Team Note of AC-complete

0xC0DEF, enochjung, jhnah917

Compiled on November 12, 2021

Contents

1 DataStructure

1.1 Bipartite Union Find

Usage: Union-Find with friend, enemy relations

```
int P[_Sz], E[_Sz]; // Parent, Enemy
void clear(){ iota(P, P+_Sz, 0); memset(E, -1, sizeof E); }
int find(int v){}
bool merge(int u, int v){}
int set_friend(int u, int v){ return merge(u, v); }
int set_enemy(int u, int v){
  int ret = 0;
  if(E[u] == -1) E[u] = v;
  else ret += merge(E[u], v);
  if(E[v] == -1) E[v] = u;
  else ret += merge(u, E[v]);
  return ret;
}
```

1.2 Erasable Priority Queue

```
template<typename T, T inf>
struct pq_set{
  priority_queue<T, vector<T>, greater<T>> in, out; // min heap, inf = 1e18
  // priority_queue<T> in, out; // max heap, inf = -1e18
  pq_set(){ in.push(inf); }
  void insert(T v){ in.push(v); }
  void erase(T v){ out.push(v); }
  T top(){
  while(out.size() && in.top() == out.top()) in.pop(), out.pop();
  return in.top();
  }
  bool empty(){
  while(out.size() && in.top() == out.top()) in.pop(), out.pop();
  return in.top() == inf;
  }
};
```

1.3 Convex Hull Trick

Usage: call init() before use

```
struct Line{
  ll a, b, c; // y = ax + b, c = line index
  Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
  ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
  return (__int128_t)(a.b - b.b) * (b.a - c.a) <= (__int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
  if(v.size() > pv && v.back().a == l.a){
    if(l.b < v.back().b) l = v.back(); v.pop_back();
  }
  while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l)) v.pop_back();
  v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
  while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
  return {v[pv].f(x), v[pv].c};
}
///// line container start (max query) /////
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<>> {
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); } // floor
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) { assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

1.4 Persistent Segment Tree

Usage: call init(root[0], s, e) before use

```
struct PSTNode{
  PSTNode *l, *r; int v;
  PSTNode(){ l = r = nullptr; v = 0; }
};
PSTNode *root[101010];
PST(){ memset(root, 0, sizeof root); } // constructor
void init(PSTNode *node, int s, int e){
  if(s == e) return;
  int m = s + e >> 1;
  node->l = new PSTNode; node->r = new PSTNode;
  init(node->l, s, m); init(node->r, m+1, e);
}
void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
  if(s == e){ now->v = prv ? prv->v + 1 : 1; return; }
  int m = s + e >> 1;
  if(x <= m){
    now->l = new PSTNode; now->r = prv->r;
    update(prv->l, now->l, s, m, x);
```

```
  }
  else{
    now->r = new PSTNode; now->l = prv->l;
    update(prv->r, now->r, m+1, e, x);
  }
  int t1 = now->l ? now->l->v : 0;
  int t2 = now->r ? now->r->v : 0;
  now->v = t1 + t2;
}
int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
  if(s == e) return s;
  int m = s + e >> 1, diff = now->l->v - prv->l->v;
  if(k <= diff) return kth(prv->l, now->l, s, m, k);
  else return kth(prv->r, now->r, m+1, e, k-diff);
}
```

1.5 Splay Tree, Link-Cut Tree

```
struct Node{
  Node *l, *r, *p;
  bool flip; int sz;
  T now, sum, lz;
  Node(){ l = r = p = nullptr; sz = 1; flip = false; now = sum = lz = 0; }
  bool IsLeft() const { return p && this == p->l; }
  bool IsRoot() const { return !p || (this != p->l && this != p->r); }
  friend int GetSize(const Node *x){ return x ? x->sz : 0; }
  friend T GetSum(const Node *x){ return x ? x->sum : 0; }
  void Rotate(){
    p->Push(); Push();
    if(IsLeft()) r && (r->p = p), p->l = r, r = p;
    else l && (l->p = p), p->r = l, l = p;
    if(!p->IsRoot()) (p->IsLeft() ? p->p->l : p->p->r) = this;
    auto t = p; p = t->p; t->p = this;
    t->Update(); Update();
  }
  void Update(){
    sz = 1 + GetSize(l) + GetSize(r);
    sum = now + GetSum(l) + GetSum(r);
  }
  void Update(const T &val){ now = val; Update(); }
  void Push(){
    Update(now + lz); if(flip) swap(l, r);
    for(auto c : {l, r}) if(c) c->flip ^= flip, c->lz += lz;
    lz = 0; flip = false;
  }
};
Node* rt;
Node* Splay(Node *x, Node *g=nullptr){
  for(g || (rt=x); x->p!=g; x->Rotate()){
    if(!x->p->IsRoot()) x->p->p->Push(); x->p->Push(); x->Push();
    if(x->p->p != g) (x->IsLeft() ^ x->p->IsLeft() ? x : x->p)->Rotate();
  }
  x->Push(); return x;
}
Node* Kth(int k){
  for(auto x=rt; ; x=x->r){
    for(; x->Push(), x->l && x->l->sz > k; x=x->l);
    if(x->l) k -= x->l->sz;
    if(!k--) return Splay(x);
  }
}
Node* Gather(int s, int e){
  auto t = Kth(e+1); return Splay(t, Kth(s-1))->l;
}
Node* Flip(int s, int e){
  auto x = Gather(s, e); x->flip ^= 1; return x;
}
Node* Shift(int s, int e, int k){
  if(k >= 0){
    k %= e-s+1;
    if(k) Flip(s, e), Flip(s, s+k-1), Flip(s+k, e);
  }
  else{
    k = -k; k %= e-s+1;
    if(k) Flip(s, e), Flip(s, e-k), Flip(e-k+1, e);
  }
  return Gather(s, e);
}
int Idx(Node *x){ return x->l->sz; }
/////////// Link Cut Tree Start ///////////
Node* Splay(Node *x){
  for(; !x->IsRoot(); x->Rotate()){
    if(!x->p->IsRoot()) x->p->p->Push(); x->p->Push(); x->Push();
    if(!x->p->IsRoot()) (x->IsLeft() ^ x->p->IsLeft() ? x : x->p)->Rotate();
  }
  x->Push(); return x;
}
void Access(Node *x){
  Splay(x); x->r = nullptr; x->Update();
  for(auto y=x; x->p; Splay(x)){
    y = x->p; Splay(y); y->r = x; y->Update();
  }
}
int GetDepth(Node *x){
  Access(x); x->Push();
  return GetSize(x->l);
}
Node* GetRoot(Node *x){
  Access(x); for(x->Push(); x->l; x->Push()) x = x->l;
  return Splay(x);
}
Node* GetPar(Node *x){
  Access(x); x->Push(); if(!x->l) return nullptr;
  x = x->l; for(x->Push(); x->r; x->Push()) x = x->r;
  return Splay(x);
}
void Link(Node *p, Node *c){
  Access(c); Access(p);
  c->l = p; p->p = c; c->Update();
}
void Cut(Node *c){
  Access(c);
```

```
  c->l->p = nullptr; c->l = nullptr; c->Update();
}
Node* GetLCA(Node *x, Node *y){
  Access(x); Access(y); Splay(x);
  return x->p ? x->p : x;
}
Node* Ancestor(Node *x, int k){
  k = GetDepth(x) - k; assert(k >= 0);
  for(;;x->Push()){
    int s = GetSize(x->l);
    if(s == k) return Access(x), x;
    if(s < k) k -= s + 1, x = x->r;
    else x = x->l;
  }
}
void MakeRoot(Node *x){ Access(x); Splay(x); x->flip ^= 1; }
bool IsConnect(Node *x, Node *y){ return GetRoot(x) == GetRoot(y); }
void PathUpdate(Node *x, Node *y, T val){
  Node *root = GetRoot(x); // original root
  MakeRoot(x); Access(y);  // make x to root, tie with y
  Splay(x); x->lz += val; x->Push();
  MakeRoot(root);    // Revert
  Node *lca = GetLCA(x, y);
  Access(lca); Splay(lca); lca->Push();
  lca->Update(lca->now - val);
}
T VertexQuery(Node *x, Node *y){
  Node *l = GetLCA(x, y);
  T ret = l->now;
  Access(x); Splay(l);
  if(l->r) ret = ret + l->r->sum;
  Access(y); Splay(l);
  if(l->r) ret = ret + l->r->sum;
  return ret;
}
Node* GetQueryResultNode(Node *u, Node *v){
  if(GetRoot(u) != GetRoot(v)) return 0;
  MakeRoot(u); Access(v);
  auto ret = v->l;
  while(ret->mx != ret->v){
    if (ret->l && ret->mx == ret->l->mx) ret = ret->l;
    else ret = ret->r;
  }
  Access(ret);
  return ret;
}
```

2 Geometry

2.1 Rotating Calipers

```
pair<Point, Point> RotatingCalipers(const vector<Point> &H){
  ll mx = 0; Point a, b;
  for(int i=0, j=0; i<H.size(); i++){
    while(j+1 < H.size() && CCW(O, H[i+1]-H[i], H[j+1]-H[j]) >= 0){
      if(ll now = D2(H[i], H[j]); mx < now) mx = now, a = H[i], b = H[j];
      j++;
    }
    if(ll now = D2(H[i], H[j]); mx < now) mx = now, a = H[i], b = H[j];
  }
  return {a, b};
}
```

2.2 Point in Convex Polygon

```
bool pip_convex(const vector<p> &v, p pt){
  int i = lower_bound(v.begin()+1, v.end(), pt, [&](const p &a, const p &b){
    int cw = ccw(v[0], a, b);
    if(cw) return cw > 0;
    return dst(v[0], a) < dst(v[0], b);
  }) - v.begin();
  if(i == v.size()) return 0;
  if(i == 1) return ccw(v[0], pt, v[1]) == 0 && v[0] <= pt && pt <= v[1];
  int t1 = ccw(v[0], pt, v[i]) * ccw(v[0], pt, v[i-1]);
  int t2 = ccw(v[i], v[i-1], v[0]) * ccw(v[i], v[i-1], pt);
  if(t1 == -1 && t2 == -1) return 0;
  return ccw(v[0], pt, v[i-1]) != 0;
}
```

2.3 Half Plane Intersection

Usage: Line : $ax + by + c = 0$

```
const pdd o = pdd(0, 0);
ld ccw(pdd a, pdd b, pdd c){} // return cross product value
struct Line{
  ld a, b, c;
  Line() : Line(0, 0, 0) {}
  Line(ld a, ld b, ld c) : a(a), b(b), c(c) {}
  bool operator < (const Line &l) const {
    bool f1 = pdd(a, b) > o;
    bool f2 = pdd(l.a, l.b) > o;
    if(f1 != f2) return f1 > f2;
    ld cw = ccw(o, pdd(a, b), pdd(l.a, l.b));
    return same(cw, 0) ? c * hypot(l.a, l.b) < l.c * hypot(a, b) : cw > 0;
  }
  pdd slope() const { return pdd(a, b); }
};
pdd lineCross(Line a, Line b){
  ld det = a.a*b.b - b.a*a.b;
  ld x = (a.c*b.b - a.b*b.c) / det, y = (a.a*b.c - a.c*b.a) / det;
  return pdd(x, y);
}
bool hpi_chk(Line a, Line b, Line c){
  if(ccw(o, a.slope(), b.slope()) <= 0) return 0;
  pdd v = lineCross(a, b);
  return v.x*c.a + v.y*c.b >= c.c;
}
vector<pdd> hpi(vector<Line> v){
  sort(v.begin(), v.end());
  deque<Line> dq; vector<pdd> ret;
  for(auto &i : v){
```

```
    if(dq.size() && same(ccw(o, dq.back().slope(), i.slope()), 0)) continue;
    while(dq.size() >= 2 && hpi_chk(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
    while(dq.size() >= 2 && hpi_chk(i, dq[0], dq[1])) dq.pop_front();
    dq.push_back(i);
  }
  while(dq.size() > 2 && hpi_chk(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
  while(dq.size() > 2 && hpi_chk(dq.back(), dq[0], dq[1])) dq.pop_front();
  for(int i=0; i<dq.size(); i++){
    Line now = dq[i], nxt = dq[(i+1)%dq.size()];
    if(ccw(o, now.slope(), nxt.slope()) <= eps) return vector<pdd>();
    ret.push_back(lineCross(now, nxt));
  }
  return ret;
}
```

2.4 K-D Tree

