template.cpp

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

#define fastio() ios_base::sync_with_stdio(false);cin.tie(
    ↪ NULL);cout.tie(NULL)
#define foreach(iterator,object) for(auto iterator = object.
    ↪ begin(); iterator != object.end(); iterator++)
#define foreach_rev(iterator,object) for(auto iterator =
    ↪ object.rbegin(); iterator != object.rend(); iterator
    ↪ ++)

typedef long long ll;
typedef long long unsigned int llu;
typedef double dbl;
typedef long double ldbl;

typedef pair <int, int> pii;
typedef pair <ll,ll> pl;
typedef vector <ll> vl;
typedef vector <vl> vvl;
typedef vector <pl> vpl;
typedef vector <vpl> vvpl;

#define sll static ll
#define sllu static llu

#define pll pl

#define VI vector <int>
#define PQ priority_queue

#define fi first
#define se second

#define sz(x) (int)x.size()
#define lng(x) (int)x.length()
#define bgn(x) x.begin()
#define end(x) x.end()
#define all(x) (x).begin(),(x).end()

#define pb push_back
#define apnd append

#define is_even(x) ((x&1)?0:1)

#define elif else if
#define ln "\n"

void solve(ll case_number){

}

#define NUMBER_OF_TESTS

int main(){
    fastio();
    ll Q = 1;

#ifdef NUMBER_OF_TESTS
    ll T;
    cin >> T;
    for(Q=1; Q<=T; Q++){
        // cout << "Case #" << Q << ": ";
        solve(Q);
    }
#else
    // while(1)
        solve(Q++);
#endif
    return 0;
}
```

alias.sh

```sh
alias g='find␣.␣-maxdepth␣1␣-regex␣".*\.c\(pp\)?"␣-exec␣bash␣
    ↪ -c␣"g++␣-std=gnu++17␣-g3␣-fdiagnostics-color=always␣-
    ↪ o␣{}.o.tmp␣-c␣{}␣2>&1␣|␣head␣-n␣10"␣";";␣g++␣-o␣"$(
    ↪ basename␣"$(pwd)")"␣*.o.tmp␣2>␣/dev/null;␣rm␣*.o.tmp␣
    ↪ 2>␣/dev/null'
```

AhoCorasick.cpp

```cpp
#include "Trie.cpp"

class AhoCorasick{
private:
    vvl ans;
    Trie trie;
    vl wl;
public:
    AhoCorasick(){}
    AhoCorasick(const vs& dictionary){
        for(string s : dictionary){
            wl.pb(s.size());
        }
        trie = Trie(dictionary);
    }
    const vvl& find_occurrences(string& text){
        ll c;
        ans.assign(wl.size(), vl());
        for(ll i=0; i<text.size(); i++){
            // i is used later as an integer, don't change
                ↪ style
            c = text[i];
            for(ll l : trie.next_leaves(c)){
                ans[l].pb(i-wl[l]+1);
            }
        }
        return ans;
    }
};
```

ArticulationBridge.cpp

```cpp
template<class T> class ArticulationBridge{
#define g (*pg)
public:
    typedef long long ll;
    typedef std::pair<ll,ll> pl;
    typedef std::vector<ll> vl;
    typedef std::vector<T> vt;
    typedef std::vector<vt> vvt;
    typedef std::pair<pl,T&> plte;
    typedef std::vector<plte> vplte;
    typedef std::function<ll(const T&)> fetnt;
    static ll def_etn(const T& e){
        return e;
    }
    vl articulations, generated_components;
    vplte bridges, trees, backs; // , forward, cross;
private:
    ll n, pos;
    vvt* pg;
    vl num, low;
    fetnt etn;
    void dfs(ll p, ll u){
        ll v, c = 0, a = 0;
        num[u] = low[u] = pos++;
        for(T& ed : g[u]){
            v = etn(ed);
            if(!num[v]){
                c++;
                trees.pb(plte(pl(u,v), ed));
                dfs(u, v);
                if(num[u] <= low[v]) a++;
                if(num[u] < low[v]) bridges.pb(plte(pl(u,v), ed
                    ↪ ));
                if(low[u] > low[v]) low[u] = low[v];
            }else if(v != p){
                if(num[u] > num[v]) backs.pb(plte(pl(u,v), ed))
                    ↪ ;
                if(low[u] > num[v]) low[u] = num[v];
            }
        }
        if(p==-1) a=c-1;
        if(a > 0){
            generated_components[u] = a;
            articulations.pb(u);
        }
```

```
    }
public:
    ArticulationBridge(){}
    ArticulationBridge(const vvt& graph, ll v = -1, fetnt
        ↪ edge_to_node = def_etn){
        g = graph;
        if(v == -1){
            n = g.size();
        }else{
            n = v;
        }
        etn = edge_to_node;
        num.assign(n, 0);
        low.resize(n);
        generated_components.assign(n, 0);
        pos = 1;
        for(ll v=0; v<n; v++){
            if(!num[v]) dfs(-1, v);
        }
    }
#undef g
};
```

ConvexHull.cpp

```
class ConvexHull{
public:
    typedef std::vector<point> vpoint;
    vpoint convex_hull; // convex_hull[0] == convex_hull[chs
        ↪ -1], points are ccw
    ll chs;
    static point pivot; // static is required by angleCmp PAY
        ↪  ATTENTION TO THIS FOR MORE INSTANCES
private:
    static bool angleCmp(const point& a, const point& b) {
        if(collinear(pivot, a, b))
            return dist(pivot, a) < dist(pivot, b);
        dbl d1x = a.x - pivot.x, d1y = a.y - pivot.y;
        dbl d2x = b.x - pivot.x, d2y = b.y - pivot.y;
        return (atan2(d1y, d1x) < atan2(d2y, d2x));
    }
public:
    // #undef end
    ConvexHull(vector<point> P) {
            ll i, j, n = (ll)P.size();
            if(n <= 3) {
                if (!(P[0] == P[n-1])) P.pb(P[0]);
                convex_hull = P;
                chs = convex_hull.size();
                return;
            }
            // first, find P0 = point with lowest Y and if tie
                ↪ : rightmost X
            ll P0 = 0;
            for(i=1; i<n; i++)
                if(P[i].y < P[P0].y || (P[i].y == P[P0].y && P[
                    ↪ i].x > P[P0].x))
                    P0 = i;
            point temp = P[0];
            P[0] = P[P0];
            P[P0] = temp;
            // second, sort points by angle w.r.t. pivot P0
            pivot = P[0]; // use this global variable as
                ↪ reference
            // sort(++P.begin(), P.end(), angleCmp); // we
                ↪ do not sort P[0]
            sort(++P.begin(), end(P), angleCmp); // we do not
                ↪ sort P[0]
            // third, the ccw tests
            vpoint& S = convex_hull;
            S.pb(P[n-1]);
            S.pb(P[0]);
            S.pb(P[1]);
            i = 2;
            while (i < n){
                // note: N must be >= 3 for this method to work
                j = (ll)S.size()-1;
                if (ccw(S[j-1], S[j], P[i])) S.pb(P[i++]); //
                    ↪ left turn, accept
                else S.pop_back(); // or pop the top of S until
                    ↪  we have a left turn
            }
            chs = convex_hull.size();
            // return S;
        }
        // return the result
};

point ConvexHull::pivot = point(0,0); // use this global
    ↪ variable as reference
```

CoreDAG.cpp

```
#include "Tarjan.cpp"

template<class T> class CoreDAG{
#define g (*pg)
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
    typedef std::vector<vl> vvl;
    typedef std::vector<T> vt;
    typedef std::vector<vt> vvt;
    typedef typename Tarjan<T>::fetnt fetnt;
    typedef std::function<T&(T&, ll)> fsedt;
    typedef std::function<T&(T&, const T&)> fmet;
    typedef std::map<ll,T> mlt;
    ll nssc;
    vvt core;
private:
    ll n;
    const vvt* pg;
    fetnt etn;
    fsedt sed;
    fmet me;
    Tarjan<T> tj;
public:
    static T& def_me(T& sv, const T& nv){
        return sv;
    }
    static T& def_sed(T& sv, ll v){
        return sv = v;
    }
    CoreDAG(){}
    CoreDAG(
        const vvt& graph,
        ll v = -1,
        fetnt edge_to_node = Tarjan<T>::def_etn,
        fsedt set_edge_destination = def_sed,
        fmet merge_edges = def_me
    ){
        std::vector<mlt> mm;
        typename mlt::iterator mit;
        ll i, x, y;
        pg = &graph;
        me = merge_edges;
        etn = edge_to_node;
        sed = set_edge_destination;
        if(v == -1){
            n = g.size();
        }else{
            n = v;
        }
        tj = Tarjan<T>(graph, v, edge_to_node);
        nssc = tj.nssc;
        vl& ssc = tj.ssc;
        mm.resize(nssc);
        for(i=0; i<n; i++){
            x = ssc[i];
            for(const T& e : g[i]){
                y = ssc[etn(e)];
                if(x != y){
                mit = mm[x].find(y);
                    if(mit == mm[x].end()){
                        mm[x][y] = e;
                    }else{
                        me(mit->second, e);
                    }
                }
            }
        }
        core.resize(nssc);
        for(i=nssc; i--;){
```

```
            for(auto& e : mm[i]){
                core[i].push_back(sed(e.second, e.first));
            }
        }
    }
    vl& get_ssc(){
        return tj.ssc;
    }
    vvt& get_sscs(){
        return tj.get_sscs();
    }
#undef g
};
```

DeterminantGaussJordan.cpp

