Contents

1 Basic

2 Data Structure

2.1 Heavy-Light Decomposition

```
int n, q, dfn = 0;
int val[maxn], sz[maxn], head[maxn], dep[maxn
    ], st[maxn * 4], par[maxn], loc[maxn], id[maxn];
vector<int> adj[maxn];

void dfs(int pos, int prev){
  sz[pos] = 1;
  if(prev != -1) adj[pos].erase
      (find(adj[pos].begin(), adj[pos].end(), prev));
  for(auto &x : adj[pos]){
    par[x] = pos, dep[x] = dep[pos] + 1;
    dfs(x, pos);
    sz[pos] += sz[x];
    if(sz[x] > sz[adj[pos][0]]) swap(x, adj[pos][0]);
  }
}
void decompose(int pos, int h){
  id[dfn++] = pos;
  head[pos] = h, loc[pos] = dfn - 1;
  // upd(loc[pos], val[pos]);
  for(auto x : adj[pos]){
    if(x == adj[pos][0]) decompose(x, h);
    else decompose(x, x);
  }
}
void build(){
  dfs(0, -1);
  decompose(0, 0);
  //build_segtree();
}
int solve(int a, int b){
  int ret = 0;
  while(head[a] != head[b]){
    if(dep[head[a]] > dep[head[b]]) swap(a, b);
    ret = max(ret, qry(loc[head[b]], loc[b]));
    b = par[head[b]];
  }
  if(dep[a] > dep[b]) swap(a, b);
  return max(ret, qry(loc[a], loc[b]));
}
```

2.2 Centroid Decomposition

```
vector<pll> adj[maxn];
ll dist[20][maxn]; // distance to kth-layer-parent
int sz[maxn], del[maxn], par[maxn], cdep[maxn];
ll cnt[maxn], sum[maxn], re[maxn]; // re: subtree->par
int n, q;

void dfssz(int pos, int prev){
    sz[pos] = 1;
    for(auto [x, w] : adj[pos]){
        if(del[x] || x == prev) continue;
        dfssz(x, pos);
        sz[pos] += sz[x];
    }
}
int get_centroid(int pos, int prev, int siz){
    for(auto [x, w] : adj[pos]){
        if(!del[x] && x != prev && sz[x] >
             siz / 2) return get_centroid(x, pos, siz);
    }
    return pos;
}
void get_dist(int pos, int prev, int layer){
    for(auto [x, w] : adj[pos]){
        if(del[x] || x == prev) continue;
        dist[layer][x] = dist[layer][pos] + w;
        get_dist(x, pos, layer);
    }
}
void cd(int pos, int layer = 1, int p = 0){
    dfssz(pos, -1);
    int cen = get_centroid(pos, -1, sz[pos]);
    del[cen] = 1;
    dist[layer][cen] = 0;
    cdep[cen] = layer;
    par[cen] = p;
    get_dist(cen, -1, layer);
    for(auto [x, w] : adj[cen]){
        if(!del[x]){
            cd(x, layer + 1, cen);
        }
    }
}
void upd(int p){
    for(int x = p, d = cdep[x]; d; x = par[x], d--){
        sum[x] += dist[d][p];
        re[x] += dist[d - 1][p];
        cnt[x] ++;
    }
}
ll qry(int p){
    ll pre = 0, ans = 0;
    for(int x = p, d = cdep[x]; d; x = par[x], d--){
        ans += sum
            [x] - re[x] + (cnt[x] - pre) * dist[d][p];
        pre = cnt[x];
    }
    return ans;
}
```

2.3 Link Cut Tree

```
struct LCT{
  int ch[maxn
      ][2], par[maxn], rev[maxn], xr[maxn], val[maxn];
  int get(int x){ return ch[par[x]][1] == x;}
  int isroot(int x){
      return ch[par[x]][0] != x && ch[par[x]][1] != x;}
  void push(int x){
    if(rev[x]){
      if(rs) swap(ch[rs][0], ch[rs][1]), rev[rs] ^= 1;
      if(ls) swap(ch[ls][0], ch[ls][1]), rev[ls] ^= 1;
      rev[x] = 0;
    }
  }
  void pull(int x){
    xr[x] = xr[ls] ^ xr[rs] ^ val[x];
  }
  void rotate(int x){
    int y = par[x], z = par[y], k = get(x);
    if(!isroot(y)) ch[z][ch[z][1] == y] = x;
    ch[y][k] = ch[x][!k], par[ch[x][!k]] = y;
    ch[x][!k] = y, par[y] = x;
    par[x] = z;
    pull(y), pull(x);
  }
  void update(int x){
    if(!isroot(x)) update(par[x]);
    push(x);
  }
  void splay(int x){
    update(x);
    for(int
        p = par[x]; !isroot(x); rotate(x), p = par[x]){
      if(!isroot(p)) rotate(get(p) == get(x) ? p : x);
    }
  }
  void access(int x){
    for(int p = 0; x != 0; p = x, x = par[x]){
```

```
      splay(x);
      ch[x][1] = p;
      pull(x);
    }
  }
  void make_root(int x){
    access(x);
    splay(x);
    swap(ls, rs);
    rev[x] ^= 1;
  }
  void link(int x, int y){
    make_root(x);
    splay(x);
    if(find_root(y) == x) return;
    par[x] = y;
  }
  void cut(int x, int y){
    make_root(x);
    access(y);
    splay(x);
    if(par[y] != x || ch[y][0]) return;
    ch[x][1] = par[y] = 0;
  }
  int find_root(int x){
    access(x);
    splay(x);
    push(x);
    while(ls) x = ls, push(x);
    splay(x);
    return x;
  }
  void split(int x, int y){
    make_root(x);
    access(y);
    splay(y);
  }
  void upd(int x, int y){
    access(x);
    splay(x);
    val[x] = y;
    pull(x);
  }
} st;
```

2.4 LiChaoST

```
struct line{
  ll m, k;
  line(){}
  line(ll _m, ll _k) : m(_m), k(_k){}
  ll val(ll x){ return m * x + k; }
};

struct node{
  line ans;
  node *l, *r;
  int siz;
  node(){}
  node(line l) : ans(l), l(nullptr), r(nullptr){ }
};
node sgt[maxn];

int root[maxn], cnt = 0;

struct segtree{
  node *rt;
  int n, siz;
  segtree() : n(maxc * 2), siz(0), rt(nullptr){}
  void insert(node* &k, int l, int r, line cur){
    if(!k){
      k = new node(cur);
      siz++;
      return;
    }
    if(l == r){
      if(k->ans.val(l) > cur.val(l)) k->ans = cur;
      return;
    }
    int m = (l + r) / 2;
    if(k->ans.val(m) > cur.val(m)) swap(k->ans, cur);
    if(cur.m > k->ans.m) insert(k->l, l, m, cur);
    else insert(k->r, m + 1, r, cur);
  }
  void insert
      (ll m, ll k) { insert(rt, 0, n, line(m, k)); }
  void insert(line l) { insert(rt, 0, n, l);}
  ll qry(node *k, int l, int r, int pos){
    if(!k) return INF;
    if(l == r) return k->ans.val(pos);
    int m = (l + r) / 2;
    return min(k->ans.val(pos), pos <= m ? qry
        (k->l, l, m, pos) : qry(k->r, m + 1, r, pos));
  }
  ll qry(int pos) { return qry(rt, 0, n, pos); }
};
```