```
#define all_range(v, s, e) v.begin()+s, v.begin()+e+1
struct KDNode{
  pll v; bool dir; ll sx, ex, sy, ey;
  KDNode(){ sx = sy = inf; ex = ey = -inf; }
};
const auto xcmp = [](pll a, pll b){ return tie(a.x, a.y) < tie(b.x, b.y); };
const auto ycmp = [](pll a, pll b){ return tie(a.y, a.x) < tie(b.y, b.x); };
struct KDTree{
  // Segment Tree Size
  static const int S = 1 << 18;
  KDNode nd[S]; int chk[S];
  vector<pll> v;
  KDTree(){ init(); }
  void init(){ memset(chk, 0, sizeof chk); }
  void _build(int node, int s, int e){
    chk[node] = 1; nd[node].dir = !nd[node/2].dir;
    nd[node].sx = min_element(all_range(v, s, e), xcmp)->x;
    nd[node].ex = max_element(all_range(v, s, e), xcmp)->x;
    nd[node].sy = min_element(all_range(v, s, e), ycmp)->y;
    nd[node].ey = max_element(all_range(v, s, e), ycmp)->y;
    if(nd[node].dir) sort(all_range(v, s, e), ycmp);
    else sort(all_range(v, s, e), xcmp);
    int m = s + e >> 1; nd[node].v = v[m];
    if(s <= m-1) _build(node<<1, s, m-1);
    if(m+1 <= e) _build(node<<1|1, m+1, e);
  }
  void build(const vector<pll> &_v){
    v = _v; sort(all(v)); _build(1, 0, v.size()-1);
  }
  ll query(pll t, int node = 1){
    ll tmp, ret = inf;
    if(t != nd[node].v) ret = min(ret, dst(t, nd[node].v));
    bool x_chk = (!nd[node].dir && xcmp(t, nd[node].v));
    bool y_chk = (nd[node].dir && ycmp(t, nd[node].v));
    if(x_chk || y_chk){
      if(chk[node<<1]) ret = min(ret, query(t, node<<1));
      if(chk[node<<1|1]){
        if(nd[node].dir) tmp = nd[node<<1|1].sy - t.y;
        else tmp = nd[node<<1|1].sx - t.x;
        if(tmp*tmp < ret) ret = min(ret, query(t, node<<1|1));
      }
    }
    else{
      if(chk[node<<1|1]) ret = min(ret, query(t, node<<1|1));
      if(chk[node<<1]){
        if(nd[node].dir) tmp = nd[node<<1].ey - t.y;
        else tmp = nd[node<<1].ex - t.x;
        if(tmp*tmp < ret) ret = min(ret, query(t, node<<1));
      }
    }
    return ret;
  }
};
```

2.5 Dual Graph

```
const int MV = 101010, ME = 101010; // MAX_V, MAX_E
p pt[MV]; // vertex's coord
vector<p> g[MV]; // g[s].emplace_back(e, edge_id);
vector<int> dual_pt; // coord compress
int par[ME * 2]; // Union Find
void uf_init(){ iota(par, par+ME*2, 0); }
int find(int v){return v == par[v] ? v : par[v] = find(par[v]);}
void merge(int u, int v){ u = find(u); v = find(v); if(u != v)par[u] = v; }
p base; // sort by angle
bool cmp_angle(const p &_a, const p &_b){
  p a = pt[_a.x], b = pt[_b.x];
  if((a > base) != (b > base)) return a > b;
  return ccw(a, base, b) > 0;
}
void addEdge(int s, int e, int id){
  g[s].emplace_back(e, id); g[e].emplace_back(s, id);
}
int out; //outer face
void getDual(int n, int m){
  uf_init();
  for(int i=1; i<=n; i++){
    base = pt[i]; sort(all(g[i]), cmp_angle);
    // up, left : *2+1 / down, right : *2
    for(int j=0; j<g[i].size(); j++){
      int k = j ? j - 1 : g[i].size()-1;
      int u = g[i][k].y << 1 | 1, v = g[i][j].y << 1;
      p p1 = pt[g[i][k].x], p2 = pt[g[i][j].x];
      if(p1 > base) u ^= 1;
      if(p2 > base) v ^= 1;
      merge(u, v);
    }
  }
  int mn_idx = min_element(pt+1, pt+n+1) - pt;
  out = find(g[mn_idx][0].y << 1 | 1);
  for(int i=1; i<=m; i++){
    dual_pt.push_back(find(i << 1));
    dual_pt.push_back(find(i << 1 | 1));
  }
  compress(dual_pt);
```

```
  // @TODO coord compress
}
```

2.6 Bulldozer Trick (Rotating Sweep Line)

```
Point v[2020];
struct Line{
  ll i, j, dx, dy;
  Line(int i, int j) : i(i), j(j) {
    dx = v[i].x - v[j].x; dy = v[i].y - v[j].y;
  }
  bool operator < (const Line &t) const {
    ll a = dy * t.dx, b = t.dy * dx;
    return tie(a, i, j) < tie(b, t.i, t.j);
  }
};
int ccw(Point a, Point b, Point c){}
int ccw(Line a, Line b){
  ll res = a.dx*b.dy - b.dx*a.dy;
  if(!res) return 0; return res > 0 ? 1 : -1;
}
int idx[2020]; vector<Line> line;
void bulldozer(int n){
  sort(v+1, v+n+1); for(int i=1; i<=n; i++) idx[i] = i;
  for(int i=1; i<=n; i++) for(int j=1; j<i; j++) line.emplace_back(i, j);
  for(int i=0, j=0; i<line.size(); ){
    int ed = i;
    while(ed < line.size() && !ccw(line[i], line[ed])) ed++;
    for(int j=i; j<ed; j++){
      int a = line[j].i, b = line[j].j;
      swap(idx[a], idx[b]); swap(v[idx[a]], v[idx[b]]);
      update(idx[a]); update(idx[b]);
    }
    ans = merge(ans, query()); i = ed;
  }
}
```

2.7 Delaunay Triangulation

```
using lll = __int128; // using T = ll; (if coords are < 2e4)
// return true if p strictly within circumcircle(a,b,c)
bool inCircle(P p, P a, P b, P c) {
  a -= p, b -= p, c -= p; // assert(cross(a,b,c)>0);
  lll x = (lll)norm(a)*cross(b,c)+(lll)norm(b)*cross(c,a)
      +(lll)norm(c)*cross(a,b);
  return x*(cross(a,b,c)>0?1:-1) > 0;
}
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point
using Q = struct Quad*;
struct Quad {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
};
Q makeEdge(P orig, P dest) {
  Q q[]{new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
      new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
  FOR(i,4) q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
  return *q;
}
void splice(Q a, Q b) { swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o); }
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next()); splice(q->r(), b);
  return q;
}
pair<Q,Q> rec(const vP& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.bk);
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = cross(s[0], s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

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

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (inCircle(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e = t; \
    }
  while (1) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && inCircle(H(RC), H(LC))))
      base = connect(RC, base->r());
    else base = connect(base->r(), LC->r());
  }
  return {ra, rb};
}
V<AR<P,3>> triangulate(vP pts) {
  sor(pts); assert(unique(all(pts)) == end(pts)); // no duplicates
  if (sz(pts) < 2) return {};
  Q e = rec(pts).f; V<Q> q = {e};
  while (cross(e->o->F(), e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.pb(c->p); \
```

```
  q.pb(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  int qi = 0; while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  V<AR<P,3>> ret(sz(pts)/3);
  FOR(i,sz(pts)) ret[i/3][i%3] = pts[i];
  return ret;
}
```

3 Graph

3.1 Euler Tour

```
// Not Directed / Cycle
constexpr int SZ = 1010;
int N, G[SZ][SZ], Deg[SZ], Work[SZ];
void DFS(int v){
  for(int &i=Work[v]; i<=N; i++) while(G[v][i])
    G[v][i]--, G[i][v]--, DFS(i);
  cout << v << " ";
}
// Directed / Path
void DFS(int v){
  for(int i=1; i<=pv; i++) while(G[v][i]) G[v][i]--, DFS(i);
  Path.push_back(v);
}
void Get(){
  for(int i=1; i<=pv; i++) if(In[i] < Out[i]){ DFS(i); return; }
  for(int i=1; i<=pv; i++) if(Out[i]){ DFS(i); return; }
}
```

3.2 SCC + 2-SAT

Usage: CNF: (A or B) / alwaysTrue: A =¿ B / setValue / mostOne / exactlyOne

```
inline int True(int x){ return x << 1; }
inline int False(int x){ return x << 1 | 1; }
inline int Inv(int x){ return x ^ 1; }
struct TwoSat{
  int n;
  vector<vector<int>> g;
  vector<int> result;
  TwoSat(int n, int m = 0) : n(n), g(n+n) { if(!m) g.reserve(m+m); }
  int addVar(){ g.emplace_back(); g.emplace_back(); return n++; }
  void addEdge(int s, int e){ g[s].push_back(e); }
  void addCNF(int a, int b){ addEdge(Inv(a), b); addEdge(Inv(b), a); } // (A or B)
  void setValue(int x){ addCNF(x, x); } // (A or A)
  void addAlwaysTrue(int a, int b){ addEdge(a, b); addEdge(Inv(b), Inv(a)); } // A => B
  void addMostOne(const vector<int> &li){
    if(li.empty()) return; int t;
    for(int i=0; i<li.size(); i++){
      t = addVar();
      addAlwaysTrue(li[i], True(t));
      if(!i) continue;
      addAlwaysTrue(True(t-1), True(t));
      addAlwaysTrue(True(t-1), Inv(li[i]));
    }
  }
  void addExactlyOne(const vector<int> &li){
    if(li.empty()) return; int t;
    for(int i=0; i<li.size(); i++){
      t = addVar();
      addAlwaysTrue(li[i], True(t));
      if(!i) continue;
      addAlwaysTrue(True(t-1), True(t));
      addAlwaysTrue(True(t-1), Inv(li[i]));
    }
    setValue(True(t));
  }
  vector<int> val, comp, z; int pv = 0;
  int dfs(int i){
    int low = val[i] = ++pv, x; z.push_back(i);
    for(int e : g[i]) if(!comp[e]) low = min(low, val[e] ?: dfs(e));
    if(low == val[i]){
      do{
        x = z.back(); z.pop_back();
        comp[x] = low;
        if (result[x>>1] == -1) result[x>>1] = ~x&1;
      }while(x != i);
    }
    return val[i] = low;
  }
  bool sat(){
    result.assign(n, -1);
    val.assign(2*n, 0); comp = val;
    for(int i=0; i<n+n; i++) if(!comp[i]) dfs(i);
    for(int i=0; i<n; i++) if(comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
  vector<int> getValue(){ return move(result); }
};
```

3.3 BCC

Usage: call tarjan() before use

```
vector<int> G[MAX_V];
int In[MAX_V], Low[MAX_V], P[MAX_V];
void addEdge(int s, int e){
  G[s].push_back(e); G[e].push_back(s);
}
void tarjan(int n){ /// Pre-Process
  int pv = 0;
  function<void(int,int)> dfs = [&pv,&dfs](int v, int b){
    In[v] = Low[v] = ++pv; P[v] = b;
    for(auto i : G[v]){
      if(i == b) continue;
      if(!In[i]) dfs(i, v), Low[v] = min(Low[v], Low[i]);
      else Low[v] = min(Low[v], In[i]);
    }
  };
  for(int i=1; i<=n; i++) if(!In[i]) dfs(i, -1);
}
```

```
vector<int> cutVertex(int n){
  vector<int> res;
  array<char,MAX_V> isCut;
  function<void(int)> dfs = [&dfs,&isCut](int v){
    int ch = 0;
    for(auto i : G[v]){
      if(P[i] != v) continue;
      dfs(i); ch++;
      if(P[v] == -1 && ch > 1) isCut[v] = 1;
      else if(P[v] != -1 && Low[i] >= In[v]) isCut[v] = 1;
    }
  };
  for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
  for(int i=1; i<=n; i++) if(isCut[i]) res.push_back(i);
  return move(res);
}
vector<PII> cutEdge(int n){
  vector<PII> res;
  function<void(int)> dfs = [&dfs,&res](int v){
    for(auto i : G[v]){
      if(P[i] != v) continue;
      dfs(i);
      if(Low[i] > In[v]) res.emplace_back(min(v,i), max(v,i));
    }
  };
  for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
  return move(res); // sort(all(res));
}
vector<int> BCC[MAX_V]; // BCC[v] = components which contains v
void vertexDisjointBCC(int n){
  int cnt = 0;
  array<char,MAX_V> vis;
  function<void(int,int)> dfs = [&dfs,&vis,&cnt](int v, int c){
    if(c > 0) BCC[v].push_back(c);
    vis[v] = 1;
    for(auto i : G[v]){
      if(vis[i]) continue;
      if(In[v] < In[i]) BCC[v].push_back(++cnt), dfs(i, cnt);
      else dfs(i, c);
    }
  };
  for(int i=1; i<=n; i++) if(!vis[i]) dfs(i, ++cnt);
}
void edgeDisjointBCC(int n){} // remove cut edge, do flood fill
```

3.4 Maximum Clique

