,,

인공지능 사용 적발 시 실격 처리됩니다

Donghyeon Son, Jeyeon Si, Yongjun Lee

July, 2025

Contents

Data structures (1)

1.1 GCC Extension Data Structures

OrderStatisticTree.h
<div align="right">22 lines</div>

```cpp
#include <bits/extc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> set_t;

void test()
{
    set_t X;
    // after insert 1, 2, 4, 8, 16
    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;   // 0
    cout<<X.order_of_key(1)<<endl;    // 0
    cout<<X.order_of_key(3)<<endl;    // 2
    cout<<X.order_of_key(4)<<endl;    // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

HashMap.h
<div align="right">10 lines</div>

```cpp
#pragma once
#pragma GCC optimize ("O3")
#pragma GCC target ("avx2")
#include <bits/extc++.h> /** keep-include */
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = ll(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

1.2 Interval Tree structures

LazySegmentTree.h
Description: Increment modifications, queries for maximum
Time: $\mathcal{O}(\log N)$
<div align="right">44 lines</div>

```cpp
int h = sizeof(int) * 8 - __builtin_clz(n);
int d[N];
void apply(int p, int value) {
    t[p] += value;
    if (p < n) d[p] += value;
}

void build(int p) {
    while (p > 1) p >>= 1, t[p] = max(t[p<<1], t[p<<1|1]) + d[p];
}
```

```cpp
void push(int p) {
    for (int s = h; s > 0; --s) {
        int i = p >> s;
        if (d[i] != 0) {
            apply(i<<1, d[i]);
            apply(i<<1|1, d[i]);
            d[i] = 0;
        }
    }
}

void inc(int l, int r, int value) {
    l += n, r += n;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) apply(l++, value);
        if (r&1) apply(--r, value);
    }
    build(l0);
    build(r0 - 1);
}

int query(int l, int r) {
    l += n, r += n;
    push(l);
    push(r - 1);
    int res = -2e9;
    for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) res = max(res, t[l++]);
        if (r&1) res = max(t[--r], res);
    }
    return res;
}
```

LazySegAssignment.h
Description: Lazy segtree with assignment modifications, sum queries. Parameter k stands for the length of the interval corrsponding to node p.
<div align="right">51 lines</div>

```cpp
void calc(int p, int k) {
    if (d[p] == 0) t[p] = t[p<<1] + t[p<<1|1]; // Assumes that 0
        is never used for modification
    else t[p] = d[p] * k;
}

void apply(int p, int value, int k) {
    t[p] = value * k;
    if (p < n) d[p] = value;
}

void build(int l, int r) {
    int k = 2;
    for (l += n, r += n-1; l > 1; k <<= 1) {
        l >>= 1, r >>= 1;
        for (int i = r; i >= l; --i) calc(i, k);
    }
}

void push(int l, int r) {
    int s = h, k = 1 << (h-1);
    for (l += n, r += n-1; s > 0; --s, k >>= 1)
        for (int i = l >> s; i <= r >> s; ++i) if (d[i] != 0) {
            apply(i<<1, d[i], k);
            apply(i<<1|1, d[i], k);
            d[i] = 0;
        }
```

```cpp
}

void modify(int l, int r, int value) {
  if (value == 0) return;
  push(l, l + 1);
  push(r - 1, r);
  int l0 = l, r0 = r, k = 1;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1, k <<= 1) {
    if (l&1) apply(l++, value, k);
    if (r&1) apply(--r, value, k);
  }
  build(l0, l0 + 1);
  build(r0 - 1, r0);
}

int query(int l, int r) {
  push(l, l + 1);
  push(r - 1, r);
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}
```

PersistentSegmentTree.h
Description: Interval incremental modification, interval sum query on $[l, r]$.
Time: $\mathcal{O}(\log N)$
                                                                66 lines

```cpp
struct Node { int l, r; ll val; };

struct PST {
  vector<Node> tree;
  vector<int> root; // 1-index (0: dummy)
  int sz;

  PST() = default;
  PST(int sz): sz(sz) {
      tree.resize(1);
      root.resize(1);
  }
  // Promote 'c' as the new root, inheriting 'p''s subtree
  void newRoot(int c, int p) {
      assert(p < root.size());
      while (root.size() <= c) root.push_back(0);
      root[c] = tree.size();
    tree.push_back(Node{tree[root[p]]});
      return;
  }
  void update(int node, int start, int end, int index, ll val)
      {
    if (index < start || index > end) return;
    tree[node].val += val;
    if (start == end) return;
    int mid = (start + end) / 2;
    if (index <= mid) {
      int lidx = tree[node].l;
      tree[node].l = tree.size();
      tree.push_back(Node{tree[lidx]});
      update(tree[node].l, start, mid, index, val);
    }
    if (index > mid) {
      int ridx = tree[node].r;
      tree[node].r = tree.size();
      tree.push_back(Node{tree[ridx]});
      update(tree[node].r, mid + 1, end, index, val);
    }
  }
```

```cpp
  ll query(int node, int start, int end, int left, int right) {
    if (left > end || right < start) return 0;
    if (left <= start && end <= right) return tree[node].val;
      int mid = (start + end) / 2;
    ll lsum = query(tree[node].l, start, mid, left, right);
    ll rsum = query(tree[node].r, mid + 1, end, left, right);
    return lsum + rsum;
  }
  int kth(int l, int r, int start, int end, ll k) {
      if (start == end) return start;
      ll tmp = tree[tree[r].l].val - tree[tree[l].l].val;
      int mid = (start + end) / 2;
      if (tmp >= k) return kth(tree[l].l, tree[r].l, start,
          mid, k);
      else return kth(tree[l].r, tree[r].r, mid + 1, end, k -
          tmp);
  }
  void update(int rt, int index, ll val) {
      assert(rt < root.size());
      update(root[rt], 0, sz - 1, index, val);
  }
  ll query(int rt, int left, int right) {
      assert(rt < root.size());
      return query(root[rt], 0, sz - 1, left, right);
  }
  int kth(int l, int r, int k) {
      assert(l - 1 < root.size() && r < root.size());
      return kth(root[l - 1], root[r], 0, sz - 1, k);
  }
};
```

LazySegRecursive.h
<bits/stdc++.h>   78 lines

```cpp
using namespace std;
typedef long long ll;

struct LazyPropagation {
  vector<ll> arr;
  vector<ll> tree;
  vector<ll> lazy;
  ll sz;

  void init(vector<ll>& a, ll sz_) {
    sz = sz_;
    arr = a;
    ll h = (ll)ceil(log2(sz));
    ll tree_size = (1 << (h + 1));
    tree = vector<ll>(tree_size);
    lazy = vector<ll>(tree_size);
    init_(1, 0, sz - 1);
  }
  void init_(ll node, ll start, ll end) {
    if (start == end) tree[node] = arr[start];
    else {
      init_(node * 2, start, (start + end) / 2);
      init_(node * 2 + 1, (start + end) / 2 + 1, end);
      tree[node] = tree[node * 2] + tree[node * 2 + 1];
    }
  }
  void update_lazy(ll node, ll start, ll end) {
    if (lazy[node] != 0) {
      tree[node] += (end - start + 1) * lazy[node];
      if (start != end) {
        lazy[node * 2] += lazy[node];
        lazy[node * 2 + 1] += lazy[node];
      }
      lazy[node] = 0;
    }
  }
```

```cpp
  void update_range(ll node, ll start, ll end, ll left, ll
      right, ll diff) {
    update_lazy(node, start, end);
    if (left > end || right < start) {
      return;
    }
    if (left <= start && end <= right) {
      tree[node] += (end - start + 1) * diff;
      if (start != end) {
        lazy[node * 2] += diff;
        lazy[node * 2 + 1] += diff;
      }
      return;
    }
    update_range(node * 2, start, (start + end) / 2, left,
        right, diff);
    update_range(node * 2 + 1, (start + end) / 2 + 1, end, left
        , right, diff);
    tree[node] = tree[node * 2] + tree[node * 2 + 1];
  }
  void update(ll left, ll right, ll val) {
    update_range(1, 0, sz - 1, left, right, val);
  }
  ll query(ll node, ll start, ll end, ll left, ll right) {
    update_lazy(node, start, end);
    if (left > end || right < start) {
      return 0;
    }
    if (left <= start && end <= right) {
      return tree[node];
    }
    ll lsum = query(node * 2, start, (start+end) / 2, left,
        right);
    ll rsum = query(node * 2 + 1, (start+end) / 2 + 1, end,
        left, right);
    return lsum + rsum;
  }
  ll query(ll left, ll right) {
    return query(1, 0, sz - 1, left, right);
  }
} seg;

void solve() {
  ll MAX_N;
  vector<ll> a(MAX_N);
    seg.init(a, MAX_N);
}
```

2DSegmentTree.h
Description: Compute sum of rectangle $[a, b) \times [c, d)$ and point modification
Time: Both operations are $\mathcal{O}(\log^2 N)$.
                                                                65 lines

```cpp
auto gif = [](int a, int b) { return a + b; };

class SEG2D {
public:
    int n;
    int m;
    vector<vector<int>> tree;
    SEG2D(int n = 0, int m = 0) {
        tree.resize(2 * n);
        for (int i = 0; i < 2 * n; i++) tree[i].resize(2 * m);
        this->n = n;
        this->m = m;
    }
    SEG2D(int n, int m, vector<vector<int>> &data) {
        tree.resize(2 * n);
        for (int i = 0; i < 2 * n; i++) tree[i].resize(2 * m);
        this->n = n;
```

```cpp
        this->m = m;
        init(data);
    }
    void init(vector<vector<int>> &data) {
        n = data.size();
        m = data.front().size();
        tree = vector<vector<int>>(2 * n, vector<int>(2 * m, 0)
            );
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                tree[i + n][j + m] = data[i][j];
        for (int i = n; i < 2 * n; i++)
            for (int j = m - 1; j > 0; j--)
                tree[i][j] = gif(tree[i][j * 2], tree[i][j * 2
                    + 1]);
        for (int i = n - 1; i > 0; i--)
            for (int j = 1; j < 2 * m; j++)
                tree[i][j] = gif(tree[i * 2][j], tree[i * 2 +
                    1][j]);
    }
    void update(int x, int y, int val) {
        tree[x + n][y + m] = val;
        for (int i = y + m; i > 1; i /= 2)
            tree[x + n][i / 2] = gif(tree[x + n][i], tree[x + n
                ][i ^ 1]);
        for (int i = x + n; i > 1; i /= 2)
            for (int j = y + m; j >= 1; j /= 2)
                tree[i / 2][j] = gif(tree[i][j], tree[i ^ 1][j
                    ]);
    }
    int query_1D(int x, int yl, int yr) {
        int res = 0;
        int u = yl + m, v = yr + m + 1;
        for (; u < v; u /= 2, v /= 2) {
            if (u & 1)
                res = gif(res, tree[x][u++]);
            if (v & 1)
                res = gif(res, tree[x][--v]);
        }
        return res;
    }
    int query_2D(int xl, int xr, int yl, int yr) {
        int res = 0;
        int u = xl + n, v = xr + n + 1;
        for (; u < v; u /= 2, v /= 2) {
            if (u & 1)
                res = gif(res, query_1D(u++, yl, yr));
            if (v & 1)
                res = gif(res, query_1D(--v, yl, yr));
        }
        return res;
    }
};
```