```
#define __GAUSS_JORDAN_EPS 10e-9

template <class T> class GaussJordan{
#define a (*A)
#define b (*B)
public:
    typedef long long ll;
    typedef double dd;
    typedef vector<T> gjvt;
    typedef vector<gjvt> gjvvt;
    static dd def_norm(const T& t){
        if(t<0) return -t;
        return t;
    }
    ll r;
    T d = 1;
    gjvvt* A;
    gjvvt* B;
private:
    ll n, ma, mb;
    dd eps;
    function <dd(const T&)> norm;
    inline bool is_zero(T& x){
        static T zero = (T)(0);
        if(norm(x) <= eps){
            x = zero;
            return true;
        }
        return false;
    }
    void print(){
        ll i, j;
        for(i=0; i<n; i++){
            for(j=0; j<ma; j++){
                cout << a[i][j] << '\t';
            }
            cout << "|\t";
            for(j=0; j<mb; j++){
                cout << b[i][j] << '\t';
            }
            cout << '\n';
        }
        cout << '\n';
    }
    void swap(gjvt& a0, gjvt& a1, gjvt& b0, gjvt& b1){
        static ll i;
        static T dt;
        for(i=ma; i--;){
            dt = a0[i];
            a0[i] = a1[i];
            a1[i] = dt;
        }
        for(i=mb; i--;){
            dt = b0[i];
            b0[i] = b1[i];
            b1[i] = dt;
        }
        d *= -1; // added wrt notebook
        // cout << "Swap\n";
        // print();
    }
    T normalize(gjvt& a0, gjvt& b0, ll c){
        static T k;
        k = (T)1/a0[c];
        d *= k; // added wrt notebook
        a0[c] = 1;
        while((++c)<ma){
            a0[c] *= k;
        }
        for(c=mb; c--;){
            b0[c] *= k;
        }
        // cout << "Normalize\n";
        // print();
        return k;
    }
    void reduce(gjvt& a0, gjvt& a1, gjvt& b0, gjvt& b1, ll c)
        ↪ {
        static T k;
        if(is_zero(a1[c])){
            return;
        }
        k = a1[c]; // k = a1[c]/a0[c]; // a0[c] == 1
        a1[c] = 0;
        // d *= k;
        while((++c)<ma){
            a1[c] -= a0[c]*k;
        }
        for(c=mb; c--;){
            b1[c] -= b0[c]*k;
        }
        // cout << "Reduce\n";
        // print();
    }
public:
    GaussJordan(gjvvt& _A, gjvvt& _B, dd _eps =
        ↪ __GAUSS_JORDAN_EPS, function<dd(const T&)> _norm =
        ↪  def_norm, ll ABrows = 0, ll Acols = 0, ll Bcols =
        ↪  0){
        A = &_A;
        B = &_B;
        if(ABrows){
            n = ABrows;
        }else{
            n = a.size();
        }
        if(Acols){
            ma = Acols;
        }else{
            ma = a[0].size();
        }
        if(Bcols){
            mb = Bcols;
        }else{
            mb = b[0].size();
        }
        eps = _eps;
        norm = _norm;
        // print();
    }
    ll solve(){
        // the absolute value of the returned value is the
            ↪ number of columns of A minus the rank of A
        // in case of a linear system, the returned value is:
            // positive if the system has multiple solutions,
                ↪ and it equals the number of "free"
                ↪ variables
            // negative if the system is not solvable
            // zero if the system has exactly one soluton (
                ↪ stored in b[:][0])
        ll c, i, u = 0, p;
        T pa, pt;
        for(r=c=0 ; r<n && c<ma ; r++, c++){
            for(; c<ma; c++){
                p = r;
                pa = norm(a[p][c]);
                for(i=r+1; i<n; i++){
                    if((pt = norm(a[i][c])) > pa){
                        p = i;
                        pa = pt;
                    }
                }
                if(!is_zero(a[p][c])) break;
            }
            if(c==ma) break;
            if(p!=r){
                swap(a[r], a[p], b[r], b[p]);
            }
```

```
            normalize(a[r], b[r], c);
            for(i=0; i<n; i++) if(r!=i){
                reduce(a[r], a[i], b[r], b[i], c);
            }
        }
        if(ma == r){ // added wrt notebook
            d = (T)1/d;
        }else{
            d = 0;
        }
        if(mb){
            for(i=r; i<n; i++){
                if(!is_zero(b[i][0])){
                    u--;
                }
            }
            if(u) return u;
        }
        return ma-r;
    }
#undef a
#undef b
};
```

Dijkstra.cpp

```
template<class T, class W> class Dijkstra{
// T: type of edge
// W: type of weight
#define g (*pg)
public:
    typedef long long ll;
    typedef std::vector<W> vw;
    typedef std::vector<T> vt;
    typedef std::vector<vt> vvt;
    typedef std::pair<W,ll> pwl;
    typedef std::priority_queue<pwl> pqt;
    typedef std::function<ll(const T&)> fetnt;
    typedef std::function<W(const T&)> fetwt;
    static ll def_etn(const T& e){
        return e.first;
    }
    static W def_etw(const T& e){
        return e.second;
    }
    vw dist;
    ll reached;
private:
    ll n, s, t;
    const vvt* pg;
    fetnt etn;
    fetwt etw;
public:
    Dijkstra(){}
    Dijkstra(
        const vvt& graph,
        ll source = 0,
        ll destination = -1,
        ll _v = -1,
        W inf_w = __DIJKSTRA_INF,
        W timeout = __DIJKSTRA_INF,
        fetnt edge_to_node = def_etn,
        fetwt edge_to_weight = def_etw
    ){
        pqt pq;
        ll u, v;
        W w, tw;
        pg = &graph;
        s = source;
        t = destination;
        if(_v == -1) n = g.size();
        else n = _v;
        etn = edge_to_node;
        etw = edge_to_weight;
        reached = 0;
        dist.assign(n, inf_w);
        dist[s] = 0;
        pq.push(pwl(0, s));
        while(!pq.empty()){
            w = -pq.top().first;
            if(w > timeout) break;
            u = pq.top().second;
            pq.pop();
            if(dist[u] < w) continue;
            reached++;
            if(u == t) break;
            for(const T& ed : g[u]){
                v = etn(ed);
                tw = etw(ed);
                if(dist[v] > dist[u] + tw){
                    dist[v] = dist[u] + tw;
                    pq.push(pwl(-dist[v], v));
                }
            }
        }
    }
#undef g
};
```

Dinic.cpp

```
#define __DINIC_FLOW_MAX (T)1000000000000000
#define __dinic_eu first.first
#define __dinic_ev first.second
#define __dinic_ec second

template <class T> class Dinic{
public:
    typedef long long ll;
    typedef pair<ll,ll> pl;
    typedef array<ll,3> dinic_node;
    typedef pair<pl,T> dinic_edge;
    T total_flow = 0;
    vector<T> rf;
    vector<bool> in_S;
    vector<ll> min_cut_edges;
private:
    enum{
        visited, deleted, level
    };
    vector<dinic_node> v;
    vector<dinic_edge> e;
    vector<vector<pl>> g;
    ll n, m, m2, s, t, i, r;
    T fm;
    static T flow_min(const T& x, const T& y){
        if(x<y) return x;
        return y;
    }
    void dinic_bfs(){
        queue<ll> q;
        ll x, y;
        v[s][visited] = r;
        q.push(s);
        while(!q.empty()){
            x = q.front();
            q.pop();
            if(x == t) continue;
            for(const pl& yy : g[x]){
                y = yy.first;
                if(rf[yy.second] != (T)0 && v[y][visited] < r){
                    v[y][level] = v[x][level] + 1;
                    v[y][visited] = r;
                    q.push(y);
                }
            }
        }
    }
    T dinic_dfs(ll x, T f){
        ll y, z;
        T ret = 0, tmp = 0;
        if(x == t) return f;
        for(const pl& yy : g[x]){
            if(f == (T)0) break;
            y = yy.first;
            z = yy.second;
            if(rf[z]!=(T)0 && v[y][deleted]<r && v[y][level]==
                ↪ v[x][level]+1){
                tmp = dinic_dfs(y, flow_min(f, rf[z]));
                rf[z] -= tmp;
                rf[(z<m)?(z+m):(z-m)] += tmp;
                f -= tmp;
                ret += tmp;
```

```
            }
        }
        if(ret != (T)0){
            v[x][deleted] = r;
        }
        return ret;
    }
    T step(){
        r++;
        dinic_bfs();
        return dinic_dfs(s, fm);
    }
public:
    Dinic(ll _v, ll _e, const vector<dinic_edge>& edges, ll
        ↪ source = 0, ll sink = -1, const T& flow_max =
        ↪ __DINIC_FLOW_MAX){
        n = _v;
        m = _e;
        m2 = m<<1;
        e = edges;
        e.resize(m2);
        for(i=m; i--;){
            e[i+m].__dinic_eu = e[i].__dinic_ev;
            e[i+m].__dinic_ev = e[i].__dinic_eu;
        }
        s = source;
        if(sink == -1){
            t = n-1;
        }else{
            t = sink;
        }
        fm = flow_max;
        r = 0;
        total_flow = 0;
        v.assign(n, {0,0,0});
        rf.reserve(m2);
        g.assign(n, vector<pl>());
        for(i=0; i<m; i++){
            rf[i] = e[i].__dinic_ec;
            g[e[i].__dinic_eu].push_back(pl(e[i].__dinic_ev, i
                ↪ ));
        }
        for(i=m; i<m2; i++){
            rf[i] = 0;
            g[e[i].__dinic_eu].push_back(pl(e[i].__dinic_ev, i
                ↪ ));
        }
    }
    void execute(){
        T ret;
        do{
            ret = step();
            total_flow += ret;
        }while(ret != (T)0);
    }
    ll min_cut(){
        ll x;
        vector<ll> q;
        ll qb;
        in_S.assign(n, false);
        in_S[s] = true;
        q.push_back(s);
        for(qb=0; qb<q.size(); qb++){
            x = q[qb];
            for(const pl& yy : g[x]){
                if(rf[yy.second]){
                    if(!in_S[yy.first]){
                        in_S[yy.first] = true;
                        q.push_back(yy.first);
                    }
                }
            }
        }
        for(ll x : q){
            for(const pl& yy : g[x]){
                if(!in_S[yy.first] && yy.second < m){
                    min_cut_edges.push_back(yy.second);
                }
            }
        }
        return q.size();
    }
};
```

DSU.cpp

```
class DSU{
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
    ll cc;
private:
    ll n;
    vl id, rk, sz;
public:
    DSU(){}
    DSU(ll n){
        static ll i;
        cc = n;
        rk.assign(n, 0);
        sz.assign(n, 1);
        id.resize(n);
        for(i=n; i--;){
            id[i] = i;
        }
    }
    ll findSet(ll i){
        if(id[i] == i) return i;
        return id[i] = findSet(id[i]);
    }
    bool isSameSet(ll i, ll j){
        return findSet(i) == findSet(j);
    }
    ll unionSet(ll i, ll j){
        static ll x, y;
        x = findSet(i);
        y = findSet(j);
        if(x != y){
            cc--;
            if(rk[x] < rk[y]){
                sz[y] += sz[x];
                return id[x] = y;
            }else{
                sz[x] += sz[y];
                if(rk[x] == rk[y]) rk[x]++;
                return id[y] = x;
            }
        }
        return x;
    }
    ll getSize(ll i){
        return sz[findSet(i)];
    }
};
```

ExpressionEvaluator.cpp

```
template <class T> class ExpressionEvaluator{
private:
    ll pos = 0;
    ll errpos = -1;
    ll errcode = 0;
    ll explen;
    string exp;
    string ops;
    string opars = "([{";
    string cpars = ")]}";
    ll npars[4] = {};
    bool sgn;
    bool chk_par;
    stack<ll> parsht;
    stack<ll> parshp;
    function<T(const string&, size_t*)> stoT;
    static T def_stoT(const string& str, size_t* idx){
        return stod(str, idx);
    }
    static bool is_mul(char c){
        return c=='*' || c=='/';
    }
    bool is_op(char c){
        return ops.find(c) != string::npos;
    }
    bool is_cpar(char c){
```

