# Contents

# 1 Data Structures

## 1.1 Disjoint Set Union

```
struct DSU
{
    int par[V], sz[V];
    DSU(){init(V);}
    void init(int n)
    {
        for (int i = 0; i<n; i++)
        par[i] = i, sz[i] = 1;
    }
    int find(int x)
    {
        return x == par[x] ? x : (par[x] = find(par[x]));
    }
    int getSize(int k){return sz[find(k)];}
    void unite(int x, int y)
    {
        int u=find(x), v=find(y);
        if(u==v) return;
        if(sz[u]>sz[v]) swap(u, v);
        sz[v]+=sz[u];
        sz[u] = 0;
        par[u] = par[v];
    }
};
```

## 1.2 Segment Tree (with Lazy propagation)

Incremental modifications with lazy propagation. All intervals are $[l, r)$.

```
const int N = 1001010;
struct Segtree
{
    int n, h;
    int T[2*N];
    int Lazy[N];
    int32_t Len[2*N];
    void apply(int pos, int val)
    {
        T[pos] += val * Len[pos];
        if (pos < n) Lazy[pos] += val;
    }
    void build(vector<int> &v)
    {
        n = v.size();
        h = sizeof(int) * 8 - __builtin_clz(n);
        for (int i = 0; i<n; i++)
        {
            T[n+i] = v[i];
            Len[n+i] = 1;
        }
        for (int i = n-1; i>0; i--)
        {
            T[i] = T[i<<1] + T[(i<<1)|1];
            Len[i] = Len[i<<1] + Len[(i<<1)|1];
        }
    }
    void pupd(int p)
    {
        while(p > 1)
        {
            p>>=1;
            T[p] = (T[p<<1] + T[(p<<1)|1] + (Lazy[p] * Len[p]));
        }
    }
    void propagate(int p)
    {
        for (int s = h; s > 0; s--)
        {
            int i = p >> s;
            if (!i) continue;
            if (Lazy[i] != 0)
            {
                apply(i << 1, Lazy[i]);
                apply((i << 1)|1, Lazy[i]);
                Lazy[i] = 0;
            }
        }
    }
    void modify(int pos, int val)
    {
        for (T[pos += n] = val; pos > 1; pos >>= 1)
            T[pos >> 1] = T[pos] + T[pos^1];
    }
    void modifyRange(int l, int r, int val)
    {
        l += n, r += n;
        int l0 = l, r0 = r;
        for (; l < r; l >>= 1, r >>= 1)
        {
            if (l & 1) apply(l++, val);
            if (r & 1) apply(--r, val);
        }
        pupd(l0), pupd(r0-1);
    }
    // query is on [l, r)!!
    int query(int l, int r)
    {
        l += n, r += n;
        propagate(l); propagate(r-1);
        int res = 0;
        for (; l < r; l >>=1, r>>=1)
        {
            if (l & 1)
                res += T[l++];
            if (r & 1)
                res += T[--r];
        }
        return res;
    }
} S;
```

