Team notebook

December 19, 2024

Contents

1 DS

1.1 LineContainer

```
/**
 * Description: Container where you can add lines of the form kx+m, and
     query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 */

struct Line {
	mutable ll k, m, p;
	bool operator<(const Line& o) const { return k < o.k; }
	bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	static const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
		if (y == end()) return x->p = inf, 0;
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p)
			isect(x, erase(y));
	}
	ll query(ll x) {
		assert(!empty());
		auto l = *lower_bound(x);
		return l.k * x + l.m;
	}
};
```

1.2 dsu

```
int getpar(int v){
	return (par[v] ? par[v] = getpar(par[v]) : v);
}

void merge(int u ,int v){
	u = getpar(u) , v = getpar(v);
	if(u == v)return;
	par[u]=v;
}
```

1.3 fenwick

```
void add(int pos,int x){
    for(pos+=5;pos<maxn;pos+=pos&(-pos))
        fen[pos]+=x;
}
int get(int pos){
    int ans = 0;
    for(pos+=5;pos;pos-=pos&(-pos))
        ans+=fen[pos];
    return(ans);
}
```

1.4 oset

```
struct oset{ // just don't use with numbers <= 0
    int maxn;
    vector < int > fen;
    oset(int n):
        maxn(n+100),
        fen(maxn){}

    void add(int x , int pos){
        for( ; pos < maxn ; pos += pos & -pos)
            fen[pos] += x;
    }
    int get(int pos){
        int sum = 0 ;
        for( ; pos ; pos -= pos & -pos)
```

```
            sum += fen[pos];
        return(sum);
    }

    void insert(int x , int cnt = 1){
        add(cnt , x);
    }
    void erase(int x , int cnt = 1){
        add(-cnt , x);
    }

    int find_by_order(int k){ //k-th element
        int sum = 0 , pos = 0;
        for(int i = log2(maxn) ; i >= 0 ; i --)
            if(pos + (1 << i) < maxn and sum + fen[pos + (1 << i)] < k)
                pos += (1 << i),
                sum += fen[pos];
        return(pos + 1);
    }
    int order_of_key(int key){ // number of elements <= key
        return(get(key));
    }
};
```

1.5 pbds

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;

template <class T> using Tree = tree<T, null_type, less<T>,
    rb_tree_tag,tree_order_statistics_node_update>;
```

1.6 segment

```
#define lc (v<<1)
#define rc (lc|1)
#define mid ((l+r)>>1)
//This is for range add range sum, modify accordingly
struct segment{
    ll seg[maxn<<2], lazy[maxn<<2];
    void build(int v = 1, int l = 1, int r = maxn){
        if(r - l == 1){
            seg[v] = a[l];
            return;
        }
        build(lc, l, mid);
        build(rc, mid, r);
        seg[v] = seg[lc] + seg[rc];
    }
    void shift(int v, int l, int r){
        if(!lazy[v])return;
        seg[v] += lazy[v]*(r-l);
        if(r - l == 1){
            lazy[v] = 0;
            return;
        }
        lazy[lc] += lazy[v];
        lazy[rc] += lazy[v];
        lazy[v] = 0;
    }
    void update(int L, int R, int val, int v = 1, int l = 1, int r =
         maxn){
        if(r <= L or R <= l)
            return;
        shift(v, l, r);
        if(L <= l and r <= R){
            lazy[v] += val;
            shift(v, l, r);
            return;
        }
        update(L, R, val, lc, l, mid);
        update(L, R, val, rc, mid, r);
        seg[v] = seg[lc] + seg[rc];
    }
    ll query(int L, int R, int v = 1, int l = 1, int r = maxn){
        if(r <= L or R <= l)
            return 0;
        shift(v, l, r);
        if(L <= l and r <= R){
            return seg[v];
        }
        return query(L, R, lc, l, mid) + query(L, R, rc, mid, r);
    }
};
```

2 Geometry

2.1 ClosestPair

```
/**
 * Source: https://codeforces.com/blog/entry/58747
 * Description: Finds the closest pair of points.
 * Time: O(n \log n)
 * Status: stress-tested
 */


typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
	assert(sz(v) > 1);
	set<P> S;
	sort(all(v), [](P a, P b) { return a.y < b.y; });
	pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
	int j = 0;
	for (P p : v) {
		P d{1 + (ll)sqrt(ret.first), 0};
		while (v[j].y <= p.y - d.x) S.erase(v[j++]);
		auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
		for (; lo != hi; ++lo)
			ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
		S.insert(p);
	}
	return ret.second;
}
```

2.2 circleIntersect

```
/**
 * Description: Computes the pair of points at which two circles
     intersect.
 * Returns false in case of no intersection.
 * Status: stress-tested
 */

typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
	if (a == b) { assert(r1 != r2); return false; }
	P vec = b - a;
	double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
	       p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
	if (sum*sum < d2 || dif*dif > d2) return false;
	P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
	*out = {mid + per, mid - per};
	return true;
}
```

2.3 circleline

