

Contents	23 歐拉定理	23
1 .vimrc	1 24 路卡斯公式	23
2 AC Actomaton	1 25 模數合成	23
3 Combinatoion	2 26 強國人說的歐拉定理	23
4 Decomposition	3 27 無權邊的生成樹個數 Kirchhoff's Theorem	23
5 Double LCA	4 28 很大的質數	23
6 Flow (Dinics)	4 29 GP 東北數學式	24
7 Geometry	5 30 圓周率 1000 位	24
8 Simple Tabulation Hash	10 31 尤拉數 e	24
9 IDA*	11 32 歐拉示性數	24
10 inverse	11 33 半平面交相關幾何轉換	24
11 Karatsuba (FFFT)	12	
12 KM	12	
13 Linear Prime	13	
14 Mod Combine	14	
15 Range Tree 2D, kth number	14	
16 Range Tree 2D, rectangle	15	
17 Scan (JAVA)	17	
18 Segment Tree	18	
19 Splay Tree	18	
20 Suffix Array	20	
21 Treap	21	
22 Z Algorithm	23	

1 .vimrc

```
set nu
set sw=4
set ts=4
set st=4
set bs=2
set cul
set ai
set ls=2
map <F5> gT
imap <F5> <ESC>gT
map <F6> gt
imap <F6> <ESC>gt
imap {<CR> {<CR><END><CR>}<UP><END>
au FileType cpp map <F9> <ESC>:w<CR>:!g++<Space>-Wall<Space>%&&./a.out
<CR>
au FileType cpp imap <F9> <ESC>:w<CR>:!g++<Space>-Wall<Space>%&&./a.
out<CR>
set encoding=UTF-8
```

2 AC Actomaton

```
#include <iostream>
#include <queue>
#include <cstring>
#include <cstdio>

using namespace std;

struct AC_Automaton {
    static const int MAX_N = 1e6+10;
    static const int MAX_CHILD = 52;

    int n;
    int fail[MAX_N];
    int trie[MAX_N][MAX_CHILD];

    void clean(int target) {
        for (int i = 0; i < MAX_CHILD; ++i) {
            trie[target][i] = -1;
        }
    }
}
```

```
}

void reset () {
    clean(0);
    n = 1;
}

void add(char* s) {
    int p = 0;
    while (*s) {
        int id = get_id(s[0]);
        if (trie[p][id] == -1) {
            clean(n);
            trie[p][id] = n++;
        }
        p = trie[p][id];
        ++s;
    }
}

void construct() {
    queue<int> que;
    fail[0] = 0;

    for (int i = 0; i < MAX_CHILD; ++i) {
        if (trie[0][i] != -1) {
            fail[trie[0][i]] = 0;
            que.push(trie[0][i]);
        }
        else {
            trie[0][i] = 0;
        }
    }

    while (que.size()) {
        int now = que.front();
        que.pop();

        for (int i = 0; i < MAX_CHILD; ++i) {
            int target = trie[now][i];
            if (target != -1) {
                que.push(target);
                fail[target] = trie[fail[now]][i];
            }
            else {

```

```

        trie[now][i] = trie[fail[now]][i];
    }
}
}

int solve() {
    int ans = fail[n-1];
    while (ans > n/2-1) ans = fail[ans];
    return ans;
}

int get_id(const char& ch) {
    if (ch <= 'z' && ch >= 'a') return ch-'a';
    else return ch-'A'+26;
}
} ac;

char input[1000010];

int main () {
    int tcase;
    scanf("%d", &tcase);
    while (tcase--) {
        ac.reset();
        scanf("%s", input);
        ac.add(input);
        ac.construct();
        printf("%d\n", ac.solve());
    }
}

```

3 Combinatoion

```

const long long MOD = 1e9+7;
const int MAX = 1e5+1;

typedef long long T;
T inverse(T mod, T b) { /* return b-1 mod a */
    T k[2][2], n[2][2], u1, u2;

    k[0][0] = k[1][1] = 1;

```

```

    k[0][1] = k[1][0] = 0;

    u1 = mod, u2 = b;

    while (u2) {
        T div = u1/u2;
        T remind = u1%u2;

        n[0][0] = k[1][0];
        n[0][1] = k[1][1];
        n[1][0] = k[0][0] - k[1][0]*div;
        n[1][1] = k[0][1] - k[1][1]*div;

        for (T i = 0; i < 2; ++i) {
            for (T j = 0; j < 2; ++j) {
                k[i][j] = n[i][j];
            }
        }
        u1 = u2;
        u2 = remind;
    }

    if (k[0][1] < 0) k[0][1] += mod;
    return k[0][1];
}

T C(T n, T m, T mod) {
    if (m < 0) return 0;
    if (n < m) return 0;
    T ans = 1;
    T base = min(n-m, m);

    for (T i = 0; i < base; ++i) {
        ans = ans*(n-i)%mod;
    }

    T inv = 1;
    for (T i = 1; i <= base; ++i) {
        inv = inv*i%mod;
    }
    return ans*inverse(mod, inv)%mod;
}

```

4 Decomposition

```
static class Decomposition{

    Map<BigInteger, Integer> prime;
    Random random;

    Decomposition(String x){
        prime = new HashMap<>();
        random = new Random();
        BigInteger in = new BigInteger(x);
        int twos = 0;
        while(!in.testBit(0)){
            in = in.shiftRight(1);
            twos++;
        }
        if(twos > 0) prime.put(BigInteger.valueOf(2), twos);
        peel(in);
    }

    void peel(BigInteger x){
        System.out.println("peel "+x);
        if(x.equals(BigInteger.ONE)) return;
        if(x.isProbablePrime(100)){
            Integer temp = prime.put(x, 1);
            if(temp!=null) prime.put(x, temp+1);
            return;
        }
        BigInteger a, b, c, next;
        do{
            a = b = new BigInteger(x.bitLength()+5, random).mod(x);
            c = new BigInteger(x.bitLength()+5, random).mod(x);
            if(c.equals(BigInteger.ZERO)) c = BigInteger.ONE;
            do{
                a = f(a, c, x);
                b = f(f(b, c, x), c, x);
                next = x.gcd(a.subtract(b).abs());
            }while(next.equals(BigInteger.ONE));
        }while(next.equals(x));
        peel(next);
        peel(x.divide(next));
    }

    BigInteger f(BigInteger x, BigInteger c, BigInteger n){
        return x.multiply(x).add(c).mod(n);
    }
}
```

```
}
}
```