```
    return cpars.find(c) != string::npos;
}
void divide(T& res, T& v){
    if(v == 0){
        if(res != 0){
            errcode = 2;
        }else if(!errcode){
            errcode = 3;
        }
        res = 0;
    }else{
        res /= v;
    }
}
ll operate(T& res, T& v, char op){
    if(op == '+'){
        res += v;
    }else if(op == '-'){
        res -= v;
    }else if(op == '*'){
        res *= v;
    }else if(op == '/'){
        divide(res, v);
    }else{
        errpos = pos;
        return 1;
    }
    v = 0;
    return 0;
}
ll read_number(T& n){
    sll i;
    size_t idx;
    for(i=pos; pos<explen && !is_op(exp[pos]); pos++);
    try{
        n = stoT(exp.substr(i, pos-i), &idx);
    }catch(const invalid_argument& e){
        errpos = i;
        return 1;
    }catch(const out_of_range& e){
        errpos = i;
        return 1;
    }
    if(pos==i || i+idx!=pos){
        errpos = pos;
        return 1;
    }
    pos--;
    return 0;
}
ll evaluate_r(T& res, bool mul, char op, ll& nel){
    ll par, i, cnel;
    char lop = 0;
    T v = 0;
    if(!mul){
        op = '+';
        if(sgn){
            res = 0;
            if(exp[pos] == '-'){
                op = '-';
                pos++;
                nel = 1;
            }else if(exp[pos] == '+'){
                pos++;
            }
        }
    }
    for(; pos<explen; pos++, nel++){
        if((par = opars.find(exp[pos])) != string::npos){
            if(!op){
                errpos = pos;
                return 1;
            }
            if(par) for(i=0; i<=par; i++){
                if(npars[i]){
                    errpos = pos;
                    return 1;
                }
            }
            npars[par]++;
            parsht.push(par);
            parshp.push(pos++);
            v = 0;
            cnel = 0;
            if(evaluate_r(v, false, 0, cnel)){
                return 1;
            }
            lop = op;
            op = 0;
        }else if((par = cpars.find(exp[pos])) != string::
            ↪ npos){
            if(op || parsht.empty() || parshp.top() == pos
                ↪ -1){
                errpos = pos;
                return 1;
            }
            if(parsht.top() != par){
                errpos = pos;
                return 1;
            }
            if(chk_par && nel<2){
                errpos = pos;
                return 1;
            }
            if(!mul){
                npars[par]--;
                parsht.pop();
                parshp.pop();
            }
            if(operate(res, v, lop)){
                return 1;
            }
            return 0;
        }else if(op){
            if(read_number(v)){
                return 1;
            }
            lop = op;
            op = 0;
        }else{
            op = exp[pos];
            if(is_mul(op)){
                if(!mul){
                    pos++;
                    if(evaluate_r(v, true, op, nel)){
                        return 1;
                    }
                    op = exp[pos];
                    if(is_cpar(op)){
                        op = 0;
                        pos--;
                    }
                }
                if(operate(res, v, lop)){
                    return 1;
                }
            }else{
                if(operate(res, v, lop)){
                    return 1;
                }
                if(mul){
                    return 0;
                }
            }
        }
    }
    if(operate(res, v, lop)){
        return 1;
    }
    return 0;
}
public:
    ExpressionEvaluator(const string& expression, bool
        ↪ signed_numbers = true, bool check_parenthesis =
        ↪ false, function<T(const std::string&, size_t*)>
        ↪ string_to_T = def_stoT, const string& operators =
        ↪ "+-*/"){
        exp = expression;
        explen = exp.size();
        sgn = signed_numbers;
        chk_par = check_parenthesis;
        ops = operators + opars + cpars;
        stoT = string_to_T;
    }
    ll evaluate(T& ans){
```

```
        ll cnel = 0;
        parshp = {};
        parsht = {};
        parshp.push(-2);
        if(evaluate_r(ans, false, 0, cnel) || parshp.size() !=
            ↪ 1){
            return 1;
        }
        return errcode;
    }
    ll get_errpos(){
        return errpos;
    }
};
```

extended_euclidean-catalan.cpp

```
typedef long long ll;
#define mod 1000000007

// implemented from https://en.wikipedia.org/wiki/
    ↪ Extended_Euclidean_algorithm#Pseudocode
#define ee_refresh(o,n) tmp=n; n=o-q*n; o=tmp
void extended_euclidean(ll a, ll b){
    ll old_r=a, r=b, old_s=1, s=0, old_t=0, t=1, tmp, q;
    while(r){
        q = old_r / r;
        ee_refresh(old_r, r);
        ee_refresh(old_s, s);
        ee_refresh(old_t, t);
    }
    // output "Bezout coefficients:", (old_s, old_t)
    // output "greatest common divisor:", old_r
    // output "quotients by the gcd:", (t, s)
}

// implemented from https://en.wikipedia.org/wiki/
    ↪ Extended_Euclidean_algorithm#Modular_integers
// #define ee_refresh(o,n) tmp=n; n=o-q*n; o=tmp
ll inv(ll a, ll m){
    ll t = 0, newt = 1, r = m, newr = a, tmp, q;
    while(newr != 0){
        q = r / newr;
        ee_refresh(t, newt);
        ee_refresh(r, newr);
    }
    // if(r > 1)
    // return "a is not invertible";
    if (t < 0)
        t = t + m;
    return t;
}
```

```
// formula taken from https://en.wikipedia.org/wiki/
    ↪ Catalan_number#Properties
#define maxc 1000
ll catalan[maxc+1];
#define precompute_catalan()\
    catalan[0] = 1;\
    for(ll i=0; i<maxc; i++){\
        catalan[i+1] = (((catalan[i]<<1)*((i<<1)+1))%mod * (
            ↪ inv(i+2, mod)))%mod;\
    }
```

Factorization.cpp

```
template <class T> class Factorization{
public:
    typedef long long ll;
    typedef std::pair<T,ll> pTl;
    typedef std::vector<pTl> vpTl;
    static ll factorize(vpTl& res, T n){
        ll x;
        T r, i;
        res.resize(0);
        if(n == (T)0){
            res.push_back(pTl((T)0, 1));
            return 1;
        }
        if(n<(T)0){
            res.push_back(pTl((T)-1,1));
            n = -n;
        }
        if(n == (T)1){
            res.push_back(pTl((T)1, 1));
            return res.size();
        }
        for(i=(T)2, r=(T)sqrt((double)n) ; i<=r ; i+=((i==(T)
            ↪ 2)?(T)1:(T)2) ){
            for(x=0 ; n%i==(T)0 ; n/=i, x++);
            if(x){
                res.push_back(pTl(i, x));
                r = (T)sqrt((double)n);
            }
        }
        if(n!=(T)1){
            res.push_back(pTl(n, 1));
        }
        return res.size();
    }
};
```

FFTit.cpp

```
template <class T, class F> class FFTit{
public:
    typedef long long int ll;
    typedef complex<F> cf;
    typedef vector<cf> vcf;
    typedef vector<T> vt;
    static constexpr F pi = acos(-1);
private:
    static T fft_round(const F& f){
        return f+(F)(0.5);
    }
    static void fft_uw(vcf& v, ll n, bool inverse = false){
        #define a (*pa)
        #define b (*pb)
        // assuming n == pow(2,k)
        vcf u, *pa, *pb, *pt;
        u.reserve(v.size());
        pa = &v;
        pb = &u;
        ll n2 = n>>1, c=0, i, j, d, x;
        cf w, dw;
        F t;
        for(d=n2, t=pi*(F)(inverse?-1:1) ; d ; d>>=1, t/=(F)2)
            ↪ {
            dw = cf(cos((double)t), sin((double)t));
            for(i=0; i<d; i++){
                for(j=i, w=cf((F)1), x=i ; j<n ; j+=(d<<1), w*=
                    ↪ dw, x+=d){
                    b[x] = a[j] + w*a[j+d];
                    b[x+n2] = a[j] - w*a[j+d];
                }
            }
            pt = pa;
            pa = pb;
            pb = pt;
            c++;
        }
        if(c&1){
            for(i=n; i--; ){
                v[i] = u[i];
            }
        }
        if(inverse){
            for(i=n; i--;){
                v[i] /= (F)n;
            }
        }
        #undef a
        #undef b
    }
public:
    static ll fft(vcf& res, const vt& p, ll n = -1){
        ll m;
        if(n == -1) n = p.size();
        for(m=1; m<n; m<<=1);
        res = vcf(p.begin(), p.end());
```

```
        res.resize(m, (F)0);
        fft_uw(res, m, false);
        return m;
    }
    static ll ifft(vt& res, vcf p, function<T(const F&)> ftot
        ↪  = fft_round, ll n = -1){
        ll m, i;
        if(n == -1) n = p.size();
        for(m=1; m<n; m<<=1);
        p.resize(m, (F)0);
        fft_uw(p, m, true);
        res.resize(m);
        for(i=m; i--;){
            res[i] = ftot(p[i].real());
        }
        return m;
    }
    static ll multiply(vt& res, const vt& p, const vt& q,
        ↪ function<T(const F&)> ftot = fft_round){
        vcf a, b;
        ll i, n, m = p.size() + q.size();
        for(n=1; n<m; n<<=1);
        fft(a, p, n);
        fft(b, q, n);
        for(i=0; i<n; i++){
            a[i] *= b[i];
        }
        ifft(res, a, ftot, n);
        return n;
    }
};
```

FordFulkersonDijkstra.cpp

```
#define __FFD_FLOW_MAX (T)1000000000000000
#define __ffd_eu first.first
#define __ffd_ev first.second
#define __ffd_ec second
#define __ffd_nflow first
#define __ffd_nvisited second.first
#define __ffd_nparent second.second
template <class T> class FFD{
public:
    typedef long long ll;
    typedef pair<ll,ll> pl;
    typedef pair<T,pl> ffd_node;
    typedef pair<pl,T> ffd_edge;
    typedef pair<T,ll> ptl;
    T total_flow = 0;
    vector<T> rf;
private:
    ll n, m, m2, s, t, i, r = 0;
    vector<ffd_node> v;
    vector<ffd_edge> e;
    vector<vector<pl>> g;
    T fm = 0;
    T vis(){
        priority_queue<ptl> q;
        ll x, y;
        T f = 0, tmp = 0;
        v[s].__ffd_nvisited = ++r;
        q.push(ptl(fm, s));
        while(!q.empty()){
            f = q.top().first;
            x = q.top().second;
            if(x == t){
                while(x != s){
                    y = v[x].__ffd_nparent;
                    rf[y] -= f;
                    rf[(y<m)?(y+m):(y-m)] += f;
                    x = e[y].__ffd_eu;
                }
                return f;
            }
            q.pop();
            if(v[x].__ffd_nflow != f) continue;
            for(pl yy : g[x]){
                y = yy.first;
                tmp = min(f,rf[yy.second]);
                if(rf[yy.second]!=(T)0 && ( v[y].__ffd_nvisited
                    ↪ <r || v[y].__ffd_nflow<tmp )){
                    v[y].__ffd_nvisited = r;
                    v[y].__ffd_nparent = yy.second;
                    v[y].__ffd_nflow = tmp;
                    q.push(ptl(tmp, y));
                }
            }
        }
        return (T)0;
    }
public:
    FFD(ll _v, ll _e, const vector<ffd_edge>& edges, ll
        ↪ source = 0, ll sink = -1, const T& flow_max =
        ↪ __FFD_FLOW_MAX){
        n = _v;
        m = _e;
        m2 = m<<1;
        e = edges;
        e.resize(m2);
        for(i=m; i--;){
            e[i+m].__ffd_eu = e[i].__ffd_ev;
            e[i+m].__ffd_ev = e[i].__ffd_eu;
        }
        s = source;
        if(sink == -1){
            t = n-1;
        }else{
            t = sink;
        }
        fm = flow_max;
        v.assign(n, ffd_node((T)0, pl(0,0)));
        v[s].__ffd_nflow = fm;
        rf.resize(m2);
        g.assign(n, vector<pl>());
        for(i=0; i<m; i++){
            rf[i] = e[i].__ffd_ec;
            g[e[i].__ffd_eu].push_back(pl(e[i].__ffd_ev, i));
        }
        for(i=m; i<m2; i++){
            rf[i] = 0;
            g[e[i].__ffd_eu].push_back(pl(e[i].__ffd_ev, i));
        }
    };
    void execute(){
        T ret;
        while((ret = vis()) != (T)0){
            total_flow += ret;
        }
    }
};
```

Fraction.cpp

