```
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
```

code/.vimrc

```
#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());
    }
}
```

code/AC_Automaton.cpp

```
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;
}
```

code/combination.cpp

```
/* 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);
}
```

code/double_lca.cpp

```
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;
```

```java
    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;
    }
  }

}
```

code/Flow_Dinics.java

```cpp
#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 -() {
        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 fisrt 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]);
}

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;
}
```

code/Geometry.cpp

```
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;
    }

}
```

code/SimpleTabulationHash.java

```
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;
    }
}
```

code/IDAstar.cpp

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

using namespace std;

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 (int i = 0; i < 2; ++i) {
            for (int 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];
}

int main () {
    int n, mod;
    cin >> n >> mod;
    cout << inverse(mod, n);
}
```

code/inverse.cpp

```
#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];

```

```cpp
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_hangarian() {

    /* 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;
}
```

code/km.cpp

```cpp
#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;
}
```

code/linear_prime.cpp

```
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;
}
```

code/modCombine.cpp

```
#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 */
```

code/rangeTree2D.cpp

```
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;
  }
};
```

code/SegmentTree.cpp

```
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;
    }
  }
}
```

code/SplayTree.java

```
/* Time Complexity=2*n*log(n)*log(n) */
#include <cstdio>
#include <algorithm>
using namespace std;

class Weight{
public:
    Weight(int a=0,int b=0,int c=0):id(a),first(b),second(c){}
    int id,first,second;
    bool operator<(const Weight &rhs)const{
        return first<rhs.first||(first==rhs.first&&second<rhs.second);
    }
    bool operator==(const Weight &rhs)const{
        return first==rhs.first&&second==rhs.second;
    }
    bool operator!=(const Weight &rhs)const{
        return!((*this)==rhs);
    }
};
```

```
class SuffixArray{
public:
    SuffixArray(char *r):refer(r){
        for(length=0;refer[length]!='\0';length++);
        rankOfIndex=new int[length];
        indexOfRank=new int[length];
        texi=new Weight[length];//=
        firstsort();
        for(int know=1;know<=length;know<<=1) doublesort(know);
    }
    ~SuffixArray() {
        delete [] rankOfIndex;
        delete [] indexOfRank;
        delete [] texi;
    }

    void firstsort(){
        for(int i=0;i<length;i++){
            texi[i]=Weight(i,refer[i]);
        }
        sort(&texi[0],&texi[length-1]+1);

        indexOfRank[rankOfIndex[texi[0].id]=0]=texi[0].id;
        int current=0;
        for(int i=1;i<length;i++){
            if(texi[i]!=texi[i-1]) current++;
            indexOfRank[i]=texi[i].id;
            rankOfIndex[texi[i].id]=current;
        }
    }

    void doublesort(int known){
        for(int i=0;i<length;i++){
            texi[i]=Weight(i,rankOfIndex[i],(i+known<length)?rankOfIndex[i+known]:-1);
        }

        sort(&texi[0],&texi[length-1]+1);

        indexOfRank[rankOfIndex[texi[0].id]=0]=texi[0].id;
        int current=0;
        for(int i=1;i<length;i++){
            if(texi[i]!=texi[i-1]) current++;
            indexOfRank[i]=texi[i].id;
            rankOfIndex[texi[i].id]=current;
        }

    }

    void print(int i,bool newline=0){
        printf("%s",&refer[indexOfRank[i]]);
        if(newline) printf("\n");
    }

    void printall(){
        for(int i=0;i<length;i++) print(i,1);
    }

    int *indexOfRank,*rankOfIndex,length;
    char *refer;
    Weight *texi;
};

int main()
{
    char str[100];
    scanf("%s", str);
    SuffixArray a(str);
    a.printall();
    return 0;
}
```

code/SuffixArray.cpp

```
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;
        }

    }

}
```

code/SuffixArray.java

```
#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);
}
```

```cpp

/* 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 */
```

code/treap.cpp

```cpp
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];
        }
    }
}
```

code/z_algorithm.cpp