Team notebook

BINUS - Among

October 16, 2021

Contents

1 Data Structures

1.1 Heavy-Light Decomposition

```
#define N 300020
vector<int> adj[N];
int memo[25][N], lvl[N], subsize[N], col[N]; //col=array input
int chainHead[N], chainInd[N], baseArray[N], posInBase[N];
int chainNo, p, n;//chainHead=nodeHead,baseArray=array tree, int st[N*4]; //posInBase=convert input
     to tree indelax
void buildtree(int v, int l, int r){
   if (l == r){
      st[v] = baseArray[l];
      return;
   }
   int m = (l+r)>>1;
   buildtree(v<<1,l,m);
```

```
    buildtree(v<<1|1,m+1,r);
    st[v] = st[v<<1]+st[v<<1|1];
}

void updatetree(int v, int l, int r, int x){
    if(l == r){
        st[v] = baseArray[x]; return;
    }
    int m = (l+r)>>1;
    if(x <= m) updatetree(v<<1,l,m,x);
    else updatetree(v<<1|1,m+1,r,x);
    st[v] = st[v<<1]+st[v<<1|1];
}
int querytree(int v, int l, int r, int ss, int se){
    if(ss > se) return 0;
    if (l == ss && r == se) return st[v];
    int m = (l+r)>>1;
    int ans = querytree(v << 1, l, m, ss, min(se,m)) + querytree(v << 1|1, m+1, r, max(m+1,ss), se);
    return ans;
}
void dfs(int cur, int par){
    lvl[cur] = lvl[par]+1;
    memo[0][cur] = par;
    subsize[cur] = 1;
    for(int to : adj[cur]){
        if (to == par) continue;
        dfs(to,cur);
        subsize[cur] += subsize[to];
    }
}
void HLD(int cur, int par){
    if(chainHead[chainNo] == -1) chainHead[chainNo] = cur;
    chainInd[cur] = chainNo;
    baseArray[p] = col[cur];
    posInBase[cur] = p++;
    int maksto = -1;
    for(int to : adj[cur]){
        if (to == par) continue;
        if (maksto == -1 || subsize[maksto] < subsize[to]){
            maksto = to;
        }
    }
    if (maksto != -1) HLD(maksto,cur);
    for(int to : adj[cur]){
        if (to == par || to == maksto) continue;
        chainNo++;
        HLD(to,cur);
    }
}
int queryup(int u, int v){
    int ans = 0;
    while(u != v){
        if (chainInd[u] == chainInd[v]){
            ans += querytree(1,0,n-1,posInBase[v]+1,posInBase[u]);
            break;
        } else {
            ans += querytree(1,0,n-1, posInBase[chainHead[chainInd[u]]] ,posInBase[u]);
            u = chainHead[chainInd[u]];
            u = memo[0][u];
        }
    }
    return ans;
}

int main()
{
    rep(i,0,n-1){ //init
        col[i] = s[i]-'0';
        chainHead[i] = -1;
        adj[i].clear();
    }
    chainNo = p = 0;
    // add edge here
    dfs(0,0); // 0-based
    sparsing();
    HLD(0,0);
    buildtree(1,0,n-1);
    return 0;
}
```

1.2 Li Chao Tree

```
typedef long long int TD;
const TD INF = 10000000000000;
namespace LICHAO {
    struct Node {
        TD m, c;
        Node *l, *r;
    };
    Node *newNode(Node *x = NULL) {
        Node *ret = (Node*)malloc(sizeof(Node));
        if (x) ret->m = x->m, ret->c = x->c;
        ret->l = ret->r = NULL;
        return ret;
    }
    void update(Node *k, TD l, TD r, TD m, TD c) {
        TD mid = l + r >> 1;
        bool le = m*l + c < k->m*l + k->c;
        bool ri = m*mid + c < k->m*mid + k->c;
        if (ri) swap(k->m, m), swap(k->c, c);
        if (r - l <= 1) return;
        else if (le != ri) update((k->l)?(k->l):(k->l=newNode(k)), l, mid, m, c);
        else update((k->r)?(k->r):(k->r=newNode(k)), mid, r, m, c);
    }
    TD query(Node *k, TD l, TD r, TD p) {
        if (!k) return INF;
        if (r - l <= 1) return p*k->m + k->c;
        if (p < (l+r >> 1)) return min(p*k->m + k->c, query(k->l, l, l+r>>1, p));
        else return min(p*k->m + k->c, query(k->r, l+r>>1, r, p));
    }
}
```

1.3 Persistent Segment Tree

```cpp
class PersistentSegtree {
    private:
        int n, ptr, sz;
        struct P {
            int val = 0, l, r;
        };
        vector<P> node;
        vector<int> root;

        int newNode() { return ptr++; }
        int copyNode(int idx) {
            node[ptr] = node[idx];
            return ptr++;
        }
        int build(int l, int r) {
            int idx = newNode();
            if(l == r) return idx;
            node[idx].l = build(l, (l+r)/2);
            node[idx].r = build((l+r)/2+1, r);
            return idx;
        }
        int update(int idx, int l, int r, int x, int val) {
            idx = copyNode(idx);
            if(l == r) {
                node[idx].val += val;
                return idx;
            }
            int mid = (l+r)/2;
            if(x <= mid) node[idx].l = update(node[idx].l, l, mid, x, val);
            else node[idx].r = update(node[idx].r, mid+1, r, x, val);
            node[idx].val = node[node[idx].l].val + node[node[idx].r].val;
            return idx;
        }
        int query(int idxl, int idxr, int l, int r, int x, int y) {
            if(y < l || r < x) return 0;
            if(x <= l && r <= y) return node[idxr].val - node[idxl].val;
            int mid = (l+r)/2;
            return query(node[idxl].l, node[idxr].l, l, mid, x, y)
                + query(node[idxl].r, node[idxr].r, mid+1, r, x, y);
        }

    public:
        PersistentSegtree(int _n) : n(_n), ptr(0) {
            sz = 30 * n;
            node.resize(sz);
            root.push_back(build(1, n));
        }
        void update(int x, int val) {
            root.push_back(update(root.back(), 1, n, x, val));
        }
        int query(int l, int r, int x, int y) {
            return query(root[l-1], root[r], 1, n, x, y);
        }
};
```

1.4 STL PBDS

```cpp
// ost = ordered set
// omp = ordered map
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<class T>
using ost = tree<T, null_type, less<T>, rb_tree_tag,
               tree_order_statistics_node_update>;
template<class T, class U>
using omp = tree<T, U, less<T>, rb_tree_tag,
               tree_order_statistics_node_update>;
```

1.5 Treap

```cpp
// Complexity: O(log N) for split and merge
//
// empty treap: Treap* tr = nullptr;
// insert v at x: [l, r] = split(tr, x), m = Treap(v), merge lmr
// delete at x: [l, r] = split(tr, x), [m, r] = split(r, 1), merge lr
// lazy prop: propagate every time a node is accessed

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

using Key = int;

struct Treap
{
    Key val;
    Treap* left;
    Treap* right;
    int prio, sz;
    Treap() {}
    Treap(int _val);
};

int size(Treap* tr)
{
    return tr ? tr->sz : 0;
}

void update(Treap* tr)
{
    tr->sz = 1 + size(tr->left) + size(tr->right);
}

Treap::Treap(Key _val) :
    val(_val), left(nullptr), right(nullptr), prio(rng())
{
    update(this);
}

pair<Treap*, Treap*> split(Treap* tr, int sz)
{
    if(!tr) return {nullptr, nullptr};
```

```
    int left_sz = size(tr->left);
    if(sz <= left_sz)
    {
        auto [left, mid] = split(tr->left, sz);
        tr->left = mid;
        update(tr);
        return {left, tr};
    }
    else
    {
        auto [mid, right] = split(tr->right, sz - left_sz - 1);
        tr->right = mid;
        update(tr);
        return {tr, right};
    }
}

Treap* merge(Treap* l, Treap* r)
{
    if(!l) return r;
    if(!r) return l;
    if(l->prio < r->prio)
    {
        l->right = merge(l->right, r);
        update(l);
        return l;
    }
    else
    {
        r->left = merge(l, r->left);
        update(r);
        return r;
    }
}
```

1.6 Unordered Map Custom Hash

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

unordered_map<int, int, custom_hash> umap;
```

2 Dynamic Programming

2.1 DP Convex Hull

```
/* dp[i] = min k<i {dp[k] + x[i]*m[k]}
   Make sure gradient (m[i]) is either non-increasing if min,
   or non-decreasing if max. x[i] must be non-decreasing. just sort */
int y[N], m[N];
// while this is true, pop back from dq. a=new line, b=last, c=2nd last
bool cekx(int a, int b, int c){
    // if not enough, change to cross mul
    // if cross mul, beware of negative denominator, and overflow
    return (double)(y[b]-y[a])/(m[a]-m[b])<=(double)(y[c]-y[b])/(m[b]-m[c]);
}
```

2.2 DP DNC

```
void f(int rem,int l,int r,int optl,int optr){
    if(l>r)return;
    int mid = l+r>>1;
    int opt = MOD, optid = mid;
    for(int i = optl; i<=mid && i<=optr; ++i){
        if(dp[rem-1][i] + c[i][mid] < opt){
            opt = dp[rem-1][i] + c[i][mid];
            optid = i;
        }
    }
    dp[rem][mid] = opt;
    f(rem, l, mid-1, optl, optid);
    f(rem, mid+1, r, optid, optr); return;
}
rep(i,1,n)dp[1][i] = c[0][i];
rep(i,2,k)f(i,i,n,i,n);
```

2.3 Knuth-Yao