```
char __FRACTION_SEPARATOR = '|';
template <class T> class Fraction{
private:
    static T gcd(T a, T b){
        static T r;
        if(a<0) a = -a;
        if(b<0) b = -b;
        if(!b){
            if(!a){
                return 1;
            }
            return a;
        }
        r = a%b;
        while(r){
            a = b;
            b = r;
            r = a%b;
        }
        return b;
    }
public:
    T num, den;
    char sep = __FRACTION_SEPARATOR;
    Fraction<T>(T numerator = (T)0, T denominator = (T)1,
        ↪ char separator = __FRACTION_SEPARATOR){
        sep = separator;
        T g = gcd(numerator, denominator);
        num = numerator/g;
        den = denominator/g;
```

```
    if(den < 0){
        num = -num;
        den = -den;
    }
}
Fraction<T>(const string& str, size_t* idx = NULL, char
    ↪ separator = __FRACTION_SEPARATOR){
    T n, d;
    size_t p, q;
    n = stoll(str, &p);
    if(str[p] == separator){
        d = stoll(str.substr(p+1), &q);
        *this = Fraction<T>(n, d, separator);
        p += q+1;
    }else{
        *this = Fraction<T>(n, (T)1, separator);
    }
    if(idx != NULL){
        *idx = p;
    }
}
string to_string() const{
    stringstream ss;
    ss << num;
    if(den != 1){
        ss << sep << den;
    }
    return ss.str();
}
Fraction<T> operator+(const Fraction<T>& x) const{
    T n,d;
    if(den && x.den){
        d = den*x.den / gcd(den, x.den);
        n = d/den*num + d/x.den*x.num;
        return Fraction<T>(n,d,sep);
    }
    return Fraction<T>((T)1, (T)0, sep);
}
Fraction<T> operator-(const Fraction<T>& x) const{
    Fraction<T> b;
    b.num = -x.num;
    b.den = x.den;
    return (*this)+b;
}
Fraction<T> operator*(const Fraction<T>& x) const{
    Fraction<T> b;
    T g;
    g = gcd(num, x.den);
    b.num = num / g;
    b.den = x.den / g;
    g = gcd(den, x.num);
    b.num *= x.num / g;
    b.den *= den / g;
    return b;
}
Fraction<T> operator/(Fraction<T> x) const{
    T t = x.num;
    x.num = x.den;
    x.den = t;
    if(x.den<0){
        x.num = -x.num;
        x.den = -x.den;
    }
    return (*this)*x;
}
bool operator==(const Fraction<T>& x) const{
    return num == x.num && den == x.den;
}
bool operator<(const Fraction<T>& x) const{
    T a = num, b = den, c = x.num, d = x.den, g;
    g = gcd(a, c);
    a /= g;
    c /= g;
    g = gcd(b, d);
    b /= g;
    d /= g;
    return a*d < b*c;
}
bool operator>(const Fraction<T>& x) const{
    return x<(*this);
}
bool operator<=(const Fraction<T>& x) const{
    return (*this)==x || (*this)<x;
}
bool operator>=(const Fraction<T>& x) const{
    return (*this)==x || (*this)>x;
}
bool operator!=(const Fraction<T>& x) const{
    return !((*this)==x);
}
void operator+=(const Fraction<T>& x){
    (*this) = (*this)+x;
}
void operator-=(const Fraction<T>& x){
    (*this) = (*this)-x;
}
void operator*=(const Fraction<T>& x){
    (*this) = (*this)*x;
}
void operator/=(const Fraction<T>& x){
    (*this) = (*this)/x;
}
/*
int operator<=>(const Fraction<T>& x) const{
    if((*this)==x){
        return 0;
    }else if((*this)<x){
        return -1;
    }else{
        return 1;
    }
}
*/
};
template <class T> Fraction<T> stofrac(const string& str,
    ↪ size_t* idx, char separator){
    return Fraction<T>(str, idx, separator);
}
template <class T> string to_string(const Fraction<T>& x){
    return x.to_string();
}
template <class T> istream& operator>>(istream& is, Fraction<
    ↪ T>& x){
    string s;
    is >> s;
    x = Fraction<T>(s, NULL, x.sep);
    return is;
}
template <class T> ostream& operator<<(ostream& os, const
    ↪ Fraction<T>& x){
    os << x.to_string();
    return os;
}
```

GaussJordan.cpp

```
#define __GAUSS_JORDAN_EPS 10e-9
template <class T> class GaussJordan{
#define a (*A)
#define b (*B)
public:
    typedef long long ll;
    typedef double dd;
    typedef vector<T> gjvt;
    typedef vector<gjvt> gjvvt;
    static dd def_norm(const T& t){
        if(t<0) return -t;
        return t;
    }
    ll r;
    T d = 1;
    gjvvt* A;
    gjvvt* B;
private:
    ll n, ma, mb;
    dd eps;
    function <dd(const T&)> norm;
    inline bool is_zero(T& x){
        static T zero = (T)(0);
        if(norm(x) <= eps){
            x = zero;
            return true;
        }
        return false;
```

```
}
void print(){
    ll i, j;
    for(i=0; i<n; i++){
        for(j=0; j<ma; j++){
            cout << a[i][j] << '\t';
        }
        cout << "|\t";
        for(j=0; j<mb; j++){
            cout << b[i][j] << '\t';
        }
        cout << '\n';
    }
    cout << '\n';
}
void swap(gjvt& a0, gjvt& a1, gjvt& b0, gjvt& b1){
    static ll i;
    static T dt;
    for(i=ma; i--;){
        dt = a0[i];
        a0[i] = a1[i];
        a1[i] = dt;
    }
    for(i=mb; i--;){
        dt = b0[i];
        b0[i] = b1[i];
        b1[i] = dt;
    }
}
T normalize(gjvt& a0, gjvt& b0, ll c){
    static T k;
    k = (T)1/a0[c];
    a0[c] = 1;
    while((++c)<ma){
        a0[c] *= k;
    }
    for(c=mb; c--;){
        b0[c] *= k;
    }
    return k;
}
void reduce(gjvt& a0, gjvt& a1, gjvt& b0, gjvt& b1, ll c)
    ↪ {
    static T k;
    if(is_zero(a1[c])){
        return;
    }
    k = a1[c]; // k = a1[c]/a0[c]; // a0[c] == 1
    a1[c] = 0;
    while((++c)<ma){
        a1[c] -= a0[c]*k;
    }
    for(c=mb; c--;){
        b1[c] -= b0[c]*k;
    }
}
public:
    GaussJordan(gjvvt& _A, gjvvt& _B, dd _eps =
        ↪ __GAUSS_JORDAN_EPS, function<dd(const T&)> _norm =
        ↪ def_norm, ll ABrows = 0, ll Acols = 0, ll Bcols =
        ↪ 0){
        A = &_A;
        B = &_B;
        if(ABrows){
            n = ABrows;
        }else{
            n = a.size();
        }
        if(Acols){
            ma = Acols;
        }else{
            ma = a[0].size();
        }
        if(Bcols){
            mb = Bcols;
        }else{
            mb = b[0].size();
        }
        eps = _eps;
        norm = _norm;
    }
    ll solve(){
        // the absolute value of the returned value is the
            ↪ number of columns of A minus the rank of A
        // in case of a linear system, the returned value is:
            // positive if the system has multiple solutions,
                ↪ and it equals the number of "free"
                ↪ variables
            // negative if the system is not solvable
            // zero if the system has exactly one soluton (
                ↪ stored in b[:][0])
        ll c, i, u = 0, p;
        T pa, pt;
        for(r=c=0 ; r<n && c<ma ; r++, c++){
            for(; c<ma; c++){
                p = r;
                pa = norm(a[p][c]);
                for(i=r+1; i<n; i++){
                    if((pt = norm(a[i][c])) > pa){
                        p = i;
                        pa = pt;
                    }
                }
                if(!is_zero(a[p][c])) break;
            }
            if(c==ma) break;
            if(p!=r){
                swap(a[r], a[p], b[r], b[p]);
            }
            normalize(a[r], b[r], c);
            for(i=0; i<n; i++) if(r!=i){
                reduce(a[r], a[i], b[r], b[i], c);
            }
        }
        d = (T)1/d;
        if(mb){
            for(i=r; i<n; i++){
                if(!is_zero(b[i][0])){
                    u--;
                }
            }
            if(u) return u;
        }
        return ma-r;
    }
#undef a
#undef b
};
```

geometry.cpp

```
#define EPS (dbl)1e-9
#define PI (dbl)(acos(-1.0))


////////////////////////////////////////
//////// OGGETTI GEOMETRICI ////////
////////////////////////////////////////


/* punti, note:
    - punti inizializzati di default a (0,0)
    - per inizializzare si puo' usare normalmente p.x = tmpx;
        ↪ p.y = tmpy o
      anche p(tmpx,tmpy);
    - overload del bool op. < per sort
    - overload del bool op. == per controlli di uguaglianza,
        ↪ per usare !=
      usare !(p1 == p2)            */

struct point {
    dbl x,y;
    point() {
        x=y=0.0;
    }
    point(dbl qx,dbl qy) : x(qx),y(qy) {}

    bool operator < (point other) const {
        if(fabs(x - other.x) > EPS)
            return (x < other.x);
        return (y < other.y);
    }

    bool operator == (point other) const {
```

```
        return (fabs(x - other.x) < EPS && (fabs(y - other.y)
            ↪ < EPS) );
    }
};


// retta: a,b,c coefficienti equazione implicita
struct line {
    dbl a,b,c;
};


// vettore
struct vec {
    dbl x,y;
    vec(dbl _x, dbl _y) : x(_x),y(_y) {}
};


/* !!! IMPORTANTE !!!: rappresentare i poligoni come vector
    ↪ di punti, con
                    con primo punto che si ripete anche
                        ↪ come ultimo. E.g.:

    vector <point> polygon;
    polygon.pb(p1);
    polygon.pb(p2);
    polygon.pb(p3);
    polygon.pb(p4);
        ...
    polygon.pb(pn);   // p1,p2,...,pn sono i punti dei
        ↪ vertici del poligono
    polygon.pb(polygon[0]);

*/

////////////////////////////////////////
////////// FUNZIONI UTILI //////////
////////////////////////////////////////


// distanza euclidea tra punti
dbl dist(point p1,point p2) {
    return sqrt( (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y
        ↪ ) * (p1.y - p2.y) );
}


// distanza di Manhattan tra punti
dbl manh_dist(point p1,point p2) {
    return ( fabs(p1.x - p2.x) + fabs(p1.y - p2.y) );
}


// rotazione punti ccw, wrt (0,0)
point rotate(point p,dbl theta) {
    dbl rad = theta * M_PI / 180.0; //se l'angolo e' gia' in
        ↪ rad cancellare questa linea
    return point( (p.x * cos(rad) - p.y * sin(rad) ),(p.x *
        ↪ sin(rad) + p.y * cos(rad) ) );
}


// retta passante per due punti
line line_2points(point p1,point p2) {
    dbl p,q,r;

    if(fabs(p1.x - p2.x) < EPS) {
        p = 1.0; q = 0.0; r = -p1.x;
    } else {
        p = -(dbl)(p1.y - p2.y) / (p1.x - p2.x);
        q = 1.0;
        r = -(dbl)(p * p1.x) - p1.y;
    }

    line tmp = {p,q,r};
    return tmp;
}


// controlla se due rette sono parallele
bool are_parallel(line l1, line l2) {
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS)
        ↪ ;
}


// controlla se due rette sono uguagli
bool are_same(line l1, line l2) {
    return are_parallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS);
}


