Team Note of Megazord da CAPES

UFCG

Compiled on March 25, 2019

Contents

1 Competition Environment

1.1 When you're stuck

- Binary search the answer
- SQRT Decomposition
- Large time/memory limit
- Max flow
- Wavelet tree
- FFT
- Gaussian elimination (subset xor)
- Random
- Preprocess the answer
- Limited answers (tricky: always 0, 1 or 2)
- Reduce to Coin DP
- 2-SAT
- Small alphabet (lowercase letters)

- Suffix Array
- Divide and Conquer

1.2 During warmup

- Test pragmas
- Test __gcd with long long and 0
- Auto and lambda
- Policy based structures (oset, omap, gp hash tables, rope)
- Check if __int128 is supported

1.3 Pragmas

```
#pragma GCC optimize ("O3")
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,
sse4,popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")
```

2 Data Structures

2.1 Persistent Segment Tree

```
struct Node {
  Node * l, * r;
  int sum;

  Node (int val) : l(NULL), r(NULL), sum(val) { }

  Node (Node * l, Node * r) : l(l), r(r), sum(0) {
    if (l)  sum += l->sum;
    if (r)  sum += r->sum;
  }
};

Node * build (int a[], int tl, int tr) {
  if (tl == tr)
    return new Node (a[tl]);
  int tm = (tl + tr) / 2;
  return new Node (build (a, tl, tm), build (a,
  tm+1, tr));
}

int get_sum(Node * t, int tl, int tr, int l, int
r) {
  if (l > r) return 0;
  if (l == tl && tr == r) return t->sum;
  int tm = (tl + tr) / 2;
  return get_sum (t->l, tl, tm, l, min(r,tm))
  + get_sum (t->r, tm+1, tr, max(l,tm+1), r);
}

Node * update (Node * t, int tl, int tr, int pos,
int new_val) {
  if (tl == tr) return new Node (new_val);
  int tm = (tl + tr) / 2;
  if (pos <= tm) {
    return new Node (update (t->l, tl, tm, pos,
    new_val), t->r);
  }
  else {
    return new Node (t->l, update (t->r, tm+1, tr,
    pos, new_val));
  }
}
```

2.2 Treap

```
struct treap {
  struct node {
    ll key, size, prior, sum, lazy;
    node *l, *r;

    node(ll key):
    key(key), size(1), prior(my_rand()), sum(key),
    lazy(0), l(NULL), r(NULL) {}

    int my_rand() {
      return ((rand()<<16) ^ rand()) & 0x7fffffff;
    }

    ~node() {
      delete l;
      delete r;
    }
  };

  typedef node* pnode;
  pnode root = NULL;

  ll get_size(pnode t) {
    return (t) ? t->size : 0;
  }

  void update_size(pnode t) {
    if(t) t->size = 1 + get_size(t->l) +
    get_size(t->r);
  }

  void push_lazy(pnode t) {
    if(t && t->lazy) {
    }
  }

  ll get_sum(pnode t) {
    return (t) ? t->sum : 0;
  }

  void update_sum(pnode t) {
    if(t) t->sum = t->key + get_sum(t->l) +
    get_sum(t->r);
  }

    // Kth element (0-indexed) [Careful with
    repeated elements]
  ll kth(pnode t, int k) {
    if(!t) return -1;
    int cur_k = get_size(t->l);
    if(cur_k == k) return t->key;
    else if(cur_k < k) return kth(t->r, k - cur_k
    - 1);
    else return kth(t->l, k);
```

```
}

  // Elements equal to key => LEFT TREE
void split(pnode t, pnode &l, pnode &r, ll key)
{
  if(!t) {
    l = r = NULL;
    return;
  }
  push_lazy(t);
  if(t->key <= key) {
    split(t->r, t->r, r, key);
    l = t;
  } else {
    split(t->l, l, t->l, key);
    r = t;
  }
  update_size(t);
  push_lazy(t);
  update_sum(t);
}

  // KEY stays on the LEFT tree
void split_by_index(pnode t, pnode &l, pnode &r,
int key) {
  if(!t) {
    l = r = NULL;
    return;
  }
  push_lazy(t);
  int cur_key = get_size(t->l);
  if(key >= cur_key) {
    split_by_index(t->r,t->r,r,key - cur_key -
    1);
    l = t;
  } else {
    split_by_index(t->l,l,t->l,key);
    r = t;
  }
  update_size(t);
  push_lazy(t);
  update_sum(t);
}

  // (L->key) <= (R->key), for every(L, R) in
  (l, r)
void merge(pnode &t, pnode l, pnode r) {
  push_lazy(l);
  push_lazy(r);
  if(!l || !r) t = l?l:r;
  else if(l->prior > r->prior) {
    merge(l->r, l->r, r);
    t = l;
  } else {
    merge(r->l, l, r->l);
    t = r;
  }
  update_size(t);
  push_lazy(t);
  update_sum(t);
}

~treap() {
    delete root;
  }
};
```

2.3 Wavelet Tree with Updates

```
int cmid;
bool f(int x) {
  return x <= cmid;
};

struct wavelet_tree{ //1-indexed
  int lo, hi;
  wavelet_tree *l, *r;

  treap t; // Implicit treap (KEY, VALUE)
  // With SUM of KEYS (0 or 1) on subtree
  // treap.erase returns VALUE of erased element

  wavelet_tree(int x, int y) {
    lo = x, hi = y;
    t.insert(0, 0);
    l = r = NULL;
  }

  wavelet_tree(int *from, int *to, int x, int y){
    lo = x, hi = y;
    t.insert(0, 0);

    if(lo == hi || from >= to) {
      l = r = NULL;
      return;
    }

    int mid = lo+(hi-lo)/2, i = 1;
    cmid = mid;

    for(int *it = from; it != to; it++, i++) {
      t.insert(f(*it), *it); //append (KEY, VALUE)
    }

    int *pivot = stable_partition(from, to, f);
    l = new wavelet_tree(from, pivot, lo, mid);
    r = new wavelet_tree(pivot, to, mid+1, hi);
  }

  int map_left(int i) {
    return t.rank(i); //how many elements with
    KEY=1
  }

  // kth smallest element in [l, r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;

    int lb = map_left(l - 1);
    int rb = map_left(r);
    int quantity = rb - lb;

    if(k <= quantity) return this->l->kth(lb + 1,
    rb , k);
```

```
    return this->r->kth(l - lb, r - rb, k -
    quantity);
  }

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

    int lb = map_left(l - 1);
    int rb = map_left(r);

    return this->l->LTE(lb + 1, rb, k) +
    this->r->LTE(l - lb, r - rb, k);
  }

  // count of numbers in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r || k < lo || k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int mid = lo+(hi-lo)/2;

    int lb = map_left(l - 1);
    int rb = map_left(r);

    if(k <= mid) return this->l->count(lb + 1, rb,
    k);
    return this->r->count(l - lb, r - rb, k);
  }

  void insert(int x, int pos) {
    if(lo == hi) return;

    int mid = lo+(hi-lo)/2;
    int left_partition = (x <= mid);
    int lb = map_left(pos - 1);

    t.insert(left_partition, x, pos);
    if(left_partition) {
      if(!l) l = new wavelet_tree(lo, mid);
      l->insert(x, lb + 1);
    } else {
      if(!r) r = new wavelet_tree(mid + 1, hi);
      r->insert(x, pos - lb);
    }
  }

  void erase(int pos) {
    if(lo == hi) return;
    int lb = map_left(pos - 1);
    int x = t.erase(pos);

    int mid = lo+(hi-lo)/2;
    if(x <= mid) this->l->erase(lb + 1);
    else this->r->erase(pos - lb);
  }

  ~wavelet_tree() { // faster: delete this
    delete l;
    delete r;
  }
};
```

2.4 2D Segment Tree

Usage: Use only *get* and *update* functions

```
#define tmax 1010
int st[tmax*4][tmax*4], sizem;

int gety(int nodex, int nodey, int iniy, int fimy,
int y, int y1) {
  if (y > fimy || y1 < iniy || iniy > fimy) return
  0;
  if (y <= iniy && y1 >= fimy) return
  st[nodex][nodey];

  int mid = (iniy + fimy) / 2;
  return gety(nodex, nodey*2, iniy, mid, y, y1) +
  gety(nodex, nodey*2+1, mid+1, fimy, y, y1);
}

int get (int nodex, int inix, int fimx, int x, int
y, int x1, int y1) {
  if (x > fimx || x1 < inix || inix > fimx) return
  0;
  if (x <= inix && x1 >= fimx) return gety(nodex,
  1, 0, sizem-1, y, y1);

  int mid = (inix + fimx) / 2;
  return get(nodex*2, inix, mid, x, y, x1, y1) +
  get(nodex*2+1, mid+1, fimx, x, y, x1, y1);
}

void updatey(int nodex, int inix, int fimx, int
nodey, int iniy, int fimy, int y, int val) {
  if (iniy == fimy) {
    if (inix == fimx) st[nodex][nodey] = val;
    else st[nodex][nodey] = st[nodex*2][nodey] +
    st[nodex*2 + 1][nodey];
    return;
  }

  int mid = (iniy + fimy) / 2;
  if (y <= mid) updatey(nodex, inix, fimx,
  nodey*2, iniy, mid, y, val);
  else updatey(nodex, inix, fimx, nodey*2+1,
  mid+1, fimy, y, val);

  st[nodex][nodey] = st[nodex][nodey*2] +
  st[nodex][nodey*2 + 1];
}

void update(int nodex, int inix, int fimx, int x,
int y, int val) {
  if (inix != fimx) {
    int mid = (inix + fimx) / 2;
    if (x <= mid) update(nodex*2, inix, mid, x, y,
    val);
    else update(nodex*2+1, mid+1, fimx, x, y,
    val);
  }
  updatey(nodex, inix, fimx, 1, 0, sizem-1, y,
  val);
}
```

