Team Notebook

ntTas

October 25, 2019

Contents

1 Data Structure

1.1 BITRange

```
/**
* Description: 1D range update, range sum query
* Alternative to lazy segment tree
* Source: GeeksForGeeks?
* Verification: ?
*/

struct FenwickTree {
    vector<int> bit; // binary indexed tree
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }

    FenwickTree(vector<int> a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

1.2 Binary Trie

```
//Binary Trie
//without deletion, comment all val change and trie.check
     condition
//Don't forget to check maxval
struct node{
    node *kiri = NULL;
    node *kanan = NULL;
    int val = 0; //Can be deleted
};

struct btrie{
    node *head = new node;
    node *cur = head;
    void insert(int x){
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                if(head->kanan == NULL)head->kanan = new node;
                head = head->kanan;
            } else{
                if(head->kiri == NULL)head->kiri = new node;
                head = head->kiri;
            }
            head->val += 1; // Can be deleted
        }
        head = cur;
    }
    void del(int x){
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                head = head->kanan;
            } else{
                head = head->kiri;
            }
            head->val -= 1; //Can be deleted
        }
        head = cur;
    }
    int max(int x){
        int res = 0;
        for(int i = 30; i >= 0; i--){
            if((x&(1<<i)) == 0){
                if(check(head->kiri)){
                    res += (1<<i);
                    head = head->kiri;
                } else if(check(head->kanan)){
                    head = head->kanan;
                }//Break can be placed here

            } else {
                if(check(head->kanan)){
                    head = head->kanan;
                } else if(check(head->kiri)){
                    head = head->kiri;
                    res += (1<<i);
                }//Break can be placed here
            }
        }
        head = cur;
        return (res^x);
    }
private:
    bool check(node *x){
        if(x != NULL and x->val > 0)return true; //Condition
             may be changed
        else return false;
    }
};
```

1.3 LazyPropagation

```
#include <stdio.h>
#include <math.h>
#define MAX 1000
int tree[MAX] = {0};
int lazy[MAX] = {0};

/*  si -> index of current node in segment tree
    ss and se -> Starting and ending indexes of elements for
                 which current nodes stores sum.
    us and ue -> starting and ending indexes of update query
    diff -> which we need to add in the range us to ue */
void updateRangeUtil(int si, int ss, int se, int us,
                     int ue, int diff) {
    if (lazy[si] != 0) {
        tree[si] += (se-ss+1)*lazy[si];
        if (ss != se) {
            lazy[si*2 + 1]  += lazy[si];
            lazy[si*2 + 2]  += lazy[si];
        }
        lazy[si] = 0;
    }
    if (ss>se || ss>ue || se<us)
        return ;
    if (ss>=us && se<=ue) {
        tree[si] += (se-ss+1)*diff;
        if (ss != se) {
            lazy[si*2 + 1]  += diff;
            lazy[si*2 + 2]  += diff;
        }
        return;
    }
    int mid = (ss+se)/2;
    updateRangeUtil(si*2+1, ss, mid, us, ue, diff);
```

```
   updateRangeUtil(0, 0, n-1, us, ue, diff);
}

    updateRangeUtil(si*2+2, mid+1, se, us, ue, diff);
    tree[si] = tree[si*2+1] + tree[si*2+2];
}

void updateRange(int n, int us, int ue, int diff) {

/*  A recursive function to get the sum of values in given
    range of the array. The following are parameters for
    this function.
    si --> Index of current node in the segment tree.
          Initially 0 is passed as root is always at'
          index 0
    ss & se --> Starting and ending indexes of the
                segment represented by current node,
                i.e., tree[si]
    qs & qe --> Starting and ending indexes of query
                range */
int getSumUtil(int ss, int se, int qs, int qe, int si) {
    if (lazy[si] != 0) {
        tree[si] += (se-ss+1)*lazy[si];
        if (ss != se) {
            // Since we are not yet updating children os si,
            // we need to set lazy values for the children
            lazy[si*2+1] += lazy[si];
            lazy[si*2+2] += lazy[si];
        }
        lazy[si] = 0;
    }
    // Out of range
    if (ss>se || ss>qe || se<qs)
        return 0;
    if (ss>=qs && se<=qe)
        return tree[si];
    int mid = (ss + se)/2;
    return getSumUtil(ss, mid, qs, qe, 2*si+1) +
           getSumUtil(mid+1, se, qs, qe, 2*si+2);
}

// Return sum of elements in range from index qs (quey
// start) to qe (query end). It mainly uses getSumUtil()
int getSum(int n, int qs, int qe) {
    // Check for erroneous input values
    if (qs < 0 || qe > n-1 || qs > qe) {
        printf("Invalid Input");
        return -1;
    }

    return getSumUtil(0, n-1, qs, qe, 0);
}

// A recursive function that constructs Segment Tree for
//  array[ss..se]. si is index of current node in segment
// tree st.
void constructSTUtil(int arr[], int ss, int se, int si) {
    if (ss > se)
        return ;
    if (ss == se) {
        tree[si] = arr[ss];
        return;
    }
    int mid = (ss + se)/2;
    constructSTUtil(arr, ss, mid, si*2+1);
    constructSTUtil(arr, mid+1, se, si*2+2);

    tree[si] = tree[si*2 + 1] + tree[si*2 + 2];
}

void constructST(int arr[], int n) {
    // Fill the allocated memory st
    constructSTUtil(arr, 0, n-1, 0);
}
```

1.4 Trie

```
struct node2{
    node2 *children[26] = {NULL};
};

struct trie{
    node2 *head = new node2;
    node2 *cur = head;
    void insert(string x){
        head = cur;
        for(int i = 0; i < x.size(); i++){
            int val = x[i]-'a';
            if(head->children[val] == NULL){
                head->children[val] = new node2;
            }
            head = head->children[val];
        }
    }

    bool find(string x){
        head = cur;
        for(int i = 0; i < x.size(); i++){
            int val = x[i]-'a';
            if(head->children[val] == NULL)return false;
            head = head->children[val];
        }
        return true;
    }
};
```

1.5 persistentSegmentTree

```
struct Node {
    int cnt;
    Node *left, *right;
    Node(int cnt, Node *left, Node *right) {
        this->cnt = cnt;
        this->left = left;
        this->right = right;
    }
    Node *insert(int l, int r, int k) {
        if (!(l <= k && k <= r)) r
            return this;
        Node *node = new
            Node(this->cnt + 1, this->left, this->right);
        if (l == r) return node;
        int m = (l + r) / 2;
        node->left = node->left->insert(l, m, k);
        node->right = node->right->insert(m + 1, r, k);
        return node;
    }

    static int query(Node *a, Node *b, int l, int r, int k) {
        if (l == r) return l;
        int m = (l + r) / 2;
        int cnt = b->left->cnt - a->left->cnt;
        if (cnt >= k + 1)
        return query(a->left, b->left, l, m, k);
        return query(a->right, b->right, m + 1, r, k - cnt);
    }
};

// MAIN
int n, m;scanf("%d %d", &n, &m);
map<int, int> M, RM;
int a[n];
for (int i = 0; i < n; i++) {
    scanf("%d", &a[i]);
    M[a[i]];
}
int cntr = 0;
for (auto it = M.begin(); it != M.end(); it++) {
    M[it->first] = cntr;
```

```
cntr--;
Node *null = new Node(0, NULL, NULL);
    RM[cntr] = it->first;
    cntr++;
}

null
    ->left = null, null->right = null;
Node *roots[n];
for (int i = 0; i < n; i++)
    roots[i] = (i == 0 ? null : roots[i - 1])->insert(0, cntr
        , M[a[i]]);
int u, v, k;
while (m--) {
    scanf("%d %d %d", &u, &v, &k);u--, v--, k--;
    Node *a = (u == 0 ? null : roots[u - 1]);
    Node *b = roots[v];
    int res = Node::query(a, b, 0, cntr, k);
    printf("%d\n", RM[res]);
}
```

1.6 treap

```
/**
* Treap uses implicit key
* This Implementation : maintain array, can insert and
    delete in any
position, can reverse interval
*/
typedef struct item * pitem;

struct item{
    int cnt, value, prior;
    bool rev;
    pitem l, r;
    item(int prior, int value) : cnt(1), rev(false), prior(
        prior),
    value(value), l(NULL), r(NULL) {}
};
int cnt(pitem t) {
    return t ? t->cnt : 0;
}
void upd_cnt(pitem it) {
    if (it) it->cnt = cnt(it->l) + cnt(it->r) + 1;
}
void push(pitem it) {
    if (it && it->rev) {
        it->rev = false;
        swap(it->l, it->r);
        if (it->l) it->l->rev ^= true;
        if (it->r) it->r->rev ^= true;
    }
}
void merge(pitem & t, pitem l, pitem r) {
    push(l);
    push(r);
    if (!l || !r) t = l ? l : r;
    else if (l->prior > r->prior)
        merge(l->r, l->r, r), t = l;
    else
        merge(r->l, l, r->l), t = r;
    upd_cnt(t);
}

void split(pitem t, pitem & l, pitem & r, int key, int add
    = 0) {
    if (!t) return void (l = r = 0);
    int cur_key = cnt(t->l)+add;
    if (key <= cur_key){
        split(t->l, l, t->l, key, add);
        r = t;
    }
    else{
        split(t->r, t->r, r, key, add + cnt(t->l) + 1);
        l = t;
    }
    upd_cnt(t);
}

void reverse(pitem t, int l, int r) {
    pitem t1, t2, t3;
    split(t, t1, t2, l);
    split(t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge(t, t1, t2);
    merge(t, t, t3);
}
void output (pitem t) {
    if (!t) return;
    push (t);
    output (t->l);
    printf ("%d ", t->value);
    output (t->r);
}
// MAIN
int n; scanf("%d", &n);
srand(time(NULL));
pitem root = NULL;
for (int i = 0; i < n; i++) {
    int a;scanf("%d", &a);
    pitem cur = new item(rand(), a);
    if (root) merge(root, root, cur);
    else root = cur;
}
int m;
scanf("%d", &m);
for (int i = 0; i < m; i++) {
    int l, r;
    scanf("%d %d", &l, &r);
    reverse(root, l, r);
    output(root);
    printf("\n");
}
```

2 Dynamic Programming

2.1 Convex Hull Trick

