Documentación Legendary Grandpupils

INDEX

1. Plantilla

```cpp
#include <bits/stdc++.h>
#include <algorithm>
#include <bitset>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <deque>
#include <fstream>
#include <functional>
#include <iomanip>
#include <iostream>
#include <limits.h>
#include <map>
#include <math.h>
#include <numeric>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <utility>
#include <vector>

#define PI 3.14159265358979323846
#define EPS 1e-6
#define INF 1000000000

#define _ ios_base::sync_with_stdio(0), cin.tie(0), cin.tie(0), cout.tie(0), cout.precision(15);
#define FOR(i, a, b) for(int i=int(a); i<int(b); i++)
#define RFOR(i, a, b) for(int i=int(a)-1; i>=int(b); i--)
#define FORC(cont, it) for(decltype((cont).begin()) it = (cont).begin(); it != (cont).end(); it++)
#define RFORC(cont, it) for(decltype((cont).rbegin()) it = (cont).rbegin(); it != (cont).rend(); it++)
#define pb push_back

using namespace std;

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

#define MAXN 10
#define MOD 1000000007

int main() { _

    return 0;
}
```

2. DP

NK

```
ll fact[MAXN], inv[MAXN], pascal[MAXN][MAXN];

ll ppow(ll n, ll p) {
    ll ret = 1;
    while (p) {
        if (p & 1) ret = (ret*n) % MOD;
        n = (n*n) % MOD;
        p>>=1;
    }
    return ret;
}

void generaFact() {
    fact[0] = 1;
    FOR(i, 1, MAXN)
        fact[i] = (fact[i-1] * i) % MOD;

    inv[MAXN-1] = ppow(fact[MAXN-1], MOD - 2);
    for(int i = MAXN-1; i>0; i--)
        inv[i-1] = inv[i] * i % MOD;
}

ll NK(ll n, ll k) {
    return fact[n] * inv[k] % MOD * inv[n-k] % MOD;
}

// pascal i,j corresponde a la formula de combinatoria i!/(j!*(i-j)!), AKA. (N, K)
void trianguloPascal() {
    pascal[0][0] = 1;
    FOR(i, 1, MAXN) {
        pascal[i][0] = 1;
        FOR(j, 1, i+1) {
            pascal[i][j] = (pascal[i - 1][j - 1] + pascal[i - 1][j]) % MOD;
        }
    }
}

int main(){
    ll n, k;
    generaFact();
    trianguloPascal();
    while (scanf("%lld %lld", &n, &k) != EOF) {
        printf("%lld\n", NK(n, k));
        printf("%lld\n", pascal[n][k]);
    }

    return 0;
}
```

Longest Common Subsequence

```
#define MAXN 1000005
#define MOD 1000000007

/* Returns length of LCS for X[0..n-1], Y[0..m-1] */
int lcs(string s1, string s2) {
    int n = s1.length();
    int m = s2.length();
    int L[n + 1][m + 1];
    memset(L, 0, sizeof(L));

    /* Following steps build L[n+1][m+1] in bottom up fashion. Note
    that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
    FOR(i, 1, n+1) {
        FOR(j, 1, m+1) {
            if (s1[i-1] == s2[j-1])
                L[i][j] = L[i-1][j-1] + 1;
            else
                L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }

    /* L[n][m] contains length of LCS for X[0..n-1] and Y[0..m-1] */
    return L[n][m];
}

int main() {
    string s1, s2;
    while (cin >> s1 >> s2)
    cout << lcs(s1, s2) << endl;
}
```

Longest Common Substring

```
#define MAXN 10
#define MOD 1000000007

/* Returns length of longest common substring of X[0..n-1] and Y[0..m-1] */
// Create a table to store lengths of longest common suffixes of
// substrings.   Notethat LCSuff[i][j] contains length of longest
// common suffix of X[0..i-1] and Y[0..j-1]. The first row and
// first column entries have no logical meaning, they are used only
// for simplicity of program
int LCSubStr(string X, string Y) {
    int n = X.length();
    int m = Y.length();

    int lcs[n+1][m+1];
    memset(lcs, 0, sizeof(lcs));
    int result = 0;  // To store length of the longest common substring

    /* Following steps build lcs[n+1][m+1] in bottom up fashion. */
    FOR(i, 1, n+1) {
        FOR(j, 1, m+1) {
            if (X[i-1] == Y[j-1]) {
                lcs[i][j] = lcs[i-1][j-1] + 1;
                result = max(result, lcs[i][j]);
            }
            else {
                lcs[i][j] = 0;
            }
        }
    }

    return result;
}

int main() {
    string X, Y;
    cin >> X >> Y;
    cout << LCSubStr(X, Y) << endl;

    return 0;
}
```

TSP

```
#define MAXN 25

int N;
int dptsp[MAXN][1 << MAXN], costViajero[MAXN][MAXN];
int maskInicial;

// Calcula costos de todos los nodos a todos los nodos.
void floyd() {
    FOR(k, 0, N) {
        FOR(i, 0, N) {
            FOR(j, 0, N) {
                costViajero[i][j] = min(costViajero[i][j], costViajero[i][k] + costViajero[k][j]);
            }
        }
    }
}

// Bits prendidos significa ciudad que aun no se visita.
int tsp(int n, int mask) {
    if (!mask) return 0;
    int &c = dptsp[n][mask];
    if (~c) return c;
    c = INF;
    FOR(i, 0, N) {
        if (mask & 1 << i) {
            c = min(c, tsp(i, mask ^ 1 << i) + costViajero[n][i]);
        }
    }
    return c;
}

int main() {
    int ans;

    floyd();
    maskInicial = (1 << N) - 1, ans = INF;
    FOR(i, 0, N) {
        ans = min(ans, tsp(i, maskInicial ^ 1 << i));
    }

    return 0;
}
```

3. Geometry

```
typedef long long ll;

double degToRad(double theta){
    return theta * PI / 180.0;
}

double radToDeg(double theta){
    return theta * 180 / PI;
}

// ******************* Struct: Punto **************************************
struct point {
    double x, y;

    point() {}
    point(double xx, double yy) {
        x = xx;
        y = yy;
    }
    point inf() {
        x = INF;
        y = INF;
    }

    double dist(point p) {
        return hypot(x - p.x, y - p.y);
    }

    point rotate(point pivot, point p, double ang) {
        double s = sin(ang);
        double c = cos(ang);

        // translate point back to origin:
        p.x -= pivot.x;
        p.y -= pivot.y;

        // rotate point
        double xaux = p.x * c - p.y * s;
        double yaux = p.x * s + p.y * c;

        // translate point back:
        p.x = xaux + pivot.x;
        p.y = yaux + pivot.y;
        return p;
    }

    double getAngle(point pivot, point p) {
        return atan2(p.y - pivot.y, p.x - pivot.x);
    }

    void swap() {
        double aux = x;
        x = y;
        y = aux;
    }

    void swap(point &a, point &b) {
        point aux = a;
        a = b;
        b = aux;
    }

    point punto(point const &p) const {
        return point(x * p.x, y * p.y);
    }

    double cruz(point const &p) const {
        return x * p.y - y * p.x;
    }

    double cruz(point const &p1, point const &p2) const {
        return (p1.x - x) * (p2.y - y) - (p1.y - y) * (p2.x - x);
```

```
    }

    point operator +(point const &p) const {
        return point(x + p.x, y + p.y);
    }

    point operator -(point const &p) const {
        return point(x + p.x, y + p.y);
    }

    point operator /(double d) const {
        return point(x / d, y / d);
    }

    bool operator <(point const &p) const {
        return (x - p.x) > EPS && x < p.x ||
                fabs(x - p.x) < EPS && (y - p.y) > EPS && y < p.y;
    }

    bool operator ==(point p) const {
        return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS;
    }

    void print() {
        cout << x << "\t" << y << endl;
    }
};
// *************************************************************************

// ******************* Struct: line ***************************************
struct line {
    point p1, p2;
    double a, b, c;

    line() {
        p1.inf();
        p2.inf();
    }
    line(double A, double B, double C) {
        a = A;
        b = B;
        c = C;

        p1.inf();
        p2.inf();
    }
    line(point pp1, point pp2) {
        p1 = pp1;
        p2 = pp2;

        if (p2 < p1)
            swap(p1, p2);

        if(fabs(p1.x - p2.x) < EPS) {
            a = 1.0;
            b = 0.0;
            c = -p1.x;
        }
        else {
            a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
            b = 1.0;
            c = -(double)(a * p1.x) - (b * p1.y);
        }
    }

    line perpendicular(point p) {
        double c = -(a * p.x - b * p.y);
        return line(-b, a, c);
    }

    bool sonParalelas(line l1, line l2){
        return fabs(l1.a - l2.a) < EPS &&
                fabs(l1.b - l2.b) < EPS;
    }
```

```cpp

    bool sonIguales(line l1, line l2){
        return sonParalelas(l1, l2) &&
                fabs(l1.c - l2.c) < EPS;
    }

    bool insideLine(point p) {
        return ((p1.x - EPS <= p.x && p.x <= p2.x + EPS ||
            p2.x - EPS <= p.x && p.x <= p1.x + EPS)     &&

            (p1.y - EPS <= p.y && p.y <= p2.y + EPS ||
            p2.y - EPS <= p.y && p.y <= p1.y + EPS));
    }

    bool intercectan(line l1, line l2, point &p){
        if (sonParalelas(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) / l1.b;
        else
            p.y = -(l2.a * p.x + l2.c) / l2.b;

        return l2.insideLine(p);
    }
};
// **************************************************************************

// ******************** Struct: triangle ***********************************
struct triangle {
    point p1, p2, p3;
    double a, b, c;
    double A, B, C;
    double area, perimetro;
    int type; // equilatero=1, isoseles=2, equilatero=3
    bool rect; // triangulo rectangulo o no

    triangle() {}
    triangle(double aa, double bb, double cc) {
        a = aa;
        b = bb;
        c = cc;
        sort();

        act();
    }

    triangle(point pp1, point pp2, point pp3) {
        p1 = pp1;
        p2 = pp2;
        p3 = pp3;

        a = p1.dist(p2);
        b = p1.dist(p3);
        c = p2.dist(p3);
        sort();

        act();
    }

    double innerCircleRadio(){
        return area / (perimetro * 2.0);
    }

    double outterCircleRadio(){
        return a * b * c / (4 * area);
    }

    int getType() {
        if(a==b && b==c)        return 1;
        else if(a==b || b==c)   return 2;
```

```
            return 3;
        }

        bool isRight(){
            return (a*a + b*b - c*c < EPS);
        }

        void sort() {
            if (a > b)
                swap(a, b);
            if (a > c)
                swap(a, c);
            if (b > c)
                swap(b, c);
        }

        double getPerimetro() {
            return a + b + c;
        }

        double getArea() {
            double s = perimetro / 2.0;
            return sqrt(s * (s-a) *(s-b) * (s-c));
        }

        double getAngles() {
            double aa = a * a;
            double bb = b * b;
            double cc = c * c;

            A = acos(bb + cc - aa) / (2 * b * c);
            B = acos(aa + cc - bb) / (2 * a * c);
            C = acos(aa + bb - cc) / (2 * a * b);
        }

        void act() {
            type = getType();
            rect = isRight();
            perimetro = getPerimetro();
            area = getArea();
            getAngles();
        }
    };
    // *************************************************************************


    // ******************** Struct: circle **************************************
    struct circle {
        point c;
        double r, circ, area;

        circle() {
            c.x = c.y = 0;
            r=1;

            act();
        }
        circle(point p, double rr){
            c = p;
            r = rr;

            act();
        }

        int insideCircle(point p){// 0-Dentro, 1-Borde, 2-Fuera
            int dx = p.x - c.x, dy = p.y - c.y;
            int Euc = dx*dx + dy*dy, rSq=r*r;
            double dist = c.dist(p);
            if (fabs(dist - r) < EPS)   return 1;
            if (dist < r)               return 0;
            return 2;
        }

        double getArc(double deg) {
```

```
        return circ * deg / 360.0;
    }
    double getChord(double deg) {
        return 2.0 * r * r * (1 - cos(degToRad(deg)));
    }
    double getSector(double deg) {
        return area * deg / 360.0;
    }
    double getSegment(double deg) {
        triangle t(r, r, getChord(deg));
        return getSector(deg) - t.getArea();
    }

    // 0 - No intercectan
    // 1 - Intersecta 1 punto
    // 2 - Intersecta 2 puntos
    // 3 - Mismo circulo
    // 4 - Circulo dentro del otro
    int intersectCirc(circle const &cir, point &p1, point &p2) {
        double d = c.dist(cir.c);

        if (d > r + cir.r)                      // No se tocan
            return 0;
        if ((r - cir.r) < EPS && c == cir.c)    // Mismo circulo
            return 3;
        if (d + min(r, cir.r) < max(r, cir.r))  // Circulo contiene otro
            return 4;

        double a = (r * r - cir.r * cir.r + d * d) / (2.0 * d);
        double h = sqrt(r * r - a * a);

        // find p2
        point pp1(c.x + (a * (cir.c.x - c.x)) / d, c.y + (a * (cir.c.y - c.y)) / d);

        // find intersection points p3
        p1 = point(pp1.x + (h * (cir.c.y - c.y)/ d),
                pp1.y - (h * (cir.c.x - c.x)/ d));

        p2 = point(pp1.x - (h * (cir.c.y - c.y)/ d),
                pp1.y + (h * (cir.c.x - c.x)/ d));

        if(fabs(d - r - cir.r) < EPS)
            return 1;
        return 2;
    }

    double getArea() {
        return PI * r * r;
    }

    double getCirc() {
        return PI * 2. * r;
    }

    bool operator ==(circle const &cir) const{
        return c == cir.c && fabs(r - cir.r) < EPS;
    }

    void act() {
        area = getArea();
        circ = getCirc();
    }
};

// ***************************************************************************

typedef vector<point> vp;
typedef vector<line> vl;
typedef vector<triangle> vt;
typedef vector<circle> vc;

// ********************** Convex Hull ***************************************
point p0;

