0    int res = -1e9;
0e4f0a    process(u, v, [&](int l, int r) {
26c08a        res = max(res, tree->query(l, r));
29a64c    });
e9dec3    return res;
f00cd2  }
4e9b11  int querySubtree(int v) { // modifySubtree is similar
7db27d    return tree->query(pos[v] + VALS_EDGES, pos[v] + siz
         [v]);
8aad63  }
6f34db};
```

## Hopcroft-Karp Bipartite Matching

**Description**: Yoinked from kactl. Fast 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**: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
**Complexity**: $\mathcal{O}(\sqrt{V}E)$.

f612e4

```
b0efcbbool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A,
         vi& B) {
59b291  if (A[a] != L) return 0;
86baa8  A[a] = -1;
77efd6  for (int b : g[a]) if (B[b] == L + 1) {
d9e76d    B[b] = 0;
1a816f    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A,
         B))
47a337      return btoa[b] = a, 1;
84f762  }
4cc63e  return 0;
9e7938}
9e7938
9e641cint hopcroftKarp(vector<vi>& g, vi& btoa) {
7f282c  int res = 0;
252756  vi A(g.size()), B(btoa.size()), cur, next;
a02d20  for (;;) {
df7680    fill(all(A), 0);
591ffa    fill(all(B), 0);
591ffa    /// Find the starting nodes for BFS (i.e. layer 0).
3bf28f    cur.clear();
69a5d0    for (int a : btoa) if(a != -1) A[a] = -1;
0fe82b    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
0fe82b    /// Find all layers using bfs.
cefa37    for (int lay = 1;; lay++) {
93e008      bool islast = 0;
786fae      next.clear();
342697      for (int a : cur) for (int b : g[a]) {
96ecca        if (btoa[b] == -1) {
17e7a8          B[b] = lay;
87b4fe          islast = 1;
4c74fe        }
```

```
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

## Link-cut tree

**Description**: Yoinked from 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.
**Usage**: See comments in code.
**Complexity**: Amortized $\mathcal{O}(\log n)$ per (any) operation.

5909e2

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

struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) { /// Move u to root of represented tree.
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) { /// Move u to root aux tree. Return the root of the root aux tree.
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

## Minimum cost maximum flow (faster)

**Description**: Yoinked from kactl. Does not support negative cost cycles. call **setpi** before **maxflow** if costs can be negative. To obtain the actual flow, look at positive values only.
**Complexity**: $\mathcal{O}(FE\log(V))$ where $F$ is max flow. $\mathcal{O}(VE)$ for **setpi**.

135b73

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

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;

  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) {
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
          else
            q.modify(its[e.to], { -dist[e.to], e.to });
        }
      }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);

      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
    return {totflow, totcost/2};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (edge& e : ed[i]) if (e.cap)
          if ((v = pi[i] + e.cost) < pi[e.to])
            pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

## Maximum clique callbacks

**Description**: Yoinked from 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.
**Usage**: `cliques(eds, callback, ...);`
**Complexity**: $\mathcal{O}(3^{n/3})$ - *much* faster for sparse graphs.

b0d5b1

```
753236typedef bitset<128> B;
6454cctemplate<class F>
05d32cvoid 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];
876203  rep(i,0,sz(eds)) if (cands[i]) {
074813    R[i] = 1;
cf4187    cliques(eds, f, P & eds[i], X & eds[i], R);
c889e0    R[i] = P[i] = 0; X[i] = 1;
2b8ca5  }
b0d5b1}
```

## Maximum clique

**Description**: Yoinked from kactl. Finds a maximum clique of a graph given as a symmetric bitset matrix. Can be used to find a maximum independent set by finding a clique of the complement graph.
**Complexity**: About 1 second for $n = 155$, worst case random graphs ($p = .90$). Runs faster for sparse graphs.

f7c0bc

```
54ea03typedef vector<bitset<200>> vb;
913d3dstruct 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;
ccd5a0  vector<vi> C;
b548bf  vi qmax, q, S, old;
f625cf  void init(vv& r) {
4a81cc    for (auto& v : r) v.d = 0;
993a80    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i
        ];
06b9b4    sort(all(r), [](auto a, auto b) { return a.d > b.d;
        });
16d40c    int mxD = r[0].d;
964a7f    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
d5dc84  }
e66dec  void expand(vv& R, int lev = 1) {
ac13ae    S[lev] += S[lev - 1] - old[lev];
8602ba    old[lev] = S[lev - 1];
67e58a    while (sz(R)) {
09eb24      if (sz(q) + R.back().d <= sz(qmax)) return;
20ce0c      q.push_back(R.back().i);
0b52a4      vv T;
b0e686      for(auto v:R) if (e[R.back().i][v.i]) T.push_back
        ({v.i});
e23129      if (sz(T)) {
c706bf        if (S[lev]++ / ++pk < limit) init(T);
86a266        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) +
         1, 1);
fb8d45        C[1].clear(), C[2].clear();
abd788        for (auto v : T) {
d6bf0a          int k = 1;
3e1b8e          auto f = [&](int i) { return e[v.i][i]; };
6fcc14          while (any_of(all(C[k]), f)) k++;
30a122          if (k > mxk) mxk = k, C[mxk + 1].clear();
f8575a          if (k < mnk) T[j++].i = v.i;
8dee8a          C[k].push_back(v.i);
5ebe7a        }
df11ee        if (j > 0) T[j - 1].d = 0;
bfcc7c        rep(k,mnk,mxk + 1) for (int i : C[k])
b4de6c          T[j].i = i, T[j++].d = k;
e72ba9        expand(T, lev + 1);
86a1f3      } else if (sz(q) > sz(qmax)) qmax = q;
ad6614      q.pop_back(), R.pop_back();
c01dd9    }
901020  }
12c3d2  vi maxClique() { init(V), expand(V); return qmax; }
6c200c  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)),
         old(S) {
64b603    rep(i,0,sz(e)) V.push_back({i});
21f145  }
f7c0bc};
```

## Minimum cost maximum flow (old version)

**Description**: Yoinked from kactl. `cap[i][j] != cap[j][i]` is allowed; double edges are not. 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. Note: duplicate edges and anti-parallel edges are not allowed.
**Complexity**: $\mathcal{O}(E^2)$ o_O.