```
/**
 * Description: Finds the intersection between a circle and a line.
 * Returns a vector of either 0, 1, or 2 intersection points.
 * P is intended to be Point<double>.
 * Status: unit tested
 */
*/

template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
	P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
	double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
	if (h2 < 0) return {};
	if (h2 == 0) return {p};
	P h = ab.unit() * sqrt(h2);
	return {p - h, p + h};
}
```

2.4 circlepolygon

```
/**
 * Description: Returns the area of the intersection of a circle with a
 * ccw polygon.
 * Time: O(n)
 * Status: Tested on GNYR 2019 Gerrymandering, stress-tested
 */

typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
```

```
	auto tri = [&](P p, P q) {
		auto r2 = r * r / 2;
		P d = q - p;
		auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
		auto det = a * a - b;
		if (det <= 0) return arg(p, q) * r2;
		auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
		if (t < 0 || 1 <= s) return arg(p, q) * r2;
		P u = p + d * s, v = p + d * t;
		return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
	};
	auto sum = 0.0;
	rep(i,0,sz(ps))
		sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
	return sum;
}
```

2.5 complex$_{geo}$

```
//THIS DOES NOT GO WELL WITH INTS
typedef complex<double> point;
#define x real()
#define y imag()

// vector addition and subtraction
cout << a + b << endl; // (5,-5)

// scalar multiplication
cout << 3.0 * a << endl; // (9,6)

//dot product: (conj(a) * b).x
//cross product: (conj(a) * b).y
//Euclidean distance: abs(a - b)
//Slope of line (a, b): tan(arg(b - a))
//Polar to cartesian: polar(r, theta)
//Cartesian to polar: point(abs(p), arg(p))
//Rotation about the origin: a * polar(1.0, theta)
//Rotation about pivot p: (a-p) * polar(1.0, theta) + p
//Angle ABC: abs(remainder(arg(a-b) - arg(c-b), 2.0 * M_PI))
//remainder normalizes the angle to be between [-PI, PI]. Thus, we can
    get the positive non-reflex angle by taking its abs value.
//Project p onto vector v: v * dot(p, v) / norm(v);
//Project p onto line (a, b): a + (b - a) * dot(p - a, b - a) / norm(b -
    a)
//Reflect p across line (a, b): a + conj((p - a) / (b - a)) * (b - a)
//Intersection of line (a, b) and (p, q):

point intersection(point a, point b, point p, point q) {
  double c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
  return (c1 * q - c2 * p) / (c1 - c2); // undefined if parallel
}
```

2.6 convex$_{hull}$

```
using P = pair<T, T>;
using vP = vector<P>;
using Line = pair<P, P>;

T sq(T a) { return a * a; }                    // square
T norm(const P &p) { return sq(p.f) + sq(p.s); } // x^2 + y^2

// basic operations
P operator-(const P &l, const P &r) { return P(l.f - r.f, l.s - r.s); }

T cross(const P &a, const P &b) { return a.f * b.s - a.s * b.f; } //
    cross product
T cross(const P &p, const P &a, const P &b) {               // cross
    product
	return cross(a - p, b - p);
}

using vi = vector<int>;
using vP = vector<P>;

vi hullInd(const vP &v) {
	int ind = int(min_element(all(v)) - begin(v));
	vi cand, hull{ind};
	FOR(i, sz(v)) if (v[i] != v[ind]) cand.pb(i);

	sort(all(cand), [&](int a, int b) { // sort by angle, tiebreak by
	    distance
		P x = v[a] - v[ind], y = v[b] - v[ind];
		T t = cross(x, y);
		return t != 0 ? t > 0 : norm(x) < norm(y);
	});
```

```
        trav(c, cand) { // for every point
                while (sz(hull) > 1 && cross(v[end(hull)[-2]], v[hull.bk],
                    v[c]) <= 0) {
                        hull.pop_back(); // pop until counterclockwise and
                            size > 1
                }
                hull.pb(c);
        }

        return hull;
}
```

2.7 lineDist

```
/**
 * Description:\\
Returns the signed distance between point p and the line containing
    points a and b.
Positive value on left side and negative on right as seen from a towards
    b. a==b gives nan.
P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long
    long.
It uses products in intermediate steps so watch out for overflow if using
    int or long long.
Using Point3D will always give a non-negative distance. For Point3D, call
    .dist on the result of the cross product.
 */

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
        return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

