



University of Copenhagen

3 little 3 late

Asbjørn Lind, Elias Rasmussen Lolck, Thor Vejen Eriksen

NWERC 2025

November 23, 2025

Setup

hash.sh

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

vimrc

```
-----39eb19
f112b5se ch=1 ic mouse=a sw=4 ts=4 nu rnu nuw=4 nowrap so=6
      iso=8 fdm=indent fdl=99 tm=100
2f1e84Ca Hash w !cpp -dD -P -fpreprocessed \l tr -d '[:space
:]' \l md5sum \l cut -c-6
6ad224Vnoremap <silent> p "-_dP
60b7c4Vnoremap <silent> <A-Down> :m '>+1<CR>gv=gv
39eb19Vnoremap <silent> <A-Up> :m '<-2<CR>gv=gv
```

Combinatorial

Permutation to Int

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

```
-----7016ba
9ab6e7int permToInt(vector<int> v) {
a6407c  int use = 0, i = 0, r = 0;
5878fd  for(int x : v) {
ba160a      r = r * ++i + __builtin_popcount(use & -(1<<x));
27b952      use |= 1 << x;
5d9fcb  }
4a7d46  return r;
7016ba}
```

Multinomial

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

```
-----7f8833
8c3101ll multinomial(vector<int> v) {
93a8d1  ll c = 1, m = v.size() ? v[0] : 1;
5019e7  for (int i = 1; i < (int)v.size(); i++)
fad3cf      for (int j = 0; j < v[i]; j++)
3daa43          c = c * ++m / (j+1);
99415d  return c;
7f8833}
```

Data_structures

Fenwick tree

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

```
-----1743e1
92f63cstruct 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
      return res;
552720  }
1c3977  int lower_bound(ll sum) { // returns first i with
348a7a      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};
```

Li-Chao tree

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

```
-----f60397
4bcbdbstruct Line { ll a, b; ll f(ll x) { return a * x + b; }
      };
7988a9constexpr const Line LINF { 0, 1LL << 60 };
ffb13astruct LCT {
358a49  vector<Line> v; // coord-compression: modify v[x] ->
v[conert(x)]
48d025  LCT(int size) { v.resize(size, LINF); }
8d520c  void insert(Line line, int l, int r) {
effece      if (l > r) return;
a07972      int mid = (l + r) >> 1;
318c53      if (line.f(mid) < v[mid].f(mid)) swap(line, v[mid]);
ec2a0e      if (line.f(l) < v[mid].f(l)) insert(line, l, mid -
1);
665fcd          else insert(line, mid + 1, r);
cba366  }
212b60  Line query(int x, int l, int r) {
8c17fb      if (l > r) return LINF;
1f9b50      int mid = (l + r) >> 1;
3bd038      if (x == mid) return v[mid]; // faster on avg. - not
      necessary
ea215f          if (x < mid) return best_of(v[mid], query(x, l, mid
- 1), x);
e40e21              return best_of(v[mid], query(x, mid + 1, r), x);
70ae78  }
2daa25  Line best_of(Line a, Line b, ll x) { return a.f(x) < b
.f(x) ? a : b; }
f60397};
```

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: O(1) per operation on average.

```
-----c7be5a
d41d8c// #include <bits/extc++.h>
d41d8c
75f3c2struct chash {
0a8969  const uint64_t C = 1l(4e18 * acos(0)) | 71;
```

```
16eb60  ll operator () (ll x) const { return __builtin_bswap64
(x * C); }
cdd37e};
cdd37e
c7be5atemplate <typename KEY_T, typename VAL_T> using hash_map
= __gnu_pbds::gp_hash_table<KEY_T, VAL_T, chash>;
```

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. 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: O(log n) per update/query, O(n) per build

```
-----3237d5
bf28eastruct Node {
24f2c2  Node* l,* r;
1edd16  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);
7dicf8}
7dicf8
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}
```

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(1, r, k); wt.LTE(1, r, k); wt.count(1, r, k);`
Complexity: $\mathcal{O}(\log n)$ per query