f0549f

```
d41d8c// #include <bits/extc++.h>
d41d8c
9f43acconst ll INF = numeric_limits<ll>::max() / 4;
e7aa4dtypedef vector<ll> VL;
e7aa4d
7600c3struct MCMF {
70940d  int N;
17badf  vector<vi> ed, red;
180f43  vector<VL> cap, flow, cost;
2da736  vi seen;
0aaea7  VL dist, pi;
8a6a40  vector<pii> par;
8a6a40
c625a3  MCMF(int N) :
40dcd7    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(
         cap),
0ff3f1    seen(N), dist(N), pi(N), par(N) {}
0ff3f1
0d5aaf  void addEdge(int from, int to, ll cap, ll cost) {
dfe9fd    this->cap[from][to] = cap;
2746cf    this->cost[from][to] = cost;
0aefe0    ed[from].push_back(to);
2e301a    red[to].push_back(from);
0d7444  }
0d7444
d51da6  void path(int s) {
1bbf83    fill(all(seen), 0);
a47d93    fill(all(dist), INF);
940b6d    dist[s] = 0; ll di;
940b6d
253bf8    __gnu_pbds::priority_queue<pair<ll, int>> q;
7cbcf8    vector<decltype(q)::point_iterator> its(N);
ceeedd    q.push({0, s});
ceeedd
131c47    auto relax = [&](int i, ll cap, ll cost, int dir) {
3e7e39      ll val = di - pi[i] + cost;
e73b04      if (cap && val < dist[i]) {
995e7c        dist[i] = val;
9ffb7f        par[i] = {s, dir};
6f572c        if (its[i] == q.end()) its[i] = q.push({-dist[i
         ], i});
b01cc7        else q.modify(its[i], {-dist[i], i});
78aeb7      }
296072    };
296072
14d0a0    while (!q.empty()) {
8a014e      s = q.top().second; q.pop();
86f88e      seen[s] = 1; di = dist[s] + pi[s];
b99962      for (int i : ed[s]) if (!seen[i])
7a591c        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
a7eccb      for (int i : red[s]) if (!seen[i])
e30ccf        relax(i, flow[i][s], -cost[i][s], 0);
471013    }
dc76f2    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
f09751  }
f09751
56b024  pair<ll, ll> maxflow(int s, int t) {
8167d2    ll totflow = 0, totcost = 0;
eafa93    while (path(s), seen[t]) {
7e7783      ll fl = INF;
c13600      for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
         p)
1deaaf        fl = min(fl, r ? cap[p][x] - flow[p][x] : flow[x
        ][p]);
ce41dc      totflow += fl;
fabd3d      for (int p,r,x = t; tie(p,r) = par[x], x != s; x =
         p)
af2fdd        if (r) flow[p][x] += fl;
d2ac45        else flow[x][p] -= fl;
d4f75d    }
eb8c78    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i
        ][j];
42f3e5    return {totflow, totcost};
74c5e6  }
74c5e6
74c5e6  // If some costs can be negative, call this before
         maxflow:
ad4fa8  void setpi(int s) { // (otherwise, leave this out)
da8610    fill(all(pi), INF); pi[s] = 0;
f5cfc8    int it = N, ch = 1; ll v;
ebc38e    while (ch-- && it--)
544ef3      rep(i,0,N) if (pi[i] != INF)
ab9631        for (int to : ed[i]) if (cap[i][to])
9b7ba1          if ((v = pi[i] + cost[i][to]) < pi[to])
10724d            pi[to] = v, ch = 1;
a95142    assert(it >= 0); // negative cost cycle
e98539  }
f0549f};
```

## Minimum vertex cover

**Description**: Yoinked from kactl. Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.
**Complexity**: Idk, look code.

da4196

```
d41d8c// #include "DFS_matching.h"
d41d8c
0ba9d3vi cover(vector<vi>& g, int n, int m) {
cb5948  vi match(m, -1);
372cb7  int res = dfsMatching(g, match);
b8f9d0  vector<bool> lfound(n, true), seen(m);
60f20a  for (int it : match) if (it != -1) lfound[it] = false;
d5d915  vi q, cover;
3be03d  rep(i,0,n) if (lfound[i]) q.push_back(i);
813047  while (!q.empty()) {
e11082    int i = q.back(); q.pop_back();
19fc1a    lfound[i] = 1;
113bda    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
1aca58      seen[e] = true;
3b97a6      q.push_back(match[e]);
b97b04    }
b9473f  }
570cd5  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
a12f34  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
8edba7  assert(sz(cover) == res);
6300f6  return cover;
da4196}
```

## Strongly connected components

**Description**: Yoinked from 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**: $\mathcal{O}(E+V)$.

76b5c9

```
508bd7 vi val, comp, z, cont;
c218d3 int Time, ncomps;
d31820 template<class G, class F> int dfs(int j, G& g, F& f) {
9b6eaf   int low = val[j] = ++Time, x; z.push_back(j);
ed28ae   for (auto e : g[j]) if (comp[e] < 0)
3cf550     low = min(low, val[e] ?: dfs(e,g,f));
3cf550
903808   if (low == val[j]) {
12fcbe     do {
b81dc2       x = z.back(); z.pop_back();
c3db61       comp[x] = ncomps;
6ddcbd       cont.push_back(x);
cf1bb0     } while (x != j);
122be2     f(cont); cont.clear();
d65942     ncomps++;
6e1ce2   }
862574   return val[j] = low;
ab59d0 }
c745fa template<class G, class F> void scc(G& g, F f) {
8d248c   int n = sz(g);
46ec08   val.assign(n, 0); comp.assign(n, -1);
011e3c   Time = ncomps = 0;
8389e2   rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
76b5c9 }
```

## Topological sort

**Description**: Yoinked from kactl. 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.
**Usage**: `vi res = topoSort(gr);`
**Complexity**: $\mathcal{O}(V+E)$.

66a137

```
01eaf1 vi topoSort(const vector<vi>& gr) {
cd1a35   vi indeg(sz(gr)), ret;
611d40   for (auto& li : gr) for (int x : li) indeg[x]++;
942024   queue<int> q; // use priority_queue for lexic. largest
           ans.
3ae360   rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
779e30   while (!q.empty()) {
1d3d39     int i = q.front(); // top() for priority queue
2a2af3     ret.push_back(i);
9447f3     q.pop();
d83bd2     for (int x : gr[i])
94c522       if (--indeg[x] == 0) q.push(x);
9b0e88   }
435aa0   return ret;
66a137 }
```

## Weighted bipartite matching

**Description**: Yoinked from kactl. Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes `cost[N][M]`, where `cost[i][j] = cost` for `L[i]` to be matched with `R[j]` and returns (min cost, match), where `L[i]` is matched with `R[match[i]]`. Negate costs for max cost. Requires $N \leq M$.
**Complexity**: $\mathcal{O}(N^2M)$.

1e0fe9

```
325ee8 pair<int, vi> hungarian(const vector<vi> &a) {
497519   if (a.empty()) return {0, {}};
ec9978   int n = sz(a) + 1, m = sz(a[0]) + 1;
0c9f93   vi u(n), v(m), p(m), ans(n - 1);
64fc2f   rep(i,1,n) {
9a06cd     p[0] = i;
c3251b     int j0 = 0; // add "dummy" worker 0
3b3e45     vi dist(m, INT_MAX), pre(m, -1);
ced645     vector<bool> done(m + 1);
564738     do { // dijkstra
2c1b77       done[j0] = true;
6773fe       int i0 = p[j0], j1, delta = INT_MAX;
0023e6       rep(j,1,m) if (!done[j]) {
264023         auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
41fd29         if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
f7e9b7         if (dist[j] < delta) delta = dist[j], j1 = j;
31ae76       }
14a4d0       rep(j,0,m) {
7ceba5         if (done[j]) u[p[j]] += delta, v[j] -= delta;
84199f         else dist[j] -= delta;
6cc461       }
b39843       j0 = j1;
45ce9c     } while (p[j0]);
6c3cef     while (j0) { // update alternating path
971e56       int j1 = pre[j0];
632eb8       p[j0] = p[j1], j0 = j1;
26ae9e     }
9e72cf   }
78ec8c   rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
ae202a   return {-v[0], ans}; // min cost
1e0fe9 }
```

## Two SAT

**Description**: Yoinked from kactl. Solves 2-SAT.
**Usage**: `TwoSat ts(n)` where $n$ is the number of variables. `ts.either(i,j)` means that either $i$ or $j$ must be true. `ts.setValue(i)` means that $i$ must be true. `ts.atMostOne(l)` means that at most one of the variables in $l$ can be true. `ts.solve()` returns true iff it is solvable. `ts.values` will contain one possible solution. Negated variables are represented by bit-inversions (~x).
**Complexity**: $\mathcal{O}(N+E)$ where $N$ is the number of variables and $E$ is the number of clauses.