2.8 lineIntersect

```
/**
 * Description:\\
If a unique intersection point of the lines going through s1,e1 and s2,e2
    exists \{1, point\} is returned.
If no intersection point exists \{0, (0,0)\} is returned and if
    infinitely many exists \{-1, (0,0)\} is returned.
The wrong position will be returned if P is Point<ll> and the
    intersection point does not have integer coordinates.
Products of three coordinates are used in intermediate steps so watch out
    for overflow if using int or ll.
 * Usage:
 *      auto res = lineInter(s1,e1,s2,e2);
 *      if (res.first == 1)
 *              cout << "intersection point at " << res.second << endl;
 * Status: stress-tested, and tested through half-plane tests
 */
#pragma once

template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
        auto d = (e1 - s1).cross(e2 - s2);
        if (d == 0) // if parallel
                return {-(s1.cross(e1, s2) == 0), P(0, 0)};
        auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
        return {1, (s1 * p + e1 * q) / d};
}
```

2.9 polyUnion

```
/**
 * Description: Calculates the area of the union of $n$ polygons (not
     necessarily
 * convex). The points within each polygon must be given in CCW order.
 * (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't
     be needed.)
 * Time: $O(N^2)$, where $N$ is the total number of points
 * Status: stress-tested, Submitted on ECNA 2017 Problem A
 */
#pragma once

#include "Point.h"
#include "sideOf.h"

typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
        double ret = 0;
```

```
	rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
		P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
		vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
		rep(j,0,sz(poly)) if (i != j) {
			rep(u,0,sz(poly[j])) {
				P C = poly[j][u], D = poly[j][(u + 1) %
				    sz(poly[j])];
				int sc = sideOf(A, B, C), sd = sideOf(A, B,
				    D);
				if (sc != sd) {
					double sa = C.cross(D, A), sb =
					    C.cross(D, B);
					if (min(sc, sd) < 0)
						segs.emplace_back(sa / (sa -
						    sb), sgn(sc - sd));
				} else if (!sc && !sd && j<i &&
				    sgn((B-A).dot(D-C))>0){
					segs.emplace_back(rat(C - A, B - A),
					    1);
					segs.emplace_back(rat(D - A, B - A),
					    -1);
				}
			}
		}
		sort(all(segs));
		for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
		double sum = 0;
		int cnt = segs[0].second;
		rep(j,1,sz(segs)) {
			if (!cnt) sum += segs[j].first - segs[j - 1].first;
			cnt += segs[j].second;
		}
		ret += A.cross(B) * sum;
	}
	return ret / 2;
}
```

2.10 polygon$_a rea$

```
vector<Point> points(n);
for (auto &p : points) { cin >> p; }
points.push_back(points[0]);
long long area = 0;
for (int i = 0; i < points.size(); i++) {
    area +=
        (1LL * points[i].x * points[i + 1].y - 1LL * points[i].y *
            points[i + 1].x);
}
cout << labs(area) << '\n';
```

2.11 segIntersect

```
/**
If a unique intersection point between the line segments going from s1 to
    e1 and from s2 to e2 exists then it is returned.
If no intersection point exists an empty vector is returned.
If infinitely many exist a vector with 2 elements is returned, containing
    the endpoints of the common line segment.
The wrong position will be returned if P is Point<ll> and the
    intersection point does not have integer coordinates.
Products of three coordinates are used in intermediate steps so watch out
    for overflow if using int or long long.
 * Usage:
 * vector<P> inter = segInter(s1,e1,s2,e2);
 * if (sz(inter)==1)
 *   cout << "segments intersect at " << inter[0] << endl;
 * Status: stress-tested, tested on kattis:intersection
 */
#pragma once

#include "Point.h"
#include "OnSegment.h"

template<class P> vector<P> segInter(P a, P b, P c, P d) {
	auto oa = c.cross(d, a), ob = c.cross(d, b),
	     oc = a.cross(b, c), od = a.cross(b, d);
	// Checks if intersection is single non-endpoint point.
	if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
		return {(a * ob - b * oa) / (ob - oa)};
	set<P> s;
	if (onSegment(c, d, a)) s.insert(a);
	if (onSegment(c, d, b)) s.insert(b);
	if (onSegment(a, b, c)) s.insert(c);
	if (onSegment(a, b, d)) s.insert(d);
	return {all(s)};
```

```
}
```

2.12 segdist

```
/**
Returns the shortest distance between point p and the line segment from
    point s to e.
 * Usage:
 *     Point<double> a, b(2,2), p(1,1);
 *     bool onSegment = segDist(a,b,p) < 1e-10;
 * Status: tested
 */


typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
       if (s==e) return (p-s).dist();
       auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
       return ((p-s)*d-(e-s)*t).dist()/d;
}
```

2.13 sideof

```
/**
 * Description: Returns where $p$ is as seen from $s$ towards $e$. 1/0/-1
    $\Leftrightarrow$ left/on line/right.
 * If the optional argument $eps$ is given 0 is returned if $p$ is within
    distance $eps$ from the line.
 * P is supposed to be Point<T> where T is e.g. double or long long.
 * It uses products in intermediate steps so watch out for overflow if
    using int or long long.
 * Usage:
 *     bool left = sideOf(p1,p2,q)==1;
 * Status: tested
 */


template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
       auto a = (e-s).cross(p-s);
       double l = (e-s).dist()*eps;
       return (a > l) - (a < -l);
}
```