2.5 Leftist Heap

```
struct LeftistTree{
  int cnt, rt[maxn
      ], lc[maxn * 20], rc[maxn * 20], d[maxn * 20];
  int v[maxn * 20];
  LeftistTree(){}
  int newnode(pll nd){
    cnt++;
    v[cnt] = nd;
    return cnt;
  }
  int merge(int x, int y){
    if(!x || !y) return x + y;
    if(v[x] > v[y]) swap(x, y);
    int p = ++cnt;
    lc[p] = lc[x], v[p] = v[x];
    rc[p] = merge(rc[x], y);
    if(d[lc[p]] < d[rc[p]]) swap(lc[p], rc[p]);
    d[p] = d[rc[p]] + 1;
    return p;
  }
} st;
```

2.6 Treap

```
struct node{
  int val, pri, c = 1;
  node *l, *r;
  node(int _val) :
       val(_val), pri(rand()), l(nullptr), r(nullptr){}
  void recalc();
} *rt;
int cnt(node *t){ return t ? t->c : 0;}
void node::recalc(){
  c = cnt(l) + cnt(r) + 1;
}
pair<node*, node*> split(node *t, int val){
  if(!t) return {nullptr, nullptr};
  if(cnt(t->l) < val){
    auto p = split(t->r, val - cnt(t->l) - 1);
    t->r = p.first;
    t->recalc();
    return {t, p.second};
  }
  else{
    auto p = split(t->l, val);
    t->l = p.second;
    t->recalc();
    return {p.first, t};
  }
}
node* merge(node *a, node *b){
  if(!a || !b) return a ? a : b;
  if(a->pri > b->pri){
    a->r = merge(a->r, b);
    a->recalc();
    return a;
  }
  else{
    b->l = merge(a, b->l);
    b->recalc();
    return b;
  }
}
node *insert(node *t, int k){
  auto [a, b] = split(t, k);
  return merge(merge(a, new node(k)), b);
}
node* remove(node *t, int k){
  auto [a, b] = split(t, k - 1);
  auto [b, c] = split(b, k);
  return merge(a, c);
}
```

2.7 pbds

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
// heap tags: paring/binary/binomial/rc_binomial/thin
template<typename T>
using pbds_heap=__gnu_pbds::prioity_queue<T,less<T>, \
                                    pairing_heap_tag>;
// pbds_heap::point_iterator
// x = pq.push(10); pq.modify(x, 87); a.join(b);
// tree tags: rb_tree_tag/ov_tree_tag/splay_tree_tag
template<typename T>
using ordered_set = tree<T, null_type, less<T>,
      rb_tree_tag, tree_order_statistics_node_update>;
// find_by_order, order_of_key
// hash tables: cc_hash_table/gp_hash_table
```

3 Graph

3.1 Dominator Tree

```cpp
int in[maxn], id[maxn], par[maxn], dfn = 0;
int mn[maxn], idom[maxn], sdom[maxn], ans[maxn];
int fa[maxn]; // dsu
int n, m;

struct edge{
  int to, id;
  edge(){}
  edge(int _to, int _id) : to(_to), id(_id){}
};
vector<edge> adj[3][maxn];

void dfs(int pos){
  in[pos] = ++dfn;
  id[dfn] = pos;
  for(auto [x, id] : adj[0][pos]){
    if(in[x]) continue;
    dfs(x);
    par[x] = pos;
  }
}

int find(int x){
  if(fa[x] == x) return x;
  int tmp = fa[x];
  fa[x] = find(fa[x]);
  if(in[sdom[mn[tmp]]] < in[sdom[mn[x]]]){
    mn[x] = mn[tmp];
  }
  return fa[x];
}

void tar(int st){
  dfs(st);
  for(int
       i = 0; i < n; i++) mn[i] = sdom[i] = fa[i] = i;
  for(int i = dfn; i >= 2; i--){
    int pos = id[i], res = INF; // res : in(x) of sdom
    for(auto [x, id] : adj[1][pos]){
      if(!in[x]) continue;
      find(x);
      if(in[pos] > in[x]) res = min(res, in[x]);
      else res = min(res, in[sdom[mn[x]]]);
    }
    sdom[pos] = id[res];
    fa[pos] = par[pos];
    adj[2][sdom[pos]].eb(pos, 0);
    pos = par[pos];
    for(auto [x, id] : adj[2][pos]){
      find(x);
      if(sdom[mn[x]] == pos){
        idom[x] = pos;
      }
      else{
        idom[x] = mn[x];
      }
    }
    adj[2][pos].clear();
  }
  for(int i = 2; i <= dfn; i++){
    int x = id[i];
    if(idom[x] != sdom[x]) idom[x] = idom[idom[x]];
  }
}
```

3.2 Maximum Clique

```cpp
struct MaximumClique{
    typedef bitset<maxn> bst;
    bst adj[maxn], empt;
    int p[maxn], n, ans;
    void init(int _n){
        n = _n;
        for(int i = 0; i < n; i++) adj[i].reset();
    }
    void BronKerbosch(bst R, bst P, bst X){
        if(P == empt && X == empt){
            ans = max(ans, (int)R.count());
            return;
        }
        bst tmp = P | X;
        if((R | P | X).count() <= ans) return;
        int u;
        for(int i = 0; i < n; i++){
            if(tmp[u = p[i]]) break;
        }
        bst lim = P & ~adj[u];
        for(int i = 0; i < n; i++){
            int v = p[i];
            if(lim[v]){
                R[v] = 1;
                BronKerbosch
                    (R, P & adj[v], X & adj[v]);
                R[v] = 0, P[v] = 0, X[v] = 1;
            }
        }
    }
    void add_edge(int a, int b){
        adj[a][b] = adj[b][a] = 1;
    }
    int solve(){
        bst R, P, X;
        ans = 0, P.flip();
        iota(p, p + n, 0);
        random_shuffle
            (p, p + n), BronKerbosch(R, P, X);
        return ans;
    }
};
```

4 Flow / Matching

4.1 Dinic