## 1.3 2D Segment Tree

Point update, rectangle query. All intervals are $[a, b] \times [c, d]$.

```
auto gif = [](int a, int b){return a+b;};
class SEG2D
{
public:
    int n;
    int m;
    vector <vector <int>> tree;
    SEG2D(int n = 0, int m = 0)
    {
        tree.resize(2*n);
        for (int i = 0; i<2*n; i++) tree[i].resize(2*m);
        this->n = n;
        this->m = m;
    }
    SEG2D(int n, int m, vector<vector<int>> &data)
    {
        tree.resize(2*n);
        for (int i = 0; i<2*n; i++) tree[i].resize(2*m);
        this->n = n;
        this->m = m;
        init(data);
    }
    void init(vector <vector <int>> & data)
    {
        n = data.size();
        m = data.front().size();
        tree = vector<vector<int>>(2*n, vector<int>(2*m, 0));
        for (int i = 0; i<n; i++)
            for (int j = 0; j<m; j++)
                tree[i+n][j+m] = data[i][j];
        for (int i = n; i<2*n; i++)
            for (int j = m-1; j>0; j--)
                tree[i][j] = gif(tree[i][j*2], tree[i][j*2+1]);
        for (int i = n-1; i>0; i--)
            for (int j = 1; j<2*m; j++)
                tree[i][j] = gif(tree[i*2][j], tree[i*2+1][j]);
    }
    void update(int x, int y, int val)
    {
        tree[x+n][y+m] = val;
        for(int i = y+m; i > 1; i /= 2)
            tree[x+n][i/2] = gif(tree[x+n][i] , tree[x+n][i^1]);
        for (int i = x+n; i>1; i/=2)
            for (int j = y+m; j>=1; j/=2)
                tree[i/2][j] = gif(tree[i][j] , tree[i^1][j]);
    }
    int query_1D(int x, int yl, int yr)
    {
        int res = 0;
        int u = yl+m, v = yr+m+1;
        for(; u<v; u/=2, v/=2)
        {
            if (u & 1)
                res = gif(res, tree[x][u++]);
            if (v & 1)
                res = gif(res, tree[x][--v]);
        }
        return res;
    }
    int query_2D(int xl, int xr, int yl, int yr)
    {
        int res = 0;
        int u = xl+n, v = xr+n+1;
        for(; u<v; u/=2, v/=2)
        {
            if (u & 1)
            {
                int k = query_1D(u++, yl, yr);
                res = gif(res, k);
            }
            if (v & 1)
            {
                int k = query_1D(--v, yl, yr);
                res = gif(res, k);
            }
        }
        return res;
    }
};
```

### 1.4 Merge Sort Tree

greater(s, e, k, 1, 0, n) : How many elements in range [s, e) are strictly greater than $k$?

```
#define MAXN (1<<18)
#define ST (1<<17)
struct merge_sort_tree
{
    vector <int> tree[MAXN];
    int n;
    void construct (vector <int> data)
    {
        n = 1;
        while(n < data.size()) n <<= 1;
        for (int i = 0; i<data.size(); i++)
            tree[i+n] = {data[i]};
        for (int i = data.size(); i<n; i++)
            tree[i+n] = {};
        for (int i = n-1; i>0; i--)
        {
            tree[i].resize(tree[i*2].size()+tree[i*2+1]
            for (int p = 0, q = 0, j = 0; j < tree[i].s
            {
                if (p == tree[i*2].size() ||
                (q<tree[i*2+1].size() && tree[i*2+1][q]
                    tree[i][j] = tree[i*2+1][q++];
                else tree[i][j] = tree[i*2][p++];
            }
        }
    }
    //greater(s,e,k,1,0,n)
    int greater(int s, int e, int k, int node, int ns,
    {
        if (ne <= s || ns >= e)
            return 0;
        if(s <= ns && ne <= e)
            return tree[node].end() - upper_bound(all(t
        int mid = (ns+ne)>>1;
        return greater(s,e,k,node*2,ns,mid) +
         greater(s,e,k,node*2+1,mid,ne);
    }
};
```

# 2 Dynamic Programming

## 2.1 Li Chao Tree

Objective

(1) Line insert query (ax + b)

(2) Max / Min on x = t query

Current Implementation : Max query

```
struct LiChao
{
    struct Line // Linear function ax + b
    {
        int a, b;
        int eval(int x)
        {
            return a*x + b;
        }
    };
    struct Node // [start, end] has line f
    {
        int left, right;
        int start, end;
        Line f;
    };

    Node new_node(int a, int b)
    {
        return {-1,-1,a,b,{0,-INF}};
        // for min, change -INF to INF
    }

    vector <Node> nodes;

    void init(int min_x, int max_x)
    {
        nodes.push_back(new_node(min_x, max_x));
    }
    void insert(int n, Line new_line)
    {
        int xl = nodes[n].start, xr = nodes[n].end;
        int xm = (xl + xr)/2;
        Line llo, lhi;
        llo = nodes[n].f, lhi = new_line;
        if (llo.eval(xl) >= lhi.eval(xl))
            swap(llo, lhi);
        if (llo.eval(xr) <= lhi.eval(xr))
        {
            nodes[n].f = lhi;
            // for min, lhi -> llo
            return;
        }
        else if (llo.eval(xm) > lhi.eval(xm))
        {
            nodes[n].f = llo;
            // for min, llo -> lhi
            if (nodes[n].left == -1)
            {
                nodes[n].left = nodes.size();
                nodes.push_back(new_node(xl,xm));
            }
            insert(nodes[n].left, lhi);
            // for min, lhi -> llo
        }
        else
        {
            nodes[n].f = lhi;
            // for min, lhi -> llo
            if (nodes[n].right == -1)
            {
                nodes[n].right = nodes.size();
                nodes.push_back(new_node(xm+1,xr));
            }
            insert(nodes[n].right,llo);
            // for min, llo -> lhi
        }
    }
    void insert(Line f)
    {
        insert(0, f);
    }
    int get(int n, int q)
    {
        // for min, max -> min, -INF -> INF
        if (n == -1) return -INF;
        int xl = nodes[n].start, xr = nodes[n].end;
        int xm = (xl + xr)/2;
        if (q > xm)
            return max(nodes[n].f.eval(q), get(nodes[n].right, q));
        else return max(nodes[n].f.eval(q), get(nodes[n].left, q));
    }
    int get(int pt)
    {
        return get(0, pt);
    }
};
```