```
// opt[i+1][j] <= opt[i][j] <= opt[i][j+1]
// dp[i][j] = min{k} dp[i][k]+dp[k][j]+cost[i][j]
for (int k = 0; k <= n; k++) {
    for (int i = 0; i + k <= n; i++) {
        if (k < 2) { dp[i][i+k] = 0, opt[i][i + k] = i; }
        else {
            int sta = opt[i][i + k - 1];
            int end = opt[i + 1][i + k];
            for (int j = sta; j <= end; j++) {
                if (dp[i][j] + dp[j][i + k] + cost[i][i + k] < dp[i][i + k]) {
                    dp[i][i + k] = dp[i][j] + dp[j][i + k] + cost[i][i + k];
                    opt[i][i + k] = j;
                }
            }
        }
    }
```

```
}
```

3 Geometry

3.1 Closest Pair of Points

```
#define fi first
#define se second
typedef pair<int, int> pii;
struct Point{
    int x, y, id;
};
int compareX(const void* a, const void* b){
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b){
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}
double dist(Point p1, Point p2) {
    return sqrt( (double)(p1.x - p2.x)*(p1.x - p2.x) +
                (double)(p1.y - p2.y)*(p1.y - p2.y)
               );
}
pair<pii, double> bruteForce(Point P[], int n){
    double min = 1e8;
    pii ret=pii(-1, -1);
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min){
                ret=pii(P[i].id, P[j].id);
                min = dist(P[i], P[j]);
            }
    return pair<pii, double> (ret, min);
}
pair<pii, double> getmin(pair<pii, double> x, pair<pii, double> y){
    if(x.fi.fi==-1 && x.fi.se==-1) return y;
    if(y.fi.fi==-1 && y.fi.se==-1) return x;
    return (x.se < y.se)? x : y;
}
pair<pii, double> stripClosest(Point strip[], int size, double d){
    double min = d;
    pii ret=pii(-1, -1);
    qsort(strip, size, sizeof(Point), compareY);
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min){
                ret=pii(strip[i].id, strip[j].id);
                min = dist(strip[i], strip[j]);
            }
    return pair<pii, double>(ret, min);
}
pair<pii, double> closestUtil(Point P[], int n){
    if (n <= 3) return bruteForce(P, n);
    int mid = n/2;
    Point midPoint = P[mid];
    pair<pii, double> dl = closestUtil(P, mid);
    pair<pii, double> dr = closestUtil(P + mid, n-mid);
    pair<pii, double> d = getmin(dl, dr);
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].x - midPoint.x) < d.second)
            strip[j] = P[i], j++;

    return getmin(d, stripClosest(strip, j, d.second));
}
pair<pii, double> closest(Point P[], int n){
    qsort(P, n, sizeof(Point), compareX);
    return closestUtil(P, n);
}
Point P[50005];
int main(){
    int n;
    scanf("%d", &n);
    for(int a=0;a<n;a++){
        scanf("%d%d", &P[a].x, &P[a].y);
        P[a].id=a;
    }
    pair<pii, double> hasil=closest(P, n);
    if(hasil.fi.fi > hasil.fi.se) swap(hasil.fi.fi, hasil.fi.se);
    printf("%d %d %.6lf\n", hasil.fi.fi, hasil.fi.se, hasil.se);
    return 0;
}
```

3.2 Convex Hull

```
typedef double TD;              // for precision shits
namespace GEOM {
    typedef pair<TD,TD> Pt;      // vector and points
    const TD EPS = 1e-9;
    const TD maxD = 1e9;
    TD cross(Pt a, Pt b, Pt c) { // right hand rule
        TD v1 = a.first - c.first; // (a-c) X (b-c)
        TD v2 = a.second - c.second;
        TD u1 = b.first - c.first;
        TD u2 = b.second - c.second;
        return v1 * u2 - v2 * u1;
    }
    TD cross(Pt a, Pt b) {       // a X b
        return a.first*b.second - a.second*b.first;
    }
    TD dot(Pt a, Pt b, Pt c) {   // (a-c) . (b-c)
        TD v1 = a.first - c.first;
        TD v2 = a.second - c.second;
        TD u1 = b.first - c.first;
        TD u2 = b.second - c.second;
        return v1 * u1 + v2 * u2;
    }
    TD dot(Pt a, Pt b) {         // a . b
        return a.first*b.first + a.second*b.second;
    }
```

```cpp
TD dist(Pt a, Pt b) {
    return sqrt((a.first-b.first)*(a.first-b.first) + (a.second-b.second)*(a.second-b.second));
}
TD shoelaceX2(vector<Pt> &convHull) {
    TD ret = 0;
    for (int i = 0, n = convHull.size(); i < n; i++)
        ret += cross(convHull[i], convHull[(i+1)%n]);
    return ret;
}
vector<Pt> createConvexHull(vector<Pt> &points) {
    sort(points.begin(), points.end());
    vector<Pt> ret;
    for (int i = 0; i < points.size(); i++) {
        while (ret.size() > 1 &&
                cross(points[i], ret[ret.size()-1], ret[ret.size()-2]) < -EPS)
            ret.pop_back();
        ret.push_back(points[i]);
    }
    for (int i = points.size() - 2, sz = ret.size(); i >= 0; i--) {
        while (ret.size() > sz &&
                cross(points[i], ret[ret.size()-1], ret[ret.size()-2]) < -EPS)
            ret.pop_back();
        if (i == 0) break;
        ret.push_back(points[i]);
    }
    return ret;
    bool isInside(Pt pv, vector<Pt> &x){//using winding number
        int n = x.size(), wn = 0;
        x.push_back(x[0]);

        for(int i = 0; i<n; ++i){
            if(((x[i+1].first<=pv.first&&x[i].first>=pv.first)||
                      (x[i+1].first>=pv.first&&x[i].first<=pv.first)) &&
                  ((x[i+1].second<=pv.second&&x[i].second>=pv.second)||
                   (x[i+1].second>=pv.second&&x[i].second<=pv.second)) ){
                if(cross(x[i],x[i+1],pv) == 0){
                    x.pop_back();
                    return true;
                }
            }
        }
        for(int i = 0; i<n; ++i){
            if(x[i].second <= pv.second) {
                if(x[i+1].second>pv.second && cross(x[i],x[i+1],pv)>0)++wn;
            }
            else if(x[i+1].second<=pv.second && cross(x[i],x[i+1],pv)<0)--wn;
        }
        x.pop_back();
        return wn!=0;
    }
}
bool isInside(Pt pv, vector<Pt> &x){//using winding number
    int n = x.size(), wn = 0;
    x.push_back(x[0]);

    for(int i = 0; i<n; ++i){
        if(((x[i+1].first<=pv.first&&x[i].first>=pv.first)||
                  (x[i+1].first>=pv.first&&x[i].first<=pv.first)) &&
              ((x[i+1].second<=pv.second&&x[i].second>=pv.second)||
               (x[i+1].second>=pv.second&&x[i].second<=pv.second)) ){
            if(cross(x[i],x[i+1],pv) == 0){
                x.pop_back();
                return true;
            }
        }
    }
    for(int i = 0; i<n; ++i){
        if(x[i].second <= pv.second) {
            if(x[i+1].second>pv.second && cross(x[i],x[i+1],pv)>0)++wn;
        }
        else if(x[i+1].second<=pv.second && cross(x[i],x[i+1],pv)<0)--wn;
    }
    x.pop_back();
    return wn!=0;
}
}
```

3.3 Geometry Template

```cpp
/*
TABLE OF CONTENT
0. Basic Rule
    0.1. Everything is in double
    0.2. Every comparison use EPS
    0.3. Every degree in rad
1. General Double Operation
    1.1. const double EPS=1E-9
    1.2. const double PI=acos(-1.0)
    1.3. const double INFD=1E9
    1.3. between_d(double x,double l,double r)
        check whether x is between l and r inclusive with EPS
    1.4. same_d(double x,double y)
        check whether x=y with EPS
    1.5. dabs(double x)
        absolute value of x
2. Point
    2.1. struct point
        2.1.1. double x,y
            cartesian coordinate of the point
        2.1.2. point()
            default constructor
        2.1.3. point(double _x,double _y)
            constructor, set the point to (_x,_y)
        2.1.4. bool operator< (point other)
            regular pair<double,double> operator < with EPS
        2.1.5. bool operator== (point other)
            regular pair<double,double> operator == with EPS
    2.2. hypot(point P)
        length of hypotenuse of point P to (0,0)
    2.3. e_dist(point P1,point P2)
        euclidean distance from P1 to P2
    2.4. m_dist(point P1,point P2)
        manhattan distance from P1 to P2
    2.5. point rotate(point P,point O,double angle)
        rotate point P from the origin O by angle ccw
3. Vector
    3.1. struct vec
```

```
        3.1.1. double x,y
            x and y magnitude of the vector
        3.1.2. vec()
            default constructor
        3.1.3. vec(double _x,double _y)
            constructor, set the vector to (_x,_y)
        3.1.4. vec(point A,point B)
            constructor, set the vector to vector AB (A->B)
*/
/*General Double Operation*/

const double PI=acos(-1.0);
const double INFD=1E9;
double between_d(double x,double l,double r) {
    return (min(l,r)<=x+EPS && x<=max(l,r)+EPS);
}
double same_d(double x,double y) {
    return between_d(x,y,y);
}
double dabs(double x) {
    if (x<EPS) return -x; return x;
}
/*Point*/
struct point {
    double x,y;
    point() {
        x=y=0.0;
    }
    point(double _x,double _y) {
        x=_x; y=_y;
    }
    bool operator< (point other) {
        if (x<other.x+EPS) return true;
        if (x+EPS>other.x) return false;
        return y<other.y+EPS;
    }
    bool operator== (point other) {
        return same_d(x,other.x)&&same_d(y,other.y);
    }
};
double e_dist(point P1,point P2) {
    return hypot(P1.x-P2.x,P1.y-P2.y);
}
double m_dist(point P1,point P2) {
    return dabs(P1.x-P2.x)+dabs(P1.y-P2.y);
}
double pointBetween(point P,point L,point R) {
    return (e_dist(L,P)+e_dist(P,R)==e_dist(L,R));
}
bool collinear(point P, point L, point R) { //newly added(luis), cek 3 poin segaris
    return P.x*(L.y-R.y)+L.x*(R.y-P.y)+R.x*(P.y-L.y)==0; // bole gnti dabs(x)<EPS
}