```
int N, M; ull G[40], MX, Clique; // 0-index, adj list with bitset, O(3^{N/3})
void get_clique(int R = 0, ull P = (1ULL<<N)-1, ull X = 0, ull V=0){
  if((P|X) == 0){ if(R > MX) MX = R, Clique = V; return; }
  int u = __builtin_ctzll(P|X); ll c = P&~G[u];
  while(c){
    int v = __builtin_ctzll(c);
    get_clique(R + 1, P&G[v], X&G[v], V | 1ULL << v);
    P ^= 1ULL << v; X |= 1ULL << v; c ^= 1ULL << v;
  }
}
```

3.5 Bipartite Matching

```
const int S = 333;
vector<int> g[S];
int dst[S], l[S], r[S], chk[S];
void clear(){ for(int i=0; i<S; i++) g[i].clear(); }
void addEdge(int s, int e){ g[s].push_back(e); }
int bfs(int n){
  queue<int> q; int ret = 0;
  memset(dst, 0, sizeof dst);
  for(int i=1; i<=n; i++) if(l[i] == -1 && !dst[i])
    q.push(i); dst[i] = 1;
  while(!q.empty()){
    int v = q.front(); q.pop();
    for(auto i : g[v]){
      if(r[i] == -1) ret = 1;
      else if(!dst[r[i]]) dst[r[i]] = dst[v] + 1, q.push(r[i]);
    }
  }
  return ret;
}
int dfs(int v){
  if(chk[v]) return 0; chk[v] = 1;
  for(auto i : g[v]){
    if(r[i] == -1 || !chk[r[i]] && dst[r[i]] == dst[v]+1 && dfs(r[i])){
      l[v] = i; r[i] = v; return 1;
    }
  }
  return 0;
}
int match(int n){
  memset(l, -1, sizeof l); memset(r, -1, sizeof r);
  int ret = 0;
  while(bfs(n)){
    memset(chk, 0, sizeof chk);
    for(int i=1; i<=n; i++) if(l[i] == -1 && dfs(i)) ret++;
  }
  return ret;
}
int track[S+S];
void rdfs(int v, int n){
  if(track[v]) return; track[v] = 1;
  for(auto i : g[v]) track[i+n] = 1, rdfs(r[i], n);
}
vector<int> getCover(int n, int m){
  match(n); memset(track, 0, sizeof track);
  for(int i=1; i<=n; i++) if(l[i] == -1) rdfs(i, n);
  vector<int> ret;
  for(int i=1; i<=n; i++) if(!track[i]) ret.push_back(i);
  for(int i=n+1; i<=n+m; i++) if(track[i]) ret.emplace_back(i);
  return ret;
}
```

3.6 Maximum Flow, Minimum Cut

```
template<typename FlowType, size_t _Sz, FlowType _Inf=1'000'000'007>
struct Dinic{
  struct Edge{ int v, dual; FlowType c; };
  int Level[_Sz], Work[_Sz];
  vector<Edge> G[_Sz];
  void clear(){ for(int i=0; i<_Sz; i++) G[i].clear(); }
  void AddEdge(int s, int e, FlowType x){
    G[s].push_back({e, (int)G[e].size(), x});
    G[e].push_back({s, (int)G[s].size()-1, 0});
  }
  bool BFS(int S, int T){
    memset(Level, 0, sizeof Level);
    queue<int> Q; Q.push(S); Level[S] = 1;
    while(Q.size()){
      int v = Q.front(); Q.pop();
      for(const auto &i : G[v]){
        if(!Level[i.v] && i.c) Q.push(i.v), Level[i.v] = Level[v] + 1;
      }
    }
    return Level[T];
  }
  FlowType DFS(int v, int T, FlowType tot){
    if(v == T) return tot;
    for(int &_i=Work[v]; _i<G[v].size(); _i++){
      Edge &i = G[v][_i];
      if(Level[i.v] != Level[v] + 1 || !i.c) continue;
      FlowType fl = DFS(i.v, T, min(tot, i.c));
      if(!fl) continue;
      i.c -= fl; G[i.v][i.dual].c += fl;
      return fl;
    }
    return 0;
  }
  FlowType MaxFlow(int S, int T){
    FlowType ret = 0, tmp;
    while(BFS(S, T)){
      memset(Work, 0, sizeof Work);
      while((tmp = DFS(S, T, _Inf))) ret += tmp;
    }
    return ret;
  }
  tuple<FlowType, vector<int>, vector<int>> MinCut(int S, int T){
    FlowType fl = MaxFlow(S, T);
    vector<int> a, b;
    const int Bias = 1e9;
    queue<int> Q; Q.push(S); Level[S] += Bias;
    while(Q.size()){
      int v = Q.front(); Q.pop();
      for(const auto &i : G[v]){
        if(Level[i.v] < Bias) Q.push(i.v), Level[i.v] += Bias;
      }
    }
    for(int i=0; i<_Sz; i++){
      if(Level[i]) a.push_back(i);
      else b.push_back(i);
    }
    return make_tuple(fl, a, b);
  }
};
```

3.7 MCMF

```
template<typename FlowType, typename CostType, int _Sz, FlowType _Inf_flow, CostType _Inf_cost>
struct MCMF{
  struct Edge{ int nxt; FlowType cap; CostType cst; int rev; };
  vector<Edge> G[_Sz];
  bitset<_Sz> Visit;
  CostType Dist[_Sz];
  int Used[_Sz];
  void AddEdge(int s, int e, FlowType cap, CostType cst){
    G[s].push_back({e, cap, cst, SZ(G[e])});
    G[e].push_back({s, 0, -cst, SZ(G[s])-1});
  }
  void Init(const int S, const int T){
    fill(Dist, Dist+_Sz, _Inf_cost);
    queue<int> Q;
    bitset<_Sz> InQ;
    Q.push(S); InQ[S] = true; Dist[S] = 0;
    while(!Q.empty()){
      int now = Q.front(); Q.pop(); InQ[now] = false;
      for(const auto &i : G[now]){
        if(i.cap > 0 && Dist[i.nxt] > Dist[now] + i.cst){
          Dist[i.nxt] = Dist[now] + i.cst;
          if(!InQ[i.nxt]) InQ[i.nxt] = true, Q.push(i.nxt);
        }
      }
    }
  }
  bool Update(){
    CostType fix = _Inf_cost;
    for(int i=0; i<_Sz; i++){
      if(!Visit[i]) continue;
      for(const auto &j : G[i]){
        if(j.cap && !Visit[j.nxt]) fix = min(fix, Dist[i] + j.cst - Dist[j.nxt]);
      }
    }
    if(fix == _Inf_cost) return false;
    for(int i=0; i<_Sz; i++) if(!Visit[i]) Dist[i] += fix;
    return true;
  }
  FlowType DFS(int now, int sink, FlowType fl){
    Visit[now] = true;
    if(now == sink) return fl;
    for(; Used[now] < G[now].size(); Used[now]++){
      auto &i = G[now][Used[now]];
      if(!Visit[i.nxt] && i.cap > 0 && Dist[i.nxt] == Dist[now] + i.cst){
        FlowType ret = DFS(i.nxt, sink, min(fl, i.cap));
        if(ret > 0){
          i.cap -= ret;
          G[i.nxt][i.rev].cap += ret;
```

```
        return ret;
      }
    }
  }
  return 0;
}
pair<FlowType, CostType> MinCostFlow(int S, int T){
  FlowType flow = 0, tmp;
  CostType cost = 0;
  Init(S, T);
  do{
    Visit.reset();
    memset(Used, 0, sizeof Used);
    while((tmp = DFS(S, T, _Inf_flow))){
      flow += tmp;
      cost += tmp * Dist[T];
      Visit.reset();
    }
  }while(Update());
  return make_pair(flow, cost);
}
};
```

3.8 LR Flow

```
addEdge(t, s, inf) // 기존 싱크 -> 기존 소스 inf
addEdge(s, nt, l) // s -> 새로운 싱크 l
addEdge(ns, e, l) // 새로운 소스 -> e l
addEdge(a, b, r-l) // s -> e (r-l)
// ns -> nt의 max flow == l들의 합 확인
// maxflow : s -> t 플로우 찾을 수 있을 때까지 반복
```

3.9 Hungarian Method

```
// 1-based, only for minimum matching, maximum matching may get TLE
template<typename cost_t=int, cost_t _INF=0x3f3f3f3f>
struct Hungarian{
  int n;
  vector<vector<cost_t>> mat;
  Hungarian(int n) : n(n), mat(n+1, vector<cost_t>(n+1, _INF)) {}
  void addEdge(int s, int e, cost_t x){
    mat[s][e] = min(mat[s][e], x);
  }
  cost_t run(){
    vector<cost_t> u(n+1), v(n+1), m(n+1);
    vector<int> p(n+1), w(n+1), c(n+1);
    for(int i=1,a,b; i<=n; i++){
      p[0] = i; b = 0;
      fill(m.begin(), m.end(), _INF);
      fill(c.begin(), c.end(), 0);
      do{
        int nxt; cost_t delta = _INF;
        c[b] = 1; a = p[b];
        for(int j=1; j<=n; j++){
          if(c[j]) continue;
          cost_t t = mat[a][j] - u[a] - v[j];
          if(t < m[j]) m[j] = t, w[j] = b;
          if(m[j] < delta) delta = m[j], nxt = j;
        }
        for(int j=0; j<=n; j++){
          if(c[j]) u[p[j]] += delta, v[j] -= delta;
          else m[j] -= delta;
        }
        b = nxt;
      }while(p[b] != 0);
      do{
        int nxt = w[b];
        p[b] = p[nxt];
        b = nxt;
      }while(b != 0);
    }
    return -v[0];
  }
};
```

3.10 Gomory-Hu Tree

```
// 0-based, S-T cut in graph == S-T cut in gomory-hu tree (path minimum)
vector<Edge> GomoryHuTree(int n, const vector<Edge> &e){
    Dinic<int,100> Flow;
    vector<Edge> res(n-1); vector<int> pr(n);
    for(int i=1; i<n; i++, Flow.clear()){
        for(const auto &[s,e,x] : e) Flow.AddEdge(s, e, x); // bi-directed
        int fl = Flow.MaxFlow(pr[i], i);
        for(int j=i+1; j<n; j++){
            if(!Flow.Level[i] == !Flow.Level[j] && pr[i] == pr[j]) pr[j] = i;
        }
        res[i-1] = Edge(pr[i], i, fl);
    }
    return res;
}
```

3.11 $O(V^3)$ Global Min Cut

```
int vertex, g[S][S], dst[S], chk[S], del[S];
void init(){
  memset(g, 0, sizeof g); memset(del, 0, sizeof del);
}
void addEdge(int s, int e, int x){ g[s][e] = g[e][s] = x; }
int minCutPhase(int &s, int &t){
  memset(dst, 0, sizeof dst);
  memset(chk, 0, sizeof chk);
  int mincut = 0;
  for(int i=1; i<=vertex; i++){
    int k = -1, mx = -1;
    for(int j=1; j<=vertex; j++) if(!del[j] && !chk[j])
      if(dst[j] > mx) k = j, mx = dst[j];
    if(k == -1) return mincut;
    s = t, t = k;
    mincut = mx, chk[k] = 1;
    for(int j=1; j<=vertex; j++){
      if(!del[j] && !chk[j]) dst[j] += g[k][j];
```

```
    }
  }
  return mincut;
}
int getMinCut(int n){
  vertex = n; int mincut = 1e9+7;
  for(int i=1; i<vertex; i++){
    int s, t;
    int now = minCutPhase(s, t);
    mincut = min(mincut, now); del[t] = 1;
    if(mincut == 0) return 0;
    for(int j=1; j<=vertex; j++){
      if(!del[j]) g[s][j] = (g[j][s] += g[j][t]);
    }
  }
  return mincut;
}
```

3.12 $O((V+E)\log V)$ Dominator Tree