```
//DP convex hull trick (Beware on overflow)
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will
    maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (long double)(x->b - y->b)*(z->m - y->m) >= (
            long double)(y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
```

```
        while (y != begin() && bad(prev(y))) erase(prev(y));
        y->succ = [=] { return next(y) == end() ? 0 : &*next(
            y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y
            ));

}

    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

3 Geometry

3.1 AreaPolygon

```
// Let a simple polygon (i.e. without self intersection, not
     necessarily convex) be given. It is required to
// calculate its area given its vertices.
double area(const vector<point>& fig) {
    double res = 0;
    for (unsigned i = 0; i < fig.size(); i++) {
        point p = i ? fig[i - 1] : fig.back();
        point q = fig[i];
        res += (p.x - q.x) * (p.y + q.y);
    }
    return fabs(res) / 2;
}
```

3.2 circleLineIntersect

```
// Given the coordinates of the center of a circle and its
     radius,
// and the equation of a line, you're required to find the
     points
// of intersection.
double r, a, b, c; // given as input
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
if (c*c > r*r*(a*a+b*b)+EPS)
    puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
    puts ("1 point");
    cout << x0 << ' ' << y0 << '\n';
}
else {
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n
        ';
}
```

3.3 convexhull

```
/**
* Description: Top-bottom convex hull
* Source: Wikibooks
* Verification:
https://open.kattis.com/problems/convexhull
*/

ll cross(pi O, pi A, pi B) {
 return (ll)(A.f-O.f)*(B.s-O.s) -(ll)(A.s-O.s)*(B.f-O.f);
}

vpi convex_hull(vpi P) {
 sort(all(P)); P.erase(unique(all(P)),P.end());
 int n = sz(P);
 if (n == 1) return P;

 vpi bot = {P[0]};
 FOR(i,1,n) {
  while (sz(bot) > 1 && cross(bot[sz(bot)-2],
  bot.back(), P[i]) <= 0) bot.pop_back();
  bot.pb(P[i]);
 }
 bot.pop_back();

 vpi up = {P[n-1]};
 FORd(i,n-1) {
  while (sz(up) > 1 && cross(up[sz(up)-2],
  up.back(), P[i]) <= 0) up.pop_back();
  up.pb(P[i]);
 }
 up.pop_back();

 bot.insert(bot.end(),all(up));
 return bot;
}
```

3.4 convexhullGrahamScan

```
struct pt {
    double x, y;
};

bool cmp(pt a, pt b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
}

bool ccw(pt a, pt b, pt c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
}

void convex_hull(vector<pt>& a) {
    if (a.size() == 1)
        return;

    sort(a.begin(), a.end(), &cmp);
    pt p1 = a[0], p2 = a.back();
    vector<pt> up, down;
    up.push_back(p1);
    down.push_back(p1);
    for (int i = 1; i < (int)a.size(); i++) {
        if (i == a.size() - 1 || cw(p1, a[i], p2)) {
            while (up.size() >= 2 && !cw(up[up.size()-2], up[
                up.size()-1], a[i]))
                up.pop_back();
            up.push_back(a[i]);
        }
        if (i == a.size() - 1 || ccw(p1, a[i], p2)) {
            while(down.size() >= 2 && !ccw(down[down.size()
                -2], down[down.size()-1], a[i]))
                down.pop_back();
            down.push_back(a[i]);
        }
    }

    a.clear();
    for (int i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (int i = down.size() - 2; i > 0; i--)
```

```
        a.push_back(down[i]);
}
```

3.5 findIntersectionTwoSegment

```
// You are given two segments AB and
// CD, described as pairs of their endpoints. Each segment
     can
// be a single point if its endpoints are the same. You have
      to
// find the intersection of these segments, which can be
     empty
// (if the segments don't intersect),
// a single point or a segment (if the given segments
     overlap).
const double EPS = 1E-9;
struct pt {
    double x, y;
    bool operator<(const pt& p) const{
        return x < p.x - EPS || (abs(x - p.x) < EPS && y < p.
            y - EPS);
    }
};
struct line {
    double a, b, c;
    line() {}
    line(pt p, pt q){
        a = p.y - q.y;
        b = q.x - p.x;
        c = -a * p.x - b * p.y;
        norm();
    }
    void norm(){
        double z = sqrt(a * a + b * b);
        if (abs(z) > EPS)
            a /= z, b /= z, c /= z;
    }
    double dist(pt p) const { return a * p.x + b * p.y + c; }
};
double det(double a, double b, double c, double d){
    return a * d - b * c;
}
inline bool betw(double l, double r, double x){
    return min(l, r) <= x + EPS && x <= max(l, r) + EPS;
}
inline bool intersect_1d(double a, double b, double c,
     double d){
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d) + EPS;
}
bool intersect(pt a, pt b, pt c, pt d, pt& left, pt& right){
    if (!intersect_1d(a.x, b.x, c.x, d.x) || !intersect_1d(a.
         y, b.y, c.y, d.y))
        return false;
    line m(a, b);
    line n(c, d);
    double zn = det(m.a, m.b, n.a, n.b);
    if (abs(zn) < EPS) {
        if (abs(m.dist(c)) > EPS || abs(n.dist(a)) > EPS)
            return false;
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        left = max(a, c);
        right = min(b, d);
        return true;
    } else {
        left.x = right.x = -det(m.c, m.b, n.c, n.b) / zn;
        left.y = right.y = -det(m.a, m.c, n.a, n.c) / zn;
        return betw(a.x, b.x, left.x) && betw(a.y, b.y, left.
            y) &&
              betw(c.x, d.x, left.x) && betw(c.y, d.y, left.y
                  );
    }
}
```