3 Graphs

3.1 2-sat

```
struct sat{ //v = 2*v , ~v = 2*v + 1 ==> ~v = v^1
       int n, c;
       vector < vector < int > > in , out;
       vector < int > col , topo;
       sat(int N):
              n(N) , c(0) , in(2*n + 5) , out(2*n + 5) , col(2*n + 5){}
       bool operator [] (int x) { return(col[2*x] > col[2*x + 1]); }
       void add_e(int v , int u){in[u].pb(v) , out[v].pb(u);}
       void add(int v , int u){add_e(u^1 , v) , add_e(v^1 , u);}
       void sfd(int v){
              col[v] = c;
              for(auto u : in[v]) if(!col[u])
                     sfd(u);
       }
       void dfs(int v){
              col[v] = 1;
              for(auto u : out[v]) if(!col[u])
                     dfs(u);
              topo.pb(v);
       }
       bool validate(){
              for(int i = 1 ; i <= 2*n+1 ; i ++ ) if(!col[i]) dfs(i);
              reverse(topo.begin() , topo.end());
              fill(col.begin() , col.end() , 0 );
              for(auto v : topo)
                     if(!col[v])
                            ++c , sfd(v);
              for(int i = 1 ; i <= n ; i ++) if(col[i * 2] == col[i * 2
                  + 1])return(0);
       return(1);
       }
};
```

3.2 Dinic

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Dinic {
    #define MAXN 100010
    int n = 0, m = 0, turn = 0;
    vector < int > a, b, h, mark, pos, adj[MAXN];
    vector < int64_t > c, d;
    queue < int > q;
    void add_edge(int u, int v, int64_t w = 1) {
        u--, v--;
        adj[u].push_back(m);
        adj[v].push_back(m);
        a.push_back(u);
        b.push_back(v);
        c.push_back(w);
        m++;
        n = max(n, max(u, v) + 1);
    }
    void bfs(int v) {
        mark[v] = turn;
        int l = 0, r = 0;
        pos[r++] = v;
        h[v] = 0;
        while (l < r) {
            int v = pos[l++];
            for (int w: adj[v]) {
                if (a[w] == v and mark[b[w]] ^ turn and c[w] - d[w] > 0) {
                    mark[b[w]] = turn, h[b[w]] = h[v] + 1;
                    pos[r++] = b[w];
                }
                if (b[w] == v and mark[a[w]] ^ turn and d[w] > 0) {
                    mark[a[w]] = turn, h[a[w]] = h[v] + 1;
                    pos[r++] = a[w];
                }
            }
        }
    }
    int64_t pump(int v, int source, int sink, int64_t cap = (1LL << 62)) {
        int64_t ans = 0;
        if (v == sink)
            return cap;
        if (v == source)
            turn++, bfs(v), fill(pos.begin(), pos.end(), 0);
        mark[v] = turn;
        for (; pos[v] < int(adj[v].size()); pos[v]++) {
            int w = adj[v][pos[v]];
            if (a[w] == v) {
                if (c[w] - d[w] == 0) continue;
                if (h[b[w]] ^ (h[v] + 1)) continue;
                int64_t res = pump(b[w], source, sink, min(cap, c[w] -
                    d[w]));
                ans += res;
                cap -= res;
                d[w] += res;
            }
            if (b[w] == v) {
                if (d[w] == 0) continue;
                if (h[a[w]] ^ (h[v] + 1)) continue;
                int64_t res = pump(a[w], source, sink, min(cap, d[w]));
                ans += res;
                cap -= res;
                d[w] -= res;
            }
        }
        return ans;
    }
    int64_t solve(int source, int sink) {
        source--, sink--;
        int64_t ans = 0;
        d.resize(m), fill(d.begin(), d.end(), 0);
        mark.resize(n), fill(mark.begin(), mark.end(), 0);
        h.resize(n);
        pos.resize(n);
        while (int64_t pumped = pump(source, source, sink))
            ans += pumped;
        return ans;
    }
};

int n, m;

Dinic crap;

int32_t main() {
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n >> m;
```

```
    for (int i = 1, a, b, c; i <= m; i++)
        cin >> a >> b >> c, crap.add_edge(a, b, c);
    cout << crap.solve(1, n);
    return (0);
}
```

3.3 Hopcroft

```
#include <bits/stdc++.h>

using namespace std;