```
vector<int> DominatorTree(const vector<vector<int>> &g, int src){ // // 0-based
  int n = g.size();
  vector<vector<int>> rg(n), buf(n);
  vector<int> r(n), val(n), idom(n, -1), sdom(n, -1), o, p(n), u(n);
  iota(all(r), 0); iota(all(val), 0);
  for(int i=0; i<n; i++) for(auto j : g[i]) rg[j].push_back(i);
  function<int(int)> find = [&](int v){
    if(v == r[v]) return v;
    int ret = find(r[v]);
    if(sdom[val[v]] > sdom[val[r[v]]]) val[v] = val[r[v]];
    return r[v] = ret;
  };
  function<void(int)> dfs = [&](int v){
    sdom[v] = o.size(); o.push_back(v);
    for(auto i : g[v]) if(sdom[i] == -1) p[i] = v, dfs(i);
  };
  dfs(src); reverse(all(o));
  for(auto &i : o){
    if(sdom[i] == -1) continue;
    for(auto j : rg[i]){
      if(sdom[j] == -1) continue;
      int x = val[find(j), j];
      if(sdom[i] > sdom[x]) sdom[i] = sdom[x];
    }
    buf[o[o.size() - sdom[i] - 1]].push_back(i);
    for(auto j : buf[p[i]]) u[j] = val[find(j), j];
    buf[p[i]].clear();
    r[i] = p[i];
  }
  reverse(all(o)); idom[src] = src;
  for(auto i : o){ // WARNING : if different, takes idom
    if(i != src) idom[i] = sdom[i] == sdom[u[i]] ? sdom[i] : idom[u[i]];
  }
  for(auto i : o) if(i != src) idom[i] = o[idom[i]];
  return idom; // unreachable -> ret[i] = -1
}
```

3.13 $O(N^2)$ Stable Marriage Problem

```
// man : 1~n, woman : n+1~2n
struct StableMarriage{
  int n;
  vector<vector<int>> g;
  StableMarriage(int n) : n(n), g(2*n+1) {
    for(int i=1; i<=n+n; i++) g[i].reserve(n);
  }
  void addEdge(int u, int v){ // insert in decreasing order of preference.
    g[u].push_back(v);
  }
  vector<int> run(){
    queue<int> q;
    vector<int> match(2*n+1), ptr(2*n+1);
    for(int i=1; i<=n; i++) q.push(i);
    while(q.size()){
      int i = q.front(); q.pop();
      for(int &p=ptr[i]; p<g[i].size(); p++){
        int j = g[i][p];
        if(!match[j]){ match[i] = j; match[j] = i; break; }
        int m = match[j], u = -1, v = -1;
        for(int k=0; k<g[j].size(); k++){
          if(g[j][k] == i) u = k;
          if(g[j][k] == m) v = k;
        }
        if(u < v){
          match[m] = 0; q.push(m);
          match[i] = j; match[j] = i;
          break;
        }
      }
    }
    return match;
  }
};
```

3.14 $O(VE)$ Vizing Theorem

```
// Graph coloring with (max-degree)+1 colors, O(N^2)
int C[MX][MX] = {}, G[MX][MX] = {}; // MX ~= 2500
void solve(vector<pii> &E, int N, int M){
  int X[MX] = {}, a, b;
  auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };
  auto color = [&](int u, int v, int c){
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v; C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if( p ) X[u] = X[v] = p;
    else update(u), update(v);
    return p; }; // end of function : color
  auto flip = [&](int u, int c1, int c2){
    int p = C[u][c1], q = C[u][c2];
    swap(C[u][c1], C[u][c2]);
    if( p ) G[u][p] = G[p][u] = c2;
```

```
    if( !C[u][c1] ) X[u] = c1;
    if( !C[u][c2] ) X[u] = c2;
    return p; }; // end of function : flip
  for(int i = 1; i <= N; i++) X[i] = 1;
  for(int t = 0; t < E.size(); t++){
    int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d;
    vector<pii> L;
    int vst[MX] = {};
    while(!G[u][v0]){
      L.emplace_back(v, d = X[v]);
      if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c);
      else if(!C[u][d])for(a=(int)L.size()-1;a>=0;a--)color(u,L[a].first,L[a].second);
      else if( vst[d] ) break;
      else vst[d] = 1, v = C[u][d];
    }
    if( !G[u][v0] ){
      for(;v; v = flip(v, c, d), swap(c, d));
      if(C[u][c0]){
        for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);
        for(; a >= 0; a--) color(u, L[a].first, L[a].second);
      } else t--;
    }
  }
}
```

3.15 $O(E \log V)$ Directed MST

```
struct Edge{
  int s, e; cost_t x;
  Edge() = default;
  Edge(int s, int e, cost_t x) : s(s), e(e), x(x) {}
  bool operator < (const Edge &t) const { return x < t.x; }
};
struct UnionFind{
  vector<int> P, S;
  vector<pair<int, int>> stk;
  UnionFind(int n) : P(n), S(n, 1) { iota(P.begin(), P.end(), 0); }
  int find(int v) const { return v == P[v] ? v : find(P[v]); }
  int time() const { return stk.size(); }
  void rollback(int t){
    while(stk.size() > t){
      auto [u,v] = stk.back(); stk.pop_back();
      P[u] = u; S[v] -= S[u];
    }
  }
  bool merge(int u, int v){
    u = find(u); v = find(v);
    if(u == v) return false;
    if(S[u] > S[v]) swap(u, v);
    stk.emplace_back(u, v);
    S[v] += S[u]; P[u] = v;
    return true;
  }
};
struct Node{
  Edge key;
  Node *l, *r;
  cost_t lz;
  Node() : Node(Edge()) {}
  Node(const Edge &edge) : key(edge), l(nullptr), r(nullptr), lz(0) {}
  void push(){
    key.x += lz;
    if(l) l->lz += lz;
    if(r) r->lz += lz;
    lz = 0;
  }
  Edge top(){ push(); return key; }
};
Node* merge(Node *a, Node *b){
  if(!a || !b) return a ? a : b;
  a->push(); b->push();
  if(b->key < a->key) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node* &a){ a->push(); a = merge(a->l, a->r); }

// 0-based
pair<cost_t, vector<int>> DirectMST(int n, int rt, vector<Edge> &edges){
  vector<Node*> heap(n);
  UnionFind uf(n);
  for(const auto &i : edges) heap[i.e] = merge(heap[i.e], new Node(i));
  cost_t res = 0;
  vector<int> seen(n, -1), path(n), par(n);
  seen[rt] = rt;
  vector<Edge> Q(n), in(n, {-1,-1, 0}), comp;
  deque<tuple<int, int, vector<Edge>>> cyc;
  for(int s=0; s<n; s++){
    int u = s, qi = 0, w;
    while(seen[u] < 0){
      if(!heap[u]) return {-1, {}};
      Edge e = heap[u]->top();
      heap[u]->lz -= e.x; pop(heap[u]);
      Q[qi] = e; path[qi++] = u; seen[u] = s;
      res += e.x; u = uf.find(e.s);
      if(seen[u] == s){ // found cycle, contract
        Node* nd = 0;
        int end = qi, time = uf.time();
        do nd = merge(nd, heap[w = path[--qi]]); while(uf.merge(u, w));
        u = uf.find(u); heap[u] = nd; seen[u] = -1;
        cyc.emplace_front(u, time, vector<Edge>{&Q[qi], &Q[end]});
      }
    }
    for(int i=0; i<qi; i++) in[uf.find(Q[i].e)] = Q[i];
  }
  for(auto& [u,t,comp] : cyc){
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.e)] = e;
    in[uf.find(inEdge.e)] = inEdge;
  }
  for(int i=0; i<n; i++) par[i] = in[i].s;
```

```
  return {res, par};
}
```

3.16 $O(V^3)$ General Matching

```
#define zer(x) memset(x, 0, sizeof x)
#define fu(x) memset(x, -1, sizeof x)
int n;
vector<int> g[SZ];
int match[SZ], par[SZ], chk[SZ], gid[SZ], color[SZ];
void addEdge(int s, int e){
  g[s].push_back(e); g[e].push_back(s);
}
int lca_chk[SZ], pv;
int lca(int rt, int u, int v){ pv++;
  while(u != rt) lca_chk[u] = pv, u = gid[par[match[u]]];
  while(v != rt){
    if(lca_chk[v] == pv) return v; v = gid[par[match[v]]];
  }
  return rt;
}
void group(int l, int u, int v){
  while(l != gid[u]){
    int vv = match[u], uu = par[vv];
    chk[vv] = 1; par[u] = v; gid[u] = gid[vv] = l;
    u = uu; v = vv;
  }
}
void add_match(int rt, int v){
  while(par[v] != rt){
    int p = par[v], vv = match[p];
    match[v] = p; match[p] = v; match[vv] = 0;
    v = vv;
  }
  match[v] = rt; match[rt] = v;
}
int arg(int st){
  zer(par); zer(chk); fu(color); iota(gid, gid+SZ, 0);
  queue<int> q; q.push(st); chk[st] = 1; color[st] = 0;
  while(q.size()){
    int v = q.front(); q.pop();
    for(auto i : g[v]){
      if(color[i] == -1){
        par[i] = v; color[i] = 1;
        if(!match[i]){ add_match(st, i); return 1; }
        color[match[i]] = 0; chk[match[i]] = 1;
        q.push(match[i]);
      }
      else if(!color[i] && gid[v] != gid[i]){
        int l = lca(gid[st], gid[v], gid[i]);
        group(l, v, i); group(l, i, v);
        for(int j=1; j<=n; j++) if(chk[j] && color[j])
          color[j] = 0, q.push(j);
      }
    }
  }
  return 0;
}
int run(int _n){
  n = _n; int ret = 0;
  for(int i=1; i<=n; i++) if(!match[i] && arg(i)) ret++;
  return ret;
}
```

3.17 $O(V^3)$ Weighted General Matching

```
int n, n_x;
edge g[N*2][N*2];
int lab[N*2];
int match[N*2], slack[N*2], st[N*2], pa[N*2];
int flo_from[N*2][N+1], S[N*2], vis[N*2];
vector<int> flo[N*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(int 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(int i=0; i<flo[x].size(); i++) set_st(flo[x][i], b);
}
int get_pr(int b, int xr){
  int pr=find(all(flo[b]), xr) - flo[b].begin();
  if(pr & 1){ reverse(1 + all(flo[b])); return flo[b].size() - pr; }
  else return pr;
}
void set_match(int u, int v){
  edge e = g[u][v]; match[u] = g[u][v].v; if(u <= n) return;
  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(), pr+all(flo[u]));
}
void augment(int u, int v){
  while(true){
    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; // new blossom
  lab[b] = 0; S[b]=0; match[b] = match[lca]; flo[b] = vector<int>{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(1 + all(flo[b]));
  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(int 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(int 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(int i=pr+1; i<flo[b].size(); i++) S[flo[b][i]] = -1, set_slack(flo[b][i]);
  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] = S[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;
  while(true){
    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, w_max = 0; long long tot_weight = 0;
  for(int u=0; u<=n; u++) st[u] = u, flo[u].clear();
  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 u, int v, int w){ g[u][v].w = g[v][u].w = w; }
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);
}
```

4 Math

4.1 Extend GCD, CRT, Combination

```
// ll gcd(ll a, ll b), ll lcm(ll a, ll b), ll mod(ll a, ll b)
ll ext_gcd(ll a, ll b, ll &x, ll &y) { //ax + by = gcd(a, b)
```