2.5 LiChao Segment Tree

Usage: Minimum value. Functions must intersect at most once.
`li_chao L(minval, maxval); L.add_func(new func(a, b)); L.query(x);`

```
struct func {
  ll a, b;
  func() {}
  func(ll a, ll b): a(a), b(b) {}
  ll y(ll x) { return a*x + b; }
};

struct li_chao {
  struct node {
    node *l, *r;
    func *f;
    node() {l = r = NULL; f = NULL;}
  };

  node *root;
  ll L, R;
  li_chao(ll l, ll r) {root = new node(); L = l; R
  = r;}

  void add_func(node *&t, ll L, ll R, func *f, ll
  lf, ll rf) {
    ll M = L + (R - L)/2;
    if(t == NULL) t = new node();
    if(f == NULL || rf < L || R < lf) return;

    else if(lf <= L && R <= rf) {
      if(t->f == NULL || f->y(M) < t->f->y(M))
        swap(t->f, f);
      if(L < R) {
        if(f == NULL || f->y(L) >= t->f->y(L))
          add_func(t->r, M+1, R, f, lf, rf);
        else
          add_func(t->l, L, M, f, lf, rf);
      }
    }
    else {
      add_func(t->l, L, M, f, lf, rf);
      add_func(t->r, M+1, R, f, lf, rf);
    }
  }

  ll query(node *t, ll L, ll R, ll x) {
    ll M = L + (R - L)/2;
    if(t == NULL) return LLONG_MAX;
    ll ret = (t->f == NULL) ? LLONG_MAX :
    t->f->y(x);
    if(L < R) {
      if(x <= M) ret = min(ret, query(t->l, L,
      M, x));
      else       ret = min(ret, query(t->r, M+1,
      R, x));
    }
    return ret;
  }

  void add_func(func *f) { add_func(root, L, R, f,
  L, R); }
  void add_func(func *f, ll l, ll r) {
  add_func(root, L, R, f, l, r); }
  ll query(ll x) { return query(root, L, R, x); }
};
```

2.6 STL Faster Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

const ll TIME =
chrono::high_resolution_clock::now().time_since_epoch().
const ll SEED = (ll)(new ll);
const ll RANDOM = TIME ^ SEED, MOD = (int)1e9+7,
MUL = (int)1e6+3;

struct chash{
  ll operator()(ll x) const { return
  std::hash<ll>{}((x ^ RANDOM) % MOD * MUL); }
};

gp_hash_table<ll, ll, chash> table;
```

2.7 Ordered Set, Ordered Map, Rope

Usage: Rope supports insert, erase, substr. Almost 4x slower.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>

using namespace __gnu_pbds;
using namespace __gnu_cxx; // rope

template<typename T, typename cmp = less<T>>
using oset = tree<T, null_type, cmp, rb_tree_tag,
tree_order_statistics_node_update>;

template<typename T, typename V, typename cmp =
less<T>>
using omap = tree<T, V, cmp, rb_tree_tag,
tree_order_statistics_node_update>;

int main() {
  omap<string, string> mm;
  mm["AAA"] = "abc";
  mm.insert(make_pair("BC", "def"));
  cout << mm.order_of_key("BC") << endl;
  // 0-indexed
  cout << (*mm.find_by_order(0)).second << endl;
  // abc

  rope <int> v(3, 7);
  cout << v[3] << endl;
  for(int x : v) cout << x << " ";
}
```

2.8 Mo's Algorithm with updates

Time Complexity: $O(N^{5/3})$

```
#define MN 100100
```

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

int B = 2000; // N ^ (2/3)
int arr[MN], freq[MN], count_different;
int l = 0, r = -1, chronol = -1;

struct query {
  int l, r, t, id;

  query(int l, int r, int t, int id) :
  l(l), r(r), t(t), id(id) {}

  bool operator <(const query &o) const {
    if(l / B != o.l / B) return l < o.l;
    if(r / B != o.r / B) return r < o.r;
    return t < o.t;
  }
};

void rem(int idx) {
  if(--freq[arr[idx]] == 0) count_different--;
}

void add(int idx) {
  if(freq[arr[idx]]++ == 0) count_different++;
}

struct update {
  int from, to, id;

  update(int to, int id) :
  id(id), to(to) {}

  void upd() {
    if(l <= id && id <= r) rem(id);
    from = arr[id];
    arr[id] = to;
    if(l <= id && id <= r) add(id);

    swap(from, to);
  }
};

vector<query> Q;
vector<update> U;
ll ans[MN];

void solve() {
  sort(Q.begin(), Q.end());
  for(int i = 0; i < Q.size(); i++) {
    int T = Q[i].t, L = Q[i].l, R = Q[i].r;

    while(chronol < T) U[++chronol].upd();
    while(chronol > T) U[chronol--].upd();

    while(l > L) add(--l);
    while(l < L) rem(l++);

    while(r < R) add(++r);
    while(r > R) rem(r--);

    ans[Q[i].id] = count_different;
  }
}

int main() {
  int q, t, a, b, c = -1;
    scanf("%d\n", &q); // (faster) B = pow(n, 0.7)
    + 1;

    printf("0 update, 1 query\n");
    for(int i = 0; i < q; i++) {
      scanf("%d %d %d", &t, &a, &b);
      if(t == 0) {
        U.push_back(update(b, a));
        c++;
      } else {
        Q.push_back(query(a, b, c, Q.size()));
      }
    }

    solve();
    for(int i = 0; i < Q.size(); i++) {
      printf("%d\n", ans[i]);
    }
    return 0;
  }
```

3 Graphs

3.1 Dinic

```
#define INF 2000000000
#define N 10100

int nodes = N, ini, fim;
int dist[N], work[N];

struct Edge {
  int to, rev, used_flow, cap;
};

vector < Edge > graph[N];

void addEdge(int from, int to, int cap) {
  Edge a = {to, graph[to].size(), 0, cap};
  Edge b = {from, graph[from].size(), 0, 0};
  graph[from].push_back(a);
  graph[to].push_back(b);
}

bool dinic_bfs() {
  fill(dist, dist + nodes, -1);
  dist[ini] = 0;

  queue < int > fila;
  fila.push(ini);

  while (!fila.empty()) {
    int u = fila.front();
    fila.pop();

    for (int i = 0; i < graph[u].size(); i++) {
      Edge &e = graph[u][i];
      int v = e.to;
```

```
      if (dist[v] < 0 && e.used_flow < e.cap) {
        dist[v] = dist[u] + 1;
        fila.push(v);
      }
    }
  }

  return dist[fim] >= 0;
}

int dinic_dfs(int u, int flow) {
  if (u == fim)
    return flow;

  for (int &i = work[u]; i < graph[u].size(); i++)
  {
    Edge &e = graph[u][i];

    if (e.cap > e.used_flow) {
      int v = e.to;

      if (dist[v] == dist[u] + 1) {
        int minf = dinic_dfs(v, min(flow, e.cap -
        e.used_flow));
        if (minf > 0) {
          e.used_flow += minf;
          graph[v][e.rev].used_flow -= minf;
          return minf;
        }
      }
    }
  }

  return 0;
}

int flow(int _ini, int _fim) {
  ini = _ini;
  fim = _fim;
  int result = 0;

  while (dinic_bfs()) {
    fill(work, work + nodes, 0);
    while (int delta = dinic_dfs(ini, INT_MAX))
      result += delta;
  }

  return result;
}
```

3.2 Min Cost Max Flow

```
#define N 200

int cap[N][N], cost[N][N];
int fnet[N][N], adj[N][N], deg[N];
int par[N], d[N];
int pi[N];

#define CLR(a, x) memset( a, x, sizeof( a ) )
#define Inf (INT_MAX/2)
#define Pot(u,v) (d[u] + pi[u] - pi[v])
```

```
bool dijkstra(int n, int s, int t) {
  for (int i = 0; i < n; i++) d[i] = Inf, par[i] =
  -1;
    d[s] = 0;
  par[s] = -n - 1;

  while (1) {
    int u = -1, bestD = Inf;
    for (int i = 0; i < n; i++)
      if (par[i] < 0 && d[i] < bestD)
        bestD = d[u = i];

    if (bestD == Inf) break;

    par[u] = -par[u] - 1;
    for (int i = 0; i < deg[u]; i++) {
      int v = adj[u][i];
      if (par[v] >= 0) continue;
      if (fnet[v][u] && d[v] > Pot(u, v) -
      cost[v][u])
        d[v] = Pot(u, v) - cost[v][u], par[v] =
        -u-1;

      if (fnet[u][v] < cap[u][v] && d[v] > Pot(u,
      v) + cost[u][v])
        d[v] = Pot(u, v) + cost[u][v], par[v] = -u
        - 1;
    }
  }

  for (int i = 0; i < n; i++)
    if (pi[i] < Inf)
      pi[i] += d[i];

  return par[t] >= 0;
}

int mcmf(int n, int s, int t, int &fcost) {
  CLR(deg, 0);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      if (cap[i][j] || cap[j][i])
        adj[i][deg[i]++] = j;

  CLR(fnet, 0);
  CLR(pi, 0);
  int flow = fcost = 0;

  while (dijkstra(n, s, t)) {
    int bot = INT_MAX;
    for (int v = t, u = par[v]; v != s; u = par[v
    = u])
      bot = min(bot, fnet[v][u] ? fnet[v][u] :
      (cap[u][v] - fnet[u][v]));

    for (int v = t, u = par[v]; v != s; u = par[v
    = u])
      if (fnet[v][u]) { fnet[v][u] -= bot; fcost
      -= bot * cost[v][u]; }
    else { fnet[u][v] += bot; fcost += bot *
    cost[u][v]; }

    flow += bot;
```

```
  }

  return flow;
}
```

3.3 Mo's algorithm on trees

```
const int L = ...
const int N = ...

