ICPC Notebook

template

hash.sh

```
# 使い方: sh hash.sh -> コピペ -> Ctrl + D
# コメント・空白・改行を削除して md5 でハッシュする
g++ -dD -E -P -fpreprocessed - | tr -d '[:space:]' | md5sum |
cut -c-6
```

settings.sh

```
# CLion の設定
Settings → Build → CMake → Reload CMake Project
add_compile_options(-D_GLIBCXX_DEBUG)
# Caps Lock を Ctrl に変更
setxkbmap -option ctrl:nocaps
```

template.hpp

md5: 015ba5

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const ll INF = LLONG_MAX / 4;
#define all(a) begin(a), end(a)
bool chmin(auto& a, auto b) { return a > b ? a = b, 1 : 0; }
bool chmax(auto& a, auto b) { return a < b ? a = b, 1 : 0; }

int main() {
   cin.tie(0)->sync_with_stdio(0);
   // your code here...
}
```

data-structure

2dBIT.hpp

md5: a0e12e

```cpp
template<typename T> struct BinaryIndexedTree2D {
   int H, W;
   vector<vector<T>> bit;

   BinaryIndexedTree2D(int H, int W) : H(H), W(W), bit(H + 3,
vector<T>(W + 3, 0)) {}
   void add(int x, int y, T w) {
      if(x < 0 || x >= H || y < 0 || y >= W) return;
      for(int a = (++y, ++x); a <= H; a += a & -a) {
         for(int b = y; b <= W; b += b & -b) { bit[a][b] += w;
}
      }
   }

   void imos(int x1, int y1, int x2, int y2, T w) {
      add(x1, y1, w);
      add(x2, y2, w);
      add(x1, y2, -w);
      add(x2, y1, -w);
   }

   T sum(int x, int y) {
      if(x < 0 || x >= H || y < 0 || y >= W) return 0;
      if(x >= H) x = H - 1;
      if(y >= W) y = W - 1;
      T ret = 0;
      for(int a = (++y, ++x); a > 0; a -= a & -a) {
         for(int b = y; b > 0; b -= b & -b) { ret += bit[a][b];
}
      }
      return ret;
   }

   T sum(int x1, int y1, int x2, int y2) {
      return sum(x2, y2) - sum(x1 - 1, y2) - sum(x2, y1 - 1) +
sum(x1 - 1, y1 - 1);
   }
};
```

BIT.hpp

md5: 4dd117

```cpp
struct BIT {
   vector<ll> a;
   BIT(ll n) : a(n + 1) {}
   void add(ll i, ll x) {  // A[i] += x
      i++;
      while(i < size(a)) {
         a[i] += x;
         i += i & -i;
      }
   }
   ll sum(ll r) {
      ll s = 0;
      while(r) {
         s += a[r];
         r -= r & -r;
      }
      return s;
   }
   ll sum(ll l, ll r) {  // sum of A[l, r)
      return sum(r) - sum(l);
   }
};
```

FastSet.hpp

md5: 5c5532

```cpp
// using u64 = uint64_t;
const u64 B = 64;
struct FastSet {
   u64 n;
   vector<vector<u64>> a;
   FastSet(u64 n_) : n(n_) {
      do a.emplace_back(n_ = (n_ + B - 1) / B);
      while(n_ > 1);
   }
   // bool operator[](ll i) const { return a[0][i / B] >> (i %
B) & 1; }
   void set(ll i) {
      for(auto& v : a) {
         v[i / B] |= 1ULL << (i % B);
         i /= B;
      }
   }
   void reset(ll i) {
      for(auto& v : a) {
         v[i / B] &= ~(1ULL << (i % B));
         if(v[i / B]) break;
         i /= B;
      }
   }
   ll next(ll i) {  // i を超える最小の要素
      for(int h = 0; h < size(a); h++) {
         i++;
         if(i / B >= size(a[h])) break;
         u64 d = a[h][i / B] >> (i % B);
         if(d) {
            i += countr_zero(d);
            while(h--) i = i * B + countr_zero(a[h][i]);
            return i;
         }
         i /= B;
      }
      return n;
   }
   ll prev(ll i) {  // i より小さい最大の要素
      for(int h = 0; h < size(a); h++) {
         i--;
         if(i < 0) break;
         u64 d = a[h][i / B] << (~i % B);
         if(d) {
            i -= countl_zero(d);
            while(h--) i = i * B + __lg(a[h][i]);
            return i;
         }
         i /= B;
      }
      return -1;
   }
};
```

binary_trie.hpp

md5: af8046

```cpp
// base: b75bb1
template<typename T, int MAX_LOG = 32> struct BinaryTrie {
   struct Node {
      array<int, 2> next;
      int common;
      T lazy;
      Node() : next{-1, -1}, common(), lazy() {}
   };
   vector<Node> nodes;
   BinaryTrie() { nodes.push_back(Node()); }
   void apply_xor(T val) { nodes[0].lazy ^= val; }
   void push(int cur, int b) {
      if((nodes[cur].lazy >> b) & 1) swap(nodes[cur].next[0],
nodes[cur].next[1]);
      for(int i = 0; i < 2; i++) {
         if(nodes[cur].next[i] == -1)
nodes[nodes[cur].next[i]].lazy ^= nodes[cur].lazy;
      }
      nodes[cur].lazy = 0;
   }
   void add(T val, int cur = 0, int b = MAX_LOG - 1) {
      nodes[cur].common++;
      if(b == -1) return;
      push(cur, b);
      int nxt = (val >> (T)b) & (T)1;
      if(nodes[cur].next[nxt] == -1) {
         nodes[cur].next[nxt] = size(nodes);
         nodes.push_back(Node());
      }
      add(val, nodes[cur].next[nxt], b - 1);
   }

   void erase(T val, int cur = 0, int b = MAX_LOG - 1) {
      nodes[cur].common--;
      if(b == -1) return;
      push(cur, b);
      erase(val, nodes[cur].next[(val >> b) & 1], b - 1);
   }

   T min_element(T val = 0, int cur = 0, int b = MAX_LOG - 1) {
      if(b == -1) return 0;
      push(cur, b);
      T nxt = (val >> b) & 1;
      int ind = nodes[cur].next[nxt];
      if(ind == -1 || nodes[ind].common == 0) nxt ^= 1;
      return min_element(val, nodes[cur].next[nxt], b - 1) |
(nxt << b);
   }  // ddf699

   T max_element(T val = 0, int cur = 0, int b = MAX_LOG - 1) {
return min_element(~val); }  // 5e1a86

   int lower_bound_rank(T val, int cur = 0, int b = MAX_LOG -
1) {
      if(cur == -1 || b == -1) return 0;
      push(cur, b);
      T nxt = (val >> b) & 1;
      int ret = lower_bound_rank(val, nodes[cur].next[nxt], b -
1);
      if(nxt == 1 && nodes[cur].next[0] != -1) { ret +=
nodes[nodes[cur].next[0]].common; }
      return ret;
   }  // 05b14c

   int upper_bound_rank(T val) { return lower_bound_rank(val +
1); }  // 70e301

   T kth_smallest(int k, int cur = 0, int b = MAX_LOG - 1) {
      if(b == -1) return 0;
      int lower_ind = nodes[cur].next[0];
      int lower_cnt = 0;
      if(lower_ind != -1) lower_cnt = nodes[lower_ind].common;
      if(k < lower_cnt) return kth_smallest(k,
nodes[cur].next[0], b - 1);
      return kth_smallest(k - lower_cnt, nodes[cur].next[1], b
- 1) | (T(1) << b);
   }  // b85845

   T kth_largest(int k) { return kth_smallest(nodes[0].common -
```

```
k); }  // 1df41b

   int count(T val) {
      int cur = 0;
      for(int b = MAX_LOG - 1; b >= 0; b--) {
         push(cur, b);
         cur = nodes[cur].next[(val >> b) & 1];
         if(cur == -1) return 0;
      }
      return nodes[cur].common;
   }  // 2a3342

   int count() { return nodes[0].common; }  // 210f0e
};
```

disjoint_sparse_table.hpp

md5: 3df31b