```cpp
struct Dinic{
  struct edge{
    ll to, cap;
    edge(){}
    edge(int _to, ll _cap) : to(_to), cap(_cap){}
  };
  vector<edge> e;
  vector<vector<int>> adj;
  vector<int> iter, level;
  int n, s, t;
  void init(int _n, int _s, int _t){
    n = _n, s = _s, t = _t;
    adj = vector<vector<int>>(n);
    iter = vector<int>(n);
    level = vector<int>(n);
    e.clear();
  }
  void add_edge(int from, int to, ll cap){
    adj[from].pb(e.size()), adj[to].pb(e.size() + 1);
    e.pb(edge(to, cap)), e.pb(edge(from, 0));
  }
  void bfs(){
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
      int cur = q.front(); q.pop();
      for(auto id : adj[cur]){
        auto [to, cap] = e[id];
        if(level[to] == -1 && cap){
          level[to] = level[cur] + 1;
          q.push(to);
        }
      }
    }
  }
  ll dfs(int pos, ll flow){
    if(pos == t) return flow;
    for(int &i = iter[pos]; i < adj[pos].size(); i++){
```

```
      auto [to, cap] = e[adj[pos][i]];
      if(level[to] == level[pos] + 1 && cap){
        ll tmp = dfs(to, min(flow, cap));
        if(tmp){
          e[adj[pos][i]].cap -= tmp;
          e[adj[pos][i] ^ 1].cap += tmp;
          return tmp;
        }
      }
    }
    return 0;
  }
} flow;
```

4.2 Min Cost Max Flow

```
struct MCMF{
    using T = ll;
    struct edge{
        int to;
        T cap, cost;
        edge(){}
        edge(int _to, T _cap, T
             _cost) : to(_to), cap(_cap), cost(_cost){}
    };
    vector<edge> e;
    vector<vector<int>> adj;
    vector<int> iter, inq;
    vector<T> dist;
    int n, s, t;
    void init(int _n, int _s, int _t){
        n = _n, s = _s, t = _t;
        adj = vector<vector<int>>(n);
        iter = vector<int>(n);
        dist = vector<T>(n);
        inq = vector<int>(n);
        e.clear();
    }
    void add_edge(int from, int to, T cap, T cost = 0){
        adj[from
            ].pb(e.size()), adj[to].pb(e.size() + 1);
        e.pb(edge(to
            , cap, cost)), e.pb(edge(from, 0, -cost));
    }
    bool spfa(){
        fill(dist.begin(), dist.end(), INF);
        queue<int> q;
        q.push(s);
        dist[s] = 0, inq[s] = 1;
        while(!q.empty()){
            int pos = q.front(); q.pop();
            inq[pos] = 0;
            for(auto id : adj[pos]){
                auto [to, cap, cost] = e[id];
                if(cap && dist[to] > dist[pos] + cost){
                    dist[to] = dist[pos] + cost;
                    if(!inq
                        [to]) q.push(to), inq[to] = 1;
                }
            }
        }
        return dist[t] != INF;
    }
    T dfs(int pos, T flow){
        if(pos == t) return flow;
        inq[pos] = 1;
        for(int
             &i = iter[pos]; i < adj[pos].size(); i++){
            auto [to, cap, cost] = e[adj[pos][i]];
            if(!inq[to] &&
                 dist[to] == dist[pos] + cost && cap){
                T tmp = dfs(to, min(flow, cap));
                if(tmp){
                    inq[pos] = 0;
                    e[adj[pos][i]].cap -= tmp;
                    e[adj[pos][i] ^ 1].cap += tmp;
                    return tmp;
                }
            }
        }
        inq[pos] = 0;
        return 0;
    }
    pair<T, T> mcmf(){
        T flow = 0, cost = 0;
        while(true){
            if(!spfa()) break;
            fill(iter.begin(), iter.end(), 0);
            T tmp;
            while((tmp = dfs(s, INF)) > 0){
                flow += tmp, cost += tmp * dist[t];
            }
        }
        return {flow, cost};
    }
} flow;
```

4.3 Gomory Hu

```
void Gomory_Hu_Tree(vector<int> st){
  if(st.size() <= 1) return;
  int s = st[0], t = st[1];
  flow.init(n, s, t);
  for(auto [a, b, w] : e) flow.add_edge(a, b, w);
  int cost = flow.flow();
  flow.bfs();
  adj[s].eb(t, cost), adj[t].eb(s, cost);
  vector<int> a, b;
  for(auto x : st){
    if(flow.level[x] == -1) a.pb(x);
    else b.pb(x);
  }
  Gomory_Hu_Tree(a);
  Gomory_Hu_Tree(b);
}
```

4.4 SW Min Cut

```
int edge[maxn][maxn], par[maxn], siz[maxn];
int dist[maxn], vis[maxn], done[maxn];
int n, m;
int root(int x)
{ return x == par[x] ? x : par[x] = root(par[x]); }
int contract(int &s, int &t){
  memset(dist, 0, sizeof(dist));
  memset(vis, 0, sizeof(vis));
  int mincut = INF, id, maxc;
  for(int i = 0; i < n; i++){
    id = maxc = -1;
    for(int j = 0; j < n; j++){
      if(!done[j] && !vis[j] && dist[j] > maxc){
        id = j;
        maxc = dist[j];
      }
    }
    if(id == -1) return mincut;
    s = t, t = id;
    mincut = maxc;
    vis[id] = true;
    for(int j = 0; j < n; j++){
      if(!done[j] && !vis[j]) dist[j] += edge[id][j];
    }
  }
  return mincut;
}
int Stoer_Wagner(){
  int mincut = INF, s, t, tmp;
  for(int i = 1; i < n; i++){
    tmp = contract(s, t);
    done[t] = true;
    mincut = min(mincut, tmp);
    if(!mincut) return 0;
    for(int j = 0; j < n; j++){
      if(!done
          [j]) edge[s][j] = (edge[j][s] += edge[j][t]);
    }
  }
  return mincut;
}
```

4.5 Hopcroft Karp

```
int mx[maxn], my[maxn], dx[maxn], dy[maxn], vis[maxn];
vector<int> adj[maxn];
int l, r, m;

int dfs(int pos){
    for(auto x : adj[pos]){
        if(!vis[x] && dy[x] == dx[pos] + 1){
            vis[x] = 1;
            if(my[x] != -1 && dy[x] == lim) continue;
            if(my[x] == -1 || dfs(my[x])){
                my[x] = pos, mx[pos] = x;
                return true;
```

```
            }
        }
    }
    return false;
}

int bfs(){
    fill(dx, dx + l, -1);
    fill(dy, dy + r, -1);
    queue<int> q;
    for(int i = 0; i < l; i++){
        if(mx[i] == -1) dx[i] = 0, q.push(i);
    }
    lim = INF;
    while(!q.empty()){
        int pos = q.front(); q.pop();
        if(dx[pos] > lim) break;
        for(auto x : adj[pos]){
            if(dy[x] == -1){
                dy[x] = dx[pos] + 1;
                if(my[x] == -1) lim = dy[x];
                else dx
                    [my[x]] = dy[x] + 1, q.push(my[x]);
            }
        }
    }
    return lim != INF;
}

void Hopcroft_Karp(){
    int res = 0;
    for(int i = 0; i < l; i++) mx[i] = -1;
    for(int i = 0; i < r; i++) my[i] = -1;
    while(bfs()){
        fill(vis, vis + l + r, 0);
        for(int i = 0; i < l; i++){
            if(mx[i] == -1 && dfs(i)) res++;
        }
    }
}
```

4.6 Kuhn Munkres