int idx;
int st[N], en[N];
int depth[N], a[N];
int parent[N][L];
vector<int> graph[N];

void dfs(int u) {
  a[idx] = u;
  st[u] = idx++;
  for (int v : graph[u]) {
    if (v != parent[u][0]) {
      parent[v][0] = u;
      depth[v] = 1 + depth[u];
      dfs(v);
    }
  }
  a[idx] = u;
  en[u] = idx++;
}

void build(int n) {
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < L; ++j) {
      parent[i][j] = -1;
    }
    depth[i] = 0;
  }
  dfs(0);
  for (int j = 1; j < L; ++j) {
    for (int i = 0; i < n; ++i) {
      if (parent[i][j - 1] != -1) {
        parent[i][j] = parent[parent[i][j - 1]][j
        - 1];
      }
    }
  }
}

int get_lca(int u, int v) {
  if (depth[u] < depth[v]) {
    swap(u, v);
  }
  for (int i = L - 1; i >= 0; --i) {
    if (depth[u] - (1 << i) >= depth[v]) {
      u = parent[u][i];
    }
  }
  if (u == v) {
    return u;
  }
  for (int i = L - 1; i >= 0; --i) {
    if (parent[u][i] != parent[v][i]) {
      u = parent[u][i];
      v = parent[v][i];
    }
  }
  return parent[u][0];
}

int S;

struct query {
  int id, lca, l, r;
  query() {}
  bool operator < (const query &other) const {
    if (l / S == other.l / S) {
      return r < other.r;
    } else {
      return l / S < other.l / S;
    }
  }
} queries[N];

void update(int u, int &res) {
  //update node u and res
}

vector<int> mos_on_tree(int n, int m) {
  vector<int> ans(m, 0);
  for (int i = 0; i < m; ++i) {
    int u, v;
    scanf("%d %d", &u, &v);
    --u, --v;
    int lca = get_lca(u, v);
    if (st[u] > st[v]) {
      swap(u, v);
    }
    queries[i].lca = lca;
    queries[i].id = i;
    if (u == lca) {
      queries[i].l = st[u];
      queries[i].r = st[v];
    } else {
      queries[i].l = en[u];
      queries[i].r = st[v];
    }
  }
  S = sqrt(2 * n);
  int prev_l = 0, prev_r = -1, res = 0;
  sort(queries, queries + m);
  for (int i = 0; i < m; ++i) {
    int l = queries[i].l, r = queries[i].r;
    while (prev_l < l) update(a[prev_l++], res);
    while (prev_l > l) update(a[--prev_l], res);
    while (prev_r < r) update(a[++prev_r], res);
    while (prev_r > r) update(a[prev_r--], res);
    int u = a[prev_l], v = a[prev_r];
    if (queries[i].lca != u and queries[i].lca !=
    v) {
      update(queries[i].lca, res);
    }
    ans[queries[i].id] = res;
    if (queries[i].lca != u and queries[i].lca !=
    v) {
      update(queries[i].lca, res);
    }
```

```
  }
  return ans;
}

int main() {
  int n, m;
  //after build graph
  for (int res : mos_on_tree(n, m)) {
    printf("%d ", res);
  }
  return 0;
}
```

3.4 2-SAT and SCC

```
#define JUMP 100050

int counter = 1, scc = 1, value[100100];
bool visited[200100], new_edges[200100];
int dfs_num[200100], dfs_low[200100], cp[200100];
vector < int > graph[200100], new_graph[200100],
comp[200100], aux, used, toposort;

void dfs(int u) {
  dfs_low[u] = dfs_num[u] = counter++;
  aux.push_back(u);
  visited[u] = true;

  for (int v : graph[u]) {
    if (!dfs_num[v])
      dfs(v);
    if (visited[v])
      dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }

  if (dfs_low[u] == dfs_num[u]) {
    while (1) {
      int v = aux.back();
      aux.pop_back();
      visited[v] = false;

      comp[scc].push_back(v);
      cp[v] = scc;

      if (u == v) break;
    }
    scc++;
  }
}

void create_new_graph() {
  for (int i = 1; i < scc; i++) {
    for (int j = 0; j < used.size(); j++)
      new_edges[used[j]] = false;

    used.clear();

    for (int j = 0; j < comp[i].size(); j++)
      for (int v : graph[comp[i][j]])
        if (cp[v] != i && !new_edges[cp[v]]) {
          new_graph[i].push_back(cp[v]);
          used.push_back(cp[v]);
          new_edges[cp[v]] = true;
        }
  }
}

void Tsort(int u) {
  dfs_num[u] = 1;
  for (int v : new_graph[u])
    if (!dfs_num[v])
      Tsort(v);
  toposort.push_back(u);
}

void set_values() {
  for (int i = 0; i < 100100; i++)
    value[i] = -1;

  for (int i = toposort.size()-1; i >= 0; i--)
    for (int v : comp[toposort[i]])
      if (v < JUMP && value[v] == -1)
        value[v] = 1;
      else if (v > JUMP && value[v-JUMP] == -1)
        value[v-JUMP] = 0;
}
```

3.5 Heavy-Light Decomposition

Usage: Remember to implement Range Data Structure

```
const int N = ...
typedef pair<int, int> edge;

int n; //number of vertices
int heavy[N], parent[N], depth[N];
int index_of[N], value[N], chain[N];

vector<edge> graph[N];

int dfs(int u) {
  int sz = 1;
  int max_sz = heavy[u] = 0;

  for (edge e : graph[u]) {
    int v = e.first;
    if (v != parent[u]) {
      parent[v] = u;
      depth[v] = 1 + depth[u];
      value[v] = e.second;
      int sub_sz = dfs(v);
      if (sub_sz > max_sz) {
        max_sz = sub_sz;
        heavy[u] = v;
      }
      sz += sub_sz;
    }
  }

  return sz;
}

void hld(int u, int &index, int head) {
  index_of[u] = index++;
  update(0, 0, n - 1, index_of[u], value[u]);
  //RANGE DS
  chain[u] = head;
```

```
  if (heavy[u]) {
    hld(heavy[u], index, head);
  }
  for (edge e : graph[u]) {
    int v = e.first;
    if (v != heavy[u] and v != parent[u]) {
      hld(v, index, v);
    }
  }
}

int get(int u, int v) {
  int asw = 0;
  while (chain[u] != chain[v]) {
    if (depth[chain[u]] < depth[chain[v]])
      swap(u, v);
    asw = max(asw, query(0, 0, n - 1,
    index_of[chain[u]], index_of[u])); //RANGE DS
    u = parent[chain[u]];
    if (u == v) {
      return asw;
    }
  }
  return max(asw, query(0, 0, n - 1,
  min(index_of[u], index_of[v]) + 1,
  max(index_of[u], index_of[v]))); //RANGE DS
}

void push(int u, int v, int cost) {
  if (parent[u] == v) {
    update(0, 0, n - 1, index_of[u], cost);
    //RANGE DS
  } else {
    update(0, 0, n - 1, index_of[v], cost);
    //RANGE DS
  }
}

void build(int root) {
  dfs(root);
  int index = 0;
  hld(root, index, root);
}
```

3.6 Centroid Decomposition

```
#define N 100100
vector < int > graph[N];

struct CentroidDec {
  int root, visited[N], siz[N], layer[N],
  parent[N];
  vector < int > centroidTree[N];

  void init() {
    for (int i = 0; i < N; i++) {
      visited[i] = siz[i] = layer[i] = parent[i] =
      0;
      centroidTree[i].clear();
    }
  }

  int getCentroid(int u, int p = 0) {
    siz[u] = 1;
    for (int v : graph[u])
      if (v != p && !visited[v]) {
        getCentroid(v, u);
        siz[u] += siz[v];
      }
    if (p) return 0;

    int par = 0, aux = u, nxt = 0;
    while (1) {
      for (int v : graph[aux])
        if (!visited[v] && v != par && siz[v] >
        siz[u] / 2)
          nxt = v;

      if (!nxt) return aux;
      else { par = aux; aux = nxt; nxt = 0; }
    }
  }

  void buildTree(int u = 0) {
    if (u == 0) {
      u = root = getCentroid(1);
      visited[u] = 1; layer[u] = 1;
    }

    for (int v : graph[u])
      if (!visited[v]) {
        int x = getCentroid(v);
        visited[x] = 1; layer[x] = layer[u] + 1;
        parent[x] = u;
        centroidTree[u].push_back(x);
        centroidTree[x].push_back(u);
        buildTree(x);
      }
  }
} centroid;
```

3.7 Cut Points, Bridges, Biconnected Components

```
const int N = ...

vector<int> graph[N];
int id; bool marked[N];

int num[N], low[N];
vector<int> cut_points, passed;
vector<pair<int, int>> bridges;
vector<vector<int>> components;

void dfs(int u, int p) {
  marked[u] = true;
  low[u] = num[u] = ++id;
  passed.push_back(u);
  int children = 0;
  bool cut_point = false;
  for (int v : graph[u]) {
    if (v != p) {
      if (marked[v]) {
        low[u] = min(low[u], num[v]);
      } else {
        dfs(v, u);
```

```
        low[u] = min(low[u], low[v]);
        cut_point |= (low[v] >= num[u]);
        if (low[v] > num[u]) {
          bridges.push_back({u, v});
        }
        ++children;
      }
    }
  }
  if (p == -1) {
    cut_point = (children >= 2);
  }
  if (cut_point) {
    cut_points.push_back(u);
  }
  if (low[u] == num[u]) {
    vector<int> component;
    while (true) {
      int v = passed.back();
      passed.pop_back();
      component.push_back(v);
      if (v == u) {
        break;
      }
    }
    components.push_back(component);
  }
}

void build(int n) {
  fill(low, low + n, -1);
  fill(num, num + n, -1);
  for (int i = 0; i < n; ++i) {
    if (num[i] == -1) {
      dfs(i, -1);
    }
  }
}
```

3.8 Dominator Tree

```
#define N 100100

int n, m, r, x, y;
vector < int > auxg[N], graph[N];

struct DominatorTree {
  int cnt, pos[N], best[N], p[N], parent[N],
  order[N];
  int link[N], idom[N], sdom[N];
  vector < int > radj[N], bucket[N];

  void dfs(int nd) {
    pos[nd] = cnt;
    order[cnt++] = nd;
    for(int ch : graph[nd])
      if(pos[ch] == -1) {
        dfs(ch);
        parent[ch] = nd;
      }
  }

  int findBest(int x) {
    if(p[x] != x) {
      int u = findBest(p[x]);
      if (pos[sdom[u]] < pos[sdom[best[x]]])
        best[x] = u;
      p[x] = p[p[x]];
    }
    return best[x];
  }

  void dominators(int n, int root) {
    cnt = 0;

    for(int i = 0; i < n+1; i++) {
      pos[i] = parent[i] = idom[i] = -1;
      p[i] = best[i] = sdom[i] = i;
    }

    for(int i = 0; i < n+1; i++)
      for(int u : graph[i])
        radj[u].push_back(i);

    dfs(root);
    for(int i = cnt-1; i >= 1; i--) {
      int w = order[i];
      for(int u : radj[w])
        if(pos[u] != -1) {
          int t = findBest(u);
          if(pos[sdom[t]] < pos[sdom[w]])
            sdom[w] = sdom[t];
        }

        bucket[sdom[w]].push_back(w);
        idom[w] = sdom[w];
        int pw = parent[w];
        for(int u : bucket[pw])
          link[u] = findBest(u);
        bucket[pw].clear();
        p[w] = pw;
    }

    for(int i = 1; i < cnt; i++) {
      int u = order[i];
      idom[u] = idom[link[u]];
    }

    for(int i = 0; i < n+1; i++)
      radj[i].clear();
  }
} DT;
```

4 Strings

4.1 KMP