/*Vector*/
struct vec {
    double x,y;
    vec() {
        x=y=0.0;
    }
    vec(double _x,double _y) {
        x=_x; y=_y;
    }
    vec(point A) {
        x=A.x; y=A.y;
    }
    vec(point A,point B) {
        x=B.x-A.x; y=B.y-A.y;
    }
};
vec scale(vec v,double s) {
    return vec(v.x*s,v.y*s);
}
vec flip(vec v) {
    return vec(-v.x,-v.y);
}
double dot(vec u,vec v) {
    return (u.x*v.x+u.y*v.y);
}
double cross(vec u,vec v) {
    return (u.x*v.y-u.y*v.x);
}
double norm_sq(vec v) {
    return (v.x*v.x+v.y*v.y);
}
point translate(point P,vec v) {
    return point(P.x+v.x,P.y+v.y);
}
point rotate(point P,point O,double angle) {
    vec v(O); P=translate(P,flip(v));
    return translate(point(P.x*cos(angle)-P.y*sin(angle),P.x*sin(angle)+P.y*cos(angle)),v);
}
point mid(point P,point Q) {
    return point((P.x+Q.x)/2,(P.y+Q.y)/2);
}
double angle(point A,point O,point B) {
    vec OA(O,A), OB(O,B);
    return acos(dot(OA,OB)/sqrt(norm_sq(OA)*norm_sq(OB)));
}
int orientation(point P,point Q,point R) {
    vec PQ(P,Q), PR(P,R);
    double c=cross(PQ,PR);
    if (c<-EPS) return -1;
    if (c>EPS) return 1;
    return 0;
}
/*Line*/
struct line {
    double a,b,c;
    line() {
        a=b=c=0.0;
    }
    line(double _a,double _b,double _c) {
        a=_a; b=_b; c=_c;
    }
    line(point P1,point P2) {
        if (P1<P2) swap(P1,P2);
        if (same_d(P1.x,P2.x)) a=1.0, b=0.0, c=-P1.x;
        else a=-(P1.y-P2.y)/(P1.x-P2.x), b=1.0, c=-(a*P1.x)-P1.y;
    }
    line (point P,double slope) {
```

```
        if (same_d(slope,INFD)) a=1.0, b=0.0, c=-P.x;
        else a=-slope, b=1.0, c=-(a*P.x)-P.y;
    }
    bool operator== (line other) {
        return same_d(a,other.a)&&same_d(b,other.b)&&same_d(c,other.c);
    }
    double slope() {
        if (same_d(b,0.0)) return INFD;
        return -(a/b);
    }
};
bool paralel(line L1,line L2) {
    return same_d(L1.a,L2.a)&&same_d(L1.b,L2.b);
}
bool intersection(line L1,line L2,point &P) {
    if (paralel(L1,L2)) return false;
    P.x=(L2.b*L1.c-L1.b*L2.c)/(L2.a*L1.b-L1.a*L2.b);
    if (same_d(L1.b,0.0)) P.y=-(L2.a*P.x+L2.c);
    else P.y=-(L1.a*P.x+L1.c);
    return true;
}
double pointToLine(point P,point A,point B,point &C) {
    vec AP(A,P), AB(A,B);
    double u=dot(AP,AB)/norm_sq(AB);
    C=translate(A,scale(AB,u));
    return e_dist(P,C);
}
double lineToLine(line L1,line L2) {
    if (!paralel(L1,L2)) return 0.0;
    return dabs(L2.c-L1.c)/sqrt(L1.a*L1.a+L1.b*L1.b);
}
/*Line Segment*/
struct segment {
    point P,Q;
    line L;
    segment() {
        point T1; P=Q=T1;
        line T2; L=T2;
    }
    segment(point _P,point _Q) {
        P=_P; Q=_Q;
        if (Q<P) swap(P,Q);
        line T(P,Q); L=T;
    }
    bool operator== (segment other) {
        return P==other.P&&Q==other.Q;
    }
};
bool onSegment(point P,segment S) {
    if (orientation(S.P,S.Q,P)!=0) return false;
    return between_d(P.x,S.P.x,S.Q.x) && between_d(P.y,S.P.y,S.Q.y);
}
bool s_intersection(segment S1,segment S2) {
    double o1=orientation(S1.P,S1.Q,S2.P);
    double o2=orientation(S1.P,S1.Q,S2.Q);
    double o3=orientation(S2.P,S2.Q,S1.P);
    double o4=orientation(S2.P,S2.Q,S1.Q);
    if (o1!=o2 && o3!=o4) return true;
    if (o1==0 && onSegment(S2.P,S1)) return true;
    if (o2==0 && onSegment(S2.Q,S1)) return true;
```

```
    if (o3==0 && onSegment(S1.P,S2)) return true;
    if (o4==0 && onSegment(S1.Q,S2)) return true;
    return false;
}
double pointToSegment(point P,point A,point B,point &C) {
    vec AP(A,P), AB(A,B);
    double u=dot(AP,AB)/norm_sq(AB);
    if (u<EPS) {
        C=A; return e_dist(P,A);
    }
    if (u+EPS>1.0) {
        C=B; return e_dist(P,B);
    }
    return pointToLine(P,A,B,C);
}
double segmentToSegment(segment S1,segment S2) {
    if (s_intersection(S1,S2)) return 0.0;
    double ret=INFD; point dummy;
    ret=min(ret,pointToSegment(S1.P,S2.P,S2.Q,dummy));
    ret=min(ret,pointToSegment(S1.Q,S2.P,S2.Q,dummy));
    ret=min(ret,pointToSegment(S2.P,S1.P,S1.Q,dummy));
    ret=min(ret,pointToSegment(S2.Q,S1.P,S1.Q,dummy));
    return ret;
}
/*Circle*/
struct circle {
    point P;
    double r;
    circle() {
        point P1; P=P1;
        r=0.0;
    }
    circle(point _P,double _r) {
        P=_P; r=_r;
    }
    circle(point P1,point P2) {
        P=mid(P1,P2); r=e_dist(P,P1);
    }
    circle(point P1,point P2,point P3) {
        vector<point> T; T.clear(); T.pb(P1); T.pb(P2); T.pb(P3);
        sort(T.begin(),T.end());
        P1=T[0]; P2=T[1]; P3=T[2];
        point M1,M2; M1=mid(P1,P2); M2=mid(P2,P3);
        point Q2,Q3; Q2=rotate(P2,P1,PI/2); Q3=rotate(P3,P2,PI/2);
        vec P1Q2(P1,Q2), P2Q3(P2,Q3);
        point M3,M4; M3=translate(M1,P1Q2); M4=translate(M2,P2Q3);
        line L1(M1,M3), L2(M2,M4);
        intersection(L1,L2,P); r=e_dist(P,P1);
    }
    bool operator==(circle other) {
        return (P==other.P && same_d(r,other.r));
    }
};
bool insideCircle(point P,circle C) {
    return e_dist(P,C.P)<=C.r+EPS;
}
bool c_intersection(circle C1,circle C2,point &P1,point &P2) {
    double d=e_dist(C1.P,C2.P);
    if (d>C1.r+C2.r) return false;//d+EPS kalo butuh
    if (d<dabs(C1.r-C2.r)+EPS) return false;
```