```

```cpp
int orientation(point p, point q, point r) {
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

bool compAng(const point p1, const point p2) {
    int o = orientation(p0, p1, p2);

    return (o == 0 &&
            hypot(p0.x - p2.x, p0.y - p2.y) >= hypot(p0.x - p1.x, p0.y - p1.y)) ||
            (o == 2);
}

bool compY(const point p1, const point p2) {
    return p1.y < p2.y || (p1.y == p2.y && p1.x < p2.x);
}

vp convexHull(vp v) {
    vp c = v;
    int n = v.size(), mini = 0;
    p0 = c[0];

    FOR(i, 1, n) {
        int y = c[i].y;

        if (compY(c[i], p0)) {
            p0 = c[i];
            mini = i;
        }
    }

    swap(c[0], c[mini]);

    sort(c.begin() + 1, c.end(), compAng);

    vp ans;
    ans.pb(c[0]);
    ans.pb(c[1]);
    ans.pb(c[2]);

    FOR (i, 3, n) {
        while (orientation(ans[ans.size()-2], ans[ans.size()-1], c[i]) != 2) {
            ans.erase(ans.end());
        }
        ans.pb(c[i]);
    }
    ans.pb(ans[0]);

    return ans;
}
// ***************************************************************************


// ********************** Convex Hull v2 ************************************
double 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);
}

vp convexHull(vp &P) {
    int n = P.size(), k = 0;
    vp H(2*n);
    sort(P.begin(), P.end());
    FOR(i, 0, n) {
        while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }
    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);
```

```
    return H;
}
// ************************************************************************



// ********************** Revisa si es CW **********************************
bool cw(vp v) {
    double ret = 0;
    FOR(i, 1, v.size()) {
        ret += v[i].x * v[i-1].y - v[i-1].x * v[i].y;
    }
    return ret < 0;
}
// ************************************************************************


// ********* Revisa si un punto esta dentro en un poligono *****************
bool PointInPolygon(vp v, point p) {
    bool ret = 0;
    int j = 0;
    FOR(i, 1, v.size()) {
        if(((v[i].y > p.y) != (v[j].y > p.y)) &&
            (p.x < (v[j].x - v[i].x) * (p.y - v[i].y) / (v[j].y - v[i].y) + v[i].x)
            )
            ret = !ret;
        j++;
    }
    return ret;
}
// ************************************************************************


// ************* Calcular perimetro de un poligono *************************
double perimetro(vp v) {
    double ret = 0;
    FOR(i, 0, v.size() - 1) {
        ret += v[i].dist(v[i + 1]);
    }
    return ret;
}
// ************************************************************************


// ****************** Calcular area de un poligono *************************
double area(vp v) {
    double ret = 0;
    FOR(i, 1, v.size()) {
        ret += v[i].x * v[i-1].y - v[i-1].x * v[i].y;
    }
    return fabs(ret) / 2.0;
}
// ************************************************************************




int main () { _
    point p(1, 2), p2(2,3);
    cout << p.x << " " << p.y << endl;

    double ang = p.getAngle(p, p2);
    cout << p.getAngle(p, p2) * 180 / PI << endl;
    p.rotate(p, p2, -ang).print();
}
```

4. Graph's Theory

Centroid decomposition

```
//CENTROID DECOMPOSITION BY ADANCITO

//NOTES:
/*
 -The consider() method contains an algorithm specifically thought of to count pairs in trees which fulfill a certain characteristic (for further details go to method). However, since this method will be called for all centroids, it can be changed to anything the problem needs to do.
 -If consider()'s efficiency is O(F(n)), then the overall complexity is O(F(n) * lgn).
 */
const int maxN = 5e5 + 6;
vector<int> adj[maxN];
int n, sz[maxN], arr[maxN];

bool dead[maxN];

void precalc(int nd, int an){
    sz[nd] = 1;
    for(int sn: adj[nd]){
        if(sn != an && !dead[sn]){
            precalc(sn, nd);
            sz[nd] += sz[sn];
        }
    }
}

int getCentroid(int nd, int an, int tot){
    for(int sn: adj[nd]){
        if(sn != an && !dead[sn] && sz[sn] * 2 > tot){
            return getCentroid(sn, nd, tot);
        }
    }
    return nd;
}

ll ans = 0;

int arr1[maxN], arr2[maxN], curs, ots;

void dfs(int nd, int an, int lev){
    // example:
    while(curs <= lev) arr2[curs++] = 0;
    arr2[lev]++;
    //bucket in arr2[lev] whatever nd represents. Afterwards, just do the same with children

    for(int sn: adj[nd]){
        if(sn != an && !dead[sn]){
            dfs(sn, nd, lev);
        }
    }

}

void mergeVec(){
    //Accumulating arrays to keep the consider() method linear
    while(ots < curs) arr1[ots++] = 0;
    for(int i = 0; i < curs; i++){
        arr1[i] += arr2[i];
    }
}
```

```
/*
 The most popular CD problem type is pair-counting in trees. This sample is a basic technique that uses bucketing and accumulating. Today, we will count pairs of trees whose path length is odd.
 */
void consider(int nd){

    //ots is the size for arr1, curs is the size for arr2

    for(int sn: adj[nd]){
        if(!dead[sn]){
            curs = 0;
            dfs(sn, nd, 0);

            //example:

            for(int j = 0; j < curs; j++){
                ans += ll(arr1[j]) * arr2[j];
            }
            //iterate through arr2, and multiply by frequency in arr1 at the bucket place it corresponds

            mergeVec();

        }
    }
}

void solve(int nd){
    precalc(nd, -1);
    nd = getCentroid(nd, -1, sz[nd]);

    consider(nd);
    dead[nd] = 1;

    for(int sn: adj[nd]){
        if(!dead[sn]){
            solve(sn);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    cin >> n;

    for(int i = 0; i < n; i++){
        cin >> arr[i];
    }


    for(int i = 1; i < n; i++){
        int u, v;
        cin >> u >> v, u--, v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    solve(0);

}
```

HLD

```cpp
const int maxN = 1e4 + 3, MOD = 1e9 + 9, LG = 21;
//NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom

int he[maxN], par[maxN], ind[maxN], cs[maxN], heavy[maxN], head[maxN], n, T[maxN << 1];
vector<int> adj[maxN];
ii ed[maxN];

int dfs(int nd){
    int ret = 1, heaS = 0, cr;
    for(int sn: adj[nd]) if(sn != par[nd]){
        par[sn] = nd, he[sn] = he[nd] + 1, cr = dfs(sn), ret += cr;
        if(heaS < cr) heaS = cr, heavy[nd] = sn;
    }
    return ret;
}

void setPaths(){
    memset(heavy, -1, sizeof(heavy)), par[0] = -1, he[0] = 0, dfs(0);
    for(int i = 0, ct = 0; i < n; i++)
        if(par[i] == -1 || heavy[par[i]] != i)
            for(int j = i; j != -1; j = heavy[j])
                head[j] = i, ind[j] = ct++;
}

int query(int l, int r){
    int ret = 0;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1){
        if(l & 1) ret = max(ret, T[l++]);
        if(r & 1) ret = max(ret, T[--r]);
    }
    return ret;
}

void update(int w, int x){
    w += n, T[w] = x;
    while(w > 1)
        w >>= 1, T[w] = max(T[w << 1], T[w << 1 | 1]);
}

int ans(int u, int v){
    int ret = 0;
    for(; head[u] != head[v]; v = par[head[v]]){
        if(he[head[u]] > he[head[v]]) swap(u, v);
        ret = max(ret, query(ind[head[v]], ind[v] + 1));
    }
    if(he[u] > he[v]) swap(u, v);
    return max(ret, query(ind[u] + 1, ind[v] + 1));
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    int t;
    cin >> t;
    while(t--){
        cin >> n;

        for(int i = 0; i < n; i++) adj[i].clear();

        for(int i = 1, u, v; i < n; i++){
            cin >> u >> v >> cs[i];
```

```
            u--, v--;
            adj[u].push_back(v), adj[v].push_back(u);
            ed[i] = ii(u, v);
        }

        setPaths();

        for(int i = 1; i < n; i++){
            if(he[ed[i].fi] < he[ed[i].se]) swap(ed[i].fi, ed[i].se);
            update(ind[ed[i].fi], cs[i]);
        }

        while(true){
            string inp;
            int a, b;
            cin >> inp;
            if(inp == "DONE") break;
            cin >> a >> b;
            if(inp == "CHANGE"){
                update(ind[ed[a].fi], b);
            }
            else{
                cout << ans(--a, --b) << '\n';
            }
        }

    }

}
```

Isomorphic Rooted Tree Test

```
const int maxN = 1e5 + 3, maxV = 1e6;

int n, rta, rtb, mla, mlb;

struct El{
    int uu;
    vector<int> vc;
    El(int uu): uu(uu), vc(vector<int>()){}
    bool operator!=(const El &ot) const{
        return vc != ot.vc;
    }
    bool operator<(const El &ot) const{
        return vc < ot.vc;
    }
};
bool notequal(const vector<El> &a, const vector<El> &b){
    if(a.size() != b.size()) return true;
    for(int i = 0; i < a.size(); i++){
        if(a[i] != b[i]) return true;
    }
    return false;
}
vector<int> one[maxN], two[maxN];
vector<El> vcs[2][maxN];
int le[maxN];

void getinp(vector<int> *adj, int &rt){
    for(int i = 0, pa; i < n; i++){
        cin >> pa;
        if(!pa) rt = i;
        else adj[--pa].push_back(i);
    }
}

void dfs(vector<int> *adj, vector<El> *vcc, int nd, int ct, int &bes){
    bes = max(bes, ct);
    for(int sn: adj[nd]){
        vcc[ct + 1].push_back(El(int(vcc[ct].size() - 1)));
        dfs(adj, vcc, sn, ct + 1, bes);
    }
}

bool isom(){
    for(int r = max(mla, mlb); r >= 0; r--){
        for(int u = 0; u < 2; u++) sort(vcs[u][r].begin(), vcs[u][r].end());
        //cout << "good " << r << '\n';
        if(notequal(vcs[0][r], vcs[1][r])) return false;
        for(int u = 0; u < 2 && r; u++){
            for(int i = 0, j = 0; i < vcs[u][r].size(); i = j){
                for(;j<vcs[u][r].size() && vcs[u][r][i].vc == vcs[u][r][j].vc; j++){
                    int uu = vcs[u][r][j].uu;
                    vcs[u][r - 1][uu].vc.push_back(i);
                }
            }
        }
    }
    return true;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

```

```
    int t;
    cin >> t;

    while(t--){
        cin >> n;

        mlb = mla = 0;
        for(int i = 0; i < n; i++)
            one[i].clear(), two[i].clear(), vcs[0][i].clear(), vcs[1][i].clear();

        getinp(one, rta), getinp(two, rtb);

        vcs[0][0].push_back(El(-1));
        vcs[1][0].push_back(El(-1));
        dfs(one, vcs[0], rta, 0, mla), dfs(two, vcs[1], rtb, 0, mlb);

        cout << isom() << '\n';

    }

}
```

Dijkstra

```
typedef long long ll;
typedef pair<ll, ll> ii;
typedef pair<ii, ll> iii;
#define FIRST first.first
#define SECOND first.second
#define THIRD second
typedef vector<ll> vi;
typedef vector<ii> vii;

#define MAXN 100000
#define MOD 1000000007

vii edges[MAXN];// first = cost, second = where
ii arr[MAXN];// shortest paths to all nodes will be stored here
ll n; // size of graph

void dijkstra(ll from, ll to) {
    for(int i = 0; i < n; i++) arr[i].first = INT_MAX;
    priority_queue<iii, vector<iii>, greater<iii> > pq;
    pq.push(iii(ii(0, -1), from));
    while(!pq.empty()){
        iii cur = pq.top();
        pq.pop();
        if(arr[cur.THIRD].first == INT_MAX)
            arr[cur.THIRD] = cur.first;
        else
            continue;
        for(int i = 0; i < edges[cur.THIRD].size(); i++){
            ll cost = edges[cur.THIRD][i].first,
            where = edges[cur.THIRD][i].second;
            if(arr[where].first == INT_MAX)
                pq.push(iii(ii(cur.FIRST + cost, cur.THIRD), where));
        }
    }
}
```

DSU

```cpp
#define MAXN 10
#define MOD 1000000007

struct DSU {
    int bel[MAXN];
    vi s[MAXN];

    void reset() {
        FOR(i, 0, MAXN)
            bel[i] = 0, s[i].clear();
    }

    void unir(int a, int b) {
        if (a > b)  swap(a, b);

        FOR(i, 0, s[a].size()) {
            s[b].pb(s[a][i]);
            bel[a] = b;
        }
        s[a].clear();
    }

    int belongs(int x) {
        return bel[x];
    }
};

struct DSU2 {
    int numSets = 0;
    int setSize[MAXN];
    int parent[MAXN];
    int rank[MAXN];

    UnionFind(int n) {
        numSets = n;
        FOR(i, 0, n)    parent[i] = i, setSize[i] = rank[i] = 0;
    }

    void make_set(int i) {
        parent[i] = i;
        rank[i] = 0;
    }

    int find_set(int i) {
        if (i != parent[i])
            parent[i] = find_set(parent[i]);
        return parent[i];
    }

    void unionSet(int i, int j) {
        if (!isSameSet(i, j)) {
            numSets--;
            int x = find_set(i), y = find_set(j);

            if (rank[x] > rank[y]) {
                parent[y] = x;
                setSize[x] += setSize[y];
            }
            else {
                parent[x] = y;
                setSize[y] += setSize[x];

                if (rank[x] == rank[y])     rank[y]++;
            }
        }
    }

    bool isSameSet(int i, int j) {
        return find_set(i) == find_set(j);
    }
```

```
};
```

Find Cycles

```
#define maxN 100005

vi edges[maxN];
bool cur[maxN], visit[maxN];
stack<int> ans;

int findCycle(int n) {
    if (cur[n])     return n;
    if (visit[n])   return -1;

    cur[n] = true;
    visit[n] = true;

    FOR(i, 0, edges[n].size()) {
        if(ans.size())  break;

        int v = findCycle(edges[n][i]);
        if (v != -1) {
            cur[n] = false;
            ans.push(n);
            if(v == n) {
                return -1;
            }
            return v;
        }
    }
    cur[n] = false;
    return -1;
}

int main() {
    int n, m, a, b;
    while (cin >> n >> m) {
        memset(cur, false, sizeof(cur));
        memset(visit, false, sizeof(visit));

        FOR(i, 0, m) {
            cin >> a >> b;
            edges[a].pb(b);
        }

        FOR(i, 0, n) {
            findCycle(i);
        }

        if(! ans.empty()) {
            cout << "YES\n";
            while(! ans.empty()) {
                int val = ans.top();
                ans.pop();
                cout << val << " \n"[ans.empty()];
            }
        }
        else {
            cout << "NO\n";
        }
    }
    return 0;
}
```

Floyd-Warshall

```
#define MAXN 10005
#define MOD 1000000007

int graph[MAXN][MAXN];
int dist[MAXN][MAXN];

void floydWarshall(int n) { // O(n^3)
    FOR(i, 0, n) FOR(j, 0, n) dist[i][j] = graph[i][j];

    FOR(k, 0, n) {
        FOR(i, 0, n) {
            FOR(j, 0, n) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}

int main() {
    FOR(i, 0, MAXN) FOR(j, 0, MAXN) dist[i][j] = INF;

    int n, m;
    cin >> n >> m;
    FOR(i, 0, m) {
        cin >> a >> b >> c;
        graph[a][b] = c;
    }

    floydWarshall(n);
    return 0;
}
```

Kruskal

```cpp
#define MAXN 10005
#define MAXM 10005
#define MOD 1000000007

struct Edge {
    int from, to, w;
    Edge() {}
    Edge(int a, int b, int c) : from(a), to(b), w(c) { }
    bool operator <(Edge const& e) const {
        return w < e.w;
    }
};

struct UnionFind {
    int numSets = 0;
    int setSize[MAXN];
    int parent[MAXN];
    int rank[MAXN];

    UnionFind(int n) {
        numSets = n;
        FOR(i, 0, n)    parent[i] = i;
    }

    void make_set(int i) {
        parent[i] = i;
        rank[i] = 0;
    }

    int find_set(int i) {
        if (i != parent[i])
            parent[i] = find_set(parent[i]);
        return parent[i];
    }

    void unionSet(int i, int j) {
        if (!isSameSet(i, j)) {
            numSets--;
            int x = find_set(i), y = find_set(j);

            if (rank[x] > rank[y]) {
                parent[y] = x;
                setSize[x] += setSize[y];
            }
            else {
                parent[x] = y;
                setSize[y] += setSize[x];

                if (rank[x] == rank[y])     rank[y]++;
            }
        }
    }

    bool isSameSet(int i, int j) {
        return find_set(i) == find_set(j);
    }
};

vector<Edge> v;
vector<Edge> tree;

int kruskal(int n, int m) { // O(ELogE + ELogV)
    sort(v.begin(), v.end());

    int mst_w = 0;
    UnionFind UF(n);

    FOR(i, 0, m) {
        Edge e = v[i];

        if (! UF.isSameSet(e.from, e.to)) {
            mst_w += e.w;
```

```
                UF.unionSet(e.from, e.to);
                tree.pb(Edge(e.from, e.to, e.w));
            }
        }

        return mst_w;
    }

    int main() {
        int n, m;
        int a, b, c;
        cin >> n >> m;
        FOR(i, 0, m) {
            cin >> a >> b >> c;
            v.pb(Edge(a, b, c));
        }

        cout << kruskal(n, m) << endl;

        return 0;
    }
```