5f9706

```
d9d94e struct TwoSat {
257c73   int N;
a0af70   vector<vi> gr;
7c0806   vi values; // 0 = false, 1 = true
c1fbac   TwoSat(int n = 0) : N(n), gr(2*n) {}
e10f30   int addVar() { // (optional)
b4b080     gr.emplace_back();
ca34a5     gr.emplace_back();
0f7e62     return N++;
8e7f67   }
1446f5   void either(int f, int j) {
5e1028     f = max(2*f, -1-2*f);
bc62d9     j = max(2*j, -1-2*j);
7f876f     gr[f].push_back(j^1);
511183     gr[j].push_back(f^1);
f602cc   }
cbc333   void setValue(int x) { either(x, x); }
69157f   void atMostOne(const vi& li) { // (optional)
7d932b     if (sz(li) <= 1) return;
7721c4     int cur = ~li[0];
66f796     rep(i,2,sz(li)) {
28a590       int next = addVar();
3de60b       either(cur, ~li[i]);
8bda68       either(cur, next);
001557       either(~li[i], next);
f470ff       cur = ~next;
7cdc2a     }
f21674     either(cur, ~li[1]);
06911d   }
594dbb   vi val, comp, z; int time = 0;
92303b   int dfs(int i) {
fa1d30     int low = val[i] = ++time, x; z.push_back(i);
c93f40     for(int e : gr[i]) if (!comp[e])
c634a9       low = min(low, val[e] ?: dfs(e));
a0ccd1     if (low == val[i]) do {
cf7006       x = z.back(); z.pop_back();
2c346c       comp[x] = low;
a8f0bd       if (values[x>>1] == -1)
fb7b0d         values[x>>1] = x&1;
5a0145     } while (x != i);
3fe09e     return val[i] = low;
088d97   }
12670e   bool solve() {
07c73a     values.assign(N, -1);
a75f85     val.assign(2*N, 0); comp = val;
27da39     rep(i,0,2*N) if (!comp[i]) dfs(i);
a77564     rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
95beae     return 1;
4fdfc4   }
5f9706 };
```

# Maths

## Chinese remainder theorem

**Description**: Yoinked from kactl. `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 $0 \leq x < \mathrm{lcm}(m,n)$. Assumes $mn < 2^{62}$.
**Complexity**: $\mathcal{O}(\log n)$.

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**: Yoinked from kactl. Given $N$ and a real number $x \geq 0$, finds the closest rational approximation $p/q$ with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$. 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 $\infty$; if $x$ is the root of a degree 2 polynomial then $a$'s eventually become cyclic.
**Complexity**: $\mathcal{O}(\log n)$.

dd6c5e

```
0705cd typedef double d; // for N ~ 1e7; long double for N ~ 1
         e9
d72231 pair<ll, ll> approximate(d x, ll N) {
709975   ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y
           = x;
63f648   for (;;) {
32bc0f     ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q :
         inf),
82cd25         a = (ll)floor(y), b = min(a, lim),
12b990         NP = b*P + LP, NQ = b*Q + LQ;
426849     if (a > b) {
426849       // If b > a/2, we have a semi-convergent that
         gives us a
426849       // better approximation; if b = a/2, we *may* have
          one.
```

```
      // Return {P, Q} here for a more canonical approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

## Determinant

**Description**: Yoinked from kactl. Calculates determinant of a matrix. Destroys the matrix.
**Complexity**: $\mathcal{O}(N^3)$.

bd5cec

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

## Divisor Count

**Description**: Counts number of divisors

2ff470

```
ll divisor_cnt(ll n) {
    ll cnt = 1;
    map<ll, ll> factors = factorize(n);
    for (auto p : factors) cnt *= p.second+1;
    return cnt;
}
```

## Sieve of Eratosthenes

**Description**: Yoinked from kactl. Prime sieve for generating all primes up to a certain limit. `isprime[i]` is true iff $i$ is a prime.
**Complexity**: $lim = 100'000'000 \approx 0.8$ s. Runs 30% faster if only odd indices are stored.

7c144c

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vi eratosthenesSieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  rep(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

## Euclid

**Description**: Yoinked from kactl. Finds two integers $x$ and $y$, such that $ax + by = \gcd(a, b)$. If $a$ and $b$ are coprime, then $x$ is the inverse of $a \pmod b$.
**Complexity**: $\mathcal{O}(\log n)$.

33ba8f

```
ll euclid(ll a, ll b, ll &x, ll &y) {
  if (!b) return x = 1, y = 0, a;
  ll d = euclid(b, a % b, y, x);
  return y -= a/b * x, d;
}
```

## Fast fourier transform

**Description**: Yoinked from kactl. `fft(a)` computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all $k$. $N$ must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice $10^{16}$; higher for random inputs). Otherwise, use NTT/FFTMod.
**Complexity**: $\mathcal{O}(n \log n)$ with $N = |A| + |B|$. ($\sim 1s$ for $N = 2^{22}$)

3dd197

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      // C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)  /// include-line
      auto x = (double *)&rt[j+k], y = (double *)&a[i+j+k];        /// exclude-line
      C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);           /// exclude-line
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

## Fast fourier transform under arbitrary MOD

**Description**: Yoinked from kactl. Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice $10^{16}$ or higher). Inputs must be in $[0, \text{mod})$.
**Complexity**: $\mathcal{O}(n \log n)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT).

b82773

```
// #include "FFT.h"

typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

## Fast sieve of Eratosthenes

**Description**: Yoinked from kactl. Prime sieve for generating all primes smaller than LIM.
**Complexity**: LIM= $1e9 \approx 1.5$s. Utalizes cache locality.

6b2912

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM / 2;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
    rep(i,0,min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

## Gauss-Jordan elimination

**Description**: Yoinked from CP-algorithms. The description is taken from CP-algorithms as well: Following is an implementation of Gauss-Jordan. Choosing the pivot row is done with heuristic: choosing maximum value in the current column. The input to the function **`gauss`** is the system matrix $a$. The last column of this matrix is vector $b$. The function returns the number of solutions of the system $(0, 1, \text{or } \infty)$. If at least one solution exists, then it is returned in the vector *ans*. Implementation notes:

- The function uses two pointers - the current column *col* and the current row *row*.
- For each variable $x_i$, the value *where*$(i)$ is the line where this column is not zero. This vector is needed because some variables can be independent.
- In this implementation, the current $i$ th line is not divided by $a_{ii}$ as described above, so in the end the matrix is not identity matrix (though apparently dividing the $i$ th line can help reducing errors).
- After finding a solution, it is inserted back into the matrix - to check whether the system has at least one solution or not. If the test solution is successful, then the function returns 1 or inf, depending on whether there is at least one independent variable.

kactl also has code for solving linear systems somewhere in the document, if needed.
**Complexity**: $\mathcal{O}(\min(n, m) \cdot nm)$ – I.e. cubic.

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be
    infinity or a big number

int gauss (vector < vector<double> > a, vector<double> &
     ans) {
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;

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

    for (int i=0; i<n; ++i)
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign (m, 0);
  for (int i=0; i<m; ++i)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i=0; i<n; ++i) {
    double sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (abs (sum - a[i][m]) > EPS)
      return 0;
  }

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

## Integer determinant

**Description**: Yoinked from kactl. Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.
**Complexity**: $\mathcal{O}(n^3)$.

```
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
  int n = sz(a); ll ans = 1;
  rep(i,0,n) {
    rep(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) rep(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

## Integration

**Description**: Yoinked from 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.
**Complexity**: $\mathcal{O}(n)$ evaluations of $f$.