```
    double x1=C1.P.x, y1=C1.P.y, r1=C1.r, x2=C2.P.x, y2=C2.P.y, r2=C2.r;
    double a=(r1*r1-r2*r2+d*d)/(2*d), h=sqrt(r1*r1-a*a);
    point T(x1+a*(x2-x1)/d,y1+a*(y2-y1)/d);
    P1=point(T.x-h*(y2-y1)/d,T.y+h*(x2-x1)/d);
    P2=point(T.x+h*(y2-y1)/d,T.y-h*(x2-x1)/d);
    return true;
}
bool lc_intersection(line L,circle O,point &P1,point &P2) {
    double a=L.a, b=L.b, c=L.c, x=O.P.x, y=O.P.y, r=O.r;
    double A=a*a+b*b, B=2*a*b*y-2*a*c-2*b*b*x, C=b*b*x*x+b*b*y*y-2*b*c*y+c*c-b*b*r*r;
    double D=B*B-4*A*C; point T1,T2;
    if (same_d(b,0.0)) {
        T1.x=c/a;
        if (dabs(x-T1.x)+EPS>r) return false;
        if (same_d(T1.x-r-x,0.0) || same_d(T1.x+r-x,0.0)) {
            P1=P2=point(T1.x,y); return true;
        }
        double dx=dabs(T1.x-x), dy=sqrt(r*r-dx*dx);
        P1=point(T1.x,y-dy); P2=point(T1.x,y+dy); return true;
    }
    if (same_d(D,0.0)) {
        T1.x=-B/(2*A); T1.y=(c-a*T1.x)/b; P1=P2=T1; return true;
    }
    if (D<EPS) return false;
    D=sqrt(D);
    T1.x=(-B-D)/(2*A);T1.y=(c-a*T1.x)/b; P1=T1;
    T2.x=(-B+D)/(2*A);T2.y=(c-a*T2.x)/b; P2=T2; return true;
}
bool sc_intersection(segment S,circle C,point &P1,point &P2) {
    bool cek=lc_intersection(S.L,C,P1,P2);
    if (!cek) return false;
    double x1=S.P.x, y1=S.P.y, x2=S.Q.x, y2=S.Q.y;
    bool b1=between_d(P1.x,x1,x2)&&between_d(P1.y,y1,y2);
    bool b2=between_d(P2.x,x1,x2)&&between_d(P2.y,y1,y2);
    if (P1==P2) return b1;
    if (b1||b2) {
        if (!b1) P1=P2; if (!b2) P2=P1; return true;
    }
    return false;
}
/*Triangle*/
double t_perimeter(point A,point B,point C) {
    return e_dist(A,B)+e_dist(B,C)+e_dist(C,A);
}
double t_area(point A,point B,point C) {
    double s=t_perimeter(A,B,C)/2;
    double ab=e_dist(A,B), bc=e_dist(B,C), ac=e_dist(C,A);
    return sqrt(s*(s-ab)*(s-bc)*(s-ac));
}
circle t_inCircle(point A,point B,point C) {
    vector<point> T; T.clear(); T.pb(A); T.pb(B); T.pb(C); sort(T.begin(),T.end());
    A=T[0]; B=T[1]; C=T[2];
    double r=t_area(A,B,C)/(t_perimeter(A,B,C)/2);
    double ratio=e_dist(A,B)/e_dist(A,C);
    vec BC(B,C); BC=scale(BC,ratio/(1+ratio));
    point P; P=translate(B,BC); line AP1(A,P);
    ratio=e_dist(B,A)/e_dist(B,C);
    vec AC(A,C); AC=scale(AC,ratio/(1+ratio));
    P=translate(A,AC); line BP2(B,P);
    intersection(AP1,BP2,P); return circle(P,r);
}
circle t_outCircle(point A,point B,point C) {
    return circle(A,B,C);
}
/*Polygon*/
struct polygon {
    vector<point> P;
    polygon() {
        P.clear();
    }
    polygon(vector<point> &_P) {
        P=_P;
    }
};
bool rayCast(point P,polygon &A) {
    point Q(P.x,10000);
    line cast(P,Q);
    int cnt=0;
    FOR(i,(int)(A.P.size())-1){
        line temp(A.P[i],A.P[i+1]);
        point I;
         bool B=intersection(cast,temp,I);
        if (!B) continue;
        else if (I==A.P[i]||I==A.P[i+1]) continue;
        else if (pointBetween(I,A.P[i],A.P[i+1])&&pointBetween(I,P,Q)) cnt++;
    }
    return cnt%2==1;
}
// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
    double a = B.y - A.y;
    double b = A.x - B.x;
    double c = B.x * A.y - A.x * B.y;
    double u = fabs(a * p.x + b * p.y + c);
    double 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));
}
// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
    vector<point> P;
    for (int i=0;i<(int)Q.size();i++) {
        double left1 = cross(toVec(a,b),toVec(a,Q[i]));
        double left2 = 0;
        if (i!=(int)Q.size()-1) left2 = cross(toVec(a,b),toVec(a,Q[i+1]));
        if (left1 > -EPS) P.push_back(Q[i]);
        if (left1*left2 < -EPS) {
            P.push_back(lineIntersectSeg(Q[i],Q[i+1],a,b));
        }
    }
    if (!P.empty() && !(P.back()==P.front())) {
        P.push_back(P.front());
    }
    return P;
}
circle minCoverCircle(polygon &A) {
    vector<point> p=A.P;
    point c; circle ret;
    double cr = 0.0;
    int i, j, k;
```

```
    c = p[0];
    for(i = 1; i < p.size(); i++) {
        if(e_dist(p[i], c) >= cr+EPS) {
            c = p[i], cr = 0;
            ret=circle(c,cr);

            for(j = 0; j < i; j++) {
                if(e_dist(p[j], c) >= cr+EPS) {
                    c=mid(p[i],p[j]);
                    cr = e_dist(p[i], c);
                    ret=circle(c,cr);
                    for(k = 0; k < j; k++) {
                        if(e_dist(p[k], c) >= cr+EPS) {
                            ret=circle(p[i],p[j],p[k]);
                            c=ret.P; cr=ret.r;
                        }
                    }
                }
            }
        }
    }
    return ret;
}
/*Geometry Algorithm*/
double DP[110][110];
double minCostPolygonTriangulation(polygon &A) {
    if (A.P.size()<3) return 0;
    FOR(i,A.P.size()) {
          for (int j=0,k=i;k<A.P.size();j++,k++) {
              if (k<j+2) DP[j][k]=0.0;
             else {
                DP[j][k]=INFD;
                REP(l,j+1,k-1) {
                    double cost=e_dist(A.P[j],A.P[k])+e_dist(A.P[k],A.P[l])+e_dist(A.P[l],A.P[j]);
                    DP[j][k]=min(DP[j][k],DP[j][l]+DP[l][k]+cost);
                }
            }
        }
    }
    return DP[0][A.P.size()-1];
}
```

3.4 Smallest Enclosing Circle

```
// Welzl's algorithm to find the smallest circle
// that encloses a group of poins in O(N * ITERS)
// returns {radius, x, y}
const int ITERS = 3e5;
const double INF = 1e12;

tuple<double, double, double> welzl(const vector<pair<int, int>>& points)
{
    double xt = 0, yt = 0;
    for(auto& [x, y] : points)
    {
        xt += x;
        yt += y;
    }
    xt /= points.size();
    yt /= points.size();
    double p = 0.1;
    double mx_d;
    for(int i = 0; i < ITERS; ++i)
    {
        mx_d = -INF;
        int mx_idx = -1;
        for(int j = 0; j < (int) points.size(); ++j)
        {
            double cx = xt - points[j].first;
            double cy = yt - points[j].second;
            double cur = cx * cx + cy * cy;
            if(cur > mx_d)
            {
                mx_d = cur;
                mx_idx = j;
            }
        }
        xt += (points[mx_idx].first - xt) * p;
        yt += (points[mx_idx].second - yt) * p;
        p *= 0.999;
    }
    return {sqrt(mx_d), xt, yt};
}
```

3.5 Sutherland-Hodgman Algorithm

```
// Complexity: linear time
// Ada 2 poligon, cari poligon intersectionnya
// poly_point = hasilnya, clipper = pemotongnya
#include<bits/stdc++.h>
using namespace std;

const double EPS = 1e-9;

struct point{
    double x,y;
    point(double _x,double _y):x(_x),y(_y){}
};
struct vec {
    double x,y;
    vec(double _x, double _y):x(_x),y(_y){}
};

point pivot(0,0);
vec toVec(point a, point b){
    return vec(b.x-a.x,b.y-a.y);
}
double dist (point a, point b){
    return hypot(a.x-b.x,a.y-b.y);
}
double cross (vec a, vec b){
    return a.x*b.y-a.y*b.x;
}
bool ccw (point p, point q, point r){
```

```
    return cross(toVec(p,q),toVec(p,r)) > 0;
}
bool collinear (point p, point q, point r){
    return fabs(cross(toVec(p,q),toVec(p,r))) < EPS;
}
bool lies(point a, point b, point c){
    if ((c.x >= min(a.x,b.x) && c.x <= max(a.x,b.x)) &&
          (c.y >= min(a.y,b.y) && c.y<= max(a.y,b.y))){
        return true;
    } else return false;
}
bool anglecmp(point a, point b){
    if (collinear(pivot,a,b)) return dist(pivot,a)<dist(pivot,b);
    double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
    double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
    return (atan2(d1y,d1x) - atan2(d2y,d2x))<0;
}

point intersect (point s1, point e1,point s2, point e2){
    double x1,x2,x3,x4,y1,y2,y3,y4;
    x1 = s1.x; y1 = s1.y;
    x2 = e1.x; y2 = e1.y;
    x3 = s2.x; y3 = s2.y;
    x4 = e2.x; y4 = e2.y;
    double num1 = (x1*y2 - y1*x2) * (x3-x4) - (x1-x2) * (x3*y4 - y3*x4);
    double num2 = (x1*y2 - y1*x2) * (y3-y4) - (y1-y2) * (x3*y4 - y3*x4);
    double den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    double new_x = num1/den;
    double new_y = num2/den;
    return point(new_x,new_y);
}

void clip(vector <point> &poly_points, point point1, point point2){
    vector <point> new_points;
    new_points.clear();
    for (int i = 0; i < poly_points.size(); i++)
    {
        int k = (i+1) % poly_points.size();
        double i_pos = ccw(point1,point2,poly_points[i]);
        double k_pos = ccw(point1,point2,poly_points[k]);
        //in in
        if (i_pos <= 0 && k_pos <= 0)
        {
            new_points.push_back(poly_points[k]);
        }
        //out in
        else if (i_pos > 0 && k_pos <= 0)
        {       new_points.push_back(intersect(point1,point2,poly_points[i],poly_points[k]));
            new_points.push_back(poly_points[k]);
        }

        // in out
        else if (i_pos <= 0 && k_pos > 0)
        {       new_points.push_back(intersect(point1,point2,poly_points[i],poly_points[k]));
        }

        //out out
        else
        {
            }
    }
    poly_points.clear();
    for (int i = 0; i < new_points.size(); i++)
        poly_points.push_back(new_points[i]);
}
double area (const vector <point> &P){
    double result =0.0;
    double x1,y1,x2,y2;
    for (int i =0; i<P.size()-1;i++){
        x1 = P[i].x;
        y1 = P[i].y;
        x2 = P[i+1].x;
        y2 = P[i+1].y;
        result += (x1*y2-x2*y1);
    }
    return fabs(result)/2;
}
void suthHodgClip(vector <point> &poly_points, vector <point> clipper_points){
    for (int i=0; i<clipper_points.size(); i++)
    {
        int k = (i+1) % clipper_points.size();
        clip(poly_points, clipper_points[i], clipper_points[k]);
    }
}
vector<point> sortku (vector<point> P){
    int P0=0;
    int i;
    for (i = 1; i<3; 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;

    pivot = P[0];
    sort(++P.begin(),P.end(),anglecmp);
    reverse(++P.begin(),P.end());
    return P;
}
int main{
    clipper_points = sortku(clipper_points);
    suthHodgClip(poly_points, clipper_points);
}
```

4 Graphs

4.1 Articulation Point and Bridge