#define endl '\n'

struct hopcroft{ //0 based
    int n , m; // size of each side
    int ans;
    vector < int > mu , mv; // u is matched with mu[u] and v with mv[v],
        -1 if not matched
    vector < vector < int > > adj;
    vector < int > layer;
    hopcroft(int n, int m):
        n(n) , m(m), ans(0),
        mu(n , -1) , mv(m , -1),
        adj(n) , layer(n){}
    void add_edge(int u, int v){
        adj[u].push_back(v);
        if(mu[u] == -1 and mv[v] == -1)
            ans ++ , mu[u] = v , mv[v] = u;
    }
    void bfs(){
        queue <int> q;
        for(int u = 0; u < n; u ++){
            if(mu[u] == -1) q.push(u), layer[u] = 0;
            else layer[u] = -1;
        }
        while(!q.empty()){
            int u = q.front(); q.pop();
            for(auto v: adj[u]) if(mv[v] != -1 and layer[mv[v]] == -1){
                layer[mv[v]] = layer[u] + 1;
                q.push(mv[v]);
            }
        }
    }
    bool dfs(int u){
        for(auto v: adj[u]) if(mv[v] == -1){
            mu[u] = v, mv[v] = u;
            return(1);
        }
        for(auto v: adj[u]) if(layer[mv[v]] == layer[u] + 1 and
            dfs(mv[v])){
            mu[u] = v, mv[v] = u;
            return(1);
        }
        return(0);
    }
    int solve(){ // O( sqrt(V) * E )
        while(true){
            bfs();
            int augment = 0;
            for(int u = 0; u < n; u ++)
                if(mu[u] == -1)
                    augment += dfs(u);
            if(!augment)
                break;
            ans += augment;
        }
        return(ans);
    }
};

int l , r;
int m;

int32_t main(){
    ios::sync_with_stdio(false);cin.tie(0);
    cin >> l >> r >> m;
    hopcroft g(l , r);
    while(m -- ){
        int u , v;
        cin >> u >> v;
        g.add_edge(u , v);
    }
    cout << g.solve() << endl;
    for(int i = 0 ; i < l ; i ++){
        if(g.mu[i] != -1)
            cout << i << ' ' << g.mu[i] << endl;
    }
```

```
    return(0);
}
```

3.4 cen

```
void plant(int v , int par = 0){
        sz[v] = 1;
        for(auto u : adj[v]) if(u != par and !hide[u])
                plant(u , v) , sz[v] += sz[u];
}
int cen(int v , int n , int par = 0 , bool found = 0){
        while(!found){
                found = 1;
                for(auto u : adj[v]) if(u!=par and !hide[u] and sz[u] * 2
                    > n){
                        par = v , v = u , found = 0;
                        break;
                }
        }
        return(v);
}
void add(int v , int par , int c){
        if(hide[v])return;
        for(auto u : adj[v])
                if(u!=par)
                        add(u , v , c);
}
void rem(int v , int par , int c){
        if(hide[v])return;
        for(auto u : adj[v])
                if(u!=par)
                        rem(u , v , c);
}
void calc(int v , int par){
        if(hide[v])return;
        for(auto u : adj[v])if(u!=par)
                calc(u , v);
}
void calc(int v){
        for(auto u : adj[v])
                add(u , v , a[v]);
        for(auto u : adj[v])
                rem(u , v , a[v]) , calc(u , v) , add(u , v , a[v]);
        for(auto u : adj[v])
                rem(u , v , a[v]);
}
void solve(int v){
        plant(v);
        v = cen(v , sz[v]);
        hide[v] = 1;
        calc(v);
        for(auto u : adj[v])
                if(!hide[u])
                        solve(u);
}
```

3.5 dijkstra

```
void djk(int source){
    ms(dist, 63);
    dist[source] = 0;
    pq.push({-dist[source], source});
    while(pq.size()){
        auto [d, v] = pq.top();
        pq.pop();
        if(mark[v])continue;
        mark[v] = 1;
        for(auto [u, w] : adj[v]){
            if(dist[u] > dist[v] + w)
                dist[u] = dist[v] + w, pq.push({-dist[u], u});
        }
    }
}
```

3.6 hld

```
void dfs_sz(int v = 0) {
    sz[v] = 1;
    for(auto &u: g[v]) {
        dfs_sz(u);
        sz[v] += sz[u];
        if(sz[u] > sz[g[v][0]]) {
            swap(u, g[v][0]);
        }
```

```
    }
}

void dfs_hld(int v = 0) {
    if(!head[v])head[v] = v;
    if(g[v].size())
        head[g[v][0]] = head[v];
    in[v] = ++t;
    for(auto u: g[v]) {
        dfs_hld(u);
    }
    out[v] = t+1;
}
```

3.7 scc