5 Double LCA

```
/* build: O(VlogV), query: O(logV) */
#include <iostream>
#include <vector>
#include <cstdio>
#define MAX 50010

using namespace std;

int a[MAX][160]; /* 160 = log2(MAX/2) */
int parent[MAX], tin[MAX], tout[MAX];
int num, root, timestamp;
bool visit[MAX];
vector<int> adj[MAX];

int log2(int n) {
    int i = 0;
    while ((1<<i) <= n) ++i;
    return i - 1;
}

/* when x == y, it's be true */
bool ancestor(int x, int y) {
    return (tin[x] <= tin[y]) && (tout[x] >= tout[y]);
}

void dfs(int x, int px) {
    tin[x] = timestamp++;
    visit[x] = true;
    a[x][0] = px;
    for (int i = 1; i < log2(num); ++i) {
        a[x][i] = a[a[x][i-1]][i-1];
    }

    for (int i = 0; i < adj[x].size(); ++i) {
        int target = adj[x][i];
        if (!visit[target]) {
```

```

        parent[target] = x;
        dfs(target, x);
    }
}
tout[x] = timestamp++;
}

int lca(int x, int y) {
    if (ancestor(x, y)) return x;
    if (ancestor(y, x)) return y;

    for (int i = log2(num); i >= 0; --i) {
        if (!ancestor(a[x][i], y)) {
            x = a[x][i];
        }
    }
    return a[x][0];
}

int main () {
    timestamp = 0;

    /* init */
    for (int i = 0; i < num; ++i) {
        parent[i] = i;
        visit[i] = false;
        adj[i].clear();
    }

    for (int i = 0; i < num-1; ++i) {
        int x, y;
        scanf("%d%d", &x, &y);
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    dfs(0, 0);
    cin >> x >> y;
    cout << lca(x, y);
}

```

6 Flow (Dinics)

```

import java.io.*;
import java.util.*;

public class Main{

    static ArrayList<ArrayList<Edge>> list;
    static Edge[][] matrix;
    static int start, finish;

    static int findFlow(){
        int[] height = new int[list.size()];
        Arrays.fill(height, -1);
        Queue<Integer> queue = new ArrayDeque<Integer>();
        height[start] = 0;
        queue.add(start);
        while(!queue.isEmpty()){
            int now = queue.poll();
            for(Edge e : list.get(now)){
                int next = e.v;
                if(e.cap == 0) continue;
                if(height[next] != -1) continue;
                height[next] = height[now]+1;
                queue.add(next);
            }
        }
        if(height[finish] == -1) return 0;
        int result = 0, flow;
        while((flow = trace(start, Integer.MAX_VALUE, height)) != 0)
            result += flow;
        return result;
    }

    static int trace(int now, int flow, int[] height){
        if(now == finish){
            return flow;
        }
        int result = 0;
        for(Edge e : list.get(now)){
            if(e.cap == 0) continue;
            int next = e.v;
            if(height[now]+1 != height[next]) continue;
            result = trace(next, Math.min(flow, e.cap), height);
            if(result != 0){
                matrix[now][next].cap -= result;
            }
        }
    }
}

```

```

        matrix[next][now].cap += result;
        break;
    }
}
return result;
}

static class Edge{
    int u, v, cap;

    public Edge(int u, int v, int cap, Edge[][] matrix){
        this.u = u;
        this.v = v;
        this.cap = cap;
        matrix[u][v] = this;
    }
}
}

```

7 Geometry

```

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

#define EPS 1e-10
#define LEFT_TOP POS(1000, 1000)
#define NO_INTERSECT POS(-1234, -1234)
#define PARALLEL POS(-1001, -1001)
#define COLINE POS(1234, 1234)
const double PI = acos(-1.0);

typedef double T;

class POS {
public:
    T x, y;
    POS(const T& x = 0, const T& y = 0) : x(x), y(y) {}
    POS(const POS& x) : x(x.x), y(x.y) {}

    bool operator==(const POS& rhs) const {
        return x == rhs.x && y == rhs.y;
    }
}

```

```

}

POS& operator+=(const POS& rhs) {
    x += rhs.x;
    y += rhs.y;
    return *this;
}

POS operator -(const POS& rhs) {
    POS tmp(-x, -y);
    return tmp;
}

double dist(const POS& rhs) const {
    T tmp_x = x-rhs.x, tmp_y = y-rhs.y;
    return sqrt(tmp_x*tmp_x+tmp_y*tmp_y);
}

friend ostream& operator<<(ostream& out, const POS& pos) {
    out << pos.x << " " << pos.y;
    return out;
}
};

POS const operator+(const POS& lhs, const POS& rhs) {
    return POS(lhs) += rhs;
}

POS const operator-(const POS& lhs, const POS& rhs) {
    POS tmp = rhs;
    tmp = -tmp;
    return POS(lhs) += (tmp);
}

bool cmp_convex(const POS& lhs, const POS& rhs) {
    return (lhs.x < rhs.x) || ( (lhs.x == rhs.x) && (lhs.y < rhs.y) );
}

inline T cross(const POS& o, const POS& a, const POS& b) {
    double value = (a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x);
    if (fabs(value) < EPS) return 0;
    return value;
}

void convex_hull(POS* points, POS* need, int& n) {

```

```

sort(points, points+n, cmp_convex);
int index = 0;
for (int i = 0; i < n; ++i) {
    while (index >= 2 && cross(need[index-2], need[index-1],
points[i]) <= 0) index--;
    need[index++] = points[i];
}
int half_point = index+1;
for (int i = n-2; i >= 0; --i) {
    while (index >= half_point && cross(need[index-2], need[index
-1], points[i]) <= 0) index--;
    need[index++] = points[i];
} /* be careful that start point will appear in first and last in
need array */
n = index;
}

class LINE {
public:
    POS start, end, vec;
    double angle;
    LINE() {}
    LINE(const T& st_x, const T& st_y, const T& ed_x, const T& ed_y) :
        start(st_x, st_y), end(ed_x, ed_y), vec(end - start), angle(
atan2(vec.x, vec.y)) {}

    LINE(const POS& start, const POS& end) :
        start(start), end(end), vec(end - start), angle(atan2(vec.x,
vec.y)) {}

    LINE(const POS& end) : /* start point is origin */
        start(0, 0), end(end), vec(end), angle(atan2(vec.x, vec.y)) {}

    LINE(const T a, const T b, const T c) : /* given line by ax+by+c =
0 */
        start(0, 0), end(0, 0), vec(-b, a) {
        if (a == 0) {
            start.y = end.y = -c/b;
            end.x = -b;
        }
        else if (b == 0) {
            start.x = end.x = -c/a;
            end.y = a;
        }
        else if (c == 0) {