```
template<typename SG> struct disjoint_sparse_table {
   using S = typename SG::S;
   vector<vector<S>> st;
   vector<int> lg;

   disjoint_sparse_table(const vector<S>& v) {
      int b = 0;
      while((1 << b) <= size(v)) b++;
      st.assign(b, vector<S>(size(v)));
      for(int i = 0; i < size(v); i++) st[0][i] = v[i];
      for(int i = 1; i < b; i++) {
         int shift = 1 << i;
         for(int j = 0; j < size(v); j += shift << 1) {
            int t = min(j + shift, (int)size(v));
            st[i][t - 1] = v[t - 1];
            for(int k = t - 2; k >= j; k--) st[i][k] =
SG::op(v[k], st[i][k + 1]);
            if(size(v) <= t) break;
            st[i][t] = v[t];
            for(int k = t + 1; k < min((int)size(v), t +
shift); k++) st[i][k] = SG::op(st[i][k - 1], v[k]);
         }
      }
      lg.resize(1 << b);
      for(int i = 2; i < size(lg); i++) lg[i] = lg[i >> 1] + 1;
   }
   S prod(int l, int r) {
      if(l >= --r) return st[0][l];
      int b = lg[l ^ r];
      return SG::op(st[b][l], st[b][r]);
   }
};
```

dsu.hpp

md5: b55e78

```
// base: d569f4
struct dsu {
   private:
   int _n;
   vector<int> p;

   public:
   dsu() : _n(0) {}
   explicit dsu(int n) : _n(n), p(n, -1) {}

   int merge(int a, int b) {
      // assert(0 <= a && a < _n);
      // assert(0 <= b && b < _n);
      int x = leader(a), y = leader(b);
      if(x == y) return x;
      if(-p[x] < -p[y]) swap(x, y);
      p[x] += p[y];
      p[y] = x;
      return x;
   }

   bool same(int a, int b) {
      // assert(0 <= a && a < _n);
      // assert(0 <= b && b < _n);
      return leader(a) == leader(b);
   }

   int leader(int a) {
      // assert(0 <= a && a < _n);
      if(p[a] < 0) return a;
      int x = a;
      while(p[x] >= 0) x = p[x];
      while(p[a] >= 0) {
         int t = p[a];
         p[a] = x;
         a = t;
      }
      return x;
   }

   int size(int a) {
      // assert(0 <= a && a < _n);
      return -p[leader(a)];
   }  // 818fe7

   vector<vector<int>> groups() {
      vector<int> leader_buf(_n), group_size(_n);
      for(int i = 0; i < _n; i++) {
         leader_buf[i] = leader(i);
         group_size[leader_buf[i]]++;
      }
      vector<vector<int>> result(_n);
      for(int i = 0; i < _n; i++)
result[i].reserve(group_size[i]);
      for(int i = 0; i < _n; i++)
result[leader_buf[i]].push_back(i);
      result.erase(remove_if(result.begin(), result.end(), [&]
(const vector<int>& v) { return v.empty(); }),
                   result.end());
      return result;
   }  // bf3a1e
};
```

lazy_segtree.hpp

md5: c86cef

```
// base: 918715
unsigned int bit_ceil(unsigned int n) {
   unsigned int x = 1;
   while(x < (unsigned int)(n)) x *= 2;
   return x;
}
int countr_zero(unsigned int n) { return __builtin_ctz(n); }
constexpr int countr_zero_constexpr(unsigned int n) {
   int x = 0;
   while(!(n & (1 << x))) x++;
   return x;
}
template<class S, S (*op)(S, S), S (*e)(), class F, S
(*mapping)(F, S), F (*composition)(F, F), F (*id)()>
struct lazy_segtree {
   public:
   lazy_segtree() : lazy_segtree(0) {}
   explicit lazy_segtree(int n) : lazy_segtree(vector<S>(n,
e())) {}
   explicit lazy_segtree(const vector<S>& v) :
_n(int(v.size())) {
      size = (int)bit_ceil((unsigned int)(_n));
      log = countr_zero((unsigned int)size);
      d = vector<S>(2 * size, e());
      lz = vector<F>(size, id());
      for(int i = 0; i < _n; i++) d[size + i] = v[i];
      for(int i = size - 1; i >= 1; i--) { update(i); }
   }
   void set(int p, S x) {
      // assert(0 <= p && p < _n);
      p += size;
      for(int i = log; i >= 1; i--) push(p >> i);
      d[p] = x;
      for(int i = 1; i <= log; i++) update(p >> i);
   }

   S get(int p) {
      // assert(0 <= p && p < _n);
      p += size;
      for(int i = log; i >= 1; i--) push(p >> i);
      return d[p];
   }

   S prod(int l, int r) {
      // assert(0 <= l && l <= r && r <= _n);
      if(l == r) return e();
```

```
    l += size;
    r += size;

    for(int i = log; i >= 1; i--) {
      if(((l >> i) << i) != l) push(l >> i);
      if(((r >> i) << i) != r) push((r - 1) >> i);
    }

    S sml = e(), smr = e();
    while(l < r) {
      if(l & 1) sml = op(sml, d[l++]);
      if(r & 1) smr = op(d[--r], smr);
      l >>= 1;
      r >>= 1;
    }

    return op(sml, smr);
  }

  void apply(int l, int r, F f) {
    assert(0 <= l && l <= r && r <= _n);
    if(l == r) return;

    l += size;
    r += size;

    for(int i = log; i >= 1; i--) {
      if(((l >> i) << i) != l) push(l >> i);
      if(((r >> i) << i) != r) push((r - 1) >> i);
    }

    {
      int l2 = l, r2 = r;
      while(l < r) {
        if(l & 1) all_apply(l++, f);
        if(r & 1) all_apply(--r, f);
        l >>= 1;
        r >>= 1;
      }
      l = l2;
      r = r2;
    }

    for(int i = 1; i <= log; i++) {
      if(((l >> i) << i) != l) update(l >> i);
      if(((r >> i) << i) != r) update((r - 1) >> i);
    }
  }

  template<class G> int max_right(int l, G g) {
    // assert(0 <= l && l <= _n);
    // assert(g(e()));
    if(l == _n) return _n;
    l += size;
    for(int i = log; i >= 1; i--) push(l >> i);
    S sm = e();
    do {
      while(l % 2 == 0) l >>= 1;
      if(!g(op(sm, d[l]))) {
        while(l < size) {
          push(l);
          l = (2 * l);
          if(g(op(sm, d[l]))) {
            sm = op(sm, d[l]);
            l++;
          }
        }
        return l - size;
      }
      sm = op(sm, d[l]);
      l++;
    } while((l & -l) != l);
    return _n;
  }  // d93691

  template<class G> int min_left(int r, G g) {
    // assert(0 <= r && r <= _n);
    // assert(g(e()));
    if(r == 0) return 0;
    r += size;
    for(int i = log; i >= 1; i--) push((r - 1) >> i);
    S sm = e();
    do {
      r--;
      while(r > 1 && (r % 2)) r >>= 1;
      if(!g(op(d[r], sm))) {
        while(r < size) {
          push(r);
          r = (2 * r + 1);
          if(g(op(d[r], sm))) {
            sm = op(d[r], sm);
            r--;
          }
        }
        return r + 1 - size;
      }
      sm = op(d[r], sm);
    } while((r & -r) != r);
    return 0;
  }  // c9a7eb

  private:
  int _n, size, log;
  vector<S> d;
  vector<F> lz;

  void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
  void all_apply(int k, F f) {
    d[k] = mapping(f, d[k]);
    if(k < size) lz[k] = composition(f, lz[k]);
  }
  void push(int k) {
    all_apply(2 * k, lz[k]);
    all_apply(2 * k + 1, lz[k]);
    lz[k] = id();
  }
};
```

potential_dsu.hpp

md5: b2e5eb

```
// base: 650ffa
template<typename Abel> struct potential_dsu {
  using T = typename Abel::T;
  int tCount;
  vector<int> p, rank;
  vector<T> potential;
  int N;
  potential_dsu(int size) {
    N = size;
    p.resize(N, -1);
    rank.resize(N, 0);
    potential.resize(N, Abel::e());
    tCount = N;
  }

  bool same(int x, int y) { return leader(x) == leader(y); }

  // w[y] - w[x] = w
  void merge(int x, int y, T w) {
    w = Abel::op(w, get_potential(x));
    w = Abel::op(w, Abel::inv(get_potential(y)));
    link(leader(x), leader(y), w);
  }

  int leader(int x) {
    if(p[x] < 0) return x;
    int l = leader(p[x]);
    potential[x] = Abel::op(potential[x], potential[p[x]]);
    return p[x] = l;
  }