# 3 Mathematics

## 3.1 Fast Fourier Transform

Multiply two polynomials in $O(n \log n)$. For

```
#define _USE_MATH_DEFINES

#define sz(v) ((int)(v).size())
#define all(v) (v).begin(),(v).end()
typedef vector<int> vi;
typedef complex<double> base;

void fft(vector <base> &a, bool invert)
{
    int n = sz(a);
    for (int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if (i < j) swap(a[i],a[j]);
    }
    for (int len=2;len<=n;len<<=1){
        double ang = 2*M_PI/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){
            base w(1);
            for (int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert){
        for (int i=0;i<n;i++) a[i] /= n;
    }
}

void multiply(const vi &a,const vi &b,vi &res)
{
    vector <base> fa(all(a)), fb(all(b));
    int n = 1;
    while (n < max(sz(a),sz(b))) n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false); fft(fb,false);
    for (int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for (int i=0;i<n;i++)
        res[i] = int(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
}
//multiply(a, b, res)
//for n = b.size
//sum(a[i+j], b[n-1-j]) -> res[n-1+i]
```

## 3.2 Berlekamp Massey Algorithm

Find minimum linear recurrence from $3n$ first terms.

```
struct Berlekamp_Massey
{
    const int mod = 1000000007;
    using lint = long long;
    lint ipow(lint x, lint p){
        lint ret = 1, piv = x;
        while(p){
            if(p & 1) ret = ret * piv % mod;
            piv = piv * piv % mod;
            p >>= 1;
        }
        return ret;
    }
    vector<int> berlekamp_massey(vector<int> x){
        vector<int> ls, cur;
        int lf, ld;
        for(int i=0; i<x.size(); i++){
            lint t = 0;
            for(int j=0; j<cur.size(); j++){
                t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
            }
            if((t - x[i]) % mod == 0) continue;
            if(cur.empty()){
                cur.resize(i+1);
                lf = i;
                ld = (t - x[i]) % mod;
                continue;
            }
            lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
            vector<int> c(i-lf-1);
            c.push_back(k);
            for(auto &j : ls) c.push_back(-j * k % mod);
            if(c.size() < cur.size()) c.resize(cur.size());
            for(int j=0; j<cur.size(); j++){
                c[j] = (c[j] + cur[j]) % mod;
            }
            if(i-lf+(int)ls.size()>=(int)cur.size()){
                tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
            }
            cur = c;
        }
        for(auto &i : cur) i = (i % mod + mod) % mod;
        return cur;
    }
    int get_nth(vector<int> rec, vector<int> dp, lint n){
        int m = rec.size();
        vector<int> s(m), t(m);
        s[0] = 1;
        if(m != 1) t[1] = 1;
        else t[0] = rec[0];
        auto mul = [&rec](vector<int> v, vector<int> w){
            int m = v.size();
            vector<int> t(2 * m);
            for(int j=0; j<m; j++){
                for(int k=0; k<m; k++){
                    t[j+k] += 1ll * v[j] * w[k] % mod;
                    if(t[j+k] >= mod) t[j+k] -= mod;
                }
            }
            for(int j=2*m-1; j>=m; j--){
                for(int k=1; k<=m; k++){
                    t[j-k] += 1ll * t[j] * rec[k-1] % mod;
                    if(t[j-k] >= mod) t[j-k] -= mod;
                }
            }
            t.resize(m);
            return t;
        };
