Kinhosz - ICPC Library

Contents

1 Data Structures

1.1 BIT - Binary Indexed Tree

```
int bit[MAXN];

int query(int x){
        int s = 0;

        while(x > 0){
                s += bit[x];
                x -= (x & -x);
        }

        return s;
}

void update(int x,int v){
        while(x < MAXN){
                bit[x] += v;
                x += (x & -x);
        }
}
```

1.2 Color Update

```
class ColorUpdate{

public:
        struct Range{

                Range(int l, int r, int c){
                        this->l = l;
                        this->r = r;
                        this->color = c;
                }

                Range(int l){
                        this->l = l;
                }

                int l;
                int r;
                int color;

                bool operator < (const Range &b) const { return l < b.l; }
        };

        bool exists(int x){

                auto it = ranges.upper_bound(Range(x));

                if(it == ranges.begin()) return false;
                it--;

                return it->l <= x && x <= it->r;
        }

        Range get(int x){

                auto it = ranges.upper_bound(Range(x));

                it--; // assuming it always exists

                return *it;
        }

        vector<Range> erase(int l, int r){

                vector<Range> ret;

                auto it = ranges.upper_bound(Range(l));

                if(it != ranges.begin()) it--;

                while(it != ranges.end()){

                        if(it->l > r) break;
                        else if(it->r >= l) ret.push_back(*it);

                        it++;
                }

                if(ret.size() > 0){

                        int sz = ret.size();

                        auto s = ranges.lower_bound(Range(ret[0].l));
                        auto e = ranges.lower_bound(Range(ret[sz-1].l));

                        Range ts = *s;
                        Range te = *e;

                        e++;
                        ranges.erase(s, e);

                        Range r1 = Range(ts.l, l-1, ts.color);
                        Range r2 = Range(r + 1, te.r, te.color);

                        ret[0].l = max(ret[0].l, l);
```

```
                ret[sz-1].r = min(ret[sz-1].r, r);

                if(r1.l <= r1.r) ranges.insert(r1);
                if(r2.l <= r2.r) ranges.insert(r2);
            }

            return ret;
        }

        vector<Range> upd(int l, int r, int color){

            vector<Range> ret = erase(l, r);

            ranges.insert(Range(l, r, color));

            return ret;
        }

    private:
        set<Range> ranges;
};
```

1.3 DSU - Disjoint Union Set

```
typedef vector<int> vi;

class DSU{

private: vi p,rank;
public:
        void create(int N){
                rank.assign(N,1);
                p.resize(N);
                for(int i=0;i<N;i++) p[i] = i;
        }
        int find(int i){
                return (p[i] == i?i:(p[i] = find(p[i])));
        }
        bool isSameSet(int u,int v){

                return (find(p[u]) == find(p[v]));
        }
        void unionSet(int u,int v){
                u = find(p[u]);
                v = find(p[v]);
                if(!isSameSet(u,v)){
                        if(rank[u] > rank[v]) swap(u,v);
                        p[u] = v;
                        rank[v]+= rank[u];
                }
        }
};
```

1.4 Merge Sort Tree

```
class MergeSortTree{
  struct Node{
    vector<int> vs;

    void add(int v){
      this->vs.push_back(v);
    }

    void build(){
      sort(vs.begin(), vs.end());
    }
  };
  vector<Node> nodes;
  int N;

  void add(int p, int l, int r, int x, int v){
    int m = (l+r)/2;
    int pl = p*2;
    int pr = p*2 + 1;

    nodes[p].add(v);
    if(l == r) return;

    if(x <= m) add(pl, l, m, x, v);
    else add(pr, m+1, r, x, v);
  }

  void build(int p){
    if(p >= nodes.size()) return;
    nodes[p].build();
    build(2*p);
    build(2*p + 1);
  }

public:
  MergeSortTree(int n){
    N = n;
    nodes.resize(4*n);
  }

  void add(int x, int v){
    add(1, 0, N-1, x, v);
  }

  void build(){
    build(1);
  }
};
```