  T get_potential(int x) {
    leader(x);
    return potential[x];
  }

  // w[y] - w[x]
  T diff(int x, int y) { return Abel::op(get_potential(y),
Abel::inv(get_potential(x))); }

  int count() { return tCount; }  // 154012

  int size(int a) {
    // assert(0 <= a && a < _n);
```

```
      return -p[leader(a)];
   }  // 818fe7

   vector<vector<int>> groups() {
      vector<int> leader_buf(N), group_size(N);
      for(int i = 0; i < N; i++) {
         leader_buf[i] = leader(i);
         group_size[leader_buf[i]]++;
      }
      vector<vector<int>> result(N);
      for(int i = 0; i < N; i++)
result[i].reserve(group_size[i]);
      for(int i = 0; i < N; i++)
result[leader_buf[i]].push_back(i);
      result.erase(remove_if(result.begin(), result.end(), [&]
(const vector<int>& v) { return v.empty(); }),
                   result.end());
      return result;
   }  // 92d7ce

   private:
   void link(int x, int y, T w) {
      if(x == y) return;
      tCount--;
      if(rank[x] < rank[y]) {
         swap(x, y);
         w = Abel::inv(w);
      }
      p[x] += p[y];
      p[y] = x;
      if(rank[x] == rank[y]) rank[x]++;
      tCount--;
      potential[y] = w;
   }
};

/*
struct Abel {
   using T = int;
   static T e() { return 0; }
   static T op(T a, T b) { return a + b; }
   static T inv(T a) { return -a; }
};

potential_dsu<Abel> dsu(N);
*/
```

range_set.hpp

md5: 1bc645

```
template<bool margeAdjacent = true> struct range_set : public
map<ll, ll> {
   auto get(ll p) const {
      auto it = upper_bound(p);
      if(it == begin() || (--it)->second < p) return end();
      return it;
   }

   void insert(ll l, ll r) {
      auto itl = upper_bound(l), itr = upper_bound(r +
margeAdjacent);
      if(itl != begin()) {
         if((--itl)->second < l - margeAdjacent) ++itl;
      }
      if(itl != itr) {
         l = min(l, itl->first);
         r = max(r, prev(itr)->second);
         erase(itl, itr);
      }
      (*this)[l] = r;
   }

   void remove(ll l, ll r) {
      auto itl = upper_bound(l), itr = upper_bound(r);
      if(itl != begin())
         if((--itl)->second < l) ++itl;
      if(itl == itr) return;
      int tl = min(l, itl->first), tr = max(r, prev(itr)-
>second);
      erase(itl, itr);
      if(tl < l) (*this)[tl] = l - 1;
      if(r < tr) (*this)[r + 1] = tr;
   }
   bool same(ll p, ll q) {
      auto it = get(p);
      return it != end() && it->first <= q && q <= it->second;
   }
};
```

range_tree.hpp

md5: 7f74d5

```
template<class K, class M> struct range_tree {
   using S = typename M::S;
   using D = typename M::D;

   private:
   vector<pair<K, K>> ps;
   vector<K> xs;
   vector<vector<K>> ys;
   vector<D> ds;
   int n;
   int id(K x) const { return lower_bound(all(xs), x) -
xs.begin(); }

   int id(int k, K y) const { return lower_bound(all(ys[k]), y)
- ys[k].begin(); }

   public:
   void add(K x, K y) { ps.emplace_back(x, y); }
   void build() {
      sort(ps.begin(), ps.end());
      ps.erase(unique(all(ps)), ps.end());
      n = size(ps);
      xs.reserve(n);
      for(auto& [x, _] : ps) xs.push_back(x);
      ys.resize(2 * n);
      ds.resize(2 * n, M::init(0));
      for(int i = 0; i < n; i++) {
         ys[i + n] = {ps[i].second};
         ds[i + n] = M::init(1);
      }
      for(int i = n - 1; i > 0; i--) {
         ys[i].resize(size(ys[i << 1]) + size(ys[(i << 1) |
1]));
         merge(all(ys[i << 1]), all(ys[(i << 1) | 1]),
ys[i].begin());
         ys[i].erase(unique(all(ys[i])), ys[i].end());
         ds[i] = M::init(size(ys[i]));
      }
   }

   void apply(K x, K y, S a) {
      int k = lower_bound(all(ps), make_pair(x, y)) -
ps.begin() + n;
      while(k > 0) {
         M::apply(ds[k], id(k, y), a);
         k >>= 1;
      }
   }

   S prod(K x1, K y1, K x2, K y2) {
      int a = id(x1), b = id(x2);
      a += n;
      b += n;
      S l = M::e(), r = M::e();
      while(a < b) {
         if(a & 1) {
            l = M::op(l, M::prod(ds[a], id(a, y1), id(a, y2)));
            ++a;
         }
         if(b & 1) {
            --b;
            r = M::op(M::prod(ds[b], id(b, y1), id(b, y2)), r);
         }
         a >>= 1;
         b >>= 1;
      }
      return M::op(l, r);
   }
};

/* 使い方

// モノイド
struct M {
```

```
    using S = ll;  // データ(モノイド)の型
    using D = BIT;  // ノードに持たせるデータ構造の型
    static S op(S a, S b) { return a + b; } // Sの二項演算
    static S e() { return 0; }  // Sの単位元
    static D init(int n) { return BIT(n); }  // Dを長さnで初期化する関数
    static void apply(D& bit, int k, const S& v) { bit.add(k, v); } // D のk番目に v を適用する関数
    static S prod(D& bit, int l, int r) { return bit.sum(l, r); } // D の[l, r) に対するクエリを行う関数
};

rt.add(x, y): 座標 (x, y) を追加
rt.build(): クエリを受け付ける準備をする
rt.apply(x, y, a): 座標 (x, y) に a を適用
rt.prod(x1, y1, x2, y2): 座標 x \in [x1, x2), y \in [y1, y2) の領域にクエリを行う
*/
```

segtree.hpp

md5: d32488

```
// base: bafcf8
unsigned int bit_ceil(unsigned int n) {
    unsigned int x = 1;
    while(x < (unsigned int)(n)) x *= 2;
    return x;
}
int countr_zero(unsigned int n) { return __builtin_ctz(n); }
constexpr int countr_zero_constexpr(unsigned int n) {
    int x = 0;
    while(!(n & (1 << x))) x++;
    return x;
}
template<class S, S (*op)(S, S), S (*e)()> struct segtree {
    public:
    segtree() : segtree(0) {}
    explicit segtree(int n) : segtree(vector<S>(n, e())) {}
    explicit segtree(const vector<S>& v) : _n(int(v.size())) {
        size = (int)bit_ceil((unsigned int)(_n));
        log = countr_zero((unsigned int)size);
        d = vector<S>(2 * size, e());
        for(int i = 0; i < _n; i++) d[size + i] = v[i];
        for(int i = size - 1; i >= 1; i--) { update(i); }
    }

    void set(int p, S x) {
        // assert(0 <= p && p < _n);
        p += size;
        d[p] = x;
        for(int i = 1; i <= log; i++) update(p >> i);
    }

    S get(int p) const {
        // assert(0 <= p && p < _n);
        return d[p + size];
    }

    S prod(int l, int r) const {
        // assert(0 <= l && l <= r && r <= _n);
        S sml = e(), smr = e();
        l += size;
        r += size;

        while(l < r) {
            if(l & 1) sml = op(sml, d[l++]);
            if(r & 1) smr = op(d[--r], smr);
            l >>= 1;
            r >>= 1;
        }
        return op(sml, smr);
    }

    S all_prod() const { return d[1]; }

    template<class F> int max_right(int l, F f) {
        // assert(0 <= l && l <= _n);
        // assert(f(e()));
        if(l == _n) return _n;
        l += size;
        S sm = e();
        do {
            while(l % 2 == 0) l >>= 1;
            if(!f(op(sm, d[l]))) {
                while(l < size) {
                    l = (2 * l);
                    if(f(op(sm, d[l]))) {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
        } while((l & -l) != l);
        return _n;
    }  // faa03f

    template<class F> int min_left(int r, F f) {
        // assert(0 <= r && r <= _n);
        // assert(f(e()));
        if(r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while(r > 1 && (r % 2)) r >>= 1;
            if(!f(op(d[r], sm))) {
                while(r < size) {
                    r = (2 * r + 1);
                    if(f(op(d[r], sm))) {
                        sm = op(d[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while((r & -r) != r);
        return 0;
    }  // efa466

    private:
    int _n, size, log;
    vector<S> d;

    void update(int k) { d[k] = op(d[2 * k], d[2 * k + 1]); }
};
```

sparse_table.hpp

md5: f3812e

```
template<typename T, auto op> struct sparse_table {
    vector<vector<T>> st;
    vector<int> lg;

    sparse_table(const vector<T>& v) {
        int b = 0;
        while((1 << b) <= v.size()) b++;
        st.assign(b, vector<T>(1 << b));
        for(int i = 0; i < size(v); i++) { st[0][i] = v[i]; }
        for(int i = 1; i < b; i++) {
            for(int j = 0; j + (1 << i) <= (1 << b); j++) st[i][j] = op(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
        }
        lg.resize(v.size() + 1);
        for(int i = 2; i < size(lg); i++) lg[i] = lg[i >> 1] + 1;
    }

    inline T prod(int l, int r) {
        int b = lg[r - l];
        return op(st[b][l], st[b][r - (1 << b)]);
    }
};
```

undo_dsu.hpp

md5: f5d93b