```
struct Hungarian{
    using T = ll;
    vector<T> lx, ly, slack;
    vector<int> vx, vy, match;
    vector<vector<T>> w;
    queue<int> q;
    int n;
    void init(int _n){
        n = _n;
        lx.resize(n), ly.resize(n), slack.resize(n);
        vx.resize
            (n), vy.resize(n), match.resize(n, -1);
        w.resize(n, vector<T>(n));
    }
    void inp(int x, int y, int val){
        w[x][y] = val;
        lx[x] = max(lx[x], val);
    }
    int dfs(int x){
        if(vx[x]) return false;
        vx[x] = 1;
        for(int i = 0; i < n; i++){
            if(lx[x] + ly[i] == w[x][i] && !vy[i]){
                vy[i] = true;
                if(match[i] == -1 || dfs(match[i])){
                    match[i] = x;
                    return true;
                }
            }

        }
        return false;
    }
    int pdfs(int x){
        fill(vx.begin(), vx.end(), 0);
        fill(vy.begin(), vy.end(), 0);
        return dfs(x);
    }
    void upd(int x){
        for(int i = 0; i < n; i++){
            if(!slack[i]) continue;
            slack[i] =
                min(slack[i], lx[x] + ly[i] - w[x][i]);
            if(!slack[i] && !vy[i]) q.push(i);
        }
    }
    void relabel(){
        T mn = numeric_limits<T>::max() / 3;
        for(int i = 0; i < n; i++){
            if(!vy[i]) mn = min(mn, slack[i]);
        }
        for(int i = 0; i < n; i++){
            if(vx[i]) lx[i] -= mn;
            if(vy[i]) ly[i] += mn;
            else{
                slack[i] -= mn;
                if(!slack[i]) q.push(i);
            }
        }
    }
    auto solve(){
        for(int i = 0; i < n; i++){
            if(pdfs(i)) continue;
            while(!q.empty()) q.pop();
            fill(slack.begin(), slack.end(), INF);
            for(int
                 j = 0; j < n; j++) if(vx[j]) upd(j);
            int ok = 0;
            while(!ok){
                relabel();
                while(!q.empty()){
                    int j = q.front(); q.pop();
                    if(match[j] == -1){
                        pdfs(i);
                        ok = 1;
                        break;
                    }
                    vy[j] = vx
                        [match[j]] = 1, upd(match[j]);
                }
            }
        }
        T ans = 0;
        for(int i = 0; i < n; i++){
            ans += w[match[i]][i];
        }
        for(int i = 0; i < n; i++) lx[match[i]] = i;
        return make_pair(ans, lx);
    }
} h;
```

4.7 General Graph Matching

```
struct Matching { // 0-based
  int n, tk;
  vector <vector <int>> g;
  vector <int> fa, pre, match, s, t;
  queue <int> q;
  int Find(int u) {
    return u == fa[u] ? u : fa[u] = Find(fa[u]);
  }
  int lca(int x, int y) {
    tk++;
    x = Find(x), y = Find(y);
    for (; ; swap(x, y)) {
      if (x != n) {
        if (t[x] == tk) return x;
        t[x] = tk;
        x = Find(pre[match[x]]);
      }
    }
  }
  void blossom(int x, int y, int l) {
    while (Find(x) != l) {
      pre[x] = y, y = match[x];
      if (s[y] == 1) q.push(y), s[y] = 0;
      if (fa[x] == x) fa[x] = l;
      if (fa[y] == y) fa[y] = l;
      x = pre[y];
    }
  }
  bool bfs(int r) {
    iota(all(fa), 0), fill(all(s), -1);
    while (!q.empty()) q.pop();
    q.push(r);
    s[r] = 0;
    while (!q.empty()) {
      int x = q.front(); q.pop();
      for (int u : g[x]) {
        if (s[u] == -1) {
          pre[u] = x, s[u] = 1;
```

```
        if (match[u] == n) {
          for (int a = u, b =
               x, last; b != n; a = last, b = pre[a])
            last =
                match[b], match[b] = a, match[a] = b;
          return true;
        }
        q.push(match[u]);
        s[match[u]] = 0;
      } else if (!s[u] && Find(u) != Find(x)) {
        int l = lca(u, x);
        blossom(x, u, l);
        blossom(u, x, l);
      }
    }
  }
  return false;
}
int solve() {
  int res = 0;
  for (int x = 0; x < n; ++x) {
    if (match[x] == n) res += bfs(x);
  }
  return res;
}
void add_edge(int u, int v) {
  g[u].push_back(v), g[v].push_back(u);
}
Matching (int _n) : n(_n), tk(0), g(n), fa(n + 1),
  pre(n + 1, n), match(n + 1, n), s(n + 1), t(n) {}
};
```

4.8 Weighted General Graph Matching

```
struct WeightGraph { // 1-based
  static const int inf = INT_MAX;
  static const int maxn = 514;
  struct edge {
    int u, v, w;
    edge(){}
    edge(int u, int v, int w): u(u), v(v), w(w) {}
  };
  int n, n_x;
  edge g[maxn * 2][maxn * 2];
  int lab[maxn * 2];
  int match[maxn *
       2], slack[maxn * 2], st[maxn * 2], pa[maxn * 2];
  int flo_from
      [maxn * 2][maxn + 1], S[maxn * 2], vis[maxn * 2];
  vector<int> flo[maxn * 2];
  queue<int> q;
  int e_delta(const edge &e) {
      return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
  void update_slack
      (int u, int x) { if (!slack[x] || e_delta(g[
      u][x]) < e_delta(g[slack[x]][x])) slack[x] = u; }
  void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
      if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
        update_slack(u, x);
  }
  void q_push(int x) {
    if (x <= n) q.push(x);
    else for (size_t i
        = 0; i < flo[x].size(); i++) q_push(flo[x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n) for (size_t i = 0;
         i < flo[x].size(); ++i) set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[
        b].begin(), flo[b].end(), xr) - flo[b].begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int)flo[b].size() - pr;
    }
    return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i
        < pr; ++i) set_match(flo[u][i], flo[u][i ^ 1]);
    set_match(xr, v);
    rotate(flo[
        u].begin(), flo[u].begin() + pr, flo[u].end());
  }
  void augment(int u, int v) {
    for (; ; ) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (!xnv) return;
      set_match(xnv, st[pa[xnv]]);
      u = st[pa[xnv]], v = xnv;
    }
  }
  int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u || v; swap(u, v)) {
      if (u == 0) continue;
      if (vis[u] == t) return u;
      vis[u] = t;
      u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x && st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo
          [b].push_back(y = st[match[x]]), q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo
          [b].push_back(y = st[match[x]]), q_push(y);
    set_st(b, b);
    for (int x
         = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w ==
            0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
  }
  void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
      set_st(flo[b][i], flo[b][i]);
    int xr =
        flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (!
          lca) return augment(u,v), augment(v,u), true;
```