```
struct KMP {
  static vector <int> prefix_function(const vector
  <int> &T) {
    int i = 0, j, n = T.size();
    vector <int> back(n + 1);
    back[0] = j = -1;
    while (i < n) {
      while (j >= 0 and T[i] != T[j]) { j =
      back[j]; }
```

```
      ++i;
      ++j;
      back[i] = j;
    }
    return back;
  }

  static vector <int> matching(const vector <int>
  &S, const vector <int> &T) {
    int i = 0, j = 0, n = S.size(), m = T.size();
    vector <int> back = prefix_function(T), at;
    while (i < n) {
      while (j >= 0 and S[i] != T[j]) { j =
      back[j]; }
      ++i;
      ++j;
      if (j == m) {
        at.push_back(i - j);
        j = back[j];
      }
    }
    return at;
  }
};
```

4.2 Z-Function

```
void z_function(string &s, int *z) {
  for(int i = 1, L = 0, R = 0; i < s.size(); i++)
  {
    if(i > R) L = R = i;
    z[i] = min(R-i, z[i-L]);
    while(i + z[i] < s.size() && s[z[i]] ==
    s[i+z[i]]) z[i]++;
    if(i + z[i] > R) L = i, R = i + z[i];
  }
}
```

4.3 Manacher's Algorithm

```
#define N 2000020

int n, lens[N];
string s;

void manacher() {
  int m = 2*n + 1;
  int C = -1, R = 0;

  for (int i = 0; i < m; i++) {
    int mirror = 2*C-i;
    int diff = R - i;

    if(diff > 0) lens[i] = min(lens[mirror],
    diff);

    while (i + lens[i] < m && i - lens[i] > 0) {
      if ((i + lens[i] + 1) % 2 == 0) lens[i]++;
      else if (s[(i+lens[i]+1)/2] ==
      s[(i-lens[i]-1)/2]) lens[i]++;
      else break;
    }
    if (i + lens[i] > R) {
      C = i;
      R = i + lens[i];
    }
  }
}
```

4.4 Aho-Corasick

```
#define N 300300
#define ALP_SIZ 26

struct Automaton {
  int nodes, fail[N], wid[N], endLink[N],
  child[N][ALP_SIZ];
  queue < int > q;
  bool leaf[N];

  int newnode() {
    fail[nodes] = wid[nodes] = endLink[nodes] =
    leaf[nodes] = 0;
    memset(child[nodes], 0, sizeof(child[nodes]));
    return nodes++;
  }

  void clear() {
    nodes = 0;
    newnode();
  }

  void insert(string s, int id) {
    int atual = 0;
    for (int i = 0; i < s.size(); i++) {
      int c = s[i] - 'a';
      if (!child[atual][c])
        child[atual][c] = newnode();
      atual = child[atual][c];
    }
    leaf[atual] = true;
    wid[atual] = id;
  }

  void getFails() {
    for (int i = 0; i < ALP_SIZ; i++)
      if (child[0][i]) {
        fail[child[0][i]] = 0,
        q.push(child[0][i]);
        if (leaf[child[0][i]])
        endLink[child[0][i]] = child[0][i];
      }

    while (!q.empty()) {
      int u = q.front(); q.pop();

      for (int i = 0; i < ALP_SIZ; i++) {
        int v = child[u][i];
        if (!v) { child[u][i] = child[fail[u]][i];
        continue; }

        q.push(v);
        int j = fail[u];
        while (j && !child[j][i]) j = fail[j];
        fail[v] = child[j][i];
```

```
        if (leaf[v]) endLink[v] = v;
        else endLink[v] = endLink[fail[v]];
      }
    }
  }
} AC;

int n;
string pat[N], s;

void findOccurs(string s) {
  int atual = 0;

  for (int i = 0; i < s.size(); i++) {
    while (!AC.child[atual][s[i] - 'a'] && atual
    != 0) atual = AC.fail[atual];
    atual = AC.child[atual][s[i] - 'a'];

    int aux = atual;
    while (true) {
      aux = AC.endLink[aux];
      if (aux == 0) break;

      cout << pat[AC.wid[aux]] << " found at
      position " << i - pat[AC.wid[aux]].size() +
      1 << endl;

      aux = AC.fail[aux];
    }
  }
}

int main() {
  AC.clear();

  scanf("%d", &n); //numero de padroes
  for (int i = 0; i < n; i++) {
    cin >> pat[i];
    AC.insert(pat[i], i);
  }
  AC.getFails();

  getchar();
  getline(cin, s);
  findOccurs(s);
}
```

4.5 Suffix Array

```
#define MAX_N 100100
#define ALP_SIZ 256

char str[MAX_N];
int N, m, SA[MAX_N], LCP[MAX_N];
int x[MAX_N], y[MAX_N], w[MAX_N], c[MAX_N];

inline bool cmp(const int a, const int b, const
int l) {
  return (y[a] == y[b] && y[a + l] == y[b + l]);
}

void Sort() {
  for (int i = 0; i < m; i++) w[i] = 0;
    for (int i = 0; i < N; i++) ++w[x[y[i]]];
      for (int i = 0; i < m - 1; i++) w[i + 1] +=
      w[i];
        for (int i = N - 1; i >= 0; i--)
        SA[--w[x[y[i]]]] = y[i];
}

void DA() {
  ++N;
  for (int i = 0; i < N; i++) x[i] = str[i], y[i]
  = i;
    Sort();

  for (int i, j = 1, p = 1; p < N; j <<= 1, m = p)
  {
    for (p = 0, i = N - j; i < N; i++) y[p++] = i;
      for (int k = 0; k < N; k++)
        if (SA[k] >= j)
          y[p++] = SA[k] - j;

    Sort();
    for (swap(x, y), p = 1, x[SA[0]] = 0, i = 1; i
    < N; ++i)
      x[SA[i]] = cmp(SA[i - 1], SA[i], j) ? p - 1
      : p++;
  }

  for (int i = 1; i < N; i++) SA[i - 1] = SA[i];
  --N;
}

void kasaiLCP() {
  for (int i = 0; i < N; i++) c[SA[i]] = i;
  LCP[0] = 0;
  for (int i = 0, h = 0; i < N; i++) {
    if (c[i] == N-1) {
      LCP[c[i]] = h = 0;
      continue;
    }

    int j = SA[c[i] + 1];
    while (i + h < N && j + h < N && str[i + h] ==
    str[j + h]) ++h;
    LCP[c[i]] = h;
    if (h > 0) --h;
  }
}

void SuffixArray() {
  m = ALP_SIZ;
  N = strlen(str);
  DA();
  kasaiLCP();
}
```

5 Math

5.1 Modular Multiplication

```
ll mul_mod(ll a, ll b, ll MOD) {
  ll x = a*b, y = (long double)a*b/MOD-0.5;
  return (x - y*MOD)%MOD;
```

```
}
```

5.2 Simpson's Integral

```
double f(double x);

double integral(double a, double b) {
  const int N = 2000000; // #STEPS * 2
  double h = (b - a) / N, s = 0;

  for(int i=0; i<=N; i++) {
    double x = a + i*h;
    if(i == 0 || i == N) s += f(x);
    else if(i % 2 == 0) s += 2 * f(x);
    else s += 4 * f(x);
  }

  s *= h / 3;
  return s;
}
```

5.3 Random Number Generator

```
typedef unsigned long long ull;

ull seed = 0;
ull nxt() {
  seed ^= ull(102938711);
  seed *= ull(109293);
  seed ^= seed >> 13;
  seed += ull(1357900102873);
  return seed;
}
```

5.4 Floyd's Cycle Finding Algorithm

```
ll a, b, c;
ll f(ll x) {
  return (a * x + (x % b)) % c;
}

ll mu, lambda; //mu -> first ocurrence, lambda ->
cycle length
void Floyd(ll x0) {
  ll hare, tortoise;

  tortoise = f(x0), hare = f(f(x0));
  while(hare != tortoise) {
    tortoise = f(tortoise);
    hare = f(f(hare));
  }

  hare = x0, mu = 0;
  while(tortoise != hare) {
    tortoise = f(tortoise);
    hare = f(hare);
    mu++;
  }

  hare = f(tortoise), lambda = 1;
  while(t != h) {
    hare = f(hare);
    lambda++;
  }
}
```

5.5 Extended GCD and Linear Diophantine Equation

```
/// ax + by = gcd(a, b) - finds x, y and returns
gcd
ll xgcd(ll a, ll b, ll &x, ll &y) {
  if(b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll g = xgcd(b, a%b, x, y), x1 = y;
  y = x - (a / b) * y;
  x = x1;
  return g;
}

/// ax + by = c - true if solution is found
/// any solution: x + m*(b/gcd(a,b)), y -
m*(a/gcd(a,b))
bool linear_diophantine(ll a, ll b, ll c, ll &x,
ll &y) {
  ll d = xgcd(a, b, x, y);
  if(c % d) return false;
  x = x * (c / d);
  y = y * (c / d);
  return true;
}

/// xi and yi are initial solutions of Diophantine
/// only works when both increases or both
decreases
bool find_first_non_negative_solution(ll a, ll b,
ll &xi, ll &yi) {
  ll g = abs(__gcd(a, b)), m;
  ll augx = b / g, augy = - a / g;

  ll m1 = -xi/augx;
  ll m2 = -yi/augy;
    if(augx > 0) { //ceiling
      if(xi % augx && (xi ^ augx) < 0) m1++;
      if(yi % augy && (yi ^ augx) < 0) m2++;
      m = max(m1, m2);
    } else { //floor
      if(xi % augx && (xi ^ augx) >= 0) m1++;
      if(yi % augy && (yi ^ augx) >= 0) m2++;
      m = min(m1, m2);
    }
    xi = xi + m * augx;
    yi = yi + m * augy;
    return true;
  }
```

5.6 Determinant

Usage: Remember to put EPS in every comparison in case of changing to double