FenwickTree.h
Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.
Time: Both operations are $\mathcal{O}(\log N)$.
<div align="right">22 lines</div>

```cpp
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
```

```cpp
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    //  Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

FenwickTree2d.h
Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j].
Time: $\mathcal{O}(\log^2 N)$.
<div align="right">"FenwickTree.h"   19 lines</div>

```cpp
struct FenwickTree2D {
    vector<vector<int>> bit;
    int n, m;
  FenwickTree2D(int _n, int _m) : n(_n), m(_m) {
    bit.assign(n, vector<int>(m, 0));
  }
    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
            for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
                ret += bit[i][j];
        return ret;
    }
    void add(int x, int y, int delta) {
        for (int i = x; i < n; i = i | (i + 1))
            for (int j = y; j < m; j = j | (j + 1))
                bit[i][j] += delta;
    }
};
```

MergeSortTree.h
Description: greater(s,e,k,1,0,n) returns number of elements strictly greater than k in range $[s, e]$. Pay attention to INTERVAL INCLUSIVENESS!!!
Time: $\mathcal{O}(\log N)$.
<div align="right">38 lines</div>

```cpp
#define MAXN (1<<18)
#define ST (1<<17)
struct merge_sort_tree
{
  vector <int> tree[MAXN];
  int n;
  void construct (vector <int> data)
  {
    n = 1;
    while(n < data.size()) n <<= 1;
    for (int i = 0; i<data.size(); i++)
      tree[i+n] = {data[i]};
    for (int i = data.size(); i<n; i++)
      tree[i+n] = {};
    for (int i = n-1; i>0; i--)
    {
      tree[i].resize(tree[i*2].size()+tree[i*2+1].size());
      for (int p = 0, q = 0, j = 0; j < tree[i].size(); j++)
      {
        if (p == tree[i*2].size() ||
        (q<tree[i*2+1].size() && tree[i*2+1][q]<tree[i*2][p]))
          tree[i][j] = tree[i*2+1][q++];
        else tree[i][j] = tree[i*2][p++];
      }
    }
  }
```

```cpp
    }
    //greater(s,e,k,1,0,n)
    int greater(int s, int e, int k, int node, int ns, int ne)
    {
        if (ne <= s || ns >= e)
            return 0;
        if(s <= ns && ne <= e)
            return tree[node].end() - upper_bound(all(tree[node]), k)
                ;
        int mid = (ns+ne)>>1;
        return greater(s,e,k,node*2,ns,mid) +
            greater(s,e,k,node*2+1,mid,ne);
    }
};
```

1.3 Miscellaneous

UnionFind.h
Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$
<div align="right">21 lines</div>

```cpp
struct DSU
{
    int par[V], sz[V];
    DSU(){init(V);}
    void init(int n) {
        for (int i = 0; i<n; i++)
            par[i] = i, sz[i] = 1;
    }
    int find(int x) {
        return x == par[x] ? x : (par[x] = find(par[x]));
    }
    bool unite(int x, int y) {
        int u=find(x), v=find(y);
        if(u==v) return false;
        if(sz[u]>sz[v]) swap(u, v);
        sz[v]+=sz[u];
        sz[u] = 0;
        par[u] = par[v];
        return true;
    }
};
```

UnionFindRollback.h
Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: int t = uf.time(); ...; uf.rollback(t);
Time: $\mathcal{O}(\log(N))$
<div align="right">21 lines</div>

```cpp
struct RollbackUF {
  vi e; vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i --> t;)
      e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b]; e[b] = a;
    return true;
  }
};
```

Treap.h
Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$
<div align="right">55 lines</div>

```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));
}
```

SparseTableRMQ.h
Description: Computes minimum of a range in $O(1)$ time.
Time: $\mathcal{O}(1)$.
<div align="right">18 lines</div>

```cpp
// update
int st[K + 1][MAXN];

std::copy(array.begin(), array.end(), st[0]);
```

```cpp
for (int i = 1; i <= K; i++)
  for (int j = 0; j + (1 << i) <= N; j++)
    st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i -
      1))]);

// query with [L, R]
int log2_floor(unsigned long long i) {
  return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}

int minimum(int L, int R) {
  int i = log2_floor(R - L + 1);
  return min(st[i][L], st[i][R - (1 << i) + 1]);
}
```

MoQueries.h
Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change step to add/remove the edge (a, c) and remove the initial add call (but keep in).
Time: $\mathcal{O}(N\sqrt{Q})$
<div align="right">49 lines</div>

```cpp
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}

vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) -> void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
                  else { add(c, end); in[c] = 1; } a = c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
}
```

```cpp
  return res;
}
```

Mathematics (2)

2.1 Matrices

Matrix.h
Description: 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 = (A^N) * vec;
<div align="right">26 lines</div>

```cpp
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  vector<T> operator*(const vector<T>& vec) const {
    vector<T> ret(N);
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

GaussianElimination.h
Time: $\mathcal{O}(n^3)$
<div align="right">43 lines</div>

```cpp
const double EPS = 1e-10;
typedef vector<vector<double>> VVD;

// Gauss-Jordan elimination with full pivoting.
// solving systems of linear equations (AX=B)
// INPUT:    a[][] = an n*n matrix
//           b[][] = an n*m matrix
// OUTPUT:   X     = an n*m matrix (stored in b[][])
//           A^{-1} = an n*n matrix (stored in a[][])
// O(n^3)
bool gauss_jordan(VVD& a, VVD& b) {
    const int n = a.size();
    const int m = b[0].size();
    vector<int> irow(n), icol(n), ipiv(n);

    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])
                    ) { pj = j; pk = k; }
        if (fabs(a[pj][pk]) < EPS) return false; // matrix is
            singular
        ipiv[pk]++;
```

```cpp
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        irow[i] = pj;
        icol[i] = pk;

        double c = 1.0 / a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c
                ;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c
                ;
        }
    }
    for (int p = n - 1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][
            icol[p]]);
    }
    return true;
}
```

Determinant.h
Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}\left(N^3\right)$
                                                                    15 lines
```cpp
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;
}
```

SolveLinear.h
Description: 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.
Time: $\mathcal{O}\left(n^2m\right)$
                                                                    38 lines
```cpp
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
```

```cpp
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinearBinary.h
Description: Solves $Ax = b$ over $\mathbb{F}_2$. If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: $\mathcal{O}\left(n^2m\right)$
                                                                    34 lines
```cpp
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
  int n = sz(A), rank = 0, br;
  assert(m <= sz(x));
  vi col(m); iota(all(col), 0);
  rep(i,0,n) {
    for (br=i; br<n; ++br) if (A[br].any()) break;
    if (br == n) {
      rep(j,i,n) if(b[j]) return -1;
      break;
    }
    int bc = (int)A[br]._Find_next(i-1);
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) if (A[j][i] != A[j][bc]) {
      A[j].flip(i); A[j].flip(bc);
    }
    rep(j,i+1,n) if (A[j][i]) {
      b[j] ^= b[i];
      A[j] ^= A[i];
    }
    rank++;
  }

  x = bs();
  for (int i = rank; i--;) {
    if (!b[i]) continue;
    x[col[i]] = 1;
    rep(j,0,i) b[j] ^= A[j][i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h
Description: 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.
Time: $\mathcal{O}\left(n^3\right)$
                                                                    35 lines
```cpp
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
```

```cpp
  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;
  }

  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;
}
```

RankOfMatrix.h
Description: Search for the rank using Gaussian elimination.
Time: $\mathcal{O}\left(n^3\right)$
                                                                    30 lines
```cpp
const double EPS = 1E-9;

int compute_rank(vector<vector<double>> A) {
    int n = A.size();
    int m = A[0].size();

    int rank = 0;
    vector<bool> row_selected(n, false);
    for (int i = 0; i < m; ++i) {
        int j;
        for (j = 0; j < n; ++j) {
            if (!row_selected[j] && abs(A[j][i]) > EPS)
                break;
        }

        if (j != n) {
            ++rank;
            row_selected[j] = true;
            for (int p = i + 1; p < m; ++p)
                A[j][p] /= A[j][i];
            for (int k = 0; k < n; ++k) {
                if (k != j && abs(A[k][i]) > EPS) {
                    for (int p = i + 1; p < m; ++p)
                        A[k][p] -= A[j][p] * A[k][i];
                }
            }
        }
    }
    return rank;
}
```

2.2   FFT, Berlekamp

FastFourierTransform.h
Description: $O(N \log N)$ Polynomial multiplication
Time: $\mathcal{O}(N \log N)$

46 lines

```
#define _USE_MATH_DEFINES

#define sz(v) ((int)(v).size())
#define all(v) (v).begin(),(v).end()
typedef complex<double> base;

void fft(vector <base> &a, bool invert)
{
    int n = sz(a);
    for (int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if (i < j) swap(a[i],a[j]);
    }
    for (int len=2;len<=n;len<<=1){
        double ang = 2*M_PI/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){
            base w(1);
            for (int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert){
        for (int i=0;i<n;i++) a[i] /= n;
    }
}

vector<int> multiply(vector<int>& a, vector<int>& b)
{
    vector<base> fa(all(a)), fb(all(b));
    int n = 1, m = sz(a)+sz(b)-1;
    while (n < m) n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa, false); fft(fb, false);
    for (int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa, true);
    vector<int> ret(m);
    for (int i=0;i<m;i++) ret[i] = fa[i].real()+(fa[i].real()
        >0?0.5:-0.5); // removed casting to int here.. should
        be fine
    return ret;
}
```

NumberTheoreticTransform.h
Description: For NTT. $119*(2^{23})+1 = 998244353$, 원시근: 3. $7*(2^{26})+1 = 469762049$, 원시근: 3. $17*(2^{27})+1 = 3221225473$, 원시근: 5
Time: $\mathcal{O}(N \log N)$