Longest Path in DAG

```cpp
#define MAXN 10
#define MOD 1000000007

vii graph[MAXN];
stack<int> s;
bool v[MAXN];
int dist[MAXN], n, m;

void topologicalSortUtil(int act) {
    v[act] = 1;

    FOR(i, 0, graph[act].size())
        if (!v[graph[act][i].second])  topologicalSortUtil(graph[act][i].second);

    s.push(act);
}

void longestPath(int source) {
    memset(v, 0, sizeof(v));

    FOR(i, 0, n)
        if (!v[i])  topologicalSortUtil(i);

    fill(dist, dist + n, -INF);
    dist[source] = 0;

    while (! s.empty()) {
        int u = s.top();
        s.pop();

        if (dist[u] != -INF) {
            FOR(i, 0, graph[u].size()) {
                dist[graph[u][i].second] = max(dist[graph[u][i].second], dist[u] + graph[u][i].first);
            }
        }
    }

    FOR(i, 0, n)
        if (dist[i] == -INF)    cout << "INF" << endl;
        else                    cout << dist[i] << endl;
}


int main() { _
    int a, b, c;
    while(cin >> n >> m) {
        FOR(i, 0, m) {
            cin >> a >> b >> c;
            graph[a].pb(ii(c, b));
        }

        int source;
        cin >> source;
        longestPath(source);
    }

    return 0;
}
```

Maximum Diameter in a Graph

```
#define MAXN 10
#define MOD 1000000007

// v es visitados localmente.
// v2 es todos los que ya he visitado (para grafos no totalmente conectados)
bool v[MAXN], v2[MAXN];
vi edges[MAXN];

ii bfs(int x, bool b) {
    queue<ii> q;
    q.push(ii(x, 0));

    v[x] = !b;
    v2[x] |= !b;
    ii last;
    while(! q.empty()) {
        last = q.front();
        q.pop();

        FOR(i, 0, edges[last.first].size()) {
            if (v[edges[last.first][i]] == b) {
                q.push(ii(edges[last.first][i], last.second + 1));
                v[edges[last.first][i]] = !b;
                v2[edges[last.first][i]] |= !b;
            }
        }
    }

    return last;
}

int maxDiameter(int x) {
    ii ret = bfs(x, false);
    return bfs(ret.first, true).second;
}
```

Maximum Bipartite Matching

```
#define MAXN 10005
#define MOD 1000000007

int n, m;
int matchL[MAXN], matchR[MAXN];
vi edge[MAXN];
bool v[MAXN];

bool dfs(int from) {
    if (v[from])   return 0;
    v[from] = 1;

    FOR(i, 0, edge[from].size()) {
        int to = edge[from][i];

        if (matchR[to] == -1 || dfs(matchR[to])) {
            matchL[from] = to;
            matchR[to] = from;
            return 1;
        }
    }
    return 0;
}

ll MBM() {
    ll ans = 0;
    bool b = 1;

    memset(matchL, -1, sizeof(matchL));
    memset(matchR, -1, sizeof(matchR));

    while (b) {
        b = 0;
        memset(v, 0, sizeof(v));
        FOR(i, 0, n) {
            if (matchL[i] == -1 && dfs(i)) {
                ans ++;
                b = 1;
            }
        }
    }
    return ans;
}

int main() { _
    int a, b;
    cin >> n >> m;
    FOR(i, 0, m) {
        cin >> a >> b;
        edge[a].pb(b);
    }
    cout << MBM() << endl;

    return 0;
}
```

Strongly Connected Components

```
#define MAXN 1000
#define MOD 1000000007

int n, m;

vi edge[MAXN];
int disc[MAXN], low[MAXN], belongs[MAXN];
bool v[MAXN];
stack<int> st;
int ttime, comp;

void SCCUtil(int u) {
    disc[u] = low[u] = ttime ++;
    st.push(u);
    v[u] = true;

    FOR(i, 0, edge[u].size()) {
        int to = edge[u][i];

        if (disc[to] == -1)
            SCCUtil(to);

        if (v[to])
            low[u]  = min(low[u], low[to]);
    }

    if (low[u] == disc[u]) {
        comp ++;
        while (1) {
            int t = st.top();
            st.pop();
            belongs[t] = comp;
            v[t] = false;

            if (t == u) break;
        }
    }
}

void SSC() {
    ttime = comp = 0;
    memset(disc, -1, sizeof(disc));
    memset(low, -1, sizeof(low));
    memset(v, 0, sizeof(v));

    FOR(i, 0, n)
        if (disc[i] == -1)
            SCCUtil(i);
}
```

Topological Sort

```
#define MAXN 10005
#define MOD 1000000007

vi edges[MAXN];
stack<int> s;
bool v[MAXN];

void topologicalSortUtil(int act) {
    v[act] = 1;

    FOR(i, 0, edges[act].size())
        if (!v[edges[act][i]])  topologicalSortUtil(edges[act][i]);

    s.push(act);
}

void topologicalSort(int n) {
    memset(v, 0, sizeof(v));

    FOR(i, 0, n)
        if (!v[i])  topologicalSortUtil(i);

    while (!s.empty()) {
        int a = s.top();
        s.pop();
        cout << a << " \n"[s.empty()];
    }
}

int main() {
    int n, m;
    int a, b;
    while (cin >> n >> m) {
        FOR(i, 0, MAXN)     edges[i].clear();

        FOR(i, 0, m) {
            cin >> a >> b;
            edges[a].pb(b);
        }

        topologicalSort(n);
    }

    return 0;
}
```

5. Number Theory

Fermat Little's theorem

```
ll mulmod(ll a, ll b, ll c) { // returns (a * b) % c, and minimize overflow
    return (ll)((__int128)(a) * (b) % c);
}

ll fastPow(ll x, ll n, ll c) { // returns (a ** b) % c, and minimize overflow
    ll ret = 1;
    while (n) {
        if (n & 1) ret = mulmod(ret, x, c);
        x = mulmod(x, x, c);
        n >>= 1;
    }
    return ret;
}



/** return modular multiplicative of: a mod p, assuming p is prime **/
ll modInverse(ll a, ll p) {
    return fastPow(a, p - 2, p);
}

/** return C(n,k) mod p, assuming p is prime **/
ll modBinomial(ll n, ll k) {
    ll numerator = 1; // n*(n-1)* ... * (n-k+1)
    FOR(i, 0, k)
        numerator = (numerator * (n - i)) % MOD;

    ll denominator = 1; // k!
    FOR (i, 1, k+1)
        denominator = (denominator * i) % MOD;

    ll res = modInverse(denominator, MOD);
    res = (res * numerator) % MOD;
    return res;
}

ll extendedGCD(ll a, ll b, ll &inva, ll &invb) {
    if (b) {
        ll g = extendedGCD(b, a%b, invb, inva);
        invb = invb - a / b*inva;
        return g;
    }
    inva = 1, invb = 0;
    return a;
}

int main() {
    ll inva=0, invb=0;
    int c = extendedGCD(62424, 13, inva, invb);
    cout << modInverse(62424, 13) << endl;
    cout << (inva + 7) % 7 << " " << (invb + 62424) % 13 << " " << c << endl;
    return 0;
}
```

Fibonacci

```cpp
const long double PHI ((1.0 + sqrt(5.0)) / 2.0);

ll O1(ll n) {
    return (ll)(floor(pow(PHI, n) / sqrt(5.0) + 0.5));
}

void mult(ll F[2][2], ll M[2][2]) {
    ll x =  (F[0][0] * M[0][0]) % MOD + (F[0][1] * M[1][0]) % MOD;
    ll y =  (F[0][0] * M[0][1]) % MOD + (F[0][1] * M[1][1]) % MOD;
    ll z =  (F[1][0] * M[0][0]) % MOD + (F[1][1] * M[1][0]) % MOD;
    ll w =  (F[1][0] * M[0][1]) % MOD + (F[1][1] * M[1][1]) % MOD;
    F[0][0] = x % MOD;
    F[0][1] = y % MOD;
    F[1][0] = z % MOD;
    F[1][1] = w % MOD;
}

void power(ll F[2][2], ll n) {
    if (n == 0 || n == 1) return;

    ll M[2][2] = {{1, 1}, {1, 0}};
    power(F, n / 2);
    mult(F, F);

    if (n % 2)
        mult(F, M);
}

ll OlnMat(ll n) {
    if (n == 0) return 0;

    ll F[2][2] = {{1,1},{1,0}};
    power(F, n - 1);

    return F[0][0];
}

/** Suma de Los primeros n numeros de fibo **/
ll fiboSuma(ll n) {
    return (MOD + OlnMat(n + 2) - 1) % MOD;
}

ll On(ll n) {
    if (n == 1) return 0;

    ll a = 0, b = 1, c = 1;

    FOR(i, 2, n) {
        c = b + a;
        a = b;
        b = c;
    }
    return c;
}

ll fibo(ll n) {
    if (n < 72)     return O1(n);
    if (n < 150)    return On(n);

    return OlnMat(n);
}

int main(){
    ll n;
    cin >> n;
    cout << fibo(n) << endl;
```

Prime Numbers

```cpp
int pc = 0, f[MAXN], primes[MAXN / 10];


// ******************************** IS PRIME() ****************************************
//Para TODOS los primos

ll mulmod2(ll a, ll b, ll c) { // returns (a * b) % c, and minimize overflow
    ll x = 0, y = a % c;
    while (b) {
        if (b & 1) x = (x + y) % c;
        y = (y << 1) % c;
        b >>= 1;
    }
    return x % c;
}

ll mulmod(ll a, ll b, ll c) {
    return (ll)((__int128)(a) * b % c);
}

ll fastPow(ll x, ll n, ll c) { // returns (a ** b) % c, and minimize overflow
    ll ret = 1;
    while (n) {
        if (n & 1) ret = mulmod(ret, x, c);
        x = mulmod(x, x, c);
        n >>= 1;
    }
    return ret;
}

// Miller-Rabin primality test
bool millerRabin(ll n) {
    ll d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        s++;
        d >>= 1;
    }
    // It's garanteed that these values will work for any number smaller than 2^64
    int a[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
    FOR(i, 0, 12)   if(n == a[i])   return true;

    FOR(i, 0, 12) {
        bool comp = fastPow(a[i], d, n) != 1;
        if(comp) FOR(j, 0, s) {
            ll fp = fastPow(a[i], (1LL << (ll)j)*d, n);
            if (fp == n - 1) {
                comp = false;
                break;
            }
        }
        if(comp) return false;
    }
    return true;
}

// Miller-Rabin primality test
ll rN(){
    return maxR * rand() + rand();
}

bool miller(ll n, ll d){
    ll a = 2 + rN() % (n - 4);
    ll x = fastPow(a, d, n);
    if(x == 1 || x == n - 1)
        return true;
    while(d < n - 1){
        x = __int128(x) * x % n;
        if(x == 1)
            return false;
```

```cpp
        if(x == n - 1)
            return true;
        d <<= 1;
    }
    return false;
}

bool isPrime(ll n, int k){
    if(n == 3 || n == 2)
        return true;
    if(n == 1 || n % 2 == 0)
        return false;
    ll d = n - 1;
    while(d % 2 == 0)
        d >>= 1;
    while(k--){
        if(!miller(n, d))
            return false;
    }
    return true;
}
// ****************************************************************************************


// ********************************
```

GET PRIME FACTORS()

```cpp
// ********************************
/**
 Devuele los factores primos de un numero. No olvidar hacer unique para quitar repetidos
 int tamano = unique(primeFactors.begin(), primeFactors.end()) - primeFactors.begin();
**/

ll A, B;
vl primeFactors;

ll funcRand(ll x, ll n) {
    return (mulmod(x, (x + A), n) + B) % n;
}

void pollardRho(ll n) {
    if (n == 1) return;
    if (millerRabin(n)) {
        primeFactors.pb(n);
        return;
    }
    ll d = n, x, y;
    while(d == n) {
        d = 1;
        x = y = 2;
        A = 2 + rand() % (n - 2);
        B = 2 + rand() % (n - 2);

        while (d == 1) {
            x = funcRand(x, n);
            y = funcRand(funcRand(y, n), n);
            d = __gcd(abs(x - y), n);
        }
    }

    if (n / d != d)
        pollardRho(n / d);
    pollardRho(d);
}
// ****************************************************************************************

// ****************************
```

GET SMALLEST DIV PRIME()

```cpp
// ******************************
// Obtener f[t], el primo mas chico que divide a f[t]

int divPrime[MAXN];

void getSmallestDivPrime() {
    FOR(i, 2, MAXN) {
        if (divPrime[i] == -1) {
            for (int j = 1; i*j < MAXN; j++) if (divPrime[i*j] == -1) divPrime[i*j] = i;
        }
```

```cpp
    }
}
// *************************************************************************************************


// ********************************   GET DIVISORS()   *************************************
vll factorsOfN;
vl divisors;

// Usar cuando se necesita sacar muchos numeros
void getFactorsOfN2(int n) {
    int t = n;
    int last = -1;
    while (divPrime[t] != -1) {
        if (divPrime[t] == last)    factorsOfN[factorsOfN.size() - 1].second ++;
        else                        factorsOfN.pb(ii(divPrime[t], 1));
        last = divPrime[t];
        t /= divPrime[t];
    }
}

// Usar cuando es numeros muy grandes
void getFactorsOfN(ll n) {
    pollardRho(n);
    sort(primeFactors.begin(), primeFactors.end());
    int ss = unique(primeFactors.begin(), primeFactors.end()) - primeFactors.begin();
    int s = 0;
    ll num = n;
    FOR(i, 0, ss) {
        bool b = false;
        while (num % primeFactors[i] == 0) {
            if (b)  factorsOfN[s].second ++;
            else    factorsOfN.pb(pll(primeFactors[i], 1)), b = 1;

            num /= primeFactors[i];
        }
        s ++;
    }
}

void getDivisorsOfN(ll n) {
    primeFactors.clear();
    factorsOfN.clear();
    divisors.clear();

    getFactorsOfN(n);
    divisors.push_back(1);

    FOR(i, 0, factorsOfN.size()) {
        int s = divisors.size(), num = 1;

        FOR(j, 0, factorsOfN[i].second) {
            num *= factorsOfN[i].first;
            FOR(k, 0, s) divisors.push_back(divisors[k] * num);
        }
    }
    sort(divisors.begin(), divisors.end());
    FOR(i, 0, divisors.size())  cout << divisors[i] << endl;
}
// *************************************************************************************************


// ********************************   Criba de Eratosthenes   *********************************
void Eratosthenes() {
    bool isPrime[MAXN];
    memset(isPrime, 0, sizeof(isPrime));
    FOR(i, 2, MAXN) {
        if (!isPrime[i]) {
            primes[pc ++] = i;
            for (int j = 2; j*i < MAXN; j++) {
                isPrime[i*j] = true;
            }
        }
```

```
    }
}
// **************************************************************************************


// ******************************** Criba de Eratosthenes O(N) ****************************
void EratosthenesON(){
    FOR(i, 2, MAXN) {
        if (! f[i]) primes[pc++] = i, f[i] = i;
        for (int j = 0; j < pc && 1LL*i*primes[j] < MAXN && primes[j] <= f[i]; j++)
            f[i*primes[j]] = primes[j];
    }
}
// **************************************************************************************

// ******************************** Criba de Atkin ****************************************
void Atkin() {
    ll sqrtArraySize = sqrt(MAXN);
    ll n;

    bool isPrime[MAXN];
    memset(isPrime, false, sizeof(isPrime));

    ll pp[MAXN];
    FOR(i, 0, sqrtArraySize + 5)    pp[i] = i * i;

    FOR(i, 1, sqrtArraySize + 1) {
        FOR(j, 1, sqrtArraySize + 1) {
            n = 4 * pp[i] + pp[j];

            if (n <= MAXN && (n % 12 == 1 || n % 12 == 5))
                isPrime[n] = !isPrime[n];

            n = 3 * pp[i] + pp[j];

            if (n <= MAXN && (n % 12 == 7))
                isPrime[n] = !isPrime[n];

            if (i > j) {
                n = 3 * pp[i] - pp[j];
                if(n <= MAXN && n % 12 == 11)
                    isPrime[n] = !isPrime[n];
            }
        }
    }

    FOR(i, 5, sqrtArraySize + 1) {
        if (isPrime[i]) {
            ll omit = pp[i];

            for (ll j = omit; j <= MAXN; j += omit) {
                isPrime[j] = false;
            }
        }
    }

    if (MAXN >= 2) {
        primes[pc ++] = 2;
        if (MAXN >= 3) {
            primes[pc ++] = 3;
        }
    }

    FOR(i, 2, MAXN) {
        if (isPrime[i]) {
            primes[pc ++] = i;
        }
    }
}
// **************************************************************************************

// ******************************** Euler Totem *********************************
```

```
// REGRESA PINCHE CANTIDAD DE COPRIMOS DE N
// Para sacar la puta suma de coprimos es n*phi(n) / 2
ll eulerTotient(ll n) {
    factorsOfN.clear();
    getFactorsOfN(n);
    FOR(i, 0, factorsOfN.size())
        n = n / factorsOfN[i].first * (factorsOfN[i].first - 1);
    return n;
}
```