// trovare punto di intersezione
point find_intersection(line l1, line l2) {
    point p;
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a *
        ↪  l2.b);
    if(fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c);

    return p;
}


// controlla se un punto e' in una retta
bool is_point_in_line(point p,line l) {
    if((l.a * p.x + l.b * p.y + l.c) < EPS) return true;
    return false;
}


// crea vettore da a -> b
vec to_vec(point a,point b) {
    return vec(b.x - a.x, b.y - a.y);
}


// scalare * vettore (?)
vec scale(vec v,dbl s) {
    return vec(v.x * s, v.y * s);
}


// traslazione di un punto p con vettore v
point translate(point p,vec v) {
    return point(p.x + v.x, p.y + v.y);
}


// prodotto scalare
dbl dot(vec a, vec b) {
    return (a.x * b.x + a.y * b.y);
}


// quadrato del vettore
dbl norm_sq(vec a) {
    return (a.x * a.x + a.y * a.y);
}


// distanza punto - retta passante per a e b
dbl dist_to_line(point p, point a, point b, point &c) {
    vec ap = to_vec(a,p); vec ab = to_vec(a,b);
    dbl u = dot(ap,ab) / norm_sq(ab);
    c = translate(a,scale(ab,u));
    return dist(p,c);
}


// distanza punto - segmento ab
dbl dist_to_segment(point p, point a, point b, point &c) {
    vec ap = to_vec(a,p); vec ab = to_vec(a,b);
    dbl u = dot(ap,ab) / norm_sq(ab);

    if(u < 0.0) {
        c = point(a.x,a.y);
        return dist(p,a);
    }
    if(u > 1.0) {
        c = point(b.x,b.y);
        return dist(p,b);
    }
```

```
    return dist_to_line(p,a,b,c);
}


// ritorna angolo aob in radianti
double angle(point a, point o, point b) {
    vec oa = to_vec(o,a), ob = to_vec(o,b);
    return acos(dot(oa,ob) / sqrt(norm_sq(oa) * norm_sq(ob)))
        ↪ ;
}


// prodotto vettoriale
dbl cross(vec a, vec b) {
    return (a.x * b.y - a.y * b.x);
}


// controlla se un punto e' a sinistra di un segmento(?) non
    ↪ so come spiegarlo
bool ccw(point p, point q, point r) {
    return (cross(to_vec(p,q),to_vec(p,r)) > 0);
}


// controlla se tre punti si trovano sulla stessa retta
bool collinear(point p, point q, point r) {
    return (fabs(cross(to_vec(p,q),to_vec(p,r))) < EPS);
}


// controlla se un punto e' interno, sulla od esterno ad una
    ↪ circonferenza di
// centro c e raggio r, ritorna 0 se interno, 1 se sulla
    ↪ circonferenza, 2 se esterno
// per pi greco usare PI, definito sopra
int inside_circle(point p, point c, dbl r) {
    dbl dx = p.x - c.x, dy = p.y - c.y;
    dbl Euc = dx * dx + dy * dy, rSq = r * r;
    if(Euc < rSq) return 0;
    if(fabs(Euc - rSq) < EPS) return 1;
    return 2;
}


// perimetro triangolo
dbl triangle_perimeter(dbl ab, dbl bc, dbl ac) {
    return (ab + bc + ac);
}


// area triangolo a partire dai lati, per partire dai punti
    ↪ usare dist()
dbl triangle_area(dbl ab, dbl bc, dbl ac) {
    dbl s = triangle_perimeter(ab,bc,ac) / 2;
    return sqrt(s * (s - ab) * (s - bc) * (s - ac));
}


// raggio circonferenza inscritta
dbl r_incircle(dbl ab, dbl bc, dbl ac) {
    return triangle_area(ab,bc,ac) / (0.5 *
        ↪ triangle_perimeter(ab,bc,ac));
}


// controlla (e trova) l'esistenza della circonferenza
    ↪ inscritta?
// i triangoli non sono sempre inscrivibili?
// cp3 non e' molto chiaro a riguardo, onestamente non
    ↪ toccherei troppo
// l'implementazione, comunque passati p1, p2, p3, ctr, r in
    ↪ ctr salva il centro
// e in r il raggio e ritorna 1 se esiste(?)
int in_circle(point p1, point p2, point p3, point &ctr, dbl &
    ↪ r) {
    r = r_incircle(dist(p1,p2),dist(p2,p3),dist(p3,p1));
    if (fabs(r) < EPS) return 0;

    line l1, l2;
    dbl ratio = dist(p1, p2) / dist(p1, p3);
    point p = translate(p2, scale(to_vec(p2, p3), ratio / (1
        ↪ + ratio)));

    l1 = line_2points(p1, p);
    ratio = dist(p2, p1) / dist(p2, p3);
    p = translate(p1, scale(to_vec(p1, p3), ratio / (1 +
        ↪ ratio)));
    l2 = line_2points(p2, p);
    ctr = find_intersection(l1, l2);
    return 1;
}


// trova raggio circonferenza circoscritta al triangolo,
    ↪ passare dist() lati
dbl r_circumcircle(double ab, double bc, double ca) {
    return ab * bc * ca / (4.0 * triangle_area(ab, bc, ca));
}


// perimetro di un poligono
dbl perimeter(const vector <point> &P) {
    dbl result = 0.0;
    for(int i=0;i<sz(P)-1;i++) result += dist(P[i],P[i+1]);

    return result;
}


// area di un poligono
dbl area(const vector <point> &P) {
    dbl result=0.0,x1,x2,y1,y2;
    result = P[0].y*P.back().x - P[0].x*P.back().y; // bug
        ↪ fix wrt notebook
    for(int i=0;i<sz(P)-1;i++) {
        x1 = P[i].x; x2 = P[i+1].x;
        y1 = P[i].y; y2 = P[i+1].y;
        result += (x1 * y2 - x2 * y1);
    }

    return fabs(result) / 2.0;
}


// controlla se un poligono e' convesso, potrebbe dare
    ↪ problemi con lati collineari
bool is_convex(const vector <point> &P) {
    int s = sz(P);
    if(s <= 3) return false;
    bool is_left = ccw(P[0],P[1],P[2]);

    for(int i=1;i<s-1;i++)
        if((ccw(P[i], P[i+1], P[(i+2) == s ? 1 : i+2]) !=
            ↪ is_left)) return false;

    return true;
}


// controlla se un punto pt e' in un poligono P
bool in_polygon(point pt, const vector <point> &P) {
    if(!sz(P)) return false;


    dbl sum = 0;
    for(int i=0;i<sz(P)-1;i++) {
        if(ccw(pt,P[i],P[i+1])) sum += angle(P[i],pt,P[i+1]);
        else sum -= angle(P[i],pt,P[i+1]);
    }

    return (fabs(fabs(sum) - 2*PI) < EPS);
}


// punto di intersezione tra linea e segmento
point line_intersect_seg(point p, point q, point A, point B)
    ↪ {
    dbl a = B.y - A.y;
    dbl b = A.x - B.x;
    dbl c = B.x * A.y - A.x - B.y;
    dbl u = fabs(a * p.x + b * p.y + c);
    dbl v = fabs(a * q.x + b * q.y + c);
    return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y
        ↪ * u) / (u+v));
```

```
}


// poligono tagliato
vector <point> cut_polygon(point a, point b, const vector <
    ↪ point> &Q) {
    vector <point> P;
    for(int i=0;i<sz(Q);i++) {
        double left1 = cross(to_vec(a, b), to_vec(a, Q[i])),
            ↪ left2 = 0;

        if (i != sz(Q)-1) left2 = cross(to_vec(a, b), to_vec(a
            ↪ , Q[i+1]));
        if (left1 > -EPS) P.pb(Q[i]);
        if (left1 * left2 < -EPS)
            P.pb(line_intersect_seg(Q[i], Q[i+1], a, b));
    }

    if (!P.empty() && !(P.back() == P.front()))
        P.pb(P.front());

    return P;
}
```

HeavyLightDecomposition.cpp

Aggiungere...

Hungarian.cpp

```
// the following code is copied from https://www.topcoder.com
    ↪ /community/competitive-programming/tutorials/
    ↪ assignment-problem-and-hungarian-algorithm/ and
    ↪ modified

class HungarianMaxMatch{
public:
    typedef long long ll;
    typedef vector<ll> vl;
    typedef vector<vl> vvl;
    #define int ll
    #define N 500 //max number of vertices in one part
    // #define INF 100000000 //just infinity
    #define INF 1000000000000 //just infinity
    int cost[N][N]; //cost matrix
    int n, max_match; //n workers and n jobs
    int lx[N], ly[N]; //labels of X and Y parts
    int xy[N]; //xy[x] - vertex that is matched with x,
    int yx[N]; //yx[y] - vertex that is matched with y
    bool S[N], T[N]; //sets S and T in algorithm
    int slack[N]; //as in the algorithm description
    int slackx[N]; //slackx[y] such a vertex, that
// l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
    int prev[N]; //array for memorizing alternating paths
    int ww; //weight of the optimal matching

    void init_labels()
    {
        memset(lx, 0, sizeof(lx));
        memset(ly, 0, sizeof(ly));
        for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
        lx[x] = max(lx[x], cost[x][y]);
    }

    void augment() //main function of the algorithm
    {
        if (max_match == n) return; //check wether matching is
            ↪  already perfect
        int x, y, root; //just counters and root vertex
        int q[N], wr, rd; //q - queue for bfs, wr,rd - write
            ↪ and read
        //pos in queue
        wr = rd = 0;
        memset(S, false, sizeof(S)); //init set S
        memset(T, false, sizeof(T)); //init set T
        memset(prev, -1, sizeof(prev)); //init set prev - for
            ↪ the alternating tree
        for (x = 0; x < n; x++) //finding root of the tree
        if (xy[x] == -1)
        {
        q[wr++] = root = x;
        prev[x] = -2;
        S[x] = true;
        break;
        }
        for (y = 0; y < n; y++) //initializing slack array
        {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
        }

        //second part of augment() function
        while (true) //main cycle
        {
            while (rd < wr) //building tree with bfs cycle
            {
            x = q[rd++]; //current vertex from X part
            for (y = 0; y < n; y++) //iterate through all
                ↪ edges in equality graph
            if (cost[x][y] == lx[x] + ly[y] && !T[y])
            {
            if (yx[y] == -1) break; //an exposed vertex in Y
                ↪ found, so
            //augmenting path exists!
            T[y] = true; //else just add y to T,
            q[wr++] = yx[y]; //add vertex yx[y], which is
                ↪ matched
            //with y, to the queue
            add_to_tree(yx[y], x); //add edges (x,y) and (y,yx
                ↪ [y]) to the tree
            }
            if (y < n) break; //augmenting path found!
            }
            if (y < n) break; //augmenting path found!
            update_labels(); //augmenting path not found, so
                ↪ improve labeling
            wr = rd = 0;
            for (y = 0; y < n; y++)
            //in this cycle we add edges that were added to
                ↪ the equality graph as a
            //result of improving the labeling, we add edge (
                ↪ slackx[y], y) to the tree if
            //and only if !T[y] && slack[y] == 0, also with
                ↪ this edge we add another one
            //(y, yx[y]) or augment the matching, if y was
                ↪ exposed
            if (!T[y] && slack[y] == 0)
            {
            if (yx[y] == -1) //exposed vertex in Y found -
                ↪ augmenting path exists!
            {
            x = slackx[y];
            break;
            }
            else
            {
            T[y] = true; //else just add y to T,
            if (!S[yx[y]])
            {
            q[wr++] = yx[y]; //add vertex yx[y], which is
                ↪ matched with
            //y, to the queue
            add_to_tree(yx[y], slackx[y]); //and add edges (x,
                ↪ y) and (y,
            //yx[y]) to the tree
            }
            }
            }
            if (y < n) break; //augmenting path found!
        }
        if (y < n) //we found augmenting path!
        {
            max_match++; //increment matching
            //in this cycle we inverse edges along augmenting
                ↪ path
            for (int cx = x, cy = y, ty; cx != -2; cx = prev[
                ↪ cx], cy = ty)
            {
            ty = xy[cx];
            yx[cy] = cx;
            xy[cx] = cy;
```