```
      else add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
      if (st[x] == x
          && !match[x]) pa[x] = 0, S[x] = 0, q_push(x);
    if (q.empty()) return false;
    for (; ; ) {
      while (q.size()) {
        int u = q.front(); q.pop();
        if (S[st[u]] == 1) continue;
        for (int v = 1; v <= n; ++v)
          if (g[u][v].w > 0 && st[u] != st[v]) {
            if (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else update_slack(u, st[v]);
          }
      }
      int d = inf;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b]
            == b && S[b] == 1) d = min(d, lab[b] / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x]) {
          if (S[x] ==
              -1) d = min(d, e_delta(g[slack[x]][x]));
          else if (S[x] == 0)
              d = min(d, e_delta(g[slack[x]][x]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1) lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0) lab[b] += d * 2;
          else if (S[st[b]] == 1) lab[b] -= d * 2;
        }
      q = queue<int>();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && slack[x] && st[slack
            [x]] != x && e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge
              (g[slack[x]][x])) return true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b && S
            [b] == 1 && lab[b] == 0) expand_blossom(b);
    }
    return false;
  }
  pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int
         u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
      }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
  }
  void add_edge(int ui, int
       vi, int wi) { g[ui][vi].w = g[vi][ui].w = wi; }
  void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v)
        g[u][v] = edge(u, v, 0);
  }
};
```

5 String

5.1 Z-Value

```
vector<int> z(string s){
    vector<int> z(s.size());
    int x = 0, y = 0;
    for(int i = 1; i < s.size(); i++){
        z[i] = max(0LL, min(z[i - x], y - i));
        while(i +
            z[i] < s.size() && s[i + z[i]] == s[z[i]]){
            x = i, y = i + z[i], z[i]++;
        }
    }
    return z;
}
```

5.2 KMP

```
vector<int> KMP(string s){
    vector<int> f(s.size());
    for(int i = 1; i < s.size(); i++){
        f[i] = f[i] - 1;
        while(f
            [i] && s[i] != s[f[i]]) f[i] = f[f[i] - 1];
        if(s[f[i]] == s[i]) f[i]++;
    }
    return f;
}
```

5.3 Manacher

```
vector<int> manacher(string s){
    int n = 2 * s.size() + 1;
    string ss(n, '#');
    for(int
         i = 0; i < n / 2; i++) ss[i * 2 + 1] = s[i];
    swap(s, ss);
    vector<int> f(n);
    int m = 0, len = 0;
    for(int i = 0; i < n; i++){
        f[i]
            = max(0LL, min(f[m + m - i], m + len - i));
        while(i + f[i] < n && i
            - f[i] >= 0 && s[i + f[i]] == s[i - f[i]]){
            m = i, len = f[i], f[i]++;
        }
    }
    return f;
}
```

5.4 Suffix Array

```
struct SuffixArray{
    int ch[2][maxn], sa[maxn], cnt[maxn], n;
    string s;
    void init(string _s){
        s = _s, n = s.size();
        Get_SA();
        Get_LCP();
    }
    void Get_SA(){
        int *x = ch[0], *y = ch[1], m = 256;
        for(int i = 0; i < m; i++) cnt[i] = 0;
        for(int i = 0; i < n; i++) cnt[x[i] = s[i]]++;
        for(int
             i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for(int i = 0; i < n; i++) sa[--cnt[x[i]]] = i;
        for(int k = 1;; k <<= 1){
            for(int i = 0; i < m; i++) cnt[i] = 0;
            for(int i = 0; i < n; i++) cnt[x[i]]++;
            for(int i
                 = 1; i < m; i++) cnt[i] += cnt[i - 1];
            int p = 0;
            for(int i = n - k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n;
                i++) if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = n - 1;
                i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
            y[sa[0]] = p = 0;
            for(int i = 1; i < n; i++){
                int a = sa[i], b = sa[i - 1];
                if(a + k < n && b + k < n && x[a
                    ] == x[b] && x[a + k] == x[b + k]);
                else p++;
                y[a] = p;
            }
            if(p == n - 1) break;
```

```
            swap(x, y);
            m = p + 1;
        }
    }
    int rnk[maxn], lcp[maxn];
    void Get_LCP(){
        for(int i = 0; i < n; i++) rnk[sa[i]] = i;
        int val = 0;
        for(int i = 0; i < n; i++){
            if(val) val--;
            if(!rnk[i]){
                lcp[0] = val = 0;
                continue;
            }
            int b = sa[rnk[i] - 1];
            while(b + val < n && i + val
                < n && s[b + val] == s[i + val]) val++;
            lcp[rnk[i]] = val;
        }
    }
} sa;
```

5.5 Suffix Automaton

```
struct SuffixAutomaton{
  int len[maxn], link[maxn]; // maxn >= 2 * n - 1
    map<char, int> nxt[maxn];
    int cnt[maxn], distinct[maxn];
  bool is_clone[maxn];
    int first_pos[maxn];
    vector<int> inv_link[maxn]; //suffix references
  int sz = 1, last = 0;
  void init(string s){
    link[0] = -1;
    for(auto x : s) sa_extend(x);
  }
  void sa_extend(char c){
    int cur = sz++;
        cnt[cur] = 1;
    len[cur] = len[last] + 1;
    first_pos[cur] = len[cur] - 1;
    int p = last;
    while(p != -1 && !nxt[p].count(c)){
      nxt[p][c] = cur;
      p = link[p];
    }
    if(p == -1) link[cur] = 0;
    else{
      int q = nxt[p][c];
      if(len[q] == len[p] + 1) link[cur] = q;
      else{
        int clone = sz++;
        is_clone[clone] = true;
        first_pos[clone] = q;
        len[clone] = len[p] + 1;
        nxt[clone] = nxt[q];
        link[clone] = link[q];
        while(p != -1 && nxt[p][c] == q) {
          nxt[p][c] = clone;
          p = link[p];
        }
        link[cur] = link[q] = clone;
      }
    }
    last = cur;
  }
  ll getDistinct(int pos){ // number
      of distinct substr. starting at pos(inc. empty)
    if(distinct[pos]) return distinct[pos];
    distinct[pos] = 1;
    for(auto [c, next]
        : nxt[pos]) distinct[pos] += getDistinct(next);
    return cnt[pos];
  }
  ll numDistinct(){
    return getDistinct
        (0) - 1; // excluding an empty string
  }
  ll numDistinct2(){
    ll tot = 0;
    for(int i
        = 1; i < sz; i++) tot += len[i] - len[link[i]];
    return tot;
  }
    void compute_cnt(){ // endpos set size
    vector<vector<int>> v(sz);
    for(int i = 1; i < sz; i++) v[len[i]].pb(i);
    for(int
        i = sz - 1; i > 0; i--) for(auto x : v[i]) {
      cnt[link[x]] += cnt[x];
    }
    }
  string distinct_kth(ll k){
        // substring
            kth (not distinct) -> compute_cnt()
    numDistinct();
    string s;
    ll cur = 0, tally = 0;
    while(tally < k){
      for(auto [c, next] : nxt[cur]){
        if(tally + distinct[next] >= k){
          tally += 1;
          s += c;
          cur = next;
          break;
        }
        tally += distinct[next];
      }
    }
    return s;
  }
  //inverse links
  void genLink(){
      for(int i = 1; i < sz; i++){
            inv_link[link[i]].pb(i);
      }
  }
  void get_all_occur(vector<int>& oc, int v){
      if(!is_clone[v]) oc.pb(first_pos[v]);
      for(auto u : inv_link[v]) get_all_occur(oc, u);
  }
  vector<int> all_occ(string s){ // get all occ of s
      int cur = 0;
      for(auto x : s){
          if(!nxt[cur].count(x)) return {};
          cur = nxt[cur][x];
      }
      vector<int> oc;
      get_all_occur(oc, cur);
      for(auto &x : oc
          ) x += 1 - s.length(); // starting positions
      sort(oc.begin(), oc.end());
      return oc;
  }
  int lcs(string t){
    int v = 0, l = 0, ans = 0;
    for(auto x : t){
      while(v && !nxt[v].count(x)){
        v = link[v];
        l = len[v];
      }
      if(nxt[v].count(x)){
        v = nxt[v][x];
        l++;
      }
      ans = max(ans, l);
    }
    return ans;
  }
};
```

5.6 Palindrome Tree