```

```

            end.x = -b; end.y = a;
        }
        else {
            start.y = -c/b; end.x = -c/a;
            vec.x = -c/a; vec.y = c/b;
        }
        angle = atan2(vec.x, vec.y);
    }

    LINE build_orthogonal(const POS& point) const {
        T c = -(vec.x*point.x + vec.y*point.y);
        return LINE(vec.x, vec.y, c);
    }

    T length2() const { /* square */
        T x = start.x - end.x, y = start.y - end.y;
        return x*x + y*y;
    }

    void modify(T x, T y) {
        this->end.x += x;
        this->end.y += y;
        this->vec.x += x;
        this->vec.y += y;
    }

    bool on_line(const POS& a) const {
        if (vec.x == 0) {
            if (start.x != a.x) return false;
            return true;
        }
        if (vec.y == 0) {
            if (start.y != a.y) return false;
            return true;
        }
        return fabs((a.x-start.x)/vec.x*vec.y + start.y) - a.y <
EPS;
    }

    bool operator/(const LINE& rhs) const { /* to see if this line
parallel to LINE rhs */
        return (vec.x*rhs.vec.y == vec.y*rhs.vec.x);
    }

    bool operator==(const LINE& rhs) const { /* to see if they are

```

```

same line */
    return (*this/rhs) && (rhs.on_line(start));
}

POS intersect(const LINE& rhs) const {
    if (*this==rhs) return COLINE; /* return co-line */
    if (*this/rhs) return PARALLEL; /* return parallel */

    double A1 = vec.y, B1 = -vec.x, C1 = end.x*start.y - start.x*
end.y;
    double A2 = rhs.vec.y, B2 = -rhs.vec.x, C2 = rhs.end.x*rhs.
start.y - rhs.start.x*rhs.end.y;
    return POS( (B2*C1-B1*C2)/(A2*B1-A1*B2), (A1*C2-A2*C1)/(A2*B1-
A1*B2) ); /* sometimes has -0 */
}

double dist(const POS& a) const {
    return fabs(vec.y*a.x - vec.x*a.y + vec.x*start.y - vec.y*
start.x)/sqrt(vec.y*vec.y+vec.x*vec.x);
}

double dist(const LINE& rhs) const {
    POS intersect_point = intersect(rhs);
    if (intersect_point == PARALLEL) {
        return dist(rhs.start);
    }
    return 0;
}

friend ostream& operator<<(ostream& out, const LINE& line) {
    out << line.start << "-->" << line.end << " vec: " << line.vec
;
    return out;
}
};

class LINESEG : public LINE {
public:
    LINESEG() : LINE(POS(0, 0)) {}
    LINESEG(const LINE& input) : LINE(input) {}
    LINESEG(const POS& start, const POS& end) : LINE(start, end) {}

    bool on_lineseg(const POS& a) const {
        if (!on_line(a)) return false;
        bool first, second;

```

```

        if (vec.x >= 0) first = (a.x >= start.x)&&(a.x <= end.x);
        else first = (a.x <= start.x)&&(a.x >= end.x);
        if (vec.y >= 0) second = (a.y >= start.y)&&(a.y <= end.y);
        else second = (a.y <= start.y)&&(a.y >= end.y);
        return first&&second;
    }

    bool operator==(const LINESEG& rhs) const {
        return ( (rhs.start == start && rhs.end == end) ||
                (rhs.start == end && rhs.end == start) );
    }

    bool operator==(const LINE& rhs) const {
        return this->LINE::operator==(rhs);
    }

    T dot(const LINESEG& rhs) const {
        return vec.x*rhs.vec.x + vec.y*rhs.vec.y;
    }

    T cross(const LINESEG& rhs) const {
        return vec.x*rhs.vec.y - vec.y*rhs.vec.x;
    }

    bool clockwise(const LINE& a) const { /* to see if LINE a is in b'
s clockwise way */
        return cross(a) > 0;
    }

    double dist(const POS& a) const {
        double ortho_dist = this->LINE::dist(a);
        LINE ortho_line = build_orthogonal(a);
        POS intersect_point = this->LINE::intersect(ortho_line);
        if (on_lineseg(intersect_point)) return ortho_dist;
        else return min(a.dist(this->start), a.dist(this->end));
    }

    double dist(const LINE& line) const {
        POS intersect_point = this->LINE::intersect(line);
        if (intersect_point == COLINE) return 0;
        if (intersect_point == PARALLEL) return dist(line.start);
        if (on_lineseg(intersect_point)) return 0;
        return min(line.dist(start), line.dist(end));
    }
}

```



```

double dist(const LINESEG& line) const {
    return min( min(dist(line.start), dist(line.end)),
                min(line.dist(start), line.dist(end)) );
}

POS intersect(const LINESEG& rhs) const {
    LINE a1b1(start, rhs.start);
    LINE a1b2(start, rhs.end);
    LINE b1a1(rhs.start, start);
    LINE b1a2(rhs.start, end);

    POS tmp(this->LINE::intersect(rhs));

    if (tmp == COLINE) {
        if ( (start==rhs.start) && (!rhs.on_lineseg(end)) && (!
on_lineseg(rhs.end)) ) return start;
        if ( (start==rhs.end) && (!rhs.on_lineseg(end)) && (!
on_lineseg(rhs.start)) ) return start;
        if ( (end==rhs.start) && (!rhs.on_lineseg(start)) && (!
on_lineseg(rhs.end)) ) return end;
        if ( (end==rhs.end) && (!rhs.on_lineseg(start)) && (!
on_lineseg(rhs.start)) ) return end;
        if (on_lineseg(rhs.start) || on_lineseg(rhs.end) || rhs.
on_lineseg(start) || rhs.on_lineseg(end)) return COLINE;
        return NO_INTERSECT;
    }

    bool intersected = ( (cross(a1b1)*cross(a1b2)<0) && (rhs.
cross(b1a1)*rhs.cross(b1a2)<0) );
    if (!intersected) return NO_INTERSECT;
    if (!on_lineseg(tmp) || !rhs.on_lineseg(tmp)) return
NO_INTERSECT;
    return tmp;
}

};

inline bool cmp_half_plane(const LINE &a, const LINE &b){
    if(fabs(a.angle-b.angle) < EPS) return cross(a.start, a.end, b.
start) < 0;
    return a.angle > b.angle;
}

void half_plane_intersection(LINE* a, LINE* need, POS* answer, int &n)
{
    int m = 1, front = 0, rear = 1;

```

```

sort(a, a+n, cmp_half_plane);
for(int i = 1; i < n; ++i){
    if( fabs(a[i].angle-a[m-1].angle) > EPS ) a[m++] = a[i];
}
need[0] = a[0], need[1] = a[1];
for(int i = 2; i < m; ++i){
    while (front<rear&&cross(a[i].start, a[i].end, need[rear].
intersect(need[rear-1]))<0) rear--;
    while (front<rear&&cross(a[i].start, a[i].end, need[front].
intersect(need[front+1]))<0) front++;
    need[++rear] = a[i];
}
while (front<rear&&cross(need[front].start,need[front].end, need[
rear].intersect(need[rear-1]))<0) rear--;
while (front<rear&&cross(need[rear].start,need[rear].end, need[
front].intersect(need[front+1]))<0) front++;
if (front==rear) return;

n = 0;
for (int i=front; i<rear; ++i) answer[n++] = need[i].intersect(
need[i+1]);
if(rear>front+1) answer[n++] = need[front].intersect(need[rear]);
}

void rotating_calipers(int& ans, POS* need, int& n) {
    --n;
    if (n == 2) {
        ans = need[0].dist(need[1]);
        return;
    }

    int now = 2;
    for (int i = 0; i < n; ++i) {
        LINE target(need[i], need[i+1]);
        double pre = target.dist(need[now]);
        for (; now != i; now = (now+1)%n) {
            double tmp = target.dist(need[now]);
            if (tmp < pre) break;
            pre = tmp;
        }
        now = (now-1+n)%n;
        ans = max(ans, max(need[i].dist(need[now]), need[i+1].dist(
need[now]))));
    }
}

