palao - ICPC Library

Contents

1 Data Structures

1.1 Fenwick

```cpp
const int mx = 2e5+5;
ll bit[mx];
int n, q;

ll qry(int i){ // [1,i] 1 indexado
        ll ret = 0;
        for(; i > 0; i -= i & -i)
                ret += bit[i];
        return ret;
}

void increment(ll i, ll v){ // 1 indexado (+= v)
        for(; i <= n; i += i & -i)
                bit[i] += v;
}
```

1.2 Segtree Lazy

```cpp
// Lazy SegTree
const int MX = 2e5+5;
vector<ll> seg(4*mx);
vector<ll> lazy(4*mx,0);
vector<ll> nums(mx);
int n,q;

void build(int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = nums[l];
                lazy[idx] = 0;
                return;
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        build(l,m,left);
        build(m+1,r,right);
        seg[idx] = seg[left] + seg[right];
}

void prop(int l = 0, int r = n-1, int idx = 0){
        seg[idx] += (ll)(r-l+1)*lazy[idx];
        if(l != r){ // nao for folha
                int left = 2*idx+1;
                int right = 2*idx+2;
                lazy[left] += lazy[idx];
                lazy[right] += lazy[idx];
        }
        lazy[idx] = 0;
}

void update(int L, int R, ll val, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return;
        prop(l,r,idx);
        if(L <= l && r <= R){
                lazy[idx] = val;
                prop(l,r,idx);
        }
        else{
                int m = (l+r)/2;
                int left = 2*idx+1;
                int right = 2*idx+2;
                update(L,R,val,l,m,left);
                update(L,R,val,m+1,r,right);
                seg[idx] = seg[left] + seg[right];
        }
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        prop(l,r,idx);
        if(R < l || L > r) return 0;
        if(L <= l && r <= R){
                return seg[idx];
        }
        int m = (l+r)/2;
        int left = 2*idx+1;
        int right = 2*idx+2;
        return query(L,R,l,m,left) + query(L,R,m+1,r,right);
}
```

1.3 Segtree Topdown

```
// SegTree
const int mx = 2e5 + 5;
ll seg[4*mx];
ll nums[mx];
int n,q;
ll merge(ll a, ll b){
        return a+b;
}

void build(int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = nums[l];
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        build(l,mid,left);
        build(mid+1,r,right);
        seg[idx] = merge(seg[left], seg[right]);
}

ll query(int L, int R, int l = 0, int r = n-1, int idx = 0){
        if(R < l || L > r) return 0; // elemento neutro
        if(L <= l && r <= R) return seg[idx];

        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        ll ql = query(L,R,l,mid,left);
        ll qr = query(L,R,mid+1,r,right);
        return merge(ql,qr);
}

void update(int pos, int num, int l = 0, int r = n-1, int idx = 0){
        if(l == r){
                seg[idx] = num;
                return;
        }
        int mid = l + (r-l)/2;
        int left = 2*idx + 1;
        int right = 2*idx + 2;
        if(pos <= mid){
                update(pos,num,l,mid,left);
        }
        else update(pos,num,mid+1,r,right);
        seg[idx] = merge(seg[left],seg[right]);
}
```

1.4 SparseTable

```
vector<vector<ll>> table;
vector<ll> lg2;
void build(int n, vector<ll> v) {
  lg2.resize(n + 1);
  lg2[1] = 0;
  for (int i = 2; i <= n; i++) {
    lg2[i] = lg2[i >> 1] + 1;
  }
  table.resize(lg2[n] + 1);
  for (int i = 0; i < lg2[n] + 1; i++) {
    table[i].resize(n + 1);
  }
  for (int i = 0; i < n; i++) {
    table[0][i] = v[i];
  }
  for (int i = 0; i < lg2[n]; i++) {    for (int j = 0; j < n; j++) {
      if (j + (1 << i) >= n) break;
      table[i + 1][j] = min(table[i][j], table[i][j + (1 << i)]);
    }
  }
}
ll get(int l, int r) { // (l,r) inclusivo
  int k = lg2[r - l + 1];
  return min(table[k][l], table[k][r - (1 << k) + 1]);
}
```