```
struct EERTREE{
  int sz, tot, last;
  int cnt[maxn], ch[maxn][26],
       len[maxn], fail[maxn], dif[maxn], slink[maxn];
  int g[maxn], dp[maxn];
  char s[maxn];
  int node(int l){
    sz++;
    memset(ch[sz], 0, sizeof(ch[sz]));
    len[sz] = l;
    fail[sz] = cnt[sz] = 0;
    return sz;
  }
  void init(){
    sz = -1;
    last = 0;
    s[tot = 0] = '$';
    node(0);
    node(-1);
    fail[0] = 1;
  }
```

```
  int getfail(int x){
    while(s[tot - len[x] - 1] != s[tot]) x = fail[x];
    return x;
  }
  void insert(char c){
    s[++tot] = c;
    int now = getfail(last);
    if(!ch[now][c - 'a']){
      int x = node(len[now] + 2);
      fail[x] = ch[getfail(fail[now])][c - 'a'];
      ch[now][c - 'a'] = x;
      dif[x] = len[x] - len[fail[x]];
      if(dif[x] == dif[fail[x]]){
        slink[x] = slink[fail[x]];
      }
      else slink[x] = fail[x];
    }
    last = ch[now][c - 'a'];
    cnt[last]++;
  }
  int process
      (string s){ // minimum palindrome partitioning
    for(int i = 0; i < s.size(); i++){
      insert(s[i]);
      dp[i] = INF;
      for(int x = last; x > 1; x = slink[x]){
        if(i - len[slink[x]] - dif[x] >=
              0) g[x] = dp[i - len[slink[x]] - dif[x]];
        if(dif[x] ==
            dif[fail[x]]) g[x] = min(g[x], g[fail[x]]);
        dp[i] = min(dp[i], g[x] + 1);
      }
    }
    return dp[s.size() - 1];
  }
} pam;
```

5.7 AC Automaton

```
namespace AC{
  int ch[maxn][26],
      fail[maxn], idx[maxn], last[maxn], pt[maxn];
  int val[maxn], cnt[maxn], tot = 0;
    // val[i] = # of times node
        (i) is visited, cnt[i] = # of occ. of str(i)
  void init(){
    memset(ch,
        0, sizeof(ch)), memset(fail, 0, sizeof(fail));
    memset(idx,
        0, sizeof(idx)), memset(last, 0, sizeof(last));
    memset(val
        , 0, sizeof(val)), memset(cnt, 0, sizeof(cnt));
    tot = 0;
  }
  void insert(string &s, int id){ // id is 1-based
    int cur = 0;
    for(int i = 0; i < s.size(); i++){
      if(!ch[cur
          ][s[i] - 'a']) ch[cur][s[i] - 'a'] = ++tot;
      cur = ch[cur][s[i] - 'a'];
    }
    if(idx[cur] == 0) idx[cur] = id;
    else pt[id] = idx[cur];
  }
  void build(){
    queue<int> q;
    for(int i = 0; i < 26; i++){
      if(ch[0][i]) q.push(ch[0][i]);
    }
    while(!q.empty()){
      int u = q.front(); q.pop();
      for(int i = 0; i < 26; i++){
        if(ch[u][i]) {
          fail[ch[u][i]] = ch[fail[u]][i];
          q.push(ch[u][i]);
        }
        else ch[u][i] = ch[fail[u]][i];
        last[ch[u][i]] = idx[fail[ch[u][i]]]
            ? fail[ch[u][i]] : last[fail[ch[u][i]]];
      }
    }
  }
  int qry(string &s){
    int u = 0, ret = 0;
    for(int i = 0; i < s.size(); i++){
      u = ch[u][s[i] - 'a'];
      for(int j = u; j; j = last[j]) val[j] ++;
    }
    for(int i = 0; i <= tot; i++){
      if(idx[i])
          ret = max(ret, val[i]), cnt[idx[i]] = val[i];
    }
    return ret;
  }
};
```

5.8 Lyndon Factorization

```
vector<string> duval(string s){
  int n = s.length(), i = 0;
    vector<string> fac;
  while(i < n){
    int j = i + 1, k = i; // i <= k < j
    while(j < n && s[k] <= s[j]){
      if(s[k] < s[j]) k = i;
      else k++;
      j++;
    }
    while(i <= k){
      fac.pb(s.substr(i, j - k));
      i += j - k;
    }
  }
  return fac;
}
```

6 Math

6.1 Miller Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 fpow(u64 a, u64 b, u64 n){
  u64 ret = 1;
  while(b > 0){
    if(b & 1) ret = (u128)ret * a % n;
    a = (u128)a * a % n;
    b >>= 1;
  }
  return ret;
}
bool check_composite(u64 n, u64 a, u64 d, int s){
  u64 x = fpow(a, d, n);
  if(x == 1 || x == n - 1) return false;
  for(int r = 1; r < s; r++){
    x = (u128)x * x % n;
    if(x == n - 1) return false;
  }
  return true;
}
bool MillerRabin(u64 n){
  if(n < 2) return false;
  int s = 0;
  u64 d = n - 1;
  while(!(d & 1)){
    d >>= 1;
    s++;
  }
  for(auto a : {2, 3, 5, 7, 11, 13, 17,
        19, 23, 29, 31, 37}){ // sufficient for n < 2^64
    if(n == a) return true;
    if(check_composite(n, a, d, s)) return false;
  }
  return true;
}
```