```cpp
  ll g = a; x = 1, y = 0;
  if (b) g = ext_gcd(b, a % b, y, x), y -= a / b * x;
  return g;
}
ll inv(ll a, ll m){ //return x when (ax mod m = 1), fail -> -1
  ll x, y, g = ext_gcd(a, m, x, y);
  if(g > 1) return -1;
  return mod(x, m);
}
pair<ll,ll> crt(ll a1, ll m1, ll a2, ll m2){ // Return (z, M), fail -> M = -1
  ll s, t; ll g = ext_gcd(m1, m2, s, t);
  if(a1 % g != a2 % g) return {0, -1};
  s = mod(s*a2, m2); t = mod(t*a1, m1);
  ll res = mod(s*(m1/g) + t*(m2/g), m1/g*m2), M = m1/g*m2;
  return {res, M};
}
pair<ll,ll> crt(const vector<ll> &a, const vector<ll> &m){
  if(a.size() == 1) return {a[0], m[0]};
  pair<ll,ll> ret = crt(a[0], m[0], a[1], m[1]);
  for(int i=2; i<a.size(); i++) ret = crt(ret.x, ret.y, a[i], m[i]);
  return ret;
}
struct Lucas{ // init : O(P), query : O(log P)
  const size_t P;
  vector<ll> fac, inv;
  ll Pow(ll a, ll b){
    ll ret = 1;
    for(; b; b>>=1, a=a*a%P) if(b&1) ret=ret*a%P;
    return ret;
  }
Lucas(size_t P) : P(P), fac(P), inv(P) {
    fac[0] = 1;
    for(int i=1; i<P; i++) fac[i] = fac[i-1] * i % P;
    inv[P-1] = Pow(fac[P-1], P-2);
    for(int i=P-2; ~i; i--) inv[i] = inv[i+1] * (i+1) % P;
  }
  ll small(ll n, ll r) const {
    if(n < r) return 0;
    return fac[n] * inv[r] % P * inv[n-r] % P;
  }
  ll calc(ll n, ll r) const {
    if(n < r || n < 0 || r < 0) return 0;
    if(!n || !r || n == r) return 1;
    return small(n%P, r%P) * calc(n/P, r/P) % P;
  }
};
template<ll p, ll e> struct CombinationPrimePower{ // init : O(p^e), query : O(log p)
  vector<ll> val; ll m;
  CombinationPrimePower(){
    val.resize(m); val[0] = 1; m = 1; for(int i=0; i<e; i++) m *= p;
    for(int i=1; i<m; i++) val[i] = val[i-1] * (i % p ? i : 1) % m;
  }
  pair<ll,ll> factorial(int n){
    if(n < p) return {0, val[n]};
    int k = n / p; auto v = factorial(k);
    int cnt = v.first + k, kp = n / m, rp = n % m;
    ll ret = v.second * Pow(val[m-1], kp % 2, m) % m * val[rp] % m;
    return {cnt, ret};
  }
  ll calc(int n, int r){
    if(n < 0 || r < 0 || n < r) return 0;
    auto v1 = factorial(n), v2 = factorial(r), v3 = factorial(n-r);
    ll cnt = v1.first - v2.first - v3.first;
    ll ret = v1.second * inv(v2.second, m) % m * inv(v3.second, m) % m;
    if(cnt >= e) return 0;
    for(int i=1; i<=cnt; i++) ret = ret * p % m;
    return ret;
  }
};
```

4.2 FloorSum

```cpp
// sum of floor((A*i+B)/M) over 0 <= i < N in O(log(N+M+A+B))
ll FloorSum(ll N, ll M, ll A, ll B){ // 1 <= N,M <= 1e9, 0 <= A,B < M
  ll R = 0;
  if(A >= M) R += N * (N - 1) / 2 * (A / M), A %= M;
  if(B >= M) R += B / M * N, B %= M;
  ll Y = (A * N + B) / M, X = Y * M - B;
  if(Y == 0) return R;
  R += (N - (X + A - 1) / A) * Y;
  R += FloorSum(Y, A, M, (A - X % A) % A);
  return R;
}
```

4.3 XOR Basis(XOR Maximization)

```cpp
// can use greedy maximize
//((staircase basis, basis coefficient),selected basis indices)
// staircase basis: has some good property
// basis coefficient and selected basis indices: for reconstruct
pair<Arr<pint>, Arr<int>> xor_basis(const Arr<int>& a) {
  Arr<pint> r(64, {-1, -1});  // descending
  Arr<int> bi;
  for(int i = 0; i < sz(a); i++) {
    int x = a[i], xc = 0;
    for(auto [b, bc] : r)
      if(~b and x > (x ^ b)) x ^= b, xc ^= bc;
    if(x) r[63 - lg2f(x)] = {x, xc ^ (1ll << sz(bi))}, bi.pushb(i);
  }
  return {move(r), move(bi)};
}
```

4.4 Gauss Jordan Elimination

```cpp
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, T, T, vector<vector<T>>> Gauss(vector<vector<T>> a, bool square=true){
  int n = a.size(), m = a[0].size(), rank = 0;
  vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
  for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
  for(int i=0; i<m; i++){
    if(rank == n) break;
    if(IsZero(a[rank][i])){
```

```
    T mx = T(0); int idx = -1; // fucking precision error
    for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx = abs(a[j][i]), idx = j;
    if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
    for(int k=0; k<m; k++){
      a[rank][k] = Add(a[rank][k], a[idx][k]);
      if(square) out[rank][k] = Add(out[rank][k], out[idx][k]);
    }
  }
  det = Mul(det, a[rank][i]);
  T coeff = Div(T(1), a[rank][i]);
  for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
  for(int j=0; j<m; j++) if(square) out[rank][j] = Mul(out[rank][j], coeff);
  for(int j=0; j<n; j++){
    if(rank == j) continue;
    T t = a[j][i]; // Warning: [j][k], [rank][k]
    for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k], Mul(a[rank][k], t));
    for(int k=0; k<m; k++) if(square) out[j][k] = Sub(out[j][k], Mul(out[rank][k], t));
  }
  rank++;
 }
 return {a, rank, det, out};
}
```

4.5 Gauss Jordan Elimination (Binary)

```
//NOTE: n*m행렬이다. 코드는 <m>(n)형태니 조심
#pragma GCC optimize ("Ofast")
int REF(bool pv_fix){
  int pi = 0;
  for(int i=0; i<n; i++){
    while(pi < m && !a[i][pi]){
      if(!pv_fix){
        for(int j=i+1; j<n; j++) if(a[j][pi]) break;
        if(j < n){ swap(a[i], a[j]); break; }
      }
      pi++;
    }
    if(pi == m)break;
    if(a[i][pi])for(int j=i+1;j<n;j++)if(a[j][pi])a[j]^=a[i];
    pi++;
  }
  return rank = i;
}
int RREF(int rm){REF(false),flipX(rank),flipY(rm),REF(true),flipY(rm),flipX(rank);return rank;}
void flipX(int rn){
  for(int i=0; i<rn/2; i++) swap(a[i], a[rn-1-i]);
}
void flipY(int rm){
  for(int i=0;i<n;i++) for(int j=0;j<rm/2;j++) swap(a[i][j], a[i][rm-1-j]);
}
```

4.6 Berlekamp + Kitamasa

Time Complexity: $O(NK + N \log mod), O(N^2 \log X)$

```
const int mod = 1e9+7;
ll pw(ll a, ll b){
  ll ret = 1; a %= mod;
  while(b){
    if(b & 1) ret = ret * a % mod;
    b >>= 1; a = a * a % mod;
  }
  return ret;
}
vector<int> berlekamp_massey(vector<int> x){
  vector<int> ls, cur;
  int lf, ld;
  for(int i=0; i<x.size(); i++){
    ll t = 0;
    for(int j=0; j<cur.size(); j++) t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
    if((t - x[i]) % mod == 0) continue;
    if(cur.empty()){
      cur.resize(i+1);
      lf = i; ld = (t - x[i]) % mod;
      continue;
    }
    ll k = -(x[i] - t) * pw(ld, mod - 2) % mod;
    vector<int> c(i-lf-1); c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % mod);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j=0; j<cur.size(); j++) c[j] = (c[j] + cur[j]) % mod;
    if(i-lf+(int)ls.size()>=(int)cur.size()){
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
    }
    cur = c;
  }
  for(auto &i : cur) i = (i % mod + mod) % mod;
  return cur;
}
int get_nth(vector<int> rec, vector<int> dp, ll n){
  int m = rec.size(); vector<int> s(m), t(m);
  s[0] = 1;
  if(m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](vector<int> v, vector<int> w){
    int m = v.size();
    vector<int> t(2 * m);
    for(int j=0; j<m; j++) for(int k=0; k<m; k++){
      t[j+k] += 1ll * v[j] * w[k] % mod;
      if(t[j+k] >= mod) t[j+k] -= mod;
    }
    for(int j=2*m-1; j>=m; j--) for(int k=1; k<=m; k++){
      t[j-k] += 1ll * t[j] * rec[k-1] % mod;
      if(t[j-k] >= mod) t[j-k] -= mod;
    }
    t.resize(m);
    return t;
  };
  while(n){
    if(n & 1) s = mul(s, t);
    t = mul(t, t); n >>= 1;
  }
  ll ret = 0;
```

```
  for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
  return ret % mod;
}
int guess_nth_term(vector<int> x, ll n){
  if(n < x.size()) return x[n];
  vector<int> v = berlekamp_massey(x);
  if(v.empty()) return 0;
  return get_nth(v, x, n);
}
```

4.7 Miller Rabin + Pollard Rho

```
ull mul(ull a, ull b, ull mod){ return (__uint128_t)a * b % mod; }
bool isp[10101010];
vector<int> prime;
void sieve(){
  memset(isp, 1, sizeof isp);
  isp[0] = isp[1] = 0;
  for(ll i=2; i<=10000000; i++){
    if(isp[i]) prime.push_back(i);
    for(auto j : prime){
      if(i*j > 10000000) break;
      isp[i*j] = 0;
      if(i % j == 0) break;
    }
  }
}
// 32bit : 2, 7, 61
// 64bit : 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool MR(ull n, ull a){ // Miller Rabin
  if(a % n == 0) return 1;
  int cnt = __builtin_ctzll(n-1);
  ull d = n >> cnt;
  ull p = pw(a, d, n);
  if(p == 1 || p == n - 1) return 1;
  while(cnt--){
    p = mul(p, p, n);
    if(p == n-1) return 1;
  }
  return 0;
}
bool isPrime(ll n){
  if(n < 10000001) return isp[n];
  if(n <= 2) return n == 2;
  if(!(n & 1)) return 0;
  if(n%3 == 0 || n%5 == 0 || n%7 == 0 || n%11 == 0) return 0;
  for(int p : {2,325,9375,28178,450775,9780504,1795265022}){
    if(!MR(n, p)) return 0;
  }
  return 1;
}
ll rho(ll n){
  while(1){
    ll x = rand() % (n - 2) + 2;
    ll y = x, c = rand() % (n-1) + 1;
    while(1){
      x = (mul(x, x, n) + c) % n;
      y = (mul(y, y, n) + c) % n;
      y = (mul(y, y, n) + c) % n;
      ll d = __gcd(abs(x - y), n);
      if(d == 1) continue;
      if(!isPrime(d)){ n = d; break; }
      else return d;
    }
  }
}
vector<ll> pollard_rho(ll n){
  vector<ll> v;
  while(~n & 1) n >>= 1, v.push_back(2);
  if(n == 1) return move(v);
  while(!isPrime(n)){
    ll d = rho(n);
    while(n % d == 0) v.push_back(d), n /= d;
    if(n == 1) break;
  }
  if(n != 1) v.push_back(n);
  return move(v);
}
```

4.8 Linear Sieve

```
// sp : 최소 소인수, 소수라면 0
// tau : 약수 개수, sigma : 약수 합
// phi : n 이하 자연수 중 n과 서로소인 개수
// mu : non square free이면 0, 그렇지 않다면 (-1)^(소인수 종류)
// e[i] : 소인수분해에서 i의 지수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
  if(!sp[i]){
    prime.push_back(i);
    e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i] = i+1;
  }
  for(auto j : prime){
    if(i*j >= sz) break;
    sp[i*j] = j;
    if(i % j == 0){
      e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
      tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
      sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j])-1)*(pw(j, e[i*j]+1)-1)/(j-1);//overflow
      break;
    }
    e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] * mu[j];
    tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] * sigma[j];
  }
}
```

4.9 Discrete Log / Sqrt

Time Complexity: Log : $O(\sqrt{P}\log P)$, $O(\sqrt{P})$ with hash set
Sqrt : $O(\log^2 P)$, $O(\log P)$ in random data

```
// Given A, B, P, solve A^x === B mod P
ll DiscreteLog(ll A, ll B, ll P){
  __gnu_pbds::gp_hash_table<ll,__gnu_pbds::null_type> st;
  ll t = ceil(sqrt(P)), k = 1; // use binary search?
  for(int i=0; i<t; i++) st.insert(k), k = k * A % P;
  ll inv = Pow(k, P-2, P);
  for(int i=0, k=1; i<t; i++, k=k*inv%P){
    ll x = B * k % P;
    if(st.find(x) == st.end()) continue;
    for(int j=0, k=1; j<t; j++, k=k*A%P){
      if(k == x) return i * t + j;
    }
  }
  return -1;
}
// Given A, P, solve X^2 === A mod P
ll DiscreteSqrt(ll A, ll P){
  if(A == 0) return 0;
  if(Pow(A, (P-1)/2, P) != 1) return -1;
  if(P % 4 == 3) return Pow(A, (P+1)/4, P);
  ll s = P - 1, n = 2, r = 0, m;
  while(~s & 1) r++, s >>= 1;
  while(Pow(n, (P-1)/2, P) != P-1) n++;
  ll x = Pow(A, (s+1)/2, P), b = Pow(A, s, P), g = Pow(n, s, P);
  for(;; r=m){
    ll t = b;
    for(m=0; m<r && t!=1; m++) t = t * t % P;
    if(!m) return x;
    ll gs = Pow(g, 1LL << (r-m-1), P);
    g = gs * gs % P;
    x = x * gs % P;
    b = b * g % P;
  }
}
```

4.10 De Bruijn Sequence