```

```

class POLYGON {
public:
    vector<POS> point;
    vector<LINESEG> line;

    void add_points(const POS& x) {
        point.push_back(x);
    }

    void add_points(const int& x, const int& y) {
        point.push_back(POS(x,y));
    }

    void build_line() {
        if (line.size() != 0) return; /* if it has build */
        for (int i = 1; i < point.size(); ++i) {
            line.push_back(LINESEG(point[i], point[i-1]));
        }
        line.push_back(LINESEG(point[0], point[point.size()-1]));
    }

    double area() {
        double ans = 0;

        vector<LINESEG> tmp;
        for (int i = 0; i < point.size(); ++i) {
            tmp.push_back(LINESEG(point[i]));
        }
        tmp.push_back(LINESEG(point[0]));

        for (int i = 1; i < tmp.size(); ++i) {
            ans += tmp[i-1].cross(tmp[i]);
        }
        return 0.5*fabs(ans);
    }

    bool in_polygon(const POS& a, const POS& left_top = LEFT_TOP) {

        for (int i = 0; i < point.size(); ++i) {
            if (a == point[i]) return true; /* a is polygon's point */
        }

        build_line();
        for (int i = 0; i < line.size(); ++i) {
            if (line[i].on_line(a)) {

```

```

                return true; /* a is on polygon's line */
            }
        }

        POS endpoint(left_top); /* should be modified according to
        problem */
        LINESEG ray(a, endpoint);
        bool touch_endpoint = false;
        do {
            touch_endpoint = false;
            for (int i = 0; i < point.size(); ++i) {
                if (ray.on_lineseg(point[i])) {
                    touch_endpoint = true;
                    break;
                }
            }
            if (touch_endpoint) ray.modify(-1, 0); /* should be
            modified according to problem */
        } while (touch_endpoint);

        int times = 0;
        for (int i = 0; i < line.size(); ++i) {
            POS tmp(ray.intersect(line[i]));
            if (tmp == NO_INTERSECT || tmp == PARALLEL) {
                continue;
            }
            ++times;
        }
        return (times&1);
    }
};

int main() {
    return 0;
}

```

8 Simple Tabulation Hash

```

import java.util.*;

class HashTable{

```

```

long[] key;
Main.Entry[] content;
SimpleTabulationHash hash;

HashTable(long universeSize, int sizeBit){
    key = new long[1<<sizeBit];
    content = new Main.Entry[1<<sizeBit];
    Arrays.fill(key, -1);
    hash = new SimpleTabulationHash(universeSize, sizeBit);
}

//returns index if found, -1 if not
int containsKey(long x){
    int hashValue = hash.hashCode(x);
    for(int i=hashValue;;i++){
        if(i == key.length) i = 0;
        if(key[i] == -1) return -1;
        if(key[i] == x) return i;
    }
}

void put(long x, Main.Entry entry){
    int hashValue = hash.hashCode(x);
    for(int i=hashValue;;i++){
        if(i == key.length) i = 0;
        if(key[i] == -1){
            key[i] = x;
            content[i] = entry;
            return;
        }
    }
}

Main.Entry get(long x){
    return content[contains(x)];
}

}

class SimpleTabulationHash{

    final static int bit = 16, mask = (1<<bit)-1;
    int C;
    int[][] table;

```

```

SimpleTabulationHash(long universeSize, int tableBit){ // table
size is givin in 2^n
    C = 0;
    while(universeSize > 0){
        universeSize >>= bit;
        C++;
    }
    table = new int[C][mask+1];
    // System.err.println("C = "+C);
    Random random = new Random();
    int cutmask = (1<<tableBit)-1;
    //System.err.println("tablebit: "+tableBit+", cutmask : "+
cutmask);
    for(int i=0;i<C;i++){
        for(int j=0;j<=mask;j++) table[i][j] = random.nextInt()&
cutmask;
    }
}

int hashCode(long x){
    int result = 0;
    for(int i=0;i<C;i++){
        result ^= table[i][(int)(x&mask)];
        x >>= bit;
    }
    return result;
}

}

```

9 IDA*

```

int search(STATE& now, int g, int bound) {
    int f = g + now.heuri;
    if (f > bound) return f;
    if (is_goal(now)) return FOUND;

    int min = INF;
    for next in successors(now):
        int t = search(state, g+cost(now,next), bound);
        if (t == FOUND) return FOUND;
        if (t < min) min = t;
}

```

```

    }
    return min;
}

void IDAStar() {
    STATE init(input);
    int bound = init.heuri;
    while (bound <= MAXI) {
        int t = search(init, 0, bound);
        if (t == FOUND) return FOUND;
        if (t == INF) return NOT_FOUND;
        bound = t;
    }
}

```

10 inverse

```

long long inverse(long long b, long long mod=MOD) {
    long long k[2][2], n[2][2], u1, u2;

    k[0][0] = k[1][1] = 1;
    k[0][1] = k[1][0] = 0;

    u1 = mod, u2 = b;

    while (u2) {
        long long div = u1/u2;
        long long remind = u1%u2;

        n[0][0] = k[1][0];
        n[0][1] = k[1][1];
        n[1][0] = k[0][0] - k[1][0]*div;
        n[1][1] = k[0][1] - k[1][1]*div;

        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                k[i][j] = n[i][j];
            }
        }

        u1 = u2;
        u2 = remind;
    }
}

```

```

    }
    while (k[0][1] < 0) k[0][1] += mod;

    if(((k[0][1]*(b%mod))%mod+mod)%mod !=1ll) printf("%lld^-1 doesn't exist under mod %lld\n",b,mod);

    return k[0][1];
}