54 lines

```
const ll _A = 119, _B = 23, _P = _A << _B | 1, _X = 3;

ll modpow(ll a, ll b, ll mod) {
  if (b == 0) return 1ll;
  ll half = modpow(a, b / 2, mod);
  half = (half * half) % mod;
  if (b % 2 == 0) return half;
  else return (half * a) % mod;
}
```

```
void fft(vector<ll>& a, bool inv) {
    ll n = a.size();
    for (ll i = 1, j = 0; i < n; i++) {
        ll bit = n >> 1;
        while (!((j ^= bit) & bit)) bit >>= 1;
        if (i < j) swap(a[i], a[j]);
    }
    for (ll i = 1; i < n; i <<= 1) {
        ll w = modpow(inv ? modpow(_X, _P - 2, _P) : _X, _P / i
            >> 1, _P);
        for (ll j = 0; j < n; j += i << 1) {
            ll th = 1ll;
            for (ll k = 0; k < i; k++) {
                ll tmp = a[i + j + k] * th % _P;
                a[i + j + k] = a[j + k] - tmp;
                if (a[i + j + k] < 0) a[i + j + k] += _P;
                a[j + k] += tmp;
                if (a[j + k] >= _P) a[j + k] -= _P;
                th = th * w % _P;
            }
        }
    }
    if (inv) {
        ll ninv = modpow(n, _P - 2, _P);
        for (ll i = 0; i < n; i++) a[i] = a[i] * ninv % _P;
    }
}

vector<ll> multiply(vector<ll>& A, vector<ll>& B) {
    vector<ll> a(A.begin(), A.end()), b(B.begin(), B.end());
    ll n = max(a.size(), b.size());
    ll i = 0;
    while ((1 << i) < (n << 1)) i++;
    n = 1 << i;
    a.resize(n);
    b.resize(n);
    fft(a, false);
    fft(b, false);
    for (ll i = 0; i < n; i++) a[i] = a[i] * b[i] % _P;
    fft(a, true);
    vector<ll> ret;
    for (auto i: a) ret.push_back(i);
    while (!ret.empty() && ret.back() == 0) ret.pop_back();
    return ret;
}
```

FWHT.h
Description: Computes XOR convolution: $\{\mathbf{a} \star \mathbf{b}\}_i = \sum_{j \oplus k = i} a_j b_k$

<bits/stdc++.h>

33 lines

```
using namespace std;
typedef long long ll;

void fwht_xor(vector<ll> &a, bool inv) {
    ll n = a.size();
    for(ll s = 2, h = 1; s <= n; s <<= 1, h <<= 1) {
        for(ll l = 0; l < n; l += s) {
            for(ll i = 0; i < h; i++) {
                ll t = a[l + h + i];
                a[l + h + i] = a[l + i] - t;
                a[l + i] += t;
                if(inv) a[l + h + i] /= 2, a[l + i] /= 2;
            }
        }
    }
}

vector<ll> convolute(vector<ll>& A, vector<ll>& B) {
    vector<ll> a(A.begin(), A.end()), b(B.begin(), B.end());
    ll n = max(a.size(), b.size());
```

```
    ll i = 0;
    while ((1 << i) < (n << 1)) i++;
    n = 1 << i;
    a.resize(n);
    b.resize(n);
    fwht_xor(a, false);
    fwht_xor(b, false);
    for (ll i = 0; i < n; i++) {
        a[i] *= b[i];
    }
    fwht_xor(a, true);
    return a;
}
```

BerlekampMassey.h
Description: Find linear recurrence when 3n terms are given.
Usage: guess_nth_term({1, 1, 2, 3, 5, 8}, 10000000);
Time: $\mathcal{O}(N^2)$

116 lines

```
struct Berlekamp_Massey
{
  const int mod = 1000000007;
  using lint = long long;
  lint ipow(lint x, lint p){
    lint ret = 1, piv = x;
    while(p){
      if(p & 1) ret = ret * piv % mod;
      piv = piv * piv % mod;
      p >>= 1;
    }
    return ret;
  }
  vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i=0; i<x.size(); i++){
      lint t = 0;
      for(int j=0; j<cur.size(); j++){
        t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
      }
      if((t - x[i]) % mod == 0) continue;
      if(cur.empty()){
        cur.resize(i+1);
        lf = i;
        ld = (t - x[i]) % mod;
        continue;
      }
      lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
      vector<int> c(i-lf-1);
      c.push_back(k);
      for(auto &j : ls) c.push_back(-j * k % mod);
      if(c.size() < cur.size()) c.resize(cur.size());
      for(int j=0; j<cur.size(); j++){
        c[j] = (c[j] + cur[j]) % mod;
      }
      if(i-lf+(int)ls.size()>=(int)cur.size()){
        tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
      }
      cur = c;
    }
    for(auto &i : cur) i = (i % mod + mod) % mod;
    return cur;
  }
  int get_nth(vector<int> rec, vector<int> dp, lint n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
```

```
auto mul = [&rec](vector<int> v, vector<int> w){
  int m = v.size();
  vector<int> t(2 * m);
  for(int j=0; j<m; j++){
    for(int k=0; k<m; k++){
      t[j+k] += 1ll * v[j] * w[k] % mod;
      if(t[j+k] >= mod) t[j+k] -= mod;
    }
  }
  for(int j=2*m-1; j>=m; j--){
    for(int k=1; k<=m; k++){
      t[j-k] += 1ll * t[j] * rec[k-1] % mod;
      if(t[j-k] >= mod) t[j-k] -= mod;
    }
  }
  t.resize(m);
  return t;
};
while(n){
  if(n & 1) s = mul(s, t);
  t = mul(t, t);
  n >>= 1;
}
lint ret = 0;
for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
return ret % mod;
}
int guess_nth_term(vector<int> x, lint n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  if(v.empty()) return 0;
  return get_nth(v, x, n);
}
struct elem{int x, y, v;};
vector<int> get_min_poly(int n, vector<elem> M)
{
  vector<int> rnd1, rnd2;
  mt19937 rng(0x14004);
  auto randint = [&rng](int lb, int ub){
    return uniform_int_distribution<int>(lb, ub)(rng);
  };
  for(int i=0; i<n; i++){
    rnd1.push_back(randint(1, mod - 1));
    rnd2.push_back(randint(1, mod - 1));
  }
  vector<int> gobs;
  for(int i=0; i<2*n+2; i++){
    int tmp = 0;
    for(int j=0; j<n; j++){
      tmp += 1ll * rnd2[j] * rnd1[j] % mod;
      if(tmp >= mod) tmp -= mod;
    }
    gobs.push_back(tmp);
    vector<int> nxt(n);
    for(auto &i : M){
      nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
      if(nxt[i.x] >= mod) nxt[i.x] -= mod;
    }
    rnd1 = nxt;
  }
  auto sol = berlekamp_massey(gobs);
  reverse(sol.begin(), sol.end());
  return sol;
}
// Usage : guess_nth_term(first_values, n);
};
```

Number Theory (3)

3.1 Primes

< 10^k	prime	# of prime
1	7	4
2	97	25
3	997	168
4	9973	1229
5	99991	9592
6	999983	78498
7	9999991	664579
8	99999989	5761455
9	999999937	50847534

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^{\times}$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

3.2 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

3.3 Modular arithmetic

ModInverse.h
Description: Pre-computation of modular inverses. Assumes LIM ≤ mod and that mod is a prime.
<div align="right">3 lines</div>

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

ModPow.h
<div align="right">8 lines</div>

```
const int mod = 16769023; // faster if const

int modpow(int b, int e) {
  int ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h
Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}(\sqrt{m})$
<div align="right">11 lines</div>

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModSum.h
Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) $= \sum_{i=0}^{to-1} (ki+c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant.
<div align="right">16 lines</div>

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModSqrt.h
Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}(\log^2 p)$ worst case, $\mathcal{O}(\log p)$ for most p
"ModPow.h"
<div align="right">24 lines</div>

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

3.4 Primality

FastEratosthenes.h
Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 ≈ 1.5s
<div align="right">20 lines</div>

```
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;
}
```

MillerRabin.h
Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \mod c$.
"ModMulLL.h"   12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
    s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {    // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

Factor.h
Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.
"ModMulLL.h", "MillerRabin.h"   18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 0, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

3.5 Divisibility

3.5.1 Bézout's identity
For $a \neq$, $b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

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

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

```
  return x = 1, y = 0, a;
}
```

CRT.h
Description: Chinese Remainder Theorem.
crt(a, m, b, n) computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$
"euclid.h"   7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

3.6 Fractions

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

```
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
      a = (ll)floor(y), b = min(a, lim),
      NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that gives us a
      // 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;
  }
}
```

FracBinarySearch.h
Description: Given f and N, finds the smallest fraction $p/q \in [0, 1]$ such that $f(p/q)$ is true, and $p, q \le N$.
Usage: fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}
Time: $\mathcal{O}(\log(N))$
25 lines

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
  bool dir = 1, A = 1, B = 1;
  Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
  if (f(lo)) return lo;
  assert(f(hi));
  while (A || B) {
    ll adv = 0, step = 1; // move hi if dir, else lo
    for (int si = 0; step; (step *= 2) >>= si) {
      adv += step;
      Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
      if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
        adv -= step; si = 2;
```

```
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

3.7 Mobius / Dirichlet

Dirichlet Convolution   For $f, g : \mathbb{N} \to \mathbb{C}$, we define
$$(f * g)(n) = \sum_{d | n} f(d)g(n/d) = \sum_{ab=n} f(a)g(b)$$

Set of arithmetic functions form a commutative ring under pointwise addition and Dirichlet convolution, equipped with the multiplicative identity $\epsilon(n) = \chi_{\{1\}}(n)$. In this ring, we have $\mathbf{1} * \mu = \epsilon$ for the Mobius function μ:

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

And also sum of divisor function $\sigma = id * \mathbf{1}$, number of divisors $d = 1 * 1$ gives $id = \sigma * \mu, \mathbf{1} = d * \mu$.

Finally, the Mobius inversion formula
$$g = f * \mathbf{1} \iff f = g * \mu$$
$$g(n) = \sum_{d|n} f(d) \iff f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Xudyh's Sieve   (1) For a multiplicative function f ($f(ab) = f(a)f(b)$ when $\gcd(a, b) = 1$), s_f can be computed in linear time (2) There exists a function g such that, s_g and s_{f*g} can be evaluated super fast.

XudyhSieve.h
Description: Sums of mod'ed arithmetic progressions.
Prefix sum of multiplicative functions : p_f : the prefix sum of $f(x)$ ($1 <= x <= $ th). p_g : the prefix sum of $g(x)$ ($0 <= x <= N$). p_c : the prefix sum of $f * g(x)$ ($0 <= x <= N$). th : the thereshold, generally should be $n^{(2/3)}$
31 lines