```
vector<vector<ll> > a(MN, vector<ll> (MN));

ll getDet(int n) {
  ll det = 1;
```

```
  for(int i = 0; i < n; i++) {
    int pivot = i;
    for(int j = i + 1; j < n; j++)
      if(abs(a[j][i]) > abs(a[pivot][i]))
        pivot = j;

    if(a[pivot][i] == 0)
      return 0;

    if(i != pivot) {
      swap(a[i], a[pivot]);
      det = -det;
    }

    det = (det * a[i][i]) % MOD;
    for(int j = i + 1; j < n; j++)
      a[i][j] = (a[i][j] * inv_mod(a[i][i])) %
      MOD;

    for(int j = 0; j < n; j++)
      if(i != j && a[j][i])
        for(int k = i + 1; k < n; k++)
          a[j][k] = (a[j][k] - (a[i][k] *
          a[j][i])%MOD + MOD) % MOD;
  }
  return (det + MOD) % MOD;
}
```

5.7 Segmented Sieve

```
#define SZ 1285 ///Make SZ equal to number of
primes in sqrt(limit)

using namespace std;
const int L1D_CACHE_SIZE = 32768;

void segmented_sieve(int limit) {
  int sqrt = (int) std::sqrt(limit);
  int segment_size = max(sqrt, L1D_CACHE_SIZE);

  int count = (limit < 2) ? 0 : 1;
  int s = 3, n = 3;

  char is_prime[sqrt + 1];
  memset(is_prime, 1, sizeof is_prime);

  for (int i = 2; i * i <= sqrt; ++i)
    if (is_prime[i])
      for (int j = i * i; j <= sqrt; j += i)
        is_prime[j] = 0;

  char sieve[segment_size];
  int primes[SZ], next[SZ], id = 0;

  for (int low = 0; low <= limit; low +=
  segment_size) {
    memset(sieve, 1, sizeof sieve);
    int high = min(low + segment_size - 1, limit);

    for (; s * s <= high; s += 2) {
      if (is_prime[s]) {
        primes[id] = (s);
        next[id++] = (s * s - low);
      }
    }

    for (int i = 0; i < id; ++i) {
      register int j = next[i];
      for (register int k = primes[i] << 1; j <
      segment_size; j += k)
        sieve[j] = 0;
      next[i] = j - segment_size;
    }

    for (; n <= high; n += 2)
      if (sieve[n - low])
        count++;
  }
  cout << count << endl;
}
```

5.8 Xor, And, Or convolutions

```
#define LOGN 20
#define MAXN (1 << LOGN)

int T[3][2][2][2] = {
  { { {1,  1}, {1, -1} }, { {1,  1}, {1, -1} } },
  { { {0,  1}, {1,  1} }, { {-1, 1}, {1,  0} } },
  { { {1,  1}, {1,  0} }, { {0,  1}, {1, -1} } }
};

void FFT(vector<ll> &a, int op, bool inverse =
false) {
  int u1 = T[op][inverse][0][0], v1 =
  T[op][inverse][0][1];
  int u2 = T[op][inverse][1][0], v2 =
  T[op][inverse][1][1];

  for(int b = 0; b < LOGN; b++)
    for(int i = 0; i < MAXN; i++)
      if((i & (1 << b)) == 0) {
        ll u = a[i], v  = a[i + (1 << b)];
        a[i]            = u*u1 + v*v1;
        a[i + (1 << b)] = u*u2 + v*v2;
      }

  if (op == 0 && inverse)
  for (int i=0; i<a.size(); i++)
    a[i] >>= LOGN;
}

/// op is 0 for XOR, 1 for AND, 2 for OR
vector<ll> convolution(vector<ll> a, vector<ll> b,
int op) {
  FFT(a, op, false);
  FFT(b, op, false);
  for(int i=0; i<a.size(); i++)
    a[i] = a[i] * b[i];
  FFT(a, op, true);
  return a;
}
```

5.9 Fast Subset Transformation

```
#define LOGN 20
#define MAXN (1 << LOGN)

void FST(vector<ll> &a, bool inverse = false) {
  for (int b = 0; b < LOGN; b++)
    for (int i = 0; i < MAXN; i++)
      if ((i & (1 << b)) == 0)
        a[i + (1 << b)] += a[i] * (inverse ? -1 :
        1);
}
```

5.10 NTT

```
const int mod = 7340033; // C*2^K+1
const int root = 5; // (root ^ root_pw) % MOD = 1
const int root_1 = 4404020; // Mod inverse of root
const int root_pw = 1<<20; // Max size

void FFT (vector<int> & a, bool invert) {
  int n = (int) a.size();

  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) {
    int wlen = invert ? root_1 : root;
    for (int i=len; i<root_pw; i<<=1)
      wlen = int (wlen * 1ll * wlen % mod);
    for (int i=0; i<n; i+=len) {
      int w = 1;
      for (int j=0; j<len/2; ++j) {
        int u = a[i+j],  v = int (a[i+j+len/2] *
        1ll * w % mod);
        a[i+j] = u+v < mod ? u+v : u+v-mod;
        a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
        w = int (w * 1ll * wlen % mod);
      }
    }
  }
  if (invert) {
    int nrev = reverse (n, mod);
    for (int i=0; i<n; ++i)
      a[i] = int (a[i] * 1ll * nrev % mod);
  }
}
```

5.11 Useful modulos for NTT

mod	root	root_1	root_pw
7340033	5	4404020	2^{20}
415236097	73362476	247718523	2^{22}
463470593	428228038	182429	2^{21}
998244353	15311432	469870224	2^{23}
918552577	86995699	324602258	2^{22}

5.12 FFT with any modulo

```
const int MOD = 1e9 + 7;
const int SQMOD = (int)sqrt(MOD);
const double PI = acos((double)-1.0);

void addeq(int &a, int b) { a += b; if(a >= MOD) a
-= MOD; }
int mul(int a, int b) { return (a * 1ll * b) %
MOD; }

struct cplex {
  double x, y;
  cplex(double _x = 0, double _y = 0) : x(_x),
  y(_y) {}
  inline cplex operator * (cplex b) { return
  cplex(x * b.x - y * b.y, x * b.y + y * b.x);}
  inline cplex operator + (cplex b) { return
  cplex(x + b.x, y + b.y); }
  inline cplex operator - (cplex b) { return
  cplex(x - b.x, y - b.y); }
  inline cplex conj() { return cplex(x, -y); }
  static cplex unitcircle (double x) { return
  cplex(cos(x), sin(x)); }
};

void fft(vector<cplex> &a) {
  int n = a.size();
  if(n < 2) return;
  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]);
  }
  vector<cplex> w(n / 2);
  w[0] = cplex(1);
  for(int len = 2; len <= n; len <<= 1) {
    for(int i = 1; i < len / 2; i++)
      w[i] = (i & (i - 1)) ? w[i & (i - 1)] * w[i
      & -i] : cplex::unitcircle(2 * i * PI / len);
    for(int i = 0; i < n; i += len) {
      for(int j = 0; j < len / 2; j++) {
        cplex u = a[i + j];
        cplex v = a[i + j + len / 2] * w[j];
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
      }
    }
  }
}

vector<int> multiply(vector<int> &a, vector<int>
&b) {
  int n = 1;
  while(n < a.size() + b.size() - 1) n <<= 1;
  vector<cplex> fa(n, 0), fb(n, 0);
  for(int i = 0; i < a.size(); i++)
    fa[i] = cplex(a[i] / SQMOD, a[i] % SQMOD);
  for(int i = 0; i < b.size(); i++)
    fb[i] = cplex(b[i] / SQMOD, b[i] % SQMOD);
  fft(fa);
```

```
  fft(fb);
  vector<cplex> ia(n), ib(n);
  for(int i = 0; i < n; i++) {
    int j = (n - i) & (n - 1);
    cplex a1 = (fa[i] + fa[j].conj()) * cplex(0.5,
    0);
    cplex a0 = (fa[i] - fa[j].conj()) * cplex(0,
    -0.5);
    cplex b1 = (fb[i] + fb[j].conj()) * cplex(0.5
    / n, 0);
    cplex b0 = (fb[i] - fb[j].conj()) * cplex(0,
    -0.5 / n);
    ia[j] = a1 * b1 + a0 * b0 * cplex(0, 1);
    ib[j] = a1 * b0 + a0 * b1;
  }
  fft(ia);
  fft(ib);
  vector<int> c(a.size() + b.size() - 1, 0);
  for(int i = 0; i < c.size(); i++) {
    addeq(c[i], mul((long long)(ia[i].x + 0.5) %
    MOD, mul(SQMOD, SQMOD)));
    addeq(c[i], mul((long long)(ib[i].x + 0.5) %
    MOD, SQMOD));
    addeq(c[i], (long long)(ia[i].y + 0.5) % MOD);
  }
  return c;
}
```

5.13 Chinese Remainder Theorem

Usage: Has answer if and only if $a[i] = a[j] \ mod \ gcd(n[i], n[j])$

```
ll inv_mod(ll a, ll mod) {
  ll x, y;
  xgcd(a, mod, x, y);
  return (x % mod + mod) % mod;
}

ll chinese_remainder(ll *a, ll *n, int size) {
  if (size == 1) return a[0];

  ll tmp = inv_mod(n[0], n[1]) * (a[1] - a[0]) %
  n[1];
  if(tmp < 0) tmp += n[1];

  ll a1 = a[1], g = __gcd(n[0], n[1]);
  a[1] = a[0] + n[0] / g * tmp;
  n[1] *= n[0] / g;
  ll ret = chinese_remainder(a + 1, n + 1, size -
  1);
  n[1] /= n[0] / g;
  a[1] = a1;
  return ret;
}
```

5.14 Gaussian Elimination

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

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

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

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

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

5.15 Gaussian Elimination Modulo Two

Usage: Returns lexicographically greatest answer. Careful: most significant bit must be $m - 1$

```
int gauss (vector < bitset<M> > &a, int n, int m,
bitset<M> &ans) {
  vector<int> which_row(m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    for (int i=row; i<n; ++i)
     if (a[i][col]) {
      swap (a[i], a[row]);
      break;
    }
    if (!a[row][col])
     continue;

   which_row[col] = row;
   for (int i=0; i<n; ++i)
     if (i != row && a[i][col])
      a[i] ^= a[row];
    ++row;
  }

  for(int i=0; i<n; i++)
    if(a[i][m] && a[i].count() == 1) return 0;

  for(int i = m-1; i >= 0; i--) {
    if(which_row[i] == -1) ans[i] = 1;
```

```
    else ans[i] = a[which_row[i]][m];
    if(ans[i]) for(int j = 0; j < n; j++)
      if(j != which_row[i] && a[j][i])
        a[j][m] = a[j][m] ^ 1;
  }
  return 1;
}
```

5.16 Simplex