```
// Create cyclic string of length k^n that contains every length n string as substring. alphabet
= [0, k - 1]
int res[10000000], aux[10000000]; // >= k^n
int de_bruijn(int k, int n) { // Returns size (k^n)
  if(k == 1) { res[0] = 0; return 1; }
  for(int i = 0; i < k * n; i++) aux[i] = 0;
  int sz = 0;
  function<void(int, int)> db = [&](int t, int p) {
    if(t > n) {
      if(n % p == 0) for(int i = 1; i <= p; i++) res[sz++] = aux[i];
    }
    else {
      aux[t] = aux[t - p]; db(t + 1, p);
      for(int i = aux[t - p] + 1; i < k; i++) aux[t] = i, db(t + 1, t);
    }
  };
  db(1, 1);
  return sz;
}
```

4.11 Simplex / LP Duality

```
//입력: Ax<=b, obj
//출력: maximize obj*x
//numeric stability is sensitive by M
//디버깅 노트
//1. T=f64 해보기(정수값만 나오는거같아도 중간에 유리수나올때 있음)
//2. M값 조절(답의 상한정도의 크기가 적절)
//듀얼후 리덕션한 결과값 primal로 복원하기
template<class T=f64,int M>
void dualize(Arr<Arr<T>> &a,Arr<T> &b,Arr<T>& obj){
  int m=sz(a), n=sz(a[0]);
  transpose(a),swap(b,obj);
  for(int i=0;i<n;i++){
    for(auto& j:a[i])j=-j;
    b[i]=-b[i];
  }
  for(auto& i:obj)i=-i;
}
template<class T=f64,int M>
tuple<T,Arr<T>,Arr<T>> simplex(Arr<Arr<T>>& a,Arr<T>& b,Arr<T>& obj){
  //return {maxval,argmax,dual_argmin}
  int m=sz(a),n=sz(a[0]),s=0;
  if(m>n){
    dualize<T,M>(a,b,obj);
    auto&& [x,y,z]=simplex<T,M>(a,b,obj);
    x*=-1;
    swap(y,z);
    return {move(x),move(y),move(z)};
  }
  func(void,elim,int r1,int r2,int c){//elim r2
    if(r1==r2){T x=a[r1][c]; for(auto& i:a[r1])i/=x;}
    else{
      T x=a[r2][c]/a[r1][c]; if(-eps<x&&x<eps)return;
      for(int i=0;i<n+s+m+2;i++)
        a[r2][i]-=x*a[r1][i];
    }
  };

  //make all b>=0
  Arr<char> geq(m);
  for(int i=0;i<m;i++)
    if(b[i]<0){
      for(auto& j:a[i])j=-j;
      for(auto& r:a)r.emplb(0);
      a[i][-1]=-1,b[i]=-b[i],geq[i]=true,s++;
    }

  //n vars, s slacks(-1), m slacks(1), 1 z, 1 b_value
  Arr<int> p(m);//행의 기본변수
  obj.resize(n+s+m+2);
  for(int i=0;i<m;i++)
    a[i].resize(n+s+m+2),a[i][p[i]=n+s+i]=1,a[i][-1]=b[i],obj[p[i]]=geq[i]?-M:0;

  //z=f(x) == z-f(x)=0
  for(auto &i:obj)i=-i;
```

```
  obj[-2]=1;
  a.emplb(obj);

  for(int i=0;i<m;i++)
    elim(i,m,p[i]);

  //now shape of a = (m+1)*(n+s+m+2)
  while(true){
    int ev=0,lvi=-1;
    for(int i=0;i<n+s+m;i++)
      ev=a[-1][ev]>a[-1][i]?i:ev;
    if(a[-1][ev]>-eps)break;
    for(int i=0;i<m;i++)
      if(a[i][ev]>eps and (!~lvi or a[i][-1]/a[i][ev]<a[lvi][-1]/a[lvi][ev]))
        lvi=i;
    if(!~lvi) throw "unbounded";
    for(int i=0;i<m+1;i++)elim(lvi,i,ev);
    p[lvi]=ev;
  }
  //if(?) throw "infeasible"
  Arr<T> ans(n+s+m+2);
  for(int i=0;i<m;i++)
    ans[p[i]]=a[i][-1];
  Arr<T> dual(m);
  for(int i=0;i<m;i++)
    dual[i]=a[-1][n+s+i]+(geq[i]?+M:0);
  return {a[-1][-1],ans,dual};
}
```

Simplex Example
Maximize $p = 6x + 14y + 13z$
Constraints
- $0.5x + 2y + z \leq 24$
- $x + 2y + 4z \leq 60$

Coding
- $n = 2, m = 3, a = \begin{pmatrix} 0.5 & 2 & 1 \\ 1 & 2 & 4 \end{pmatrix}, b = \begin{pmatrix} 24 \\ 60 \end{pmatrix}, c = [6, 14, 13]$

LP Duality & Example
tableu를 대각선으로 뒤집고 음수 부호를 붙인 답 = -(원 문제의 답)
- Primal : $n = 2, m = 3, a = \begin{pmatrix} 0.5 & 2 & 1 \\ 1 & 2 & 4 \end{pmatrix}, b = \begin{pmatrix} 24 \\ 60 \end{pmatrix}, c = [6, 14, 13]$
- Dual : $n = 3, m = 2, a = \begin{pmatrix} -0.5 & -1 \\ -2 & -2 \\ -1 & -4 \end{pmatrix}, b = \begin{pmatrix} -6 \\ -14 \\ -13 \end{pmatrix}, c = [-24, -60]$

공식
- Primal : $\max_x c^T x$, Constraints $Ax \leq b, x \geq 0$
- Dual : $\min_y b^T y$,Constraints $A^T y \geq c, y \geq 0$

4.12 FFT, NTT, Polynomial, Fast Kitamasa

```
template<int M>
struct MINT{
  int v;
  MINT() : v(0) {}
  MINT(ll val){
    v = (-M <= val && val < M) ? val : val % M;
    if(v < 0) v += M;
  }
  // @TODO : pw, operator >> << == != + - * /
  friend MINT pw(MINT a, ll b){
    MINT ret= 1;
    while(b){
      if(b & 1) ret *= a;
      b >>= 1; a *= a;
    }
    return ret;
  }
  friend MINT inv(const MINT a) { return pw(a, M-2); }
};
namespace fft{
  using real_t = double; using cpx = complex<real_t>;
  void FFT(vector<cpx> &a, bool inv_fft = false){
    int N = a.size();
    vector<cpx> root(N/2);
    for(int i=1, j=0; i<N; i++){
      int bit = (N >> 1);
      while(j >= bit) j -= bit, bit >>= 1;
      j += bit;
      if(i < j) swap(a[i], a[j]);
    }
    real_t ang = 2 * acos(-1) / N * (inv_fft ? -1 : 1);
    for(int i=0; i<N/2; i++) root[i] = cpx(cos(ang * i), sin(ang * i));
    /*
    XOR Convolution : set roots[*] = 1.
    OR Convolution : set roots[*] = 1, and do following:
    if (!inv) a[j + k] = u + v, a[j + k + i/2] = u;
    else a[j + k] = v, a[j + k + i/2] = u - v;
    */
    for(int i=2; i<=N; i<<=1){
      int step = N / i;
      for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
        cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
        a[j+k] = u+v; a[j+k+i/2] = u-v;
      }
    }
    if(inv_fft) for(int i=0; i<N; i++) a[i] /= N; // skip for OR convolution.
  }
  vector<ll> multiply(const vector<ll> &_a, const vector<ll> &_b){
    vector<cpx> a(all(_a)), b(all(_b));
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    a.resize(N); b.resize(N);
    FFT(a); FFT(b);
    for(int i=0; i<N; i++) a[i] *= b[i];
    FFT(a, 1);
    vector<ll> ret(N);
    for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
    return ret;
  }
  vector<ll> multiply_mod(const vector<ll> &a, const vector<ll> &b, const ull mod){
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    vector<cpx> v1(N), v2(N), r1(N), r2(N);
    for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i] >> 15, a[i] & 32767);
```

```
  for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i] >> 15, b[i] & 32767);
  FFT(v1); FFT(v2);
  for(int i=0; i<N; i++){
    int j = i ? N-i : i;
    cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
    cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
    cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
    cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
    r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
    r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
  }
  FFT(r1, true); FFT(r2, true);
  vector<ll> ret(N);
  for(int i=0; i<N; i++){
    ll av = llround(r1[i].real()) % mod;
    ll bv = ( llround(r1[i].imag()) + llround(r2[i].real()) ) % mod;
    ll cv = llround(r2[i].imag()) % mod;
    ret[i] = (av << 30) + (bv << 15) + cv;
    ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
  }
  return ret;
}
// 104,857,601   =  25 * 2^22 + 1, w = 3
// 998,244,353   = 119 * 2^23 + 1, w = 3
// 2,281,701,377 =  17 * 2^27 + 1, w = 3
// 2,483,027,969 =  37 * 2^26 + 1, w = 3
// 2,113,929,217 =  63 * 2^25 + 1, w = 5
// 1,092,616,193 = 521 * 2^21 + 1, w = 3
template<int W, int M>
static void NTT(vector<MINT<M>> &f, bool inv_fft = false){
  using T = MINT<M>;
  int N = f.size();
  vector<T> root(N >> 1);
  for(int i=1, j=0; i<N; i++){
    int bit = N >> 1;
    while(j >= bit) j -= bit, bit >>= 1;
    j += bit;
    if(i < j) swap(f[i], f[j]);
  }
  T ang = pw(T(W), (M-1)/N); if(inv_fft) ang = inv(ang);
  root[0] = 1; for(int i=1; i<N>>1; i++) root[i] = root[i-1] * ang;
  for(int i=2; i<=N; i<<=1){
    int step = N / i;
    for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
      T u = f[j+k], v = f[j+k+(i>>1)] * root[k*step];
      f[j+k] = u + v; f[j+k+(i>>1)] = u - v;
    }
  }
  if(inv_fft){
    T rev = inv(T(N));
    for(int i=0; i<N; i++) f[i] *= rev;
  }
}
template<int W, int M>
vector<MINT<M>> multiply_ntt(vector<MINT<M>> a, vector<MINT<M>> b){
  int N = 2; while(N < a.size() + b.size()) N <<= 1;
    a.resize(N); b.resize(N);
    NTT<W, M>(a); NTT<W, M>(b);
    for(int i=0; i<N; i++) a[i] *= b[i];
    NTT<W, M>(a, true);
    return a;
  }
}
template<int W, int M>
struct PolyMod{
  using T = MINT<M>;
  vector<T> a;
  PolyMod(){}
  PolyMod(T a0) : a(1, a0) { normalize(); }
  PolyMod(const vector<T> a) : a(a) { normalize(); }
  int size() const { return a.size(); }
  int deg() const { return a.size() - 1; }
  void normalize(){ while(a.size() && a.back() == T(0)) a.pop_back(); }
  T operator [] (int idx) const { return a[idx]; }
  typename vector<T>::const_iterator begin() const { return a.begin(); }
  typename vector<T>::const_iterator end() const { return a.end(); }
  void push_back(const T val) { a.push_back(val); }
  void pop_back() { a.pop_back(); }
  T evaluate(T x) const {
    T ret = T(0);
    for(int i=deg(); i>=0; i--) ret = ret * x + a[i];
    return ret;
  }
  PolyMod reversed() const {
    vector<T> b = a;
    reverse(b.begin(), b.end());
    return b;
  }
  PolyMod trim(int n) const {
    return vector<T>(a.begin(), a.begin() + min(n, size()));
  }
  // @TODO : operator + - *(with scala) /(with scala)
  PolyMod inv(int n){
    PolyMod q(T(1) / a[0]);
    for(int i=1; i<n; i<<=1){
      PolyMod p = PolyMod(2) - q * trim(i * 2);
      q = (p * q).trim(i * 2);
    }
    return q.trim(n);
  }
  PolyMod operator *= (const PolyMod &b){
    *this = fft::multiply_ntt<W, M>(a, b.a);
    normalize(); return *this;
  }
  PolyMod operator /= (const PolyMod &b){
    if(deg() < b.deg()) return *this = PolyMod();
    int sz = deg() - b.deg() + 1;
    PolyMod ra = reversed().trim(sz), rb = b.reversed().trim(sz).inv(sz);
    *this = (ra * rb).trim(sz);
    for(int i=sz-size(); i; i--) push_back(T(0));
    reverse(all(a)); normalize();
    return *this;
```