1.5 Segment Tree

```
/*
add: add a value to current node on tree
join: join a query on lazy node
merge: combine two nodes into one */
class SegTree{
        int N;
        vector<ll> tree;
        vector<ll> lazy;

        ll add(ll curr, int i, int j, ll v){
                return curr + v * ll(j - i + 1);
        }

        ll join(ll curr, ll v){
                return curr + v;
        }

        ll merge(ll v1, ll v2){
                return v1 + v2;
        }

        void propagate(int pos, int i, int j){
                int esq = 2*pos;
                int dir = 2*pos + 1;

                if(lazy[pos]){
                        tree[pos] = add(tree[pos], i, j, lazy[pos]);
                        if(i < j){
                                lazy[esq] = join(lazy[esq], lazy[pos]);
                                lazy[dir] = join(lazy[dir], lazy[pos]);
                        }
                        lazy[pos] = 0;
```

```
        }
    }

    void upd(int pos, int i, int j, int l, int r, ll v){
        int esq = 2*pos;
        int dir = 2*pos + 1;
        int mid = (i+j)/2;

        propagate(pos, i, j);

        if(i > r || j < l) return;
        else if(i >= l && j <= r){
            tree[pos] = add(tree[pos], i, j, v);
            if(i < j){
                lazy[esq] = join(lazy[esq], v);
                lazy[dir] = join(lazy[dir], v);
            }
        }
        else{
            upd(esq, i, mid, l, r, v);
            upd(dir, mid+1, j, l, r, v);
            tree[pos] = merge(tree[esq], tree[dir]);
        }
    }

    ll qry(int pos, int i, int j, int l, int r){
        int esq = 2*pos;
        int dir = 2*pos + 1;
        int mid = (i+j)/2;

        propagate(pos, i, j);

        if(i > r || j < l) return 0;
        if(i >= l && j <= r) return tree[pos];
        else return merge(qry(esq, i, mid, l, r), qry(dir, mid+1, j
            , l, r));
    }

public:
    SegTree(int n){
        N = n;
        tree.resize(4*N + 3);
        lazy.resize(4*N + 3);
    }

    void upd(int l, int r, ll v){
        upd(1, 0, N-1, l, r, v);
    }

    ll qry(int l, int r){
        return qry(1, 0, N-1, l, r);
    }
};
```

1.6 Tetrix - Stack of Segments

```
class Tetrix{
  private:

  struct Range{
    int l, r;
    int id;
    bool active;

    Range(int l, int r, int id): l(l), r(r), id(id), active(true) {}
  };

  int MAXN;
  vector<stack<int>> st;
  vector<Range> ranges;
  vector<int> tree;

  bool isCovered(int i, int j, int l, int r){
    return (l <= i && r >= j);
  }

  bool isDisjoint(int i, int j, int l, int r){
    return (l > j || r < i);
  }

  int lazyTop(int pos){
    return (st[pos].empty()? -1 : st[pos].top());
  }

  void lazy(int pos){
    while(!st[pos].empty()){
      int id = st[pos].top();
      if(!ranges[id].active) st[pos].pop();
      else break;
    }
  }

  int add(int pos, int i, int j, Range &range){
    int esq = 2*pos;
    int dir = 2*pos + 1;
    int mid = (i+j)/2;

    lazy(pos);

    if(isDisjoint(i, j, range.l, range.r));
    else if(isCovered(i, j, range.l, range.r)){
      st[pos].push(range.id);
    }
    else{
      tree[pos] = max(add(esq, i, mid, range), add(dir, mid+1, j, range));
    }

    return max(tree[pos], lazyTop(pos));
  }

  int remove(int pos, int i, int j, Range &range){
    int esq = 2*pos;
    int dir = 2*pos + 1;
    int mid = (i+j)/2;

    lazy(pos);

    if(isDisjoint(i, j, range.l, range.r) || isCovered(i, j, range.l, range
        .r));
    else{
      tree[pos] = max(remove(esq, i, mid, range), remove(dir, mid+1, j,
          range));
    }

    return max(tree[pos], lazyTop(pos));
  }

  int query(int pos, int i, int j, int l, int r){
    int esq = 2*pos;
    int dir = 2*pos + 1;
    int mid = (i+j)/2;

    if(isDisjoint(i, j, l, r)) return -1;
    else if(isCovered(i, j, l, r)) return max(tree[pos], lazyTop(pos));
    else{
      return max({query(esq, i, mid, l, r), query(dir, mid+1, j, l, r),
          lazyTop(pos)});
    }
  }

  public:

  Tetrix(int maxn){
    MAXN = maxn;
```

```
    tree.resize(4*MAXN + 3, -1);
    st.resize(4*MAXN + 3);
  }

  void push(int l, int r, int id){
    Range range(l, r, id);
    ranges.push_back(range);
    add(1, 0, MAXN-1, range);
  }

  void pop(int id){
    Range range = ranges[id];
    ranges[id].active = false;
    remove(1, 0, MAXN-1, range);
  }

  int get(int l, int r){
    int id = query(1, 0, MAXN-1, l, r);
    return id;
  }
};
```