```
// gr -> adj list
// vector vis, low -> initialize to -1
// int timer -> initialize to 0
void dfs(int pos, int dad = -1)
{
```

```
    vis[pos] = low[pos] = timer++;
    int kids = 0;
    for(auto& i : gr[pos])
    {
        if(i == dad) continue;
        if(vis[i] >= 0)
            low[pos] = min(low[pos], vis[i]);
        else
        {
            dfs(i, pos);
            low[pos] = min(low[pos], low[i]);
            if(low[i] > vis[pos])
                is_bridge(pos, i)
            if(low[i] >= vis[pos] && dad >= 0)
                is_articulation_point(pos)
            ++kids;
        }
    }
    if(dad == -1 && kids > 1)
        is_articulation_point(pos)
}
```

4.2 Centroid Decomposition

```
int build_cen(int nw){
    com_cen(nw,0); //fungsi untuk itung size subtree
    int siz = sz[nw]/2; bool found = false;
    while(!found){
        found = true;
        for(int i:v[nw]){
            if(!rem[i] && sz[i]<sz[nw]){
                if(sz[i] > siz){found = false; nw = i; break;}
            }
        }
    }big
    rem[nw] = true;
    for(int i:v[nw])if(!rem[i])par_cen[build_cen(i)] = nw;
    return nw;
}
```

4.3 Dinic's Maximum Flow

```
// O(VE log(max_flow)) if scaling == 1
// O((V + E) sqrt(E)) if unit graph (turn scaling off)
// O((V + E) sqrt(V)) if bipartite matching (turn scaling off)
// indices are 0-based
const ll INF = 1e18;

struct Dinic
{
    struct Edge
    {
        int v;
        ll cap, flow;
        Edge(int _v, ll _cap) : v(_v), cap(_cap), flow(0) {}
    };

    int n;
    ll lim;
    vector<vector<int>> gr;
    vector<Edge> e;
    vector<int> idx, lv;

    bool has_path(int s, int t)
    {
        queue<int> q;
        q.push(s);
        lv.assign(n, -1);
        lv[s] = 0;
        while(!q.empty())
        {
            int c = q.front();
            q.pop();
            if(c == t) break;
            for(auto& i : gr[c])
            {
                ll cur_flow = e[i].cap - e[i].flow;
                if(lv[e[i].v] == -1 && cur_flow >= lim)
                {
                    lv[e[i].v] = lv[c] + 1;
                    q.push(e[i].v);
                }
            }
        }
        return lv[t] != -1;
    }

    ll get_flow(int s, int t, ll left)
    {
        if(!left || s == t) return left;
        while(idx[s] < (int) gr[s].size())
        {
            int i = gr[s][idx[s]];
            if(lv[e[i].v] == lv[s] + 1)
            {
                ll add = get_flow(
                    e[i].v,
                    t,
                    min(left, e[i].cap - e[i].flow)
                );
                if(add)
                {
                    e[i].flow += add;
                    e[i ^ 1].flow -= add;
                    return add;
                }
            }
            ++idx[s];
        }
        return 0;
    }

    Dinic(int vertices, bool scaling = 1) // toggle scaling here
        : n(vertices), lim(scaling ? 1 << 30 : 1), gr(n) {}
```

```
    void add_edge(int from, int to, ll cap, bool directed = 1)
    {
        gr[from].push_back(e.size());
        e.emplace_back(to, cap);
        gr[to].push_back(e.size());
        e.emplace_back(from, directed ? 0 : cap);
    }

    ll get_max_flow(int s, int t) // call this
    {
        ll res = 0;
        while(lim) // scaling
        {
            while(has_path(s, t))
            {
                idx.assign(n, 0);
                while(ll add = get_flow(s, t, INF)) res += add;
            }
            lim >>= 1;
        }
        return res;
    }
};
```

4.4 Edmonds' Blossom

```
// Maximum matching on general graphs in O(V^2 E)
// Indices are 1-based
// Stolen from ko_osaga's cheatsheet
struct Blossom
{
    vector<int> vis, dad, orig, match, aux;
    vector<vector<int>> conn;
    int t, N;
    queue<int> Q;

    void augment(int u, int v)
    {
        int pv = v;
        do
        {
            pv = dad[v];
            int nv = match[pv];
            match[v] = pv;
            match[pv] = v;
            v = nv;
        } while(u != pv);
    }

    int lca(int v, int w)
    {
        ++t;
        while(true)
        {
            if(v)
            {
                if(aux[v] == t) return v;
                aux[v] = t;
                v = orig[dad[match[v]]];
            }
            swap(v, w);
        }
    }

    void blossom(int v, int w, int a)
    {
        while(orig[v] != a)
        {
            dad[v] = w;
            w = match[v];
            if(vis[w] == 1)
            {
                Q.push(w);
                vis[w] = 0;
            }
            orig[v] = orig[w] = a;
            v = dad[w];
        }
    }

    bool bfs(int u)
    {
        fill(vis.begin(), vis.end(), -1);
        iota(orig.begin(), orig.end(), 0);
        Q = queue<int>();
        Q.push(u);
        vis[u] = 0;
        while(!Q.empty())
        {
            int v = Q.front(); Q.pop();
            for(int x : conn[v])
            {
                if(vis[x] == -1)
                {
                    dad[x] = v; vis[x] = 1;
                    if(!match[x])
                    {
                        augment(u, x);
                        return 1;
                    }
                    Q.push(match[x]);
                    vis[match[x]] = 0;
                }
                else if(vis[x] == 0 && orig[v] != orig[x])
                {
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a);
                    blossom(v, x, a);
                }
            }
        }
        return false;
    }

    Blossom(int n) : // n = vertices
        vis(n + 1), dad(n + 1), orig(n + 1), match(n + 1),
```

```
        aux(n + 1), conn(n + 1), t(0), N(n)
    {
        for(int i = 0; i <= n; ++i)
        {
            conn[i].clear();
            match[i] = aux[i] = dad[i] = 0;
        }
    }

    void add_edge(int u, int v)
    {
        conn[u].push_back(v);
        conn[v].push_back(u);
    }

    int solve() // call this for answer
    {
        int ans = 0;
        vector<int> V(N - 1);
        iota(V.begin(), V.end(), 1);
        shuffle(V.begin(), V.end(), mt19937(0x94949));
        for(auto x : V)
        {
            if(!match[x])
            {
                for(auto y : conn[x])
                {
                    if(!match[y])
                    {
                        match[x] = y, match[y] = x;
                        ++ans;
                        break;
                    }
                }
            }
        }
        for(int i = 1; i <= N; ++i)
        {
            if(!match[i] && bfs(i)) ++ans;
        }
        return ans;
    }
};
```

4.5 Eulerian Path or Cycle

```
// finds a eulerian path / cycle
// visits each edge only once
// properties:
// - cycle: degrees are even
// - path: degrees are even OR degrees are even except for 2 vertices
// how to use: g = adjacency list g[n] = connected to n, undirected
// if there is a vertex u with an odd degree, call dfs(u)
// else call on any vertex
// ans = path result

vector<set<int>> g;
vector<int> ans;

void dfs(int u)
{
    while(g[u].size())
    {
        int v = *g[u].begin();
        g[u].erase(v);
        g[v].erase(u);
        dfs(v);
    }
    ans.push_back(u);
}
```

4.6 Hierholzer's Algorithm

```
// Eulerian on Directed Graph
stack<int> path; vector<int> euler;
inline void hierholzer()
{
    path.push(0); int cur=0;
    while (!path.empty())
    {
        if (!adj[cur].empty())
        {
            path.push(cur);
            int next=adj[cur].back();
            adj[cur].pob();
            cur=next;
        }
        else
        {
            euler.pb(cur);
            cur=path.top();
            path.pop();
        }
    }
    reverse(euler.begin(),euler.end());
}
```

4.7 Hungarian

```
template <typename TD> struct Hungarian {
    TD INF = 1e9; //max_inf
    int n; vector<vector<TD> > adj; // cost[left][right]
    vector<TD> hl,hr,slk;
    vector<int> fl,fr,vl,vr,pre;
    deque<int> q;
    Hungarian(int _n) {
        n=_n; adj = vector<vector<TD> >(n, vector<TD> (n, 0));
    }
    int check(int i) {
        if (vl[i]=1,fl[i]!=-1) return q.push_back(fl[i]), vr[fl[i]]=1;
        while (i!=-1) swap(i,fr[fl[i]]=pre[i]); return 0;
```

```
    }
    void bfs(int s) {
        slk.assign(n,INF); vl.assign(n,0); vr=vl; q.assign(vr[s]=1,s);
        for (TD d;;) {
            for (;!q.empty();q.pop_front()) {
                for (int i=0,j=q.front();i<n;i++) {
                    if (d=hl[i]+hr[j]-adj[i][j],!vl[i]&&d<=slk[i]) {
                        if (pre[i]=j,d) slk[i]=d; else if (!check(i)) return;
                    }}}
            d=INF;
            for (int i = 0; i < n; i++) if (!vl[i]&&d>slk[i]) d=slk[i];
            for (int i = 0; i < n; i++) {
                if (vl[i]) hl[i]+=d; else slk[i]-=d;
                if (vr[i]) hr[i]-=d;
            }
            for (int i = 0; i < n; i++) if (!vl[i]&&!slk[i]&&!check(i)) return;
        }}
    TD solve() {
        fl.assign(n,-1); fr=fl; hl.assign(n,0); hr=hl; pre.assign(n,0);
        for (int i = 0; i < n; i++) hl[i]=*max_element(adj[i].begin(),adj[i].begin()+n);
        for (int i = 0; i < n; i++) bfs(i);
        TD ret=0;
        for (int i = 0; i < n; i++) if (adj[i][fl[i]]) ret+=adj[i][fl[i]];
        return ret;
    }
}; //i wiLL be matched with fl[i]
```

4.8 Minimum Cost Maximum Flow