        while(n){
            if(n & 1) s = mul(s, t);
            t = mul(t, t);
            n >>= 1;
        }
        lint ret = 0;
        for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i
        return ret % mod;
    }
    int guess_nth_term(vector<int> x, lint n){
        if(n < x.size()) return x[n];
        vector<int> v = berlekamp_massey(x);
        if(v.empty()) return 0;
        return get_nth(v, x, n);
    }
    struct elem{int x, y, v;};
    vector<int> get_min_poly(int n, vector<elem> M)
    {
        vector<int> rnd1, rnd2;
        mt19937 rng(0x14004);
        auto randint = [&rng](int lb, int ub){
            return uniform_int_distribution<int>(lb, ub
        };
        for(int i=0; i<n; i++){
            rnd1.push_back(randint(1, mod - 1));
            rnd2.push_back(randint(1, mod - 1));
        }
        vector<int> gobs;
        for(int i=0; i<2*n+2; i++){
            int tmp = 0;
            for(int j=0; j<n; j++){
                tmp += 1ll * rnd2[j] * rnd1[j] % mod;
                if(tmp >= mod) tmp -= mod;
            }
            gobs.push_back(tmp);
            vector<int> nxt(n);
            for(auto &i : M){
                nxt[i.x] += 1ll * i.v * rnd1[i.y] % mo
                if(nxt[i.x] >= mod) nxt[i.x] -= mod;
            }
            rnd1 = nxt;
        }
        auto sol = berlekamp_massey(gobs);
        reverse(sol.begin(), sol.end());
        return sol;
    }
    // Usage : guess_nth_term(first_values, n);
};
```

# 4 Geometry

## 4.1 Point header

Use as `typedef Point<double> P;`

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template <class T>
struct Point {
    typedef Point P_;
    T x, y;
    explicit Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P_ p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P_ p) const { return tie(x,y)==tie(p.x,p.y); }
    P_ operator+(P_ p) const { return P_(x+p.x, y+p.y); }
    P_ operator-(P_ p) const { return P_(x-p.x, y-p.y); }
    P_ operator*(T d) const { return P_(x*d, y*d); }
    P_ operator/(T d) const { return P_(x/d, y/d); }
    T dot(P_ p) const { return x*p.x + y*p.y; }
    T cross(P_ p) const { return x*p.y - y*p.x; }
    T cross(P_ a, P_ b) const { return (a-*this).cross(b-*this); }
    T dist2() const { return x*x + y*y; }
    double dist() const { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() const { return atan2(y, x); }
    P_ unit() const { return *this/dist(); } // makes dist()=1
    P_ perp() const { return P_(-y, x); } // rotates +90 degrees
    P_ normal() const { return perp().unit(); }
    // returns point rotated ’a’ radians ccw around the origin
    P_ rotate(double a) const {
        return P_(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
    friend ostream& operator<<(ostream& os, P_ p) {
        return os << "(" << p.x << "," << p.y << ")"; }
};
```

## 4.2 Line / Line segment

`lineDist` : 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.

`lineInter, segInter` : Unique intersection : (1, point), No intersection : (0, (0, 0)), Infinitely many : (-1, (0, 0)).

`segDist` : Return shortest distance between p and segment $[s, e]$.

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

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};
}

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;
}

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)};
}

bool onSegment(P s, P e, P p) {
    return p.cross(s, e) == 0 && (s - p).dot(e - p) <=
}
```