2 DP

2.1 Knapsack

```
// Knapsack
const int MXW = 1e5+5;
const int MXN = 105;

int n, max_w;
vector<int> weight(MXN), value(MXN);
vector<vector<ll>> dp(MXN,vector<ll>(MXW,-1));

ll solveDp(int i, int k){ // k -> peso atual
        if(i == n) return 0;
        if(dp[i][k] != -1) return dp[i][k];

        ll ignore = solveDp(i+1,k);
        ll add = -1;
        if(weight[i] + k <= max_w){
                add = value[i] + solveDp(i+1, weight[i] + k);
        }
        return dp[i][k] = max(ignore,add);
}

// iterativo

ll knapsack(){
  vector<ll> dp(dpmx,0);
  for(int i = 0; i < n; i++){
    ll w = weight[i];
    ll v = value[i];
    for(int sz = max_w; sz >= w; sz--){
      dp[sz] = max(dp[sz],dp[sz-w]+v);
    }
  }
  return *max_element(begin(dp),end(dp));
}
```

2.2 Longest Increasing Sequence

```
// Longest Increasing Sequence
int lis(vector<ll>& nums){
        int n = nums.size();
        vector<ll> s;
        for(int i = 0; i < n; i++){
                auto it = lower_bound(s.begin(),s.end(),nums[i]);
                if(it == s.end()){
                        s.PB(nums[i]);
                }
                else{
                        *it = nums[i];
                }
        }
```

```
        return (int)s.size();
}
```

3 Geometry

3.1 Point

```
const double inf = 1e100, eps = 1e-9;
const double PI = acos(-1.0L);
int cmp (double a, double b = 0) {
  if (abs(a-b) < eps) return 0;
  return (a < b) ? -1 : +1;
}
struct Point{
        double x,y;
        Point(double x = 0, double y = 0) : x(x),y(y){}
        Point(const Point& p): x(p.x), y(p.y){}
  bool operator < (const Point &p) const {
    if(cmp(x, p.x) != 0) return x < p.x;
    return cmp(y, p.y) < 0;
  }
  bool operator == (const Point &p) const {return !cmp(x, p.x) && !cmp(y, p
      .y);}
  bool operator != (const Point &p) const {return !(p == *this);}

        // basic ops
        Point operator + (const Point& p) const {return Point(x+p.x,y+p.y)
            ;}
        Point operator - (const Point& p) const {return Point(x-p.x,y-p.y)
            ;}
        Point operator * (const double k) const {return Point(x*k,y*k);}
        Point operator / (const double k) const {return Point(x/k,y/k);}
};
        // points ops
double dot (const Point& p,const Point& q) { return p.x*q.x + p.y*q.y; }
double cross (const Point& p,const Point& q) { return p.x*q.y - p.y*q.x; }
double norm(const Point& p) { return hypot(p.x,p.y); }
double dist(const Point& p, const Point& q) { return hypot(p.x-q.x,p.y-q.y)
    ; }
double dist2(const Point& p, const Point& q) { return dot(p-q,p-q); }
Point normalize(const Point &p) { return p/hypot(p.x, p.y); }
double angle (const Point& p, Point& q) { return atan2(cross(p, q), dot(p,
    q)); }
double angle (const Point& p) { return atan2(p.y, p.x); }

ostream &operator<<(ostream &os, const Point &p) {
        return os << "(" << p.x << "," << p.y << ")";
}
```

```
struct Line{
        Point p, vd;
        Line(){}
        Line(const Point& p, const Point& vd) : p(p), vd(vd) {};
};

double distPointLine(const Point& p, const Line& l){
        Point vp = p-l.p;
        return abs(cross(vp,l.vd))/norm(l.vd);
}

ostream &operator<<(ostream &os, const Line &l) {
        return os << "(" << l.p.x << "," << l.p.y << ")" << "+ t(" << l.vd.
            x << "," << l.vd.y << ")";
}
```