```
struct prefix_mul {

  typedef long long (*func) (long long);

  func p_f, p_g, p_c;
  long long n, th;
  std::unordered_map <long long, long long> mem;

  prefix_mul (func p_f, func p_g, func p_c) : p_f (p_f), p_g (
    p_g), p_c (p_c) {}

  long long calc (long long x) {
    if (x <= th) return p_f (x);
    auto d = mem.find (x);
    if (d != mem.end ()) return d -> second;
    long long ans = 0;
    for (long long i = 2, la; i <= x; i = la + 1) {
      la = x / (x / i);
      ans = ans + (p_g (la) - p_g (i - 1) + mod) * calc (x / i)
        ;
    }
    ans = p_c (x) - ans; ans = ans / inv;
    return mem[x] = ans;
  }
}
```

```
  long long solve (long long n, long long th) {
    if (n <= 0) return 0;
    prefix_mul::n = n; prefix_mul::th = th;
    inv = p_g (1);
    return calc (n);
  }
};
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h
<div align="right">17 lines</div>

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h
Description: Finds the real roots to a polynomial.
Usage: polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0
Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$
<div align="right">"Polynomial.h"   23 lines</div>

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

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

Time: $\mathcal{O}\left(n^2\right)$
<div align="right">13 lines</div>

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

4.2 Optimization

IntegrateAdaptive.h
Description: Fast integration using an adaptive Simpson's rule.
Usage: double sphereVolume = quad(-1, 1, [](double x) {
return quad(-1, 1, [&](double y) {
return quad(-1, 1, [&](double z) {
return x*x + y*y + z*z < 1; });});});
<div align="right">15 lines</div>

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6

template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

Simplex.h
Description: 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);
Time: $\mathcal{O}\left(NM * \#pivots\right)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}\left(2^n\right)$ in the general case.
<div align="right">68 lines</div>

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
    rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
```

```
    rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
    rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
  }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

Combinatorics (5)

5.1 Permutations

Cycles   Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

Derangement   Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n - 1)(D(n - 1) + D(n - 2)) = nD(n - 1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

Burnside's lemma Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x=x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G=\mathbb{Z}_n$ to get

$$g(n)=\frac{1}{n}\sum_{k=0}^{n-1}f(\gcd(n,k))=\frac{1}{n}\sum_{k|n}f(k)\phi(n/k).$$

5.2 Partitions and subsets
5.2.1 Partition function
Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.
$$p(0)=1,\ p(n)=\sum_{k\in\mathbb{Z}\setminus\{0\}}(-1)^{k+1}p(n-k(3k-1)/2)$$
$$p(n)\sim 0.145/n\cdot\exp(2.56\sqrt{n})$$

5.2.2 Lucas' Theorem
Let n,m be non-negative integers and p a prime. Write $n=n_kp^k+...+n_1p+n_0$ and $m=m_kp^k+...+m_1p+m_0$. Then $\binom{n}{m}\equiv\prod_{i=0}^{k}\binom{n_i}{m_i}\pmod p$.

5.2.3 Binomials
multinomial.h
Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n}=\frac{(\sum k_i)!}{k_1!k_2!...k_n!}$.
<div align=right>6 lines</div>

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

5.3 Miscellaneous Sequences
5.3.1 Bernoulli numbers
EGF of Bernoulli numbers is $B(t)=\frac{t}{e^t-1}$ (FFT-able).
$B[0,\ldots]=[1,-\frac12,\frac16,0,-\frac1{30},0,\frac1{42},\ldots]$

Sums of powers:
$$\sum_{i=1}^n n^m=\frac{1}{m+1}\sum_{k=0}^m\binom{m+1}{k}B_k\cdot(n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:
$$\sum_{i=m}^\infty f(i)=\int_m^\infty f(x)dx-\sum_{k=1}^\infty\frac{B_k}{k!}f^{(k-1)}(m)$$
$$\approx\int_m^\infty f(x)dx+\frac{f(m)}{2}-\frac{f'(m)}{12}+\frac{f'''(m)}{720}+O(f^{(5)}(m))$$

5.3.2 Stirling numbers of the first kind
Number of permutations on n items with k cycles.
$$c(n,k)=c(n-1,k-1)+(n-1)c(n-1,k),\ c(0,0)=1$$
$$\sum_{k=0}^n c(n,k)x^k=x(x+1)\ldots(x+n-1)$$

$c(8,k)=8,0,5040,13068,13132,6769,1960,322,28,1$
$c(n,2)=0,0,1,3,11,50,274,1764,13068,109584,\ldots$

5.3.3 Eulerian numbers
Number of permutations $\pi\in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j)>\pi(j+1)$, $k+1$ j:s s.t. $\pi(j)\ge j$, k j:s s.t. $\pi(j)>j$.
$$E(n,k)=(n-k)E(n-1,k-1)+(k+1)E(n-1,k)$$
$$E(n,0)=E(n,n-1)=1$$
$$E(n,k)=\sum_{j=0}^k(-1)^j\binom{n+1}{j}(k+1-j)^n$$

5.3.4 Stirling numbers of the second kind
Partitions of n distinct elements into exactly k groups.
$$S(n,k)=S(n-1,k-1)+kS(n-1,k)$$
$$S(n,1)=S(n,n)=1$$
$$S(n,k)=\frac{1}{k!}\sum_{j=0}^k(-1)^{k-j}\binom{k}{j}j^n$$

5.3.5 Bell numbers
Total number of partitions of n distinct elements. $B(n)=1,1,2,5,15,52,203,877,4140,21147,\ldots$. For p prime,
$$B(p^m+n)\equiv mB(n)+B(n+1)\pmod p$$

5.3.6 Labeled unrooted trees
on n vertices: n^{n-2}
on k existing trees of size n_i: $n_1n_2\cdots n_kn^{k-2}$
with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

5.3.7 Catalan numbers
$$C_n=\frac{1}{n+1}\binom{2n}{n}=\binom{2n}{n}-\binom{2n}{n+1}=\frac{(2n)!}{(n+1)!n!}$$
$$C_0=1,\ C_{n+1}=\frac{2(2n+1)}{n+2}C_n,\ C_{n+1}=\sum C_iC_{n-i}$$
$C_n=1,1,2,5,14,42,132,429,1430,4862,16796,58786,\ldots$
- sub-diagonal monotone paths in an $n\times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (6)
6.1 Theorems
Konig's Theorem In any bipartite graph,
$|\text{max flow}|=|\text{max matching}|=|\text{min vertex cover}|$. Also,
$V-|\text{max indep set}|$.

In any graph without isolated vertices,
$|\text{min edge cover}|+|\text{max matching}|=V$

$\Rightarrow$ For bipartite graph of no isolated edge,
$|\text{min edge cover}|=|\text{max indep set}|$

Flow complexity When all edges are unit capacities (or more generally for all intermediate vertices, there are one in/out edge of unit capacity), Dinic's algorithm runs in $O(E\min(V^{2/3},E^{1/2}))$

6.2 Trees
LCA.h
Description: LCA in $O(N\log N+Q\log N)$
<div align=right>65 lines</div>

```
int n, k;
bool visited[101010];
int par[101010][21], maxedge[101010][21], minedge[101010][21];
int d[101010];
vector <pii> graph[101010]; // {destination, weight}
void dfs(int here, int depth) { // run dfs(root,0)
    visited[here] = true;
    d[here] = depth;
    for (auto there: graph[here]) {
        if (visited[there.first])
            continue;
        dfs(there.first, depth + 1);
        par[there.first][0] = here;
        maxedge[there.first][0] = there.second;
        minedge[there.first][0] = there.second;
    }
}

void precomputation() {
    for (int i = 1; i < 21; i++) {
        for (int j = 1; j <= n; j++) {
            par[j][i] = par[par[j][i - 1]][i - 1];
            maxedge[j][i] = max(maxedge[j][i - 1],
                            maxedge[par[j][i - 1]][i - 1]);
            minedge[j][i] = min(minedge[j][i - 1],
                            minedge[par[j][i - 1]][i - 1]);
        }
    }
}

pii lca(int x, int y) {
    int maxlen = INT_MIN;
    int minlen = INT_MAX;
    if (d[x] > d[y])
        swap(x, y);
    for (int i = 20; i >= 0; i--) {
        if (d[y] - d[x] >= (1 << i)) {
            minlen = min(minlen, minedge[y][i]);
            maxlen = max(maxlen, maxedge[y][i]);
            y = par[y][i];
        }
    }
    if (x == y)
        return {minlen, maxlen};
    // x is lca point
    for (int i = 20; i >= 0; i--) {
        if (par[x][i] != par[y][i]) {
```

```
            minlen = min(minlen, min(minedge[x][i], minedge[y][
                i]));
            maxlen = max(maxlen, max(maxedge[x][i], maxedge[y][
                i]));
            x = par[x][i];
            y = par[y][i];
        }
    }
    minlen = min(minlen, min(minedge[x][0], minedge[y][0]));
    maxlen = max(maxlen, max(maxedge[x][0], maxedge[y][0]));

    int lca_point = par[x][0];
    return {minlen, maxlen};
    // lca_point is lca point
}

void lca_construction() {
    dfs(1, 0);
    precomputation();
}
```

HLD.h
Description: 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.
Time: $\mathcal{O}\left((\log N)^2\right)$

<div align="right">72 lines</div>

```
const int N = 2e5+5;
const int D = 19;
const int S = (1<<D);

int n, q, v[N];
vector<int> adj[N];

int sz[N], p[N], dep[N];
int st[S], id[N], tp[N];

void update(int idx, int val) {
  st[idx += n] = val;
  for (idx /= 2; idx; idx /= 2)
    st[idx] = max(st[2 * idx], st[2 * idx + 1]);
}

int query(int lo, int hi) {
  int ra = 0, rb = 0;
  for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
    if (lo & 1)
      ra = max(ra, st[lo++]);
    if (hi & 1)
      rb = max(rb, st[--hi]);
  }
  return max(ra, rb);
}

int dfs_sz(int cur, int par) {
  sz[cur] = 1;
  p[cur] = par;
  for(int chi : adj[cur]) {
    if(chi == par) continue;
    dep[chi] = dep[cur] + 1;
    p[chi] = cur;
    sz[cur] += dfs_sz(chi, cur);
  }
  return sz[cur];
}

int ct = 1;

void dfs_hld(int cur, int par, int top) {
```

```
  id[cur] = ct++;
  tp[cur] = top;
  update(id[cur], v[cur]);
  int h_chi = -1, h_sz = -1;
  for(int chi : adj[cur]) {
    if(chi == par) continue;
    if(sz[chi] > h_sz) {
      h_sz = sz[chi];
      h_chi = chi;
    }
  }
  if(h_chi == -1) return;
  dfs_hld(h_chi, cur, top);
  for(int chi : adj[cur]) {
    if(chi == par || chi == h_chi) continue;
    dfs_hld(chi, cur, chi);
  }
}

int path(int x, int y){
  int ret = 0;
  while(tp[x] != tp[y]){
    if(dep[tp[x]] < dep[tp[y]])swap(x,y);
    ret = max(ret, query(id[tp[x]],id[x]));
    x = p[tp[x]];
  }
  if(dep[x] > dep[y])swap(x,y);
  ret = max(ret, query(id[x],id[y]));
  return ret;
}
```

LinkCutTree.h
Description: 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.
Time: All operations take amortized $\mathcal{O}(\log N)$.