```
template<class F> double quad(double a, double b, F f,
     const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

## Linear Recurrences

**Description**: Having a linear recurrence of the form $f(n) = a_1 \cdot f(n-1) + a_2 \cdot f(n-2) \cdots$ can be solved in log time with matrix exponentation.

```
#define Matrix vector<vector<ll>>
const ll m = 1000000007;
Matrix operator*(const Matrix& a, const Matrix& b) {
    Matrix c = Matrix(len(a), vector<ll>(len(b[0])));
    for (int i = 0; i < len(a); i++) {
        for (int j = 0; j < len(b[0]); j++) {
            for (int k = 0; k < len(b); k++) {
                c[i][j] += a[i][k]*b[k][j]%m;
                c[i][j] %= m;
            }
        }
    }
    return c;
}
// DOES THIS WORK? Why dp needed?
Matrix fast_exp(const Matrix& a, ll b, map<ll, Matrix>&
     dp) {
    if (dp.count(b)) return dp[b];
    if (b == 1) return a;
    if (b%2) return dp[b] = fast_exp(a, b/2, dp)*
     fast_exp(a, b/2, dp)*a;
    return dp[b] = fast_exp(a, b/2, dp)*fast_exp(a, b/2,
      dp);
}
Matrix operator^(const Matrix& a, ll b) {
    map<ll, Matrix> dp;
    return fast_exp(a, b, dp);
}
void linear_recurrence() {
    /*
        dp[j] += dp[i] * X[i][j] <-- genral case
    */
}
```

## Matrix inverse

**Description**: Yoinked from 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**: $\mathcal{O}(n^3)$.

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c])
   ;
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  /// forget A at this point, just eliminate tmp
   backward
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

## Matrix inverse mod prime

**Description**: Yoinked from kactl. 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**: $\mathcal{O}(n^3)$.

```
// #include "Mod_pow.h"

int matInv(vector<vector<ll>>& A) {
  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]);
994d92     rep(j,0,n) 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)
43e42e     A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0
           ? mod : 0);
d429b2   return n;
a6f68f }
```

## Millar-Rabin primality test

**Description**: Yoinked from kactl. Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$.
**Complexity**: 7 times the complexity of $a^b \mod c$.

60dcd1

```
d41d8c // #include "Mod_mul_LL.h"
d41d8c
da49ed bool isPrime(ull n) {
6e0366   if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
ad415b   ull A[] = {2, 325, 9375, 28178, 450775, 9780504,
          1795265022},
13b9b1       s = __builtin_ctzll(n-1), d = n >> s;
60a421   for (ull a : A) {   // ^ count trailing zeroes
29e314     ull p = modpow(a%n, d, n), i = s;
4ab836     while (p != 1 && p != n - 1 && a % n && i--)
f7944d       p = modmul(p, p, n);
56ff8c     if (p != n-1 && i != s) return 0;
1fad05   }
3c0060   return 1;
60dcd1 }
```

## Modular inverses

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

b4a981

```
d41d8c // const ll mod = 1000000007, LIM = 200000; ///include-
          line
66d058 ll* inv = new ll[LIM] - 1; inv[1] = 1;
b4a981 rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] %
          mod;
```

## Modulo multiplication for 64-bit integers

**Description**: Yoinked from kactl. Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$. This runs 2x faster than the naive `(__int128_t)a * b % M`.
**Complexity**: $\mathcal{O}(1)$ for `modmul`, $\mathcal{O}(\log b)$ for `modpow`.

bbbd8f

```
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 }
```

## Mod pow

**Description**: Yoinked from kactl. What u think mans. (this interface is used by a few other things, hence included in the document)
**Complexity**: $\mathcal{O}(\log e)$.

b83e45

```
e2e0e3 const ll mod = 1000000007; // faster if const
e2e0e3
cceb35 ll modpow(ll b, ll e) {
cd37e8   ll ans = 1;
8bc5f9   for (; e; b = b * b % mod, e /= 2)
c96cd7     if (e & 1) ans = ans * b % mod;
a23ec3   return ans;
b83e45 }
```

## Modular arithmetic

**Description**: Yoinked from kactl. Simple operators for modular arithmetic. You need to set `mod` to some number first and then you can use the structure.

35bfea

```
d41d8c // #include "Euclid.h"
d41d8c
4eb587 const ll mod = 17; // change to something else
6655aa struct Mod {
e58fcd   ll x;
316e8f   Mod(ll xx) : x(xx) {}
9af9c9   Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
88d537   Mod operator-(Mod b) { return Mod((x - b.x + mod) %
          mod); }
622079   Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
bac0da   Mod operator/(Mod b) { return *this * invert(b); }
727966   Mod invert(Mod a) {
ef23f5     ll x, y, g = euclid(a.x, mod, x, y);
8ac2fc     assert(g == 1); return Mod((x + mod) % mod);
f65d71   }
f9c260   Mod operator^(ll e) {
cc1619     if (!e) return Mod(1);
9708c3     Mod r = *this ^ (e / 2); r = r * r;
1f093c     return e&1 ? *this * r : r;
f16184   }
35bfea };
```

## Number theoretic transform

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

ced03d

```
d41d8c // #include "Mod_pow.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 {};
cb0e4e   int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
         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);
1c4346   rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod *
          inv % mod;
4af30c   ntt(out);
70c6bc   return {out.begin(), out.begin() + s};
ced03d }
```

## Polynomial root finding

**Description**: Yoinked from kactl. Finds the real roots to a polynomial.
**Usage**: `polyRoots({{2,-3,1}}, -1e9, 1e9); // solve x^2-3x+2 = 0`
**Complexity**: $\mathcal{O}(n^2 \log(\frac{1}{\epsilon}))$.

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 thing

**Description**: Yoinked from kactl. Some poly things I guess.

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 };
```

## SOS DP

**Description**: Some solution from some problem Elias solved. For each of $n$ elements $x$: The number of elements $y$ such that $x \mid y = x$. The number of elements $y$ such that $x \& y = x$. The number of elements $y$ such that $x \& y \neq 0$. NOTE: if TLE issues, try loop unrolling or C style arrays.
**Complexity**: $\mathcal{O}(V \log V + n)$ where $V$ is the maximum value.

29cc3d