1.7 Treap

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

class Treap{

private:
        struct Node{

                ll key; // key
                ll prior; // randomized
                ll value; // value for this node
                ll tree; // value for this subtree

                Node *l; // data of l node
                Node *r; // data of r node
                Node(){}
                Node(ll key, ll value): key(key),
                 value(value), prior(uniform_int_distribution<int>()(rng)),
                      l(NULL), r(NULL){}
        };
        typedef Node* node;

        node root = NULL;

        void split(node t, ll key, node &l, node &r){

                if(!t){
                        l = r = NULL;
                }
                else if(t->key <= key){
                        split(t->r, key, t->r, r), l = t;
                }
                else{
                        split(t->l, key, l, t->l), r = t;
                }

                update(t);
        }

        void insert(node &t, node item){

                if(!t){
                        t = item;
                }
                else if(item->prior > t->prior){
                        split(t, item->key, item->l, item->r), t = item;
                }
                else if(t->key <= item->key){
                        insert(t->r, item);
                }
                else{
                        insert(t->l, item);
                }

                update(t);
        }

        void merge(node &t, node l, node r) {
            if (!l || !r)
                t = l ? l : r;
            else if (l->prior > r->prior)
                merge (l->r, l->r, r),  t = l;
            else
                merge (r->l, l, r->l),  t = r;

            update(t);
        }

        void erase(node &t, ll key){
                node th;

                if(!t) return ;

                if(t->key == key){
                        th = t;
                        merge(t, t->l, t-> r);
                        delete th;
                }
                else if(t->key < key){
                        erase(t->r, key);
                }
                else{
                        erase(t->l, key);
                }

                update(t);
        }

        ll getValue(node t){

                if(!t) return -1;

                return t->value;
        }

        ll getTree(node t){

                if(!t) return 0;

                return t->tree;
        }

        void update(node t){

                if(!t) return ;
                t->tree = t->value;
                t->tree += getTree(t->l);
                t->tree += getTree(t->r);
        }

        ll prefix(node t, ll key){

                if(!t) return 0;

                if(t->key <= key){
                        return getTree(t->l) + prefix(t->r, key) + t->value
                            ;
                }
                else{
                        return prefix(t->l, key);
                }
```

```
        }

        ll suffix(node t, ll key){

                if(!t) return 0;

                if(t->key < key){
                        return suffix(t->r, key);
                }
                else{
                        return suffix(t->l, key) + getTree(t->r) + t->value
                            ;
                }
        }

        ll find(node t, ll key){

                if(!t) return -1;

                if(t->key == key) return t->value;
                if(t->key < key) return find(t->r, key);
                else return find(t->l, key);
        }

public:
        void add(ll key, ll value){
                node t = new Node(key, value);
                insert(root, t);
        }

        void remove(ll key){
                erase(root, key);
        }

        ll prefix(ll key){
                return prefix(root, key);
        }

        ll suffix(ll key){
                return suffix(root, key);
        }

        ll find(ll key){
                return find(root, key);
        }
};
```

1.8 Segment Tree 2D