364273

```
137e8b struct wavelet_tree{
2f784e     #define vi vector<int>
6a3389     #define pb push_back
bd5515     int lo, hi;
441687     wavelet_tree *l, *r;
d7a498     vi b;

d7a498     //nos are in range [x,y]
d7a498     //array indices are [from, to)
4907d3     wavelet_tree(int *from, int *to, int x, int y){
50c38b         lo = x, hi = y;
15e543         if(lo == hi || from >= to) return;
034eb1         int mid = (lo+hi)/2;
276c4a         auto f = [mid](int x){
4d4ca8             return x <= mid;
dc9b96         };
290aa3         b.reserve(to-from+1);
80c53a         b.pb(0);
55caf2         for(auto it = from; it != to; it++){
9e0a5f             b.pb(b.back() + f(*it));
9e0a5f             //see how lambda function is used here
f87134             auto pivot = stable_partition(from, to, f);
834105             l = new wavelet_tree(from, pivot, lo, mid);
765e4a             r = new wavelet_tree(pivot, to, mid+1, hi);
ee8856         }

ee8856         //kth smallest element in [l, r]
6a485a         int kth(int l, int r, int k){
161294             if(l > r) return 0;
000e05             if(lo == hi) return lo;
515897             int inLeft = b[r] - b[l-1];
1c793f             int lb = b[l-1]; //amt of nos in first (l-1) nos
that gb in left
5207bc             int rb = b[r]; //amt of nos in first (r) nos that go
in left
491f0c             if(k <= inLeft) return this->l->kth(lb+1, rb , k);
ba11bf             return this->r->kth(l-lb, r-rb, k-inLeft);
408cd0         }
408cd0     }

408cd0     //count of nos in [l, r] Less than or equal to k
d6b496     int LTE(int l, int r, int k) {
56eb2f         if(l > r || k < lo) return 0;
5c546e         if(hi <= k) return r - l + 1;
b5a26e         int lb = b[l-1], rb = b[r];
9638eb         return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb
, r-rb, k);
b8e885     }
b8e885     //count of nos in [l, r] equal to k
59067a     int count(int l, int r, int k) {
431d4b         if(l > r || k < lo || k > hi) return 0;
49fc8e         if(lo == hi) return r - l + 1;
1dcf86         int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
6c2de0         if(k <= mid) return this->l->count(lb+1, rb, k);
d7dcf8         return this->r->count(l-lb, r-rb, k);
de1518     }
c5a5e8     ~wavelet_tree(){
400d14         delete l;
80917d         delete r;
98e8a4     }
364273};
```

Strings

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] = (1ll) p[i - 1] * b % MD;
4870cc             ip[i] = (1ll) ip[i - 1] * ib % MD;
66aa32             hs[i] = ((1ll) 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 (1ll) (hs[r] - (l ? hs[l - 1] : 0) + MD) * ip[
l] % MD;
1379de     }
2e25f9};
```

Suffix automaton

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

3d234e

```
10747d struct SA {
31fdad     struct State {
fad143         int length;
7e049f         int link;
ec43e2         int next[26];
209696         int cnt;
0a95ea         bool is_clone;
dafc14         int first_pos;
0fbc43         State(int _length, int _link) :
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;
dadfd1     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),
14383e         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;
106fd4     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 {
ae05d         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         }
98914e         states[q].link = states[cur].link = clone;
0461f9     }
591347     last = cur;
301567 }
d0cce2 bool exists(const std::string& pattern) {
Offabb     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) {
        node = states[node].next[pattern[index] - 'a'];
        index++;
    }
652cc2    return index == (int) pattern.size() ? states[node].
8e968d    first_pos - (int) pattern.size() + 1 : -1;
ef6d88    }
a59113    }
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;
14020f            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    }
4b005c};
4b005c
4b005c// usage example: Repeating Substring submission on cses
.fi
2f5768int 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}
```

Various

Xor basis

Description: Basis of vectors in Z_2^d

61b70d

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