3.6 geometry

```
Proyeksi segitiga: BC^2 = AC^2 + AB{^2 2AD.AC
#define EPS 1E-9
#define PI acos(-1)
// >>>> Constructor of point
struct point {
  double x,y;
  point() { x = y = 0.0; }
  point(double _x, double _y) : x(_x), y(_y) {}
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y - other.y) <
        EPS));
  }
};
// >>>> Constructor of vector
struct vec {
  double x, y;
  vec(double _x, double _y) : x(_x), y(_y) {}
};
// >>>> Constructor of line (ax + by = c)
struct line {
  double a,b,c;
};
// Distance of two points
double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y);
}
double DEG_to_RAD(double theta) {
  return theta * PI / 180.0;
}
// Rotate a point THETA degrees
point rotate(point p, double theta) {
  double rad = DEG_to_RAD(theta);
  return point(p.x * cos(rad) - p.y * sin(rad),
      p.x * sin(rad) + p.y * cos(rad));
}
// Make a line l from 2 given points
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {
   l.a = 1.0 ; l.b = 0.0 ; l.c = -p1.x;
  } else {
   l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
   l.b = 1.0;
   l.c = -(double)(l.a * p1.x) - p1.y;
  }
}
// Check if two lines are parallel
bool areParallel(line l1, line l2) {
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);
}
// Check if two lines are same
bool areSame(line l1, line l2) {
  return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);
}
// Check if two lines are intersect (at point P)
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a *
       l2.b);
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c); return true;
}
// Convert 2 points to vector A -> B
vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y);
}
// Scale a vector
```

```
  return point(p.x + v.x, p.y + v.y);
vec scale(vec v, double s) {
  return vec(v.x * s, v.y * s);
}
// Translate P according to v
point translate(point p, vec v) {

}

// Dot product of two vectors
double dot(vec a, vec b) {
  return a.x * b.x + a.y * b.y;
}
// Cross product of two vectors
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
double norm_sq(vec v) {
  return v.x * v.x + v.y * v.y;
}
// Get the minimum distance of point P and line AB
// Line PC is the minimum distance
double distToLine(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));
  return dist(p,c);
}
// Get the minimum distance of point P and line segment AB
// Line PC is the minimum distance
double distToLineSegment(point p, point a, point b, point &c
    ) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) {
   c = point(a.x, a.y);
   return dist(p,a);
  }
  if (u > 1.0) {
    c = point(b.x, b.y);
    return dist(p, b);
  }
  return distToLine(p, a, b, c);
}
// Returns angle AOB in RADIANS
double angle(point a, point o, point b) {
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa,ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}
// Heron's Formula : Find the area of triangle double
heronsFormula(double a, double b, double c) {
  double s = perimeter(a, b, c) * 0.5;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}
// Find the radius incircle of triangle ABC (lengths)
double rInCircle(double ab, double bc, double ca) {
  return heronsFormula(ab,bc,ca) / (0.5 * perimeter(ab, bc,
      ca));
}
// Find the radius incircle of triangle ABC (points)
double rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c), dist (c, a));
}
// Returns 1 if there is an incircle center, return 0
    otherwise
// ctr will be the incircle center
// r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr,
    double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0;

  line l1, l2;
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3), ratio / (1 +
      ratio)));
  pointsToLine(p1, p, l1);
  ratio = dist(p2, p1) / dist(p2 , p3);
  p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)
      ));
  pointsToLine(p2, p, l2);

  areIntersect(l1, l2, ctr);
  return 1;
}
// Find the radius circumcircle of triangle ABC (lengths)
double rCircumCircle(double ab, double bc, double ca) {
  return ab * bc * ca / (4.0 * heronsFormula(ab, bc, ca));
}
// Find the radius circumcircle of triangle ABC (points)
double rCircumCircle(point a, point b, point c) {
  return rCircumCircle(dist(a, b), dist(b, c), dist(c , a));
}
// Polygon Representation :
// 4 points, entered in counter clockwise order, 0-based
    indexing
// vector<point> P;
// P.push_back(point(1,1)); // P[0]
// P.push_back(point(3,3)); // P[1]
// P.push_back(point(9,7)); // P[2]
// P.push_back(point(1,7)); // P[3]
// P.push_back(P[0]); // P[n-1] = P[0]
// Checks if a polygon is convex or not
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
   bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i > sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft
        )
     return false;
  return true;
}
// Line segment PQ intersect with line AB at this point
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 AB
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])), left2
       = 0;
   if (i != (int)Q.size()-1) left2 = cross(toVec(a, b),
       toVec(a, Q[i+1]));
   // Q[i] is on the left of AB
   // edge(Q[i], Q[i+1]) crosses line AB
   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;
}
//-- Line Segment Intersection
int pyt(PII a, PII b){
   int dx=a.x-b.x;
   int dy=a.y-b.y;
   return (dx*dx + dy*dy);
}
int det(PII a, PII b, PII c){
   return ((a.x*b.y)+(b.x*c.y)+(c.x*a.y)
```

```
    h2=det(t1.F,t1.S, t2.S);
          -(a.x*c.y)-(b.x*a.y)-(c.x*b.y));
}
bool insec(pair<PII,PII> t1, pair<PII,PII> t2){
    bool hsl;
    h1=det(t1.F,t1.S, t2.F);

h3
    =det(t2.F,t2.S, t1.F);
    h4=det(t2.F,t2.S, t1.S);
    hsl=false;
    if ((h1*h2<=0) && (h3*h4<=0) && !((h1==0) && (h2==0) && (
        h3==0) && (h4==0))){
       hsl=true;
    }
    return hasil;
}
...
//sg1 dan sg2 adalah pair<PII,PII>
if (insec(sg1,sg2)){
   le=sqrt((double)pyt(sg2.x, sg2.y));
   r1=fabs(crosp(MP(sg2.x, sg1.x),sg2)/le);
   r2=fabs(crosp(MP(sg2.x, sg1.y),sg2)/le);
   r2=r1+r2;
   dix=sg1.x.x + (r1/r2)*(sg1.y.x - sg1.x.x);
   diy=sg1.x.y + (r1/r2)*(sg1.y.y - sg1.x.y);
   //intersect here
   return MP(dix,diy);
}
// returns the area, which is half the determinant
// works for both convex and concave polygons
double area(vector<point> P) {
 double result = 0.0, x1, y1, x2, y2;
 for (int i = 0; i < P.size() - 1; i++) {
  x1 = P[i].x;
  x2 = P[i + 1].x;
  y1 = P[i].y;
  y2 = P[i + 1].y;
  result += (x1 * y2 - x2 * y1);
 }
 return fabs(result) / 2.0;
}
// returns true if point p is in either convex/concave
    polygon P
bool inPolygon(point p, const vector<point> &P) {
 if ((int) P.size() == 0) return false;
 double sum = 0; // assume first vertex = last vertex
 for (int i = 0; i < (int) P.size() - 1; i++) {
  if (ccw(p, P[i], P[i + 1]))
   sum += angle(P[i], p, P[i + 1]); // left turn/ccw
  else
   sum -= angle(P[i], p, P[i + 1]);
 } // right turn/cw
 return fabs(fabs(sum) - 2 * PI) < EPS;
}
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
} // compute distance between point (x,y,z) and plane ax+by+
    cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double
                              d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
//circle-circle intersect
for(int i = 1; i < n; i++) {
 for(int j = i + 1; j <= n; j++) {
  double d = dist(P[i], P[j]);
  double r0 = P[i].r, x0 = P[i].x, y0 = P[i].y
  double r1 = P[j].r, x1 = P[j].x, y1 = P[j].y;
  point center;
  if (d > r0 + r1) continue;
  if (d < fabs(r0 - r1) || fabs(d) < EPS) {
   if (r0 < r1) center = P[i];
   else center = P[j];
  } else {
   double a = (r0*r0 - r1*r1 + d*d)/(2*d);
   double h = sqrt(r0*r0 - a*a);
   double x2 = x0 + a*(x1 - x0)/d;
   double y2 = y0 + a*(y1 - y0)/d;
   double translationY = h*(y1 - y0)/d;
   double translationX = h*(x1 - x0)/d;
   center.x = x2 + translationY;
   center.y = y2 - translationX;
   ans = max(ans, go(center));
   center.x = x2 - translationY;
   center.y = y2 + translationX;
  }
  ans = max(ans, go(center));
 }
}
// line segment with circle intersect
private int FindLineCircleIntersections(
    float cx, float cy, float radius,
    PointF point1, PointF point2,
    out PointF intersection1, out PointF intersection2)
{
    float dx, dy, A, B, C, det, t;
    dx = point2.X - point1.X;
    dy = point2.Y - point1.Y;
    A = dx * dx + dy * dy;
    B = 2 * (dx * (point1.X - cx) + dy * (point1.Y - cy));
    C = (point1.X - cx) * (point1.X - cx) +
        (point1.Y - cy) * (point1.Y - cy) -
        radius * radius;
    det = B * B - 4 * A * C;
    if ((A <= 0.0000001) || (det < 0)) {
        // No real solutions.
        intersection1 = new PointF(float.NaN, float.NaN);
        intersection2 = new PointF(float.NaN, float.NaN);
        return 0;
    }   else if (det == 0)  {
        // One solution.
        t = -B / (2 * A);
        intersection1 =
            new PointF(point1.X + t * dx, point1.Y + t * dy);
        intersection2 = new PointF(float.NaN, float.NaN);
        return 1;
    } else {
      // Two solutions.
      t = (float)((-B + Math.Sqrt(det)) / (2 * A));
      intersection1 = new PointF(point1.X + t * dx, point1.Y
            + t * dy);
      t = (float)((-B - Math.Sqrt(det)) / (2 * A));
      intersection2 = new PointF(point1.X + t * dx, point1.Y
            + t * dy);
      return 2;
    }
 }
```

3.7 monotoneChain

```
// Implementation of Andrew's monotone chain 2D convex hull
    algorithm.
// Asymptotic complexity: O(n log n).
// Practical performance: 0.5-1.0 seconds for n=1000000 on a
    1GHz machine.
#include <algorithm>
#include <vector>
using namespace std;
typedef double coord_t;      // coordinate type
```

```
 }
typedef double coord2_t; // must be big enough to hold 2*max
    (|coordinate|)^2
struct Point {
 coord_t x, y;
 bool operator <(const Point &p) const {
  return x < p.x || (x == p.x && y < p.y);

};

// 2D cross product of OA and OB vectors, i.e. z-component
    of their 3D cross product.
// Returns a positive value, if OAB makes a counter-
    clockwise turn,
// negative for clockwise turn, and zero if the points are
    collinear.
coord2_t cross(const Point &O, const Point &A, const Point &
    B){
 return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x
     );
}
// Returns a list of points on the convex hull in counter-
    clockwise order.
// Note: the last point in the returned list is the same as
    the first one.
vector<Point> convex_hull(vector<Point> P){
 int n = P.size(), k = 0;
 vector<Point> H(2*n);
 // Sort points lexicographically
 sort(P.begin(), P.end());
 // Build lower hull
 for (int i = 0; i < n; ++i) {
  while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
  H[k++] = P[i];
 }
 // Build upper hull
 for (int i = n-2, t = k+1; i >= 0; i--) {
  while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
  H[k++] = P[i];
 }
 H.resize(k-1);
 return H;
}
```

3.8 smallestEnclosingCircle

```
//declare variables here.
struct point {
 double x;
 double y;
};

struct circle {
 double x; double y; double r;
 circle() {}
 circle(double x, double y, double r): x(x), y(y), r(r) {}
};

circle f(vector<point> R) {
 if (R.size() == 0) {
  return circle(0, 0, -1);
 }
 else if (R.size() == 1) {
  return circle(R[0].x, R[0].y, 0);
 }
 else if (R.size() == 2) {
  return circle((R[0].x+R[1].x)/2.0, (R[0].y+R[1].y)/2.0,
      hypot(R[0].x-R[1].x, R[0].y-R[1].y)/2.0);
 } else {
  double D = (R[0].x - R[2].x)*(R[1].y - R[2].y) - (R[1].x
      - R[2].x)*(R[0].y - R[2].y);
  double p0 = (((R[0].x - R[2].x)*(R[0].x + R[2].x) + (R[0].
      y - R[2].y)*(R[0].y + R[2].y)) / 2 * (R[1].y - R[2].y)
       - ((R[1].x - R[2].x)*(R[1].x + R[2].x) + (R[1].y - R
      [2].y)*(R[1].y + R[2].y)) / 2 * (R[0].y - R[2].y))/D;
  double p1 = (((R[1].x - R[2].x)*(R[1].x + R[2].x) + (R[1].
      y - R[2].y)*(R[1].y + R[2].y)) / 2 * (R[0].x - R[2].x)
       - ((R[0].x - R[2].x)*(R[0].x + R[2].x) + (R[0].y - R
      [2].y)*(R[0].y + R[2].y)) / 2 * (R[1].x - R[2].x))/D;
  return circle(p0, p1, hypot(R[0].x - p0, R[0].y - p1));
 }
}

circle fmini(vector<point>& P, int i, vector<point> R) {
 if (i == P.size() || R.size() == 3) {
  return f(R);
 } else {
  circle D = fmini(P, i+1, R);
  if (hypot(P[i].x-D.x, P[i].y-D.y) > D.r) {
   R.push_back(P[i]);
   D = fmini(P, i+1, R);
  }
  return D;
 }
}

circle minidisk(vector<point> P) {
 random_shuffle(P.begin(), P.end());
 return fmini(P, 0, vector<point>());
}
```