## 4.3 Polygons

```
vector<P> convexHull(vector<P> pts)
{
    if (sz(pts) <= 1) return pts;
    sort(all(pts));
    vector<P> h(sz(pts)+1);
    int s = 0, t = 0;
    for (int it = 2; it--; s = --t, reverse(all(pts)))
        for (P p : pts) {
            while (t >= s + 2 && h[t-2].cross(h[t-1], p
            h[t++] = p;
        }
    return {h.begin(), h.begin() + t - (t == 2 && h[0]
}

//Returns two points with max distance on a convex hull
array<P, 2> hullDiameter(vector<P> S) {
    int n = sz(S), j = n < 2 ? 0 : 1;
    pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
    rep(i,0,j)
        for (;; j = (j + 1) % n) {
            res = max(res, {(S[i] - S[j]).dist2(), {S[i
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1]
                break;
        }
    return res.second;
}
```

- $(-1,-1)$ if no collision,
- $(i,-1)$ if touching the corner $i$,
- $(i,i)$ if along side $(i,i+1)$,
- $(i,j)$ if crossing sides $(i,i+1)$ and $(j,j+1)$.

```
bool inPolygon(vector<P> &p, P a, bool strict = true) {
    int cnt = 0, n = sz(p);
    rep(i,0,n) {
        P q = p[(i + 1) % n];
        if (onSegment(p[i], q, a)) return !strict;
        //or: if (segDist(p[i], q, a) <= eps) return !strict;
        cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
    }
    return cnt;
}

// Twice the signed area of polygon
template<class T>
T polygonArea2(vector<Point<T>>& v) {
    T a = v.back().cross(v[0]);
    rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
    return a;
}

#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
    return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P> poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

# 5 Graph

## 5.1 Network Flow

### 5.1.1 Dinic's Maxflow

Fast max flow algorithm. Use `maxflow` and `AddEdge`.

```
struct Edge
{
    int u, v;
    ll cap, flow;
    Edge() {}
    Edge(int u, int v, ll cap): u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic
{
    int N;
    vector<Edge> E;
    vector<vector<int>> g;
    vector<int> d, pt;

    Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

    void AddEdge(int u, int v, ll cap)
    {
        if (u != v)
        {
            E.push_back(Edge(u, v, cap));
            g[u].push_back(E.size() - 1);
            E.push_back(Edge(v, u, 0));
            g[v].push_back(E.size() - 1);
        }
    }

    bool BFS(int S, int T)
    {
        queue<int> q({S});
        fill(d.begin(), d.end(), N + 1);
        d[S] = 0;
        while(!q.empty())
        {
            int u = q.front();
            q.pop();
            if (u == T) break;
            for (int k: g[u])
            {
                Edge &e = E[k];
                if (e.flow < e.cap && d[e.v] > d[e.u] +
                {
                    d[e.v] = d[e.u] + 1;
                    q.push(e.v);
                }
            }
        }
        return d[T] != N + 1;
    }

    ll DFS(int u, int T, ll flow = -1)
    {
        if (u == T || flow == 0) return flow;
        for (int &i = pt[u]; i < g[u].size(); i++)
        {
            Edge &e = E[g[u][i]];
            Edge &oe = E[g[u][i]^1];
            if (d[e.v] == d[e.u] + 1)
            {
                ll amt = e.cap - e.flow;
                if (flow != -1 && amt > flow) amt = flo
                if (ll pushed = DFS(e.v, T, amt))
                {
                    e.flow += pushed;
                    oe.flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
    }

    ll MaxFlow(int S, int T)
    {
        ll total = 0;
        while (BFS(S, T))
        {
            fill(pt.begin(), pt.end(), 0);
            while (ll flow = DFS(S, T))
                total += flow;
        }
        return total;
    }
};
```

### 5.1.2 Min cost Max Flow

Fast min cost max flow algorithm. Use `solveMCMF` and `AddEdge`.
solveMCMF returns pair of (max flow, cost of such flow).