3.2 Convex Hull

```
// O(nlogn) sorted = false
// O(n) sorted = true
vector<Point> convexHull(vector<Point> points, bool sorted = false) {
  if(!sorted)  sort(begin(points), end(points));
  vector<Point> hull;
  hull.reserve(points.size() + 1);
  for (int phase = 0; phase < 2; ++phase) {
    int start = hull.size();
    for (Point& c : points) {
      while (hull.size() >= start+2){
        Point a = hull[hull.size()-2], b = hull.back();
        if(cross(b-a,c-a) > 0) break; // '>' descarta pontos colineares,
            '>=' nao, '<' sentido horario
        hull.pop_back();
      }
      reverse(begin(points), end(points));
      hull.push_back(c);
    }
    hull.pop_back();
  }
  if (hull.size() == 2 && hull[0] == hull[1]) hull.pop_back();
  return hull;
}
```

```
// pegar half-hull 0 -> n
vector<Point> halfHull(vector<Point>& pts, bool upper = 0){
  int n = pts.size();
  vector<Point> hull(n + 1);
  int s = 0;
  for(int i = 0; i < n; i++){
    hull[s++] = pts[i];
    while(s >= 3){
      Point a = hull[s-3], b = hull[s-2], c = hull[s-1];
      Point v1 = b-a, v2 = c-b;
      if((upper?-1:1)*cross(v1,v2) >= 0)  break;
      hull[s-2] = hull[s-1];
      s--;
    }
  }
  hull.resize(s);
  return hull;
}

bool isInside(const vector<Point> &hull, Point pt) {
  int n = hull.size();
  Point v0 = pt - hull[0], v1 = hull[1] - hull[0], v2 = hull[n-1] - hull
      [0];
  if(cross(v0,v1) > 0 || cross(v0,v2) < 0){
    return false;
  }
  int l = 1, r = n - 1;
  while(l != r){
    int mid = (l + r + 1) / 2;
    Point v0 = pt - hull[0], v1 = hull[mid] - hull[0];
    if(cross(v0,v1) < 0) {
      l = mid;
    } else {
      r = mid - 1;
    }
  }
  v0 = hull[(l+1)%n] - hull[l], v1 = pt - hull[l];
  return cross(v0,v1) >= 0;
}
// poligonos
ll polygon_area_db(const vector<Point>& poly){
```

```
  ll area = 0;
  for(int i = 0, n = (int)poly.size(); i < n; ++i) {
    int j = i + 1 == n ? 0 : i + 1;
    area += cross(poly[i], poly[j]);
  }
  return abs(area);
}
// Teorema de Pick para lattice points
// Area = insidePts + boundPts/2 - 1
// 2A - b + 2 = 2i
// usar gcd dos lados pra contar bound pts
ll cntInsidePts(ll area_db, ll bound){
  return (area_db + 2LL - bound)/2;
}
```

4 ETC

4.1 Bitset

```
// Bitset operations
__builtin_popcount(int x);
__builtin_popcountll(ll x);
const int SZ = 1e6;
bitset<SZ> b;
b.reset(); // 00 ... 00
b.set();   // 11 ... 11
b.flip();
b._Find_first(); // retorna SZ se nao tiver
b._Find_next(i);
b.to_ulong();
b.to_string();
b.count();

// rng
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
shuffle(begin(x),end(x),rng);
uniform_int_distribution<int>(0,x)(rng);
```

4.2 Ternary Search

```
double f(double t){
        // alguma funcao unimodal -> maximo ou minimo
        //      /\
        //     /  \
        // __/    \__
}

double tern_search(double l, double r){ // achar o maximo
        double eps = 1e-8;
        while(r - l > eps){
                double m1 = l + (r-l)/3;
                double m2 = r - (r-l)/3;
                double f1 = f(m1), f2 = f(m2);
                if(f1 < f2)      l = m1; // (m1,r)
                else r = m2; // (l,m2);
        }
        return max(f(l),f(r));
}
// retorna mais a esquerda no empate
int int_tern_search(int l, int r){
        int lo = l - 1, hi = r;
        while(hi - lo > 1){
                int m = (lo+hi)/2;
                if(f(m) >= f(m+1)){ // decrescendo
                        hi = m;
                }else{ // crescendo;
                        lo = m;
                }
        }
        return lo + 1;
}
```