```
int n , m , cnt = 1 ;
vector < int > adj[maxn] , radj[maxn] , order;
int mark[maxn] , c[maxn];

void sfd(int v){
    c[v] = cur;
    for (auto u : radj[v])
        if(!c[u])
            sfd(u);
}

void dfs(int v){
    mark[v] = 1;
    for (auto u : adj[v])
        if(!mark[u])
            dfs(u);
    order.pb(v);
}

int32_t main(){
    for (int i = 1 ; i <= n ; i ++)
        if(!mark[i])
            dfs(i);
    reverse(order.begin() , order.end());
    for (int i = 0 ; i < n ; i ++)
        if(!c[order[i]])
            ++cnt, sfd(order[i]);
    return(0);
}
```

3.8 tur

```
int pointer[MAXN];
vector<pii> adj[MAXN];
bool mark[MAXN];

void tour(int v){
    while(pointer[v] < (int)adj[v].size()){
        if(!mark[adj[v][pointer[v]].S]){
            mark[adj[v][pointer[v]].S] = 1;
            tour(adj[v][pointer[v]].F);
        }
        pointer[v]++;
    }
    ans.push_back(v);
}
```

3.9 virtual$_{tree}$

```
/**
 * Description: Given a rooted tree and a subset S of nodes, compute the
     minimal
 * subtree that contains all the nodes by adding all (at most $|S|-1$)
 * pairwise LCA's and compressing edges.
 * Returns a list of (par, orig\_index) representing a tree rooted at 0.
 * The root points to itself.
 * Time: $O(|S| \log |S|)$
 */

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
        static vi rev; rev.resize(sz(lca.time));
        vi li = subset, &T = lca.time;
        auto cmp = [&](int a, int b) { return T[a] < T[b]; };
        sort(all(li), cmp);
        int m = sz(li)-1;
        rep(i,0,m) {
                int a = li[i], b = li[i+1];
```

```
            li.push_back(lca.lca(a, b));
        }
        sort(all(li), cmp);
        li.erase(unique(all(li)), li.end());
        rep(i,0,sz(li)) rev[li[i]] = i;
        vpi ret = {pii(0, li[0])};
        rep(i,0,sz(li)-1) {
            int a = li[i], b = li[i+1];
            ret.emplace_back(rev[lca.lca(a, b)], b);
        }
        return ret;
}
```

4 Misc

4.1 CRT

```
struct Congruence {
    long long a, m;
};

long long chinese_remainder_theorem(vector<Congruence> const&
     congruences) {
    long long M = 1;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }

    long long solution = 0;
    for (auto const& congruence : congruences) {
        long long a_i = congruence.a;
        long long M_i = M / congruence.m;
        long long N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

4.2 LIS

```
int LIS(vector <int> &vec){
    multiset <int> st;
    for(int x : vec){
        st.insert(x);
        auto it = st.lower_bound(x);
        it++;
        if(it != st.end())
            st.erase(it);
    }
    return st.size();
}
```

4.3 codeKalak

```
freopen("input.txt", "r", stdin);
int main(int argc, char* argv[]) {
    // Print each argument
    for (int i = 0; i < argc; ++i) {
        std::cout << "Argument " << i << ": " << argv[i] << std::endl;
    }
}
```

4.4 dp$_{d}ivide$

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
    }
```

```
    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

4.5 masks

```
//all submasks
for (int s=m; ; s=(s-1)&m) {
    ... you can use s ...
    if (s==0) break;
}
//all masks and their submasks in 3^n
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
        ... s and m ...
```

4.6 mat

```
struct Mat{
    int m[K][K];
    Mat(int diag = -1){
        ms(m , 0);
        if(diag==0)for(int i = 0 ; i < K ; i ++)m[i][i] = 1;
        if(diag>0)for(int i = 0 ; i < diag ; i ++)m[i][i+1]=1;
    }
    Mat operator* (const Mat &b) const{
        Mat c = Mat();
        for(int i = 0 ; i < K ; i ++)
            for(int k = 0 ; k < K ; k ++)
                for(int j = 0 ; j < K ; j ++)
                    c.m[i][j] = (ll(c.m[i][j]) + ll(m[i][k]) *
                        b.m[k][j])%mod;
        return(c);
    }
};
Mat pw(Mat a, ll b){Mat
    res(0);while(b){if(b&1){res=(a*res);}a=(a*a);b>>=1;}return(res);}
```

4.7 phi

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

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

4.8 sosDP

```
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
	F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
	if(mask & (1<<i))
		F[mask] += F[mask^(1<<i)];
}


//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
	dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
	for(int i = 0;i < N; ++i){
		if(mask & (1<<i))
			dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
		else
			dp[mask][i] = dp[mask][i-1];
	}
	F[mask] = dp[mask][N-1];
}
```

4.9 time

```
chrono::steady_clock::time_point begin = chrono::steady_clock::now();
chrono::steady_clock::time_point end = chrono::steady_clock::now();
chrono::duration_cast<std::chrono::milliseconds>(end - begin).count();

auto start_time = chrono::high_resolution_clock::now();
auto end_time = chrono::high_resolution_clock::now();
auto elapsed_time = chrono::duration_cast<chrono::milliseconds>(end_time
    - start_time);
std::cout << "Elapsed time: " << elapsed_time.count() << " ms\n";
```