<div align="right">90 lines</div>

```
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() {
    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() {
    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) {
    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) {
    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;
  }
};
```

DirectedMST.h
Description: Finds a minimum spanning tree/arborescence of a directed
graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

`"../data-structures/UnionFindRollback.h"`

<div align="right">60 lines</div>

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
```

```
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

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

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

CentroidDecomp.h
Description: JusticeHui implementation of Centroid Decomposition   11 lines

```
int sz[101010]; // subtree size
vector<int> g[101010]; // adj list
int getSize(int v, int b = -1){ // get sz
  sz[v] = 1;
  for(auto i : g[v]) if(i != b) sz[v] += getSize(i, v);
  return sz[v];
}
int getCent(int v, int b = -1, int cap = n){ // find centroid
  for(auto i : g[v]) if(&& i != b && sz[i]*2 > cap) return
      getCent(i, v, cap);
```

```
  return v;
}
```

6.3 Strongly Connected Components
SCC.h
Description: Finds sccs in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: build graph only with add_edge. use C.find_scc();
Time: $\mathcal{O}(E + V)$ 37 lines

```
vector<int> adj[10020];
vector<int> revadj[10020];
vector<vector<int>> scc;
stack<int> s;
int visited[10020];
void DFS(int node) {
  visited[node] = 1;
  for(auto N: adj[node]) {
    if(!visited[N]) {
      DFS(N);
    }
  }
  s.push(node);
}
void revDFS(int node, vector<int>& v) {
  visited[node] = 1;
  for(auto N: revadj[node]) {
    if(!visited[N]) {
      revDFS(N, v);
    }
  }
  v.push_back(node);
}
void findSCC() {
  fill(visited, visited+10002, 0);
  while(!s.empty()) {
    int tmp = s.top();
    s.pop();
    if(visited[tmp]) {
      continue;
    }
    vector<int> v;
    revDFS(tmp, v);
    sort(v.begin(), v.end());
    scc.push_back(v);
  }
}
```

BiconnectedComponents.h
Description: 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_back(b, eid);
ed[b].emplace_back(a, eid++); }
bicomps([&](const vi& edgelist) {...});
Time: $\mathcal{O}(E + V)$ 33 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, e, y, top = me;
  for (auto pa : ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
```

```
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}

template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h
    74 lines

```
struct SAT {
    vector<vector<int>> adj;
    vector<int> val, num, low, stk, scc, visiting;
    int nxt, n, cnt;
    int f(int i) {
        return i < 0 ? -i+n : i;
    }
    int neg(int i) {
        return i >= n ? i-n : i+n;
    }
    SAT(int _n, vector<pair<int, int>>& edges): {
        n = _n+1;
        adj.resize(2*n+1);
        val.resize(2*n+1);
        num.resize(2*n+1);
        low.resize(2*n+1);
        scc.resize(2*n+1);
        visiting.resize(2*n+1);
        nxt = 1;
        for(auto [a, b]: edges) {
            a = f(a), b = f(b);
            adj[neg(a)].push_back(b);
            adj[neg(b)].push_back(a);
        }
        compute();
    }
    void compute() {
        for(int i=1; i<=n; i++) {
            if(!num[i])
                dfs(i);
        }
        for(int i=n+1; i<=n+n; i++) {
            if(!num[i])
                dfs(i);
        }
    }
    void dfs(int i) {
        num[i] = low[i] = nxt++;
        visiting[i] = 1;
        stk.push_back(i);
        for(int a: adj[i]) {
```

```
        if(!num[a]) {
            dfs(a);
        }
        if(visiting[a]) {
            low[i] = min(low[i], low[a]);
        }
    }
    if(num[i] == low[i]) {
        int c = cnt++;
        while(1) {
            int t = stk.back();
            stk.pop_back();
            scc[t] = c;
            visiting[t] = 0;
            if(t == i) break;
        }
    }
}
bool ok() {
    for(int i=1; i<=n; i++) {
        if(scc[i] == scc[neg(i)])
            return 0;
    }
    return 1;
}
vector<int> result() {
    vector<int> ans(n+1);
    for(int i=1; i<=n; i++) {
        ans[i] = (scc[i] < scc[neg(i)]);
    }
    return ans;
}
};
```

6.4 Network flow

Dinic.h
Description: Maxflow

74 lines

```
struct Edge {
    int u, v;
    int cap, flow;

    Edge() {}

    Edge(int u, int v, int cap) : u(u), v(v), cap(cap), flow(0)
        {}
};

struct Dinic {
    int N;
    vector <Edge> E;
    vector <vector<int>> g;
    vector<int> d, pt;

    Dinic(int N) : N(N), E(0), g(N), d(N), pt(N) {}

    void AddEdge(int u, int v, int cap) {
        if (u != v) {
            E.push_back(Edge(u, v, cap));
            g[u].push_back(E.size() - 1);
            E.push_back(Edge(v, u, 0));
            g[v].push_back(E.size() - 1);
        }
    }

    bool BFS(int S, int T) {
        queue<int> q({S});
        fill(d.begin(), d.end(), N + 1);
        d[S] = 0;
```

```
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (u == T) break;
            for (int k: g[u]) {
                Edge &e = E[k];
                if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
                    d[e.v] = d[e.u] + 1;
                    q.push(e.v);
                }
            }
        }
        return d[T] != N + 1;
    }

    int DFS(int u, int T, int flow = -1) {
        if (u == T || flow == 0) return flow;
        for (int &i = pt[u]; i < g[u].size(); i++) {
            Edge &e = E[g[u][i]];
            Edge &oe = E[g[u][i] ^ 1];
            if (d[e.v] == d[e.u] + 1) {
                int amt = e.cap - e.flow;
                if (flow != -1 && amt > flow) amt = flow;
                if (int pushed = DFS(e.v, T, amt)) {
                    e.flow += pushed;
                    oe.flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
    }

    int MaxFlow(int S, int T) {
        int total = 0;
        while (BFS(S, T)) {
            fill(pt.begin(), pt.end(), 0);
            while (int flow = DFS(S, T)) {
                total += flow;
            }
        }
        return total;
    }
};
```

MinCostMaxFlow.h
Description: Min cost max flow
Time: $\mathcal{O}(VEf)$ (practically $\mathcal{O}(Ef)$) where f is maximum flow

69 lines

```
struct Edge{
    int u, v, w, cap, flow;
};
const int INF = 0x3f3f3f3f;
struct MCMF {
    int n;
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<int> par, dist, inque;
    MCMF(int _n) {
        n = _n;
        adj.resize(n+1);
        par.resize(n+1);
        dist.resize(n+1);
        inque.resize(n+1);
    }
    void addedge(int u, int v, int cap, int w) {
        adj[u].push_back(edges.size());
        edges.push_back({u, v, w, cap, 0});
        adj[v].push_back(edges.size());
```

```
        edges.push_back({v, u, -w, cap, cap});
    }
    void spfa(int src) {
        fill(par.begin(), par.end(), -1);
        fill(dist.begin(), dist.end(), INF);
        fill(inque.begin(), inque.end(), 0);
        dist[src] = 0;
        queue<int> que({src});
        while(que.size()) {
            int q = que.front();
            que.pop();
            inque[q] = 0;
            for(int i: adj[q]) {
                Edge& e = edges[i];
                if(e.flow < e.cap && dist[e.v] > dist[e.u]+e.w)
                    {
                    dist[e.v] = dist[e.u]+e.w;
                    par[e.v] = i;
                    if(!inque[e.v]) {
                        inque[e.v] = 1;
                        que.push(e.v);
                    }
                }
            }
        }
    }
    pair<int, int> solve(int src, int sink) {
        int mc = 0, mf = 0;
        while(1) {
            spfa(src);
            if(par[sink] == -1)
                return {mc, mf};
            int flow = INF, c = sink;
            while(c != src) {
                Edge &e = edges[par[c]];
                flow = min(flow, e.cap-e.flow);
                c = e.u;
            }
            c = sink;
            while(c != src) {
                Edge &e = edges[par[c]], &ie = edges[par[c]^1];
                e.flow += flow;
                ie.flow -= flow;
                c = e.u;
            }
            mf += flow;
            mc += dist[sink]*flow;
        }
    }
};
```

L-R MaxFlow (1) MCMF solution. For edge (a→b) with bound $[l, r]$, make two edges as $(l, -1), (r - l, 0)$. MCMF forces flow to remaining -1 edges. COST : is there possible lr-flow. FLOW : actual lr-flow. (2) Dinic solution. Make new SRC and SNK. Make (SNK, SRC, INF) edge. For edge (a→b) with bound $[l, r]$, make $(a \rightarrow NEWSNK, l)$, $(NEWSRC \rightarrow b, l)$, $(a \rightarrow b, r - l)$. Is sum of maxflow $NEWSRC \rightarrow NEWSNK$ equal to sum of l? To find actual flow, run dinic again with original SRC-SNK (without resetting).