4.3 Mo Algorithm

```
// Mo apelao
// Ordering based on the Hilbert curve
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;
}

const int MXN = 2e5;
ll a[MXN];
ll resp = 0;
void add(int x);
void remove(int x);

struct Query{
    int l, r, idx;
    int64_t ord;
    Query(int l, int r, int idx) : l(l), r(r), idx(idx) {
        ord = hilbertOrder(l, r, 21, 0);
    }
    bool operator < (Query &other){
        return ord < other.ord;
    }
};

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n, q; cin >> n >> q;
    for(int i = 0; i < n; i++)
        cin >> a[i];
    vector<Query> queries;
    for(int i = 0; i < q; i++){
        int l, r; cin >> l >> r;
        queries.push_back(Query(l-1,r-1,i));
    }
    sort(begin(queries),end(queries));
    vector<ll> answers(q);
    int L = 0, R = -1;
    for(Query qr : queries){
      while (L > qr.l) add(--L);
      while (R < qr.r) add(++R);
      while (L < qr.l) remove(L++);
      while (R > qr.r) remove(R--);
      answers[qr.idx] = resp;
    }
    for(int i = 0; i < q; i++)
        cout << answers[i] << "\n";
}
```

5 Graph

5.1 Dijkstra

```cpp
// Dijkstra
#define pii pair<ll,ll>
const ll MXN = 2e5+5;
const ll INF = LLONG_MAX;
int v,e;
vector<pii> adj[MXN];
vector<ll> parent(MXN,-1);
vector<ll> dist(MXN,INF);

void dijkstra(ll node){
        dist[node] = 0;
        priority_queue<pii,vector<pii>, greater<pii>> pq;
        pq.push({0,node});
        while(!pq.empty()){
                auto [d,u] = pq.top();
                pq.pop();
                if(d > dist[u]) continue;
                for(auto [cost, v] : adj[u]){
                        ll currD = dist[u] + cost;
                        if(currD < dist[v]){
                                dist[v] = currD;
                                parent[v] = out;
                                pq.push({currD,v});
                        }
                }
        }
}
```

5.2 DSU

```cpp
struct DSU{
        vector<int> p;
        vector<int> sz;
        int n;
        DSU(int nodes){
                n = nodes;
                p.resize(nodes);
                sz.resize(nodes,1);
                iota(begin(p), end(p), 0);
        }
        int size(int a){ return sz[root(a)]; }
        int root(int a){ return p[a] = (p[a] == a ? a : root(p[a])); }
        bool unite(int a, int b){
                int ra = root(a), rb = root(b);
                if(ra != rb){
                        if(sz[ra] < sz[rb]) swap(ra,rb);
                        p[rb] = ra;
                        sz[ra] += sz[rb];
                        return 1;
                }
                return 0;
        }
};
```

5.3 Floyd Warshall

```cpp
const int inf = 0x3f3f3f3f;
int g[ms][ms], dis[ms][ms], n;

void clear() {
  memset(g, 0x3f, sizeof g);
  for(int i = 0; i < n; i++) g[i][i] = 0;
}

void add(int u, int v, int w) {
  g[u][v] = min(w, g[u][v]);
}

void floydWarshall() {
  memcpy(g, dis, sizeof g);
  for(int k = 0; k < n; k++) {
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
        dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
      }
    }
  }
}
```