3.9 twoSegmentIntersect

```
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator-(const pt& p) const { return pt(x - p.x, y -
        p.y); }
    long long cross(const pt& p) const { return x * p.y - y
        * p.x; }
    long long cross(const pt& a, const pt& b) const { return
        (a - *this).cross(b - *this); }
};

int sgn(const long long& x) { return x >= 0 ? x
    ? 1 : 0 : -1; }

bool inter1(long long a, long long b, long long c, long long
    d) {
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    return max(a, c) <= min(b, d);
}

bool check_inter(const pt& a, const pt& b, const pt& c,
    const pt& d) {
    if (c.cross(a, d) == 0 && c.cross(b, d) == 0)
        return inter1(a.x, b.x, c.x, d.x) && inter1(a.y, b.y
            , c.y, d.y);
    return sgn(a.cross(b, c)) != sgn(a.cross(b, d)) &&
        sgn(c.cross(d, a)) != sgn(c.cross(d, b));
}
```

4 Graph

4.1 Flow

4.1.1 Bipartite Matching

```
//To handle some corner cases, don't forget to randomize the
    edge order
struct BipartiteMatcher {
```

```
  G(n), L(n, -1), R(m, -1), Viz(n) {}
  vector<vector<int>> G;
  vector<int> L, R, Viz;

  BipartiteMatcher(int n, int m) :

  void AddEdge(int a, int b) {
    G[a].push_back(b);
  }

  bool Match(int node) {
    if (Viz[node])
      return false;
    Viz[node] = true;

    for (auto vec : G[node]) {
      if (R[vec] == -1) {
        L[node] = vec;
        R[vec] = node;
        return true;
      }
    }

    for (auto vec : G[node]) {
      if (Match(R[vec])) {
        L[node] = vec;
        R[vec] = node;
        return true;
      }
    }

    return false;
  }

  int Solve() {
    int ok = true;
    while (ok--) {
      fill(Viz.begin(), Viz.end(), 0);
      for (int i = 0; i < (int)L.size(); ++i)
        if (L[i] == -1)
          ok |= Match(i);
    }

    int ret = 0;
    for (int i = 0; i < L.size(); ++i)
      ret += (L[i] != -1);
    return ret;
  }
};
```

4.1.2 Dinic

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
        cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
            ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap
                - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

4.1.3 Edmond-Karp Max flow

```
#define MAXNODE 1000
#define INF 1000000007
int capacity[MAXNODE+5][MAXNODE+5];
vector<vector<int>> adj;

int bfs(int s, int t, vector<int>& parent) {
    fill(parent.begin(), parent.end(), -1);
    parent[s] = -2;
    queue<pair<int, int>> q;
```

```
        int cur = q.front().first;
    q.push({s, INF});

    while (!q.empty()) {

int
     flow = q.front().second;
        q.pop();

        for (int next : adj[cur]) {
            if (parent[next] == -1 && capacity[cur][next]) {
                parent[next] = cur;
                int new_flow = min(flow, capacity[cur][next]);
                if (next == t)
                    return new_flow;
                q.push({next, new_flow});
            }
        }
    }

    return 0;
}

int maxflow(int s, int t) {
    int flow = 0;
    vector<int> parent(MAXNODE);
    int new_flow;

    while (new_flow = bfs(s, t, parent)) {
        flow += new_flow;
        int cur = t;
        while (cur != s) {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }

    return flow;
}
```

4.1.4 Min-cost flow

```
struct Edge
{
    int from, to, capacity, cost;
    Edge(int from, int to, int capacity, int cost){
     this->from = from;
     this->to = to;
     this->capacity = capacity;
     this->cost = cost;
    }
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<
     int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<int> m(n, 2);
    deque<int> q;
    q.push_back(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        m[u] = 0;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v
                 ]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (m[v] == 2) {
                    m[v] = 1;
                    q.push_back(v);
                } else if (m[v] == 0) {
                    m[v] = 1;
                    q.push_front(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s,
     int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return 1;
    else
        return cost;
}
```

4.1.5 hopcoft

```
// C++ implementation of Hopcroft Karp algorithm for
// maximum matching
#include <iostream>
#include <cstdlib>
#include <queue>
#include <list>
#include <climits>
#define NIL 0
#define INF INT_MAX

// A class to represent Bipartite graph for
```

```
    // and right sides of Bipartite Graph
// Hopcroft Karp implementation
class BGraph
{
    // m and n are number of vertices on left

int
     m, n;

    // adj[u] stores adjacents of left side
    // vertex 'u'. The value of u ranges from 1 to m.
    // 0 is used for dummy vertex
    std::list<int> *adj;

    // pointers for hopcroftKarp()
    int *pair_u, *pair_v, *dist;

public:
    BGraph(int m, int n);   // Constructor
    void addEdge(int u, int v); // To add edge

    // Returns true if there is an augmenting path
    bool bfs();

    // Adds augmenting path if there is one beginning
    // with u
    bool dfs(int u);

    // Returns size of maximum matching
    int hopcroftKarpAlgorithm();
};

// Returns size of maximum matching
int BGraph::hopcroftKarpAlgorithm()
{
    // pair_u[u] stores pair of u in matching on left side of
         Bipartite Graph.
    // If u doesn't have any pair, then pair_u[u] is NIL
    pair_u = new int[m + 1];

    // pair_v[v] stores pair of v in matching on right side
        of Biparite Graph.
    // If v doesn't have any pair, then pair_u[v] is NIL
    pair_v = new int[n + 1];

    // dist[u] stores distance of left side vertices
    dist = new int[m + 1];

    // Initialize NIL as pair of all vertices
    for (int u = 0; u <= m; u++)
        pair_u[u] = NIL;
    for (int v = 0; v <= n; v++)
        pair_v[v] = NIL;

    // Initialize result
    int result = 0;

    // Keep updating the result while there is an
    // augmenting path possible.
    while (bfs())
    {
        // Find a free vertex to check for a matching
        for (int u = 1; u <= m; u++)

            // If current vertex is free and there is
            // an augmenting path from current vertex
            // then increment the result
            if (pair_u[u] == NIL && dfs(u))
                result++;
    }
    return result;
}

// Returns true if there is an augmenting path available,
     else returns false
bool BGraph::bfs()
{
    std::queue<int> q; //an integer queue for bfs

    // First layer of vertices (set distance as 0)
    for (int u = 1; u <= m; u++)
    {
        // If this is a free vertex, add it to queue
        if (pair_u[u] == NIL)
        {
            // u is not matched so distance is 0
            dist[u] = 0;
            q.push(u);
        }

        // Else set distance as infinite so that this vertex
             is considered next time for availibility
        else
            dist[u] = INF;
    }

    // Initialize distance to NIL as infinite
    dist[NIL] = INF;

    // q is going to contain vertices of left side only.
    while (!q.empty())
    {
        // dequeue a vertex
        int u = q.front();
        q.pop();

        // If this node is not NIL and can provide a shorter
             path to NIL then
        if (dist[u] < dist[NIL])
        {
            // Get all the adjacent vertices of the dequeued
                 vertex u
            std::list<int>::iterator it;
            for (it = adj[u].begin(); it != adj[u].end(); ++
                 it)
            {
                int v = *it;

                // If pair of v is not considered so far
                // i.e. (v, pair_v[v]) is not yet explored
                     edge.
                if (dist[pair_v[v]] == INF)
                {
                    // Consider the pair and push it to queue
                    dist[pair_v[v]] = dist[u] + 1;
                    q.push(pair_v[v]);
                }
            }
        }
    }

    // If we could come back to NIL using alternating path of
         distinct
    // vertices then there is an augmenting path available
    return (dist[NIL] != INF);
}

// Returns true if there is an augmenting path beginning
     with free vertex u
bool BGraph::dfs(int u)
{
    if (u != NIL)
    {
        std::list<int>::iterator it;
        for (it = adj[u].begin(); it != adj[u].end(); ++it)
        {
            // Adjacent vertex of u
            int v = *it;

            // Follow the distances set by BFS search
```

```
            {   // new matching possible, store the
                matching
            if (dist[pair_v[v]] == dist[u] + 1)
            {
                // If dfs for pair of v also returnn true then
                if (dfs(pair_v[v]) == true)

pair_v
    [v] = u;
                    pair_u[u] = v;
                    return true;
                }
            }
        }

        // If there is no augmenting path beginning with u
            then.
        dist[u] = INF;
        return false;
    }
    return true;
}

// Constructor for initialization
BGraph::BGraph(int m, int n)
{
    this->m = m;
    this->n = n;
    adj = new std::list<int>[m + 1];
}

// function to add edge from u to v
void BGraph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add v to 'us list.
}

int main()
{
    int v1, v2, e;
    std::cin >> v1 >> v2 >> e; // vertices of left side,
        right side and edges
    BGraph g(v1, v2); //
    int u, v;
    for (int i = 0; i < e; ++i)
    {
        std::cin >> u >> v;
        g.addEdge(u, v);
    }
    int res = g.hopcroftKarpAlgorithm();
    std::cout << "Maximum matching is " << res <<"\n";

    return 0;
}
```

4.2 LCABinaryLifting

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

4.3 LCARmq

```
/**
* Description: Euler Tour LCA w/ O(1) query
* Source: own
* Verification: Debug the Bugs
* Dependency: Range Minimum Query
*/

template<int SZ> struct LCA {
 vi adj[SZ];
 RMQ<pi,2*SZ> r;
 vpi tmp;
 int depth[SZ], pos[SZ];
 int N, R = 1;

 void addEdge(int u, int v) {
  adj[u].pb(v), adj[v].pb(u);
 }

 void dfs(int u, int prev){
  pos[u] = sz(tmp); depth[u] = depth[prev]+1;
  tmp.pb({depth[u],u});
  for (int v: adj[u]) if (v != prev) {
   dfs(v, u);
   tmp.pb({depth[u],u});
  }
 }

 void init(int _N) {
  N = _N;
  dfs(R, 0);
  r.build(tmp);
 }

 int lca(int u, int v){
  u = pos[u], v = pos[v];
  if (u > v) swap(u,v);
  return r.query(u,v).s;
 }

 int dist(int u, int v) {
```

```
  return depth[u]+depth[v]-2*depth[lca(u,v)];
 }
};
```