EdmondsKarp.h
Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

35 lines

```cpp
template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
    graph, int source, int sink) {
  assert(source != sink);
  T flow = 0;
  vi par(sz(graph)), q = par;

  for (;;) {
    fill(all(par), -1);
    par[source] = 0;
    int ptr = 1;
    q[0] = source;

    rep(i,0,ptr) {
      int x = q[i];
      for (auto e : graph[x]) {
        if (par[e.first] == -1 && e.second > 0) {
          par[e.first] = x;
          q[ptr++] = e.first;
          if (e.first == sink) goto out;
        }
      }
    }
    return flow;
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
      inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
      int p = par[y];
      if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
      graph[y][p] += inc;
    }
  }
}
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph. (Stoer-Wagner) MinCut mc; mc.init(n); for (each edge) mc.addEdge(a,b,weight); mincut = mc.solve(); mc.cut = $\{0,1\}^n$ describing which side the vertex belongs to.
Time: $\mathcal{O}\left(V^3\right)$

<div align="right">63 lines</div>

```cpp
struct MinCutMatrix {
    int n;
    vector<vector<int>> graph;

    void init(int _n) {
        n = _n;
        graph = vector<vector<int>>(n, vector<int>(n, 0));
    }
    void addEdge(int a, int b, int w) {
        if (a == b) return;
        graph[a][b] += w;
        graph[b][a] += w;
    }

    pair<int, pair<int, int>> stMinCut(vector<int> &active) {
        vector<int> key(n);
        vector<int> v(n);
        int s = -1, t = -1;
        for (int i = 0; i < active.size(); i++) {
```

```cpp
            int maxv = -1;
            int cur = -1;
            for (auto j : active) {
                if (v[j] == 0 && maxv < key[j]) {
                    maxv = key[j];
                    cur = j;
                }
            }
            t = s; s = cur;
            v[cur] = 1;
            for (auto j : active) key[j] += graph[cur][j];
        }
        return make_pair(key[s], make_pair(s, t));
    }

    vector<int> cut;

    int solve() {
        int res = numeric_limits<int>::max();
        vector<vector<int>> grps;
        vector<int> active;
        cut.resize(n);
        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
        for (int i = 0; i < n; i++) active.push_back(i);
        while (active.size() >= 2) {
            auto stcut = stMinCut(active);
            if (stcut.first < res) {
                res = stcut.first;
                fill(cut.begin(), cut.end(), 0);
                for (auto v : grps[stcut.second.first]) cut[v]
                    = 1;
            }

            int s = stcut.second.first, t = stcut.second.second
                ;
            if (grps[s].size() < grps[t].size()) swap(s, t);

            active.erase(find(active.begin(), active.end(), t))
                ;
            grps[s].insert(grps[s].end(), grps[t].begin(), grps
                [t].end());
            for (int i = 0; i < n; i++) { graph[i][s] += graph[
                i][t]; graph[i][t] = 0; }
            for (int i = 0; i < n; i++) { graph[s][i] += graph[
                t][i]; graph[t][i] = 0; }
            graph[s][s] = 0;
        }
        return res;
    }
};
```

6.5 Matching

hopcroftKarp.h

Description: 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);
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

<div align="right">42 lines</div>

```cpp
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
      return btoa[b] = a, 1;
  }
}
```

```cpp
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        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);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

Hungarian.h

Description: Min-cost matching on bipartite graphs, in $O(n^3)$ time

<div align="right">108 lines</div>

```cpp
int w[N][N];
int match_x[N];
int match_y[N];

int l_x[N], l_y[N];
bool s[N], t[N];
int slack[N];
int slack_x[N];

int tree_x[N];
int tree_y[N];

int hungarian(int n) {
  memset(match_x, -1, sizeof(match_x));
  memset(match_y, -1, sizeof(match_y));
  int ret = 0;

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      l_x[i] = max(l_x[i], w[i][j]);
    }
  }
  memset(l_y, 0, sizeof(l_y));

  int m = 0;
  while (m != n) { // repeat at most V times
    memset(tree_x, -1, sizeof(tree_x));
    memset(tree_y, -1, sizeof(tree_y));
    memset(s, 0, sizeof(s));
    memset(t, 0, sizeof(t));
```

```cpp
int s_start;
for (int i = 0; i < n; ++i) { // O(V)
  if (match_x[i] == -1) {
    s[i] = 1;
    s_start = i;
    break;
  }
}

for (int i = 0; i < n; ++i) { // init slack
  slack[i] = l_x[s_start] + l_y[i] - w[s_start][i];
  slack_x[i] = s_start;
}

here:
int y = -1;
for (int i = 0; i < n; ++i) { // compare: O(V)
  if (slack[i] == 0 && !t[i]) y = i;
}

if (y == -1) { // n_l = t
  // update label
  int alpha = INF;
  for (int i = 0; i < n; ++i) { // O(V)
    if (!t[i]) {
      alpha = min(alpha, slack[i]);
    }
  }
  for (int i = 0; i < n; ++i) { // O(V)
    if (s[i]) l_x[i] -= alpha;
    if (t[i]) l_y[i] += alpha;
  }
  for (int i = 0; i < n; ++i) { // O(V)
    if (!t[i]) {
      slack[i] -= alpha;
      if (slack[i] == 0) {
        y = i;
      }
    }
  }
}

// n_l != t is guaranteed
if (match_y[y] == -1) { // free
  tree_y[y] = slack_x[y];
  while (y != -1) {
    int x = tree_y[y];
    match_y[y] = x;
    int next_y = match_x[x];
    match_x[x] = y;
    y = next_y;
  }
  m++;
}
else { // matched
  int z = match_y[y];
  tree_x[z] = y;
  tree_y[y] = slack_x[y];
  s[z] = 1;
  t[y] = 1;
    // z is added - update slack and n_l
  for (int i = 0; i < n; ++i) { // O(V)
    if (l_x[z] + l_y[i] - w[z][i] < slack[i]) {
      slack[i] = l_x[z] + l_y[i] - w[z][i];
      slack_x[i] = z;
    }
  }
  goto here;
}
```

```cpp
}

  for (int i = 0; i < n; ++i) {
    ret += l_x[i];
    ret += l_y[i];
  }
  return ret;
}
```

DFSMatching.h
Description: 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);
Time: $\mathcal{O}(VE)$
<div align="right">22 lines</div>

```cpp
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h
Description: 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.
"DFSMatching.h"
<div align="right">20 lines</div>

```cpp
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1) {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

WeightedMatching.h
Description: 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.
Time: $\mathcal{O}(N^2M)$
<div align="right">31 lines</div>

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

GeneralMatching.h
Description: Matching for general graphs. Fails with probability N/mod.
Time: $\mathcal{O}(N^3)$
"../numerical/MatrixInverse-mod.h"
<div align="right">40 lines</div>

```cpp
vector<pii> generalMatching(int N, vector<pii>& ed) {
  vector<vector<ll>> mat(N, vector<ll>(N)), A;
  for (pii pa : ed) {
    int a = pa.first, b = pa.second, r = rand() % mod;
    mat[a][b] = r, mat[b][a] = (mod - r) % mod;
  }

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

  if (M != N) do {
    mat.resize(M, vector<ll>(M));
    rep(i,0,N) {
      mat[i].resize(M);
      rep(j,N,M) {
        int r = rand() % mod;
        mat[i][j] = r, mat[j][i] = (mod - r) % mod;
      }
    }
  } while (matInv(A = mat) != M);

  vi has(M, 1); vector<pii> ret;
  rep(it,0,M/2) {
    rep(i,0,M) if (has[i])
      rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
```

```
    fi = i; fj = j; goto done;
  } assert(0); done:
  if (fj < N) ret.emplace_back(fi, fj);
  has[fi] = has[fj] = 0;
  rep(sw,0,2) {
    ll a = modpow(A[fi][fj], mod-2);
    rep(i,0,M) if (has[i] && A[i][fj]) {
      ll b = A[i][fj] * a % mod;
      rep(j,0,M) A[i][j] = (A[i][j] - A[fi][j] * b) % mod;
    }
    swap(fi,fj);
  }
}
  return ret;
}
```

6.6   Heuristics

MaximalCliques.h
Description: 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.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs
                                                                    12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

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

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
      if (sz(T)) {
```

```
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

MaximumIndependentSet.h
Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

6.7   Other Stuff

EulerWalk.h
Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V + E)$
                                                                    15 lines

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

FloydWarshall.h
Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] = \text{inf}$ 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.
Time: $\mathcal{O}(N^3)$
                                                                    12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
```

```
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h
Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$
                                                                    14 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), ret;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  queue<int> q; // use priority queue for lexic. smallest ans.
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(-i);
  while (!q.empty()) {
    int i = -q.front(); // top() for priority queue
    ret.push_back(i);
    q.pop();
    for (int x : gr[i])
      if (--indeg[x] == 0) q.push(-x);
  }
  return ret;
}
```

ArticulationBridge.h
Time: $\mathcal{O}(V + E)$
                                                                    27 lines

```
const int MX = 100000;
vector<int> adj[MX];
int num[MX], low[MX], parent[MX], ind = 1;
bool isarti[MX]; // for articulation point
vector<pair<int, int>> ans; // for articulation bridge
int dfs(int u, bool isroot) {
  num[u] = low[u] = ind++;
  int childcnt = 0;
  for(int a: adj[u]) {
    if(a == parent[u]) continue;
    if(num[a]) {
      low[u] = min(low[u], num[a]);
    } else {
      childcnt++;
      parent[a] = u;
      low[u] = min(low[u], dfs(a, false));
      // for articulation point:
      if(low[a] >= num[u])
        isarti[u] = true;
      // for articulation bridge:
      if(low[a] > num[u])
        ans.emplace_back(min(a,u), max(a,u));
    }
  }
  if(isroot) isarti[u] = (childcnt>=2); // for articulation
      point
  return low[u];
}
```

dijkstra.h
Description: Computes shortest path from a single source vertex to all of the other vertices in a weighted digraph.
Time: $\mathcal{O}(E \log V)$
                                                                    28 lines

```
const int MX = 2e5 + 5;
struct edge {
  int v, w;
```

```cpp
  bool operator<(const edge &p) const{
    return w > p.w;
  }
};
vector<edge> adj[MX];
int dist[MX];

void dijkstra(int s) {
    memset(dist, 0x3f, sizeof dist);
    dist[s] = 0;
    priority_queue<edge> pq;
    pq.push({s, 0});
    while(!pq.empty()) {
      auto t = pq.top();
      int v = t.v, w = t.w;
      pq.pop();
      if(w > dist[v]) continue;
      for(auto p: adj[v]) {
        if(dist[p.v] > dist[v]+p.w) {
          dist[p.v] = dist[v]+p.w;
          pq.push({p.v, dist[p.v]});
        }
      }
    }
}
```

Geometry (7)

7.1 Formulae

Triangles

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

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

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

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

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

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

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

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

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

Quadrilaterals

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

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

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

Spherical coordinates

$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$

$$y = r\sin\theta\sin\phi \qquad \theta = \mathrm{acos}(z/\sqrt{x^2+y^2+z^2})$$

$$z = r\cos\theta \qquad \phi = \mathrm{atan2}(y, x)$$

7.2 Geometric primitives

Point.h

Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)
<div align="right">28 lines</div>

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

lineDistance.h

Description: 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.
"Point.h" <div align="right">4 lines</div>

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

SegmentDistance.h

Description: 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;
"Point.h" <div align="right">6 lines</div>