5.4 Kosaraju

```cpp
// Kosaraju
vector<vector<int>> G, Gt;
vector<int> id;
vector<int> order;
vector<bool> vis;
int n;

void dfs1(int v){ // ordem de saida
  vis[v] = true;
  for(int u : G[v]){
    if(!vis[u]) dfs1(u);
  }
  order.PB(v);
}
void dfs2(int v, int idx, vector<int>& component){ // pegar um componente todo
    vis[v] = true;
    id[v] = idx;
    component.PB(v);
    for(int u : Gt[v]){
        if(!vis[u]) dfs2(u);
    }
}
vector<vector<int>> kosaraju(){
  vector<vector<int>> components;
  vis.assign(n,false);
  for(int i = 0; i < n; i++){
    if(!vis[i]) dfs1(i);
  }
  vis.assign(n,false);
  reverse(begin(order),end(order));
  int idx = 0;
  for(int v : order){
    if(!vis[v]){
      vector<int> component;
      dfs2(v, idx++, component);
      // sort(begin(component),end(component));
      components.PB(component);
    }
  }
  return components;
}
```

5.5 Kruskal

```
int n = 1e5;
DSU dsu = DSU(n+5);
using tp = tuple<ll,int,int>
vector<tp> edges(e);
for(auto& [w, u, v] : edges){
        cin >> u >> v >> w;
}
sort(begin(edges),end(edges));
ll cost = 0;
int cnt = 0;
for(auto [w, u, v] : edges){
        if(dsu.unite(u,v)){
                cost += w;
                cnt++;
        }
}
// if(cnt != n-1) cout << "IMPOSSIBLE" << br;
```

5.6 Lowest Common Ancestor

```
const int mxn = 2e5+5;
const int LOG = 22;
int n, q;
int tin[mxn], tout[mxn];
vector<vector<int>> up; // up[v][k] = 2^k-esimo ancestor de v
vector<int> g[mxn];
int lvl[mxn];
int timer = 0;
void dfs(int u, int p){
    tin[u] = ++timer;
    lvl[u] = lvl[p] + 1;
    up[u][0] = p;
    for(int i = 1; i <= LOG; i++){
        up[u][i] = up[ up[u][i-1] ][i-1];
    }
    for(int v : g[u]){
        if(v != u && !tin[v])
            dfs(v,u);
    }
    tout[u] = ++timer;
}

bool is_ancestor(int u, int v){
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int a, int b){
    if(is_ancestor(a,b)) return a;
    if(is_ancestor(b,a)) return b;
    for(int i = LOG; i >= 0; i--){
        if(!is_ancestor(up[a][i], b)){
            a = up[a][i];
        }
    }
    return up[a][0];
}
```

5.7 Max Flow

```
template <class T = int>
class MCMF {
public:
  struct Edge {
    Edge(int a, T b, T c) : to(a), cap(b), cost(c) {}
    int to;
    T cap, cost;
  };
  MCMF(int size) {
    n = size;
    edges.resize(n);
    pot.assign(n, 0);
    dist.resize(n);
    visit.assign(n, false);
  }
  pair<T, T> mcmf(int src, int sink) {
    pair<T, T> ans(0, 0);
    if(!SPFA(src, sink)) return ans;
    fixPot();
    // can use dijkstra to speed up depending on the graph
    while(SPFA(src, sink)) {
      auto flow = augment(src, sink);
      ans.first += flow.first;
      ans.second += flow.first * flow.second;
      fixPot();
    }
    return ans;
  }
  void addEdge(int from, int to, T cap, T cost) {
    edges[from].push_back(list.size());
    list.push_back(Edge(to, cap, cost));
    edges[to].push_back(list.size());
    list.push_back(Edge(from, 0, -cost));
  }
private:
  int n;
  vector<vector<int>> edges;
  vector<Edge> list;
  vector<int> from;
  vector<T> dist, pot;
  vector<bool> visit;

  /*bool dij(int src, int sink) {
    T INF = std::numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    visit.assign(n, false);
    dist[src] = 0;
    for(int i = 0; i < n; i++) {
      int best = -1;
      for(int j = 0; j < n; j++) {
        if(visit[j]) continue;
        if(best == -1 || dist[best] > dist[j]) best = j;
      }
      if(dist[best] >= INF) break;
      visit[best] = true;
      for(auto e : edges[best]) {
        auto ed = list[e];
        if(ed.cap == 0) continue;
        T toDist = dist[best] + ed.cost + pot[best] - pot[ed.to];
        assert(toDist >= dist[best]);
        if(toDist < dist[ed.to]) {
          dist[ed.to] = toDist;
          from[ed.to] = e;
        }
      }
    }
    return dist[sink] < INF;
  }*/

  pair<T, T> augment(int src, int sink) {
    pair<T, T> flow = {list[from[sink]].cap, 0};
    for(int v = sink; v != src; v = list[from[v]^1].to) {
      flow.first = min(flow.first, list[from[v]].cap);
      flow.second += list[from[v]].cost;
    }
    for(int v = sink; v != src; v = list[from[v]^1].to) {
      list[from[v]].cap -= flow.first;
```

```
      list[from[v]^1].cap += flow.first;
    }
    return flow;
  }
  queue<int> q;
  bool SPFA(int src, int sink) {
    T INF = numeric_limits<T>::max();
    dist.assign(n, INF);
    from.assign(n, -1);
    q.push(src);
    dist[src] = 0;
    while(!q.empty()) {
      int on = q.front();
      q.pop();
      visit[on] = false;
      for(auto e : edges[on]) {
        auto ed = list[e];
        if(ed.cap == 0) continue;
        T toDist = dist[on] + ed.cost + pot[on] - pot[ed.to];
        if(toDist < dist[ed.to]) {
          dist[ed.to] = toDist;
          from[ed.to] = e;
          if(!visit[ed.to]) {
            visit[ed.to] = true;
            q.push(ed.to);
          }
        }
      }
    }
    return dist[sink] < INF;
  }
  void fixPot() {
    T INF = numeric_limits<T>::max();
    for(int i = 0; i < n; i++) {
      if(dist[i] < INF) pot[i] += dist[i];
    }
  }
};
```