```
const int SZ = 1<<20;
template<class T> struct node_t {
        T delta = -1; node_t<T>* c[2];
        node_t() { c[0] = c[1] = nullptr; }
        void upd(int pos, T v, int L = 0, int R = SZ-1) { // upd v
                if (L == pos && R == pos) { delta = v; return; }
                int M = (L + R)>>1;
                if (pos <= M) {
                        if (!c[0]) c[0] = new node_t();
                        c[0]->upd(pos, v, L, M);
                } else {
                        if (!c[1]) c[1] = new node_t();
                        c[1]->upd(pos, v, M+1, R);
                }
                delta = -1;
                for (int i = 0; i < 2; ++i) if (c[i]) delta = max(delta, c[
                    i]->delta);
        }
        T query(int lx, int rx, int L = 0, int R = SZ-1) { // query sum of
             segment
                if (rx < L || R < lx) return -1;
                if (lx <= L && R <= rx) return delta;
                int M = (L + R)>>1; T res = -1;
                if (c[0]) res = max(res, c[0]->query(lx, rx, L, M));
                if (c[1]) res = max(res, c[1]->query(lx, rx, M+1, R));
                return res;
        }
        void upd(int pos, node_t *a, node_t *b, int L = 0, int R = SZ-1) {
                if (L != R) {
                        int M = (L + R)>>1;
                        if (pos <= M) {
                                if (!c[0]) c[0] = new node_t();
                                c[0]->upd(pos, a ? a->c[0] : nullptr, b ? b
                                    ->c[0] : nullptr, L, M);
                        } else {
                                if (!c[1]) c[1] = new node_t();
                                c[1]->upd(pos,a ? a->c[1] : nullptr, b ? b
                                    ->c[1] : nullptr, M+1, R);
                        }
                }
                delta = max((a ? a->delta : -1), (b ? b->delta : -1));
        }
};

template<class T> struct Node {
    node_t<T> seg; Node* c[2];
    Node() { c[0] = c[1] = nullptr; }
    void upd(int x, int y, T v, int L = 0, int R = SZ-1) {
        if (L == x && R == x) { seg.upd(y,v); return; }
        int M = (L+R)>>1;
        if (x <= M) {
            if (!c[0]) c[0] = new Node();
            c[0]->upd(x,y,v,L,M);
        } else {
            if (!c[1]) c[1] = new Node();
            c[1]->upd(x,y,v,M+1,R);
        }
        //seg.upd(y,v); // only for addition
        seg.upd(y,c[0]?&c[0]->seg:nullptr,c[1]?&c[1]->seg:nullptr);
    }
    T query(int x1, int x2, int y1, int y2, int L = 0, int R = SZ-1) { //
         query sum of rectangle
        if (x1 <= L && R <= x2) return seg.query(y1,y2);
        if (x2 < L || R < x1) return -1;
        int M = (L+R)>>1; T res = -1;
        if (c[0]) res = max(res, c[0]->query(x1, x2, y1, y2, L, M));
        if (c[1]) res = max(res, c[1]->query(x1, x2, y1, y2, M+1, R));
        return res;
    }
};
// by: LeticiaFCS
```

2 Dynamic Programming

2.1 CHT - Convex Hull Trick (Persistent)

```
class CHTPersistent{
  struct Line{
    ll m;
    ll c;
    Line(){}
    Line(ll _m, ll _c): m(_m), c(_c){}
  };

  vector<vector<Line>> hull;
  int SZ = 0;
  vector<int> version_idx;
  vector<int> version_sz;

  double inter(Line t1, Line t2){
    double ret;
```

```
    ret = (double)(t2.c - t1.c)/(t1.m - t2.m);
    return ret;
  }

  void add(Line curr){
    Line temp, temp2;
    version_sz.push_back(SZ);

    if(SZ > 1){
      int s = 0;
      int e = SZ-1;

      while(s < e){
        int p = (s+e)/2;

        temp = hull[p+1].back();
        temp2 = hull[p].back();

        double point = inter(temp, temp2);
        double point2 = inter(temp, curr);

        if(point < point2){
          s = p+1;
        }
        else{
          e = p;
        }
      }
      SZ = s+1;
    }

    if(hull.size() == SZ){
      vector<Line> x;
      hull.push_back(x);
    }

    hull[SZ].push_back(curr);
    version_idx.push_back(SZ);
    SZ++;
  }

public:
  void add(ll m, ll c){
    add(Line(m, c));
  }

  ll query(ll find){
    int s = 0;
    int e = SZ-1;

    while(s < e){
      int p = (s+e)/2;

      double point = inter(hull[p].back(), hull[p+1].back());
      if(point < find){
        s = p+1;
      }
      else{
        e = p;
      }
    }

    ll ret = (hull[s].back().m * find) + hull[s].back().c;
    return ret;
  }

  void rollback(){
    SZ = version_sz.back();
    version_sz.pop_back();
    hull[version_idx.back()].pop_back();
    version_idx.pop_back();
  }

  int size(){
    return SZ;
  }
};