```cpp
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description: 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;
"Point.h", "OnSegment.h" <div align="right">13 lines</div>

```cpp
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description: 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;
"Point.h" <div align="right">8 lines</div>

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

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument eps is given 0 is returned if p is within distance eps from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: bool left = sideOf(p1,p2,q)==1;
"Point.h" <div align="right">9 lines</div>

```cpp
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

Description: Returns true iff p lies on the line segment from s to e. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.
"Point.h" <div align="right">3 lines</div>

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

linearTransformation.h
Description: Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.
"Point.h"   6 lines

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

LineProjectionReflection.h
Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. 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.
"Point.h"   5 lines

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

Angle.h
Description: 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   35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}

Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}

Angle angleDiff(Angle a, Angle b) { // angle b - angle a
```

```
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

7.3 Circles

CircleIntersection.h
Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.
"Point.h"   11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

CircleTangents.h
Description: 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.
"Point.h"   13 lines

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

CircleLine.h
Description: 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>.
"Point.h"   9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$
"../../content/geometry/Point.h"   19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
```

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

circumcircle.h
Description: The circumcircle 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.
"Point.h"   9 lines

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

MinimumEnclosingCircle.h
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$
"circumcircle.h"   17 lines

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

7.4 Polygons

InsidePolygon.h
Description: 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);
Time: $\mathcal{O}(n)$
"Point.h", "OnSegment.h", "SegmentDistance.h"   11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
```

```
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

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

"Point.h"   8 lines
```
// ! Results can be negative. use abs() if needed.
template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    for (int i = 0; i < v.size()-1; i++)
        a += v[i].cross(v[i+1]);
    return a;
}
```

PolygonCenter.h
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h"   9 lines
```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h
Description: 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));

"Point.h", "lineIntersection.h"   13 lines
```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h
Description: 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.)
Time: $\mathcal{O}(N^2)$, where N is the total number of points

"Point.h", "sideOf.h"   33 lines
```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
```

```
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h
Description: 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.
Time: $\mathcal{O}(n \log n)$

"Point.h"   13 lines
```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h
Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/colinear points).

"Point.h"   12 lines
```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

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

"Point.h", "sideOf.h", "OnSegment.h"   14 lines
```
typedef Point<ll> P;

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

LineHullIntersection.h
Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear 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. extrVertex returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(N + Q \log n)$

"Point.h"   39 lines
```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

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

```
    return res;
}
```

7.5   Misc. Point Set Problems

ClosestPair.h
Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$
"Point.h"   17 lines

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

kdTree.h
Description: KD-tree (2d, can be extended to 3d)
"Point.h"   63 lines

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

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

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

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

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

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

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

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

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

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

FastDelaunay.h
Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], . . .}, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$
"Point.h"   88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
};

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
           new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
  rep(i,0,4)
    q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
  return *q;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
```

```
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

ccw.h
Description: CCW
         7 lines

```
struct point {double x, y;};
int ccw(point a, point b, point c) {
    double v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}
```

IntersectionOfSegment.h
         58 lines

```
const double EPS = 1E-9;
struct pt {
    double x, y;
```

```cpp
    bool operator<(const pt& p) const {
        return x < p.x - EPS || (abs(x - p.x) < EPS && y < p.y
            - EPS);
    }
};
struct line {
    double a, b, c;
    line() {}
    line(pt p, pt q) {
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }
    void norm() {
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS)
            a /= z, b /= z, c /= z;
    }
    double dist(pt p) const { return a * p.x + b * p.y + c; }
};
double det(double a, double b, double c, double d) {
    return a * d - b * c;
}
inline bool betw(double l, double r, double x) {
    return min(l, r) <= x + EPS && x <= max(l, r) + EPS;
}
inline bool intersect_1d(double a, double b, double c, double d
    ) {
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}
bool intersect(pt a, pt b, pt c, pt d, pt& left, pt& right) {
    if (!intersect_1d(a.x, b.x, c.x, d.x) || !intersect_1d(a.y,
        b.y, c.y, d.y))
        return false;
    line m(a, b);
    line n(c, d);
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
        if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
            return false;
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        left = max(a, c);
        right = min(b, d);
        return true;
    } else {
        left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
        left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
        return betw(a.x, b.x, left.x) && betw(a.y, b.y, left.y)
            &&
            betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y)
                ;
    }
}
```

Strings (8)

KMP.h
<div align="right">32 lines</div>

```cpp
vector<int> getPi(string p) {
    int j = 0, plen = p.length();
```

```cpp
    vector<int> pi;
    pi.resize(plen);
    for(int i = 1; i< plen; i++) {
        while((j > 0) && (p[i] != p[j]))
            j = pi[j-1];
        if(p[i] == p[j]) {
            j++;
            pi[i] = j;
        }
    }
    return pi;
}
vector <int> kmp(string s, string p) {
    vector<int> ans;
    auto pi = getPi(p);
    int slen = s.length(), plen = p.length(), j = 0;
    for(int i = 0 ; i < slen ; i++) {
        while(j>0 && s[i] != p[j])
            j = pi[j-1];
        if(s[i] == p[j]) {
            if(j==plen-1) {
                ans.push_back(i-plen+1);
                j = pi[j];
            }
            else
                j++;
        }
    }
    return ans;
}
```

Zfunc.h
Description: z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$
<div align="right">12 lines</div>

```cpp
vi Z(string S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

Manacher.h
<div align="right">17 lines</div>

```cpp
int N, A[MAXN];
char S[MAXN];

void Manachers() {
    int r = 0, p = 0;
    for (int i = 1; i <= N; i++) {
        if (i <= r)
            A[i] = min(A[2*p-i],r-i);
        else
            A[i] = 0;
        while (i-A[i]-1 > 0 && i+A[i]+1 <= N
        && S[i-A[i]-1] == S[i+A[i]+1])
            A[i]++;
        if (r < i+A[i])
            r = i+A[i], p = i;
    }
}
```

MinRotation.h
Description: Finds the lexicographically smallest rotation of a string.
Usage: rotate(v.begin(), v.begin()+minRotation(v), v.end());
Time: $\mathcal{O}(N)$
<div align="right">8 lines</div>

```cpp
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    }
    return a;
}
```

SuffixArray.h
Description: Builds suffix array for a string. Note, that this algorithm only sorts the cycle shifts. We can generate the sorted order of the suffixes by appending a character that is smaller than all other characters of the string, and sorting this resulting string by cycle shifts, e.g. by sorting the cycle shifts of s + $.
<div align="right">72 lines</div>

```cpp
vector<int> sort_cyclic_shifts(string &s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n), cnt(max(alphabet, n)), pn(n), cn(n)
        ;
    // p[i] = locaction of ith string
    for(int i=0; i<n; i++)
        cnt[s[i]]++;
    for(int i=1; i<alphabet; i++)
        cnt[i] += cnt[i-1];
    for(int i=n-1; i>=0; i--)
        p[--cnt[s[i]]] = i;
    // c[i] = equivalence class of position i
    c[p[0]] = 0;
    int classes = 1;
    for(int i=1; i<n; i++) {
        if(s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    for(int h=0; (1<<h) < n; h++) {
        // preprocess pn
        for(int i=0; i<n; i++) {
            pn[i] = p[i] - (1<<h);
            if(pn[i] < 0)
                pn[i] += n;
        }

        // find next p by counting sort stably
        fill(cnt.begin(), cnt.end(), 0);
        for(int i=0; i<n; i++)
            cnt[c[i]]++;
        for(int i=1; i<classes; i++)
            cnt[i] += cnt[i-1];
        for(int i=n-1; i>=0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        // find next c
        cn[p[0]] = 0;
        classes = 1;
        for(int i=1; i<n; i++) {
            auto prv = make_pair(c[p[i]], c[(p[i] + (1<<h)) % n
                ]),
                nxt = make_pair(c[p[i-1]], c[(p[i-1] + (1<<h))
                    % n]);

            if(prv != nxt)
                classes++;
            cn[p[i]] = classes - 1;
```

```
        }
        swap(cn, c);
    }
    return p;
}
vector<int> lcp_construction(string const& s, vector<int> const
    & p) {
    int n = s.size(), k = 0;
    // rank[i] = order of [string on location i]
    vector<int> rank(n), lcp(n-1);
    for(int i=0; i<n; i++)
        rank[p[i]] = i;

    for(int i=0; i<n; i++) {
        if(rank[i] == n-1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while(i+k<n && j+k<n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if(k)
            k--;
    }
    return lcp;
}
```

SuffixTree.h

Description: 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).
Time: $\mathcal{O}(26N)$

<div align="right">50 lines</div>

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m;  l[m]=i;
                p[m++]=v; v=s[v]; q=r[v];  goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i;  p[m+1]=m;  l[m]=l[v];  r[m]=q;
            p[m]=p[v];  t[m][c]=m+1;  t[m][toi(a[q])]=v;
            l[v]=q;  p[v]=m;  t[p[m]][toi(a[l[m]])]=m;
            v=s[p[m]];  q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])];  q+=r[v]-l[v]; }
            if (q==r[m])  s[m]=v;  else s[m]=m+2;
            q=r[v]-(q-r[m]);  m+=2;  goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r,r+N,sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1],t[1]+ALPHA,0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
        rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
    }
```

```
    // example: find longest common substring (uses ALPHA = 28)
    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
        rep(c,0,ALPHA) if (t[node][c] != -1)
            mask |= lcs(t[node][c], i1, i2, len);
        if (mask == 3)
            best = max(best, {len, r[node] - len});
        return mask;
    }
    static pii LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
        st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
        return st.best;
    }
};
```

Hashing.h

Description: Rabin-Karp rolling hash, always use multiple modulos and bases when unsure.

<div align="right">18 lines</div>

```
vector <ll> RabinKarp(string &s, ll window, ll mod, ll p) {
    vector <ll> hash_values;
    ll cur = 0;
    ll hash_offset = 1;
    for (int i = 0; i < window; i++) {
        cur *= p;
        cur += s[i];
        cur %= mod;
        if (i > 0) hash_offset = (hash_offset * p)%mod;
    }
    hash_values.push_back(cur);
    for (int i = window; i < s.length(); i++) {
        cur = (mod + cur - (s[i-window] * hash_offset)%mod)%mod
            ;
        cur = (cur * p + s[i]) % mod;
        hash_values.push_back(cur);
    }
    return hash_values;
}
```

AhoCorasick.h

Description: Each Vertex contains the flag output and the edges in the form of an array next[], where next[i] is the index of the vertex that we reach by following the character i, or −1 if there is no such edge.