```
        }
        augment(); //recall function, go to step 1 of the
            ↪ algorithm
    }
}//end of augment() function


void update_labels()
{
    int x, y, delta;
    delta = INF; //init delta as infinity
    for (y = 0; y < n; y++) //calculate delta using slack
    if (!T[y])
    delta = min(delta, slack[y]);
    for (x = 0; x < n; x++) //update X labels
    if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++) //update Y labels
    if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++) //update slack array
    if (!T[y])
    slack[y] -= delta;
}

void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in
    ↪ the alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
{
    S[x] = true; //add x to S
    prev[x] = prevx; //we need this when augmenting
    for (int y = 0; y < n; y++) //update slacks, because
        ↪ we add new vertex to S
    if (lx[x] + ly[y] - cost[x][y] < slack[y])
    {
    slack[y] = lx[x] + ly[y] - cost[x][y];
    slackx[y] = x;
    }
}

int hungarian()
{
    ww = 0;
    max_match = 0; //number of vertices in current
        ↪ matching
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels(); //step 0
    augment(); //steps 1-3
    for (int x = 0; x < n; x++) //forming answer there
    ww += cost[x][xy[x]];
    return ww;
}
HungarianMaxMatch(vvl& cost_matrix, ll rows, ll columns){
    ll i,j;
    n = max(rows, columns);
        for(i=columns; i-->rows;)
            for(j=columns; j--;)
                cost[i][j] = 0;
        for(i=rows; i--;){
            for(j=rows; j-->columns;)
                cost[i][j] = 0;
            for(j=columns; j--;)
                cost[i][j] = cost_matrix[i][j];
        }
        hungarian();
    }
    #undef int
    #undef N
    #undef INF
};
```

Hungarian-double.cpp

```
// the following code is copied from https://www.topcoder.com
    ↪ /community/competitive-programming/tutorials/
    ↪ assignment-problem-and-hungarian-algorithm/ and
    ↪ modified

class HungarianMaxMatch{
public:
    typedef long long ll;
    typedef double ct;
    typedef vector<ct> vc;
    typedef vector<vc> vvc;
    #define int ll
    #define N 500 //max number of vertices in one part
    // #define INF 100000000 //just infinity
    #define INF 1000000000000 //just infinity
    ct cost[N][N]; //cost matrix
    int n, max_match; //n workers and n jobs
    ct lx[N], ly[N]; //labels of X and Y parts
    int xy[N]; //xy[x] - vertex that is matched with x,
    int yx[N]; //yx[y] - vertex that is matched with y
    bool S[N], T[N]; //sets S and T in algorithm
    ct slack[N]; //as in the algorithm description
    int slackx[N]; //slackx[y] such a vertex, that
    // l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
    int prev[N]; //array for memorizing alternating paths
    ct ww; //weight of the optimal matching

    void init_labels()
    {
        memset(lx, 0, sizeof(lx));
        memset(ly, 0, sizeof(ly));
        for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
        lx[x] = max(lx[x], cost[x][y]);
    }

void augment() //main function of the algorithm
{
    if (max_match == n) return; //check wether matching is
        ↪ already perfect
    int x, y, root; //just counters and root vertex
    int q[N], wr, rd; //q - queue for bfs, wr,rd - write
        ↪ and read
    //pos in queue
    wr = rd = 0;
    memset(S, false, sizeof(S)); //init set S
    memset(T, false, sizeof(T)); //init set T
    memset(prev, -1, sizeof(prev)); //init set prev - for
        ↪ the alternating tree
    for (x = 0; x < n; x++) //finding root of the tree
    if (xy[x] == -1)
    {
        q[wr++] = root = x;
        prev[x] = -2;
        S[x] = true;
        break;
    }
    for (y = 0; y < n; y++) //initializing slack array
    {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }

    //second part of augment() function
    while (true) //main cycle
    {
        while (rd < wr) //building tree with bfs cycle
        {
        x = q[rd++]; //current vertex from X part
        for (y = 0; y < n; y++) //iterate through all
            ↪ edges in equality graph
        if (cost[x][y] == lx[x] + ly[y] && !T[y])
        {
        if (yx[y] == -1) break; //an exposed vertex in Y
            ↪ found, so
        //augmenting path exists!
        T[y] = true; //else just add y to T,
        q[wr++] = yx[y]; //add vertex yx[y], which is
            ↪ matched
        //with y, to the queue
        add_to_tree(yx[y], x); //add edges (x,y) and (y,yx
            ↪ [y]) to the tree
        }
        if (y < n) break; //augmenting path found!
        }
        if (y < n) break; //augmenting path found!
        update_labels(); //augmenting path not found, so
            ↪ improve labeling
        wr = rd = 0;
        for (y = 0; y < n; y++)
        //in this cycle we add edges that were added to
```

```
        ↪ the equality graph as a
        //result of improving the labeling, we add edge (
            ↪ slackx[y], y) to the tree if
        //and only if !T[y] && slack[y] == 0, also with
            ↪ this edge we add another one
        //(y, yx[y]) or augment the matching, if y was
            ↪ exposed
        if (!T[y] && slack[y] == 0)
        {
        if (yx[y] == -1) //exposed vertex in Y found -
            ↪ augmenting path exists!
        {
        x = slackx[y];
        break;
        }
        else
        {
        T[y] = true; //else just add y to T,
        if (!S[yx[y]])
        {
        q[wr++] = yx[y]; //add vertex yx[y], which is
            ↪ matched with
        //y, to the queue
        add_to_tree(yx[y], slackx[y]); //and add edges (x,
            ↪ y) and (y,
        //yx[y]) to the tree
        }
        }
        }
        if (y < n) break; //augmenting path found!
    }
    if (y < n) //we found augmenting path!
    {
        max_match++; //increment matching
        //in this cycle we inverse edges along augmenting
            ↪ path
        for (int cx = x, cy = y, ty; cx != -2; cx = prev[
            ↪ cx], cy = ty)
        {
        ty = xy[cx];
        yx[cy] = cx;
        xy[cx] = cy;
        }
        augment(); //recall function, go to step 1 of the
            ↪ algorithm
    }
}//end of augment() function


void update_labels()
{
    int x, y;
    ct delta;
    delta = INF; //init delta as infinity
    for (y = 0; y < n; y++) //calculate delta using slack
        if (!T[y])
        delta = min(delta, slack[y]);
    for (x = 0; x < n; x++) //update X labels
    if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++) //update Y labels
    if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++) //update slack array
    if (!T[y])
    slack[y] -= delta;
}

void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in
    ↪ the alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
{
    S[x] = true; //add x to S
    prev[x] = prevx; //we need this when augmenting
    for (int y = 0; y < n; y++) //update slacks, because
        ↪ we add new vertex to S
    if (lx[x] + ly[y] - cost[x][y] < slack[y])
    {
    slack[y] = lx[x] + ly[y] - cost[x][y];
    slackx[y] = x;
    }
}

int hungarian()
{
    ww = 0;
    max_match = 0; //number of vertices in current
        ↪ matching
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels(); //step 0
    augment(); //steps 1-3
    for (int x = 0; x < n; x++) //forming answer there
    ww += cost[x][xy[x]];
    return ww;
}
HungarianMaxMatch(vvc& cost_matrix, ll rows, ll columns){
    ll i,j;
    n = max(rows, columns);
    for(i=columns; i-->rows;)
        for(j=columns; j--;)
            cost[i][j] = 0;
    for(i=rows; i--;){
        for(j=rows; j-->columns;)
            cost[i][j] = 0;
        for(j=columns; j--;)
            cost[i][j] = cost_matrix[i][j];
    }
    hungarian();
}
#undef int
    #undef N
    #undef INF
};
```

IndexMap.cpp

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

typedef long long ll;
typedef ll K;
typedef ll V;

#define succ(x) ((x)+1)
#define prec(x) ((x)-1)
const K dummyk = -1;

ll accmod = 0;
K tofind = dummyk;
ll kfound = 0;
K imgk, imok;
vector<K> imh, imhi, imhf;

ll imkpos = 0;

typedef pair<pair<ll,ll>,V> IMV;
typedef map<K,IMV,bool(*)(K, K)> mymap;
mymap indexmap;

bool imcmp (K lhs, K rhs) {
    if(accmod == 0){
        return lhs < rhs;
    }
    if(accmod == 1){
        ll tk = lhs ^ rhs ^ imok;
        if(imh[imh.size()-1] != tk) imh.push_back(tk);
        if(kfound && tk!=tofind){
            imgk = tk;
            kfound = 0;
            tofind = dummyk;
        }else if(lhs == tofind || rhs == tofind){
            kfound = 1;
        }
        return ((lhs==imok)?imgk:lhs) < ((rhs==imok)?imgk:rhs)
            ↪ ;
    }
    ll tk = lhs ^ rhs ^ dummyk;
    if(imh[imh.size()-1] == tk){
        accmod = 0;
        imkpos += indexmap.find(tk)->second.first.first + 1;
        accmod = 2;
    }
    ll sl;
    imh.push_back(tk);
```

```
    accmod = 0;
    sl = indexmap.find(tk)->second.first.first;
    if(sl == imkpos){
        kfound = tk;
        return false;
    }
    accmod = 2;
    if(sl < imkpos){
        imkpos -= sl + 1;
        return rhs == dummyk;
    }
    return lhs == dummyk;
}


bool(*ptcmp)(K,K) = imcmp;
mymap imtmpm(ptcmp);
mymap::iterator node, nodec;

bool testc(K k, K kc){
    for(K x : imh){
        if(x == k) return true;
        if(x == kc) return false;
    }
    return false;
}

ll updwc(K k, K kc){
    imgk = imok = k;
    node = indexmap.find(k);
    // child
    imgk = imok = kc;
    tofind = k;
    kfound = 0;
    imh.clear();
    imh.push_back(dummyk);
    nodec = indexmap.find(imgk);
    imok = imgk;
    tofind = dummyk;
    kfound = 0;
    imh.clear();
    imh.push_back(dummyk);
    nodec = indexmap.find(imgk);
    tofind = dummyk;
    kfound = 0;
    if(nodec!=indexmap.end() && testc(k, nodec->first)){
        return nodec->second.first.second;
    }
    return 0;
}

void updw(K k){
    node->second.first.first = updwc(k, prec(k));
    node->second.first.second = 1 + node->second.first.first
        ↪ + updwc(k, succ(k));
}

}

void insert(K k, V v){
    accmod = 1;
    imh.clear();
    imh.push_back(dummyk);
    imgk = imok = k;
    indexmap[k] = IMV({{0,0},v});
    imhi = imh;
    imh.clear();
    imh.push_back(dummyk);
    imgk = imok = k;
    indexmap.find(k);
    imhf = imh;
    set<K> hs(imhf.begin(), imhf.end());
    for(ll i=imhi.size(); --i; ) if(!hs.count(imhi[i])){
        updw(imhi[i]);
    }
    for(ll i=imhf.size(); --i; ){
        updw(imhf[i]);
    }
    accmod = 0;
}

K iloc(ll pos){
    ll r;
    accmod = 2;
    imkpos = pos;
    imh.clear();
    imh.push_back(dummyk);
    indexmap.find(dummyk);
    accmod = 0;
    r = kfound;
    kfound = 0;
    return r;
}

V get(K k){
    return indexmap.find(k)->second.second;
}

int test(const vector<K>& vk, const vector<K>& v){
    for(K kk : vk){
        insert(kk, kk);
        iloc(kk);
        // for(K k : vk){
        // node = indexmap.find(k);
        // cout << node->first << ' ' << node->second.first.
            ↪ first << ' ' << node->second.first.second <<
            ↪ '\n';
        // if(k == kk) break;
        // }
        // cout << '\n';
    }
    K kk;
    for(ll i=0; i<indexmap.size(); i++){
        kk = iloc(i);
        if(kk != v[i] || get(kk) != v[i]) return 0;
    }
    return 1;
}

int main(){
    indexmap = imtmpm;
    vector<K> v, vk({1,2,3,5,7,8,9,11,13});//,14,15});
    cerr << vk.size() << '\n';
    v = vk;
    sort(v.begin(), v.end());
    do{
        indexmap.clear();
        if(!test(vk, v)){
            for(ll x : vk) cout << x << ' ';
            cout << '\n';
        }
        // break;
    }while(next_permutation(vk.begin(), vk.end()));
    return 0;
}
```