5.8 Policy Based

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
ordered_set X;
X.insert(1); X.find_by_order(0); // Acha a key na ordem Y
X.order_of_key(-5); // Acha a ordem da key Y
end(X), begin(X);
```

5.9 2-sat

```
#define PB push_back
// Kosaraju
struct TwoSat{
        int n;
        vector<vector<int>> G, Gt;
        vector<int> id, order, ans;
        vector<bool> vis;

        TwoSat(){}
        TwoSat(int n) : n(n){init();}
        void init(){
                G.resize(2*n);
                Gt.resize(2*n);
                id.resize(2*n);
                ans.resize(n);
        }
        void add_edge(int u, int v){
                G[u].PB(v);
                Gt[v].PB(u);
        }
        // Algum ser 1
        void add_or(int a, bool neg1, int b, bool neg2){
                // (neg1 A | neg2 B) = (!neg1 A -> neg2 B) & (!neg2 B ->
                    neg1 A)
                add_edge(a + (neg1 ? 0 : n), b + (neg2 ? n : 0));
                add_edge(b + (neg2 ? 0 : n), a + (neg1 ? n : 0));
        }
        // Apenas algum ser 1
        void add_xor(int a, bool neg1, int b, bool neg2){
                 add_or(a,neg1,b,neg2);
                 add_or(a,!neg1,b,!neg2);
        }
        // Setar variavel a pra b
        void set(int a, bool b){ // (a|a)
                add_or(a,!b,a,!b);
        }
        // Mesmo valor
    void add_xnor(int a, bool neg1, int b, bool neg2) {
        add_xor(a, !neg1, b, neg2);
    }
    //

        void dfs1(int v){ // ordem de saida
          vis[v] = true;
          for(int u : G[v]){
            if(!vis[u]){
              dfs1(u);
            }
          }
          order.PB(v);
        }
        void dfs2(int v, int idx){ // pegar um componente todo
            vis[v] = true;
            id[v] = idx;
            for(int u : Gt[v]){
                if(!vis[u]) dfs2(u,idx);
            }
        }
        void kosaraju(){
          vis.assign(2*n,false);
          for(int i = 0; i < 2*n; i++){
                if(!vis[i]) dfs1(i);
          }
          vis.assign(2*n,false);
          reverse(begin(order),end(order));
          int idx = 0;
          for(int v : order){
            if(!vis[v]) dfs2(v, idx++);
          }
        }
        bool satisfiable(){
                kosaraju();
                for(int i = 0; i < n; i++){
                        if(id[i] == id[i+n]) return false;
                        ans[i] = (id[i] > id[i+n]);
                }
                return true;
        }
};
```

6 Math

6.1 Extended Euclidean

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

// inverso modular de a
int inv, y;
int g = gcd(a,mod,inv,y);
inv = (inv % m + m) % m;
```