```
// base: edf246
struct undo_dsu {
    private:
    int _n;
    vector<int> p;
```

```
   public:
   undo_dsu() : _n(0) {}
   explicit undo_dsu(int n) : _n(n), p(n, -1) {}

   int merge(int a, int b) {
      // assert(0 <= a && a < _n);
      // assert(0 <= b && b < _n);
      int x = leader(a), y = leader(b);
      if(x == y) {
         history.emplace(x, p[x]);
         history.emplace(y, p[y]);
         return x;
      }
      if(-p[x] < -p[y]) swap(x, y);
      history.emplace(x, p[x]);
      history.emplace(y, p[y]);
      p[x] += p[y];
      p[y] = x;
      return x;
   }

   bool same(int a, int b) {
      // assert(0 <= a && a < _n);
      // assert(0 <= b && b < _n);
      return leader(a) == leader(b);
   }

   int leader(int a) {
      // assert(0 <= a && a < _n);
      while(p[a] >= 0) a = p[a];
      return a;
   }

   void undo() {
      p[history.top().first] = history.top().second;
      history.pop();
      p[history.top().first] = history.top().second;
      history.pop();
   }

   int snapshot() { return history.size(); }

   void rollback(int snapshot = 0) {
      while(history.size() > snapshot) undo();
   }

   int size(int a) {
      // assert(0 <= a && a < _n);
      return -p[leader(a)];
   }  // 818fe7
};
```

math

BinaryGCD.hpp

md5: f3ab31

```
u64 ctz(u64 x) { return countr_zero(x); }
u64 binary_gcd(u64 x, u64 y) {
   if(!x || !y) return x | y;
   u64 n = ctz(x), m = ctz(y);
   x >>= n, y >>= m;
   while(x != y) {
      if(x > y) x = (x - y) >> ctz(x - y);
      else y = (y - x) >> ctz(y - x);
   }
   return x << min(n, m);
}
```

ExtGCD.hpp

md5: c3fa9b

```
// returns gcd(a, b) and assign x, y to integers
// s.t. ax + by = gcd(a, b) and |x| + |y| is minimized
ll extgcd(ll a, ll b, ll& x, ll& y) {
   // assert(a >= 0 && b >= 0);
   if(!b) return x = 1, y = 0, a;
   ll d = extgcd(b, a % b, y, x);
   y -= a / b * x;
   return d;
}
```

floor_sum.hpp

md5: 0f7242

```
ll floor_sum(const ll& n, const ll& m, ll a, ll b) {
   ll ret = 0;
   if(a >= m) ret += (n - 1) * n * (a / m) / 2, a %= m;
   if(b >= m) ret += n * (b / m), b %= m;
   ll y = (a * n + b) / m;
   if(y == 0) return ret;
   ll x = y * m - b;
   ret += (n - (x + a - 1) / a) * y;
   ret += floor_sum(y, a, m, (a - x % a) % a);
   return ret;
}
```

modint

BarrettReduction.hpp

md5: 2ca7f3

```
// using u64 = uint64_t;
struct Barrett {  // mod < 2^32
   u64 m, im;
   Barrett(u64 mod) : m(mod), im(-1ULL / m + 1) {}
   // input: a * b < 2^64, output: a * b % mod
   u64 mul(u64 a, u64 b) const {
      a *= b;
      u64 x = ((__uint128_t)a * im) >> 64;
      a -= x * m;
      if((ll)a < 0) a += m;
      return a;
   }
};
```

modint.hpp

md5: 81b530

```
const ll mod = 998244353;
struct mm {
   ll x;
   mm(ll x_ = 0) : x(x_ % mod) {
      if(x < 0) x += mod;
   }
   friend mm operator+(mm a, mm b) { return a.x + b.x; }
   friend mm operator-(mm a, mm b) { return a.x - b.x; }
   friend mm operator*(mm a, mm b) { return a.x * b.x; }
   friend mm operator/(mm a, mm b) { return a * b.inv(); }
   // 4 行コピペ  Alt + Shift + クリックで複数カーソル
   friend mm& operator+=(mm& a, mm b) { return a = a.x + b.x; }
   friend mm& operator-=(mm& a, mm b) { return a = a.x - b.x; }
   friend mm& operator*=(mm& a, mm b) { return a = a.x * b.x; }
   friend mm& operator/=(mm& a, mm b) { return a = a * b.inv();
}
   mm inv() const { return pow(mod - 2); }
   mm pow(ll b) const {
      mm a = *this, c = 1;
      while(b) {
         if(b & 1) c *= a;
         a *= a;
         b >>= 1;
      }
      return c;
   }
};
```

FPS

FFT.hpp

md5: 6e60c3

```
// {998244353, 3}, {1811939329, 13}, {2013265921, 31}
mm g = 3;  // 原始根
void fft(vector<mm>& a) {
   ll n = size(a), lg = __lg(n);
   assert((1 << lg) == n);
   vector<mm> b(n);
   for(int l = 1; l <= lg; l++) {
      ll w = n >> l;
      mm s = 1, r = g.pow(mod >> l);
      for(ll u = 0; u < n / 2; u += w) {
         for(int d = 0; d < w; d++) {
            mm x = a[u << 1 | d], y = a[u << 1 | w | d] * s;
            b[u | d] = x + y;
```

```
            b[n >> 1 | u | d] = x - y;
        }
        s *= r;
    }
    swap(a, b);
  }
}
vector<mm> conv(vector<mm> a, vector<mm> b) {
  if(a.empty() || b.empty()) return {};
  size_t s = size(a) + size(b) - 1, n = bit_ceil(s);
  // if(min(sz(a), sz(b)) <= 60) 愚直に掛け算
  a.resize(n);
  b.resize(n);
  fft(a);
  fft(b);
  mm inv = mm(n).inv();
  for(int i = 0; i < n; i++) a[i] *= b[i] * inv;
  reverse(1 + all(a));
  fft(a);
  a.resize(s);
  return a;
}
```

FFT_fast.hpp

md5: 33f77f

```
// modint を u32 にして加減算を真面目にやると速い
mm g = 3;  // 原始根
void fft(vector<mm>& a) {
  ll n = size(a), lg = __lg(n);
  static auto z = [] {
    vector<mm> z(30);
    mm s = 1;
    for(int i = 2; i < 32; i++) {
      z[i - 2] = s * g.pow(mod >> i);
      s *= g.inv().pow(mod >> i);
    }
    return z;
  }();
  for(int l = 0; l < lg; l++) {
    ll w = 1 << (lg - l - 1);
    mm s = 1;
    for(int k = 0; k < (1 << l); k++) {
      ll o = k << (lg - l);
      for(ll i = o; i < o + w; i++) {
        mm x = a[i], y = a[i + w] * s;
        a[i] = x + y;
        a[i + w] = x - y;
      }
      s *= z[countr_zero<uint64_t>(~k)];
    }
  }
}
// コピペ
void ifft(vector<mm>& a) {
  ll n = size(a), lg = __lg(n);
  static auto z = [] {
    vector<mm> z(30);
    mm s = 1;
    for(int i = 2; i < 32; i++) {  // g を逆数に
      z[i - 2] = s * g.inv().pow(mod >> i);
      s *= g.pow(mod >> i);
    }
    return z;
  }();
  for(ll l = lg; l--;) {  // 逆順に
    ll w = 1 << (lg - l - 1);
    mm s = 1;
    for(int k = 0; k < (1 << l); k++) {
      ll o = k << (lg - l);
      for(ll i = o; i < o + w; i++) {
        mm x = a[i], y = a[i + w];  // *s を下に移動
        a[i] = x + y;
        a[i + w] = (x - y) * s;
      }
      s *= z[countr_zero<uint64_t>(~k)];
    }
  }
}
vector<mm> conv(vector<mm> a, vector<mm> b) {
  if(a.empty() || b.empty()) return {};
  size_t s = size(a) + size(b) - 1, n = bit_ceil(s);
  // if(min(size(a), size(b)) <= 60) 愚直に掛け算
  a.resize(n);
  b.resize(n);
  fft(a);
  fft(b);
  mm inv = mm(n).inv();
  for(int i = 0; i < n; i++) a[i] *= b[i] * inv;
  ifft(a);
  a.resize(s);
  return a;
}
```

graph

low_link.hpp

md5: d5d2f7