KMP.cpp

```
int n,i,Q,T,m,j,k,x,y,b[maxn];
string t,p;

void preprocessing(){
    int i=0,j=-1;b[0]=-1;
    while(i<m){
        while(j>=0 && p[i]!=p[j])j=b[j];
        i++; j++;
        b[i]=j;
    }
}

void search(){
    int i=0,j=0;
    while(i<n){
        while(j>=0 && t[i]!=p[j])j=b[j];
        i++;j++;
        if(j==m){
            cout<<"P found at index  "<<i-j<<ln;
        }
    }
}
```

Kruskal.cpp

```
#include "DSU.cpp"
```

```cpp
template<class T> class Kruskal{
#define e (*pe)
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
    typedef std::pair<ll,ll> pl;
    typedef std::pair<T,pl> ptel;
    typedef std::vector<ptel> vptel;
    typedef std::vector<bool> vb;
    vb used;
    vl tree_edges;
    T cost;
    ll ncc;
private:
    bool all_en = false;
    ll n,m;
    vb en;
    vptel* pe;
    DSU dsu;
    Kruskal(char dummy_pvt, vptel& edges, ll v, const vb&
        ↪ enabled, ll cc = 1, bool sorted = false, ll _e =
        ↪ -1){
        ll i;
        pe = &edges;
        n = v;
        if(_e == -1){
            m = e.size();
        }else{
            m = _e;
        }
        if(!sorted){
            sort(e.begin(), e.end());
        }
        en = enabled;
        cost = 0;
        used.assign(m, false);
        dsu = DSU(n);
        for(i=0; i<m && cc!=dsu.cc; i++){
            if(en[i] && !dsu.isSameSet(e[i].second.first, e[i
                ↪ ].second.second)){
                dsu.unionSet(e[i].second.first, e[i].second.
                    ↪ second);
                used[i] = true;
                tree_edges.push_back(i);
                cost += e[i].first;
            }
        }
        ncc = dsu.cc;
    }
public:
    Kruskal(){}
    Kruskal(vptel& edges, ll v, ll cc = 1, bool sorted =
        ↪ false, ll _e = -1){
        vb ten;
        if(_e == -1){
            _e = edges.size();
        }
        ten.assign(_e, 1);
        *this = Kruskal(0, edges, v, ten, cc, sorted, _e);
    }
    Kruskal(vptel& sorted_edges, ll v, const vb& enabled, ll
        ↪ cc = 1, ll _e = -1){
        *this = Kruskal(0, sorted_edges, v, enabled, cc, true,
            ↪  _e);
    }
    ll get_tree(ll v){
        return dsu.findSet(v);
    }
    bool is_same_tree(ll u, ll v){
        return dsu.isSameSet(u,v);
    }
#undef e
};
```

LazySegmentTree.cpp

```cpp
class ST{
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
private:
    // functions for range sum query with incremental range
        ↪ updates (increment all elements between 3 and 7 by
        ↪  2)
    // these must be adapted to the actual problem
    #define stadd(left,right) ((left)+(right))
    // stadd is used to add results of subrange queries in st
    #define lzadd(st_old,up_val) ((st_old)+(up_val))
    // lzadd is used to add a new lazy value (in lz) where an
        ↪  old one is already present
    #define lz2st(st_old, lz_val, rng_width) ((st_old)+(
        ↪ lz_val)*(rng_width))
    // lz2st is used to update st with lazy value in lz,
        ↪ rngwidth is the width of the range being updated
    // then lz_val is automatically cleared
    #define lc(x) ((x)<<1)
    #define rc(x) (lc(x)+1)
    #define lavg(x,y) (((x)+(y))>>1)
    #define ravg(x,y) (lavg(x,y)+1)
    #define width(l,r) ((r)-(l)+1)
    #define updateson(ps,v) lz[ps]=lzadd(lz[ps],v)
    ll n, i, j, v, LZD;
    const vl* a;
    vl st, lz;
    ll build(ll p, ll l, ll r){
        if(l == r) return st[p] = (*a)[l];
        return st[p] = stadd(
            build(lc(p), l, lavg(l,r)),
            build(rc(p), ravg(l,r), r)
        );
    }
    void update(ll p, ll v, ll l, ll r){
        if(v != LZD){
            st[p] = lz2st(st[p], v, width(l,r));
            updateson(lc(p), v);
            updateson(rc(p), v);
            lz[p] = LZD;
        }
    }
    ll rqr(ll p, ll l, ll r){
        update(p, lz[p], l, r);
        if(i>r || l>j) return 0;
        if(i<=l && r<=j) return st[p];
        return stadd(
            rqr(lc(p), l, lavg(l,r)),
            rqr(rc(p), ravg(l,r), r)
        );
    }
    ll rur(ll p, ll l, ll r){
        update(p, lz[p], l, r);
        if(i>r || l>j) return st[p];
        if(i<=l && r<=j){
            update(p, v, l, r);
            return st[p];
        }
        return st[p] = stadd(
            rur(lc(p), l, lavg(l,r)),
            rur(rc(p), ravg(l,r), r)
        );
    }
public:
    ST(const vl& _a, ll lazy_default_value=0){
        // LZD value is used to determine whether there is
            ↪ already a value that shuld be updated (lazily)
        // each query and update the lazy value is checked,
            ↪ and if it is LZD no updates occur
        a = &_a;
        LZD = lazy_default_value;
        n = _a.size();
        st.resize(n<<2);
        lz.assign(n<<3, LZD);
        build(1, 0, n-1);
    }
    ll rq(ll l, ll r){
        i = l;
        j = r;
        return rqr(1, 0, n-1);
    }
    void ru(ll l, ll r, ll newval){
        v = newval;
        i = l;
        j = r;
        rur(1, 0, n-1);
```

```
    }
    #undef updateson
    #undef width
    #undef ravg
    #undef lavg
    #undef rc
    #undef lc
};
```

LCA.cpp

```
#include "LazySegTree.cpp"

template<class T> class LCA{
#define t (*pt)
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
    typedef std::pair<ll,ll> pl;
    typedef std::vector<pl> vpl;
    typedef std::vector<T> vt;
    typedef std::vector<vt> vvt;
    typedef std::function<ll(const T&)> fetnt;
    static inline ll def_etn(const T& e){
        return e;
    }
private:
    LazySegTree<pl> st;
    const vvt* pt;
    ll n, es;
    vl e, h, p;
    vpl l;
    fetnt etn;
    static pl& lmin(pl& r, const pl& a, const pl& b){
        if(a.second < b.second) return r = a;
        return r = b;
    }
    void dfs(ll v, ll d){
        ll u;
        h[v] = es;
        l[es] = pl(es,d);
        e[es++] = v;
        for(const T& ed : t[v]){
            u = etn(ed);
            if(h[u] == -1){
                p[u] = v;
                dfs(u, d+1);
                l[es] = pl(es,d);
                e[es++] = v;
            }
        }
    }
public:
    LCA(){}
    LCA(const vvt& tree, ll vertex = 0, ll dimension = -1,
        ↪ fetnt edge_to_node = def_etn){
        pt = &tree;
        if(dimension == -1){
            n = tree.size();
        }else{
            n = dimension;
        }
        etn = edge_to_node;
        e.resize(n<<1);
        l.resize(n<<1);
        h.assign(n, -1);
        p.resize(n);
        p[vertex] = vertex;
        es = 0;
        dfs(vertex, 0);
        st = LazySegTree<pl>(l, pl(-1,(ll)1e18), lmin);
    }
    ll lca(ll u, ll v){
        if(h[u] < h[v]){
            return e[st.range_query(h[u], h[v]).first];
        }
        return e[st.range_query(h[v], h[u]).first];
    }
    ll depth(ll u){
        return l[h[u]].second;
    }
    ll distance(ll u, ll v){
        ll a = lca(u,v);
        return depth(u) + depth(v) - (depth(a)<<1);
    }
    ll parent(ll v, ll d){
        while(d--) v = p[v];
        return v;
    }
    ll parent(ll v){
        return p[v];
    }
    ll low_midpoint(ll u, ll v){
        static ll a, d0, d1;
        a = lca(u, v);
        d0 = distance(a, u);
        d1 = distance(a, v);
        if(d0 < d1) return parent(v, (d0+d1)>>1);
        return parent(u, (d0+d1)>>1);
    }
#undef t
};
```