```
typedef long long ll;
typedef long double ld;
const int inf = int(1e9) + int(1e5);
const ll infl = ll(2e18) + ll(1e10);

const int maxn = 505;
const int maxm = 505;

const ld eps = 1e-9;

bool eq(ld a, ld b) {
  return fabsl(a - b) < eps;
}

namespace Simplex {

ld D[maxm][maxn]; // [n+2][m+2]
int B[maxm];
int N[maxn];
ld x[maxn];
int n, m;

//x >= 0, Ax <= b, c^Tx -> max
void init(int _n, int _m, ld A[][maxn], ld *b, ld
*c) {
  n = _n, m = _m;
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      D[i][j] = -A[i][j];
    for (int i = 0; i < m; i++) {
      D[i][n] = 1;
      D[i][n + 1] = b[i];
    }
    for (int j = 0; j < n; j++) {
      D[m][j] = c[j];
      D[m + 1][j] = 0;
    }
    D[m][n + 1] = D[m][n] = D[m + 1][n + 1] = 0;
    D[m + 1][n] = -1;
    iota(B, B + m, n);
    iota(N, N + n, 0);
    N[n] = -1;
  }

  void pivot(int b, int nb) {
    assert(D[b][nb] != 0);
    ld q = 1. / -D[b][nb];
    D[b][nb] = -1;
    for (int i = 0; i < n+2; i++)
      D[b][i] *= q;
    for (int i = 0; i < m+2; i++) {
      if (i == b)
        continue;
      ld coef = D[i][nb];
      D[i][nb] = 0;
      for (int j = 0; j < n+2; j++)
        D[i][j] += coef * D[b][j];
    }
    swap(B[b], N[nb]);
  }

  bool betterN(int f, int i, int j) {
    if (eq(D[f][i], D[f][j]))
      return N[i] < N[j];
    return D[f][i] > D[f][j];
  }

  bool betterB(int nb, int i, int j) {
    ld ai = D[i][n + 1] / D[i][nb];
    ld aj = D[j][n + 1] / D[j][nb];
    if (eq(ai, aj))
      return B[i] < B[j];
    return ai > aj;
  }

  bool simplex(int phase) {
    int f = phase == 1 ? m : m + 1;
    while (true) {
      int nb = -1;
      for (int i = 0; i < n+1; i++) {
        if (N[i] == -1 && phase == 1)
          continue;
        if (nb == -1 || betterN(f, i, nb))
          nb = i;
      }
      if (D[f][nb] <= eps)
        return phase == 1;
      assert(nb != -1);

      int b = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][nb] >= -eps)
          continue;
        if (b == -1 || betterB(nb, i, b))
          b = i;
      }
      if (b == -1)
        return false;
      pivot(b, nb);
      if (N[nb] == -1 && phase == 2)
        return true;
    }
  }

  ld solve() {
    int b = -1;
    for (int i = 0; i < m; i++) {
      if (b == -1 || D[i][n + 1] < D[b][n + 1])
        b = i;
    }
    assert(b != -1);
    if (D[b][n + 1] < -eps) {
      pivot(b, n);
      if (!simplex(2) || D[m + 1][n + 1] < -eps)
        return -infl;
    }
```

```
    if (!simplex(1))
      return infl;

    for (int i = 0; i < n; i++)
      x[i] = 0;
    for (int i = 0; i < m; i++)
      if (B[i] < n)
        x[B[i]] = D[i][n + 1];

      return D[m][n + 1];
    }

} //Simplex

ld a[maxm][maxn];
ld b[maxm];
ld c[maxn];

int main() {
  int n, m;
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      cin >> a[i][j];
    cin >> b[i];
  }
  for (int i = 0; i < n; i++)
    cin >> c[i];
  Simplex::init(n, m, a, b, c);
  cout << Simplex::solve() << '\n';
  for (int i = 0; i < n; i++)
    cerr << Simplex::x[i] << ' ';
  cerr << '\n';
}
```

5.17 Linear Recurrence with Divide and Conquer

Usage: `dp[0] = x0; solve(0, N+1);`. Computes every $dp[i]$ in interval $[L, R)$.
Time Complexity: $O(N\ log^2\ N)$

```
ll dp[200020], coef[200020];

void solve(int l, int r) {
  if(l + 1 == r) return;
  int m = (l + r) / 2;

  solve(l, m);
  vector<int> a(dp + l, dp + m), b(coef, coef + (r
  - l + 1)), c;
  if(1LL * a.size() * b.size() < 2000) {
    c.resize(a.size() + b.size());
    for(int i = 0; i < a.size(); i++)
      for(int j = 0; j < b.size(); j++)
        c[i+j] = (c[i+j] + 1LL*a[i]*b[j]) % MOD;
  }
  else c = multiply(a, b);

  for(int i = m; i < r; i++) {
    dp[i] += c[i - l - 1];
    if(dp[i] >= MOD) dp[i] -= MOD;
  }
  solve(m, r);
}
```

5.18 Catalan Trapezoid

Usage: How many strings with x 0s and y 1s such that in every prefix 1s does not exceed 0s by m or more

```
ll trapezoid(ll x, ll y, ll m) {
  if(0 <= y && y < m)return nCr(x + y, y);
  else if(m <= y && y <= x + m - 1) return (nCr(x
  + y, y) - nCr(x + y, y - m) + MOD) % MOD;
  else return 0;
}
```

5.19 Random Walk

Usage: Probability p, position i, goal N

```
double random_walk(double p, int i, int N) {
  double q = 1 - p;
  if(fabs(p - q) < EPS) return 1.0 * i / N;
  return (1 - expo(q / p, i)) / (1 - expo(q / p,
  N));
}
```

6 Optimizations

6.1 Divide and Conquer Optimization

From $O(kn^2)$ to $O(kn \log n)$

$$dp[i][j] = \min_{k<j}(dp[i-1][k] + C[k][j]) \tag{1}$$

$$optK[i][j] \le optK[i][j+1] \tag{2}$$

Calculate for layers d, set $optK[d][0] = 0$ and $optK[d][n+1] = n$;
Calculate $optK[d][(l+r)/2]$, recursively call for two halves.

6.2 Knuth Optimization

From $O(n^3)$ to $O(n^2)$

$$dp[i][j] = \min_{i<k<j}(dp[i][k] + dp[k][j] + C[i][j]) \tag{3}$$

$$optK[i][j-1] \le optK[i][j] \le optK[i+1][j] \tag{4}$$

Calculate by layers $delta = j - i$, iterate k between $optK[i][j-1]$ and $optK[i+1][j]$;
Calculate $optK[i][j]$, repeat for $delta + 1$.

6.3 Convex Hull Trick

Usage: How many strings with x 0s and y 1s such that in every prefix 1s does not exceed 0s by m or more

```
struct CHT { // struct func -> y = ax + b
  deque<func> cht;
  bool bad(func a, func b, func c) {
    return a.intersec(c) <= a.intersec(b); // >=
    for max
  }
```

```
  void add(func f) { // only works if f.a is
  always decreasing
    while(cht.size() > 1 && bad(cht[cht.size()-2],
    cht.back(), f))
      cht.pop_back();
    cht.push_back(f);
  }
  double query(double x) { // only works if x is
  always increasing
    while(cht.size() > 1 && cht[0].y(x) >=
    cht[1].y(x)) cht.pop_front(); // <= for max
    return cht.front().y(x);
  }
};
```

6.4 Aliens Trick

Usage: Let K be the number of item choices or array partitions. Solve the problem without restriction on K (any number of partitions). Binary search a constant C to add in every partition, such that number of partitions will be K. C may be decimal, or negative.

Final answer will be $DP - C * K$, where DP is the answer without restriction on K.

If using double, use $l = -EPS$ instead of 0, to avoid precision issues

C may be large (greater than N or MAX - consider using $r = N * MAX$

```
ll l = 0, r = (1LL << 21), m;
pair<ll, int> ans; // ans.first: cost; ans.second:
#partitions
while(l < r) {
  m = (l + r + 1) / 2;
  // max/minimize ans.first, in case of draw,
  maximize ans.second
  ans = solve(m);
  if(ans.second >= k) {
    l = m;
    if(ans.second == k) break;
  }
  else r = m - 1;
}
ans = solve(l);
printf("%lld\n", ans.first + l * k);
```

6.5 1D1D Optimization

Usage:

$$dp[i] = \min_{j<i}(dp[j] + cost(j, i))$$

Keep list of intervals $optK$ where $optK[i]$ keeps the best option for i

```
dp[0] = 0;  // best if 1-indexed
deque<pair<int, int> > optK; // {position, best
option for that position}
optK.emplace_back(1, 0);
for(int i = 1; i <= n; i++) {
  while(optK.size() > 1 && optK[1].first <= i)
  optK.pop_front();
  dp[i] = dp[optK[0].second] +
  cost(optK[0].second, i);
  while(!optK.empty()) {
    int pos = optK.back().first, j =
    optK.back().second;
    if(pos > i && dp[j] + cost(j, pos) >= dp[i] +
    cost(i, pos)) optK.pop_back();
    else break;
  }
  if(optK.empty()) {
    optK.emplace_back(i+1, i);
  } else {
    int L = i + 1, R = n + 1, M, j =
    optK.back().second;
    while(L < R) {
      M = (L + R) / 2;
      if(dp[i] + cost(i, M) <= dp[j] + cost(j, M))
      R = M;
      else L = M + 1;
    }
    if(L <= n) optK.emplace_back(L, i);
  }
}
```

7 Geometry

7.1 2D Geometry

```
#define EPS 1e-9
typedef double T;

int cmp(T x, T y = 0) {
  return (x <= y + EPS)?(x + EPS < y)?-1:0:1;
}

struct point {
  T x, y;
  int id;
  point(T x = 0, T y = 0, int id = -1): x(x),
  y(y), id(id) {}
  point operator + (point b) { return point(x +
  b.x, y + b.y); }
  point operator - (point b) { return point(x -
  b.x, y - b.y); }
  point operator * (T c) { return point(x*c, y*c);
  }
  point operator / (T c) { return point(x/c, y/c);
  }
  bool operator < (const point b) const {
    return pair<T, T>(x, y) < pair<T, T>(b.x,
    b.y);
  }
};
typedef pair<point, point> segm;
typedef vector<point> polygon;

inline T distPoints(point a) {
  return sqrt(a.x * a.x + a.y * a.y);
}
/** Dot product **/
inline T escalar(point a, point b) {
  return a.x * b.x + a.y * b.y;
}
/** Cross product **/
inline T vetorial(point a, point b) {
```