```

11 Karatsuba (FFT)

```

static class Karatsuba{

    int maxHeight;
    long[][][] buffer; //h1, l1, m1, h2, l2, m2, hh, ll, mm

    Karatsuba(int maxHeight){
        this.maxHeight = maxHeight;
        buffer = new long[maxHeight][9][];
        for(int i=6;i<maxHeight;i++){
            for(int j=0;j<6;j++){ buffer[i][j] = new long[(1<<i)>>1];
                for(int j=6;j<9;j++) buffer[i][j] = new long[1<<i];
            }
        }

        void multiply(long[] a, long[] b, long[] result, int depth){
            int size = 1<<depth, mid = size>>1;
            if(depth <= 5){
                Arrays.fill(result, 0);
                for(int i=0;i<a.length;i++){
                    for(int j=0;j<b.length;j++) result[i+j] += a[i]*b[j];
                }
                return;
            }
            for(int i=0;i<mid;i++){
                buffer[depth][0][i] = a[i+mid];
                buffer[depth][1][i] = a[i];
                buffer[depth][2][i] = a[i+mid] + a[i];
                buffer[depth][3][i] = b[i+mid];
                buffer[depth][4][i] = b[i];
                buffer[depth][5][i] = b[i+mid] + b[i];
            }
        }
    }
}

```

```

        multiply(buffer[depth][0], buffer[depth][3], buffer[depth][6],
depth-1);
        multiply(buffer[depth][1], buffer[depth][4], buffer[depth][7],
depth-1);
        multiply(buffer[depth][2], buffer[depth][5], buffer[depth][8],
depth-1);
        Arrays.fill(result, 0);
        for(int i=0;i<size;i++){
            result[i+size] += buffer[depth][6][i];
            result[i] += buffer[depth][7][i];
            result[i+mid] += buffer[depth][8][i] - buffer[depth][6][i]
- buffer[depth][7][i];
        }
    }
}

```

12 KM

```

#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#define MAX 404
#define INF 0x7fffffff

using namespace std;

int num; // total num of node
int path[MAX][MAX];
bool visit_x[MAX], visit_y[MAX];
int parent[MAX], weight_x[MAX], weight_y[MAX];

bool find(int i) {
    visit_x[i] = true;
    for (int j = 0; j < num; ++j) {
        if (visit_y[j]) continue;
        if (weight_x[i] + weight_y[j] == path[i][j]) {
            visit_y[j] = true;
            if (parent[j] == -1 || find(parent[j])) {
                parent[j] = i;
                return true;
            }
        }
    }
}

```

```

    }
}
return false;
}

int weighted_hungarian() {

    /* remember to initial weight_x (max weight of node's edge)*/
    /* initialize */
    for (int i = 0; i < num; ++i) {
        weight_y[i] = 0;
        parent[i] = -1;
    }

    for (int i = 0; i < num; ++i) {
        while (1) {
            memset(visit_x, false, sizeof(visit_x));
            memset(visit_y, false, sizeof(visit_y));
            if (find(i)) break;

            int lack = INF;
            for (int j = 0; j < num; ++j) {
                if (visit_x[j]) {
                    for (int k = 0; k < num; ++k) {
                        if (!visit_y[k]) {
                            lack = min(lack, weight_x[j] + weight_y[k]
- path[j][k]);
                        }
                    }
                }
            }
            if (lack == INF) break;
            // renew label
            for (int j = 0; j < num; ++j) {
                if (visit_x[j]) weight_x[j] -= lack;
                if (visit_y[j]) weight_y[j] += lack;
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < num; ++i) {
        ans += weight_x[i];
    }
}

```

```

        ans += weight_y[i];
    }
    return ans;
}

```

13 Linear Prime

```

#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;
#define N (100000000+5)

bool killed[N]={0};
int kill[N]={0};
int prime[N];
long long numOfPrime=0;

void makeTable(){
    long long limit;
    for(long long i=2;i<N;i++){
        if(kill[i]==0){
            prime[numOfPrime++] = i;
            limit = i;
        }
        else{
            limit = kill[i];
        }
        for(int j=0;j<numOfPrime;j++){
            long long get = prime[j];
            if(get>limit||get*i>=N) break;
            kill[get*i] = get;
        }
    }
}

int main()
{
    makeTable();
    int num=0;
    printf("%d\n",prime[numOfPrime-1]);
}

```

```

    return 0;
}

```

14 Mod Combine

```

int modCombine(int x,int a,int y,int b){//ans mod x = a,ans mod y =b;

    int ans = x * (x^(-1))(mod(y)) * b + y * (y^(-1))(mod(x)) * a;
    ans %=(x*y);
    return ans;
}

```

15 Range Tree 2D, kth number

```

#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

struct COORDINATE {
    int x, y;
};

bool cmp(const COORDINATE& x, const COORDINATE& y) {
    return x.x < y.x;
}

/* x: data, y: index */
struct RangeTree2D {
    COORDINATE **container;
    bool **is_left;
    int **left, **right, *input, length, rank, capacity;
    void init(int *input, int length) {
        this->input = input;
        this->length = length;
        rank = 1;
        while ( (1<<rank++) < length );
        capacity = 1<<(rank-1);
    }
}

```

```

    container = new COORDINATE*[rank], left = new int*[rank],
    right = new int*[rank];
    is_left = new bool*[rank];
    for (int i = 0; i < rank; ++i) {
        container[i] = new COORDINATE[capacity];
        left[i] = new int[capacity];
        right[i] = new int[capacity];
        is_left[i] = new bool[capacity];
    }
    for (int i = 0; i < capacity; ++i) {
        container[0][i].x = i>=length?0:input[i];
        container[0][i].y = i;
    }
    sort(container[0], container[0]+length, cmp);
    build(rank-1, 0, capacity-1);
}

void build(int height, int start, int finish) {
    if (height == 0) return;
    if (start == finish) {
        build(height-1, start, finish);
        container[height][start] = container[height-1][start];
        return;
    }

    int middle = start+(1<<(height-1));
    build(height-1, start, middle-1);
    build(height-1, middle, finish);
    int now = start, l_index = start, r_index = middle;

    while (now <= finish) {
        left[height][now] = l_index;
        right[height][now] = r_index;

        if (l_index < middle && (r_index > finish || container[
height-1][l_index].y <= container[height-1][r_index].y)) {
            container[height][now] = container[height-1][l_index];
            is_left[height][now] = true;
            ++l_index;
        }
        else {
            container[height][now] = container[height-1][r_index];
            is_left[height][now] = false;
            ++r_index;
        }
    }
}

```

```

    }
    ++now;
}

/* 0-base index, k 1-base */
int query(int start, int finish, int k) {
    return query(rank-1, start, finish, k);
}

int query(int height, int start, int finish, int k) {
    if (height == 0) return container[height][start].x;
    int left_size = left[height][finish] - left[height][start];
    if (is_left[height][finish]) ++left_size;
    int right_size = finish-start+1-left_size;
    if (left_size >= k) return query(height-1, left[height][start
], min(left[height][finish], left[height][start]+left_size-1), k);
    else return query(height-1, right[height][start], min(right[
height][finish], right[height][start]+right_size-1), k-left_size);
}
};

int input[100005];
int main () {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &input[i]);
    }
    RangeTree2D range;
    range.init(input, n);
    for (int i = 0; i < m; ++i) {
        int a, b, k;
        scanf("%d%d%d", &a, &b, &k);
        printf("%d\n", range.query(a-1, b-1, k));
    }
    return 0;
}
/* Pass POJ 2104 */