// log(n) for query & add. O(1) for rollback. All lines should be added in
    crescent angular coef order
```

2.2 SOS - Subset Over Sum DP

```
void sosdp(){
        int bmask;

        for(int i=0;i<22;i++){
                for(int mask=0;mask<maxn;mask++){
                        if(mask & (1<<i)){
                                bmask = mask^(1<<i);
                                dp[mask] = max(dp[mask],dp[bmask]);
                        }
                }
        }
}
```

3 Graph

3.1 Dinic

```
const ll INF = 1e9 + 7;

class Dinic{
        int N;
        vector<ll> level;
        vector<bool> dead;

public:
        struct Edge{
                Edge(int a,ll x){
                        v = a;
                        cap = x;
                }
                int v;
                ll cap;
        };
        int source;
        int sink;
        vector<Edge> edge;
        vector<vector<int>> g;

        Dinic(int n){
                g.resize(n);
                N = n;
                level.resize(n);
        }

        void setInit(int u,int v){
                source = u;
                sink = v;
        }

        void addEdge(int u,int v,ll cap){
                g[u].push_back(edge.size());
                edge.push_back(Edge(v,cap));
                g[v].push_back(edge.size());
                edge.push_back(Edge(u,0));
        }
private:

        bool BFS(){
```

```
                for(int i=0;i<N;i++) level[i] = INF;
                dead.clear();
                dead.resize(N, false);
                level[source] = 0;
                queue<int> q;
                q.push(source);

                while(!q.empty()){

                        int u = q.front();
                        q.pop();

                        if(u == sink) return true;

                        for(auto x: g[u]){
                                if(level[edge[x].v] == INF && edge[x].cap >
                                      0){
                                   level[edge[x].v] = level[u] + 1;
                                   q.push(edge[x].v);
                                }
                        }
                }
                return false;
        }

        ll maxflow(int u,ll flow){

                if(dead[u]) return 0;

                ll ret = 0;
                ll f = 0;
                if(flow == 0) return 0;
                if(u == sink) return flow;

                for(auto i: g[u]){
                        if(level[edge[i].v] != level[u] + 1) continue;
                        f = maxflow(edge[i].v,min(edge[i].cap,flow));
                        int x = (i%2 == 0?i+1:i-1);
                        flow -= f;
                        ret += f;
                        edge[i].cap -= f;
                        edge[x].cap += f;
                }

                if(ret == 0) dead[u] = true;

                return ret;
        }
public:
        ll run(){

                ll flow = 0;
                while(BFS()){
                        flow += maxflow(source,INF);
                }

                return flow;
        }
};
```

3.2 HLD - Heavy-Ligth Decomposition

```
typedef long long ll;
typedef pair<int,int> ii;

class HLD{

        vector<int> pos;
        vector<int> parent;
        vector<int> sz;
        vector<int> level;
        vector<int> head;
        vector<vector<ii>> g;

        // segment tree
        vector<int> tree;
        vector<int> lazy;

        int N;

        void propagate(int pos,int i,int j){

        }

        int query(int pos,int i,int j,int l,int r){

        }

        void update(int pos,int i,int j,int l,int r,int w){

        }

        void dfs(int u,int lv){

                level[u] = lv;
                sz[u] = 1;
                int bigChild = 0;

                for(int i=0;i<g[u].size();i++){
                        ii topo = g[u][i];
                        int v = topo.first;
                        if(v == parent[u]) continue;
                        parent[v] = u;
                        dfs(v,lv+1);
                        sz[u] += sz[v];
                        if(sz[v] > bigChild) swap(g[u][i],g[u][0]);
                        bigChild = max(bigChild,sz[v]);
                }
        }

        void decompose(int u,int &x,bool keep){

                if(keep){
                        head[u] = head[parent[u]];
                }
                else head[u] = u;

                pos[u] = x++;

                if(sz[u] > 1) decompose(g[u][0].first,x,true);

                for(int i=1;i<g[u].size();i++){
                        ii topo = g[u][i];
                        int v = topo.first;
                        if(v == parent[u]) continue;
                        decompose(v,x,0);
                }
        }

public:

        HLD(int n){
                N = n;
                pos.resize(n,-1);
                parent.resize(n,-1);
                sz.resize(n,-1);
                level.resize(n,-1);
                head.resize(n,-1);
                g.resize(n);
                // segment tree
                tree.resize(4*n+3,-1);
                lazy.resize(4*n+3,-1);
        }

        void addEdge(int u,int v,int w=-1){
```

```
        g[u].push_back({v,w}); // vertex, weight
    }

    void init(){
        parent[0] = -1;
        dfs(0,0);
        int x=0;
        decompose(0,x,0);
    }

    int LCA(int u,int v){
        while(head[u] != head[v]){
            if(level[head[u]] > level[head[v]]) u = parent[head
                [u]];
            else v = parent[head[v]];
        }
        return (level[u] < level[v]?u:v);
    }

    int join(int a,int b){

        return a+b;
    }

    int get(int u,int v){
        int l = LCA(u,v);
        int ret = 0;
        int add;
        while(head[u] != head[l]){
            add = query(1,0,N-1,pos[head[u]],pos[u]);
            ret = join(add,ret);
            u = parent[head[u]];
        }
        add = query(1,0,N-1,pos[l]+1,pos[u]);
        ret = join(add,ret);
        while(head[v] != head[l]){
            add = query(1,0,N-1,pos[head[v]],pos[v]);
            ret = join(add,ret);
            v = parent[head[v]];
        }
        add = query(1,0,N-1,pos[l],pos[v]);// para hld de arestas,
             mude isso aqui para pos[l]+1
        ret = join(add,ret);
        return ret;
    }

    void flip(int u,int v,int w){
        int l = LCA(u,v);
        while(head[u] != head[l]){
            update(1,0,N-1,pos[head[u]],pos[u],w);
            u = parent[head[u]];
        }
        update(1,0,N-1,pos[l]+1,pos[u],w);
        while(head[v] != head[l]){
            update(1,0,N-1,pos[head[v]],pos[v],w);
            v = parent[head[v]];
        }
        update(1,0,N-1,pos[l],pos[v],w); // para hld de arestas,
             mude isso aqui para pos[l]+1
    }

    int lenPath(int u,int v){
        int l = LCA(u,v);
        int ret = level[u] - level[l];
        ret += level[v] - level[l];
        ret++;
        return ret;
    }
};
```

3.3 LCA - Lowest Common Ancestor

```
int parent[MAXN][MAXL];
int level[MAXN];
vector<vector<int>> g;

void dfs(int u){
    for(auto &v: g[u]){
        if(level[v] == -1){
            level[v] = level[u] + 1;
            parent[v][0] = u;
            dfs(v);
        }
    }
}

void init(int root,int n){
    for(int i=0;i<n;i++){
        parent[i][0] = -1;
        level[i] = -1;
    }

    level[root] = 0;
    dfs(root);
    for(int j=1;j<MAXL;j++){
        for(int i=0;i<n;i++){
            parent[i][j] = parent[parent[i][j-1]][j-1]; // meu
                 avo eh pai do meu pai
        }
    }
}

int LCA(int u,int v){

    if(level[u] < level[v]) swap(u,v);

    for(int i=MAXL-1;i>=0;i--){
        if(level[u] - (1<<i) >= level[v]){
            u = parent[u][i];
        }
    }

    if(u == v) return u;

    for(int i=MAXL-1;i>=0;i--){
        if(parent[u][i] != -1 && parent[u][i] != parent[v][i]){
            u = parent[u][i];
            v = parent[v][i];
        }
    }