```
  return a.x * b.y - a.y * b.x;
}
/** Counter-clockwise (o is the pivot)**/
int ccw(point p, point q, point o) {
  return cmp(vetorial(p - o, q - o));
}
T angle(point p, point q, point o) {
  point u = p - q, v = o - q;
  return atan2(vetorial(u, v), escalar(u, v));
}

struct line {
  T a, b, c;
  line(T a, T b, T c): a(a), b(b), c(c) {}
  line(point p1, point p2):
  a(p1.y - p2.y),
  b(p2.x - p1.x),
  c(p1.x*p2.y - p2.x*p1.y) {}
};
/** Distance from point to line **/
inline T point_line(point a, line l) {
  return abs(l.a*a.x + l.b*a.y + l.c) /
  sqrt(l.a*l.a + l.b*l.b);
}
/** Distance from point to segment **/
inline T point_segment(segm s, point a) {
  if(escalar(s.second - s.first, a - s.first) < 0)
    return distPoints(a - s.first);
  if(escalar(s.first - s.second, a - s.second) <
  0)
    return distPoints(a - s.second);
  return point_line(a, s.first, s.second);
}

bool parallel(line a, line b) {
  return abs(a.b * b.a - a.a * b.b) < EPS;
}
/** Distance between two lines **/
T distLines(line a, line b) {
  if(!parallel(a,b)) return 0;
  if(a.a == 0) return point_line(point(0,
  -a.c/a.b), b);
  return point_line(point(-a.c/a.a, 0), b);
}
/**Intersection between two lines **/
point intersection(line a, line b) {
  point ret;
  ret.x = (b.c * a.b - a.c * b.b) / (b.b * a.a -
  a.b * b.a);
  ret.y = (b.c * a.a - a.c * b.a) / (a.b * b.a -
  b.b * a.a);
  return ret;
}
bool equal_lines(line a, line b) {
  if(!parallel(a,b)) return false;
  return abs(distLines(a,b)) < EPS;
}
/** Line perpendicular to line A at point B **/
line perpendicular(line a, point b) {
  T la, lb, lc;
  la = a.b;
  lb = -a.a;
  lc = -la * b.x - lb * b.y;
  return line(la, lb, lc);
}

struct circle {
  point center;
  T rad;
  circle(point c, T r): center(c), rad(r) {}
  circle(point p1, point p2) {
    center = (p1 + p2) / 2;
    rad = distPoints(p2 - p1) / 2;
  }
  circle(point p1, point p2, point p3) {
    line l12 (p1, p2);
    line l23 (p2, p3);
    assert(!equal_lines(l12, l23));
    point p12 = (p1 + p2) / 2;
    point p23 = (p2 + p3) / 2;
    center = intersection(perpendicular(l12, p12),
    perpendicular(l23, p23));
    rad = distPoints(center - p1);
  }
};

T circle_intersec(T x1, T y1, T r1, T x2, T y2, T
r2) {
  T d = sqrtl((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
  if (d >= (r1+r2)) return 0;
  else if (max(r1, r2) >= d+min(r1,r2)) return
  PI*min(r1,r2)*min(r1,r2);
  else {
    T a1=2*acosl((d*d+r1*r1-r2*r2)/(2*d*r1));
    T a2=2*acosl((d*d+r2*r2-r1*r1)/(2*d*r2));
    T num1=(T)a1/2*r1*r1-r1*r1*sin(a1)*0.5;
    T num2=(T)a2/2*r2*r2-r2*r2*sin(a2)*0.5;
    return num1+num2;
  }
}

circle mec_solve1(vector<point> &p, int i, int j)
{
  circle c(p[i], p[j]);
  for(int k = 0; k < j; k++)
    if(cmp(distPoints(c.center - p[k]), c.rad) ==
    1)
      c = circle(p[i], p[j], p[k]);
  return c;
}
circle mec_solve2(vector<point> &p, int i) {
  circle c(p[0], p[i]);
  for(int j = 1; j < i; j++)
    if(cmp(distPoints(c.center - p[j]), c.rad) ==
    1)
      c = mec_solve1(p, i, j);
  return c;
}
circle mec_solve3(vector<point> &p) {
  circle c(p[0], p[1]);
  for(int i = 2; i < (int)p.size(); i++)
    if(cmp(distPoints(c.center - p[i]), c.rad) ==
    1)
      c = mec_solve2(p, i);
  return c;
}
```

```
circle minimum_enclosing_circle(vector<point> p) {
  if(p.size() == 1) return circle{p[0], 0};
  random_shuffle(p.begin(), p.end());
  return mec_solve3(p);
}
```

7.2 Polygons

```
/** Area of polygon **/
T polygon_area(vector <point> &points) {
  T area = vetorial(points.back(), points[0]);
  for(int i=1; i < (int)points.size(); i++)
    area += vetorial(points[i-1], points[i]);
  return abs(area) / 2;
}

/** Center of mass of a polygon with uniform mass
distribution **/
point polygon_centroid(vector <point> &poly) {
  point ret;
  T area = polygon_area(poly);
  ret.x = (poly.back().x + poly[0].x) *
  vetorial(poly.back(), poly[0]);
  ret.y = (poly.back().y + poly[0].y) *
  vetorial(poly.back(), poly[0]);
  for(int i = 1; i < (int)poly.size(); i++) {
    ret.x += (poly[i-1].x + poly[i].x) *
    vetorial(poly[i-1], poly[i]);
    ret.y += (poly[i-1].y + poly[i].y) *
    vetorial(poly[i-1], poly[i]);
  }
  ret.x /= (6 * area);
  ret.y /= (6 * area);
  return ret;
}

point pivot;
bool polar_cmp(point a, point b) {
  T cross = vetorial(a - pivot, b - pivot);
  return cmp(cross) == 1 || (cmp(cross) == 0 &&
   cmp(distPoints(pivot - a), distPoints(pivot -
   b)) == -1);
}

vector<point> convex_hull(vector<point> p, bool
repeat_last = false) {
  if(p.size() <= 2) return p;
  int pi = 0;
  for(int i = 1; i < p.size(); i++)
    if(p[i] < p[pi]) pi = i;
  swap(p[0], p[pi]); pivot = p[0];
  sort(p.begin()+1, p.end(), polar_cmp);
  vector<point> s;
  s.push_back(p.back()); s.push_back(p[0]);
  s.push_back(p[1]);
  for(int i = 2; i < p.size();) {
    int j = s.size()-1;
    if(s.size() == 2 || ccw(s[j], p[i], s[j-1]) ==
    1) s.push_back(p[i++]);
    else s.pop_back();
  }
  if(!repeat_last) s.pop_back();
  return s;
}

/// 0 - Border / 1 - Outside / -1 - Inside
int point_inside_polygon(point p, vector<point>
&poly) {
  int n = poly.size(), windingNumber = 0;
  for(int i = 0; i < n; i++) {
    if(distPoints(p - poly[i]) < EPS) return 0;
    int j = (i + 1) % n;
    if(cmp(poly[i].y, p.y) == 0 && cmp(poly[j].y,
    p.y) == 0) {
      if(cmp(min(poly[i].x, poly[j].x), p.x) <= 0
      &&
        cmp(p.x, max(poly[i].x, poly[j].x)) <= 0)
        return 0;
    }
    else {
      bool below = (cmp(poly[i].y, p.y) == -1);
      if(below != (cmp(poly[j].y, p.y) == -1)) {
        int orientation = ccw(p, poly[j],
        poly[i]);
        if(orientation == 0) return 0;
        if(below == (orientation > 0))
        windingNumber += below ? 1 : -1;
      }
    }
  }
  return windingNumber == 0 ? 1 : -1;
}

long long areaTriangle(const point &a, const point
&b, const point &c) {
  return (a.x * (b.y - c.y) + b.x * (c.y - a.y) +
  c.x * (a.y - b.y));
}

//Check if point is inside of a convex polygon in
O(log(n))
bool inConvexPoly(const vector < point >
&convPoly, point p) {
  long long start = 1, last = (int)
  convPoly.size() - 1;

  while (last - start > 1) {
    long long mid = (start + last) / 2;
    if (areaTriangle(convPoly[0], convPoly[mid],
    p) < 0) last = mid;
    else start = mid;
  }

  long long r0 = abs(areaTriangle(convPoly[0],
  convPoly[start], convPoly[last]));
  long long r1 = abs(areaTriangle(convPoly[0],
  convPoly[start], p));
  long long r2 = abs(areaTriangle(convPoly[0],
  convPoly[last], p));
  long long r3 = abs(areaTriangle(convPoly[start],
  convPoly[last], p));

  // if you need strictly inside
  long long r4 = areaTriangle(convPoly[0],
  convPoly[1], p);
```

```
  long long r5 = areaTriangle(convPoly[0],
  convPoly[convPoly.size() - 1], p);

  // if you need strictly inside, add '&& r3 != 0
  && r4 != 0 && r5 != 0' to return
  return r0 == (r1 + r2 + r3);
}
```

7.3 3D Geometry