Triangle of Mahonian

```
#define MAXN 35
#define MAXM 5005

ll arr[MAXN][MAXM];

int main() {
    memset(arr, 0, sizeof(arr));
    arr[0][0] = arr[1][0] = 1;

    FOR(i, 0, MAXN) {
        for(int m = 0; m < MAXM - 5 && arr[i][m] != 0; m ++) {
            while(m < 5000 && ) {
                FOR(j, 0, i+1) {
                    arr[i+1][j+m] += arr[i][m];
                }
            }
        }
    }

    return 0;
}
```

6. Segment Tree

Lazy Propagation

```
const int maxN = 1e5, MOD = 1E9 + 7;
int HEIGHT, n;
ll d[maxN], t[maxN << 1];

void apply(int p, ll value) {
    t[p] = value;
    if (p < n) d[p] = value;
}

void build(int p) {
    while (p > 1) p >>= 1, t[p] = max(max(t[p<<1], t[p<<1|1]), d[p]);
}

void push(int p) {
    for (int s = HEIGHT; s > 0; --s) {
        int i = p >> s;
        if (d[i] != 0) {
            apply(i<<1, d[i]);
            apply(i<<1|1, d[i]);
            d[i] = 0;
        }
    }
}

void change(int l, int r, ll value) {
    l += n, r += n;
    int l0 = l, r0 = r;
    for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) apply(l++, value);
        if (r&1) apply(--r, value);
    }
    build(l0);
    build(r0 - 1);
}

ll query(int l, int r) {
    l += n, r += n;
    push(l);
    push(r - 1);
    ll res = LLONG_MIN;
    for (; l < r; l >>= 1, r >>= 1) {
        if (l&1) res = max(res, t[l++]);
        if (r&1) res = max(t[--r], res);
    }
    return res;
}
```

Persistent Segment Tree

```
const int maxN = 1e6;

struct Node{
    Node *l, *r;
    int count;
    Node(int count, Node *l, Node *r):count(count), l(l), r(r){}
    Node * insert(int, int, int);
};
Node * segTrees[maxN + 1];

Node * Node::insert(int left, int right, int pos){
    if(pos < left || pos >= right)
        return this;
    if(left + 1 == right)
        return new Node(count + 1, segTrees[0], segTrees[0]);
    int m = (left + right) >> 1;
    return new Node(count + 1, l -> insert(left, m, pos), r -> insert(m, right, pos));
}
int solve(Node * last, Node * first, int left, int right, int k){
    if(left + 1 == right)
        return left;
    int m = (left + right) >> 1, dif = last -> l -> count - first -> l -> count;
    if(dif < k)
        return solve(last -> r, first -> r, m, right, k - dif);
    return solve(last -> l, first -> l, left, m, k);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, q, x, y, k, arr[maxN], vals[maxN];
    segTrees[0] = new Node(0, NULL, NULL);
    segTrees[0] -> l = segTrees[0] -> r = segTrees[0];
    while(cin >> n >> q){
        map<int, int> myMap;
        for(int i = 0; i < n; i++){
            cin >> arr[i];
            myMap[arr[i]];
        }
        int valSize = 0;
        for(map<int, int>::iterator it = myMap.begin(); it != myMap.end(); it++){
            it -> second = valSize;
            vals[valSize++] = it -> first;
        }
        for(int i = 1; i <= n; i++)
            segTrees[i] = segTrees[i - 1] -> insert(0, valSize, myMap[arr[i - 1]]);
        while(q--){
            cin >> x >> y >> k;
            cout << vals[solve(segTrees[y], segTrees[x - 1], 0, valSize, k)] << '\n';
        }
    }
}
```

7. Treap

```
const int maxN = 1e5 + 3, MOD = 1e9 + 7, AL = 10;

ll fc[maxN][2];

ll perm(int *bu){
    ll den = 1, tot = 0, imp = 0;
    for(int i = 0; i < AL; i++){
        //cout << bu[i] << ' ';
        imp += bu[i] & 1;
        den = den * fc[bu[i] >> 1][1] % MOD;
        tot += bu[i] >> 1;
    }
    //cout  << '\n';
    if(imp > 1) return 0;
    return fc[tot][0] * den % MOD;
}

struct Node{
    int ca, pri, bu[AL], cnt;
    Node *l, *r;
    bool rev;
    Node(int ca): ca(ca), l(NULL), r(NULL), rev(false){
        memset(bu, 0, sizeof(bu)), pri = (rand() << 15) + rand();
    }
};
typedef Node* pnode;

int cnt(pnode nd){
    return nd? nd->cnt: 0;
}

void upd(pnode nd){
    if(!nd) return;
    nd->cnt = 1 + cnt(nd->l) + cnt(nd->r);

    for(int i = 0; i < AL; i++)
        nd->bu[i] = (nd->l? nd->l->bu[i]: 0) + (nd->r? nd->r->bu[i]: 0);
    nd->bu[nd->ca]++;
}

void apply(pnode nd){
    if(nd) nd->rev ^= 1;
}

void push(pnode nd){
    if(nd && nd->rev)
        swap(nd->l, nd->r), apply(nd-> l), apply(nd -> r), nd->rev = 0;
}

void split(pnode t, pnode &l, pnode &r, int key, int add){
    if(!t)
        return void(l = r = NULL);
    push(t);
    int mykey = add + cnt(t->l) + 1;
    if(mykey <= key)
        split(t->r, t->r, r, key, mykey), l = t;
    else
        split(t->l, l, t->l, key, add), r = t;
    upd(t);
}

void merge(pnode &t, pnode l, pnode r){
    push(l), push(r);
    if(!l || !r)
```

```
        t = l? l: r;
    else if(l->pri > r->pri)
        merge(l->r, l->r, r), t = l;
    else
        merge(r->l, l, r->l), t = r;
    upd(t);
}

void insert(pnode &t, pnode it){
    if(!t)
        t = it;
    else if(it->pri > t->pri)
        it->l = t, t = it;
    else
        insert(t->r, it);
    upd(t);
}

int n, m;
string st;

ll fastPow(ll a, ll b){
    ll ret = 1;
    while(b){
        if(b & 1)
            ret = ret * a % MOD;
        a = a * a % MOD, b >>= 1;
    }
    return ret;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    srand(8000000);

    fc[0][0] = fc[0][1] = 1;
    for(ll i = 1; i < maxN; i++)
        fc[i][0] = fc[i - 1][0] * i % MOD, fc[i][1] = fastPow(fc[i][0], MOD - 2);

    cin >> n >> m >> st;

    pnode head = NULL;
    for(char c: st) insert(head, new Node(c - 'a'));

    while(m--){
        int ty, l, r;
        cin >> ty >> l >> r;
        pnode two, thr;
        split(head, head, two, l - 1, 0);
        split(two, two, thr, r - l + 1, 0);
        if(ty == 1)
            apply(two);
        else
            cout << perm(two->bu) << '\n';
        merge(head, head, two);
        merge(head, head, thr);
    }

}
```

8. Strings

Aho-Corasick

```
const int maxN = 1e3 + 3, MOD = 1e9 + 9, LG = 21;
//NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom
const int dr[8][2] = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}, {1, 1}, {-1, -1}, {-1, 1}, {1, -1}};
const string chr = "ECAGDHBF";

string mat[maxN], dic[maxN], ans[maxN];
int n, m, w;
bool v[maxN][maxN];

vector<ii> ini[8];

bool ins(int x, int y){
    return x >= 0 && x < n && y >= 0 && y < m;
}

void findInits(int d, int r, int c){
    int x = r - dr[d][0], y = c - dr[d][1];

    v[r][c] = 1;

    if(!ins(x, y)) ini[d].push_back(ii(r, c));
    else if(!v[x][y]) findInits(d, x, y);
}

int tr[maxN * maxN][26], dp[maxN * maxN], lf[maxN * maxN], tc, fWord[maxN * maxN], lastInd[maxN], sz[maxN];

int ins(int par, char c){
    if(tr[par][c - 'A']) return tr[par][c - 'A'];
    int k, ret = tr[par][c - 'A'] = tc++;

    for(k = dp[par]; k && !tr[k][c - 'A']; k = dp[k]);

    if(k != par) k = tr[k][c - 'A'];

    if(fWord[k] != -1)
        lf[ret] = k;
    else
        lf[ret] = lf[k];

    dp[ret] = k;

    return ret;
}

string bld(int sz, int r, int c, int u){
    r -= dr[u][0] * sz, c -= dr[u][1] * sz;
    return to_string(r) + " " + to_string(c) + " " + chr[u];
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    int t;
    cin >> t;
    while(t--){
        cin >> n >> m >> w;

        queue<int> q;
```

```
    for(int i = 0; i < n; i++) cin >> mat[i];
    for(int i = 0; i < w; i++) {
        cin >> dic[i];
        sz[i] = int(dic[i].size());
        reverse(dic[i].begin(), dic[i].end());
        q.push(i);
    }

    for(int u = 0; u < 8; u++){
        memset(v, 0, sizeof(v)), ini[u].clear();
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                if(!v[i][j])
                    findInits(u, i, j);
    }

    memset(tr, 0, tc * sizeof(tr[0])), memset(fWord, -1, sizeof(fWord));
    memset(lastInd, 0, sizeof(lastInd));
    tc = 1;

    while(q.size()){
        int c = q.front(); q.pop();

        lastInd[c] = ins(lastInd[c], dic[c].back());

        //cout << c << ' ' << dic[c].back() << '\n';

        dic[c].pop_back();

        if(!dic[c].empty())
            q.push(c);
        else
            fWord[lastInd[c]] = c;
    }

    //for(int i = 0; i < 26; i++) cout << tr[0][i] << ' '; cout << '\n';
    map<int, string> myMap;

    for(int u = 0; u < 8; u++){
                for(ii el: ini[u]){
                    int c = 0;

                    for(; ins(el.fi, el.se); el.fi += dr[u][0], el.se += dr[u][1]){
                        while(c && !tr[c][mat[el.fi][el.se] - 'A'])
                            c = dp[c];
                        c = tr[c][mat[el.fi][el.se] - 'A'];

                        //cout << c << '\n';

                        for(int i = c; i; i = lf[i])
                            if(fWord[i] != -1)
                                myMap[i] = bld(sz[fWord[i]]-1, el.fi, el.se, u);
                    }
                }
            }

            for(int i = 0; i < w; i++)
                cout << myMap[lastInd[i]] << '\n';
            cout << '\n';
        }
    }
```

KMP

```
int lps[MAXN];
vi v;

void computeLPSArray(string pat, int M) {
    int len = 0;
    int i = 1;
    lps[0] = 0;
    while (i < M) {
        if (pat[i] == pat[len])
            lps[i ++] = ++ len;
        else {
            if (len != 0)   len = lps[len - 1];
            else            lps[i ++] = 0;
        }
    }
}
void KMPSearch(string pat, string txt) {
    int M = pat.length();
    int N = txt.length();
    int i = 0;
    int j = 0;

    computeLPSArray(pat, M);

    while (i < N) {
        if (pat[j] == txt[i]) {
            j ++;
            i ++;
        }

        if (j == M) {
            v.pb(i - j);
            j = lps[j - 1];
        }
        else if (i < N && pat[j] != txt[i]) {
            if (j != 0)     j = lps[j - 1];
            else            i = i + 1;
        }
    }
}

int main() {
    string txt = "ABABDABACDABABCABAB";
    string pat = "AB";
    KMPSearch(pat, txt);

    FOR(i, 0, v.size())
        cout << v[i] << " \n"[i==v.size()-1];

    return 0;
}
```

Suffix Array

```
typedef pair<int, int> ii;
typedef pair<ii, int> iii;
typedef long long ll;
typedef vector<ll> vd;
typedef vector<vd> Matrix;
const int maxN = 1e6;

iii sa[maxN], aux[maxN];
int inv[maxN], lcp[maxN + 1], values[maxN + 1], n;
string st;

#define a(i) (fir? sa[i].FIRST: sa[i].SECOND)

void radiixSort(bool fir){
    memset(values, 0, sizeof(values));
    for(int i = 0; i < n; i++)
        values[a(i) + 1]++;
    for(int i = 1; i <= n; i++)
        values[i] += values[i - 1];
    for(int i = n - 1; i >= 0; i--)
        aux[--values[a(i) + 1]] = sa[i];
    for(int i = 0; i < n; i++)
        sa[i] = aux[i];
}

void createSuffixArray(){
    for(int i = 0; i < n; i++)
        sa[i] = iii(ii(st[i], 0), i);
    sort(sa, sa + n);
    for(int cnt = 1; cnt <= n; cnt <<= 1){
        for(int i = 0, j = 0; i < n; i = j)
            for(ii cur = sa[i].first; j < n && sa[j].first == cur; j++)
                sa[j].FIRST = inv[sa[j].THIRD] = i;
        for(int i = 0; i < n; i++)
            sa[i].SECOND = (sa[i].THIRD + cnt < n)? sa[inv[sa[i].THIRD + cnt]].FIRST: -1;
        radiixSort(false);
        radiixSort(true);
    }
}

void createLCPArray(){
    for(int i = 0; i < n; i++)
        inv[sa[i].THIRD] = i;
    for(int i = 0, k = 0; i < n; i++, k? k--: k){
        if(inv[i] + 1 == n){
            k = lcp[n - 1] = 0;
            continue;
        }
        int cur = sa[inv[i]].THIRD, next = sa[inv[i] + 1].THIRD;
        for(; max(cur, next) + k < n && st[cur + k] == st[next + k]; k++);
        lcp[inv[i] + 1] = k;
    }
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n >> st;
    createSuffixArray();
    createLCPArray();
}
```