<div align="right">53 lines</div>

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
```

```
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}


int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}


int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

DP Optimization (9)

9.1 Convex Hull Trick

점화식이 다음과 같은 형태를 만족할 때,
$$D(i) = \min_{j<i}(D(j) + B(j) \times A(i))$$

이를 직선 $l_j : y = D(j) + B(j)x$ 들을 삽입한 상황에서, $x = A(i)$를 쿼리하여
최솟값을 찾는 문제로 본다. $O(n^2) \to O(n \log n)$ (Linecontainer, Li-Chao Tree),
추가로 $B(j)$ 가 단조성을 갖는 경우 $O(n)$.

CHT.h

Description: insertion queries come in decreasing v.y/v.x order, queries come in increasing x order.

<div align="right">22 lines</div>

```
const int MEM=50003;
struct cht{
    int s=0,e=0,id[MEM];
    pll f[MEM];
    double cross(int a,int b){
        return 1.0*(f[a].y-f[b].y)/(f[b].x-f[a].x);
    }
    void insert(pll v,int l){
        f[e]=v;
        id[e]=l;
        while(s+1<e && cross(e-2,e-1)>cross(e-1,e)){
            f[e-1]=f[e];
            id[e-1]=id[e];
            --e;
        }
        ++e;
    }
    ll query(ll x){
        while(s+1<e && f[s+1].y-f[s].y <= x*(f[s].x-f[s+1].x))
            ++s;
```

```
      return f[s].x * x + f[s].y;
  }
} CHT;
```

LiChaoTree.h

Description: Convex hull trick. Current implementation is for max query. Be especially aware of overflow. Let M be maximum x coordinate, $aM + b$ should be less than LLMAX.
Time: $\mathcal{O}(\log N)$

87 lines

```
struct LiChao
{
  struct Line // Linear function ax + b
  {
    int a, b;
    int eval(int x)
    {
      return a*x + b;
    }
  };
  struct Node // [start, end] has line f
  {
    int left, right;
    int start, end;
    Line f;
  };

  Node new_node(int a, int b)
  {
    return {-1,-1,a,b,{0,-INF}};
    // for min, change -INF to INF
  }

  vector <Node> nodes;

  void init(int min_x, int max_x)
  {
    nodes.push_back(new_node(min_x, max_x));
  }
  void insert(int n, Line new_line)
  {
    int xl = nodes[n].start, xr = nodes[n].end;
    int xm = (xl + xr)/2;
    Line llo, lhi;
    llo = nodes[n].f, lhi = new_line;
    if (llo.eval(xl) >= lhi.eval(xl))
      swap(llo, lhi);
    if (llo.eval(xr) <= lhi.eval(xr))
    {
      nodes[n].f = lhi;
      // for min, lhi -> llo
      return;
    }
    else if (llo.eval(xm) > lhi.eval(xm))
    {
      nodes[n].f = llo;
      // for min, llo -> lhi
      if (nodes[n].left == -1)
      {
        nodes[n].left = nodes.size();
        nodes.push_back(new_node(xl,xm));
      }
      insert(nodes[n].left, lhi);
      // for min, lhi -> llo
    }
    else
    {
      nodes[n].f = lhi;
      // for min, lhi -> llo
```

```
      if (nodes[n].right == -1)
      {
        nodes[n].right = nodes.size();
        nodes.push_back(new_node(xm+1,xr));
      }
      insert(nodes[n].right,llo);
      // for min, llo -> lhi
    }
  }
  void insert(Line f)
  {
    insert(0, f);
  }
  int get(int n, int q)
  {
    // for min, max -> min, -INF -> INF
    if (n == -1) return -INF;
    int xl = nodes[n].start, xr = nodes[n].end;
    int xm = (xl + xr)/2;
    if (q > xm)
      return max(nodes[n].f.eval(q), get(nodes[n].right, q));
    else return max(nodes[n].f.eval(q), get(nodes[n].left, q));
  }
  int get(int pt)
  {
    return get(0, pt);
  }
};
```

LineContainer.h

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

29 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

SOS.h

Description: Find $F[mask] = \sum_{i \in mask} A[i]$ in $O(N * 2^N)$.

3 lines

```
for(int i = 0; i < (1 << N); i++) F[i] = A[i];
```

```
for(int j = 0; j < N; j++) for(int i = 0; i < (1 << N); i++)
  if(i & (1 << j)) F[i] += F[i ^ (1 << j)];
```

SubmaskEnum.h

Description: Iterating through all masks with their submasks. Complexity $O(3^n)$

3 lines

```
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
        (... s and m ...)
```

9.2 Divide and Conquer Optimization

C가 Monge Array, 즉 $C(a,c) + C(b,d) \leq C(a,d) + C(b,c)$ 이고, 점화식이
$$D(i,j) = \min_{k<j}(D(i-1,k) + C(k,j))$$
형태를 만족할 때, j값에 따라 최적인 k인 opt_j가 단조증가함을 관찰하여 확인할 후보를 줄인다. $D(k, s \ldots e)$ 를 구하기 위해, (1) 중간값 m에 대해 $D(k,m)$ 을 구한다. (2) 각각 재귀적으로 좌우를 호출하되, 봐야 할 j의 범위를 호출과정에서 관리한다.

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \leq k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R - 1$.
Time: $\mathcal{O}((N + (hi - lo))\log N)$

18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

9.3 Monotone Queue Optimization

Deque Trick 수열에서 sliding window 형태의 최솟값을 빠르게 확인해야 할 때, deque에 pair형으로 (값, 인덱스) 를 저장한다. 이때 front가 우리가 확인하고자 하는 구간을 벗어났다면 계속 pop front 하고, 뒤에 삽입할 때 삽입하려는 값보다 deque의 back이 더 크다면 이를 제거하는 형태로 deque를 monotonic하게 유지한다.

DequeTrick.h

Description: Monotone Deque-DP for sliding-window like query.

12 lines

```
dq.push_back({-1, 0});
for (int i = 0; i < n; i++) {
    while (!dq.empty() and dq.front().first < i - d) {
        dq.pop_front();
    }
    dp[i] = max(arr[i], dq.front().second + arr[i]);
    while (!dq.empty() and dq.back().second <= dp[i]) {
        dq.pop_back();
    }
    dq.push_back({i, dp[i]});
    ans = max(ans, dp[i]);
}
```

점화식 D가 $D(i) = \min_{j<i}(D(j) + C(j,i))$ 형태이고, C가 Monge array일 때, $O(n^2)$ DP를 $O(n \log n)$ 으로 줄이는 기법이다.

가정 : 다음을 만족하는 cross(i, j) 를 찾을 수 있다.

$$\text{cross}(i,j) > k \iff D(i) + C(i,k) < D(j) + C(j,k)$$

Queue Q에 앞으로 계산할 점화식에서 답이 될 수 있는 후보들을 차례로 저장한다. 즉, $D(i...n)$ 이 답이 되는 j들. 따라서, Q의 모든 원소들이 cross$(Q_i, Q_{i+1}) <$ cross(Q_{i+1}, Q_{i+2}) 를 만족하고, cross$(Q_0, Q_1) \geq i$ 를 만족하도록 한다. 이때, $D(i)$ 를 $D(Q_0) + C(Q_0, i)$를 구함으로써 해결.

각 i에 대해, cross$(Q_0, Q_1) \geq i$ 을 만족시키는 것은 Q에서 필요한 만큼 pop 하여 구할 수 있고, 원소를 Q에 삽입할 때 Cross 부등식을 유지하기 위해 CHT에서와 같이 조건이 성립할때까지 맨 뒤 원소를 제거한다. Deque DP랑 비슷한 concept.

큐의 맨 뒤 원소 두개를 x, y라 할 때, cross$(x, y) \geq$ cross(y, i) 이면 y가 최적인 위치가 없으므로 이를 계속 pop한다. Cross 한번은 이분탐색으로 $O(\log n)$ 에 찾을 수 있고, 이를 n번 호출하므로 $O(n \log n)$.

<u>Various</u> (10)

10.1   Intervals

IntervalContainer.h
Description: 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].
Time: $\mathcal{O}(\log N)$
<div align="right">23 lines</div>

```cpp
set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h
Description: 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).
Time: $\mathcal{O}(N \log N)$
<div align="right">19 lines</div>

```cpp
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b]; });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
```

```cpp
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h
Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$
<div align="right">19 lines</div>

```cpp
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T q) {
  if (p == q) return;
  if (from == to) {
    g(i, to, p);
    i = to; p = q;
  } else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
  }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
  if (to <= from) return;
  int i = from; auto p = f(i), q = f(to-1);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2   Misc. algorithms

TernarySearch.h
Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows nonstrict inequalities, change the < marked with (A) to <=, and reverse the loop at (B). To minimize f, change it to >, also at (B).
Usage: int ind = ternSearch(0,n-1,[&](int i){return a[i];});
Time: $\mathcal{O}(\log(b - a))$
<div align="right">11 lines</div>

```cpp
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

LIS.h
Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$
<div align="right">17 lines</div>

```cpp
template<class I> vi lis(const vector<I>& S) {
  if (S.empty()) return {};
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
```

```cpp
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

Bucket.h
Description: Be careful and double check for off by 1 error
<div align="right">44 lines</div>

```cpp
struct bset {
  bitset<64> arr[1577]; // bitset is a size-64 bucket
  bset() {}
  // single-query
  inline int bitquery(int idx) {
    return arr[idx >> 6][idx&63]?1:0;
  }
  // single-operation
  inline void bitflip(int idx) {
    arr[idx >> 6][idx & 63].flip();
  }
  void flip(int l, int r) {
    if (r - l < 200) {
      for (int i = l; i <= r; i++)
        bitflip(i);
      return;
    }
    int u = l >> 6;
    int v = r >> 6;
    for (int i = u+1; i < v; i++) {
      // bucket-operation
      arr[i].flip();
    }
    for (int i = l; i < ((u+1) << 6); i++) bitflip(i);
    for (int i = (v << 6); i <= r; i++) bitflip(i);
  }
  int query(int l, int r) {
    int ans = 0;
    if (r - l < 200) {
      for (int i = l; i <= r; i++)
        ans += bitquery(i);
      return ans;
    }
    int u = l >> 6;
    int v = r >> 6;
    for (int i = u+1; i < v; i++) {
      // bucket-query
      ans += arr[i].count();
    }
    for (int i = l; i < ((u+1) << 6); i++) ans += bitquery(i);
    for (int i = (v << 6); i <= r; i++) ans += bitquery(i);
    return ans;
  }
};
```