```
struct point {
  double x, y, z;
  point() { x = y = z = 0.0; }
  point(double x, double y, double z) : x(x),
  y(y), z(z) {}
};

double dist(point a, point b) {
  return sqrt((a.x - b.x)*(a.x - b.x) + (a.y -
  b.y)*(a.y - b.y) + (a.z - b.z)*(a.z - b.z));
}

struct vec {
  double x, y, z;
  vec(double x, double y, double z) : x(x), y(y),
  z(z) {}
};

vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y, b.z - a.z);
}

vec scale(vec v, double s) {
  return vec(v.x * s, v.y * s, v.z * s);
}

point translate(point p, vec v) {
  return point(p.x + v.x, p.y + v.y, p.z + v.z);
}

vec add(vec p, vec v) {
  return vec(p.x + v.x, p.y + v.y, p.z + v.z);
}

double dot(vec a, vec b) { return a.x * b.x + a.y
* b.y + a.z * b.z; }

double norm_sq(vec v) { return v.x * v.x + v.y *
v.y + v.z * v.z; }

double distToLine(point p, point a, point b) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  point c = translate(a, scale(ab, u));
  return dist(p, c);
}

double distToLineS(point p, point a, point b) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) return dist(p, a);
  if (u > 1.0) return dist(p, b);
  return distToLine(p, a, b);
}

vec norm(vec a, vec b) {
  return vec(a.y * b.z - a.z * b.y, a.z * b.x -
  a.x * b.z, a.x * b.y - a.y * b.x);
}

double area(point a, point b, point c) {
  return sqrt(norm_sq(norm(toVec(a, b), toVec(a,
  c)))) / 2.0;
}

double inside(point p, point a, point b, point c)
{
  vec ab = toVec(a, b), ac = toVec(a, c);
  vec n = norm(ab, ac);
  n = scale(n, 1.0 / sqrt(norm_sq(n)));

  double d = -a.x*n.x - a.y*n.y - a.z*n.z;
  double dst = dot(n, vec(p.x, p.y, p.z)) + d;

  point pj = translate(p, scale(n, -dot(toVec(a,
  p), n)));

  double area1 = area(a, b, c), area2 = 0.0;
  area2 += area(a, b, pj) + area(b, c, pj) +
  area(c, a, pj);

  if (fabs(area1 - area2) < EPS) return fabs(dst);
  return -1.0;
}

double distseg(point p1, point p2, point p3, point
p4) {
  vec u = toVec(p1, p2), v = toVec(p3, p4), w =
  toVec(p3, p1);
  double a = dot(u, u), b = dot(u, v), c = dot(v,
  v);
  double d = dot(u, w), e = dot(v, w);
  double D = a*c - b*b;
  double sc, sN, sD = D;
  double tc, tN, tD = D;

  if (D < EPS) {
    sN = 0.0; sD = 1.0; tN = e; tD = c;
  } else {
    sN = (b*e - c*d);
    tN = (a*e - b*d);
    if (sN < 0.0) {
      sN = 0.0;
      tN = e;
      tD = c;
    } else if (sN > sD) {
      sN = sD;
      tN = e + b;
      tD = c;
    }
  }

  if (tN < 0.0) {
    tN = 0.0;
    if (-d < 0.0) sN = 0.0;
    else if (-d > a) sN = sD;
```

```
    else {
      sN = -d;
      sD = a;
    }
  } else if (tN > tD) {
    tN = tD;
    if ((-d + b) < 0.0) sN = 0;
    else if ((-d + b) > a) sN = sD;
    else {
      sN = (-d +  b);
      sD = a;
    }
  }

  sc = (abs(sN) < EPS ? 0.0 : sN / sD);
  tc = (abs(tN) < EPS ? 0.0 : tN / tD);

  vec dP = add(w, scale(u, sc));
  dP = add(dP, scale(v, -tc));

  return sqrt(norm_sq(dP));
}
```

7.4 Half Plane Intersection

```
bool comp(point a, point b){
  if((a.x > 0 || (a.x==0 && a.y>0) ) && (b.x < 0
  || (b.x==0 && b.y<0))) return 1;
  if((b.x > 0 || (b.x==0 && b.y>0) ) && (a.x < 0
  || (a.x==0 && a.y<0))) return 0;
  T R = vetorial(a, b);
  if(R) return R > 0;
  return escalar(a, a) < escalar(b, b);
}

namespace halfplane{
  struct L{
    point p,v;
    L(){}
    L(point P, point V):p(P),v(V){}
    bool operator<(const L &b)const{ return
    comp(v, b.v); }
  };
  vector<L> line;
  void addL(point a, point
  b){line.push_back(L(a,b-a));}
  bool left(point &p, L &l){ return
  cmp(vetorial(l.v, p-l.p)) > 0;}
  bool left_equal(point &p, L &l){ return
  cmp(vetorial(l.v, p-l.p)) >= 0;}
  void init(){ line.clear(); }

  point pos(L &a, L &b){
    point x=a.p-b.p;
    T t = vetorial(b.v, x)/ vetorial(a.v, b.v);
    return a.p+a.v*t;
  }

  polygon intersect(){
    sort(line.begin(), line.end());
    deque<L> q; //linhas da intersecao
    deque<point> p; //pontos de intersecao entre
    elas
    q.push_back(line[0]);

    for(int i = 1; i < (int)line.size(); i++) {
      while(q.size() > 1 && !left(p.back(),
      line[i]))
        q.pop_back(), p.pop_back();
      while(q.size() > 1 && !left(p.front(),
      line[i]))
        q.pop_front(), p.pop_front();
      if(!cmp(vetorial(q.back().v, line[i].v)) &&
      !left(q.back().p,line[i]))
        q.back() = line[i];
      else if(cmp(vetorial(q.back().v,
      line[i].v)))
        q.push_back(line[i]),
        p.push_back(point());
      if(q.size() > 1)
        p.back() = pos(q.back(),q[q.size()-2]);
    }

    while(q.size() > 1 &&
    !left(p.back(),q.front()))
      q.pop_back(), p.pop_back();
    if(q.size() <= 2) return polygon(); //Nao
    forma poligono (pode nao ter intersecao)
    if(!cmp(vetorial(q.back().v, q.front().v)))
    return polygon(); //Lados paralelos -> area
    infinita

    point ult = pos(q.back(),q.front());
    bool ok = 1;
    for(int i = 0; ok && i < (int)line.size();
    i++)
      if(!left_equal(ult,line[i])) ok = 0;

    if(ok) p.push_back(ult); //Se formar um
    poligono fechado
    return polygon(p.begin(), p.end());
  }
};
```

8 Cheat Sheet

8.1 Binomial Transform

If $a_n = \sum_{k=0}^{n} \binom{n}{k} b_k$, then $b_n = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} a_k$

8.2 Stars and Bars

N identical objects into K distinct groups $\binom{n+k-1}{n}$

8.3 Lucas' Theorem

$$\binom{n}{m} \equiv \binom{\lfloor \frac{n}{p} \rfloor}{\lfloor \frac{m}{p} \rfloor} \binom{n \bmod p}{m \bmod p} \pmod{p} \tag{5}$$

If $p = 2$, $\binom{n}{m} \equiv 0$ if in some bit $n_i < m_i$.

8.4 Derangement

Number of permutations with no fixed points

$$!n \;=\; (n-1)\cdot[!(n-1)+!(n-2)] \;=\; n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

8.5 Burnside's Lemma

- Square, C colors, rotations : $\frac{1}{4}(C^4 + C + C^2 + C)$
- Cube, C colors, rotations: $\frac{1}{24}(C^6 + 6C^3 + 3C^4 + 6C^3 + 8C^2)$

8.6 Summations

$$\sum_{k=0}^{n} k^2 \;=\; \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=0}^{n} k^3 \;=\; \frac{n^2(n+1)^2}{4}$$

$$\sum_{k=0}^{n} a^k \;=\; \frac{a^{n+1}-1}{a-1}$$

$$\sum_{k=0}^{\infty} a^k \;=\; \frac{1}{1-a}\;,\; |a|<1$$

$$\sum_{k=0}^{\infty} ka^k \;=\; \frac{a}{(1-a)^2}\;,\; |a|<1$$

$$\sum_{k=0}^{n-1} \frac{1}{k(k+1)} \;=\; 1-\frac{1}{n}$$

$$(x+y)^n \;=\; \sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k}$$

$$\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$$

$$\sum_{k=0}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$$

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

$$\sum_{m=k}^{n} \binom{m}{k} = \binom{n+1}{k+1}$$

$$\sum_{k=q}^{n} \binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$$

$$\sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$$

8.7 Stirling numbers of the first kind

$\left[{n \atop k}\right]$ is the number of $n-permutations$ with k cycles.
Base cases: $\left[{0 \atop 0}\right]=1$ and $\left[{0 \atop n}\right]=\left[{n \atop 0}\right]=0$

$$\left[{n+1 \atop k}\right] = n\left[{n \atop k}\right] + \left[{n \atop k-1}\right]$$

$$\sum_{k=0}^{n}\left[{n \atop k}\right] = n! = \left[{n+1 \atop 1}\right]$$

$$\sum_{k=0}^{n}\left[{n \atop k}\right]x^k = x(x+1)(x+2)\ldots(x+n-1)$$

Calculate the polynomial using FFT and Divide and Conquer, get entire row of Stirling numbers in $O(n\cdot log^2 n)$.

8.8 Stirling numbers of the second kind

$\left\{{n \atop k}\right\}$ is the number of ways to partition n distinct objects into k non-empty subsets.
Base cases: $\left\{{n \atop n}\right\}=1$ and $\left\{{n \atop 1}\right\}$, for $n\geq 1$

$$\left\{{n+1 \atop k}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$$

$$\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

Calculate entire row of Stirling numbers with FFT. Polynomials are:

$$a_i = \frac{-1^i}{i!} \text{ and } b_i = \frac{i^n}{i!}$$

$$\left\{{n \atop k}\right\} = \sum_{i=0}^{k} a_i\cdot b_{k-i}$$

n\k	0	1	2	3	4
0	1	0	0	0	0
1	1	1	0	0	0
2	1	2	1	0	0
3	1	3	3	1	0
4	1	4	6	4	1
5	1	5	10	10	5

Table 1: $\binom{n}{k}$

n\k	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	1	1	0	0
3	0	2	3	1	0
4	0	6	11	6	1
5	0	24	50	35	10

Table 2: $\left[{n \atop k}\right]$

n\k	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	1	1	0	0
3	0	1	3	1	0
4	0	1	7	6	1
5	0	1	15	25	10

Table 3: $\left\{{n \atop k}\right\}$

8.9 Useful Primes

311

317

$2\cdot 10^5+17$

$2\cdot 10^5+23$

10^9+9

10^9+21

$1.2\cdot 10^{18}+11$

$1.2\cdot 10^{18}+13$

8.10 Dates

```
import datetime
date_a = datetime.datetime(2018, 09, 15) # year
between 1 and 9999
index = datetime.date.toordinal(date_a)
date_b = datetime.date.fromordinal(index)
print date_a.day, date_a.month, date_a.year
print datetime.datetime.now().weekday() # monday
is 0
```

8.11 Partial sum of f

$$S_f(n) = \frac{1}{g(1)}\left(S_{f*g(n)} - \sum_{i=2}^{n} S_f\left(\left\lfloor\frac{n}{i}\right\rfloor\right)g(i)\right)$$