```
struct LowLink {
  public:
  vector<vector<int>> g;
  vector<int> ord, low;
  vector<int> articulation;
  vector<bool> visited;
  vector<pair<int, int>> bridge;

  void dfs(int cur, int pre, int& k) {
    visited[cur] = true;
    ord[cur] = low[cur] = k++;
    bool isArticulation = false;
    int cnt = 0;
    for(auto to : g[cur]) {
      if(!visited[to]) {
        cnt++;
        dfs(to, cur, k);
        chmin(low[cur], low[to]);
        if(pre != -1 && ord[cur] <= low[to]) isArticulation
= true;
        if(ord[cur] < low[to]) bridge.emplace_back(min(cur,
to), max(cur, to));
      } else if(to != pre) chmin(low[cur], ord[to]);
    }
    if(pre == -1 && cnt > 1) isArticulation = true;
    if(isArticulation) articulation.push_back(cur);
  }

  void build(const vector<vector<int>>& g) {
    int n = g.size();
    this->g = g;
    ord.assign(n, -1);
    low.assign(n, -1);
    visited.assign(n, false);
    int k = 0;
    for(int i = 0; i < n; i++)
      if(!visited[i]) dfs(i, -1, k);
  }
  LowLink(const vector<vector<int>>& g) { build(g); }

  vector<int>& getArticulations() { return articulation; }
  vector<pair<int, int>>& getBridges() { return bridge; }
  vector<int>& getOrd() { return ord; }
  vector<int>& getLowlink() { return low; }
};
```

max_flow.hpp

md5: a7f1d5

```
// base: 9927a4
template<class Cap> struct mf_graph {
  public:
  mf_graph() : _n(0) {}
  mf_graph(int n) : _n(n), g(n) {}

  int add_edge(int from, int to, Cap cap) {
    // assert(0 <= from && from < _n);
    // assert(0 <= to && to < _n);
    // assert(0 <= cap);
    int m = size(pos);
    pos.push_back({from, size(g[from])});
    int from_id = size(g[from]);
    int to_id = size(g[to]);
    if(from == to) to_id++;
    g[from].push_back(_edge{to, to_id, cap});
    g[to].push_back(_edge{from, from_id, 0});
```

```
      return m;
   }

   Cap flow(int s, int t, Cap flow_limit =
numeric_limits<Cap>::max()) {
      // assert(0 <= s && s < _n);
      // assert(0 <= t && t < _n);
      // assert(s != t);

      vector<int> level(_n), iter(_n);
      queue<int> que;
      auto bfs = [&]() {
         fill(all(level), -1);
         level[s] = 0;
         while(!que.empty()) que.pop();
         que.push(s);
         while(!que.empty()) {
            int v = que.front();
            que.pop();
            for(auto e : g[v]) {
               if(e.cap == 0 || level[e.to] >= 0) continue;
               level[e.to] = level[v] + 1;
               if(e.to == t) return;
               que.push(e.to);
            }
         }
      };
      auto dfs = [&](auto self, int v, Cap up) {
         if(v == s) return up;
         Cap res = 0;
         int level_v = level[v];
         for(int& i = iter[v]; i < size(g[v]); i++) {
            _edge& e = g[v][i];
            if(level_v <= level[e.to] || g[e.to][e.rev].cap ==
0) continue;
            Cap d = self(self, e.to, min(up - res, g[e.to]
[e.rev].cap));
            if(d <= 0) continue;
            g[v][i].cap += d;
            g[e.to][e.rev].cap -= d;
            res += d;
            if(res == up) break;
         }
         return res;
      };

      Cap flow = 0;
      while(flow < flow_limit) {
         bfs();
         if(level[t] == -1) break;
         fill(all(iter), 0);
         while(flow < flow_limit) {
            Cap f = dfs(dfs, t, flow_limit - flow);
            if(!f) break;
            flow += f;
         }
      }
      return flow;
   }

   vector<bool> min_cut(int s) {
      vector<bool> visited(_n);
      queue<int> que;
      que.push(s);
      visited[s] = true;
      while(!que.empty()) {
         int v = que.front();
         que.pop();
         for(auto e : g[v]) {
            if(e.cap && !visited[e.to]) {
               visited[e.to] = true;
               que.push(e.to);
            }
         }
      }
      return visited;
   }  // 8735cf

   struct edge {
      int from, to;
      Cap cap, flow;
   };  // 9fe107

   edge get_edge(int i) {
      int m = size(pos);
      // assert(0 <= i && i < m);
      auto _e = g[pos[i].first][pos[i].second];
      auto _re = g[_e.to][_e.rev];
      return edge{pos[i].first, _e.to, _e.cap + _re.cap,
_re.cap};
   }  // ad4299

   vector<edge> edges() {
      int m = size(pos);
      vector<edge> result;
      for(int i = 0; i < m; i++) result.push_back(get_edge(i));
      return result;
   }  // 5948b8

   void change_edge(int i, Cap new_cap, Cap new_flow) {
      int m = size(pos);
      // assert(0 <= i && i < m);
      // assert(0 <= new_flow && new_flow <= new_cap);
      auto& _e = g[pos[i].first][pos[i].second];
      auto& _re = g[_e.to][_e.rev];
      _e.cap = new_cap - new_flow;
      _re.cap = new_flow;
   }  // 558c35

   private:
   int _n;
   struct _edge {
      int to, rev;
      Cap cap;
   };
   vector<pair<int, int>> pos;
   vector<vector<_edge>> g;
};
```

min_cost_flow.hpp

md5: 17d51b

```
// base: 4e9f1c
template<class Cap, class Cost> struct mcf_graph {
   public:
   mcf_graph() {}
   mcf_graph(int n) : _n(n), g(n) {}

   int add_edge(int from, int to, Cap cap, Cost cost) {
      // assert(0 <= from && from < _n);
      // assert(0 <= to && to < _n);
      int m = size(pos);
      pos.push_back({from, size(g[from])});
      int from_id = size(g[from]);
      int to_id = size(g[to]);
      if(from == to) to_id++;
      g[from].push_back(_edge{to, to_id, cap, cost});
      g[to].push_back(_edge{from, from_id, 0, -cost});
      return m;
   }

   pair<Cap, Cost> flow(int s, int t, Cap flow_limit =
numeric_limits<Cap>::max()) {
      return slope(s, t, flow_limit).back();
   }
   vector<pair<Cap, Cost>> slope(int s, int t, Cap flow_limit =
numeric_limits<Cap>::max()) {
      // assert(0 <= s && s < _n);
      // assert(0 <= t && t < _n);
      // assert(s != t);
      vector<Cost> dual(_n, 0), dist(_n);
      vector<int> pv(_n), pe(_n);
      vector<bool> vis(_n);
      auto dual_ref = [&]() {
         fill(all(dist), numeric_limits<Cost>::max());
         fill(all(pv), -1);
         fill(all(pe), -1);
         fill(all(vis), false);
         struct Q {
            Cost key;
            int to;
            bool operator<(const Q& r) const { return key >
r.key; }
         };
         priority_queue<Q> que;
         dist[s] = 0;
```

```
        que.push(Q{0, s});
        while(!que.empty()) {
           int v = que.top().to;
           que.pop();
           if(vis[v]) continue;
           vis[v] = true;
           if(v == t) break;
           for(int i = 0; i < size(g[v]); i++) {
              auto e = g[v][i];
              if(vis[e.to] || !e.cap) continue;
              Cost cost = e.cost - dual[e.to] + dual[v];
              if(chmin(dist[e.to], dist[v] + cost)) {
                 pv[e.to] = v;
                 pe[e.to] = i;
                 que.push(Q{dist[e.to], e.to});
              }
           }
        }
        if(!vis[t]) return false;
        for(int v = 0; v < _n; v++)
           if(vis[v]) dual[v] -= dist[t] - dist[v];
        return true;
     };
     Cap flow = 0;
     Cap cost = 0, prev_cost_per_flow = -1;
     vector<pair<Cap, Cost>> result;
     result.push_back({flow, cost});
     while(flow < flow_limit) {
        if(!dual_ref()) break;
        Cap c = flow_limit - flow;
        for(int v = t; v != s; v = pv[v]) { c = min(c,
g[pv[v]][pe[v]].cap); }
        for(int v = t; v != s; v = pv[v]) {
           auto& e = g[pv[v]][pe[v]];
           e.cap -= c;
           g[v][e.rev].cap += c;
        }
        Cost d = -dual[s];
        flow += c;
        cost += c * d;
        if(prev_cost_per_flow == d) { result.pop_back(); }
        result.push_back({flow, cost});
        prev_cost_per_flow = d;
     }
     return result;
  }

  struct edge {
     int from, to;
     Cap cap, flow;
  };  // 9fe107

  edge get_edge(int i) {
     int m = size(pos);
     // assert(0 <= i && i < m);
     auto _e = g[pos[i].first][pos[i].second];
     auto _re = g[_e.to][_e.rev];
     return edge({pos[i].first, _e.to, _e.cap + _re.cap,
_re.cap});
  }  // d7bd7e

  vector<edge> edges() {
     int m = size(pos);
     vector<edge> result;
     for(int i = 0; i < m; i++) result.push_back(get_edge(i));
     return result;
  }  // 5948b8

  void change_edge(int i, Cap new_cap, Cap new_flow) {
     int m = size(pos);
     // assert(0 <= i && i < m);
     // assert(0 <= new_flow && new_flow <= new_cap);

     auto& _e = g[pos[i].first][pos[i].second];
     auto& _re = g[_e.to][_e.rev];
     _e.cap = new_cap - new_flow;
     _re.cap = new_flow;
  }  // 558c35

  private:
  int _n;
  struct _edge {
     int to, rev;
     Cap cap;
     Cost cost;
  };

  vector<pair<int, int>> pos;
  vector<vector<_edge>> g;
};
```

scc.hpp

md5: 9f5fd6