Z Function

```cpp
// Devuelve el arreglo Z
vector<int> z_function(string &s){
    int L = 0, R = 0,  n = s.length();
    vector<int> z(n);
     for(int i = 1; i < n; i++){
        if(i <= R)
            z[i] = min(z[i-L], R - i + 1);
        while(i + z[i] < n && s[i + z[i]] == s[z[i]])
            z[i]++;
        if(i + z[i] - 1 > R){
            L = i;
            R = i + z[i] - 1;
        }
    }
    return z;
}

int main(){ io
    string A, B;
    cin >> A >> B;
    string z_Arg = B + '$' + A;
    vector<int> z = z_function(z_Arg);
}
```

9. Flows

Dinic

```
/*
    O(V^2E)
    On unit networks O(Esqrt(V))
    Unit network = edges with capacity = 1
*/
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.push_back(FlowEdge(v, u, cap));
        edges.push_back(FlowEdge(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;
    }
};
```

Edmons-Karp

```
/*
Edmonds-Karp algorithm O(VE^2)
Max flow
The matrix capacity stores the capacity for every pair of vertices.
adj is the adjacency list of the undirected graph,
since we have also to use the reversed of directed edges when we are looking for augmenting paths.

The function maxflow will return the value of the maximal flow.
During the algorithm the matrix capacity will actually store the residual capacity of the network.
The value of the flow in each edge will actually no stored,
but it is easy to extent the implementation - by using an additional matrix - to also store the flow and return it.
*/

int n;
vector<vector<int>> capacity;
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;
    q.push({s, INF});

    while (!q.empty()) {
        int cur = q.front().first;
        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(n);
    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;
}
```

Hungarian

```
const int maxN = 50 + 9, MOD = 1e9 + 7, INF = 2e7;

int n, m, a[maxN][maxN];

/*

 HUNGARIAN ALGORITHM

 -efficiency O(n ^ 2 * m)
 -input matrix a[1..n][1..m] ONE BASED!
 -n <= m
 -need to fix an INF value!
 -it can handle negative values :O
 -returns MINIMUM cost (just *= -1 to change it to maximum cost)

 Such an incredible implementation demands credit for the author. Thanks again, Andrei Lopatin, we remember your genious every time we look at this beautiful piece of code!

 "This implementation was actually developed by Andrei Lopatin several years ago. It is distinguished by an amazing brevity: the whole algorithm is placed in 30 lines of code ."

 */

int hungarian(){
    vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
    for (int i=1; i<=n; ++i) {
        p[0] = i;
        int j0 = 0;
        vector<int> minv (m+1, INF);
        vector<char> used (m+1, false);
        do {
            used[j0] = true;
            int i0 = p[j0],  delta = INF,  j1;
            for (int j=1; j<=m; ++j)
                if (!used[j]) {
                    int cur = a[i0][j]-u[i0]-v[j];
                    if (cur < minv[j])
                        minv[j] = cur,  way[j] = j0;
                    if (minv[j] < delta)
                        delta = minv[j],  j1 = j;
                }
            for (int j=0; j<=m; ++j)
                if (used[j])
                    u[p[j]] += delta,  v[j] -= delta;
                else
                    minv[j] -= delta;
            j0 = j1;
        } while (p[j0] != 0);
        do {
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }

    /*
     Recovery of the answer in a more usual form, i.e. finding for each row the i = 1 \ ldots nnumber of the column selected in it ans [i]is done as follows:*/
    vector<int> ans (n+1);
    for (int j=1; j<=m; ++j)
        ans[p[j]] = j;

```

```
    /*

     The value of the matching found can simply be taken as the potential of the zero column (taken with the opposite sign). In fact, how easy it is to trace the code -v [0]contains a sum of all quantities delta, i.e. total potential change. Although at each potential change several values could change at once u [i]and v [j], the total change in the potential value is exactly the same delta, since as long as there is no increasing chain, the number of reachable rows is exactly one more than the number of reachable columns (only the current row idoes not have a pair in the form of a visited column )*/

    return -v[0];
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    cin >> n >> m;

    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            cin >> a[i][j];

    cout << hungarian();
}
```

Min-cost flow

```
/*
Minimum-cost flow
O(n^3m)
*/

struct Edge
{
    int from, to, capacity, 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;
}
```

10. Variuos

Gauss-Jordan

```
// Regresa el vector con los valores de cada variable.
// Si no es posible obtener solucion, regresa un vector vacio.
vector<double> gaussJordan(vector<vector<double> > v) {
    int n = v.size();
    double val;
    FOR(i, 0, n) {
        if (v[i][i] == 0) {
            bool b = 1;
            FOR(j, i+1, n) {
                if (v[i][j] != 0) {
                    swap(v[i], v[j]);
                    b = 0;
                    break;
                }
            }
            if (b) return vector<double>();
        }

        val = v[i][i];
        FOR(j, i, n+1) {
            v[i][j] /= val;
        }

        FOR(k, 0, n) {
            if (i == k)     continue;
            val = -v[k][i];
            FOR(j, i, n+1) {
                v[k][j] += v[i][j] * val;
            }
        }
    }

    vector<double> vd(n);
    FOR(i, 0, n)    vd[i] = v[i][n];
    return vd;
}

int main() { _
    vector<double> v(4);
    vector<vector<double> > vv;
    v[0] = 1;
    v[1] = 1;
    v[2] = 1;
    v[3] = 5;
    vv.pb(v);

    v[0] = 2;
    v[1] = 3;
    v[2] = 5;
    v[3] = 8;
    vv.pb(v);

    v[0] = 4;
    v[1] = 0;
    v[2] = 5;
    v[3] = 2;
    vv.pb(v);

    FOR(i, 0, 3) {
        FOR(j, 0, 4)
            cout << vv[i][j] << " ";
```

```
        cout << endl;
    }
    cout << endl << endl;

    vector<double> vd = gaussJordan(vv);
    printArr(vd, 0, 3);

    return 0;
}
```

2-Sat

```cpp
#define MAXN 100005
#define MOD 1000000007

struct TwoSAT{
    int n;
    vector<int> G[MAXN*2];
    int S[MAXN*2], c;
    bool mark[MAXN*2];

    bool dfs(int x){
        if(mark[x^1]) return false;
        if(mark[x]) return true;
        mark[x] = true;
        S[c++] = x;

        FOR(i, 0, G[x].size())
            if(!dfs(G[x][i])) return false;
        return true;
    }

    void init(int n){
        this->n = n;
        FOR(i, 0, 2*n) G[i].clear();
        memset(mark, 0, sizeof(mark));
    }

    /*
     * Each clause is in the form x or y
     * x is abs(x) - 1 and xval is x < 0 ? 1 : 0
     */
    void add_clause(int x, int xval, int y, int yval){
        x = x*2 + xval;
        y = y*2 + yval;
        G[x^1].push_back(y);
        G[y^1].push_back(x);
    }

    bool solve(){
        for(int i = 0; i < 2*n; i += 2){
            if(!mark[i] && !mark[i+1]){
                c = 0;
                if(!dfs(i)){
                    while(c > 0) mark[S[--c]] = false;
                    if(!dfs(i + 1)) return false;
                }
            }
        }
        return true;
    }
}TS;

int main(){
    int n, m;
    while(scanf("%d %d", &n, &m) != EOF){
        TS.init(n);
        // scan m clauses
        FOR(i, 0, m){
            int a, b;
            scanf("%d %d", &a, &b);
            TS.add_clause(abs(a) - 1, a < 0 ? 1 : 0, abs(b) - 1, b < 0 ? 1 : 0);
        }
        printf("%d\n", TS.solve()? 1 : 0);
    }
    return 0;
}
```

Bits

```
typedef unsigned int ui;
typedef unsigned long long ull;

ull BitCount(ull u) {
     ull uCount = u - ((u >> 1) & 033333333333) - ((u >> 2) & 011111111111);
     return ((uCount + (uCount >> 3)) & 030707070707) % 63;
}

ull flipBits(ull n, int k) {
    ull mask = (1 << k) - 1;
    return ~n & mask;
}

ull flipBits(ull n) {
    return ~n;
}

ull differentBits(ull a, ull b) {
    return BitCount(a ^ b);
}

void getEvenOddBits(ull n) {
    // Para ui, hacerlo con 8 A's o 5's
    ull evenBits = n & 0xAAAAAAAAAAAAAAAA;
    ull oddBits = n & 0x5555555555555555;
}

ull rotateBits(ull n, int d) {
    // d negative for left rotation, positive for right.
    d %= 64;
    return (n >> d) | (n << (64 - d));
}

string toBinary(ull n) {
    string s = "";
    while(n) {
        if (n & 1)  s = "1" + s;
        else        s = "0" + s;
        n >>= 1;
    }
    return s;
}

ui getSetBitsFromOneToN(ull n){
    ui two = 2, ans = 0;
    ull N = n;
    while(n) {
        ans += (N / two) * (two >> 1);
        if ((N & (two - 1)) > (two >> 1) - 1)
            ans += (N & (two - 1)) - (two >> 1) + 1;
        two <<= 1;
        n >>= 1;
    }
    return ans;
}
```

Fast IO

```
const int BUFFSIZE = 10240;
char BUFF[BUFFSIZE + 1], *ppp = BUFF;
int RR, CHAR, SIGN, BYTES = 0;
#define GETCHAR(c) { \
    if(ppp-BUFF==BYTES && (BYTES==0 || BYTES==BUFFSIZE)) { BYTES = fread(BUFF,1,BUFFSIZE,stdin); ppp=BUFF; } \
    if(ppp-BUFF==BYTES && (BYTES>0 && BYTES<BUFFSIZE)) { BUFF[0] = 0; ppp=BUFF; }\
    c = *ppp++; \
}
#define DIGIT(c) (((c) >= ’0’) && ((c) <= ’9’))
#define MINUS(c) ((c)== ’-’)
#define GETNUMBER(n) { \
    n = 0; SIGN = 1; do { GETCHAR(CHAR); } while(!(DIGIT(CHAR) || MINUS(CHAR))); \
    if(MINUS(CHAR)) { SIGN = -1; GETCHAR(CHAR); } \
    while(DIGIT(CHAR)) { n = ((n<<3) + (n << 1)) + CHAR-’0’; GETCHAR(CHAR); } if(SIGN == -1) { n = -n; } \
}
```

FFT 1

```
struct Complex{
    double real, imag;
    Complex(){}
    Complex(const complex<double> &a): real(a.real()), imag(a.imag()){}
    Complex(double real, double imag): real(real), imag(imag){}
    Complex operator*(const Complex & a){
        return Complex(real * a.real - imag * a.imag, real * a.imag + a.real * imag);
    }
    Complex operator+(const Complex & a){return Complex(real + a.real, imag + a.imag);}
};
Complex conj(const Complex & a){return Complex(a.real, -a.imag);}

const int maxN = 1e6;
int maxL;

int reverses[maxN << 2];
bool v[maxN << 2];
Complex steps[30], unityRoots[maxN << 2], first[maxN << 2], second[maxN << 2];
ll sums[maxN << 2];

int reverse(int num, int exp){
    int reverse = 0, pot = 1, inv = 1 << (exp - 1);
    while(inv >= 1){
        if(num & pot) reverse |= inv;
        inv >>= 1, pot <<= 1;
    }
    return reverse;
}

void sqRoot(Complex *vec, int len, int loog){
    for(int i = len - 1; i >= 0; i--){
        vec[i << 1] = vec[i];
        vec[(i << 1) + 1] = vec[i] * steps[loog];
    }
}

void FFT(Complex *coef, int arrS){
    const int saiz = ceil(log2(arrS));

    memset(v, false, sizeof(v));
    for(int i = 0; i < (1 << saiz); i++)
        if(!v[reverses[i] >> (maxL - saiz)])
            swap(coef[i], coef[reverses[i] >> (maxL - saiz)]), v[i] = true;
    unityRoots[0] = Complex(1, 0);

    //DIVIDE AND CONQUER...
    for(int T = 1, u = 0; T < (1 << saiz); T <<= 1, u++){
        sqRoot(unityRoots, T, u);
        for(int i = 0; i < (1 << saiz); i += (T << 1)){
            for(int j = 0; j < T; j++){
                Complex lTmp = coef[i + j], rTmp = coef[i + j + T];

                coef[i + j] = lTmp + rTmp * unityRoots[j];
                coef[i + j + T] = lTmp + rTmp * unityRoots[j + T];
            }
        }
    }
}

//one & two = coefficient arrays w/sizes, res is where results are stored
void polynomMultiplication(int *one, int *two, int *res, int oS, int tS){
    const int saiz = 1 << int(ceil(log2(oS + tS)));
```

```
    for(int i = 0; i < saiz; i++)
        first[i] = Complex(i < oS? double(one[i]): 0.0, 0.0);
    for(int i = 0; i < saiz; i++)
        second[i] = Complex(i < tS? double(two[i]): 0.0, 0.0);

    //FFT
    FFT(first, saiz), FFT(second, saiz);

    //INVERSE FFT = FFT(conj(C1 * C2) / N)
    for(int i = 0; i < saiz; i++)
        first[i] = conj(first[i] * second[i]);
    FFT(first, saiz);

    for(int i = 0; i < saiz; i++)
        res[i] = int(round(first[i].real / saiz));
}
```

FFT 2

```
const int maxN = 2.1e6 + 3, MOD = 1e9 + 7, LG = 21;
//NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom

cd ur[maxN], res[maxN], ors[maxN], cf[maxN];
//assumes sz and step are 2-powers
void FFT(int cfs, int step, int his, int rs, int sz, cd *res){
    if(cfs == 1)
        for(int i = 0; i < sz; i++) res[i] = cf[his];
    else{

        FFT(ceil(cfs/2.0), step+1, his, rs<<1, sz>>1, res);
        FFT(cfs>>1, step+1, 1 << step | his, rs<<1, sz>>1, res + (sz>>1));

        for(int i = 0, m = sz >> 1; i < m; i++){
            cd cr = res[i];
            res[i] = cr + res[i + m] * ur[i * rs];
            res[i + m] = cr + res[i + m] * ur[(i + m) * rs];
        }
    }
}

void mu(const vector<ll> &a, const vector<ll> &b, vector<ll> &ans){
    int sz = 1<<int(ceil(log2(a.size() + b.size() - 1)));

    for(int i = 0; i < a.size(); i++) cf[i] = a[i];
    FFT((int)a.size(), 0, 0, (1<<LG)/sz, sz, res);

    for(int i = 0; i < b.size(); i++) cf[i] = b[i];
    FFT((int)b.size(), 0, 0, (1<<LG)/sz, sz, ors);

    for(int i = 0; i < sz; i++) cf[i] = conj(res[i] * ors[i]);
    FFT(sz, 0, 0, (1<<LG)/sz, sz, res);



    for(int i = 0; i < a.size() + b.size() - 1; i++)
        ans.push_back(ll(round(res[i].real() / sz)));

}

string st;
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    ur[0] = 1, ur[1] = polar(1.0, M_PI * 2 / (1 << LG));

    for(int i = 2; i < (1 << LG); i++)
        ur[i] = ur[i - 1] * ur[1];

    cin >> st;

    vector<ll> a, b, ans;
    for(int i = 0; i < st.size(); i++)
        a.push_back((st[i] == 'B')? 1: 0);
    for(int i = int(st.size() - 1); i >= 0; i--)
        b.push_back((st[i] == 'A')? 1: 0);

    mu(a, b, ans);
    for(int i = int(st.size() - 2); i >= 0; i--) cout << ans[i] << '\n';

}
//1010
```

```
//1010

//coefficients can be handled with an integer for how many, and a pair of ints: step and history, which will tell the index of the coefficient in the base case (the only time it will be used)
//now, for the unity roots, use formula x * 2 % sz to get square simply, in this way we can keep only one array of unity roots.
//for m current unity roots, when we squere them, all indices will have at least one zero to left more in binary notation. So we just need to calculate squares of the upper side of argand's diagram. So its new size will be a power of two and the step unity will be twice bigger.
//args: cfs(how many coefficients), step(for coef), history(parities defined), rootstep(size of unity root step), rootam(amount of roots to consider)
//so process is simple now:
//base case: return coefficient rootam times
//regular case: rec(ceil(n/2), step+1, history, rootstep*2, next2pow(rootam))
//              rec(floor(n/2), step+1, 1<< st|history, rootstep*2, next2pow(rootam))

//now where should we store them?
//create array that will store evaluations. So add new parameter Foo *pl, where we will store results of shit. When we call recursively, store at end all even- coef evals. Then put in current and forget, so use same place for unevens
```