4.4 LCATarjan

```
vector<vector<int>> adj;
vector<vector<int>> queries;
vector<int> ancestor;
vector<bool> visited;

void dfs(int v)
{
    visited[v] = true;
    ancestor[v] = v;
    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
            union_sets(v, u);
            ancestor[find_set(v)] = v;
        }
    }
    for (int other_node : queries[v]) {
        if (visited[other_node])
            cout << "LCA of " << v << " and " << other_node
                 << " is " << ancestor[find_set(other_node)
                     ] << ".\n";
    }
}

void compute_LCAs() {
    // initialize n, adj and DSU
    // for (each query (u, v)) {
    //    queries[u].push_back(v);
    //    queries[v].push_back(u);
    // }

    ancestor.resize(n);
    visited.assign(n, false);
    dfs(0);
}
```

4.5 MST

4.5.1 secondBestMSTLCA

```
struct edge {
    int s, e, w, id;
    bool operator<(const struct edge& other) { return w <
        other.w; }
};
typedef struct edge Edge;

const int N = 2e5 + 5;
long long res = 0, ans = 1e18;
int n, m, a, b, w, id, l = 21;
vector<Edge> edges;
vector<int> h(N, 0), parent(N, -1), size(N, 0), present(N
    , 0);
vector<vector<pair<int, int>>> adj(N), dp(N, vector<pair<int
    , int>>(l));
vector<vector<int>> up(N, vector<int>(l, -1));

pair<int, int> combine(pair<int, int> a, pair<int, int> b) {
    vector<int> v = {a.first, a.second, b.first, b.second};
    int topTwo = -3, topOne = -2;
    for (int c : v) {
        if (c > topOne) {
            topTwo = topOne;
            topOne = c;
        } else if (c > topTwo && c < topOne) {
            topTwo = c;
        }
    }
    return {topOne, topTwo};
}

void dfs(int u, int par, int d) {
    h[u] = 1 + h[par];
    up[u][0] = par;
    dp[u][0] = {d, -1};
    for (auto v : adj[u]) {
        if (v.first != par) {
            dfs(v.first, u, v.second);
        }
    }
}

pair<int, int> lca(int u, int v) {
    pair<int, int> ans = {-2, -3};
    if (h[u] < h[v]) {
        swap(u, v);
    }
    for (int i = l - 1; i >= 0; i--) {
        if (h[u] - h[v] >= (1 << i)) {
            ans = combine(ans, dp[u][i]);
            u = up[u][i];
        }
    }
    if (u == v) {
        return ans;
    }
    for (int i = l - 1; i >= 0; i--) {
        if (up[u][i] != -1 && up[v][i] != -1 && up[u][i] !=
            up[v][i]) {
            ans = combine(ans, combine(dp[u][i], dp[v][i]));
            u = up[u][i];
            v = up[v][i];
        }
    }
    ans = combine(ans, combine(dp[u][0], dp[v][0]));
    return ans;
}

int main(void) {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        parent[i] = i;
        size[i] = 1;
    }
    for (int i = 1; i <= m; i++) {
        cin >> a >> b >> w; // 1-indexed
        edges.push_back({a, b, w, i - 1});
    }
    sort(edges.begin(), edges.end());
    for (int i = 0; i <= m - 1; i++) {
        a = edges[i].s;
        b = edges[i].e;
        w = edges[i].w;
        id = edges[i].id;
        if (unite_set(a, b)) {
            adj[a].emplace_back(b, w);
            adj[b].emplace_back(a, w);
            present[id] = 1;
            res += w;
        }
    }
    dfs(1, 0, 0);
    for (int i = 1; i <= l - 1; i++) {
        for (int j = 1; j <= n; ++j) {
            if (up[j][i - 1] != -1) {
                int v = up[j][i - 1];
                up[j][i] = up[v][i - 1];
                dp[j][i] = combine(dp[j][i - 1], dp[v][i - 1])
                    ;
            }
        }
```

```
            if (rem.first != w) {
    }
    for (int i = 0; i <= m - 1; i++) {
        id = edges[i].id;
        w = edges[i].w;
        if (!present[id]) {
            auto rem = lca(edges[i].s, edges[i].e);

if
     (ans > res + w - rem.first) {
                    ans = res + w - rem.first;
                }
            } else if (rem.second != -1) {
                if (ans > res + w - rem.second) {
                    ans = res + w - rem.second;
                }
            }
        }
    }
    cout << ans << "\n";
    return 0;
}
```

4.6 SCC

4.6.1 2SAT

```
int n;
vector<vector<int>> g, gt;
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    used[v] = true;
    for (int u : g[v]) {
        if (!used[u])
            dfs1(u);
    }
    order.push_back(v);
}

void dfs2(int v, int cl) {
    comp[v] = cl;
    for (int u : gt[v]) {
        if (comp[u] == -1)
            dfs2(u, cl);
    }
}

bool solve_2SAT() {
    used.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!used[i])
            dfs1(i);
    }

    comp.assign(n, -1);
    for (int i = 0, j = 0; i < n; ++i) {
        int v = order[n - i - 1];
        if (comp[v] == -1)
            dfs2(v, j++);
    }

    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}
```

4.6.2 Kosaraju

```
vector < vector<int> > g, gr;
vector<bool> used;
vector<int> order, component;

void dfs1 (int v) {
 used[v] = true;
 for (size_t i=0; i<g[v].size(); ++i)
  if (!used[ g[v][i] ])
   dfs1 (g[v][i]);
 order.push_back (v);
}

void dfs2 (int v) {
 used[v] = true;
 component.push_back (v);
 for (size_t i=0; i<gr[v].size(); ++i)
  if (!used[ gr[v][i] ])
   dfs2 (gr[v][i]);
}

int main() {
 int n;
 // ... reading n ...
 for (;;) {
  int a, b;
  // ... reading next edge (a,b) ...
  g[a].push_back (b);
  gr[b].push_back (a);
 }
 used.assign (n, false);
 for (int i=0; i<n; ++i)
  if (!used[i])
   dfs1 (i);
 used.assign (n, false);
 for (int i=0; i<n; ++i) {
  int v = order[n-1-i];
  if (!used[v]) {
   dfs2 (v);
   // ... printing next component ...
   component.clear();
  }
 }
}
```

4.7 Shortest Path

4.7.1 BellmanFord

```
vi dist(V, INF); dist[s] = 0;

for (int i = 0; i < V - 1; i++) // relax all E edges V-1
     times
 for (int u = 0; u < V; u++) // these two loops = O(E),
      overall O(VE)
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
   ii v = AdjList[u][j];
   // record SP spanning here if needed
   dist[v.first] = min(dist[v.first], dist[u] + v.second);
   // relax
  }

// after running the O(VE) Bellman 'Fords algorithm shown
     above
bool hasNegativeCycle = false;
for (int u = 0; u < V; u++) // one more pass to check
 for (int j = 0; j < (int)AdjList[u].size(); j++) {
  ii v = AdjList[u][j];
  if (dist[v.first] > dist[u] + v.second) // if this is
       still possible
  hasNegativeCycle = true;
  // then negative cycle exists!
 }
```

```
printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes
     " : "No");
```

4.7.2 Dijkstra

```
/**
* Description: shortest path!
* Works with negative edge weights (aka SPFA?)
*/

template<class T> using pqg = priority_queue<T,vector<T>,
     greater<T>>;

template<class T> T poll(pqg<T>& x) {
 T y = x.top(); x.pop();
 return y;
}

template<int SZ> struct Dijkstra {
 ll dist[SZ];
 vpi adj[SZ];
 pqg<pl> q;

 void addEdge(int A, int B, int C) {
  adj[A].pb({B,C}), adj[B].pb({A,C});
 }

 void gen(int st) {
  fill_n(dist,SZ,INF);
  q = pqg<pl>(); q.push({dist[st] = 0,st});
  while (sz(q)) {
   auto x = poll(q);
   if (dist[x.s] < x.f) continue;
   for (auto y: adj[x.s]) if (x.f+y.s < dist[y.f])
    q.push({dist[y.f] = x.f+y.s,y.f});
  }
 }
};
```

4.8 centroidDecomposition

```
const int maxn = 1e5 + 5;
vector<vi> AdjList, centroid;
int id[maxn],sz[maxn];

void dfsSizeSubtree(int node, int p) {
    sz[node] = 1;
    for(int i=0;i<AdjList[node].size();i++){
        int v = AdjList[node][i];
        if(v!=p) {
        dfsSizeSubtree(v, node);
            sz[node] += sz[v];
        }
    }
}

int findCentroid(int node, int ukuran, int label) {
    bool valid = true;
    for(int i=0;i<AdjList[node].size();i++){
        int v = AdjList[node][i];
        if(sz[v] > ukuran/2 && id[v] == -1) {
            //pindah ke centroid itu
            valid = false;
            sz[node] = ukuran-sz[v];
            sz[v] = ukuran;
            int cen = findCentroid(v, ukuran, label);
            id[cen] = label;
            return cen;
        }
    }
    if(valid) {
        id[node] = label;
        return node;
    }
    return -1;
}

int dfsCentroid(int node, int label) {
    int root = findCentroid(node, sz[node], label);
    id[root] = label
    for(int i=0;i<AdjList[root].size();i++){
        int v = AdjList[root][i];
        if(id[v] == -1){
            int cen = dfsCentroid(v, label+1);
            centroid[root].pb(cen); centroid[cen].pb(root);
        }
    }
    return root;
}

//MAIN
int n,i,j; scanf("%d",&n);
AdjList.assign(maxn, vi());
centroid.assign(maxn, vi());
for(i=0;i<n-1;i++){
    int a,b; scanf("%d %d",&a,&b);
    AdjList[a].pb(b); AdjList[b].pb(a);
}
dfsSizeSubtree(1, -1);
memset(id, -1, sizeof id);
int rootCentroid = dfsCentroid(1,0);
```