```
// 1-based index
template<class T>
using rpq = priority_queue<T, vector<T>, greater<T>>;

const ll INF = 1e18;

struct MCMF
{
    struct Edge
    {
        int v;
        ll cap, cost;
        int rev;
        Edge(int _v, ll _cap, ll _cost, int _rev) :
            v(_v), cap(_cap), cost(_cost), rev(_rev) {}
    };

    ll flow, cost;
    int st, ed, n;
    vector<ll> dist, H;
    vector<int> pv, pe;
    vector<vector<Edge>> adj;

    bool dijkstra()
    {
        rpq<pair<ll, int>> pq;
        dist.assign(n + 1, INF);
        dist[st] = 0;
        pq.emplace(0, st);
        while(!pq.empty())
        {
            auto [cst, pos] = pq.top();
            pq.pop();
            if(dist[pos] < cst) continue;
            for(int i = 0; i < (int) adj[pos].size(); ++i)
            {
                auto& e = adj[pos][i];
                int nxt = e.v;
                ll nxt_cst = dist[pos] + e.cost + H[pos] - H[nxt];
                if(e.cap > 0 && nxt_cst < dist[nxt])
                {
                    dist[nxt] = nxt_cst;
                    pe[nxt] = i;
                    pv[nxt] = pos;
                    pq.emplace(nxt_cst, nxt);
                }
            }
        }
        return dist[ed] != INF;
    }

    MCMF(int _n) : n(_n), pv(n + 1), pe(n + 1), adj(n + 1) {}

    void add_edge(int u, int v, ll cap, ll cst)
    {
        adj[u].emplace_back(v, cap, cst, adj[v].size());
        adj[v].emplace_back(u, 0, -cst, adj[u].size() - 1);
    }

    pair<ll, ll> solve(int _st, int _ed)
    {
        st = _st, ed = _ed;
        flow = 0, cost = 0;
        H.assign(n + 1, 0);
        while(dijkstra())
        {
            for(int i = 0; i <= n; ++i)
                H[i] += dist[i];
            ll f = INF;
            for(int i = ed; i != st; i = pv[i])
                f = min(f, adj[pv[i]][pe[i]].cap);
            flow += f;
            cost += f * H[ed];
            for(int i = ed; i != st; i = pv[i])
            {
                auto& e = adj[pv[i]][pe[i]];
                e.cap -= f;
                adj[i][e.rev].cap += f;
            }
        }
        return {flow, cost};
    }
};
```

4.9 SCC and Strong Orientation

```
#define N 10020
vector<int> adj[N];
bool vis[N], ins[N];
int disc[N], low[N], gr[N];
stack<int> st;
int id,grid;
void scc(int cur, int par)
{
    disc[cur]=low[cur]=++id;
    vis[cur]=ins[cur]=1;
    st.push(cur);
    for(int to : adj[cur])
    {
        //if (to==par) continue; // ini untuk SO(scc undirected)
        if (!vis[to]) scc(to,cur);
        if (ins[to]) low[cur]=min(low[cur],low[to]);
    }
    if(low[cur]==disc[cur])
    {
        grid++; // group id
        while(ins[cur])
        {
            gr[st.tp]=grid; ins[st.tp]=0; st.pop();
        }
    }
}
```

5 Math

5.1 Berlekamp-Massey

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
typedef long long ll;
#define SZ 233333
const int MOD=1e9+7; //or any prime
ll qp(ll a,ll b)
{
    ll x=1; a%=MOD;
    while(b)
    {
        if(b&1) x=x*a%MOD;
        a=a*a%MOD; b>>=1;
    }
    return x;
}
namespace linear_seq {
    vector<int> BM(vector<int> x)
    {
        //ls: (shortest) relation sequence (after filling zeroes) so far
        //cur: current relation sequence
        vector<int> ls,cur;
        //lf: the position of ls (t')
        //ld: delta of ls (v')
        int lf = -1,ld = -1;
        for(int i=0;i<int(x.size());++i)
        {
            ll t=0;
            //evaluate at position i
            for(int j=0;j<int(cur.size());++j)
                t=(t+x[i-j-1]*(ll)cur[j])%MOD;
            if((t-x[i])%MOD==0) continue; //good so far
            //first non-zero position
            if(!cur.size())
            {
                cur.resize(i+1);
                lf=i; ld=(t-x[i])%MOD;
                continue;
            }
            //cur=cur-c/ld*(x[i]-t)
            ll k=-(x[i]-t)*qp(ld,MOD-2)%MOD/*1/ld*/;
            vector<int> c(i-lf-1); //add zeroes in front
            c.pb(k);
            for(int j=0;j<int(ls.size());++j)
                c.pb(-ls[j]*k%MOD);
            if(c.size()<cur.size()) c.resize(cur.size());
            for(int j=0;j<int(cur.size());++j)
                c[j]=(c[j]+cur[j])%MOD;
            //if cur is better than ls, change ls to cur
            if(i-lf+(int)ls.size()>=(int)cur.size())
                ls=cur,lf=i,ld=(t-x[i])%MOD;
            cur=c;
        }
        for(int i=0;i<int(cur.size());++i)
            cur[i]=(cur[i]%MOD+MOD)%MOD;
        return cur;
    }
    int m; //length of recurrence
    //a: first terms
    //h: relation
    ll a[SZ],h[SZ],t_[SZ],s[SZ],t[SZ];
    //calculate p*q mod f
    void mull(ll*p,ll*q)
    {
        for(int i=0;i<m+m;++i) t_[i]=0;
        for(int i=0;i<m;++i) if(p[i])
            for(int j=0;j<m;++j)
                t_[i+j]=(t_[i+j]+p[i]*q[j])%MOD;
        for(int i=m+m-1;i>=m;--i) if(t_[i])
            //miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1} x^{m-j-1}h_j)
            for(int j=m-1;~j;--j)
                t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%MOD;
        for(int i=0;i<m;++i) p[i]=t_[i];
    }
    ll calc(ll K)
    {
        for(int i=m;~i;--i)
            s[i]=t[i]=0;
        //init
        s[0]=1; if(m!=1) t[1]=1; else t[0]=h[0];
        //binary-exponentiation
        while(K)
        {
            if(K&1) mull(s,t);
            mull(t,t); K>>=1;
```

```
        }
        ll su=0;
        for(int i=0;i<m;++i) su=(su+s[i]*a[i])%MOD;
        return (su%MOD+MOD)%MOD;
    }
    int work(vector<int> x,ll n)
    {
        if(n<int(x.size())) return x[n];
        vector<int> v=BM(x); m=v.size(); if(!m) return 0;
        for(int i=0;i<m;++i) h[i]=v[i],a[i]=x[i];
        return calc(n);
    }
}
using linear_seq::work;

const vector<int> sequence = {
    0, 2, 2, 28, 60, 836, 2766
};
int main()
{
    cout<<work(sequence, 7) << '\n';
}
```

5.2 Catalan

```
long long cat(long long n){
    long long ret = 1;
    for(long long i = 0; i < n; i++){
        ret = ret*(2*n-i);
        ret = ret/(i+1);
    }
    ret = ret/(n+1);
    return ret;
}

ll superCatalan(int n){
    if(n <= 2)return 1;
    return (3*(2*n-3)*sc(n-1)-(n-3)*sc(n-2)) / n;
} // 1,1,1,3,11,45, 197, 903, 4279, 20793, 103049
```

5.3 Extended Euclidean GCD

```
// computes x and y such that ax + by = gcd(a, b) in O(log (min(a, b)))
// returns {gcd(a, b), x, y}
tuple<int, int, int> gcd(int a, int b)
{
    if(b == 0) return {a, 1, 0};
    auto [d, x1, y1] = gcd(b, a % b);
    return {d, y1, x1 - y1 * (a / b)};
}
```

5.4 Fast Fourier Transform

```
using ld = double; // change to long double if reach 10^18
using cd = complex<ld>;
const ld PI = acos(-(ld)1);

void fft(vector<cd> &a, int sign = 1)
{
    int n = a.size();
    ld theta = sign * 2 * PI / n;
    for(int i = 0, j = 1; j < n-1; j++)
    {
        for(int k = n >> 1; k > (i ^= k); k >>= 1);
        if(j < i) swap(a[i], a[j]);
    }
    for(int m, mh = 1; (m = mh << 1) <= n; mh = m)
    {
        int irev = 0;
        for(int i = 0; i < n; i += m)
        {
            cd w = exp(cd(0, theta*irev));
            for(int k = n >> 2; k > (irev ^= k); k >>= 1);
            for(int j = i; j < mh + i; j++)
            {
                int k = j+mh;
                cd x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
    }
    if(sign == -1) for(cd &i : a) i /= n;
}

vector<ll> multiply(vector<ll> const& a, vector<ll> const &b)
{
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while(n < a.size() + b.size()) n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa); fft(fb);
    for(int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, -1);
    vector<ll> res(n);
    for(int i = 0; i < n; i++) res[i] = round(fa[i].real());
    return res;
}
```

5.5 Fibonacci Check

```
bool is_fibonacci(int n)
{
    return is_perfect_square(5 * n * n + 4)
         || is_perfect_square(5 * n * n - 4);
}
```