Fraction

```
struct Fraction {
    ll num, den;

    Fraction() {num = den = 1;}

    Fraction(ll numm, ll denn) : num(numm), den(denn) {
        simplify();
    }

    void simplify() {
        bool signo = false;
        if (num < 0)    signo = !signo;
        if (den < 0)    signo = !signo;
        ll g = __gcd(num, den);
        if (g) {
            num /= g;
            den /= g;
        }
        if (signo)  num = -num;
    }

    Fraction operator + (const Fraction &r) const {
        Fraction ret;
        ret.num = num*r.den + r.num*den;
        ret.den = r.den*den;
        ret.simplify();
        return ret;
    }

    Fraction operator + (const ll &r) const {
        Fraction ret;
        ret.num = num + r*den;
        ret.den = den;
        return ret;
    }

    Fraction operator - (const Fraction &r) const {
        Fraction ret;
        ret.num = num*r.den - r.num*den;
        ret.den = r.den*den;
        ret.simplify();
        return ret;
    }

    Fraction operator - (const ll &r) const {
        Fraction ret;
        ret.num = num - r * den;
        ret.den = den;
        return ret;
    }

    Fraction operator * (const Fraction &r) const {
        Fraction ret;
        ret.num = num*r.num;
        ret.den = den*r.den;
        ret.simplify();
        return ret;
    }

    Fraction operator * (const ll &r) const {
        Fraction ret;
        ret.num = num*r;
        ret.den = den;
        ret.simplify();
        return ret;
    }

    Fraction operator / (const Fraction &r) const {
        Fraction ret;
        ret.num = num*r.den;
        ret.den = den*r.num;
```

```
        ret.simplify();
        return ret;
    }

    Fraction operator / (const ll &r) const {
        Fraction ret;
        ret.num = num;
        ret.den = den*r;
        ret.simplify();
        return ret;
    }

    Fraction pow(int n) const {
        Fraction ret(1, 1);
        Fraction x(num, den);
        while (n) {
            if (n & 1) ret = ret * x;
            x = x * x;
            n >>= 1;
        }
        return ret;
    }

    bool operator <(const Fraction &r) const {
        return num * r.den < den * r.num;
    }

    bool operator ==(const Fraction &r) const {
        return num == r. num && den == r.den;
    }

    string to_str() {
        return to_string(num) + "/" + to_string(den);
    }
};

int main() { _
    Fraction f;
    while(cin >> f.num >> f.den) {
        f.simplify();
        cout << f.num << "/" << f.den << endl;
    }

    return 0;
}
```

Hour

```
struct Hora {
    int h, m, s, t;

    Hora() { t = h = m = s = 0; }

    Hora(int hh, int mm, int ss) : h(hh), m(mm), s(ss) {
        simplify();
    }

    Hora(string ss) {
        int f1 = ss.find(":");
        h = atoi(ss.substr(0, f1).c_str());

        int f2 = ss.find(":", f1+1);
        if (f2 != -1) {
            m = atoi(ss.substr(f1+1, f2-f1).c_str());
            s = atoi(ss.substr(f2+1).c_str());
        }
        else {
            m = atoi(ss.substr(f1+1).c_str());
            s = 0;
        }

        simplify();
    }

    Hora operator +(const Hora &r) const {
        Hora ret;

        ret.h = h + r.h;
        ret.m = m + r.m;
        ret.s = s + r.s;

        ret.simplify();

        return ret;
    }

    Hora operator -(const Hora &r) const {
        Hora ret;

        ret.h = h - r.h;
        ret.m = m - r.m;
        ret.s = s - r.s;

        ret.simplify();

        return ret;
    }

    bool operator <(const Hora &r) const {
        return t < r.t;
    }

    bool operator ==(const Hora &r) const {
        return t == r.t;
    }

    void simplify() {
        t = 3600 * h + 60 * m + s;
        t = (t + 86400) % 86400;

        int tt = t;
        h = tt / 3600;
        tt %= 3600;
        m = tt / 60;
        s = tt % 60;
    }

    string to_str() {
        string ret = "";

```

```
        if (h < 10) ret = "0";
        ret += std::to_string(h) + ":";

        if (m < 10) ret += "0";
        ret += std::to_string(m) + ":";

        if (s < 10) ret += "0";
        ret += std::to_string(s);

        return ret;
    }

    string to_str(const bool b) {
        string ret = "";

        if (h < 10) ret = "0";
        ret += to_string(h) + ":";

        if (m < 10) ret += "0";
        ret += to_string(m);

        return ret;
    }
};

int main() { _
    Hora h1, h2;
    string s1, s2;

    while(cin >> s1 >> s2) {
        h1 = Hora(s1);
        h2 = Hora(s2);

        cout << (h1 - h2).to_str() << endl;
    }

    return 0;
}
```

Matrix Exponentiation

```
#define FIRST first
#define SECOND second.first
#define THIRD second.second

using namespace std;
typedef pair<int, int> ii;
typedef vector<double> vd;
typedef vector<vd> Matrix;
typedef long long ll;

const int maxN = 100;

Matrix operator *(const Matrix &first, const Matrix &second){
    Matrix ret(first.size(), vd(second[0].size(), 0.0));
    for(int i = 0; i < first.size(); i++)
        for(int j = 0; j < second[0].size(); j++)
            for(int k = 0; k < second.size(); k++)
                ret[i][j] += first[i][k] * second[k][j];
    return ret;
}
Matrix operator^(Matrix mat, int coef){
    Matrix ret(mat.size(), vd(mat.size(), 0.0));
    for(int i = 0; i < mat.size(); i++) ret[i][i] = 1.0;
    while(coef){
        if(coef & 1)
            ret = ret * mat;
        coef >>= 1, mat = mat * mat;
    }
    return ret;
}
```

Roman Numbers

```cpp
#define MAXN 10
#define MOD 1000000007

int mil[MAXN];

string fill(char c, int n) {
    string s;
    while (n--) s += c;
    return s;
}

string toRoman(ll n, int nivel) {
    if (n == 0) return "";
    if (n < 4) return fill('I', n);
    if (n < 6) return fill('I', 5 - n) + "V";
    if (n < 9) return string("V") + fill('I', n - 5);
    if (n < 11) return fill('I', 10 - n) + "X";
    if (n < 40) return fill('X', n / 10) + toRoman(n % 10, nivel);
    if (n < 60) return fill('X', 5 - n / 10) + 'L' + toRoman(n % 10, nivel);
    if (n < 90) return string("L") + fill('X', n / 10 - 5 ) + toRoman(n % 10, nivel);
    if (n < 110) return fill('X', 10 - n / 10) + "C" + toRoman(n % 10, nivel);
    if (n < 400) return fill('C', n / 100) + toRoman(n % 100, nivel);
    if (n < 600) return fill('C', 5 - n / 100) + 'D' + toRoman(n % 100, nivel);
    if (n < 900) return string("D" ) + fill('C', n / 100 - 5) + toRoman(n % 10, nivel);
    if (n < 1100) return fill('C', 10 - n / 100) + "M" + toRoman(n % 100, nivel);
    if (n < 4000) return fill('M', n / 1000) + toRoman(n % 1000, nivel);

    string ret = toRoman(n / 1000, nivel + 1);
    string ret2 = toRoman(n % 1000, nivel);
    mil[nivel] = ret.length();
    if (ret2 == "")    return ret;
    return ret + " " + toRoman(n % 1000, nivel);
}

string toRoman(ll n) {
    string ret = toRoman(n, 0);
    for(int i = 0; mil[i]; i ++) {
        ret = fill('_', mil[i]) + "\n" + ret;
    }

    return ret;
}

int main() { _
    ll n;
    while (cin >> n) {
        cout << toRoman(n) << endl;
    }
    return 0;
}
```

Rotate Matrix

```
#define MAXN 10
#define MOD 1000000007

int mat[MAXN][MAXN];

void giraMat(int k, int n) {
    k %= 4;

    FOR(p, 0, k) {
        FOR(i, 0, n/2) {
            FOR(j, i, n - i - 1) {
                ii p1(i, j);
                ii p2(j, n - i - 1);
                ii p3(n - i - 1, n - j - 1);
                ii p4(n - j - 1, i);

                int aux = mat[p1.first][p1.second];
                mat[p1.first][p1.second] = mat[p4.first][p4.second];
                mat[p4.first][p4.second] = mat[p3.first][p3.second];
                mat[p3.first][p3.second] = mat[p2.first][p2.second];
                mat[p2.first][p2.second] = aux;
            }
        }
    }
}

int main() { _
    int cant = 1;
    FOR(i, 0, MAXN) FOR(j, 0, MAXN) {
        mat[i][j] = cant;
        cant ++;
    }

    giraMat(6, MAXN);

    FOR(i, 0, MAXN) FOR(j, 0, MAXN)
        cout << mat[i][j] << "\t\n"[j == MAXN-1];
    return 0;
}
```

Sorts

```
#define MAXN 15
#define MOD 1000000007

int arr[MAXN];
int comp, inter;

void print() {
    FOR(i, 0, MAXN) {
        cout << arr[i] << " \n"[i == MAXN-1];
    }
    cout << endl;
}

void bubbleSort() {
    int n = MAXN;
    bool b = true;

    for(int i = 0; i < n - 1 && b; i ++){
        b = false;
        FOR(j, 0, n - 1 - i) {
            if (arr[j+1] < arr[j]) {
                swap(arr[j], arr[j + 1]);
                b = true;

                inter ++;
            }
            comp ++;
        }
    }
}

void selectionSort() {
    int n = MAXN;
    FOR(i, 0, n - 1) {
        int mini = i;
        FOR(j, i + 1, n) {
            comp ++;
            if (arr[j] < arr[mini]) mini = j;

        }

        if (i != mini) {
            swap(arr[i], arr[mini]);
            inter ++;
        }
    }
}

void intercambioSort() {
    int n = MAXN;
    FOR(i, 0, n - 1) {
        FOR(j, i + 1, n) {
            if (arr[i] > arr[j]) {
                swap(arr[i], arr[j]);
                inter++;
            }
            comp++;
        }
    }
}

void insertionSort() {
    int j, key, n = MAXN;
    FOR(i, 1, n) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            inter ++;
            arr[j+1] = arr[j];
            j = j - 1;
        }

```

```
        comp = inter;
        if(j)   comp ++;

        arr[j+1] = key;
    }
}

void insertionBinarySort() {
    int n = MAXN;
    vi v;
    FOR(i, 0, n) {
        v.insert(upper_bound(v.begin(), v.end(), arr[i]), arr[i]);
    }

    copy(v.begin(), v.end(), arr);
}


void Une(int ini, int mitad, int fin) {
    int aux[fin - ini + 1];
    int i = ini, j = mitad + 1, k = 0;

    while(i <= mitad && j <= fin) {
        if (arr[i] < arr[j]) {
            aux[k] = arr[i];
            i ++;
        }
        else {
            aux[k] = arr[j];
            j ++;
        }
        k ++;
    }

    for(; i <= mitad; i ++, k ++)    aux[k] = arr[i];
    for(; j <= fin; j ++, k ++)      aux[k] = arr[j];

    for(k = 0; ini <= fin; ini ++, k ++)
        arr[ini] = aux[k];
}

void mergeSort(int ini, int fin){
    if(ini < fin){
        int mitad = (ini + fin) / 2;
        mergeSort(ini, mitad);
        mergeSort(mitad + 1, fin);
        Une(ini, mitad, fin);
    }
}


void heapify(int n, int i) {
    int maxi = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] > arr[maxi])
        maxi = l;

    if (r < n && arr[r] > arr[maxi])
        maxi = r;

    if (maxi != i) {
        swap(arr[i], arr[maxi]);
        heapify(n, maxi);
    }
}

void heapSort() {
    int n = MAXN;
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(n, i);

    for (int i = n-1; i >= 0; i--) {
```

```
        swap(arr[0], arr[i]);
        heapify(i, 0);
    }
}

int part(int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    FOR(j, low, high) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// QuickSort with less recursion calls.
void quickSortNoTail(int low, int high) {
    while (low < high) {
        int pi = part(low, high);

        if (pi - low < high - pi) {
            quickSortNoTail(low, pi - 1);
            low = pi + 1;
        }
        else {
            quickSortNoTail(pi + 1, high);
            high = pi - 1;
        }
    }
}

void quickSort(int low, int high) {
    if (low < high) {
        int pi = part(low, high);
        quickSort(low, pi - 1);
        quickSort(pi + 1, high);
    }
}
```