```
  }
  PolyMod operator %= (const PolyMod &b){
    if(deg() < b.deg()) return *this;
    PolyMod tmp = *this; tmp /= b; tmp *= b;
    *this -= tmp; normalize();
    return *this;
  }
  PolyMod operator * (const PolyMod &b) const { return PolyMod(*this) *= b; }
  PolyMod operator / (const PolyMod &b) const { return PolyMod(*this) /= b; }
  PolyMod operator % (const PolyMod &b) const { return PolyMod(*this) %= b; }
};
using mint = MINT<998244353>;
using poly = PolyMod<3, 998244353>;
mint Kitamasa(poly c, poly a, ll n){
  poly d = vector<mint>{1};
  poly xn = vector<mint>{0, 1};
  poly f;
  for(int i=0; i<c.size(); i++) f.push_back(-c[i]);
  f.push_back(1);
  while(n){
    if(n & 1) d = d * xn % f;
    n >>= 1; xn = xn * xn % f;
  }
  mint ret = 0;
  for(int i=0; i<=a.deg(); i++) ret += a[i] * d[i];
  return ret;
}
```

5 String

5.1 KMP, Hash, Manacher, Z

```
vector<int> getFail(const container &pat){
    vector<int> fail(pat.size());
    // match: pat[0..j] and pat[j-i..i] is equivalent
    // ins/del: manipulate corresponding range to pattern starts at 0
    //      (insert/delete pat[i], manage pat[j-i..i])
    function<bool(int, int)> match = [&](int i, int j){ };
    function<void(int)> ins = [&](int i){ };
    function<void(int)> del = [&](int i){ };
    for(int i=1, j=0; i<pat.size(); i++){
        while(j && !match(i, j)){
            for(int s=i-j; s<i-fail[j-1]; s++) del(s);
            j = fail[j-1];
        }
        if(match(i, j)) ins(i), fail[i] = ++j;
    }
    return fail;
}
vector<int> doKMP(const container &str, const container &pat){
    vector<int> ret, fail = getFail(pat);
    // match: pat[0..j] and str[j-i..i] is equivalent
    // ins/del: manipulate corresponding range to pattern starts at 0
    //      (insert/delete str[i], manage str[j-i..i])
    function<bool(int, int)> match = [&](int i, int j){ };
    function<void(int)> ins = [&](int i){ };
    function<void(int)> del = [&](int i){ };
    for(int i=0, j=0; i<str.size(); i++){
        while(j && !match(i, j)){
            for(int s=i-j; s<i-fail[j-1]; s++) del(s);
            j = fail[j-1];
        }
        if(match(i, j)){
            if(j+1 == pat.size()){
                ret.push_back(i-j);
                for(int s=i-j; s<i-fail[j]+1; s++) del(s);
                j = fail[j];
            }
            else ++j;
            ins(i);
        }
    }
    return ret;
}
// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<ll P, ll M> struct Hashing {
    vector<ll> H, B;
    void Build(const string &S){
        H.resize(S.size()+1);
        B.resize(S.size()+1);
        B[0] = 1;
        for(int i=1; i<=S.size(); i++) H[i] = (H[i-1] * P + S[i-1]) % M;
        for(int i=1; i<=S.size(); i++) B[i] = B[i-1] * P % M;
    }
    ll sub(int s, int e){
        ll res = (H[e] - H[s-1] * B[e-s+1]) % M;
        return res < 0 ? res + M : res;
    }
};
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1;
    vector<int> ret(n);
    string s = "#";
    for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n && s[i-ret[i]-1] == s[i+ret[i]+1]) ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    }
    return ret;
}
// input: manacher array, 1-based hashing structure
// output: set of pair(hash_val, length)
set<pair<hash_t,int>> UniquePalindrome(const vector<int> &dp, const Hashing &hashing){
    set<pair<hash_t,int>> st;
    for(int i=0,s,e; i<dp.size(); i++){
        if(!dp[i]) continue;
        if(i & 1) s = i/2 - dp[i]/2 + 1, e = i/2 + dp[i]/2 + 1;
        else s = (i-1)/2 - dp[i]/2 + 2, e = (i+1)/2 + dp[i]/2;
```

```
        for(int l=s, r=e; l<=r; l++, r--){
            auto now = hashing.get(l, r);
            auto [iter,flag] = st.emplace(now, r-l+1);
            if(!flag) break;
        }
    }
    return st;
}
//z[i]=match length of s[0,n-1] and s[i,n-1]
vector<int> Z(const string &s){
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++){
        if(i < r) z[i] = min(r-i-1, z[i-l]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    }
    return z;
}
```

5.2 Aho-Corasick

```
struct Node{
  map<char, Node*> ch; int terminal;
  Node() : terminal(-1) {}
  ~Node(){
    for(auto &i : ch) delete i.second;
    ch.clear();
  }
  void insert(const char *key, int num){
    if(*key == 0){ terminal = num; return; }
    if(!ch[*key]) ch[*key] = new Node();
    ch[*key]->insert(key+1, num);
  }
  Node *fail; vector<int> out;
};
void aho_getFail(Node *root){
  queue<Node*> q; q.push(root);
  root->fail = root;
  while(q.size()){
    Node *now = q.front(); q.pop();
    for(auto &i : now->ch){
      Node *ch = i.second;
      if(!ch) continue;
      if(root == now) ch->fail = root;
      else{
        Node *t = now->fail;
        while(t != root && !t->ch[i.first]) t = t->fail;
        if(t->ch[i.first]) t = t->ch[i.first];
        ch->fail = t;
      }
      ch->out = ch->fail->out;
      if(ch->terminal != -1) ch->out.push_back(ch->terminal);
      q.push(ch);
    }
  }
}
vector<p> aho_find(const string &s, Node *root){
  vector<p> ret; auto state = root;
  for(int i=0; i<s.size(); i++){
    while(state != root && !state->ch[s[i]]) state = state->fail;
    if(state->ch[s[i]]) state = state->ch[s[i]];
    for(int j=0; j<state->out.size(); j++){
      ret.emplace_back(i, state->out[j]);
    }
  }
  return ret;
}
```

5.3 $O(N \log N)$ SA + LCP

```
// O(N \log N) + O(N)
// 서로 다른 부분 문자열의 개수 : n(n+1)/2 - sum(lcp)
// LCS : A+#+B, then do
/* int result = 0, pos = 0, B = N - A;
   for(int i=0; i<N-1; i++) if((sa[i] >= A) != (sa[i+1] >= A)){
   int t = min(lcp[i], A-1 - min(sa[i], sa[i+1]));
   if(t > res) res = t, pos = sa[i]; } */
int sa[1010101], lcp[1010101], pos[1010101];
void getSA(const string &s){
  int n = s.size(), m = 500;
  vector<int> cnt(max(n, m)+1), x(n+1), y(n+1);
  for(int i=1; i<=n; i++) cnt[x[i]=s[i-1]]++;
  for(int i=1; i<=m; i++) cnt[i] += cnt[i-1];
  for(int i=n; i; i--) sa[cnt[x[i]]--] = i;
  for(int len=1, pv=0, i; pv<n; len<<=1, m=pv){
    for(pv=0, i=n-len+1; i<=n; i++) y[++pv] = i;
    for(i=1; i<=n; i++) if(sa[i] > len) y[++pv] = sa[i] - len;
    for(i=0; i<=m; i++) cnt[i] = 0;
    for(i=1; i<=n; i++) cnt[x[y[i]]]++;
    for(i=1; i<=m; i++) cnt[i] += cnt[i-1];
    for(i=n; i>=1; i--) sa[cnt[x[y[i]]]--] = y[i];
    swap(x, y); pv = 1; x[sa[1]] = 1;
    for(i=1; i<n; i++){
      int a = sa[i], b = sa[i+1], a_len = a + len, b_len = b + len;
      if(a_len <= n && b_len <= n && y[a] == y[b] && y[a_len] == y[b_len]) x[sa[i+1]] = pv;
      else x[sa[i+1]] = ++pv;
    }
  }
  for(int i=0; i<n; i++) sa[i] = sa[i+1]-1, pos[sa[i]] = i;
}
void getLCP(const string &s){
  int n = s.size();
  for(int i=0,k=0; i<n; i++, k=max(k-1, 0)){
    if(pos[i] == n-1) continue;
    for(int j=sa[pos[i]+1]; s[i+k]==s[j+k]; k++);
    lcp[pos[i]] = k;
  }
}
```

5.4 Bitset LCS

```
#include <x86intrin.h>
template<size_t _Nw> void _M_do_sub(_Base_bitset<_Nw> &A, const _Base_bitset<_Nw> &B){
  for(int i=0, c=0; i<_Nw; i++) c = _subborrow_u64(c, A._M_w[i], B._M_w[i], (ull*)&A._M_w[i]);
}
void _M_do_sub(_Base_bitset<1> &A, const _Base_bitset<1> &B){ A._M_w -= B._M_w; }
template<size_t _Nb> bitset<_Nb>& operator-=(bitset<_Nb> &A, const bitset<_Nb> &B){
  _M_do_sub(A, B); return A;
}
template<size_t _Nb> inline bitset<_Nb> operator-(const bitset<_Nb> &A, const bitset<_Nb> &B){
  bitset<_Nb> C(A); return C -= B;
}
char s[50050], t[50050];
int lcs(){ // O(NM/64)
  bitset<50050> dp, ch[26];
  int n = strlen(s), m = strlen(t);
  for(int i=0; i<m; i++) ch[t[i]-'A'].set(i);
  for(int i=0; i<n; i++){ auto x = dp | ch[s[i]-'A']; dp = dp - (dp ^ x) & x; }
  return dp.count();
}
```

5.5 Lyndon Factorization, Minimum Rotation

```
// factorize string into w1 >= w2 >= ... >= wk, wi is smallest cyclic shift of suffix.
vector<string> Lyndon(const string &s){ // O(N)
  int n = s.size(), i = 0, j, k;
  vector<string> res;
  while(i < n){
    for(j=i+1, k=i; i<n && s[k]<=s[j]; j++) k = s[k] < s[j] ? i : k + 1;
    for(; i<=k; i+=j-k) res.push_back(s.substr(i, j-k));
  }
  return res;
}
// rotate(v.begin(), v.begin()+min_rotation(v), v.end());
template<typename T> int min_rotation(T s){ // O(N)
  int a = 0, N = s.size();
  for(int i=0; i<N; i++) s.push_back(s[i]);
  for(int b=0; b<N; b++) for(int k=0; k<N; k++){
    if(a+k == b || s[a+k] < s[b+k]){ b += max(0, k-1); break; }
    if(s[a+k] > s[b+k]){ a = b; break; }
  }
  return a;
}
```

6 Misc

6.1 Ternary Search

```
// get minimum / when multiple answer, find minimum `s`
while(s + 3 <= e){
  T l = (s + s + e) / 3, r = (s + e + e) / 3;
  if(Check(l) > Check(r)) s = l;
  else e = r;
}
T mn = INF, idx = s;
for(T i=s; i<=e; i++){
  T now = Check(i);
  if(now < mn) mn = now, idx = i;
}
```

6.2 Aliens Trick

```
// 점화식에 min이 들어가는 경우: 구간을 쪼갤 때마다 +lambda
while(l <= r){
  ll m = l + r >> 1;
  [dp,cnt] = Solve(m);
  res = max(res, dp - k*m);
  if(cnt <= k) r = m - 1;
  else l = m + 1;
}
// 점화식에 max가 들어가는 경우: 구간을 쪼갤 때마다 +lambda
while(l <= r){
  ll m = l + r >> 1;
  [dp,cnt] = Solve(m);
  res = min(res, dp - k*m);
  if(cnt <= k) l = m + 1;
  else r = m - 1;
}
```

6.3 Slope Trick