```
ac9985 constexpr const int lgmxV = 20;
e1ff58 constexpr const int mxV = 1 << lgmxV;
e1ff58
28d892 int main(){
d90c3e     ios_base::sync_with_stdio(0); cin.tie(0); cout.tie
       (0);
c1cd68     int n; cin >> n;
5d1c88     vector<int> v(n);
b9f5bb     for(auto &x : v)cin >> x;
924113     vector<vector<int>> sos1(mxV, vector<int> (lgmxV +
       1, 0));
657ed7     vector<vector<int>> sos2(mxV, vector<int> (lgmxV +
       1, 0));
2d7593     for(int i = 0; i < n; ++i){
22d226         sos1[v[i]][0]++;
a24c4a         sos2[v[i] ^ (mxV - 1)][0]++;
932fe0     }
5de047     for(int i = 0; i < mxV; ++i){
b88e0d         for(int j = 0; j < lgmxV; ++j){
6965f4             sos1[i][j + 1] = sos1[i][j];
55b56f             sos2[i][j + 1] = sos2[i][j];
73e5b1             if(i & (1 << j)) { sos1[i][j + 1] += sos1[i
       - (1 << j)][j]; };
cf7af2             if(i & (1 << j)) { sos2[i][j + 1] += sos2[i
       - (1 << j)][j]; };
54565b         }
2735ac     }
61582a     for(int i = 0; i < n; ++i){
b94f88         cout << sos1[v[i]][lgmxV] << ' ' << sos2[v[i] ^
       (mxV - 1)][lgmxV] << ' ' << n - sos1[v[i] ^ (mxV - 1)
       ][lgmxV] << '\n';
f1eea6     }
29cc3d }
```

## Simplex

**Description**: Yoinked from 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**: $\mathcal{O}(NM \cdot \#pivots)$, 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) :
09ecbe     m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
a00ca8       rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
eab15d       rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] =
         b[i];}
03bb56       rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
4c20cd       N[n] = -1; D[m+1][n] = 1;
dcadf8     }
dcadf8
d2dadf   void pivot(int r, int s) {
72cb06     T *a = D[r].data(), inv = 1 / a[s];
93b9bd     rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
a86c76       T *b = D[i].data(), inv2 = b[s] * inv;
c1f31d       rep(j,0,n+2) b[j] -= a[j] * inv2;
ee22d8       b[s] = a[s] * inv2;
df792b     }
d3cb55     rep(j,0,n+2) if (j != s) D[r][j] *= inv;
9e2376     rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
6bf9c5     D[r][s] = inv;
b3404b     swap(B[r], N[s]);
193de8   }
193de8
ede257   bool simplex(int phase) {
f695c2     int x = m + phase - 1;
0aa9db     for (;;) {
8b65cd       int s = -1;
96f50e       rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
e72781       if (D[x][s] >= -eps) return true;
fcd18c       int r = -1;
a7d0e5       rep(i,0,m) {
f65882         if (D[i][s] <= eps) continue;
01fd61         if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
8af3f7                      < MP(D[r][n+1] / D[r][s], B[r])) r
         = i;
170720       }
23b7a6       if (r == -1) return false;
100fe3       pivot(r, s);
d81c2f     }
62b7d3   }
62b7d3
48ae53   T solve(vd &x) {
b0718e     int r = 0;
cc8cd8     rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
dc34d7     if (D[r][n+1] < -eps) {
fbfb80       pivot(r, n);
09ceea       if (!simplex(2) || D[m+1][n+1] < -eps) return -inf
     ;
6b2bed       rep(i,0,m) if (B[i] == -1) {
9aa881         int s = 0;
db9144         rep(j,1,n+1) ltj(D[i]);
d11ba5         pivot(i, s);
213eb8       }
36d5c1     }
e286bf     bool ok = simplex(1); x = vd(n);
002972     rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
8dddea     return ok ? D[m][n+1] : inf;
bc3870   }
aa8530 };
```

## Solve linear equations

**Description**: Yoinked from 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**: $\mathcal{O}(n^2 m)$.

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 equations extended

**Description**: Yoinked from kactl. To get all uniquely determined values of $x$ back from SolveLinear, make the following changes:

08e495

```
d41d8c // #include "Solve_linear.h"
d41d8c
f9498c rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
```

```
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

## Phi function

**Description**: Yoinked from kactl. *Euler's* $\phi$ function is defined as $\phi(n) := \#$ of positive integers $\leq n$ that are coprime with $n$. $\phi(1) = 1$, $p$ prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, $m, n$ coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} \dots (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$. $\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$ **Euler's thm**: $a, n$ coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
**Fermat's little thm**: $p$ prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$.

cf7d6d

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

# Misc

## (FROM CHAT-GPT) Lazy Treap

**Description**: Yoinked from CHAT-GPT. Treap supporting entire tree updates/queries + split/merge.
**Complexity**: $\mathcal{O}(\log n)$ or better on average.

079e94

```
namespace gpt {

  struct Node {
    int val;              // Value of the node
    int priority;         // Priority for treap balancing
    int size;             // Size of the subtree rooted at this node
    int subtree_min;      // Minimum value in the subtree
    int lazy_add;         // Lazy propagation value
    std::shared_ptr <Node> left;         // Left child
    std::shared_ptr <Node> right;        // Right child

    Node(int _val) : val(_val), priority(std::rand()), size(1), subtree_min(_val),
              lazy_add(0), left(nullptr), right(nullptr) {}
  };

  int Size(std::shared_ptr <Node> node) {
    return node ? node->size : 0;
  }

  int SubtreeMin(std::shared_ptr <Node> node) {
    return node ? node->subtree_min : std::numeric_limits<int>::max();
  }

  void Propagate(std::shared_ptr <Node> node) {
    if (node && node->lazy_add != 0) {
      node->val += node->lazy_add;

      if (node->left) {
        node->left->lazy_add += node->lazy_add;
        node->left->subtree_min += node->lazy_add;
      }
      if (node->right) {
        node->right->lazy_add += node->lazy_add;
        node->right->subtree_min += node->lazy_add;
      }

      node->lazy_add = 0;

      // Update subtree_min after propagation
      node->subtree_min = std::min(node->val, std::min(SubtreeMin(node->left), SubtreeMin(node->right)));
    }
  }

  void Update(std::shared_ptr <Node> node) {
    if (node) {
      node->size = 1 + Size(node->left) + Size(node->right);
      node->subtree_min = std::min(node->val, std::min(SubtreeMin(node->left), SubtreeMin(node->right)));
    }
  }

  void Split(std::shared_ptr <Node> node, int index, std::shared_ptr <Node>& left, std::shared_ptr <Node>& right) {
    if (!node) {
      left = right = nullptr;
      return;
    }
    Propagate(node);
    int curr_index = Size(node->left);
    if (index <= curr_index) {
      Split(node->left, index, left, node->left);
      right = node;
    } else {
      Split(node->right, index - curr_index - 1, node->right, right);
      left = node;
    }
    Update(node);
  }

  std::shared_ptr <Node> Merge(std::shared_ptr <Node> left, std::shared_ptr <Node> right) {
    Propagate(left);
    Propagate(right);
    if (!left || !right) {
      return left ? left : right;
    }
    if (left->priority > right->priority) {
      left->right = Merge(left->right, right);
      Update(left);
      return left;
    } else {
      right->left = Merge(left, right->left);
      Update(right);
      return right;
    }
  }

  void AddValue(std::shared_ptr <Node> node, int value) {
    if (!node) return;
    node->lazy_add += value;
    node->subtree_min += value;
  }

  int GetMin(std::shared_ptr <Node> node) {
    if (!node) return std::numeric_limits<int>::max();
    Propagate(node);
    return node->subtree_min;
  }

  int GetMinIndex(std::shared_ptr <Node> node, int offset = 0) {
    if (!node) return -1;
    Propagate(node);
    int min_val = node->subtree_min;

    if (node->val == min_val) {
      return offset + Size(node->left);
    } else if (SubtreeMin(node->left) == min_val) {
      return GetMinIndex(node->left, offset);
    } else {
      return GetMinIndex(node->right, offset + Size(node->left) + 1);
    }
  }

  int GetValue(std::shared_ptr <Node> node, int index) {
    if (!node) return std::numeric_limits<int>::max(); // Or throw an exception
    Propagate(node);
    int curr_index = Size(node->left);
    if (index < curr_index) {
      return GetValue(node->left, index);
    } else if (index == curr_index) {
      return node->val;
    } else {
      return GetValue(node->right, index - curr_index - 1);
    }
  }

  void SetValue(std::shared_ptr <Node> node, int index, int value) {
    if (!node) return;  // Or throw an exception
    Propagate(node);
    int curr_index = Size(node->left);
    if (index < curr_index) {
      SetValue(node->left, index, value);
    } else if (index == curr_index) {
      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:
```