    return parent[u][0];
}
```

4 Number Theory

4.1 Miller Rabin

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
ll rnd(ll a, ll b){
    return uniform_int_distribution<ll>(a, b)(rng);
}

//Retorna a*b % mod. eh necessario pra multiplicar
//dois long long sem dar overflow.
//Se os numeros que vc quer testar sao menores que 2*10^9,
//pode deixar o return (a*b)%mod.
ll mod_mul(ll a, ll b, ll mod){
    //return (a*b)%mod;
    ll cur_mod = a;
    ll ans = 0;
```

```
    for (int i=0; i<63; i++){
        if ((b>>i)&1) ans = (ans + cur_mod)%mod;
        cur_mod = (2*cur_mod) % mod;
    }
    return ans;
}

//Exponenciacao rapida - calcula (a^e)%mod em O(log(e)).
ll fexp(ll a, ll e, ll mod){
    if (e == 0) return 1;
    ll p = fexp(a, e/2, mod);
    p = mod_mul(p, p, mod);
    if (e%2 == 1)
        p = mod_mul(p, a, mod);
    return p;
}

//Miller-Rabin. Checa se o numero p eh fortemente pseudoprimo na base a.
//Complexidade: O(log(p)) sem modmul, O(log^2(p)) com modmul.
bool miller_rabin (ll p, ll a){

    ll q = p-1, k=0;
    while(q % 2 == 0){
        q /= 2;
        k++;
    }

    ll cur = fexp(a, q, p);
    if (cur == 1 or cur == p-1) return true;

    for (int i=0; i<k; i++){
        if (cur == p-1) return true;
        if (cur == 1) return false;
        cur = mod_mul(cur, cur, p);
    }

    return false;
}

//Testa o algoritmo de miller rabin pra varias bases.
//p => numero testado, k => numero de bases.
//A probabilidade de erro (identificar que um
//numero eh primo quando na vdd eh composto) eh de (1/4)^k
//k = 40 eh uma boa opcao, se o TL apertar vc pode diminuir isso ai
bool is_probably_prime(ll p, int k){
    if (p == 0 or p == 1) return false;
    if (p == 2 or p == 3) return true;
    if (p%2 == 0) return false;

    for (int i=0; i<k; i++) {
        if (!miller_rabin(p, rnd(1, p-1))) return false;
    }

    return true;
}
```

5 String

5.1 Aho-corasick

```
class Aho{
        vector<map<char, int>> to;
        vector<int> link, term, exit, sobe;
        int idx = 0;

public:
        Aho(int maxn){
                to.resize(maxn);
                link.resize(maxn, 0);
                term.resize(maxn, 0);
                exit.resize(maxn, 0);
                sobe.resize(maxn, 0);
        }

        void insert(string &s){
                int at = 0;
                for(char c: s){
                        auto it = to[at].find(c);
                        if(it == to[at].end()) to[at][c] = ++idx;
                        it = to[at].find(c);
                        at = it->second;
                }
                term[at]++, sobe[at]++;
        }

        // nao esquecer de chamar o build dps de inserir
        void build(){
                queue<int> q;
                q.push(0);
                link[0] = exit[0] = -1;
                while(q.size()){
                        int i = q.front(); q.pop();
                        for(auto p: to[i]){
                                int c = p.first, j = p.second;
                                int l = link[i];
                                while(l != -1 and !to[l].count(c)) l = link
                                    [l];
                                link[j] = l == -1 ? 0 : to[l][c];
                                exit[j] = term[link[j]] ? link[j] : exit[
                                    link[j]];
                                if(exit[j]+1) sobe[j] += sobe[exit[j]];
                                q.push(j);
                        }
                }
        }

        int query(string &s){
                int at=0, ans=0;
                for(char c: s){
                        while(at != -1 and !to[at].count(c)) at = link[at];
                        at = at == -1 ? 0 : to[at][c];
                        ans += sobe[at];
                }

                return ans;
        }
};
```

5.2 Palindromic Tree

```
struct eertree {
        vector<vector<int>> t;
        int n, last, sz;
        vector<int> s, len, link;
        vector<ll> qt;
        const int SIGMA = 26;

        eertree(int N) {
                t.resize(N+2, vector<int>(SIGMA));
                s = len = link = vector<int>(N+2);
                qt = vector<ll>(N+2);
                s[0] = -1;
                link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
                sz = 2, last = 0, n = 1;
        }

        void add(char c) {
                s[n++] = c -= 'a';
                while (s[n-len[last]-2] != c) last = link[last];
```

```
                if (!t[last][c]) {
                        int prev = link[last];
                        while (s[n-len[prev]-2] != c) prev = link[prev];
                        link[sz] = t[prev][c];
                        len[sz] = len[last]+2;
                        t[last][c] = sz++;
                }
                qt[last = t[last][c]]++;
        }
        int size() { return sz-2; }
        ll propagate() {
                ll ret = 0;
                for (int i = n; i > 1; i--) {
                        qt[link[i]] += qt[i];
                        ret += qt[i];
                }
                return ret;
        }
};
```