```
//NOTE: f(x)=min{f(x+i),i<a}+|x-k|+m -> pf(k)sf(k)ab(-a,m)
//NOTE: sf_inc에 답구하는게 들어있어서, 반드시 한 연산에 대해 pf_dec->sf_inc순서로 호출
struct LeftHull{
  void pf_dec(int x){pq.empl(x-bias);}//x이하의 기울기들 -1
  int sf_inc(int x){//x이상의 기울기들 +1, pop된 원소 반환(Right Hull관리에 사용됨)
    if(pq.empty() or argmin()<=x)return x;
    ans+=argmin()-x;//이 경우 최솟값이 증가함
    pq.empl(x-bias);//x 이하 -1
    int r=argmin();pq.pop();//전체 +1
    return r;
  }
  void add_bias(int x,int y){bias+=x;ans+=y;}//그래프 x축 평행이동
  int minval(){return ans;}//최소값
  int argmin(){return pq.empty()?-inf<int>():pq.top()+bias;}//최소값 x좌표
  void operator+=(LeftHull& a){
    ans+=a.ans;
    while(sz(a.pq))pf_dec(a.argmin()), a.pq.pop();
  }
  int size()const{return sz(pq);}
// private:
  PQMax<int> pq;
  int ans=0,bias=0;
};
//NOTE: f(x)=min{f(x+i),a<i<b}+|x-k|+m -> pf(k)sf(k)ab(-a,b,m)
struct SlopeTrick{
  void pf_dec(int x){l.pf_dec(-r.sf_inc(-x));}
  void sf_inc(int x){r.pf_dec(-l.sf_inc(x));}
  void add_bias(int lx,int rx,int y){l.add_bias(lx,0),r.add_bias(-rx,0),ans+=y;}
  int minval(){return ans+l.minval()+r.minval();}
  pint argmin(){return {l.argmin(),-r.argmin()};}
```

```
  void operator+=(SlopeTrick& a){
    while(sz(a.l.pq)) pf_dec(a.l.argmin()),a.l.pq.pop();
    l.ans+=a.l.ans;
    while(sz(a.r.pq)) sf_inc(-a.r.argmin()),a.r.pq.pop();
    r.ans+=a.r.ans;
    ans+=a.ans;
  }
  int size()const{return l.size()+r.size();}
// private:
  LeftHull l,r;
  int ans=0;
};
//LeftHull 역추적 방법: 스텝i의 argmin값을 am(i)라고 하자. 스텝n부터 스텝1까지
ans[i]=min(ans[i+1],am(i))하면 된다. 아래는 증명..은 아니고 간략한 이유
//am(i)<=ans[i+1]일때: ans[i]=am(i)
//x[i]>ans[i+1]일때: ans[i]=ans[i+1] 왜냐하면 f(i,a)는 a<x[i]에서 감소함수이므로 가능한 최대로
오른쪽으로 붙은 ans[i+1]이 최적.
//스텝i에서 add_bias(k,0)한다면 간격제한k가 있는것이므로 ans[i]=min(ans[i+1]-k,x[i])으로 수정.
//LR Hull 역추적은 케이스나눠서 위 방법을 확장하면 될듯
```

6.4 Random, PBDS, Bit Trick

```
mt19937 rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> rnd_int(l, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1); // rnd_real(rd)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>
using namespace __gnu_pbds; //ordered_set : find_by_order(order), order_of_key(key)
using namespace __gnu_cxx; //crope : append(str), substr(s, e), at(idx)
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
int __builtin_clz(int x);// number of leading zero
int __builtin_ctz(int x);// number of trailing zero
int __builtin_popcount(int x);// number of 1-bits in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_clzll(n | 1);
long long next_perm(long long v){
  long long t = v | (v-1);
  return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
int main2(){ return 0; }
int main(){
  size_t  sz = 1<<29;  // 512MB
  void* newstack = malloc(sz);
  void* sp_dest = newstack + sz - sizeof(void*);
  asm  __volatile__("movq %0, %%rax\n\t"
          "movq %%rsp , (%%rax)\n\t"
          "movq %0, %%rsp\n\t": : "r"(sp_dest): );
  main2();
  asm  __volatile__("pop %rsp\n\t");
  return  0;
}
```

6.5 Fast I/O, Fast Div/Mod, Hilbert Mo's

```
static char buf[1 << 19]; // size : any number geq than 1024
static int idx = 0, bytes = 0;
static inline int _read() {
  if (!bytes || idx == bytes) {
    bytes = (int)fread(buf, sizeof(buf[0]), sizeof(buf), stdin);
    idx = 0;
  }
  return buf[idx++];
}
static inline int readInt() {
  int x = 0, s = 1, c = _read();
  while (c <= 32) c = _read();
  if (c == '-') s = -1, c = _read();
  while (c > 32) x = 10 * x + (c - '0'), c = _read();
  if (s < 0) x = -x; return x;
}
typedef __uint128_t L;
struct FastMod{
  ull b, m;
  FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}
  ull reduce(ull a){
    ull q = (ull)((L(m) * a) >> 64), r = a - q * b; // can be proven that 0 <= r < 2*b
    return r >= b ? r - b : r;
  }
};
inline int64_t hilbertOrder(int x, int y, int pow, int rotate) {
  if(pow == 0) return 0;
  int hpow = 1 << (pow-1);
  int seg = (x<hpow) ? ( (y<hpow) ? 0 : 3 ) : ( (y<hpow) ? 1 : 2 );
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  int64_t subSquareSize = int64_t(1) << (2*pow - 2);
  int64_t ans = seg * subSquareSize;
  int64_t add = hilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
  return ans;
}
struct Query{
  int s, e, x; ll order;
  void init(){ order = hilbertOrder(s, e, 21, 0); }
  bool operator < (const Query &t) const { return order < t.order; }
};
```

6.6 DP Opt, Tree Opt, Well-Known Ideas

```
// Quadrangle Inequality : C(a, c)+C(b, d) ≤ C(a, d)+C(b, c)
// Monotonicity : C(b, c) ≤ C(a, d)
// CHT, DnC Opt(Quadrangle), Knuth(Quadrangle and Monotonicity)

// 크기가 A, B인 두 서브트리의 결과를 합칠 때 O(AB)이면 O(N^3)이 아니라 O(N^2)
// 각 정점마다 sum(2 ~ C번째로 높이가 작은 정점의 높이)에 결과를 구할 수 있으면 O(N^2)이 아니라 O(N)
```

```
// IOI 16 Alien(Lagrange Multiplier), IOI 11 Elephant(sqrt batch process)
// IOI 09 Region
// 서로소 합집합의 크기가 적당히 bound 되어 있을 때 사용
// 쿼리 메모이제이션 / 쿼리 하나에 O(A log B), 전체 O(N√Q log N)
```

6.7 Catalan, Burnside, Grundy, Pick, Hall, Simpson, Kirchhoff

- 카탈란 수
  1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,742900
  $C_n = binomial(n*2, n)/(n+1)$;
  - 길이가 2n인 올바른 괄호 수식의 수
  - n + 1개의 리프를 가진 풀 바이너리 트리의 수
  - n + 2각형을 n개의 삼각형으로 나누는 방법의 수
- Burnside's Lemma
  - 수식
    G=(X,A): 집합X와 액션A로 정의되는 군G에 대해, $|A||X/A| = sum(|$Fixed points of a$|$, for all a in A)
    X/A 는 Action으로 서로 변형가능한 X의 원소들을 동치로 묶었을때 동치류(파티션) 집합
  - 풀어쓰기
    orbit: 그룹에 대해 두 원소 a,b와 액션f에 대해 f(a)=b인거에 간선연결한 컴포넌트(연결집합)
    orbit개수 = sum(각 액션 g에 대해 f(x)=x인 x(고정점)개수)/액션개수
  - 자유도 치트시트 회전 n개: 회전i의 고정점 자유도=gcd(n,i)
    임의뒤집기 n=홀수: n개 원소중심축(자유도 (n+1)/2)
    임의뒤집기 n=짝수: n/2개 원소중심축(자유도 n/2+1) + n/2개 원소안지나는축(자유도 n/2)
- 알고리즘 게임
  - Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.
  - Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.
  - Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 k + 1로 나눈 나머지를 XOR 합하여 판단한다.
  - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.
- Pick's Theorem
  격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$
- 홀의 결혼 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S의 크기) <= (S와 연결되어있는 모든 R의 크기)이다.
- Simpson 공식 (적분) : Simpson 공식, $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{2i+1}) + 2\sum f(x_{2i})]$
  - $M = \max|f^4(x)|$이라고 하면 오차 범위는 최대 $E_n \le \frac{M(b-a)}{180}h^4$
- Kirchhoff's Theorem : 그래프의 스패닝 트리 개수
  - m[i][j] := -(i-j 간선 개수) (i ≠ j)
  - m[i][i] := 정점 i의 degree
  - res = (m의 첫 번째 행과 첫 번째 열을 없앤 (n-1) by (n-1) matrix의 행렬식)
- Tutte Matrix : 그래프의 최대 매칭
  - m[i][j] := 간선 (i, j)가 없으면 0, 있으면 $i < j ? r : -r$, r은 $[0, P)$ 구간의 임의의 정수
  - $rank(m)/2$가 높은 확률로 최대 매칭

6.8 About Graph Matching(Graph with $|V| \le 500$)

- **Game on a Graph** : s에 토큰이 있음. 플레이어는 각자의 턴마다 토큰을 인접한 정점으로 옮기고 못 옮기면 짐. s를 포함하지 않는 최대 매칭이 존재함 ↔ 후공이 이김
- **Chinese Postman Problem** : 모든 간선을 방문하는 최소 가중치 Walk를 구하는 문제.
  Floyd를 돌린 다음, 홀수 정점들을 모아서 최소 가중치 매칭 (홀수 정점은 짝수 개 존재)
- **Unweighted Edge Cover** : 모든 정점을 덮는 가장 작은(minimum cardinality/weight) 간선 집합을 구하는 문제
  $|V| - |M|$, 길이 3짜리 경로 없음, star graph 여러 개로 구성
- **Weighted Edge Cover** : $sum_{v \in V}(w(v)) - sum_{(u,v) \in M}(w(u) + w(v) - d(u, v))$, $w(x)$는 x와 인접한 간선의 최소 가중치
- **NEERC'18 B** : 각 기계마다 2명의 노동자가 다뤄야 하는 문제.
  기계마다 두 개의 정점을 만들고 간선으로 연결하면 정답은 $|M| - |$기계$|$임. 정답에 1/2씩 기여한다는 점을 생각해 보면 좋음.
- **Min Disjoint Cycle Cover** : 정점이 중복되지 않으면서 모든 정점을 덮는 길이 3 이상의 사이클 집합을 찾는 문제.
  모든 정점은 2개의 서로 다른 간선, 일부 간선은 양쪽 끝점과 매칭되어야 하므로 플로우를 생각할 수 있지만 용량 2짜리 간선에 유량을 1만큼 흘릴 수 있으므로 플로우는 불가능.
  각 정점과 간선을 2개씩((v, v'), $(e_{i,u}, e_{i,v})$)로 복사하자. 모든 간선 $e = (u, v)$에 대해 e_u와 e_v를 잇는 가중치 w짜리 간선을 만들고(like NEERC18), $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$를 연결하는 가중치 0짜리 간선을 만들자.
  Perfect 매칭이 존재함 ↔ Disjoint Cycle Cover 존재. 최대 가중치 매칭 찾은 뒤 모든 간선 가중치 합에서 매칭 빼면 됨.
- **Two Matching** : 각 정점이 최대 2개의 간선과 인접할 수 있는 최대 가중치 매칭 문제.
  각 컴포넌트는 정점 하나/경로/사이클이 되어야 함. 모든 서로 다른 정점 쌍에 대해 가중치 0짜리 간선 만들고, 가중치 0짜리 (v, v') 간선 만들면 Disjoing Cycle Cover 문제가 됨. 정점 하나만 있는 컴포넌트는 self-loop, 경로 형태의 컴포넌트는 양쪽 끝점을 연결한다고 생각하면 편함.

6.9 Checklist

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 구간을 통째로 가져간다 : 플로우 + 적당한 자료구조 $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- a = b : a만 움직이기, b만 움직이기, 두 개 동시에 움직이기, 반대로 움직이기
- 말도 안 되는 것들을 한 번은 생각해보기 / "당연하다고 생각한 것" 다시 생각해보기
- Directed MST / Dominator Tree
- 일정 비율 충족 or 2 3개로 모두 커버 : 랜덤
- 확률 : DP, 이분 탐색(NYPC 2019 Finals C)
- 최대/최소 : 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)