```
// base: 3085f6
struct scc_graph {
  public:
  explicit scc_graph(int _n = 0) : n(_n), G(_n), rG(_n),
comp(_n, -1), visited(_n, 0) {}

  void add_edge(int from, int to) {
     // assert(0 <= from && from < n);
     // assert(0 <= to && to < n);
     G[from].push_back(to);
     rG[to].push_back(from);
  }

  vector<vector<int>> scc() {
     fill(all(visited), 0);
     fill(all(comp), -1);
     order.clear();
     for(int i = 0; i < n; i++)
        if(!visited[i]) dfs(i);
     comp_size = 0;
     for(int i = size(order) - 1; i >= 0; i--) {
        if(comp[order[i]] < 0) rdfs(order[i], comp_size++);
     }
     vector<vector<int>> v(comp_size);
     for(int i = 0; i < n; i++) v[comp[i]].push_back(i);
     return v;
  }

  vector<int> get_comp() { return comp; }  // bdafc0

  vector<vector<int>> dag() {
     vector<vector<int>> res(comp_size);
     for(int i = 0; i < n; i++)
        for(auto j : G[i]) {
           if(comp[i] != comp[j])
res[comp[i]].push_back(comp[j]);
        }
     for(int i = 0; i < comp_size; i++) {
        sort(all(res[i]));
        res[i].erase(unique(all(res[i])), res[i].end());
     }
     return res;
  }  // 312650

  private:
  vector<vector<int>> G, rG;
  vector<int> order, comp;
  vector<bool> visited;
  int n, comp_size;

  void dfs(int v) {
     visited[v] = true;
     for(auto to : G[v])
        if(!visited[to]) dfs(to);
     order.push_back(v);
  }

  void rdfs(int v, int k) {
     comp[v] = k;
     for(auto to : rG[v]) {
        if(comp[to] < 0) rdfs(to, k);
     }
  }
};
```

topological_sort.hpp

md5: 024d40

```
vector<int> topological_sort(vector<vector<int>>& g) {
  int n = g.size();
  vector<int> indeg(n);
  for(int i = 0; i < n; i++)
```

```
      for(int j : g[i]) indeg[j]++;
   vector<int> res;
   queue<int> q;
   for(int i = 0; i < n; i++)
      if(indeg[i] == 0) q.push(i);
   while(!q.empty()) {
      auto v = q.front();
      q.pop();
      res.push_back(v);
      for(auto u : g[v]) {
         indeg[u]--;
         if(indeg[u] == 0) q.push(u);
      }
   }
   return res;
}
```

two_sat.hpp

md5: 681721

```
struct two_sat {
   public:
   two_sat() : _n(0), scc(0) {}
   two_sat(int n) : _n(n), scc(2 * n), _answer(n) {}

   void add_clause(int i, bool f, int j, bool g) {
      // assert(0 <= i && i < _n);
      // assert(0 <= j && j < _n);
      scc.add_edge(2 * i + (f ? 0 : 1), 2 * j + (g ? 1 : 0));
      scc.add_edge(2 * j + (g ? 0 : 1), 2 * i + (f ? 1 : 0));
   }

   bool satisfiable() {
      scc.scc();
      auto comp = scc.get_comp();
      for(int i = 0; i < _n; i++) {
         if(comp[2 * i] == comp[2 * i + 1]) return false;
         _answer[i] = comp[2 * i] < comp[2 * i + 1];
      }
      return true;
   }

   vector<bool> answer() { return _answer; }

   private:
   int _n;
   vector<bool> _answer;
   scc_graph scc;
};
```

graph/tree

cartesian_tree.hpp

md5: ac77a5

```
template<class T> struct cartesian_tree {
   int root;
   vector<int> par, left, right;

   cartesian_tree(const vector<T>& v) : root(0), par(size(v),
-1), left(size(v), -1), right(size(v), -1) {
      stack<int> st;
      int N = size(v);
      for(int i = 0; i < N; i++) {
         int prev = -1;
         while(!st.empty() && v[st.top()] > v[i]) {
            prev = st.top();
            st.pop();
         }
         if(prev != -1) par[prev] = i;
         if(!st.empty()) par[i] = st.top();
         st.push(i);
      }

      root = -1;
      for(int i = 0; i < N; i++) {
         if(par[i] == -1) root = i;
         else if(par[i] < i) right[par[i]] = i;
         else left[par[i]] = i;
      }
   }
};
```

hld.hpp

md5: 10247f

```
class HLDcomposition {
   private:
   int V;
   vector<vector<int>> G;
   vector<int> stsize, parent, pathtop, in, out;
   int root;
   void build_stsize(int u, int p) {
      stsize[u] = 1, parent[u] = p;
      for(auto&& v : G[u]) {
         if(v == p) {
            if(v == G[u].back()) break;
            else swap(v, G[u].back());
         }
         build_stsize(v, u);
         stsize[u] += stsize[v];
         if(stsize[v] > stsize[G[u][0]]) swap(v, G[u][0]);
      }
   }

   void build_path(int u, int p, int& tm) {
      in[u] = tm++;
      for(auto v : G[u]) {
         if(v == p) continue;
         pathtop[v] = (v == G[u][0] ? pathtop[u] : v);
         build_path(v, u, tm);
      }
      out[u] = tm;
   }

   public:
   void add_edge(int u, int v) {
      G[u].push_back(v);
      G[v].push_back(u);
   }

   void build(int _root = 0) {
      root = _root;
      int tm = 0;
      build_stsize(root, -1);
      pathtop[root] = root;
      build_path(root, -1, tm);
   }

   inline int index(int a) { return in[a]; }

   int lca(int a, int b) {
      int pa = pathtop[a], pb = pathtop[b];
      while(pa != pb) {
         if(in[pa] > in[pb]) {
            a = parent[pa], pa = pathtop[a];
         } else {
            b = parent[pb], pb = pathtop[b];
         }
      }
      if(in[a] > in[b]) swap(a, b);
      return a;
   }

   pair<int, int> subtree_query(int a) { return {in[a],
out[a]}; }

   vector<tuple<int, int, bool>> path_query(int from, int to) {
      int pf = pathtop[from], pt = pathtop[to];
      using T = tuple<int, int, bool>;
      deque<T> front, back;
      while(pf != pt) {
         if(in[pf] > in[pt]) {
            front.push_back({in[pf], in[from] + 1, true});
            from = parent[pf], pf = pathtop[from];
         } else {
            back.push_front({in[pt], in[to] + 1, false});
            to = parent[pt], pt = pathtop[to];
         }
      }
      if(in[from] > in[to]) front.push_back({in[to], in[from] +
1, true});
      else front.push_back({in[from], in[to] + 1, false});
      vector<T> ret;
      while(!front.empty()) {
```

```
        ret.push_back(front.front());
        front.pop_front();
      }
      while(!back.empty()) {
        ret.push_back(back.front());
        back.pop_front();
      }
      return ret;
   }

   HLDcomposition(int node_size)
       : V(node_size), G(V), stsize(V, 0), parent(V, -1),
pathtop(V, -1), in(V, -1), out(V, -1) {}
};
```

rerooting.hpp

md5: 3bb537

```
// base: b7fc0f
template<class M, bool calc_contribution = false> struct
Rerooting {
   using S = typename M::S;
   using C = typename M::C;
   vector<S> dp, memo;
   vector<vector<pair<int, C>>> g;
   map<ll, S> hash;
   int N;