```
7225eb
92f289      Tree() : m_data(nullptr) {
3b8c66        this->m_init_random();
cf0a25      }
99910e      Tree(const std::vector <int>& data) {
03cb7c        this->m_init_random();
564ef7        data.empty() ? this->m_data = nullptr :
5f7420        this->m_data = Build(data.data(), 0, data.size() -
      1);
280fff      }
280fff
47a4cf      Tree split(int at) {
5391b1        assert(this->m_data && 0 <= at && at <= Size(this
    ->m_data));
ad4d40        if (Size(this->m_data) == 0) {
ce6b3e          return Tree();
2bb1a6        }
8d403f        Tree result;
7016ae        std::shared_ptr <Node> left, right;
651518        Split(this->m_data, at, left, right);
46a5cc        this->m_data = left;
27f751        result.m_data = right;
2d4bac        return result;
50f92c      }
50f92c
79d288      void merge(Tree& rhs) {
06f135        this->m_data = Merge(this->m_data, rhs.m_data);
2e5a63        rhs.m_data = nullptr;
1d1ae0      }
1d1ae0
c7731a      void add_value(int value) {
eb2f98        if (this->m_data) {
e1a0a7          AddValue(this->m_data, value);
fa6857        }
9e4289      }
9e4289
02b8a3      int get_min_index() {
412710        assert(this->m_data);
c7cf2a        return GetMinIndex(this->m_data);
8d3f4a      }
8d3f4a
575847      int at(int index) {
34381e        assert(this->m_data && 0 <= index && index < Size(
    this->m_data));
13ff1b        return GetValue(this->m_data, index);
9d5b4c      }
9d5b4c
d224a4      void set(int index, int value) {
ee69b5        assert(this->m_data && 0 <= index && index < Size(
    this->m_data));
c5549c        SetValue(this->m_data, index, value);
02eb63      }
02eb63
45a7ba      int size() {
e98290        return Size(this->m_data);
eec66f      }
eec66f
8cc9f9    private:
8cc9f9
c699fb      std::shared_ptr <Node> m_data;
c699fb
63db02    private:
63db02
8c081a      void m_init_random() {
944111        static bool initialized = false;
d35700        if (!initialized) {
8fb079          std::srand(std::time(nullptr));
0fb68e          initialized = true;
478ca4        }
a8512c      }
a8512c
b35a10    };
b35a10
079e94  }
```

# Strings

## Aho-Corasick automaton

**Description**: Yoinked from kactl. Used for multiple pattern matching. Initialize with `AhoCorasick ac(patterns);` the automaton start node will be at index 0. `find(word)` returns for each position the index of the longest word that ends there, or -1 if none. `findAll(-, word)` finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.
**Complexity**: $26 \cdot \mathcal{O}(N)$ to construct, where $N$ = sum of length of patterns. `find(x)` is $\mathcal{O}(N)$, where $N$ = length of $x$. `findAll` is $\mathcal{O}(NM)$.

f35677

```
51f3fc  struct AhoCorasick {
ba2f89    enum {alpha = 26, first = 'A'}; // change this!
b513ea    struct Node {
b513ea      // (nmatches is optional)
724017      int back, next[alpha], start = -1, end = -1,
    nmatches = 0;
c328c4      Node(int v) { memset(next, v, sizeof(next)); }
9e9dd8    };
99b2f6    vector<Node> N;
b8ee95    vi backp;
abd02c    void insert(string& s, int j) {
77af37      assert(!s.empty());
77757e      int n = 0;
503315      for (char c : s) {
77af36        int& m = N[n].next[c - first];
eebd80        if (m == -1) { n = m = sz(N); N.emplace_back(-1);
    }
d6cab2        else n = m;
4fcd60      }
66f79f      if (N[n].end == -1) N[n].start = j;
695d21      backp.push_back(N[n].end);
c0e36e      N[n].end = j;
702d53      N[n].nmatches++;
10d352    }
bc03d5    AhoCorasick(vector<string>& pat) : N(1, -1) {
32aea4      rep(i,0,sz(pat)) insert(pat[i], i);
514ae6      N[0].back = sz(N);
704bf0      N.emplace_back(0);
704bf0
8b68c2      queue<int> q;
d647a6      for (q.push(0); !q.empty(); q.pop()) {
e8cf90        int n = q.front(), prev = N[n].back;
eab923        rep(i,0,alpha) {
644642          int &ed = N[n].next[i], y = N[prev].next[i];
fe3435          if (ed == -1) ed = y;
040a99          else {
0ea4fc            N[ed].back = y;
1c31f5            (N[ed].end == -1 ? N[ed].end : backp[N[ed].
    start])
912740              = N[y].end;
3ab361            N[ed].nmatches += N[y].nmatches;
021cd1            q.push(ed);
4d5a32          }
ddd9e1        }
22d688      }
7258f5    }
0341b5    vi find(string word) {
8fa219      int n = 0;
af991b      vi res; // ll count = 0;
7a9eea      for (char c : word) {
a99d9f        n = N[n].next[c - first];
8d0b0d        res.push_back(N[n].end);
8d0b0d        // count += N[n].nmatches;
a23882      }
85de21      return res;
33c731    }
32ec5f    vector<vi> findAll(vector<string>& pat, string word) {
801c1a      vi r = find(word);
7ddfe9      vector<vi> res(sz(word));
9106bf      rep(i,0,sz(word)) {
0f304f        int ind = r[i];
0ec3da        while (ind != -1) {
0795af          res[i - sz(pat[ind]) + 1].push_back(ind);
328439          ind = backp[ind];
2f76e2        }
6756ad      }
cbc7fa      return res;
e9b14e    }
f35677  };
```

## String hashing

**Description**: Yoinked from kactl. Self-explanatory methods for string hashing.

2d2a67

```
d41d8c  // Arithmetic mod 2^64-1. 2x slower than mod 2^64 and
      more
d41d8c  // code, but works on evil test data (e.g. Thue-Morse,
      where
d41d8c  // ABBA... and BAAB... of length 2^10 hash the same mod
      2^64).
d41d8c  // "typedef ull H;" instead if you think test data is
      random,
d41d8c  // or work mod 10^9+7 if the Birthday paradox is not a
      problem.
41d24d  typedef uint64_t ull;
95307e  struct H {
80cf70    ull x; H(ull x=0) : x(x) {}
1f9d48    H operator+(H o) { return x + o.x + (x + o.x < x); }
98ccfa    H operator-(H o) { return *this + ~o.x; }
df4ab4    H operator*(H o) { auto m = (__uint128_t)x * o.x;
4eff44      return H((ull)m) + (ull)(m >> 64); }
f17b1d    ull get() const { return x + !~x; }
f8f361    bool operator==(H o) const { return get() == o.get();
      }
442de3    bool operator<(H o) const { return get() < o.get(); }
40d284  };
27469d  static const H C = (ll)1e11+3; // (order ~ 3e9; random
      also ok)
27469d
ff1702  struct HashInterval {
5588fd    vector<H> ha, pw;
76de09    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
ed388c      pw[0] = 1;
77ec24      rep(i,0,sz(str))
aadaa5        ha[i+1] = ha[i] * C + str[i],
7ccac8        pw[i+1] = pw[i] * C;
47478c    }
90ccf6    H hashInterval(int a, int b) { // hash [a, b)
143f12      return ha[b] - ha[a] * pw[b - a];
c82f9f    }
91d0ec  };
91d0ec
690e88  vector<H> getHashes(string& str, int length) {
6b6485    if (sz(str) < length) return {};
355ef9    H h = 0, pw = 1;
22cbb3    rep(i,0,length)
a4ed43      h = h * C + str[i], pw = pw * C;
707e0f    vector<H> ret = {h};
e1bef5    rep(i,length,sz(str)) {
54d254      ret.push_back(h = h * C + str[i] - pw * str[i-length
      ]);
0f079f   }
4311cc   return ret;
2f26bc }
2f26bc
2d2a67 H hashString(string& s){H h{}; for(char c:s) h=h*C+c;
         return h;}
```

## Knuth-Morris-Pratt algorithm

**Description**: Yoinked from kactl. Finds all occurrences of a pattern in a string. `p[x]` computes the length of the longest prefix of `s` that ends at `x`, other than `s[0...x]` itself (`abacaba -> 0010123`).
**Complexity**: $\mathcal{O}(n)$.

d4375c