MinCostMaxFlow.cpp

```
#define __MCMF_FLOW_MAX (T)1000000000000000
#define __mcmf_eu first.first
#define __mcmf_ev first.second
#define __mcmf_ec second.second
#define __mcmf_ecost second.first
#define __mcmf_ncost first.first
#define __mcmf_nflow first.second
#define __mcmf_nvisited second.first
#define __mcmf_nparent second.second

template <class W,class T> class MinCostMaxFlow{
public:
    typedef long long ll;
    typedef pair<ll,ll> pl;
    typedef pair<W,T> pwt;
    typedef pair<pwt,pl> mcmf_node;
    typedef pair<pl,pwt> mcmf_edge;
    typedef pair<T,ll> ptl;
    typedef pair<pwt,ll> pwtl;
    T total_flow = 0;
    W total_cost = 0;
    vector<T> rf;
private:
    ll n, m, m2, s, t, i, r = 0;
    vector<mcmf_node> v;
    vector<mcmf_edge> e;
    vector<vector<pl>> g;
    T fm = 0;
    T vis(){
        priority_queue<pwtl> q;
        ll x, y;
        T f = 0, tf = 0;
        W c, tc;
        v[s].__mcmf_nvisited = ++r;
        v[s].__mcmf_ncost = 0;
        v[s].__mcmf_nflow = fm-total_flow;
        q.push(pwtl(pwt(0,fm-total_flow), s));
        while(!q.empty()){
            c = -q.top().first.first;
            f = q.top().first.second;
            x = q.top().second;
            q.pop();
            if(v[x].__mcmf_ncost != c || v[x].__mcmf_nflow !=
                ↪ f) continue;
            for(pl yy : g[x]){
                y = yy.first;
                tc = c + e[yy.second].__mcmf_ecost;
                tf = min(f,rf[yy.second]);
                if(
                    rf[yy.second]!=(T)0 && (
                        v[y].__mcmf_nvisited<r ||
                        v[y].__mcmf_ncost>tc ||
                        (v[y].__mcmf_ncost==tc && v[y].
                            ↪ __mcmf_nflow<tf)
                    )
                ){
                    v[y].__mcmf_nvisited = r;
                    v[y].__mcmf_nparent = yy.second;
```

```
                v[y].__mcmf_nflow = tf;
                v[y].__mcmf_ncost = tc;
                q.push(pwtl(pwt(-tc, tf), y));
            }
        }
    }
    if(v[t].__mcmf_nvisited == r){
        f = v[t].__mcmf_nflow;
        total_cost += v[t].__mcmf_ncost * f;
        for(x=t; x!=s; x=e[y].__mcmf_eu){
            y = v[x].__mcmf_nparent;
            rf[y] -= f;
            rf[(y<m)?(y+m):(y-m)] += f;
        }
        return f;
    }
    return (T)0;
  }
public:
  MinCostMaxFlow(
    ll _v,
    ll _e,
    const vector<mcmf_edge>& edges,
    ll source = 0,
    ll sink = -1,
    const T& flow_max = __MCMF_FLOW_MAX
  ){
    n = _v;
    m = _e;
    m2 = m<<1;
    e = edges;
    e.resize(m2);
    for(i=m; i--;){
        e[i+m].__mcmf_eu = e[i].__mcmf_ev;
        e[i+m].__mcmf_ev = e[i].__mcmf_eu;
        e[i+m].__mcmf_ecost = -e[i].__mcmf_ecost;
    }
    s = source;
    if(sink == -1){
        t = n-1;
    }else{
        t = sink;
    }
    fm = flow_max;
    v.assign(n, mcmf_node(pwt((W)1e12, (T)0), pl(0,0)));
    v[s].__mcmf_ncost = 0;
    v[s].__mcmf_nflow = fm;
    rf.resize(m2);
    g.assign(n, vector<pl>());
    for(i=0; i<m; i++){
        rf[i] = e[i].__mcmf_ec;
        g[e[i].__mcmf_eu].push_back(pl(e[i].__mcmf_ev, i))
            ↪ ;
    }
    for(i=m; i<m2; i++){
        rf[i] = 0;
        g[e[i].__mcmf_eu].push_back(pl(e[i].__mcmf_ev, i))
            ↪ ;
    }
  }
  void execute(){
    T ret;
    while((ret = vis()) != (T)0){
        total_flow += ret;
    }
  }
};
```

Nim game e riconducibili

Considera n pile di oggetti. A e B giocano togliendo quanti oggetti vogliono da una di queste pile. Chi toglie l'ultimo oggetto rimanente perde, l'altro vince.

Chiamiamo vincente uno stato del gioco in cui la xor-sum (d'ora in poi chiamata s) del numero di oggetti nelle pile è pari a zero, $a_i, 1 \leq i \leq n$ è il numero di elementi nella i-esima pila:

- stato vincente: $a_1 \oplus a_2 \oplus ... \oplus a_n \neq 0$
- stato perdente: $a_1 \oplus a_2 \oplus ... \oplus a_n = 0$

Ciò è valido anche nel caso del Nim Game modificato in cui si possono aggiungere elementi. **Importante:** il gioco deve continuare ad essere aciclico.

Teorema di Sprague-Grundy e Grundy's values

Considera uno stato v di un gioco Nim-like e v_i tutti i possibili stati del gioco raggiungibili da v ($i \in 1, 2, ..., k, k \leq 0$). A questo stato possiamo associare un Nim-game equivalente con una pila di altezza x. x è chiamato *Grundy's value* associato allo stato v.

x può essere calcolato ricorsivamente:

$$x = mex\{x_1, ..., x_k\}$$

dove: x_i è il Grundy's value associato allo stato v_i; la funzione mex identifica il *minimum excludant*.

Per visualizzare il gioco può essere utilizzato un albero, le cui foglie sono gli stati perdenti e avranno Grundy's value uguale a 0.

```
/* considerare come *base* il seguente codice.
 * state_t: tipo della variabile dello stato
 * der_states: vector con stati derivati dallo stato corrente
 */

map <state_t, int> dp;
map <state_t, vector <state_t> > der_states

int grundy_value(state_t state) {
    if(dp[state]) {
        return dp[state];
    }
    set <int> x;
    for(auto i:der_states[state]) {
        x.insert(grundy_value(i, der_states[state]));
    }

    dp[state] = mex(x);
    return dp[state];
}
```

node_split.cpp

```
typedef long long ll;
typedef std::pair<ll,ll> pl;
#define plT std::pair<pl,T>
#define vplt std::vector<plT>
#define ic(x) ((x)<<1)
#define og(x) (ic(x)^1)

template<class T> void node_split(vplt& new_edges, const vplt
    ↪ & edges, ll v, const std::vector<T>& node_cap, ll&
    ↪ source, ll& sink){
    new_edges.resize(0);
    for(ll i=v; i--;){
        new_edges.push_back(plT(pl(ic(i),og(i)),node_cap[i]));
    }
    for(const plT& x : edges){
        new_edges.push_back(plT(pl(og(x.first.first), ic(x.
            ↪ first.second)), x.second));
    }
    source = og(source);
    sink = ic(sink);
}

#undef og
#undef ic
#undef vplt
#undef plT
```

Rabin_karp.cpp

```
int n,i,q,t;

vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
```

```
    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

Suffix_array.cpp

```
int n,i,Q,T,j,k,t,x,y;
int ra[maxn],sa[maxn],tmpra[maxn],tmpsa[maxn],c[maxn];
string s;

void counting(int k){
    int i,sum,maxi=max(300,n);
    for(i=0;i<maxn;i++)c[i]=0;
    for(i=0;i<n;i++)c[i+k<n ? ra[i+k]:0]++;
    for(i=sum=0;i<maxi;i++){
        int t=c[i];
        c[i]=sum;
        sum+=t;
    }
    for(i=0;i<n;i++){
        tmpsa[c[sa[i]+k <n ? ra[sa[i]+k]:0]++]=sa[i];
    }
    for(i=0;i<n;i++)sa[i]=tmpsa[i];
}

void build(){
    int i,k,r;
    for(i=0;i<n;i++)ra[i]=s[i],sa[i]=i;
    for(k=1;k<n;k<<=1){
        counting(k);
        counting(0);
        tmpra[sa[0]]=r=0;
        for(i=1;i<n;i++){
            tmpra[sa[i]]= (ra[sa[i]]==ra[sa[i-1]] && ra[sa[i]+
                ↪ k] == ra[sa[i-1]+k] )? r:++r;
        }
        for(i=0;i<n;i++)ra[i]=tmpra[i];
        if(ra[sa[n-1]]==n-1)break;
    }
}
```

Tarjan.cpp

```
template<class T> class Tarjan{
#define G (*pg)
public:
    typedef long long ll;
    typedef std::vector<ll> vl;
    typedef std::vector<vl> vvl;
    typedef std::vector<bool> vb;
    typedef std::stack<ll> sl;
    typedef std::vector<T> vt;
    typedef std::vector<vt> vvt;
    typedef std::function<ll(const T&)> fetnt;
    ll nssc;
    vl ssc;
    static ll def_etn(const T& edge){
        return edge;
    }
private:
    vvl sscs;
    ll n, vis;
    vl id, lk;
    vb os;
    sl s;
    const vvt* pg;
    fetnt etn;
    void dfs(ll v){
        static ll t;
        ll u;
        id[v] = vis;
        lk[v] = vis++;
        s.push(v);
        os[v] = true;
        for(const T& e : G[v]){
            u = etn(e);
            if(id[u] == -1) dfs(u);
            if(os[u] && lk[v]>lk[u]) lk[v] = lk[u];
        }
        if(id[v] == lk[v]){
            do{
                t = s.top();
                s.pop();
                os[t] = false;
                ssc[t] = nssc;
            }while(t != v);
            nssc++;
        }
    }
public:
    Tarjan(){}
    Tarjan(const vvt& graph, ll v = -1, fetnt edge_to_node =
        ↪ def_etn){
        pg = &graph;
        etn = edge_to_node;
        if(v == -1){
            n = G.size();
        }else{
            n = v;
        }
        id.assign(n, -1);
        os.assign(n, false);
        lk.resize(n);
        ssc.resize(n);
        sscs.resize(0);
        vis = 0;
        nssc = 0;
        for(ll i=0; i<n; i++) if(id[i] == -1){
            dfs(i);
        }
    }
    vvl& get_sscs(){
        if(!sscs.size()){
            sscs.resize(nssc);
            for(ll i=0; i<n; i++){
                sscs[ssc[i]].push_back(i);
            }
        }
        return sscs;
    }
#undef G
};
```

Trie.cpp

```
#define __TRIE_AB_SIZE 256
typedef vector<ll> vl;
typedef vector<vl> vvl;
typedef vector<string> vs;

class Trie{
private:
    ll states;
    ll state;
    vvl trie;
    vl fail;
    vvl leaves;
public:
    Trie(){}
    Trie(const vs& dictionary){
        ll tstate;
        state = 0;
        queue<ll> fq;
        states = 1;
        trie.push_back(vl());
        trie[0].assign(__TRIE_AB_SIZE, 0);
        leaves.push_back(vl());
        for(ll s=0; s<dictionary.size(); s++){
            // s is used later as an integer, don't change
                ↪ style
```

```
    for(auto c : dictionary[s]){
        if( !(tstate = trie[state][c]) ){
            tstate = trie[state][c] = states++;
            trie.push_back(vl());
            trie[tstate].assign(__TRIE_AB_SIZE, 0);
            leaves.push_back(vl());
        }
        state = tstate;
    }
    leaves[state].push_back(s);
    state = 0;
}
// find fails
fail.resize(states);
for(ll c=0; c<__TRIE_AB_SIZE; c++){
    if(trie[0][c]){
        fail[trie[0][c]] = 0;
        fq.push(trie[0][c]);
    }
}
while(!fq.empty()){
    state = fq.front();
    fq.pop();
    for(ll c=0; c<__TRIE_AB_SIZE; c++) if(trie[state][
        ↪ c]){
        tstate = fail[state];
        while(tstate && !trie[tstate][c]){
            tstate = fail[tstate];
        }
        if(trie[tstate][c]){
            fail[trie[state][c]] = trie[tstate][c];
            for(ll l : leaves[trie[tstate][c]]){
                leaves[trie[state][c]].push_back(l);
            }
        }else{
            fail[trie[state][c]] = 0;
        }
        fq.push(trie[state][c]);
    }
}
state = 0; // reset_state();
}
void reset_state(){
    state = 0;
}
ll get_initial_state(){
    return 0;
}
const vl& get_leaves(ll state){
    return leaves[state];
}
ll next_state(ll state, char c){
    while(state && !trie[state][c]){
        state = fail[state];
    }
    if(trie[state][c]){
        return trie[state][c];
    }
    return state;
}
const vl& next_leaves(char c){
    // return get_leaves( state = next_state(state, c) );
    return leaves[ state = next_state(state, c) ];
}
};
```