4.9 findBridge

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
                //IS_BRIDGE(a, b) function that process fact
                     edge (a, b)
                //is a bridge
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

4.10 heavyLightDecomposition

```
const int maxn = 1e4 + 4;
vector<vii> AdjList;
```

```
    int maks = 0;
vi parent, depth, heavy, head, pos;
vector<vi> indexx;
int curPos,sz[maxn],w[maxn],tempW[maxn],akhir[maxn];

int dfs(int node, int p, int dalam) {
    sz[node] = 1;

depth
    [node] = dalam;
    for(int i=0;i<AdjList[node].size();i++){
        ii v = AdjList[node][i];
        int tetangga = v.first, berat = v.second;
        if(tetangga != p) {
            int idx = indexx[node][i];
            akhir[idx] = tetangga;
            parent[tetangga] = node;
            int ukuranTetangga = dfs(tetangga, node, dalam+1)
                 ;
            sz[node] += ukuranTetangga;
            tempW[tetangga] = berat;
            if(maks < ukuranTetangga) {
                maks = ukuranTetangga;
                heavy[node] = tetangga;
            }
        }
    }
    return sz[node];
}

void decompose(int node, int h) {
    head[node] = h, pos[node] = curPos++;
    if(heavy[node] != -1)
    decompose(heavy[node], h);
    for(int i=0;i<AdjList[node].size();i++){
        ii v = AdjList[node][i];
        int tetangga = v.first, berat = v.second;
        if(parent[node] != tetangga && heavy[node] !=
             tetangga)
            decompose(tetangga, tetangga);
    }
}

void assignWeight(int n) {
    for(int i=1;i<=n;i++){
        int posisi = pos[i];
        w[posisi] = tempW[i];
    }
}
```

```
int st[4*maxn];
int left(int n){return n<<1;}
int right(int n){return (n<<1) + 1;}

void build(int node, int l, int r) {
    if(l == r) {
        st[node] = w[l];
        return;
    }
    int mid = (l+r)/2;
    build(left(node), l, mid);
    build(right(node), mid+1, r);
    st[node] = max(st[left(node)], st[right(node)]);
}

int query(int node, int l, int r, int i, int j) {
    if(j < l || r < i) {return -inf;}
    if(i <= l && r <= j){return st[node];}
    int mid = (l+r)/2;
    int ans1 = query(left(node), l, mid, i, j);
    int ans2 = query(right(node), mid+1, r, i, j);
    return max(ans1, ans2);
}

void update(int node, int l, int r, int idx, int val) {
    if(r < idx || l > idx){return;}
    if(idx == r && idx == l){
        st[node] = val;
        return;
    }
    int mid = (l+r)/2;
    update(left(node), l, mid, idx, val);
    update(right(node), mid+1, r, idx, val);
    st[node] = max(st[left(node)], st[right(node)]);
}

void updateQuery(int idx, int val) {
    int nodeAkhir = akhir[idx];
    update(1,1,curPos,pos[nodeAkhir], val);
}

int jawabQuery(int a, int b) {
    if(a == b) return 0;
    int ans = -inf;
    for(; head[a] != head[b]; b = parent[head[b]]) {
        if(depth[head[a]] > depth[head[b]]) swap(a,b);
        int cur = query(1,1,curPos,pos[head[b]], pos[b]);
        ans = max(ans, cur);
    }
    if(depth[a] > depth[b]) swap(a,b);
```

```
    int cur = query(1,1,curPos,pos[a]+1, pos[b]);
    ans = max(ans, cur);
    return ans;
}

void preprocess() {
    int n = AdjList.size();
    parent = vi(n+1);
    depth = vi(n+1);
    heavy = vi(n+1, -1);
    head = vi(n+1);
    pos = vi(n+1);
    curPos = 1;
    dfs(1, -1,0);
    decompose(1,1);
    assignWeight(n);
    build(1,1,curPos);
}

// MAIN
// INPUT
preprocess()
```

5 Math

5.1 BigInt

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;

string int_to_string(int i, int width, bool zero) {
    string res = "";
    while (width--) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}

struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }
```

```
            s[len] = num % BASE;

    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {

num
     /= BASE;
            len ++;
        }
        s[len++] = num;
    }

    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -= BASE_LENGTH) {
            int tmp = 0;
            int k = i - BASE_LENGTH + 1;
            if (k < 0) k = 0;
            for (int j = k; j <= i; j++) {
                tmp = tmp*10 + num[j] - '0';
            }
            s[index++] = tmp;
        }
    }

    void clean() {
        while(len > 1 && !s[len-1]) len--;
    }

    string str() const {
        string ret = "";
        if (len == 1 && !s[0]) return "0";
        for(int i = 0; i < len; i++) {
            if (i == 0) {
                ret += int_to_string(s[len - i - 1],
                     BASE_LENGTH, false);
            } else {
                ret += int_to_string(s[len - i - 1],
                     BASE_LENGTH, true);
            }
        }
        return ret;
    }

    unsigned long long ll() const {
        unsigned long long ret = 0;
        for(int i = len-1; i >= 0; i--) {
            ret *= BASE;
            ret += s[i];
        }
        return ret;
    }

    bigint operator + (const bigint& b) const {
        bigint c = b;
        while (c.len < len) c.s[c.len++] = 0;
        c.s[c.len++] = 0;
        bool r = 0;
        for (int i = 0; i < len || r; i++) {
            c.s[i] += (i<len)*s[i] + r;
            r = c.s[i] >= BASE;
            if (r) c.s[i] -= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator - (const bigint& b) const {
        if (operator < (b)) throw "cannot do subtract";
        bigint c = *this;
        bool r = 0;
        for (int i = 0; i < b.len || r; i++) {
            c.s[i] -= b.s[i];
            r = c.s[i] < 0;
            if (r) c.s[i] += BASE;
        }
        c.clean();
        return c;
    }

    bigint operator * (const bigint& b) const {
        bigint c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++)
            for(int j = 0; j < b.len; j++)
                c.s[i+j] += s[i] * b.s[j];
        for(int i = 0; i < c.len-1; i++){
            c.s[i+1] += c.s[i] / BASE;
            c.s[i] %= BASE;
        }
        c.clean();
        return c;
    }

    bigint operator / (const int b) const {
        bigint ret;
        int down = 0;
        for (int i = len - 1; i >= 0; i--) {
            ret.s[i] = (s[i] + down * BASE) / b;
            down = s[i] + down * BASE - ret.s[i] * b;
        }
        ret.len = len;
        ret.clean();
        return ret;
    }

    bool operator < (const bigint& b) const {
        if (len < b.len) return true;
        else if (len > b.len) return false;
        for (int i = 0; i < len; i++)
            if (s[i] < b.s[i]) return true;
            else if (s[i] > b.s[i]) return false;
        return false;
    }

    bool operator == (const bigint& b) const {
        return !(*this<b) && !(b<(*this));
    }

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};
```

5.2 Number Theory

5.2.1 CRT

```
#include <bits/stdc++.h>
using namespace std;
  // Returns modulo inverse of a with respect to m using
      extended
// Euclid Algorithm. Refer below post for details:
// https://www.geeksforgeeks.org/multiplicative-inverse-
     under-modulo-m/
int inv(int a, int m) {
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;
    if (m == 1)
      return 0;
    // Apply extended Euclid Algorithm
    while (a > 1) {
        // q is quotient
        q = a / m;
        t = m;
        // m is remainder now, process same as
```

```
        x1 = t;
        // euclid's algo
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;

}

    // Make x1 positive
    if (x1 < 0)
       x1 += m0;
    return x1;
}
// k is size of num[] and rem[]. Returns the smallest
// number x such that:
//  x % num[0] = rem[0],
//  x % num[1] = rem[1],
//  ..................
//  x % num[k-2] = rem[k-1]
// Assumption: Numbers in num[] are pairwise coprime
// (gcd for every pair is 1)
int findMinX(int num[], int rem[], int k) {
    // Compute product of all numbers
    int prod = 1;
    for (int i = 0; i < k; i++)
        prod *= num[i];
    // Initialize result
    int result = 0;
    // Apply above formula
    for (int i = 0; i < k; i++) {
        int pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) * pp;
    }
    return result % prod;
}
```

5.2.2 InverseModulo

```
/**
* Description : find x such that ax = 1 mod m
*/
/* case 1 : when(gcd(a,m) = 1) */
/* use extended euclid : find x such that ax + my = 1 */
/* store x, y, and d as global variables */
/* d = gcd */
void extendedEuclid(int a, int b) {
  if (b == 0) { x = 1; y = 0; d = a; return; }
  /* base case */
  extendedEuclid(b, a % b);
  /* similar as the original gcd */
  int x1 = y;
  int y1 = x - (a / b) * y;
  x = x1;
  y = y1;
}
/* compute the first case inverse modulo*/
int firstInverseModulo(int a, int m){
  /* produces x and y, such that ax + my = 1 */
  /* return a^-1 mod m */
  extendedEuclid(a, m);
  return (x + m)%m;
}
/* case 2 : m is prime */
/* a^(m-1) = 1 mod m */
/* a^(m-2) = a^-1 mod m */
int power(int a,int b){
  int res = 1;
  while (b > 0){
    if (b%2 == 1)
      res *= a;
    b /= 2;
    a *= a;
  }
  return res;
}
int secondInverseModulo(int a,int m){
  return power(a, m-2);
}
```

5.2.3 PrimeFactor