6.2 Pollard Rho

```
ll f(ll t, ll c, ll n){
  return (t * t + c) % n;
}

ll Pollard_Rho(ll x){
  ll t = 0;
  ll c = rand() % (x - 1) + 1;
  ll s = t;
  ll val = 1;
  for(int goal = 1;; goal <<= 1, s = t, val = 1){
    for(int step = 1; step <= goal; step++){
      t = f(t, c, x);
      val = val * abs(t - s) % x;
      if(!val) return x;
      if(step % 127 == 0){
```

```
        ll d = __gcd(val, x);
        if(d > 1) return d;
      }
    }
    ll d = __gcd(val, x);
    if(d > 1) return d;
  }
}
```

6.3 EXT GCD

```
ll extgcd(ll a, ll b, ll &x, ll &y){
  if(b == 0){
    x = 1, y = 0;
    return a;
  }
  int res = extgcd(b, a % b, y, x);
  y -= (a / b) * x;
  return res;
}
```

6.4 Chinese Remainder Theorem

```
ll CRT(vector<ll> p, vector<ll> a){
  ll n = p.size(), prod = 1, ret = 0;
  for(int i = 0; i < n; i++) prod *= p[i];
  for(int i = 0; i < n; i++){
    ll m = (prod / p[i]);
    ll x, y;
    extgcd(m, p[i], x, y);
    ret = ((ret + a[i] * m * x) % prod + prod) % prod;
  }
  return ret;
}
```

6.5 Powerful Number Sieve

```
void linearsieve(){
  phi[1] = 1;
  for(int i = 2; i < maxn; i++){
    if(!lp[i]) pr.pb(i), lp[i] = i, phi[i] = i - 1;
    for(auto x : pr){
      if(i * x >= maxn) break;
      lp[i * x] = x;
      if(lp[i] == x){
        phi[i * x] = phi[i] * x;
        break;
      }
      phi[i * x] = phi[i] * (x - 1);
    }
  }
  for(int i = 1; i < maxn
      ; i++) sum[i] = (sum[i - 1] + i * phi[i]) % N;
}

int s2(int n){
  static const int inv6 = inv(6);
  n %= N;
  return n * (n + 1) % N * (2 * n + 1) % N * inv6 % N;
}

int G(int n){
  static const int inv2 = inv(2);
  if(n < maxn) return sum[n];
  if(mp_G.count(n)) return mp_G[n];
  int ans = s2(n);
  for(int i = 2, j; i <= n; i = j + 1){
    j = n / (n / i);
    (ans -= (i + j) % N * (j -
        i + 1) % N * inv2 % N * G(n / i) % N - N) %= N;
  }
  return mp_G[n] = ans;
}

void dfs(int d, int hd, int p){ // dfs 出所有 PN
  (ans += hd * G(n / d)) %= N;
  for(int i = p; i < pr.size(); i++){
    if(d > n / pr[i] / pr[i]) break;
    int c = 2;
    for(int x
         = d * pr[i] * pr[i]; x <= n; x *= pr[i], c++){
      if(!vis[i][c]){
        int f = fpow(pr[i], c);
        f = f * (f - 1) % N;
        int g = pr[i] * (pr[i] - 1) % N;
        int t = pr[i] * pr[i] % N;
        for(int j = 1; j <= c; j++){
          (f -= g * h[i][c - j] % N - N) %= N;
          (g *= t) %= N;
        }
        h[i][c] = f;
        vis[i][c] = true;
      }
      if(h[i][c]) dfs(x, hd * h[i][c] % N, i + 1);
    }
  }
}

linearsieve();
for(int i = 0; i < pr.size(); i++) h[i][0] = 1;
dfs(1, 1, 0);
```

6.6 Fast Walsh Transform

```
void fwt(vector<int> &a, bool inv){
  int n = 1;
  while(n < a.size()) n *= 2;
  a.resize(n);
  for(int len = 1; 2 * len <= n; len <<= 1){
    for(int i = 0; i < n; i += 2 * len){
      for(int j = 0; j < len; j++){
        int &u =
             a[i + j], &v = a[i + j + len]; tie(u, v) =
        // inv ? pll(u - v, v) : pll(u + v, v); // and
        // inv ? pll(u, v - u) : pll(u, u + v); // or
        pll(u + v, u - v); // xor
      }
    }
  }
  if(inv) for(auto &x : a) x /= n; // xor only
}
```

6.7 Floor Sum

```
//f(n, a, b, c) = sum_{0<=i<=n}{(ai + b)/c},
//g(n, a, b, c) = sum_{0<=i<=n}{i(ai + b)/c},
//h(n, a, b, c) = sum_{0<=i<=n}{((ai + b)/c)^2},
const int N = 998244353;
const int i2 = (N + 1) / 2, i6 = 166374059;
struct info{
  ll f, g, h;
  info(){f = g = h = 0;}
};
info calc(ll n, ll a, ll b, ll c){
  ll ac = a / c, bc = b / c,
     m = (a * n + b) / c, n1 = n + 1, n21 = n * 2 + 1;
  info d;
  if(a == 0){
    d.f = bc * n1 % N;
    d.g = bc * n % N * n1 % N * i2 % N;
    d.h = bc * bc % N * n1 % N;
    return d;
  }
  if(a >= c || b >= c){
    d.f = n * n1 % N * i2 % N * ac % N + bc * n1 % N;
    d.g = ac * n % N * n1 % N * n21
         % N * i6 % N + bc * n % N * n1 % N * i2 % N;
    d.h = ac * ac
        % N * n % N * n1 % N * n21 % N * i6 % N + bc *
        bc % N * n1 % N + ac * bc % N * n % N * n1 % N;

    info e = calc(n, a % c, b % c, c);

    d.h +=
         e.h + 2 * bc * e.f % N + 2 * ac % N * e.g % N;
    d.g += e.g, d.f += e.f;
    d.f %= N, d.g %= N, d.h %= N;
    return d;
  }
  info e = calc(m - 1, c, c - b - 1, a);
  d.f = (n * m % N - e.f + N) % N;
  d.g = m * n % N *
      n1 % N - e.h - e.f; d.g = (d.g * i2 % N + N) % N;
  d.h = n * m % N * (m + 1) % N -
      2 * e.g - 2 * e.f - d.f; d.h = (d.h % N + N) % N;
  return d;
}
```