```

16 Range Tree 2D, rectangle

```

struct POS {
    int x, y, value, cost, segid;
    POS(){}
    POS(int x, int y, int value, int cost):x(x), y(y), value(value),
    cost(cost) {}
    bool operator<(const POS &rhs) const {
        return this->y < rhs.y;
    }
} pos[100005];

struct SegmentTree{

    unordered_map<int, int> trans;
    int rank, capacity, length;
    POS *input;
    int *tree;

    SegmentTree() {}
    void init(POS* input, int length){
        trans.clear();
        this->input = input;
        this->length = length;
        rank = 1;
        while((1<<rank++) < length);
        capacity = 1<<(rank-1);
        tree = new int[capacity << 1];
        build(1, capacity, capacity<<1);
    }

    ~SegmentTree(){
        delete[] tree;
    }

    int build(int index, int left, int right){
        if(index >= left){
            tree[index] = getInput(index);
            trans[tree[index]] = index;
            return tree[index];
        }
        int middle = (left+right) >> 1;
        int left_value = build(lc(index), left, middle);
        int right_value = build(rc(index), middle, right);
        return tree[index] = max(left_value, right_value);
    }
}

```

```

void update(int origin_value, int value) {
    int index = trans[origin_value];
    tree[index] = value;
    maintain(index>>1);
}

void maintain(int index){
    tree[index] = max(tree[lc(index)], tree[rc(index)]);
    if(index == 1) return;
    maintain(index>>1);
}

int query(int start, int finish){
    return query(1, capacity, capacity<<1, capacity+start, capacity+
    finish+1);
}

int query(int index, int left, int right, int start, int finish){
    if(left == start && right == finish) return tree[index];
    int middle = (left+right) >> 1;
    if(finish <= middle) return query(lc(index), left, middle, start,
    finish);
    if(start >= middle) return query(rc(index), middle, right, start,
    finish);
    int left_value = query(lc(index), left, middle, start, middle);
    int right_value = query(rc(index), middle, right, middle, finish);
    return max(left_value, right_value);
}

int getInput(int index){
    index -= capacity;
    if(index < length) return input[index].value;
    return 0;
}

int lc(int x){
    return x<<1;
}

int rc(int x){
    return (x<<1)+1;
}
};

```



```

bool cmp(const POS& x, const POS& y) {
    return x.x==y.x? x.y<y.y: x.x<y.x;
}

struct rangeTree2D {
    unordered_map<int, int> trans;
    POS **container, *input;
    SegmentTree *seg;
    int rank, capacity, length;
    int *idx;
    void init(POS* input, int length) {
        trans.clear();
        sort(input, input+length, cmp);
        for (int i = 0; i < length; ++i) this->trans[input[i].value] =
i;
        this->input = input;
        this->length = length;
        rank = 1;
        while ( (1<<rank++) < length) ;
        capacity = 1<<(rank-1);
        container = new POS*[rank];
        seg = new SegmentTree[capacity<<1];
        idx = new int[length];
        POS tmp(input[length-1].x+1, input[length-1].y+1, 0, 0);
        for (int i = 0; i < rank; ++i) {
            container[i] = new POS[capacity];
        }
        for (int i = 0; i < length; ++i) {
            container[0][i] = input[i];
            idx[i] = input[i].x;
        }
        for (int i = length; i < capacity; ++i) container[0][i] = tmp;
        sort(idx, idx+length);

        // build
        int segid = 0;
        for (int height = 0; height < rank-1; ++height) {
            for (int i = 0; i < capacity; i += (2<<height)) {
                merge(container[height]+i, container[height]+i+(1<<
height),
                    container[height]+i+(1<<height), container[
height]+i+(2<<height),
                    container[height+1]+i);
                container[height+1][i].segid = segid;
                seg[segid++].init(container[height+1]+i, (2<<height));
            }
        }
    }
};

```

```

    }
}

void decrease(int value) {
    int index = trans[value];
    container[0][index].value = 0;
    maintain(1, (index>>1)<<1, value);
}

int range_query(int left, int right, int bottum, int top) {
    left = lower_bound(idx, idx+length, left)-idx;
    right = upper_bound(idx, idx+length, right)-idx;

    POS _bottum(0, bottum, 0, 0), _top(0, top, 0, 0);
    int ans = range_query(rank-1, 0, left, right, _bottum, _top);
    if (ans != 0) decrease(ans);
    if (ans == 0) return 0;
    return container[0][trans[ans]].cost;
}

void maintain(int height, int start, int value) {
    if (height == rank) return;
    int myId = container[height][start].segid;
    seg[myId].update(value, 0);
    maintain(height+1, (start>>(height+1))<<(height+1), value);
}

int range_query(int height, int start, int left, int right, const
POS& bottum, const POS& top) {
    if (start >= right || start+(1<<height) <= left) return 0;
    if (start >= left && start+(1<<height)<= right) {
        int st = lower_bound(container[height]+start, container[
height]+start+(1<<height), bottum)-container[height]-start;
        int ed = upper_bound(container[height]+start, container[
height]+start+(1<<height), top)-container[height]-start;
        --ed;
        if (ed < st) return 0;
        if (height == 0) return container[0][start].value;
        int myId = container[height][start].segid;
        return seg[myId].query(st, ed);
    }
    --height;
    return max(range_query(height, start, left, right, bottum, top
),

```

```

        range_query(height, start+(1<<height), left, right,
        bottum, top));
    }
};

```

17 Scan (JAVA)

```

import java.io.*;
import java.util.*;

public class Scan{

    BufferedReader buffer;
    StringTokenizer tok;

    Scan(){
        buffer = new BufferedReader(new InputStreamReader(System.in));
    }

    boolean hasNext(){
        while(tok==null || !tok.hasMoreElements()){
            try{
                tok = new StringTokenizer(buffer.readLine());
            }catch(Exception e){
                return false;
            }
        }
        return true;
    }

    String next(){
        if(hasNext()) return tok.nextToken();
        return null;
    }

    String nextLine(){
        if(hasNext()) return tok.nextToken("\n");
        return null;
    }

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

```

```

}

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

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

```

18 Segment Tree

```

struct SegmentTree{

    int rank, capacity, length;
    int *input, *tree;

    SegmentTree() {}
    void init(int* input, int length){
        this->input = input;
        this->length = length;
        rank = 1;
        while((1<<rank++) < length);
        capacity = 1<<(rank-1);
        tree = new int[capacity << 1];
        build(1, capacity, capacity<<1);
    }

    ~SegmentTree(){
        delete[] tree;
    }

    int build(int index, int left, int right){
        if(index >= left){
            return tree[index] = getInput(index);
        }
        int middle = (left+right) >> 1;
        int left_value = build(lc(index), left, middle);
        int right_value = build(rc(index), middle, right);
        return tree[index] = max(left_value, right_value);
    }
}