5.3 Suffix Array

```
typedef pair<int,int> ii;

class SArray{

public:
        vector<int> idx;
        vector<int> lcp;
        vector<int> rank;
        string word;

        void process(string &text){

                text += '$';
                word = text;
                int n = text.size();
                rank.resize(n);

                vector<ii> lista;
                for(int i=0;i<n;i++){
                        lista.push_back({text[i],i});
                }
                sort(lista.begin(),lista.end());
                for(int i=0;i<n;i++){
                        idx.push_back(lista[i].second);
                }
                rank[idx[0]] = 0;
                int classe =0;
                for(int i=1;i<n;i++){
                        if(text[idx[i]] != text[idx[i-1]]) classe++;
                        rank[idx[i]] = classe;
                }

                int k = 1;
                while(k < n){

                        vector<int> aux(n);
                        vector<int> count(n,0);

                        for(int i=0;i<n;i++){
                                count[rank[idx[i]]]++;
                        }
                        for(int i=1;i<n;i++) count[i] += count[i-1];
                        for(int i=n-1;i>=0;i--){
                                int x = (idx[i]-k+n)%n;
                                aux[count[rank[x]] - 1] = x;
                                count[rank[x]]--;
                        }
                        swap(idx,aux);

                        vector<int> novo(n);

                        novo[idx[0]] = 0;
                        classe = 0;
                        for(int i=1;i<n;i++){
                                if(rank[idx[i]] != rank[idx[i-1]] || rank[(
                                    idx[i] + k)%n] != rank[(idx[i-1] + k)%
                                    n]){
                                        classe++;
                                }
                                novo[idx[i]] = classe;
                        }
                        swap(rank,novo);
                        k += k;
                }

                // lcp
                rank.clear();
                rank.resize(n);
                lcp.resize(n-1);

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

bool find(string &pat){

        int sz = pat.size();
        int dl = 0;
        int dr = idx.size()-1;

        for(int i=0;i<sz;i++){

                int s = dl;
                int e = dr;
                while(s < e){
                        int p = (s+e)/2;

                        int x = idx[p] + i;
                        if(word[x] >= pat[i]){
                                e = p;
                        }
                        else{
                                s = p+1;
                        }
                }
                dl = s;
                e = dr;
                while(s < e){
                        int p = (s+e)/2;

                        int x = idx[p] + i;
                        if(word[x] > pat[i]){
                                e = p;
                        }
                        else{
                                s = p+1;
                        }
```

```
                }
                if(word[idx[s]+i] > pat[i]) s--;
                dr = s;
            }

            return dl <= dr;
        }
};
```

6 Notes

6.1 Linear Progamming Primal/Dual Conversion

```
Primal <-> Dual

Seja o primal definido por:

1) max{ sum[1<=i<=N]:{ X(i) * B(i) }}
                B(i) pertence a um set B fixo.
                X(i) pertence a um set X -> variaveis

2) constraints, assumindo que existem M constraints:
                a) tipo ">=": sum[1<=i<=N]:{ A(j, i) * X(i) } >= C(j)
                b) tipo "<=": sum[1<=i<=N]:{ A(j, i) * X(i) } <= C(j)
                c) tipo "=":  sum[1<=i<=N]:{ A(j, i) * X(i) } = C(j)

                j: j-th constraint
                A: matrix que representa os valores de todas as constraints
                A(j, i): i-esimo valor da constraint j.
                C: resultado da inequacao/equacao da constraint

3) variaveis podendo ser:
                a) X(i) >= 0
                b) X(i) <= 0
                C) X(i) livre

a transformacao pro dual, eh dado por:

1) min{ sum[1<=j<=M]:{ Y(j) * C(j) }}
                Y(j) eh o novo conjunto de variaveis

2) novos dominios das variaveis:
                a) se o tipo da j-th constraint no primal eh ">=", entao Y(
                    j) <= 0.
                b) se o tipo da j-th constraint no primal eh "<=", entao Y(
                    j) >= 0.
                c) se o tipo da j-th constraint no primal eh "=", entao Y(j
                    ) eh livre.

3) novos constraints, N agora:
                a) se X(i) >= 0 no primal, entao:
                                sum[1<=j<=M]:{ A(i, j) * Y(j) } >= B(i)

                b) se X(i) <= 0 no primal, entao:
                                sum[1<=j<=M]:{ A(i, j) * Y(j) } <= B(i)

                c) se X(i) eh livre no primal, entao:
                                sum[1<=j<=M]:{ A(i, j) * Y(j) } = B(i)

A reciproca eh valida!
```