   Rerooting(int n) : N(n), g(n) {}

   void add_edge(int f, int t, const C& c) {
      g[f].emplace_back(t, c);
      g[t].emplace_back(f, c);
   }

   vector<S> build() {
      memo.resize(N, M::e());
      dp.resize(N, M::e());
      dfs(0, -1);
      reroot(0, M::e());
      return dp;
   }

   private:
   void dfs(int cur, int pre = -1) {
      bool is_leaf = true;
      for(auto [to, c] : g[cur]) {
        if(to == pre) continue;
        is_leaf = false;
        dfs(to, cur);
        memo[cur] = M::merge(memo[cur], M::apply(memo[to], to,
cur, c));
      }
      if(is_leaf) { memo[cur] = M::leaf(); }
   }

   void reroot(int cur, const S val, int pre = -1) {
      vector<S> ds;
      ds.push_back(val);
      if(calc_contribution) {
        if(pre == -1) hash[cur * N + pre] = val;
      }
      for(auto [to, c] : g[cur]) {
        if(to == pre) continue;
        ds.push_back(M::apply(memo[to], to, cur, c));
        if(calc_contribution) { hash[cur * N + to] =
ds.back(); }
      }
      int n = size(ds);
      vector<S> l(n + 1, M::e()), r(n + 1, M::e());
      for(int i = 0; i < n; i++) l[i + 1] = M::merge(l[i],
ds[i]);
      for(int i = n - 1; i >= 0; i--) r[i] = M::merge(ds[i],
r[i + 1]);
      dp[cur] = r[0];
      int ind = 1;
      for(auto [to, c] : g[cur]) {
        if(to == pre) continue;
        S sub = M::merge(l[ind], r[ind + 1]);
        reroot(to, M::apply(sub, cur, to, c), cur);
        ind++;
      }
   }
   public:
   S get_contribution(int p, int c) {
      if(hash.count(p * N + c)) return hash[p * N + c];
      return M::e();
   }  // e6000f
};

/*

struct M {
   using S = pair<mm, int>; // DPの型
   using C = pair<mm, mm>;  // 辺コストの型
   static S merge(S a, S b) { return {a.first + b.first,
a.second + b.second}; } // DPのマージ
   static S apply(S a, int from, int to, C b) {  // DPの親への寄
与
      return {(a.first + A[from]) * b.first + b.second *
(a.second + 1), a.second + 1};
   }
   static S e() { return {0, 0}; } // 単位元
   static S leaf() { return {0, 0}; } // 葉の値
};

Rerooting<M> reroot;
*/
```

flow

燃やす埋める.md

変形前の制約	変形後の制約
x が 0 のとき z 失う	(x, T, z)
x が 0 のとき z 得る	無条件で z 得る; (S, x, z)
x が 1 のとき z 失う	(S, x, z)
x が 1 のとき z 得る	無条件で z 得る; (x, T, z)
$x, y, \ldots$ がすべて 0 のとき z 得る	無条件で z 得る; $(S, w, z), (w, x, \infty), (w, y, \infty)$
$x, y, \ldots$ がすべて 1 のとき z 得る	無条件で z 得る; $(w, T, z), (x, w, \infty), (y, w, \infty)$

string

KMP.hpp

md5: 298f79

```
// kmp[i] := max{ l ≤ i | s[:l] == s[(i+1)-l:i+1] }
// abacaba -> 0010123
auto KMP(string s) {
   vector<ll> p(size(s));
   for(int i = 1; i < size(s); i++) {
      ll g = p[i - 1];
      while(g && s[i] != s[g]) g = p[g - 1];
      p[i] = g + (s[i] == s[g]);
   }
   return p;
}
```

Manacher.hpp

md5: 20c548

```
// 各位置での回文半径を求める
// aaabaaa -> 1214121
// 偶数長の回文を含めて直径を知るには，N+1 個の $ を挿入して 1 を引く
// $a$a$a$b$a$a$a$ -> 123432181234321
auto manacher(string s) {
   ll n = size(s), i = 0, j = 0;
   vector<ll> r(n);
   while(i < n) {
      while(i >= j && i + j < n && s[i - j] == s[i + j]) j++;
      r[i] = j;
      ll k = 1;
      while(i >= k && i + k < n && k + r[i - k] < j) {
        r[i + k] = r[i - k];
        k++;
      }
```

```
      i += k, j -= k;
   }
   return r;
}
```

RollingHash.hpp

md5: 7403a8

```
// using u64 = uint64_t;
const u64 mod = INF;
u64 add(u64 a, u64 b) {
   a += b;
   if(a >= mod) a -= mod;
   return a;
}
u64 mul(u64 a, u64 b) {
   auto c = (__uint128_t)a * b;
   return add(c >> 61, c & mod);
}
random_device rnd;
const u64 r = ((u64)rnd() << 32 | rnd()) % mod;
struct RH {
   ll n;
   vector<u64> hs, pw;
   RH(string s) : n(size(s)), hs(n + 1), pw(n + 1, 1) {
      for(int i = 0; i < n; i++) {
         pw[i + 1] = mul(pw[i], r);
         hs[i + 1] = add(mul(hs[i], r), s[i]);
      }
   }
   u64 get(ll l, ll r) const { return add(hs[r], mod -
mul(hs[l], pw[r - l])); }
};
```

SuffixArray.hpp

md5: cbf1dc

```
// returns pair{sa, lcp}
// sa 長さ n : s[sa[0]:] < s[sa[1]:] < … < s[sa[n-1]:]
// lcp 長さ n-1 : lcp[i] = LCP(s[sa[i]:], s[sa[i+1]:])
auto SA(string s) {
   ll n = size(s) + 1, lim = 256;
   // assert(lim > ranges::max(s));
   vector<ll> sa(n), lcp(n), x(all(s) + 1), y(n), ws(max(n,
lim)), rk(n);
   iota(all(sa), 0);
   for(ll j = 0, p = 0; p < n; j = max(1LL, j * 2), lim = p) {
      p = j;
      iota(all(y), n - j);
      for(int i = 0; i < n; i++)
         if(sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      for(int i = 0; i < n; i++) ws[x[i]]++;
      for(int i = 1; i < lim; i++) ws[i] += ws[i - 1];
      for(ll i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y);
      p = 1;
      x[sa[0]] = 0;
      for(int i = 1; i < n; i++) {
         ll a = sa[i - 1], b = sa[i];
         x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1
: p++;
      }
   }
   for(int i = 1; i < n; i++) rk[sa[i]] = i;
   for(int i = 0, k = 0; i < n - 1; lcp[rk[i++]] = k) {
      if(k) k--;
      while(s[i + k] == s[sa[rk[i] - 1] + k]) k++;
   }
   sa.erase(begin(sa));
   lcp.erase(begin(lcp));
   return pair{sa, lcp};
}
```

Zalgorithm.hpp

md5: 6388f3

```
// Z[i] := LCP(s, s[i:])
// abacaba -> 7010301
auto Z(string s) {
   ll n = size(s), l = -1, r = -1;
   vector<ll> z(n, n);
   for(int i = 1; i < n; i++) {
      ll& x = z[i] = i < r ? min(r - i, z[i - l]) : 0;
      while(i + x < n && s[i + x] == s[x]) x++;
      if(i + x > r) l = i, r = i + x;
   }
   return z;
}
```

aho_corasick.hpp

md5: f30f1f

```
// base: 822702
template<int char_size, int margin> struct AhoCorasick :
Trie<char_size + 1, margin> {
   using Trie<char_size + 1, margin>::Trie;

   const int FAIL = char_size;
   vector<int> correct;

   void build(bool heavy = true) {
      correct.resize(this->nodes.size());
      for(int i = 0; i < size(this->nodes); ++i) { correct[i] =
size(this->nodes[i].accept); }
      queue<int> que;
      for(int i = 0; i < char_size; ++i) {
         if(~this->nodes[0].nxt[i]) {
            this->nodes[this->nodes[0].nxt[i]].nxt[FAIL] = 0;
            que.emplace(this->nodes[0].nxt[i]);
         } else {
            this->nodes[0].nxt[i] = 0;
         }
      }
      while(!que.empty()) {
         auto& now = this->nodes[que.front()];
         int fail = now.nxt[FAIL];
         correct[que.front()] += correct[fail];
         que.pop();
         for(int i = 0; i < char_size; i++) {
            if(~now.nxt[i]) {
               this->nodes[now.nxt[i]].nxt[FAIL] = this-
>nodes[fail].nxt[i];
               if(heavy) {
                  auto& u = this->nodes[now.nxt[i]].accept;
                  auto& v = this->nodes[this-
>nodes[fail].nxt[i]].accept;
                  vector<int> accept;
                  set_union(all(u), all(v),
back_inserter(accept));
                  u = accept;
               }
               que.emplace(now.nxt[i]);
            } else {
               now.nxt[i] = this->nodes[fail].nxt[i];
            }
         }
      }
   }

   vector<int> match(const char& c, int now = 0) {
      vector<int> res;
      now = this->nodes[now].nxt[c - margin];
      for(auto& v : this->nodes[now].accept) res.push_back(v);
      return res;
   }  // 68ef6b

   unordered_map<int, int> match(const string& str, int now =
0) {
      unordered_map<int, int> res, visit_cnt;
      for(auto& c : str) {
         now = this->nodes[now].nxt[c - margin];
         visit_cnt[now]++;
      }
      for(auto& [now, cnt] : visit_cnt) {
         for(auto& v : this->nodes[now].accept) res[v] += cnt;
      }
      return res;
   }  // 36fe6c

   pair<ll, int> move(const char& c, int now = 0) {
      now = this->nodes[now].nxt[c - margin];
      return {correct[now], now};
   }  // 43ccad

   pair<ll, int> move(const string& str, int now = 0) {
      ll res = 0;
```

```
      for(auto& c : str) {
         auto [cnt, nxt] = move(c, now);
         res += cnt;
         now = nxt;
      }
      return {res, now};
   }  // b1949a
};
```

trie.hpp

md5: 09415e