```

```

int query(int start, int finish){
    return query(1, capacity, capacity<<1, capacity+start, capacity+
        finish+1);
}

int query(int index, int left, int right, int start, int finish){
    if(left == start && right == finish) return tree[index];
    int middle = (left+right) >> 1;
    if(finish <= middle) return query(lc(index), left, middle, start,
        finish);
    if(start >= middle) return query(rc(index), middle, right, start,
        finish);
    int left_value = query(lc(index), left, middle, start, middle);
    int right_value = query(rc(index), middle, right, middle, finish);
    return max(left_value, right_value);
}

int getInput(int index){
    index -= capacity;
    if(index < length) return input[index];
    return 0;
}

int lc(int x){
    return x<<1;
}

int rc(int x){
    return (x<<1)+1;
}
};

```

19 Splay Tree

```

public class SplayTree{

    Node root;
    int size;

    SplayTree(){
        root = null;
    }
}

```

```

    size = 0;
}

public boolean containsKey(int target){
    return splay(target);
}

public void add(int target){
    // System.out.println("add "+target);
    if(root == null){
        root = new Node(null, target);
        return;
    }
    Node now = root;
    while(true){
        if(now.key == target) break;
        if(target < now.key){
            if(now.lchild == null){
                now.lchild = new Node(now, target);
                break;
            }else now = now.lchild;
        }else{
            if(now.rchild == null){
                now.rchild = new Node(now, target);
                break;
            }else now = now.rchild;
        }
    }
    splay(target);
}

public void delete(int target){
    // System.out.println("delete "+target);
    if(!containsKey(target)) return;
    Node l = root.lchild;
    Node r = root.rchild;
    if(l == null){
        root = r;
    }else l.parent = null;
    if(r == null){
        root = l;
    }else r.parent = null;
    if(root==null || root.key != target) return;
    Node lMax = l;
}

```

```

while(lMax.rchild != null) lMax = lMax.rchild;
splay(lMax.key);
lMax.rchild = r;
}

private boolean splay(int target){
//    System.out.println("splay "+target);
while(true){
    if(root == null) return false;
    if(root.key == target) return true;
    if(target < root.key){
        if(root.lchild == null) return false;
        Node l = root.lchild;
        if(l.key == target){
            root = l;
            rightRoatation(l);
            return true;
        }
        if(target < l.key){
            if(l.lchild == null) return false;
            Node a = l.lchild;
            root = a;
            rightRoatation(l);
            rightRoatation(a);
        }else{
            if(l.rchild == null) return false;
            Node b = l.rchild;
            root = b;
            leftRoatation(b);
            rightRoatation(b);
        }
    }else{
        if(root.rchild == null) return false;
        Node r = root.rchild;
        if(r.key == target){
            root = r;
            leftRoatation(r);
            return true;
        }
        if(target > r.key){
            if(r.rchild == null) return false;
            Node d = r.rchild;
            root = d;
            leftRoatation(r);
            leftRoatation(d);
        }
    }
}
}

```

```

}else{
    if(r.lchild == null) return false;
    Node c = r.lchild;
    root = c;
    rightRoatation(c);
    leftRoatation(c);
}
}
}

void print(Node now){
    if(now == null){
        System.out.print("-1 ");
        return;
    }
    System.out.print(now.key+" ");
    print(now.lchild);
    print(now.rchild);
}

void rightRoatation(Node x){
    Node r = x.parent.parent;
    Node p = x.parent;
    Node b = x.rchild;

    x.rchild = p;
    if(p != null) p.parent = x;
    if(p != null) p.lchild = b;
    if(b != null) b.parent = p;
    x.parent = r;
    if(r != null) r.lchild = x;
}

void leftRoatation(Node x){
    Node r = x.parent.parent;
    Node p = x.parent;
    Node b = x.lchild;

    x.lchild = p;
    if(p != null) p.parent = x;
    if(p != null) p.rchild = b;
    if(b != null) b.parent = p;
}

```

```

    x.parent = r;
    if(r != null) r.rchild = x;
}

class Node{

    Node parent, lchild, rchild;
    int key;

    Node(Node parent, int key){
        this.parent = parent;
        lchild = rchild = null;
        this.key = key;
    }
}

```

20 Suffix Array

```

import java.io.*;
import java.util.*;

class SuffixArray{

    Entry[] entries;
    int[] rank;
    int length;

    SuffixArray(CharSequence S){
        length = S.length();
        rank = new int[length];
        entries = new Entry[length];
        int[] temp = new int[length];
        int counter;
        for (int i=0;i<length;i++){
            entries[i] = new Entry(i);
            entries[i].a = S.charAt(i) - 'a';
        }
        Arrays.parallelSort(entries);
        rank[entries[0].index] = temp[0] = counter = 0;
        for(int i=1;i<length;i++){
            if(entries[i].a != entries[i-1].a) counter++;

```

```

            rank[entries[i].index] = temp[i] = counter;
        }
        int step = 1;
        while(step < length){
            for(int i=0;i<length;i++){
                entries[i].a = temp[i];
                entries[i].b = rank[(entries[i].index+step)%length];
            }
            countingSort(entries);
            rank[entries[0].index] = temp[0] = counter = 0;
            for(int i=1;i<length;i++){
                if(entries[i].a != entries[i-1].a || entries[i].b !=
entries[i-1].b) counter++;
                rank[entries[i].index] = temp[i] = counter;
            }
            step <= 1;
        }
    }

    void countingSort(Entry[] input){
        int[] counter = new int[length];
        Entry[] temp = new Entry[length];
        for(int i=0;i<length;i++) counter[input[i].b]++;
        for(int i=1;i<length;i++) counter[i] += counter[i-1];
        for(int i=length-1;i>=0;i--) temp[--counter[input[i].b]] =
input[i];
        Arrays.fill(counter, 0);
        for(int i=0;i<length;i++) counter[temp[i].a]++;
        for(int i=1;i<length;i++) counter[i] += counter[i-1];
        for(int i=length-1;i>=0;i--) input[--counter[temp[i].a]] =
temp[i];
    }

    class Entry implements Comparable<Entry>{

        int a, b, index;

        Entry(int index){
            this.index = index;
        }

        void assign(Entry rhs){
            a = rhs.a;
            b = rhs.b;
        }
    }
}

```

```

    @Override
    public int compareTo(Entry rhs){
        return a - rhs.a;
    }

}
}