```
/**
* Description : some function that have relation with prime
    factor
*/

/* find prime factor */
vector<long long> primefactor(long long N){
    vector<long long> factors;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
       while (N%PF == 0){
           N /= PF;
           factors.push_back(PF);
       }
       PF = primes[++idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}

/* number of divisor */
long long numDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        ans *= (power + 1);
        PF = primes[++idx];
    }
    if (N != 1) ans *= 2;
    return ans;
}

/* sum of divisor */
long long sumDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        /* 1 + PF + PF^2 + PF^3 + ... + PF^pow = (a.r^n - 1)
              / (r-1) */
        ans *= ((long long)pow((double)PF, power + 1.0) - 1)
              / (PF - 1);
        PF = primes[++idx];
    }
    if (N != 1) ans *= ((long long)pow((double)N, 2.0) - 1)
          / (N - 1);
    return ans;
}

/* Euler Phi */
long long eulerPhi(long long N){
    long long idx = 0;
    long long PF = primes[idx];
    long long ans = N;
    while (PF <= (long long)sqrt(N)){
```

```
    if (N != 1) ans -= ans / N;
        if (N%PF == 0) ans -= ans / PF;
        while (N%PF == 0) N /= PF;
        PF = primes[++idx];
    }

return
     ans;
}
```

5.2.4 Sieve

```
/**
* Description :Test Primality up to n in O(log(logn))
*/

const int SZ = 1e7;
bitset<SZ> bs;
vector<long long> primes;

void sieve(){
    bs.set();
    bs[0] = false; bs[1] = false;
    for (long long i = 2; i <= SZ; i++){
        if (bs[i]){
            primes.push_back(i);
            for (long long j = i * i; j <= SZ; j+=i)
                bs[j] = false;
        }
    }
}
```

5.2.5 millerRabin

```
def millerTest(d, n):
 lon = int(math.log(n))
 # b = min(n-2, 2*lon*lon)
 a = random.randrange(2, n-2)
 x = power(a, d, n)
 if (x == 1 or x == n-1):
  return True
 while (d != n-1):
  x = (x *x) % n
  d *= 2
  if (x == 1):
   return False
  if (x == n-1):
   return True
 return False

def isPrime(n, k):
 if (n <= 1 or n == 4):
  return False
 if (n <= 3):
  return True

 d = n-1
 while (d % 2 == 0):
  d /= 2

 for i in range(0, k):
  if (not(millerTest(d, n))):
   return False
 return True
# factorization a number in O(n^1/3)
def fastFactorization(n):
 res = 1
 for pf in primes:
  if (pf * pf * pf > n):
   break
  cnt = 1
  while (n%pf == 0):
   n /= pf
   cnt+=1
  res *= (cnt)
 sqt = int(math.sqrt(n))
 if (isPrime(n, 10)):
  res *= 2
 elif (sqt * sqt == n and isPrime(sqt, 10)):
  res *= 3
 elif (n != 1):
  res *= 4
 return res
```

5.2.6 multiplicationOverflow

```
// calculate a * b % mod. mod may fit in 64-bit,
// but a*b might exceed. two ways, one using
// fast exponentiation style which is safe but
// slower, or using long double as written below (safe
     enough i guess)

long long mul(long long a, long long b, long long mod) {
  long double res = a;
  res *= b;
  long long c = (long long)(res / mod);
  a *= b;
  a -= c * mod;
  a %= mod;
  if (a < 0) a += mod;
  return a;
}
```

5.3 Polynomial

5.3.1 FFT mod

```
/*
Description: Allows multiplication of polynomials in
general moduli.
Verification:
http://codeforces.com/contest/960/submission/37085144
*/

namespace FFTmod {
 int get(int s) {
  return s > 1 ? 32 - __builtin_clz(s - 1) : 0;
 }
 void fft(vcd& a, bool inv){
  int n = sz(a), j = 0;
  vcd roots(n/2);
  FOR(i,1,n) {
   int bit = (n >> 1);
   while (j >= bit){
    j -= bit;
    bit >>= 1;
   }
   j += bit;
   if(i < j) swap(a[i], a[j]);
  }

  ld ang = 2 * M_PIl / n * (inv ? -1 : 1);
  FOR(i,n/2) roots[i] = cd(cos(ang * i), sin(ang * i));

  for (int i=2; i<=n; i<<=1){
   int step = n / i;
   for(int j=0; j<n; j+=i){
    for(int k=0; k<i/2; k++){
     cd u = a[j+k], v = a[j+k+i/2] * roots[step * k];
     a[j+k] = u+v;
     a[j+k+i/2] = u-v;
    }
   }
  }
  if (inv) FOR(i,n) a[i] /= n;
 }
```

```
  vcd v1(n), v2(n), r1(n), r2(n);
  FOR(i,sz(a)) v1[i] = cd(a[i] >> 15, a[i] & 32767);

 vl conv(vl a, vl b, ll mod){
  int s = sz(a)+sz(b)-1, L = get(s), n = 1<<L;

FOR
    (i,sz(b)) v2[i] = cd(b[i] >> 15, b[i] & 32767);
  fft(v1, 0); fft(v2, 0);

  FOR(i,n) {
   int j = (i ? (n - i) : i);
   cd ans1 = (v1[i] + conj(v1[j])) * cd(0.5, 0);
   cd ans2 = (v1[i] - conj(v1[j])) * cd(0,-0.5);
   cd ans3 = (v2[i] + conj(v2[j])) * cd(0.5, 0);
   cd ans4 = (v2[i] - conj(v2[j])) * cd(0,-0.5);
   r1[i] = (ans1 * ans3) + (ans1 * ans4) * cd(0, 1);
   r2[i] = (ans2 * ans3) + (ans2 * ans4) * cd(0, 1);
  }
  fft(r1, 1); fft(r2, 1);
  vl ret(n);
  FOR(i,n) {
   ll av = (ll)round(r1[i].real());
   ll bv = (ll)round(r1[i].imag()) + (ll)round(r2[i].real())
       ;
   ll cv = (ll)round(r2[i].imag());
   av %= mod, bv %= mod, cv %= mod;
   ret[i] = (av << 30) + (bv << 15) + cv;
   ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
  }
  ret.resize(s);
  return ret;
 }
}

using namespace FFTmod;
```

5.3.2 FFT

```
#include <algorithm>
#include <cstdio>
#include <ctime>
#include <vector>
#include <complex>

using namespace std;

typedef complex<double> cd;
typedef vector<cd> vcd;

vcd fft(const vcd &as) {
  int n = as.size();
  int k = 0; // n
  while ((1 << k) < n) k++;
  vector<int> rev(n);
  rev[0] = 0;
  int high1 = -1;
  for (int i = 1; i < n; i++) {
    if ((i & (i - 1)) == 0) // .  i  ,  i-1     .
      high1++;
    rev[i] = rev[i ^ (1 << high1)]; //
    rev[i] |= (1 << (k - high1 - 1)); //
  }

  vcd roots(n);
  for (int i = 0; i < n; i++) {
    double alpha = 2 * M_PI * i / n;
    roots[i] = cd(cos(alpha), sin(alpha));
  }

  vcd cur(n);
  for (int i = 0; i < n; i++)
    cur[i] = as[rev[i]];

  for (int len = 1; len < n; len <<= 1) {
    vcd ncur(n);
    int rstep = roots.size() / (len * 2);
    for (int pdest = 0; pdest < n;) {
      int p1 = pdest;
      for (int i = 0; i < len; i++) {
        cd val = roots[i * rstep] * cur[p1 + len];
        ncur[pdest] = cur[p1] + val;
        ncur[pdest + len] = cur[p1] - val;
        pdest++, p1++;
      }
      pdest += len;
    }
    cur.swap(ncur);
  }
  return cur;
}

vcd fft_rev(const vcd &as) {
  vcd res = fft(as);
  for (int i = 0; i < (int)res.size(); i++) res[i] /= as.
      size();
  reverse(res.begin() + 1, res.end());
  return res;
}

int main() {
  int n;
  scanf("%d", &n);
  vcd as(n);
  for (int i = 0; i < n; i++) {
    int x;
    scanf("%d", &x);
    as[i] = x;
  }

  clock_t stime = clock();
  vcd res = fft(as);
  fprintf(stderr, "%d\n", (int)(clock() - stime));
  for (int i = 0; i < n; i++)
    printf("%.4lf %.4lf\n", res[i].real(), res[i].imag());

  stime = clock();
  vcd as2 = fft_rev(res);
  fprintf(stderr, "%d\n", (int)(clock() - stime));
  for (int i = 0; i < n; i++)
    printf("%.4lf %.4lf\n", as2[i].real(), as2[i].imag());
  return 0;
}
```

5.3.3 FFT2

```
**
* Description:
* Source: KACTL, https://pastebin.com/3Tnj5mRu
* Verification: SPOJ polymul, CSA manhattan
*/

namespace FFT {
 int get(int s) {
  return s > 1 ? 32 - __builtin_clz(s - 1) : 0;
 }

 vcd fft(vcd& a) {
  int n = sz(a), x = get(n);
  vcd res, RES(n), roots(n);
  FOR(i,n) roots[i] = cd(cos(2*M_PIl*i/n),sin(2*M_PIl*i/n));

  res = a;
  FOR(i,1,x+1) {
   int inc = n>>i;
   FOR(j,inc) for (int k = 0; k < n; k += inc){
    int t = 2*k%n+j;
    RES[k+j] = res[t]+roots[k]*res[t+inc];
```