```
template<int char_size> struct TrieNode {
   int nxt[char_size];
   int exist;
   vector<int> accept;

   TrieNode() : exist(0) { memset(nxt, -1, sizeof(nxt)); }
};

template<int char_size, int margin> struct Trie {
   using Node = TrieNode<char_size>;

   vector<Node> nodes;
   int root;
   Trie() : root(0) { nodes.push_back(Node()); }

   void update_direct(int node, int id) {
nodes[node].accept.push_back(id); }

   void update_child(int node, int child, int id) {
++nodes[node].exist; }

   void add(const string& str, int str_index, int node_index,
int id) {
      if(str_index == size(str)) {
         update_direct(node_index, id);
      } else {
         const int c = str[str_index] - margin;
         if(nodes[node_index].nxt[c] == -1) {
            nodes[node_index].nxt[c] = size(nodes);
            nodes.push_back(Node());
         }
         add(str, str_index + 1, nodes[node_index].nxt[c], id);
         update_child(node_index, nodes[node_index].nxt[c],
id);
      }
   }

   void add(const string& str, int id = -1) {
      if(id == -1) id = nodes[0].exist;
      add(str, 0, 0, id);
   }

   void query(const string& str, const function<void(int)>& f,
int str_index, int node_index) {
      for(auto& idx : nodes[node_index].accept) f(idx);
      if(str_index == size(str)) {
         return;
      } else {
         const int c = str[str_index] - margin;
         if(nodes[node_index].nxt[c] == -1) return;
         query(str, f, str_index + 1,
nodes[node_index].nxt[c]);
      }
   }

   void query(const string& str, const function<void(int)>& f)
{ query(str, f, 0, 0); }

   int count() const { return nodes[0].exist; }
};
```

algorithm

doubling.hpp

md5: df858f

```
template<int L> struct Doubling {
   private:
   vector<vector<int>> V;

   public:
   Doubling(const vector<int>& v) {
      int N = size(v);
      V = vector<vector<int>>(L, vector<int>(N));
      for(int i = 0; i < N; i++) V[0][i] = v[i];
      for(int i = 0; i < L - 1; i++)
         for(int j = 0; j < N; j++) {
            if(V[i][j] != -1) V[i + 1][j] = V[i][V[i][j]];
         }
   }

   int jump(int from, ll k) {
      for(int cnt = 0; k > 0; k >>= 1, ++cnt) {
         if((k & 1) && from != -1) from = V[cnt][from];
      }
      return from;
   }
};
```

doubling_monoid.hpp

md5: 530e69

```
template<int L, class T, auto op, auto e> struct Doubling {
   private:
   vector<vector<int>> V;
   vector<vector<T>> data;

   public:
   Doubling(const vector<int>& to, const vector<T>& v) {
      int N = size(to);
      V = vector<vector<int>>(L, vector<int>(N, -1));
      data = vector<vector<T>>(L, vector<T>(N, e()));
      for(int i = 0; i < N; i++) {
         V[0][i] = to[i];
         data[0][i] = v[i];
      }

      for(int i = 0; i < L - 1; i++) {
         for(int j = 0; j < N; j++) {
            if(V[i][j] != -1) {
               V[i + 1][j] = V[i][V[i][j]];
               data[i + 1][j] = op(data[i][j], data[i][V[i]
[j]]);
            } else {
               V[i + 1][j] = V[i][j];
               data[i + 1][j] = data[i][j];
            }
         }
      }
   }

   pair<int, T> jump(int from, ll k) {
      T res = e();
      for(int cnt = 0; k > 0; k >>= 1, ++cnt) {
         if((k & 1) && from != -1) {
            res = op(res, data[cnt][from]);
            from = V[cnt][from];
         }
      }
      return {from, res};
   }
};
```

geometry

memo

Primes.md

素数の個数

n	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
$\pi(n)$	25	168	1229	9592	78498	664579	5.76e+6	5.08e+7	4.55e+8

高度合成数

$\leq n$	10^3	10^4	10^5	10^6	10^7	10^8	10^9
x	840	7560	83160	720720	8648640	73513440	735134400
$d^0(x)$	32	64	128	240	448	768	1344

$\leq n$	10^{10}	10^{11}	10^{12}	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$d^0(x)$	2304	4032	6720	10752	17280	26880	41472	64512	103680

素数階乗

n	2	3	5	7	11	13	17	19	23	29
$n\#$	2	6	30	210	2310	30030	510510	9.70e+6	2.23e+8	6.47e+9

階乗

4!	5!	6!	7!	8!	9!	10!	11!	12!	13!
24	120	720	5040	40320	362880	3.63e+6	3.99e+7	4.79e+8	6.23e+9

ext.cpp

md5: 64e006

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/rope>

using namespace __gnu_pbds;
using namespace __gnu_cxx;  // for ext/rope
using namespace std;

int main() {
    tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> tree;
    tree.insert(1);                         // 1
    tree.insert(10);                        // 1 10
    tree.insert(6);                         // 1 6 10
    tree.insert(7);                         // 1 6 7 10
    tree.insert(4);                         // 1 4 6 7 10
    tree.erase(6);                          // 1 4 7 10
    assert(tree.order_of_key(5) == 2);      // 5未満の要素数
    assert(*tree.find_by_order(2) == 7);  // 0-indexedで2番目の要
素

    gp_hash_table<int, int> m;
    m[2] = 5;
    m[1] = 10;
    m[3] = 7;

    __gnu_pbds::priority_queue<int, greater<int>,
rc_binomial_heap_tag> pq;
    auto it = pq.push(1);  // 1
    pq.push(10);           // 1 10
    assert(pq.top() == 1);
    pq.modify(it, 20);  // 10 20
    assert(pq.top() == 10);
    pq.erase_if([](int x) { return x <= 10; });  // 20
    assert(pq.top() == 20);
    assert(pq.size() == 1);
    pq.erase(it);  // empty
    assert(pq.empty());

    // access, insert, erase: O(log n)
    rope<int> v;
    v.insert(0, 1);  // 1
    v.insert(0, 2);  // 2 1
    v.insert(1, 3);  // 2 3 1
    v.insert(2, 4);  // 2 3 4 1
    v.erase(1, 2);   // 2 1
    assert(v.size() == 2);
    assert(v[0] == 2);
    assert(v[1] == 1);

    return 0;
}
```

priority_queue.md

	push	pop	modify	erase	join
std::priority_queue	最悪$\Theta(n)$, 償却$\Theta(\log(n))$	最悪$\Theta(\log(n))$	最悪$\Theta(n\log(n))$	$\Theta(n\log(n))$	$\Theta(n\log$
priority_queue	$O(1)$	最悪$\Theta(n)$, 償却$\Theta(\log n)$	最悪$\Theta(n)$, 償却$\Theta(\log(n))$	最悪$\Theta(n)$, 償却$O(\log(n))$	O(1)
priority_queue	最悪$\Theta(n)$, 償却$O(\log(n))$	最悪$\Theta(n)$, 償却$O(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
priority_queue	最悪$\Theta(n)$, 償却$O(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log($
priority_queue	$O(1)$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log($
priority_queue	$O(1)$	最悪$\Theta(n)$, 償却$O(\log(n))$	最悪$\Theta(\log(n))$, 償却$O(1)$ or 償却$\Theta(\log n)$	最悪$\Theta(n)$, 償却$O(\log(n))$	$\Theta(n)$