```

21 Treap

```

#include <bits/stdc++.h>

using namespace std;

typedef int T;
typedef char T1;

struct Treap {
    T key, priority, size;
    Treap *lc, *rc;

    T1 value;
    bool reverse;
    Treap(T key, T1 value): key(key), priority(rand()),
        size(1), lc(NULL), rc(NULL), value(value), reverse(false) {}
};

inline int size(Treap *target) {
    if (!target) return 0;
    return target->size;
}

inline void pull(Treap *target) {
    target->size = size(target->lc) + size(target->rc) + 1;
}

void reverseIt(Treap *target) {
    if (!(target->reverse)) return;
    Treap *lc = target->lc;
    target->lc = target->rc;

```

```

    target->rc = lc;
    target->reverse = false;
    if (target->lc) (target->lc->reverse) ^= true;
    if (target->rc) (target->rc->reverse) ^= true;
}

Treap* merge(Treap *lhs, Treap *rhs) {
    if (!lhs || !rhs) return lhs? lhs: rhs;
    if (lhs->priority > rhs->priority) {
        reverseIt(lhs);
        lhs->rc = merge(lhs->rc, rhs);
        pull(lhs);
        return lhs;
    }
    else {
        reverseIt(rhs);
        rhs->lc = merge(lhs, rhs->lc);
        pull(rhs);
        return rhs;
    }
}

void split(Treap *target, Treap *&lhs, Treap *&rhs, int k) {
    if (!target) lhs = rhs = NULL;
    else if (k > target->key) {
        lhs = target;
        split(target->rc, lhs->rc, rhs, k);
        pull(lhs);
    }
    else {
        rhs = target;
        split(target->lc, lhs, rhs->lc, k);
        pull(rhs);
    }
}

Treap* insert(Treap *target, int key, int value) {
    Treap *lhs, *rhs;
    split(target, lhs, rhs, key);
    return merge(merge(lhs, new Treap(key, value)), rhs);
}

/* split by size */
void splitSize(Treap *target, Treap *&lhs, Treap *&rhs, int k) {
    if (!target) lhs = rhs = NULL;

```

```

    else {
        reverseIt(target);
        if (size(target->lc) < k) {
            lhs = target;
            splitSize(target->rc, lhs->rc, rhs, k-size(target->lc)-1);
            pull(lhs);
        }
        else {
            rhs = target;
            splitSize(target->lc, lhs, rhs->lc, k);
            pull(rhs);
        }
    }
}

/* do lazy tag */
Treap* reverseIt(Treap *target, int lp, int rp) {
    Treap *A, *B, *C, *D;
    splitSize(target, A, B, lp-1);
    splitSize(B, C, D, rp-lp+1);
    C->reverse ^= true;
    return merge( merge(A, C), D);
}

/* delete singal key */
Treap* del(Treap *target, int key) {
    if (target->key == key) return merge(target->lc, target->rc);
    else if (target->key > key) target->lc = del(target->lc, key);
    else target->rc = del(target->rc, key);
    pull(target);
    return target;
}

T findK(Treap *target, int k) {
    if (size(target->lc)+1 == k) return target->key;
    else if (size(target->lc) < k) return findK(target->rc, k-size(
target->lc)-1);
    else return findK(target->lc, k);
}

/* find the kth's value */
T1 findK(Treap *target, int k) {
    reverseIt(target);
    if (size(target->lc)+1 == k) return target->value;
    else if (size(target->lc) < k) return findK(target->rc, k-size(

```

```

target->lc)-1);
    else return findK(target->lc, k);
}

int main () {
    return 0;
}

/* pass POJ2761, CF gym 100488 pL */

```

22 Z Algorithm

```

void z_algorithm(string& input) {
    int z[1000005];
    memset(z, 0, sizeof(z));
    z[0] = input.size();
    int L = 0, R = 1;
    for (int i = 1; i < input.size(); ++i) {
        if (R <= i || z[i-L] >= R-i) {
            int x = ((i>=R)? i: R);
            while (x < input.size() && input[x] == input[x-i]) x++;
            z[i] = x-i;
            if (i < x) {
                L = i;
                R = x;
            }
        }
        else {
            z[i] = z[i-L];
        }
    }
}

```

23 歐拉定理

假若 a 與 n 互質，那麼 $a^{\phi(n)} - 1$ 可被 n 整除。亦即， $a^{\phi(n)} \equiv 1 \pmod{n}$ 。若 m, n 互質，則 $\phi(mn) = \phi(m)\phi(n)$ 。

24 路卡斯公式

$$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$$

where $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$ and $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$.

25 模數合成

見 codeBook: modCombine

26 強國人說的歐拉定理

如果 a 和 n 互質，那麼 $a^{\phi(n)} \equiv 1 \pmod{n}$ ，對於任意 a, n 和較大的 b ，有 $a^b \equiv a^{b \bmod \phi(n) + \phi(n)} \pmod{n}$

27 無權邊的生成樹個數 Kirchhoff's Theorem

- 定義 $n \times m$ 矩陣 $E = (a_{i,j})$ ， n 為點數， m 為邊數，若 i 點在 j 編上， i 為小點 $a_{i,j} = 1$ ， i 為大點 $a_{i,j} = -1$ ，否則 $a_{i,j} = 0$ 。
(證明省略)
- 令 $E(E^T) = Q$ ，他是一種有負號的 kirchhoff 的矩陣，取 Q 的子矩陣即為 $F(F^T)$
結論：做 Q 取子矩陣算 \det 即為所求。(除去第一行第一列 by mz)

28 很大的質數

18446744082299486207

29 GP 東北數學式

$$(p-1)!/p \pmod{p} = p-1$$

$$C(n, m) = C(n/p, m/p) * C(n \% p, m \% p)$$

30 圓周率 1000 位

3.141592653589793238462643383279502884197169399375105820974944592307816406286
20899862803482534211706798214808651328230664709384460955058223172535940812848
11174502841027019385211055596446229489549303819644288109756659334461284756482
33786783165271201909145648566923460348610454326648213393607260249141273724587
00660631558817488152092096282925409171536436789259036001133053054882046652138
41469519415116094330572703657595919530921861173819326117931051185480744623799
62749567351885752724891227938183011949129833673362440656643086021394946395224
73719070217986094370277053921717629317675238467481846766940513200056812714526
35608277857713427577896091736371787214684409012249534301465495853710507922796
89258923542019956112129021960864034418159813629774771309960518707211349999998
37297804995105973173281609631859502445945534690830264252230825334468503526193
11881710100031378387528865875332083814206171776691473035982534904287554687311
59562863882353787593751957781857780532171226806613001927876611195909216420199

31 尤拉數 e

2.718281828459045235360287471352662497757247093699959574966967627724076630353
54759457138217852516642742746

32 歐拉示性數

$\chi = F - E + V$
幾何中同類的形狀， χ 為相同值

33 半平面交相關幾何轉換

$$(a, b) \Leftrightarrow y = ax + b$$