```
const int MAXN = 1010;
struct MCMF
{
    struct edg { int pos, cap, rev, cost; };
    vector<edg> gph[MAXN];
    void clear(){
        for(int i=0; i<MAXN; i++) gph[i].clear();
    }
    void AddEdge(int s, int e, int cap, int cst)
    // add edge (s to e, capacity = x, cost = c)
    {
        gph[s].push_back({e, cap, (int)gph[e].size(), cst});
        gph[e].push_back({s, 0, (int)gph[s].size()-1, -cst});
    }
    int dist[MAXN], pa[MAXN], pe[MAXN];
    bool inque[MAXN];
    bool spfa(int src, int sink)
    {
        memset(dist, 0x3f, sizeof(dist));
        memset(inque, 0, sizeof(inque));
        queue<int> que;
        dist[src] = 0;
        inque[src] = 1;
        que.push(src);
        bool ok = 0;
        while(!que.empty()){
            int x = que.front();
            que.pop();
            if(x == sink) ok = 1;
            inque[x] = 0;
            for(int i=0; i<gph[x].size(); i++)
            {
                edg e = gph[x][i];
                if(e.cap > 0 && dist[e.pos] > dist[x] + e.cost)
                {
                    dist[e.pos] = dist[x] + e.cost;
                    pa[e.pos] = x;
                    pe[e.pos] = i;
                    if(!inque[e.pos]){
                        inque[e.pos] = 1;
                        que.push(e.pos);
                    }
                }
            }
        }
        return ok;
    }
    pii solveMCMF(int src, int sink){
        int MCMF_COST = 0;
        int MCMF_FLOW = 0;
        while(spfa(src, sink)){
            int cap = 1e9;
            for(int pos = sink; pos != src; pos = pa[pos]){
                cap = min(cap, gph[pa[pos]][pe[pos]].cap);
            }
            MCMF_COST += dist[sink] * cap;
            for(int pos = sink; pos != src; pos = pa[po
                int rev = gph[pa[pos]][pe[pos]].rev;
                gph[pa[pos]][pe[pos]].cap -= cap;
                gph[pos][rev].cap += cap;
            }
            MCMF_FLOW += cap;
        }
        return {MCMF_FLOW, MCMF_COST};
    }
};
```

# 6 String

## 6.1 KMP

Pi 배열의 정의 : *str*[0] 부터 *str*[*i*] 까지 중 접두사가 접미사와 같은 부분만큼의 길이.

```
vector<int> getPi(string p)
{
    int j = 0;
    int plen = p.length();
    vector<int> pi;
    pi.resize(plen);
    for(int i = 1; i< plen; i++)
    {
        while((j > 0) && (p[i] !=  p[j]))
            j = pi[j-1];
        if(p[i] == p[j])
        {
            j++;
            pi[i] = j;
        }
    }
    return pi;
}
vector <int> kmp(string s, string p)
{
    vector<int> ans;
    auto pi = getPi(p);
    int slen = s.length(), plen = p.length(), j = 0;
    for(int i = 0 ; i < slen ; i++)
    {
        while(j>0 && s[i] != p[j])
```

```
            j = pi[j-1];
        if(s[i] == p[j])
        {
            if(j==plen-1)
            {
                ans.push_back(i-plen+1);
                j = pi[j];
            }
            else
                j++;
        }
    }
    return ans;
}
```

## 6.2 Manacher

A[i] = $i$ 번을 중심으로 하는 가장 긴 팰린드롬이 되는 반지름.

```
int N,A[MAXN];
char S[MAXN];

void Manachers()
{
    int r = 0, p = 0;
    for (int i=1;i<=N;i++)
    {
        if (i <= r)
            A[i] = min(A[2*p-i],r-i);
        else
            A[i] = 0;
        while (i-A[i]-1 > 0 && i+A[i]+1 <= N
        && S[i-A[i]-1] == S[i+A[i]+1])
            A[i]++;
        if (r < i+A[i])
            r = i+A[i], p = i;
    }
}
```

# 7 Misc

## 7.1 GCC Extensions : OST

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

void test()
{
   ordered_set X;
   X.insert(1);
   X.insert(2);
   X.insert(4);
   X.insert(8);
   X.insert(16);

   cout<<*X.find_by_order(1)<<endl; // 2
   cout<<*X.find_by_order(2)<<endl; // 4
   cout<<*X.find_by_order(4)<<endl; // 16
   cout<<(end(X)==X.find_by_order(6))<<endl; // true

   cout<<X.order_of_key(-5)<<endl;  // 0
   cout<<X.order_of_key(1)<<endl;   // 0
   cout<<X.order_of_key(3)<<endl;   // 2
   cout<<X.order_of_key(4)<<endl;   // 2
   cout<<X.order_of_key(400)<<endl; // 5
}
```