6.8 NTT

```
const int N = 998244353, g = 3;
using cd = complex<double>;
const double PI = acos(-1);
```

```
void NTT(vector
    <ll> &a, bool invert = 0){ // interative version
  int n = a.size();
  int lg_n = __lg(n);
  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){
    ll wn = fpow(g, (N - 1) / len);
    if(invert) wn = inv(wn);
    for(int i = 0; i < n; i += len){
      ll w = 1;
      for(int j = 0; j < len / 2; j++){
        ll u
            = a[i + j], v = a[i + j + len / 2] * w % N;
        a[i + j] = (u + v) % N;
        a[i + j + len / 2] = (u - v + N) % N;
        (w *= wn) %= N;
      }
    }
  }
  ll n_1 = inv(n);
  if(invert) for(auto &x : a) (x *= n_1) %= N;
}
```

7 Geometry

7.1 Basic

```
struct pt{
    double x, y;
    pt(){}
    pt(double _x, double _y) : x(_x), y(_y){}
};
pt operator + (pt a, pt b)
{ return pt(a.x + b.x, a.y + b.y); }
pt operator - (pt a, pt b)
{ return pt(a.x - b.x, a.y - b.y); }
pt operator * (pt a, double p)
{ return pt(a.x * p, a.y * p); }
pt operator / (pt a, double p)
{ return pt(a.x / p, a.y / p); }
bool operator < (const pt &a, const pt &b)
{ return a.x < b.x || (a.x == b.x && a.y < b.y); }
bool operator == (const pt &a, const pt &b)
{ return a.x == b.x && a.y == b.y; }
double dot(pt a, pt b)
{ return a.x * b.x + a.y * b.y; }
double cross(pt a, pt b)
{ return a.x * b.y - a.y * b.x; }
double len(pt a)
{ return sqrt(dot(a, a)); }
double angle(pt a, pt b)
{ return acos(dot(a, b) / len(a) / len(b)); }
double area2(pt a, pt b, pt c)
{ return cross(b - a, c - a); }

const double eps = 1e-9;
int dcmp(double x){
  if(fabs(x) < eps) return 0;
  return x < 0? -1 : 1;
}

inline int ori(pt a, pt b, pt c){
  double area = cross(b - a, c - a);
  if(area > -eps && area < eps) return 0;
  return area > 0 ? 1 : -1;
}

inline int btw(pt a, pt b, pt c){ // [a, c, b]
  if(fabs(cross(b - a, c - a)) > eps) return false;
  if(dot(b - a, c - a)
       > -eps && len(c - a) <= len(b - a)) return true;
  return false;
}

bool intersect(pt a, pt b, pt c, pt d){
  if(a == c || a == d || b == c || b == d) return true;
  int a123 = ori(a, b, c), a124 = ori(a,
       b, d), a341 = ori(c, d, a), a342 = ori(c, d, b);
  if(a123 == 0 && a124 == 0){
    if(btw(a, b, c) || btw(a, b, d
         ) || btw(c, d, a) || btw(c, d, b)) return true;
    else return false;
  }
  else if(a123
       * a124 <= 0 && a341 * a342 <= 0) return true;
  return false;
}

istream &operator>>(istream &s, pt &a){
  s >> a.x >> a.y;
  return s;
}
```

7.2 Minkowski Sum

```
void reorder(vector<pt> &a){
    int pos = 0;
    for(int j = 1; j < a.size(); j++){
        if(a[j].x < a[pos].x || (a[j].x
            == a[pos].x && a[j].y < a[pos].y)) pos = j;
    }
    rotate(a.begin(), a.begin() + pos, a.end());
}

vector<pt> minkowski(vector<pt> a, vector<pt> b){
    // for(int i = 0;
         i < b.size(); i++) b[i] = {-b[i].x, -b[i].y};
        最短距離：把 Q 鏡像，找凸包到 (0, 0) 的最短距離
    reorder(a), reorder(b);
    a.pb(a[0]), a.pb(a[1]);
    b.pb(b[0]), b.pb(b[1]);
    vector<pt> res;
    int i = 0, j = 0;
    while(i < a.size() - 2 || j < b.size() - 2){
        res.pb(a[i] + b[j]);
        int c
             = cross(a[i + 1] - a[i], b[j + 1] - b[j]);
        if(c >= 0 && i < a.size() - 2) i++;
        if(c <= 0 && j < b.size() - 2) j++;
    }
    return res;
}
```

8 Misc

8.1 SMAWK

```
bool select(int r, int u, int v){
    // if f(r, v) is better than f(r, u), return true
  return f(r, u) < f(r, v);
}
// For all 2x2 submatrix: (x < y => y is better than x)
// If M[1][0] < M[1][1], M[0][0] < M[0][1]
// If M[1][0] == M[1][1], M[0][0] <= M[0][1]
// M[i][ans[i]] is the best value in the i-th row
vector<int> solve(vector<int> &r, vector<int> &c){
  if(r.size() == 1){
    vector<int> opt(1, 0);
    for(int i = 1; i < c.size(); i++){
      if(select(r[0], c[opt[0]], c[i])){
        opt[0] = i;
      }
    }
    return opt;
  }
  //reduce
  vector<int> st, rev;
  for(int i = 0; i < c.size(); i++){
    while(!st.empty()
         && select(r[st.size() - 1], st.back(), c[i])){
      st.pop_back();
      rev.pop_back();
    }
    if(st.size() < r.size()){
      st.pb(c[i]);
      rev.pb(i);
    }
  }
  //interpolate
  vector<int> half;
  for(int i = 0; i < r.size(); i += 2){
    half.pb(r[i]);
  }
  vector<int> ans(r.size());
  auto interp = solve(half, st);
  for(int i = 0;
       i < interp.size(); i++) ans[i * 2] = interp[i];
  for(int i = 1; i < ans.size(); i += 2){
    int s = ans[i - 1], e = (i
         + 1 < ans.size() ? ans[i + 1] : st.size() - 1);
```

```
    ans[i] = s;
    for(int j = s + 1; j <= e; j++){
      if(select(r[i], st[ans[i]], st[j])) ans[i] = j;
    }
  }
  for(int
      i = 0; i < ans.size(); i++) ans[i] = rev[ans[i]];
  return ans;
}

vector<int> smawk(int n, int m){
  vector<int> r(n), c(m);
  iota(r.begin(), r.end(), 0);
  iota(c.begin(), c.end(), 0);
  return solve(r, c);
}
```

8.2 Simulate Annealing

```
double anneal() {
  mt19937 rnd_engine(time(0));
  uniform_real_distribution<double> rng(0, 1);
  const double dT = 0.001;
  // Argument p
  double S_cur = calc(p), S_best = S_cur;
  for (double T = 2000; T > eps; T -= dT) {
    // Modify p to p_prime
    const double S_prime = calc(p_prime);
    const double delta_c = S_prime - S_cur;
    double prob = min((double)1, exp(-delta_c / T));
    if (rng(rnd_engine) <= prob)
      S_cur = S_prime, p = p_prime;
    if (S_prime < S_best) // find min
      S_best = S_prime, p_best = p_prime;
  }
  return S_best;
}
```