11. Otros

Series

$$\sum_{i=0}^{n} 1 = n$$

$$\log x^y = y \ \log x$$

$$\sum_{i=0}^{n} i = \frac{n * (n+1)}{2}$$

$$\log xy = \log x + \log y$$

$$\sum_{i=0}^{n} i^2 = \frac{n * (n+1) * (2n+1)}{6}$$

$$\log \frac{x}{y} = \log x - \log y$$

$$\sum_{i=0}^{n} i^3 = \frac{n^2 * (n+1)^2}{4}$$

$$\sum_{i=0}^{n} i^k \approx \frac{1}{k+1} * \ n^{k+1}$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=0}^{n} 2^i = \ 2^{n+1} - 1$$

$$\sum_{i=0}^{n} \binom{i}{n} = \ 2^n$$

First 5000 digits of PI

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808
6513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933
4461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748815
2092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193
2611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190
7021798609437027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371787214
6844090122495343014654958537105079227968925892354201995611212902196086403441815981362977477130996051870721134
9999998372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137838752886
5875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321712268066130019278
7661119590921642019893809525720106548586327886593615338182796823030195203530185296899577362259941389124972177
5283479131515574857242454150695950829533116861727855889075098381754637464939319255060400927701671139009848824
0128583616035637076601047101819429555961989467678374494482553797747268471040475346462080466842590694912933136
7702898915210475216205696602405803815019351125338243003558764024749647326391419927260426992279678235478163600
9341721641219924586315030286182974555706749838505494588586926995690927210797509302955321165344987202755960236
4806654991198818347977535663698074265425278625518184175746728909777727938000816470600161452491921732172147723
5014144197356854816136115735255213347574184946843852332390739414333454776241686251898356948556209921922218427
2550254256887671790494601653466804988627232791786085784383827967976681454100953883786360950680064225125205117
3929848960841284886269456042419652850222106611863067442786220391949450471237137869609563643719172874677646575
7396241389086583264599581339047802759009946576407895126946839835259570982582262052248940772671947826848260147
6990902640136394437455305068203496252451749399651431429809190659250937221696461515709858387410597885959772975
4989301617539284681382686838689427741559918559252459539594310499725246808459872736446958486538367362226260991
2460805124388439045124413654976278079771569143599770012961608944169486855584840635342207222582848864815845602
8506016842739452267467678895252138522549954666727823986456596116354886230577456498035593634568174324112515076
0694794510965960940252288797108931456691368672287489405601015033086179286809208747609178249385890097149096759
8526136554978189312978482168299894872265880485756401427047755513237964145152374623436454285844479526586782105
1141354735739523113427166102135969536231442952484937187110145765403590279934403742007310578539062198387447808
4784896833214457138687519435064302184531910484810053706146806749192781911979399520614196634287544406437451237
1819217999839101591956181467514269123974894090718649423196156794520809514655022523160388193014209376213785595
6638937787083039069792077346722182562599661501421503068038447734549202605414665925201497442850732518666002132
4340881907104863317346496514539057962685610055081066587969981635747363840525714591028970641401109712062804390
3975951567715770042033786993600723055876317635942187312514712053292819182618612586732157919841484882916447060
9575270695722091756711672291098169091528017350671274858322287183520935396572512108357915136988209144421006751
0334671103141267111369908658516398315019701651511685171437657618351556508849099898599823873455283316355076479
1853589322618548963213293308985706420467525907091548141654985946163718027098199430992448895757128289059232332
6097299712084433573265489382391193259746366730583604142813883032038249037589852437441702913276561809377344403
0707469211201913020330380197621101100449293215160842444859637669838952286847831235526582131449576857262433441
8930396864262434107732269780280731891544110104468232527162010526522721116603966655730925471105578537634668206
5310989652691862056476931257058635662018558100729360659876486117910453348850346113657686753249441668039626579
7877185560845529654126654085306143444318586769751456614068007002378776591344017127494704205622305389945613140
7112700040785473326993908145466464588079727082668306343285878569830523580893306575740679545716377525420211495
5761581400250126228594130216471550979259230990796547376125517656751357517829666454779174501129961489030463994
7132962107340437518957359614589019389713111790429782856475032031986915140287080859904801094121472213179476477
7262241425485454033215718530614228813758504306332175182979866223717215916077166925474873898665494945011465406
2843366393790039769265672146385306736096571209180763832716641627488880078692560290228472104031721186082041900
0422966171196377921337575114959501566049631862947265473642523081770367515906735023507283540567040386743513622
2247715891504953098444893330963408780769325993978054193414473774418426312986080998886874132604721

First 150 Fibonacci numbers

0 : 0
1 : 1
2 : 1
3 : 2
4 : 3
5 : 5
6 : 8 = 2^3
7 : 13
8 : 21 = 3 x 7
9 : 34 = 2 x 17
10 : 55 = 5 x 11
11 : 89
12 : 144 = 2^4 x 3^2
13 : 233
14 : 377 = 13 x 29
15 : 610 = 2 x 5 x 61
16 : 987 = 3 x 7 x 47
17 : 1597
18 : 2584 = 2^3 x 17 x 19
19 : 4181 = 37 x 113
20 : 6765 = 3 x 5 x 11 x 41
21 : 10946 = 2 x 13 x 421
22 : 17711 = 89 x 199
23 : 28657
24 : 46368 = 2^5 x 3^2 x 7 x 23
25 : 75025 = 5^2 x 3001
26 : 121393 = 233 x 521
27 : 196418 = 2 x 17 x 53 x 109
28 : 317811 = 3 x 13 x 29 x 281
29 : 514229
30 : 832040 = 2^3 x 5 x 11 x 31 x 61
31 : 1346269 = 557 x 2417
32 : 2178309 = 3 x 7 x 47 x 2207
33 : 3524578 = 2 x 89 x 19801
34 : 5702887 = 1597 x 3571
35 : 9227465 = 5 x 13 x 141961
36 : 14930352 = 2^4 x 3^3 x 17 x 19 x 107
37 : 24157817 = 73 x 149 x 2221
38 : 39088169 = 37 x 113 x 9349
39 : 63245986 = 2 x 233 x 135721
40 : 102334155 = 3 x 5 x 7 x 11 x 41 x 2161
41 : 165580141 = 2789 x 59369
42 : 267914296 = 2^3 x 13 x 29 x 211 x 421
43 : 433494437
44 : 701408733 = 3 x 43 x 89 x 199 x 307
45 : 1134903170 = 2 x 5 x 17 x 61 x 109441
46 : 1836311903 = 139 x 461 x 28657
47 : 2971215073
48 : 4807526976 = 2^6 x 3^2 x 7 x 23 x 47 x 1103
49 : 7778742049 = 13 x 97 x 6168709
50 : 12586269025 = 5^2 x 11 x 101 x 151 x 3001
51 : 20365011074 = 2 x 1597 x 6376021
52 : 32951280099 = 3 x 233 x 521 x 90481
53 : 53316291173 = 953 x 55945741
54 : 86267571272 = 2^3 x 17 x 19 x 53 x 109 x 5779
55 : 139583862445 = 5 x 89 x 661 x 474541
56 : 225851433717 = 3 x 7^2 x 13 x 29 x 281 x 14503
57 : 365435296162 = 2 x 37 x 113 x 797 x 54833
58 : 591286729879 = 59 x 19489 x 514229
59 : 956722026041 = 353 x 2710260697
60 : 1548008755920 = 2^4 x 3^2 x 5 x 11 x 31 x 41 x 61 x 2521
61 : 2504730781961 = 4513 x 555003497
62 : 4052739537881 = 557 x 2417 x 3010349
63 : 6557470319842 = 2 x 13 x 17 x 421 x 35239681
64 : 10610209857723 = 3 x 7 x 47 x 1087 x 2207 x 4481
65 : 17167680177565 = 5 x 233 x 14736206161
66 : 27777890035288 = 2^3 x 89 x 199 x 9901 x 19801
67 : 44945570212853 = 269 x 116849 x 1429913
68 : 72723460248141 = 3 x 67 x 1597 x 3571 x 63443
69 : 117669030460994 = 2 x 137 x 829 x 18077 x 28657
70 : 190392490709135 = 5 x 11 x 13 x 29 x 71 x 911 x 141961
71 : 308061521170129 = 6673 x 46165371073
72 : 498454011879264 = 2^5 x 3^3 x 7 x 17 x 19 x 23 x 107 x 103681
73 : 806515533049393 = 9375829 x 86020717

74 : 1304969544928657 = 73 x 149 x 2221 x 54018521

75 : 2111485077978050 = 2 x 5^{2} x 61 x 3001 x 230686501

76 : 3416454622906707 = 3 x 37 x 113 x 9349 x 29134601

77 : 5527939700884757 = 13 x 89 x 988681 x 4832521

78 : 8944394323791464 = 2^{3} x 79 x 233 x 521 x 859 x 135721

79 : 14472334024676221 = 157 x 92180471494753

80 : 23416728348467685 = 3 x 5 x 7 x 11 x 41 x 47 x 1601 x 2161 x 3041

81 : 37889062373143906 = 2 x 17 x 53 x 109 x 2269 x 4373 x 19441

82 : 61305790721611591 = 2789 x 59369 x 370248451

83 : 99194853094755497

84 : 160500643816367088 = 2^{4} x 3^{2} x 13 x 29 x 83 x 211 x 281 x 421 x 1427

85 : 259695496911122585 = 5 x 1597 x 9521 x 3415914041

86 : 420196140727489673 = 6709 x 144481 x 433494437

87 : 679891637638612258 = 2 x 173 x 514229 x 3821263937

88 : 1100087778366101931 = 3 x 7 x 43 x 89 x 199 x 263 x 307 x 881 x 967

89 : 1779979416004714189 = 1069 x 1665088321800481

90 : 2880067194370816120 = 2^{3} x 5 x 11 x 17 x 19 x 31 x 61 x 181 x 541 x 109441

91 : 4660046610375530309 = 13^{2} x 233 x 741469 x 159607993

92 : 7540113804746346429 = 3 x 139 x 461 x 4969 x 28657 x 275449

93 : 12200160415121876738 = 2 x 557 x 2417 x 4531100550901

94 : 19740274219868223167 = 2971215073 x 6643838879

95 : 31940434634990099905 = 5 x 37 x 113 x 761 x 29641 x 67735001

96 : 51680708854858323072 = 2^{7} x 3^{2} x 7 x 23 x 47 x 769 x 1103 x 2207 x 3167

97 : 83621143489848422977 = 193 x 389 x 3084989 x 361040209

98 : 135301852344706746049 = 13 x 29 x 97 x 6168709 x 599786069

99 : 218922995834555169026 = 2 x 17 x 89 x 197 x 19801 x 18546805133

100 : 354224848179261915075 = 3 x 5^{2} x 11 x 41 x 101 x 151 x 401 x 3001 x 570601

101 : 573147844013817084101 = 743519377 x 770857978613

102 : 927372692193078999176 = 2^{3} x 919 x 1597 x 3469 x 3571 x 6376021

103 : 1500520536206896083277 = 519121 x 5644193 x 512119709

104 : 2427893228399975082453 = 3 x 7 x 103 x 233 x 521 x 90481 x 102193207

105 : 3928413764606871165730 = 2 x 5 x 13 x 61 x 421 x 141961 x 8288823481

106 : 6356306993006846248183 = 953 x 55945741 x 119218851371

107 : 10284720757613717413913 = 1247833 x 8242065050061761

108 : 16641027750620563662096 = 2^{4} x 3^{4} x 17 x 19 x 53 x 107 x 109 x 5779 x 11128427

109 : 26925748508234281076009 = 827728777 x 32529675488417

110 : 43566776258854844738105 = 5 x 11^{2} x 89 x 199 x 331 x 661 x 39161 x 474541

111 : 70492524767089125814114 = 2 x 73 x 149 x 2221 x 1459000305513721

112 : 114059301025943970552219 = 3 x 7^{2} x 13 x 29 x 47 x 281 x 14503 x 10745088481

113 : 184551825793033096366333 = 677 x 272602401466814027129

114 : 298611126818977066918552 = 2^{3} x 37 x 113 x 229 x 797 x 9349 x 54833 x 95419

115 : 483162952612010163284885 = 5 x 1381 x 28657 x 2441738887963981

116 : 781774079430987230203437 = 3 x 59 x 347 x 19489 x 514229 x 1270083883

117 : 1264937032042997393488322 = 2 x 17 x 233 x 29717 x 135721 x 39589685693

118 : 2046711111473984623691759 = 353 x 709 x 8969 x 336419 x 2710260697

119 : 3311648143516982017180081 = 13 x 1597 x 159512939815855788121

120 : 5358359254990966640871840 = 2^{5} x 3^{2} x 5 x 7 x 11 x 23 x 31 x 41 x 61 x 241 x 2161 x 2521 x 20641

121 : 8670007398507948658051921 = 89 x 97415813466381445596089

122 : 14028366653498915298923761 = 4513 x 555003497 x 5600748293801

123 : 22698374052006863956975682 = 2 x 2789 x 59369 x 68541957733949701

124 : 36726740705505779255899443 = 3 x 557 x 2417 x 3010349 x 3020733700601

125 : 59425114757512643212875125 = 5^{3} x 3001 x 158414167964045700001

126 : 96151855463018422468774568 = 2^{3} x 13 x 17 x 19 x 29 x 211 x 421 x 1009 x 31249 x 35239681

127 : 155576970220531065681649693 = 27941 x 5568053048227732210073

128 : 251728825683549488150424261 = 3 x 7 x 47 x 127 x 1087 x 2207 x 4481 x 186812208641

129 : 407305795904080553832073954 = 2 x 257 x 5417 x 8513 x 39639893 x 433494437

130 : 659034621587630041982498215 = 5 x 11 x 131 x 233 x 521 x 2081 x 24571 x 14736206161

131 : 1066340417491710595814572169

132 : 1725375039079340637797070384 = 2^{4} x 3^{2} x 43 x 89 x 199 x 307 x 9901 x 19801 x 261399601

133 : 2791715456571051233611642553 = 13 x 37 x 113 x 3457 x 42293 x 351301301942501

134 : 4517090495650391871408712937 = 269 x 4021 x 116849 x 1429913 x 24994118449

135 : 7308805952221443105020355490 = 2 x 5 x 17 x 53 x 61 x 109 x 109441 x 1114769954367361

136 : 11825896447871834976429068427 = 3 x 7 x 67 x 1597 x 3571 x 63443 x 23230657239121

137 : 19134702400093278081449423917

138 : 30960598847965113057878492344 = 2^{3} x 137 x 139 x 461 x 691 x 829 x 18077 x 28657 x 1485571

139 : 50095301248058391139327916261 = 277 x 2114537501 x 85526722937689093

140 : 81055900096023504197206408605 = 3 x 5 x 11 x 13 x 29 x 41 x 71 x 281 x 911 x 141961 x 12317523121

141 : 131151201344081895336534324866 = 2 x 108289 x 1435097 x 142017737 x 2971215073

142 : 212207101440105399533740733471 = 6673 x 46165371073 x 688846502588399

143 : 343358302784187294870275058337 = 89 x 233 x 8581 x 1929584153756850496621

144 : 555565404224292694404015791808 = 2^{6} x 3^{3} x 7 x 17 x 19 x 23 x 47 x 107 x 1103 x 103681 x 10749957121

145 : 898923707008479989274290850145 = 5 x 514229 x 349619996930737079890201

146 : 1454489111232772683678306641953 = 151549 x 9375829 x 86020717 x 11899937029

147 : 2353412818241252672952597492098 = 2 x 13 x 97 x 293 x 421 x 3529 x 6168709 x 347502052673

148 : 3807901929474025356630904134051 = 3 x 73 x 149 x 2221 x 11987 x 54018521 x 81143477963

149 : 6161314747715278029583501626149 = 110557 x 162709 x 4000949 x 85607646594577

150 : 9969216677189303386214405760200 = 2^{3} x 5^{2} x 11 x 31 x 61 x 101 x 151 x 3001 x 12301 x 18451 x 23068650