5 Strings

5.1 Aho

```
#define SIGMA 26
int nxt[SIGMA][MAXN] , f[MAXN] , ext[MAXN] , sz = 0;
bool endpoint[MAXN];

int add(string &s){
	int cur = 0;
	for(char c : s){
		if(!nxt[c - 'a'][cur])nxt[c - 'a'][cur] = ++sz;
		cur = nxt[c - 'a'][cur];
	}
	endpoint[cur] = 1;
	return cur;
}

void build(){//q is a queue
	for(int i = 0 ; i < SIGMA ; i ++)if(nxt[i][0])q.push(nxt[i][0]);
	while(q.size()){
		int v = q.front();
		q.pop();
		if(endpoint[f[v]])ext[v] = f[v];
		else ext[v] = ext[f[v]];
		for(int i = 0 ; i < SIGMA ; i ++)
			if(nxt[i][v])f[nxt[i][v]] = nxt[i][f[v]] ,
			    q.push(nxt[i][v]);
			else nxt[i][v] = nxt[i][f[v]];
	}
}
```

5.2 Strtable

```
struct strtable{
	#define MAXN 500010
	#define LG 20

	int rnk[LG][MAXN], n;
	int tmp[MAXN];
	int LST[MAXN], NXT[MAXN];
	int lg[MAXN];
	void build(string s){
		n = s.size();
		for(int i = 0 ; i < n ; i ++)
			tmp[i] = s[i] - 'a';
		sort(tmp , tmp + n);
```

```
        int sz = unique(tmp , tmp + n) - tmp;
        for(int i = 0 ; i < sz ; i ++)
                LST[tmp[i]] = i;
        for(int i = 0 ; i < n ; i ++)
                rnk[0][i] = LST[s[i] - 'a'];
        for(int j = 1 ; (1 << j) - 1 < n ; j ++){
                for(int i = 0 ; i + (1 << (j-1)) - 1 < n ; i
                    ++)LST[i] = -1;
                for(int i = n - (1 << j) ; ~i ; i --)
                        NXT[i] = LST[rnk[j - 1][i + (1 << (j - 1))]]
                            , LST[rnk[j - 1][i + (1 << (j - 1))]] =
                            i;
                int pos = 0;
                for(int i = 0 ; i + (1 << (j-1)) - 1 < n ; i ++)
                        for(int k = LST[i] ; ~k ; k = NXT[k])
                                tmp[pos++] = k;
                for(int i = 0 ; i + (1 << (j-1)) - 1 < n ; i
                    ++)LST[i] = -1;
                for(int i = n - (1 << j) ; ~i ; i --)
                        NXT[i] = LST[rnk[j - 1][tmp[i]]] , LST[rnk[j
                            - 1][tmp[i]]] = i;
                pos = 0;
                for(int i = 0 ; i + (1 << (j-1)) - 1 < n ; pos +=
                    (LST[i] > -1) , i ++)
                        for(int k = LST[i] ; ~k ; k = NXT[k])
                                rnk[j][tmp[k]] = pos ,
                                pos = ((~NXT[k]) ? ((rnk[j -
                                    1][tmp[k] + (1 << (j - 1))] ^
                                    rnk[j - 1][tmp[NXT[k]] + (1 << (j
                                    - 1))]) ? pos + 1 : pos) : pos);
        }
        for(int i = 2 ; i <= n ; i ++)
                lg[i] = lg[i >> 1] + 1;
}
pair < int , int > get(int l , int r){
        return pair < int , int > (rnk[lg[r - l]][l] , rnk[lg[r -
            l]][r - (1 << lg[r - l]) + 1]);
}
bool cmp(int l, int r, int L, int R){
        int sz = min(r - l , R - L);
        if(get(l , l + sz) == get(L , L + sz))
                return (r - l) < (R - L);
        return get(l , l + sz) < get(L , L + sz);
}
int Lcp(int l , int r , int L , int R){
        int ans = 0;
        for(int i = 0 ; i < n ; i ++){
                for(int j = LG ; ~j ; j --)if(l + (1 << j) -1 <= r
                    and L + (1 << j) - 1 <= R){
                        if(rnk[j][l] == rnk[j][L]){
                                ans += (1 << j);
                                l += (1 << j);
                                L += (1 << j);
                        }
                }
        }
        return ans;
    }
};

int sa[MAXN];
strtable *st;
bool SAcmp(int i , int j){
        return st->cmp(i , st->n - 1 , j , st->n - 1);
}
void SA(strtable *ST){
        st = ST;
        for(int i = 0 ; i < st->n ; i ++) sa[i] = i;
        sort(sa , sa + st->n , SAcmp);
}

int lcp[MAXN];
void LCP(strtable *st){
        for(int i = 1 ; i < st->n ; i ++){
                int u = sa[i - 1], v = sa[i];
                for(int j = LG ; ~j ; j --)if(u + (1 << j) - 1 < st->n and
                    v + (1 << j) - 1 < st->n){
                        if(st->rnk[j][u] == st->rnk[j][v]){
                                lcp[i] += (1 << j);
                                u += (1 << j);
                                v += (1 << j);
                        }
                }
        }
}
```