6.2 Factorization

```
// Factorization
vector<pii> getFact(int n){
        vector<pii> primes;
        for(int p = 2; p*p <= n; p++){
                if(n % p == 0){
                        int exp = 0;
                        while(n % p == 0){
                                exp++;
                                n /= p;
                        }
                        primes.PB({p,exp});
                }
        }
        if(n > 1) primes.PB({n,1});
        return primes;
}
```

6.3 Fastexp

```
// Fast Exp
const ll mod = 1e9+7;


ll fexpll(ll a, ll n){
        ll ans = 1;
        while(n){
                if(n & 1) ans = (ans * a) % mod;
                a = (a * a) % mod;
                n >>= 1;
        }
        return ans;
}
// matriz quadrada
class Matrix{
        public:
        vector<vector<ll>> mat;
        int m;
        Matrix(int m): m(m){
                mat.resize(m);
                for(int i = 0; i < m; i++) mat[i].resize(m,0);
        }
        Matrix operator * (const Matrix& rhs){
                Matrix ans = Matrix(m);
                for(int i = 0; i < m; i++)
                        for(int j = 0; j < m; j++)
                                for(int k = 0; k < m; k++)
                                        ans.mat[i][j] = (ans.mat[i][j] + (
                                                mat[i][k] * rhs.mat[k][j]) %
                                                mod) % mod;
                return ans;
        }
};

Matrix fexp(Matrix a, ll n){
        int m = a.m;
        Matrix ans = Matrix(m);
        for(int i = 0; i < m; i++) ans.mat[i][i] = 1;
        while(n){
                if(n & 1) ans = ans * a;
                a = a * a;
                n >>= 1;
        }
        return ans;
}
```

6.4 Phi

```
const int LIM = 1e6+5;
int phi[LIM];
void sieve(){
        iota(phi, phi + LIM, 0);
        for(int i = 2; i < LIM; i++){
                if(phi == i){
                        for(int j = i; j < LIM; j += i){
                                phi[j] -= phi[j] / i;
                        }
                }
        }
}

template<typename T>
T phi(T n) {
  T ans = n;
  for(T p = 2; p * p <= n; p++) {
    if(n % p == 0) {
      ans -= ans / p;
      while(n % p == 0) {
        n /= p;
      }
    }
  }
  if(n > 1) {
    ans -= ans / n;
  }
  return ans;
}
```

6.5 Pollard Rho

```
// from: https://github.com/kth-competitive-programming/kactl
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
        ll ret = a * b - M * ull(1.L / M * a * b);
        return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
        ull ans = 1;
        for (; e; b = modmul(b, b, mod), e /= 2)
```

```
            if (e & 1) ans = modmul(ans, b, mod);
        return ans;
}

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

ull pollard(ull n) {
        ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
        auto f = [&](ull x) { return modmul(x, x, n) + i; };
        while (t++ % 40 || __gcd(prd, n) == 1) {
                if (x == y) x = ++i, y = f(x);
                if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
                x = f(x), y = f(f(y));
        }
        return __gcd(prd, n);
}
vector<ull> factor(ull n) {
        if (n == 1) return {};
        if (isPrime(n)) return {n};
        ull x = pollard(n);
        auto l = factor(x), r = factor(n / x);
        l.insert(l.end(), begin(r),end(r));
        return l;
}
```