```
  return res;
   }
   swap(res,RES);
  }


}


 vcd fft_rev(vcd& a) {
  vcd res = fft(a);
  FOR(i,sz(res)) res[i] /= sz(a);
  reverse(res.begin() + 1, res.end());
  return res;
 }

 vcd brute(vcd& a, vcd& b) {
  vcd c(sz(a)+sz(b)-1);
  FOR(i,sz(a)) FOR(j,sz(b)) c[i+j] += a[i]*b[j];
  return c;
 }

 vcd conv(vcd a, vcd b) {
  int s = sz(a)+sz(b)-1, L = get(s), n = 1<<L;
  if (s <= 0) return {};
  if (s <= 200) return brute(a,b);
  a.resize(n); a = fft(a);
  b.resize(n); b = fft(b);
  FOR(i,n) a[i] *= b[i];
  a = fft_rev(a);

  a.resize(s);

  return a;
 }

 vl convll(vl a, vl b) {
  vcd A(sz(a)); FOR(i,sz(a)) A[i] = a[i];
  vcd B(sz(b)); FOR(i,sz(b)) B[i] = b[i];
  vcd X = conv(A,B);
  vl x(sz(X)); FOR(i,sz(X)) x[i] =
  round(X[i].real());
  return x;
 }
}
```

6 Misc

6.1 Mo

```
bool comp(query a, query b){
 if (a.L / block == b.L/block)
  return a.R < b.R;
 return a.L/block < b.L/block;
}

void add(int x){
 cnt[x]++;
 if (cnt[x] == 1) distinct++;
}

void del(int x){
 cnt[x]--;
 if (cnt[x] == 0) distinct--;
}

int main(){
  OPTIMATION
  cin >> N;
  for (int i = 0; i < N; i++)
   cin >> arr[i];
  block = (int)sqrt(N) + 1;
  cin >> Q;
  for (int i = 0; i < Q; i++){
   int tl, tr;
   cin >> tl >> tr;
   tl--; tr--;
   q[i].L = tl;
   q[i].R = tr;
   q[i].no = i;
  }
  sort(q, q+Q, comp);
  currL = 0;
  currR = 0;
  for (int i = 0; i < Q; i++){
   int L = q[i].L;
   int R = q[i].R;
   while (currL < L) {
    del(arr[currL]);
    currL++;
   }
   while (currL > L){
    add(arr[currL-1]);
    currL--;
   }
   while (currR <= R) {
    add(arr[currR]);
    currR++;
   }
   while (currR > R+1){
    del(arr[currR-1]);
    currR--;
   }
   ans[q[i].no] = distinct;
  }
  for (int i = 0; i < Q; i++){
   cout << ans[i] << '\n';
  }
  return 0;
}
```

6.2 josephus

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

6.3 mt19937

```
#include <bits/stdc++.h>
using namespace std;
//Application of mt199937

const int N = 3000000;

double average_distance(const vector<int> &permutation) {
    double distance_sum = 0;

    for (int i = 0; i < N; i++)
        distance_sum += abs(permutation[i] - i);
```

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

    return distance_sum / N;
}

int main() {

vector
    <int> permutation(N);

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    shuffle(permutation.begin(), permutation.end(), rng);
    cout << average_distance(permutation) << '\n';

    for (int i = 0; i < N; i++)
        permutation[i] = i;

    for (int i = 1; i < N; i++)
        swap(permutation[i], permutation[
            uniform_int_distribution<int>(0, i)(rng)]);

    cout << average_distance(permutation) << '\n';
}
```

6.4 pbds

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
#include <iostream>
using namespace __gnu_pbds;
using namespace std;

// a new data structure defined. Please refer below
// GNU link : https://goo.gl/WVDL6g
typedef tree<int, null_type, less<int>, rb_tree_tag,
            tree_order_statistics_node_update>
    new_data_set;

// Driver code
int main()
{
    new_data_set p;
    p.insert(5);
    p.insert(2);
    p.insert(6);
    p.insert(4);

    // value at 3rd index in sorted array.
    cout << "The value at 3rd index ::"
         << *p.find_by_order(3) << endl;

    // index of number 6
    cout << "The index of number 6::"
         << p.order_of_key(6) << endl;

    // number 7 not in the set but it will show the
    // index number if it was there in sorted array.
    cout << "The index of number seven ::"
         << p.order_of_key(7) << endl;

    return 0;
}
```

7 Setup

7.1 C++Template

```
#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")

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

#define fi first
#define se second
#define pb push_back

typedef long long LL;
typedef vector<int> vi;
typedef pair<int,int> ii;

const int MOD = 1e9 + 7;
const LL INF = 1e18;

void fastscan(int &number) {
    //variable to indicate sign of input number
    bool negative = false;
    register int c;
    number = 0;
    c = getchar();
    if (c=='-') {
        negative = true;
        c = getchar();
    }
    for (; (c>47 && c<58); c=getchar())
        number = number *10 + c - 48;
    if (negative)
        number *= -1;
}

/**
* Description: Custom comparator for map / set
* Source: StackOverflow
* Verification: ?
*/
struct cmp {
 bool operator()(const int& l, const int& r) const {
  return l > r;
 }
};
set<int,cmp> s;

int main(){
    //ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0)

    return 0;
}
```

7.2 FastScanner

```
class FastScanner {
    private InputStream stream;
    private byte[] buf = new byte[1024];
    private int curChar;
    private int numChars;

    public FastScanner(InputStream stream) {
        this.stream = stream;
    }

    int read() {
        if (numChars == -1)
            throw new InputMismatchException();
        if (curChar >= numChars) {
            curChar = 0;
            try {
                numChars = stream.read(buf);
            } catch (IOException e) {
                throw new InputMismatchException();
            }
            if (numChars <= 0) return -1;
        }
```

```
    boolean isSpaceChar(int c) {
        return buf[curChar++];
    }


return
     c' == ' || c' == \'n || c' == \'r || c'== \'t || c
              == -1;
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }

    public long nextLong() {
        return Long.parseLong(next());
    }

    public double nextDouble() {
        return Double.parseDouble(next());
    }

    public String next() {
        int c = read();
        while (isSpaceChar(c)) c = read();
        StringBuilder res = new StringBuilder();
        do {
            res.appendCodePoint(c);
            c = read();
        } while (!isSpaceChar(c));
        return res.toString();
    }

    public String nextLine() {
        int c = read();
        while (isEndline(c))
            c = read();
        StringBuilder res = new StringBuilder();
        do {
            res.appendCodePoint(c);
            c = read();
        } while (!isEndline(c));
        return res.toString();
    }
}
```

7.3 sublimesetup

```
{
 "cmd": ["g++", "-std=c++11", "$file_name", "-o", "${
     file_base_name}.exe", "&&", "start", "cmd", "/k" , "
     $file_base_name"],
 "selector": "source.cpp",
 "file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
 "working_dir": "${file_path}",
 "shell": true
}
```

8 String

8.1 Hashing

```
/*use double hashing */

long long compute_hash(string const& s) {
    const int p = 31; //another good option : p = 53
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m
            ;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

8.2 KMP

```
#define HHH 10003

int ne[HHH]; // next[], if par[i] not matched, jump to i =
     ne[i]
int kmp(string& par, string& ori) {
    ne[0] = -1;
    for (int p = ne[0], i = 1; i < par.length(); i++) {
        while (p >= 0 && par[p+1] != par[i])
            p = ne[p];
        if (par[p+1] == par[i])
            p++;
        ne[i] = p;
    }

    int match = 0;
    for (int p = -1, q = 0; q < ori.length(); q++) {
        while (p >= 0 && par[p+1] != ori[q])
            p = ne[p];
        if (par[p+1] == ori[q])
            p++;
        if (p + 1 == par.length()) { // match!
            p = ne[p];
            match++;
        }
    }

    return match; // return number of occurance
}

int main () {
    int n; cin >> n;
    string par, ori;
    while (cin >> par >> ori)
        cout << kmp(par, ori) << endl;
    return 0;
}
```

8.3 Manacher

```
int dp[HHH];
int lengthLongestPalindromSubstring(string& s) {
    memset(dp, 0, sizeof(dp));
    int ans = 0;
    int pivot = 1;
    int len = s.length() * 2; // _s0_s1_s2 = 2 * length
    for (int i = 1; i < len; i++) {
        int pBorder = pivot + dp[pivot];
        int iBorder = i;
        if (iBorder < pBorder && 2 * pivot - i > 0) {
            dp[i] = dp[2*pivot-i];
            iBorder = min(pBorder, i + dp[i]);
        }

        if (iBorder >= pBorder) {
            int j = iBorder + (iBorder % 2 ? 2 : 1);
            for (; j < len && 2*i-j > 0 && s[j/2] == s[(2*i-j
                 )/2]; j += 2)
                ;
            iBorder = j - 2;
            dp[i] = iBorder - i;
            pivot = i;
        }
        ans = max(ans, dp[i] + 1);
    }
```

```
    int n; cin >> n;
    return ans;
}

int main () {

string
     s;
    while (cin >> s)
        cout << lengthLongestPalindromSubstring(s) << endl;
    return 0;
}
```

8.4 rabinkarp

```
/* Problem: Given two strings - a pattern s and a text t,
determine if the pattern appears in the text and if it does,
enumerate all its occurrences in O(|s|+|t|) time.*/
vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

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

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

8.5 zAlgorithm

```
string s; cin >> s;
int L = 0, R = 0;
int n = s.size();
for (int i = 1; i < n; ++i) {
    if (i > R) {
        L = R = i;
        while (R < n && s[R] == s[R-L]) ++R;
        Z[i] = R-L; --R;
    }
    else {
        int k = i-L;
        if (Z[k] < R-i+1) Z[i] = Z[k];
        else {
            L = i;
            while (R < n && s[R] == s[R-L]) ++R;
            Z[i] = R-L; --R;
        }
    }
}
```