First 1000 prime numbers

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
53	59	61	67	71	73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349	353	359	367	373	379
383	389	397	401	409	419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541	547	557	563	569	571
577	587	593	599	601	607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733	739	743	751	757	761
769	773	787	797	809	811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941	947	953	967	971	977
983	991	997	1009	1013	1019	1021	1031	1033	1039	1049	1051	1061	1063	1069
1087	1091	1093	1097	1103	1109	1117	1123	1129	1151	1153	1163	1171	1181	1187
1193	1201	1213	1217	1223	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291
1297	1301	1303	1307	1319	1321	1327	1361	1367	1373	1381	1399	1409	1423	1427
1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
1523	1531	1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607	1609	1613
1619	1621	1627	1637	1657	1663	1667	1669	1693	1697	1699	1709	1721	1723	1733
1741	1747	1753	1759	1777	1783	1787	1789	1801	1811	1823	1831	1847	1861	1867
1871	1873	1877	1879	1889	1901	1907	1913	1931	1933	1949	1951	1973	1979	1987
1993	1997	1999	2003	2011	2017	2027	2029	2039	2053	2063	2069	2081	2083	2087
2089	2099	2111	2113	2129	2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
2221	2237	2239	2243	2251	2267	2269	2273	2281	2287	2293	2297	2309	2311	2333
2339	2341	2347	2351	2357	2371	2377	2381	2383	2389	2393	2399	2411	2417	2423
2437	2441	2447	2459	2467	2473	2477	2503	2521	2531	2539	2543	2549	2551	2557
2579	2591	2593	2609	2617	2621	2633	2647	2657	2659	2663	2671	2677	2683	2687
2689	2693	2699	2707	2711	2713	2719	2729	2731	2741	2749	2753	2767	2777	2789
2791	2797	2801	2803	2819	2833	2837	2843	2851	2857	2861	2879	2887	2897	2903
2909	2917	2927	2939	2953	2957	2963	2969	2971	2999	3001	3011	3019	3023	3037
3041	3049	3061	3067	3079	3083	3089	3109	3119	3121	3137	3163	3167	3169	3181
3187	3191	3203	3209	3217	3221	3229	3251	3253	3257	3259	3271	3299	3301	3307
3313	3319	3323	3329	3331	3343	3347	3359	3361	3371	3373	3389	3391	3407	3413
3433	3449	3457	3461	3463	3467	3469	3491	3499	3511	3517	3527	3529	3533	3539
3541	3547	3557	3559	3571	3581	3583	3593	3607	3613	3617	3623	3631	3637	3643
3659	3671	3673	3677	3691	3697	3701	3709	3719	3727	3733	3739	3761	3767	3769
3779	3793	3797	3803	3821	3823	3833	3847	3851	3853	3863	3877	3881	3889	3907
3911	3917	3919	3923	3929	3931	3943	3947	3967	3989	4001	4003	4007	4013	4019
4021	4027	4049	4051	4057	4073	4079	4091	4093	4099	4111	4127	4129	4133	4139
4153	4157	4159	4177	4201	4211	4217	4219	4229	4231	4241	4243	4253	4259	4261
4271	4273	4283	4289	4297	4327	4337	4339	4349	4357	4363	4373	4391	4397	4409
4421	4423	4441	4447	4451	4457	4463	4481	4483	4493	4507	4513	4517	4519	4523
4547	4549	4561	4567	4583	4591	4597	4603	4621	4637	4639	4643	4649	4651	4657
4663	4673	4679	4691	4703	4721	4723	4729	4733	4751	4759	4783	4787	4789	4793
4799	4801	4813	4817	4831	4861	4871	4877	4889	4903	4909	4919	4931	4933	4937
4943	4951	4957	4967	4969	4973	4987	4993	4999	5003	5009	5011	5021	5023	5039
5051	5059	5077	5081	5087	5099	5101	5107	5113	5119	5147	5153	5167	5171	5179
5189	5197	5209	5227	5231	5233	5237	5261	5273	5279	5281	5297	5303	5309	5323
5333	5347	5351	5381	5387	5393	5399	5407	5413	5417	5419	5431	5437	5441	5443
5449	5471	5477	5479	5483	5501	5503	5507	5519	5521	5527	5531	5557	5563	5569
5573	5581	5591	5623	5639	5641	5647	5651	5653	5657	5659	5669	5683	5689	5693
5701	5711	5717	5737	5741	5743	5749	5779	5783	5791	5801	5807	5813	5821	5827
5839	5843	5849	5851	5857	5861	5867	5869	5879	5881	5897	5903	5923	5927	5939
5953	5981	5987	6007	6011	6029	6037	6043	6047	6053	6067	6073	6079	6089	6091
6101	6113	6121	6131	6133	6143	6151	6163	6173	6197	6199	6203	6211	6217	6221
6229	6247	6257	6263	6269	6271	6277	6287	6299	6301	6311	6317	6323	6329	6337
6343	6353	6359	6361	6367	6373	6379	6389	6397	6421	6427	6449	6451	6469	6473
6481	6491	6521	6529	6547	6551	6553	6563	6569	6571	6577	6581	6599	6607	6619
6637	6653	6659	6661	6673	6679	6689	6691	6701	6703	6709	6719	6733	6737	6761
6763	6779	6781	6791	6793	6803	6823	6827	6829	6833	6841	6857	6863	6869	6871
6883	6899	6907	6911	6917	6947	6949	6959	6961	6967	6971	6977	6983	6991	6997
7001	7013	7019	7027	7039	7043	7057	7069	7079	7103	7109	7121	7127	7129	7151
7159	7177	7187	7193	7207	7211	7213	7219	7229	7237	7243	7247	7253	7283	7297
7307	7309	7321	7331	7333	7349	7351	7369	7393	7411	7417	7433	7451	7457	7459
7477	7481	7487	7489	7499	7507	7517	7523	7529	7537	7541	7547	7549	7559	7561
7573	7577	7583	7589	7591	7603	7607	7621	7639	7643	7649	7669	7673	7681	7687
7691	7699	7703	7717	7723	7727	7741	7753	7757	7759	7789	7793	7817	7823	7829
7841	7853	7867	7873	7877	7879	7883	7901	7907	7919					

First 25 Catalan numbers

1	1
2	1
3	2
4	5
5	14
6	42
7	132
8	429
9	1430
10	4862
11	16796
12	58786
13	208012
14	742900
15	2674440
16	9694845
17	35357670
18	129644790
19	477638700
20	1767263190
21	6564120420
22	24466267020
23	91482563640
24	343059613650
25	1289904147324
26	4861946401452
27	18367353072152
28	69533550916004
29	263747951750360
30	1002242216651360
31	3814986502092300

First 100 powers of 2

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^10	1024
2^11	2048
2^12	4096
2^13	8192
2^14	16384
2^15	32768
2^16	65536
2^17	131072
2^18	262144
2^19	524288
2^20	1048576
2^21	2097152
2^22	4194304
2^23	8388608
2^24	16777216
2^25	33554432
2^26	67108864
2^27	134217728
2^28	268435456
2^29	536870912
2^30	1073741824
2^31	2147483648
2^32	4294967296
2^33	8589934592
2^34	17179869184
2^35	34359738368
2^36	68719476736
2^37	137438953472
2^38	274877906944
2^39	549755813888
2^40	1099511627776
2^41	2199023255552
2^42	4398046511104
2^43	8796093022208
2^44	17592186044416
2^45	35184372088832
2^46	70368744177664
2^47	140737488355328
2^48	281474976710656
2^49	562949953421312
2^50	1125899906842624
2^51	2251799813685248
2^52	4503599627370496
2^53	9007199254740992
2^54	18014398509481984
2^55	36028797018963968
2^56	72057594037927936
2^57	144115188075855872
2^58	288230376151711744
2^59	576460752303423488
2^60	1152921504606846976
2^61	2305843009213693952
2^62	4611686018427387904
2^63	9223372036854775808
2^64	18446744073709551616
2^65	36893488147419103232
2^66	73786976294838206464
2^67	147573952589676412928
2^68	295147905179352825856
2^69	590295810358705651712
2^70	1180591620717411303424
2^71	2361183241434822606848
2^72	4722366482869645213696
2^73	9444732965739290427392
2^74	18889465931478580854784
2^75	37778931862957161709568
2^76	75557863725914323419136
2^77	151115727451828646838272
2^78	302231454903657293676544
2^79	604462909807314587353088
2^80	1208925819614629174706176
2^81	2417851639229258349412352
2^82	4835703278458516698824704
2^83	9671406556917033397649408
2^84	19342813113834066795298816
2^85	38685626227668133590597632
2^86	77371252455336267181195264
2^87	154742504910672534362390528
2^88	309485009821345068724781056
2^89	618970019642690137449562112
2^90	1237940039285380274899124224
2^91	2475880078570760549798248448
2^92	4951760157141521099596496896
2^93	9903520314283042199192993792
2^94	19807040628566084398385987584
2^95	39614081257132168796771975168
2^96	79228162514264337593543950336
2^97	158456325028528675187087900672
2^98	316912650057057350374175801344
2^99	633825300114114700748351602688
2^100	1267650600228229401496703205376
2^101	2535301200456458802993406410752
2^102	5070602400912917605986812821504
2^103	10141204801825835211973625643008
2^104	20282409603651670423947251286016
2^105	40564819207303340847894502572032
2^106	81129638414606681695789005144064
2^107	162259276829213363391578010288128
2^108	324518553658426726783156020576256
2^109	649037107316853453566312041152512
2^110	1298074214633706907132624082305024
2^111	2596148429267413814265248164610048
2^112	5192296858534827628530496329220096
2^113	10384593717069655257060992658440192
2^114	20769187434139310514121985316880384
2^115	41538374868278621028243970633760768
2^116	83076749736557242056487941267521536
2^117	166153499473114484112975882535043072
2^118	332306998946228968225951765070086144
2^119	664613997892457936451903530140172288
2^120	1329227995784915872903807060280344576
2^121	2658455991569831745807614120560689152
2^122	5316911983139663491615228241121378304
2^123	10633823966279326983230456482242756608
2^124	21267647932558653966460912964485513216
2^125	42535295865117307932921825928971026432
2^126	85070591730234615865843651857942052864
2^127	170141183460469231731687303715884105728
2^128	340282366920938463463374607431768211456
2^129	680564733841876926926749214863536422912
2^130	1361129467683753853853498429727072845824
2^131	2722258935367507707706996859454145691648
2^132	5444517870735015415413993718908291383296
2^133	10889035741470030830827987437816582766592
2^134	21778071482940061661655974875633165533184
2^135	43556142965880123323311949751266331066368
2^136	87112285931760246646623899502532662132736
2^137	174224571863520493293247799005065324265472
2^138	348449143727040986586495598010130648530944
2^139	696898287454081973172991196020261297061888
2^140	1393796574908163946345982392040522594123776
2^141	2787593149816327892691964784081045188247552
2^142	5575186299632655785383929568162090376495104
2^143	11150372599265311570767859136324180752990208
2^144	22300745198530623141535718272648361505980416
2^145	44601490397061246283071436545296723011960832
2^146	89202980794122492566142873090593446023921664
2^147	178405961588244985132285746181186892047843328
2^148	356811923176489970264571492362373784095686656
2^149	713623846352979940529142984724747568191373312
2^150	1427247692705959881058285969449495136382746624

First 30 Rows of Pascal Triangle

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1
1 20 190 1140 4845 15504 38760 77520 125970 167960 184756 167960 125970 77520 38760 15504 4845 1140 190 20 1
1 21 210 1330 5985 20349 54264 116280 203490 293930 352716 352716 293930 203490 116280 54264 20349 5985 1330 210 21
1
1 22 231 1540 7315 26334 74613 170544 319770 497420 646646 705432 646646 497420 319770 170544 74613 26334 7315 1540 231
22 1
1 23 253 1771 8855 33649 100947 245157 490314 817190 1144066 1352078 1352078 1144066 817190 490314 245157 100947 33649 8855 1771
253 23 1
1 24 276 2024 10626 42504 134596 346104 735471 1307504 1961256 2496144 2704156 2496144 1961256 1307504 735471 346104 134596 42504 10626
2024 276 24 1
1 25 300 2300 12650 53130 177100 480700 1081575 2042975 3268760 4457400 5200300 5200300 4457400 3268760 2042975 1081575 480700 177100 53130
12650 2300 300 25 1
1 26 325 2600 14950 65780 230230 657800 1562275 3124550 5311735 7726160 9657700 10400600 9657700 7726160 5311735 3124550 1562275 657800 230230
65780 14950 2600 325 26 1
1 27 351 2925 17550 80730 296010 888030 2220075 4686825 8436285 13037895 17383860 20058300 20058300 17383860 13037895 8436285 4686825 2220075 888030
296010 80730 17550 2925 351 27 1
1 28 378 3276 20475 98280 376740 1184040 3108105 6906900 13123110 21474180 30421755 37442160 40116600 37442160 30421755 21474180 13123110 6906900
3108105 1184040 376740 98280 20475 3276 378 28 1
1 29 406 3654 23751 118755 475020 1560780 4292145 10015005 20030010 34597290 51895935 67863915 77558760 77558760 67863915 51895935 34597290 20030010
10015005 4292145 1560780 475020 118755 23751 3654 406 29 1
1 30 435 4060 27405 142506 593775 2035800 5852925 14307150 30045015 54627
```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1
1 20 190 1140 4845 15504 38760 77520 125970 167960 184756 167960 125970 77520 38760 15504 4845 1140 190 20 1
1 21 210 1330 5985 20349 54264 116280 203490 293930 352716 352716 293930 203490 116280 54264 20349 5985 1330 210 21 1
1 22 231 1540 7315 26334 74613 170544 319770 497420 646646 705432 646646 497420 319770 170544 74613 26334 7315 1540 231 22 1
1 23 253 1771 8855 33649 100947 245157 490314 817190 1144066 1352078 1352078 1144066 817190 490314 245157 100947 33649 8855 1771 253 23 1
1 24 276 2024 10626 42504 134596 346104 735471 1307504 1961256 2496144 2704156 2496144 1961256 1307504 735471 346104 134596 42504 10626 2024 276 24 1
1 25 300 2300 12650 53130 177100 480700 1081575 2042975 3268760 4457400 5200300 5200300 4457400 3268760 2042975 1081575 480700 177100 53130 12650 2300 300 25 1
1 26 325 2600 14950 65780 230230 657800 1562275 3124550 5311735 7726160 9657700 10400600 9657700 7726160 5311735 3124550 1562275 657800 230230 65780 14950 2600 325 26 1
1 27 351 2925 17550 80730 296010 888030 2220075 4686825 8436285 13037895 17383860 20058300 20058300 17383860 13037895 8436285 4686825 2220075 888030 296010 80730 17550 2925 351 27 1
1 28 378 3276 20475 98280 376740 1184040 3108105 6906900 13123110 21474180 30421755 37442160 40116600 37442160 30421755 21474180 13123110 6906900 3108105 1184040 376740 98280 20475 3276 378 28 1
1 29 406 3654 23751 118755 475020 1560780 4292145 10015005 20030010 34597290 51895935 67863915 77558760 77558760 67863915 51895935 34597290 20030010 10015005 4292145 1560780 475020 118755 23751 3654 406 29 1
1 30 435 4060 27405 142506 593775 2035800 5852925 14307150 30045015 54627300 86493225 119759850 145422675 155117520 145422675 119759850 86493225 54627300 30045015 14307150 5852925 2035800 593775 142506 27405 4060 435 30 1

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1
1 18 153 816 3060 8568 18564 31824 43758 48620 43758 31824 18564 8568 3060 816 153 18 1
1 19 171 969 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 171 19 1
1 20 190 1140 4845 15504 38760 77520 125970 167960 184756 167960 125970 77520 38760 15504 4845 1140 190 20 1

Código ASCII

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	32	20	(space)	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	⍰

Laws and facts

<u>Sum of squares</u>
An integer greater than one can be written as a sum of two squares if and only if its prime decomposition contains no prime congruent to 3 (mod 4) raised to an odd power.

<u>Goldbach's conjecture</u>
Every even integer greater than 2 can be expressed as the sum of two primes.

<u>Is Divisible by prime</u>
Given a number M, we want to know if it is divisible by a prime P.

1. Find the smallest K for this condition (KP + 1) MOD 10.
2. N = (KP + 1) / 10
3. Multiply the last digit of M times N and add the result to M without the last digit. (You might use (N-P) as well for the multiplication instead of N, note that it would be a negative number).
4. Repeat until M is short enough.

- $\sum_{i=0}^{a}\binom{a}{i}\binom{b}{i} = \binom{a+b}{a}$ when a <= b

- **Euler's formula** states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and *v* is the number of vertices, *e* is the number of edges and *f* is the number of faces (regions bounded by edges, including the outer, infinitely large region), then v – e + f = 2

- Inclusion-exclusion

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \cdots$$
$$\cdots + \sum_{1\le i<j<k\le n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|.$$

-