```
132da4 vi pi(const string& s) {
adaa84   vi p(sz(s));
049209   rep(i,1,sz(s)) {
c043e7     int g = p[i-1];
f6d6b9     while (g && s[i] != s[g]) g = p[g-1];
21a657     p[i] = g + (s[i] == s[g]);
6c1f11   }
63e9df   return p;
9cb7fc }
9cb7fc
7c0957 vi match(const string& s, const string& pat) {
58166d   vi p = pi(pat + '\0' + s), res;
608c16   rep(i,sz(p)-sz(s),sz(p))
68390b     if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
c66a2a   return res;
d4375c }
```

## Manacher

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

e7ad79

```
fc122b array<vi, 2> manacher(const string& s) {
f03a96   int n = sz(s);
daf4bc   array<vi,2> p = {vi(n+1), vi(n)};
4d112b   rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
510161     int t = r-i+!z;
2a504d     if (i<r) p[z][i] = min(t, p[z][l+t]);
3613b8     int L = i-p[z][i], R = i+p[z][i]-!z;
508df3     while (L>=1 && R+1<n && s[L-1] == s[R+1])
c79056       p[z][i]++, L--, R++;
168507     if (R>r) l=L, r=R;
21a1fb   }
4ae824   return p;
e7ad79 }
```

## Min rotation

**Description**: Yoinked from kactl. Finds the lexicographically smallest rotation of a string.
**Usage**: `rotate(v.begin(), v.begin() + minRotation(v), v.end());`
**Complexity**: $\mathcal{O}(n)$.

d07a42

```
5fa8d6 int minRotation(string s) {
e7cf68   int a=0, N=sz(s); s += s;
5ca080   rep(b,0,N) rep(k,0,N) {
f656d5     if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
          break;}
20f912     if (s[a+k] > s[b+k]) { a = b; break; }
b2e25e   }
5fafdc   return a;
d07a42 }
```

## 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 array

**Description**: Yoinked from kactl. Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is $i$'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any zero bytes.
**Complexity**: $\mathcal{O}(n \log n)$ per update/query

38db9f

```
3f48c2 struct SuffixArray {
1ff472   vi sa, lcp;
e88c75   SuffixArray(string& s, int lim=256) { // or
          basic_string<int>
91bda5     int n = sz(s) + 1, k = 0, a, b;
58cf39     vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
74da6a     sa = lcp = y, iota(all(sa), 0);
c642ff     for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim
          = p) {
323b5c       p = j, iota(all(y), n - j);
70faae       rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
066cf5       fill(all(ws), 0);
499169       rep(i,0,n) ws[x[i]]++;
3549fa       rep(i,1,lim) ws[i] += ws[i - 1];
fa1cc1       for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
f9337c       swap(x, y), p = 1, x[sa[0]] = 0;
3a641e       rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
248123         (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
          p++;
f30252     }
0b3927     rep(i,1,n) rank[sa[i]] = i;
271bfc     for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
f2d8ac       for (k && k--, j = sa[rank[i] - 1];
2b582e           s[i + k] == s[j + k]; k++);
22a139   }
38db9f };
```

## 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));
c13476     }
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 }
```

## Suffix tree

**Description**: Yoinked from kactl. Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r) into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r) substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).

**Complexity**: $26 \cdot \mathcal{O}(n)$.

aae0b8

```
3a1cf8 struct SuffixTree {
749ba4   enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
e2aa04   int toi(char c) { return c - 'a'; }
c011c3   string a; // v = cur node, q = cur position
b1f1b1   int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;
b1f1b1
557107   void ukkadd(int i, int c) { suff:
a822f9     if (r[v]<=q) {
f7dbb8       if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
810ece         p[m++]=v; v=s[v]; q=r[v];  goto suff; }
54a4b2       v=t[v][c]; q=l[v];
6b58ee     }
8cb728     if (q==-1 || c==toi(a[q])) q++; else {
141302       l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
694340       p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
d13a03       l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
d6dde8       v=s[p[m]];  q=l[m];
afa4f2       while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v];
          }
8b395d       if (q==r[m])  s[m]=v;  else s[m]=m+2;
3ea06d       q=r[v]-(q-r[m]);  m+=2;  goto suff;
451104     }
0b7995   }
0b7995
d444af   SuffixTree(string a) : a(a) {
88dda6     fill(r,r+N,sz(a));
cb23ac     memset(s, 0, sizeof s);
ab059b     memset(t, -1, sizeof t);
0a991e     fill(t[1],t[1]+ALPHA,0);
3b133d     s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p
          [1] = 0;
372f3e     rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
e6a350   }
e6a350
e6a350   // example: find longest common substring (uses ALPHA
          = 28)
66b1ec   pii best;
66dfb7   int lcs(int node, int i1, int i2, int olen) {
dc2e91     if (l[node] <= i1 && i1 < r[node]) return 1;
bb99fd     if (l[node] <= i2 && i2 < r[node]) return 2;
088265     int mask = 0, len = node ? olen + (r[node] - l[node
          ]) : 0;
3619dd     rep(c,0,ALPHA) if (t[node][c] != -1)
7c5642       mask |= lcs(t[node][c], i1, i2, len);
f72e9f     if (mask == 3)
f77c80       best = max(best, {len, r[node] - len});
2dd106     return mask;
526a4c   }
e2ea9b   static pii LCS(string s, string t) {
a8444c     SuffixTree st(s + (char)('z' + 1) + t + (char)('z' +
          2));
fb62d1     st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
cbb226     return st.best;
9dc48b   }
aae0b8 };
```

## Z function

**Description**: Yoinked from kactl. z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

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

ee09e2

```
44dbe3 vi Z(const string& S) {
cf9608   vi z(sz(S));
661ccc   int l = -1, r = -1;
4fa4a3   rep(i,1,sz(S)) {
fc3afa     z[i] = i >= r ? 0 : min(r - i, z[i - l]);
145eb7     while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
d97d13       z[i]++;
18e9a9     if (i + z[i] > r)
0d1f1b       l = i, r = i + z[i];
b90033   }
cc04e8   return z;
ee09e2 }
```

# Various

## Bump allocator

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

745db2

```
d41d8c // Either globally or in a single class:
2b9528 static char buf[450 << 20];
73a19f void* operator new(size_t s) {
3d5bc2   static size_t i = sizeof buf;
c17d54   assert(s < i);
e69924   return (void*)&buf[i -= s];
0c4c77 }
745db2 void operator delete(void*) {}
```

## Fast integer input

**Description**: Yoinked from kactl. USE THIS IF TRYING TO CONSTANT TIME OPTIMIZE SOLUTION READING IN LOTS OF INTEGERS!!! Read an integer from stdin. Usage requires your program to pipe in input from file.
**Usage**: `./a.out < input.txt`
**Complexity**: About 5x as fast as cin/scanf.