6.6 Polynomial

```
template<typename T>
struct Poly {
  int n;
  vector<T> v;
  Poly(int sz) : n(sz+1) { v.resize(sz+1,0);}
  friend Poly operator*(const Poly& lhs, const Poly& rhs) {
    int grauL = (int)lhs.n - 1;
    int grauR = (int)rhs.n - 1;
    Poly ans(grauR+grauL);
    for(int i = 0; i <= grauL; ++i) {
      for(int j = 0; j <= grauR; ++j) {
        ans.v[i + j] += lhs.v[i] * rhs.v[j];
      }
    }
    return ans;
  }
  void set_identity() { // 1
    v[0] = T(1);
    for(int i = 1; i < n; ++i) {
      v[i] = T(0);
    }
  }
};
template<typename T>
Poly<T> poly_exp(Poly<T> a, long long e) {
  Poly<T> r(0);
  r.set_identity();
  for(; e > 0; e >>= 1) {
    if(e & 1) {
      r = r * a;
    }
    a = a * a;
  }
  return r;
}
```

7 String

7.1 KMP

```
vector<int> getBorder(string str) {
  int n = str.size();
  vector<int> border(n, -1);
  for(int i = 1, j = -1; i < n; i++) {
    while(j >= 0 && str[i] != str[j + 1]) {
      j = border[j];
    }
    if(str[i] == str[j + 1]) {
      j++;
    }
    border[i] = j;
  }
  return border;
}
int matchPattern(const string &txt, const string &pat, const vector<int> &
    border) {
  int freq = 0;
  for(int i = 0, j = -1; i < txt.size(); i++) {
    while(j >= 0 && txt[i] != pat[j + 1]) {
      j = border[j];
    }
    if(pat[j + 1] == txt[i]) {
      j++;
    }
    if(j + 1 == (int) pat.size()) {
      //found occurence
      freq++;
      j = border[j];
    }
  }
  return freq;
}
```

7.2 RabinKarp

```
// Rabin Karp
const ll base = 997;
const ll mod[] ={1000000007, 1000000009};
using Hash = pair<ll,ll>;
const int str_mxsz = 1e5+2;
ll pot[str_mxsz][2];
void buildPots(){ // lembrar de chamar essa funcao
        pot[0][0] = 1;
        pot[0][1] = 1;
        for(int i = 1; i < str_mxsz; i++)
                for(int j = 0; j < 2; j++)
                        pot[i][j] = (pot[i-1][j]*base) % mod[j];
}

class RabinKarp{
public:
        string s;
        int sz;
        vector<vector<ll>> has;
        RabinKarp(){}
        RabinKarp(const string& str): s(str){
                sz = str.size();
                has.assign(sz+1,vector<ll>(2));
```

```
        build();
    }
    void build(){
        has[0] = {s[0],s[0]};
        for(int i = 1; i < sz; i++)
            for(int j = 0; j < 2; j++)
                has[i][j] = ((has[i-1][j]*base)+s[i])%mod[j
                    ];
    }
    Hash getKey(ll l, ll r){ // inclusivo
        Hash ans = {has[r][0],has[r][1]};
        if(l > 0){
            ans.first = (((ans.first - pot[r-l+1][0]*has[l
                -1][0])%mod[0] + mod[0])%mod[0]);
            ans.second = (((ans.second - pot[r-l+1][1]*has[l
                -1][1])%mod[1] + mod[1])%mod[1]);
        }
        return ans;
    }
};
```

7.3 Trie

```
int trie[ms][sigma], terminal[ms], z = 1;

void insert(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      trie[cur][id] = z++;
    }
    cur = trie[cur][id];
  }
  terminal[cur]++;
}

int count(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = p[i]-'a';
    if(!trie[cur][id]) {
      return false;
    }
    cur = trie[cur][id];
  }
  return terminal[cur];
}
```

7.4 Z Function

```
vector<int> Zfunction(string &s){
    int n = s.size();
    vector<int> z (n, 0);
    for(int i=1, l=0, r=0; i<n;  i++)       {
        if(i <= r) z[i] = min(z[i-l], r-i+1);
        while(z[i] + i < n && s[z[i]] == s[i+z[i]]) z[i]++;
        if(r < i+z[i]-1) l = i, r = i+z[i]-1;
    }
    return z;
}
```