5.3 SuffixArray

```
int sa[maxn];
int rk[maxn];
int tp[maxn];
int cnt[maxn];
int lcp[maxn];

void SA(string &s){
	int A = 'z' , p = 0 , n = s.size();
	if(n == 1){
		sa[0] = rk[0] = 0;
		return;
	}
	for(int i = 0 ; i < n ; i ++)
		sa[i] = i , rk[i] = s[i];
	for(int j = 1 ; p < n - 1 ; j<<=1){
		p = 0;
		int k = (j>>1);
		for(int i = n - k ; i < n ; i ++)
			tp[p ++ ] = i;
		for(int i = 0 ; i < n ; i ++)
			if(sa[i] >= k)
				tp[p ++] = sa[i] - k;
		for(int i = 0 ; i <= A ; i ++)
			cnt[i] = 0;
		for(int i = 0 ; i < n ; i ++)
			cnt[rk[i]] ++;
		for(int i = 1 ; i <= A ; i ++)
			cnt[i] += cnt[i - 1];
		for(int i = n - 1 ; i >= 0 ; i --)
			sa[--cnt[rk[tp[i]]]] = tp[i];
		swap(rk , tp);
		rk[sa[0]] = p = 0;
		for(int i = 1 ; i < n ; i ++)
			p += (tp[sa[i - 1]]!=tp[sa[i]] || sa[i - 1] + k >=
			    n ||
			 tp[sa[i - 1]+k] != tp[sa[i] + k]) , rk[sa[i]] = p;
		A = p;
	}
}

void LCP(string &s){
	for(int i = 0 , k = 0 ; i < s.size() ; i ++){
		if(rk[i] == 0)continue;
		if(k) k --;
		while(s[i + k] == s[sa[rk[i] - 1] + k]) k ++;
		lcp[rk[i]] = k;
	}
}
```

5.4 Zalgo

```
int zlcp[MAXN]; //zlcp[i] == lcp(s[i , .... , n-1] , s[0 , ... , n-1])
void ZAlgo(strtable *st){
	for(int i = 0 ; i < st-> n ; i ++){
		int u = i , v = 0;
		for(int j = LG ; ~j ; j --)if(u + (1 << j) -1 < st->n and
		    v + (1 << j) - 1 < st->n){
			if(st->rnk[j][u] == st->rnk[j][v]){
				zlcp[i] += (1 << j);
				u += (1 << j);
				v += (1 << j);
			}
		}
	}
}
```

5.5 kmp

```
//1 indexed
vector < int > kmp(string s){
    int i = -1 ;
    vector < int > f(s.size() + 1);
    f[0] = -1;
    for (int j = 0 ; j < s.size() ; j ++){
        while(s[j]!=s[i] and i!=-1)
            i = f[i];
        f[j + 1] = ++i;
    }
    return(f);
}
//0 indexed
vector < int > kmp(string s){
	int n = s.size();
    vector < int > f(n);
    for(int i = 1; i < n ; i ++){
		int j = f[i-1];
```

```
            while(j and s[i]!=s[j])
                    j = f[j-1];
            j+=(s[i]==s[j]) , f[i] = j;
        }
    return(f);
}
```

5.6 string$_{s}tuff$

```
getline(input_stream, string_variable);
getline(cin, line);
stringstream ss(input);
//do not forget cin.ignore
```

6 Templates

6.1 temp

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

typedef long long ll;
typedef long double ld;
typedef pair<int , int> pii;

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

const int maxn = 3e6;
const ll mod = 1e9+7;
#define pb push_back
#define endl '\n'
#define dokme(x) cout << x , exit(0)
#define ms(x , y) memset(x , y , sizeof x)
ll pw(ll a, ll b, ll md = mod){ll res =
    1;while(b){if(b&1){res=(a*res)%md;}a=(a*a)%md;b>>=1;}return(res);}


int32_t main(){
        cin.tie(0)->sync_with_stdio(0);

        return(0);
}
```

6.2 tester

```
#!/bin/bash
echo "" > main.txt
echo "" > naive.txt
g++ -std=c++17 -o main main.cpp
g++ -std=c++17 -o naive naive.cpp
g++ -std=c++17 -o gen gen.cpp
((i = 1))
while diff main.txt naive.txt -Bb
do
  echo $i
  ((i++))
  ./gen > test.txt
  ./main < test.txt > main.txt
  ./naive < test.txt > naive.txt
done
```