7b3c70

```
c304cb inline char gc() { // like getchar()
b5396f   static char buf[1 << 16];
0c057f   static size_t bc, be;
62a7c2   if (bc >= be) {
c5125f     buf[0] = 0, bc = 0;
bba013     be = fread(buf, 1, sizeof(buf), stdin);
e9a035   }
973215   return buf[bc++]; // returns 0 on EOF
0261eb }
0261eb
b36081 int readInt() {
b8176b   int a, c;
d5554c   while ((a = gc()) < 40);
bc51ee   if (a == '-') return -readInt();
e7b4e7   while ((c = gc()) >= 48) a = a * 10 + c - 480;
5eb5ba   return a - 48;
7b3c70 }
```

## Fast knapsack

**Description**: Yoinked from 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**: $\mathcal{O}(N \max(w_i))$.

b20ccc

```
4d398e int knapsack(vi w, int t) {
cca251   int a = 0, b = 0, x;
ba551c   while (b < sz(w) && a + w[b] <= t) a += w[b++];
3f688f   if (b == sz(w)) return a;
e2b1c9   int m = *max_element(all(w));
11fd10   vi u, v(2*m, -1);
7d8c93   v[a+m-t] = b;
682d61   rep(i,b,sz(w)) {
c83dfe     u = v;
e898de     rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
51a6b1     for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
5e6b65       v[x-w[j]] = max(v[x-w[j]], j);
d2bd39   }
700a1e   for (a = t; v[a+m-t] < 0; a--) ;
e4db33   return a;
b20ccc }
```

## Fast mod reduction

**Description**: Yoinked from 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 $[0, 2b)$. (proven correct)

751a02

```
f4cf5b typedef unsigned long long ull;
a7a66a struct FastMod {
a51f1f   ull b, m;
551bab   FastMod(ull b) : b(b), m(-1ULL / b) {}
010304   ull reduce(ull a) { // a % b + (0 or b)
c7e7c1     return a - (ull)((__uint128_t(m) * a) >> 64) * b;
03d237   }
751a02 };
```

## Interval container

**Description**: Yoinked from kactl. Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are `[inclusive, exclusive)`.
**Complexity**: $\mathcal{O}(\log n)$ per update/query

edce47

```
d91403 set<pii>::iterator addInterval(set<pii>& is, int L, int
         R) {
905a62   if (L == R) return is.end();
117079   auto it = is.lower_bound({L, R}), before = it;
b0184b   while (it != is.end() && it->first <= R) {
ba1bdc     R = max(R, it->second);
a98b04     before = it = is.erase(it);
381108   }
6d817b   if (it != is.begin() && (--it)->second >= L) {
7d7c26     L = min(L, it->first);
d2faed     R = max(R, it->second);
8ea38c     is.erase(it);
5783d8   }
72f28b   return is.insert(before, {L,R});
d57d47 }
d57d47
154403 void removeInterval(set<pii>& is, int L, int R) {
969cd4   if (L == R) return;
f20f53   auto it = addInterval(is, L, R);
51cff5   auto r2 = it->second;
d09c40   if (it->first == L) is.erase(it);
c1de31   else (int&)it->second = L;
b4d977   if (R != r2) is.emplace(R, r2);
edce47 }
```

## Interval cover

**Description**: Yoinked from 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**: $\mathcal{O}(n \log n)$.

9e9d8d

```
4fce64 template<class T>
68ecb6 vi cover(pair<T, T> G, vector<pair<T, T>> I) {
1a8df4   vi S(sz(I)), R;
fa5016   iota(all(S), 0);
0e2216   sort(all(S), [&](int a, int b) { return I[a] < I[b];
          });
a166e4   T cur = G.first;
4d4739   int at = 0;
7cae10   while (cur < G.second) { // (A)
3ef5e6     pair<T, int> mx = make_pair(cur, -1);
bfcc53     while (at < sz(I) && I[S[at]].first <= cur) {
201b40       mx = max(mx, make_pair(I[S[at]].second, S[at]));
9b4b3e       at++;
470978     }
f1e40b     if (mx.second == -1) return {};
206822     cur = mx.first;
93267c     R.push_back(mx.second);
cd0c49   }
c6ad6f   return R;
9e9d8d }
```

## Knuth DP optimization

**Description**: Yoinked from 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 $a \le b \le c \le d$.
**Complexity**: $\mathcal{O}(N^2)$.

210610

```
d41d8c // generic implementation frmo cp-algorithms:
f5cbfa int solve() {
90d984     int N;
c75671     ... // read N and input
f99dd3     int dp[N][N], opt[N][N];
7ed932     auto C = [&](int i, int j) {
163e32         ... // Implement cost function C.
a996a2     };
c67594     for (int i = 0; i < N; i++) {
ea7c0e         opt[i][i] = i;
cd2bbe         ... // Initialize dp[i][i] according to the
       problem
eb6571     }
82a94b     for (int i = N-2; i >= 0; i--) {
6c2b32         for (int j = i+1; j < N; j++) {
1bdeb6             int mn = INT_MAX;
b4c515             int cost = C(i, j);
0b2d26             for (int k = opt[i][j-1]; k <= min(j-1, opt[
       i+1][j]); k++) {
27debe                 if (mn >= dp[i][k] + dp[k+1][j] + cost)
       {
4b284e                     opt[i][j] = k;
e7e3b1                     mn = dp[i][k] + dp[k+1][j] + cost;
3a469f                 }
96a5d4             }
03175a             dp[i][j] = mn;
1659d3         }
ce8b93     }
008b9c     cout << dp[0][N-1] << endl;
210610 }
```

## Longest increasing subsequence

**Description**: Yoinked from kactl. Computes the longest increasing subsequence of a sequence.
**Complexity**: $\mathcal{O}(n \log n)$.

2932a0

```
8d31fc template<class I> vi lis(const vector<I>& S) {
be1376   if (S.empty()) return {};
67f1da   vi prev(sz(S));
c1cccf   typedef pair<I, int> p;
47f7ae   vector<p> res;
5dc126   rep(i,0,sz(S)) {
5dc126     // change 0 -> i for longest non-decreasing
       subsequence
f6ef94     auto it = lower_bound(all(res), p{S[i], 0});
b92110     if (it == res.end()) res.emplace_back(), it = res.
       end()-1;
26a0a3     *it = {S[i], i};
e3bc5b     prev[i] = it == res.begin() ? 0 : (it-1)->second;
f2ee22   }
b33eaf   int L = sz(res), cur = res.back().second;
3f07f2   vi ans(L);
ffa1d5   while (L--) ans[L] = cur, cur = prev[cur];
4593b0   return ans;
2932a0 }
```

## Small ptr

**Description**: Yoinked from kactl. A 32-bit pointer that points into BumpAllocator memory.

2dd6c9

```
d41d8c // #include "Bump_allocator.h"
d41d8c
0cae25 template<class T> struct ptr {
dfb41f   unsigned ind;
953a17   ptr(T* p = 0) : ind(p ? unsigned((char*)p - buf) : 0)
          {
347c02     assert(ind < sizeof buf);
bda3ee   }
36a0d6   T& operator*() const { return *(T*)(buf + ind); }
c82e36   T* operator->() const { return &**this; }
dd2aa9   T& operator[](int a) const { return (&**this)[a]; }
881391   explicit operator bool() const { return ind; }
2dd6c9 };
```

## 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 };
```