5.6 Gauss-Jordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:   a[][] = an nxn matrix
//          b[][] = an nxm matrix
//
// OUTPUT:  X      = an nxm matrix (stored in b[][])
//          A^{-1} = an nxn matrix (stored in a[][])
//          returns determinant of a[][]
const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
T GaussJordan(VVT &a, VVT &b) {
    const int n = a.size();
    const int m = b[0].size();
    VI irow(n), icol(n), ipiv(n);
    T det = 1;
    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
        if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        if (pj != pk) det *= -1;
        irow[i] = pj;
        icol[i] = pk;
        T c = 1.0 / a[pk][pk];
        det *= a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return det;
}
int main() {
    const int n = 4;
    const int m = 2;
    double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
    double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
    VVT a(n), b(n);
    for (int i = 0; i < n; i++) {
        a[i] = VT(A[i], A[i] + n);
        b[i] = VT(B[i], B[i] + m);
    }
    double det = GaussJordan(a, b);
    // expected: 60
    cout << "Determinant: " << det << endl;
    // expected: -0.233333 0.166667 0.133333 0.0666667
    //           0.166667 0.166667 0.333333 -0.333333
    //           0.233333 0.833333 -0.133333 -0.0666667
    //           0.05 -0.75 -0.1 0.2
    cout << "Inverse: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
    // expected: 1.63333 1.3
    //           -0.166667 0.5
    //           2.36667 1.7
    //           -1.85 -1.35
    cout << "Solution: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            cout << b[i][j] << ' ';
        cout << endl;
    }
}
```

5.7 Generalized CRT

```
template<typename T>
T extended_euclid(T a, T b, T &x, T &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    T xx, yy, gcd;
    gcd = extended_euclid(b, a % b, xx, yy);
    x = yy;
    y = xx - (yy * (a / b));
    return gcd;
}
template<typename T>
T MOD(T a, T b) { return (a%b + b) % b; }
// return x, lcm. x = a % n && x = b % m
template<typename T>
pair<T,T> CRT(T a, T n, T b, T m) {
    T _n, _m;
    T gcd = extended_euclid(n, m, _n, _m);
    if (n == m) {
```

```
        if (a == b) return pair<T,T>(a, n);
        else return pair<T,T>(-1, -1);
    } else if (abs(a-b) % gcd != 0) return pair<T,T>(-1, -1);
    else {
        T lcm = m * n / gcd;
        T x = MOD(a + MOD(n*MOD(_n*((b-a)/gcd), m/gcd), lcm), lcm);
        return pair<T,T>(x, lcm);
    }}
```

5.8 Generalized Lucas Theorem

```
/*Special Lucas : (n,k) % p^x
  fctp[n] = Product of the integers less than or equal
  to n that are not divisible by p
  Precompute fctp*/
LL p
LL E(LL n,int m){
    LL tot = 0;
    while(n!=0){
        tot += n/m,n/=m;
    }
    return tot;
}
LL funct(LL n,LL base){
    LL ans = fast(fctp[base],n/base,base) * fctp[n%base] %base;
    return ans;
}
LL F(LL n,LL base){
    LL ans = 1;
    while(n!=0){
        ans = (ans * funct(n,base))%base;
        n/=p;
    }
    return ans;
}
LL special_lucas(LL n,LL r,LL base){
    p = fprime(base);
    LL pow = E(n,p) - E(n-r,p) - E(r,p);
    LL TOP = fast(p,pow,base) * F(n,base)%base;
    LL BOT = F(r,base) * F(n-r,base)%base;
    return (TOP * fast(BOT,totien(base) - 1,base))%base;
}
//End of Special Lucas
```

5.9 Linear Diophantine

```
//FOR SOLVING MINIMUM ABS(X) + ABS(Y)
ll x,y,newX,newY,target=0;
ll extGcd(ll a,ll b){
    if(b==0){
        x=1,y=0;
        return a;
    }
    ll ret = extGcd(b,a%b);
    newX = y;
    newY = x - y * (a/b);
    x = newX;
    y = newY;
    return ret;
}
ll fix(ll sol,ll rt){
    ll ret = 0;
    //CASE SOLUTION(X/Y) < TARGET
    if(sol < target)ret = -floor(abs(sol+target)/(double)rt);
    //CASE SOLUTION(X/Y) > TARGET
    if(sol > target)ret = ceil(abs(sol-target)/(double)rt);
    return ret;
}
ll work(ll a,ll b,ll c){
    ll gcd = extGcd(a,b);
    ll solX = x*(c/gcd);
    ll solY = y*(c/gcd);
    a/=gcd;b/=gcd;
    ll fi = abs(fix(solX,b));
    ll se = abs(fix(solY,a));
    ll lo = min(fi,se);
    ll hi = max(fi,se);
    ll ans = abs(solX) + abs(solY);
    for(ll i = lo; i<=hi; i++){
        ans = min(ans, abs(solX+i*b) + abs(solY-i*a));
        ans = min(ans, abs(solX-i*b) + abs(solY+i*a));
    }
    return ans;
}
```

5.10 Miller-Rabin and Pollard's Rho

```
namespace MillerRabin
{
    const vector<ll> primes = { // deterministic up to 2^64 - 1
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37
    };
    ll gcd(ll a, ll b)
    {
        return b ? gcd(b, a % b) : a;
    }
    ll powa(ll x, ll y, ll p) // (x ^ y) % p
    {
        if(!y) return 1;
        if(y & 1) return ((__int128) x * powa(x, y - 1, p)) % p;
        ll temp = powa(x, y >> 1, p);
        return ((__int128) temp * temp) % p;
    }
    bool miller_rabin(ll n, ll a, ll d, int s)
    {
        ll x = powa(a, d, n);
        if(x == 1 || x == n - 1) return 0;
        for(int i = 0; i < s; ++i)
        {
            x = ((__int128) x * x) % n;
            if(x == n - 1) return 0;
```

```
        }
        return 1;
    }
    bool is_prime(ll x) // use this
    {
        if(x < 2) return 0;
        int r = 0;
        ll d = x - 1;
        while((d & 1) == 0)
        {
            d >>= 1;
            ++r;
        }
        for(auto& i : primes)
        {
            if(x == i) return 1;
            if(miller_rabin(x, i, d, r)) return 0;
        }
        return 1;
    }
}

namespace PollardRho
{
    mt19937_64 generator(chrono::steady_clock::now()
                     .time_since_epoch().count());
    uniform_int_distribution<ll> rand_ll(0, LLONG_MAX);
    ll f(ll x, ll b, ll n) // (x^2 + b) % n
    {
        return (((__int128) x * x) % n + b) % n;
    }
    ll rho(ll n)
    {
        if(n % 2 == 0) return 2;
        ll b = rand_ll(generator);
        ll x = rand_ll(generator);
        ll y = x;
        while(1)
        {
            x = f(x, b, n);
            y = f(f(y, b, n), b, n);
            ll d = MillerRabin::gcd(abs(x - y), n);
            if(d != 1) return d;
        }
    }
    void pollard_rho(ll n, vector<ll>& res)
    {
        if(n == 1) return;
        if(MillerRabin::is_prime(n))
        {
            res.push_back(n);
            return;
        }
        ll d = rho(n);
        pollard_rho(d, res);
        pollard_rho(n / d, res);
    }
    vector<ll> factorize(ll n, bool sorted = 1) // use this
    {
        vector<ll> res;
        pollard_rho(n, res);
        if(sorted) sort(res.begin(), res.end());
        return res;
    }
}
```

5.11 Modular Linear Equation

```
// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver(int a, int b, int n) {
    int x, y; vi ret; int g = extended_euclid(a, n, x, y);
    if (!(b%g)) {
        x = mod(x*(b / g), n);
        for (int i = 0; i < g; i++)
            ret.push_back(mod(x + i*(n / g), n));
    }
    return ret;
}
```

5.12 Number Theoretic Transform

```
namespace FFT {
    /* ----- Adjust the constants here ----- */
    const int LN = 24; //23
    const int N = 1 << LN;
    typedef long long LL; // 2**23 * 119 + 1. 998244353
    // 'MOD' must be of the form 2**'LN' * k + 1, where k odd.
    const LL MOD = 9223372036737335297; // 2**24 * 54975513881 + 1.
    const LL PRIMITIVE_ROOT = 3; // Primitive root modulo 'MOD'.
    /* ----- End of constants ----- */
    LL root[N];
    inline LL power(LL x, LL y) {
        LL ret = 1;
        for (; y; y >>= 1) {
            if (y & 1) ret = (__int128) ret * x % MOD;
            x = (__int128) x * x % MOD;
        }
        return ret;
    }
    inline void init_fft() {
        const LL UNITY = power(PRIMITIVE_ROOT, MOD-1 >> LN);
        root[0] = 1;
        for (int i=1; i<N; i++) {
            root[i] = (__int128) UNITY * root[i-1] % MOD;
        }return;
    }
    // n = 2^k is the length of polynom
    inline void fft(int n, vector<LL> &a, bool invert) {
        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) {
            LL wlen = (invert ? root[N - N/len] : root[N/len]);
            for (int i=0; i<n; i+=len) {
                LL w = 1;
                for (int j=0; j<len>>1; j++) {
                    LL u = a[i+j];
                    LL v = (__int128) a[i+j + len/2] * w % MOD;
                    a[i+j] = ((__int128) u + v) % MOD;
                    a[i+j + len/2] = ((__int128) u - v + MOD) % MOD;
                    w = (__int128) w * wlen % MOD;
                }
            }
        }
        if (invert) {
            LL inv = power(n, MOD-2);
            for (int i=0; i<n; i++) a[i] = (__int128) a[i] * inv % MOD;
        }return;
    }
    inline vector<LL> multiply(vector<LL> a, vector<LL> b) {
        vector<LL> c;
        int len = 1 << 32 - __builtin_clz(a.size() + b.size() - 2);
        a.resize(len, 0); b.resize(len, 0);
        fft(len, a, false); fft(len, b, false);
        c.resize(len);
        for (int i = 0; i < len; ++i)c[i] = (__int128) a[i]*b[i]%MOD;
        fft(len, c, true);
        return c;
    }
    //FFT::init_fft(); wajib di panggil init di awal
}
```

6 Miscellaneous

6.1 Dates

6.1.1 Day of Date

```
// 0-based
const vector<int> T = {
    0, 3, 2, 5, 0, 3,
    5, 1, 4, 6, 2, 4
}

int day(int d, int m, int y)
{
    y -= (m < 3);
    return (y + y / 4 - y / 100 + y / 400 + T[m - 1] + d) % 7;
}
```

6.1.2 Number of Days since 1-1-1

```
int rdn(int d, int m, int y)
{
    if(m < 3) --y, m += 12;
    return 365 * y + y / 4 - y / 100 + y / 400
        + (153 * m - 457) / 5 + d - 306;
}
```

6.2 Enumerate Subsets of a Bitmask

```
int x = 0; do {
    // do stuff with the bitmask here
    x = (x + 1 + ~m) & m;
} while(x != 0);
```

6.3 Fast IO

```
int read()
{
    char c;
    do
    {
        c = getchar_unlocked();
    } while(c < 33);
    int res = 0;
    int mul = 1;
    if(c == '-')
    {
        mul = -1;
        c = getchar_unlocked();
    }
    while('0' <= c && c <= '9')
    {
        res = res * 10 + c - '0';
        c = getchar_unlocked();
    }
    return res * mul;
}

void write(int x)
{
    static char wbuf[10];
    if(x < 0)
    {
        putchar_unlocked('-');
        x = -x;
    }
    int idx = 0;
    while(x)
    {
        wbuf[idx++] = x % 10;
        x /= 10;
    }
    if(idx == 0)
    {
        putchar_unlocked('0');
    }
```

```
    for(int i = idx - 1; i >= 0; --i)
    {
        putchar_unlocked(wbuf[i] + '0');
    }
}

void write(const char* s)
{
    while(*s)
    {
        putchar_unlocked(*s);
        ++s;
    }
}
```

6.4 Int to Roman

```
const string R[] = {
    "M", "CM", "D", "CD", "C", "XC", "L",
    "XL", "X", "IX", "V", "IV", "I"
};

const int N[] = {
    1000, 900, 500, 400, 100, 90,
    50, 40, 10, 9, 5, 4, 1
};

string to_roman(int x)
{
    if (x == 0) return "O"; // Not decimal 0!
    string res = "";
    for (int i = 0; i < 13; ++i)
        while (x >= N[i]) x -= N[i], res += R[i];
    return res;
}
```

6.5 Josephus Problem

```
ll josephus(ll n, ll k) // O(k log n)
{
    if(n == 1) return 0;
    if(k == 1) return n - 1;
    if(k > n) return (josephus(n - 1, k) + k) % n;
    ll cnt = n / k;
    ll res = josephus(n - cnt, k);
    res -= n % k;
    if(res < 0) res += n;
    else res += res / (k - 1);
    return res;
}

int josephus(int n, int k) // O(n)
{
    int res = 0;
    for(int i = 1; i <= n; ++i)
        res = (res + k) % i;
    return res + 1;
}
```

6.6 Random

```
// RNG - rand_int(min, max), inclusive

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

template<class T>
T rand_int(T mn, T mx)
{
    return uniform_int_distribution<T>(mn, mx)(rng);
}
```

7 Strings

7.1 Aho-Corasick

```
const int K = 26;
struct Vertex {
    int next[K];
    bool leaf = 0;
    int p = -1, ans = 0;
    char pch;
    int link = -1, mlink = -1;
    //magic link, is the link to find the nearest leaf
    int go[K];
    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};
vector<Vertex> t;
int add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        } v = t[v].next[c];
    }
    t[v].leaf = 1;
    return v;
}
int go(int v, char ch);
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0) t[v].link = 0;
        else t[v].link = go(get_link(t[v].p), t[v].pch);
```

```cpp
    }
    return t[v].link;
}
int get_mlink(int v) {
    if (t[v].mlink == -1) {
        if (v == 0 || t[v].p == 0) t[v].mlink = 0;
        else{
            t[v].mlink = go(get_link(t[v].p), t[v].pch);
            if(t[v].mlink && !t[t[v].mlink].leaf){
                if(t[t[v].mlink].mlink==-1)get_mlink(t[v].mlink);
                t[v].mlink = t[t[v].mlink].mlink;
            }
        }
    } return t[v].mlink;
}
int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1) t[v].go[c] = t[v].next[c];
        else t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    } return t[v].go[c];
}
//t.pb(Vertex());
```

7.2 Eertree

```cpp
/*
   Eertree - keep track of all palindromes and its occurences
   This code refers to problem Longest Palindromic Substring
https://www.spoj.com/problems/LPS/
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct node {
    int next[26];
    int sufflink;
    int len, cnt;
};

const int N = 1e5+69;
int n;
string s;
node tree[N];
int idx, suff;
int ans = 0;

void init_eertree() {
    idx = suff = 2;
    tree[1].len = -1, tree[1].sufflink = 1;
    tree[2].len = 0, tree[2].sufflink = 1;
}

bool add_letter(int x) {
    int cur = suff, curlen = 0;
    int nw = s[x] - 'a';
    while(1) {
        curlen = tree[cur].len;
        if(x-curlen-1 >= 0 && s[x-curlen-1] == s[x])
            break;
        cur = tree[cur].sufflink;
    }

    if(tree[cur].next[nw]) {
        suff = tree[cur].next[nw];
        return 0;
    }


    tree[cur].next[nw] = suff = ++idx;
    tree[idx].len = tree[cur].len + 2;
    ans = max(ans, tree[idx].len);

    if(tree[idx].len == 1) {
        tree[idx].sufflink = 2;
        tree[idx].cnt = 1;
        return 1;
    }

    while(1) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if(x-curlen-1 >= 0 && s[x-curlen-1] == s[x]) {
            tree[idx].sufflink = tree[cur].next[nw];
            break;
        }
    }
    tree[idx].cnt = tree[tree[idx].sufflink].cnt + 1;

    return 1;
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    cin >> n >> s;
    init_eertree();
    for(int i = 0; i < n; i++) {
        add_letter(i);
    }
    cout << ans << '\n';
    return 0;
}
```

7.3 Manacher's Algorithm

```cpp
// Computes lps array. lps[i] means the longest palindromic substring centered at i (when i is even,
     it is between characters. when it is odd, it is on characters)lps[0] = 0; lps[1] = 1;
REP(i,2,2*str.size()){
    int l = i/2 - lps[i]/2;
    int r = (i-1)/2 + lps[i]/2;
    while(1){ // widen
        if(l == 0 || r+1 == str.size())break;
```

```
        if(str[l-1] != str[r+1])break;
        l--, r++;
    }
    lps[i] = r - l + 1;
    // jump
    if(lps[i] > 2){
        int j = i-1, k = i+1; // while lps[j] inside lps[i]
        while(lps[j] - j < lps[i] - i) lps[k++] = lps[j--];
        lps[k] = lps[i] - (i - j); // set lps[k] to edge of lps[i]
        i = k-1; // jump to mirror, which is k
    }
}
```

7.4 Suffix Array

```
// stores result in sa and lcp
// if lcp is needed, call SuffixArray(str, 1)
struct SuffixArray
{
    int n;
    vector<int> sa, lcp, rnk, cnt;
    vector<pair<int, int>> p;
    SuffixArray(const string& s, bool calc_lcp = 0) :
        n(s.length()), sa(n), lcp(calc_lcp ? n : 0), rnk(n),
        cnt(max(n, 256)), p(n)
    {
        for(int i = 0; i < n; ++i) rnk[i] = s[i];
        iota(sa.begin(), sa.end(), 0);
        for(int i = 1; i < n; i <<= 1) update_sa(i);
        if(!calc_lcp) return;
        vector<int> phi(n), plcp(n);
        phi[sa[0]] = -1;
        for(int i = 1; i < n; ++i) phi[sa[i]] = sa[i - 1];
        int l = 0;
        for(int i = 0; i < n; ++i)
        {
            if(phi[i] == -1) plcp[i] = 0;
            else
            {
                while((i + l < n) && (phi[i] + l < n)
                      && (s[i + l] == s[phi[i] + l])) ++l;
                plcp[i] = l;
                l = max(l - 1, 0);
            }
        }
        for(int i = 0; i < n; ++i) lcp[i] = plcp[sa[i]];
    }
    void update_sa(int len)
    {
        sort_sa(len); sort_sa(0);
        for(int i = 0; i < n; ++i) p[i] = {rnk[i], rnk[(i + len) % n]};
        auto lst = p[sa[0]];
        rnk[sa[0]] = 0;
        int cur = 0;
        for(int i = 1; i < n; ++i)
        {
            if(lst != p[sa[i]])
            {
                lst = p[sa[i]];
                ++cur;
            }
            rnk[sa[i]] = cur;
        }
    }
    void sort_sa(int offset)
    {
        fill(cnt.begin(), cnt.end(), 0);
        for(int i = 0; i < n; ++i) ++cnt[rnk[(i + offset) % n]];
        int sum = 0;
        for(int i = 0; i < (int) cnt.size(); ++i)
        {
            int temp = cnt[i];
            cnt[i] = sum;
            sum += temp;
        }
        vector<int> temp(n);
        for(int i = 0; i < n; ++i)
        {
            int cur = cnt[rnk[(sa[i] + offset) % n]]++;
            temp[cur] = sa[i];
        }
        sa = move(temp);
    }
};
```